AirBnB Classification Project 3

Aman Gangwani, Ben Giles, Owen FinkBeiner

our kaggle score

In [120]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from datetime import date, time, datetime
from xgboost.sklearn import XGBClassifier
import seaborn as sns
```

In [ ]:

```

```

Sections ::

Introduction to the problem :

Recently there's been a surge in the peer-to-peer economy that AirBnB encapsulates. AirBnB is a business that allows homeowners to rent out their homes or properties for cheap to travelers who are also looking for equally cheap accomodations. Rather than booking costly hotels, travelers now have the ability to support local economies and enjoy further immersion into the host country's culture.

Throughout this project, we will be looking at data provided by AirBnB to predict where a new user will book their first travel experience. By doing so, AirBnB hopes to share personalized content with those users, decrease the wait time until the first booking, and better anticipate the demand of the consumers.

To complete this project, we took a features from 2 of the 3-4 datasets provided, the major train_users features that contain all of the training labels and data, and the sessions data which provides information on how people navigated the website. After processing all our data, we used an xgboost classifier to predict our values. We tried a few different classifiers but found that an XGBoost gave us our largest base value that we could improve upon.

Dataset exploration and data creation

Here we're importing our dataset, we use the sessions data later on to get the information of how many seconds they were on certain parts of the website. Our approach was to essentially get as many features as possible and then perform feature selection afterwards to see how the model changes.

In [118]:

```
train_users = pd.read_csv("proj3_data/train_users.csv",parse_dates=['timestamp_first_acti
ve','date_account_created','date_first_booking'])
test_users = pd.read_csv("proj3_data/test_users.csv",parse_dates=['timestamp_first_active
','date_account_created','date_first_booking'])
sessions = pd.read_csv("proj3_data/sessions.csv")
```

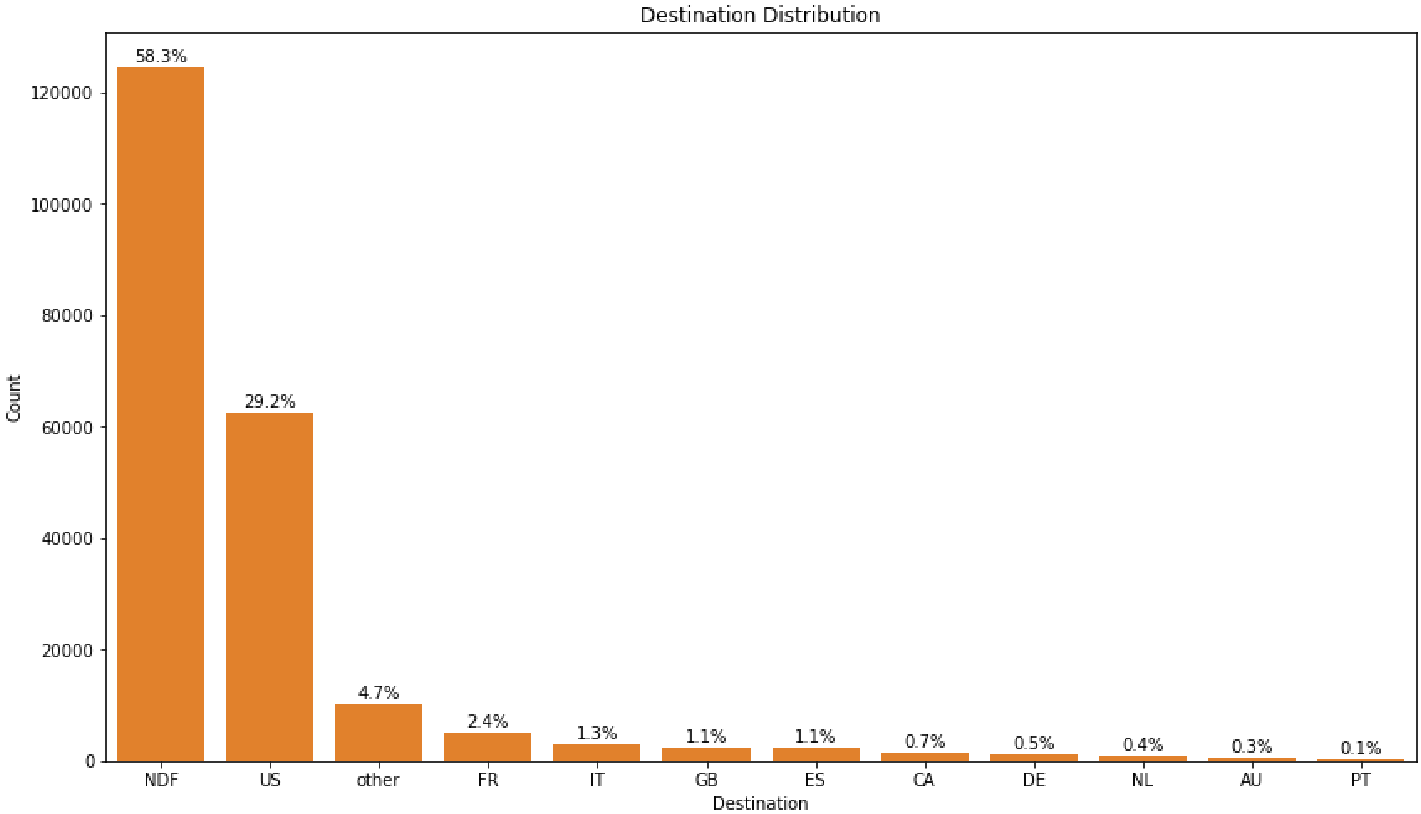
In [121]:

```
# Exploration before we move forward
# Plot ideas came from https://www.kaggle.com/krutarthhd/airbnb-eda-and-xgboost so we cou
```

```
ld view the distribution
# after we make certain changes.
#Finding Destination Distribution.
df_train = train_users


plt.figure(figsize=(14,8))
order1 = df_train['country_destination'].value_counts().index
sns.countplot(data = df_train, x = 'country_destination', order = order1, color = sns.co
lor_palette()[1])
plt.xlabel('Destination')
plt.ylabel('Count')
plt.title('Destination Distribution')
order2 = df_train['country_destination'].value_counts()

for i in range(order2.shape[0]):
    count = order2[i]
    strt='{:0.1f}%'.format(100*count / df_train.shape[0])
    plt.text(i,count+1000,strt,ha='center')
```



In [107]:

```
Y = train_users['country_destination'] # these are our labels
train_users = train_users.drop('country_destination', axis =1)
```

In [123]:

```
#data is pretty badly skewed.
Y.value_counts()
```

Out[123]:

```
NDF      124543
US        62376
other     10094
FR         5023
IT         2835
GB         2324
ES         2249
CA         1428
DE         1061
NL          762
AU          539
PT          217
Name: country_destination, dtype: int64
```

```
Name: country_destination, dtype: int64
```

In [124]:

```
train_users.columns # exploratory
```

Out[124]:

```
Index(['id', 'date_account_created', 'timestamp_first_active',
       'date_first_booking', 'gender', 'age', 'signup_method', 'signup_flow',
       'language', 'affiliate_channel', 'affiliate_provider',
       'first_affiliate_tracked', 'signup_app', 'first_device_type',
       'first_browser', 'country_destination'],
      dtype='object')
```

Columns:

id: user id

date_account_created: the date of account creation

timestamp_first_active: timestamp of the first activity, note that it can be earlier than date_account_created or date_first_booking because a user can search before signing up

date_first_booking: date of first booking

gender

age

signup_method

signup_flow: the page a user came to signup up from

language: international language preference

affiliate_channel: what kind of paid marketing

affiliate_provider: where the marketing is e.g. google, craigslist, other

first_affiliate_tracked: whats the first marketing the user interacted with before the signing up

signup_app

first_device_type

first_browser

country_destination: this is the target variable you are to predict

In [125]:

```
sessions.head()
```

Out[125]:

	user_id	action	action_type	action_detail	device_type	secs_elapsed
0	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	319.0
1	d1mm9tcy42	search_results	click	view_search_results	Windows Desktop	67753.0
2	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	301.0
3	d1mm9tcy42	search_results	click	view_search_results	Windows Desktop	22141.0
4	d1mm9tcy42	lookup	NaN	NaN	Windows Desktop	435.0

sessions.csv - web sessions log for users

user_id: to be joined with the column 'id' in users table

action - lookup, search results, etc. large number of them

action_type - clicks, scrolls, etc.

action_detail - view search results, etc.

device_type - windows, mac, etc.

secs_elapsed - this is what we want we're basically aggregating the seconds for each person.

Now we're just going to be adding the train and test sets for data processing so we don't need to do these steps for both of them separately

In [126]:

```
df_all = pd.concat((train_users, test_users), axis = 0, ignore_index= True)
```

Create some variables that involve splitting up our DAC and TFA into the year month and day of week. No processing needed because we're parsing using parse_dates parameter

In [127]:

```
from sklearn.preprocessing import MinMaxScaler

# Splitting date time data for date account created
df_all['dac_year'] = df_all.date_account_created.dt.year
df_all['dac_month'] = df_all.date_account_created.dt.month
df_all['dac_day'] = df_all.date_account_created.dt.day
df_all["dac_day_of_week"] = df_all.date_account_created.dt.dayofweek

# Splitting date time data for time first active
df_all['tfa_year'] = df_all.timestamp_first_active.dt.year
df_all['tfa_month'] = df_all.timestamp_first_active.dt.month
df_all['tfa_day'] = df_all.timestamp_first_active.dt.day
df_all['tfa_day_of_week'] = df_all.timestamp_first_active.dt.dayofweek
```

In [128]:

```
#Now that we created the variables let's drop these columns so we don't forget to do that later.
df_all.drop('date_account_created',1, inplace=True)
df_all.drop('timestamp_first_active',1, inplace=True)
```

In [129]:

```
df_all
# View the data after creating those new variables to make sure they're created properly
```

Out[129]:

	id	date_first_booking	gender	age	signup_method	signup_flow	language	affiliate_channel	affiliate_pr
0	gxn3p5htnn	NaT	-unknown-	NaN	facebook	0	en	direct	
1	820tgsjxq7	NaT	MALE	38.0	facebook	0	en	seo	
2	4ft3gnwmtx	2010-08-02	FEMALE	56.0	basic	3	en	direct	
3	bjjt8pjhuk	2012-09-08	FEMALE	42.0	facebook	0	en	direct	
4	87mebub9p4	2010-02-18	-unknown-	41.0	basic	0	en	direct	
...	...	...	...	...	...	...	...	...	
275542	cv0na2lf5a	NaT	-unknown-	31.0	basic	0	en	direct	
275543	zp8xfonng8	NaT	-unknown-	NaN	basic	23	ko	direct	
275544	fa6260ziny	NaT	-unknown-	NaN	basic	0	de	direct	
275545	87k0fy4ugm	NaT	-unknown-	NaN	basic	0	en	sem-brand	
275546	9uqfg8txu3	NaT	FEMALE	49.0	basic	0	en	other	

275547 rows × 22 columns

In [130]:

```
df_all.isna().sum()
```

Out[130]:

```
id                              0
date_first_booking         186639
gender                          0
age                        116866
signup_method                   0
signup_flow                     0
language                        0
affiliate_channel               0
affiliate_provider              0
first_affiliate_tracked      6085
signup_app                      0
first_device_type               0
first_browser                   0
country_destination         62096
dac_year                        0
dac_month                       0
dac_day                         0
dac_day_of_week                 0
tfa_year                        0
tfa_month                       0
tfa_day                         0
tfa_day_of_week                 0
dtype: int64
```

here we see date_first_booking has a lot of NA values and so does age and first_affiliate_trakced

here we see date_first_booking has a lot of NA values and so does age and first_affiliate_tracked

Let's explore the age values to see what's happening here

In [131]:

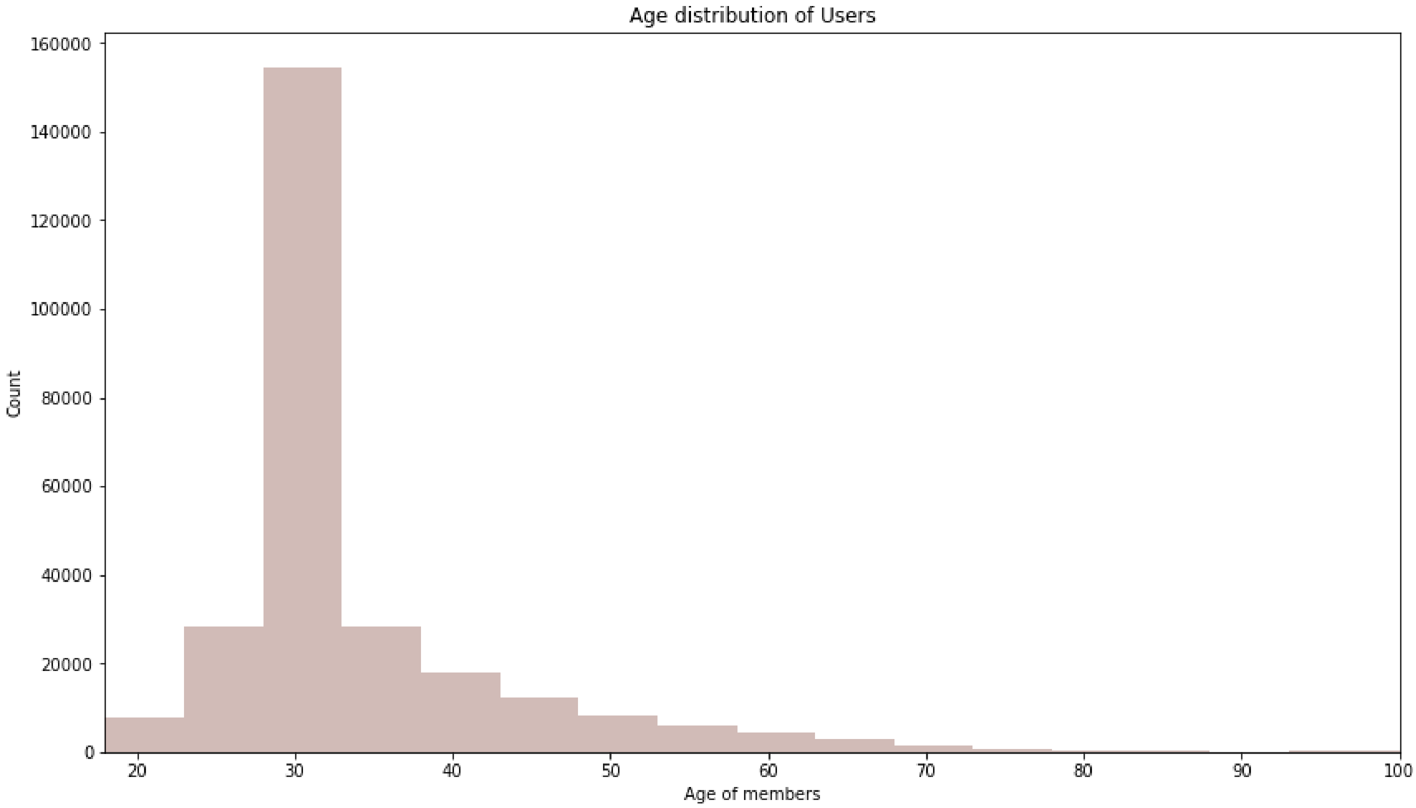
```
df_all.age.value_counts()
df_all.age.describe()
```

Out[131]:

```
count    158681.000000
mean         47.145310
std         142.629468
min           1.000000
25%          28.000000
50%          33.000000
75%          42.000000
max        2014.000000
Name: age, dtype: float64
```

In [137]:

```
#Age distribution before we normalized. There are a LOT of NA values that aren't included
in this because we're
# dropping the NA values
plt.figure(figsize=[14,8])
sns.distplot(df_all.age.dropna(),bins=np.arange(18,100+5,5),color=sns.color_palette()[5]
,kde=False);
plt.xlabel('Age of members')
plt.ylabel('Count')
plt.title('Age distribution of Users')
plt.xlim(18,100);
```



As we can see, there is a ton of variablility in the age column but that seemed to be a good indicator of where people would book their flights so we wanted to normalize this in some way.

The approach we took for this one was to find where the age values were greater than 1000 i.e. there was human error in inputting them, and then fill those values with a random int from 28 to 43 which was our 25% to 75% quantile range roughly. For the na values, we also just filled those up. We tried several different methods of filling the age values and found that this produced the best results by and large.

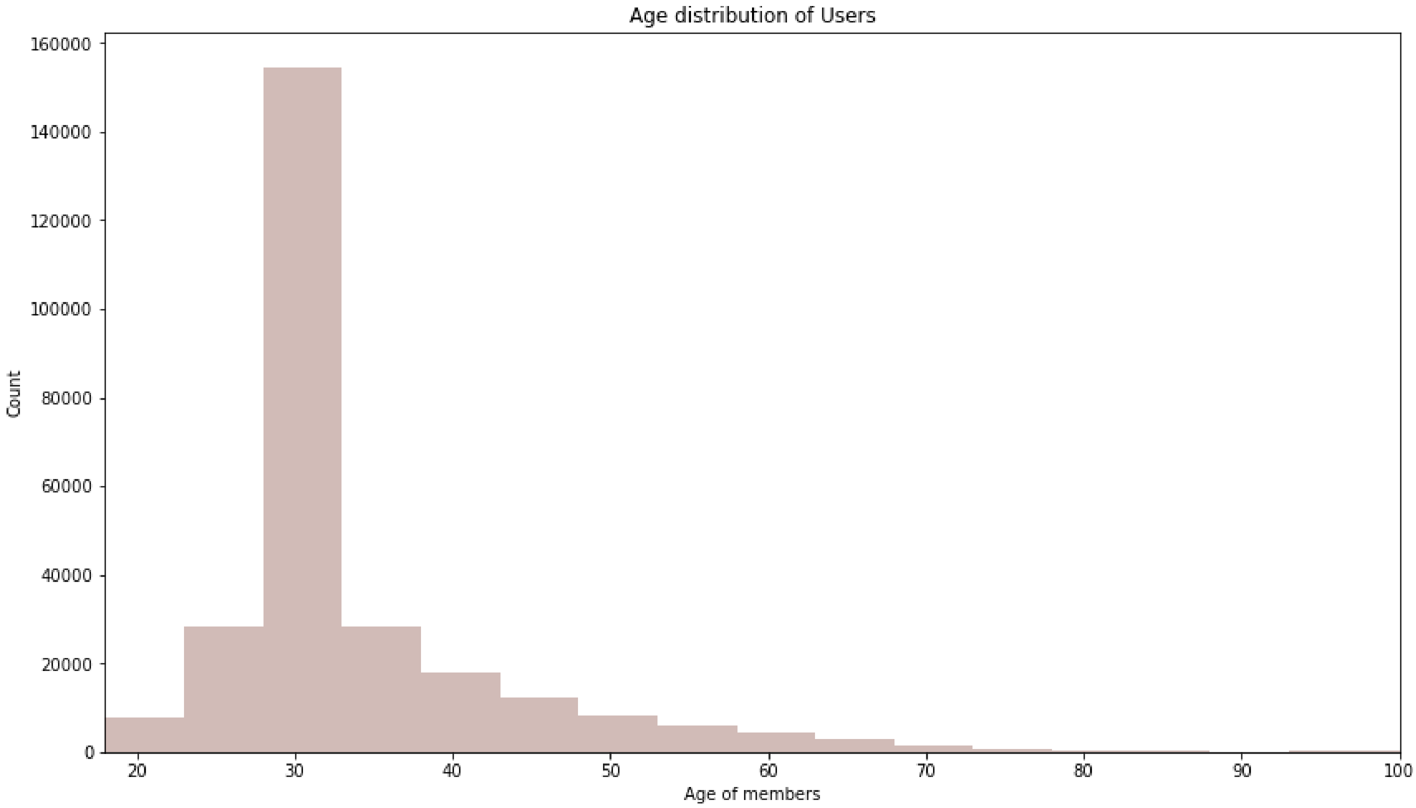
In [133]:

```
#df_all.loc[df_all.age > 100, 'age'] = np.nan
#df_all.loc[df_all.age < 18, 'age'] = np.nan

age_values = df_all.age.values
df_all['age'] = np.where(age_values>1000, np.random.randint(28, 43), age_values)
df_all['age'] = df_all['age'].fillna(np.random.randint(28, 43))
```

In [134]:

```
#Plotting Age distribution of the members
# After normalization with the random values
plt.figure(figsize=[14,8])
sns.distplot(df_all.age.dropna(),bins=np.arange(18,100+5,5),color=sns.color_palette()[5]
,kde=False);
plt.xlabel('Age of members')
plt.ylabel('Count')
plt.title('Age distribution of Users')
plt.xlim(18,100);
```



In [13]:

```
df_all.age.describe()
# We still have some very obvious outliers, max age = 150, but this is much better than b
efore
# and we didn't mess with the distributions as much.
```

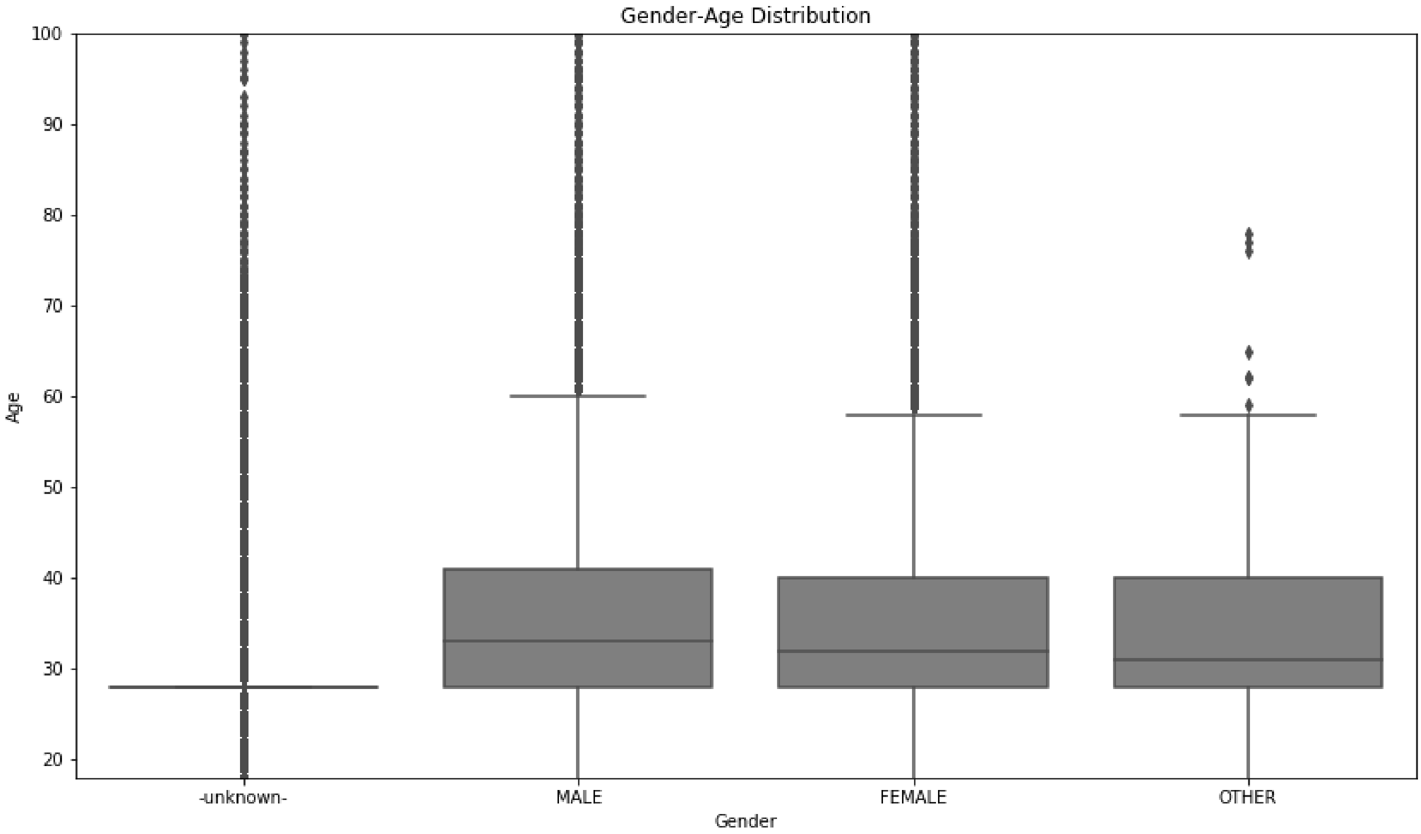
Out[13]:

```
count    275547.000000
mean         35.234896
std          10.645925
min           1.000000
25%          32.000000
50%          33.000000
75%          35.000000
max         150.000000
Name: age, dtype: float64
```

In [135]:

```
# relationship between gender and age
```

```
plt.figure(figsize=[14,8])
sns.boxplot(data=df_all,y='age',x='gender',color=sns.color_palette()[7]);
plt.ylim(18,100)
plt.xlabel('Gender');
plt.ylabel('Age');
plt.title('Gender-Age Distribution');
```



In [14]:

```
#We're grouping up the signup devices and then filling the na withe the medians of our se
ries.
# this accomplishes normalization for our signup methods and first_device types

by_signup_device = df_all.groupby(['signup_method', 'first_device_type'])
def impute_median(series):
    return series.fillna(series.median())
df_all.age = by_signup_device['age'].transform(impute_median)
```

In [15]:

```
# first affiliate tracked isn't necessarily important but we saw a few notebooks that use
d this
# it basically tells us how the user found airbnb. We thought this would be a useful feat
ure and turned this into a
# a one hot encoded column.

tracked = []
for i in df_all['first_affiliate_tracked']:
    if i == "untracked" or i == "":
        isTracked = 0
    else:
        isTracked = 1
    tracked.append(isTracked)

df_all['is_first_affiliate_tracked'] = tracked
```

This part is pretty important, we decided to group up the seconds information for sessions because we thought people who spent more time browsing the website would choose different locations. We decided to group these people all together and then merged that with our

In [16]:

```
seconds = sessions.groupby('user_id', as_index=False).agg({"secs_elapsed": "sum"})
df_all = pd.merge(df_all, seconds, left_on="id", right_on="user_id", how="left")
df_all['secs_elapsed'] = df_all['secs_elapsed'].fillna(0)
```

In [17]:

```
df_all.secs_elapsed.value_counts()
# so it looks like seconds elapsed has way too many 0 or NA values so we ended up droppin
g this from our dataframe

df_all = df_all.drop('secs_elapsed', axis=1)
```

In [18]:

```
df_all.columns
```

Out[18]:

```
Index(['id', 'date_first_booking', 'gender', 'age', 'signup_method',
       'signup_flow', 'language', 'affiliate_channel', 'affiliate_provider',
       'first_affiliate_tracked', 'signup_app', 'first_device_type',
       'first_browser', 'dac_year', 'dac_month', 'dac_day', 'dac_day_of_week',
       'tfa_year', 'tfa_month', 'tfa_day', 'tfa_day_of_week',
       'is_first_affiliate_tracked', 'user_id'],
      dtype='object')
```

Here we're cerating dummy variables for each of the features seen below. These features are ones that contain alphanumeric values such as names. We're turning them into a dummy features to basically turn all of this into one hot encoding. Finally we added that to our original dataframe which is now called df2.

In [19]:

```
method = pd.get_dummies(df_all[["signup_method"]])
affch = pd.get_dummies(df_all[["affiliate_channel"]])
affprov = pd.get_dummies(df_all[["affiliate_provider"]])
firstdevice = pd.get_dummies(df_all[["first_device_type"]])
signupFlow = pd.get_dummies(df_all[["signup_flow"]].astype(str))
signup = pd.get_dummies(df_all[["signup_app"]])
genderdum = pd.get_dummies(df_all[["gender"]])
langdum = pd.get_dummies(df_all[["language"]])
browser = pd.get_dummies(df_all[["first_browser"]])

df2 = df_all

df2 = pd.concat([df2.reset_index(drop=True),method.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),affch.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),affprov.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),firstdevice.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),signupFlow.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),genderdum.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),signup.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),langdum.reset_index(drop=True)], axis=1)
df2 = pd.concat([df2.reset_index(drop=True),browser.reset_index(drop=True)], axis=1)
```

In [20]:

```
df2 = df2.drop(['id',  'date_first_booking','gender',
          'signup_method',  'affiliate_channel', 'affiliate_provider', 'first_device_typ
e',
          'first_browser', 'signup_app', 'first_browser', 'language', 'signup_flow', 'fi
rst_affiliate_tracked'], axis=1)
```

In [21]:

```
id_test = test_users['id']
```

In [22]:

```
print(len(Y.unique()))
print(Y.unique())
```

```
12
['NDF' 'US' 'other' 'FR' 'CA' 'GB' 'ES' 'IT' 'PT' 'NL' 'DE' 'AU']
```

In [23]:

```
print(df2.shape)
print(len(Y))
#print(train.shape)
print(df_all.shape)
```

```
(275547, 157)
213451
(275547, 23)
```

In [24]:

```
df_all.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 275547 entries, 0 to 275546
Data columns (total 23 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   id                        275547 non-null  object
 1   date_first_booking        88908 non-null   datetime64[ns]
 2   gender                    275547 non-null  object
 3   age                       275547 non-null  float64
 4   signup_method             275547 non-null  object
 5   signup_flow               275547 non-null  int64
 6   language                  275547 non-null  object
 7   affiliate_channel         275547 non-null  object
 8   affiliate_provider        275547 non-null  object
 9   first_affiliate_tracked   269462 non-null  object
 10  signup_app                275547 non-null  object
 11  first_device_type         275547 non-null  object
 12  first_browser             275547 non-null  object
 13  dac_year                  275547 non-null  int64
 14  dac_month                 275547 non-null  int64
 15  dac_day                   275547 non-null  int64
 16  dac_day_of_week           275547 non-null  int64
 17  tfa_year                  275547 non-null  int64
 18  tfa_month                 275547 non-null  int64
 19  tfa_day                   275547 non-null  int64
 20  tfa_day_of_week           275547 non-null  int64
 21  is_first_affiliate_tracked 275547 non-null  int64
 22  user_id                   135483 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(10), object(11)
memory usage: 50.5+ MB
```

In [25]:

```
df2 = df2.drop("user_id", axis = 1)
```

In [26]:

```
#We got this from feature selection

df2 = df2.drop(['gender_-unknown-', 'signup_method_facebook', 'age', 'affiliate_channel_c
ontent', 'first_browser_-unknown-', 'signup_app_Android', 'signup_app_iOS', 'signup_flow_
1', 'signup_method_basic', 'first_device_type_Other/Unknown', 'gender_MALE', 'signup_app_
Web', 'signup_flow_2', 'language_en', 'tfa_year', 'signup_flow_3', 'signup_flow_0', 'sign
up_app_Moweb', 'affiliate_channel_direct', 'first_device_type_Mac Desktop', 'signup_flow_
25', 'is_first_affiliate_tracked', 'affiliate_provider_meetup', 'affiliate_provider_faceb
ook', 'gender_FEMALE', 'affiliate_channel_sem-non-brand', 'signup_flow_12', 'dac_year', '
affiliate_channel_other', 'first_device_type_Android Phone', 'affiliate_provider_other',
'first_browser_Chrome', 'first_device_type_SmartPhone (Other)', 'first_browser_AOL Explor
er', 'signup_flow_24', 'signup_flow_5', 'affiliate_channel_api', 'first_browser_Firefox',
'language_ko', 'language_it', 'language_zh', 'affiliate_provider_craigslist', 'first_brow
ser_Camino', 'affiliate_provider_vast', 'first_browser_Silk', 'language_ja', 'signup_flow
_8', 'language_fi', 'language_fr', 'dac_month', 'first_device_type_Windows Desktop', 'aff
iliate_channel_seo', 'tfa_month', 'first_device_type_iPhone', 'first_browser_IE', 'first_
```

```
browser_Android Browser', 'affiliate_channel_remarketing', 'language_es', 'language_da',
'first_browser_Mobile Safari', 'language_de', 'first_browser_Safari', 'affiliate_provider
_google', 'affiliate_provider_facebook-open-graph', 'first_browser_Chrome Mobile', 'affil
iate_provider_padmapper', 'signup_flow_23', 'signup_flow_21', 'affiliate_provider_direct'
, 'affiliate_channel_sem-brand', 'first_device_type_iPad', 'language_pt', 'dac_day', 'fir
st_browser_Chromium', 'gender_OTHER', 'dac_day_of_week', 'tfa_day', 'affiliate_provider_b
ing', 'tfa_day_of_week', 'language_ru', 'first_device_type_Desktop (Other)', 'affiliate_p
rovider_yahoo', 'affiliate_provider_gsp', 'first_device_type_Android Tablet', 'first_brow
ser_Opera', 'signup_flow_6', 'affiliate_provider_email-marketing', 'language_nl', 'langua
ge_sv', 'signup_method_google', 'first_browser_BlackBerry Browser'], axis =1)
```

In [28]:

```
Y
```

Out[28]:

```
0           NDF
1           NDF
2            US
3         other
4            US
          ...
213446      NDF
213447      NDF
213448      NDF
213449      NDF
213450      NDF
Name: country_destination, Length: 213451, dtype: object
```

In [82]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn import model_selection

# We took this label encoding stuff from a separate notebook.
vals = df2.values
X = vals[:train_users.shape[0]]
le = LabelEncoder()
y = le.fit_transform(Y)
X_test = vals[train_users.shape[0]:]
```

In [83]:

```
xgb = XGBClassifier(max_depth=6, learning_rate=0.3, n_estimators=25,
                    objective='multi:softprob', subsample=0.5, colsample_bytree=0.5, see
d=0, nthread = -1)
xgb.fit(X, y)
```

Out[83]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.3, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=25, n_jobs=-1, nthread=-1, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, seed=0, subsample=0.5,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [84]:

```
y_pred = xgb.predict_proba(X_test)
```

In [85]:

```
ids = []  #list of ids
cts = []  #list of countries
for i in range(len(id_test)):
    idx = id_test[i]
    ids += [idx] * 5
```

```
    cts += le.inverse_transform(np.argsort(y_pred[i])[::-1])[:5].tolist()


#Generate submission
sub = pd.DataFrame(np.column_stack((ids, cts)), columns=['id', 'country'])
sub.to_csv('sub.csv',index=False)
```

In [99]:

```
set(cts)
exploreCountries = pd.DataFrame(cts, columns = ['country'])
exploreCountries.country.value_counts()
```

Out[99]:

```
NDF      62096
FR       62096
US       62096
other    62096
IT       62018
ES          44
GB          34
Name: country, dtype: int64
```

as we can see we're predicting roughly the same values for each of the classes but some classes are not included

In [60]:

```
# train test split for analysis
#Y = train_users['country_destination'] # these are our labels
#train_users = train_users.drop('country_destination', axis =1)


# Here we're using label encoding for the Y_train and y_test. This helped our accuracy al
ot.
# We're honestly not entirely sure why this helped so much but it could be because it's e
asier for the model
# to predict numerical values than it is to predict strings but we saw a bunch of kaggle
notebooks doing this
# thought it could be important, and our accuracy shot up alot.


train = df2.iloc[0:len(train_users),:]
le = LabelEncoder()
#Ynew = le.fit_transform(Y)

X_train, X_test, y_train, y_test = model_selection.train_test_split(
    train, Y, test_size=0.33, random_state=42)


#train_users_n = train_users.shape[0] # comment
#X_train = df2.values[:train_users_n] # comment
#le = LabelEncoder() # don't uncommment
#y_train = le.fit_transform(Y)

y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)


#X_test = df2.values[train_users_n:]



xgb = XGBClassifier(max_depth=6, learning_rate=0.3, n_estimators=25,
                    objective='multi:softprob', subsample=0.5, colsample_bytree=0.5, see
d=0)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict_proba(X_test)
```

In [47]:

```
y_pred_xgb
```

Out[47]:

```
array([[0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335],
       [0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335],
       [0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335],
       ...,
       [0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335],
       [0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335],
       [0.00351094, 0.00767762, 0.00580891, ..., 0.00207162, 0.28722212,
        0.04675335]], dtype=float32)
```

In [64]:

```
from sklearn.metrics import classification_report, ndcg
ids = []  #list of ids
cts = []  #list of countries
for i in range(len(y_pred_xgb)):
    #idx = id_test[i]
    #ids += [idx] * 5
    cts += le.inverse_transform(np.argsort(y_pred_xgb[i])[::-1])[:1].tolist()

y_test = le.inverse_transform(y_test)
#cts is our inverse transformed of y_pred_xgb
#print(classification_report(y_test, cts))
```

In [113]:

```
print(set(y_test))
print(set(cts))
```

```
{'AU', 'ES', 'CA', 'DE', 'other', 'US', 'GB', 'PT', 'IT', 'FR', 'NL', 'NDF'}
{'ES', 'other', 'US', 'GB', 'FR', 'IT', 'NDF'}
```

In [79]:

```
from sklearn.metrics import multilabel_confusion_matrix

print(classification_report(y_test, cts))
multilabel_confusion_matrix(y_test,cts, labels=list(set(Y)))
```

```
              precision    recall  f1-score   support

          AU       0.00      0.00      0.00       182
          CA       0.00      0.00      0.00       456
          DE       0.00      0.00      0.00       358
          ES       0.00      0.00      0.00       743
          FR       0.00      0.00      0.00      1645
          GB       0.00      0.00      0.00       837
          IT       0.00      0.00      0.00       952
         NDF       0.58      1.00      0.73     40905
          NL       0.00      0.00      0.00       241
          PT       0.00      0.00      0.00        86
          US       0.17      0.00      0.00     20714
       other       0.00      0.00      0.00      3320

    accuracy                           0.58     70439
   macro avg       0.06      0.08      0.06     70439
weighted avg       0.39      0.58      0.43     70439
```

Out[79]:

```
array([[[70257,     0],
```

```
        [  182,      0]],

       [[69696,      0],
        [  743,      0]],

       [[69983,      0],
        [  456,      0]],

       [[70081,      0],
        [  358,      0]],

       [[70198,      0],
        [  241,      0]],

       [[67119,      0],
        [ 3320,      0]],

       [[49715,     10],
        [20712,      2]],

       [[69602,      0],
        [  837,      0]],

       [[70353,      0],
        [   86,      0]],

       [[68794,      0],
        [ 1645,      0]],

       [[69487,      0],
        [  952,      0]],

       [[    3, 29531],
        [    9, 40896]]]])
```

So as we can see from the classificaiton report, accuracy is not the best metric to use with this problem. From the train test split, it's obvious that our model is mostly predicting US and NDF values. This isn't very odd since the actual dataset is incredibly skewed towards the US and NDF labels.

However, when we look at the predictions on the entire dataset and not a split of it, we can see that our data is better distributed. We're predicting US, FR, NDF, etc. with the same amount of accuracy. This is odd because our data is skewed towards NDF and the US. We're not really sure why this is happening but it is noteworthy to report

Applications: So a real life application of this project is discussed in the introduction. AirBnB could use this report to help market to people if they were able to predict where that person is going to be making their first booking. They would be able to better forcast demand and also recruit new hosts in those countries. Based off of our analysis, it looks like currently most people are going to the US/France/Spain/Italy. Therefore, AirBnB could look into expanding into high demand areas within each of these countries. There is also an incredible amount of people that don't end up booking vacations (NDF values) which is an issue for AirBnB. A short survey on their

values) which is an issue for AirBnB. A short survey on their website could reveal some more information but it could be dangerous to speculate on this. We believe that if AirBnB augmented the dataset, they could discover where people were looking before not choosing a location. There may be overlap in these areas and therefore this gives AirBnB a clear path forward on where to expand their operations to.

Appendix, extra information/ trial and error things to show our thought process, not going to be annotated.

In [ ]:

```

```

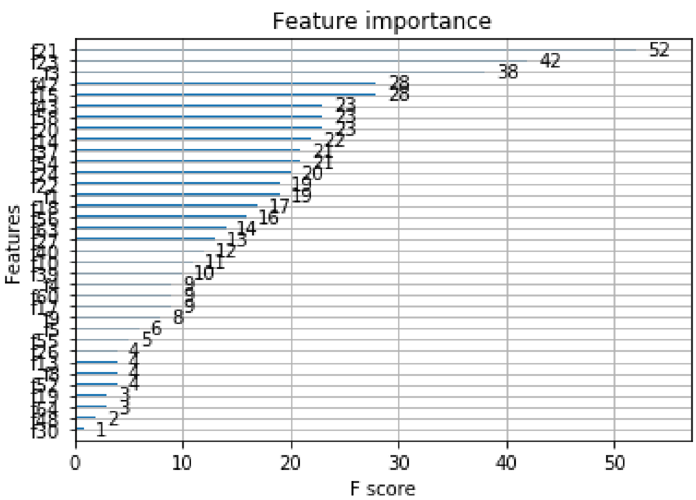
From the above section you saw we performed feature selection, we do that here. FEATURE SELECTION SECTION don't run this section below since it'll probably throw some errors. we ran this once with our model fit on a huge number of features. This is only for documentation purposes

In [36]:

```
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot
```

In [115]:

```
plot_importance(xgb)
pyplot.show()
```



In [114]:

```
importance = xgb.feature_importances_
importance_df = pd.DataFrame(importance, index=X_train.columns,
                             columns=["Importance"])
```

In [ ]:

```
importance_df = importance_df.sort_values(by = ['Importance'], ascending = False)
```

In [ ]:

```
importance_df.to_csv('importance3.csv')
```

In [ ]:

```
trim = importance_df[importance_df.Importance > 0]
```

In [ ]:

```
trim.to_csv('trim.csv')
```

In [ ]:

```
indexNamesArr = trim.index.values
listoftrim = list(indexNamesArr)
```

In [ ]:

```
train = train[listoftrim]
test = test[listoftrim]
```

In [ ]:

```
print(listoftrim)
```

''' Our trimmed features that we took out. these all had a 0 F1 score. ['gender_-unknown-', 'signup_method_facebook', 'age', 'affiliate_channel_content', 'first*browser*-unknown-', 'signup_app_Android', 'signup_app_iOS', 'signup_flow_1', 'signup_method_basic', 'first_device_type_Other/Unknown', 'gender_MALE', 'signup_app_Web', 'signup_flow_2', 'language_en', 'tfa_year', 'signup_flow_3', 'signup_flow_0', 'signup_app_Moweb', 'affiliate_channel_direct', 'first_device_type_Mac Desktop', 'signup_flow_25', 'is_first_affiliate_tracked', 'affiliate_provider_meetup', 'affiliate_provider_facebook', 'gender_FEMALE', 'affiliate_channel_sem-non-brand', 'signup_flow_12', 'dac_year', 'affiliate_channel_other', 'first_device_type_Android Phone', 'affiliate_provider_other', 'first_browser_Chrome', 'first_device_type_SmartPhone (Other)', 'first_browser_AOL Explorer', 'signup_flow_24', 'signup_flow_5', 'affiliate_channel_api', 'first_browser_Firefox', 'language_ko', 'language_it', 'language_zh', 'affiliate_provider_craigslist', 'first_browser_Camino', 'affiliate_provider_vast', 'first_browser_Silk', 'language_ja', 'signup_flow_8', 'language_fi', 'language_fr', 'dac_month', 'first_device_type_Windows Desktop', 'affiliate_channel_seo', 'tfa_month', 'first_device_type_iPhone', 'first_browser_IE', 'first_browser_Android Browser', 'affiliate_channel_remarketing', 'language_es', 'language_da', 'first_browser_Mobile Safari', 'language_de', 'first_browser_Safari', 'affiliate_provider_google', 'affiliate_provider_facebook-open-graph', 'first_browser_Chrome Mobile', 'affiliate_provider_padmapper', 'signup_flow_23', 'signup_flow_21', 'affiliate_provider_direct', 'affiliate_channel_sem-brand', 'first_device_type_iPad', 'language_pt', 'dac_day', 'first_browser_Chromium', 'gender_OTHER', 'dac_day_of_week', 'tfa_day', 'affiliate_provider_bing', 'tfa_day_of_week', 'language_ru', 'first_device_type_Desktop (Other)', 'affiliate_provider_yahoo', 'affiliate_provider_gsp', 'first_device_type_Android Tablet', 'first_browser_Opera', 'signup_flow_6', 'affiliate_provider_email-marketing', 'language_nl', 'language_sv', 'signup_method_google', 'first_browser_BlackBerry Browser'] trim list '''''

In [ ]:

```
Y_Predict = xgb.predict_proba(test)
```

In [ ]:

```
len(y_pred_xgb)
```

In [ ]:

```
generate_answer(y_pred_xgb, 'XGB')
```

In [ ]:

```
submit = generate_answer(Y_Predict, 'XGB')
```

In [ ]:

```
submit
```

In [ ]:

```
# This was us experimenting with the for loop that does inverse_transform. again,
#do not run this it is not necessary this is only for documentation purposes
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
ids = []  #list of ids
cts = []  #list of countries
y_predicted = y_pred_xgb
for i in range(len(y_pred_xgb)):
    #idx = id_test[i]
    #ids += [idx] * 5
    cts += le.inverse_transform(np.argsort(y_predicted[i])[::-1])[:1].tolist()

#scores = cross_val_score(xgb, X_train, y_train, cv = 10)
#multilabel_confusion_matrix(y_test, y_pred_xgb)
```

In [ ]:

```
# here we're experimenting with how to get the NDCG score for our report but we weren't a
ble to figure it out
# we're going to be showing the classification report instead.

print(len(cts))
print(len(y_test))
print(len(y_pred_xgb))
print(y_pred_xgb[0])
print(le.inverse_transform(np.argsort(y_pred_xgb[0])[::-1])[:1].tolist())
```

In [ ]:

```
from sklearn.metrics import ndcg_score

# part 2 of experimenting with NDCG score, not required to run.

#print(y_test)
#print(y_pred_xgb)
#print(len(y_test))
#print(len(y_pred_xgb))

print(y_test[0])
print(y_pred_xgb[0])
print(len(y_pred_xgb[0]))



testArray = [[0,0,0,1,0,0,0,0,0,0,0,0]]
print(len(testArray))
testArray2 = [y_pred_xgb[0]]
#print(cts[0])
ndcg_score(testArray, testArray2)
```