In [ ]:
```
print("hello world")
```

```
hello world
```

One Arm Bandit Simuls

In [ ]:
```
import numpy as np
import math
import random
import pandas as pd
import altair as alt
```

In [ ]:
```
class EpsilonGreedy():
    def __init__(self, epsilon, counts, values):
        self.epsilon = epsilon
        self.counts = counts
        self.values = values
        return

    # Initialise k number of arms
    def initialize(self, n_arms):
        self.counts = [0 for col in range(n_arms)]
        self.values = [0.0 for col in range(n_arms)]
        return

    # Epsilon greedy arm selection
    def select_arm(self):
        # If prob is not in epsilon, do exploitation of best arm so far
        if random.random() > self.epsilon:
            return np.argmax(self.values)
        # If prob falls in epsilon range, do exploration
        else:
            return random.randrange(len(self.values))

    # Choose to update chosen arm and reward
    def update(self, chosen_arm, reward):
        # update counts pulled for chosen arm
        self.counts[chosen_arm] = self.counts[chosen_arm] + 1
        n = self.counts[chosen_arm]

        # Update average/mean value/reward for chosen arm
        value = self.values[chosen_arm]
        new_value = ((n-1)/float(n)) * value + (1 / float(n)) * reward
        self.values[chosen_arm] = new_value
        return
```

In [ ]:
```
class BernoulliArm():
    def __init__(self, p):
        self.p = p

    # Reward system based on Bernoulli
    def draw(self):
        if random.random() > self.p:
            return 0.0
        else:
            return 1.0
```

In [ ]:
```
def simuls(algo, arms, num_sims, time):

    chosen_arms = [0.0 for i in range(num_sims * time)]
    rewards = [0.0 for i in range(num_sims * time)]
    cumulative_rewards = [0 for i in range(num_sims * time)]
    sim_nums = [0.0 for i in range(num_sims *time)]
    times = [0.0 for i in range (num_sims*time)]

    for sim in range(num_sims):
        sim = sim + 1
        algo.initialize(len(arms))

        for t in range(time):
            t = t + 1
            index = (sim -1) * time + t -1
            sim_nums[index] = sim
            times[index] = t

            # Selection of best arm and engaging it
            chosen_arm = algo.select_arm()
            chosen_arms[index] = chosen_arm

            # Engage chosen Bernoulli Arm and obtain reward info
            reward = arms[chosen_arm].draw()
            rewards[index] = reward

            if t ==1:
                cumulative_rewards[index] = reward
            else:
                cumulative_rewards[index] = cumulative_rewards[index-1] + re

            algo.update(chosen_arm, reward)

    return [sim_nums, times, chosen_arms, rewards, cumulative_rewards]
```

In [ ]:
```
import random
random.seed(1)
# out of 5 arms, 1 arm is clearly the best
means = [0.1, 0.1, 0.1, 0.1, 0.9]
means = [0.8,0.8,0.8,0.8,0.9]
n_arms = len(means)
time=1000
simulCount=5000
# Shuffling arms
random.shuffle(means)
best_arm=np.argmax(means)
arms = list(map(lambda mu: BernoulliArm(mu), means))
print("Best arm is " + str(best_arm)+ " with mean "+ str(means[best_arm]))
f = open("standard_results_epsg.tsv", "w+")
for epsilon in [0.1, 0.2, 0.3, 0.4, 0.5]:
    algo = EpsilonGreedy(epsilon, [], [])
    algo.initialize(n_arms)
    results = simuls(algo, arms, simulCount, time)

    # Store data
    for i in range(len(results[0])):
        f.write(str(epsilon) + "\t")
        f.write("\t".join([str(results[j][i]) for j in range(len(results))]))
f.close()
```

```
Best arm is 2 with mean 0.9
```

In [ ]:
```
df = pd.read_csv("standard_results_epsg.tsv", sep = "\t", header = None, nam
df.head()
```

Out[ ]:

	epsilon	simulation_num	step	chosen_arm	reward	cum_reward
0	0.1	1	1	0	1.0	1.0
1	0.1	1	2	0	1.0	2.0
2	0.1	1	3	0	0.0	2.0
3	0.1	1	4	0	1.0	3.0
4	0.1	1	5	0	1.0	4.0

