In [1]:
```
# Energy Data Analysis: Visualization and Outlier Detection

#1. Data preparation

#import Pkg; Pkg.add("DataFrames")
#import Pkg; Pkg.add("CSV")
#import Pkg; Pkg.add("Plots")
using CSV, DataFrames, Plots
```

In [3]:
```
# Load in the TSV data

#df = CSV.read("C:/Users/zizhe/Desktop/411 Labs/Assignment 1/households-24hrs.tsv", DataFrame)
df = CSV.read("households-24hrs.tsv", DataFrame)
```

Out[3]: 48 rows × 248 columns (omitted printing of 240 columns)

	ID	Time	Year	Month	Day	Hour	Minute	Reading.207860297
	Int64	String31	Int64	Int64	Int64	Int64	Int64	Float64
1	1	2009+AC0-03+AC0-04 00:30:01	2009	3	4	0	0	0.299
2	2	2009+AC0-03+AC0-04 01:00:01	2009	3	4	0	30	0.176
3	3	2009+AC0-03+AC0-04 01:30:01	2009	3	4	1	0	0.373
4	4	2009+AC0-03+AC0-04 02:00:01	2009	3	4	1	30	0.409
5	5	2009+AC0-03+AC0-04 02:30:01	2009	3	4	2	0	0.658
6	6	2009+AC0-03+AC0-04 03:00:01	2009	3	4	2	30	0.167
7	7	2009+AC0-03+AC0-04 03:30:01	2009	3	4	3	0	0.206
8	8	2009+AC0-03+AC0-04 04:00:01	2009	3	4	3	30	0.173
9	9	2009+AC0-03+AC0-04 04:30:01	2009	3	4	4	0	0.184
10	10	2009+AC0-03+AC0-04 05:00:01	2009	3	4	4	30	0.18
11	11	2009+AC0-03+AC0-04 05:30:01	2009	3	4	5	0	0.197
12	12	2009+AC0-03+AC0-04 06:00:01	2009	3	4	5	30	0.609
13	13	2009+AC0-03+AC0-04 06:30:01	2009	3	4	6	0	0.24
14	14	2009+AC0-03+AC0-04 07:00:01	2009	3	4	6	30	1.078
15	15	2009+AC0-03+AC0-04 07:30:01	2009	3	4	7	0	1.164
16	16	2009+AC0-03+AC0-04 08:00:01	2009	3	4	7	30	0.163
17	17	2009+AC0-03+AC0-04 08:30:01	2009	3	4	8	0	0.683
18	18	2009+AC0-03+AC0-04 09:00:01	2009	3	4	8	30	0.173
19	19	2009+AC0-03+AC0-04 09:30:01	2009	3	4	9	0	0.619
20	20	2009+AC0-03+AC0-04 10:00:01	2009	3	4	9	30	0.186
21	21	2009+AC0-03+AC0-04 10:30:01	2009	3	4	10	0	0.153
22	22	2009+AC0-03+AC0-04 11:00:01	2009	3	4	10	30	0.184
23	23	2009+AC0-03+AC0-04 11:30:01	2009	3	4	11	0	0.187

	ID	Time	Year	Month	Day	Hour	Minute	Reading.207860297
	Int64	String31	Int64	Int64	Int64	Int64	Int64	Float64
24	24	2009+AC0-03+AC0-04 12:00:01	2009	3	4	11	30	0.158
25	25	2009+AC0-03+AC0-04 12:30:01	2009	3	4	12	0	0.189
26	26	2009+AC0-03+AC0-04 13:00:01	2009	3	4	12	30	0.335
27	27	2009+AC0-03+AC0-04 13:30:01	2009	3	4	13	0	0.438
28	28	2009+AC0-03+AC0-04 14:00:01	2009	3	4	13	30	0.188
29	29	2009+AC0-03+AC0-04 14:30:01	2009	3	4	14	0	0.163
30	30	2009+AC0-03+AC0-04 15:00:01	2009	3	4	14	30	0.186
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Convert the DataFrame into a Matrix

data = Matrix(df)[:,8:end]

In [9]:
```
# Convert to float64 otherwise would have error message later on

data=Float64.(data)
```

Out[9]:
```
48×241 Matrix{Float64}:
 0.299  0.208  0.214  0.064  0.211  …  0.373  0.186  0.121  0.484  0.214
 0.176  0.336  0.127  0.08   0.149     0.162  0.608  0.296  0.42   0.196
 0.373  0.342  0.111  0.188  0.109     0.16   0.172  0.739  0.368  0.192
 0.409  0.532  0.089  0.177  0.131     0.16   0.18   0.129  0.616  0.21
 0.658  0.202  0.533  0.077  0.137     0.16   0.166  0.138  0.194  0.166
 0.167  0.337  0.12   0.06   0.108  …  0.158  0.168  0.136  0.194  0.178
 0.206  0.328  0.089  0.061  0.132     0.16   0.176  0.127  0.2    0.158
 0.173  0.224  0.646  0.213  0.141     0.158  0.17   0.138  0.232  0.164
 0.184  0.199  0.167  0.044  0.096     0.096  0.166  0.278  1.198  0.176
 0.18   0.315  0.144  0.052  0.225     0.126  1.826  0.629  0.276  0.16
 0.197  0.332  0.146  0.061  0.122  …  0.846  0.732  0.559  0.214  0.174
 0.609  0.69   0.116  0.236  0.124     0.365  1.928  0.162  0.702  0.213
 0.24   0.451  1.195  0.288  0.418     0.162  0.438  0.855  0.384  0.478
 ⋮                                ⋱                             ⋮
 0.31   0.222  0.926  0.4    0.497     0.553  0.524  0.822  0.828  0.693
 0.738  0.667  0.959  0.472  0.522     0.612  1.042  0.681  0.654  0.762
 0.933  0.415  0.335  0.156  0.825     0.632  0.546  0.33   0.86   0.764
 0.535  0.469  0.933  0.594  0.512     0.557  1.082  0.286  0.926  0.748
 0.684  1.019  1.076  0.242  0.631  …  0.6    1.998  1.499  0.32   0.807
 0.502  1.05   1.383  0.407  1.099     0.902  0.82   0.816  0.422  0.566
 0.588  2.14   2.02   0.247  0.534     0.365  0.598  0.52   0.584  0.268
 1.076  2.067  2.251  0.434  0.218     0.184  0.538  0.147  0.3    0.182
 0.57   0.675  0.64   0.465  0.394     0.184  0.19   0.127  0.296  0.18
 0.166  0.368  1.032  0.166  0.121  …  0.162  0.194  0.111  0.466  0.185
 0.182  0.741  0.77   0.168  0.102     0.16   0.168  0.115  0.49   0.195
 0.376  0.356  0.88   0.297  0.221     0.201  0.634  0.123  0.668  0.191
```

JD: Good start! Do remember to remove the absolute path ("C:\...") when doing and submitting your work - I don't use the same path as yours. Also comment off "Pkg.add()" lines.

In [ ]:

In [4]:
```
# 2. Visual exploratory analysis

#import Pkg; Pkg.add("StatsBase")

using Statistics, StatsBase, LinearAlgebra
```

In [5]:
```
#import Pkg; Pkg.add("Distances")

#import Pkg; Pkg.add("Clustering")
```

```
using Clustering, Distances
```

In [10]:
```
# get the Silhouette Coefficients of kmeans when k = 2 to 10

dists = pairwise(SqEuclidean(), data) # get the pairwise distance

result = kmeans(data, 2; maxiter=100, display=:iter) #group all data points into 2 clusters
memb = assignments(result)
sc=mean(silhouettes(memb, dists))

for i in 3:10
    result = kmeans(data, i; maxiter=100, display=:iter) #group all data points into 3 to 10 clusters
    memb = assignments(result)
    sc=vcat(sc,mean(silhouettes(memb, dists)))
end

sc # to show sc
```

```
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       2.513344e+03
      1       1.376400e+03      -1.136943e+03 |        2
      2       1.284294e+03      -9.210632e+01 |        2
      3       1.275578e+03      -8.715907e+00 |        2
      4       1.275326e+03      -2.524212e-01 |        0
      5       1.275326e+03       0.000000e+00 |        0
K-means converged with 5 iterations (objv = 1275.325833169753)
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       2.028534e+03
      1       1.241334e+03      -7.871996e+02 |        3
      2       1.224779e+03      -1.655492e+01 |        3
      3       1.222127e+03      -2.652626e+00 |        2
      4       1.220360e+03      -1.766097e+00 |        2
      5       1.220093e+03      -2.670869e-01 |        0
      6       1.220093e+03       0.000000e+00 |        0
K-means converged with 6 iterations (objv = 1220.0934086291277)
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       1.707883e+03
      1       1.234155e+03      -4.737283e+02 |        4
      2       1.178506e+03      -5.564851e+01 |        4
      3       1.172484e+03      -6.021920e+00 |        4
      4       1.167296e+03      -5.187708e+00 |        4
      5       1.162964e+03      -4.332058e+00 |        4
      6       1.161495e+03      -1.469117e+00 |        3
      7       1.160413e+03      -1.082037e+00 |        4
      8       1.156900e+03      -3.512825e+00 |        3
      9       1.155819e+03      -1.081657e+00 |        2
     10       1.155568e+03      -2.505365e-01 |        2
     11       1.155119e+03      -4.493992e-01 |        0
     12       1.155119e+03       0.000000e+00 |        0
K-means converged with 12 iterations (objv = 1155.118774248904)
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       1.757072e+03
      1       1.182732e+03      -5.743402e+02 |        5
      2       1.156445e+03      -2.628688e+01 |        5
      3       1.148339e+03      -8.105236e+00 |        3
      4       1.145695e+03      -2.644270e+00 |        5
      5       1.139405e+03      -6.289887e+00 |        5
      6       1.130237e+03      -9.168356e+00 |        4
      7       1.126074e+03      -4.163132e+00 |        4
```

