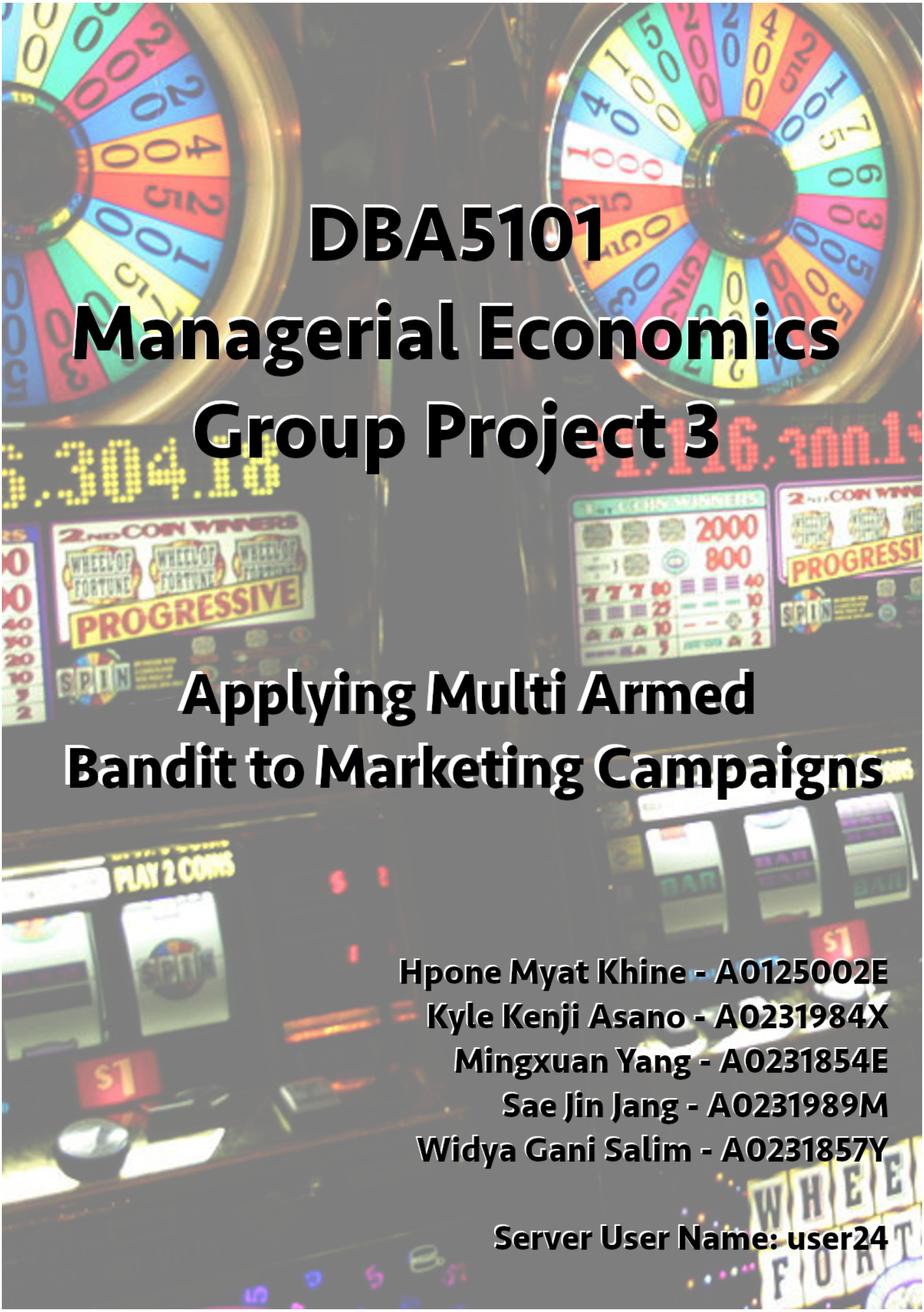


DBA5101 Managerial Economics Group Project 3

Applying Multi Armed Bandit to Marketing Campaigns

Hpone Myat Khine - A0125002E
Kyle Kenji Asano - A0231984X
Mingxuan Yang - A0231854E
Sae Jin Jang - A0231989M
Widya Gani Salim - A0231857Y

Server User Name: user24

1. Introduction & Problem Statement

This project is based on an Obama's fundraising campaign simulation where we are trying to find the best combinations of a sign-up button and a background media that maximises the number of visitor sign-ups. There are 4 buttons and 6 media available, giving a total of 24 possible combinations. Similar to real life, a budget constraint is imposed by capping the number of combination tries to 100 million views. Each try reflects the results of 100 visitors' views, and so, we have a maximum of 1 million tries.

Key objective: find the best sign-up button and media combination as efficiently as possible.
Approach: We choose to adopt a Multi Armed Bandit (MAB) approach instead of A/B split testing which is commonly used to determine best combinations. MAB is more efficient - able to find the best combination faster - as it adapts and shifts towards winning variations throughout the experiment, instead of waiting until the end.

2. Overview of Multi Armed Bandits (MAB)

MAB algorithm works to make an optimal decision based on available information. It finds the best option, among a myriad of options (called 'arms') with different rewards, over a defined period of time (called 'rounds'). At every round, the algorithm chooses an arm and collects the reward for the chosen arm.

MAB strives to find the best trade-off between exploring the distribution of rewards of the different arms (***exploration phase***) and optimising the collection of rewards from the currently known best arms (***exploitation phase***). Put simply, exploring too much will lead to fewer chances for exploitation and insufficient exploring may result in higher chances of missing the best arms.

MAB performance can be estimated using ***Cumulative Regret***, which is defined below:

$$Regret = T\mu^* - \sum_{t=1}^{T} R_t$$

μ^* = Mean reward of optimal arm

T = Total number of rounds

R_t = current reward obtained according to implemented algorithm for the arm chosen

Every round there is an optimal arm that can be chosen, and so, a maximum reward is obtained only if the best arm was chosen every time. When a non-optimal arm was chosen, the total reward that can be obtained is lower than the maximum reward. Regret measures the difference between the maximum possible reward and the rewards from the chosen algorithm. The MAB algorithm seeks to minimise regret.

3. Model Selection

3.1 Assumptions

For this project, we make several assumptions in which the MAB operates under:

- I.I.D rewards: rewards are drawn ***independently*** from an unknown ***fixed*** distribution.
- Rewards distribution does not change over time.
- Bandit feedback model: reward is observed only when an arm is chosen.

