Logistic & SVM Regression of Airbnb New User Booking

In [1]:
```
# Import mathematical libraries for Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Draw inline
%matplotlib inline

# Set figure aesthetics
sns.set_style("white", {'ytick.major.size': 10.0})
sns.set_context("poster", font_scale=1.1)

# Create our dataframes
train_users = pd.read_csv('train_users_2.csv')
```

In [2]:
```
print ('We have', train_users.shape[0], 'users in the training set.')

train_users.head(5)
train_users.info()
```

```
('We have', 213451, 'users in the training set.')
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 27.7+ MB
```

Data Cleansing and Statistics

Our dataset began with 213,451 observations. After going through our data cleansing process, our dataset included approximately 55,000 observations. Since we wanted to maintain data integrity, we systematically went through the dataset and eliminated all observations that were missing values.

In [3]:
```
# Delete these dimensions because they were not of use to our analysis

del train_users['first_affiliate_tracked']
del train_users['affiliate_channel']
del train_users['timestamp_first_active']
del train_users['id']

train_users.head(10)
```

Out[3]:

	date_account_created	date_first_booking	gender	age	signup_method	si
0	2010-06-28	NaN	-unknown-	NaN	facebook	0
1	2011-05-25	NaN	MALE	38	facebook	0
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
4	2010-09-14	2010-02-18	-unknown-	41	basic	0
5	2010-01-01	2010-01-02	-unknown-	NaN	basic	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
9	2010-01-04	2010-01-04	-unknown-	46	basic	0

In [4]:
```
# cast to Dates
train_users['date_account_created'] = pd.to_datetime(train_users['date_a
ccount_created'])
train_users['date_first_booking'] = pd.to_datetime(train_users['date_fir
st_booking'])
```

Looking into the Age variable, we determined that about 90,000 of our observations have a null or invalid value. We noticed that the Age variable had many errors as there were values from 0 to 2014, which leads us to beleive that users errored in adding those values. As a result, we set limits of less than 16 and greater than 99

In [5]:
```
# Count to see how many of our Age observations are invalid

age_NaN = train_users['age'].isnull().sum()
age_Invalid_Less = train_users[train_users.age <= 15]['age'].count()
age_Invalid_Greater = train_users[train_users.age > 99]['age'].count()
total_age = len(train_users)

print ('Total Null Age Count:' , age_NaN)
print ('Total Invalid Age Count:' , (age_Invalid_Less + age_Invalid_Grea
ter))
print ('Total Age Count:', total_age)
```

```
('Total Null Age Count:', 87990)
('Total Invalid Age Count:', 2436)
('Total Age Count:', 213451)
```

Looking into the Gender variable, we determined that about 95,000 of our observations have a null value. For this mini lab, we chose not to extrapolate the Gender variable and instead decided to delete all the null rows. Perhaps in another lab, we could preserve this data by extrapolating this variable.

In [6]:
```
# Count to see how many of our Gender observations are invalid where gen
der = unknown

train_users.gender.replace('-unknown-', np.nan, inplace=True)

gen_NaN = train_users['gender'].isnull().sum()
gen_O = train_users[train_users.gender == 'OTHER']['gender'].count()
gen_M = train_users[train_users.gender == 'MALE']['gender'].count()
gen_F = train_users[train_users.gender == 'FEMALE']['gender'].count()
total_gen = len(train_users)

print ('Total Null Gender Count:' , gen_NaN)
print ('Total Other Gender Count:' , gen_O)
print ('Total Male Gender Count:' , gen_M)
print ('Total Female Gender Count:' , gen_F)
print ('Total Gender Count:', total_gen)
```

```
('Total Null Gender Count:', 95688)
('Total Other Gender Count:', 282)
('Total Male Gender Count:', 54440)
('Total Female Gender Count:', 63041)
('Total Gender Count:', 213451)
```

