Multi-Arm Bandits

A Multi-Arm Bandit is a concept from Reinforcement Learning which is useful for decision making under uncertainty. Take a situation where you have limited time or resources to test multiple drugs, perhaps the COVID vaccines for example. While you could use the standard approach of a clinical trial with randomization, a balanced design, and performing AB tests, this results in many participants not recieving the best treatment, as some of them will be allocated to inferior choices. Ideally, we would like to do both: 1) Show a given treatment is the best choice and 2) Have most people end up benefitting from it.

Using a strategy known as ϵ-greedy Q learning, the Multi Arm bandit can help accomplish this task in an online manner where the treatments are allocated on the spot. The arm refers to different treatment groups that the bandit can select from. In the beginning, each treatment arm is assigned the same value of $Q_{tmt} = Q_0$. Some small amount of noise can be added to break ties. At each step, $(1 - \epsilon)$ of the time the bandit randomly selects from the treatment with the highest Q_{tmt} and updates its corresponding value. ϵ of the time, the bandit will select a random treatment instead of the one with the highest Q. The update rule is as follows:

$$Q_{tmt,t} = Q_{tmt,t-1} + \frac{R - Q_{tmt,t-1}}{N_{tmt,t}}$$

where R is the reward, which is usually just the response Y corresponding to the treatment given. N_{t+1} corresponds to the sample size in the given treatment group after the sample from the group is randomly selected. The bandit will be trying to go for the treatment which maximizes the reward, and so care should be taken that the response is transformed or sign-flipped if necessary so that higher is better. The above update rule can be shown to correspond to the calculation of the sample mean in an online manner if $Q_0 = 0$. However, we can modify the updates by adding an explicit learning rate parameter η:

$$Q_{tmt,t+1} = Q_{tmt,t} + \eta(R - Q_{tmt,t})$$

This can help improve learning. In this notebook, we will simulate data from 5 groups and examine the impact of setting different parameters for Q_0 and η.

The source code for the functions used in this demo Pluto notebook can be found in the **MultiArmBandit.jl** module in this repo.

```
begin

include("MultiArmBandit.jl")

using .MultiArmBandit
using DataFrames
using StatsBase, Statistics
using Distributions
using Gadfly
import Random

end
```

Simulate Data

Below, we simulate n=10000 points from 5 groups each A, B, C, D, E from a normal distibution with mean $\mu = (-0.3, 0.0, 0.5, 0.8, 1.1)$ and $\sigma = (1, 0.5, 0.3, 0.3, 0.5)$ respectively. The bandit will select points from this simulated dataset. Note that in the ideal situation the ground truth in this simulated example is:

$$E > D > C > B > A$$

Ideally, the bandit will come to this same conclusion and choose E most of the time. However, the different SDs for the groups can complicate this choice, as the bandit is greedy and thus converging to E in a finite number of iterations (in the real world, this would correspond to resources or number of patients in the trial) can be more complicated. We can track the bandit's history and plot it over time to examine when convergence occurs.

The sample mean/SD of each Arm are shown below. The bandit will be selecting samples from each group in an online manner, and so at each step it does NOT have access to every single data point. Instead it must figure out in the finite number of iterations which group is the best

	Arm	nrow	meanY	sdY
1	"A"	1000	-0.340486	1.03546
2	"B"	1000	0.0161643	0.494901
3	"C"	1000	0.497106	0.304793
4	"D"	1000	0.803976	0.296274
5	"E"	1000	1.09454	0.505926

```
begin

n = 1000
Random.seed!(101)
A= rand(Normal(-0.3,1),n)
B= rand(Normal(0,0.5),n)
C = rand(Normal(0.5,0.3),n)
D = rand(Normal(0.8,0.3),n)
E = rand(Normal(1.1,0.5),n)

simdata = DataFrame(Arm =
vcat(fill("A",n),fill("B",n),fill("C",n),fill("D",n),fill("E",n)),
                    Y = vcat(A,B,C,D,E))

summary_simdata =
combine(groupby(simdata,:Arm),nrow,:Y=>mean=>:meanY,:Y=>std=>:sdY)
summary_simdata
end
```

