Take_Home_Ultimate_Challenge

In [121]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
import calendar
from ast import literal_eval
import json
```

In [122]:

```
df_logins = pd.read_json(r'F:\spring board\ultimate_challenge\logins.json')
df_logins.head()
```

Out[122]:

	login_time
0	1970-01-01 20:13:18
1	1970-01-01 20:16:10
2	1970-01-01 20:16:37
3	1970-01-01 20:16:36
4	1970-01-01 20:26:21

In [123]:

```
df_logins.shape
```

Out[123]:

```
(93142, 1)
```

In [124]:

```
df_logins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 93142 entries, 0 to 93141
Data columns (total 1 columns):
login_time    93142 non-null datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 727.8 KB
```

In [125]:

```
df_logins['count'] = 1
df_logins = df_logins.set_index('login_time')
```

In [126]:

```
# Aggregate login counts based on 15-minute time intervals
df_15login = df_logins.resample('15T', label='right').sum()
df_15login.head()
```

Out[126]:

count

	count
login_time	
1970-01-01 20:15:00	2
1970-01-01 20:30:00	6
1970-01-01 20:45:00	9
1970-01-01 21:00:00	7
1970-01-01 21:15:00	1

In [127]:

```
df_15login['time'] = pd.to_datetime(df_15login.index)
```

In [128]:

```
df_15login = df_15login.fillna(0)
```

In [129]:

```
df_15login['month'] = df_15login.time.dt.month
df_15login['day'] = df_15login.time.dt.day
df_15login['hour'] = df_15login.time.dt.hour
df_15login['week'] = df_15login.time.dt.week
df_15login['weekday'] = df_15login.time.dt.weekday
```

In [130]:

```
df_15login.head()
```

Out[130]:

	count	time	month	day	hour	week	weekday
login_time							
1970-01-01 20:15:00	2	1970-01-01 20:15:00	1	1	20	1	3
1970-01-01 20:30:00	6	1970-01-01 20:30:00	1	1	20	1	3
1970-01-01 20:45:00	9	1970-01-01 20:45:00	1	1	20	1	3
1970-01-01 21:00:00	7	1970-01-01 21:00:00	1	1	21	1	3
1970-01-01 21:15:00	1	1970-01-01 21:15:00	1	1	21	1	3

EDA

Now let us group the logins by months weeks and days to see the pattern of logins

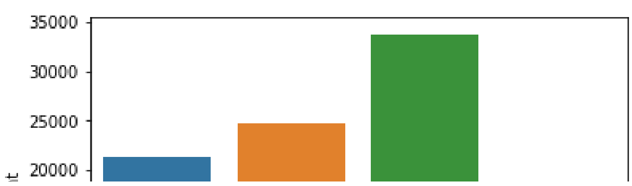
Months

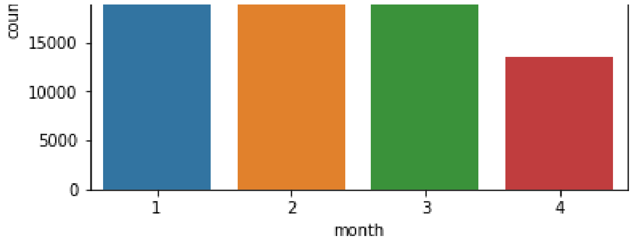
In [131]:

```
month_counts = df_15login.groupby('month')['count'].sum()
sns.barplot(x=month_counts.index, y=month_counts)
```

Out[131]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12a0f97a588>
```





As yo see logins are high in month 3 and low in month 4, and it is increasing from month 1 to 3 and suddenly decreased in month 4

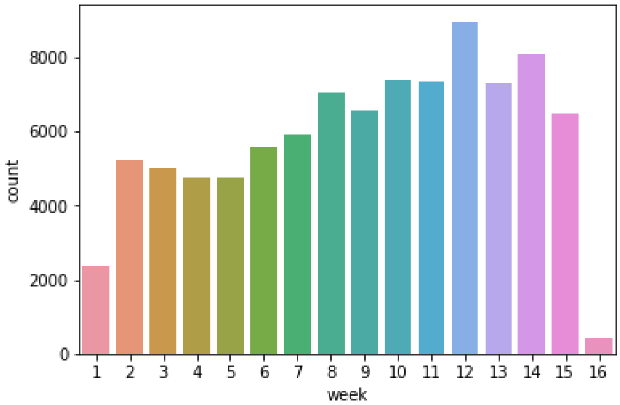
Weeks

In [132]:

```
week_counts = df_15login.groupby('week')['count'].sum()
sns.barplot(x=week_counts.index, y=week_counts)
```

Out[132]:

<matplotlib.axes._subplots.AxesSubplot at 0x12a10d5eb00>



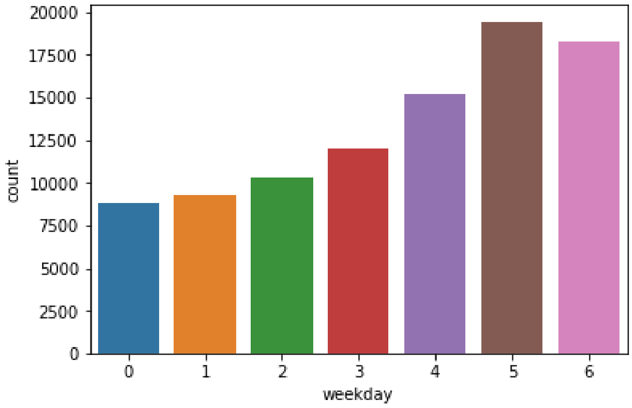
Days

In [133]:

```
day_counts = df_15login.groupby('weekday')['count'].sum()
sns.barplot(x=day_counts.index, y=day_counts)
```

Out[133]:

<matplotlib.axes._subplots.AxesSubplot at 0x12a10954400>



Login number kept increasing from Monday to Saturday. There are more logins in weekends than in weekdays. Logins on Saturday are the most, and logins on Sunday are the second most, while logins on Friday are the third most.

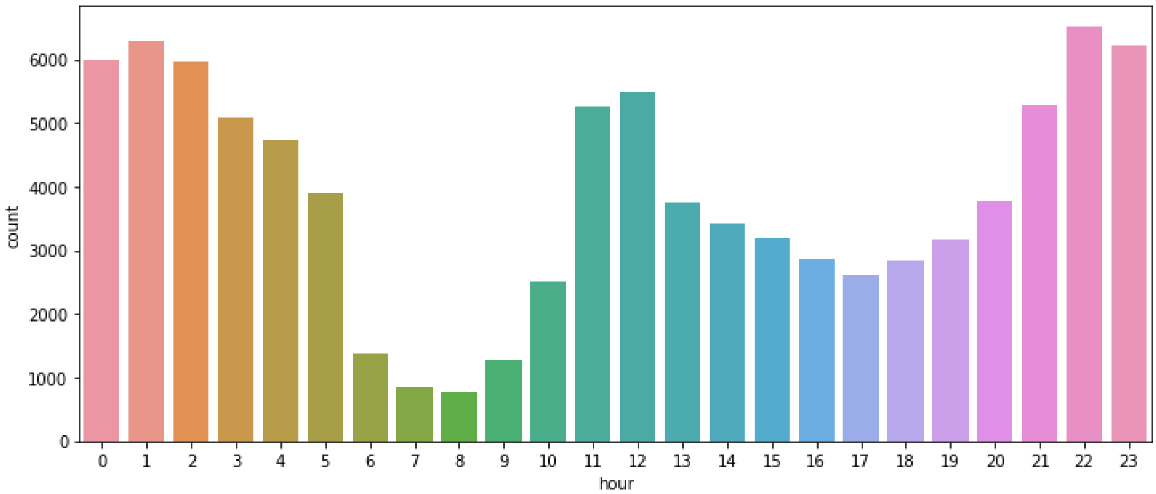
Hours

In [134]:

```
hour_counts = df_15login.groupby('hour')['count'].sum()
plt.figure(figsize=(12,5))
sns.barplot(x=hour_counts.index, y=hour_counts)
```

Out[134]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12a10d432e8>
```



The above graph shows that maximum activity is between 10 PM and 1 AM and minimum is between early morning between 6 AM and 8 AM.

Experiment and Metrics Design

The neighboring cities of Gotham and Metropolis have complementary circadian rhythms: on weekdays, Ultimate Gotham is most active at night, and Ultimate Metropolis is most active during the day. On weekends, there is reasonable activity in both cities.

However, a toll bridge, with a twoway toll, between the two cities causes driver partners to tend to be exclusive to each city. The Ultimate managers of city operations for the two cities have proposed an experiment to encourage driver partners to be available in both cities, by reimbursing all toll costs

My Solution

The key Measure of success for this experiment i will choose is the number of toll entries for each city, because if this experiment is successfull then the drivers will move to other cities, in that process they need to give their car details in the tolls of each city and hence the toll entries will increase.

1 Collect the details of toll entries before and after implementing the experiment and if toll logins were increased after the experiment this shows that the experiment was successfull. 2 Perform hypothesis testing using the Difference of Proportions test on the two samples. Calculate the Z-Statisitc and the p- value and compare it with an arbitrary significance level, α. 3 If the Null Hypothesis holds, it implies that the experiment has been a failure. If the Alternate hypothesis holds, it implies that it has been a success

Predictive Modelling

In [135]:

```
file = open(r'F:\spring board\ultimate_challenge\ultimate_data_challenge.json')
df2 = pd.DataFrame(json.load(file))
file.close()
```

```
file.close()

df2.head()
```

Out[135]:

	avg_dist	avg_rating_by_driver	avg_rating_of_driver	avg_surge	city	last_trip_date	phone	signup_date	surge_pct	trips_in_f
0	3.67	5.0	4.7	1.10	King's Landing	2014-06-17	iPhone	2014-01-25	15.4	
1	8.26	5.0	5.0	1.00	Astapor	2014-05-05	Android	2014-01-29	0.0	
2	0.77	5.0	4.3	1.00	Astapor	2014-01-07	iPhone	2014-01-06	0.0	
3	2.36	4.9	4.6	1.14	King's Landing	2014-06-29	iPhone	2014-01-10	20.0	
4	3.13	4.9	4.4	1.19	Winterfell	2014-03-15	Android	2014-01-27	11.8	

In [136]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 12 columns):
avg_dist                  50000 non-null float64
avg_rating_by_driver      49799 non-null float64
avg_rating_of_driver      41878 non-null float64
avg_surge                 50000 non-null float64
city                      50000 non-null object
last_trip_date            50000 non-null object
phone                     49604 non-null object
signup_date               50000 non-null object
surge_pct                 50000 non-null float64
trips_in_first_30_days    50000 non-null int64
ultimate_black_user       50000 non-null bool
weekday_pct               50000 non-null float64
dtypes: bool(1), float64(6), int64(1), object(4)
memory usage: 4.2+ MB
```

In [137]:

```
df2['phone'].value_counts()
```

Out[137]:

```
iPhone     34582
Android    15022
Name: phone, dtype: int64
```

In [138]:

```
df2['phone'] = df2['phone'].fillna('iPhone')
```

In [139]:

```
df2['avg_rating_by_driver'].describe()
```

Out[139]:

```
count    49799.000000
mean         4.778158
std          0.446652
min          1.000000
25%          4.700000
50%          5.000000
75%          5.000000
max          5.000000
Name: avg_rating_by_driver, dtype: float64
```

In [140]:

```
df2['avg_rating_by_driver'].isnull().sum()
```

Out[140]:

```
201
```

In [141]:

```
df2[df2['avg_rating_by_driver'].notnull()]['avg_rating_by_driver'].value_counts()
```

Out[141]:

```
5.0    28508
4.8     4537
4.7     3330
4.9     3094
4.5     2424
4.6     2078
4.0     1914
4.3     1018
4.4      860
3.0      602
4.2      342
3.5      199
3.7      195
1.0      181
2.0      126
4.1      125
3.8      111
3.3       47
3.9       41
2.5       31
3.6       19
3.4        5
1.5        4
2.8        3
3.2        2
2.7        2
2.3        1
Name: avg_rating_by_driver, dtype: int64
```

In [142]:

```
df2['avg_rating_by_driver'] = df2['avg_rating_by_driver'].interpolate()
```

In [143]:

```
df2['avg_rating_by_driver'].isnull().sum()
```

Out[143]:

```
0
```

In 'avg_rating_by_driver' column there are 201 missing values and i have filled that using interpolation method because this is t]one of the best mthods for filling the missing values of Continous variables

In [144]:

```
df2['avg_rating_of_driver'].describe()
```

Out[144]:

```
count    41878.000000
mean         4.601559
std          0.617338
min          1.000000
25%          4.300000
50%          4.900000
75%          5.000000
max          5.000000
```

Name: avg_rating_of_driver, dtype: float64

In [145]:

```
df2['avg_rating_of_driver'].isnull().sum()
```

Out[145]:

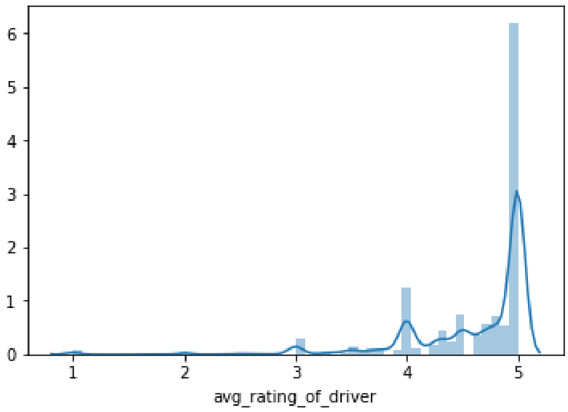
8122

In [146]:

```
sns.distplot(df2[df2['avg_rating_of_driver'].notnull()]['avg_rating_of_driver'])
```

Out[146]:

<matplotlib.axes._subplots.AxesSubplot at 0x12a0c2c2588>



In [147]:

```
df2['avg_rating_of_driver'] =
df2['avg_rating_of_driver'].fillna(df2['avg_rating_of_driver'].mean())
```

Now lets check if there are any null values in the data frame

In [148]:

```
df2.isnull().sum()
```

Out[148]:

```
avg_dist                  0
avg_rating_by_driver      0
avg_rating_of_driver      0
avg_surge                 0
city                      0
last_trip_date            0
phone                     0
signup_date               0
surge_pct                 0
trips_in_first_30_days    0
ultimate_black_user       0
weekday_pct               0
dtype: int64
```

Now lets conert all Date Variables to Datetime

In [149]:

```
df2['signup_date'] = df2['signup_date'].apply(lambda x: pd.Timestamp(x, tz=None))
df2['last_trip_date'] = df2['last_trip_date'].apply(lambda x: pd.Timestamp(x, tz=None))
```

In [150]:

```
df2['last_trip_date'].sort_values(ascending=False).head()
```

Out[150]:

```
45357   2014-07-01
22735   2014-07-01
14473   2014-07-01
38651   2014-07-01
45126   2014-07-01
Name: last_trip_date, dtype: datetime64[ns]
```

We see that this data was taken on 1st July, 2014. Therefore, a user is considered retained if the passenger took a trip after June 1st, 2014.

In [151]:

```
df2['retained'] = df2['last_trip_date'].apply(lambda x: 1 if x >= pd.Timestamp('2014-06-01', tz=None) else 0)
```

In [152]:

```
df2['retained'].value_counts()
```

Out[152]:

```
0    31196
1    18804
Name: retained, dtype: int64
```

As you see the retention rate is not so big i.e only 37.6% of users have been retained. Now lets drop 'signup_date' and 'last_trip_date' columns because these are usefull to retention of the customers and we got that

In [153]:

```
df2 = df2.drop('signup_date', axis=1)
df2 = df2.drop('last_trip_date', axis=1)
```

In [154]:

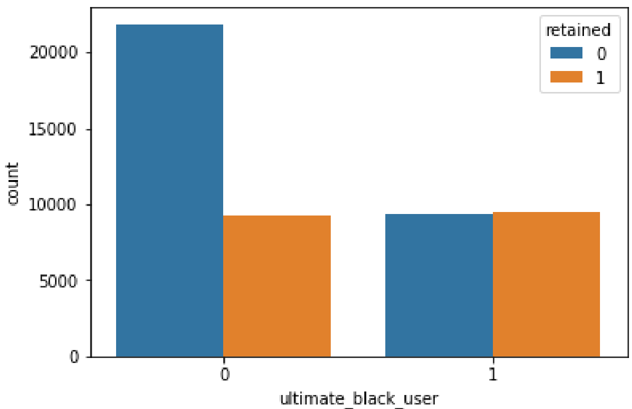
```
df2['ultimate_black_user'] = df2['ultimate_black_user'].apply(lambda x: 1 if x else 0)
```

In [155]:

```
sns.countplot(x='ultimate_black_user', data=df2, hue='retained')
```

Out[155]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12a10811160>
```



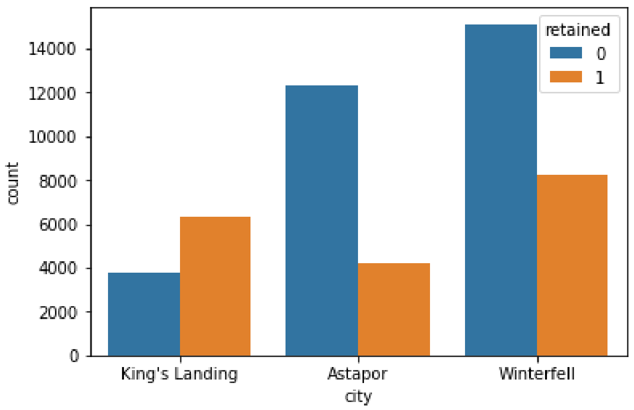
As you see in the above graph retention rate of 'ultimate_black_user' was equal to the number of 'ultimate_black_users' but for non 'ultimate_black_user' retention rate is not greater than 40%

In [156]:

```
sns.countplot(x='city', data=df2, hue='retained')
```

Out[156]:

<matplotlib.axes._subplots.AxesSubplot at 0x12a0fe6db70>



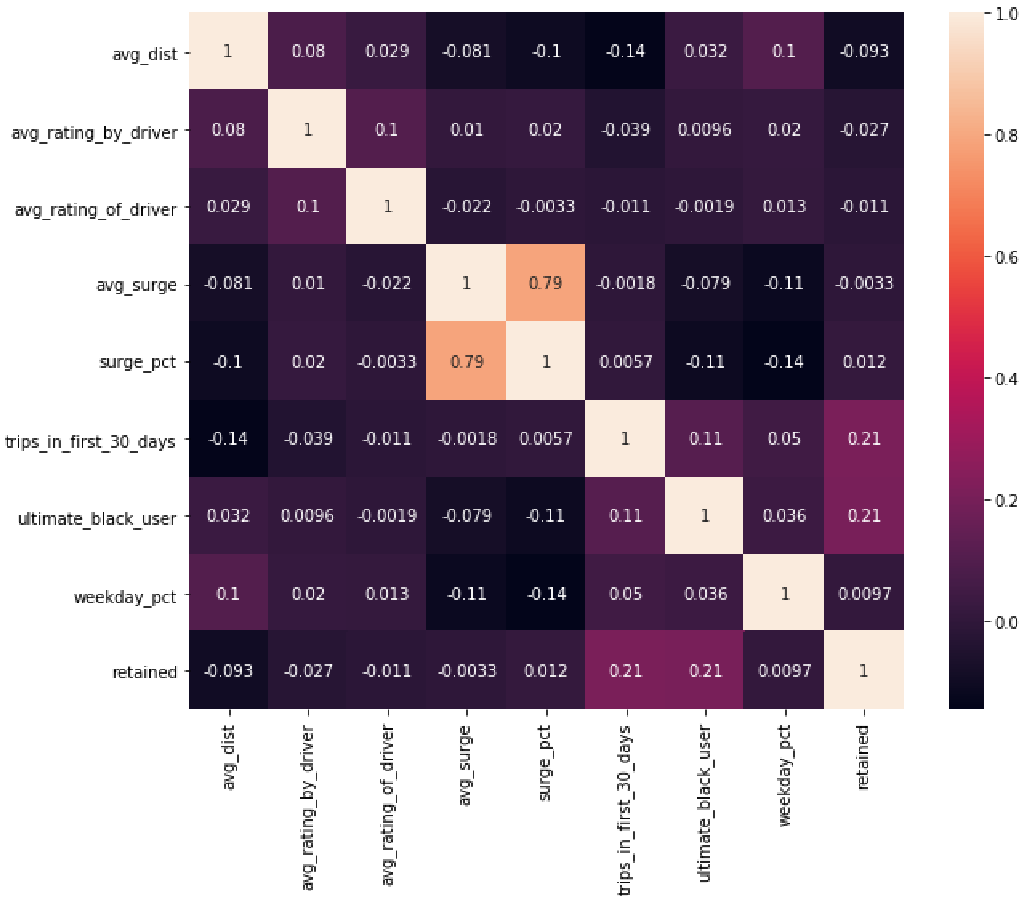
King's Landing seems to be especially successful in retaining users whereas Astapor is the least successful.

In [157]:

```
plt.figure(figsize=(10,8))
sns.heatmap(df2.corr(), annot=True)
```

Out[157]:

<matplotlib.axes._subplots.AxesSubplot at 0x12a1053c5c0>



In [158]:

```
df2 = pd.get_dummies(df2, prefix='is')
```

In [159]:

```
df2 = df2.drop(['avg_surge'], axis=1)
```

In [160]:

```
X = df2.drop(['retained'], axis=1)
y = df2['retained']
```

In [161]:

```
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.7, test_size=0.3, stratify=y
)
```

In [164]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import cross_val_score
```

In [167]:

```
classifier = RandomForestClassifier()
classifier.fit(train_X, train_y)
classifier.score(test_X, test_y)
```

```
C:\Users\user\anaconda\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The defaul
t value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[167]:

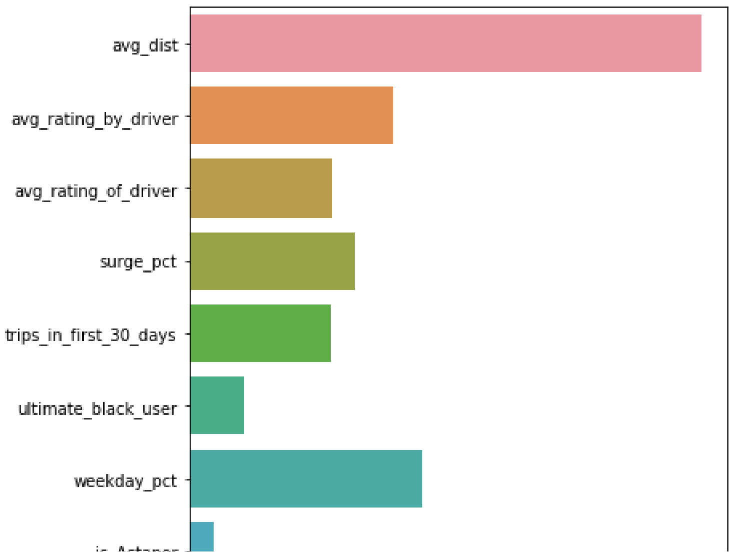
```
0.7518666666666667
```

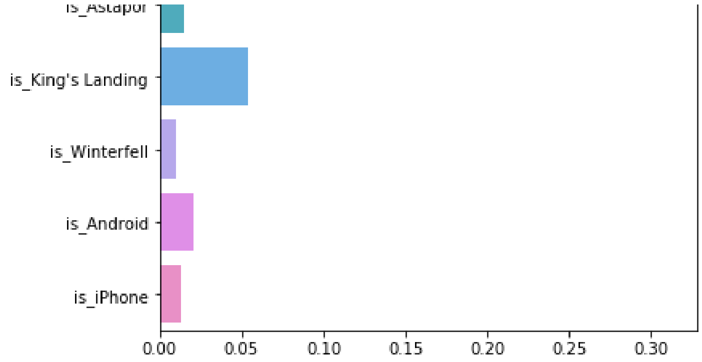
In [168]:

```
plt.figure(figsize=(6,10))
sns.barplot(y=X.columns, x=classifier.feature_importances_)
```

Out[168]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x12a12f3c7f0>
```





Recommendations

1 Increase operations in King's Landing as they tend to have greater probability of conversion. Know what makes ratention rate in King's Landing is high and implement same in other cities to get better retention rate

2 Retention Rate of Ultimate_Black_User is high Know what makes ratention rate of Ultimate_Black_User is high and implement same to other people to get better retention rate

3 From the model we can see that avg_distance is the important feature for retention

4 avg_rating_by_driver is also the important feature for retention so provide bettef facilities to back the customers

In [ ]:

```

```

In [ ]:

```

```