3.2 Chosen Models

In this project, we will be focusing on the following MAB algorithms:

- Uniform Distribution
- ε-greedy
- Upper confidence bound (UCB1)

	Uniform Exploitation	ε-greedy	Upper confidence bound (UCB1)
Exploration Phase	Each arm is tried N times Arm with highest average reward, $\hat{a}$ is chosen (tie-breaker is randomly decided)	Arm is chosen uniformly at random at each turn with probability $\boldsymbol{\varepsilon}$	UCB1 changes its exploration-exploitation ratio as it gains more knowledge. It initially starts off with exploration before settling on exploitation as the optimal estimated reward becomes clear.
Exploitation Phase	Estimated optimal arm $\hat{a}$ is chosen repeatedly.	By default, the arm with the highest average thus far is chosen with probability 1 - $\boldsymbol{\varepsilon}$	$\hat{\mu}_a(t)$ = estimated mean of arm a at round t
Limitation	Choice of N: Too high will lead to exploitation not being maximised Too low, there might be insufficient information for exploitation (not optimal arm being chosen)	Given fixed ε, algorithm would still switch to other arms at random even when the optimal arm has been found. Adaptive ε-greedy seeks to improve on this, but not used as no practical advantage to it (Vermorel and Mohri 2005)	The algorithm could be further fine-tuned with a parameter* of α In addition, UCB1 can be iteratively improved upon by having the arm trialed for a certain period before trying a new arm. This is known as UCB2. *See paragraph under UCB below this table
Formulation	Try each arm: N times Select arm $\hat{a}$ with highest average reward For remaining rounds, play arm $\hat{a}$, to maximise exploitation.	For each round: Arm is chosen at random with probability ε Highest average reward so far $\hat{\mu}_a(t)$ is chosen with probability 1 - ε	UCB Index of arm: $\hat{\mu}_a(t) + \sqrt{\frac{2ln(t)}{n_a(t)}}$ $n_a(t)$ = number of times arm a has been chosen $\hat{\mu}_a(t)$= estimated mean of arm a at round t Arm with highest UCB index is chosen and steps are incremented

			and repeated $\hat{\mu}_{a}(t + 1) = \frac{\hat{\mu}_{\hat{a}}(t)\times n_{\hat{a}}(t) + X_{\hat{a}}}{n_{\hat{a}}(t)+1}$ $X_{\hat{a}}$= Observed reward of chosen arm $\hat{a}$ $n_{\hat{a}}(t + 1) = n_{\hat{a}}(t) + 1$

*From UCB, it is worth noting that the authors implemented UCB1 algorithm as provided by Keppo, Tan & Zhuang, with no extra penalty on arms pulled. In the original UCB1 formula proposed by Li, Chu, Langford & Schapire, there was a parameter of α, used as $\alpha \times \sqrt{\frac{2ln(t)}{n_a(t)}}$, that would control the exploration of other arms (with $\alpha \geq 0$). As this was not discovered until after data collection, the UCB1 model was run with an implied $\alpha = 1$; as noted later, this had strong implications on arms pulled.

4. Implementation and Results

4.1 Model Implementation

Uniform Distribution: We set N to be 10, and so each arm was pulled 4 times during the exploration phase. This means that the exploration phase was set to be 96 times. Post-exploration, arm 0 was chosen for the remaining pulls, giving 84 pulls on the optimal arm.

ε-greedy: We set the probability of ε-Greedy exploring other arms (p) to be 30%.

UCB1: We set an exploration parameter ɑ to be equal to 1.

Number of Rounds: For each algorithm, we decided to select an arm 2000 times.

4.2 Results & Insights

Table 1 - Result Summary by Model

	Uniform Exploitation	ε-greedy	Upper confidence bound (UCB1)
Cumulative Rewards	51,157	42,988	48,289
Mean Rewards	25.5785	21.494	24.1445
Mean Rewards of Optimal Arms (Arm 0)	22.9	24.587302	24.492231
Cumulative Regret	-5357.0	6186.60	99.46

We found that our best performing algorithm is the uniform distribution, followed by UCB1 and in the last place, ε-greedy (see Table 1). Uniform gave the lowest cumulative regret and best reward overall as it exploited the optimal arm 0 immediately after its exploitation phase - pulling it 1938 times. UCB1 pulled the optimal arm 1802 times, taking some time to converge as it explored other arm 8, which gave the second highest mean during UCB1 exploration phase. ε-greedy, which produced the highest cumulative regret, was unable to converge as effectively as Uniform or UCB1 due to randomisation, only converging to the optimal arm after around 380 pulls. We can see the performance of all models overtime in **Figure 0** (in the Appendix).

4.2.1 Further Insights on UCB1 Results

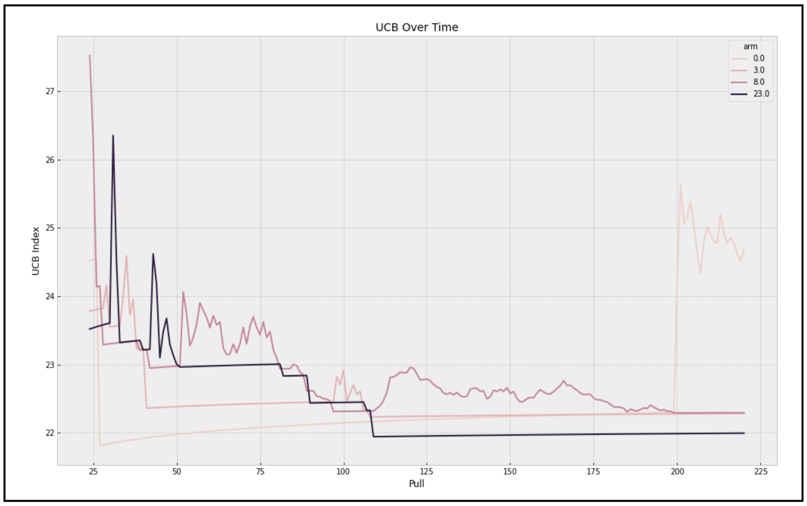


Figure 1 - UCB Over Time Result

Although the pulling of suboptimal arms slowed the convergence of UCB1, it allowed the model to discover the mean rewards of the 'next best' optimal arms better, as compared to the uniform distribution model (see table 2). This information might be useful for campaign managers who would want to know the response for next best alternative combinations.

As in Figure 1, our UCB1 model opted for Arm 0 (1st arm, 0 index) at pull 200 (overall 201st; pull 176 of exploitation phase) with mean reward of 24.5, after pulling 5 other arms prior to convergence (and ignoring 19 other arms after exploration phase).

Table 2 - Mean Reward of Top Arms by Model

Mean Reward of Arm	UCB1	ε-greedy	Uniform Distribution
Arm 0	24.49	24.59	22.9
Arm 3	21.55	22.95	22.3
Arm 8	22.01	21.84	21.7
Arm 21	21.22	23.68	20.6
Arm 23	18.67	21.25	22.5

4.3 Implications and Insights

Uniform distribution gave the highest reward for the campaign managers. However, it requires a waiting period to explore each arm before taking an action post-exploration phase. In real life, this might not be ideal as campaign managers might also face time constraint, on top of budget constraint. In that case, UCB1 offers a relatively faster alternative to determine a winning variation on a sign-up page. It also offers a better insight into the distribution of the next best alternative, as discussed in part 4.2.1. Nonetheless, the strength of UCB1 is two-sided, as the current model will not explore unknown variations until a high degree of uncertainty is reached. This may merit either increasing α (and promoting more exploration), or using the unused variations in other pages where the model is not deployed (for gathering data to create a prior).