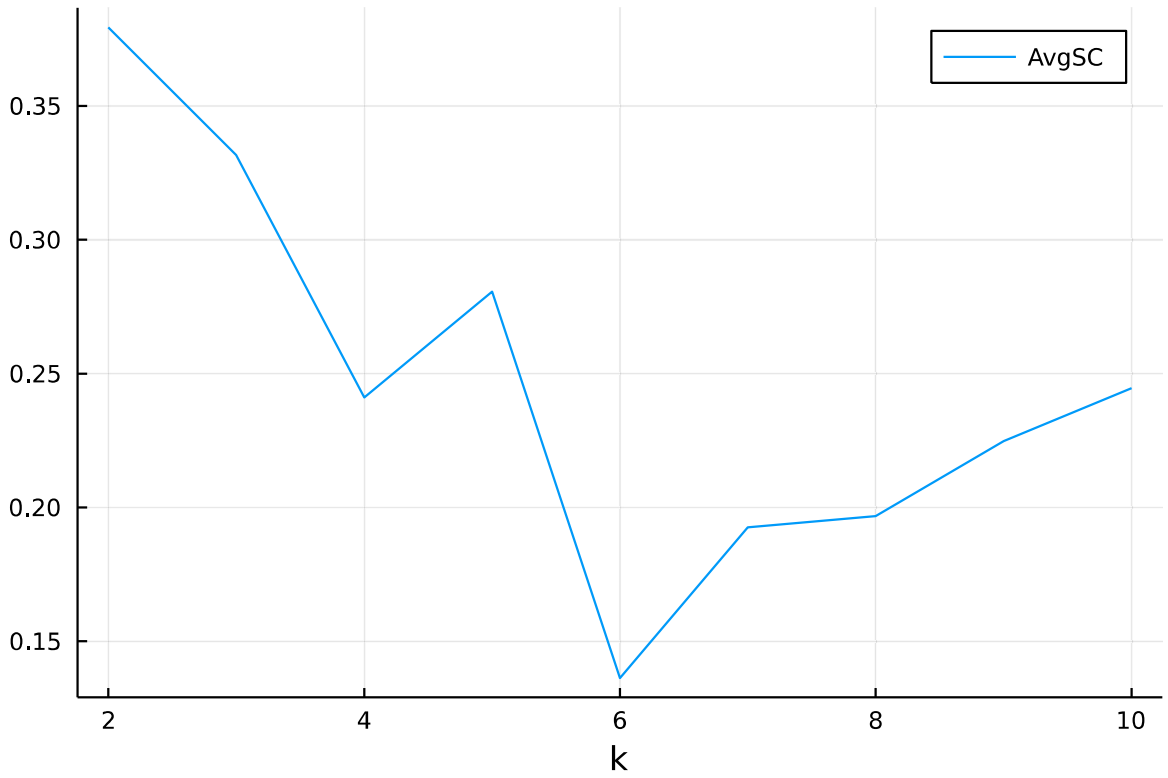
```
      8       1.122703e+03      -3.370777e+00 |        3
      9       1.121285e+03      -1.418217e+00 |        3
     10       1.120422e+03      -8.631301e-01 |        0
     11       1.120422e+03       0.000000e+00 |        0
K-means converged with 11 iterations (objv = 1120.421655189311)
  Iters               objv        objv-change | affected
-------------------------------------------------------------
      0       1.504465e+03
      1       1.135961e+03      -3.685041e+02 |        6
      2       1.114867e+03      -2.109392e+01 |        5
      3       1.103958e+03      -1.090945e+01 |        4
      4       1.102005e+03      -1.952732e+00 |        3
      5       1.100677e+03      -1.328441e+00 |        3
      6       1.099286e+03      -1.390672e+00 |        0
      7       1.099286e+03       0.000000e+00 |        0
K-means converged with 7 iterations (objv = 1099.2859909841748)
  Iters               objv        objv-change | affected
-------------------------------------------------------------
      0       1.855961e+03
      1       1.147396e+03      -7.085654e+02 |        6
      2       1.104873e+03      -4.252288e+01 |        5
      3       1.081340e+03      -2.353275e+01 |        5
      4       1.069668e+03      -1.167290e+01 |        4
      5       1.065497e+03      -4.170853e+00 |        3
      6       1.063537e+03      -1.959938e+00 |        3
      7       1.062009e+03      -1.527850e+00 |        2
      8       1.060990e+03      -1.018724e+00 |        3
      9       1.059762e+03      -1.228426e+00 |        4
     10       1.058488e+03      -1.273867e+00 |        2
     11       1.057485e+03      -1.002969e+00 |        2
     12       1.057322e+03      -1.628683e-01 |        0
     13       1.057322e+03       0.000000e+00 |        0
K-means converged with 13 iterations (objv = 1057.3220159001473)
  Iters               objv        objv-change | affected
-------------------------------------------------------------
      0       1.880478e+03
      1       1.093896e+03      -7.865817e+02 |        7
      2       1.067912e+03      -2.598424e+01 |        6
      3       1.060873e+03      -7.039186e+00 |        6
      4       1.056623e+03      -4.249776e+00 |        7
      5       1.051056e+03      -5.566976e+00 |        4
      6       1.049106e+03      -1.950007e+00 |        5
      7       1.047190e+03      -1.916143e+00 |        0
      8       1.047190e+03       0.000000e+00 |        0
K-means converged with 8 iterations (objv = 1047.1896845157)
```

```
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       1.530727e+03
      1       1.110119e+03      -4.206078e+02 |        6
      2       1.085290e+03      -2.482837e+01 |        4
      3       1.078871e+03      -6.419886e+00 |        4
      4       1.074773e+03      -4.097955e+00 |        3
      5       1.071333e+03      -3.439547e+00 |        3
      6       1.070326e+03      -1.007142e+00 |        3
      7       1.069703e+03      -6.225802e-01 |        3
      8       1.069171e+03      -5.321547e-01 |        0
      9       1.069171e+03       0.000000e+00 |        0
K-means converged with 9 iterations (objv = 1069.1711396692815)
  Iters               objv        objv-change | affected
------------------------------------------------------------
      0       1.472950e+03
      1       1.067774e+03      -4.051760e+02 |        8
      2       1.032745e+03      -3.502927e+01 |        5
      3       1.018427e+03      -1.431780e+01 |        5
      4       1.014909e+03      -3.518545e+00 |        4
      5       1.012940e+03      -1.968958e+00 |        0
      6       1.012940e+03       0.000000e+00 |        0
K-means converged with 6 iterations (objv = 1012.9398276738517)
```

Out[10]:
```
9-element Vector{Float64}:
 0.37930497978972294
 0.33165054790216514
 0.24113335764183705
 0.2805950103258804
 0.13632453222011737
 0.1925480402734555
 0.19671791885935533
 0.22485502733783472
 0.24461051983205975
```

In [11]:
```
plot(2:10, sc, label="AvgSC", xlabel="k")

#the higher the Silhouette Coefficient is better, that means when k=2, the clustering is more accurate. So we choose to
```

Out[11]:



In [12]:

```
# get the membership of kmeans when k=2

km = kmeans(data, 2; maxiter=100)
a = assignments(km)
```

Out[12]:
```
241-element Vector{Int64}:
 1
 2
 2
 1
 1
 2
 1
 2
 1
 2
 1
 1
 2
 ⋮
 2
 1
 1
 2
 1
 2
 2
 1
 2
 1
 2
 1
```

In [13]:
```
a1=findall(a -> a == 1,a) # find the index of data when membership = 1

c1=data[:,a1] # get the original data that belongs to membership = 1

c1=c1' # transpose as needed later on
```

Out[13]:

```
160×48 adjoint(::Matrix{Float64}) with eltype Float64:
 0.299  0.176  0.373  0.409  0.658  …  1.076  0.57   0.166  0.182  0.376
 0.064  0.08   0.188  0.177  0.077     0.434  0.465  0.166  0.168  0.297
 0.211  0.149  0.109  0.131  0.137     0.218  0.394  0.121  0.102  0.221
 0.383  0.303  0.363  0.208  0.121     0.36   0.25   0.215  0.128  0.148
 0.142  0.393  0.195  0.131  0.198     0.426  0.706  0.713  0.079  0.349
 0.084  0.273  0.653  0.215  0.243  …  0.202  0.784  1.758  0.775  0.308
 0.175  0.39   0.227  0.098  0.149     0.528  0.156  0.165  0.148  0.129
 0.098  0.161  0.173  0.15   0.187     0.155  0.135  0.157  0.208  0.108
 0.107  0.054  0.079  0.105  0.045     0.639  1.661  0.74   0.101  0.068
 0.69   0.016  0.016  0.018  0.015     0.065  0.024  0.024  0.085  0.085
 0.113  0.16   0.143  0.181  0.08   …  0.307  0.202  0.187  0.176  0.18
 0.078  0.107  0.222  0.097  0.1       0.052  0.234  0.06   0.222  0.059
 0.802  0.199  0.233  0.19   0.208     0.128  0.665  1.121  0.247  0.416
 ⋮                                ⋱              ⋮
 0.044  0.144  0.118  0.116  0.118     1.084  1.208  1.184  1.13   0.79
 0.172  0.196  0.126  0.16   0.176     0.224  0.178  0.196  0.65   0.188
 0.082  0.076  0.064  0.08   0.074  …  0.188  0.106  0.098  0.58   0.122
 0.07   0.09   0.086  0.09   0.113     0.71   1.11   0.733  0.26   0.106
 0.058  0.016  0.03   0.056  0.049     0.09   0.777  0.073  0.035  0.068
 0.278  0.188  0.214  0.194  0.216     0.166  0.398  1.426  1.046  0.206
 0.094  0.096  0.098  0.096  0.1       0.44   0.102  0.102  0.1    0.09
 0.11   0.112  0.12   0.116  0.097  …  0.108  0.116  0.114  0.114  0.11
 0.062  0.055  0.117  0.1    0.078     0.741  0.597  1.089  0.395  0.271
 0.373  0.162  0.16   0.16   0.16      0.184  0.184  0.162  0.16   0.201
 0.121  0.296  0.739  0.129  0.138     0.147  0.127  0.111  0.115  0.123
 0.214  0.196  0.192  0.21   0.166     0.182  0.18   0.185  0.195  0.191
```

In [14]:

```
# same as code above, but get the original data that belongs to membership = 2

a2=findall(a -> a == 2,a)

c2=data[:,a2]

c2=c2'
```

