Real-Time Bidding (RTB) System for Digital Advertising

Team: Access Denied

Members:

1. **Samay Jain (23/MC/128)**
2. **Ibrahim Haneef (23/IT/72)**
3. **Mohd. Hassan (23/IT/100)**
4. **Hemant Singh (23/IT/69)**

Table of Contents

1. Introduction

This project develops a real-time bidding (RTB) system for digital advertising. Our primary goal is to build a bidding engine that uses historical data to predict:

- **CTR (Click-Through Rate):** The probability that an ad impression leads to a click.
- **CVR (Conversion Rate):** The probability that a click results in a conversion (e.g., sale or signup).
- **Market Price:** The expected price paid for an impression in a second-price auction.

Using these predictions, the system computes an expected value (EV) for each bid request and makes bid decisions under a fixed budget.:

$$\textbf{Score} = \textbf{Clicks} + N \times \textbf{Conversions}$$

2. Problem Statement

In real-time bidding, decisions must be made in milliseconds, with limited budget and incomplete future knowledge. The system must decide:

- **Whether to bid on a given impression.**
- **At what price to bid.**

The decision is based on the trade-off between bidding costs and the expected value from clicks and conversions. Our system leverages historical bidding logs to predict the outcomes and set bid prices accordingly, all while respecting budget constraints.

3. Dataset Description

- **Data Sources:**
  - Bid logs, impression logs, click logs, and conversion logs.
- **Format:**
  - Data is provided as plain text files with tab-separated values.
- **Key Fields:**
  - `BidID`: **Primary key** used to uniquely identify each bid request.
  - `Timestamp`, `VisitorID`, `User-Agent`, `IP`, `Region`, `City`, `Adexchange`, `Domain`, `URL`, `AdslotID`, `Adslotwidth`, `Adslotheight`, `Adslotvisibility`, `Adslotformat`, `Adslotfloorprice`, `CreativeID`, `Biddingprice`, `Payingprice`, `KeypageURL`, `AdvertiserID`.
- **Volume:**
  - Approximately 9.6 million bids, 1.8 million impressions (18.93%), 1,159 clicks (≈0.0121%), and 38 conversions (≈0.0004%). (*Only for Day 06.06.2023)*

4. Data Processing & Feature Engineering

4.1 Data Loading & Primary Key Usage

- **Primary Key:**
  - `BidID` is used as the unique identifier to merge data from bid, impression, click, and conversion logs.
- **Chunked Loading:**
  - Data is loaded in chunks to handle memory constraints.
  - Each file is read with a `chunksize` of **100,000 rows** to efficiently process large datasets.

4.2 Chunk Processing & Batch Sizes

- **Batch Sizes:**
  - **100,000 rows** per chunk for initial data loading.
  - **50,000 rows** per chunk for feature extraction in the `extract_features()` function.
- **Memory Optimization:**
  - Data types are optimized (e.g., converting integers to `np.uint8` or `np.int32`, floats to `np.float32`) to reduce memory usage.

4.3 Feature Engineering: Design and Justification

The following features are engineered based on domain knowledge and empirical analysis:

- **Time Features:**
  - **Timestamp Conversion:** Converted from string format (e.g., '20231010120000000') to a datetime object.
  - **Hour, Day of Week, Is Weekend:**
    - *Hour:* Reflects user activity during the day.
    - *Day of Week:* Captures weekday vs. weekend behavior.
    - *Is Weekend:* Boolean flag (1 if Saturday or Sunday, 0 otherwise).
- **User-Agent Features:**
  - **is_mobile:** Checks if the User-Agent string contains “Mobile”, “Android”, or “iOS.”
  - **is_chrome, is_firefox, is_safari:** Flags for popular browsers.
  - **Justification:** Browser and device type influence ad viewability and interaction.
- **Ad Slot Features:**
  - **Adslotwidth, Adslotheight:** Converted to numerical values (float32) and scaled.
  - **ad_area:** Computed as `Adslotwidth * Adslotheight` to capture the ad size.
  - **is_premium_ad:** Set to 1 if `ad_area` ≥ 100,000 pixels, which might correlate with higher-quality placements.

