The Problem

From FiveThirtyEight, a problem is presented:

"*Two players go on a hot new game show called “Higher Number Wins.” The two go into separate booths, and each presses a button, and a random number between zero and one appears on a screen. (At this point, neither knows the other’s number, but they do know the numbers are chosen from a standard uniform distribution.) They can choose to keep that first number, or to press the button again to discard the first number and get a second random number, which they must keep. Then, they come out of their booths and see the final number for each player on the wall. The lavish grand prize — a case full of gold bullion — is awarded to the player who kept the higher number. Which number is the optimal cutoff for players to discard their first number and choose another? Put another way, within which range should they choose to keep the first number, and within which range should they reject it and try their luck with a second number?*" -- *http://fivethirtyeight.com/features/can-you-win-this-hot-new-game-show/*

Naive Approach

First, lets just grab a number

In [15]:

```
import random
num = random.random()
print num
```

```
0.376410076349
```

Now lets build a function that takes a threshold, and if the first generation is less than that threshold, tries another

In [99]:

```
def pickNumbers(cutoff,tries=2):
    pick = -1.0
    for i in range(1,tries):
        if pick < cutoff:
            pick = random.random()
            #print "Try %d (%f)" % (i,pick)
    return pick
```

In [100]:

```
pickNumbers(.5)
```

Out[100]:

```
0.9374545091324559
```

Now we need to run a simulation to test a sample cutoff, to determine what the typical value it determines

In [138]:

```
import graphlab
def runSimulation(numSim,cutoff,function=pickNumbers):
    picks = []
    cutoffs = [cutoff]*numSim

    # Run the simulation
    for i in range(0,numSim):
        pick = function(cutoff)
        picks.append(pick)

    #print picks
    #print len(picks),len(cutoffs)

    return graphlab.SFrame({'cutoff':cutoffs,'pick':picks})
```

In [127]:

```
dataset = graphlab.SFrame()
dataset=dataset.append(runSimulation(1000,.5))
```

```
1000 1000
```

Now we have a simple simulation, lets take a look at some descriptive stats

In [134]:

```
graphlab.canvas.set_target('ipynb')
dataset.show(view="Summary")
```

cutoff

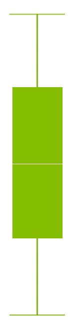
dtype:	**float**
num_unique (est.):	**1**
num_undefined:	**0**
min:	**0.5**
max:	**0.5**
median:	**0.5**
mean:	**0.5**
std:	**0**

distribution of values:

pick

dtype:	**float**
num_unique (est.):	**998**
num_undefined:	**0**
min:	**5.203e-4**
max:	**1**
median:	**0.505**
mean:	**0.501**
std:	**0.286**

distribution of values:



In [135]:

```
import graphlab.aggregate as agg
dataset.groupby(key_columns='cutoff',
               operations={"numSim":agg.COUNT(),
                          "mean_pick":agg.MEAN('pick')})
```

Out[135]:

cutoff	mean_pick	numSim
0.5	0.501407230709	1000

[1 rows x 3 columns]

Lets try a few different options

More than just .5, lets look at all combos at .01 intervals

In [148]:

```
import numpy as np
large_dataset = graphlab.SFrame()
for cutoff in np.arange(0,1,.01):
    large_dataset=large_dataset.append(runSimulation(100000,cutoff))
```

In [149]:

```
large_dataset.show()
```

cutoff

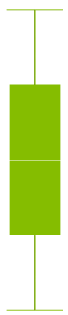
dtype:	**float**
num_unique (est.):	**100**
num_undefined:	**0**
min:	**0**
max:	**0.99**
median:	**0.5**
mean:	**0.495**
std:	**0.289**

distribution of values:

pick

dtype:	**float**
num_unique (est.):	**9,957,802**
num_undefined:	**0**
min:	**8.253e-8**
max:	**1**
median:	**0.501**
mean:	**0.5**
std:	**0.289**

distribution of values:



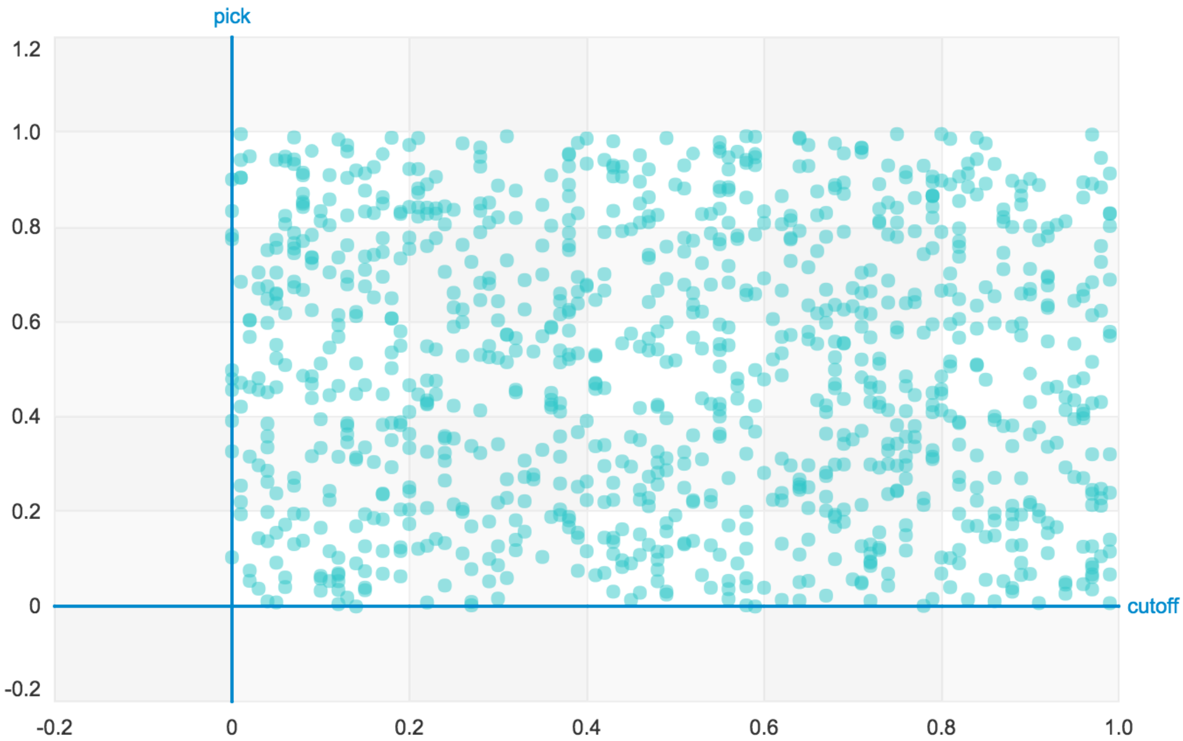
In [161]:

```
large_dataset.groupby(key_columns='cutoff',
               operations={"mean_pick":agg.MEAN('pick'),
                           "stdDev":agg.STD('pick')}).sort('mean_pick',ascending=False).print_
rows(num_rows=20)
```

```
+--------+----------------+----------------+
| cutoff |   mean_pick    |     stdDev     |
+--------+----------------+----------------+
|  0.63  | 0.501757069779 |  0.2881923323  |
|  0.99  | 0.501750080796 | 0.288382179342 |
|  0.94  | 0.501695482386 | 0.289530081979 |
|  0.28  | 0.501559455908 | 0.288510217352 |
|  0.43  | 0.501356004517 | 0.287676019932 |
|  0.76  | 0.501346458365 | 0.288098905934 |
|  0.15  | 0.501288800369 | 0.288465576877 |
|  0.42  | 0.501278989494 | 0.289236706126 |
|  0.31  | 0.501226430097 | 0.288716071103 |
|  0.33  | 0.501176844303 | 0.289013671667 |
|  0.77  | 0.501123788633 | 0.288777041285 |
|  0.81  | 0.501089691801 | 0.288603112659 |
|  0.12  | 0.50106957873  | 0.290128423622 |
|  0.05  | 0.501054476795 | 0.288669078019 |
|  0.88  | 0.501052252264 | 0.289067603594 |
|  0.3   | 0.500914916297 | 0.289441136272 |
|  0.29  | 0.500884426395 | 0.289066400489 |
|  0.18  | 0.500873659138 | 0.289281249597 |
|  0.89  | 0.500862851866 | 0.28843690675  |
|  0.47  | 0.500816461414 | 0.288850699842 |
+--------+----------------+----------------+
[100 rows x 3 columns]
```

In [162]:

```
large_dataset.show(view="Scatter Plot",x='cutoff',y='pick')
```



The cutoff value doesn't increase the ultimate pick

Conclusion

Over sufficiently large simulations (10M per cutoff selection at .01 intervals), cutoff isn't correlated with any meaningful higher final pick. You might as well just stick with the first pick.