Out[14]:

```
81×48 adjoint(::Matrix{Float64}) with eltype Float64:
 0.208  0.336  0.342  0.532  0.202  …  2.067  0.675  0.368  0.741  0.356
 0.214  0.127  0.111  0.089  0.533     2.251  0.64   1.032  0.77   0.88
 1.366  0.555  0.41   0.31   0.309     0.815  0.632  0.576  0.46   0.413
 0.353  0.232  0.37   0.233  0.343     1.934  1.011  0.21   0.351  0.242
 0.681  0.241  0.269  0.238  0.247     0.751  1.05   0.51   0.252  0.562
 0.39   0.449  0.348  0.423  0.419  …  0.547  0.521  0.444  0.433  0.402
 0.353  0.321  0.278  0.33   0.345     2.449  1.793  1.622  0.277  0.334
 0.333  0.393  0.382  0.376  0.943     0.465  0.235  0.234  0.647  0.194
 0.193  0.184  0.148  0.185  0.56      0.266  0.294  0.639  0.676  1.284
 0.18   0.067  0.414  0.066  0.076     0.384  0.159  0.131  0.463  0.139
 0.145  0.621  0.16   0.145  0.151  …  0.628  0.297  0.163  0.162  0.751
 0.186  0.48   0.182  0.234  0.136     1.115  0.38   0.95   0.348  0.282
 0.156  0.15   0.09   0.094  0.148     0.911  0.701  0.738  0.21   0.146
 ⋮                                ⋱                ⋮
 0.222  0.162  0.226  0.928  0.29      0.454  0.212  0.204  0.766  0.132
 0.616  1.262  0.332  0.276  0.426  …  2.041  0.507  0.991  1.335  0.493
 0.139  0.198  0.185  0.138  0.148     0.482  0.475  0.877  0.261  0.208
 0.532  0.638  0.244  0.484  0.362     0.243  0.597  0.238  0.541  0.452
 0.071  0.085  0.069  0.081  0.071     0.921  1.194  0.731  0.144  0.133
 0.234  0.198  0.142  0.174  0.132     0.485  0.479  0.514  0.462  0.197
 0.128  0.132  0.11   0.102  0.142  …  0.356  0.201  0.123  0.153  0.102
 0.15   0.13   0.102  0.158  0.458     1.013  0.675  0.26   0.161  0.183
 0.162  0.112  0.154  0.162  0.138     0.16   0.158  0.138  0.118  0.162
 0.136  0.564  0.134  0.096  0.096     0.272  1.208  0.126  0.126  0.18
 0.186  0.608  0.172  0.18   0.166     0.538  0.19   0.194  0.168  0.634
 0.484  0.42   0.368  0.616  0.194  …  0.3    0.296  0.466  0.49   0.668
```

In [18]:

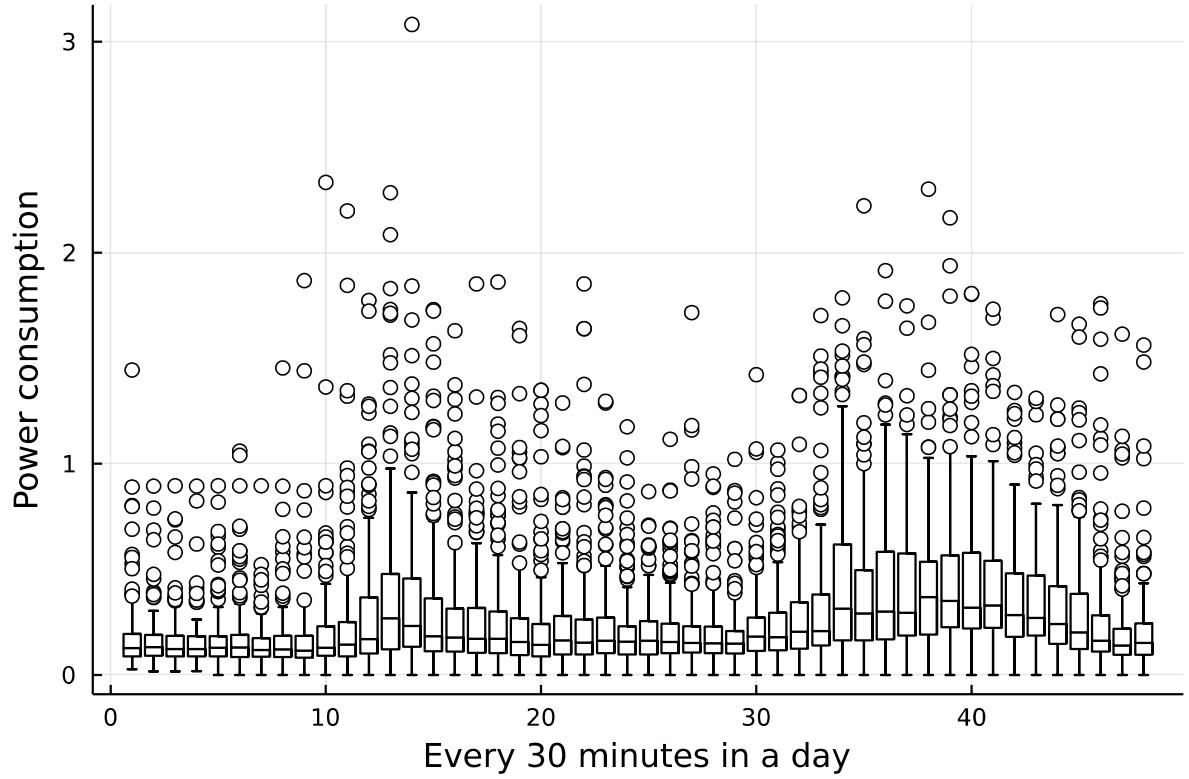
```
#Pkg.add("StatsPlots");
using StatsPlots
```

In [19]:

```
# plot the original data when membership = 1

boxplot(c1,legend = false,color="white",ylabel="Power consumption",xlabel="Every 30 minutes in a day")
```

```
┌ Warning: Keyword argument hover not supported with Plots.GRBackend().  Choose from: Set([:top_margin, :group, :inset_
subplots, :background_color, :ytickfontsize, :yforeground_color_text, :yguidefontcolor, :tickfontfamily, :show_empty_bi
ns, :seriesalpha, :seriescolor, :ztick_direction, :xgrid, :ygridalpha, :zlims, :xtick_direction, :colorbar, :legend_fon
t_family, :zflip, :ticks, :linealpha, :overwrite_figure, :arrow, :xguidefonthalign, :normalize, :linestyle, :xtickfontv
align, :xflip, :zgrid, :fillcolor, :ygrid, :bar_width, :colorbar_scale, :background_color_inside, :zguidefonthalign, :b
ins, :zguide, :zforeground_color_text, :legend_font_valign, :yscale, :legend_font_color, :weights, :xgridalpha, :ygrids
tyle, :clims, :xtickfontcolor, :fill_z, :xguide, :markershape, :background_color_subplot, :ztickfontfamily, :fillalpha,
:markerstrokewidth, :tick_direction, :xguidefontvalign, :xguidefontfamily, :gridlinewidth, :foreground_color_subplot, :
xgridlinewidth, :yguidefontsize, :foreground_color, :foreground_color_text, :titlefonthalign, :yerror, :x, :xtickfontha
lign, :zgridlinewidth, :ytickfontrotation, :discrete_values, :ytick_direction, :grid, :xguidefontrotation, :ribbon, :xg
uidefontsize, :tickfontrotation, :xforeground_color_axis, :xdiscrete_values, :background_color_outside, :titlefontcolo
r, :xgridstyle, :line_z, :size, :orientation, :gridstyle, :projection, :markersize, :legend_foreground_color, :camera,
:zguidefontrotation, :ydiscrete_values, :xforeground_color_grid, :seriestype, :yflip, :quiver, :zticks, :markerstrokeco
lor, :ztickfontrotation, :ztickfonthalign, :fillrange, :ztickfontvalign, :xlims, :xforeground_color_border, :markercolo
r, :xtickfontsize, :ylink, :levels, :color_palette, :connections, :yforeground_color_grid, :lims, :zgridstyle, :foregro
und_color_border, :zguidefontvalign, :xscale, :marker_z, :markerstrokealpha, :left_margin, :markeralpha, :legend_font_h
align, :annotations, :window_title, :tickfontvalign, :foreground_color_axis, :zguidefontcolor, :ygridlinewidth, :zlink,
:zscale, :smooth, :yguidefontrotation, :xticks, :guidefontsize, :zguidefontsize, :y, :margin, :ytickfontcolor, :zdiscre
te_values, :tickfonthalign, :bottom_margin, :yforeground_color_border, :zguidefontfamily, :framestyle, :yguidefontvalig
n, :yguidefonthalign, :zerror, :zgridalpha, :ztickfontcolor, :scale, :legend_position, :linecolor, :html_output_format,
:legend_title, :zforeground_color_border, :legend_font_pointsize, :title, :tickfontcolor, :subplot_index, :flip, :title
fontrotation, :legend_background_color, :tickfontsize, :titlefontvalign, :z, :yforeground_color_axis, :foreground_color
_grid, :xtickfontrotation, :linewidth, :ztickfontsize, :gridalpha, :xerror, :guidefontfamily, :ylims, :contour_labels,
:xguidefontcolor, :primary, :xtickfontfamily, :ytickfontvalign, :guidefonthalign, :ytickfontfamily, :aspect_ratio, :xfo
reground_color_text, :show, :link, :colorbar_title, :guidefontrotation, :subplot, :label, :ytickfonthalign, :guide, :gu
idefontcolor, :yguide, :titlefontsize, :titlefontfamily, :guidefontvalign, :zforeground_color_axis, :zforeground_color_
grid, :layout, :legend_font_rotation, :colorbar_entry, :yguidefontfamily, :polar, :right_margin, :xlink, :series_annota
tions, :yticks])
└ @ Plots /Users/denda47p/.julia/packages/Plots/Xv2qA/src/args.jl:1607
```