References

Gittins, Glazebrook, and Weber (2011), “Multi Armed Bandit Allocation Indices”.
Keppo, Tan, and Zhuang (2021), “Lecture Notes (Multi-Armed Bandits)”.
Li, Chu, Langford, and Schapire (2010), “A Contextual-Bandit Approach to Personalized News Article Recommendation”. https://arxiv.org/pdf/1003.0146.pdf
Liu (2021), “Multi Armed Bandits”.
http://ethen8181.github.io/machine-learning/bandits/multi_armed_bandits.html
Roberts (2020), “The Upper Confidence Bound (UCB) Bandit Algorithm”.
https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f
Roberts (2020), “Thompson Sampling”.
https://towardsdatascience.com/thompson-sampling-fc28817eacb8
Roberts (2021), “Thompson Sampling using Conjugate Priors”.
https://towardsdatascience.com/thompson-sampling-using-conjugate-priors-e0a18348ea2d

Burtini, G., Loeppky, J., & Lawrence, R. (n.d.). *A Survey of Online Experiment Design with the Stochastic Multi-Armed Bandit*. Retrieved November 14, 2021, from https://arxiv.org/pdf/1510.00757.pdf.

Figure 0 - Cumulative Reward By Model

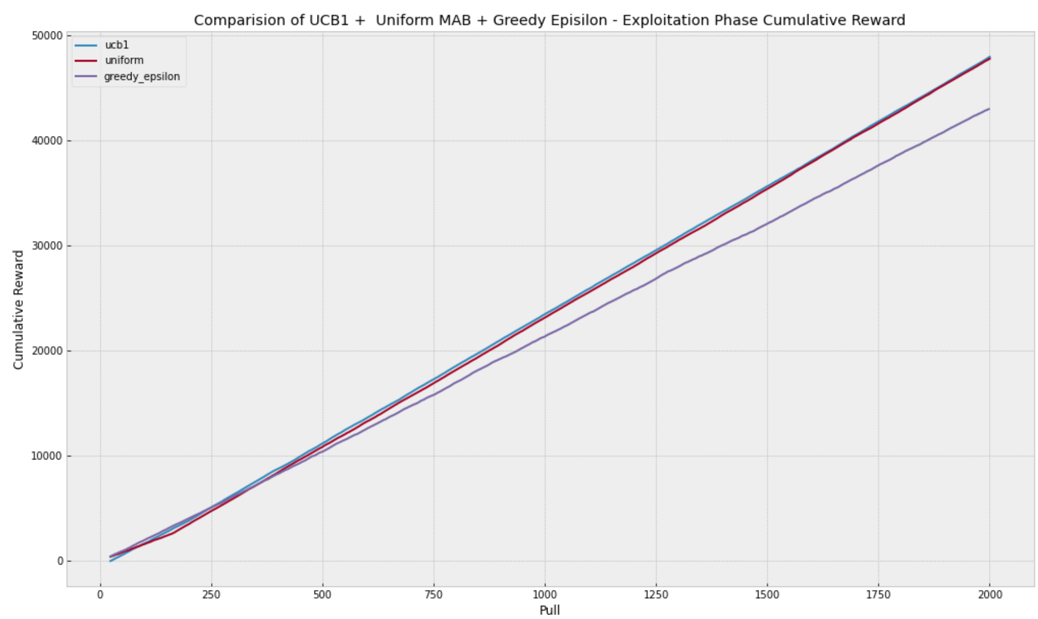


Figure 1 - Table of Arm Rewards During Exploration & Exploitation, UCB1

Exploration

Pull Number	Arm	Reward
0	0	22
1	1	12
2	2	7
3	3	27
4	4	5

5	5	10
6	6	5
7	7	19
8	8	25
9	9	19
10	10	13
11	11	18
12	12	18
13	13	12
14	14	17
15	15	8
16	16	8
17	17	20
18	18	11
19	19	16
20	20	19
21	21	20
22	22	12
23	23	21

Exploitation

arm	count	mean	std	min	25%	50%	75%	max
0	1826	24.496714	4.284952	11.0	22.00	24.5	27.0	42.0
1	1	12.0	0	12.0	12.00	12.0	12.0	12.0
2	1	7.0	0	7.0	7.00	7.0	7.0	7.0
3	20	21.550	3.993086	14.0	17.75	22.0	25.0	27.0
4	1	5.0	0	5.0	5.00	5.0	5.0	5.0
5	1	10.0	0	10.0	10.00	10.0	10.0	10.0
6	1	5.0	0	5.0	5.00	5.0	5.0	5.0
7	1	19.0	0	19.0	19.00	19.0	19.0	19.0
8	137	22.014599	3.974154	13.0	19.00	22.0	25.0	34.0
9	1	19.0	0	19.0	19.00	19.0	19.0	19.0
10	1	13.0	0	13.0	13.00	13.0	13.0	13.0
11	1	18.0	0	18.0	18.00	18.0	18.0	18.0
12	1	18.0	0	18.0	18.00	18.0	18.0	18.0
13	1	12.0	0	12.0	12.00	12.0	12.0	12.0
14	1	17.0	0	17.0	17.00	17.0	17.0	17.0
15	1	8.0	0	8.0	8.00	8.0	8.0	8.0
16	1	8.0	0	8.0	8.00	8.0	8.0	8.0
17	2	20.0	0	20.0	20.00	20.0	20.0	20.0
18	1	11.0	0	11.0	11.00	11.0	11.0	11.0
19	1	16.0	0	16.0	16.00	16.0	16.0	16.0
20	1	19.0	0	19.0	19.00	19.0	19.0	19.0
21	3	18.666667	3.214550	15.0	17.50	20.0	20.5	21.0
22	1	12.0	0	12.0	12.00	12.0	12.0	12.0
23	18	21.222222	4.194612	15.0	19.25	21.0	22.0	31.0

