Data Pre-processing:

Imported all different libraries:

In [1]:
```
import pandas as pd
import numpy as  np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pickle
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, Model
from keras.layers import Activation, Dense, Dropout
from sklearn.preprocessing import LabelBinarizer
import sklearn.datasets as skds
from pathlib import Path
import itertools
from sklearn.metrics import confusion_matrix
```

```
Using TensorFlow backend.
```

Read required input files:

In [2]:
```
train_users = pd.read_csv("train_users_2.csv")
test_users = pd.read_csv("test_users.csv")
```

Concatenated both training and testing data:

In [4]:
```
df = pd.concat((train_users, test_users), axis = 0, ignore_index = True, sort = True)
```

Head of data:

In [5]:
```
df.head()
```

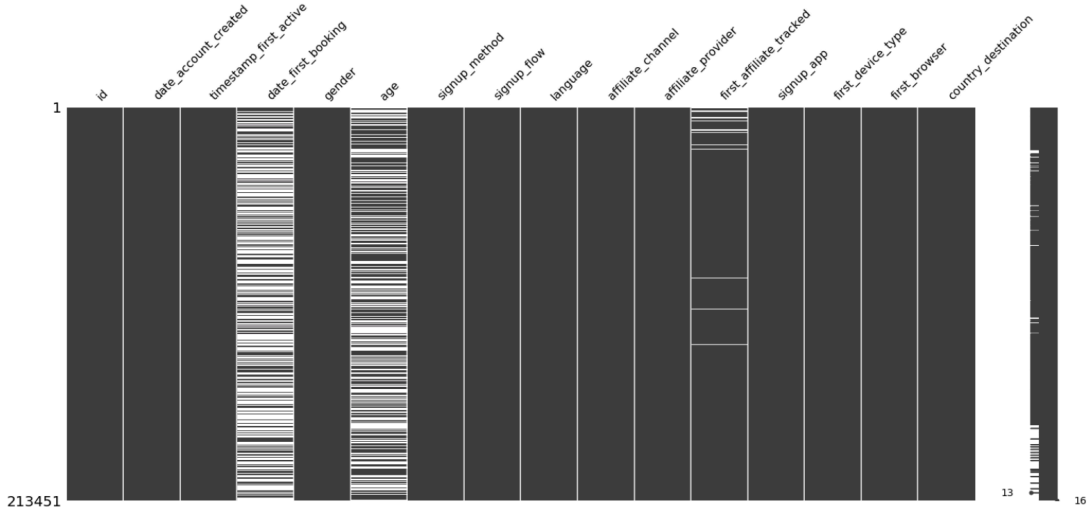
Out[5]:

affiliate_provider	age	country_destination	date_account_created	date_first_booking	first_affiliate_tracked	first_browser	first_de
direct	NaN	NDF	6/28/10	NaN	untracked	Chrome	Ma
google	38.0	NDF	5/25/11	NaN	untracked	Chrome	Ma
direct	56.0	US	9/28/10	8/2/10	untracked	IE	
direct	42.0	other	12/5/11	9/8/12	untracked	Firefox	Ma
direct	41.0	US	9/14/10	2/18/10	untracked	Chrome	Ma

Missing Values in data:

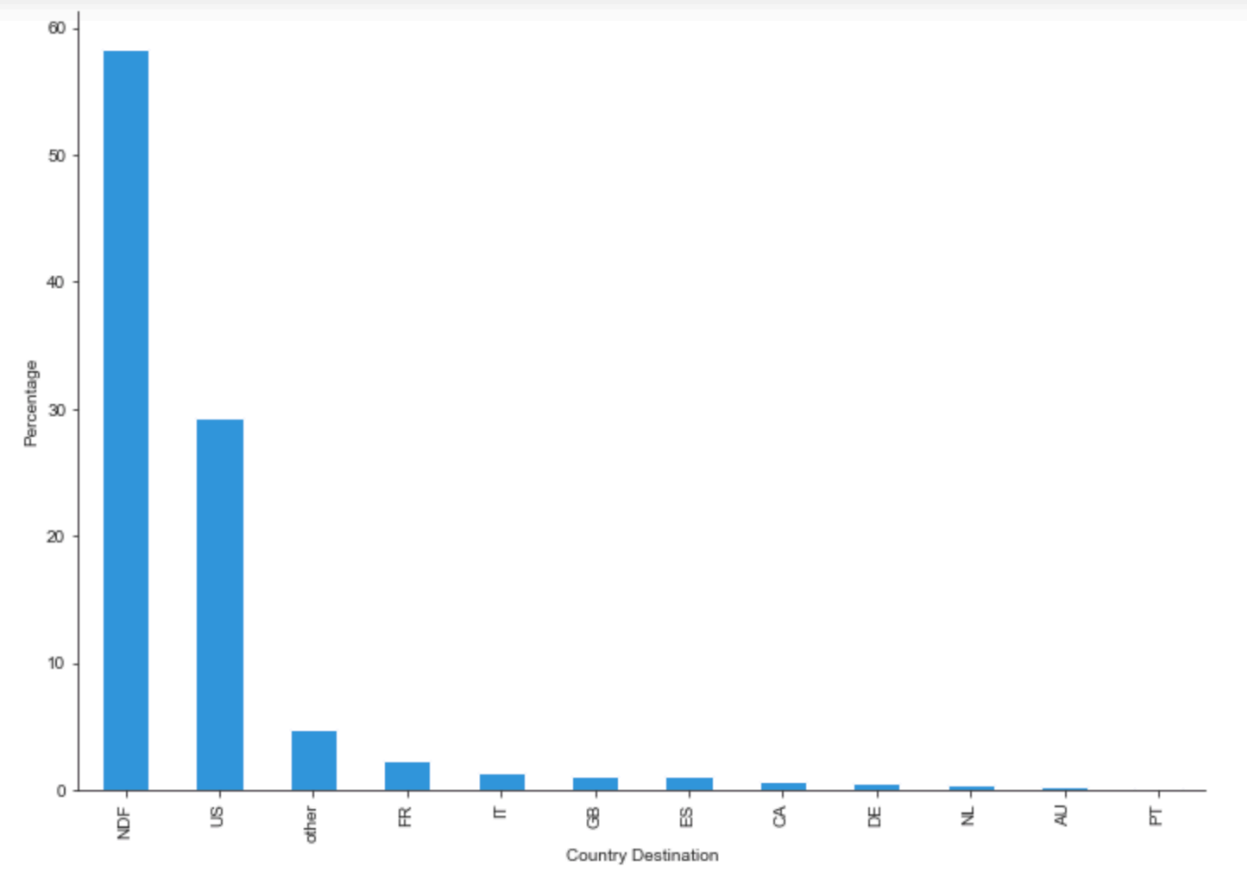
```
In [6]: import missingno as msno
        msno.matrix(train_users)
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1a31dcc470>
```



Distribution of our label which is country destination:

```
In [7]: sns.set_style('ticks')
        fig, ax = plt.subplots()
        fig.set_size_inches(11.7, 8.27)
        dest_per = train_users.country_destination.value_counts() / train_users.shape[0] * 100
        dest_per.plot(kind='bar',color='#3498DB')
        plt.xlabel('Country Destination')
        plt.ylabel('Percentage')
        sns.despine()
```



Count of gender value:

```
In [8]: df['gender'].value_counts()

Out[8]: -unknown-    129480
        FEMALE        77524
        MALE          68209
        OTHER           334
        Name: gender, dtype: int64
```

Count of first_browser value:

```
In [9]: df['first_browser'].value_counts()
```

```
Out[9]: Chrome                  78671
        Safari                  53302
        -unknown-               44394
        Firefox                 38665
        Mobile Safari           29636
        IE                      24744
        Chrome Mobile            3186
        Android Browser          1577
        AOL Explorer              254
        Opera                     228
        Silk                      172
        IE Mobile                 118
        BlackBerry Browser         89
        Chromium                   83
        Mobile Firefox             64
        Maxthon                    60
        Apple Mail                 45
        Sogou Explorer             43
        SiteKiosk                  27
        Iron                       24
        RockMelt                   24
        Yandex.Browser             14
        IceWeasel                  14
        Pale Moon                  13
        CometBird                  12
```

Replacing of age with median value:

```
In [10]: df['age'] = df['age'].fillna((df['age'].median()))
```

```
In [11]: df.head()
```

Out[11]:

	affiliate_channel	affiliate_provider	age	country_destination	date_account_created	date_first_booking	first_affiliate_tracked	f
0	direct	direct	33.0	NDF	6/28/10	NaN	untracked	
1	seo	google	38.0	NDF	5/25/11	NaN	untracked	
2	direct	direct	56.0	US	9/28/10	8/2/10	untracked	
3	direct	direct	42.0	other	12/5/11	9/8/12	untracked	
4	direct	direct	41.0	US	9/14/10	2/18/10	untracked	

Dropping ‘ date_first_booking’ column:

```
In [16]: df=df.drop('date_first_booking',1)
```

```
In [17]: from sklearn import preprocessing
         from sklearn.model_selection import cross_validate
         from sklearn import tree
         from sklearn.preprocessing import LabelBinarizer
         from sklearn.metrics import make_scorer
         df_label = df.loc[:,'country_destination']
         data = df.drop(['country_destination'], axis=1)
```

Coverting features into numerical:

Gender: As per the feedback the Gender column missing value is kept as –unknown- and then changed to numerical rather than replacing it with the most used value in this column.

```
In [18]: # Replacing unknown values in gender with -1 and null values with
         df.loc[ df['gender'] == '-unknown-', 'gender'] = -1
         df.loc[ df['gender'].isnull(), 'gender' ] = -1
```

Language:

```
In [23]: df['language'] = df['language'].astype('category')
         df['language'] = df['language'].cat.codes
```

affiliate_channel:

```
In [25]: # Encoding for affiliate_channel
         df['affiliate_channel'] = df['affiliate_channel'].astype('category')
         df['affiliate_channel'] = df['affiliate_channel'].cat.codes
```

affiliate_provider:

In [26]:
```
# Encoding for affiliate_provider
affiliate_provider_encoding = {'direct':1,
'google':2,
'other':3,
'craigslist':4,
'bing':5,
'facebook':6,
'vast':7,
'padmapper':8,
'facebook-open-graph':9,
'yahoo':10,
'gsp':11,
'meetup':12,
'email-marketing':13,
'naver':14,
'baidu':15,
'yandex':16,
'wayn':17,
'daum':18}

for data in [df]:
    data['affiliate_provider'] = data['affiliate_provider'].apply(lambda x: affiliate_provider_
```

first_affiliate_tracked:

In [27]:
```
# Encoding for first_affiliate_tracked
df['first_affiliate_tracked'] = df['first_affiliate_tracked'].astype('category')
df['first_affiliate_tracked'] = df['first_affiliate_tracked'].cat.codes
```

signup_app:

In [31]:
```
# Encoding for signup_app
signup_app_encoding = {'Web' : 1,
                       'iOS' : 2,
                       'Android' : 3,
                       'Moweb' : 4}
for data in [df]:
    data['signup_app'] = data['signup_app'].apply(lambda x: signup_app_encoding[x])
```

first_device_type:

In [32]:
```
# Encoding for first_device_type
df['first_device_type'] = df['first_device_type'].astype('category')
df['first_device_type'] = df['first_device_type'].cat.codes
```

first_browser:

```
In [33]: # Encoding for first_browser
first_browser_encoding = {'Chrome':1,
'Safari':2,
'Firefox':3,
'-unknown-':4,
'IE':5,
'Mobile Safari':6,
'Chrome Mobile':7,
'Android Browser':8,
'AOL Explorer':9,
'Opera':10,
'Silk':11,
'Chromium':12,
```

Date Preprocessing 1:

```
In [45]: # Date is converted to numerical
# new features in data
import numpy as  np
df['date_account_created'] = df['date_account_created'].astype(str)
df['date_account_created'] = df['date_account_created'].str.replace('\D', '').astype(int)


df['date_account_created'].head()
```

```
Out[45]: 0    62810
1    52511
2    92810
3    12511
4    91410
Name: date_account_created, dtype: int64
```

Date Preprocessing 2:

```
In [64]: data_wout_label['Month'] = data_wout_label.date_account_created.dt.month
In [65]: data_wout_label.head()
```

Out[65]:

ed	first_affiliate_tracked	first_browser	first_device_type	gender	language	signup_app	signup_flow	signup_method	timestamp_first_active	Year	Month
28	1	1	1	-1	1	1	0	0	20090319043255	2010	6
25	1	1	1	1	1	1	0	0	20090523174809	2011	5
28	1	5	3	0	1	1	3	2	20090609231247	2010	9
05	1	3	1	0	1	1	0	0	20091031060129	2011	12
14	1	1	1	-1	1	1	0	2	20091208061105	2010	9

```
In [66]: import calendar
         data_wout_label['Month'] = data_wout_label['Month'].apply(lambda x: calendar.month_abbr[x])
In [67]: data_wout_label.head()
```

Out[67]:

ed	first_affiliate_tracked	first_browser	first_device_type	gender	language	signup_app	signup_flow	signup_method	timestamp_first_active	Year	Month
28	1	1	1	-1	1	1	0	0	20090319043255	2010	Jun
25	1	1	1	1	1	1	0	0	20090523174809	2011	May
28	1	5	3	0	1	1	3	2	20090609231247	2010	Sep
05	1	3	1	0	1	1	0	0	20091031060129	2011	Dec
14	1	1	1	-1	1	1	0	2	20091208061105	2010	Sep

Dropped Id column as it has unique values:

```
In [39]: data = df.drop(['id'], axis=1)
         data.head()
```

Out[39]:

	affiliate_channel	affiliate_provider	age	country_destination	date_account_created	fir
0	2	1	33.0	NDF	6/28/10	
1	7	2	38.0	NDF	5/25/11	
2	2	1	56.0	US	9/28/10	
3	2	1	42.0	other	12/5/11	
4	2	1	41.0	US	9/14/10	

Converting Labels into numeric:

```
:  df_label = df_label.astype('category')
   df_label = df_label.cat.codes
```

Scaling Data:

In [57]:
```
#Scaling Data
from sklearn import preprocessing
data_scaled = pd.DataFrame(preprocessing.StandardScaler().fit_transform(data))
data_scaled.head(n=5)
```

Out[57]:

	0	1	2	3	4	5	6	7	8	9	
0	-0.530050	-0.457366	-0.075104	-0.537362	0.847061	-1.014921	-0.874795	-0.950224	-0.148954	-0.43807	-0.4
1	2.355723	0.275219	-0.029005	-0.538589	0.847061	-1.014921	-0.874795	1.486051	-0.148954	-0.43807	-0.4
2	-0.530050	-0.457366	0.136952	-0.533789	0.847061	0.985196	0.544437	0.267914	-0.148954	-0.43807	-0.1
3	-0.530050	-0.457366	0.007874	-0.543353	0.847061	-0.014862	-0.874795	0.267914	-0.148954	-0.43807	-0.4
4	-0.530050	-0.457366	-0.001346	-0.533956	0.847061	-1.014921	-0.874795	-0.950224	-0.148954	-0.43807	-0.4

Models:

1. Naïve Bayes (Date preprocessing 1):

In [90]:
```
#Naive Bayes using Cross validation on normal Data
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
x = np.array(data)
y=np.array(df_label)
clf_cv = GaussianNB()
clf_cv.fit(data,df_label)
score_nb = cross_val_score(clf_cv, data, df_label, cv=10, scoring='accuracy')
print(score_nb)
```

```
[0.64750916 0.67724964 0.67727421 0.67728708 0.67733624 0.67736082
 0.67737369 0.67739828 0.67742287 0.67747205]
```

In [91]:
```
print(score_nb.mean())
```

```
0.6743684032645229
```

2. Naïve Bayes (Date preprocessing 2):

In [88]:
```
#Naive Bayes using Cross validation on Normal Data
x = np.array(data_scaled)
y=np.array(df_label)
clf_cv2 = GaussianNB()
clf_cv2.fit(data,df_label)
score_nb2 = cross_val_score(clf_cv2, data, df_label, cv=10, scoring='accuracy')
print(score_nb2.mean())
```

```
0.7434765412986523
```

3. KNN (Date preprocessing 1):

In [90]:
```
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# 10-fold cross-validation with K=20 for KNN (the n_neighbors parameter)
knn = KNeighborsClassifier(n_neighbors=20)
# scoring='accuracy' for evaluation metric
scores = cross_val_score(knn, data_scaled, df_label, cv=10, scoring='accuracy')
print(scores)
```

```
[0.44116687 0.47851959 0.46921151 0.4572341  0.47759027 0.49350367
 0.53037892 0.57667598 0.6123049  0.67213591]
```

In [91]:
```
print(scores.mean())
```

```
0.5208721716814246
```

4. KNN (Date preprocessing 2):

```
from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# 10-fold cross-validation with K=20 for KNN (the n_neighbors parameter)
knn = KNeighborsClassifier(n_neighbors=20)
# scoring='accuracy' for evaluation metric
scores = cross_val_score(knn, data_scaled, df_label, cv=10, scoring='accuracy')
print(scores.mean())
```

```
0.748732284537623
```

5. Decision tree (Date preprocessing 1):

```
# Decision Tree
# 2
# Import the model we are using
import random
from sklearn.model_selection import cross_validate
#from random import randint
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

#Hyperparameter tuning
param_dist = {"max_depth": [3, 20], 'splitter': ['random'],
              "criterion": ["gini", "entropy"]}
# Instantiate model with 1000 decision trees
dt = tree.DecisionTreeClassifier()
grid_search = GridSearchCV(dt, param_grid=param_dist, cv=10)
#Fit it to data
dt=grid_search.fit(data_scaled, df_label)

#print the tuned parameters and score

print("Tuned Decision Tree Parameters:{}".format(grid_search.best_params_))
print("Best score is:{}".format(grid_search.best_score_))
```

```
Tuned Decision Tree Parameters:{'criterion': 'gini', 'max_depth': 3, 'splitter': 'random'}
Best score is:0.6848196496423478
```

6. Decision tree (Date preprocessing 2):

```
# Decision Tree
# 2
# Import the model we are using
import random
from sklearn.model_selection import cross_validate
#from random import randint
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

#Hyperparameter tuning
param_dist = {"max_depth": [3, 20], 'splitter': ['random'],
              "criterion": ["gini", "entropy"]}
# Instantiate model with 1000 decision trees
dt = tree.DecisionTreeClassifier()
grid_search = GridSearchCV(dt, param_grid=param_dist, cv=10)
#Fit it to data
dt=grid_search.fit(data_scaled, df_label)

#print the tuned parameters and score
print("Tuned Decision Tree Parameters:{}".format(grid_search.best_params_))
print("Best score is:{}".format(grid_search.best_score_))
```

```
Tuned Decision Tree Parameters:{'criterion': 'gini', 'max_depth': 3, 'splitter': 'random'}
Best score is:{0.7683216782034567}
```

7. Random Forest (Date preprocessing 1):

In [55]:
```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
rf_2 = RandomForestClassifier(n_estimators = 500, max_depth=8)
rf_2.fit(data_scaled, df_label)
rf_score2 = cross_val_score(rf_2, data_scaled, df_label, cv =10)
print(rf_score2.mean())
```

```
0.7023943628743212
```

8. Random Forest (Date preprocessing 2):

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
rf_2 = RandomForestClassifier(n_estimators = 600, max_depth=8)
rf_2.fit(data_scaled, df_label)
rf_score2 = cross_val_score(rf_2, data_scaled, df_label, cv =10)
print(rf_score2.mean())
```

```
0.7994295412387653
```

9. Support Vector Machines (Date Preprocessing 1):

In [65]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

SVM_ncross1 = OneVsRestClassifier(LinearSVC(random_state=0)).fit(data_scaled, df_label)
score = (cross_val_score(SVM_ncross1, data_scaled, df_label, cv=10, scoring='accuracy'))
print(score.mean())
```

```
0.6801902989687368
```

10. Support Vector Machines (Date Preprocessing 2):

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

SVM_ncross1 = OneVsRestClassifier(LinearSVC(random_state=0)).fit(data_scaled, df_label)
score = (cross_val_score(SVM_ncross1, data_scaled, df_label, cv=10, scoring='accuracy'))
print(score.mean())
```

```
0.7912386452736485
```

11. XGBoost (Date Preprocessing 1):

In [84]:

```
# k-fold cross validation evaluation of xgboost model
from numpy import loadtxt
import xgboost
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# CV model
model = xgboost.XGBClassifier()
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(model,data_scaled, df_label, cv=kfold)
print("Accuracy: %.2f%% " % (results.mean()*100))
```

```
Accuracy: 70.93%
```

12. XGBoost (Date Preprocessing 2):

```
# k-fold cross validation evaluation of xgboost model
from numpy import loadtxt
import xgboost
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


# CV model
model = xgboost.XGBClassifier()
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(model,data_scaled, df_label, cv=kfold)
print("Accuracy: %.2f%% " % (results.mean()*100))
```

```
Accuracy: 81.43%
```

Exploratory Analysis:

Figure 1:

In [35]:
```
plt.figure(figsize=(12,6))
sns.distplot(df.age.dropna(), rug=True)
sns.despine()
plt.show()
```

```
D:\Anaconda\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence
for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the f
uture this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either i
n an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

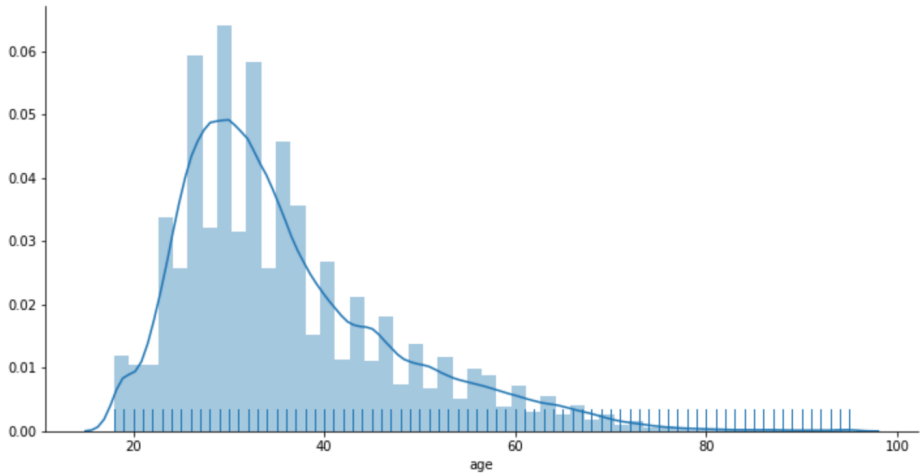


Figure 2:

[34]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='gender', data=df)
plt.ylabel('Number of users')
plt.title('Users gender distribution')
plt.show()
```

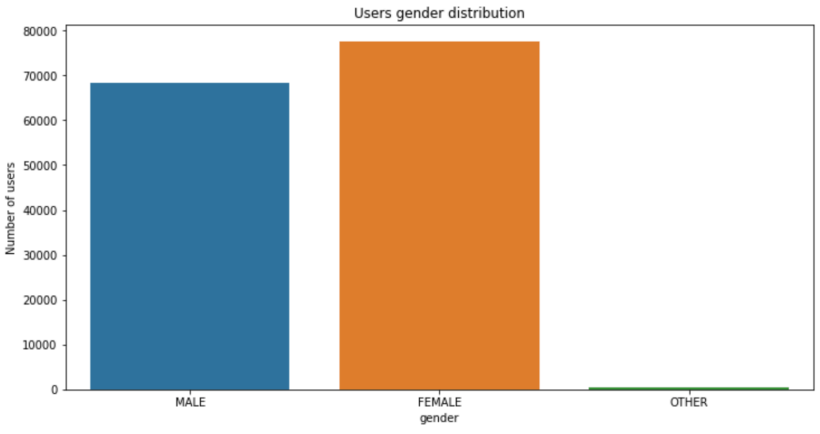


Figure 3:

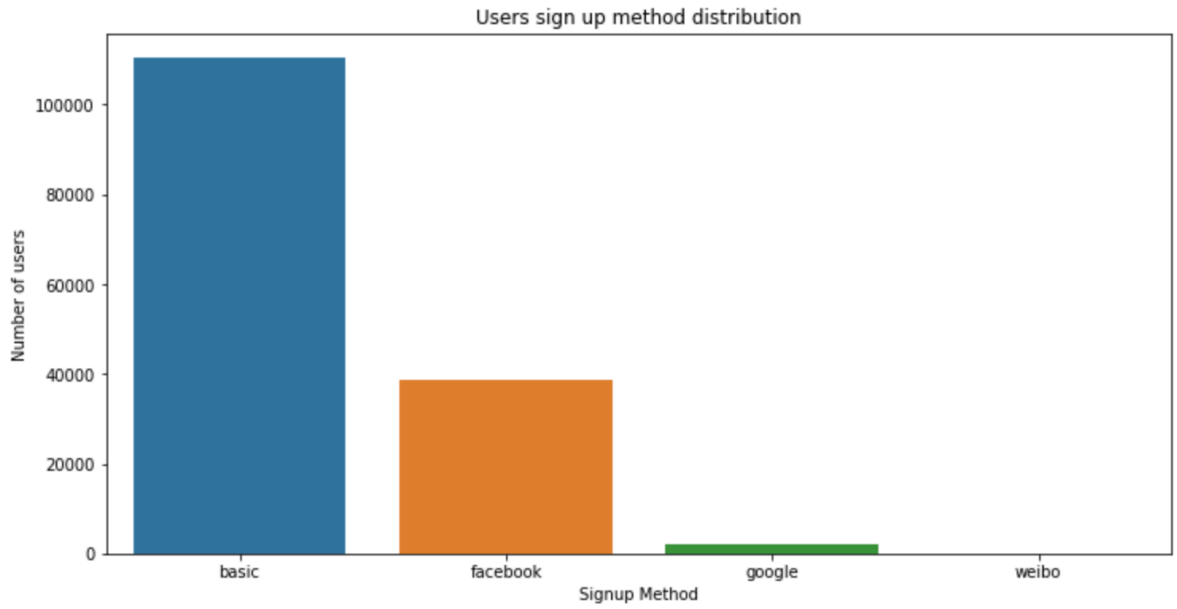


Figure 4:

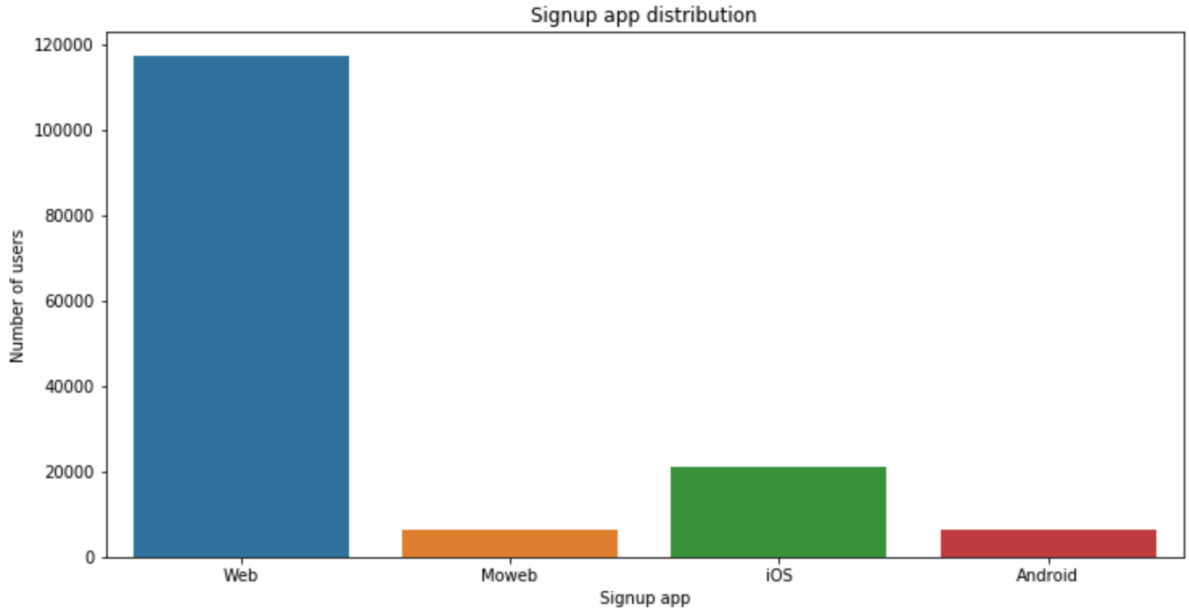


Figure 5:

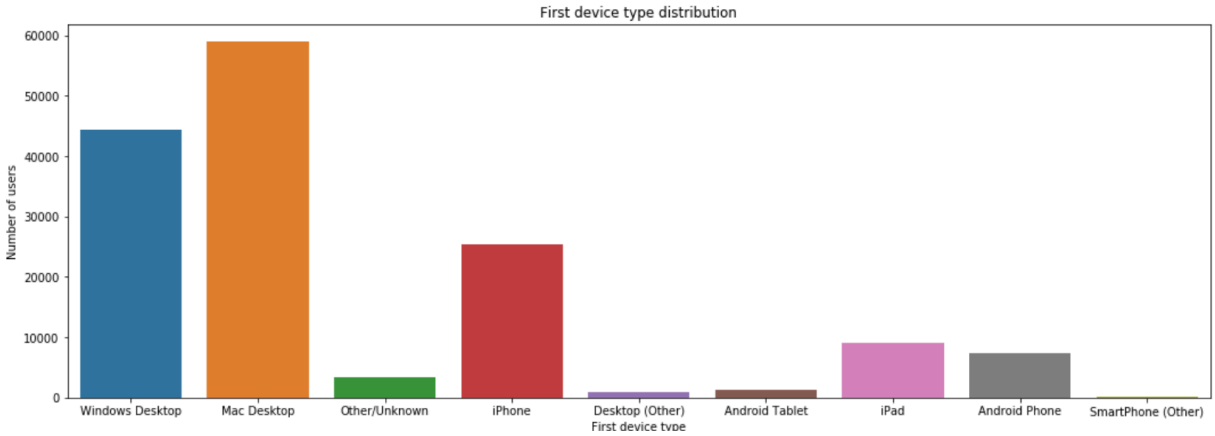


Figure 6:

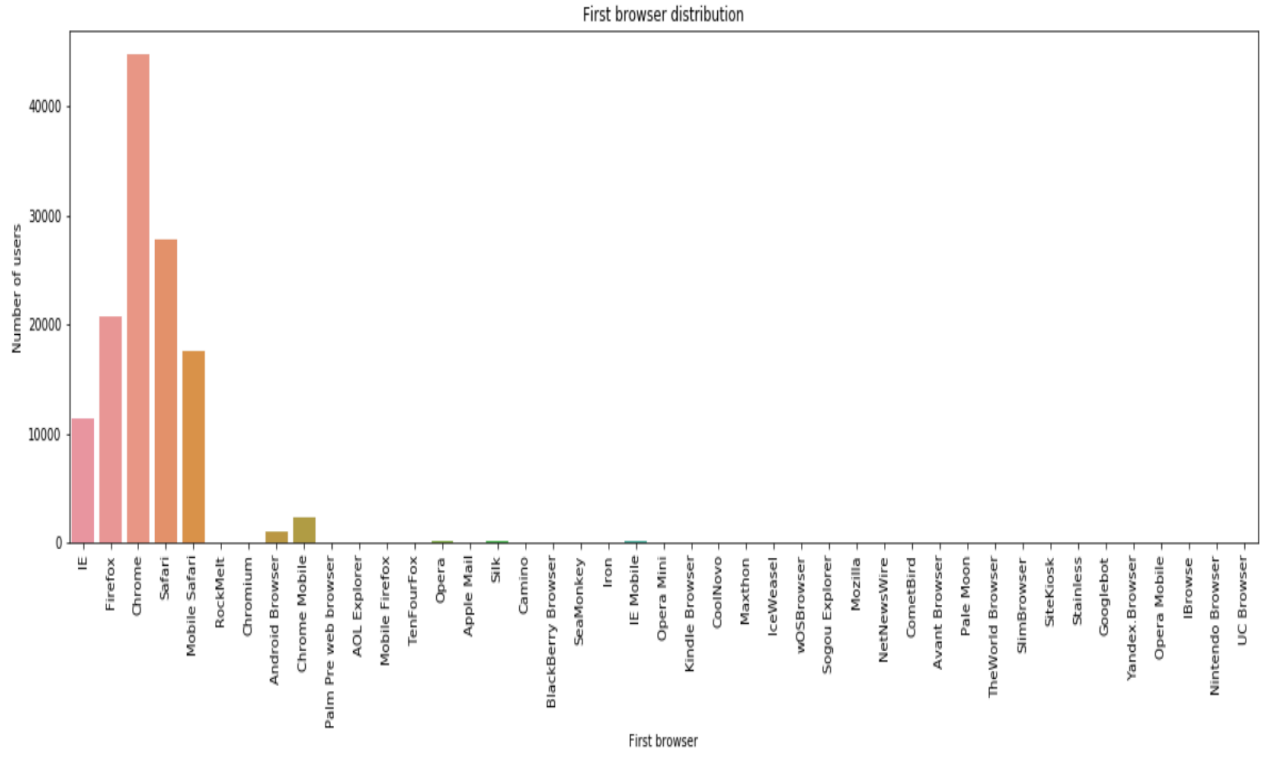


Figure 7:

In [41]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='affiliate_channel', data=df_without_NDF)
plt.xlabel('Affiliate channel')
plt.ylabel('Number of users')
plt.title('Affiliate channel distribution')
plt.show()
```

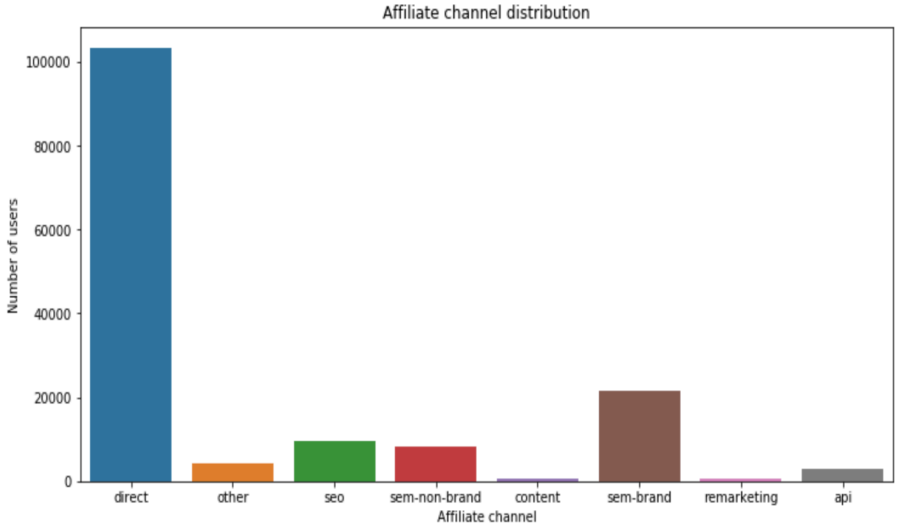
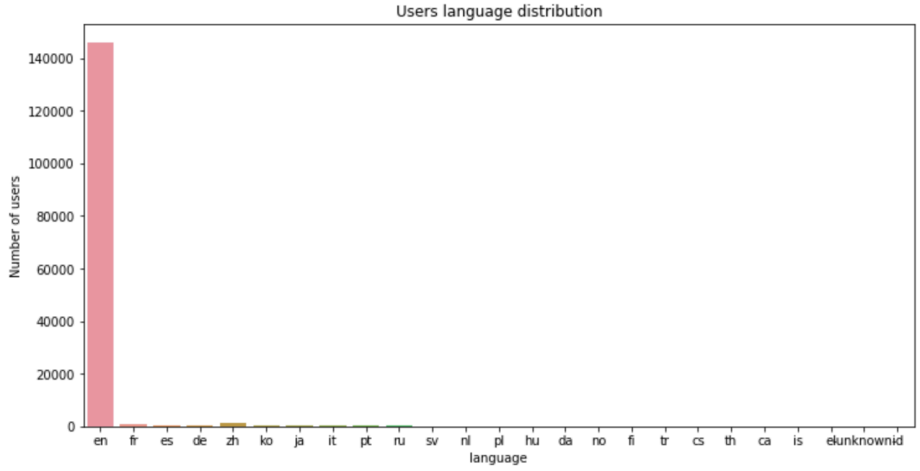
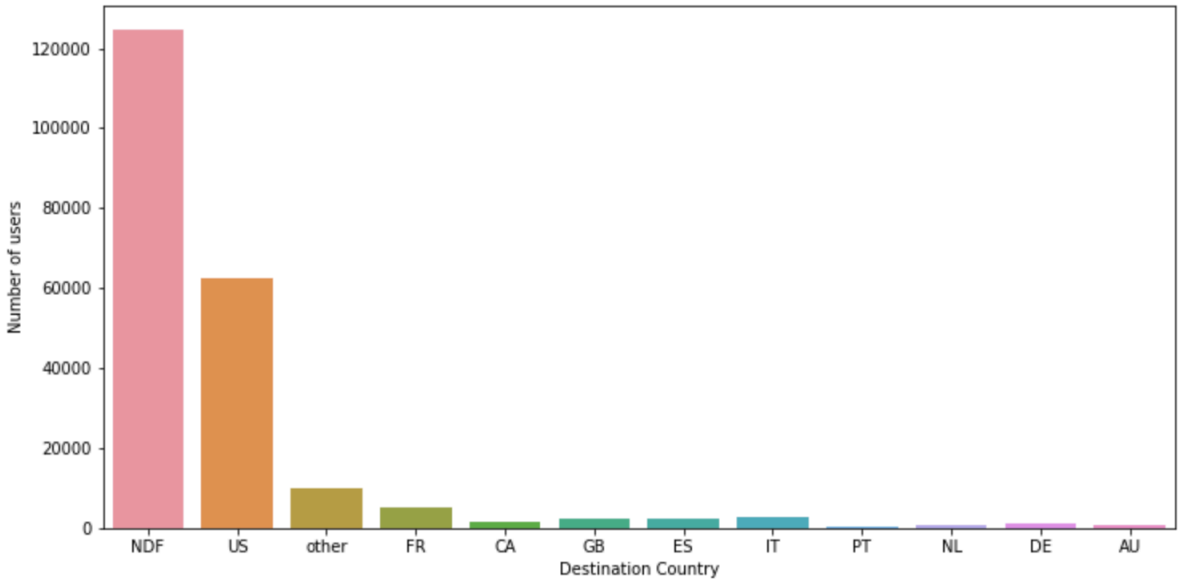


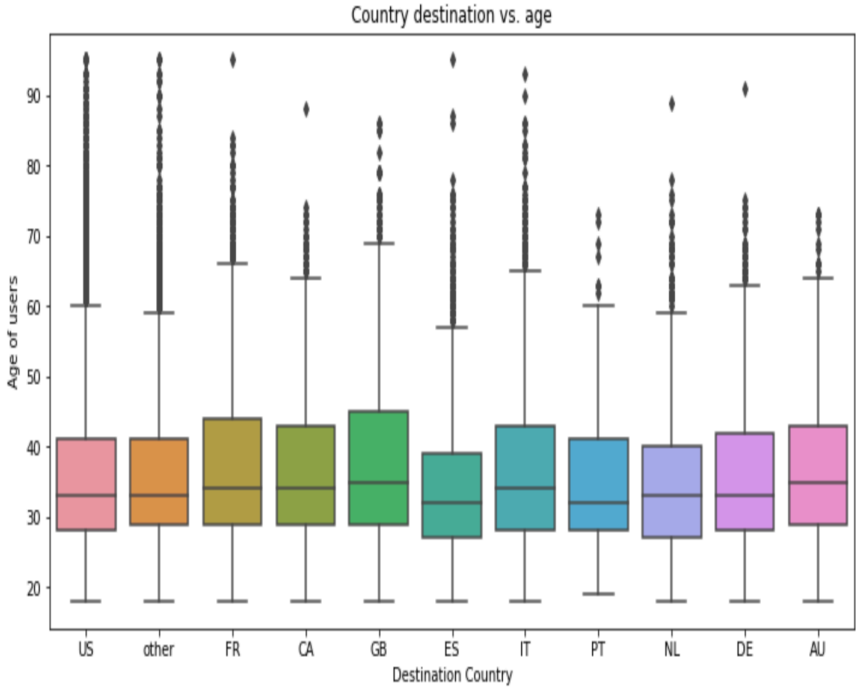
Figure 8:

In [80]:
```
plt.figure(figsize=(12,6))
sns.countplot(x='language', data=df_without_NDF)
plt.xlabel('language')
plt.ylabel('Number of users')
plt.title('Users language distribution')
plt.show()
```



Label:


120000
100000
80000
60000
40000
20000
0
Number of users
NDF
US
other
FR
CA
GB
ES
IT
PT
NL
DE
AU
Destination Country


Country destination vs. age
90
80
70
60
50
40
30
20
Age of users
US
other
FR
CA
GB
ES
IT
PT
NL
DE
AU
Destination Country