Airbnb Data Analysis and predicting the destination country for a new user

4th Sept 2020

OVERVIEW:

Airbnb is a renowned American vacation rental online marketplace which offers arrangements for lodging, primarily homestays or tourism experiences. It has a wide range of options to choose from 34000+ cities around 190 countries.

BUSINESS REQUEST/GOALS:

- Predicting where the new user will book their first travel experience has a great value.

- Such insights or information can help Airbnb share more personalised content with the community, decrease the average time for first booking, understand how a user engages with the service, understand what factors would encourage them to engage more deeply and better forecast demand and many more.

WHO CARES ABOUT THIS ?

- Knowing where a new user will book their first travel experience is of great value to Airbnb.
- As a new user getting a personalised treatment is of great value.

DATA COLLECTION AND WRANGLING:

- The data is collected from Kaggle. Ref Data
- Data mainly comprises demographics information, web session records of the user and some summary statistics.
- Most of the data is clean.
- 'US', 'FR', 'CA', 'GB', 'ES', 'IT', 'PT', 'NL','DE', 'AU', 'NDF' are possible destination countries(classes of target variable)
- Timings are transformed to datetime formats.
- Missing values are transformed to np.NAN.
- Some outliers were observed, like in user age which were replaced by mean age.

EXPLORATORY DATA ANALYSIS SUMMARY

Ref script: EDA

- We majorly have the following datasets.
  - Train, sessions and countries.
- Quick glance of data

Train data

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213451 entries, 0 to 213450
Data columns (total 16 columns):
id                           213451 non-null object
date_account_created         213451 non-null object
timestamp_first_active       213451 non-null int64
date_first_booking           88908 non-null object
gender                       117763 non-null object
age                          125461 non-null float64
signup_method                213451 non-null object
signup_flow                  213451 non-null int64
language                     213451 non-null object
affiliate_channel            213451 non-null object
affiliate_provider           213451 non-null object
first_affiliate_tracked      207386 non-null object
signup_app                   213451 non-null object
first_device_type            213451 non-null object
first_browser                186185 non-null object
country_destination          213451 non-null object
dtypes: float64(1), int64(2), object(13)
memory usage: 26.1+ MB
```

Session data

```
df_sessions.info()
```

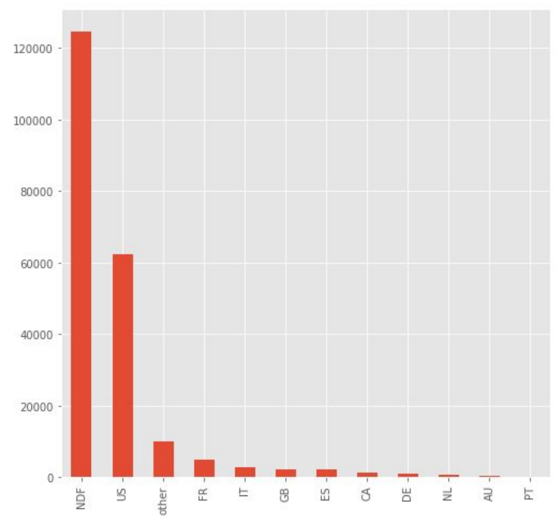
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10567737 entries, 0 to 10567736
Data columns (total 6 columns):
user_id          object
action           object
action_type      object
action_detail    object
device_type      object
secs_elapsed     float64
dtypes: float64(1), object(5)
memory usage: 483.8+ MB
```

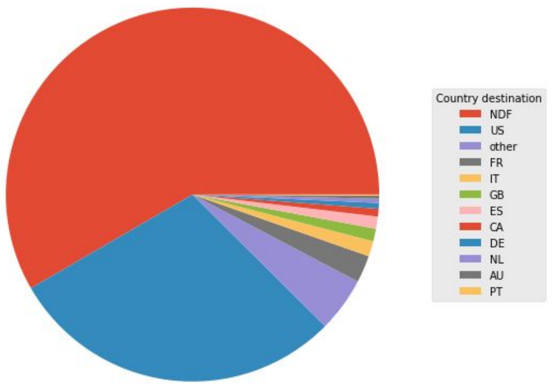
Countries data

```
df_countries.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 7 columns):
country_destination              10 non-null object
lat_destination                  10 non-null float64
lng_destination                  10 non-null float64
distance_km                      10 non-null float64
destination_km2                  10 non-null int64
destination_language             10 non-null object
language_levenshtein_distance    10 non-null float64
dtypes: float64(4), int64(1), object(2)
memory usage: 688.0+ bytes
```

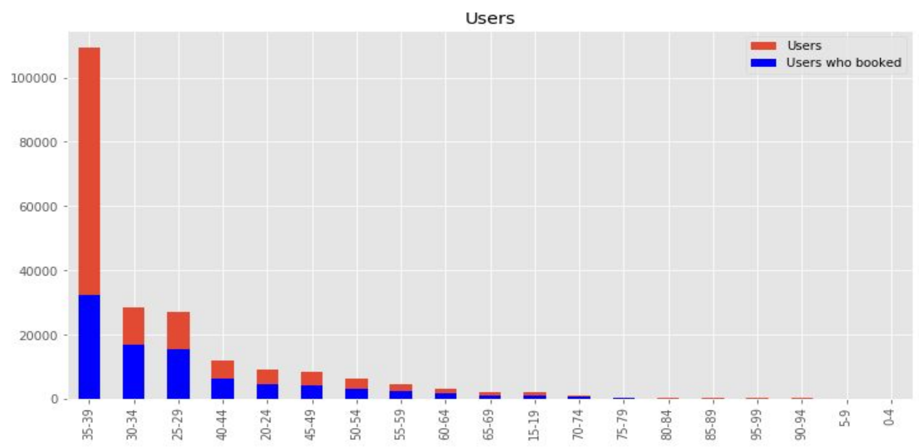
- ***Distribution of destination countries:***





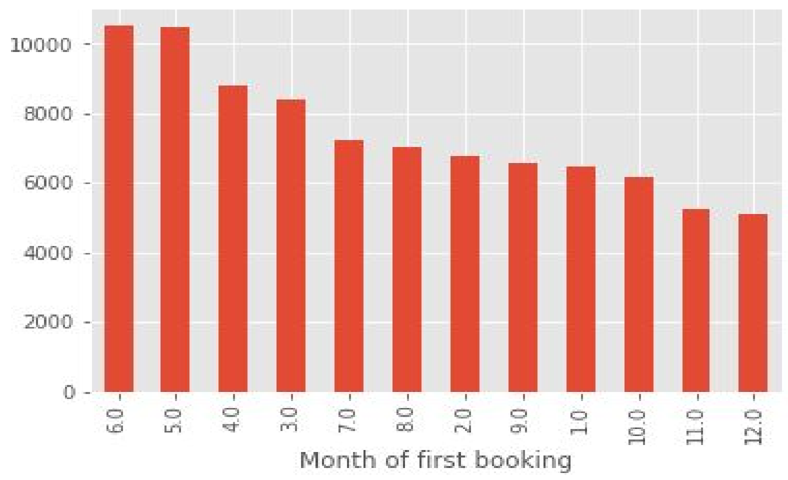
- Most of the users land up doing no bookings(NDF).
- US is the destination country for most of the users, could be because all user data are from people of US which also implies that most users do bookings within the country.
- US and NDF are the most favourable classes making it an imbalance set.

- ***Age group with max users and users who booked:***



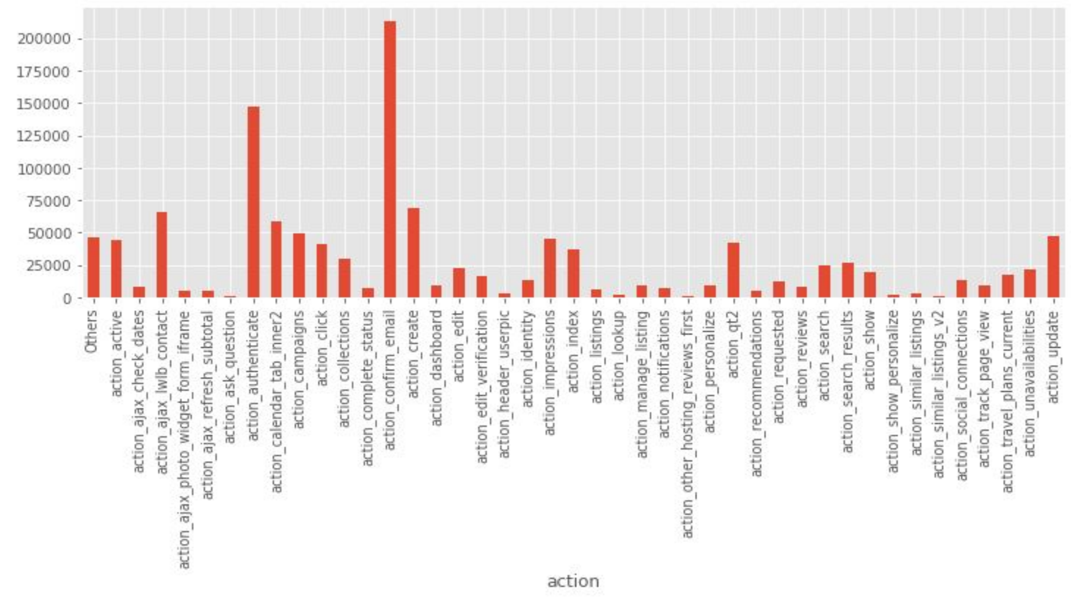
- Most users are from age bucket 35-39.
- There is a lot of variance in count as age bucket varies.
- Age bucket 35-39 has relatively low booking to not booking ratio.
- Users of Age bucket 30-34 and 25-29 has relatively higher booking to Non booking ratio.

- ***Monthly count of first bookings:***



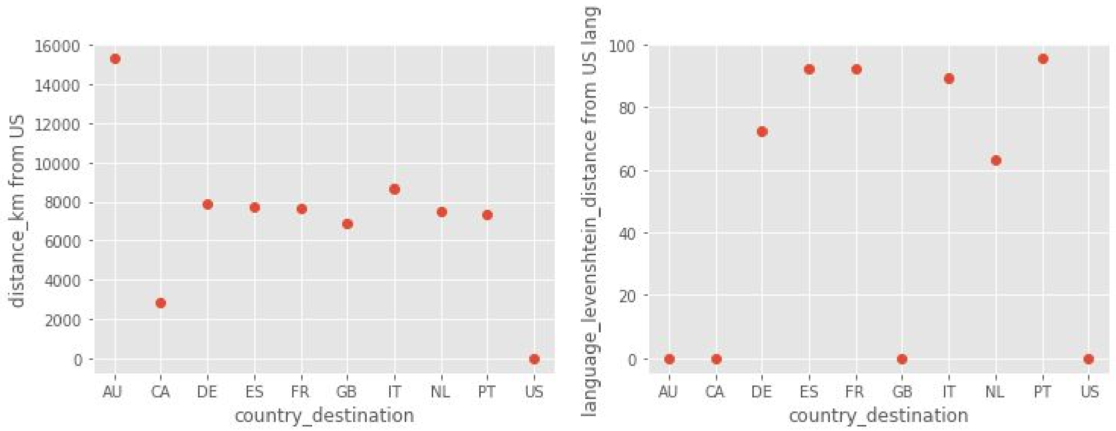
- Mid year(ie May, June) seems to have relatively higher first time bookings.
- Year end has relatively low first bookings.

- ***User session action having highest time elapsed:***



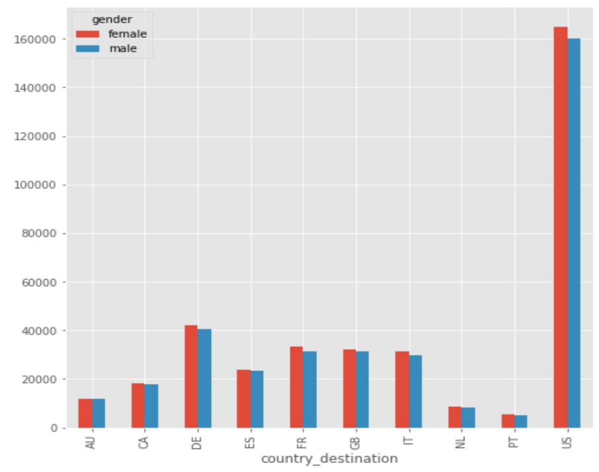
- Action 'confirm_email' and 'authenticate' has the highest mean secsElapsed in a user session.

- ***Language difference and km distance for a US user:***



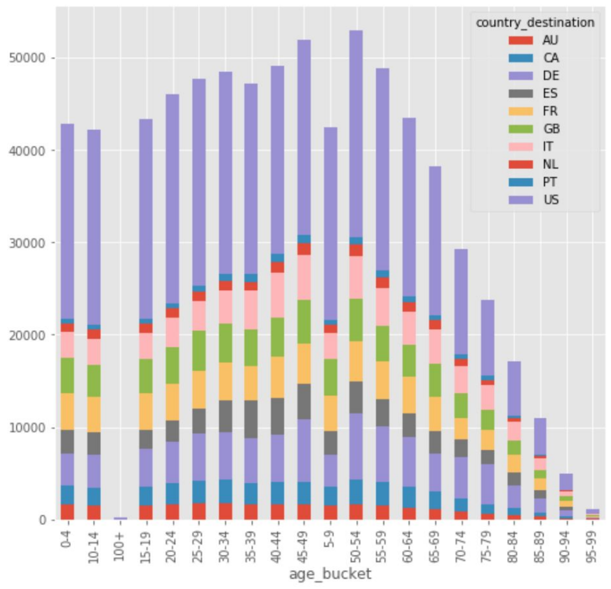
  - From plot 1, AU looks farest from the US in km distance.ES, FR, PT have the highest language_levenshtein_distance i.e these languages have the highest difference score from US english.

- ***Demographic information of cities***



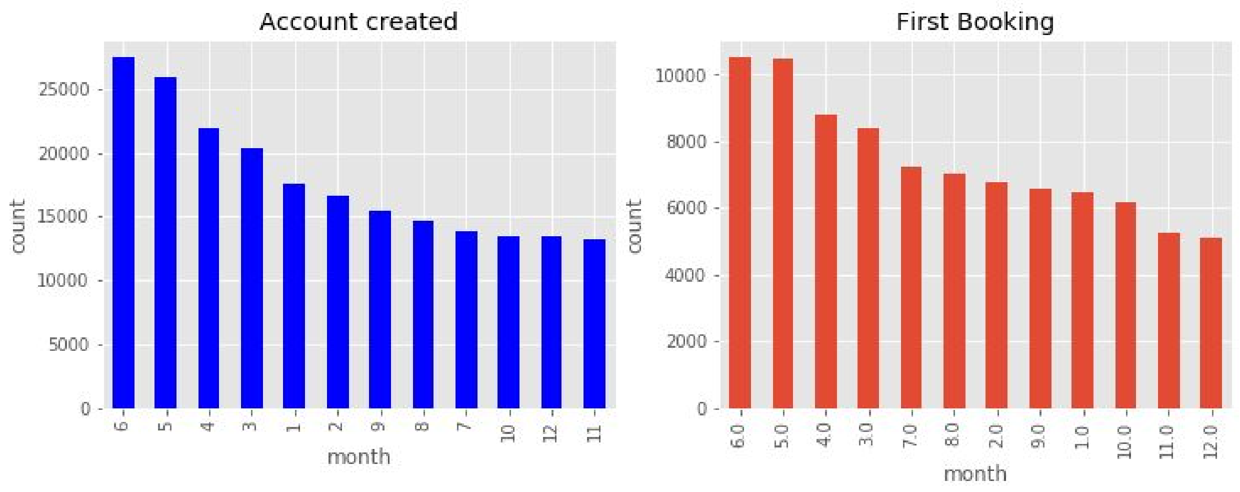
- The US seems to have the highest population, also female population is higher compared to male for all destination countries.

- ***Age bucket wise distribution of destination country***



- There is no significant variation in the segments with age buckets.

- ***Highest first bookings and accounts created***



- From the plot Shapes of Account created and first bookings over months are almost same.
- First half of the year has the max accounts created.
- June and May are months of highest first bookings.

DATA PREPROCESSING AND FEATURE ENGINEERING:

As a part of data preprocessing and feature engineering following steps were performed.

- Datetime format transformations.
- Extracting important features from datetime like month were added as separate features.
- Less frequent categories considering a threshold were transformed to single categories like 'Others'.
- Grouping and aggregations.
- Dropping redundant columns.
- Joining eg. Session data was joined with train data.
- Age to Age_group transformation.

- Adding features like user language, age group preferences from the demographics information of the destination countries.

Now that we have a cleaner and well engineered data set we can move to predictive modelling of the rating ,

PREDICTIVE MODELING FOR CLASSIFICATION OF DESTINATION COUNTRIES:

Before modeling, let's perform feature selection and remove redundancy.

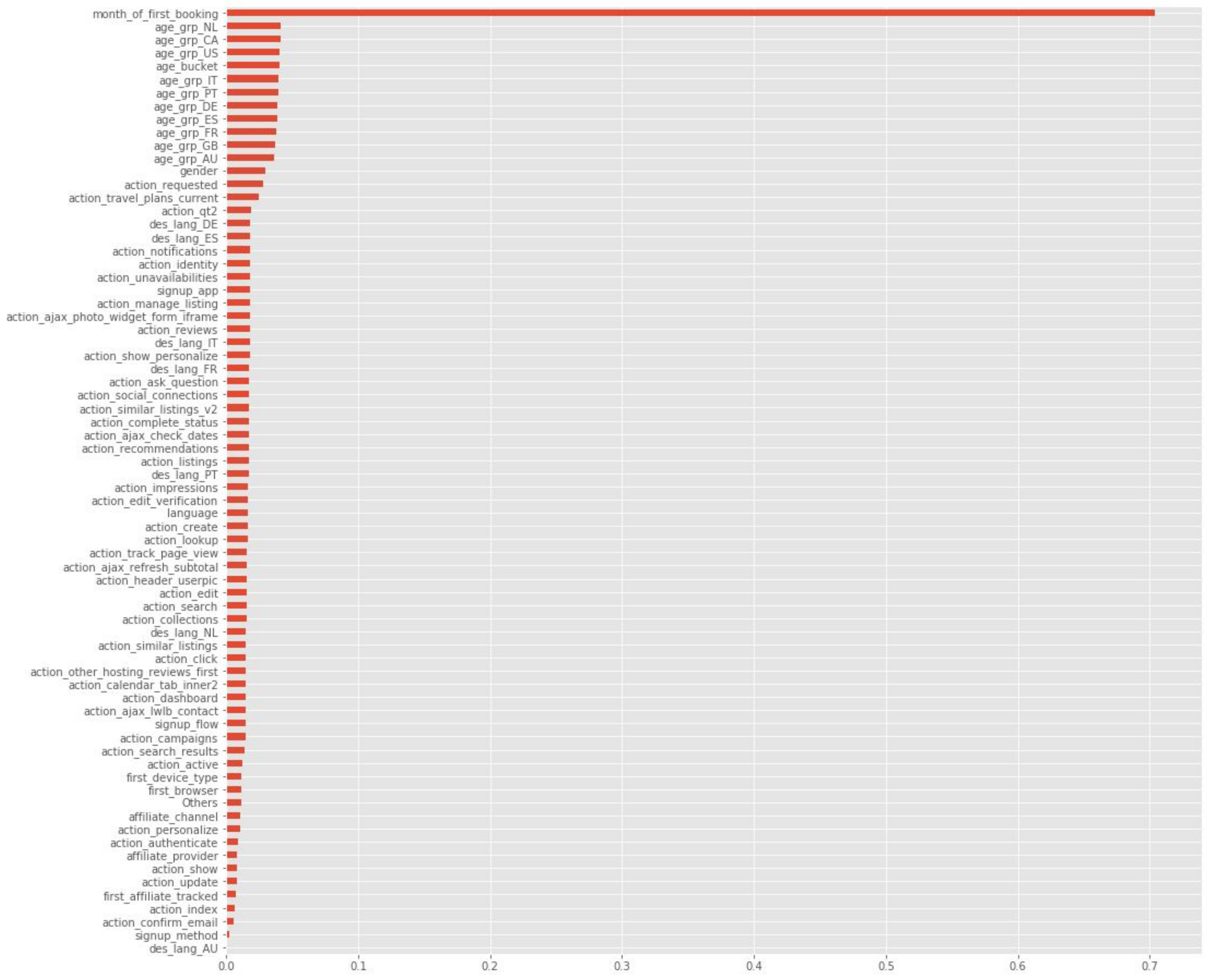
- ***Features Selection based on Mutual information***(Ref script modeling)

  Mutual Information is the measure of dependence between two discrete random variables. In this context, it is a measure of dependence between two features, higher the number more is the dependence.

  Before getting the mutual info, we remove quasi constants and duplicate features.

```
from sklearn.feature_selection import VarianceThreshold, mutual_info_classif
from sklearn.feature_selection import SelectKBest, SelectPercentile
```

  Sklearn.feature_selection module provides mutual_info_classif for getting mutual info.



- From the plot we see, the month_of_first_booking has the highest mutual information value for classification.This could be because for major NDF i.e for users not doing bookings have month_of_first_booking as NaN as seen in our analysis earlier, which makes it high dependent variable and thus high mutual information.
- Next ,we select the best features keeping a threshold of 40 percentile.

Now that we have a reduced optimal feature set we can start modeling the data and attempt the classification problem.

***Note*:**

- From our the previous analysis we have seen that the dataset is imbalanced with class 7(NDF) and 10(US).
- We are not handling this imbalance as we want our model to learn the bias and try to classify the minor classes taking this into account.

To start with lets try some decision trees models,

Random Forest Classifier:

```
clf = RandomForestClassifier(n_estimators=100,random_state=0,n_jobs=1)
clf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                       oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
y_pred = clf.predict(X_test)
```

```
print("accuracy_score",accuracy_score(y_test,y_pred))
```

```
accuracy_score 0.8701043163220689
```

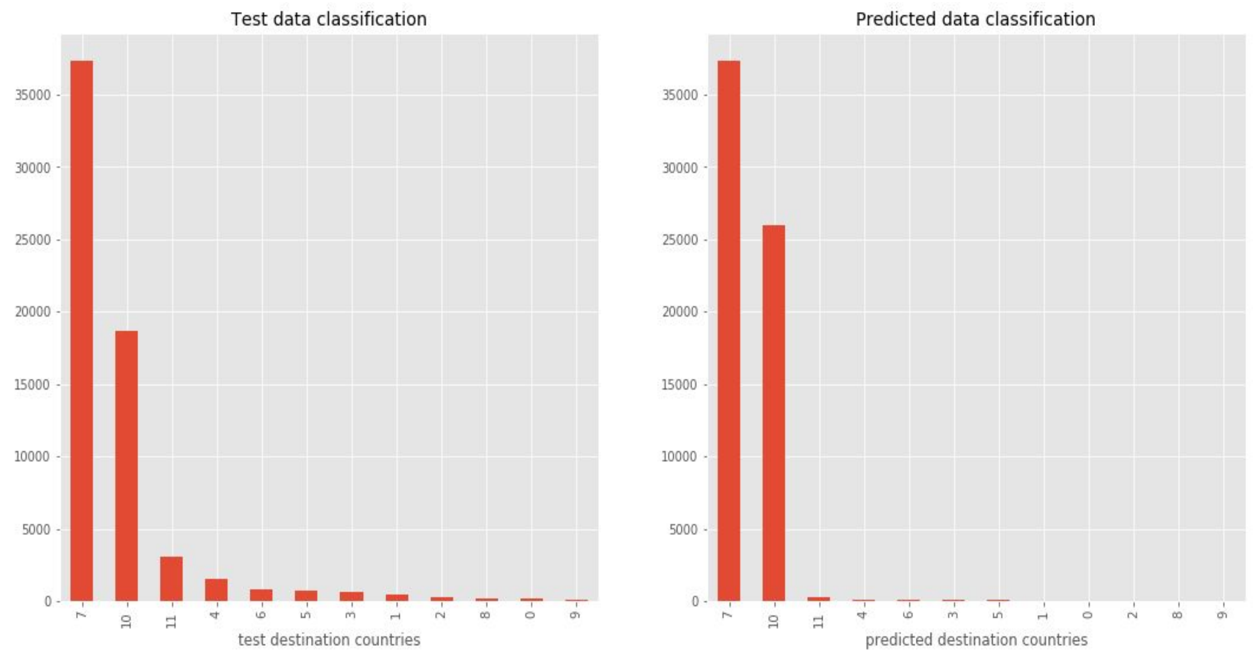
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       162
           1       0.00      0.00      0.00       428
           2       0.00      0.00      0.00       318
           3       0.02      0.00      0.00       675
           4       0.11      0.01      0.01      1507
           5       0.04      0.00      0.01       697
           6       0.04      0.00      0.01       851
           7       1.00      1.00      1.00     37363
           8       0.00      0.00      0.00       229
           9       0.00      0.00      0.00        65
          10       0.70      0.98      0.82     18713
          11       0.14      0.01      0.03      3028

    accuracy                           0.87     64036
   macro avg       0.17      0.17      0.16     64036
weighted avg       0.80      0.87      0.82     64036
```

- Random classifier gives a test accuracy score of 0.87.
- Following plot shows comparison of real test and predictions and with reference to the classification report we see that, class 7(NDF) is very well predicted with precision of 1.Class 10(US) has been predicted with a precision of 0.7,this class is a bit overclassified.
- The model is capable of predicting some minor classes like class 11, 4, 6, 3, 5 with relatively lower precision.



Further, hyperparameter tuning using Randomised CV gave a slightly higher test accuracy of **0.875.**

Neural Networks:

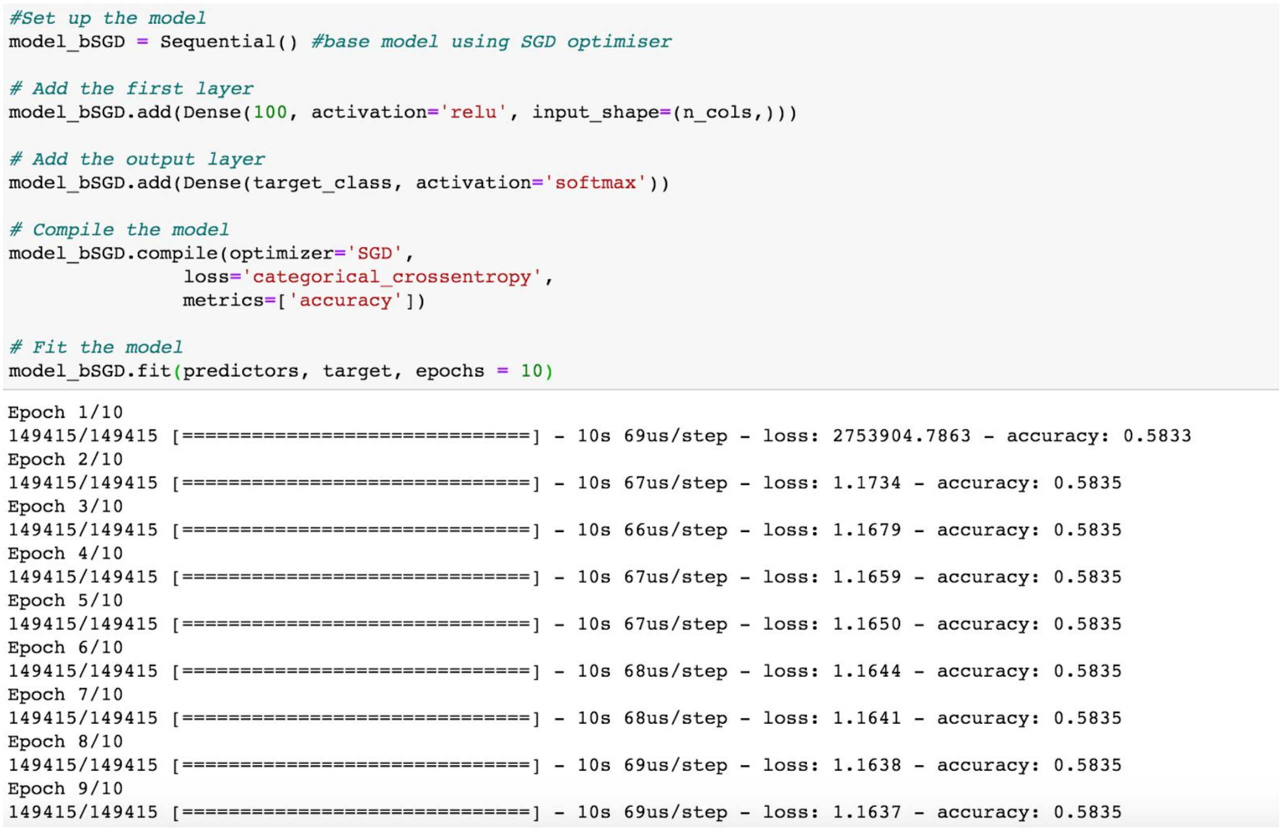
- Neurals networks take into account interactions between features.

- The hidden layers between the input and output layers capture the interactions or in other words the nodes in the hidden layers represent the aggregation of information at each node and adds to models capability to capture interactions.
- More nodes, more interactions we capture.
- We would be using Keras interface to the Tensorflow Deep learning library.

  Workflow steps for Keras:

  - Specify Architecture
  - Compile
  - Fit
  - Predict

Further let's start with defining a baseline model,



```
#Set up the model
model_bSGD = Sequential() #base model using SGD optimiser

# Add the first layer
model_bSGD.add(Dense(100, activation='relu', input_shape=(n_cols,)))

# Add the output layer
model_bSGD.add(Dense(target_class, activation='softmax'))

# Compile the model
model_bSGD.compile(optimizer='SGD',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Fit the model
model_bSGD.fit(predictors, target, epochs = 10)
```

```
Epoch 1/10
149415/149415 [==============================] - 10s 69us/step - loss: 2753904.7863 - accuracy: 0.5833
Epoch 2/10
149415/149415 [==============================] - 10s 67us/step - loss: 1.1734 - accuracy: 0.5835
Epoch 3/10
149415/149415 [==============================] - 10s 66us/step - loss: 1.1679 - accuracy: 0.5835
Epoch 4/10
149415/149415 [==============================] - 10s 67us/step - loss: 1.1659 - accuracy: 0.5835
Epoch 5/10
149415/149415 [==============================] - 10s 67us/step - loss: 1.1650 - accuracy: 0.5835
Epoch 6/10
149415/149415 [==============================] - 10s 68us/step - loss: 1.1644 - accuracy: 0.5835
Epoch 7/10
149415/149415 [==============================] - 10s 68us/step - loss: 1.1641 - accuracy: 0.5835
Epoch 8/10
149415/149415 [==============================] - 10s 69us/step - loss: 1.1638 - accuracy: 0.5835
Epoch 9/10
149415/149415 [==============================] - 10s 69us/step - loss: 1.1637 - accuracy: 0.5835
```

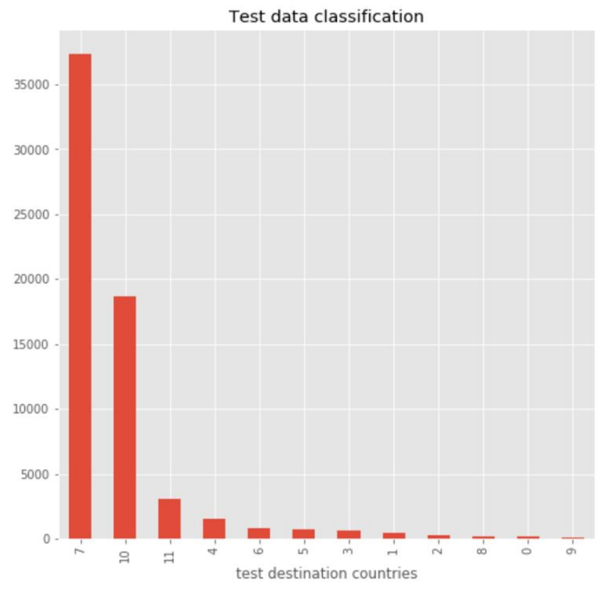
- From above baseline model setting single layer with nodes = 100

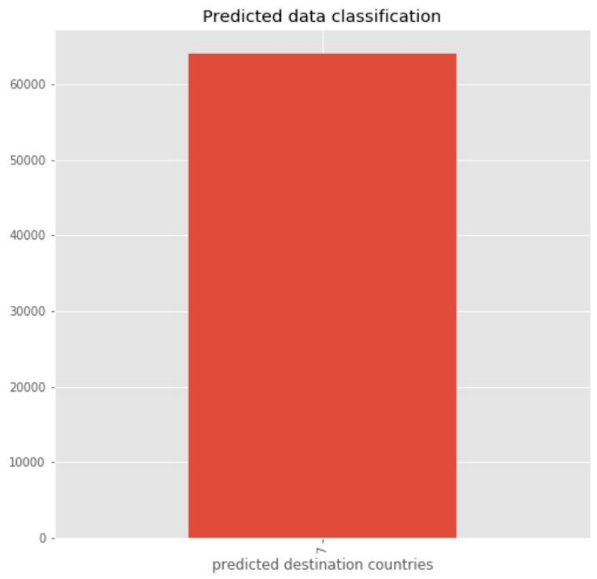
Epoch = 10

Optimiser = **‘SGD’(Stochastic Gradient Descent)**

Activation = **‘relu’**

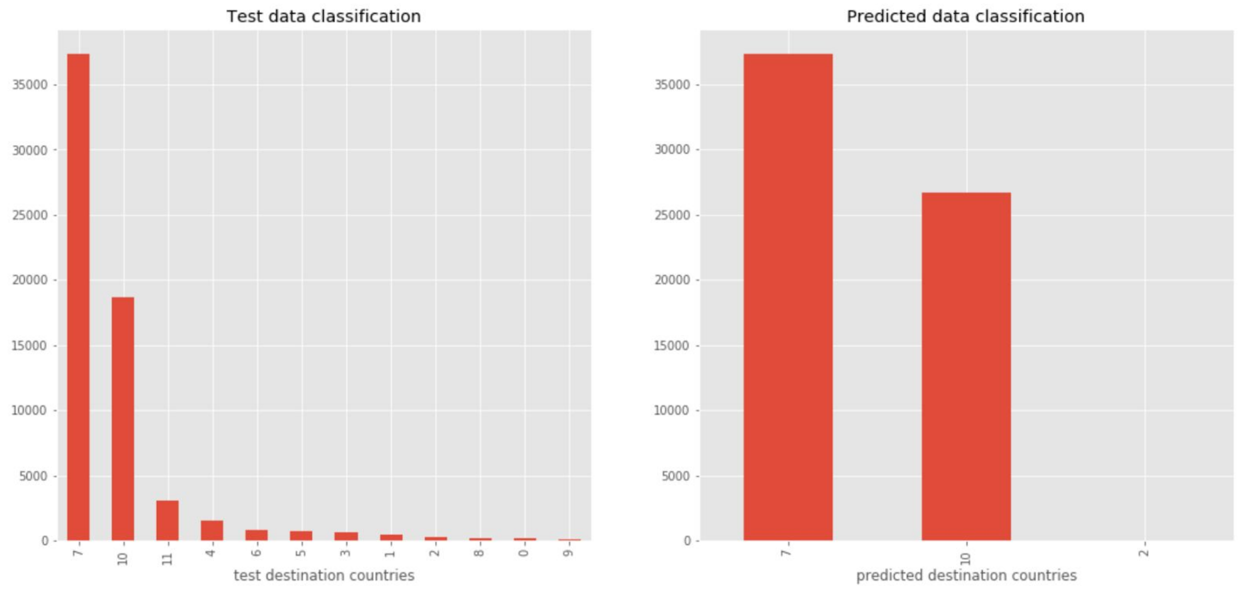
- Results test accuracy of **0.583**



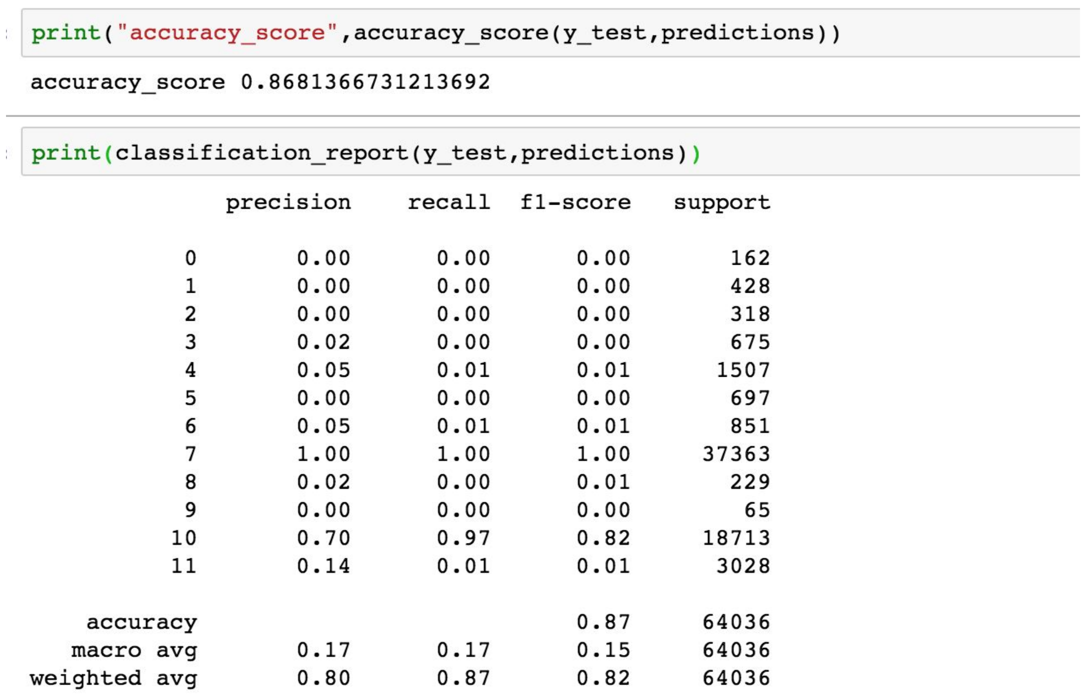


From the above plot comparing the real test data and predictions shows that all classes are labelled as class 7 and the model is highly baised by its weight.

- Further using optimiser ‘**Adam**’ gave slightly better results with test accuracy of **0.875** and identification of some rare classes.



- Further Parameter optimisation on Learning rate, epoch, nodes, layers, EarlyStopping,Model Checkpoint and Cross validation gave a validation accuracy of **0.868** along with some rare class identification.



```
print("accuracy_score",accuracy_score(y_test,predictions))
```

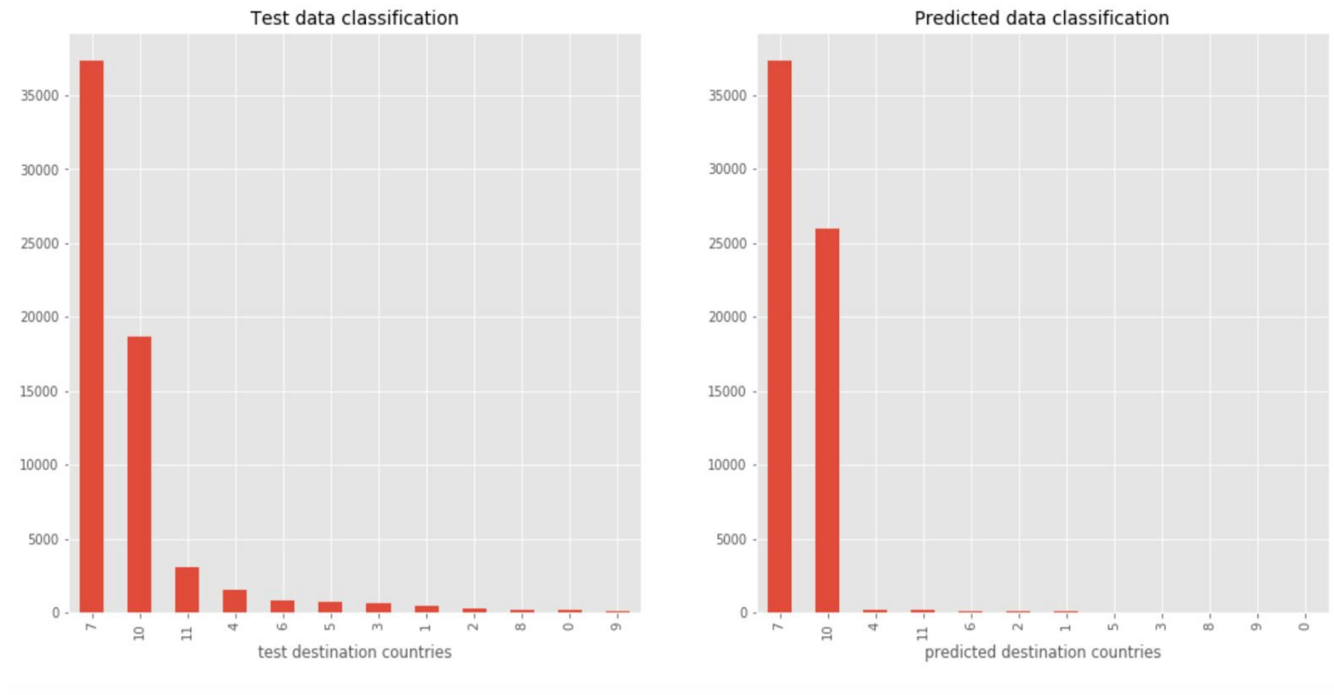
```
accuracy_score 0.8681366731213692
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       162
           1       0.00      0.00      0.00       428
           2       0.00      0.00      0.00       318
           3       0.02      0.00      0.00       675
           4       0.05      0.01      0.01      1507
           5       0.00      0.00      0.00       697
           6       0.05      0.01      0.01       851
           7       1.00      1.00      1.00     37363
           8       0.02      0.00      0.01       229
           9       0.00      0.00      0.00        65
          10       0.70      0.97      0.82     18713
          11       0.14      0.01      0.01      3028

    accuracy                           0.87     64036
   macro avg       0.17      0.17      0.15     64036
weighted avg       0.80      0.87      0.82     64036
```



Insights from model analysis:

- As the dataset was imbalanced, and because we wanted this bias in our predictions we did not handle this imbalance and let our model learn it. As a result, the model at times overclassifies these major classes.
- Month_of_first_booking feature has a good mutual information value.
- NDF class(Class 7) is very well predicted, with a good precision, and could be of a good value to primarily classify if a user will book or not book and treat him accordingly.
- RandomForest classifier gave a good validation accuracy of **0.875**, identifying even some minor classes.
- Neural Networks classifier gave a validation accuracy **0.868**, and could identify some of the rare classes.
- Both these models have almost equal performance and are capable of predicting even the minor classes to some extent.This information is of a great value in understanding the user and give a more personalised treatment.

Future Development: As a part of future development and refinements,

- Would like to include more features and more intutuive visualizations using seaborn.
- Hypothesis test on some key insights we derived from visualisations.
- Try out more Hyperparameter tuning iterations using Gridsearch.
- Try out a different architecture like Residual Neural Network.