**Purpose:**

Model the ad auction and develop an effective algorithm for bidding in a landscape of strategic competitors.

**Building a Second-Price Auction (Modeling and Strategy)**

You have been hired by a major retailer to develop algorithms for an online ad auction. Your client knows a little about the multi-armed bandit literature and recognizes that it can spend money to explore and learn how likely users are to click on ads or to exploit, spending on the most promising users to maximize immediate payoffs. At the same time, other companies are participating in auctions that may outbid your client, potentially interfering with these goals.

Your task is to model the ad auction and develop an effective algorithm for bidding in a landscape of strategic competitors. Your client plans to test your bidding algorithm against other bidding algorithms contributed by other data scientists in order to select the most promising algorithm.

**The Auction Rules**

The Auction is a game involving a set of Bidders on one side and a set of Users on the other. Each round represents an event where a user navigates to a website with a space for an ad. When this happens, the Bidders will place bids, and the winner gets to show their ad to the User. The user may click on the ad or not click, and the winning Bidder gets to observe the user's behavior. This is a second-price sealed-bid Auction. There are num\_users Users, numbered from 0 to num\_users - 1. The number corresponding to a user will be called by its user\_id \_. Each user has a secret probability of clicking whenever an ad is shown; the probability is the same, no matter which Bidder gets to show the ad, and the probability never changes. The events of clicking on each ad are mutually independent. When a user is created, the secret probability is drawn from a uniform distribution from 0 to 1.

There is a set of Bidders. Each Bidder begins with a balance of 0 dollars. The objective is to finish the game with as high a balance as possible. At some points during the game, the Bidder's balance may become negative. If your Bidder's balance goes below -1000 dollars, then your bidder will be disqualified from the Auction and further bidding.

**A user class that includes:**

• an initializer method with the definition def \_\_init (self)

• a private \_probability attribute to represent the probability of clicking on an ad. When a

user is created, the secret probability is drawn from a uniform distribution from 0 to 1

• a show\_ad method with the definition def that represents showing an ad to this user \_ This method should retum True to represent the user clicking on and ad and False otherwise.

**A Bidder class that includes:**

• an initializer with the definition **def \_\_init\_\_ (self, num\_users, num\_rounds**) , in which **num\_users** contains the number of **user** objects in the game, and **num\_rounds** contains the total number of rounds to be played- The Bidder might want to use this info to help plan its strategy.

• a **bid** method with the definition **def bid(self, user\_id)** , which retums a non-negative amount of money, in dollars round to three (3) decimal places.

• a **notify method** with the definition def notify**(self, auction\_winner, price, clicked)**, which is used to send information about what happened in a round back to the **Bidder**. Here, **auction\_winner** is a boolean to represent whether the given **Bidder** won the auction **( True )** or not **( False**). **price** is the amount of the second bid, which the winner

pays. If the given **Bidder** won the auction, **clicked** will contain a boolean value representing whether the **user** clicked on the ad. If the given Bidder did not win the auction,

**clicked** will always contain **None**.

**An Auction class that includes:**

• an initializer with the definition **def \_\_ init\_\_ (self, users, bidders)** \_ Here, **users** is expected to contain a list of all user objects. **bidders** are expected to contain a list of all **Bidder** objects.

• an **execute\_round method** with the header **def execute\_round** (self). This method should execute all steps within a single round of the game.

• a **balances** attribute, which contains a dictionary of the current balance of every Bidder with the initial bid being 0

• a plot\_history method with the definition def plot\_history(self), which creates a visual representation of how the auction has proceeded.

**Code Documentation**

1. **import random**: Imports the **random** module, which is used for generating random numbers.
2. **import matplotlib.pyplot as plt**: Imports the **pyplot** module from the **matplotlib** library and aliases it as **plt** for easier reference.

Then, there are three class definitions:

**Class User:**

* Represents a user with a secret probability of clicking an ad.
* **\_\_init\_\_(self)**: Initializes a User object with a secret probability of clicking an ad.
* **\_\_repr\_\_(self)**: Returns a string representation of the User object.
* **\_\_str\_\_(self)**: Returns a more descriptive string representation of the User object.
* **show\_ad(self)**: Simulates the user clicking on an ad based on their probability.
* Attributes:
  + **\_\_probability**: Secret probability of clicking an ad.

**Class Bidder:**

* Represents a bidder in an online second-price ad auction.
* **\_\_init\_\_(self, num\_users, num\_rounds)**: Initializes a Bidder object with the specified number of users and rounds.
* **\_\_repr\_\_(self)**: Returns a string representation of the Bidder object.
* **\_\_str\_\_(self)**: Returns a more descriptive string representation of the Bidder object.
* **bid(self, user\_id)**: Places a bid for a user based on their ID.
* **notify(self, auction\_winner, price, clicked)**: Receives notification about the auction outcome and updates the bidder's strategy.
* Attributes:
  + **num\_users**: Number of user objects in the game.
  + **num\_rounds**: Total number of rounds to be played.
  + **balance**: Initial balance of the bidder.
  + **user\_click\_prob\_estimates**: List of click probability estimates for each user.
  + **bid\_history**: List to keep track of all bids placed.
  + **is\_qualified**: Boolean indicating if the bidder is qualified.

**Class Auction:**

* Represents an online second-price ad auction.
* **\_\_init\_\_(self, users, bidders)**: Initializes the Auction with lists of User and Bidder objects.
* **\_\_repr\_\_(self)**: Returns a string representation of the Auction object.
* **\_\_str\_\_(self)**: Returns a more descriptive string representation of the Auction object.
* **execute\_round(self)**: Executes a single round of the auction.
* **plot\_history(self)**: Creates a plot of balance history for each bidder.
* Attributes:
  + **users**: List of User objects participating in the auction.
  + **bidders**: List of Bidder objects participating in the auction.
  + **balances**: Dictionary to store balances for each bidder in the auction.

return random.random() <= self.\_\_probability: This line returns **True** if a randomly generated number between 0 and 1 is less than or equal to the **\_\_probability** attribute of the **User** object, otherwise, it returns **False**.

**These lines calculate a safe bid amount based on the bidder's balance and the number of rounds left.**

**rounds\_left = max(self.num\_rounds - len(self.bid\_history), 1)**

**safe\_bid\_amount = min(0.2, self.balance / (rounds\_left \* 2))**

**These lines calculate the bid amount based on the estimated click probability of the user.**

**estimated\_click\_prob = self.user\_click\_prob\_estimates[user\_id]**

**dynamic\_bid = safe\_bid\_amount \* estimated\_click\_prob**

**bid\_amount = max(dynamic\_bid, minimum\_bid)**

**This line retrieves the user ID from the bidder's bid history.**

**user\_id = self.bid\_history[-1][0] if self.bid\_history else 0**

**The click probability estimate for the user is updated based on the adjustment factor.**

**self.user\_click\_prob\_estimates[user\_id] += adjustment\_factor**

**The updated click probability estimate is constrained to be within the range of 0 to 1.**

**self.user\_click\_prob\_estimates[user\_id] = min(max(self.user\_click\_prob\_estimates[user\_id], 0), 1)**