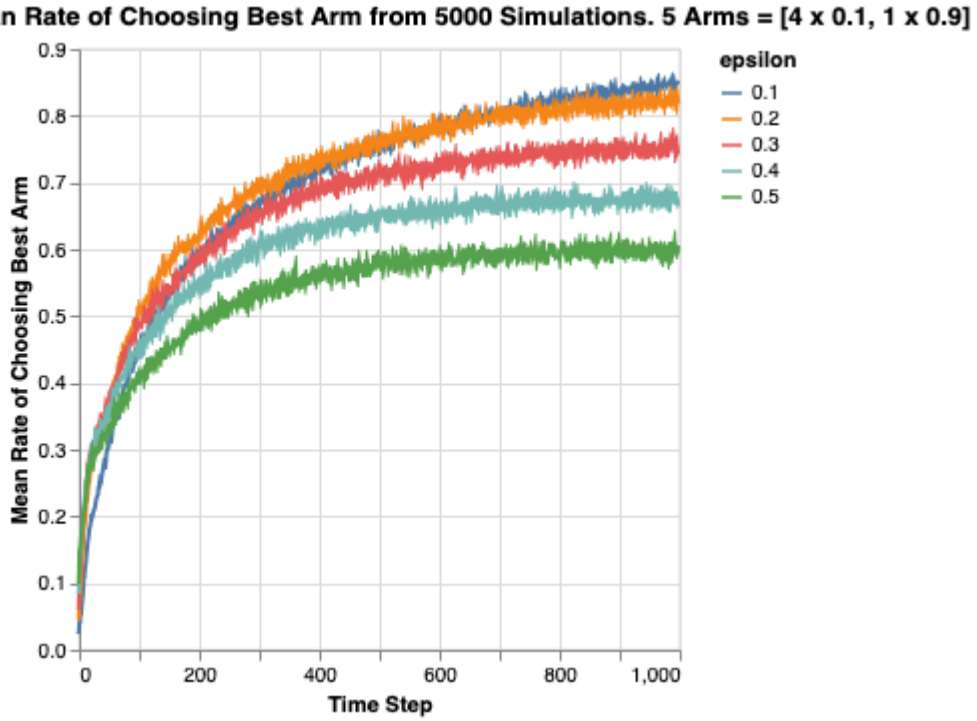
In [ ]:
```
df["chose_correct"] = np.select(
    [
        df["chosen_arm"] == best_arm,
        df["chosen_arm"] != best_arm
    ],
    [
        1,
        0
    ]
)
```

In [ ]:
```
df_chose_correctly = df.loc[:,["epsilon","step", "chose_correct"]].groupby(["

df_chose_correctly = df_chose_correctly.reset_index()
```

In [ ]:
```
alt.Chart(df_chose_correctly).mark_line().encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("chose_correct:Q", title = "Mean Rate of Choosing Best Arm"),
    color = alt.Color("epsilon:N")
).properties(
    title = "Eps-Greedy: Mean Rate of Choosing Best Arm from 5000 Simulatior
)
```