Create Bandit: $Q_0 = 0, \eta = \frac{1}{N_{tmt,t}}, \epsilon = 0.1$

Below we instantiate the first bandit with all default settings, and allow it to explore 10% of the time while 90% of the time it selects the current best group in a greedy manner.

```
band1 =
  Bandit(Dict((Arm = "B") ⇒ Dict(:n ⇒ 0.0,  :Q ⇒ 6.95901e-11), (Arm = "A") ⇒ Dict(:
```

```
band1=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,track_hist=true)
```

```
  Bandit(Dict((Arm = "B") ⇒ Dict(:n ⇒ 11.0,  :Q ⇒ -0.0573318), (Arm = "A") ⇒ Dict(:
```

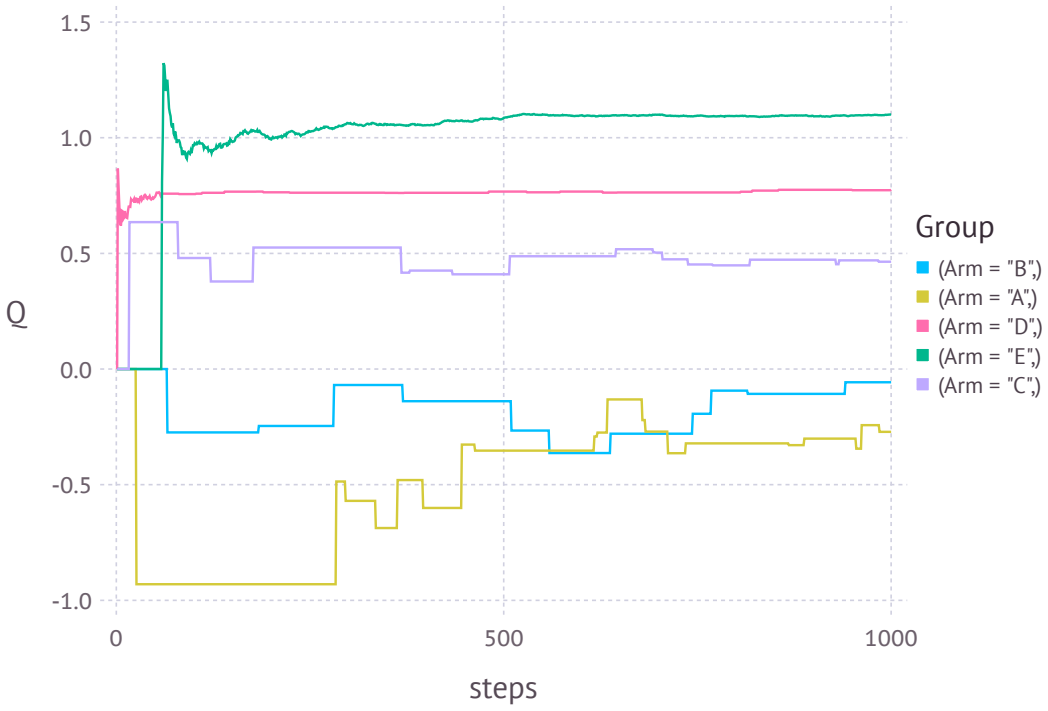
```
begin
Random.seed!(1001)
MultiArmBandit.Train!(band1,999,ϵ = 0.1)
end
```

band1_df =

	steps	Group	Q	n_Arm
1	1	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
2	2	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
3	3	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
4	4	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
5	5	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
6	6	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
7	7	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
8	8	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
9	9	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0
10	10	CategoricalValue{String, UInt32} "(Arr	6.95901e-11	0.0

