Introduction

In today's market, many companies have a mobile presence. Often these companies provide free products/services in their mobile apps in an attempt to transition their customers to a paid membership. Some examples of paid products, which originate from free ones, are YouTube Red, Pandora Premium, Netflix, Disney+Hotstar, AmazonPrime, Audible Subscription and Spotify. Since marketing efforts are never free, these companies need to know exactly who to target with offers and promotions.

1. **Market**: The target audience is customers who use a company's free product. In this case study, this refers to users who installed (and used) the company's free mobile app.
2. **Product**: The paid memberships often provide enhanced versions of the free products already given for free, alongside new features. For example, YouTube Red allows you to leave the app while still listening to a video.
3. **Goal** : The objective of this model is to predict which users will not subscribe to the paid membership, so that greater marketing efforts can go into trying to 'convert' them to paid users.

Importing Essential Libraries and Our Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from dateutil import parser
```

In [2]:
```
dataset = pd.read_csv('appdata10.csv')
```

In [3]:
```
dataset.head()
```

Out[3]:

	user	first_open	dayofweek	hour	age	screen_list	n
0	235136	2012-12-27 02:14:51.273	3	02:00:00	23	idscreen,joinscreen,Cycle,product_review,ScanP...	
1	333588	2012-12-02 01:16:00.905	6	01:00:00	24	joinscreen,product_review,product_review2,Scan...	
2	254414	2013-03-19 19:19:09.157	1	19:00:00	23	Splash,Cycle,Loan	
3	234192	2013-07-05 16:08:46.354	4	16:00:00	28	product_review,Home,product_review,Loan3,Finan...	
4	51549	2013-02-26 18:50:48.661	1	18:00:00	31	idscreen,joinscreen,Cycle,Credit3Container,Sca...	

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
user                    0
first_open              0
dayofweek               0
hour                    0
```

```
age                        0
screen_list                0
numscreens                 0
minigame                   0
used_premium_feature       0
enrolled                   0
enrolled_date          18926
liked                      0
dtype: int64
```

In [5]:
```
dataset.describe()
```

Out[5]:

	user	dayofweek	age	numscreens	minigame	used_premium_feature
count	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	50000.000000
mean	186889.729900	3.029860	31.72436	21.095900	0.107820	0.172020
std	107768.520361	2.031997	10.80331	15.728812	0.310156	0.377402
min	13.000000	0.000000	16.00000	1.000000	0.000000	0.000000
25%	93526.750000	1.000000	24.00000	10.000000	0.000000	0.000000
50%	187193.500000	3.000000	29.00000	18.000000	0.000000	0.000000
75%	279984.250000	5.000000	37.00000	28.000000	0.000000	0.000000
max	373662.000000	6.000000	101.00000	325.000000	1.000000	1.000000

As seen above, the hour column is not present. This is because it is of a string type. Now lets convert it to an int type.

In [6]:
```
dataset['hour'] = dataset['hour'].str.slice(1, 3).astype(int)
```

In [7]:
```
dataset.head()
```

Out[7]:

	user	first_open	dayofweek	hour	age	screen_list	num
0	235136	2012-12-27 02:14:51.273	3	2	23	idscreen,joinscreen,Cycle,product_review,ScanP...	
1	333588	2012-12-02 01:16:00.905	6	1	24	joinscreen,product_review,product_review2,Scan...	
2	254414	2013-03-19 19:19:09.157	1	19	23	Splash,Cycle,Loan	
3	234192	2013-07-05 16:08:46.354	4	16	28	product_review,Home,product_review,Loan3,Finan...	
4	51549	2013-02-26 18:50:48.661	1	18	31	idscreen,joinscreen,Cycle,Credit3Container,Sca...	

In [8]:
```
dataset.describe()
```

Out[8]:

	user	dayofweek	hour	age	numscreens	minigame	used_pr
count	50000.000000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	
mean	186889.729900	3.029860	12.557220	31.72436	21.095900	0.107820	
std	107768.520361	2.031997	7.438072	10.80331	15.728812	0.310156	

	user	dayofweek	hour	age	numscreens	minigame	used_pr
min	13.000000	0.000000	0.000000	16.00000	1.000000	0.000000	
25%	93526.750000	1.000000	5.000000	24.00000	10.000000	0.000000	
50%	187193.500000	3.000000	14.000000	29.00000	18.000000	0.000000	
75%	279984.250000	5.000000	19.000000	37.00000	28.000000	0.000000	
max	373662.000000	6.000000	23.000000	101.00000	325.000000	1.000000	

Visualization

We will perform this on a copy of our dataset so that we don't end up messing our original dataset.

In [9]:

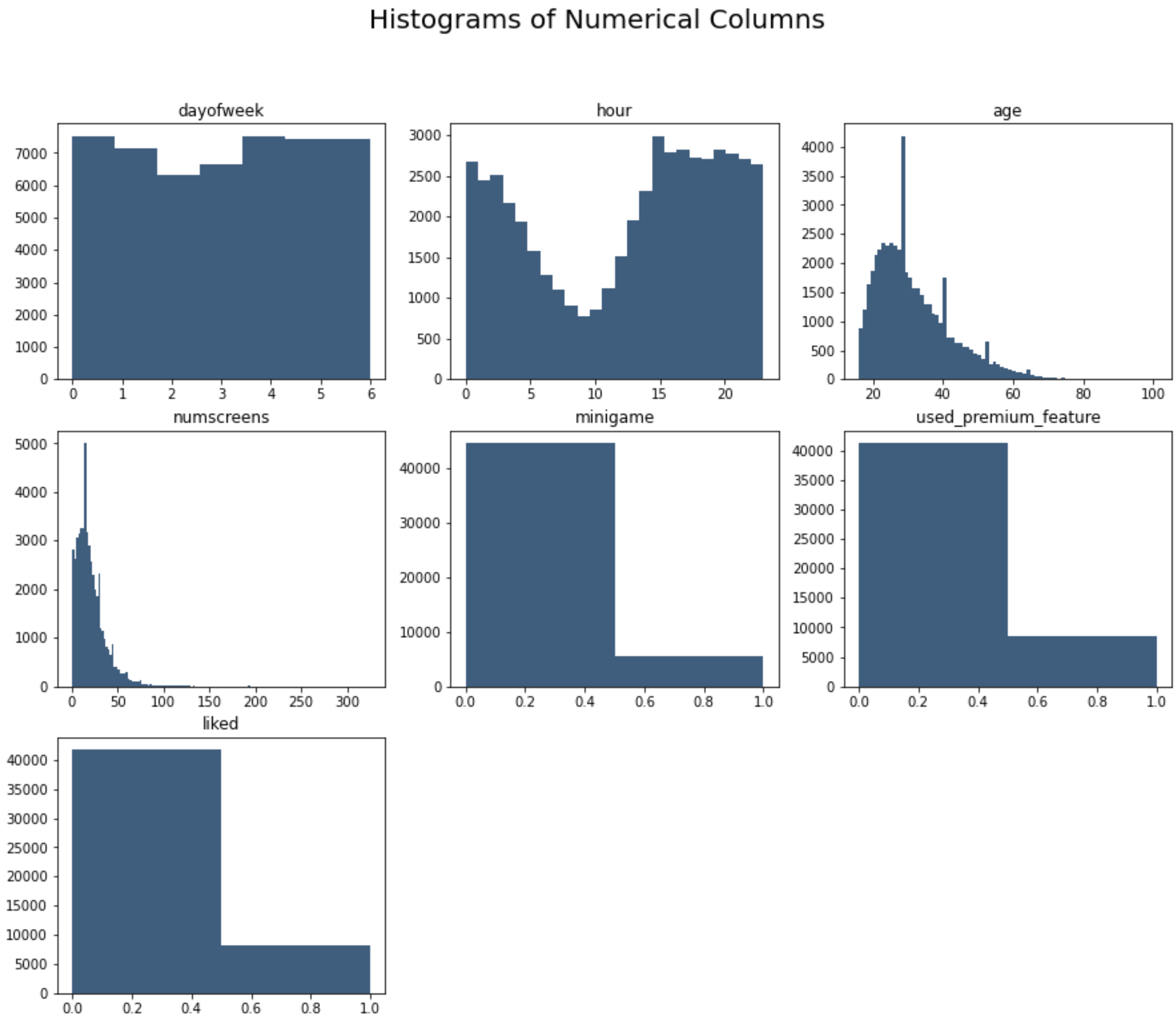
```
dataset_copy = dataset.copy().drop(columns = ['user', 'screen_list', 'enrolled_date'
dataset_copy.head()
```

Out[9]:

	dayofweek	hour	age	numscreens	minigame	used_premium_feature	liked
0	3	2	23	15	0	0	0
1	6	1	24	13	0	0	0
2	1	19	23	3	0	1	1
3	4	16	28	40	0	0	0
4	1	18	31	32	0	0	1

In [10]:

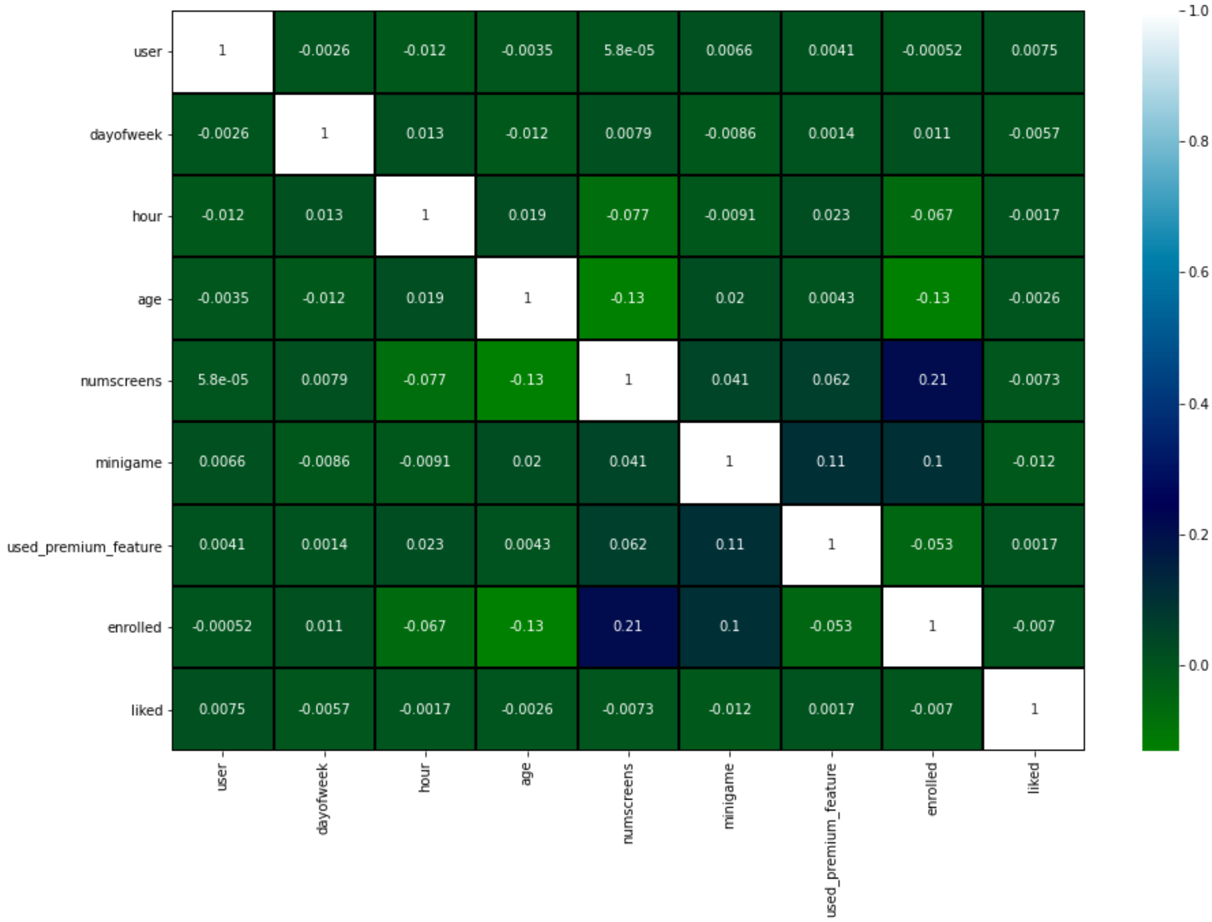
```
plt.figure(figsize=(15, 12))
plt.suptitle('Histograms of Numerical Columns', fontsize = 20)
for i in range(1, dataset_copy.shape[1] + 1):
    plt.subplot(3, 3, i)
    f = plt.gca()
    f.set_title(dataset_copy.columns.values[i - 1])
    vals = np.size(dataset_copy.iloc[:, i - 1].unique())
    plt.hist(dataset_copy.iloc[:, i - 1], bins = vals, color = '#3F5D7D')
#plt.savefig('Histograms of Numerical Columns.png')
```



In [11]:

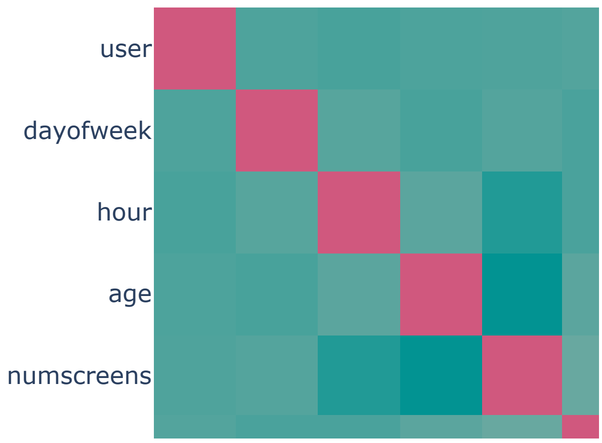
```
# Heatmap Using Seaborn
plt.figure(figsize = (15, 10))
sns.heatmap(dataset.corr(), annot = True, cmap = 'ocean', linewidths= 1, linecolor =
#plt.savefig('Heatmap Using Seaborn.png')
```

Out[11]: <AxesSubplot:>

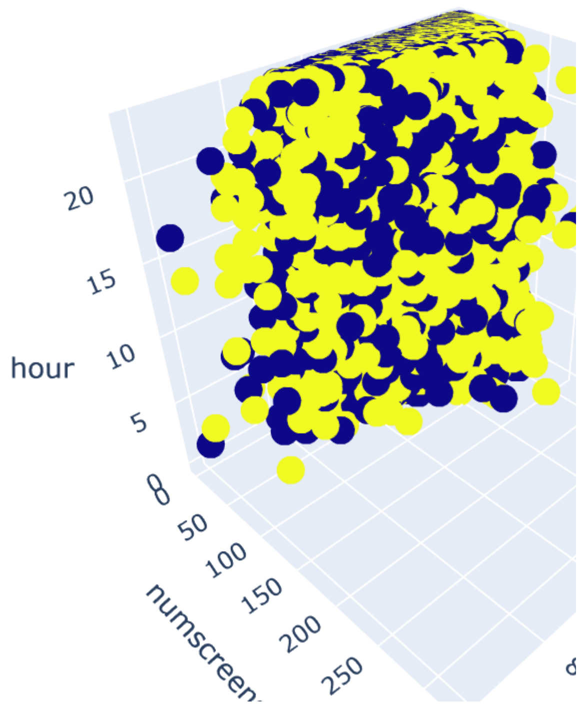


In [12]:

```python
# Heatmap Using Plotly
px.imshow(dataset.corr(), color_continuous_scale = 'tealrose')
```



In [13]:
```
# Just trying a 3d Scatter PLot
px.scatter_3d(dataset, 'age', 'numscreens', 'hour', color = 'enrolled')
```



Feature Engineering

Response

In [14]:
```
dataset.dtypes
```

Out[14]:
```
user                      int64
first_open               object
dayofweek                 int64
hour                      int32
age                       int64
screen_list              object
numscreens                int64
minigame                  int64
used_premium_feature      int64
enrolled                  int64
enrolled_date            object
liked                     int64
dtype: object
```

In [15]:
```
dataset['first_open'] = [parser.parse(row_data) for row_data in dataset['first_open'
dataset['enrolled_date'] = [parser.parse(row_data)if isinstance(row_data, str) else
```

In [16]:
```
dataset.dtypes
```

Out[16]:
```
user                               int64
first_open                datetime64[ns]
dayofweek                          int64
hour                               int32
age                                int64
screen_list                       object
numscreens                         int64
minigame                           int64
used_premium_feature               int64
enrolled                           int64
enrolled_date             datetime64[ns]
liked                              int64
dtype: object
```

In [17]:
```
dataset['difference'] = (dataset.enrolled_date - dataset.first_open).astype('timedel
```

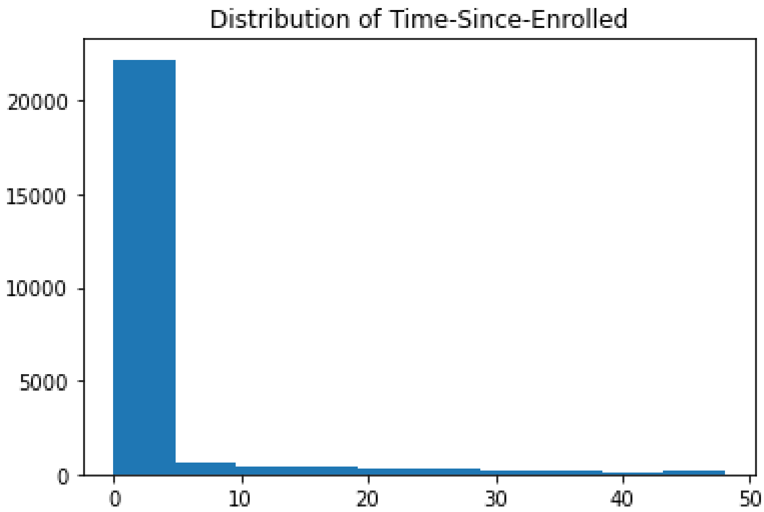
In [18]:
```
dataset.head(20)
```

Out[18]:

	user	first_open	dayofweek	hour	age	screen_list	nu
0	235136	2012-12-27 02:14:51.273	3	2	23	idscreen,joinscreen,Cycle,product_review,ScanP...	
1	333588	2012-12-02 01:16:00.905	6	1	24	joinscreen,product_review,product_review2,Scan...	
2	254414	2013-03-19 19:19:09.157	1	19	23	Splash,Cycle,Loan	
3	234192	2013-07-05 16:08:46.354	4	16	28	product_review,Home,product_review,Loan3,Finan...	
4	51549	2013-02-26 18:50:48.661	1	18	31	idscreen,joinscreen,Cycle,Credit3Container,Sca...	
5	56480	2013-04-03 09:58:15.752	2	9	20	idscreen,Cycle,Home,ScanPreview,VerifyPhone,Ve...	
6	144649	2012-12-25 02:33:18.461	1	2	35	product_review,product_review2,ScanPreview	
7	249366	2012-12-11 03:07:49.875	1	3	26	Splash,Cycle,Home,Credit3Container,Credit3Dash...	
8	372004	2013-03-20 14:22:01.569	2	14	29	product_review,product_review2,ScanPreview,Ver...	
9	338013	2013-04-26 18:22:16.013	4	18	26	Home,Loan2,product_review,product_review,produ...	
10	43555	2013-05-14 04:48:27.597	1	4	39	Splash,idscreen,Home,RewardsContainer,Settings...	
11	317454	2013-05-28 11:07:07.358	1	11	32	product_review,Home,Loan2,Credit3Container,Ver...	
12	205375	2012-12-17 06:28:45.903	0	6	25	idscreen,joinscreen,Cycle,product_review,produ...	
13	307608	2013-05-25 19:52:31.798	5	19	23	Alerts,ProfilePage,Home,Credit3Container	
14	359855	2013-02-18 04:48:48.912	0	4	17	joinscreen,product_review,product_review2,Scan...	

	user	first_open	dayofweek	hour	age	screen_list	nu
15	284938	2013-02-02 18:41:35.724	5	18	25	idscreen,joinscreen,Cycle,Loan2,product_review...	
16	235143	2013-07-07 16:07:35.057	6	16	21	product_review,product_review,product_review,p...	
17	141402	2013-02-02 21:12:46.888	5	21	55	joinscreen,Cycle,product_review,Loan2,product_...	
18	257945	2013-05-10 05:59:43.405	4	5	32	Splash,product_review,Home,Loan2,product_revie...	
19	54931	2013-07-06 17:34:46.439	5	17	25	idscreen,Loan3,product_review,product_review,Home	

In [19]:
```
plt.hist(dataset['difference'].dropna(), range = [0, 48])
plt.title('Distribution of Time-Since-Enrolled')
plt.show()
#plt.savefig('Distribution of Time Since Enrolled.png')
```



In [20]:
```
# First we remove every user who took more than 48 hours to enroll (mark them as 0).
# Next we remove some columns which no longer serve the purpose
dataset.loc[dataset.difference > 48, 'enrolled'] = 0
dataset = dataset.drop(columns = ['difference', 'enrolled_date', 'first_open'])
```

Screen

In [21]:
```
top_screens = pd.read_csv('top_screens.csv').top_screens.values
```

In [22]:
```
dataset['screen_list'] = dataset['screen_list'].astype(str) + ','
```

In [23]:
```
for sc in top_screens:
    dataset[sc] = dataset.screen_list.str.contains(sc).astype(int)
    dataset['screen_list'] = dataset['screen_list'].str.replace(sc + ',', '')
```

In [24]:
```
dataset['other'] = dataset['screen_list'].str.count(',')
dataset = dataset.drop(columns = ['screen_list'])
```

In [25]:
```
saving_screens = ['Saving1', 'Saving2', 'Saving2Amount', 'Saving4','Saving5','Saving
```

In [26]:
```
dataset['SavingsCount'] = dataset[saving_screens].sum(axis = 1)
dataset = dataset.drop(columns = saving_screens)
```

In [27]:
```
cm_screens = ['Credit1', 'Credit2','Credit3','Credit3Container','Credit3Dashboard']
dataset['CMCount'] = dataset[cm_screens].sum(axis = 1)
dataset = dataset.drop(columns = cm_screens)
```

In [28]:
```
cc_screens = ['CC1', 'CC1Category','CC3']
dataset['CCCount'] = dataset[cc_screens].sum(axis = 1)
dataset = dataset.drop(columns = cc_screens)
```

In [29]:
```
loan_screens = ['Loan', 'Loan2','Loan3','Loan4']
dataset['LoansCount'] = dataset[loan_screens].sum(axis = 1)
dataset = dataset.drop(columns = loan_screens)
```

In [30]:
```
dataset.head()
```

Out[30]:

	user	dayofweek	hour	age	numscreens	minigame	used_premium_feature	enrolled	liked
0	235136	3	2	23	15	0	0	0	0
1	333588	6	1	24	13	0	0	0	0
2	254414	1	19	23	3	0	1	0	1
3	234192	4	16	28	40	0	0	1	0
4	51549	1	18	31	32	0	0	1	1

5 rows × 50 columns

In [31]:
```
dataset.columns
```

Out[31]:
```
Index(['user', 'dayofweek', 'hour', 'age', 'numscreens', 'minigame',
       'used_premium_feature', 'enrolled', 'liked', 'location', 'Institutions',
       'VerifyPhone', 'BankVerification', 'VerifyDateOfBirth', 'ProfilePage',
       'VerifyCountry', 'Cycle', 'idscreen', 'Splash', 'RewardsContainer',
       'EditProfile', 'Finances', 'Alerts', 'Leaderboard', 'VerifyMobile',
       'VerifyHousing', 'RewardDetail', 'VerifyHousingAmount',
       'ProfileMaritalStatus', 'ProfileChildren ', 'ProfileEducation',
       'ProfileEducationMajor', 'Rewards', 'AccountView', 'VerifyAnnualIncome',
       'VerifyIncomeType', 'ProfileJobTitle', 'Login',
       'ProfileEmploymentLength', 'WebView', 'SecurityModal', 'ResendToken',
       'TransactionList', 'NetworkFailure', 'ListPicker', 'other',
       'SavingsCount', 'CMCount', 'CCCount', 'LoansCount'],
      dtype='object')
```

In [32]:
```
dataset.to_csv('new_appdata10.csv', index = False)
```

Model Building

In [33]:
```
dataset = pd.read_csv('new_appdata10.csv')
```

In [34]:
```
response = dataset['enrolled']
dataset = dataset.drop(columns = 'enrolled')
```

In [35]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset, response, test_size = 0
```

In [36]:
```
# We don't need the user identifier column for our model right now but will need it
# for each user. So we save it first and then remove it from our X_train & X_test co
train_identifier = X_train['user']
X_train = X_train.drop(columns = 'user')

test_identifier = X_test['user']
X_test = X_test.drop(columns = 'user')
```

In [37]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [38]:
```
X_train2 = pd.DataFrame(sc.fit_transform(X_train))
X_test2 = pd.DataFrame(sc.transform(X_test))
```

In [39]:
```
X_train2.columns = X_train.columns.values
X_test2.columns = X_test.columns.values
```

In [40]:
```
X_train2.index = X_train.index.values
X_test2.index = X_test.index.values
```

In [41]:
```
X_train = X_train2
X_test = X_test2
```

In [42]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, penalty = 'l1', solver = 'liblinea
```

In [43]:
```
classifier.fit(X_train, y_train)
```

Out[43]:
```
LogisticRegression(penalty='l1', random_state=0, solver='liblinear')
```

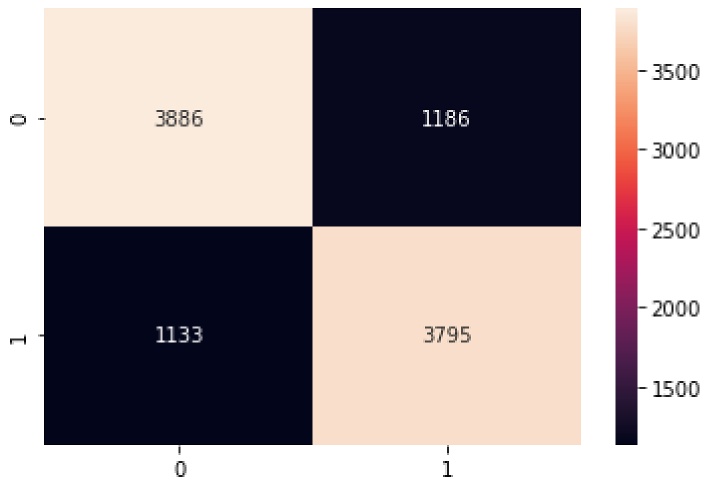
In [44]:
```
y_pred = classifier.predict(X_test)
```

Results

In [45]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_sc
```

In [46]:
```
# Confusion Matrix & Accuracy Score
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot = True, fmt = 'g')
accuracy_score(y_test, y_pred)
#plt.savefig('Confusion Matrix.png')
```

Out[46]:
```
0.7681
```



In [47]:
```
# Precision Score
precision_score(y_test, y_pred)
```

Out[47]: 0.7618952017667135

In [48]:
```
# Recall Score
recall_score(y_test, y_pred)
```

Out[48]: 0.7700892857142857

In [49]:
```
# F1 Score
f1_score(y_test, y_pred)
```

Out[49]: 0.7659703300030276

In [50]:
```
# A full Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.77      0.77      5072
           1       0.76      0.77      0.77      4928

    accuracy                           0.77     10000
   macro avg       0.77      0.77      0.77     10000
weighted avg       0.77      0.77      0.77     10000
```

In [51]:
```
# K-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv =
print("Logistic Regression Mean Accuracy: %0.3f" % (accuracies.mean()))
print("Logistic Regression Standard Deviation: %0.3f" % (accuracies.std() * 2))
```

```
Logistic Regression Mean Accuracy: 0.767
Logistic Regression Standard Deviation: 0.009
```

In [52]:
```
final_results = pd.concat([y_test, test_identifier], axis = 1)
final_results['predicted_results'] = y_pred
final_results[['user', 'enrolled', 'predicted_results']].reset_index(drop = True)
```

Out[52]:

	user	enrolled	predicted_results
0	239786	1	1
1	279644	1	1
2	98290	0	0
3	170150	1	1
4	237568	1	1
...	...	...	...
9995	143036	1	0
9996	91158	1	1
9997	248318	0	0
9998	142418	1	1
9999	279355	1	1

10000 rows × 3 columns