DATA SCIENCE ASSIGNMENT ZEPTO

INTRODUCTION

Ads Click-Through Rate or CTR in short prediction is a crucial task in online advertising, where the goal is to predict the probability that a user will click on a particular advertisement. It plays a vital role in optimizing ad campaigns, as it helps advertisers allocate their resources effectively, target specific audience segments, and maximize the return on investment (ROI). Machine learning techniques are commonly employed in CTR prediction due to their ability to analyze vast amounts of data and identify patterns that can help predict user behavior.

Using machine learning to predict Click Through Rate (CTR) offers both advantages and drawbacks. Here's a detailed look at each:

ADVANTAGES:

1. **Handling Complexity:**
   - Machine learning models can *capture complex patterns* and interactions in data that may be difficult to identify with traditional statistical methods.
2. **Scalability:**
   - Machine learning models can *handle large volumes of data efficiently,* making them suitable for applications with extensive user interactions and features.
3. **Adaptability:**
   - Models can be *updated and retrained as new data* becomes available, allowing them to adapt to changing user behaviors and trends.

DRAWBACKS:

1. **Data Dependency:**
   - Machine learning models require large amounts of *high-quality data for training*. Insufficient or noisy data can lead to poor model performance.
2. **Overfitting:**
   - Models can become *too complex and overfit the training data*, resulting in poor generalization to new, unseen data.
3. **Interpretability:**
   - Many machine learning models, *especially complex ones like deep neural networks, are often considered "black boxes*," making it challenging to interpret how predictions are made.

MY APPROACH TO THE PROBLEM STATEMENT:

1. Exploratory Data Analysis (EDA):

- **Analyze Data:** Perform exploratory analysis to understand the distribution of features, correlations, and patterns in the data.
- **Visualization:** Use visualizations (e.g., histograms, scatter plots) to gain insights into relationships between features and the target variable (CTR).

2. Data Preprocessing:

- **Data Cleaning:** Handle missing values, outliers, and errors in the dataset. This might involve filling missing values, removing or correcting erroneous data, and normalizing or scaling features.
- **Feature Engineering:** Create new features from existing data to capture important information. This could include:
    - Aggregating user activity (e.g., total clicks in the past month).
    - Creating interaction features (e.g., user-ad pair).
    - Encoding categorical variables (e.g., one-hot encoding for categorical features).
- **Feature Selection:** Identify and select the most relevant features to improve model performance and reduce complexity.

3. Split Data:

- **Training and Testing:** Divide the dataset into training and testing subsets. A common approach is to use an 80/20 split or cross-validation techniques like k-fold cross-validation.

4. Model Selection:

- **Choose Algorithms:** Select machine learning algorithms suitable for classification tasks. Common choices for CTR prediction include:
    - Logistic Regression
    - Decision Trees
    - Random Forests
    - Gradient Boosting Machines (e.g., XGBoost, LightGBM)
    - Neural Networks
- **Baseline Model:** Start with a simple model to establish a baseline performance.

5. Model Training:

- **Train Models:** Fit the selected models to the training data, adjusting parameters as needed.

- **Hyperparameter Tuning:** Optimize model performance by tuning hyperparameters using techniques like Grid Search or Randomized Search.

6. Model Evaluation:

- **Evaluate Performance:** Assess the model using the testing data. Key evaluation metrics include:
  - Accuracy
  - Precision
  - Recall
  - F1-Score
  - Area Under the ROC Curve (AUC-ROC)
- **Confusion Matrix:** Analyze the confusion matrix to understand the model's performance in terms of true positives, false positives, true negatives, and false negatives.

7. Model Interpretation:

- **Feature Importance:** Identify which features contribute most to the model's predictions. This helps in understanding model decisions and improving transparency.
- **SHAP Values:** Use SHAP (SHapley Additive exPlanations) values for detailed interpretation of how features impact predictions.

1.Exploratory Data Analysis (EDA)

Assignment _Zepto_visualization.ipynb

General Information

- **Total Entries**: 504,996 rows, indicating a substantial amount of data.
- **Total Columns**: 33, covering a wide range of information related to search terms and associated metrics.

Column Breakdown

1. **search_term**: The actual search term entered by users. Data type: `object`.
2. **product_variant_id**: Identifier for product variants. Data type: `object`.
3. **city_id**: Identifier for cities. Data type: `object`.
4. **query_type**: Type of query (e.g., head or tail). Data type: `object`.
5. **is_clicked**: Binary indicator of whether the search term was clicked (1 for clicked, 0 for not clicked). Data type: `float64`.

6. **total_clicks**: Total number of clicks for the search term. Data type: `float64`.
7. **session_views**: Number of views in a session for the search term. Data type: `float64`.
8. **query_products_clicks_last_30_days**: Clicks on products from the query in the last 30 days. Data type: `int64`.
9. **CTR_last_30_days**: Click-Through Rate (CTR) for the last 30 days. Data type: `float64`.
10. **CTR_last_7_days**: CTR for the last 7 days. Data type: `float64`.
11. **CTR_product_30_days**: CTR for the product in the last 30 days. Data type: `float64`.
12. **query_product_plt_clicks_60_days**: Clicks on the product from the query in the last 60 days. Data type: `float64`.
13. **query_product_plt_ctr_60_days**: CTR for the product from the query in the last 60 days. Data type: `float64`.
14. **CTR_plt_30_days**: CTR for the product in the last 30 days. Data type: `float64`.
15. **predicted_category_name**: Predicted category name for the search term. Data type: `object`.
16. **predicted_subcategory_name**: Predicted subcategory name for the search term. Data type: `object`.
17. **query_product_plt_clicks_30_days**: Clicks on the product from the query in the last 30 days. Data type: `float64`.
18. **product_name**: Name of the product. Data type: `object`.
19. **brand_name**: Brand name of the product. Data type: `object`.
20. **category_name**: Actual category name of the product. Data type: `object`.
21. **subcategory_name**: Actual subcategory name of the product. Data type: `object`.
22. **latest_margin**: Latest profit margin of the product. Data type: `float64`.
23. **savings**: Savings on the product. Data type: `float64`.
24. **savings_with_pass**: Savings with a membership pass. Data type: `float64`.
25. **ad_revenue**: Ad revenue generated from the product. Data type: `float64`.
26. **total_unique_orders**: Total number of unique orders for the product. Data type: `float64`.
27. **product_atcs_30_days**: Add to carts for the product in the last 30 days. Data type: `float64`.
28. **product_atcs_plt_30_days**: Add to carts for the product in the last 30 days across the platform. Data type: `float64`.
29. **total_unique_orders_plt_30_days**: Total unique orders for the product in the last 30 days across the platform. Data type: `float64`.
30. **product_ctr_city_30_days**: CTR for the product in a specific city in the last 30 days. Data type: `float64`.
31. **query_product_similarity**: Similarity score between the search term and the product. Data type: `float64`.
32. **search_term_length**: Length of the search term in characters. Data type: `int64`.

33. **is_popular_search_term**: Indicator if the search term is popular (1 for popular, 0 for not popular). Data type: `int64`.

Data Types

- **Object (10 columns)**: Mostly categorical data such as search terms, product and brand names, category information.
- **Float64 (20 columns)**: Numerical data including click metrics, CTRs, margins, savings, ad revenue, orders, etc.
- **Int64 (3 columns)**: Numerical data related to counts and binary indicators.

Memory Usage

- **Memory Usage**: 127.1+ MB, indicating the dataset is large but manageable for most modern systems.

DATA VISUALIZATION:

Inferred from the heat correlation map:

Highly Correlated Features:

- **`query_products_clicks_last_30_days` and `total_clicks` (0.97)**:
  - These two features are almost perfectly correlated. Including both in a model could lead to multicollinearity issues. It might be better to use one of them or create a composite feature.
- **`product_atcs_plt_30_days` and `product_atcs_30_days` (0.97)**:
  - Similar to the above, these features are highly correlated and may provide redundant information. Consider removing one or combining them.
- **`total_unique_orders` and `product_atcs_plt_30_days` (0.94)**:
  - This strong correlation suggests that customers who add products to their cart (ATCs) also tend to place unique orders. These could be combined into a single metric reflecting user engagement or purchase intent.

Moderately Correlated Features:

- **`CTR_product_30_days` and `CTR_last_30_days` (0.79)**:
  - Click-through rates (CTRs) over different periods are moderately correlated, indicating that past engagement is a good predictor of future engagement.
- **`query_product_plt_clicks_60_days` and `query_product_plt_clicks_30_days` (0.86)**:
  - This suggests that click behavior is consistent over these two time frames. Similar to the CTR, past clicks can be indicative of future behavior.
- **`CTR_plt_30_days` and `query_product_plt_ctr_60_days` (0.49)**:

    - There is a moderate correlation between the product click-through rate (CTR) and query product plot CTR over different time frames, indicating consistent user behavior patterns.