    - **Adslotfloorprice:** The minimum price set by the publisher, used as a baseline for bidding.
    - **Justification:** Ad dimensions and floor price directly affect the cost and performance of an ad impression.
- **Categorical Features:**

    - **Columns:** `Region`, `City`, `Adexchange`, `Domain`, `URL`, `AdslotID`, `Adslotvisibility`, `Adslotformat`, `CreativeID`, `AdvertiserID`
    - **Transformation:** These are label encoded to convert string identifiers into numerical values.
    - **Justification:** Categorical variables often capture essential market or contextual information that can influence ad performance.

5. Modeling Approach

We trained three separate models using LightGBM, chosen for its efficiency, speed, and accuracy in handling large datasets and imbalanced classes.

5.1 CTR Prediction Model

- **Type:** LightGBM Classifier
- **Parameters:**
    - `n_estimators=100`, `learning_rate=0.05`, `max_depth=7`, `num_leaves=31`, `min_child_samples=20`, `class_weight='balanced'`, `random_state=42`
- **Training Data:**
    - Trained on all bid data with label `is_click`.
- **Evaluation:**
    - **AUC:** 0.8803
    - **PR-AUC:** 0.0049
- **Justification:**
    - Despite a low PR-AUC (common in highly imbalanced datasets), the AUC indicates good ranking capability.

5.2 CVR Prediction Model

- **Type:** LightGBM Classifier
- **Parameters:** Same as CTR model.
- **Training Data:**
    - Trained only on clicked impressions (where `is_click` == 1) to reduce noise.
- **Evaluation:**
    - **AUC:** 0.8516
    - **PR-AUC:** 0.5911

- **Justification:**
    - By focusing only on clicks, the model learns a more precise mapping for conversion prediction.

5.3 Market Price Prediction Model

- **Type:** LightGBM Regressor
- **Parameters:**
    - `n_estimators=100`, `learning_rate=0.05`, `max_depth=7`, `num_leaves=31`, `min_child_samples=20`, `random_state=42`
- **Training Data:**
    - Trained on won impressions (where `is_impression == 1`) with `Payingprice` as the target.
- **Evaluation:**
    - **RMSE:** ~47.92
- **Justification:**
    - Accurate market price estimation is crucial for bidding competitively while controlling costs.

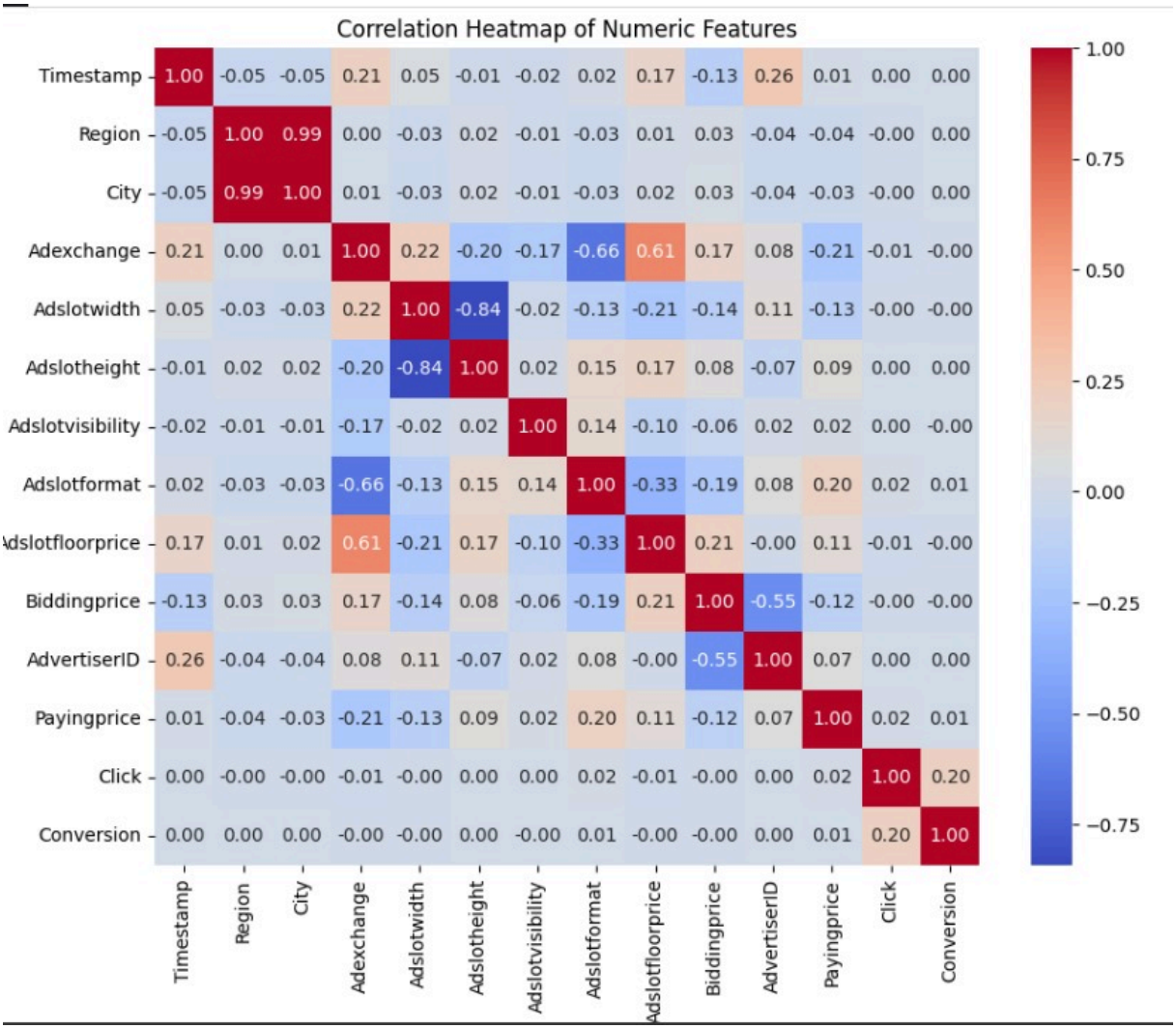
5.4 Model Training, Validation, and Evaluation

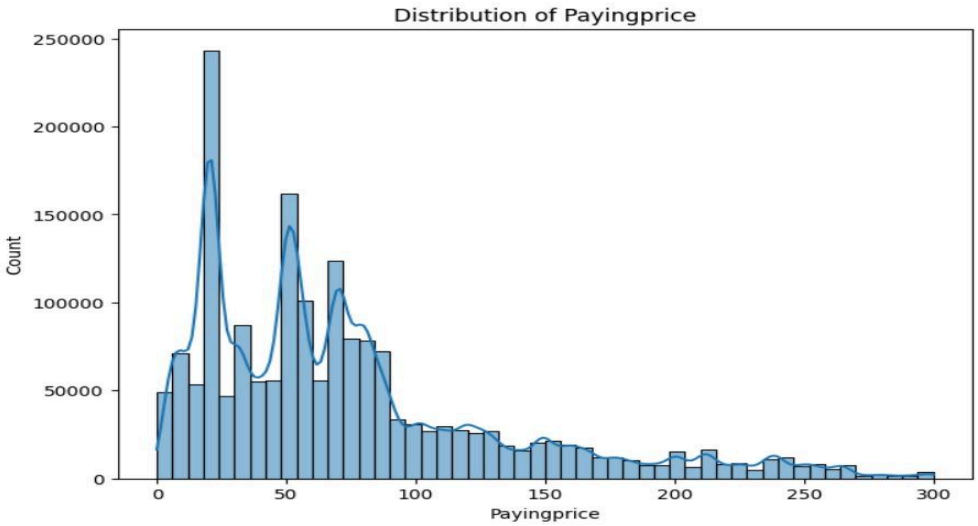
- **Data Splitting:**
    - **Train Set:** ~7,669,559 rows
    - **Validation Set:** ~958,695 rows
    - **Test Set:** ~958,695 rows
- **Early Stopping:**
    - Used with a stopping round of 10 iterations to prevent overfitting.
- **Evaluation Metrics:**
    - **CTR & CVR:** ROC AUC and PR-AUC.
    - **Market Price:** RMSE.
- **Justification:**
    - These metrics provide insight into both ranking performance and cost estimation accuracy.

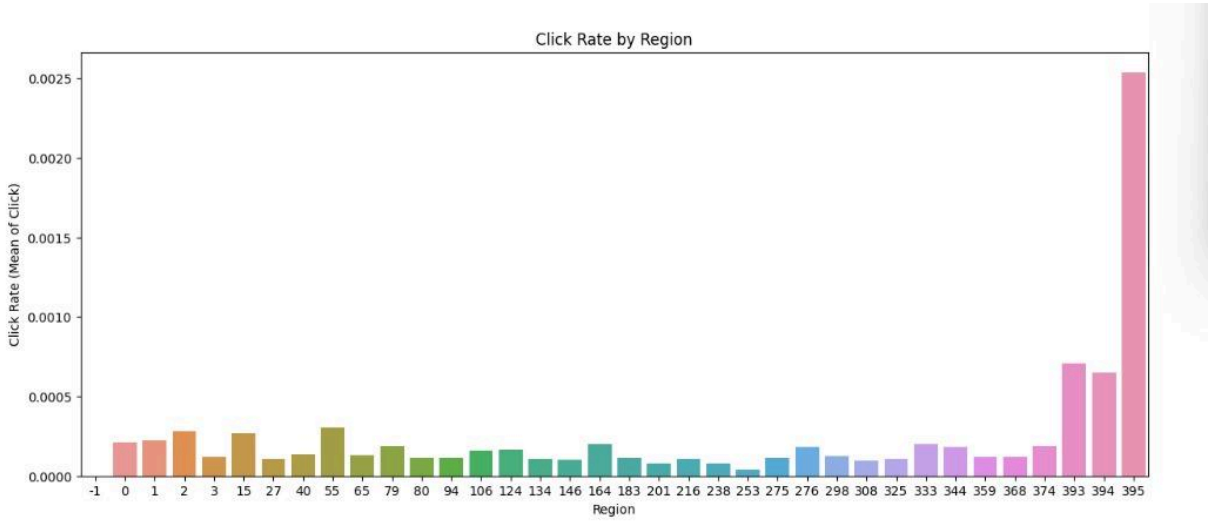
6. Exploratory Data Analysis (EDA)

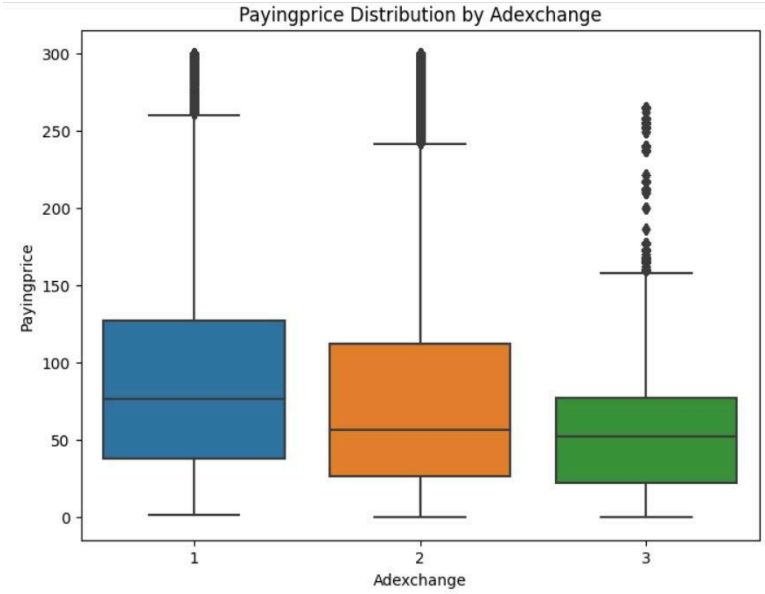
6.1 Initial Data Exploration

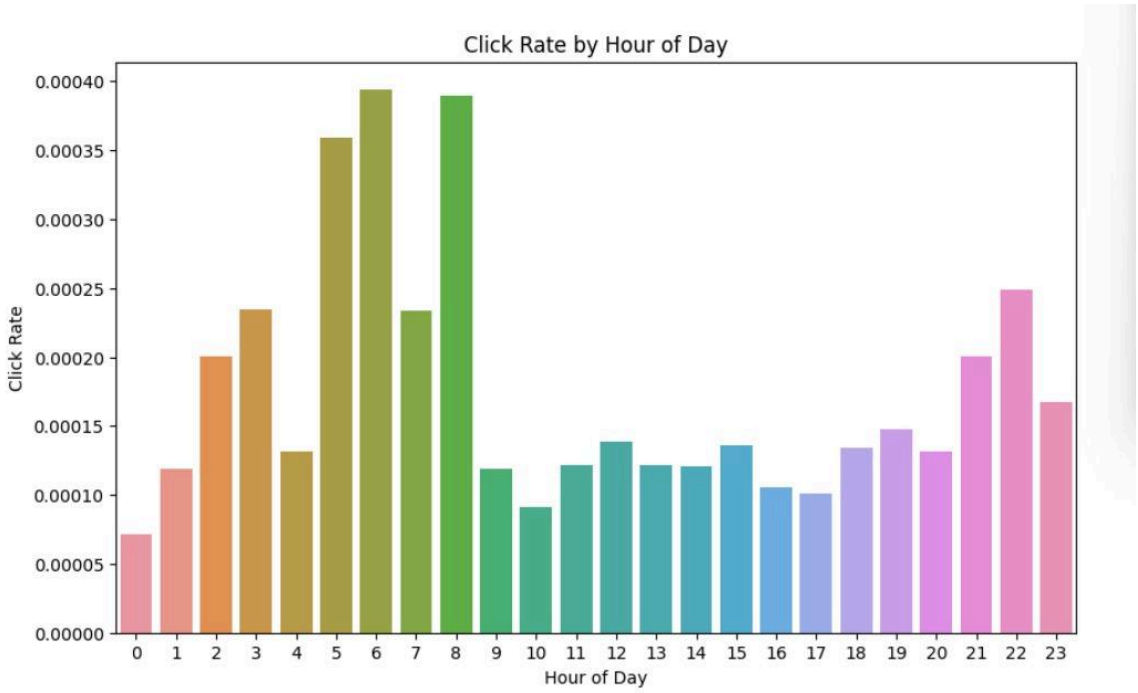
Objective: Understand the distribution and relationships of key variables.

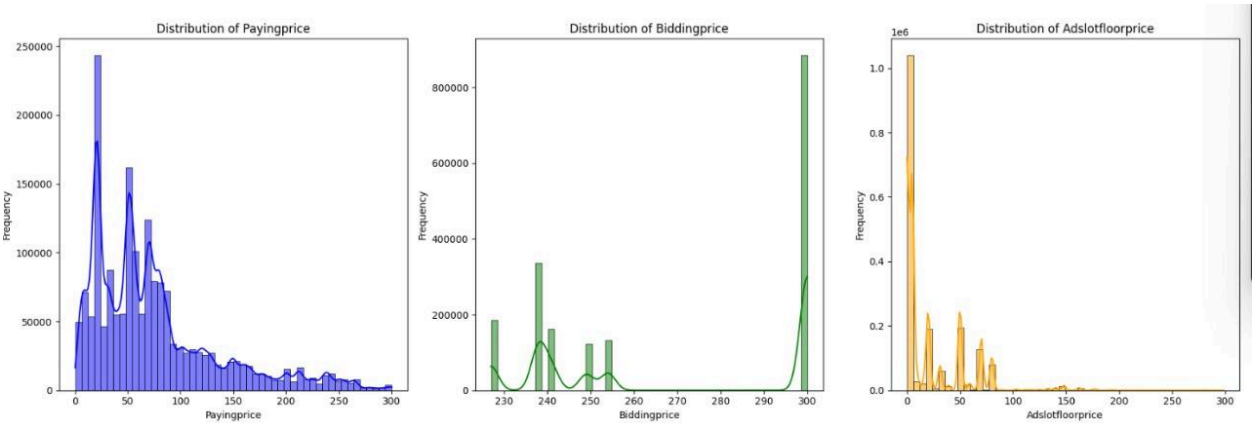



Distribution of Payingprice
Count
Payingprice
0
50000
100000
150000
200000
250000
0
50
100
150
200
250
300


Click Rate by Region
Click Rate (Mean of Click)
Region
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
-1 0 1 2 3 15 27 40 55 65 79 80 94 106 124 134 146 164 183 201 216 238 253 275 276 298 308 325 333 344 359 368 374 393 394 395


Payingprice Distribution by Adexchange
Payingprice
Adexchange
0
50
100
150
200
250
300
1
2
3


Click Rate by Hour of Day
Click Rate
Hour of Day
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23


Distribution of Payingprice
Frequency
Payingprice
Distribution of Biddingprice
Frequency
Biddingprice
Distribution of Adslotfloorprice
1e6
Frequency
Adslotfloorprice

7. System Architecture & Code Overview

7.1 RTBDataProcessor and Data Utility Functions

- **RTBDataProcessor:**
  - Reads bid, impression, click, and conversion logs in chunks of **100,000 rows**.
  - Uses `BidID` as the unique key to join the datasets.
  - Extracts features in chunks of **50,000 rows** to manage memory usage.
- **optimize_dtypes:**
  - Converts DataFrame columns to efficient types (e.g., from `int64` to `uint8`, `float64` to `float32`).

7.2 Model Training & Evaluation Code

- **train_and_evaluate():**
  - Merges features with labels (joined on `BidID`).
  - Splits the merged dataset into train, validation, and test sets.
  - Trains three LightGBM models (CTR, CVR, and Market Price) with early stopping.
  - Evaluates models using ROC AUC, PR-AUC, and RMSE.
  - Outputs top feature importance for the CTR model.

7.3 Online Bidding Engine Components

7.3.1 BidRequest.py

- **Function:**
  - Defines the `BidRequest` class that encapsulates all fields of a bid request.
- **Key Attributes:**
  - `bidId`, `timestamp`, `visitorId`, `userAgent`, `ipAddress`, `region`, `city`, `adExchange`, `domain`, `url`, `adSlotID`, `adSlotWidth`, `adSlotHeight`, `adSlotVisibility`, `adSlotFormat`, `adSlotFloorPrice`, `creativeID`, `advertiserId`, `userTags`.

7.3.2 Bidder.py

- **Function:**
  - Provides the abstract interface with the `getBidPrice(bidRequest: BidRequest) -> int` method.
- **Usage:**
  - The `Bid` class inherits from this interface.

