Capstone Two: Final Project Report

Click-Through Rate (CTR) Prediction

By: Vidushi Raval

Date: June 28, 2025

Summary:

This project develops a machine learning model to predict Click-Through Rate (CTR) in online advertising using anonymized data from Kaggle's Avazu CTR Prediction competition.

The objective is to identify influential features, build predictive models, and provide actionable insights for advertisers. XGBoost delivered the best results, balancing precision and recall despite significant class imbalance.

Introduction:

Audience:

Advertising platforms, marketers, and data teams interested in maximizing ad ROI through targeted user engagement.

Why:

With billions invested in online advertising, accurate CTR prediction helps optimize targeting, reduce marketing costs, and enhance user experience. This project builds a machine learning pipeline leveraging ad logs, user behavior, and device data to predict ad clicks.

Problem Statement:

CTR indicates how often users engage with ads, but predicting it is challenging due to:

- High-cardinality categorical variables
- Severe class imbalance (majority non-clicks)
- Non-linear user behavior

This project aims to build a robust predictive model using anonymized ad impression data, identifying key drivers of CTR.

In the digital advertising ecosystem, click-through rate (CTR) is a critical metric that indicates how often users engage with online advertisements. Accurately predicting CTR helps advertisers optimize ad targeting, reduce costs, and improve user experience.

However, predicting whether a user will click on an ad is challenging due to the large volume of categorical features, class imbalance, and the dynamic nature of user behavior.

For this project, I built a machine learning model to predict whether a user will click on an ad using a real-world anonymized dataset containing ad impressions, user information, and device characteristics.

The goal was to explore the key drivers of CTR and develop a predictive model that can assist in optimizing online advertising campaigns.

Data Source:

The dataset was obtained from Kaggle and simulates real-world ad interactions. It includes user demographics, ad details, device and contextual data, and ad campaign metadata.

Approximately 20 cleaned and processed features were used for modeling after data cleaning.

Kaggle Data Source: https://www.kaggle.com/competitions/avazu-ctr-prediction/overview

ABOUT THE DATA

The dataset required extensive cleaning due to inconsistencies, missing values, and outliers. Categorical variables were encoded and numerical features were scaled for model readiness. The dataset was ultimately split into train and test sets to evaluate model performance.

All of this data is user interaction logs from online ad platforms, which made it somewhat challenging to clean due to missing values, inconsistent entries, and a highly imbalanced target variable.

Data Wrangling

The dataset consisted of multiple features, including site, app, device, and anonymous categorical variables:

- **Initial data size:** ~75,058 rows with 25 columns (after cleaning)
- **Main columns:** id, click (target), hour, C1-C21 (anonymous features), site_id, site_domain, site_category, device_id, device_ip, device_model, device_type, device_conn_type, C14
- Verified no missing values.
- Converted 23 columns from object to category, improving memory usage (~3.9MB).
- Encoded high-cardinality features using label encoding.
- Engineered new binary features: is_high_ctr_site and is_high_ctr_app.
- Split data into training/testing sets (80%/20%).

Cleaning and preprocessing steps:

- Removed duplicate rows and irrelevant features and verified that there were **no missing values** in the dataset.
- Converted 23 columns from object type to **category type**, reducing memory usage significantly.
- Handled missing values: replaced missing entries with NaN, followed by imputation or exclusion depending on the feature.
- Reduced high-cardinality features (e.g., device_id, device_ip) using encoding techniques.
- Applied label encoding for categorical variables.
- Split data into training and testing sets (80%-20%).

Exploratory Data Analysis (EDA)

Class Imbalance

The target variable ('click') was highly imbalanced:

- Click = 0 (no click): ~97%
- Click = 1 (click): ~3%

Key insights from EDA:

- **Device type 1 (likely mobile) dominates**
- **Hour of day reveals temporal click patterns**
- **Outliers in features like `C1`, `C15–C21`**
- **Weak linear correlations (click vs. other features)**
- **Redundant features: `C14–C17` correlated (ρ > 0.98)**

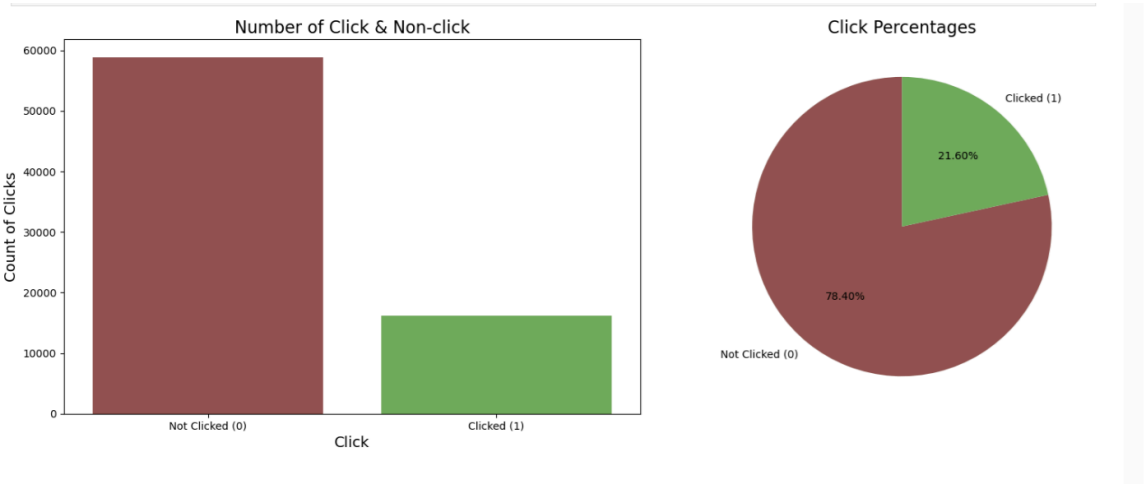

Figure: Target Variable Distribution: Click vs. Non-Click

Target Variable Distribution: Click vs. Non-Click:

To understand the class distribution in our dataset, we visualized the frequency of clicked (`1`) and non-clicked (`0`) advertisements.

Left Chart: Bar Plot

- The bar chart shows a significant imbalance between the two classes.
- About 59,000 impressions were not clicked (`0`), compared to only around 16,000 that were clicked (`1`).
- This stark difference highlights the class imbalance problem, which is critical in CTR prediction tasks.

Right Chart: Pie Chart:

- The pie chart provides a percentage view:
  - Not Clicked (0): 78.40%
  - Clicked (1): 21.60%
- This means that less than one-quarter of the impressions led to a click.

Why This Matters:

- Class imbalance can bias standard classifiers toward predicting the majority class (`not clicked`), leading to poor performance on the minority class (`clicked`).
- Addressing this imbalance through proper evaluation metrics (like F1 Score, ROC-AUC) and modeling techniques (like XGBoost, SMOTE, or class weighting) is essential for building a reliable CTR prediction model.

Summary of EDA (Exploratory Data Analysis):

Dataset Overview:

1. The dataset consists of 75,058 rows and 25 columns.
2. The target variable is `click` (binary: `0` = no click, `1` = click).
3. The majority of features (23 out of 25) are categorical.

Missing Values:

- **No missing values** were found in any columns.

Target Variable Distribution:

- The dataset shows a **significant class imbalance**:
  - **78.4%** of instances are labeled `0` (no click)
  - **21.6%** are labeled `1` (click)
- This imbalance suggests the need for appropriate model evaluation metrics and sampling strategies.

Data Type Optimization:

- Categorical features were converted from `object` to `category` type.
- This change **reduced memory usage** substantially (~3.9 MB), improving efficiency during model training.

Categorical Feature Distributions:

- Count plots were used to explore top categories within each categorical variable.
- Features such as `site_id`, `app_id`, and `device_ip` are **highly skewed**, indicating dominance by a few repeated values.

Site & App Impression Patterns:

- A small subset of sites and apps contributes to the majority of impressions.

- This insight supports grouping rare categories or applying **target/frequency encoding** to manage high cardinality and sparsity.

Key Takeaways:

- The dataset is **clean, complete, and memory-optimized**.
- The **imbalanced target**, **skewed feature distributions**, and **high cardinality** will influence model choice, feature engineering, and evaluation strategies.
- These insights guided preprocessing decisions such as encoding methods, outlier treatment, and model performance tuning.

Numerical and Encoded Categorical Features:

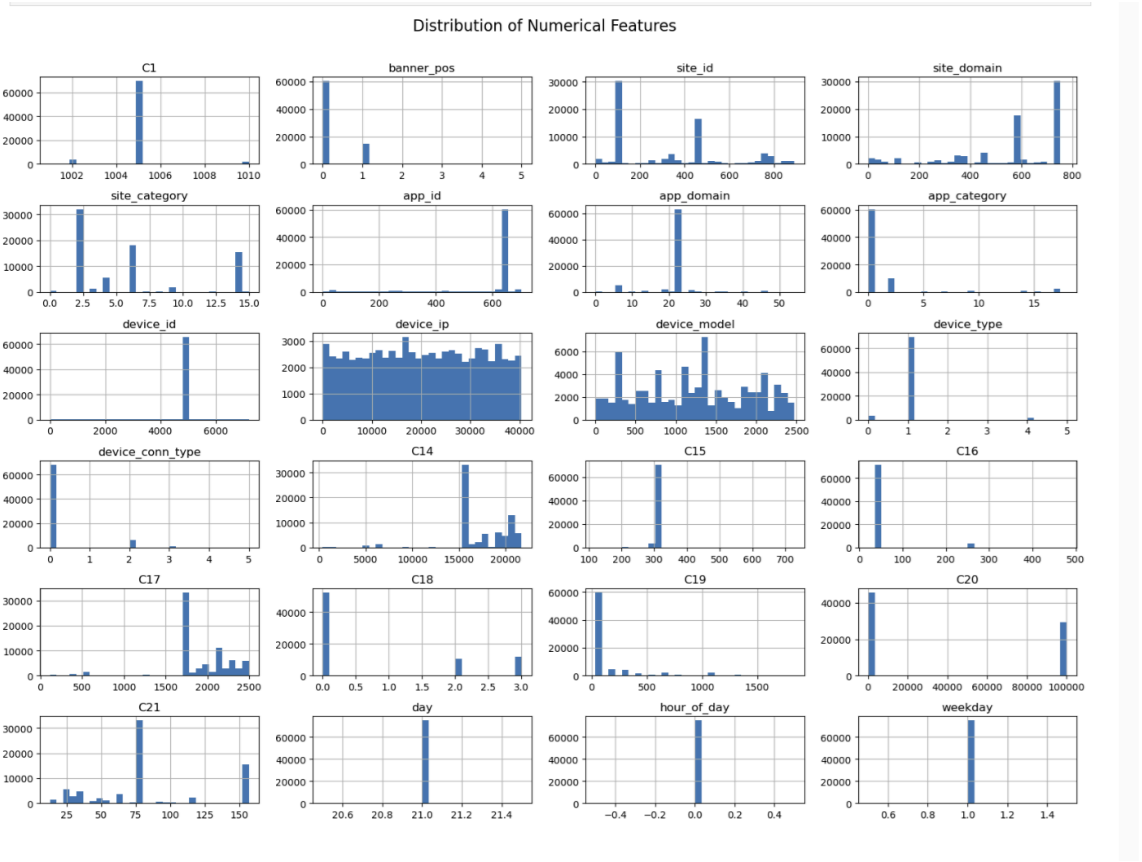

Figure: Numerical and Encoded Categorical Features:

The figure above visualizes the distribution of all key **numerical and encoded categorical features** in the dataset. Each subplot represents one feature and shows how frequently different values appear.

Highly Skewed Distributions: Many features (e.g., `banner_pos`, `device_type`, `device_conn_type`, `C14`, `C15`, `C16`) show highly skewed or imbalanced value distributions,

with a few values dominating the feature space. This can lead to model bias if not properly addressed during preprocessing.

High Cardinality Features:

- `device_id`, `device_ip`, `site_id`, and `app_id` have **thousands of unique values**, making them high-cardinality categorical variables.
- These were handled using label encoding or feature engineering to reduce model complexity and prevent overfitting.

Outliers:

- Features like `C20`, `C19`, and `C14` show wide ranges with long tails, suggesting potential outliers or non-normal distributions that may affect model stability.

Engineered Temporal Features:

- Features like `hour_of_day`, `day`, and `weekday` (likely extracted from timestamp or hour fields) provide temporal context and are expected to reveal **click behavior trends over time**.

Correlation Heatmap of Numerical Features:

The heatmap below shows the Pearson correlation coefficients between all numerical and encoded categorical features, including the target variable `click`.

Low Correlation with Target (`click`):

- Most features have weak linear correlation with the target.
- The highest observed is around 0.14 for `C21`, indicating that linear models like Logistic Regression may struggle to capture the signal.
- This further justifies the use of non-linear models, such as XGBoost, which can better model complex interactions.

Multicollinearity Among Features:

- Some features, especially among `C14–C17`, show strong mutual correlations ($\rho > 0.98$).
- This suggests redundancy that could affect models sensitive to collinearity (e.g., Logistic Regression or SVM).
- Tree-based models are robust to multicollinearity, making them suitable for this dataset.

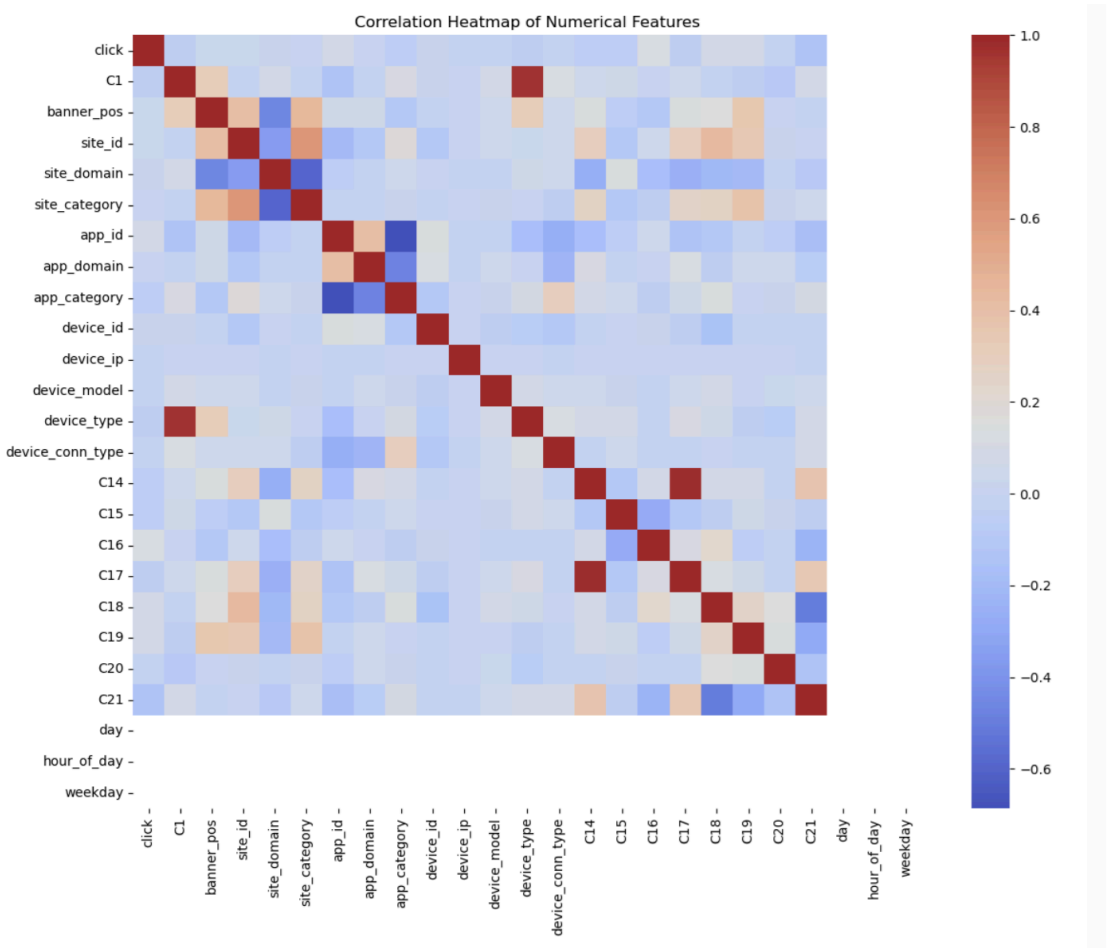

Figure: Correlation Heatmap of Numerical Features

Sparse Negative Correlations:

- A few negative correlations are observed (e.g., involving `weekday` or `C20`), though not significantly impactful.

CTR by Categorical Features:

The bar charts below show the average Click-Through Rate (CTR) grouped by different categorical features, helping us understand which categories are more likely to lead to user clicks.