more

```
band1_df = MultiArmBandit.GetHistoryDF(band1)
```




```
plot(band1_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

band1_summary =

	Group	n	Q
1	(Arm = "B")	11.0	-0.0573318
2	(Arm = "A")	20.0	-0.271369
3	(Arm = "D")	78.0	0.775631
4	(Arm = "E")	872.0	1.10056
5	(Arm = "C")	18.0	0.464202

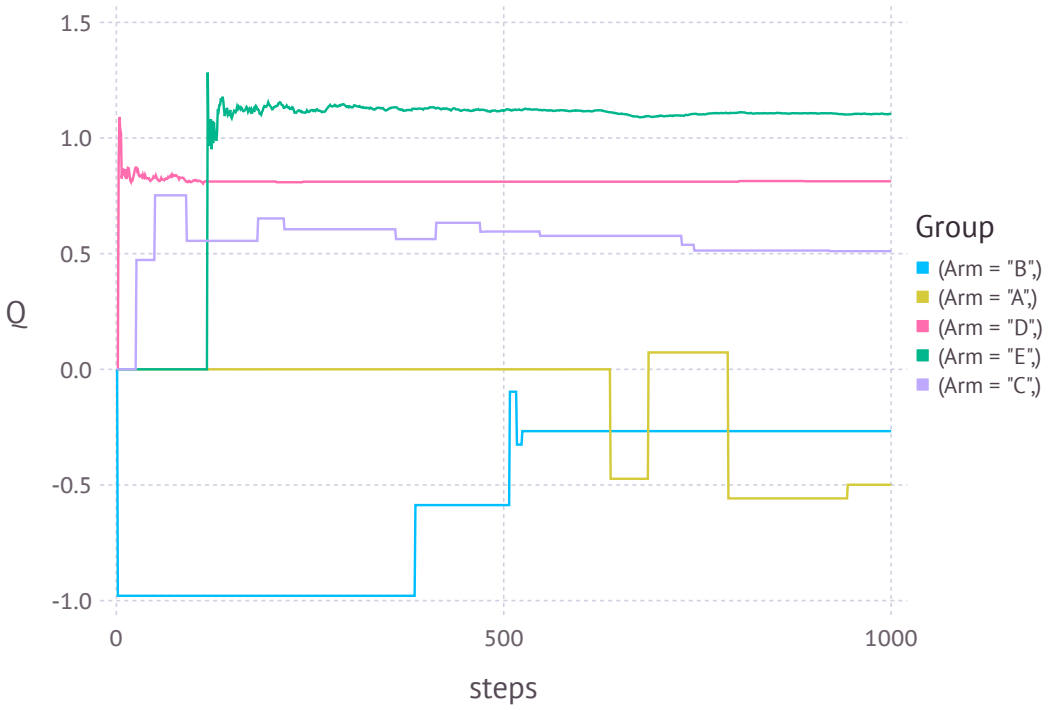
```
band1_summary = summary(band1)
```

Based on the above figure, the correct ordering is found after around 250 steps. For the first few steps, the bandit picks Group D which is the 2nd best but eventually finds Group C as a result of the exploration. Lets try to see what happens if we lower the bandit's exploration by setting $\epsilon = 0.03$. That is, the bandit will select a group randomly (instead of the group it deems the best group at the moment) 3% of the time.

Create Bandit: $Q_0 = 0, \eta = \frac{1}{N_{tmt,t}}, \epsilon = 0.03$

	Group	n	Q
1	(Arm = "B")	5.0	-0.267508
2	(Arm = "A")	4.0	-0.49938
3	(Arm = "D")	116.0	0.813019
4	(Arm = "E")	861.0	1.1042
5	(Arm = "C")	13.0	0.510682

```
begin
band2=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,track_hist=true)
Random.seed!(1102)
MultiArmBandit.Train!(band2,999,ϵ = 0.03)
band2_df = MultiArmBandit.GetHistoryDF(band2)
band2_summary = summary(band2)
end
```