Figure 2 - UCB Index During Early Exploitation Phase, UCB1

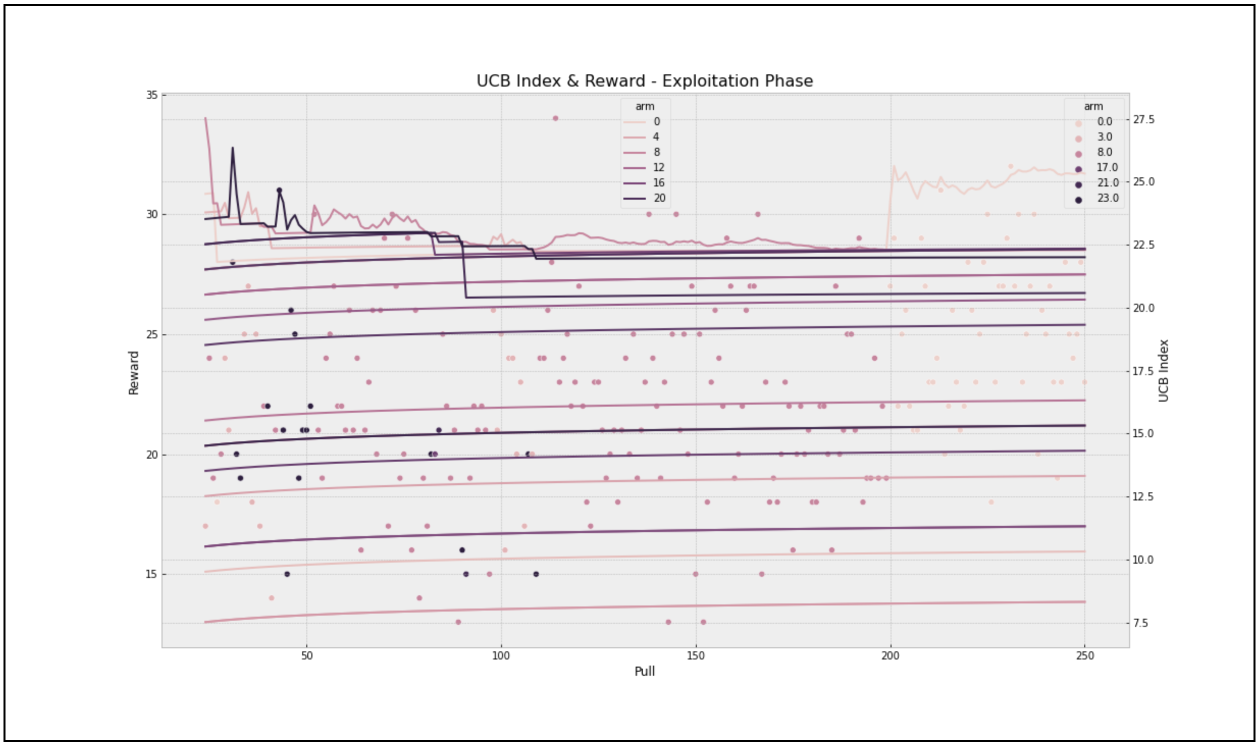


Figure 3 - Violin Plot of Arm Rewards During Exploitation, UCB1

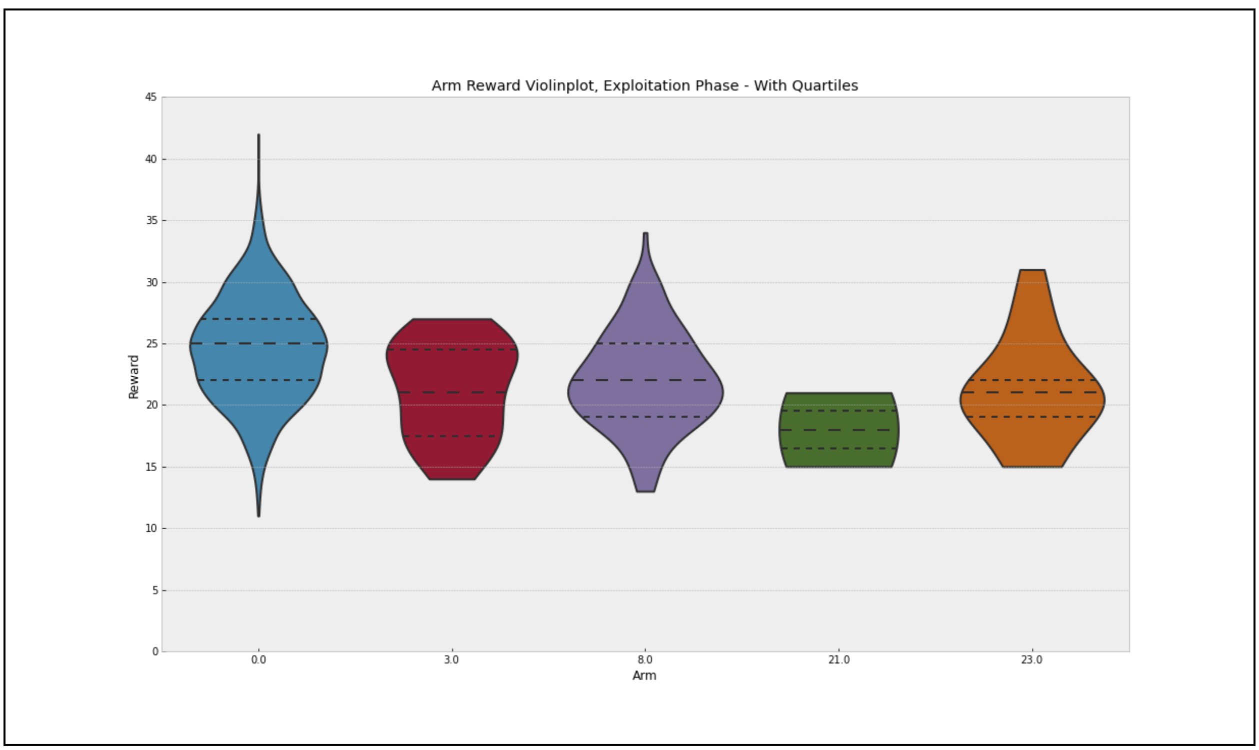


Figure 4 - Table of Arm Rewards & Index At Exploitation Pull 2000, UCB1

arm	arm_pull	cumulative_reward	contemporary_linucb	ucb_mean	penalty
0	1823	44651.0	24.584520	24.493143	0.091377
7	1	19.0	22.901500	19.000	3.901500

20	1	19.0	22.901500	19.000	3.901500
9	1	19.0	22.901500	19.000	3.901500
17	2	40.0	22.758777	20.000	2.758777