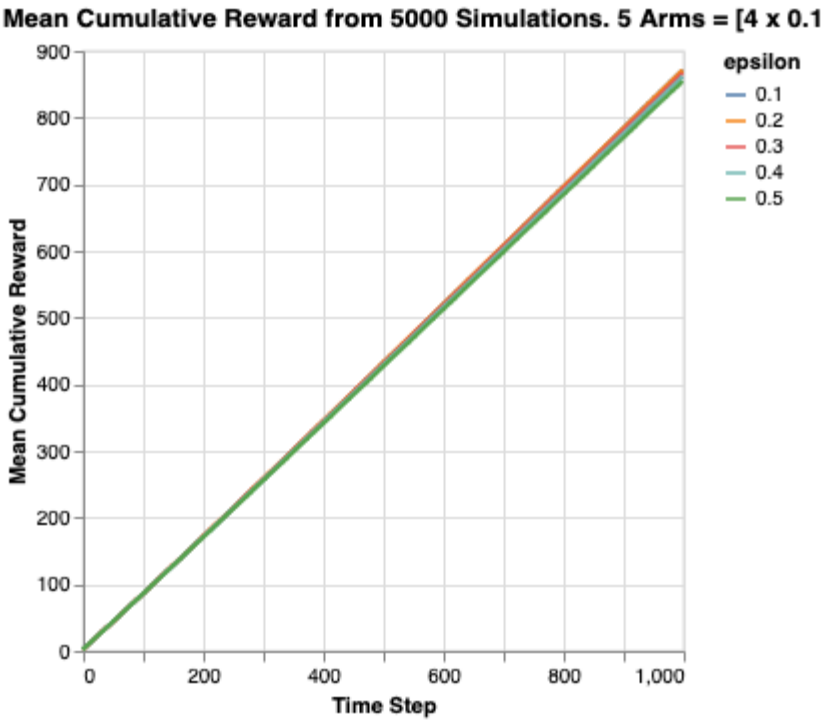
Out[ ]:



In [ ]:
```
df_cumreward = df.loc[:,["epsilon","step", "cum_reward"]].groupby(["epsilon"
```

In [ ]:
```
alt.Chart(df_cumreward).mark_line().encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("cum_reward:Q", title = "Mean Cumulative Reward"),
    color = alt.Color("epsilon:N")
).properties(
    title = "Eps-Greedy: Mean Cumulative Reward from 5000 Simulations. 5 Arm
)
```

Out[ ]:


Eps-Greedy: Mean Cumulative Reward from 5000 Simulations. 5 Arms = [4 x 0.1, 1 x 0.9]
Mean Cumulative Reward
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1,000
Time Step
epsilon
0.1
0.2
0.3
0.4
0.5

UCB1

In [ ]:
```
class UCB1():
    def __init__(self, counts, values):
        self.counts = counts
        self.values = values
        return

    def initialize(self, n_arms):
        self.counts = [0 for col in range(n_arms)]
        self.values = [0.0 for col in range(n_arms)]
        return

    def select_arm(self):
        n_arms = len(self.counts)
        for arm in range(n_arms):
            if self.counts[arm] == 0:
                return arm

        ucb_values = [0.0 for arm in range(n_arms)]
        total_counts = sum(self.counts)

        for arm in range(n_arms):
            bonus = math.sqrt((2 * math.log(total_counts)) / float(self.cour
            ucb_values[arm] = self.values[arm] + bonus
        return ucb_values.index(max(ucb_values))

    def update(self, chosen_arm, reward):
        self.counts[chosen_arm] = self.counts[chosen_arm] + 1
        n = self.counts[chosen_arm]

        value = self.values[chosen_arm]
        new_value = ((n - 1) / float(n)) * value + (1 / float(n)) * reward
        self.values[chosen_arm] = new_value
        return
```

In [ ]:
```
import random

random.seed(1)
# out of 5 arms, 1 arm is clearly the best
means = [0.1, 0.1, 0.1, 0.1, 0.9]
means = [0.8, 0.8, 0.8, 0.8, 0.9]
n_arms = len(means)
# Shuffling arms
random.shuffle(means)
best_arm=np.argmax(means)
time=250
simulCount=5000

# Create list of Bernoulli Arms with Reward Information
arms = list(map(lambda mu: BernoulliArm(mu), means))
print("Best arm is " + str(best_arm))

f = open("standard_ucb_results_2.tsv", "w+")

# Create 1 round of 5000 simulations
algo = UCB1([], [])
algo.initialize(n_arms)
results = simuls(algo, arms, simulCount, time)

# Store data
for i in range(len(results[0])):
    f.write("\t".join([str(results[j][i]) for j in range(len(results))]) + '
f.close()
print("done")
```

```
Best arm is 2
done
```

In [ ]:
```
df = pd.read_csv("standard_ucb_results_2.tsv", sep = "\t", header = None, na

df.head()
```

Out[ ]:

	simulation_num	step	chosen_arm	reward	cum_reward
0	1	1	0	1.0	1.0
1	1	2	1	1.0	2.0
2	1	3	2	1.0	3.0
3	1	4	3	1.0	4.0
4	1	5	4	1.0	5.0

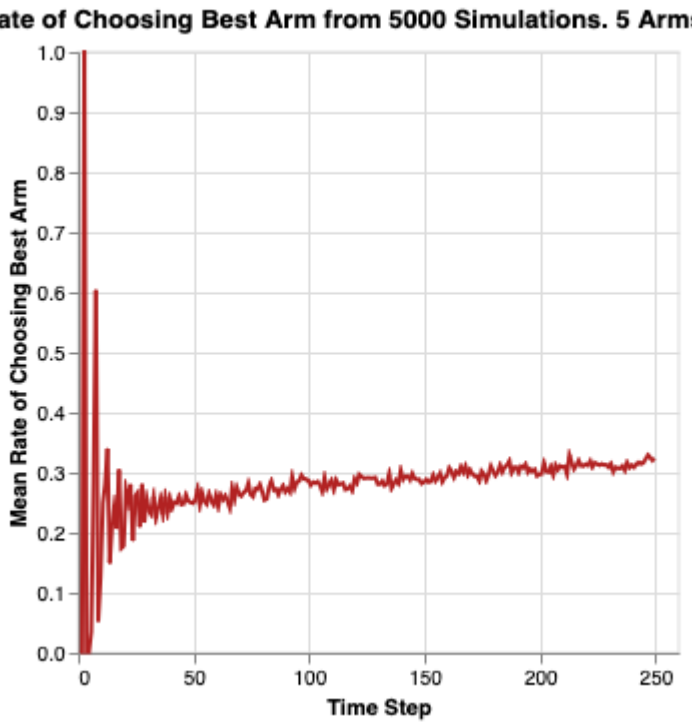
In [ ]:
```
df["chose_correct"] = np.select(
    [
        df["chosen_arm"] == best_arm,
        df["chosen_arm"] != best_arm
    ],
    [
        1,
        0
    ]
)
```

In [ ]:
```
# Perform average/mean for each step for all simulations and epsilon
df_chose_correctly = df.loc[:,["step", "chose_correct"]].groupby(["step"]).a

# Remove multi index grouping
df_chose_correctly = df_chose_correctly.reset_index()
```

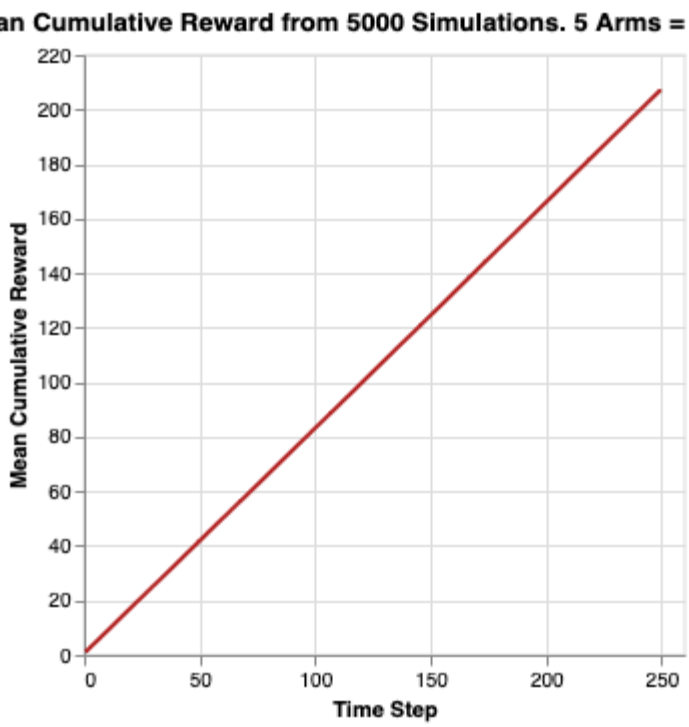
In [ ]:
```
alt.Chart(df_chose_correctly).mark_line(color='firebrick').encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("chose_correct:Q", title = "Mean Rate of Choosing Best Arm", scale
).properties(
    title = "UCB: Mean Rate of Choosing Best Arm from 5000 Simulations. 5 Ar
)
```

Out[ ]:



In [ ]:
```
df_cumreward = df.loc[:,["step", "cum_reward"]].groupby(["step"]).agg("mean"
alt.Chart(df_cumreward).mark_line(color="firebrick").encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("cum_reward:Q", title = "Mean Cumulative Reward")
).properties(
    title = "UCB: Mean Cumulative Reward from 5000 Simulations. 5 Arms = [4
)
```

Out[ ]:



In [ ]:
```
from scipy.stats import beta
```

In [ ]:
```
class ThompsonSampling():
    def __init__(self, counts, values, a, b):
        self.counts = counts
        self.values = values

        # Beta parameters
        self.a = a
        self.b = b
        return

    def initialize(self, n_arms):
        self.counts = [0 for col in range(n_arms)]
        self.values = [0.0 for col in range(n_arms)]

        # Uniform distribution of prior beta (A,B)
        self.a = [1 for arm in range(n_arms)]
        self.b = [1 for arm in range(n_arms)]
        return

    def select_arm(self):
        n_arms = len(self.counts)

        # Pair up all beta params of a and b for each arm
        beta_params = zip(self.a, self.b)

        # Perform random draw for all arms based on their params (a,b)
        all_draws = [beta.rvs(i[0], i[1], size = 1) for i in beta_params]

        # return index of arm with the highest draw
        return all_draws.index(max(all_draws))

    def update(self, chosen_arm, reward):
        self.counts[chosen_arm] = self.counts[chosen_arm] + 1
        n = self.counts[chosen_arm]

        value = self.values[chosen_arm]
        new_value = ((n - 1) / float(n)) * value + (1 / float(n)) * reward
        self.values[chosen_arm] = new_value

        # Update a and b

        # a is based on total counts of rewards of arm
        self.a[chosen_arm] = self.a[chosen_arm] + reward

        # b is based on total counts of failed rewards on arm
        self.b[chosen_arm] = self.b[chosen_arm] + (1-reward)

        return
```

In [ ]:
```
import random

random.seed(1)
# out of 5 arms, 1 arm is clearly the best
means = [0.1, 0.1, 0.1, 0.1, 0.9]
n_arms = len(means)
# Shuffling arms
random.shuffle(means)

# Create list of Bernoulli Arms with Reward Information
arms = list(map(lambda mu: BernoulliArm(mu), means))
print("Best arm is " + str(np.argmax(means)))

f = open("ts_results.tsv", "w+")

# Create simulations for ThompsonSampling
algo = ThompsonSampling([], [], [], [])
algo.initialize(n_arms)
results = simuls(algo, arms, 5000, 250)

# Store data
for i in range(len(results[0])):
    f.write("\t".join([str(results[j][i]) for j in range(len(results))]) + "
f.close()
print("done")
```

```
Best arm is 2
done
```

In [ ]:
```
df = pd.read_csv("ts_results.tsv", sep = "\t", header = None, names = ["simu

df.head()
```

Out[ ]:

	simulation_num	step	chosen_arm	reward	cum_reward
0	1	1	1	0.0	0.0
1	1	2	1	0.0	0.0
2	1	3	3	0.0	0.0
3	1	4	4	0.0	0.0
4	1	5	0	1.0	1.0