```
plot(band2_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

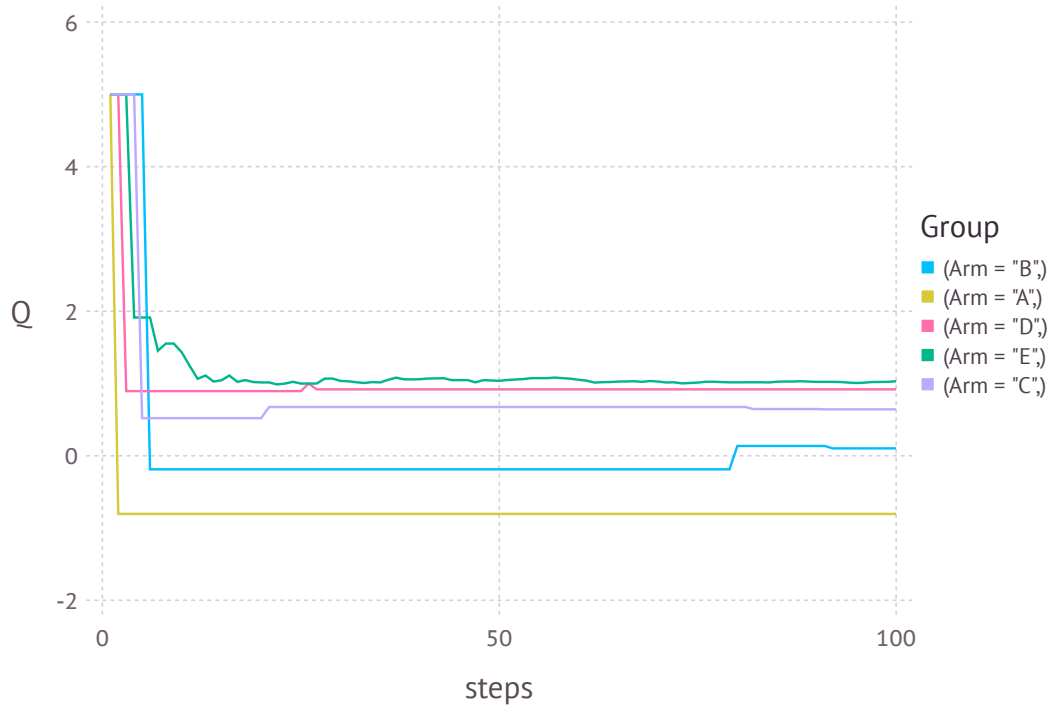
In this situation, it took longer (~100 iterations) for the bandit to arrive at the best group

In the next section, we will change the starting value to be higher at $Q_0 = 5$. This will tend to encourage a lot more early exploration as this value is much higher than what is possible based on sampling.

Create Bandit: $Q_0 = 5, \eta = \frac{1}{N_{tmt,t}}, \epsilon = 0.1$

	Group	n	Q
1	(Arm = "B")	3.0	0.1028
2	(Arm = "A")	1.0	-0.804429
3	(Arm = "D")	3.0	0.918634
4	(Arm = "E")	88.0	1.03159
5	(Arm = "C")	4.0	0.643146

```
begin
band3=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,Q₀=5,track_hist=true)
Random.seed!(901)
MultiArmBandit.Train!(band3,99,ϵ = 0.1)
band3_df = MultiArmBandit.GetHistoryDF(band3)
band3_summary = summary(band3)
end
```




```
plot(band3_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

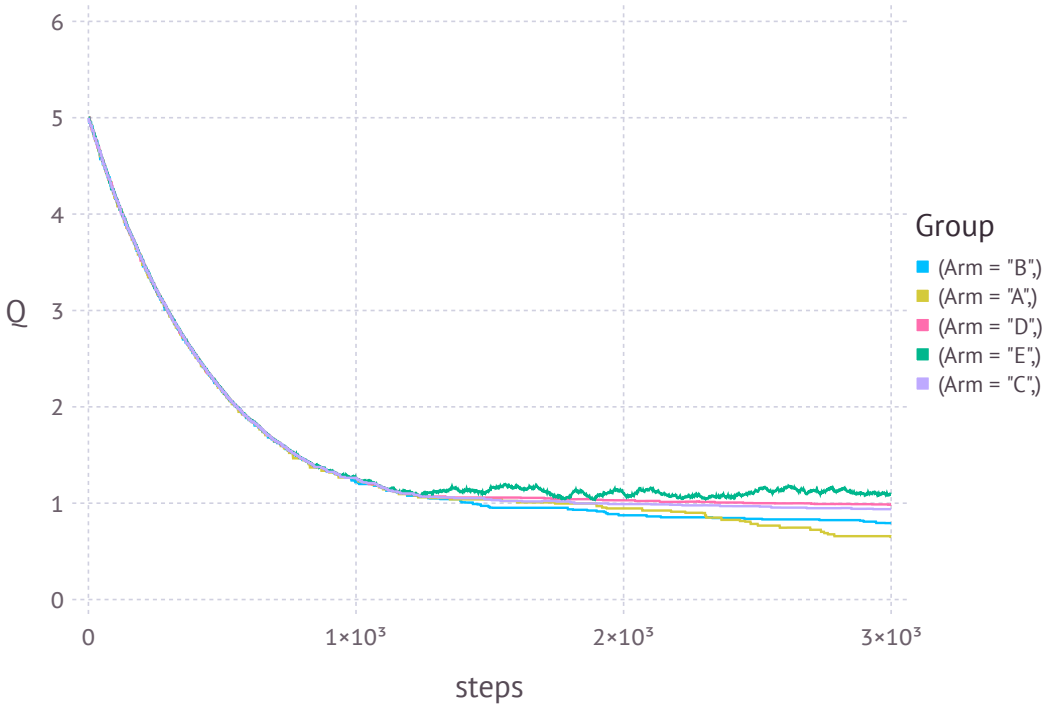
In this case, starting at a value much higher than possible leads to earlier exploration, resulting in the Q value for each group falling rapidly. However, the bandit arrives at the best option E much quicker since it is pulled down less and is essentially guaranteed to be selected since the bandit desperately switches options more in the beginning to try to maintain a high Q value.

In this next part, we keep $Q_0 = 5$ but set the learning rate $\eta = 0.01$

Create Bandit: $Q_0 = 5, \eta = 0.01, \epsilon = 0.1$

	Group	n	Q
1	(Arm = "B")	185.0	0.793885
2	(Arm = "A")	160.0	0.650098
3	(Arm = "D")	308.0	0.988423
4	(Arm = "E")	2113.0	1.09454
5	(Arm = "C")	233.0	0.939054

```
begin
band4=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,Q₀=5,track_hist=true)
Random.seed!(555)
MultiArmBandit.Train!(band4,2999,ϵ = 0.1,η=0.01)
band4_df = MultiArmBandit.GetHistoryDF(band4)
band4_summary = summary(band4)
end
```




```
plot(band4_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

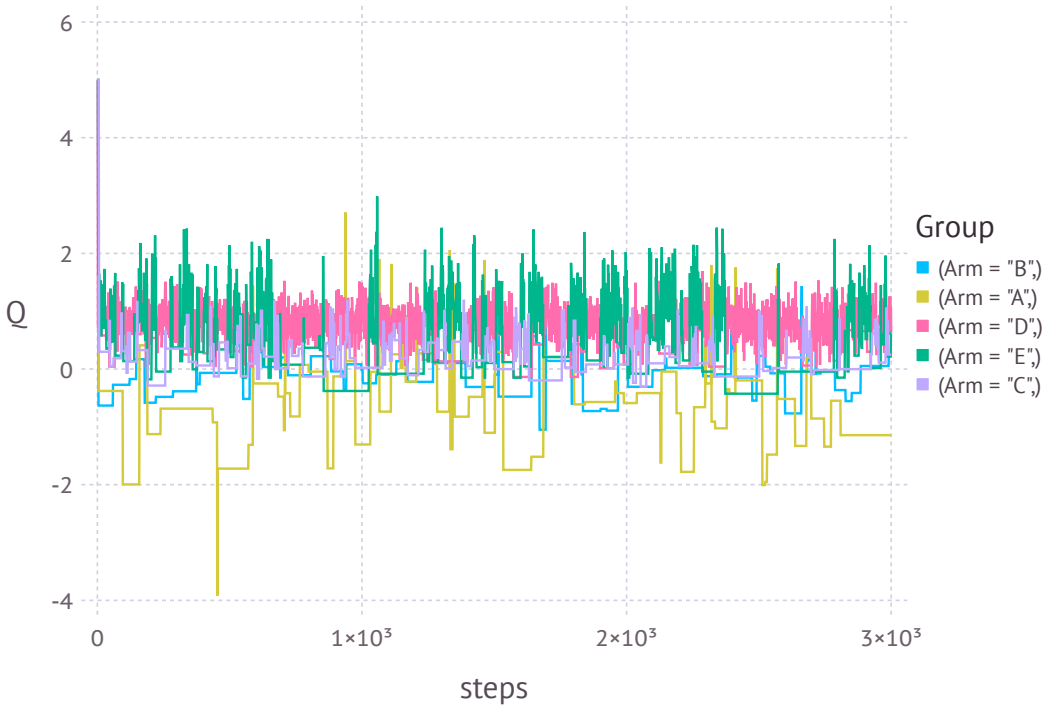
In this case, setting the learning rate to 0.01 leads the bandit to take longer to reach the optimal solution E. Until around 1500 steps, the Q value for all the groups is even.

What happens if we increase the learning rate to 1 (but keep $Q_0 = 5$) ?

Create Bandit: $Q_0 = 5, \eta = 1, \epsilon = 0.1$

	Group	n	Q
1	(Arm = "B")	93.0	0.22619
2	(Arm = "A")	88.0	-1.14306
3	(Arm = "D")	1608.0	0.629405
4	(Arm = "E")	859.0	0.624888
5	(Arm = "C")	351.0	0.249597

```
begin
band5=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,Q₀=5,track_hist=true)
Random.seed!(1556)
MultiArmBandit.Train!(band5,2999,ϵ = 0.1,η=1)
band5_df = MultiArmBandit.GetHistoryDF(band5)
band5_summary = summary(band5)
end
```




```
plot(band5_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

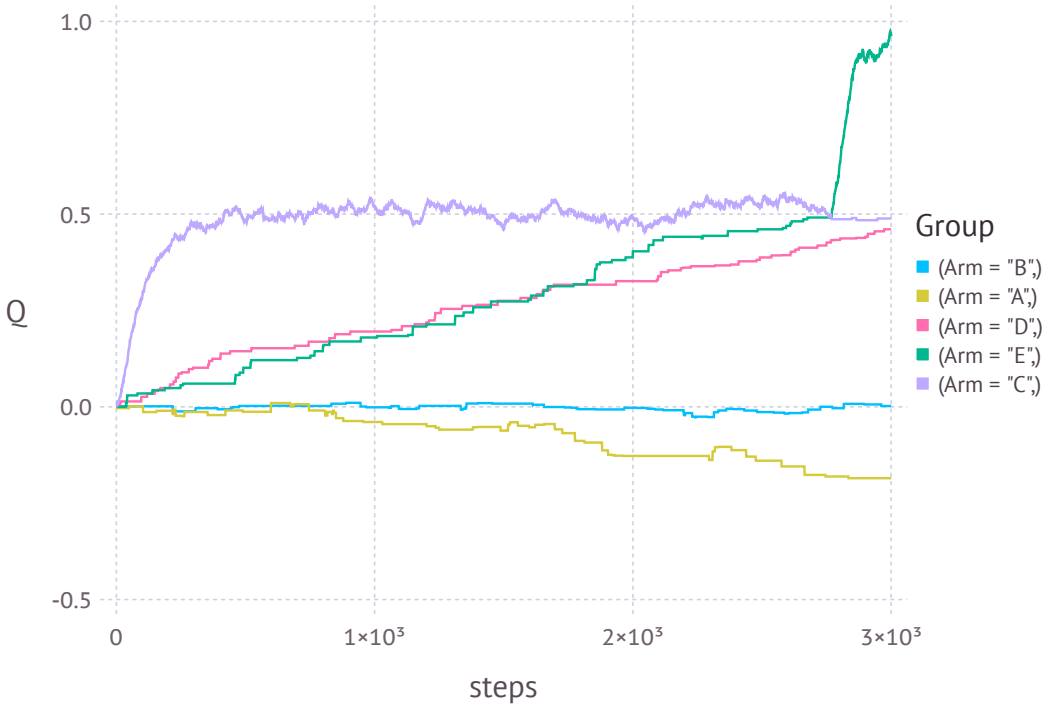
Now the plot is extremely noisy, and the bandit cannot seem to differentiate between Group D and Group E very well. We can see in the summary that the bandit tended to select D the most but in the end the Q values of both D/E are about the same.

In this final parts, we now set $Q_0 = 0$ again and examine the impact of the two learning rates.

Create Bandit: $Q_0 = 0, \eta = 0.01, \epsilon = 0.1$

	Group	n	Q
1	(Arm = "B")	64.0	0.00217161
2	(Arm = "A")	60.0	-0.185236
3	(Arm = "D")	74.0	0.460756
4	(Arm = "E")	273.0	0.961716
5	(Arm = "C")	2528.0	0.488628