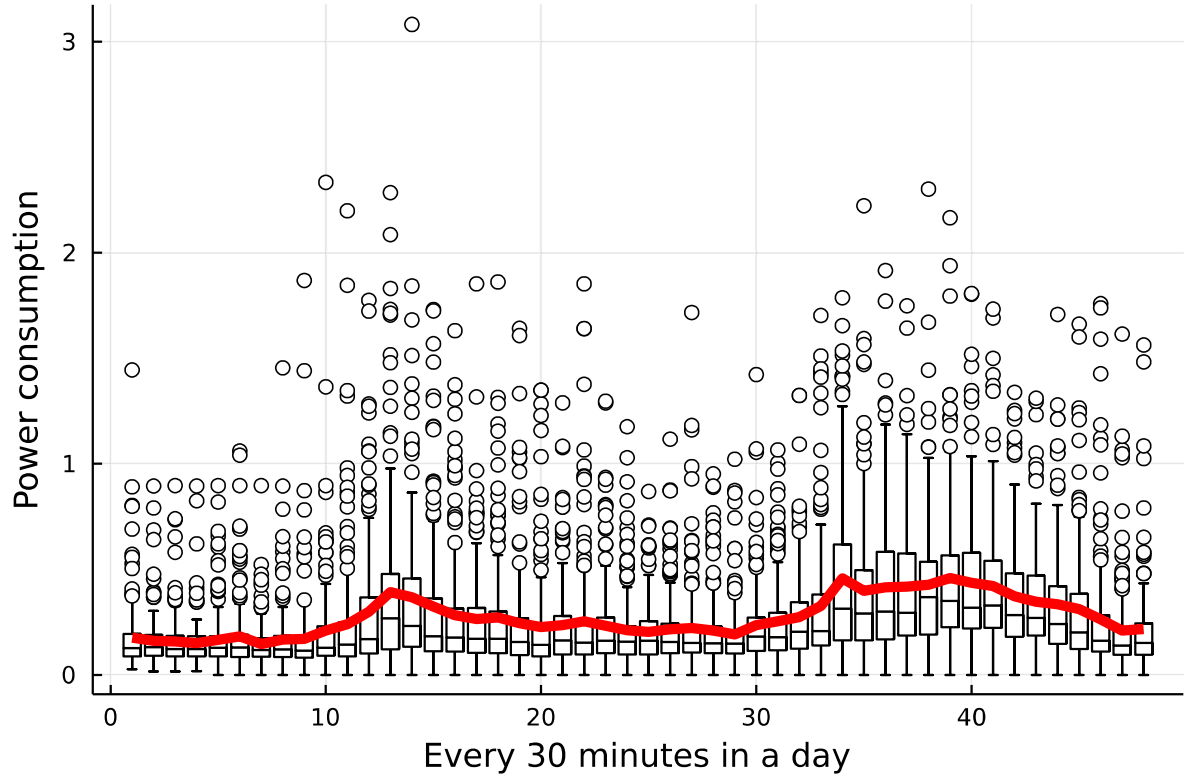
Out[19]:



In [20]:

```
# plot the mean when membership = 1

mc1=mean(c1,dims=1)
mc1=mc1'

plot!(mc1,color="red",lw = 5)
```

Out[20]:

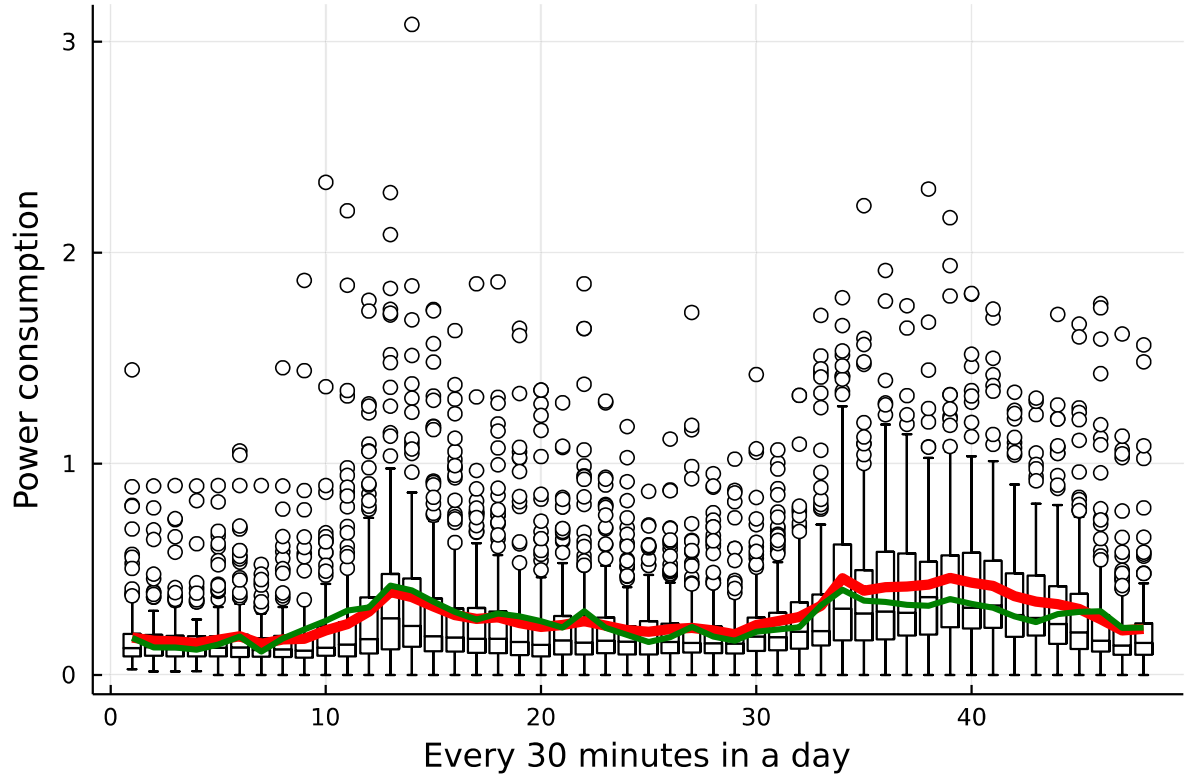


In [21]:
```
sd1=std(c1,dims=1)
```

Out[21]:
```
1×48 Matrix{Float64}:
 0.17366  0.128363  0.128214  0.117674  …  0.300264  0.220931  0.222449
```

In [22]:
```
# plot the standard deviation when membership = 1

sd1=std(c1,dims=1)
sd1=sd1'

plot!(sd1,color="green",lw=3)
```

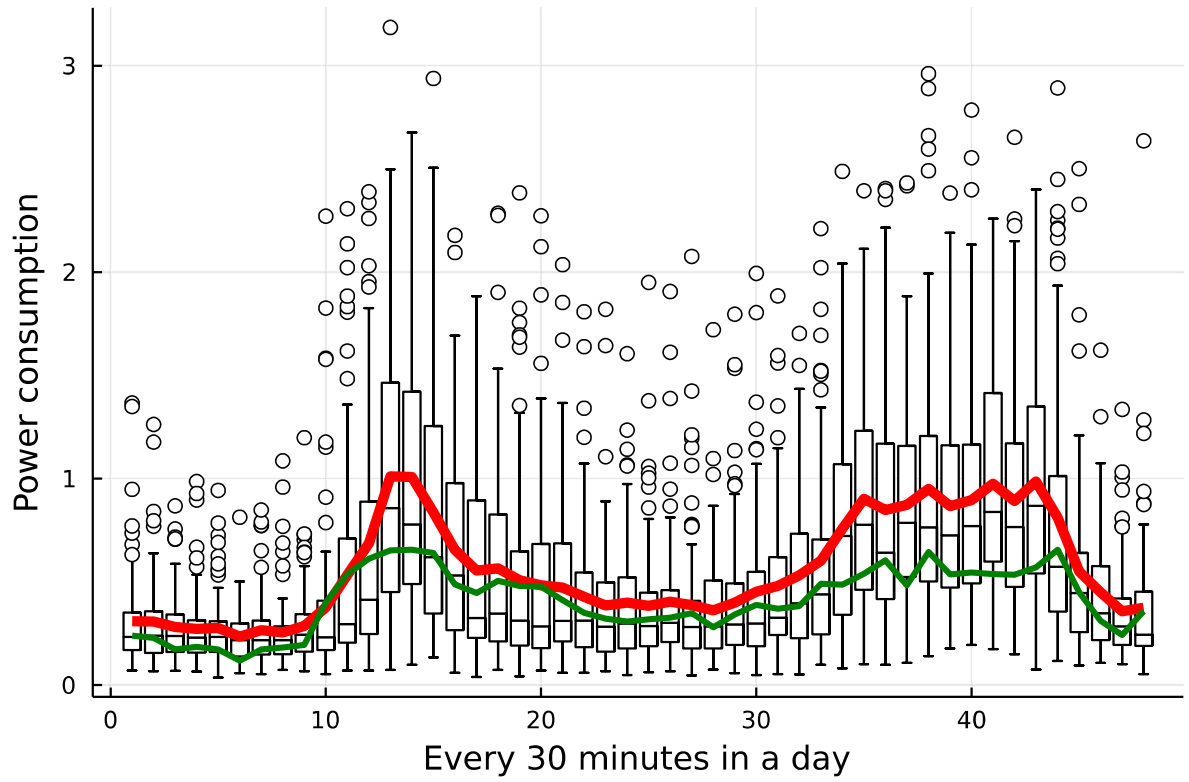
Out[22]:



In [23]:
```
# in the above plot, red line represents the mean, green line represents the standard deviation
```

In [24]:
```
# same as the above code, plot the original data when membership = 2, plus to plot the mean and the standard deviation

boxplot(c2,legend = false,color="white",ylabel="Power consumption",xlabel="Every 30 minutes in a day")

mc2=mean(c2,dims=1)
mc2=mc2'
plot!(mc2,color="red",lw = 5)

sd2=std(c2,dims=1)
sd2=sd2'
plot!(sd2,color="green",lw = 3)
```

Out[24]:



In [80]:
```
# In the above plot, red line represents the mean, green line represents the standard deviation.
```

In [ ]:
```
# What do these patterns inform?
```

In [ ]:
```
# I tried to cluster the original data by using kmeans when k = 2 to 10, the result shows when k = 2 is the best settin
# So I choose to cluster the original data into 2 groups and plot them out.

# From the 2 plots, I can see that when membership = 1, the households tend to have a relatively stable and less eletri
# That could implies the household in group 1, may have less people, or just lower users in general.

# When membership = 2, the households tend to have more eletricity usage during breakfast and dinner time, the mean and
# That could implies the household in group 2, may have more people, or just higher users in general, or people need to
```

JD: Change the cells into "Markdown" when writing comments. !!

In [ ]:

In [25]:
```
#3.PCA and visualization

m = mean(data, dims=2)   # get the mean vector (per attribute across all entries)
R = (data.- m)*(data.- m)' # get the covariance
```

Out[25]:
```
48×48 Matrix{Float64}:
 10.3419   3.92279  2.82327  2.30583  …   4.28155    4.26517   4.48548
  3.92279  7.93575  3.14158  2.57431       2.73171    2.59158   2.90492
  2.82327  3.14158  5.81741  2.63273       2.6125     2.11868   2.52624
  2.30583  2.57431  2.63273  5.72686       2.15039    2.33695   3.24244
  2.142    2.37876  2.06883  2.06062       1.87537    1.69752   2.39697
  1.87811  2.08944  2.06741  2.13032  …    2.05145    1.42079   1.09672
  2.5621   2.55499  2.74824  2.23828       2.1541     1.61529   2.50115
  2.02433  2.27448  1.8335   1.63519       1.82915    2.06677   1.81685
  2.89665  2.9294   2.70886  2.89694       1.13219    1.43097   1.27455
  2.27099  3.20802  3.50744  4.64424       2.36666    1.74521   6.78037
  2.33127  3.1493   3.30487  4.85765  …  -0.970507   0.30273   2.54072
  4.01436  6.26199  3.85453  5.0462        0.03606    2.44808   2.16064
  4.76285  5.77524  5.60526  3.92701       4.47856    4.81058   4.03231
  ⋮                                   ⋱   ⋮
  5.95256  4.74852  3.50033  2.84276       4.43677    5.81232   4.49598
  5.77216  6.13001  6.32356  4.84293       8.83121    7.29454  13.025
  4.13337  3.65088  4.20201  4.38137       5.91869    6.45431   8.26145
  6.32913  6.01747  3.88619  6.267         7.31411    7.67594   5.60565
  6.13883  6.63999  5.12745  5.06632  …    8.01842    7.27027   4.37876
  8.60196  6.97379  5.95281  5.37657      10.5278     8.70444   6.29316
  5.93954  8.17987  5.60458  6.36613       9.55942    8.85428   8.78888
  5.5303   4.40552  3.71891  3.35207      13.1825     9.07235   8.22666
  6.24468  2.69885  1.70222  1.70805      14.8616     6.09165   7.43099
  4.28155  2.73171  2.6125   2.15039  …   23.9727    10.1335    7.35093
  4.26517  2.59158  2.11868  2.33695      10.1335    13.6703    8.25054
  4.48548  2.90492  2.52624  3.24244       7.35093    8.25054  19.408
```

In [26]:
```
# get the eigenvalues

eigvals(R)
```

Out[26]:
```
48-element Vector{Float64}:
   1.5364440681859448
   2.062677539091256
   2.2070763112172664
   2.921442335205112
   3.1227003869010588
   3.426901173454448
   3.826417821744972
   4.050980872996767
   4.373717663019209
   4.733117552981962
   4.815706421026518
   5.239048577524982
   6.06722844262388
   ⋮
  32.72228741744937
  36.36948032051839
  39.61659278065371
  41.30121377953084
  50.79018145630751
  53.941150431645994
  58.079053078948526
  77.53024444319897
  97.2537587458017
  99.61983549907653
 115.53501890157493
 412.2258586029577
```

In [27]:
```
# get eigenvectors

evecs = eigvecs(R)
```

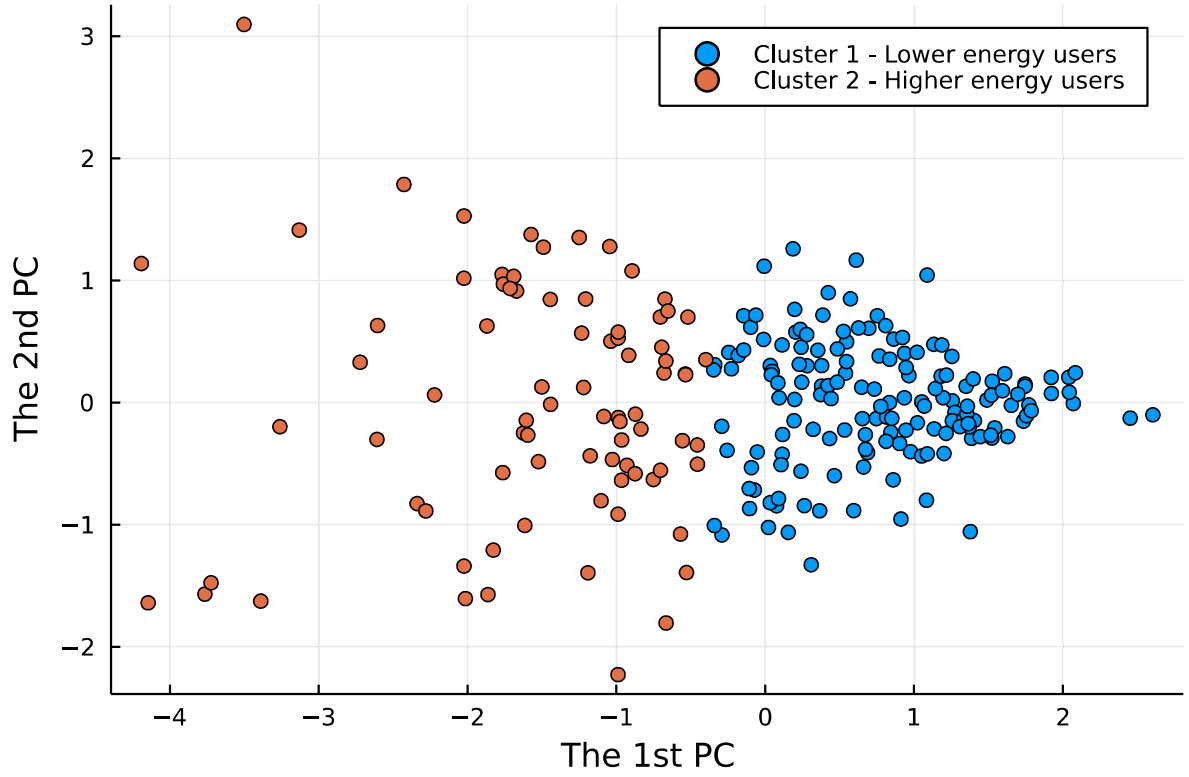
Out[27]:

```
48×48 Matrix{Float64}:
 -0.0189238    0.0510614   -0.0801123   …   0.0366548  -0.0639892
  0.150441    -0.123895    -0.438807        0.0214334  -0.0616843
 -0.323514    -0.241641     0.635778        0.0296949  -0.0596783
  0.0753618    0.803282    -0.0193944       0.0106112  -0.0553129
  0.1969       0.0677541    0.22993         0.0159943  -0.0524941
 -0.204001    -0.161874    -0.2692      …   0.0207941  -0.0400981
  0.801231    -0.228995     0.171711        0.0443434  -0.0537433
 -0.197825     0.0821634    0.0552272       0.0179742  -0.0434374
 -0.0270177   -0.0627653   -0.0270425       0.0278564  -0.0498034
 -0.0906456   -0.0179507   -0.105169       -0.0474405  -0.0740393
  0.0291991   -0.0449131   -0.00768005  …  -0.128979   -0.112417
 -0.0462299    0.00502154   0.0480983      -0.23976    -0.166192
 -0.0212112    0.0191595   -0.0354043      -0.373745   -0.269708
  ⋮                                   ⋱
 -0.00662025   0.00735072   0.0306393      -0.149103   -0.185448
  0.0564218    0.0181749    0.00429062     -0.135458   -0.248403
 -0.039005     0.00342648  -0.0561595      -0.237986   -0.208153
  0.0232137   -0.0646321    0.0511323      -0.223506   -0.220103
 -0.00553257   0.0248865   -0.0148162   …  -0.154926   -0.225145
  0.0178484    0.0200019    0.011941       -0.0186772  -0.239102
 -0.0539938   -0.00185071   0.0633764      -0.0510295  -0.260054
  0.0311248   -0.0311246   -0.0613282       0.14523    -0.203736
 -0.0440664    0.0864358    0.0915216       0.111855   -0.111267
  0.015703    -0.0539716   -0.0453335   …   0.116219   -0.088303
  0.046045    -0.0332255   -0.0953079       0.0280232  -0.0719005
 -0.0487879   -0.0753207    0.0787615       0.0413667  -0.0795272
```

In [28]:

```
# take the last two columns (actually in maths the 2nd and 1st eigenvectors with the largest eigenvalues) to produce 2-

proj = evecs[:, end-1:end]' * (data.- m)

cl1=proj[:,a1]
cl2=proj[:,a2]

scatter(cl1[2, :], cl1[1, :], label="Cluster 1 - Lower energy users", xlabel="The 1st PC", ylabel="The 2nd PC")
scatter!(cl2[2, :], cl2[1, :], label="Cluster 2 - Higher energy users", xlabel="The 1st PC", ylabel="The 2nd PC")
```

Out[28]:



In [ ]:

In [ ]:

In [29]:
```
#4. Online PCA
```

In [30]:
```
# get the first 20 data vectors, the mean and the covariance

X20 = data[:,1:20]
m20 = mean(X20, dims=2)
R20 = (X20 .- m20)*(X20 .- m20)'
```

Out[30]:
```
48×48 Matrix{Float64}:
  1.71221    0.354513   0.100056   …  -0.225808   -0.0319372   0.066903
  0.354513   0.407448   0.205781       0.252344    0.151068    0.034105
  0.100056   0.205781   0.430735       0.408361    0.329845   -0.096509
  0.141724   0.218919   0.199672       0.111474    0.240958    0.076173
  0.0792026  0.215137   0.168779       0.254147    0.572813    0.5908
  0.0728882  0.208773   0.0665314  …   0.774256   -0.0703087  -0.14762
  0.405259   0.213373   0.162815       0.101965    0.161452    0.17443
  0.165357   0.204138   0.014782       0.237666    0.285729    0.294747
  0.425838   0.259288   0.191305       0.0488872   0.269815    0.0990358
  0.340398   0.388046   0.48459        0.938207    0.719116    0.0548848
  0.407499   0.283103   0.138449   …   0.0249027   0.180423    0.0399635
  2.07163    0.76783    0.469233      -0.133416    0.444031    0.22249
  1.48747    0.54191    0.198785       2.98292     1.42406     1.91296
  ⋮                                ⋱   ⋮
  0.823685   0.581844   0.32723        0.829403    0.947759    0.806794
  1.27135    0.17003    0.104304       1.35494     0.388343    0.45904
  1.78509    0.71446    0.384105       1.35301     0.0559881   0.481912
  0.927629   0.443478  -0.000882       0.897634    0.100556    0.921545
  1.85774    0.611484   0.628704   …   0.0622579   0.801853    0.52953
  2.26084    0.735037   0.552865       0.765472    1.34444     1.34271
  0.875818   0.545656   0.460585       1.2523      1.65856     0.82068
  0.174216   0.278345   0.139109       2.47689     1.14123     0.919026
  0.027379   0.071172   0.059096       2.78348    -0.0939995   0.0763898
 -0.225808   0.252344   0.408361   …   4.46864     0.950845    0.825597
 -0.0319372  0.151068   0.329845       0.950845    1.18975     0.735011
  0.066903   0.034105  -0.096509       0.825597    0.735011    1.60498
```

In [31]:
```
# function to calculate the final covariance by using Online PCA and the first 20 data vectors

function cov(r,data,m,R)
    for i in 21:size(data,2)
    m += r*(data[:,i] - m)
    R += r*((data[:,i] - m)*(data[:,i] - m)'- R)
    end
 R
end
```

Out[31]:
```
cov (generic function with 1 method)
```

In [32]:
```
# final covirance when learning rate is 0.2

R02=cov(0.2,data,m20,R20)
```

Out[32]:
```
48×48 Matrix{Float64}:
  0.0144798     0.00496919    0.00155406   …   0.0101615     0.0155836
  0.00496919    0.02438       0.00513176       0.00174235    0.0200118
  0.00155406    0.00513176    0.0321594       -0.000121456   0.000216138
  0.0164867     0.00808893    0.00540415       0.0152172     0.0229821
  0.00219957    0.000112247  -0.000306744      0.00147445    0.00261856
  0.00232406    0.000459827  -0.000171755  …   0.0023616     0.00259111
  0.00327683    0.00196398    0.000658674      0.00253239    0.00431225
  0.00335097    5.02951e-5   -3.55003e-5       0.00696323    0.00289896
  0.0318408     0.0252108     0.0188912        0.03042       0.0457923
 -0.0055982     0.0483124     0.0139675       -0.0116297     0.0446817
  0.000309761   0.0147901     0.00803378   …  -0.0128241     0.00277311
  0.0105606     0.0484229    -0.0139929        0.00326553    0.0659034
 -0.00816391   -0.00556985    0.0205251       -0.0112647    -0.0118864
  ⋮                                        ⋱
 -0.000619207   0.0135199     0.00633714      -0.00726267   -0.00501117
 -0.00471908    0.000541327  -0.0035055        0.00101197    0.0118277
  0.0151089    -0.00203439   -0.00722105       0.0157161     0.0155152
  0.0116323     0.0215365    -0.0185524        0.0118178     0.0295694
 -0.0185904     0.0281263     0.0331084   …  -0.0268011     0.0106457
 -5.50762e-5    0.00186327    0.012745        -0.00527287    0.000573681
  0.000752009   0.0168723     0.00418376       0.00201646    0.0117544
 -0.00358815    0.00304393   -0.0117016        0.00514586    0.0120877
 -0.00558003    0.00899411   -0.0149811        0.00360264   -0.00193632
  0.00536672   -0.00425574   -0.00499244   …   0.0298784     0.0147992
  0.0101615     0.00174235   -0.000121456      0.025397      0.0163762
  0.0155836     0.0200118     0.000216138      0.0163762     0.0394964
```

In [33]:
```
# final covirance when learning rate is 0.05

R005=cov(0.05,data,m20,R20)
```

Out[33]:
```
48×48 Matrix{Float64}:
  0.0211036    0.0130787    0.00737694  …   0.011059      0.0172703
  0.0130787    0.0390991    0.0107267       0.00700877    0.0128052
  0.00737694   0.0107267    0.0246641       0.00268474    0.00406073
  0.0130524    0.0112224    0.00848717      0.0096416     0.0110715
  0.00608447   0.00610336   0.00440084      0.00248626    0.00447987
  0.00535877   0.00484338   0.00303858  …   0.00466404    0.00062117
  0.00767272   0.00843898   0.00670547      0.00490346    0.0062188
  0.00687959   0.0043543    0.00465572      0.0109155    -0.000719911
  0.0213544    0.0232187    0.0172061       0.0101482     0.0170315
  0.00927879   0.0349653    0.015678       -0.00842967    0.0399812
  0.00815138   0.0203188    0.0101851   …  -0.00794548    0.00543471
  0.0102704    0.0398134    0.00152834     -0.00223324    0.0298965
 -0.00108711   0.00701766   0.0109874      -0.0104669    -0.0133085
  ⋮                                   ⋱
  0.00659052   0.0256995    0.00810418     -0.0040313    -0.0037103
  0.00712325   0.00709869   0.00193447      0.000826666   0.0294161
  0.008536    -0.00124222   0.00197916      0.0221628     0.0141187
  0.0124821    0.0251705   -0.0041585       0.0309346     0.0106407
  0.00666      0.0296225    0.0268511   …  -0.0018182     0.0171142
  0.0102132    0.0145579    0.0130008       0.0159114     0.00835068
  0.00743082   0.0230107    0.00814684      0.00942873    0.0118673
  0.00267702   0.00974033  -0.00197432      0.0264669     0.0290943
 -0.00101944   0.00432572  -0.0108096       0.0180781     0.0174737
  0.00545413  -0.00140472  -0.00156327  …   0.0608973     0.0189428
  0.011059     0.00700877   0.00268474      0.0669271     0.0203538
  0.0172703    0.0128052    0.00406073      0.0203538     0.0662483
```

In [34]:
```
# final covirance when learning rate is 0.01

R001=cov(0.01,data,m20,R20)
```

Out[34]:

```
48×48 Matrix{Float64}:
  0.220656    0.0552747  0.0229908    0.026041   …   0.0155015    0.0271636
  0.0552747   0.0807498  0.0355246    0.0351969       0.0280079    0.0151384
  0.0229908   0.0355246  0.0703572    0.0323688       0.0423762    0.000191864
  0.026041    0.0351969  0.0323688    0.0625352       0.0347058    0.0205854
  0.0177512   0.0334154  0.0271333    0.0425022       0.0668894    0.0706455
  0.015556    0.0310717  0.0152576    0.0306443   …  -0.00271831  -0.0127295
  0.0536489   0.0349568  0.0298565    0.0353917       0.0228029    0.029559
  0.0255586   0.0315564  0.0101556    0.017173        0.0382667    0.0364416
  0.059334    0.0426185  0.0328386    0.044004        0.0335471    0.0153038
  0.0460614   0.0583495  0.0660554    0.0580001       0.078863     0.0394923
  0.0512292   0.0464025  0.0273122    0.0389758   …   0.0184248    0.015196
  0.231564    0.109944   0.0639433    0.074821        0.0537504    0.0359316
  0.170853    0.0799687  0.0425526    0.0519133       0.161236     0.209996
  ⋮                                                  ⋱
  0.107013    0.083162   0.0460491    0.0406797       0.120066     0.0993186
  0.154131    0.0414977  0.032444     0.042395        0.0651699    0.103389
  0.201888    0.0872162  0.0556799    0.080234        0.0347591    0.0855868
  0.119688    0.0715941  0.0111457    0.0340288       0.0444484    0.115662
  0.219195    0.0914553  0.0879055    0.0632093   …   0.108624     0.0731481
  0.26869     0.106702   0.0811423    0.0721067       0.17546      0.167111
  0.116177    0.0912471  0.0717033    0.0998747       0.208178     0.121979
  0.0385452   0.04905    0.0307517    0.0940466       0.158689     0.136881
  0.0218776   0.0159534  0.00973112   0.0308499       0.0165303    0.0384486
 -0.00757028  0.0359286  0.0516558    0.017959    …   0.149794     0.117659
  0.0155015   0.0280079  0.0423762    0.0347058       0.189791     0.111631
  0.0271636   0.0151384  0.000191864  0.0205854       0.111631     0.256212
```

In [35]:

```
# final covirance when learning rate is 0.001

R0001=cov(0.001,data,m20,R20)
```

Out[35]:
```
48×48 Matrix{Float64}:
  1.38221    0.28807     0.0839928   …  -0.17387    -0.0200485   0.0589688
  0.28807    0.333729    0.168059        0.205504    0.123875    0.0303373
  0.0839928  0.168059    0.351081        0.331471    0.267143   -0.0740764
  0.116121   0.177828    0.16264         0.0920143   0.195499    0.064283
  0.0669477  0.174955    0.138138        0.207797    0.461577    0.476309
  0.0610842  0.169408    0.0558471   …   0.623443   -0.0541586  -0.116541
  0.32734    0.173382    0.133253        0.0843839   0.131162    0.142321
  0.134846   0.165741    0.0139565       0.192982    0.231192    0.238053
  0.343884   0.210408    0.155817        0.0405192   0.217526    0.0805911
  0.274922   0.313797    0.391414        0.753742    0.577572    0.0504245
  0.326453   0.229005    0.112509    …   0.0158263   0.143045    0.0329858
  1.6612     0.620159    0.378377       -0.10881     0.356648    0.179323
  1.19533    0.43916     0.164275        2.39253     1.14452     1.5353
  ⋮                                  ⋱   ⋮
  0.665808   0.470548    0.265776        0.66974     0.764935    0.650691
  1.0232     0.141676    0.0891913       1.09307     0.317485    0.379459
  1.43237    0.575112    0.31086         1.08767     0.0501324   0.392923
  0.74935    0.360836    0.00335038      0.726766    0.0882956   0.743532
  1.49591    0.496522    0.510005    …   0.0614968   0.650956    0.429898
  1.82313    0.596559    0.451597        0.630859    1.08879     1.08443
  0.711453   0.44586     0.377202        1.01929     1.34016     0.668434
  0.149958   0.228694    0.118616        2.00447     0.926827    0.747338
  0.0312515  0.0606107   0.0514601       2.2486     -0.0663765   0.0705664
 -0.17387    0.205504    0.331471    …   3.60489     0.773406    0.669934
 -0.0200485  0.123875    0.267143        0.773406    0.966628    0.597186
  0.0589688  0.0303373  -0.0740764       0.669934    0.597186    1.30384
```

In [36]:
```
# final covirance when learning rate is 0.0001

R00001=cov(0.0001,data,m20,R20)
```

Out[36]:
```
48×48 Matrix{Float64}:
  1.67589    0.347203    0.0983187    …  -0.219998   -0.0305857   0.0660622
  0.347203   0.399316    0.201633          0.247215    0.148083    0.0337041
  0.0983187  0.201633    0.421978          0.39995     0.32297    -0.0940117
  0.138915   0.214392    0.1956            0.109361    0.235963    0.0748738
  0.0778924  0.21072     0.165429          0.249112    0.560594    0.578219
  0.0716161  0.204443    0.0653743    …    0.757694   -0.068499   -0.144169
  0.396685   0.208965    0.159562          0.100056    0.158129    0.170898
  0.162014   0.19991     0.0147002         0.232773    0.279737    0.288517
  0.41681    0.253897    0.187397          0.0479835   0.264063    0.097012
  0.333191   0.379858    0.474326          0.917908    0.703535    0.0543822
  0.398518   0.277113    0.135553     …    0.0238226   0.176259    0.0391504
  2.02638    0.751537    0.459198         -0.130755    0.434379    0.217702
  1.45529    0.530586    0.194981          2.91788     1.39328     1.87136
  ⋮                                   ⋱   ⋮
  0.806316   0.569581    0.320481          0.811846    0.927637    0.789615
  1.24401    0.166905    0.102643          1.32607     0.380543    0.450255
  1.74621    0.699102    0.376022          1.32373     0.0553162   0.472082
  0.908016   0.434378   -0.000388999       0.878861    0.0992218   0.901956
  1.81794    0.59884     0.615673     …    0.0623158   0.785302    0.518608
  2.21274    0.719818    0.5418            0.750862    1.31639     1.31434
  0.857828   0.5347      0.451481          1.22683     1.6236      0.803995
  0.171692   0.272919    0.136949          2.42507     1.11773     0.9002
  0.027926   0.0700511   0.0583363         2.72472    -0.090863    0.0758269
 -0.219998   0.247215    0.39995      …    4.3736      0.931349    0.808502
 -0.0305857  0.148083    0.32297           0.931349    1.16518     0.719851
  0.0660622  0.0337041  -0.0940117         0.808502    0.719851    1.5718
```

In [37]:
```
# to use the Power method to get (first) eigenvector w, firstly need to generate random first

using Random
Random.seed!(1)
w = rand(48)
```

Out[37]:
```
48-element Vector{Float64}:
 0.0491718221481211
 0.11907881640750706
 0.3932710232252806
 0.024094310524527707
 0.6918572875342215
 0.7675180540873912
 0.08725304891274233
 0.8557176841095734
 0.8403841370820818
 0.8907696748195567
 0.138227024723224
 0.3477368477058109
 0.19852090963358837
 ⋮
 0.2794235399691639
 0.6820050160336559
 0.8050245353412276
 0.21100502806555133
 0.2671882107649258
 0.3438753675415538
 0.4408314844038338
 0.11604055879014974
 0.5475146896778691
 0.5892167646241558
 0.1993200843257663
 0.3641283484132992
```

In [38]:
```
# function to use power method to calculate the eigenvector w

function vectorw(R,w)
    for i in 1:10
    w = R * w
    w /= sqrt(sum(abs2, w))
    end
  w
end
```

Out[38]:
```
vectorw (generic function with 1 method)
```

In [39]:
```
# eigenvector w when learning rate is 0.2

w02=vectorw(R02,w)
```

Out[39]:
```
48-element Vector{Float64}:
  0.08415435810436389
  0.02831201520008852
 -0.019334285728736383
  0.11419897717550741
  0.00865448149281041
  0.008770494484486623
  0.012679360445251297
  0.0186373161862267
  0.2653793389708861
 -0.16480168409226362
 -0.06896190831627555
 -0.018733044978583555
 -0.0804563181766276
  ⋮
  0.07052421830278764
 -0.05094625937161482
  0.09044538585801669
  0.0842708168875579
 -0.32186770997972897
 -0.09343983749532568
  0.04590741386876707
 -0.00023195765433091514
  0.07066397811841088
  0.07663929100875991
  0.08807507052881745
  0.0924666997187018
```

In [40]:
```
# When learning rate is 0.2, calculate the correlation between w and the first eigenvector e1 (biggest/last one) obtain

d02=dot(w02, evecs[:, end])
```

Out[40]:
```
-0.3251190747457914
```

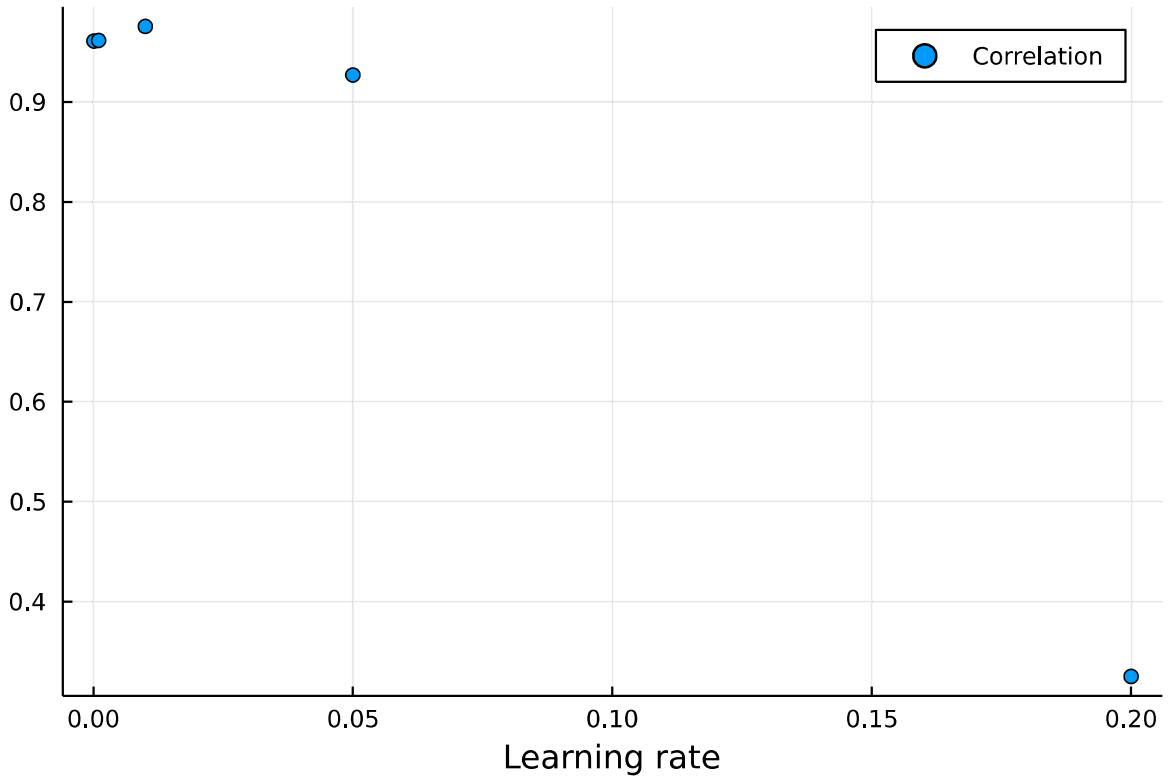
In [41]:
```
# same as the above codes, get the eigenvector when learning rate is 0.05, 0.01, 0.001, 0.0001 respectively, and calcul

w005=vectorw(R005,w)
d005=dot(w005, evecs[:, end])

w001=vectorw(R001,w)
d001=dot(w001, evecs[:, end])

w0001=vectorw(R0001,w)
d0001=dot(w0001, evecs[:, end])
```

```
w00001=vectorw(R00001,w)
d00001=dot(w00001, evecs[:, end])

println("when learning rate = 0.2, the correlation is ",abs(d02))
println("when learning rate = 0.05, the correlation is ",abs(d005))
println("when learning rate = 0.01, the correlation is ",abs(d001))
println("when learning rate = 0.001, the correlation is ",abs(d0001))
println("when learning rate = 0.0001, the correlation is ",abs(d00001))
```

```
when learning rate = 0.2, the correlation is 0.3251190747457914
when learning rate = 0.05, the correlation is 0.9271017127432803
when learning rate = 0.01, the correlation is 0.9757769037083995
when learning rate = 0.001, the correlation is 0.9617270730746834
when learning rate = 0.0001, the correlation is 0.9611347362340328
```

In [42]:
```
# plot the relationship between learning rate and correlation

scatter([0.0001,0.001,0.01,0.05,0.2], [abs(d00001),abs(d0001),abs(d001),abs(d005),abs(d02)], label="Correlation", xlabe
```

Out[42]:



In [ ]:
```
scatter([0.0001,0.001,0.01,0.05,0.2], [abs(d00001),abs(d0001),abs(d001),abs(d005),abs(d02)], label="Correlation", xlabe
```

In [ ]:
```
# From the above plot we can see that when learning rate = 0.01,the correlation is 0.97578, means it is the best setting
# When learning rate = 0.2, it has a very small correlation, that means when learning rate is big (e.g. 0.2), the eigen
# so when learning rate is big, the calculation tends to converge very quickly and cannot get to the point that we need
# In summary, when learning rate is 0.01, it has the best result for all the experiments that I have done so far.
```

JD: good effort in visualizing the result. Note x-axis should be on log-scale so as to separate the small values.

