

# Capstone Project - Machine Learning Engineer Nanodegree

## Definition

### Project Overview

New users on Airbnb can book a place to stay in 34,000+ cities across 190+ countries. By accurately predicting where a new user will book their first travel experience, Airbnb can share more personalized content with their community, decrease the average time to first booking, and better forecast demand. Airbnb challenges us to predict in which country a new user will make his or her first booking.



Figure 1: Airbnb Logo

It is a [Kaggle Competition](#). The datasets are available [here](#). Kaggle is a wonderful platform for Data Scientists and Machine Learning Engineers. The goal is to work on this challenge to gain knowledge and practice Machine Learning Techniques.

### Problem Statement

In this challenge, I am given a list of users along with their demographics, web session records, and some summary statistics. I am asked to predict which country a new user's first booking destination will be.

- All the users in the dataset are from the USA,
- 12 possible outcomes of the destination country: US, FR, CA, GB, ES, IT, PT, NL, DE, AU, NDF (no destination found), and 'other'
- Datasets
- countries.csv

- sessions.csv
- train\_users\_2.csv
- age\_gender\_bkts.csv
- The training and test sets are split by dates
- Test set: activity after 12/5/2015
- Sessions dataset: the data only dates back to 1/1/2014 while the users dataset dates back to 2010

### Datasets Description

The file description can be found on [Kaggle](#).

**train\_users.csv** - the training set of users.

**test\_users.csv** - the test set of users.

- 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

**sessions.csv** - web sessions log for users.

- user\_id: to be joined with the column 'id' in users table
- action
- action\_type
- action\_detail
- device\_type
- secs\_elapsed

**countries.csv** - summary statistics of destination countries in this dataset and their locations.

**age\_gender\_bkts.csv** - summary statistics of users' age group, gender, country of destination.

**sample\_submission.csv** - correct format for submitting your predictions.

## Metrics

For each new user, I am to make a maximum of **5 predictions** on the country of the first booking. The ground truth country is marked with relevance = 1, while the rest have relevance = 0.

### Definition

The Metrics used in this competition is the **Normalized discounted cumulative gain (NDCG)**. It measures the performance of a recommendation system based on the graded relevance of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities. It is commonly used in Information Retrieval and to evaluate performance of web search engines.

$$DCG_k = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\lg(i + 1)} \quad (1)$$

$$nDCG_k = \frac{DCG_k}{IDCG_k} \quad (2)$$

where  $rel_i$  is the relevance of the result at position  $i$ .  $IDCG_k$  is the maximum possible  $DCG$  for a given set of queries.

### Justification

It makes a lot of sense to use an NCDG metrics in this challenge because the predictions can be seen as documents returned from a Search Engine. The ordering matters and it has to be graded accordingly.

The **NCDG** metrics is an enhancement over the simple Cumulative Gain (CG) metrics and the Discounted Cumulative Gain (DCG) metrics.

- Cumulative Gain does not include the position of a result in the consideration of the usefulness of a result set. Hence, the ordering does not change the CG values.
- With DCG, highly relevant documents that appear lower in the result should be penalized. The graded relevance value is reduced logarithmically proportional to the position of the result. The problem with DCG is that result lists vary in length depending on the query. Consequently, comparing performance from one query to the next cannot be consistently achieved using DCG alone. This is why the DCG is normalized with the IDCG.

### **Limitations of the NDCG metrics**

- NDCG does not penalize for bad documents in the result
- NDCG does not penalize for missing documents in the result. One workaround is to enforce fixed set size for the result set and use minimum score for the missing documents
- See this [Wikipedia entry](#) for more details.

# Capstone Project - Airbnb New user booking

April 26, 2016

## 1 Analysis

### 1.1 Data Exploration and Exploratory Visualization

Some data exploration and exploratory visualization will be performed in order to better understand the data I am dealing with in this Kaggle Competition.

```
In [2]: # Import libraries
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
```

```
# Math
from math import floor
```

```
%matplotlib inline
```

```
In [3]: # Loading data in Memory
country_data = pd.read_csv("../data/countries.csv")
gender_data = pd.read_csv("../data/age_gender_bkts.csv")

# train data / test data
train_data = pd.read_csv("../data/train_users_2.csv")
target = train_data['country_destination']
test_data = pd.read_csv("../data/test_users.csv")
```

```
In [8]: # Some useful constants
n_train_data = len(train_data)
n_test_data = len(test_data)
print("#training samples: {}".format(n_train_data))
```

```
#training samples: 213451
```

```
In [4]: # summary statistics of destination countries in this dataset and their locations
country_data.head()
```

```
Out[4]:  country_destination  lat_destination  lng_destination  distance_km \
0              AU          -26.853388        133.275160      15297.7440
1              CA           62.393303        -96.818146       2828.1333
2              DE           51.165707         10.452764       7879.5680
```

3	ES	39.896027	-2.487694	7730.7240
4	FR	46.232193	2.209667	7682.9450

	destination_km2	destination_language	language_levenshtein_distance
0	7741220.0	eng	0.00
1	9984670.0	eng	0.00
2	357022.0	deu	72.61
3	505370.0	spa	92.25
4	643801.0	fra	92.06

```
In [5]: # summary statistics of users' age group, gender, country of destination
gender_data.head()
```

```
Out[5]:
```

	age_bucket	country_destination	gender	population_in_thousands	year
0	100+	AU	male	1.0	2015.0
1	95-99	AU	male	9.0	2015.0
2	90-94	AU	male	47.0	2015.0
3	85-89	AU	male	118.0	2015.0
4	80-84	AU	male	199.0	2015.0

```
In [6]: # What are the train data like?
train_data.head()
```

```
Out[6]:
```

	id	date_account_created	timestamp_first_active	date_first_booking
0	gxn3p5htnn	2010-06-28	20090319043255	NaN
1	820tgsjxq7	2011-05-25	20090523174809	NaN
2	4ft3gnwmtx	2010-09-28	20090609231247	2010-08-02
3	bjjt8pjhuk	2011-12-05	20091031060129	2012-09-08
4	87mebub9p4	2010-09-14	20091208061105	2010-02-18

	gender	age	signup_method	signup_flow	language	affiliate_channel
0	-unknown-	NaN	facebook	0	en	direct
1	MALE	38.0	facebook	0	en	seo
2	FEMALE	56.0	basic	3	en	direct
3	FEMALE	42.0	facebook	0	en	direct
4	-unknown-	41.0	basic	0	en	direct

	affiliate_provider	first_affiliate_tracked	signup_app	first_device_type
0	direct	untracked	Web	Mac Desktop
1	google	untracked	Web	Mac Desktop
2	direct	untracked	Web	Windows Desktop
3	direct	untracked	Web	Mac Desktop
4	direct	untracked	Web	Mac Desktop

	first_browser	country_destination
0	Chrome	NDF
1	Chrome	NDF
2	IE	US
3	Firefox	other
4	Chrome	US

- Some data samples are presented in the previous table.
- Some features might need some cleaning
- Age (a lot of them are NaN and wrong values)

- Dates - We'll need to parse them and transform them into some other features (seasons, years, ...)