In [7]:
```
# Drop all of our invalid observations where Age were less than or equal
to 15, greater than to equal 100
# or null

clean_users = train_users.dropna(subset=['age'])
clean_users = clean_users.drop(clean_users[clean_users.age <=15].index)
clean_users = clean_users.drop(clean_users[clean_users.age > 99].index)

print ('After dropping rows with invalid/null ages, we have', clean_user
s.shape[0], 'observations in the training set.')
```

```
('After dropping rows with invalid/null ages, we have', 123025, 'observ
ations in the training set.')
```

In [8]:
```
# Drop all of our invalid observations where gender = unknown

clean_users = clean_users.dropna(subset=['gender'])

print ('After dropping rows with null gender, we have', clean_users.shap
e[0], 'observations in the training set.')
```

```
('After dropping rows with null gender, we have', 106905, 'observations
in the training set.')
```

In [9]:
```
clean_users.head(10)
```

Out[9]:

	date_account_created	date_first_booking	gender	age	signup_method	si
1	2011-05-25	NaT	MALE	38	facebook	0
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
11	2010-01-05	NaT	FEMALE	47	basic	0
13	2010-01-05	NaT	FEMALE	37	basic	0
14	2010-01-07	NaT	FEMALE	36	basic	0

We decided to clean up the date_first_booking variable, because it was highly correlated with booking and traveling in our previous lab. We found that certain days of the week and certain months saw high traffic in users.

In [10]:
```
# Count to see how many of our Date of First Booking observations are nu
ll

dfb_NaN = clean_users['date_first_booking'].isnull().sum()
total_dfb = len(clean_users)

print ('Total Null Date First Booking:' , dfb_NaN)
print ('Total DFB Count:', total_dfb)
```

```
('Total Null Date First Booking:', 51250)
('Total DFB Count:', 106905)
```

In [11]:
```
# Drop all of the null Date of First Booking observations

clean_users = clean_users.dropna(subset=['date_first_booking'])

print ('After dropping rows with null Date of First Booking, we have', c
lean_users.shape[0], 'observations in the training set.')
```

```
('After dropping rows with null Date of First Booking, we have', 55655,
'observations in the training set.')
```

In [12]:
```
clean_users.head(10)
```

Out[12]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

In [13]:
```
# Check if we have any more missing values in our dataset

(clean_users.isnull().sum() / clean_users.shape[0]) * 100
```

Out[13]:
```
date_account_created    0
date_first_booking      0
gender                  0
age                     0
signup_method           0
signup_flow             0
language                0
affiliate_provider      0
signup_app              0
first_device_type       0
first_browser           0
country_destination     0
dtype: float64
```

In [14]:
```
clean_users.describe().transpose()
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
age	55655	36.011589	11.067755	16	28	33	41	99
signup_flow	55655	2.367981	6.354137	0	0	0	0	25

One-Hot Encoding

For this section, we implemented one-hot encoding to change the datatypes from categorical values to ordinal integers. Using this method, we were able to clean-up our dataset for Logistic analysis and SVM modeling.

In [15]:
```
# perform one-hot encoding of the categorical data "gender"

tmp_df = pd.get_dummies(clean_users.gender,prefix='gender')
clean_users = pd.concat((clean_users,tmp_df),axis=1) # add back into the dataframe
clean_users.head(10)
```

Out[15]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

Instead of creating a new column for every country, we decided to stick with 1 Country_Destination column and assign each different country a discrete value. We chose this approach for the Country_Destination column, because this is the column we are trying to predict.

In [16]:

```
# Encoding our categorical variables with a numeric value

clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='AU',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='CA',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='DE',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='ES',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='FR',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='GB',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='IT',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='NL',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='PT',value=0)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='US',value=1)
clean_users['country_destination'] = clean_users.country_destination.rep
lace(to_replace='other',value=0)

clean_users.head(10)
```

Out[16]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

In [17]:

```
# perform one-hot encoding of the categorical data "signup_method"

tmp_df = pd.get_dummies(clean_users.signup_method,prefix='signup_metho
d')
clean_users = pd.concat((clean_users,tmp_df),axis=1) # add back into the
dataframe
clean_users.head(10)
```

Out[17]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

In [18]:

```
# perform one-hot encoding of the categorical data "affiliate_provider"

clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('baidu', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('email-marketing', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place("facebook-open-graph", 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('gsp', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('meetup', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('naver', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('padmapper', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('vast', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('yahoo', 'other')
clean_users['affiliate_provider'] = clean_users['affiliate_provider'].re
place('yandex', 'other')

tmp_df = pd.get_dummies(clean_users.affiliate_provider,prefix='affiliate
_provider')
clean_users = pd.concat((clean_users,tmp_df),axis=1) # add back into the
dataframe
clean_users.head(10)
```

Out[18]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 24 columns

In [19]:

```
# perform one-hot encoding of the categorical data "signup_app"

tmp_df = pd.get_dummies(clean_users.signup_app,prefix='signup_app')
clean_users = pd.concat((clean_users,tmp_df),axis=1) # add back into the
dataframe
clean_users.head(10)
```

Out[19]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 28 columns

In [20]:
```
# perform one-hot encoding of the categorical data "first_device_type"

tmp_df = pd.get_dummies(clean_users.first_device_type,prefix='first_devi
ce_type')
clean_users = pd.concat((clean_users,tmp_df),axis=1) # add back into the
dataframe
clean_users.head(10)
```

Out[20]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 37 columns

In [21]:

```
# perform one-hot encoding of the categorical data "first_browser"

# replace the current first_browser attribute with something slightly mo
re intuitive and readable
clean_users['first_browser_Chrome'] = clean_users.first_browser == 'Chro
me'
clean_users.first_browser_Chrome = clean_users.first_browser_Chrome.asty
pe(np.int)

clean_users['first_browser_Safari'] = clean_users.first_browser == 'Safa
ri'
clean_users.first_browser_Safari = clean_users.first_browser_Safari.asty
pe(np.int)

clean_users['first_browser_IE'] = clean_users.first_browser == 'IE'
clean_users.first_browser_IE = clean_users.first_browser_IE.astype(np.in
t)

clean_users['first_browser_Mobile_Safari'] = clean_users.first_browser =
= 'Mobile Safari'
clean_users.first_browser_Mobile_Safari = clean_users.first_browser_Mobi
le_Safari.astype(np.int)

clean_users['first_browser_Firefox'] = clean_users.first_browser == 'Fir
efox'
clean_users.first_browser_Firefox = clean_users.first_browser_Firefox.as
type(np.int)

clean_users['first_browser_Other'] = ~clean_users.first_browser.isin(['C
hrome', 'Safari', 'IE', 'Mobile Safari', 'Firefox'])
clean_users.first_browser_Other = clean_users.first_browser_Other.astype
(np.int)

clean_users.head(10)
```

Out[21]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 43 columns

After looking into the portions of the language column, we decided to encode this value as language_en and language _other. About 97% of our dataset had the value of 'en' in the language column.

In [22]:
```
# perform one-hot encoding of the categorical data "language"

lang_en = clean_users[clean_users.language == 'en']['language'].count()
total = len(clean_users)

print ('count EN:' , lang_en)
print ('Total:', total)

print ('Percentage English:' , (float(lang_en)/total) * 100)

clean_users['language_en'] = clean_users.language == 'en'
clean_users.language_en = clean_users.language_en.astype(np.int)

clean_users['language_other'] = clean_users.language != 'en'
clean_users.language_other = clean_users.language_other.astype(np.int)
clean_users.head(10)
```

```
('count EN:', 54157)
('Total:', 55655)
('Percentage English:', 97.30841793190189)
```

Out[22]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 45 columns

We chose to encode the date_account_created by month and weekday, because they would repeat more than 3 times. If we had chose week and/or year it would only show up 3 times, whereas weekday would give us a solid breakdown of when accounts were being created.

In [23]:

```
# perform one-hot encoding of the categorical data "date_account_create
d"

clean_users['date_account_created_Month'] = clean_users['date_account_cr
eated'].map(lambda x: x.month)
clean_users['date_account_created_WeekDay'] = clean_users['date_account_
created'].map(lambda x: x.weekday())

clean_users.head(10)
```

Out[23]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 47 columns

Also, we chose to encode the date_first_booking by month and weekday, because they would repeat more than 3 times. If we had chose week and/or year it would only show up 3 times, whereas weekday would give us a solid breakdown of when the most bookings take place.

In [24]:

```
# perform one-hot encoding of the categorical data "date_first_booking"

clean_users['date_first_booking_Month'] = clean_users['date_first_bookin
g'].map(lambda x: x.month)
clean_users['date_first_booking_WeekDay'] = clean_users['date_first_book
ing'].map(lambda x: x.weekday())

clean_users.head(10)
```

Out[24]:

	date_account_created	date_first_booking	gender	age	signup_method	si
2	2010-09-28	2010-08-02	FEMALE	56	basic	3
3	2011-12-05	2012-09-08	FEMALE	42	facebook	0
6	2010-01-02	2010-01-05	FEMALE	46	basic	0
7	2010-01-03	2010-01-13	FEMALE	47	basic	0
8	2010-01-04	2010-07-29	FEMALE	50	basic	0
10	2010-01-04	2010-01-06	FEMALE	36	basic	0
15	2010-01-07	2010-01-08	FEMALE	33	basic	0
19	2010-01-10	2010-01-10	FEMALE	29	basic	0
21	2010-01-10	2010-01-11	MALE	30	basic	0
25	2010-01-12	2010-01-15	FEMALE	26	basic	0

10 rows × 49 columns

In [25]:
```
clean_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55655 entries, 2 to 213445
Data columns (total 49 columns):
date_account_created                    55655 non-null datetime64[ns]
date_first_booking                      55655 non-null datetime64[ns]
gender                                  55655 non-null object
age                                     55655 non-null float64
signup_method                           55655 non-null object
signup_flow                             55655 non-null int64
language                                55655 non-null object
affiliate_provider                      55655 non-null object
signup_app                              55655 non-null object
first_device_type                       55655 non-null object
first_browser                           55655 non-null object
country_destination                     55655 non-null int64
gender_FEMALE                           55655 non-null float64
gender_MALE                             55655 non-null float64
gender_OTHER                            55655 non-null float64
signup_method_basic                     55655 non-null float64
signup_method_facebook                  55655 non-null float64
signup_method_google                    55655 non-null float64
affiliate_provider_bing                 55655 non-null float64
affiliate_provider_craigslist           55655 non-null float64
affiliate_provider_direct               55655 non-null float64
affiliate_provider_facebook             55655 non-null float64
affiliate_provider_google               55655 non-null float64
affiliate_provider_other                55655 non-null float64
signup_app_Android                      55655 non-null float64
signup_app_Moweb                        55655 non-null float64
signup_app_Web                          55655 non-null float64
signup_app_iOS                          55655 non-null float64
first_device_type_Android Phone         55655 non-null float64
first_device_type_Android Tablet        55655 non-null float64
first_device_type_Desktop (Other)       55655 non-null float64
first_device_type_Mac Desktop           55655 non-null float64
first_device_type_Other/Unknown         55655 non-null float64
first_device_type_SmartPhone (Other)    55655 non-null float64
first_device_type_Windows Desktop       55655 non-null float64
first_device_type_iPad                  55655 non-null float64
first_device_type_iPhone                55655 non-null float64
first_browser_Chrome                    55655 non-null int64
first_browser_Safari                    55655 non-null int64
first_browser_IE                        55655 non-null int64
first_browser_Mobile_Safari             55655 non-null int64
first_browser_Firefox                   55655 non-null int64
first_browser_Other                     55655 non-null int64
language_en                             55655 non-null int64
language_other                          55655 non-null int64
date_account_created_Month              55655 non-null int64
date_account_created_WeekDay            55655 non-null int64
date_first_booking_Month                55655 non-null int64
date_first_booking_WeekDay              55655 non-null int64
dtypes: datetime64[ns](2), float64(26), int64(14), object(7)
```

```
memory usage: 21.2+ MB
```

In [26]:
```python
# Clean up the dataset by removing our categorical variables

if 'date_account_created' in clean_users:
    del clean_users['date_account_created']

if 'date_first_booking' in clean_users:
    del clean_users['date_first_booking']

if 'gender' in clean_users:
    del clean_users['gender']

if 'signup_method' in clean_users:
    del clean_users['signup_method']

if 'affiliate_provider' in clean_users:
    del clean_users['affiliate_provider']

if 'signup_app' in clean_users:
    del clean_users['signup_app']

if 'first_device_type' in clean_users:
    del clean_users['first_device_type']

if 'first_browser' in clean_users:
    del clean_users['first_browser']

if 'language' in clean_users:
    del clean_users['language']

clean_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55655 entries, 2 to 213445
Data columns (total 40 columns):
age                                     55655 non-null float64
signup_flow                             55655 non-null int64
country_destination                     55655 non-null int64
gender_FEMALE                           55655 non-null float64
gender_MALE                             55655 non-null float64
gender_OTHER                            55655 non-null float64
signup_method_basic                     55655 non-null float64
signup_method_facebook                  55655 non-null float64
signup_method_google                    55655 non-null float64
affiliate_provider_bing                 55655 non-null float64
affiliate_provider_craigslist           55655 non-null float64
affiliate_provider_direct               55655 non-null float64
affiliate_provider_facebook             55655 non-null float64
affiliate_provider_google               55655 non-null float64
affiliate_provider_other                55655 non-null float64
signup_app_Android                      55655 non-null float64
signup_app_Moweb                        55655 non-null float64
signup_app_Web                          55655 non-null float64
signup_app_iOS                          55655 non-null float64
first_device_type_Android Phone         55655 non-null float64
first_device_type_Android Tablet        55655 non-null float64
first_device_type_Desktop (Other)       55655 non-null float64
first_device_type_Mac Desktop           55655 non-null float64
first_device_type_Other/Unknown         55655 non-null float64
first_device_type_SmartPhone (Other)    55655 non-null float64
first_device_type_Windows Desktop       55655 non-null float64
first_device_type_iPad                  55655 non-null float64
first_device_type_iPhone                55655 non-null float64
first_browser_Chrome                    55655 non-null int64
first_browser_Safari                    55655 non-null int64
first_browser_IE                        55655 non-null int64
first_browser_Mobile_Safari             55655 non-null int64
first_browser_Firefox                   55655 non-null int64
first_browser_Other                     55655 non-null int64
language_en                             55655 non-null int64
language_other                          55655 non-null int64
date_account_created_Month              55655 non-null int64
date_account_created_WeekDay            55655 non-null int64
date_first_booking_Month                55655 non-null int64
date_first_booking_WeekDay              55655 non-null int64
dtypes: float64(26), int64(14)
memory usage: 17.4 MB
```

Logistic Regression

In [27]:
```
from sklearn.cross_validation import ShuffleSplit

# breaking our dataset into:
# (y) values we are trying to predict
# (x) attributes used to predict y
if 'country_destination' in clean_users:
    y = clean_users['country_destination'].values
    del clean_users['country_destination']
    X = clean_users.values

# creating a cross-validation object to help split our data into a train
ing (80%) and testing (20%) sets
num_cv_iterations = 1
num_instances = len(y)
cv_object = ShuffleSplit(n=num_instances,
                         n_iter=num_cv_iterations,
                         test_size  = 0.2)

print (cv_object)
```

```
ShuffleSplit(55655, n_iter=1, test_size=0.2, random_state=None)
```

In [29]:

```
from sklearn.cross_validation import ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn import metrics as mt
from sklearn.cross_validation import StratifiedKFold


num_cv_iterations = 3
num_instances = len(y)

cv = StratifiedKFold(y, n_folds=10)

print cv

lr_clf = LogisticRegression(penalty='l2', C=0.2, class_weight='auto') #
get object

iter_num=0

# the indices are the rows used for training and testing in each iterati
on
for train, test in cv:
    X_train = X[train]
    y_train = y[train]

    X_test = X[test]
    y_test = y[test]

    # train the reusable logisitc regression model on the training data
    lr_clf.fit(X_train,y_train)  # train object
    y_hat = lr_clf.predict(X_test) # get test set precitions

    # now let's get the accuracy and confusion matrix for this iteration
s of training/testing
    acc = mt.accuracy_score(y_test,y_hat)
    conf = mt.confusion_matrix(y_test,y_hat)
    print "====Iteration",iter_num," ===="
    print "accuracy", acc
    print "confusion matrix\n",conf
    iter_num+=1
```

```
sklearn.cross_validation.StratifiedKFold(labels=[1 0 1 ..., 0 1 1], n_f
olds=10, shuffle=False, random_state=None)
====Iteration 0  ====
accuracy 0.510959396335
confusion matrix
[[ 946  681]
 [2041 1898]]
====Iteration 1  ====
accuracy 0.49101688825
confusion matrix
[[1038  589]
 [2244 1695]]
====Iteration 2  ====
accuracy 0.557671577434
confusion matrix
[[ 698  929]
 [1533 2406]]
====Iteration 3  ====
accuracy 0.372619475386
confusion matrix
[[1351  276]
 [3216  723]]
====Iteration 4  ====
accuracy 0.472152353575
confusion matrix
[[1083  544]
 [2394 1545]]
====Iteration 5  ====
accuracy 0.495058400719
confusion matrix
[[ 938  689]
 [2121 1817]]
====Iteration 6  ====
accuracy 0.581850853549
confusion matrix
[[ 342 1285]
 [1042 2896]]
====Iteration 7  ====
accuracy 0.420305480683
confusion matrix
[[1239  388]
 [2838 1100]]
====Iteration 8  ====
accuracy 0.443486073675
confusion matrix
[[1172  455]
 [2642 1296]]
====Iteration 9  ====
accuracy 0.486433063792
confusion matrix
[[ 886  741]
 [2117 1821]]
```

In [ ]:

```
weights = lr_clf.coef_.T # take transpose to make a column vector
variable_names = train_users.columns
for coef, name in zip(weights,variable_names):
    print name, 'has weight of ', coef[0]

print lr_clf.coef_
```

In [ ]:

```
# importing Logistic Regression packages from SciKit Learn

from sklearn.linear_model import LogisticRegression
from sklearn import metrics as mt
from sklearn.preprocessing import StandardScaler

# creating our Logistic Regression object
lr_clf = LogisticRegression(penalty='l2', C=0.2, class_weight='auto')

# the indices are the rows used for training and testing in each iterati
on
for train_indices, test_indices in cv_object:

    # explicitly breaking our data set into Testing/Training sets
    # this will be usefull to prevent data snooping
    X_train = X[train_indices]
    y_train = y[train_indices]

    X_test = X[test_indices]
    y_test = y[test_indices]

    # use the Standard Scaler to scale our columns
    # this will allow us to interpret the weights from the Logistic Regr
ession Model
    # between the different columns
    scl_obj = StandardScaler()
    scl_obj.fit(X_train)

    X_train_scaled = scl_obj.transform(X_train)
    X_test_scaled = scl_obj.transform(X_test)

    # train the Logistic Regression model using the training data
    lr_clf.fit(X_train_scaled,y_train)  # train object

    # predict the values of the test data
    y_hat = lr_clf.predict(X_test_scaled)

    # print the accuracy score and confusion matrix this model and data
    acc = mt.accuracy_score(y_test,y_hat)
    conf = mt.confusion_matrix(y_test,y_hat)
    print ("==== Iteration ====")
    print ("accuracy", acc)
    print ("confusion matrix\n",conf)

    # combine the column names and the weights into a list
    zip_vars = zip(lr_clf.coef_.T, clean_users.columns)

    # print the column name and its weight
    for coef, name in zip_vars:
        print (name, 'has weight of', coef[0])
```

Confusion matrix shows that no rows were predicted to book their first AirBNB outside the United States. We believe that this is due to a even mixing of individuals who booked in and out of the US. An example of this would be if two individuals with the same values in every attribute booked in different countries (one within the US and one outside the US).

In [ ]:

```
# import the pyplot library
from matplotlib import pyplot as plt
%matplotlib inline

# create pandas series of column names and weights
weights = pd.Series(lr_clf.coef_[0],index=clean_users.columns)

# print bar chart of column names and weights
weights.plot(kind='bar',figsize=(30,4))
plt.show()
```

SVM

In [ ]:

```
# import the SciKit Learn SVC library
from sklearn.svm import SVC

# build and train the SVC model using the training data
svm_clf = SVC(C=0.1, kernel='poly', degree=3, gamma='auto', class_weight
='balanced')
svm_clf.fit(X_train, y_train)  # train object

# predict the values of the test data
y_hat = svm_clf.predict(X_test) # get test set precitions

# print the accuracy score and confusion matrix
acc = mt.accuracy_score(y_test,y_hat)
conf = mt.confusion_matrix(y_test,y_hat)
print ('accuracy:', acc )
print (conf)
```

In [32]:

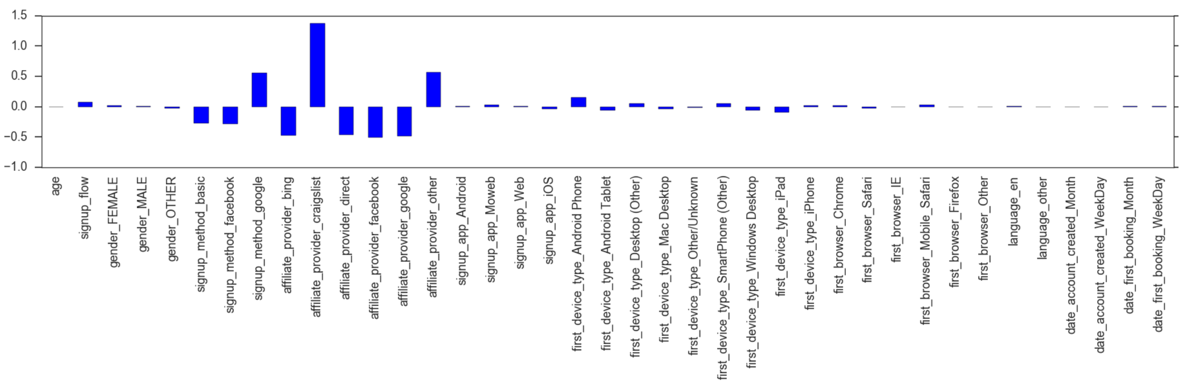
```
# print support vectors
print (svm_clf.support_vectors_.shape)
print (svm_clf.support_.shape)
print (svm_clf.n_support_)
```

```
(42843L, 39L)
(42843L,)
[12602 30241]
```

In [33]:
```
# Support Vector Coefficients
print (svm_clf.coef_)
weights = pd.Series(svm_clf.coef_[0],index=clean_users.columns)
weights.plot(kind='bar', figsize=(30,4))
```

```
[[ -4.40175812e-04   7.62055642e-02   1.75318063e-02   7.03466040e-03
   -2.45664670e-02  -2.74995234e-01  -2.84065889e-01   5.59061123e-01
   -4.75887306e-01   1.36985181e+00  -4.66697243e-01  -5.07953277e-01
   -4.81753642e-01   5.62439660e-01   5.41865285e-03   2.80138506e-02
    6.08989686e-03  -3.95224000e-02   1.48745775e-01  -6.05036818e-02
    4.94500885e-02  -3.43900240e-02  -1.84506027e-02   5.03573665e-02
   -5.70166963e-02  -9.48300257e-02   1.66378001e-02   2.23611657e-02
   -2.83364570e-02  -7.71071017e-03   2.70269841e-02  -9.79284780e-03
   -3.54813486e-03   4.40997968e-03  -4.40997926e-03   3.74588084e-04
   -3.14765929e-03   2.58155621e-03   1.02209927e-03]]
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0xaeb5710>



In [ ]:
```
# Review the support of the vectors

# create a dataframe of the training data
df_tested_on = clean_users.iloc[train_indices] # saved from above, the i
ndices chosen for training
# now get the support vectors from the trained model
df_support = df_tested_on.iloc[svm_clf.support_,:]

df_support['US'] = y[svm_clf.support_] # Place in the 'US' Column to the
pandas dataframe
clean_users['US'] = y # and for the original data
df_support.info()
```

In [ ]:

```
# view attribute statistics
from pandas.tools.plotting import boxplot

# seperately group the original data and the support vectors
df_grouped_support = df_support.groupby(['US']) # support vectors
df_grouped = clean_users.groupby(['US']) # original data

# plot Kernel Density Estimation of Different variables
vars_to_plot = ['date_account_created_Month', 'age', 'gender_MALE']

for v in vars_to_plot:
    plt.figure(figsize=(18,8))

    # plot original distributions
    plt.subplot(1,2,2)
    ax = df_grouped[v].plot.kde()
    plt.legend(['non-US', 'US'])
    plt.title(v+' (Original)')

    # plot support vector stats
    plt.subplot(1,2,1)
    ax = df_grouped_support[v].plot.kde()
    plt.legend(['non-US', 'US'])
    plt.title(v+' (Chosen)')
```

The values chosen to as support vectors have less seperation than those of the original data set because the support vectors are on the edge of class boundaries. The differences between the two sets seems to be small, this could be due to the lack of differences between individuals who first booked outside of the US vs inside of the US.

By taking the dot product of an individual observation and our support vector, we can classify our data to find out where the user will vacation to first usign Airbnb.

The KDE plot for gender are symmetric meaning that our linear model was able to separate the gender data relatively well. On the other hand, our KDE plot for date_account_created does not look very symmetric at all. In a further study, we may want to use a non-linear kernel for our SVM in order to help split the categories better.

Comparison: Logistic Regresion vs. SVM

Our linear models performed very well in our opinion. Using the 80/20 split we were able to consistently get scores of 70+% accuracy. Additionally when running our SVM we also were able to consistently get scores greater than 70%. We didn't perform gradient descent as our dataset did not require it, due to the relative small size of our dataset (approx. 50k observations). The scores for both models gave us approximitely the same accuracy scores. The SVM was significantly more expensive to run and complete on our systems (approximate 5 times longer than logistic regression. Both Models are very similar in regards to the outputs/results. SVM is a better classification method than Logistic Regression as it is fundamentally used to seperate via the hyperplane. Whereas Logistic regression is fundamentally used to predict using the logit function. Additionally, SVM is best suited for high dimensionality whereas Logistic Regression is best suited for low dimensionality. With a dataset of 50 dimensions, it appears they are both. Logistic Regression seems to be the better of the two simply becuase it places less stress on the machine(s).

In a further study, we would like to investigate this data using a non-linear kernel. In the numerous models we built for this dataset, we were not able to successfully predict users whose first Airbnb booking was outside of the US. After looking over our dataset, the weights from our models and the KDE plots, we believe that a higher degree polynomial would be best suited as a model for our data.

Model change for weight fix

To fix the SVC and Logistic regression we simply need to add a parameter setting for weight. We added class-weight='auto' which decreased our accuracy rating but dramatically increased our precision and recall due to the model including all class variables (previously it only classified all destinations as the US). Additionally, we changed our model to use stratifiedkfold for validation which saw also saw an increase in the accuracy.

Model Comparison

As seen by the weights plot The models had similar weight values.

the logistic model's weights are faily evenly distributed with high weights found in affiliate providers, signup method and age. Below are the five highest weights per model:

Logistic:

1. affiliateprovider(Craigslist)
2. affiliateprovider(other)
3. devicetype(Ipad)
4. age
5. date_first_booking_Month

SVM:

In the SVM the weights are primarily spread across the device type, the browser type and signup flow. Additionally signup flow held more strength than the age variabl.e

When running the SVM with a linear kernal the model's accuracy was arround 37%, however after switching up to an RBF kernal and changing the class weights to balanced we were able to dramatically increase our SVM score to around 53%. (tweaked and changed the default parameter settings to increase model efficiency)