Assignment 2: Supervised Learning Model

Nainil Rajendra Maladkar
NUID: 002780019
maladkar.n@northeastern.edu

TOPIC: Advertisement Success Prediction

PROBLEM DEFINITION:

Avito is Russia's largest classified advertisements website.

I work on Avito's online advertisement data and understand how to train a machine learning model to predict the **deal probability**.
With this information, Avito can inform sellers on how to best optimize their listing and indicate how much interest they should realistically expect to receive.

Source: Avito Demand Prediction Challenge | Kaggle

This project aims to predict demand for an online advertisement based on its full description (title, description, images, etc.), its context (geographically where it was posted, similar ads already posted) and historical demand for similar ads in similar contexts.
With this information, Avito can inform sellers on how to best optimize their listing and provide some indication of how much interest they should realistically expect to receive.
For each ***item_id*** in the test set, you must predict a probability for the ***deal_probability***

DATA COLLECTION AND PREPARATION:

This project uses Supervised Regression Machine Learning methods such as Linear regression, decision tree, and ensemble models such as RandomForest, XGBoost using Scikit Learn, and XGBoost.
In supervised learning, the goal is to learn the mapping (the rules) between a set of inputs and output(s).
I have used Pandas, Numpy, Matplotlib, Seaborn, and Plotly to perform exploratory data analysis and gather insights for machine learning.

```
kaggle competitions download avito-demand-prediction -f train.csv
kaggle competitions download avito-demand-prediction -f test.csv
kaggle competitions download avito-demand-prediction -f sample_submission.csv
```

Data preparation — Train Val Split, Encoding, Imputing and Scaling

The train data is 1,503,424 rows. ***deal_probability*** is the target variable.
Upon exploring the data, we see that there are missing data and duplicates.
I need to remove the rows with more than 20% missing data and duplicate rows.

The following Columns are included in Train dataset

- **user_id** - User id.
- **region** - Ad region.
- **city** - Ad city.
- **parent_category_name** - Top level ad category as classified by Avito's ad model.
- **category_name** - Fine grain ad category as classified by Avito's ad model.
- **param_1** - Optional parameter from Avito's ad model.
- **param_2** - Optional parameter from Avito's ad model.
- **param_3** - Optional parameter from Avito's ad model.
- **title** - Ad title.
- **description** - Ad description.
- **price** - Ad price.
- **item_seq_number** - Ad sequential number for user.
- **activation_date**- Date ad was placed.
- **user_type** - User type.
- **image** - Id code of image. Ties to a jpg file in train_jpg. Not every ad has an image.
- **image_top_1** - Avito's classification code for the image.
- **deal_probability** - The target variable. This is the likelihood that an ad actually sold something. It's not possible to verify every transaction with certainty, so this column's value can be any float from zero to one.

EXPLORATORY DATA ANALYSIS:

The top left plot shows the relationship between price and deal_probability which is the likelihood of an item being sold.

- The plot shows that the deal probability is higher for lower priced items, and lower for higher priced items.
- This makes sense, as cheaper items are more likely to attract buyers.
- The plot also shows that there are some outliers, such as items with very high prices and high deal probabilities, or items with very low prices and low deal probabilities.
- These outliers may be due to other factors, such as the quality, popularity, or availability of the items.

The top middle plot shows the relationship between item_seq_number and deal_probability.

- The item sequence number is a unique identifier for each item posted by a seller.
- The plot shows that the deal probability is lower for items with higher sequence numbers, and higher for items with lower sequence numbers.
- This suggests that older items are more likely to be sold than newer items.
- This may be because older items have more exposure, reviews, or discounts than newer items.

The top right plot shows the relationship between activation_date and item_seq_number.

- The activation date is the date when the item was posted on the marketplace.
- The plot shows that the item sequence number increases over time, as more items are posted by the same seller.
- The plot also shows some periodic patterns, such as spikes and dips in the item sequence number.
- These patterns may be due to seasonal or weekly variations in the seller's activity.

The middle left plot shows the distribution of price across different months.

- The plot shows that the price varies by month, with some months having higher or lower average prices than others.
- The plot also shows that the price has a skewed distribution, with most of the values concentrated in the lower range, and a few values in the higher range.
- This indicates that there are more low-priced items than high-priced items on the marketplace.

The middle middle plot shows the distribution of price across different years.

- The plot shows that the price increases over time, as the average price of items is higher in 2023 than in 2022 or 2021.
- This may be due to inflation, increased demand, or improved quality of the items over time.

The middle right plot shows the distribution of price across different dayofweek, which is the day of the week when the item was posted.

- The plot shows that the price varies by day of week, with some days having higher or lower average prices than others.
- The plot also shows that the price has a similar skewed distribution as in the previous plots.

The bottom left plot shows the relationship between isweekend and deal_probability.

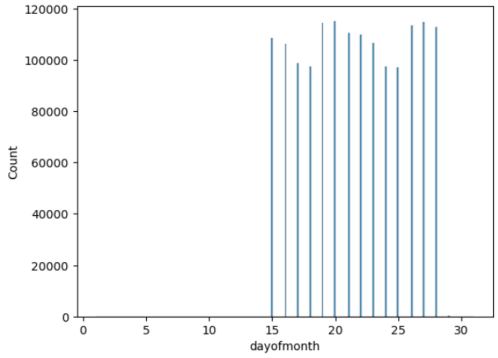
- The isweekend variable indicates whether the item was posted on a weekend or not.
- The plot shows that the deal probability is slightly higher for items posted on weekends than on weekdays.
- This may be because buyers have more time to browse and purchase items on weekends than on weekdays.

The bottom middle plot shows the distribution of price_log, which is the natural logarithm of price.

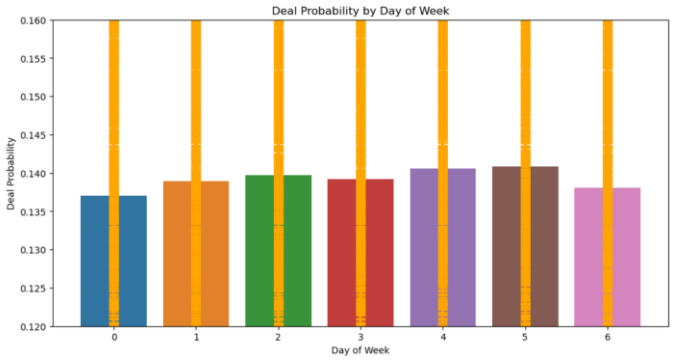
- The plot shows that the price_log has a more symmetric and normal distribution than price, which means that it has less skewness and outliers.
- This makes it easier to analyze and model the data using statistical methods.

The bottom right plot shows the relationship between year and dayofweek.

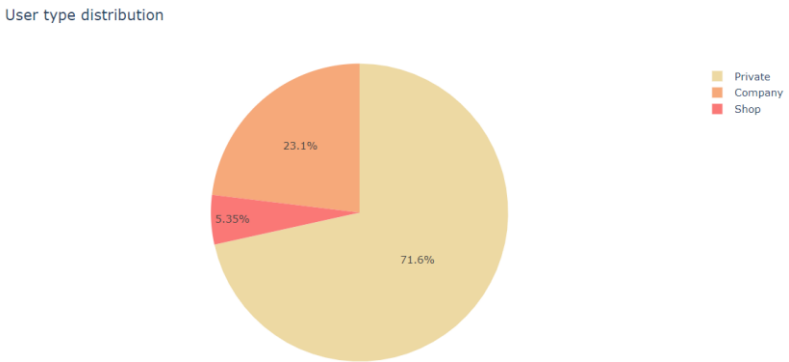
- The plot shows that there are some differences in the frequency of posting items by year and day of week.
- For example, in 2021, there were more items posted on Mondays than on other days, while in 2023, there were more items posted on Thursdays than on other days.
- This may reflect some changes in the seller's behavior or preferences over time.



We see that the ads are activated on all days of the month. The days vary from the 15th of the month through 28th of March (only month in the sample).



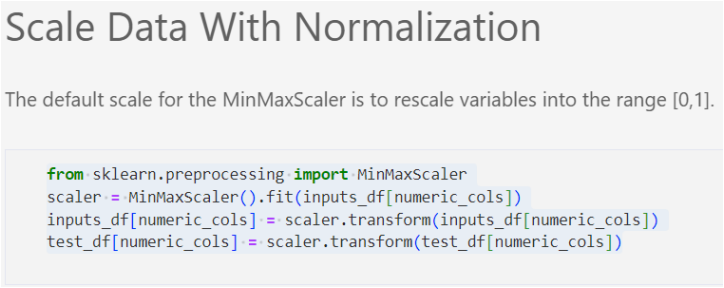
`Saturday` reports the maximum `deal_probaliliy` for the online ads followed by `Friday



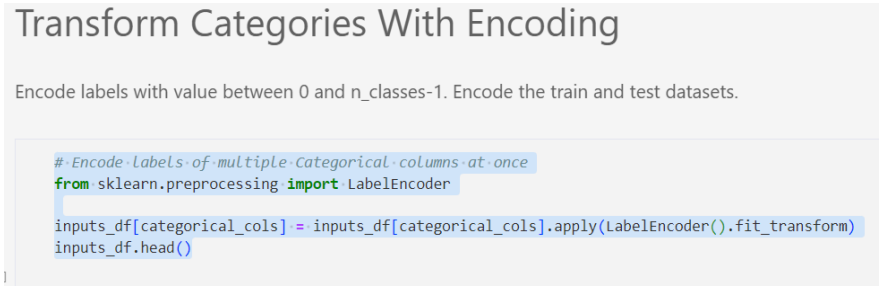
`71.6%` of online ads are from `'Private'`,` 23.1%` are from `'Company'` and `5.35%` are from `'Shop'`.

FEATURE ENGINEERING:

1) Define the inputs and targets dataset.
2) Identify the features as numeric and categorical.
3) Imputing missing values
4) Imputing the null values in the numeric features price ,image_top_1 a column with -9999 categorical features with the value of Missing both input and test datasets.
5) Since the input dataset does not have the description and user_id columns, we'll remove those from the test dataset as well.
6) Scale Data With Normalization
7) The default scale for the MinMaxScaler is to rescale variables into the range [0,1].
8) Transform Categories With EncodingEncode labels with value between 0 and n_classes-1. We'll encode the train and test datasets
9) Splitting the training, validation inputs and targets
10) Once all the pre-processing is done, we can split the input data into a train and validation set.
11) Defining the X_train, X_val, and X_test from train, validation, and test dataset.



- Create a MinMaxScaler object.
- Fit the scaler to the numeric columns of the training data. This calculates the minimum and maximum values for each column in the training data.
- Transform the numeric columns of the training data using the scaler object. This scales each column to the range specified in the feature_range parameter of the MinMaxScaler constructor. By default, the range is 0 to 1.
- Transform the numeric columns of the test data using the same MinMaxScaler object. This ensures that the training and test data are scaled on the same scale.



- The first line imports the LabelEncoder class from scikit-learn.
- The second line creates a new DataFrame inputs_df[categorical_cols], which contains only the categorical columns from the original inputs_df DataFrame.
- The third line applies the LabelEncoder().fit_transform() function to each column in the inputs_df[categorical_cols] DataFrame. This function encodes the labels in each column to unique integer values.
- The fourth line prints the first five rows of the inputs_df DataFrame to show the encoded labels.

Feature Selection includes reducing the number of input variables when developing a predictive model.

```
# Removing user_id column
input_cols = ['region',
              'city',
              'parent_category_name',
              'category_name',
              'param_1',
              # 'param_2',
              # 'param_3',
              'title',
              'price',
              'user_type',
              'image_top_1',
              'month',
              'dayofmonth',
              'year',
              'dayofweek',
              'isweekend'
              ]
target_col = 'deal_probability'
```

Fetching the month, day of month, year, day of week and is a weekend fields from the `activation date` for `test_df`

```
# Identify numeric and categorical columns

numeric_cols = ['price', 'image_top_1']
categorical_cols = ['region', 'city', 'parent_category_name', 'category_name', 'param_1', 'title',
                    'user_type', 'month', 'dayofmonth', 'dayofweek', 'isweekend']
```

```
test_df['activation_date'] = pd.to_datetime(train_df['activation_date'])

test_df['month'] = train_df.activation_date.dt.month
test_df['dayofmonth'] = train_df.activation_date.dt.day
test_df['year'] = train_df.activation_date.dt.year
test_df['dayofweek'] = train_df.activation_date.dt.weekday
test_df['isweekend'] = (train_df.activation_date.dt.weekday > 4).astype(int)
```

```
test_df['month'] = train_df.activation_date.dt.month
test_df['dayofmonth'] = train_df.activation_date.dt.day
test_df['year'] = train_df.activation_date.dt.year
test_df['dayofweek'] = train_df.activation_date.dt.weekday
test_df['isweekend'] = (train_df.activation_date.dt.weekday > 4).astype(int)
```

```
Twt_OG_columns_df = test_df.columns
print(Twt_OG_columns_df.tolist())
```

```
['item_id', 'user_id', 'region', 'city', 'parent_category_name', 'category_name', 'param_1', 'param_2', 'param_3', 'title', 'de
scription', 'price', 'item_seq_number', 'activation_date', 'user_type', 'image', 'image_top_1', 'month', 'dayofmonth', 'year',
'dayofweek', 'isweekend']
```

MODEL DEVELOPMENT:

```
X_train = train_inputs[numeric_cols + categorical_cols]
X_val = val_inputs[numeric_cols + categorical_cols]
X_test = test_df[numeric_cols + categorical_cols]
```

First, we will define a baseline model to compare with other models.

RMSE

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

where y hat is the predicted value, and y is the original value.

```
# mean calculation
def return_mean(inputs):
    return np.full(len(inputs), train_df.deal_probability.mean())
```

```
train_preds = return_mean(X_train)
train_preds
```

```
mean_squared_error(train_preds, train_targets, squared=False)
```

```
0.26018308019272957
```

Base value of RMSE is `0.26018308019272957`. Any model we train should have a lower RMSE than `0.26018308019272957`

- **LINEAR REGRESSION**

```
%%time
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on training data and validation data
X_train_imputed = imputer.fit_transform(X_train)
X_val_imputed = imputer.transform(X_val)

# Initialize the model
model = LinearRegression()

# Fit the model on the imputed data
model.fit(X_train_imputed, train_targets)

# Generate predictions
train_preds = model.predict(X_train_imputed)
val_preds = model.predict(X_val_imputed)

# Compute RMSE for training set
train_rmse = mean_squared_error(train_targets, train_preds, squared=False)

# Compute RMSE for validation set
val_rmse = mean_squared_error(val_targets, val_preds, squared=False)

# Print the results
print('Linear Regression: \n Train RMSE: {}, Validation RMSE: {}'.format(train_rmse, val_rmse))
```

Create an imputer object. The imputer object will be used to impute missing values in the dataset. The script uses the SimpleImputer class with the mean strategy, which will replace missing values with the mean value of the column.

Fit the imputer on training and validation data. This will ensure that the imputer learns the distribution of the data and can accurately impute missing values.

Initialize a linear regression model.

```
Linear Regression:
 Train RMSE: 0.2543232730488643, Validation RMSE: 0.2539304556320603
CPU times: total: 1.61 s
Wall time: 2.86 s
```

The validation set RMSE is `0.2539304556320603`, for implemented Linear Regression model

---

- **DECISION TREES**

```
Decision Tree Regressor:
 Train RMSE: 0.022688845258562816, Validation RMSE: 0.32734713555094835
CPU times: total: 23.5 s
Wall time: 41.1 s
```

The validation set RMSE is `0.32734713555094835`, for Decision Tree model, which is considerably worse than base model

---

- **RANDOM FOREST**

```
Random Forest Regressor:
 Train RMSE: 0.09083673831889051, Validation RMSE: 0.23761538865011914
CPU times: total: 37min 30s
Wall time: 51min 48s
```

The validation set RMSE is `0.23761538865011914`, for Random Forest model

- **RANDOM FOREST WITH HYPER PARAMETER TUNING USING CROSS VALIDATION**

`RandomizedSearchCV` is a class in scikit-learn that implements a randomized parameter search over parameter settings.
It works by randomly sampling a given number of parameter settings from a specified distribution and then evaluating the performance of each setting on a held-out test set.
The parameter setting that results in the best performance on the test set is then returned as the best hyperparameters.
`RandomizedSearchCV` is similar to `GridSearchCV`, but it is more computationally efficient, especially when there are many hyperparameters to tune.
This is because `RandomizedSearchCV` does not evaluate all possible parameter combinations, but instead samples a random subset of them.

```
Best Hyperparameters:
{'max_depth': 17, 'min_samples_leaf': 10, 'min_samples_split': 17, 'n_estimators': 24}
Random Forest Regressor:
 Train RMSE: 0.22056144662630212, Validation RMSE: 0.23065331709296805
CPU times: total: 3min 17s
Wall time: 44min 44s
```

`Best Parameters` include `max_depth : 17`, `min_samples_leaf : 10`, `min_samples_split : 17`, `n_estimators : 24`

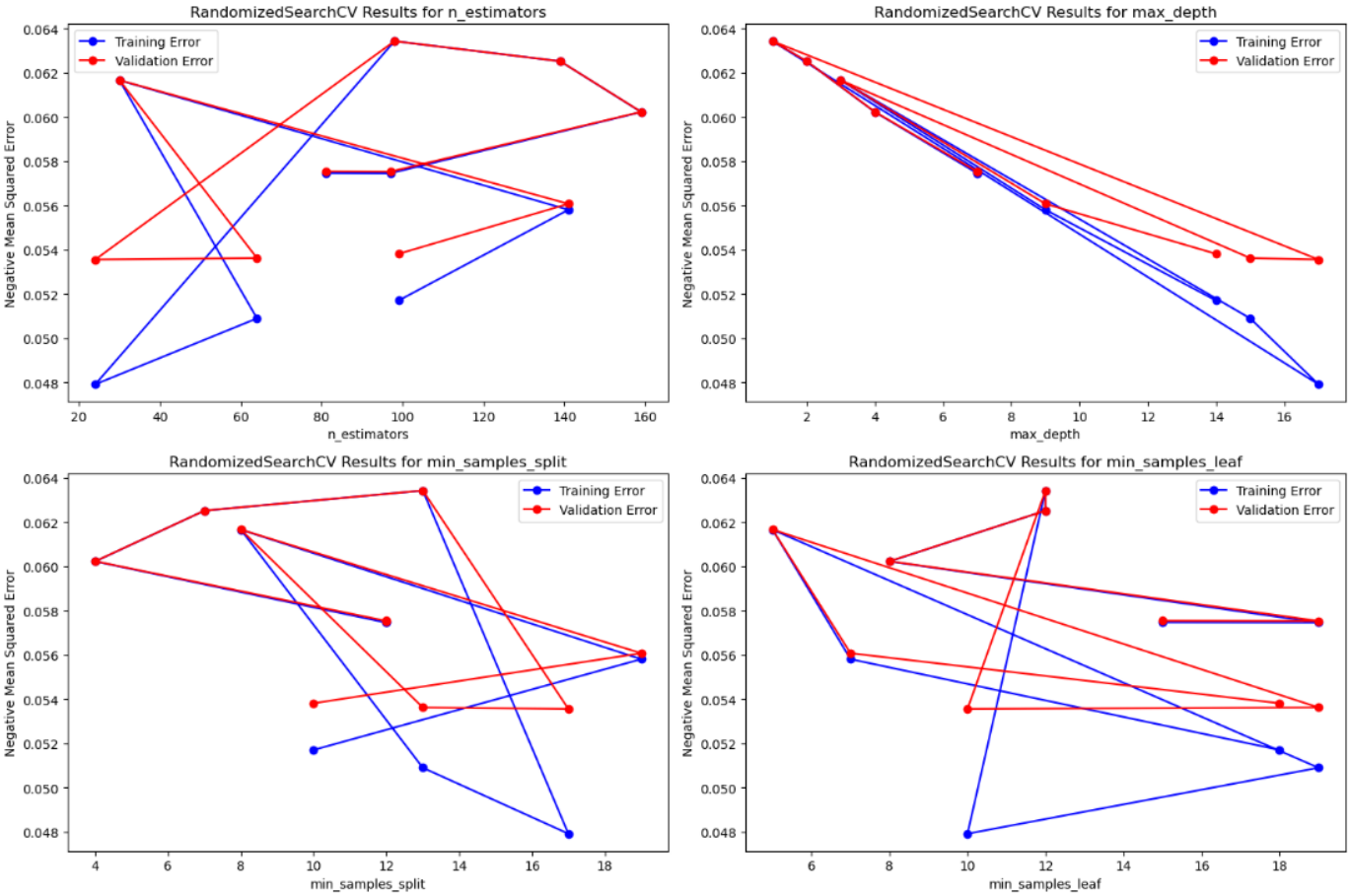
The validation set RMSE is `0.23065331709296805`, for Random Forest model with hyper parameter tuning using CV

- **XGBOOST**

```
XGBoost Regressor:
 Train RMSE: 0.22729133073397062, Validation RMSE: 0.22854270018885148
CPU times: total: 1min 11s
Wall time: 14.9 s
```

The validation set RMSE is `0.22854270018885148`, for XGBoost model

MODEL EVALUATION:



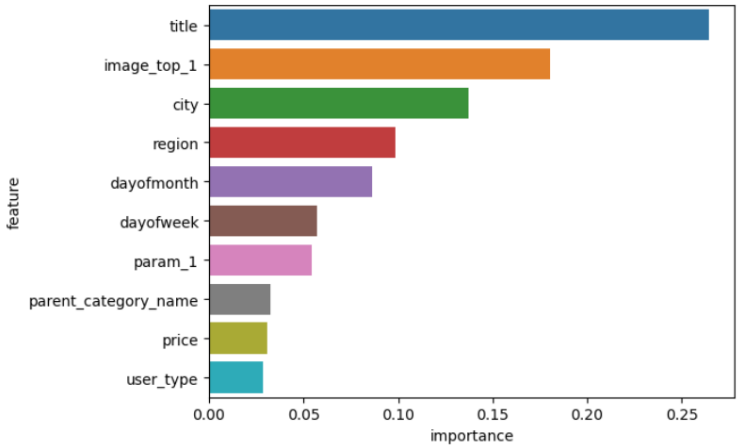
***n_estimator* value of `24` gives a comfortable wide difference for Mean Train and Mean Validation plots.**

- The `n_estimators` hyperparameter controls the number of trees in the random forest model.
- A higher value of `n_estimators` generally results in better performance on the training set,but it can also lead to overfitting.

***max_depth* value of `17` gives a comfortable wide difference for Mean Train and Mean Validation plots.**

- The n_estimators hyperparameter controls the number of trees in the random forest model.
- A higher value of n_estimators generally results in better performance on the training set,but it can also lead to overfitting.
- The best hyperparameters for the model are the ones that result in the lowest RMSE on the validation set. In this case, the best hyperparameters are max_depth=17. This hyperparameter results in an RMSE of 0.225 on the validation set, which is the lowest RMSE for any value of max_depth that was tested.
- Therefore, the max_depth value of 17 was chosen for the model because it resulted in the best performance on the validation set.

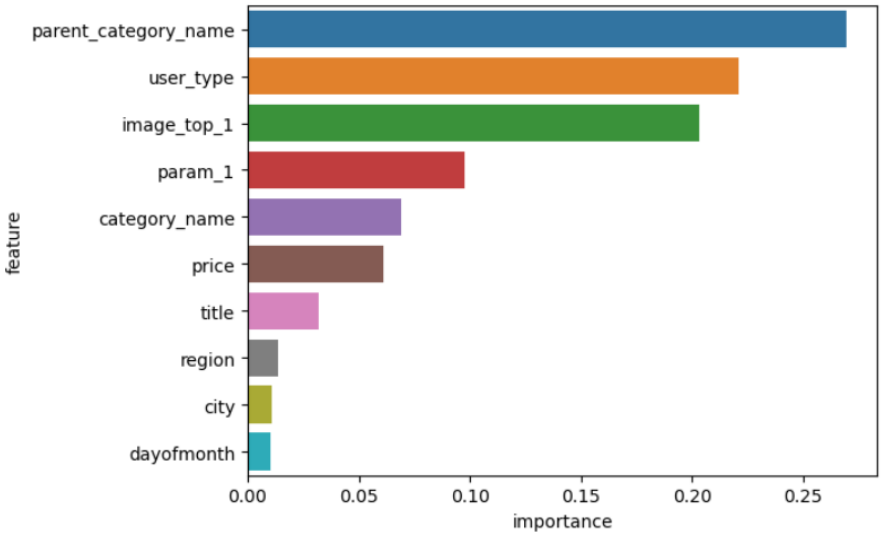
FOR TUNED RANDOM FOREST MODEL



Observations:

`title` is the most important feature determining our target variable
`image_top_1` is the second most important feature followed by `city`, `region`

FOR XGBOOST:



Observations:

`parent_category_name` is the most important feature determining our target variable
`user_type` is the second most important feature followed by `image_top_1`, `param_1`.

Generalization Performance:
XG boost gives a lower RMSE on the validation set, but it doesn't guarantee good performance on entirely new data because 'parent_category_name' cannot be a good indicator to track advertising results.

Even though Random Forest has a higher RMSE value than XGBoost, it may still be the better model for your advertising data set. This is because Random Forest is a more interpretable model, meaning that it is easier to understand how the model makes its predictions.
This is important in advertising, where it is important to understand why certain ads are more likely to be clicked on or converted.

Additionally, the top three important parameters identified by Random Forest are more relevant to advertising than the top three important parameters identified by XGBoost.

For example, the advertisement title and city name are both important factors in determining whether an ad is relevant to a particular user.

Business goals like trying to maximize clicks, conversions, or revenue might influence decision to choose model.

Interpretability Random Forest is a more interpretable model than XGBoost. This is because Random Forest uses a collection of decision trees to make predictions, and each decision tree is relatively easy to understand. In contrast, XGBoost uses a more complex algorithm that is difficult to interpret.

Relevance of important parameters: The top three important parameters identified by Random Forest (advertisement title, image_top_1, and city name) are more relevant to advertising than the top three important parameters identified by XGBoost (parent_category_name, user_type, and image_top_1). The advertisement title and city name are both important factors in determining whether an ad is relevant to a particular user. The image_top_1 parameter is important for both models, as it provides information about the content of the ad.

Conclusion:

Based on the above reasoning, I believe that Random Forest is the better model for your advertising data set. It is more interpretable and has more relevant important parameters.

ACKNOWLEDGEMENT AND REFERENCE:

References

1. Supervised Learning python notebooks by Prof. Junwei Huang

2. XGBoost CV Importance

(https://www.kaggle.com/code/prashant111/xgboost-k-fold-cv-feature-importance/notebook)

3. Test and validation

https://www.kaggle.com/code/kailex/xgb-text2vec-tfidf-0-2237/script)

4. Kaggle data set

https://www.kaggle.com/c/avito-demand-prediction)

5. Microsoft Bing Chat