In [ ]:
```

```

In [ ]:
```
# 5. Outlier test using online PCA
```

In [43]:
```
# get the eigenvector w of the first 20 data vectors by using power method

w20=vectorw(R20,w)
```

Out[43]:
```
48-element Vector{Float64}:
 0.10107035769615955
 0.048510664483675206
 0.03536943462845019
 0.04767459474394368
 0.07235538564541115
 0.05091074234232311
 0.062401025773508416
 0.039818887915165135
 0.0707076288455528
 0.08076193295072764
 0.050966026406571115
 0.20177373536663432
 0.29682320654489347
 ⋮
 0.20405046344526742
 0.1595399481926389
 0.23576529747436287
 0.12821284857031842
 0.22905156686044206
 0.319770006604531
 0.26426914545646685
 0.26237217094269616
 0.12735870957713413
 0.11067235875993532
 0.0849606282792476
 0.0790579091386057
```

In [44]:
```
# scaning through all data vectors, for each household, get new estimates m1 and R1 by using online PCA method, then ge

function evw1(r)

     m1 = m20 + r*(data[:,1] - m20) # use online PCA method to calculate the mean for the first household
     R1 = R20 + r*((data[:,1] - m1)*(data[:,1] - m1)'- R20) # use online PCA method to calculate the covariance for the
     w1 = vectorw(R1,w) # use power method to calculate the eigenvector for the first household

    for i in 2:size(data,2)

         m1 = m20 + r*(data[:,i] - m20) # use online PCA method to calculate the mean for each household
         R1 = R20 + r*((data[:,i] - m1)*(data[:,i] - m1)'- R20) # use online PCA method to calculate the covariance for
         w1 = hcat(w1, vectorw(R1,w)) # use power method to calculate the eigenvector for each household, and also comb
    end
w1
end
```

Out[44]:
```
evw1 (generic function with 1 method)
```

In [45]:
```
# function to get all the outlier correlations, outliers vector eigenvector vs. the first 20 data vectors eigenvector

function corout(w1)

a=[6,33,40,50,56,157,183,199] # The known outliers

corout=dot(w1[:, a[1]], w20) # get the correlations between the 6th column vector in w1 vs. w vector that produced by f

#repeat the same process, to get the correlations for all the outliers

for i in 2:length(a)
    corout=vcat(corout,dot(w1[:, a[i]], w20)) # combine all the correlations for outliers together
    end

corout
end
```

Out[45]:
```
corout (generic function with 1 method)
```

In [46]:
```
# function to get the threshold of outliers, that is the max correlation for the already known outliers,anything less t

function threshold(corout)
T = findmax(corout)[1]
end
```

Out[46]:
```
threshold (generic function with 1 method)
```

In [47]:
```
# function to get all the data vector correlations, each vector eigenvector vs. the first 20 data vectors eigenvector

function corall(w1)

corall = dot(w1[:, 1], w20) # get the correlations between the 1st column vector in w1 vs. w vector that produced by fi

#repeat the same process, to get the correlations for all the data

for i in 2:size(data,2)

corall = vcat(corall,dot(w1[:, i], w20)) # combine all the correlations for all the data together
end
```

```
    corall
end
```

Out[47]: corall (generic function with 1 method)

In [48]:
```
# function to get outlier, which is true positives. Which are the ones that the correlations are smaller than or equal

function tp(corall,T)
    tp=findall(corall -> corall <= T,corall) # in all the correlations for all the data, find the ones that are smaller
end
```

Out[48]: tp (generic function with 1 method)

In [49]:
```
# function to get the non-outlier, which is false positives. That are the ones that the correlation is bigger than the

function fp(corvalall,T)
fp=findall(corvalall -> corvalall > T,corvalall) # in all the correlations for all the data, find the ones that are big
end
```

Out[49]: fp (generic function with 1 method)

In [50]:
```
# combine all the functions above to get a function to calculate the threshold, outliers and non-outliers

function outlier(r)
    w1=evw1(r)
    corout1=corout(w1)
    T=threshold(corout1)
    corall1=corall(w1)

    tp1=tp(corall1,T)
    fp1=fp(corall1,T)

println("Threshold is ",T)

println("Outliers are ",tp1)

println("Non-outliers are ",fp1)


end
```

Out[50]: outlier (generic function with 1 method)

In [51]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.0001

out00001=outlier(0.0001)
```

```
Threshold is 0.999999999970851
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 80, 91, 96, 99, 107, 110, 114, 1
15, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 199, 201, 202, 213, 223, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 10
9, 111, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 1
42, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170,
171, 173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 200,
203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 219, 220, 221, 222, 224, 225, 226, 227, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241]
```

In [52]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.001

outlier(0.001)
```

```
Threshold is 0.9999999970913889
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 80, 91, 96, 99, 107, 110, 114, 1
15, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 199, 201, 202, 213, 223, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 10
9, 111, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 1
42, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170,
171, 173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 200,
203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 219, 220, 221, 222, 224, 225, 226, 227, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241]
```

In [53]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.01

outlier(0.01)
```

```
Threshold is 0.9999997153827496
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 80, 91, 96, 99, 107, 110, 114, 1
15, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 199, 201, 202, 213, 223, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 10
9, 111, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 1
42, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170,
171, 173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 200,
203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 219, 220, 221, 222, 224, 225, 226, 227, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241]
```

In [54]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.05

outlier(0.05)
```

```
Threshold is 0.9999935465378407
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 72, 80, 91, 96, 99, 107, 110, 11
4, 115, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 190, 199, 201, 202, 213, 219, 2
23, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 73, 74, 75, 76, 77,
78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 109, 11
1, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 142, 1
43, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170, 171,
173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 191, 192, 193, 194, 195, 196, 197, 198, 200, 203, 204,
205, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 220, 221, 222, 224, 225, 226, 227, 229, 230, 231, 232,
233, 234, 235, 236, 237, 238, 239, 240, 241]
```

In [55]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.1

outlier(0.1)
```

```
Threshold is 0.9999772234774909
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 72, 80, 91, 96, 99, 107, 110, 11
4, 115, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 190, 199, 201, 202, 213, 219, 2
23, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 73, 74, 75, 76, 77,
78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 109, 11
1, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 142, 1
43, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170, 171,
173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 191, 192, 193, 194, 195, 196, 197, 198, 200, 203, 204,
205, 206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 220, 221, 222, 224, 225, 226, 227, 229, 230, 231, 232,
233, 234, 235, 236, 237, 238, 239, 240, 241]
```

In [56]:
```
# get the threshold, outliers, non-outliers, when learning rate is 0.2

outlier(0.2)
```

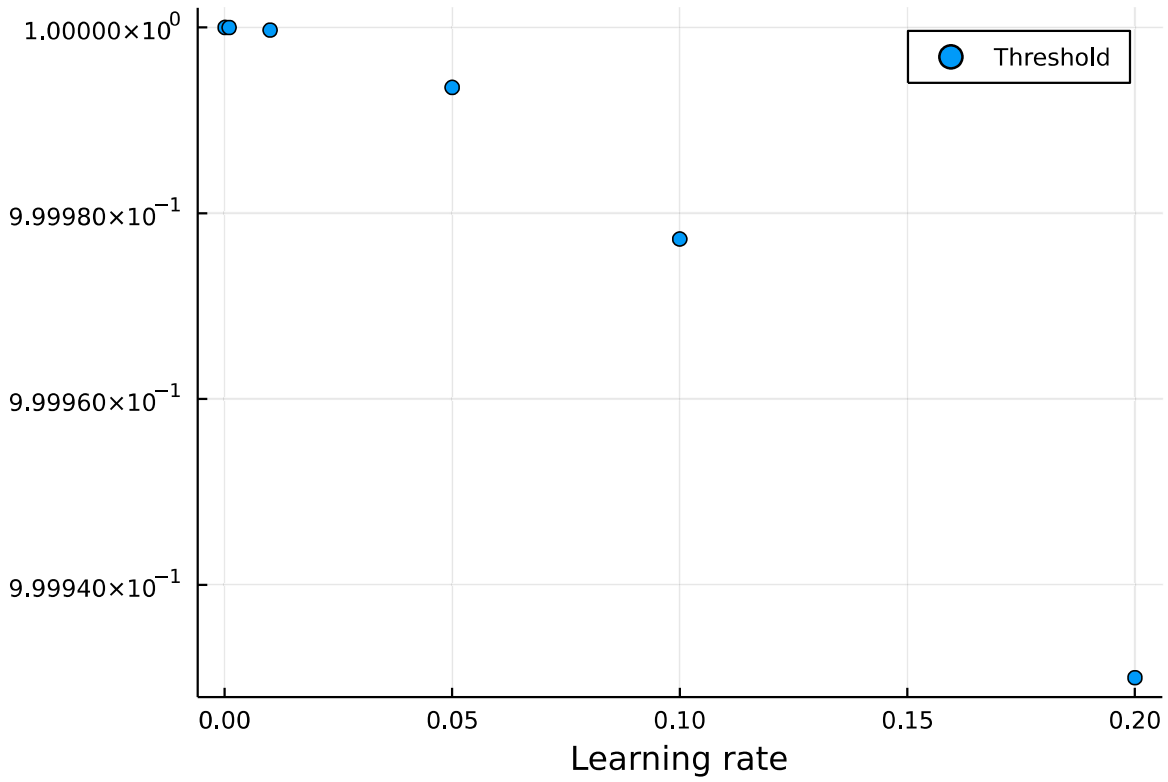
```
Threshold is 0.9999299928974149
Outliers are [2, 6, 8, 14, 15, 16, 20, 32, 33, 36, 40, 49, 50, 56, 60, 65, 66, 68, 69, 72, 80, 91, 96, 99, 107, 110, 11
4, 115, 116, 117, 118, 127, 129, 138, 145, 154, 157, 159, 164, 172, 175, 183, 184, 186, 190, 194, 199, 201, 202, 213, 2
19, 223, 228]
Non-outliers are [1, 3, 4, 5, 7, 9, 10, 11, 12, 13, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 37,
38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 51, 52, 53, 54, 55, 57, 58, 59, 61, 62, 63, 64, 67, 70, 71, 73, 74, 75, 76, 77,
78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 100, 101, 102, 103, 104, 105, 106, 108, 109, 11
1, 112, 113, 119, 120, 121, 122, 123, 124, 125, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 142, 1
43, 144, 146, 147, 148, 149, 150, 151, 152, 153, 155, 156, 158, 160, 161, 162, 163, 165, 166, 167, 168, 169, 170, 171,
173, 174, 176, 177, 178, 179, 180, 181, 182, 185, 187, 188, 189, 191, 192, 193, 195, 196, 197, 198, 200, 203, 204, 205,
206, 207, 208, 209, 210, 211, 212, 214, 215, 216, 217, 218, 220, 221, 222, 224, 225, 226, 227, 229, 230, 231, 232, 233,
234, 235, 236, 237, 238, 239, 240, 241]
```

In [57]:
```
#function to get threshold

function thresh(r)
    w1=evw1(r)
    corout1=corout(w1)
    T=threshold(corout1)
end
```

Out[57]:
```
thresh (generic function with 1 method)
```

In [58]:
```
scatter([0.0001,0.001,0.01,0.05,0.1,0.2], [thresh(0.0001),thresh(0.001),thresh(0.01),thresh(0.05),thresh(0.1),thresh(0.
```

Out[58]:



In [59]:
```
# The above plot shows that the threshold for outliers decreases when the learning rate increases.
# This means when the learning rate increases, threshold decreases, tolerance level increases, therefore there would be
# The results also confirm my conclusion too.
```

Good observation on the threshold's trend. Thresholds' getting lower doesn't mean that you'll have fewer outliers - it may also have something to do with the quality of the eigenvector for testing.

In [ ]: