Airbnb New User Booking

Table of Contents

Problem Statement

The goal is to accurately predict where a new Airbnb user will book their first travel experience, in order to share more personalized content with their community, decrease the average time to first booking, and better forecast demand.

There are 12 possible outcomes of the destination country: 'US', 'FR', 'CA', 'GB', 'ES', 'IT', 'PT', 'NL','DE', 'AU', 'NDF' (no destination found), and 'other'.
'NDF' is different from 'other' because 'other' means there was a booking, but is to a country not included in the list, while 'NDF' means there wasn't a booking.

Datasets

There are several datasets varying between a list of users along with their demographics, web session records, and some summary statistics. It's mentioned on the competition's page that all the users in this dataset are from the USA.

1. train_users.csv - test_users.csv

   Generally include the following features:
   - user id, date_account_created, timestamp_first_active, which can be earlier than date_account_created or date_first_booking because a user can search before signing up
   - Demographic features like gender and age, language (international language preference)
   - Features that describe marketing methods and providers such as signup_method, signup_flow, affiliate_channel, affiliate_provider, first_affiliate_tracked, signup_app
   first_device_type
   first_browser
   - Train dataset includes date_first_booking and **country_destination** which is the target variable we want to predict

2. Sessions.csv

   Describes web sessions log for users who are present in train and test sets, with the following features: user_id, action, action_type, action_detail, device_type and secs_elapsed describing the duration of each session

Other datasets were available such as countries.csv and age_gender_bkts.csv, however, Sessions and Train_user datasets captured the most important information so I decided to proceed with both of them only.

1. Sessions Dataset

Features describe the interactions of users with the Airbnb platform before they make a booking. insights into user behavior, the typical user journey, and potential factors that influence a new user's likelihood of making a booking.

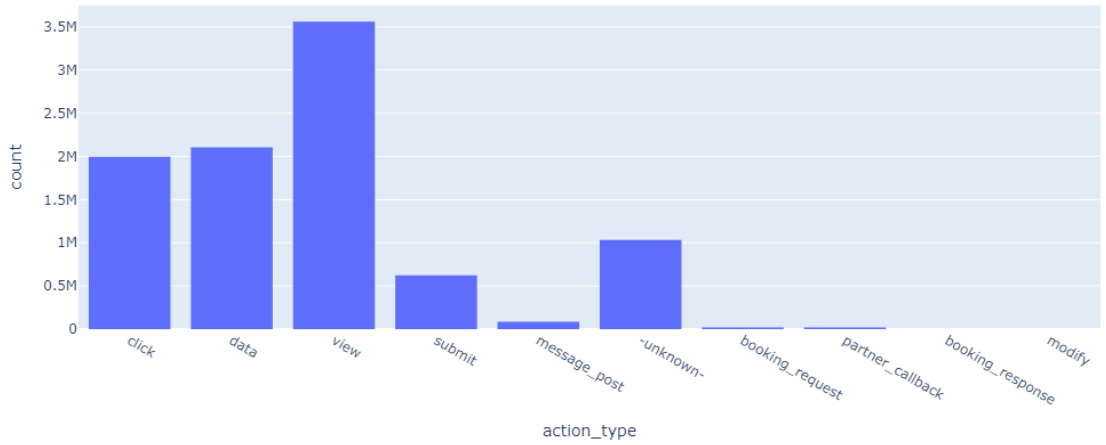
We aim to explore the relationship between user actions and whether they result in a booking.

1.1 EDA & Data Cleaning

A. Action Features

I started with preprocessing action columns (action , action_type, and action_detail) to address any missing values. I noticed that action column had a large number of unique values (359), AKA being a high cardinality feature. It also had the least number of null values and I noticed a relationship between missing action values and their corresponding action_type category, as all of them belonged to the action_type “message_post”. Therefore, I decided to impute the missing action values on the mode of action among instances that belong to the action_type “message_post”.

Then, I started to analyze the action_type column. With a large number of missing values and an “-unknown-” category, I decided to set the “-unknown-” values to null and impute on the mode action_type across each corresponding action. As there definitely is a relationship between action and its corresponding action_type categories. Then, I applied the same techniques on action_detail column.

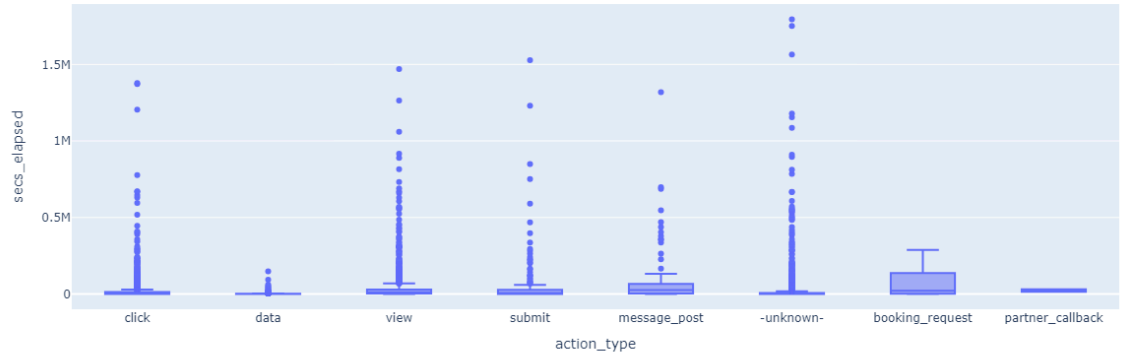


(Figure1: Different categories of action_type value counts)

After ensuring action features had no missing values anymore and before proceeding with feature engineering, rows where essential information, such as **user_id** was missing were removed, as I won’t be able to gain insights out of them.

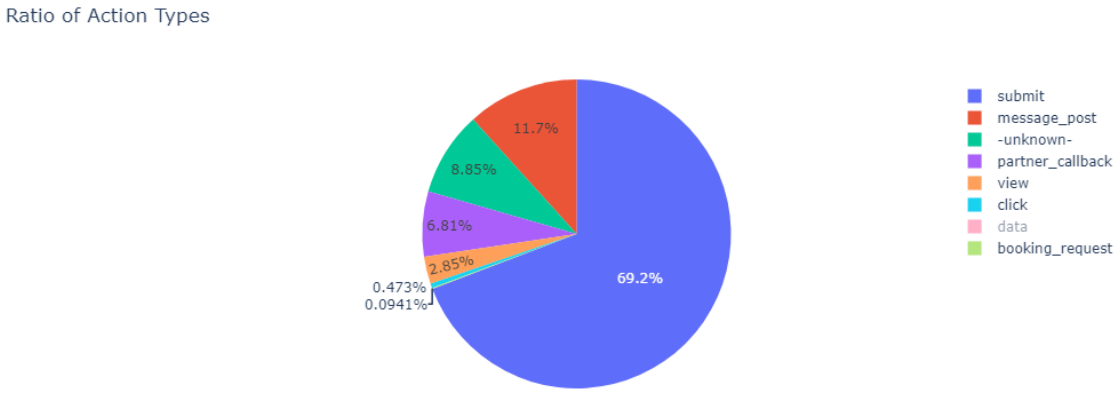
B. Secs Elapsed

Secs_elapsed column also had a large number of missing values. I started to explore the relationship between secs_elapsed and other features using descriptive statistical methods and drawing box plots showing secs_elapsed distribution across different action types.



(Figure2: secs_elapsed distribution across different action types)

I noticed a possibility of a relationship between missing secs_elapsed values and action_type “submit”.



(Figure3: action types of missing secs_elapsed values)

I noticed extreme outliers that are equivalent to up-to 20 days of session duration. I decided to remove outliers based on z-score with threshold =3. Then decided to impute missing values on their mean across different action types.

1.2 Feature Engineering

A. Total Session Duration

One of the first things to check is how long users are staying on the platform

during their sessions. So I created an aggregate of users' secs_elapsed values into a total_session_duration feature.

B. **Most Used Device**
   I created a most_used_device feature for each user and replaced it with the second most common value for users with the most common device "unknown".

C. **Total Actions**
   Total actions feature representing the count of sessions per user.

D. **Frequency of Selected Action Categories**
   I selected a number of categories from each action column, some are common and some are rare to capture the possibility of underlying patterns, and created one-hot-encoding features out of their normalized count.

2. Train Users

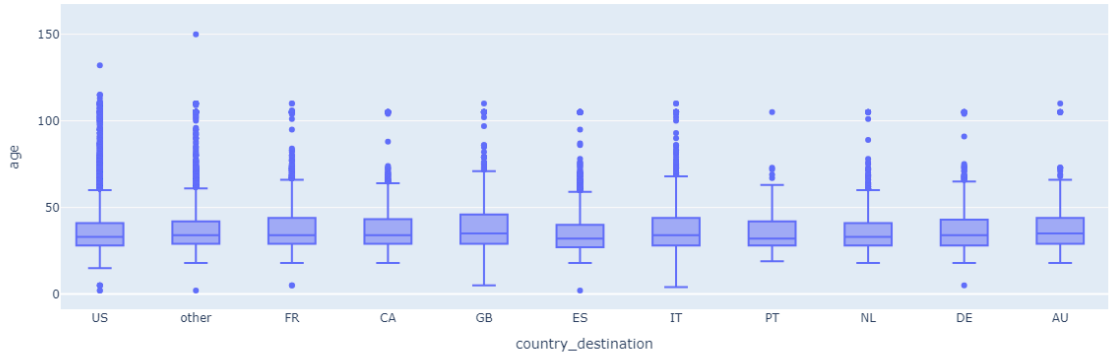
The train_users dataset contains information about users who signed up on Airbnb, their demographics and marketing related information.

We aim to explore patterns in user behavior and demographics that correlate with booking decisions.

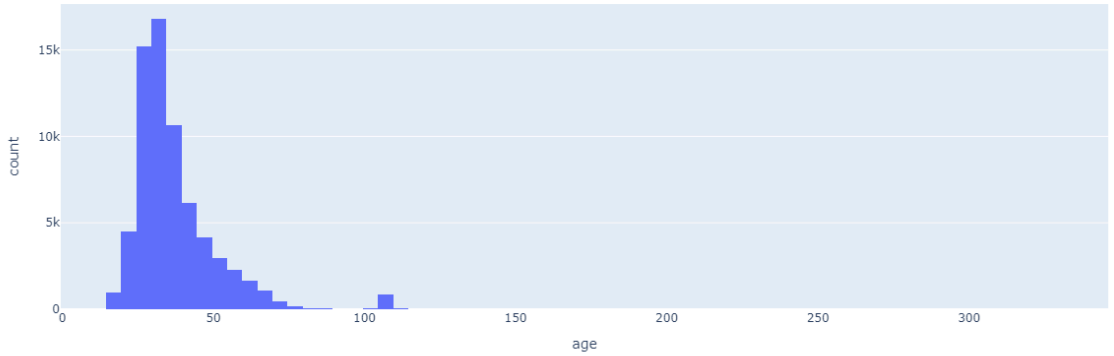
2.1 EDA & Data Cleaning

First, I dropped the date_first_booking feature as it was not present in the test set. I also dropped rows with label "NDF", because it's not a desirable model output, we will aim to predict the first upcoming destination of users who haven't made a booking yet.

Regarding the age column, I started to explore its overall distribution and distribution across different country destinations. I noticed a right skewed distribution and slight differences across different countries that might be useful for the model.

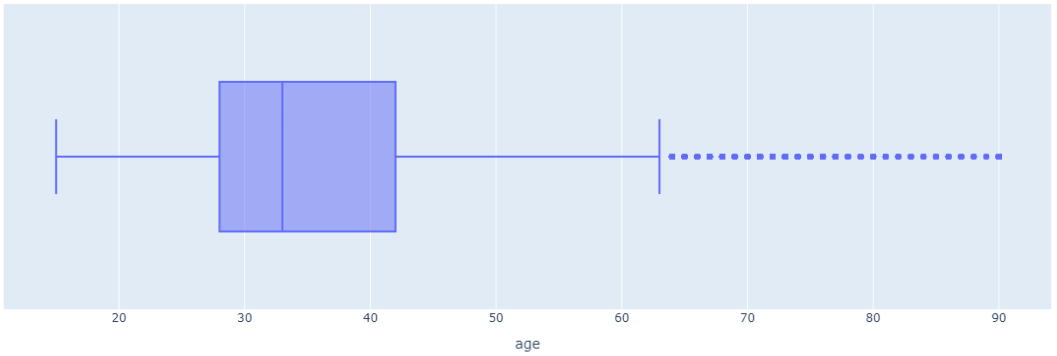


(Figure4: age distribution across different country destinations)



(Figure5: age distribution)

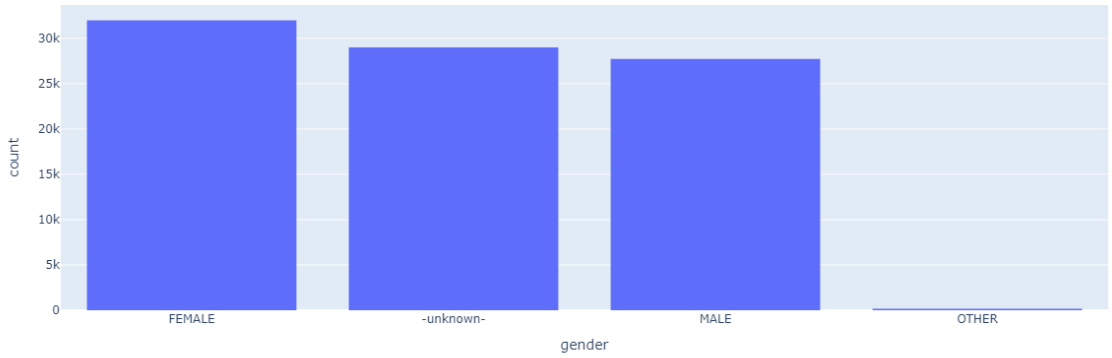
I also noticed outliers of age values above 90 and below 15. Values above 90 were capped to 90 and values below 15 were replaced with null values to be imputed among other missing values. I also noticed values that seem like a year of birth rather than an age (e.g. 1900's and 2000's), I replaced these with the difference between them and year_account_created to produce a valid age value.



(Figure6:age after cleaning outliers)

Finally, I imputed missing values on the global median as it's ideal for the right skewed distribution.

Gender has four categories (male, female, -unknown-, and other). I Decided to leave the -unknown- category as it is to avoid imputation bias, as I cannot reliably impute the missing values based on available data, and it represents a valid category of users who prefer not to specify their gender and might capture an underlying behavioural pattern.



(Figure7: distribution of gender values)

The feature first_affiliate_tracked had a number of missing values, so I decided to replace them with the category "untracked".

Next, categorical features with high cardinality were processed to reduce their valid categories by mapping the categories with value counts below a specified threshold to one "other" category.

Finally, Train dataset was merged with Sessions dataset on user_id, and the same cleaning techniques and merging were applied to Test dataset.

2.2 Feature Engineering

A. Date Features

I parsed the date features date_account_created and timestamp_first_active into their correct formats and created meaningful hour, day, weekday, month, and year features out of them and dropped original columns.

2.3 Encoding

1. Label Encoding

Label encoding was applied to the target column country_destination and class mappings were saved as a pickle file to be used later during inference.

2. **Target Encoding**
   I decided to encode categorical features using target encoding as one hot encoding would lead to very high dimensional data with mostly zero values and would make it harder for the model to learn. Rare grouping was applied to account for possible data leakage when using target encoding. Target encoder mappings were also saved as a pickle file for later use at inference time.

Kindly refer back to the notebooks for any further details regarding EDA & preprocessing.

Modeling

For the classification task, I chose to implement a neural network using PyTorch. In the sections that follow, I will delve into the details of the model architecture, training process, and the rationale behind key design decisions and a summary of findings from other methods tested.

1. Data Preparation

First, I split the data into train and test sets, and further split the train set into train and validation, using scikit-learn's random split with stratify parameter to maintain a unified distribution across different sets.
Then, data was scaled using a Min-Max scaler and scaler object was saved for later use during inference. Finally, data was converted to PyTorch tensors to match the model's expected input shape.

2. Oversampling

We notice a significant class imbalance, as the vast majority of data belongs to US class that we end up with the model only predicting the US class. Oversampling, undersampling, and a hybrid of both techniques were explored to address class imbalance. However, I observed that both undersampling and hybrid techniques negatively impacted the model's performance. These approaches led to a decline in recall, causing the model to lose its ability to accurately predict the majority class while still failing to correctly identify the minority classes.
At last, I decided to go with oversampling minority classes using Imbalanced Learn's random over sampler, increasing the number of instances in minority classes to approach majority class while maintaining their relative distributions.

3. Model Architecture

The model I decided to implement is a simple feedforward neural network that consists of an input layer, two hidden layers with hidden_size = 256, and an output layer. Also, applying Batch Normalization after the first and second hidden layers to improve training stability and convergence, and ReLU Activation to allow the network to learn complex, non-linear patterns in the data, leveraging both batch normalization and non-linear activation to improve performance and robustness.
During training, I monitored the training loss and validation loss to detect signs of overfitting and assess whether the model was generalizing well to unseen data.

4. Loss Function

Initially, I experimented with standard cross-entropy loss; however, it performed poorly due to the class imbalance in the dataset, resulting in high recall and very low precision. I then tried setting custom class weights to address the imbalance, but this approach also performed poorly, leading to a high loss.

To address this, I switched to focal loss and tuned the alpha parameter while monitoring precision and recall. After fine-tuning and experimenting with different values for alpha and gamma parameters, I found that setting alpha = 0.25, and gamma = 2 achieved a reasonable balance.
While the model still demonstrated a preference for the majority class, it also improved predictions for the next most common classes, effectively balancing precision and recall.

5. Evaluation

The final evaluation of the model demonstrated its performance in terms of precision, recall, and F1-score.
Precision achieved was 0.5574.
Recall was 0.6785
F1-score was 0.5752.

Finally, I saved the state dictionary of the model in a .pth file format, which contains the learned weights and parameters. For later retrieval, enabling the model to be reloaded and used for inference without needing to retrain from scratch.

In the future, there are several potential improvements that could further enhance the model's performance. Areas to explore include:

- Experimenting with more advanced sampling techniques to better address the class imbalance
- Incorporating more sophisticated feature engineering, such as experimenting with advanced NLP techniques on action features in sessions dataset
- Exploring alternative model architectures, or ensemble methods such as balanced random forest.
- Tuning hyperparameters further.

Deployment

For deployment, I used FastAPI along with an HTML template to create a simple interactive interface, ensuring smooth deployment, and allowing for easy inference.

To ensure a clean and efficient structure, I implemented a modular design, encapsulating the entire process within a single inference function.
I created a preprocessing pipeline class that prepares the input data for inference by utilizing the saved target encoder, scaler, and transforming the data into tensors.

The prediction function utilizes the saved model and mappings, selects a random sample from the test set, applies the preprocessing pipeline, and returns the predicted label and sample row printed by tabulate.

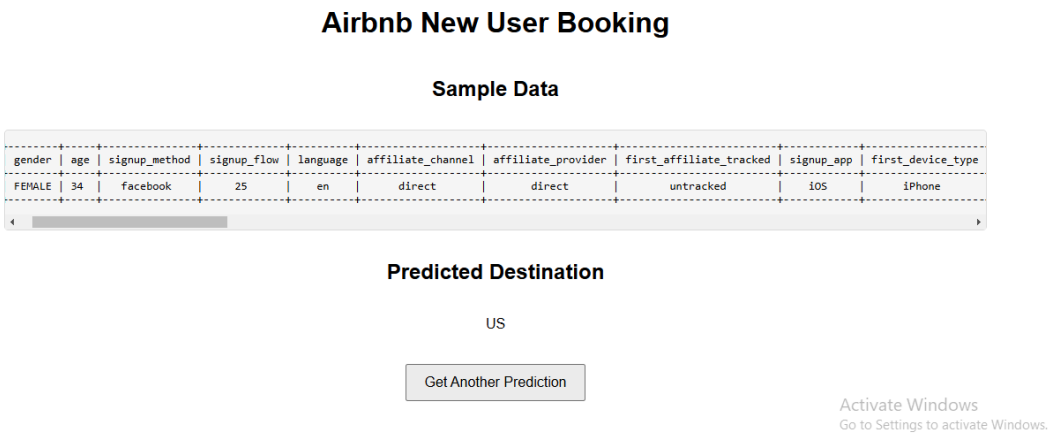
FastAPI runs this prediction function in the main application, displaying both the sample row and its corresponding prediction. Additionally, when the "Get Another Prediction" button is pressed, the function is executed again, providing a new random sample and its prediction each time, ensuring a dynamic and interactive user experience.

To streamline the deployment process, I created a Docker image that includes all the necessary dependencies and configurations. The image contains a requirements.txt file listing all the Python packages required for the application, and a .dockerignore file to exclude unnecessary files from the build. Once the Docker image is built and run, it automatically starts the FastAPI server, enabling the API to be accessed seamlessly in any environment. This approach ensures consistent and isolated deployment, simplifying both development and production workflows.
However, I encountered an issue when trying to push the image to Docker Hub, as the image was too large. Despite optimizing the image size by using base image python-slim and including only the test set csv file, it was still unable to be pushed successfully.

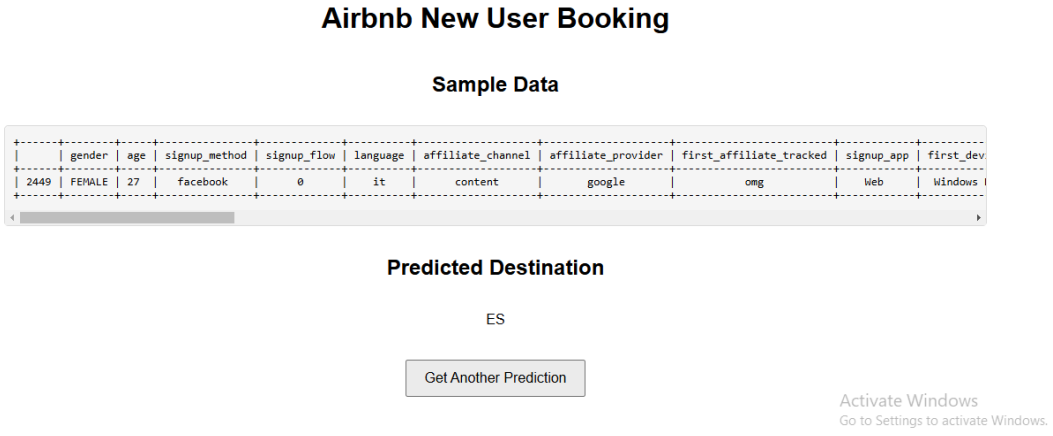
Usage Instructions

- Open the root folder of the project in Visual Studio Code.
- Open the terminal in VS Code and run the following command to start the FastAPI server:
  python -m uvicorn src.main:app --reload
- Once the server is running, navigate to the provided API link in your web browser.

Sample Runs



(Figure8: sample run 1)



(Figure9: sample run 2)