7.3.3 Bid.py

- **Function:**
    1. Implements the bidding logic.
- **Key Steps:**
    1. **Initialization:**
        - Loads pre-trained models (`ctr_model.pkl`, `cvr_model.pkl`, `market_price_model.pkl`) and preprocessing objects (`label_encoders.pkl`, `scaler.pkl`).
        - Sets the campaign budget (e.g., $10,000) and advertiser-specific conversion weight (e.g., 10 for AdvertiserID "3476").
    2. **Feature Extraction:**
        - Converts an incoming `BidRequest` into a feature vector using a helper function (`extract_features_from_bidrequest`).
    3. **Prediction & Decision:**
        - Predicts CTR, CVR, and expected market price.
        - Computes the expected value (EV = CTR + N × CVR).
        - Compares EV (or the ratio EV/market_price) to a threshold (e.g., 0.05).
        - Returns a bid price slightly above the predicted market price (e.g., 2% higher) if conditions are met, otherwise returns -1.
    4. **Budget Pacing:**
        - Checks the remaining budget before bidding.

7.4 Testing Framework

- **Test Script (test_bid.py):**
    - Simulates a bid request by instantiating a `BidRequest` with sample data (e.g., realistic timestamp, ad slot sizes, and AdvertiserID "3476").
    - Creates an instance of the `Bid` class.
    - Calls the `getBidPrice()` method and prints the result.
- **Purpose:**
    - Ensures the bidding engine responds correctly under simulated conditions.
    - Helps verify that feature extraction, model prediction, and bid decision logic operate as intended.

8. Results & Evaluation

8.1 Performance Metrics

- **CTR Model:**
  - **ROC AUC:** 0.8803
  - **PR-AUC:** 0.0049
- **CVR Model:**
  - **ROC AUC:** 0.8516
  - **PR-AUC:** 0.5911
- **Market Price Model:**
  - **RMSE:** 47.92
- **Interpretation:**
  - The high AUC values indicate strong ranking performance.
  - The low PR-AUC for CTR reflects the severe class imbalance; however, the CVR model shows much improved precision-recall characteristics within the clicked impressions subset.

8.2 Further Testing

```
PS C:\Users\SAMAY-JAIN\Desktop\Adobe Hackathon> python test_bid.py
(1, 23)
CTR: 0.48416749088648886, CVR: 0.21124015735485296, Market Price: 87.83544419779706, EV: 2.596569064435019, Ratio: 0.029561745695596327

Bid Price: -1
(1, 23)
CTR: 0.44725753915127303, CVR: 0.7716489187266614, Market Price: 70.76715078599463, EV: 8.163746726417887, Ratio: 0.11536068127294954

Bid Price: 72
(1, 23)
CTR: 0.766672591135116, CVR: 0.3190189249107126, Market Price: 103.9387536332674, EV: 3.956861840242242, Ratio: 0.03806916767737515

Bid Price: -1
PS C:\Users\SAMAY-JAIN\Desktop\Adobe Hackathon>
```

9. Instructions for Running the Code

1. **Environment Setup:**
    - Python Version: 3.9

      Install dependencies using:
      pip install -r requirements.txt

2. **Training the Models:**
    - Run your training script (e.g., `python train_and_evaluate.py`) to produce:
        - `ctr_model.pkl`
        - `cvr_model.pkl`
        - `market_price_model.pkl`
        - `label_encoders.pkl`
        - `scaler.pkl`
3. **Testing the Bidding Engine:**

   Run the test script:
   python test_bid.py

    - Verify the output bid price.
4. **Packaging:**
    - Include all Python files, model files, and this documentation in a ZIP file for submission.

10. Future Work and Improvements

- **Model Optimization:**
    - Further hyperparameter tuning and model ensembling.
    - Currently only the data for 06.06.2023 has been used due to time and resource constraints. Further training on the entire dataset will help in considerably increasing accuracy and improving model reliability.
- **Dynamic Budget Pacing:**
    - Develop a more sophisticated mechanism to adjust bids based on remaining budget and real-time performance.
- **Feature Enrichment:**
    - Integrate additional contextual features or real-time user behavior signals.
- **Scalability Testing:**
    - Stress-test the bidding engine under simulated high-load conditions.