Low or Negative Correlations:

- **`query_product_similarity` and `is_clicked` (-0.38)**:
    - This negative correlation suggests that higher product similarity scores are associated with a lower likelihood of clicks. It could indicate that users prefer diverse or unique products rather than similar ones.
- **`search_term_length` and `is_clicked` (-0.09)**:
    - A negative but weak correlation indicates that longer search terms slightly reduce the probability of a click. It might be interesting to explore further to understand user search behavior.
- **`ad_revenue` and `latest_margin` (-0.02)**:
    - There is almost no correlation between ad revenue and latest margin, suggesting that these two variables are independent of each other.

Implications for Feature Engineering:

- **Combining Features**:
    - Given the high correlations, consider creating new features that combine related ones. For example, a composite score of user engagement could include clicks, ATCs, and unique orders.
- **Removing Redundant Features**:
    - To reduce multicollinearity, remove one of each pair of highly correlated features, such as `query_products_clicks_last_30_days` or `total_clicks`.
- **Interaction Terms**:
    - Investigate the interaction between features with moderate correlations. For instance, combining `CTR_product_30_days` and `CTR_last_30_days` might yield a feature that captures longer-term user engagement.

Predictive Modeling Considerations:

- **Target Variable Insights**:
    - If `is_clicked` is the target variable, focus on features with moderate to high positive or negative correlations, as they will likely have more predictive power.
- **Dimensionality Reduction**:
    - Use techniques like Principal Component Analysis (PCA) to reduce the feature space, especially given the high correlations between many features.

From **scatterplot**, titled "CTR Last 30 Days," we can infer several points regarding the Click-Through Rate (CTR) for various search terms over the last 30 days:

1. **Distribution of CTR**: Most search terms have a lower CTR, as evidenced by the concentration of points near the bottom of the plot. However, there are several search terms with significantly higher CTR values, suggesting that some terms are much more effective in generating clicks.
2. **Outliers**: There are a few search terms with exceptionally high CTR values (close to 1), which are outliers compared to the rest of the data. These terms are particularly effective in generating user engagement.
3. **Density of Search Terms**: The plot is densely packed with search terms on the x-axis, indicating a large number of different search terms being analyzed. The x-axis labels are heavily crowded, making individual terms difficult to read, but the overall trend can still be observed.
4. **CTR Spread Over Time**: There is a visible spread in the CTR values as we move from left to right on the x-axis, suggesting that the effectiveness of search terms varies widely. Some terms consistently perform better than others.
5. **Error Bars**: The vertical lines (error bars) extending from each data point might represent variability or uncertainty in the CTR measurements. Larger error bars indicate greater variability or less confidence in the CTR value for that search term.

Interpretation

- **Commonality of Low CTR**: Most search terms are less effective, with CTR values clustered near the bottom of the plot. This suggests that many search terms do not capture user attention effectively.
- **Moderate Effectiveness**: There are a significant number of terms with moderate effectiveness, as indicated by the spread of points in the middle range of the CTR values.
- **High Effectiveness**: A few search terms are highly effective, achieving near-perfect CTR values. These are the outliers that perform exceptionally well.

This box plot, titled "Box Plot of Total Clicks by Query Type," visualizes the distribution of total clicks for two different types of queries: "head" and "tail." Here's a detailed explanation of what this visualization signifies:

Box Plot Components

1. **Boxes**: The main part of each box represents the interquartile range (IQR), which contains the middle 50% of the data. The bottom and top edges of the boxes are the 25th and 75th percentiles, respectively.

2. **Median**: The horizontal line inside each box represents the median (50th percentile) of the total clicks.
3. **Whiskers**: The lines extending from the boxes (whiskers) indicate variability outside the upper and lower quartiles. They typically extend to 1.5 times the IQR from the quartiles, but in this plot, they reach the minimum and maximum data points within that range.
4. **Outliers**: If there were any data points outside the whiskers' range, they would be plotted individually as outliers. In this plot, there do not appear to be any outliers.

Analysis of the Box Plot

1. **Head Queries**:
   - **Median**: The median total clicks for head queries is around 125.
   - **Interquartile Range (IQR)**: The IQR for head queries ranges from approximately 110 to 140 clicks.
   - **Whiskers**: The whiskers extend from about 100 to 150 clicks, showing the overall range of clicks for head queries within this range.
2. **Tail Queries**:
   - **Median**: The median total clicks for tail queries is slightly higher than for head queries, around 130.
   - **Interquartile Range (IQR)**: The IQR for tail queries is wider, ranging from approximately 110 to 160 clicks.
   - **Whiskers**: The whiskers for tail queries extend from about 70 to 200 clicks, indicating a larger spread and more variability in the total clicks.

Comparison Between Head and Tail Queries

1. **Median Comparison**: The median total clicks for tail queries is slightly higher than for head queries, suggesting that, on average, tail queries receive more clicks than head queries.
2. **IQR Comparison**: The IQR for tail queries is wider than for head queries. This indicates that tail queries have a greater variability in the middle 50% of their total clicks.
3. **Range Comparison**: The whiskers for tail queries extend further than those for head queries, indicating that tail queries have a broader range of total clicks. This suggests that while some tail queries can perform very well, others may perform poorly.

Significance

- **Variability in Tail Queries**: The greater variability and wider range of total clicks for tail queries suggest that they can be highly unpredictable. While some tail queries can attract a high number of clicks, others might attract significantly fewer.
- **Performance of Head Queries**: Head queries tend to be more consistent, with a narrower IQR and less variability in total clicks. This suggests that head queries perform more reliably but may not achieve the highest possible number of clicks as frequently as some tail queries.

2. Feature Engineering: Develop and engineer relevant features to build a prediction model.

Assignment_Zepto_Feature_Selection.ipynb

Feature Engineering

Feature Engineering is a crucial step in developing a predictive model. It involves creating new features or modifying existing ones to improve the model's performance and its ability to make accurate predictions. Here's a short overview:

1. **Understanding the Data:**
    - Begin by analyzing the raw data to understand the underlying patterns, relationships, and potential issues.
2. **Creating New Features:**
    - **Transformations:** Apply mathematical transformations to existing features (e.g., log transformation, polynomial features) to capture nonlinear relationships.
    - **Aggregations:** Combine features to create summary statistics (e.g., mean, median, sum) for groups of data.
    - **Encoding Categorical Data:** Convert categorical variables into numerical form using techniques like one-hot encoding or label encoding.
    - **Feature Extraction:** Extract meaningful information from complex data (e.g., text, images) using domain-specific methods or algorithms.
3. **Feature Selection:**
    - **Importance Scores:** Identify which features are most influential in predicting the target variable using techniques like feature importance from models, correlation analysis, or statistical tests.
    - **Dimensionality Reduction:** Apply methods like Principal Component Analysis (PCA) to reduce the number of features while retaining essential information.
4. **Feature Scaling:**
    - Standardize or normalize features to ensure they contribute equally to the model, especially for algorithms sensitive to feature scales.
5. **Interaction Features:**
    - Create features that represent interactions between existing features (e.g., multiplying two features) to capture more complex relationships.

Model Building: Pre-process and Clean the Raw Features

Pre-processing and Cleaning Raw Features are essential steps in model building to ensure the data is suitable for training and to improve the model's performance. Here's a brief outline:

1. **Handle Missing Values:**
    - **Imputation:** Fill missing values with statistical measures (mean, median, mode) or use advanced techniques like interpolation or model-based imputation.
    - **Deletion:** Remove rows or columns with excessive missing values if imputation is not feasible.
2. **Remove Outliers:**
    - Identify and handle outliers that can skew the results, using methods like statistical thresholds or visualization techniques.
3. **Normalize and Scale Features:**
    - Ensure all features are on a similar scale by applying normalization or standardization, which helps algorithms converge faster and perform better.
4. **Encode Categorical Variables:**
    - Convert categorical variables into numerical format using encoding techniques to make them compatible with machine learning algorithms.
5. **Data Splitting:**
    - Split the dataset into training, validation, and test sets to evaluate the model's performance effectively and avoid overfitting.
6. **Feature Engineering:**
    - Apply feature engineering techniques (described above) to enhance the dataset before model training.

2. Feature Pre-Processing

a. Feature Engineering:

- **Categorical Encoding:** Techniques such as one-hot encoding, embeddings, and target encoding.
- **Feature Transformation:** Normalization, scaling, and discretization.
- **Interaction Features:** Creating features that capture interactions between different categorical and numerical features.

b. Feature Selection:

- **Filter Methods:** Statistical tests like chi-squared, mutual information.
- **Wrapper Methods:** Using models to assess feature subsets.
- **Embedded Methods:** Feature importance from models like decision trees and regularization methods like Lasso.

c. Handling Missing Data:

- **Imputation:** Using mean, median, mode, or more complex methods like K-nearest neighbors (KNN) imputation.
- **Feature Engineering:** Creating indicators for missing data.

Option 1: Remove Rows with Any Missing Values

1. **Remove rows with missing values:** This approach removes all rows that contain any missing values, resulting in a DataFrame that has no missing data but may have fewer rows.

Option 2: Fill Missing Values (Imputation)

2. **Fill missing values in numerical columns with the median:** This approach identifies numerical columns and fills any missing values in these columns with the median value of the respective column.
3. **Fill missing values in categorical columns with the most frequent value:** This approach identifies categorical columns and fills any missing values in these columns with the most frequent value (mode) of the respective column.

Saving the Cleaned Data

4. **Save the cleaned data to new CSV files:** Both the DataFrame with removed rows and the DataFrame with imputed values are saved to separate CSV files for further use or analysis.

Print the Cleaned Data

5. **Print the first few rows of the cleaned data:** The first few rows of both cleaned datasets are printed to the console to provide a quick view of the results.

Cleaned and Preprocessed Data

1. **First Few Rows of the Cleaned Data:**
   - The `print(data_cleaned.head())` statement displays the first few rows of the cleaned and preprocessed data.
   - This output shows the transformed numerical and categorical columns, now ready for further analysis or modeling.
   - It demonstrates that missing values have been filled, numerical columns have been scaled, and categorical columns have been one-hot encoded.

Correlation Analysis

2. **Correlation Matrix:**
   - The `correlation_matrix = data_cleaned.corr()` calculates the Pearson correlation coefficients between pairs of features in the cleaned data.
   - This matrix shows how strongly each feature is linearly related to every other feature. Values range from -1 to 1, where:
     - 1 indicates a perfect positive linear relationship.

        - -1 indicates a perfect negative linear relationship.
        - 0 indicates no linear relationship.

3. **Heatmap of the Correlation Matrix:**
    - The `sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm')` visualizes the correlation matrix as a heatmap.
    - The color intensity and annotations represent the correlation values between features.
    - This plot helps to quickly identify pairs of features that are highly correlated.

Insights from the Correlation Matrix and Heatmap

- **High Correlation:** Features with high positive or negative correlations might provide redundant information. For example, if two features have a correlation close to 1 or -1, they essentially carry the same information. One of these features could potentially be dropped to simplify the model.
- **Low or No Correlation:** Features with low or no correlation (values close to 0) are less likely to provide redundant information. These features can independently contribute to the predictive power of a model.
- **Multicollinearity:** High correlation among predictor variables can cause multicollinearity issues in regression models, leading to less reliable coefficient estimates. Identifying and addressing multicollinearity is crucial in building robust models.

Mean Squared Error (MSE)

Mean Squared Error: 0.12659483881126682

- **Interpretation:**
    - The MSE is a measure of the average squared difference between the predicted values and the actual values. A lower MSE indicates that the model's predictions are closer to the actual values.
    - In this case, an MSE of 0.1266 suggests that, on average, the squared difference between the predicted and actual values is relatively small, implying that the model is performing reasonably well.

Feature Importance

Feature Importance:

The feature importance scores indicate the relative importance of each feature in making predictions with the model. Higher absolute values signify greater importance, while lower absolute values (including negative values) indicate lesser importance.

- **Top Features:**
  - `query_type_head` (27.737006): This feature has the highest importance score, suggesting it is the most significant predictor in the model.
  - `query_type_tail` (27.486191): This feature is also highly important.
  - `query_type_5.342857142857146` (18.541543) and `query_type_54.01` (18.429675): These features are also key predictors.
  - `predicted_category_name_{'Cleaning Essentials': 1, 'Electricals & Accessories': 0.5, 'Zepto Cafe': 2}` (17.673990): This categorical feature has a significant impact on the model's predictions.
- **Least Important/Negative Impact Features:**
  - `query_type_82.537090909091` (-17.933973): This feature has a negative importance score, indicating it negatively affects the model's predictions.
  - `product_variant_id_Breakfast & Sauces` (-55.223197), `city_id_Honey & Spreads` (-55.223197), `predicted_category_name_0` (-55.223197): These features have large negative importance scores, suggesting they significantly detract from the model's predictive power.
  - `query_type_66.51890909090915` (-78.733301): This feature has the most substantial negative impact on the model.

Implications

1. **Model Performance (MSE):**
   - The MSE indicates the model's accuracy, and a value of 0.1266 shows that the model performs reasonably well in predicting the target variable.
2. **Feature Importance:**
   - **Highly Important Features:** Features like `query_type_head` and `query_type_tail` are crucial for the model, meaning any changes to these features will significantly impact predictions. These should be carefully monitored and potentially prioritized in data collection and preprocessing.
   - **Negatively Impactful Features:** Features with large negative importance scores, such as `query_type_66.51890909090915`, may indicate issues like multicollinearity, noise, or features that confuse the model. These should be reviewed, and if necessary, removed or transformed.
   - **Moderate and Less Important Features:** Features with smaller importance scores might still contribute to the model but to a lesser extent. These can be considered for feature selection or dimensionality reduction to simplify the model without losing significant predictive power.

WHAT CAN BE DONE NEXT?

- **Refinement:**

  - Investigate and potentially remove or transform features with large negative importance scores to improve the model.
  - Consider feature engineering or interaction terms for highly important features to enhance their predictive power further.
- **Validation:**
  - Perform cross-validation to ensure the MSE and feature importance scores are consistent across different data splits, confirming the model's robustness.
- **Optimization:**
  - Fine-tune hyperparameters and experiment with different algorithms to see if a better performance (lower MSE) can be achieved.

CTR PREDICTION AND RANKING PRODUCTS MODEL

Training model.ipynb

1. Ranking Techniques

a. Logistic Regression:

A baseline model for CTR prediction. It estimates the probability of a click using a logistic function and is simple yet effective for small to medium-sized datasets.

b. Gradient Boosting Machines (GBM):

Techniques like XGBoost, LightGBM, and CatBoost are popular for their efficiency and performance in CTR prediction. They work by iteratively improving the model based on errors from previous iterations.

c. RankNet, LambdaMART:

These are specific algorithms for learning to rank, which can be applied to CTR prediction. RankNet uses neural networks, while LambdaMART improves on traditional gradient boosting by incorporating rank-aware loss functions.

d. Deep Learning-based Ranking Models:

Neural networks that can learn complex representations of features and interactions. Examples include DeepFM and Wide & Deep models.

3. Machine Learning Architectures

a. Linear Models:

- **Logistic Regression:** For baseline CTR prediction.

- **Generalized Linear Models (GLM):** Can be extended to handle various distributions.

b. Tree-Based Models:

- **Decision Trees:** Basic building blocks for ensemble methods.
- **Random Forests:** Ensemble of decision trees for better performance.
- **Gradient Boosting Models (GBM):** Including XGBoost, LightGBM, and CatBoost.

c. Neural Networks:

- **Feedforward Neural Networks (FNN):** Standard architecture for learning complex patterns.
- **Wide & Deep Learning:** Combines linear models (wide) with deep neural networks to capture both memorization and generalization.
- **DeepFM (Deep Factorization Machine):** Combines factorization machines with deep learning to capture both feature interactions and non-linearities.
- **Deep Neural Networks (DNN):** For high-dimensional and complex datasets.

d. Hybrid Models:

- **Wide & Deep Learning:** Integrates wide linear models and deep neural networks to leverage both feature memorization and generalization.
- **Neural Collaborative Filtering (NCF):** Combines neural networks with collaborative filtering techniques for recommendation systems.

APPROACH 1 : XGBOOST MODEL

Load the Data: The code reads a CSV file into a DataFrame.

Define Features and Target: It separates the data into features (`X`) and the target variable (`y`).

Convert Target Variable: The target variable is converted to integer type to prepare it for modeling.

Handle Categorical Features: The code identifies columns with string data and converts them to categorical type, which is necessary for the model.

Train XGBoost Model: An XGBoost classifier is trained on the features and target variable, with categorical handling enabled.

Extract Feature Importances: After training, the importance of each feature is extracted from the model.

Create and Sort DataFrame: A DataFrame is created to display feature importances and is sorted to show the most important features first.

This code is used to visualize the feature importances from the XGBoost model and display the top features. Here's what it does:

1. **Plot Feature Importances:** It creates a horizontal bar chart where each bar represents the importance of a feature in predicting the target variable (`is_clicked`). The length of each bar indicates the relative importance of the feature.
2. **Set Plot Details:** The x-axis is labeled "Feature Importance," and the y-axis is labeled "Feature." The chart's title is set to "Feature Importances for Predicting is_clicked."
3. **Invert Y-Axis:** The y-axis is inverted so that the most important features appear at the top of the chart.
4. **Display the Plot:** The chart is displayed using `plt.show()`.
5. **Print Top Features:** It prints the top 10 features with the highest importance, showing which features contribute the most to the model's predictions.

This output provides an evaluation of the model's performance. Here's a detailed explanation of each part:

1. Model Accuracy:

- **Accuracy: 1.00**
  - The model's accuracy is 100%, meaning it correctly predicted the target variable for all the samples in the dataset. This suggests the model perfectly classified every instance.

2. Classification Report:

The classification report provides detailed metrics for each class in the target variable. In this case, the target variable has two classes: `0` and `1`. The report includes the following metrics for each class:

- **Precision:**
  - Measures the proportion of true positive predictions among all positive predictions made by the model.
  - For both classes (`0` and `1`), precision is 1.00, indicating that every prediction made as a positive (either `0` or `1`) was correct.
- **Recall:**
  - Measures the proportion of true positive predictions among all actual positives in the data.
  - For both classes, recall is 1.00, meaning the model identified every actual positive instance correctly.

- **F1-Score:**
  - The harmonic mean of precision and recall, providing a single metric that balances both precision and recall.
  - For both classes, the F1-score is 1.00, indicating a perfect balance between precision and recall.
- **Support:**
  - The number of actual occurrences of each class in the dataset.
  - For class `0`, there are 70,393 instances, and for class `1`, there are 30,607 instances.

3. Overall Metrics:

- **Accuracy:**
  - The overall accuracy of the model, which is 1.00 or 100%.
- **Macro Average:**
  - Averages the precision, recall, and F1-score across all classes, treating all classes equally.
  - All macro averages are 1.00, indicating perfect performance across all classes.
- **Weighted Average:**
  - Averages the precision, recall, and F1-score across all classes, weighted by the number of instances in each class.
  - All weighted averages are 1.00, reflecting perfect performance when taking into account the class distribution.

This code performs several key steps to evaluate and optimize a machine learning model using a Random Forest classifier.

1. **Logging Configuration:**
   - Sets up logging to record information about the process, including timestamps and messages.
2. **Feature and Target Definition:**
   - Defines lists of numerical and categorical features to be used in the model, along with the target variable (`is_clicked`).
3. **Data Preparation:**
   - Splits the dataset into features (`X`) and the target variable (`y`).
   - Handles missing values by filling them with zeros.
   - Converts the target variable to integer type if necessary.
4. **Subsampling:**
   - Creates a smaller subset of the data for faster processing, specifying a sample size and splitting it into training and test sets.
5. **Data Preprocessing:**
   - Sets up preprocessing steps to standardize numerical features and one-hot encode categorical features using a column transformer.
6. **Model Pipeline:**

    - Constructs a pipeline that includes preprocessing steps followed by a Random Forest classifier.
7. **Hyperparameter Tuning:**
    - Defines a parameter grid with different values for the Random Forest hyperparameters.
    - Uses RandomizedSearchCV to search for the best combination of hyperparameters through cross-validation.
8. **Model Training:**
    - Splits the subsampled data into training and testing sets.
    - Fits the RandomizedSearchCV object on the training data and logs the time taken for this process.
9. **Model Evaluation:**
    - Extracts the best model from the RandomizedSearchCV.
    - Makes predictions on the test data and evaluates the model using accuracy, confusion matrix, and classification report.
    - Prints the evaluation results and the best hyperparameters found during the search.

Overall, this code performs data preparation, sets up a machine learning pipeline, tunes hyperparameters, trains the model, and evaluates its performance, while also logging and reporting key results.

APPROACH 2: Using RandomForest model:

1. Model Performance Metrics:

- **Accuracy:**
    - The model achieved an accuracy of 92%, meaning it correctly predicted the target variable for 92% of the samples in the dataset.
- **Confusion Matrix:**
    - **True Negatives:** The model correctly identified 13,932 instances as class `0`.
    - **False Positives:** There were no instances where the model incorrectly predicted class `1` when the true class was `0`.
    - **False Negatives:** The model incorrectly identified 1,632 instances as class `0` when the true class was `1`.
    - **True Positives:** The model correctly identified 4,436 instances as class `1`.
- **Classification Report:**
    - **Class `0`:**
        - **Precision:** 90% of the predictions for class `0` were correct.

        - **Recall:** The model correctly identified 100% of the actual class `0` instances.
        - **F1-Score:** The balance between precision and recall for class `0` is 94%.
    - **Class `1`:**
        - **Precision:** 100% of the predictions for class `1` were correct.
        - **Recall:** The model correctly identified 73% of the actual class `1` instances.
        - **F1-Score:** The balance between precision and recall for class `1` is 84%.
    - **Overall Metrics:**
        - **Accuracy:** 92% overall accuracy.
        - **Macro Average:** Averages precision, recall, and F1-score across both classes, with high values indicating strong performance across classes.
        - **Weighted Average:** Takes into account the number of instances in each class, showing good performance when considering the class distribution.

2. Hyperparameters:

- **Best Hyperparameters:**
    - The model's performance was optimized with specific settings, including the use of bootstrapping, a maximum tree depth of 20, and 206 trees in the forest.
- **Inference:**
    - The model performs well overall with high accuracy and balanced metrics for class `0`. However, there's a trade-off in recall for class `1`, where some instances are missed. The model is well-tuned, but attention to the balance between precision and recall is important, especially if misclassifications have significant implications.
- ANOTHER PERFORMANCE METRIC:

To calculate the Normalized Discounted Cumulative Gain (NDCG) for evaluating the ranking performance

1. **Define the DCG and NDCG functions**.
2. **Train the model** using the relevant feature and target variable.
3. **Generate predictions** and compute NDCG for the predictions.

The `ndcg_score` function from `sklearn.metrics` computes the Normalized Discounted Cumulative Gain, which is a common metric for evaluating the quality of rankings.

HISTORY OF CTR PREDICTION MODELS:

Early Models and Feature Engineering

1. Logistic Regression and Linear Models (2000s):
    - Initially, simple logistic regression and other linear models were used for CTR prediction. These models relied heavily on manual feature engineering to capture interactions and dependencies between different features.
    - Studies by McMahan et al. (2013) and Rendle (2010) contributed significantly during this period, with logistic regression being a popular choice due to its interpretability and simplicity (ar5iv).

Incorporation of Factorization Machines

2. Factorization Machines (FMs):
    - Introduced by Rendle (2010), Factorization Machines enhanced the ability to model feature interactions without extensive manual feature engineering. FMs became a foundational technique for CTR prediction, especially in recommendation systems.
    - The combination of FMs with logistic regression provided a robust way to handle large sparse datasets typical in CTR prediction tasks (ar5iv).

Rise of Deep Learning

3. Deep Learning Models (mid-2010s):
    - The advent of deep learning brought significant advancements to CTR prediction models. Deep neural networks (DNNs) could automatically learn complex feature interactions and hierarchical representations from raw data.
    - DeepFM (2017): Guo et al. introduced DeepFM, combining factorization machines with deep neural networks to leverage the strengths of both. This model captured low- and high-order feature interactions, improving prediction accuracy (ar5iv).
    - PNN (Product-based Neural Networks): Proposed by Qu et al. (2016), PNNs used a product layer to capture feature interactions explicitly within the neural network structure, offering a different approach from traditional deep learning models (ar5iv).

REFERENCES AND LITERATURE REVIEW:

For research on Click-Through Rate (CTR) prediction models, several recent papers offer valuable insights and advancements:

1. **Helen: Optimizing CTR Prediction Models with Frequency-wise Hessian Eigenvalue Regularization** - This paper addresses the challenge of skewed feature distribution in CTR prediction models. It introduces a method to clip the gradient of feature embeddings based on feature frequencies and focuses on the impact of feature frequency on the dominant eigenvalue of the Hessian matrix, enhancing the optimization process for better model performance (ar5iv).
2. **Star+: A New Multi-Domain Model for CTR Prediction** - The Star+ model extends the Star model by incorporating domain-specific and shared fully connected networks (FCNs) for different domains, along with various fusion strategies to balance the influence of domain-specific and shared information. This approach aims to improve the precision of CTR predictions across multiple domains by learning both commonalities and domain-specific characteristics (ar5iv).
3. **Deep Pattern Network (DPN) for Click-Through Rate Prediction** - DPN introduces a novel architecture that leverages user behavior patterns to improve CTR prediction. It employs a Target-ware Pattern Retrieval Model (TPRM) to identify relevant patterns from user behavior sequences and refines these patterns using a pre-trained network, ultimately enhancing the model's ability to predict user interactions (ar5iv).

HOW CAN THE ACCURACY OF THE MODEL BE INCREASED MORE?

1. Feature Engineering

- Domain-Specific Features: Develop features that capture the nuances of your specific domain. For example, if you are predicting CTR for e-commerce, features like time since the last purchase, browsing behavior, and product categories can be significant.
- Interaction Features: Create features that represent interactions between different variables, which might capture more complex patterns.
- Temporal Features: Incorporate time-based features like time of day, day of the week, or seasonality to capture temporal trends.

2. Model Selection and Ensemble Methods

- Advanced Algorithms: Experiment with advanced algorithms like gradient boosting machines (e.g., XGBoost, LightGBM), deep learning models, and ensemble methods.
- Ensemble Methods: Combine different models to improve performance. Stacking, bagging, and boosting can provide significant improvements.

3. Hyperparameter Tuning

- Use automated hyperparameter optimization techniques like Grid Search, Random Search, or Bayesian Optimization to fine-tune your models for better performance.

4. Evaluation Metrics and Validation

- Robust Validation: Use techniques like cross-validation and hold-out validation to ensure your model's robustness.
- Multiple Metrics: Evaluate your model using various metrics like AUC-ROC, Precision-Recall AUC, F1-score, in addition to just accuracy.

5. Explainability and Interpretability

- Use tools like SHAP or LIME to make your model interpretable. Providing insights into why your model makes certain predictions can build trust and reveal valuable business insights.

6. Regular Updates and Monitoring

- Model Drift: Continuously monitor and update your model to handle changes in user behavior and underlying data distribution.
- A/B Testing: Implement A/B testing to measure the impact of your model in a live environment and gather continuous feedback.