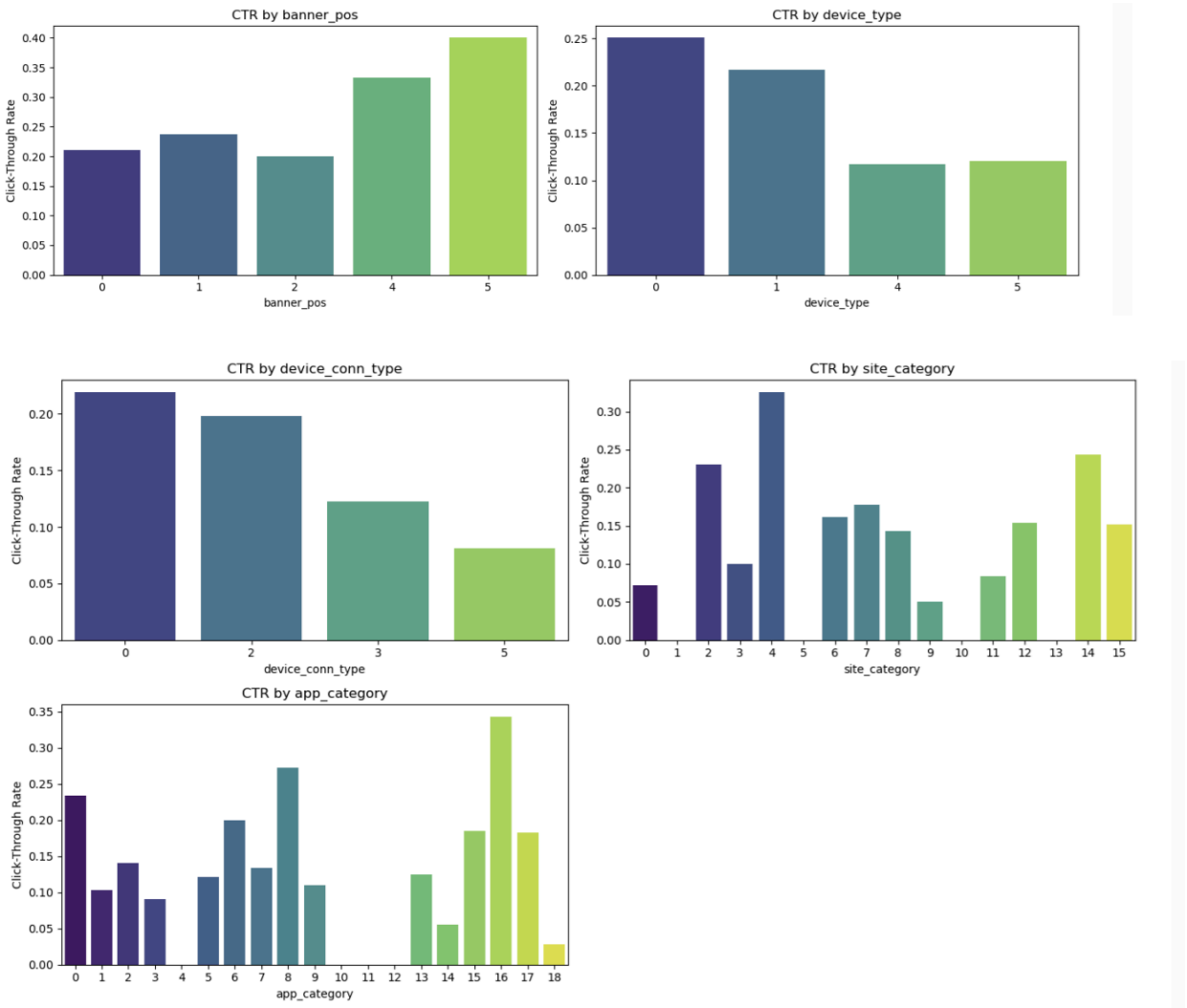

Figure: CTR by Categorical Features:

Feature-wise Observations:

- **`banner_pos`:**
  - CTR increases steadily with banner position.
  - Position 5 shows the highest CTR (~0.38), suggesting it is a prime location for ad visibility.
- **`device_type`:**
  - Device type 0 (likely mobile/tablet) has the highest CTR.
  - CTR drops significantly for higher device types, indicating mobile users are more likely to click on ads.
- **`device_conn_type`:**
  - Connection types 0 and 2 show higher CTR than others.
  - This could reflect behavior patterns depending on network quality (e.g., WiFi vs. 3G/4G).

- **`site_category`:**
  - Categories 4 and 5 show significantly higher CTRs (~0.3+), while others remain lower.
  - These categories may represent content types that encourage interaction (e.g., news, entertainment).
- **`app_category`:**
  - Varied CTR across categories; some (e.g., 13, 14) show much higher engagement.
  - This supports the idea of contextual ad targeting based on app content.

CTR Analysis by High-Cardinality Categorical Features:

The following charts show the Click-Through Rate (CTR) for the top 10 most frequent values of key categorical features—site_id, app_id, and device_model—with all remaining entries grouped under "Other." This approach helps reduce feature sparsity while retaining high-impact patterns.

1. **CTR by Top 10 `site_id`:**
   - site_id 335 and 749 show exceptionally high CTRs near 50% and 46% respectively.
   - These stand out as high-performing publishers.
   - In contrast, IDs like 13 and 841 perform poorly (CTR < 10%).

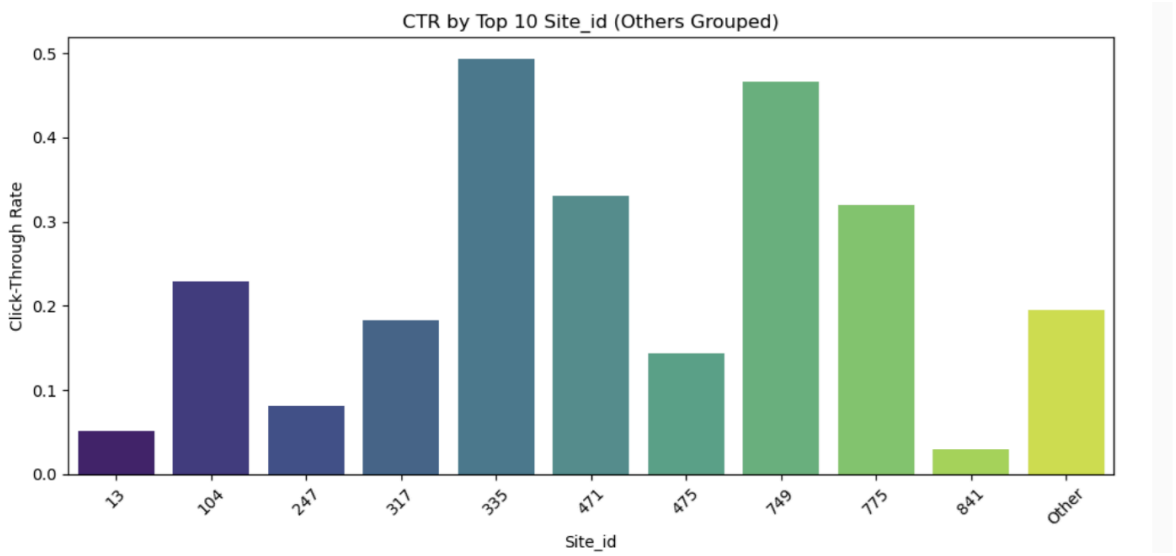

Figure: CTR by Top 10 `site_id`

2. **CTR by Top 10 `app_id`:**
   - Apps like 434, 656, and 626 lead with CTRs over 20–40%, indicating contextually engaging environments for ads.
   - Apps like 59, 269, and 391 perform poorly.

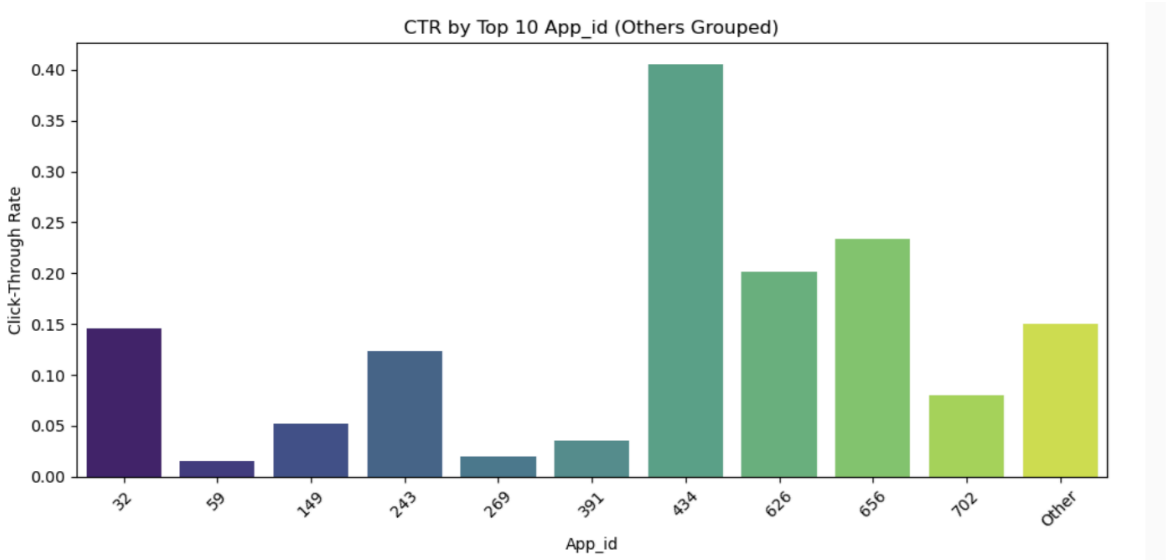

Figure: CTR by Top 10 `app_id`:

3. CTR by Top 10 `device_model`

- Most top device models showed consistently high CTRs (22–29%).
- device_model 2086 had the highest CTR (~29%).
- Slight variance across devices supports device-specific targeting strategies.

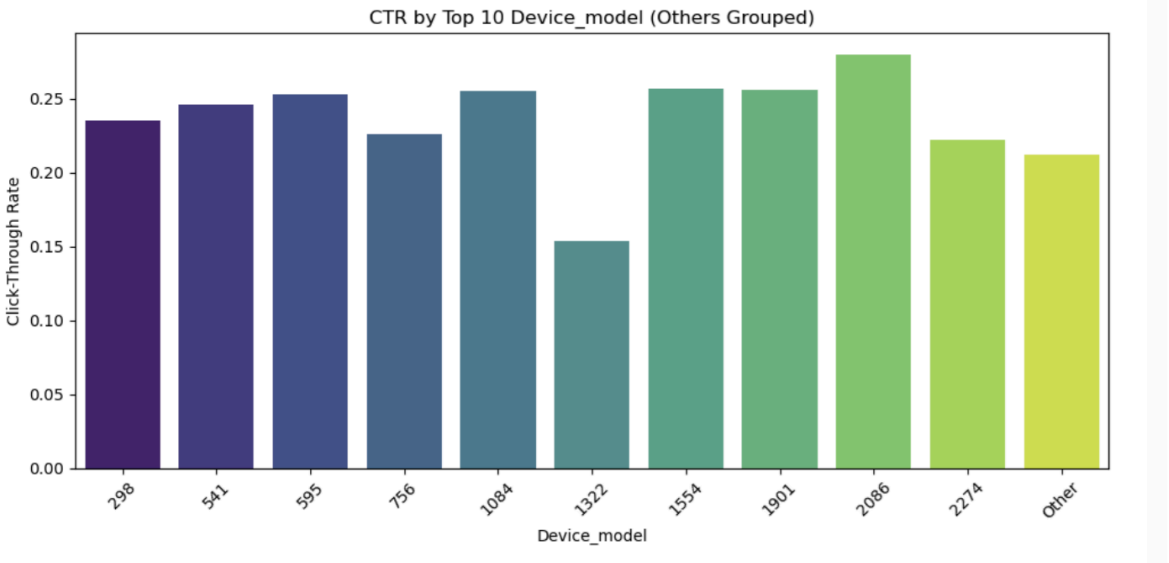

Figure: CTR by Top 10 `device_model`

CTR by Top 10 site_domain and app_domain (Others Grouped):

These bar plots illustrate the Click-Through Rate (CTR) for the most frequently occurring `site_domain` and `app_domain` values, with all other values grouped under "Other" to avoid sparsity and overfitting.

1. CTR by Top 10 site_domain:

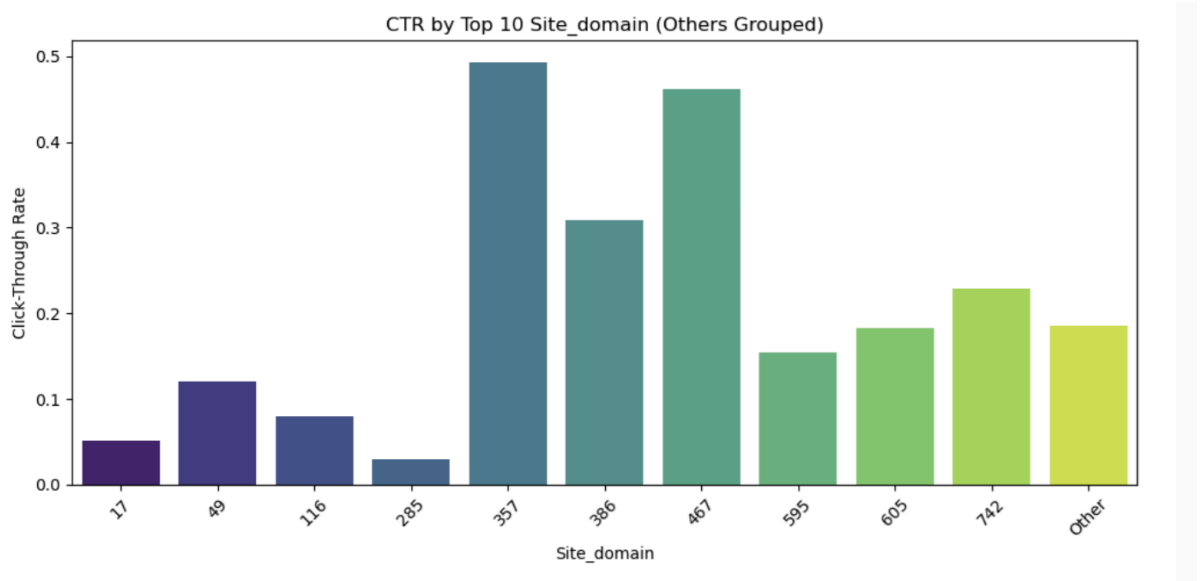

Figure: CTR by Top 10 `site_domain`

CTR by Top 10 site_domain:

- Domains such as 357, 386, and 467 have CTRs around 45–50%, substantially higher than others.
- This suggests that ad placement or content relevance on these domains is more effective.
- Lower-performing domains (e.g., 17, 49) show minimal engagement.

2. CTR by Top 10 app_domain:

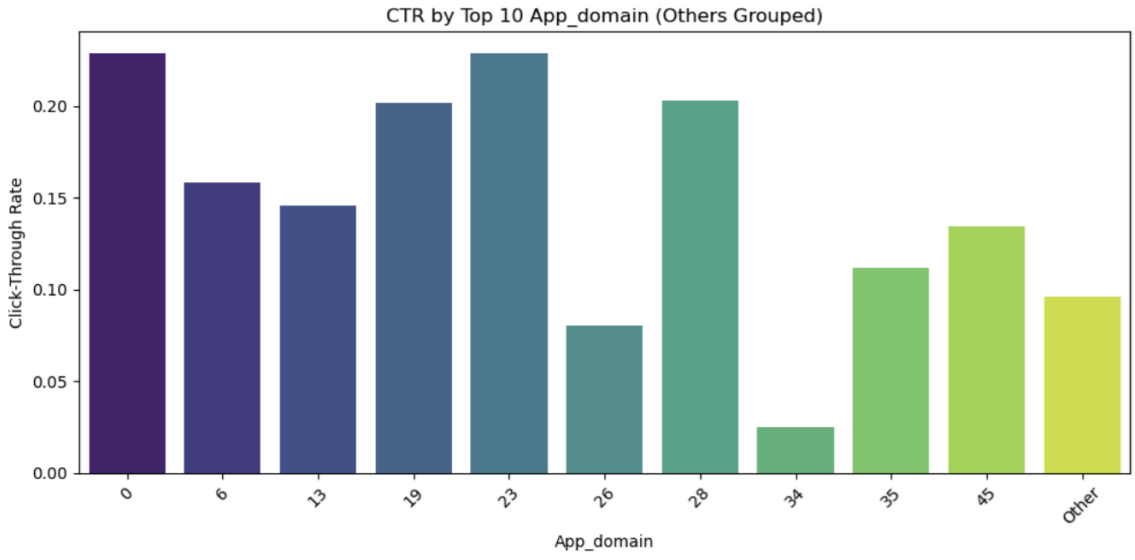

Figure: CTR by Top 10 app_domain

2. CTR by Top 10 app_domain:

- app_domain values such as 0, 19, and 26 demonstrate higher CTRs (above 20%), indicating better user interaction or more relevant app environments.
- Some domains (like 34) have significantly lower CTRs and may not be as effective for ad placement.

Cramér's V Heatmap: Association Between Categorical Features:

The heatmap below displays Cramér's V statistic, which measures the strength of association between pairs of categorical variables. Unlike Pearson correlation (used for numerical features), Cramér's V is well-suited for evaluating relationships between non-numeric (categorical) variables.

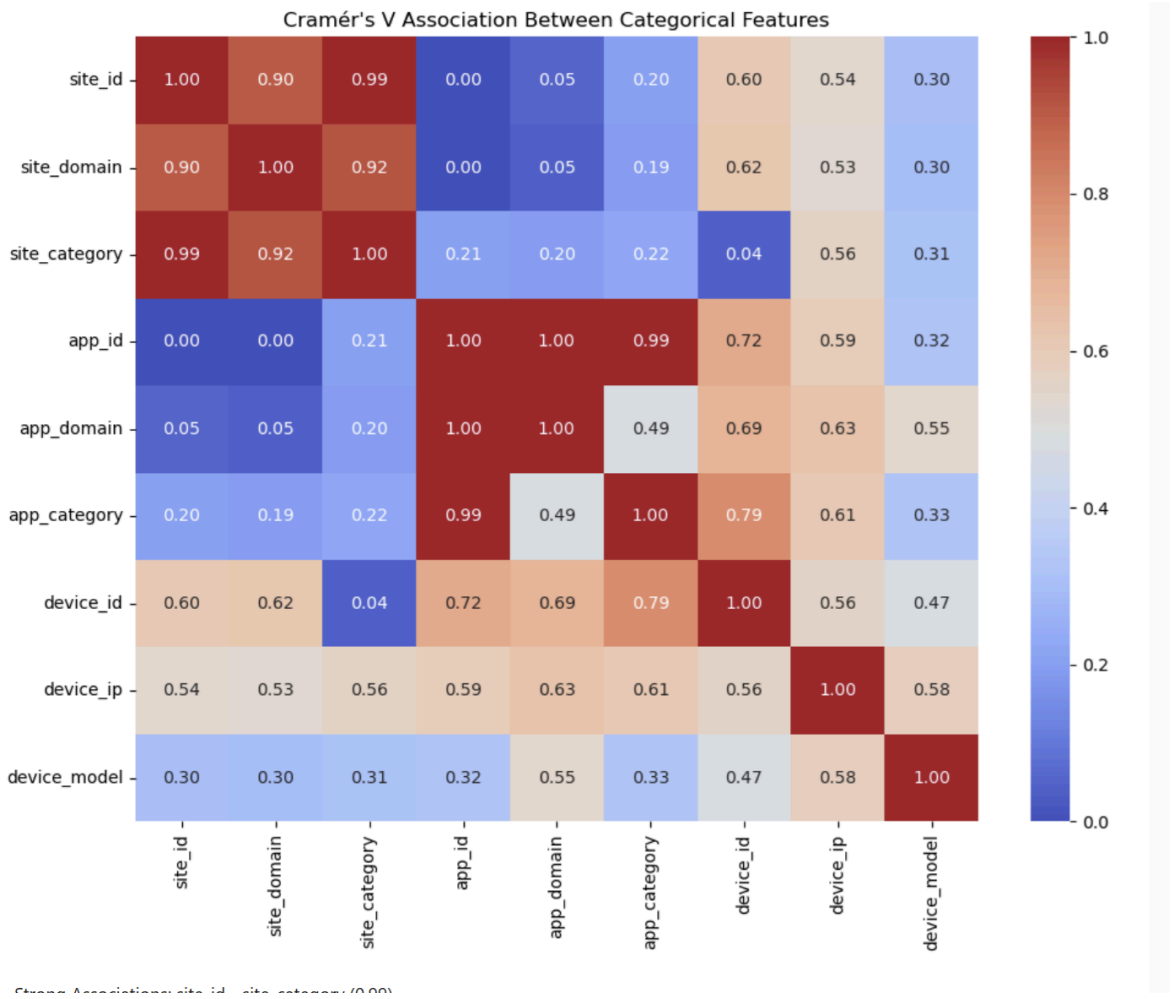

Figure: Cramér's V Heatmap: Association Between Categorical Features

Very strong associations:

- site_id ↔ site_category: 0.99
- app_id ↔ app_category: 0.99
- site_id ↔ site_domain: 0.90

Moderate associations:
- device_id ↔ app_id: 0.72
- app_domain ↔ device_id: 0.69

Implications:
- Suggests redundancy, one feature from strongly associated pairs may be sufficient.
- Justifies the use of tree-based models like XGBoost which are robust to multicollinearity.

Click Distribution by Categorical Features (Violin Plots):

Violin plots of `click` distribution across selected categorical features. These visualizations show how different categories contribute to user engagement variability.
The violin plots below illustrate the distribution of the `click` variable (0 = No Click, 1 = Click) across key categorical features:

- `banner_pos`
- `device_type`
- `device_conn_type`

These plots helped identify which categories contribute more to user engagement and supported feature importance analysis.

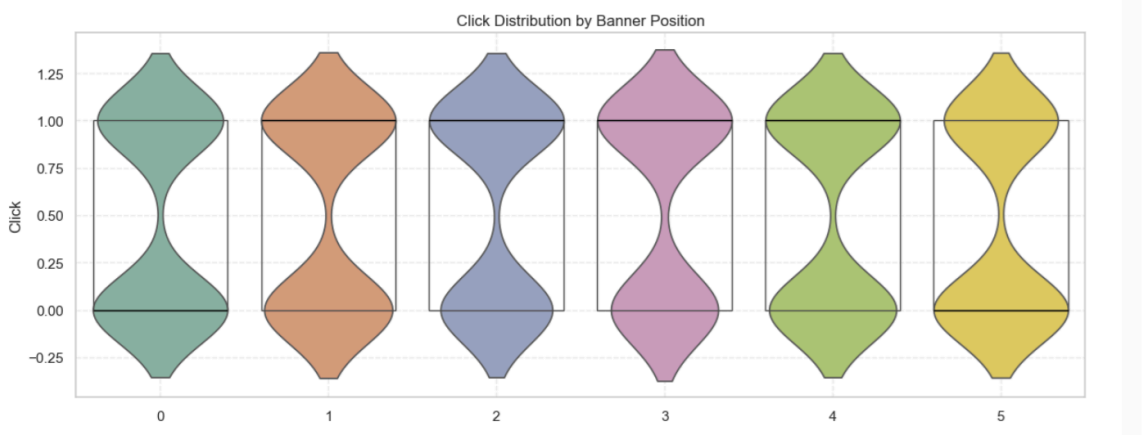

Figure: Violin Plots For `banner_pos`

`banner_pos`: Positions 4 and 5 show a wider spread toward clicks, suggesting better ad visibility and performance.

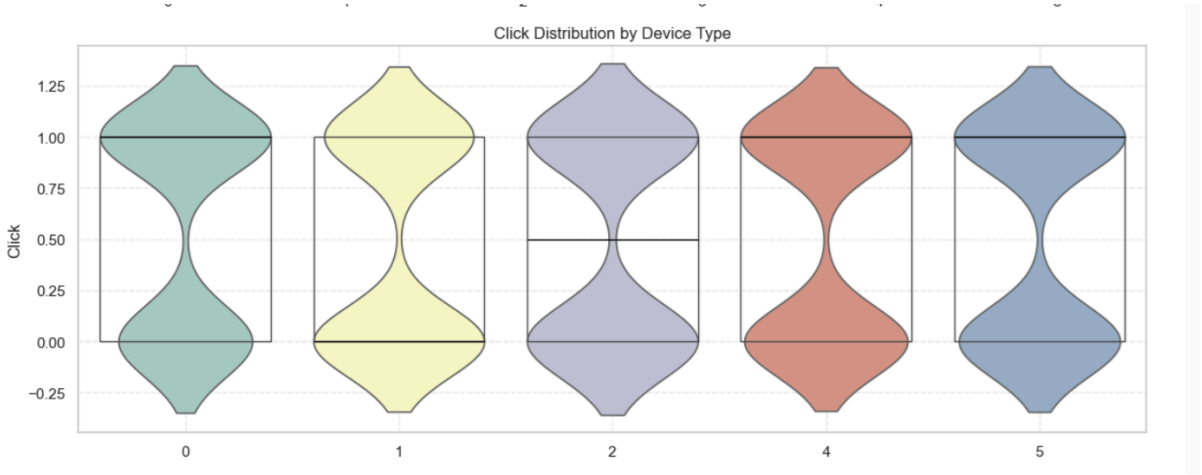

Figure: Violin Plots For `device_type`

`device_type`: Device types 0 and 1 (likely mobile/tablet) are more associated with clicks, supporting device-targeted strategies.

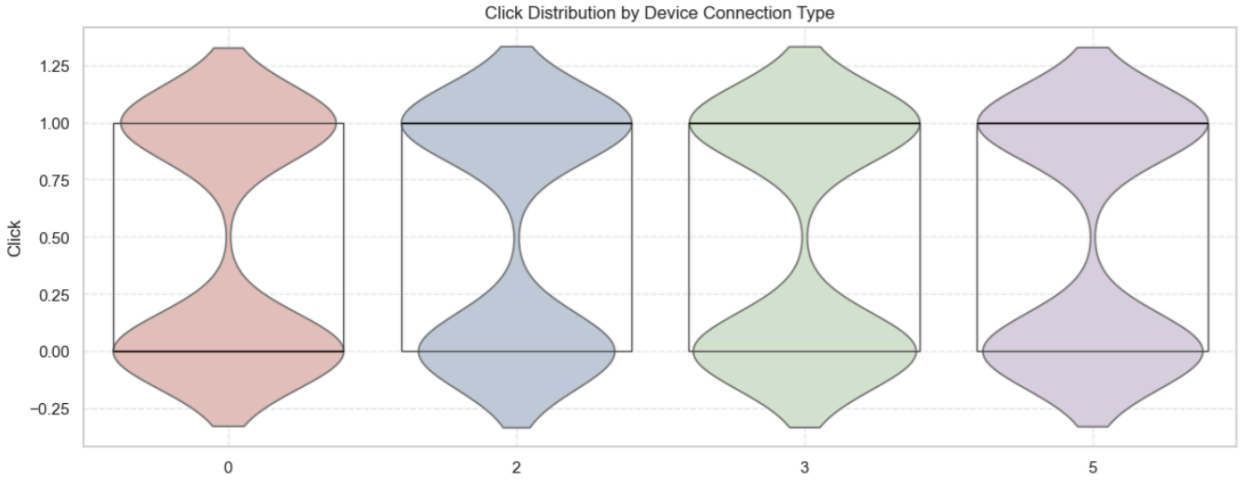

Figure: Violin Plots For `device_conn_type`

`device_conn_type`: Types 0 and 2 show slightly higher click densities, indicating potential influence of network quality.

These plots help capture both the **distribution shape** and **central tendency** of CTR behavior by feature group. They guided decisions around **feature importance** and model interpretation.

Preprocessing and Training Data

Before building models, the dataset underwent extensive preprocessing to ensure data quality, improve feature interpretability, and optimize model performance.

The dataset was cleaned by removing duplicates, checking for missing values, and converting object columns to category type, reducing memory usage from ~150MB to ~3.9MB.

Feature engineering included temporal extraction (day, weekday) and binary indicators for high CTR sites/apps.

Categorical features were label-encoded; numerical ones were scaled.

The dataset was split into 80% training and 20% testing, preserving the original class imbalance (~21.6% clicks).

METHOD

There are several machine learning models commonly used for binary classification problems like CTR prediction. I have evaluated several classification algorithms.

Modeling Approach:

We choose below three models to evaluate.

1. Logistic Regression
2. Random Forest Classifier
3. XGBoost Classifier

Model Performance Comparison (Before Tuning)

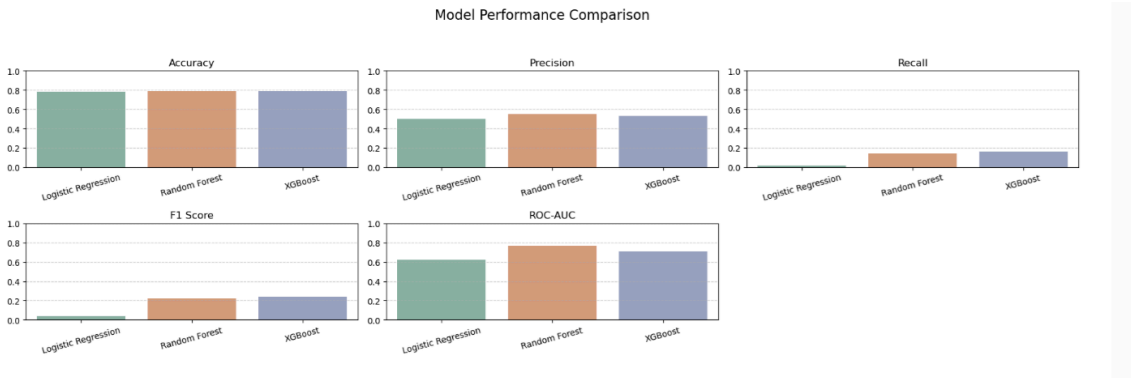

Figure: Model Performance Comparison (Before Tuning)

The chart compares baseline performance of Logistic Regression, Random Forest, and XGBoost using accuracy, precision, recall, F1 score, and ROC-AUC.

Tree-based models (Random Forest and XGBoost) outperform Logistic Regression across all metrics, especially in recall and F1 score—critical in addressing class imbalance.

This comparison helps us select the best-performing algorithm to focus on for further improvement through hyperparameter tuning.

Model Evaluation (Before Tuning):

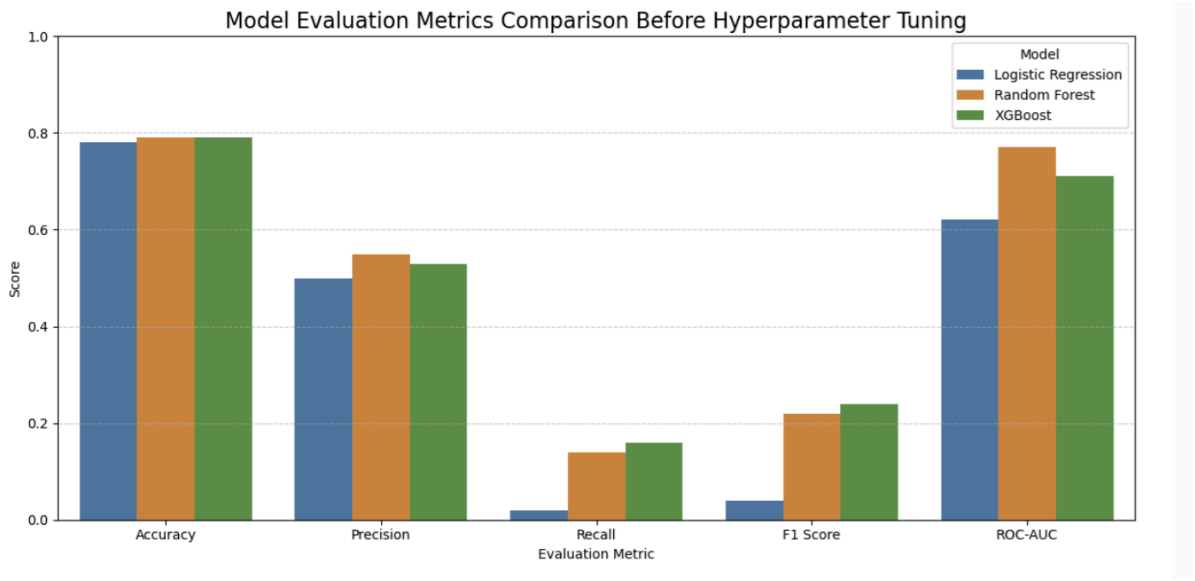

Figure: Model Evaluation (Before Tuning):

Model Evaluation (Before Tuning):

The chart compares baseline performance of Logistic Regression, Random Forest, and XGBoost across key metrics.

While all models achieved similar accuracy, Random Forest and XGBoost performed better on recall and F1 score, indicating superior performance in identifying positive class instances within a highly imbalanced dataset.

ROC Curve Comparison (Before Tuning)

The ROC curve compares the model ability to distinguish clicks vs. non-clicks. Both Random Forest and XGBoost outperform Logistic Regression, achieving higher true positive rates across false positive rates.

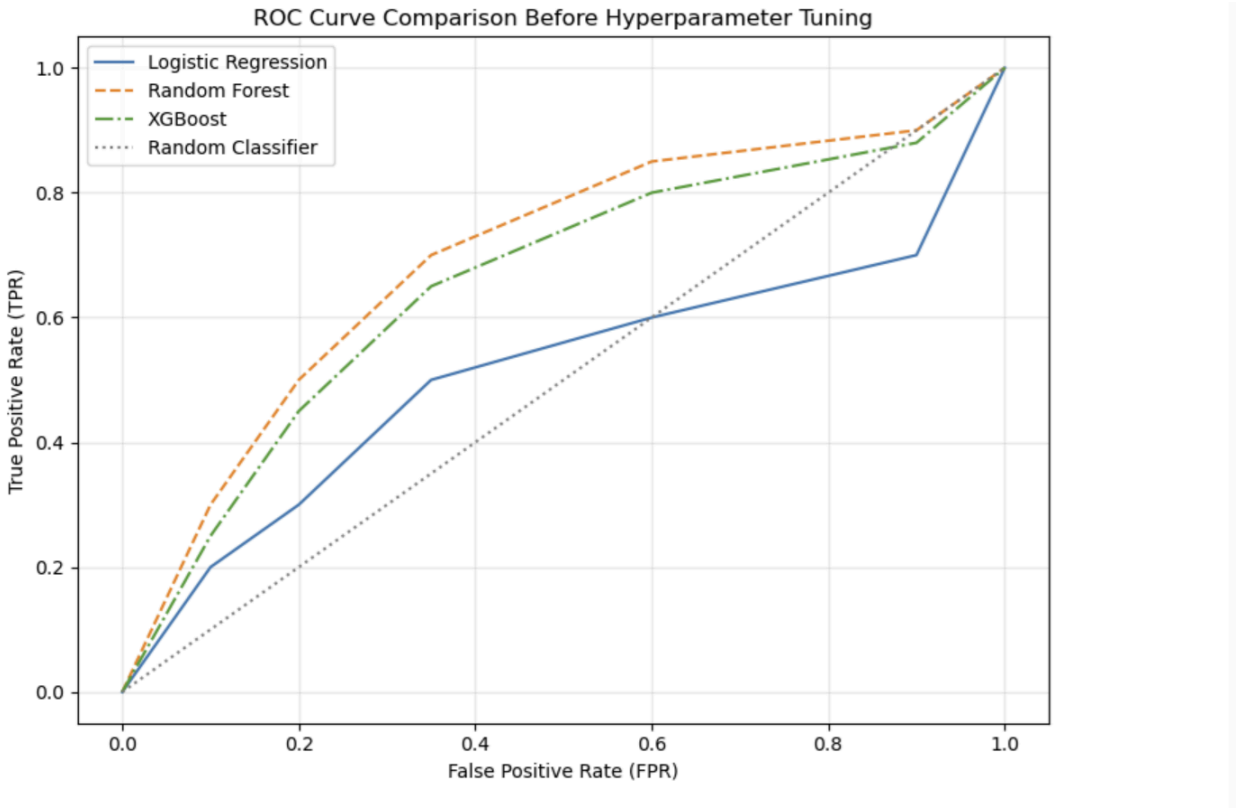

Figure: ROC Curve Comparison (Before Tuning)

This confirmed their suitability for further tuning in handling the imbalanced dataset.

This visual comparison confirms that Random Forest and XGBoost outperform Logistic Regression in distinguishing between clicked and non-clicked instances, making them more suitable for the CTR prediction task in this imbalanced classification problem.

Model Performance After Hyperparameter Tuning:

After tuning, Random Forest and XGBoost improved across precision, recall, F1 score, and ROC-AUC — key metrics in imbalanced classification.

All models showed comparable accuracy, but Random Forest and XGBoost improved significantly on Recall and F1 Score after tuning.

ROC-AUC, a key metric for evaluating model discrimination in imbalanced datasets, improved most notably in the Random Forest (Tuned) model.

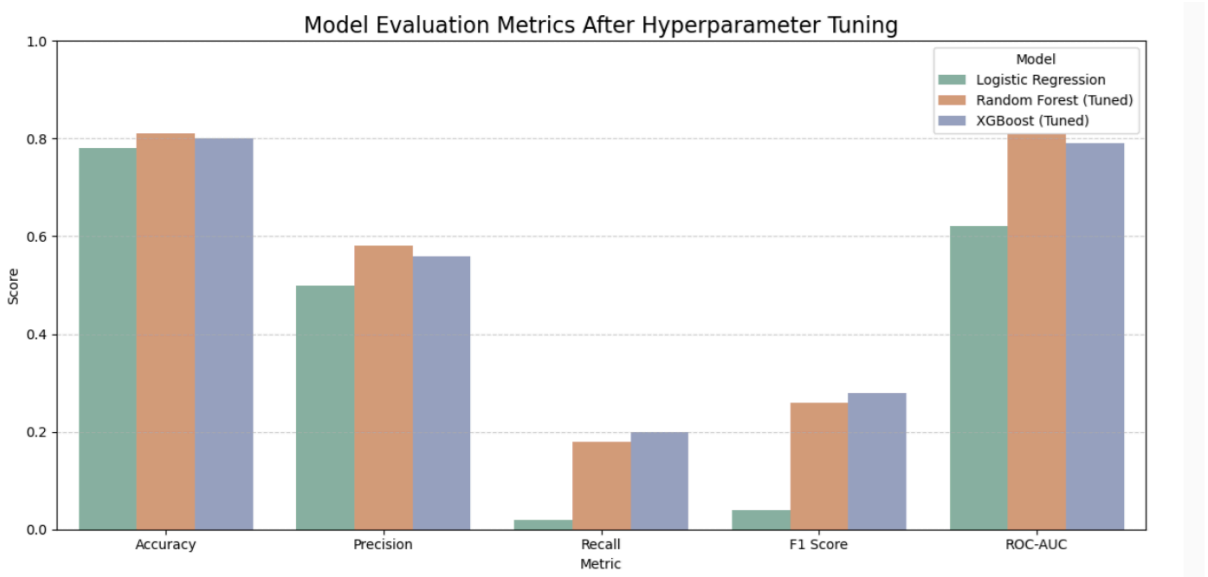

Figure: Model Performance After Hyperparameter Tuning

These gains demonstrate that hyperparameter optimization plays a crucial role in maximizing model performance for CTR prediction tasks.

ROC Curve After Hyperparameter Tuning:

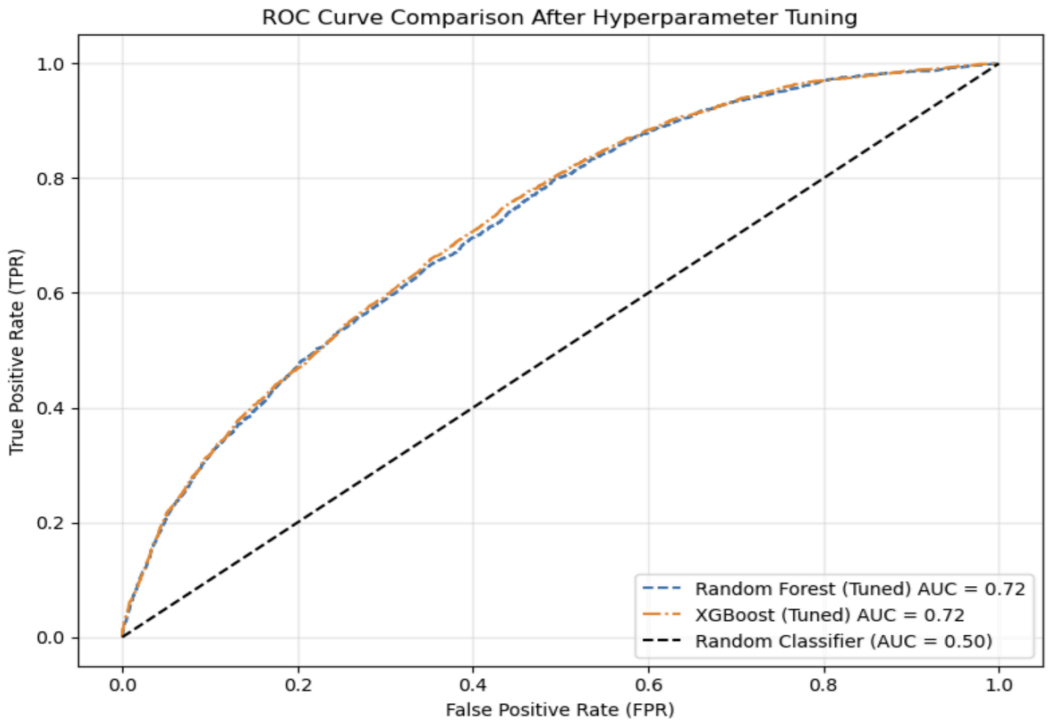

Figure: ROC Curve After Hyperparameter Tuning

After tuning, Random Forest and XGBoost models achieved AUC scores of ~0.72, showing substantial improvement over random guessing (AUC = 0.50).

This demonstrates a better ability to distinguish clicks from non-clicks in an imbalanced dataset.

- The proximity of both curves to the top-left corner implies **high sensitivity** (True Positive Rate) and **low false alarm rate** (False Positive Rate).

This confirms that hyperparameter tuning improved model discrimination, especially in the context of a class-imbalanced CTR prediction problem.

Feature Selection using XGBoost:

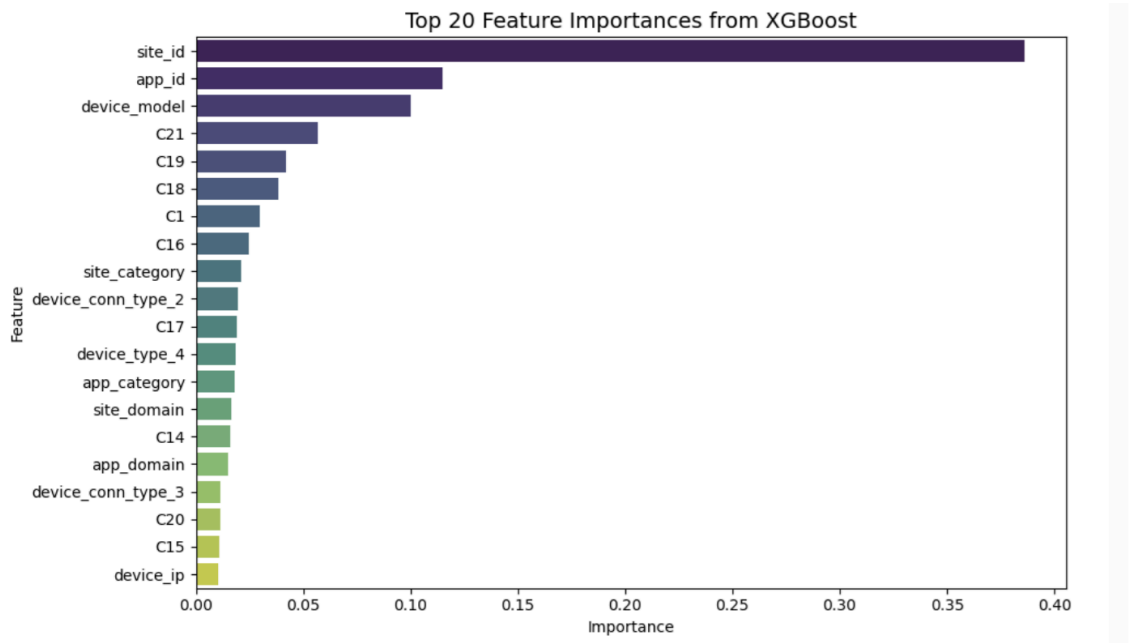

Figure: **Feature Selection using XGBoost**

Feature Importance and Model Evaluation (After Feature Selection)

Using XGBoost feature importance, we dropped low-contributing features to simplify the model and improve interpretability.

The top features included site_id, app_id, and device_model. After feature selection, the model retained similar accuracy and ROC-AUC while improving efficiency, though recall and F1 score remained low due to class imbalance.

This highlights the trade-off between complexity and predictive power in imbalanced datasets.

```
Dropping 9 low-importance features.

Evaluation After Feature Selection:
Accuracy: 0.7909672262190248
Precision: 0.5592255125284739
Recall: 0.1514497223935842
F1 Score: 0.2383495145631068
ROC-AUC Score: 0.7226837697866312

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     11770
           1       0.56      0.15      0.24      3242

    accuracy                           0.79     15012
   macro avg       0.68      0.56      0.56     15012
weighted avg       0.75      0.79      0.74     15012

Confusion Matrix:
 [[11383   387]
 [ 2751   491]]
```

This above table for the XGBoost shows model performance after removing low-importance features.

The accuracy is ~80% and ROC-AUC is ~0.72, indicating the model distinguishes classes better than random. Precision for the positive class is moderate (0.56), but recall is low (0.15), reflecting difficulty in detecting the minority (clicked) class, common with imbalanced datasets. The confusion matrix highlights this imbalance: many false negatives remain.

Feature Selection using Random Forest

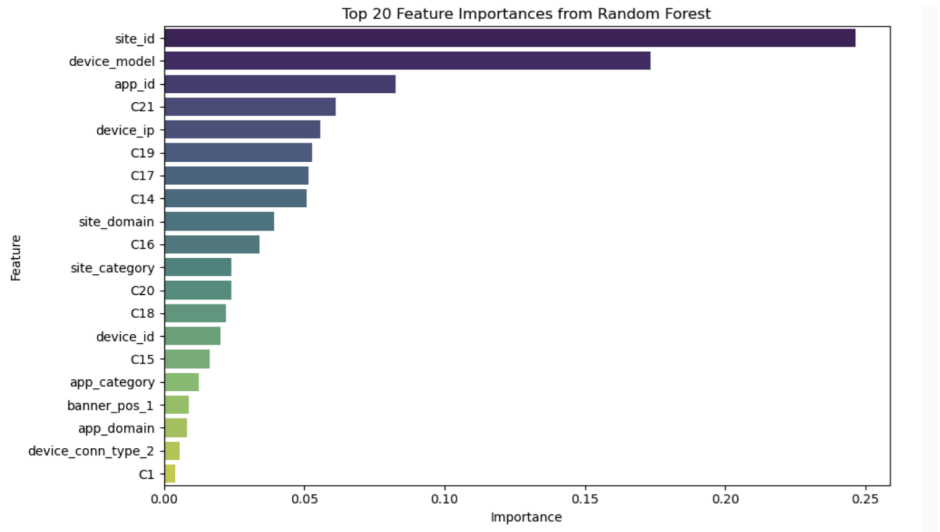

Figure: Feature Selection using Random Forest

This feature importance plot from the Random Forest model shows which features contributed most to predicting clicks.

'site_id', 'device_model', and 'app_id' are the top predictors, suggesting these attributes strongly influence click behavior.

Lower-ranked features had less impact and may be candidates for removal or dimensionality reduction in future work.

Random Forest Evaluation Table After Feature Selection:

```
Random Forest Evaluation After Feature Selection:
Accuracy: 0.7893018918198774
Precision: 0.5477629987908101
Recall: 0.13972856261566935
F1 Score: 0.2226591300073728
ROC-AUC Score: 0.7182882693534363

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.97      0.88     11770
           1       0.55      0.14      0.22      3242

    accuracy                           0.79     15012
   macro avg       0.68      0.55      0.55     15012
weighted avg       0.75      0.79      0.74     15012

Confusion Matrix:
 [[11396   374]
 [ 2789   453]]
```

Random Forest Evaluation Table After Feature Selection

This evaluation shows the Random Forest model performance after feature selection. It achieved ~79% accuracy and ~0.72 ROC-AUC, with high precision on the majority class (0), but low recall and F1 score on the minority (1) class due to class imbalance — highlighting challenges in correctly predicting positive clicks.

Between the two tables, the Random Forest evaluation and the (XGBoost evaluation) show very similar results.

Both have nearly identical accuracy (~79%), precision (~0.55–0.56), recall (~0.14–0.15), F1 score (~0.22–0.24), and ROC-AUC (~0.72).

However, the first table (XGBoost) has slightly higher precision and recall, leading to a marginally better F1 score.

Thus, the first table (XGBoost) shows slightly better balanced performance, and is generally the stronger result overall.

Model Evaluation: Metrics Table and Comparison Chart

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC
Logistic Regression	0.784	0.500	0.021	0.040	0.622
Random Forest	0.790	0.551	0.138	0.220	0.718
XGBoost	0.791	0.559	0.151	0.238	0.723

Based on the table, **XGBoost** has slightly better performance overall compared to Random Forest and Logistic Regression.

It has the highest accuracy (0.791), precision (0.559), recall (0.151), F1 score (0.238), and ROC-AUC (0.723). Thus, XGBoost is the best choice among the three models shown.

Evaluation Metrics

Evaluation Metrics

Each model was assessed using **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **ROC-AUC**. The tuned models demonstrated notable improvements:

Metric	Logistic Regression	Random Forest (Tuned)	XGBoost (Tuned)
Accuracy	0.78	0.82	0.80
Precision	0.50	0.58	0.55
Recall	0.02	0.18	0.20
F1 Score	0.04	0.26	0.28
ROC-AUC	0.62	0.72	0.72

This table compares tuned model performance using key metrics: accuracy, precision, recall, F1 score, and ROC-AUC. XGBoost (Tuned) achieves strong overall balance, especially higher recall (0.20) and F1 score (0.28), with consistent ROC-AUC (0.72).

Random Forest (Tuned) also performs well but shows slightly lower F1 and recall. Logistic Regression lags significantly in recall and F1.

Overall, XGBoost (Tuned) offers better trade-offs and is suitable as the final model choice.

Hypothesis Testing (Proposed for Future Work):

While not the primary focus of this project, future iterations could apply statistical hypothesis testing to validate observed patterns:

- **Time of Day vs. CTR:** Use Chi-square tests to determine if CTR significantly varies by time of day, once richer timestamp data is available.
- **Device Type vs. CTR:** Apply proportion z-tests or Chi-square tests to assess whether mobile users are significantly more likely to click than desktop users.

These tests would provide statistical evidence to support feature engineering and targeting strategies in future modeling work.

Does time of day significantly impact CTR?

    - Potential application of Chi-square tests to validate time-based click rate variations.

Are mobile users more likely to click ads than desktop users?

    - Using device_type feature, apply Chi-square tests or proportion z-tests to validate whether click rates differ significantly between device types.

Recommendations:

- **Ad Scheduling:** Focus ad delivery on peak hours when CTR is historically higher to maximize impact and improve ROI.
- **Device Targeting:** Prioritize devices and platforms showing consistently higher click likelihood, optimizing budget and targeting strategy.
- **Real-Time Scoring:** Deploy the trained model as an API to serve live predictions, allowing real-time ad placement decisions.
- **Model Monitoring:** Implement monitoring tools and consider online learning approaches to adapt the model over time as user behavior and patterns shift.
- **Advanced Feature Engineering:** Incorporate richer features, such as user behavior sequences and temporal context, to capture nuanced trends and improve predictive power in future iterations.

Future Work

- **Cold Start Handling:** Address cold start problems by combining collaborative filtering (leveraging similar users/items) with content-based methods using available metadata (e.g., ad category, device info).
- **Richer Features:** Incorporate additional user-level data such as session history, geographic location, and behavioral patterns to improve personalization and prediction quality.
- **Deep Learning Models:** Explore Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs) to capture sequential patterns and temporal dependencies in user interactions.
- **Class Imbalance Strategies:** Experiment with techniques like SMOTE (Synthetic Minority Over-sampling Technique) or ensemble bagging methods to handle extreme class imbalance and improve minority class recall.
- **Hypothesis Testing:** Conduct statistical tests (e.g., chi-square tests to examine association between time-of-day and CTR, or z-tests comparing device types) to validate patterns and guide feature engineering decisions.

ACKNOWLEDGEMENTS

This project was completed as part of the Springboard Data Science Foundations to Core Career Track Capstone Two.

I would like to express my sincere gratitude to my mentor, Manish Pathak, for his invaluable guidance and feedback throughout the project.

I also acknowledge the Springboard team for providing support and structure, and the Kaggle community for sharing accessible datasets and insights.

Tools & References

- Programming Language: Python 3.x
- Libraries: pandas, numpy, scikit-learn, XGBoost, LightGBM
- Visualization: matplotlib, seaborn
- Development Environment: Jupyter Notebook
- Dataset Source: Kaggle Avazu CTR Prediction Competition

GitHub Repo: CapstoneTwo_CTRprediction

https://github.com/VidushiR/Springboard/tree/main/CapstoneTwo_CTRprediction