Xia Fu
Project: Airbnb New User Bookings

1. Summary

Problem:
Airbnb is an online marketplace for arranging or offering lodging, primarily homestays, or tourism experiences. New users on Airbnb can book a place to stay in 34000+ cities and across 190+ countries. Through analyzing past data provided by Airbnb, accurately predicting where a new user will book their first travel. As a result, Airbnb can share more personalized content with their community, decrease the average time to first booking, and better forecast demand. In this case, we need to predict the first travel destination of a new user based on his/her registration information.

Data:

1. **Train/Test_users.cvs**: there are 16 features, include User_ID, gender, age, signup_method, language, affiliate_channel, affiliate_provider, signup_app, first_device_type, first_brower, etc.
2. **Age_gender_bkts.cvs**: summary statistics of users' age group, gender, country of destination
3. **Countries.csv**: summary statistics of destination countries and their locations
4. **Sessions.csv**: web sessions log for users, includes user's actions, device type, etc.

Findings:
There is a weak signal in user features and session data.
From XGB feature importance analysis, we know that the age of people and the date of registration is the most important features.
Develop high resolution cross validation can help with better feature selection and model ensembling.

2. Benchmarking of Other Solutions

Kernal Name	Feature Approach	Model Approach	Train/Test Perf
airbnb-new-user-bookings	LabelEncoder, sklearn.preprocessing.StandardScaler	Gaussian Naive Bayes	0.545
		Linear Discriminant Analysis	0.5896
		Gradient Boosting Classifier	0.631
airbnb-first-sample	LabelEncoder sklearn.preprocessing.StandardScaler	RandomForestClassifier	0.573
		DecisionTreeClassifier	0.520
		XGBClassifier	0.627
dsbook-benchmark	LabelEncoder, Get Dummies	Cross validation--XGBoost	0.863

All the three kernels use LabelEncoder and get dummies to process their data, because most of the features are categorical variables, such as signup_app, first_device_type, language, and destination. In this way, categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

The first two standardize the data by using sklearn.preprocessing.StandardScaler, so that all the data aggregrate around 0 and has the variance of 1. And all the variables are comparable to each other.

The first kernel shows three different approaches that are Gaussian Naive Bayes, Linear Discriminant Analysis, and Gradient Boosting Classifier. Clearly Gradient Boosting Classifier gives the highest performance which is 0.631.

The **Gaussian Naive Bayes** assumes that all the observations of these features belong to a certain category conform to the Gaussian distribution. **Linear Discriminant Analysis** finds a linear combination of features that characterizes or separates two or more classes of objects or events. However, our data might not be able to align greatly to either a linear or gaussian distribution with so many features.

Gradient Boosting vs XGBoost

The first two kernels used Gradient Boosting and XGBoost model with one coding data. This two approaches gives similar results. The main difference between the two models is that XGBoost makes a few more modifications, which allows it to be more flexible and robust than Gradient Boosting. In particular, XGBoost uses second-order gradients of the loss function, which transforms the loss function into a more sophisticated objective function containing regularisation terms.

However, the use of Gradient Boosting and XGBoost in this problem gives the similar performance. The third kernel uses cross validation together with XGBoost that largely increased the performance of the XGBoost approach, which gives 0.863 accuracy. Because cross validation helps avoiding over-fitting and achieving more generalized relationships.

Random forest vs Decision tree.
In kernel 2, we see that Random forest works better than decision tree. The reason is clear that Randomforest is a forest formed by randomly calculating many decision trees with training data. And then use the forest to predict the unknown data. Our data has many different features, so choosing random forest to predict will be definitely better than using a single tree.

Random Forest vs. XGBoost
For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data. In addition, XGBoost build trees one at a time, where each new tree helps to correct errors made by previously trained tree. Maybe that's why XGBoost shows a little better performance than Random forest.

3. Data Description and Initial Processing:

```python
train = pd.read_csv('train_users_2.csv')
test = pd.read_csv('test_users.csv')
```

```python
countries = pd.read_csv('countries.csv')
age_gender = pd.read_csv('age_gender_bkts.csv')
sessions = pd.read_csv('sessions.csv')
```

```python
train.info()
```

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

```python
train.head()
```

	id	date_account_created	timestamp_first_active	date_first_booking	gender	age	signup_method	signup_flow	language
0	gxn3p5htnn	2010-06-28	20090319043255	NaN	-unknown-	NaN	facebook	0	en
1	820tgsjxq7	2011-05-25	20090523174809	NaN	MALE	38.0	facebook	0	en
2	4ft3gnwmtx	2010-09-28	20090609231247	2010-08-02	FEMALE	56.0	basic	3	en
3	bjjt8pjhuk	2011-12-05	20091031060129	2012-09-08	FEMALE	42.0	facebook	0	en
4	87mebub9p4	2010-09-14	20091208061105	2010-02-18	-unknown-	41.0	basic	0	en

```python
train.shape
```

```
(213451, 16)
```

Drop 'id', 'date_first_booking' and 'destination' column

'date_first_booking' is useless because we want to predict before their first booking

Contatenate train and test dataset

```python
X_train = train.drop(['id', 'date_first_booking', 'country_destination'], axis=1)
X_test = test.drop(['id', 'date_first_booking'], axis=1)
```

```python
y_des = train['country_destination'].values
```

```python
X=pd.concat((X_train, X_test), axis=0, ignore_index=True)
```

```python
X.shape
```

```
(275547, 13)
```

Fill missing data with its last value

```
X.fillna(method='pad').head()
```

	age	dac_year	dac_month	dac_day	tfa_year	tfa_month	tfa_day	gender_-unknown-	gender_FEMALE	gender_MALE	...	first_browser_Silk
0	NaN	2010	6	28	2009	3	19	1	0	0	...	0
1	38.0	2011	5	25	2009	5	23	0	0	1	...	0
2	56.0	2010	9	28	2009	6	9	0	1	0	...	0
3	42.0	2011	12	5	2009	10	31	0	1	0	...	0
4	41.0	2010	9	14	2009	12	8	1	0	0	...	0

5 rows × 160 columns

Delete unmeaningful value in age column

```
X.loc[X.age > 95, 'age'] = -1
X.loc[X.age < 13, 'age'] = -1
```

```
X['age'].describe()
```

```
count    158681.000000
mean         35.346254
std          12.461746
min          -1.000000
25%          28.000000
50%          33.000000
75%          41.000000
max          95.000000
Name: age, dtype: float64
```

Split 'date_account_created' as year, month, day

```
dac = np.vstack(
    X.date_account_created.astype(str).apply(
        lambda x: list(map(int, x.split('-')))
    ).values
)
X['dac_year'] = dac[:, 0]
X['dac_month'] = dac[:, 1]
X['dac_day'] = dac[:, 2]
X = X.drop(['date_account_created'], axis=1)
```

```
X.head()
```

	timestamp_first_active	gender	age	signup_method	signup_flow	language	affiliate_channel	affiliate_provider	first_affiliate_track
0	20090319043255	-unknown-	NaN	facebook	0	en	direct	direct	untrack
1	20090523174809	MALE	38.0	facebook	0	en	seo	google	untrack
2	20090609231247	FEMALE	56.0	basic	3	en	direct	direct	untrack
3	20091031060129	FEMALE	42.0	facebook	0	en	direct	direct	untrack
4	20091208061105	-unknown-	41.0	basic	0	en	direct	direct	untrack

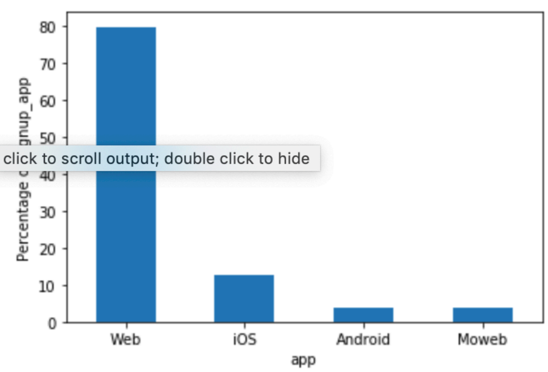
Do same thing for 'timestamp_first_active'

```
tfa = np.vstack(
    X.timestamp_first_active.astype(str).apply(
        lambda x: list(map(int, [x[:4], x[4:6], x[6:8],
                                 x[8:10], x[10:12],
                                 x[12:14]]))
    ).values
)
X['tfa_year'] = tfa[:, 0]
X['tfa_month'] = tfa[:, 1]
X['tfa_day'] = tfa[:, 2]
X = X.drop(['timestamp_first_active'], axis=1)
```

some visualizations

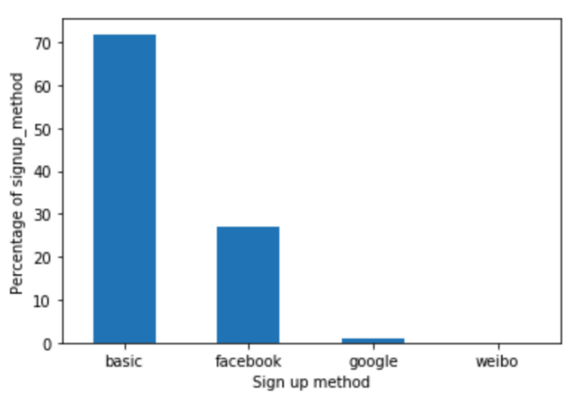
```
signup_app = X.signup_app.value_counts(dropna = False) / len(X) * 100
signup_app.plot('bar', rot = 0)
plt.xlabel('app')
plt.ylabel('Percentage of signup_app')
```

```
Text(0, 0.5, 'Percentage of signup_app')
```



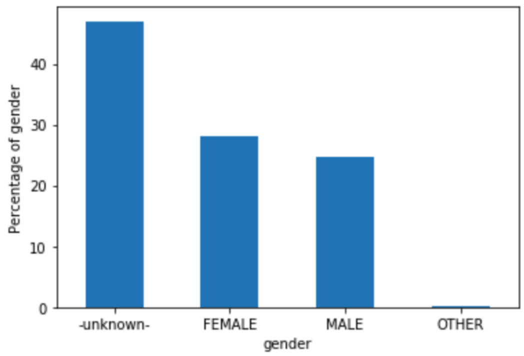

```
signup_method = X.signup_method.value_counts(dropna = False) / len(X) * 100
signup_method.plot('bar', rot = 0)
plt.xlabel('Sign up method')
plt.ylabel('Percentage of signup_method')
```

```
Text(0, 0.5, 'Percentage of signup_method')
```



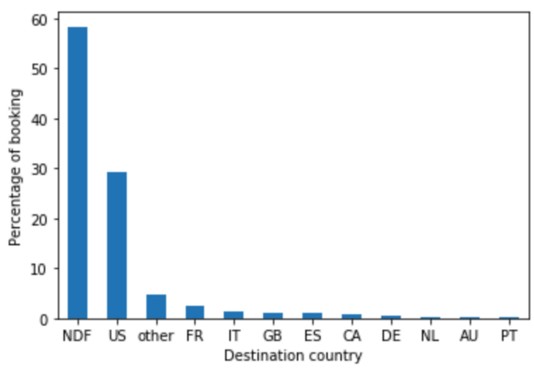
```
gender = X.gender.value_counts(dropna = False) / len(X) * 100
gender.plot('bar', rot = 0)
plt.xlabel('gender')
plt.ylabel('Percentage of gender')
```

```
Text(0, 0.5, 'Percentage of gender')
```




```
des_countries = train.country_destination.value_counts(dropna = False) / len(train) * 100
des_countries.plot('bar', rot = 0)
plt.xlabel('Destination country')
plt.ylabel('Percentage of booking')
```

```
Text(0, 0.5, 'Percentage of booking')
```



One hot coding--by get.dummies

```
oh_features = ['gender', 'signup_method', 'signup_flow', 'language',
               'affiliate_channel', 'affiliate_provider',
               'first_affiliate_tracked', 'signup_app',
               'first_device_type', 'first_browser']
```

```
for feature in oh_features:
    X_dummy = pd.get_dummies(X[feature], prefix=feature)
    X = X.drop([feature], axis=1)
    X = pd.concat((X, X_dummy), axis=1)
```

```
X.head()
```

	age	dac_year	dac_month	dac_day	tfa_year	tfa_month	tfa_day	gender_-unknown-	gender_FEMALE	gender_MALE	...	first_browser_Silk	first_browser_SiteKiosk
0	NaN	2010	6	28	2009	3	19	1	0	0	...	0	0
1	38.0	2011	5	25	2009	5	23	0	0	1	...	0	0
2	56.0	2010	9	28	2009	6	9	0	1	0	...	0	0
3	42.0	2011	12	5	2009	10	31	0	1	0	...	0	0
4	41.0	2010	9	14	2009	12	8	1	0	0	...	0	0

5 rows × 160 columns

split the well processed dataset into X_train and X_test

```
X_train = X.iloc[:len(train), :]
X_test = X.iloc[len(train):, :]
```

```
X_train.shape
```

```
(213451, 160)
```

Label Encode target y colunm

```
le = LabelEncoder()
y_trans = le.fit_transform(y_des)
```

```
y_trans.shape
```

```
(213451,)
```