Figure 5 - UCB Index During Late Exploitation Phase, UCB1

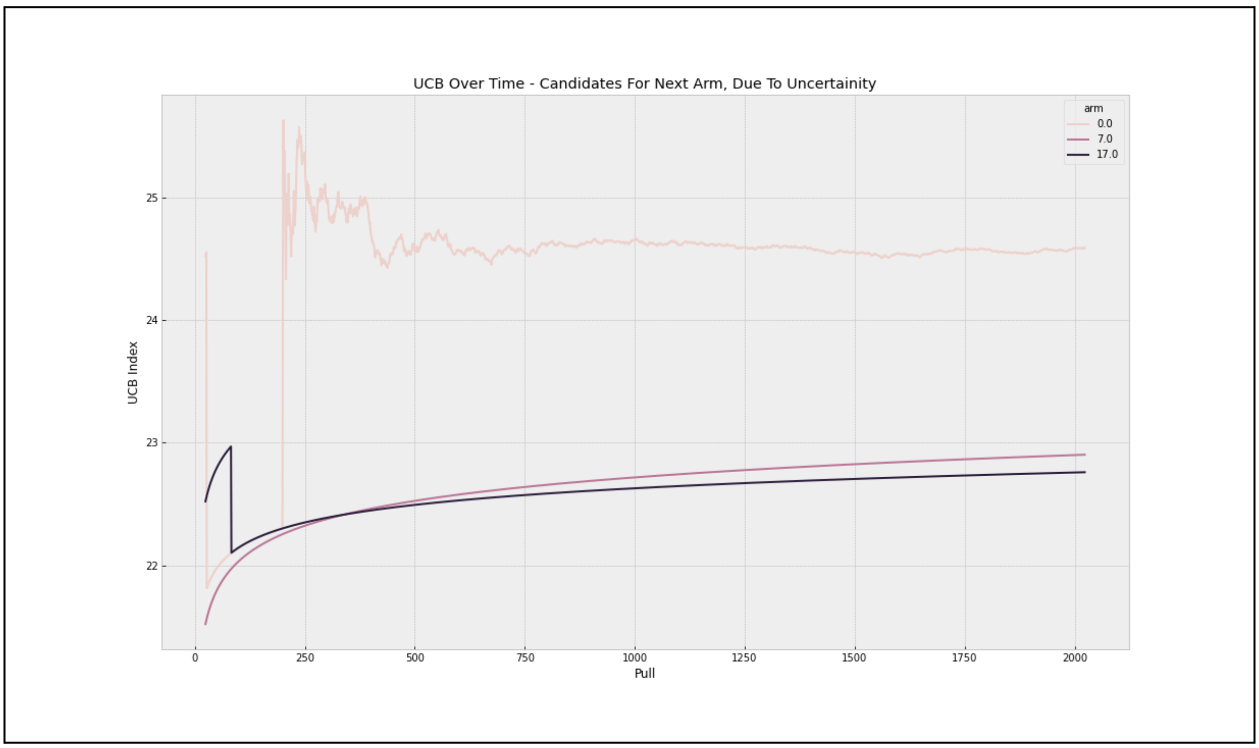
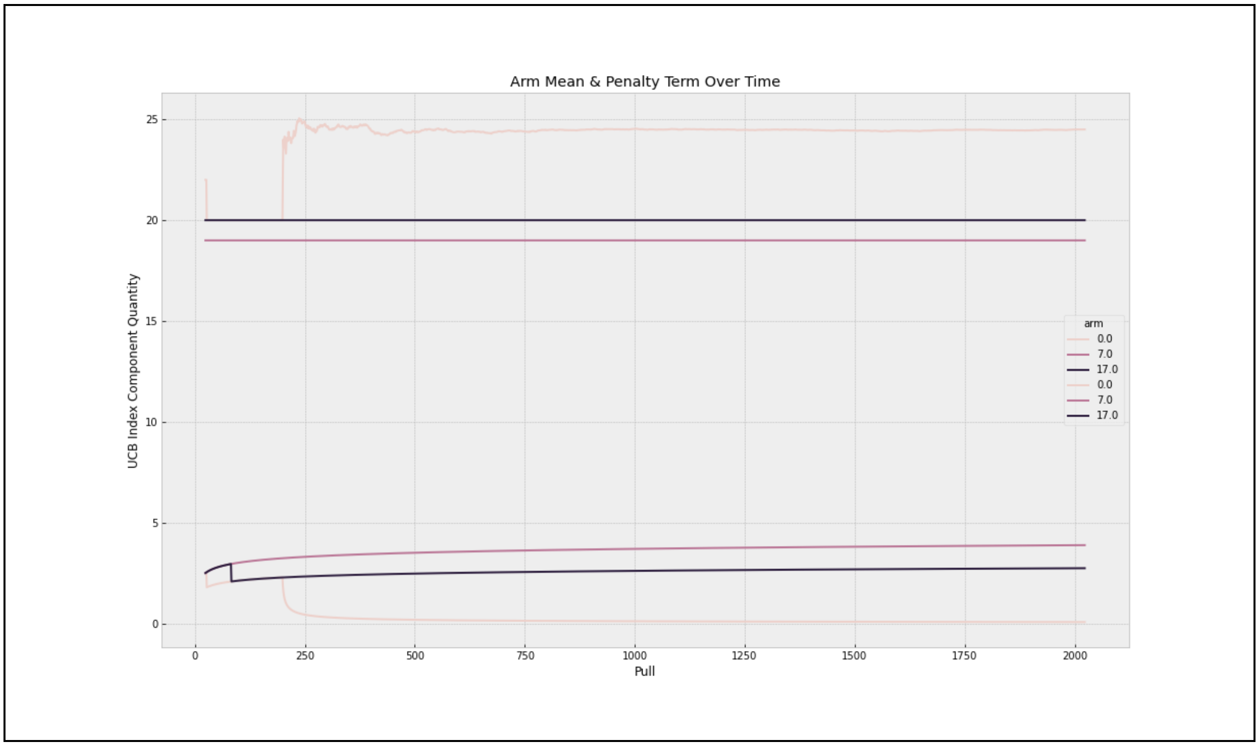


Figure 6 - Components of UCB Index, UCB1



Arm Mean & Penalty Term Over Time
UCB Index Component Quantity
Pull
arm
0.0
7.0
17.0
0.0
7.0
17.0
0
5
10
15
20
25
0
250
500
750
1000
1250
1500
1750
2000

Uniform Exploitation

In [61]:

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

import requests

def pull(user_group, secret_key, arm):
        url = ('http://10.243.255.29:5787/pull_arm/%s/%s/%s' % (user_group,secret_key,str(arm)))

        while True:
                r = requests.get(url)
                if r.ok:
                        output = r.json()['result']
                        return(output)
```

In [97]:

```
arms = [str(x) for x in range(24)]
rewards = [0]*24
tries = [0]*24

result = pd.DataFrame({'arm': arms, 'reward': rewards, 'tries': tries})
```

In [18]:

```
dict = {"result":{"Arm":"4","NetReward":49382,"Pull":2047,"Reward":10},"status":200}
{'Arm': '1', 'NetReward': 49395, 'Pull': 2048, 'Reward': 13}
{'Arm': '11', 'NetReward': 49415, 'Pull': 2049, 'Reward': 20}
{'Arm': '2', 'NetReward': 49451, 'Pull': 2051, 'Reward': 9}
{'Arm': '0', 'NetReward': 53588, 'Pull': 2256, 'Reward': 27}
```

In [98]:

```
result
```

Out[98]:

	arm	reward	tries
0	0	0	0
1	1	0	0
2	2	0	0
3	3	0	0
4	4	0	0
5	5	0	0
6	6	0	0
7	7	0	0
8	8	0	0
9	9	0	0
10	10	0	0
11	11	0	0
12	12	0	0
13	13	0	0
14	14	0	0

15	15	0	0
16	16	0	0
17	17	0	0
18	18	0	0
19	19	0	0
20	20	0	0
21	21	0	0
22	22	0	0
23	23	0	0

In [99]:

```
def uniform_bandit(n):
    for j in range(4):
        for arm in arms:
            output_dict = pull('user24','IyqHJZcK',arm)
            result['reward'].loc[result['arm'] == arm] += output_dict['Reward']
            result['tries'].loc[result['arm'] == arm] += 1

    leftover_n = n - (3*len(arms))
    winning_arm = result['arm'].loc[result['reward'] == result['reward'].max()].values[0]
    for i in range(leftover_n):
        output_dict = pull('user24','IyqHJZcK',winning_arm)
        result['reward'].loc[result['arm'] == winning_arm] += output_dict['Reward']
        result['tries'].loc[result['arm'] == winning_arm] += 1
    return result
```

In [100…

```
uniform_bandit(180)
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Out[100…

	arm	reward	tries
0	0	2769	112
1	1	65	4
2	2	29	4
3	3	83	4
4	4	27	4
5	5	57	4
6	6	26	4
7	7	72	4
8	8	101	4
9	9	44	4
10	10	60	4
11	11	85	4
12	12	44	4
13	13	44	4
14	14	78	4
15	15	28	4

16	16	35	4
17	17	84	4
18	18	78	4
19	19	59	4
20	20	82	4
21	21	94	4
22	22	55	4
23	23	79	4

In [101…

```
result.to_csv('reward_pull_2.csv')
```

In [93]:

```
result
```

Out[93]:

	arm	reward	tries
0	0	0	0
1	1	0	0
2	2	0	0
3	3	0	0
4	4	0	0
5	5	0	0
6	6	0	0
7	7	0	0
8	8	0	0

9	9	0	0
10	10	0	0
11	11	0	0
12	12	0	0
13	13	0	0
14	14	0	0
15	15	0	0
16	16	0	0
17	17	0	0
18	18	0	0
19	19	0	0
20	20	0	0
21	21	0	0
22	22	0	0
23	23	0	0

In [94]:

```
arm = '0'
output_dict = pull('user24','IyqHJZcK',arm)
result['reward'].loc[result['arm'] == arm] += output_dict['Reward']
result['tries'].loc[result['arm'] == arm] += 1
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

In [96]:

```
output_dict
```

Out[96]:

```
{'Arm': '0', 'NetReward': 53588, 'Pull': 2256, 'Reward': 27}
```

In [ ]:

$\epsilon - Greedy$

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import random

import requests

def pull(user_group, secret_key, arm):
        url = ('http://10.243.255.29:5787/pull_arm/%s/%s/%s' % (user_group,secret_key,str(arm)))

        while True:
                r = requests.get(url)
                if r.ok:
                        output = r.json()['result']
                        return(output)
```

In [2]:

```
arms = [str(x) for x in range(24)]
rewards = [0]*24
tries = [0]*24
mean_reward = [0]*24

result = pd.DataFrame({'arm': arms, 'reward': rewards, 'tries': tries, 'mean_reward': mean_reward})
```

In [3]:
```
def epsilon_greedy(n, p):
    winning_arm = str(random.randint(0,23))
    arms.remove(winning_arm)

    for i in range(n):
        if np.random.uniform() <= p:
            output_dict = pull('user24','IyqHJZcK', random.choice(arms))

        else:
            output_dict = pull('user24','IyqHJZcK', winning_arm)

        result['reward'].loc[result['arm'] == output_dict['Arm']] += output_dict['Reward']
        result['tries'].loc[result['arm'] == output_dict['Arm']] += 1
        result['mean_reward'].loc[result['arm'] == output_dict['Arm']] = result['reward']/result['tries']

        new_winning_arm = result['arm'].loc[result['mean_reward'] == result['mean_reward'].max()].values[0]

        if new_winning_arm in arms:
            arms.remove(new_winning_arm)
            arms.append(winning_arm)

            winning_arm = new_winning_arm

        else:
            continue


    return result
```

In [4]:
```
epsilon_greedy(180, 0.3)
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/indexing.py:1637: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

Out[4]:

	arm	reward	tries	mean_reward
0	0	2277	90	25.300000

1	1	0	0	0.000000
2	2	26	4	6.500000
3	3	35	2	17.500000
4	4	12	2	6.000000
5	5	30	2	15.000000
6	6	6	1	6.000000
7	7	66	4	16.500000
8	8	89	4	22.250000
9	9	18	2	9.000000
10	10	15	1	15.000000
11	11	43	2	21.500000
12	12	48	4	12.000000
13	13	17	2	8.500000
14	14	31	2	15.500000
15	15	45	6	7.500000
16	16	45	7	6.428571
17	17	15	1	15.000000
18	18	48	3	16.000000
19	19	23	1	23.000000
20	20	638	30	21.266667
21	21	84	4	21.000000
22	22	40	3	13.333333
23	23	67	3	22.333333

In [5]:
```
random.randint(0,23)
```

Out[5]: 2

In [7]:
```
result.to_csv('greedy_result.csv')
```

In [ ]:
```
```

UCB1

In [ ]:
```
import pandas as pd,\
  utils.server_pull as usp,\
  numpy as np
```

Intialise desired pulls, API secret

In [ ]:
```
desired_pulls = 2000
team = ''
group_key = ''
```

Create function for getting top linUCB index of all arms

In [ ]:
```
def get_max_ucb_of_arms(input_df, input_global_round):
# Get the mean reward of each arm, and count of runs so far
# TODO: currently not safe for historical runs, would need indexing on inputdf
  df_mean_reward = input_df.pivot_table(
    index='arm',
    values=['arm_reward','arm_pull'],
    aggfunc={'arm_reward': np.mean, 'arm_pull': np.max }
  ).reset_index()
  df_mean_reward['ucb_index'] = df_mean_reward.apply(
    lambda x: x['arm_reward'] + np.sqrt(
      (2 * np.log( input_global_round ) ) / (x['arm_pull'] + 1)
    ),
    axis=1
  )
  return df_mean_reward['ucb_index'].idxmax()
# TODO: there is a non trivial circumstance where this can return multiple rows, rather than a scalar;
#   could be improved with getting random as typebreaker
```

Get historical data

In [ ]:

```
df_historical_pulls = pd.read_csv(
  './data/pulls.csv',
  header='infer',
  index_col=False
)
```

Get current arm pulls so far

In [ ]:

```
existing_arm_pulls = df_historical_pulls['global_pull'].max()
```

safety check if in exploration phase

In [ ]:

```
if existing_arm_pulls >= 23:
  # pre compute UCB for best arm
  target_arm = get_max_ucb_of_arms( df_historical_pulls, existing_arm_pulls )
  for i in range(existing_arm_pulls+1,existing_arm_pulls+desired_pulls+1):
    # pull arm, get output
    arm_output = usp.pull(team,group_key,target_arm)
    arm_pull_count = df_historical_pulls[df_historical_pulls['arm'] == target_arm  ]['arm_pull'].count()
    # append output
    df_historical_pulls = df_historical_pulls.append(
      {'arm_pull': arm_pull_count, 'arm':target_arm,'global_pull':i ,'arm_reward':arm_output['Reward'] },
      ignore_index=True
    )
    # get next arm
    target_arm = get_max_ucb_of_arms( df_historical_pulls,  i )
else:
  raise
```

write out results

In [ ]:
```
df_historical_pulls.to_csv(
  './data/pulls_output.csv',
  index=False
)
```

TODO: get arm pulls for exploration phase

Analysis

In [1]:
```
import pandas as pd,\
  seaborn as sns,\
  matplotlib.pyplot as pypl,\
  numpy as np

#  plotly.express as px,\
```

In [2]:
```
pypl.style.use('bmh')
```

In [3]:
```
df_historical_pulls = pd.read_csv(
  '../dba5101_gp3/data/pulls_output.csv',
  header='infer',
  index_col=False
)
```

In [4]:
```
## add_uniform_algo

df_uniform_pulls_input = pd.read_json(
  '../dba5101_gp3/data/uniform_pulls_1.json',
  orient='records',
  lines=True
)
```

In [5]:
```
## add_greed_algo

df_greedy_pulls_input = pd.read_json(
  '../dba5101_gp3/data/greedy_pull_3.json',
  orient='records',
  lines=True
)
```

In [6]:
```
df_greedy_pulls_input['index_pull'] = df_greedy_pulls_input['Pull'].rank() - 1
df_uniform_pulls_input['index_pull'] = df_uniform_pulls_input['Pull'].rank() - 1
```

In [7]:
```
df_greedy_pulls = df_greedy_pulls_input[df_greedy_pulls_input['index_pull'] >= 1]
```

In [8]:
```
df_uniform_pulls =  df_uniform_pulls_input[df_uniform_pulls_input['index_pull'] >= (24*3 )]
```

In [9]:
```
df_ucb1_exploit = df_historical_pulls[df_historical_pulls['global_pull'] >= 24 ]
```

In [ ]:
```

```

In [10]:
```
df_uniform_pulls['local_pull'] = df_uniform_pulls['Pull'].rank() - 1
```

```
/var/folders/88/st0km2xx06b1nm8trg753nqm0000gn/T/ipykernel_19265/1785843218.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  df_uniform_pulls['local_pull'] = df_uniform_pulls['Pull'].rank() - 1
```

In [ ]:

In [11]:

```
df_greedy_pulls['local_pull'] = df_greedy_pulls['Pull'].rank() - 1
```

```
/var/folders/88/st0km2xx06b1nm8trg753nqm0000gn/T/ipykernel_19265/3050684879.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_greedy_pulls['local_pull'] = df_greedy_pulls['Pull'].rank() - 1
```

In [12]:

```
df_greedy_pulls['rolling_sum'] = df_greedy_pulls['Reward'].cumsum()

df_uniform_pulls['rolling_sum'] = df_uniform_pulls['Reward'].cumsum()

df_historical_pulls['rolling_sum'] = df_historical_pulls['arm_reward'].cumsum()
```

```
/var/folders/88/st0km2xx06b1nm8trg753nqm0000gn/T/ipykernel_19265/4265629572.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_greedy_pulls['rolling_sum'] = df_greedy_pulls['Reward'].cumsum()
/var/folders/88/st0km2xx06b1nm8trg753nqm0000gn/T/ipykernel_19265/4265629572.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_uniform_pulls['rolling_sum'] = df_uniform_pulls['Reward'].cumsum()
```

In [13]:

```
df_ucb1_exploit['rolling_sum'] = df_ucb1_exploit['arm_reward'].cumsum()
```

```
/var/folders/88/st0km2xx06b1nm8trg753nqm0000gn/T/ipykernel_19265/3231549739.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  df_ucb1_exploit['rolling_sum'] = df_ucb1_exploit['arm_reward'].cumsum()
```

In [14]:

```
df_comparative = df_ucb1_exploit.merge(
  right=df_uniform_pulls,
  left_on ='global_pull',
  right_on = 'local_pull',
  how='left' ,
  suffixes=('_ucb1', '_uniform')
)

df_comparative = df_comparative.merge(
  right=df_greedy_pulls,
  left_on ='global_pull',
  right_on = 'local_pull',
  how='left' ,
  suffixes=('', '_greedy')
)
```

In [15]:

```
df_comparative
```

Out[15]:

	global_pull	arm	arm_pull	arm_reward	next_chosen_arm	rolling_sum_ucb1	Arm	NetReward	Pull	Reward	index_pull	local_pu
0	24.0	3.0	1.0	17.0	NaN	17.0	0	194485	8959	27	96.0	24.
1	25.0	8.0	1.0	24.0	NaN	41.0	1	194501	8960	16	97.0	25.
2	26.0	8.0	2.0	19.0	NaN	60.0	2	194509	8961	8	98.0	26.
3	27.0	0.0	1.0	18.0	NaN	78.0	3	194533	8962	24	99.0	27.
4	28.0	8.0	3.0	20.0	NaN	98.0	4	194541	8963	8	100.0	28.
...	...	...	...	...	...	...	...	...	...	...	...	.
1995	2019.0	0.0	1821.0	22.0	NaN	48414.0	0	242273	10954	22	2091.0	2019.
1996	2020.0	0.0	1822.0	27.0	NaN	48441.0	0	242295	10955	22	2092.0	2020.
1997	2021.0	0.0	1823.0	31.0	NaN	48472.0	0	242315	10956	20	2093.0	2021.
1998	2022.0	0.0	1824.0	20.0	NaN	48492.0	0	242337	10957	22	2094.0	2022.
1999	2023.0	0.0	1825.0	29.0	NaN	48521.0	0	242366	10958	29	2095.0	2023.

2000 rows × 20 columns

In [19]:
```
df_comparative [df_comparative['global_pull'] <= 2000 ] [
  ['global_pull','rolling_sum_ucb1','rolling_sum_uniform','rolling_sum']
].rename(columns={'rolling_sum':'greedy_epsilon','rolling_sum_uniform':'uniform','rolling_sum_ucb1':'ucb1'}) .plot
  kind ='line',
  x = 'global_pull'
)



pypl.title("Comparision of UCB1 +  Uniform MAB + Greedy Episilon - Exploitation Phase Cumulative Reward")

pypl.gcf().set_size_inches(17, 10)

pypl.ylabel('Cumulative Reward')
pypl.xlabel('Pull')

#pypl.show()

pypl.savefig(fname='./model_cumulative_sum.png')

pypl.close('all')
```

In [ ]:

In [ ]:

In [4]:
```
# Add in historical pulls over all time

df_total_pulls = df_historical_pulls.groupby(['arm'])['global_pull'].count().reset_index()

df_total_pulls.rename(columns={"global_pull": "total_pulls"},inplace=True)

df_historical_pulls =  df_historical_pulls.merge(right =df_total_pulls , how ='left', on ='arm' )
```

In [5]:
```
df_arms = pd.DataFrame( {'arm': df_historical_pulls['arm'].unique().tolist()  } )

df_pulls = pd.DataFrame( {'global_pull': df_historical_pulls['global_pull'].unique().tolist()  } )

df_cartesian = df_arms.merge(right=df_pulls, how='cross')

df_cartesian= df_cartesian.merge( right=df_historical_pulls, on=['global_pull','arm'] ,how='left' )

df_cartesian['arm_reward'].fillna(value=0,inplace=True)

df_cartesian['cumulative_reward'] = df_cartesian.groupby(['arm'])['arm_reward'].cumsum(skipna=True)

df_cartesian['arm_pull'].ffill(inplace=True)
```

In [6]:
```
df_cartesian = df_cartesian[df_cartesian['global_pull'] >= 24]
```

In [7]:

```
#df_cartesian['ucb_index'] =
df_cartesian['contemporary_linucb'] = (df_cartesian['cumulative_reward'] / (df_cartesian['arm_pull'] + 1) )  + np.
  (2 * np.log( df_cartesian['global_pull']  ) ) / (df_cartesian['arm_pull'] + 1)
)

df_cartesian['ucb_mean'] = (df_cartesian['cumulative_reward'] / (df_cartesian['arm_pull'] + 1) )

df_cartesian['penalty'] = np.sqrt(
  (2 * np.log( df_cartesian['global_pull']  ) ) / (df_cartesian['arm_pull'] + 1)
)
```

In [8]:

```
sns.lineplot(
  x='global_pull',
  y='contemporary_linucb',
  hue='arm',
  data=df_cartesian[
    (df_cartesian['global_pull'] <= 220)
    & (df_cartesian['global_pull'] >= 24)
    & (df_cartesian['arm'].isin([0,3,8,23]) )
  ]
)

pypl.title("UCB Over Time")

pypl.gcf().set_size_inches(17, 10)

pypl.ylabel('UCB Index')
pypl.xlabel('Pull')

#pypl.show()

pypl.savefig(fname='./ucb_exploitation_line.png')

pypl.close('all')
```

In [9]:
```
df_cartesian[
  df_cartesian['global_pull'] == 2020
].sort_values(
  by=['contemporary_linucb'],
  ascending=False
)[['arm','arm_pull','cumulative_reward','contemporary_linucb','ucb_mean','penalty']].head(n=5)
```

Out[9]:

	arm	arm_pull	cumulative_reward	contemporary_linucb	ucb_mean	penalty
2020	0.0	1822.0	44651.0	24.584520	24.493143	0.091377
16188	7.0	0.0	19.0	22.901500	19.000000	3.901500
42500	20.0	0.0	19.0	22.901500	19.000000	3.901500
20236	9.0	0.0	19.0	22.901500	19.000000	3.901500
36428	17.0	1.0	40.0	22.758777	20.000000	2.758777

In [10]:

```
sns.lineplot(
  x='global_pull',
  y='contemporary_linucb',
  hue='arm',
#  color='b',
  data=df_cartesian[
    (df_cartesian['global_pull'] >= 24)
    & (df_cartesian['arm'].isin([0,7,17]) )
  ]
)

pypl.title("UCB Over Time - Candidates For Next Arm, Due To Uncertainity")

pypl.gcf().set_size_inches(17, 10)

pypl.ylabel('UCB Index')
pypl.xlabel('Pull')

#pypl.show()
pypl.savefig(fname='./ucb_future_line.png')

pypl.close('all')
```

In [11]:

```
f, ax = pypl.subplots(figsize=(17, 10))

sns.lineplot(
  x='global_pull',
  y='ucb_mean',
  hue='arm',
#  color='b',
  data=df_cartesian[
    (df_cartesian['global_pull'] >= 24)
    & (df_cartesian['arm'].isin([0,7,17]) )
  ]
)

sns.lineplot(
  x='global_pull',
  y='penalty',
  hue='arm',
  data=df_cartesian[
    (df_cartesian['global_pull'] >= 24)
    & (df_cartesian['arm'].isin([0,7,17]) )
  ]
)


pypl.title("Arm Mean & Penalty Term Over Time")

pypl.ylabel('UCB Index Component Quantity')
pypl.xlabel('Pull')


#pypl.show()
pypl.savefig(fname='./ucb_penalty_line.png')

pypl.close('all')
```

In [12]:

```
#Create combo chart
fig, ax1 = pypl.subplots(figsize=(17,10))


#bar plot creation
ax1.set_title('UCB Index & Reward - Exploitation Phase', fontsize=16)
ax1.set_xlabel('Pull')
ax1.set_ylabel('Reward')
ax1 = sns.scatterplot(x='global_pull', y='arm_reward', hue='arm', data = df_historical_pulls[
(df_historical_pulls['global_pull'] <= 250)
& (df_historical_pulls['global_pull'] >= 24)
#& (df_historical_pulls['arm'].isin([0,3,8,23]) )
])
ax1.tick_params(axis='y')
#specify we want to share the same x-axis
ax2 = ax1.twinx()

#line plot creation
ax2.set_ylabel('UCB Index')
ax2 = sns.lineplot(
  x='global_pull',
  y='contemporary_linucb',
  hue='arm',
  data=df_cartesian[
    (df_cartesian['global_pull'] <= 250)
    & (df_cartesian['global_pull'] >= 24)
#     & (df_cartesian['arm'].isin([0,3,8,23]) )
  ]
)


#show plot
#pypl.show()
pypl.savefig(fname='./ucb_index_reward_line.png')

pypl.close('all')
```

In [48]:
```
#sns.scatterplot( x='global_pull', y='arm_reward',   hue='arm',
#  data = df_historical_pulls[   (df_historical_pulls['global_pull'] <= 250)
#    & (df_historical_pulls['global_pull'] >= 24) ])

#pypl.gcf().set_size_inches(17, 10)

#pypl.show()

#pypl.close('all')
```

In [47]:
```
#sns.displot(  data=df_historical_pulls[ df_historical_pulls['global_pull'] >= 24] ,
#  x="arm_reward", hue="arm", kind="kde")

#pypl.gcf().set_size_inches(17, 10)

#pypl.show()

#pypl.close('all')
```

In [13]:

```
sns.violinplot(
  cut=0,
  inner='quartiles',
  data=df_historical_pulls[ (df_historical_pulls['global_pull'] >= 24) & (df_historical_pulls[ 'total_pulls'] > 2
  x='arm',
  y='arm_reward',
)

pypl.gcf().set_size_inches(17, 10)
pypl.ylim([0,45])

pypl.title("Arm Reward Violinplot, Exploitation Phase - With Quartiles")
pypl.ylabel('Reward')
pypl.xlabel('Arm')

#pypl.show()
pypl.savefig(fname='./reward_violin.png')

pypl.close('all')
```