```
begin
band6=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,Q₀=0,track_hist=true)
Random.seed!(121)
MultiArmBandit.Train!(band6,2999,ϵ = 0.1,η=0.01)
band6_df = MultiArmBandit.GetHistoryDF(band6)
band6_summary = summary(band6)
end
```




```
plot(band6_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

In this case, starting at $Q_0 = 0$ and setting the learning rate $\eta = 0.01$ leads to the bandit favoring Group C, a suboptimal group, for a long time. It is only after ~2700 iterations that it picks up on

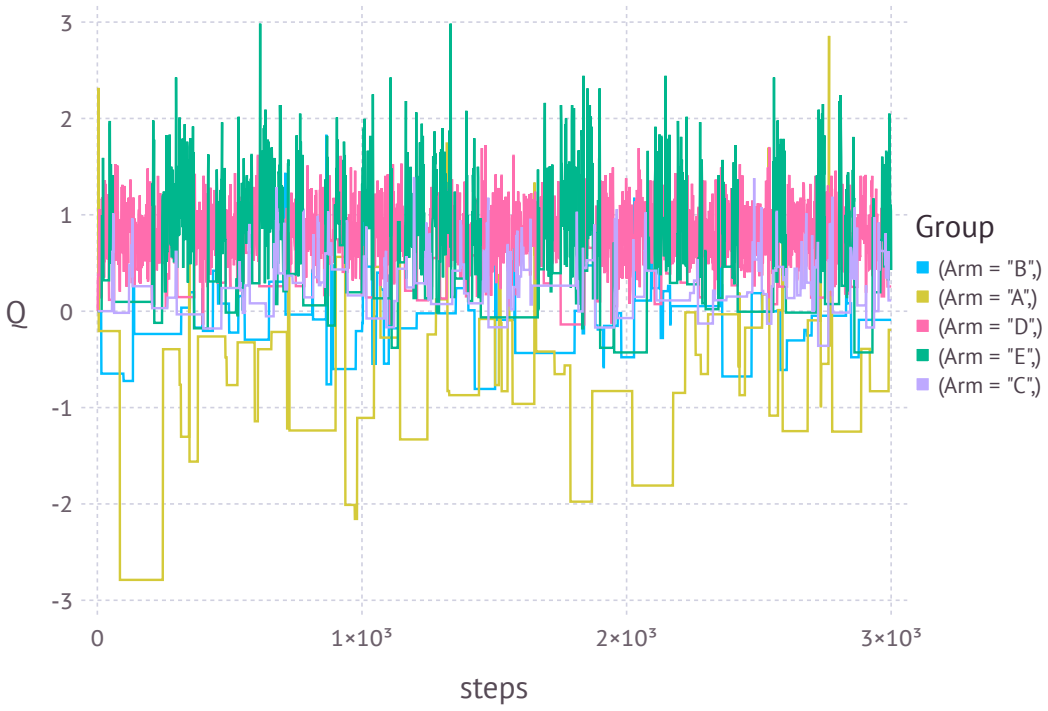
Group E.

What happens if the learning rate is increased? Will we see the same noisy behavior as before when starting from $Q_0 = 0$?

Create Bandit: $Q_0 = 0, \eta = 1, \epsilon = 0.1$

	Group	n	Q
1	(Arm = "B")	79.0	-0.0897333
2	(Arm = "A")	78.0	-0.193803
3	(Arm = "D")	1653.0	0.256227
4	(Arm = "E")	914.0	0.427666
5	(Arm = "C")	275.0	0.111905

```
begin
band7=MultiArmBandit.CreateBandit(simdata,:Arm,:Y,Q₀=0,track_hist=true)
Random.seed!(13)
MultiArmBandit.Train!(band7,2999,ϵ = 0.1,η=1)
band7_df = MultiArmBandit.GetHistoryDF(band7)
band7_summary = summary(band7)
end
```




```
plot(band7_df,x=:steps,y=:Q,color=:Group,Geom.line)
```

The higher learning rate again leads to noise, and in the end, the bandit has trouble differentiating Group D and E just as before.

In summary, lowering the learning rate appears to increase the chance of arriving at a suboptimal solution. Increasing the learning rate too much confuses the Bandit and the Q value curves become extremely noisy.

Starting at a higher Q (with no set learning rate) encourages quicker exploration between the groups