Some statistics about the provided features and dataset

- 15 features.
- Mix of continuous and discrete features (age, timestamp, categorical...).
- Size of training set: 213451 instances.
- 12 classes: NDF, US, other, FR, IT, GB, ES, CA, DE, NL, AU, PT

Here is a complete description of the provided features

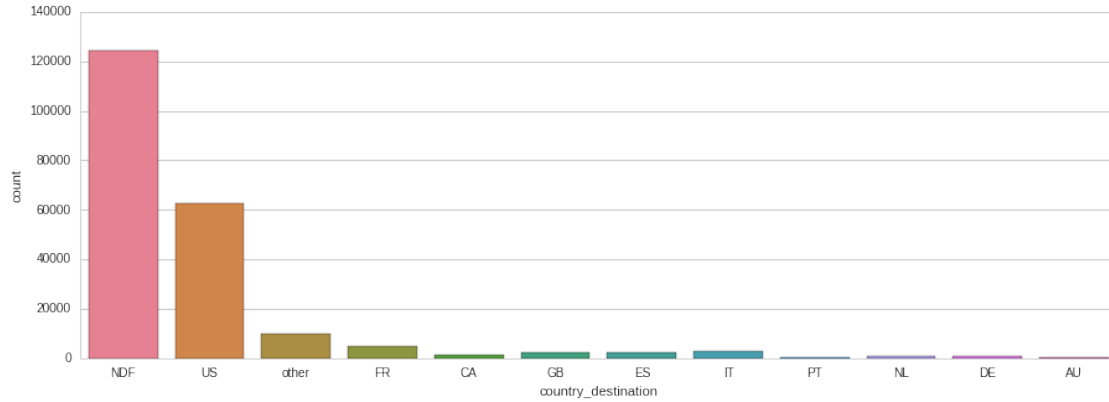
- id: user id (sequence of characters and digits)
- date\_account\_created: the date of account creation, format "YYYY-mm-dd" (eg. 2010-06-28)
- timestamp\_first\_active: timestamp of the first activity, note that it can be earlier than date\_account\_created or date\_first\_booking because a user can search before signing up (integer, eg. 20090319043255)
- date\_first\_booking: date of first booking, format "YYYY-mm-dd"
- gender: categorical (MALE, FEMALE, OTHER, unknown)
- age: discrete variable (from 1 to 2014 with NaNs)
- signup\_method: categorical variable (facebook, google, basic, ...)
- signup\_flow: the page a user came to signup up from, categorical variable (0 to 5)
- language: international language preference, categorical variable (eg. en)
- affiliate\_channel: what kind of paid marketing, categorical variable (eg. seo, channel, ...)
- affiliate\_provider: where the marketing is e.g. google, craigslist, other, categorical variable (eg. direct, google, ...)
- first\_affiliate\_tracked: what's the first marketing the user interacted with before the signing up, categorical variable
- signup\_app: categorical variable (eg. Web)
- first\_device\_type: categorical variable (eg. Mac Desktop, Windows Desktop, ...)
- first\_browser: categorical variable (eg. Chrome, Safari, ...)
- country\_destination: this is the target variable you are to predict (14 classes)

```
In [9]: # Let's have a look at the training data
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
sns.countplot(x='country_destination', data=train_data, palette="husl", ax=axis1)

pd.DataFrame({'country_destination_percent' : target.value_counts() * 100 / n_train_data})
```

```
Out[9]:
```

	country_destination_percent
NDF	58.347349
US	29.222632
other	4.728954
FR	2.353233
IT	1.328174
GB	1.088774
ES	1.053638
CA	0.669006
DE	0.497070
NL	0.356991
AU	0.252517
PT	0.101663



The data suggest that

- 58% of the new users do not make any booking.
- 30% book in the US
- 5% are other

It seems that americans tend to prefer traveling in their own country rather than traveling abroad.

## 1.2 Benchmark

In order to assess our model's performances, some benchmarks are required.

### 1.2.1 Sample submission Benchmark 1

The first idea that comes to mind is to create a classifier that outputs only NDF as predictions. It should provide a simple benchmark for our next iterations. Below is the code used to generate this simple benchmark and writes it in a CSV file so that it can be submitted to Kaggle.

```
def df_submission(ids, f):
    ids_result = [] # list of ids
    cts = [] # list of countries
    for i in range(len(ids)):
        idx = ids[i]
        ids_result += [idx] * 5
        cts += f(i)
    return pd.DataFrame(np.column_stack((ids_result, cts)),
                        columns=['id', 'country'])

def generate_benchmark1_submission(df_train, target, ids):
    df_sub = df_submission(ids, lambda i: ['NDF'])
    df_sub.to_csv(SUBMISSION_PATH + 'benchmark1.csv', index=False)
    return df_sub
```

The NDCG@5 score is: **0.68411** (from Kaggle's scorer)

### 1.2.2 Sample submission Benchmark 2

The second benchmark that can be used to assess our model's performance is the following: Always predict the 5 top country destinations sorted by frequencies (see previous table). \* NDF \* US \* other \* FR \* IT

Here is the code for generating this benchmark and saving it into a csv file:



```
def generate_benchmark2_submission(df_train, target, ids):
    prediction = target.value_counts().index.tolist()[:5]
    df_sub = df_submission(ids, lambda i: prediction)
    df_sub.to_csv(SUBMISSION_PATH + 'benchmark2.csv', index=False)
    return df_sub
```

The NDCG@5 score is: **0.85670** (from Kaggle's scorer)

### 1.2.3 Best entry on this competition

The current best entry is scored **0.88697** in the leaderboard. It is relatively close to the sample submission benchmark 2 (score difference of **0.03**).

### 1.2.4 Goal

The goal of this project is to create a model that is in between the sample submission benchmark 2 and the best entry in the competition. It means that my model's score should be in the range [**0.85670**, **0.88697**]

## 1.3 Age

In this section, I will explore and discuss the age data.

```
In [149]: # Age exploration
age = train_data['age'].dropna() # A lot of NaN -> fix it with ML?
real_age = age[(age >= 5) & (age <= 95)] # A lot of outliers
print(real_age.describe())

# % of NaNs
percentage_age_NaN = train_data['age'].isnull().sum() * 100 / real_age.describe()['count']
print("% of NaN entries: {0:.1f}%".format(percentage_age_NaN))
```

count	123011.000000
mean	36.501142
std	11.585328
min	5.000000
25%	28.000000
50%	34.000000
75%	42.000000
max	95.000000

Name: age, dtype: float64  
% of NaN entries: 71.5%

```
In [201]: # Exploring the Age Distribution

# Training Data
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Age Distribution - Training Data')
sns.countplot(x='age', data=train_data, palette="husl", ax=axis1)

# Removing the age values
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Age Distribution with the years only - Training Data')
sns.countplot(x='age', data=train_data[train_data['age'] > 300], palette="husl", ax=axis1)

# Removing the year values
fig, (axis1,axis2) = plt.subplots(1,2,figsize=(15,5))
```

```

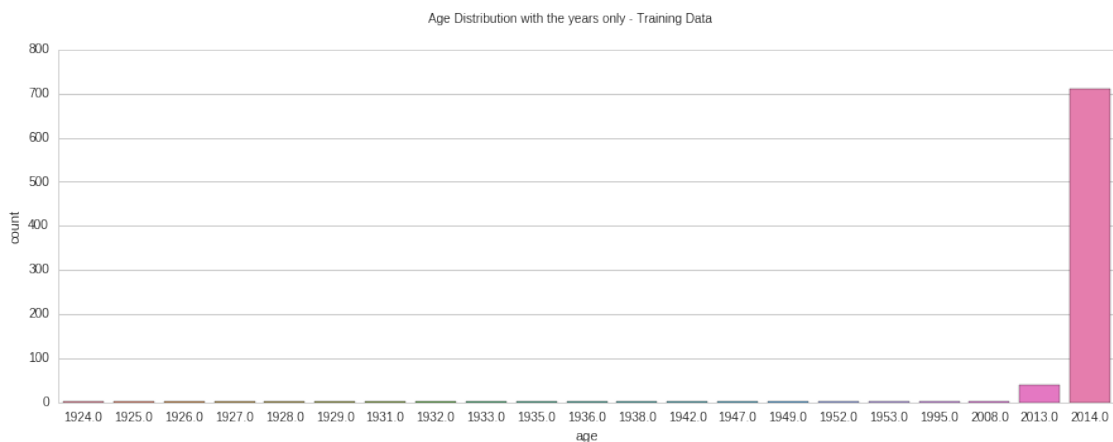
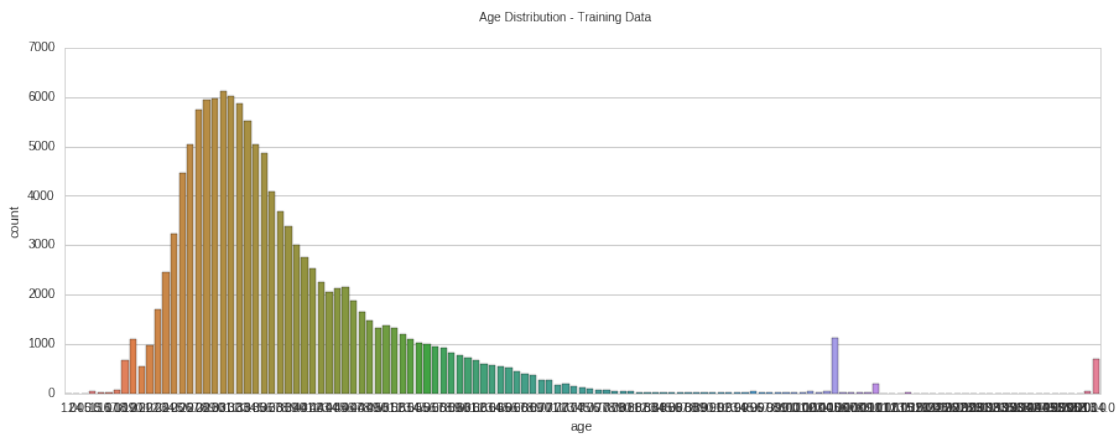
fig.suptitle('Age Distribution without the years - Training Data')
sns.countplot(x='age', data=train_data[train_data['age'] < 100], palette="husl", ax=axis1)
sns.distplot(train_data[train_data['age'] < 100]['age'], ax=axis2)

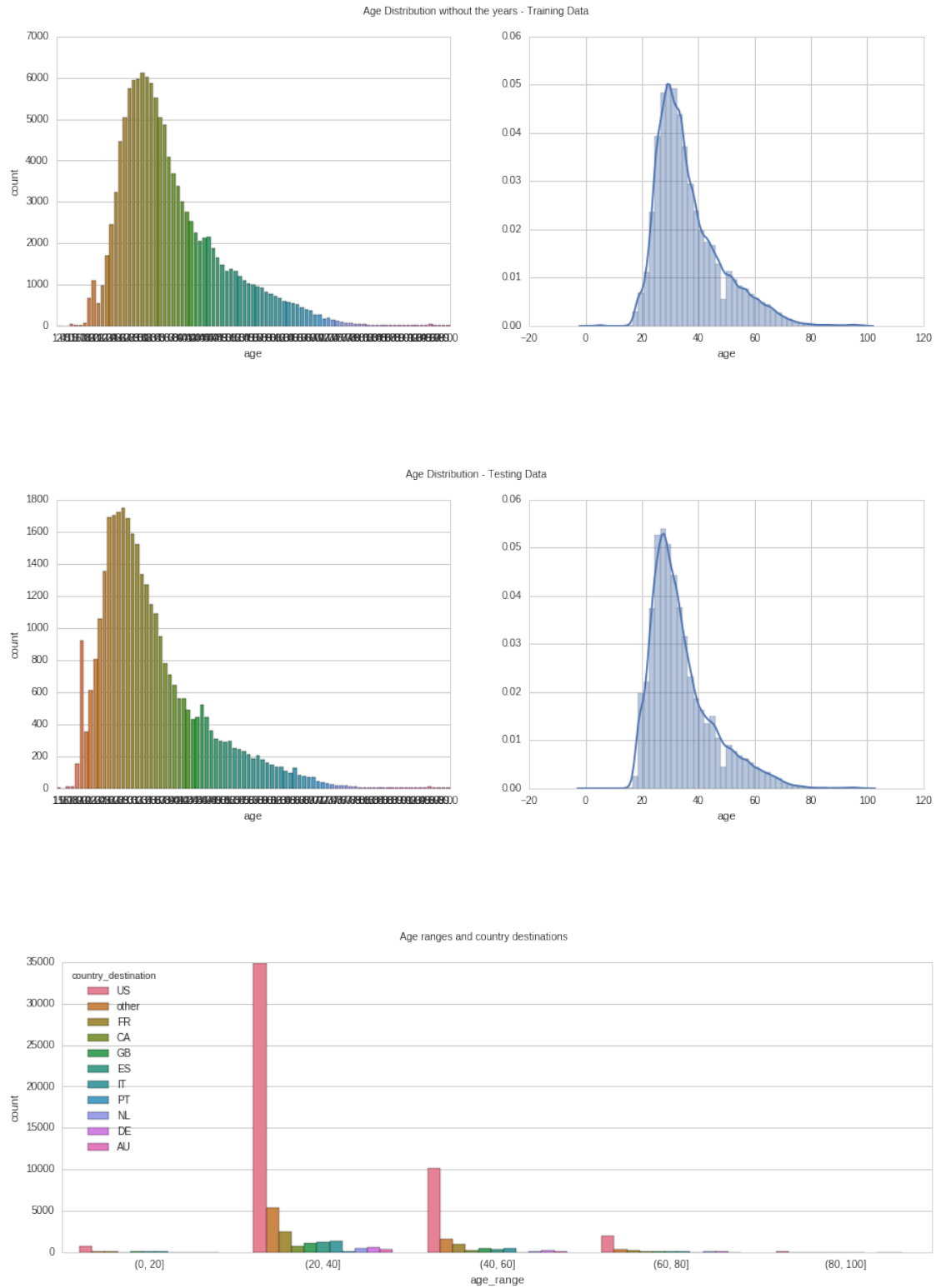
# Testing Data
fig, (axis1, axis2) = plt.subplots(1,2,figsize=(15,5))
fig.suptitle('Age Distribution - Testing Data')
sns.countplot(x='age', data=test_data[test_data['age'] < 100], palette="husl", ax=axis1)
sns.distplot(test_data[test_data['age'] < 100]['age'], ax=axis2)

# cut age values into ranges
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Age ranges and country destinations')
train_data['age_range'] = pd.cut(train_data["age"], [0, 20, 40, 60, 80, 100])
sns.countplot(x="age_range",hue="country_destination", data=train_data[train_data['country_destination'] != 'Other'])

```

Out[201]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f2902496210>





I notice that the age data is malformed

- Sometimes it is an age and sometimes a year date,

- Some values do not make much sense: age  $\leq 5$ , age  $\geq 95$ ,
- There are a lot of NaN values meaning that these people did not fill out this form input.

Some thoughts on the data \* The age distribution seems close to a **Poisson distribution**, \* Training and testing data follow the same distribution for age. Which is very important for the training and predicting.

\* I will need to clean up the data by transforming years

### Cleaning the data

- Set a valid range (x0, x1) where x0 = 15, x1 = 90
- any values between 1915 and 2000 are birth years
- any age outside the valid range is set to NaN

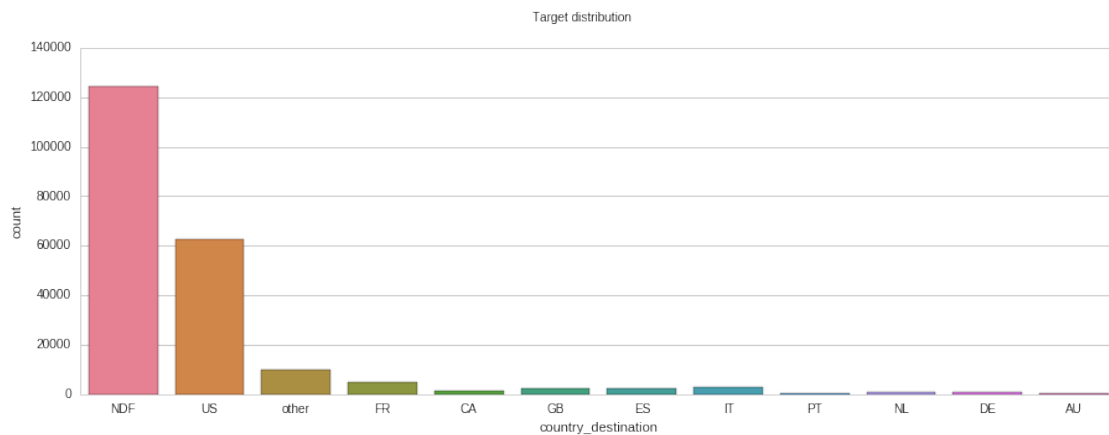
What is the correlation between users not filling their ages and the destination? Let's have a look.

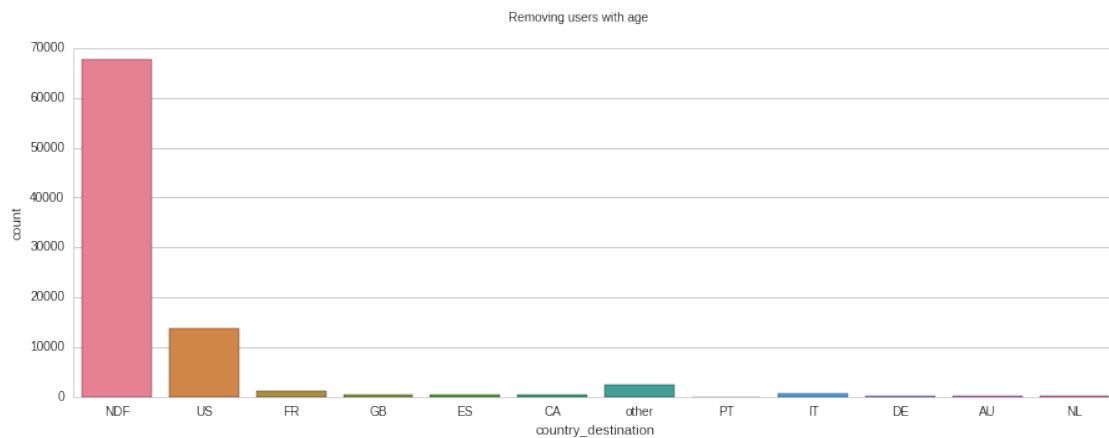
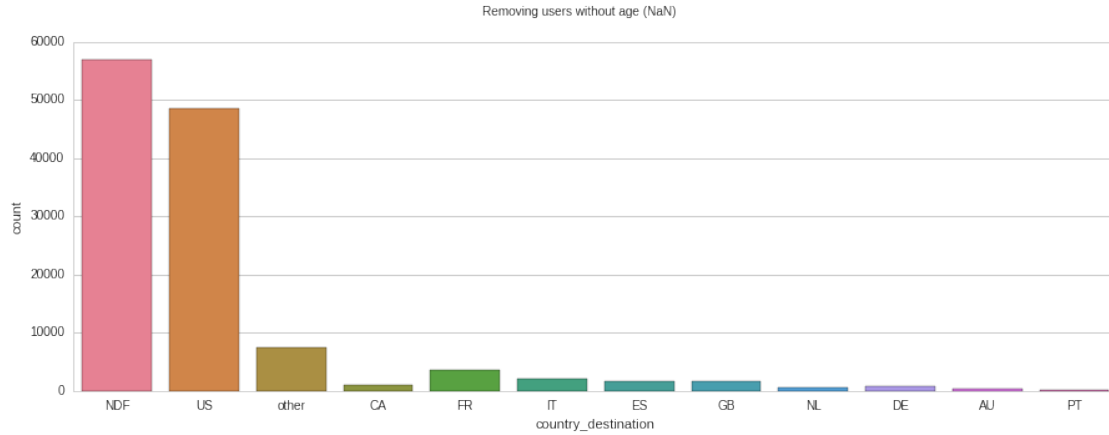
```
In [151]: # Without changing the target data
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Target distribution')
sns.countplot(x='country_destination', data=train_data, palette="husl", ax=axis1)

# Removing users without age
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Removing users without age (NaN)')
sns.countplot(x='country_destination', data=train_data[train_data['age'].notnull()], palette="husl", ax=axis1)

# Removing users with age
fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Removing users with age')
sns.countplot(x='country_destination', data=train_data[train_data['age'].isnull()], palette="husl", ax=axis1)
```

Out[151]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f290985fa10>





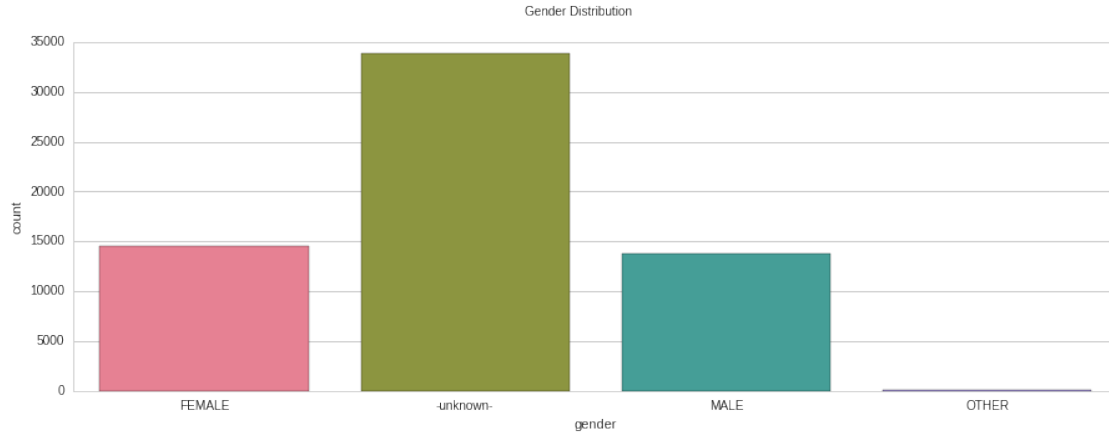
From the plots above, the data suggests that only 25% of users with missing ages book while 55% of users with age book a trip. **NDF** stands for No Destination Found.

## 1.4 Gender

```
In [153]: # Gender Exploration
gender = train_data['gender']
gender.value_counts(normalize=True) # How to fix unknown? and OTHER?

fig, (axis1) = plt.subplots(1,1,figsize=(15,5))
fig.suptitle('Gender Distribution')
sns.countplot(x='gender', data=test_data, palette="husl", ax=axis1)

Out[153]: <matplotlib.axes._subplots.AxesSubplot at 0x7f290b5ee310>
```

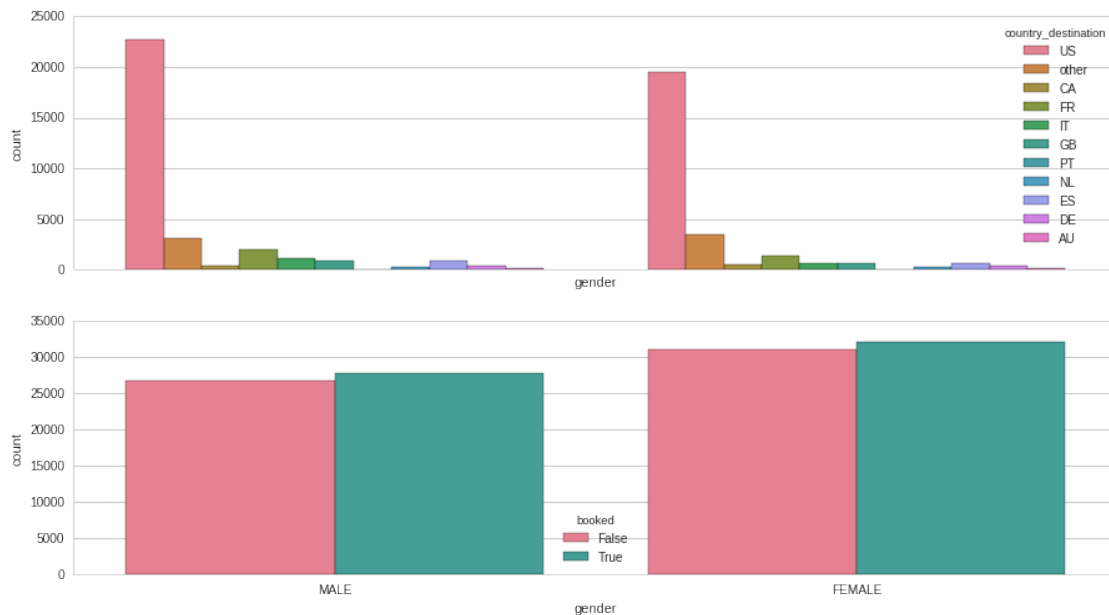


This column needs some cleaning by removing the unknown and other.

```
In [154]: # Keeping only
train_data_clean_gender = train_data.query('["MALE", "FEMALE"] in gender')
train_data_clean_gender.loc[:, 'booked'] = train_data["country_destination"].apply(lambda x :

fig, (axis1, axis2) = plt.subplots(2,1,sharex=True,figsize=(15,8))
sns.countplot(x="gender",hue="country_destination", data=train_data_clean_gender[train_data_c:
sns.countplot(x="gender",hue="booked", data=train_data_clean_gender, palette="husl", ax=axis2)
```

Out[154]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f2909925f90>



The data suggests that the gender has no impact on the country destination and if the first destination was booked.

```
In [169]: train_data['signup_method'].value_counts(normalize=True)
```

```
Out[169]: basic      0.716310
          facebook   0.281132
          google     0.002558
          Name: signup_method, dtype: float64
```

```
In [168]: train_data['first_device_type'].value_counts(normalize=True)
```

```
Out[168]: Mac Desktop      0.419768
          Windows Desktop  0.340668
          iPhone           0.097254
          iPad             0.067177
          Other/Unknown    0.049974
          Android Phone    0.013132
          Android Tablet   0.006053
          Desktop (Other)  0.005617
          SmartPhone (Other) 0.000356
          Name: first_device_type, dtype: float64
```

```
In [171]: train_data['first_browser'].value_counts(normalize=True)
```

```
Out[171]: Chrome      0.299108
          Safari       0.211613
          Firefox      0.157671
          -unknown-    0.127739
          IE           0.098702
          Mobile Safari 0.090297
          Chrome Mobile 0.005950
          Android Browser 0.003987
          AOL Explorer  0.001148
          Opera         0.000881
          Silk          0.000581
          Chromium      0.000342
          BlackBerry Browser 0.000248
          Maxthon       0.000216
          IE Mobile    0.000169
          Apple Mail    0.000169
          Sogou Explorer 0.000155
          Mobile Firefox 0.000141
          SiteKiosk     0.000112
          RockMelt      0.000112
          Iron          0.000080
          IceWeasel     0.000061
          Pale Moon     0.000056
          Yandex.Browser 0.000052
          SeaMonkey     0.000052
          CometBird     0.000052
          Camino        0.000042
          TenFourFox    0.000037
          wOSBrowser    0.000028
          CoolNovo      0.000028
          Avant Browser  0.000019
          Opera Mini    0.000019
          Mozilla       0.000014
          Comodo Dragon  0.000009
          Flock         0.000009
```

Opera Mobile	0.000009
SlimBrowser	0.000009
OmniWeb	0.000009
Crazy Browser	0.000009
TheWorld Browser	0.000009
PS Vita browser	0.000005
Googlebot	0.000005
IceDragon	0.000005
Stainless	0.000005
Conkeror	0.000005
Outlook 2007	0.000005
Palm Pre web browser	0.000005
Kindle Browser	0.000005
Epic	0.000005
Google Earth	0.000005
Arora	0.000005
NetNewsWire	0.000005

Name: first\_browser, dtype: float64

Some values are not useful if the frequency is too low. A cleaning step is to remove/change the value if the frequency is under a threshold.

```
In [190]: def clean_under_threshold(df, column, f, freq_threshold=.001):
            frequencies = df[column].value_counts(normalize=True)
            return apply_on(df, column, lambda x: f(x) if frequencies[x] < freq_threshold else x)

clean_data_first_browser = clean_under_threshold(train_data, 'first_browser', lambda x: 'OHTER' if x == 'OHTER' else x)
print(clean_data_first_browser['first_browser'].value_counts(normalize=True))

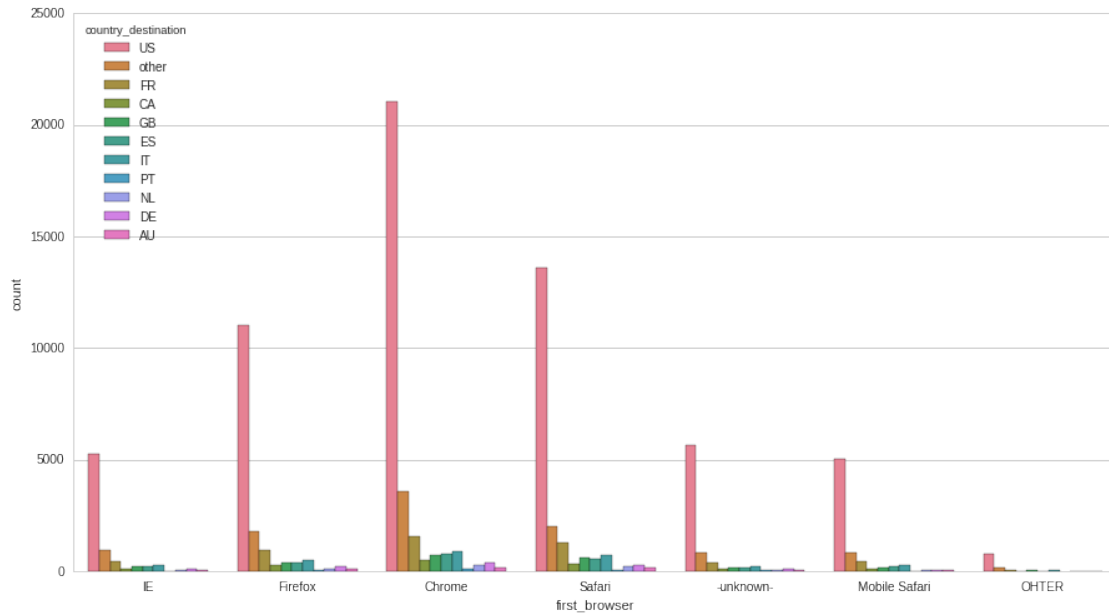
# Plot the first browsers
fig, (axis1) = plt.subplots(1,1,sharex=True,figsize=(15,8))
sns.countplot(x="first_browser",hue="country_destination", data=clean_data_first_browser[clean_data_first_browser['first_browser'] != 'OHTER'])
```

Chrome	0.299108
Safari	0.211613
Firefox	0.157671
-unknown-	0.127739
IE	0.098702
Mobile Safari	0.090297
OHTER	0.014870

Name: first\_browser, dtype: float64

Out[190]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f2903fa7e50>





```
In [188]: print(train_data['signup_app'].value_counts(normalize=True))
```

```
# Plot the signup app
```

```
fig, (axis1) = plt.subplots(1,1,sharex=True,figsize=(15,8))
```

```
sns.countplot(x="signup_app",hue="country_destination", data=train_data[train_data['country_d
```

```
Web          0.856014
```

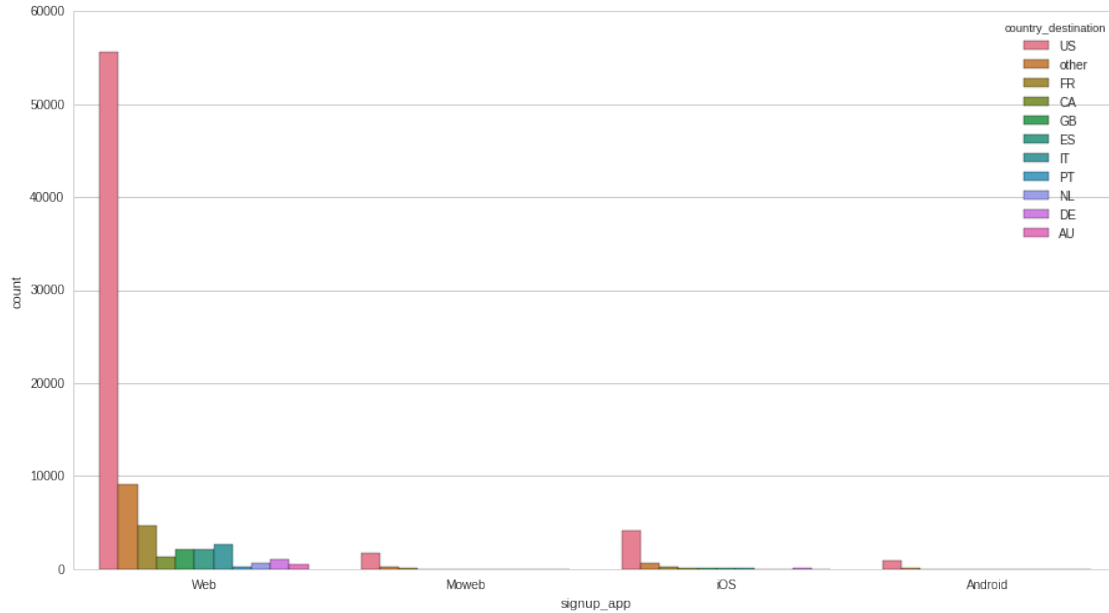
```
iOS          0.089102
```

```
Moweb        0.029332
```

```
Android      0.025552
```

```
Name: signup_app, dtype: float64
```

```
Out[188]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2903efab90>
```



## 1.5 Algorithms and Techniques

- The trained model will return probabilities for each country destination given a new unseen datapoint. The 5 most likely classes will be used to make a prediction in that order.
- The datasets provided by Kaggle are quite small (they fit in RAM) which makes model training and evaluation simpler since I do not need to use distributed computing or large EC2 instances to handle it.
- Computation will be done on my laptop (at least at the beginning) so that I can iterate faster on the core algorithms and data processing. Later, when hyperparameter tuning is performed, I might need to run it in on an other machine to get it done faster.

With small datasets, batch training can be done and many algorithms can be tested and compared

- SVMs
- Decision Trees
- Naive Bayes
- ...

### 1.5.1 Cleaning and missing values

The age feature is very messy and some cleaning is needed to make use of it.

- The outliers will be removed,
- The years will be transformed in ages,
- The missing values will be replaced with the collection's mean.

### 1.5.2 Feature Engineering

The date features need to be transformed to numbers. Some feature engineering is needed to make that transformation. We may want to keep or engineer the following

- Year
- Month
- Day
- Weekday
- Season
- ...

The first step is to use the train\_users.csv dataset only. If needed, I'll make use of the sessions.csv and the other csv files.

### 1.5.3 Data Transformation

Some data transformation will be needed

- Dimensionality reduction (with Principal Component Analysis): The training set is not large and the curse of dimensionality can be the root cause for bad performances.
- Data normalization: Some algorithms I want to try require data normalization (SVM and Naive Bayes for instance).

### 1.5.4 Boosting methods

Some boosting methods will be used to get better scores. I am planning on trying

- Random Forests (Adaboost or Bagging),
- Extreme Gradient Boosting - xgboost - that usually works well on Kaggle competitions.

In [ ]:

## Methodology

### Data Processing

#### 1. Data Cleaning

Some data cleaning is needed prior to training a Machine Learning model. The age feature is cleaned with the following steps

- All the values outside the range [14, 90] are removed and replaced with the mean age value,
- All the years values are turned into years,

The other categorical features are cleaned with the following processing steps

- Remove or rename values that are not frequent (under a frequency threshold)

#### 2. Feature Engineering

- Age\_interval: encodes year values into buckets so that the continuous variable age takes fewer values.
- timestamp\_first\_active is turned into 4 additional features
  - first\_active\_year
  - first\_active\_month
  - first\_active\_day
  - first\_active\_season
- date\_account\_created is turned into 4 additional features
  - account\_created\_year
  - account\_created\_month
  - account\_created\_day
  - account\_created\_season
- One Hot Encoding is performed on the categorical features

In total, there are now 121 features on this training set. that can potentially be too many features and I will apply PCA to this dataset to lower the dimensionality of the feature space.

## Implementation

At a high level, the data processing pipeline looks like the following:

loading -> cleaning -> feature engineering -> data normalization -> data transformation  
data transformation -> model training and evaluation -> hyperparameter optimization

1. **Loading:** Loads the data in memory.
2. **Cleaning:** cleans up age feature and remove the ones that are under a frequency threshold. See `data_preparation.py`.

```
def apply_on(df, column, f):
    """Returns a new dataframe after applying f on column"""
    result = df.copy()
    result.loc[:, column] = df[column].apply(f)
    return result

def apply_under_threshold(df, column, f, freq_threshold=.001):
    """Applies F on values in a column if under a freq_threshold"""
    frequencies = df[column].value_counts(normalize=True)
    return apply_on(df, column,
                    lambda x: f(x) if frequencies[x] < freq_threshold else x)
```

3. **Feature Engineering:** Adds `age_interval` and derived date features. See `data_preparation.py`.

```
def feature_transform_date_account_created(df):
    df_result = apply_on(
        df, "date_account_created", compose(parse, str))
    df_result[DATE_ACCOUNT_CREATED_YEAR] = \
        df_result.date_account_created.apply(lambda x: x.year)
    df_result[DATE_ACCOUNT_CREATED_MONTH] = \
        df_result.date_account_created.apply(lambda x: x.month)
    df_result[DATE_ACCOUNT_CREATED_DAY] = \
        df_result.date_account_created.apply(lambda x: x.day)
    df_result[DATE_ACCOUNT_CREATED_SEASON] = \
        df_result.date_account_created.apply(
            lambda x: day_of_year_to_season(x.timetuple().tm_yday))
    return df_result
```

4. **Data Normalization:** Some algorithms (PCA, SVM, ...) require prior data normalization to work. I decided to apply data normalization with the `sklearn.StandardScaler()` object. See `data_transformation.py`.

```
def make_scaler(X):
    scaler = StandardScaler()
    scaler.fit(X)
    return scaler
```

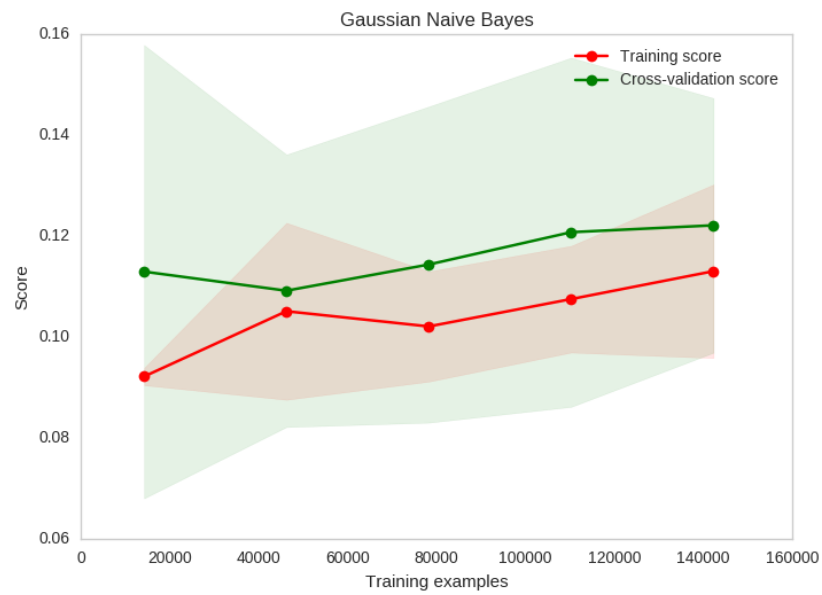
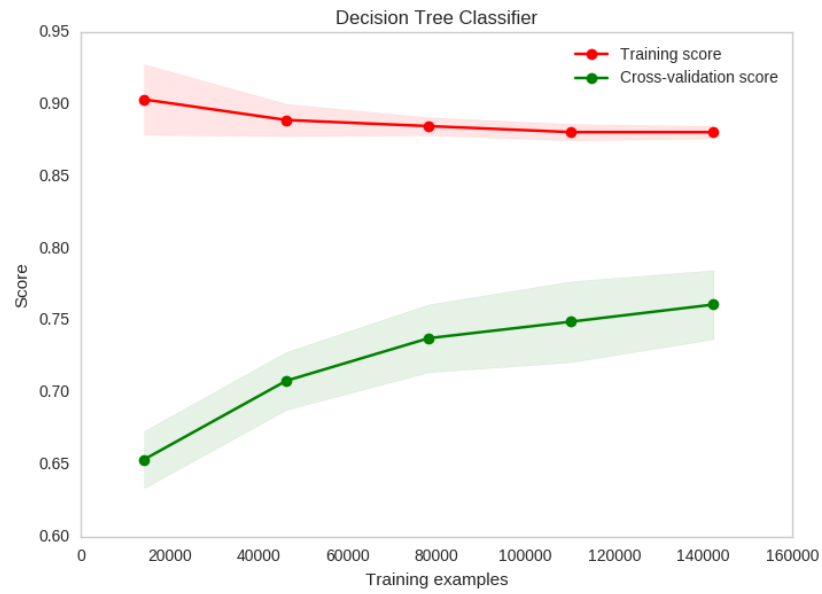
5. **Data Transformation:** there are a lot of features now and the curse of dimensionality tells us that we should only keep as few features as possible. PCA is used to project the feature space onto a smaller feature space (I decide to keep X% of the variance of the dataset). See `data_transformation.py`.

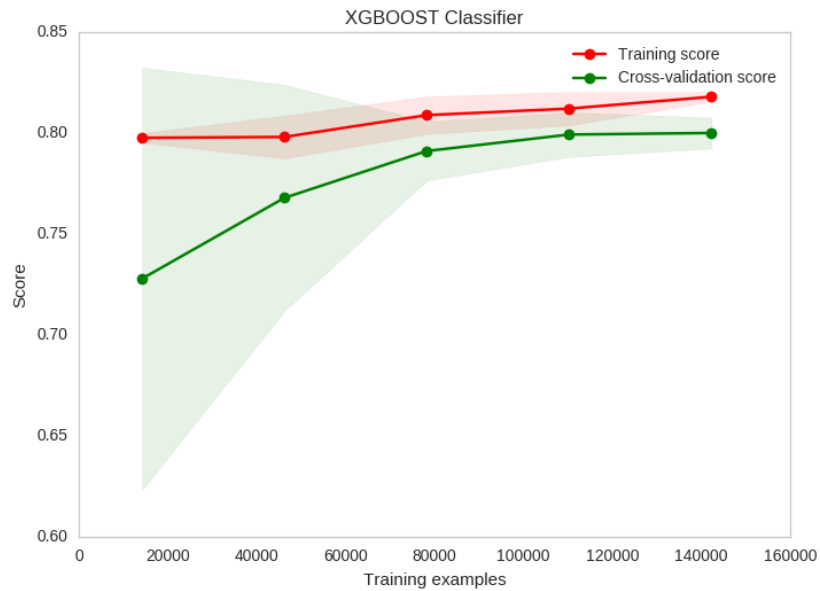
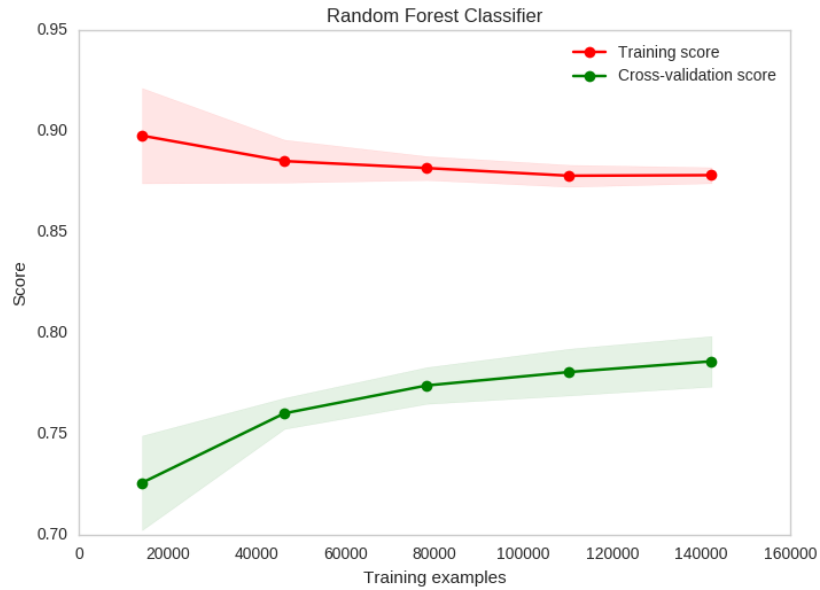
```
def make_transformer(X):
    pca = PCA(n_components=.94) # May vary to tune performance of the classifier
    pca.fit(X)
    return pca
```

I implemented the NDCG@5 score in Python so that I can use it for Cross Validation and Hyperparameter tuning.

## 6. Model Training and evaluation

The first thing to do is to try out some ML algorithms with default hyperparameters and plot their learning curves. In order to make the classifiers robust to noise, I perform 5 fold cross validation when plotting the learning curves. The mean and variance can be seen on the plots.





- The learning curves suggest that Naive Bayes perform really bad on this dataset. It is due to the fact that it assumes that each feature is independent from the others.
- Decision Tree and Random Forest seem to overfit but they are very promising. They will require some fine hyperparameter tuning to beat the sample



benchmark #2. It is already better than the sample benchmark #1.

- XGBoost (Extreme Gradient Boosting) has a really interesting learning curve: the training score and the cross validation score increase both with the training samples. It may require some hyperparameter tuning as well to perform better than the sample benchmark #2. It is also better than the sample benchmark #1.

## 7. Hyperparameter optimization (see hyperparameter\_tuning.py)

The final step is to fine tune the models selected from the previous step. Random Forest and XGBOOST seem to be the most promising ML algorithms on this dataset.

The hyperparameter space is very large. GridSearch will exhaust every possible combination of parameters in a combinatorial manner. It is too slow for my personal laptop. I will perform multiple passes of RandomizeSearchCV and keep reasonable values for the models. Also, RandomizedSearch should theoretically find faster good hyperparameters combinations.

```
def main(clf_name='rf'):

    (X_train, X_test, y_train, y_test) = datasets(scaling=False)

    print("Training classifier..." + clf_name)
    if clf_name == 'xgb':
        clf = XGBClassifier()
        parameters = PARAMETERS_XG
    else:
        clf = RandomForestClassifier(n_jobs=-1)
        parameters = PARAMETERS_RF

    # Hyperparameter tuning
    clf = hyperparameter_tuning(clf, parameters, X_train, y_train)

    print('Persisting classifier...' + clf_name)
    persist_clf(clf, clf_name + '-' + str(datetime.now()))

    # Score on test data
    best_clf = clf.best_estimator_
    score = score_classifier(best_clf, X_test, y_test)
    print("NDCG@5 score: {}".format(score))
    print("Persisting score...")
    persist_score(clf_name, best_clf, score)
```

I keep track of all the best hyperparameter combinations found by Randomized-SearchCV and run it in batches of 100 iterations.

## Refinement

The initial solution can be benchmark #2 that always predicts the top 5 countries sorted by frequencies. This does not use any ML algorithm but it is already a pretty good solution given the best scores obtained on that Kaggle Competition -  $\text{NDCG@5}=0.85670$

The first refinement is to use a ML algorithm instead of the naive and static initial solution. Using an XGBOOST classifier with default hyperparameters and  $n=50$  estimators yields  **$\text{NDCG@5}=0.85814$**  on the Kaggle LeaderBoard.

The next refinement is to fine tune the XGBOOST classifier using hyperparameter optimization and RandomizedGridSearch. It yields  **$\text{NDCG@5}=0.86681$**  on the Kaggle LeaderBoard

The next refinement steps can be

- tuning feature scaling and feature transformation: for xgboost, the PCA with  $n\_components=.94$  might be too small to improve performance. Also, the scaling step is not necessary for xgboost or random forests.
- adding additional features using **sessions.csv**.

## Results

### Model Evaluation and Validation

Q: Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?

The final model is built using extreme gradient boosting, XGBOOST classifier, which is an ensemble method. The final parameters of the model are found with a RandomizedSearch on the hyperparameter space.

- ESTIMATORS\_XG = 500
- OBJECTIVE\_XG = 'multi:softprob'
- DEPTH\_XG = 6
- LEARNING\_RATE\_XG = 0.3
- SUBSAMPLE\_XG = 0.5
- COLSAMPLE\_BYTREE\_XG = 0.5

Having a large number of estimators makes a lot of sense: It makes the model more robust by combining votes of weak classifiers. The more the better (too many estimators decreases prediction time since each weak classifier needs to vote). The objective function is **softprob**. It is used for multiclass classification. Probabilities for each classes are needed for the evaluation metrics NDCG@5 (That's why softmax is not used here). The Learning rate is 30% which is a common value among practitioners. The max\_depth parameter is 6. It can control the model complexity and the bias/variance tradeoff. The parameters found via hyperparameter tuning are relevant.

Q: Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?

Cross validation has been used to evaluate model performance. In other words, the model generalizes well on unseen data and does not overfit. 10 fold cross validation is used for hyperparameter tuning. And then all the models found are tested on a separate test set.

Q: Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?

The model robustness is mostly due to the Ensemble and Boosting method. To check whether or not a my model is robust to small changes in the training data I can randomly drop N (small N) datapoints in the training set and measure how far away the performances of the new model are.

Q: Can results found from the model be trusted?

The results found from the model can be trusted. This is confirmed by the submission made on Kaggle with the model predictions.

An ensemble method is used to boost classifier performance and also to make it more resistant to noise and small perturbations (more robust). This can be seen on the learning curves (the standard deviation is very small for ensemble methods: Random Forest and XGBOOST). Cross validation scores and scores obtained from Kaggle are very similar. In other words, the method used to improve and refine the models is relevant. The results predicted by the model can be trusted.

In practice, ensemble methods and their classifiers can be distributed easily across a cluster of machines. And since predicting where a new user is likely to book a first trip does not need real time prediction, the network overhead to make predictions on unseen datapoints is more than fine.

### Justification

Q: Are the final results found stronger than the benchmark result reported earlier?

- benchmark #1:  $NDCG@5=0.68411$
- benchmark #2:  $NDCG@5=0.85670$
- best entry:  $NDCG@5=0.88697$
- My best score  $NDCG@5=0.86681$

The final results found are stronger than the 2 benchmark result reported earlier. The final model is aligned with the scope of this project.

Q: Have you thoroughly analyzed and discussed the final solution?

To see how much the solution is improved, let's compute some scores:

$$\frac{|NDCG@5_{benchmark_1} - NDCG@5_{bestentry}|}{NDCG@5_{bestentry}} = 23.3\% \quad (3)$$

$$\frac{|NDCG@5_{benchmark_2} - NDCG@5_{bestentry}|}{NDCG@5_{bestentry}} = 3.5\% \quad (4)$$

$$\frac{|NDCG@5_{mybestentry} - NDCG@5_{bestentry}|}{NDCG@5_{bestentry}} = 2.3\% \quad (5)$$

As the scores suggest, the first benchmark is really far (23.3%) from the optimal solution submitted by the winner of the competition. The second benchmark is already very close to the optimal solution (3.5%). My final solution improves the second further. It is only 2.3% away from the best entry on the leaderboard.

Q: Is the final solution significant enough to have solved the problem?

Obviously, there is always room for improvement. I will consider that the final solution is significant enough to have solved the problem.

## Conclusion

### Free-From Visualization

Please refer to visualizations made in analysis and the learning curves.

### Reflection

A kaggle competition is an effective way of getting to work on real world problems and with other data scientists (Kaggle forums and community).

In this challenge, I was given a list of users along with their demographics, web session records, and some summary statistics. I was asked to predict which country a new user's first booking destination will be.

The initial step is to understand the metrics that we are optimizing for. In this challenge, the NDCG@5 or **Normalized discounted cumulative gain** measures the performance of a recommendation system based on the graded relevance of the recommended entities. Our models can output up to 5 predictions but the order of the predictions matters.

The second step was to get some insight at the data (Analysis part). The features were explored and plotted in order to understand if some preprocessing and cleaning was needed. Some features required some cleaning (eg. age feature).

Then, some feature engineering work was achieved in order to make use of the dates and timestamps. One Hot Encoding is also performed in order to make the data usable by the different ML algorithms (xgboost takes only numerical values, random forest can handle categorical features, ...)

The next step was to try different ML algorithms to see how well they could perform on the dataset. Some learning curves for different ML algorithms were plotted and discussed. Naive Bayes perform poorly whereas Decision Trees and xgboost were able to consistently learn and generalize on unseen data.

After deciding which ML algorithms were the best fit for the dataset, hyperparameter tuning was performed to improve their performances. Ensemble methods were used to make the models more resistant to small perturbations in the training set.

- Random Forest yields a NDCG@5=TODO
- XGBOOST yields a NDCG@5=TODO

### Improvement

There is a lot of room for improvement for my models. Some that seem the most important to me are listed below.

- A naive approach for imputing missing values for the age features is used. Instead of using the mean, one could design a supervised ML model that predicts the age given the other features.
- Some extra features from the file sessions.csv can be added to the training and testing set. That will require some feature engineering.
- If I had to start over a Kaggle competition I would probably invest more time in automating tasks like: feature engineering, model training, model submission through their API so that I would keep track of all the model performances and parameters and iterate faster.