# Airbnb New User Bookings

Prepared by
**Nam Pham**
**Sudip Baral,**
**Dibakar Barua,**
**Ruturaj Joshi,**
**Jiajun Du**

November 26, 2017

**Project Description:**

In this project, we are given a training dataset of 213451 users along with 16 features including their demographics, web session record, and some summary statistics. A testing dataset of 62096 users with 15 features except country_destination feature. We are required to predict which country a new user's first booking destination. All the users in this dataset are from the USA. The training and test sets are split by dates. In the test set, you will predict all the new users with first activities after **7/1/2014.**

**Approach:**

- We thought that it is essential to explore the data sets before going to built the model to make sure there are no missing data unprocessed.

- Processing the data so that it would be able to used by variety classifier.

- Applying variety classifier to predict, comparing the result, then choosing the best classifier.

**DATA EXPLORATION**

The goals of this step is to figure out and solve the following problem:

- is there any mistakes in the data?

- fixing the data so that it could look more realistic.

In [417]:
```
# Read the data into DataFrams
train_users = pd.read_csv('train_users_2.csv')
test_users = pd.read_csv('test_users.csv')
```

In [418]:
```
# checking the training test
train_users.head()
```

Out[418]:

| | 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 |

The data seems to be included missing data and unusable format.

**MISSING DATA**

In the gender column some values being -unknown- . We need to transform it to NaN so that later on we could use pandas built in function to process the data.

```
In [422]: # Deal with Missing Data: Convert -unkown- in "gender" into NaN
          train_users.gender.replace('-unknown-', np.nan, inplace=True)
          test_users.gender.replace('-unknown-', np.nan, inplace=True)
```

Compute the percentage of missing data in each column in order to exclude those column with maximum missing data rate.

```
In [424]: # The percentage of missing data in each colum in training data
          print(users_nan)

          id                          0.000000
          date_account_created        0.000000
          timestamp_first_active      0.000000
          date_first_booking         58.347349
          gender                     44.829024
          age                        41.222576
          signup_method               0.000000
          signup_flow                 0.000000
          language                    0.000000
          affiliate_channel           0.000000
          affiliate_provider          0.000000
          first_affiliate_tracked     2.841402
          signup_app                  0.000000
          first_device_type           0.000000
          first_browser               0.000000
          country_destination         0.000000
          dtype: float64
```

The feature date_first_booking has 58.35% of NaN or missing data, so to maximine our performing of our classifiers we will build we better not include this feature at the modeling part.

According to the statistics, the other features we could explore more are gender and age.

**AGE:**

In [426]:
```
train_users.age.describe()
```

Out[426]:
```
count    125461.000000
mean         49.668335
std         155.666612
min           1.000000
25%          28.000000
50%          34.000000
75%          43.000000
max        2014.000000
Name: age, dtype: float64
```

In [427]:
```
test_users.age.describe()
```

Out[427]:
```
count    33220.000000
mean        37.616677
std         74.440647
min          1.000000
25%         26.000000
50%         31.000000
75%         40.000000
max       2002.000000
Name: age, dtype: float64
```

**The statictis in both data sets shows the inconsistency in the age of the users.**

In [428]:
```
print("Total number of users over 90 year olds is:",
      sum(test_users.age > 90)+sum(train_users.age > 90))
print("Total number of users less than 13 year olds is:",
      sum(test_users.age < 13)+sum(train_users.age < 13))
```

```
Total number of users over 90 year olds is: 2928
Total number of users less than 13 year olds is: 59
```

So, we need to fix the data look like more realistic by set all the users whose age are over 90 and less than 13 to NaN.

In [429]:
```
# set the ouliers of age to NaN
train_users.loc[train_users.age > 90, 'age'] = np.nan
test_users.loc[test_users.age > 90, 'age'] = np.nan
train_users.loc[train_users.age < 13, 'age'] = np.nan
test_users.loc[test_users.age < 13, 'age'] = np.nan
```

From now we could able to merge both data set into one 'users' dataset in other to visualizing the data.

In [430]:
```
# Merge train and test users
users = pd.concat((train_users, test_users), axis=0,
                  ignore_index=True)
# Remove ID's
users.drop('id',axis=1, inplace=True)
users.shape
```

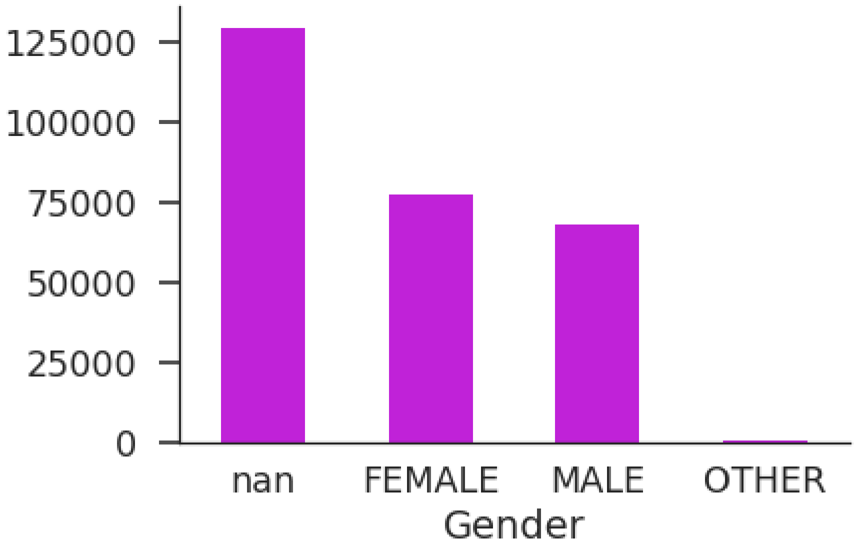
Out[430]: (275547, 15)

## VISUALIZING THE DATA

Visualization able us to see the outliers and error immediately.

### Gender

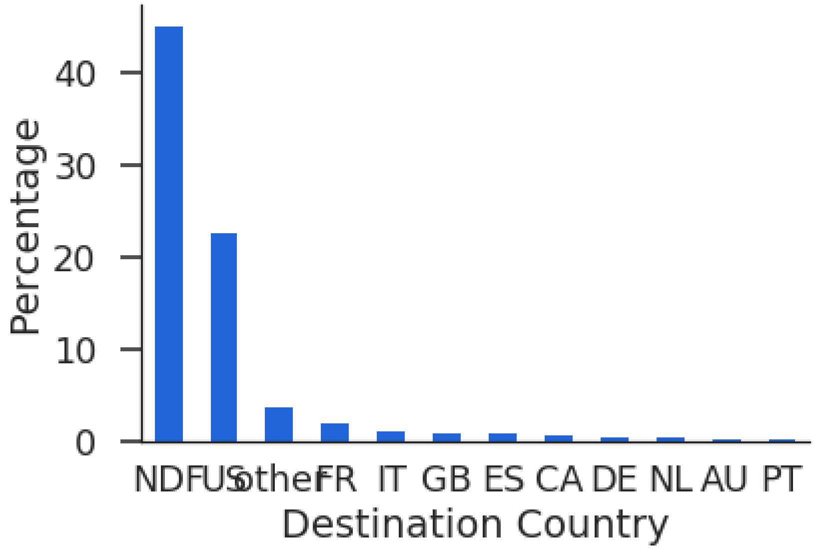
In [432]:
```
users.gender.value_counts(dropna=False).plot(kind='bar', color=
plt.xlabel('Gender')
sns.despine()
```




**The ammount of missing data is too many and there is a slight difference between user gender.**

### Destination

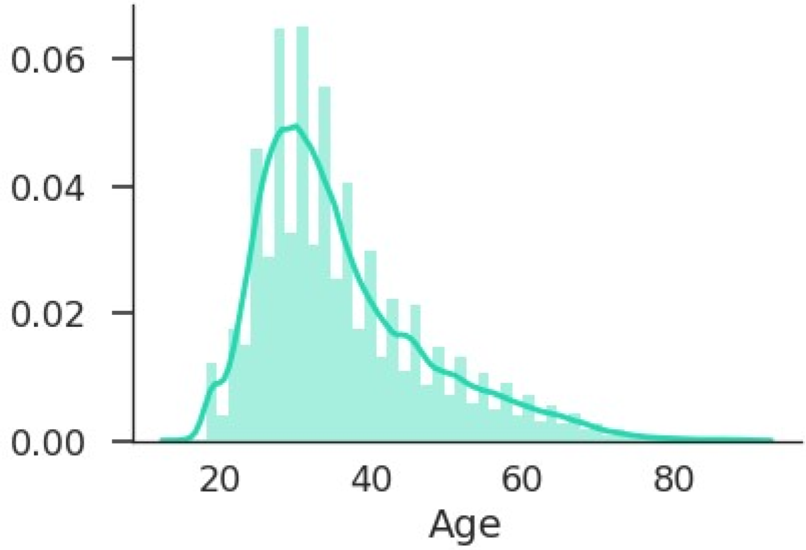
```
In [433]: destination_percentage = users.country_destination.value_counts
          destination_percentage.plot(kind='bar',color='#2265d8', rot=0)
          plt.xlabel('Destination Country')
          plt.ylabel('Percentage')
          sns.despine()
```




Looking at the bar graph we see that more than 40% users not booking. For those who booked via airbnb the US is the most chose, more than sum of other destinations.

**Age**

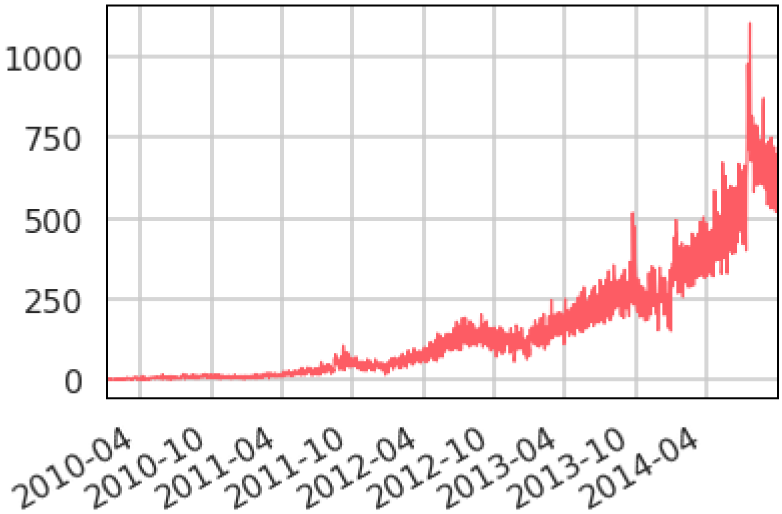
```
In [434]: sns.distplot(users.age.dropna(), color='#22d8b1')
          plt.xlabel('Age')
          sns.despine()
```



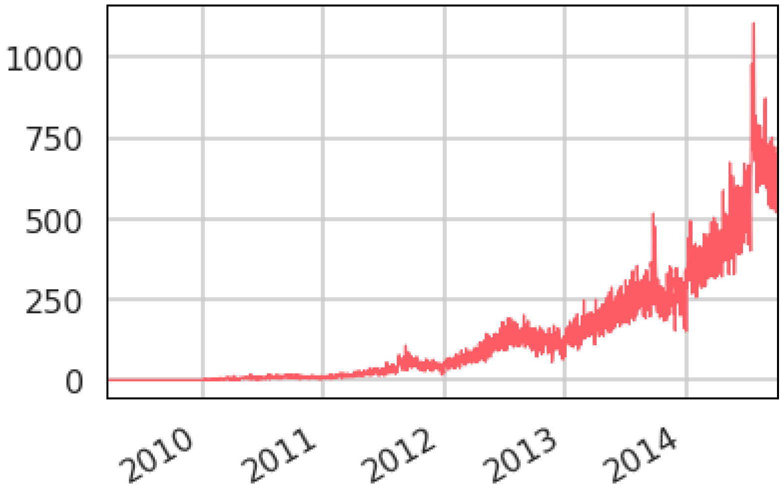

Based on the graph, the common age to travel is between 25 and 40.

**DATE:**

```
In [436]: sns.set_style("whitegrid", {'axes.edgecolor': '0'})
          sns.set_context("poster", font_scale=1.0)
          users.date_account_created.value_counts().plot(kind='line', line
          plt.show()
```

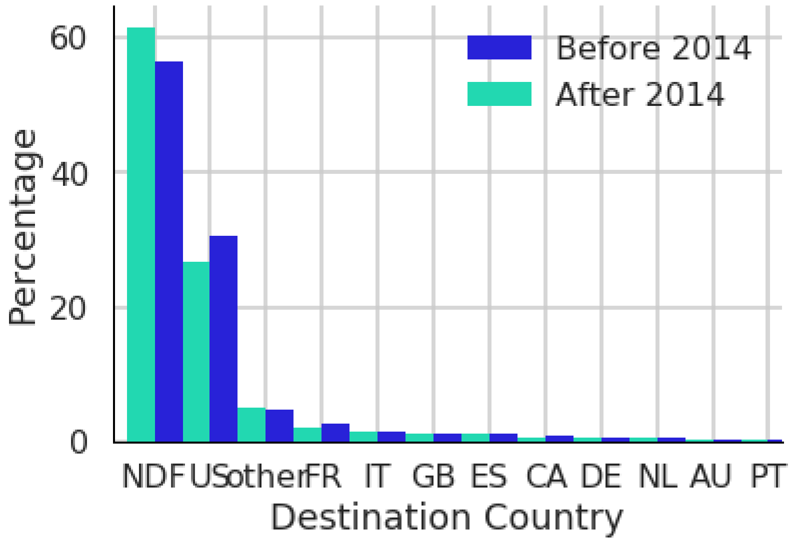


In [437]:
```
users.date_first_active.value_counts().plot(kind='line', linewi
plt.show()
```




As usual, users who create accounts and activate the accounts almost at the same time.

```
after = sum(users.loc[users['date_first_active'] > date, 'count
after_destinations = users.loc[users['date_first_active'] > dat
                               'country_destination'].value_cou
after_destinations.plot(kind='bar',  color='#22d8b1', position=

plt.legend()
plt.xlabel('Destination Country')
plt.ylabel('Percentage')
sns.despine()
plt.show()
```




Look at the graph we see that more new users after 2014 but they booked less than the previous.

**PREDICTION:**

Our approaches are to use different classifier to built the predict model and to make prediction. After comparing the accuracy will we chose the best classifier that provide the most accuracy.

Before building our predict model we need to drop some column features that have peculiar behavior or those will make the predict model less accuracy.

## Predict

```
In [440]: from scipy import stats
          from sklearn.naive_bayes import GaussianNB
          from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import accuracy_score
          from sklearn.ensemble import RandomForestClassifier
          # Create label and id_test vector
          labels = train_users['country_destination']
          id_test = test_users['id']
          # train_users = train_users.drop(['country_destination'], axis=
          end_train = train_users.shape[0]
```

```
In [443]: # Remove 'country_destination'
          users.drop('country_destination',axis=1, inplace=True)
```

```
In [446]: # Remove 'date_first_booking'
          users.drop('date_account_created',axis=1, inplace=True)
```

```
In [448]: users.drop('date_first_booking',axis=1, inplace=True)
```

```
In [449]: list(users)

Out[449]: ['affiliate_channel',
           'affiliate_provider',
           'age',
           'first_affiliate_tracked',
           'first_browser',
           'first_device_type',
           'gender',
           'language',
           'signup_app',
           'signup_flow',
           'signup_method',
           'timestamp_first_active',
           'date_first_active']
```

**for "NaN", find the value that is most common in the column it is found and replace the "NaN" with it**

In [456]:
```
categorical_features = [
    'affiliate_channel',
    'affiliate_provider',
    'first_affiliate_tracked',
    'first_browser',
    'first_device_type',
    'gender',
    'language',
    'signup_app',
    'signup_method'
]
fcc_list = len(categorical_features)
from collections import Counter
print("The most common in the column")
for i in range(fcc_list):
    lst = users[categorical_features[i]]
    data = Counter(lst)
    most_common = max(lst, key=data.get)
    print("most_common:",most_common)
    users[categorical_features[i]].replace(" NaN", most_common,
```

```
The most common in the column
most_common: direct
most_common: direct
most_common: untracked
```

In [457]:
```
# Replace NaN in age by the average of its column
users.age.fillna(users.age.mean(), inplace =True)
```

In [458]:
```
users.head()
```

Out[458]:

| | affiliate_channel | affiliate_provider | age | first_affiliate_tracked | first_brows |
|---|---|---|---|---|---|
| **0** | direct | direct | 35.963871 | untracked | Chro |
| **1** | seo | google | 38.000000 | untracked | Chro |
| **2** | direct | direct | 56.000000 | untracked | |
| **3** | direct | direct | 42.000000 | untracked | Fire |
| **4** | direct | direct | 41.000000 | untracked | Chro |

Using One-hot-encoding to transform categorical features in to dummy data so that we could able to use the data for many classifiers.

In [459]:
```
#One-hot-encoding features
for f in categorical_features:
    users_dummy = pd.get_dummies(users[f], prefix=f)
    users = users.drop([f], axis=1)
    users = pd.concat((users, users_dummy), axis=1)
```

Divide the training data into a training and testing set

In [468]:
```
# Divide the training data into a training and test set
from sklearn.model_selection import train_test_split
# y = users['country_destination']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

From now we apply different classifier to build the predict model and make prediction.

## Using Random Forest Classifier to predict

In [470]:
```
my_RandomForest = RandomForestClassifier(n_estimators = 19, boo
my_RandomForest.fit(X_train, y_train)

y_training = my_RandomForest.predict(X_train)
acc_train = accuracy_score(y_train, y_training)

y_predict = my_RandomForest.predict(X_test)
acc_test = accuracy_score(y_test, y_predict)

print("Accuracy on training data:", acc_train)
print("Accuracy on test data:", acc_test)
```

```
Accuracy on training data: 0.730665958101
Accuracy on test data: 0.59776544244
```

### Using Logistic Regression Classifier to predict

In [405]:
```
my_logReg = LogisticRegression()
my_logReg.fit(X_train, y_train)
acc_train = my_logReg.score(X_train, y_train)
acc_test = my_logReg.score(X_test, y_test)

print("Accuracy on training data:", acc_train)
print("Accuracy on test data:", acc_test)
```

```
Accuracy on training data: 0.602229183565
Accuracy on test data: 0.603813228467
```

### Using Gaussian Naive Bayes to predict

In [471]:
```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
acc_train = gnb.score(X_train, y_train)
acc_test = gnb.score(X_test, y_test)

print("Accuracy on training data:", acc_train)
print("Accuracy on test data:", acc_test)
```

```
Accuracy on training data: 0.00553100439124
Accuracy on test data: 0.00515339513622
```

### Using Decition Tree Classfier to predict

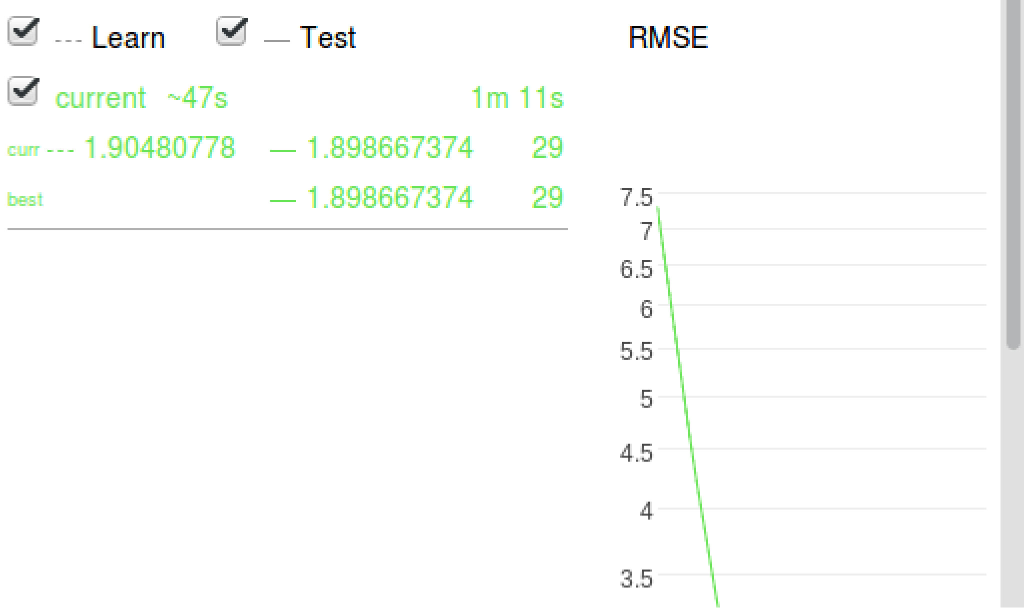
In [472]:
```
# we can simply use decision tree to clarify our prediction
from sklearn.tree import DecisionTreeClassifier
decisiontree = DecisionTreeClassifier()
# fitting the model
decisiontree.fit(X_train, y_train)
# predict the response
y_predict = decisiontree.predict(X_test)
y_training = decisiontree.predict(X_train)

acc_train= accuracy_score(y_train, y_training)
acc_test= accuracy_score(y_test, y_predict)

print("Accuracy on training data:", acc_train)
print("Accuracy on test data:", acc_test)
```

```
Accuracy on training data: 0.733099320337
Accuracy on test data: 0.586408097787
```

## Using Gradient Boosting Algorithms

### Using Catboost

In [414]:
```
from catboost import CatBoostRegressor
model=CatBoostRegressor(iterations=50, depth=3,
                        learning_rate=0.1,
                        loss_function='RMSE')
categorical_features_indices = np.where(
    users.dtypes != np.float)[0]
model.fit(X_train, y_train,
          cat_features=categorical_features_indices,
          eval_set=(X_test, y_test),plot=True)
```




**Out of the classifiers we have used, Catboost Classifier give us the most accuracy: 100 – rmse*rmse = 100 – 1.898667374* 1.898667374 = 96.39506220291**