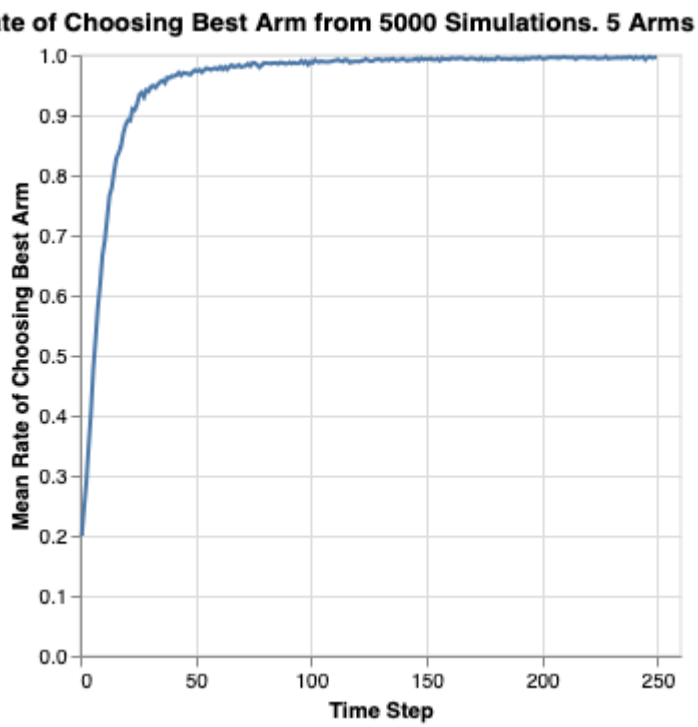
In [ ]:
```
df["chose_correct"] = np.select(
    [
        df["chosen_arm"] == 2,
        df["chosen_arm"] != 2
    ],
    [
        1,
        0
    ]
)
```

In [ ]:
```
# Perform average/mean for each step for all simulations and epsilon
df_chose_correctly = df.loc[:,["step", "chose_correct"]].groupby(["step"]).a

df_chose_correctly = df_chose_correctly.reset_index()
```

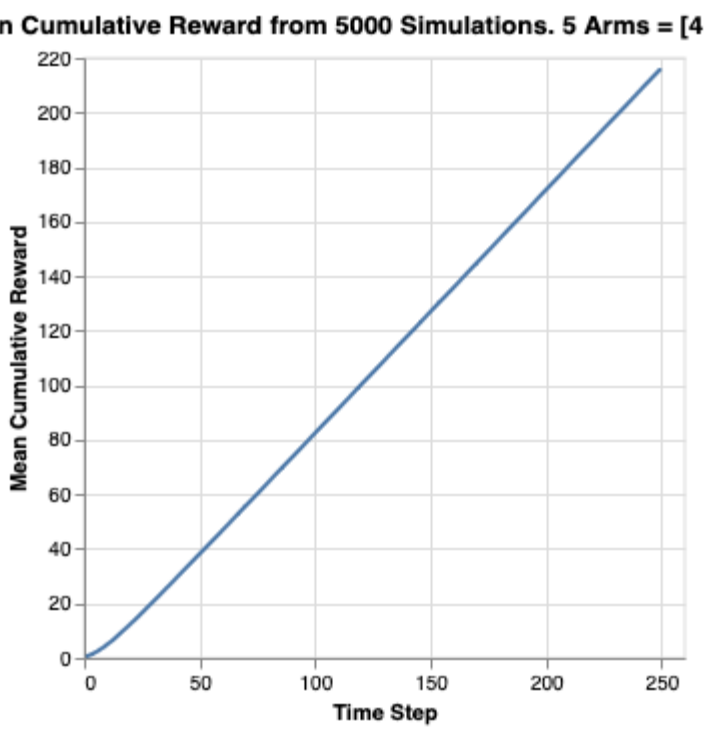
In [ ]:
```
alt.Chart(df_chose_correctly).mark_line().encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("chose_correct:Q", title = "Mean Rate of Choosing Best Arm", scale
).properties(
    title = "TS: Mean Rate of Choosing Best Arm from 5000 Simulations. 5 Arm
)
```

Out[ ]:



In [ ]:
```
df_cumreward = df.loc[:,["step", "cum_reward"]].groupby(["step"]).agg("mean"
alt.Chart(df_cumreward).mark_line().encode(
    alt.X("step:Q", title = "Time Step"),
    alt.Y("cum_reward:Q", title = "Mean Cumulative Reward")
).properties(
    title = "TS: Mean Cumulative Reward from 5000 Simulations. 5 Arms = [4 x
)
```

Out[ ]:


TS: Mean Cumulative Reward from 5000 Simulations. 5 Arms = [4 x 0.1, 1 x 0.9]
Mean Cumulative Reward
0
20
40
60
80
100
120
140
160
180
200
220
0
50
100
150
200
250
Time Step

In [ ]: