In [793]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from dateutil import parser
```

1 Obtaining Data

In [794]:

```
!ls
```

```
CleanAppData.csv
P39-CS3-Data
P39-CS3-Python-Code
P39-CS3-Python-Code.zip
Predict Users more likely to pay subscription.ipynb
Untitled.ipynb
```

In [795]:

```
data = pd.read_csv("P39-CS3-Data/appdata10.csv")
```

In [796]:

```
data.describe
```

Out[796]:

```
<bound method NDFrame.describe of          user               first_o
pen  dayofweek       hour  age  \
0       235136  2012-12-27 02:14:51.273           3   02:00:00   23
1       333588  2012-12-02 01:16:00.905           6   01:00:00   24
2       254414  2013-03-19 19:19:09.157           1   19:00:00   23
3       234192  2013-07-05 16:08:46.354           4   16:00:00   28
4        51549  2013-02-26 18:50:48.661           1   18:00:00   31
5        56480  2013-04-03 09:58:15.752           2   09:00:00   20
6       144649  2012-12-25 02:33:18.461           1   02:00:00   35
7       249366  2012-12-11 03:07:49.875           1   03:00:00   26
8       372004  2013-03-20 14:22:01.569           2   14:00:00   29
9       338013  2013-04-26 18:22:16.013           4   18:00:00   26
10       43555  2013-05-14 04:48:27.597           1   04:00:00   39
11      317454  2013-05-28 11:07:07.358           1   11:00:00   32
12      205375  2012-12-17 06:28:45.903           0   06:00:00   25
13      307608  2013-05-25 19:52:31.798           5   19:00:00   23
14      359855  2013-02-18 04:48:48.912           0   04:00:00   17
15      284938  2013-02-02 18:41:35.724           5   18:00:00   25
```

In [797]:

```
data['hour'].head()
```

Out[797]:

```
0     02:00:00
1     01:00:00
2     19:00:00
3     16:00:00
4     18:00:00
Name: hour, dtype: object
```

In [798]:

```
data.columns
```

Out[798]:

```
Index(['user', 'first_open', 'dayofweek', 'hour', 'age', 'screen_lis
t',
       'numscreens', 'minigame', 'used_premium_feature', 'enrolled',
       'enrolled_date', 'liked'],
      dtype='object')
```

In [799]:

```
data['hour'] = data.hour.str.slice(1,3).astype(int)
data['hour'].head()
```

Out[799]:

```
0     2
1     1
2    19
3    16
4    18
Name: hour, dtype: int64
```

In [800]:

```
data.dtypes
```

Out[800]:

```
user                    int64
first_open             object
dayofweek               int64
hour                    int64
age                     int64
screen_list            object
numscreens              int64
minigame                int64
used_premium_feature    int64
enrolled                int64
enrolled_date          object
liked                   int64
dtype: object
```

In [801]:

```
numdata = data.copy().drop(['user','screen_list','enrolled_date','first_open','enrol
                           axis=1)
```

In [802]:

```
numdata.head()
```

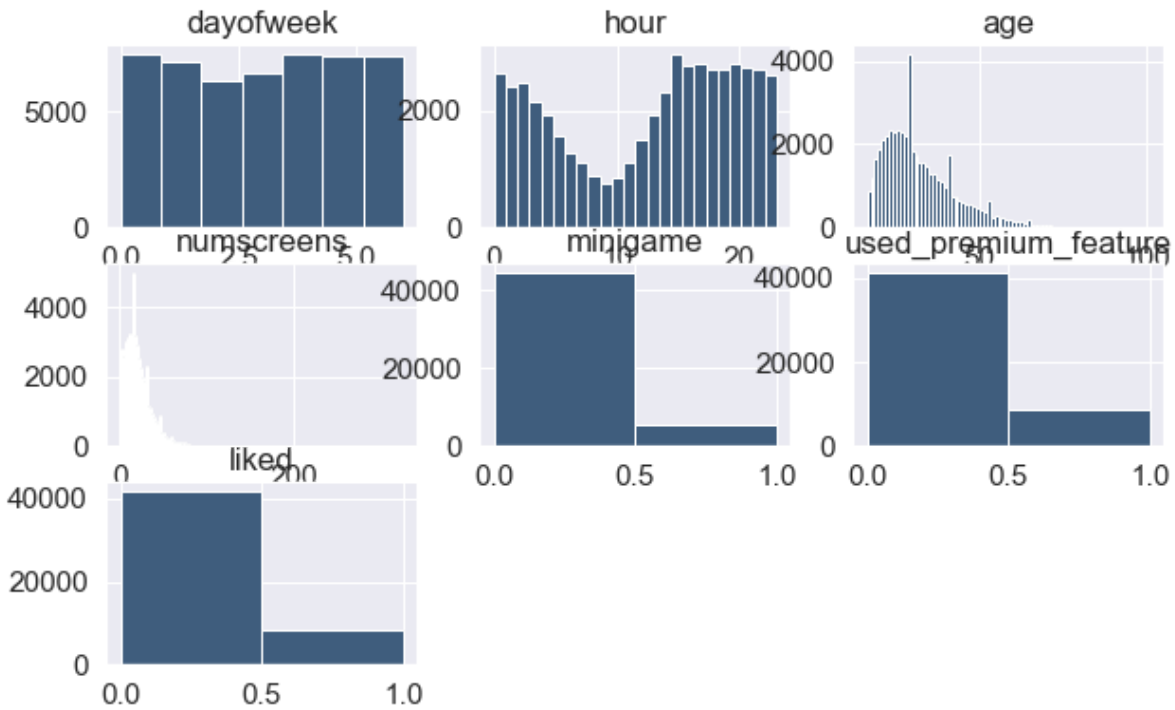
Out[802]:

	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

2 Data Visualization

In [803]:

```
plt.figure(figsize=(10,6))
for i in range(numdata.shape[1]):
    plt.subplot(3,3,i+1)
    f=plt.gca()
    f.set_title(numdata.columns.values[i])
    valu = np.size(numdata.iloc[:,i].unique())
    plt.hist(numdata.iloc[:, i], bins=valu, color='#3F5D7D')
```



In [804]:

```
numdata.corrwith(data.enrolled)
```

Out[804]:

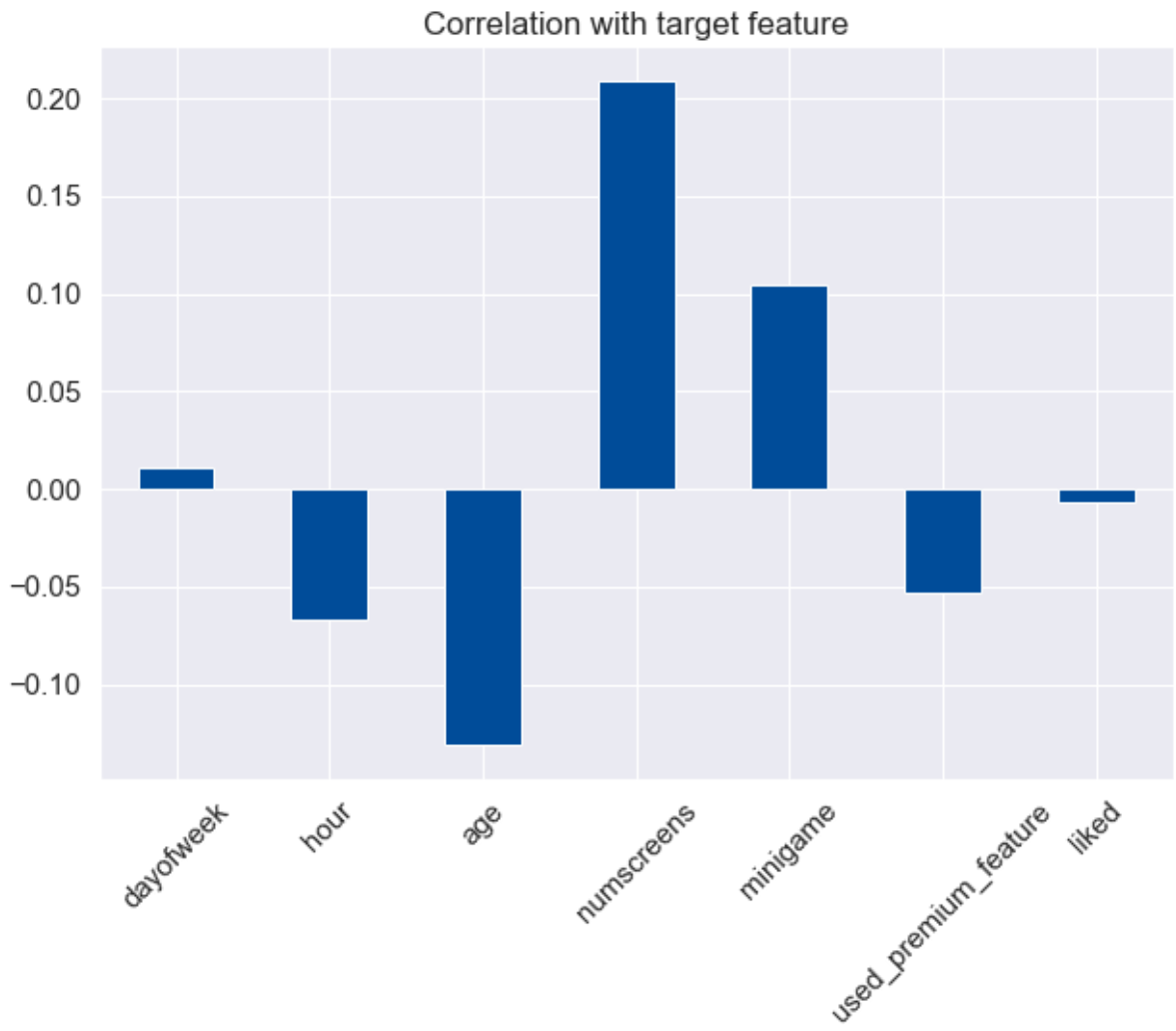
```
dayofweek               0.011326
hour                   -0.066694
age                    -0.131303
numscreens              0.209457
minigame                0.104979
used_premium_feature   -0.052703
liked                  -0.007022
dtype: float64
```

In [805]:

```
numdata.corrwith(data.enrolled).plot.bar(figsize=(10,7),
                                         title='Correlation with target feature',
                                         fontsize=15,rot=45,
                                         grid=True,
                                         color='#004C99')
```

Out[805]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16e70bef0>
```



In [806]:

```
numdata.shape
```

Out[806]:

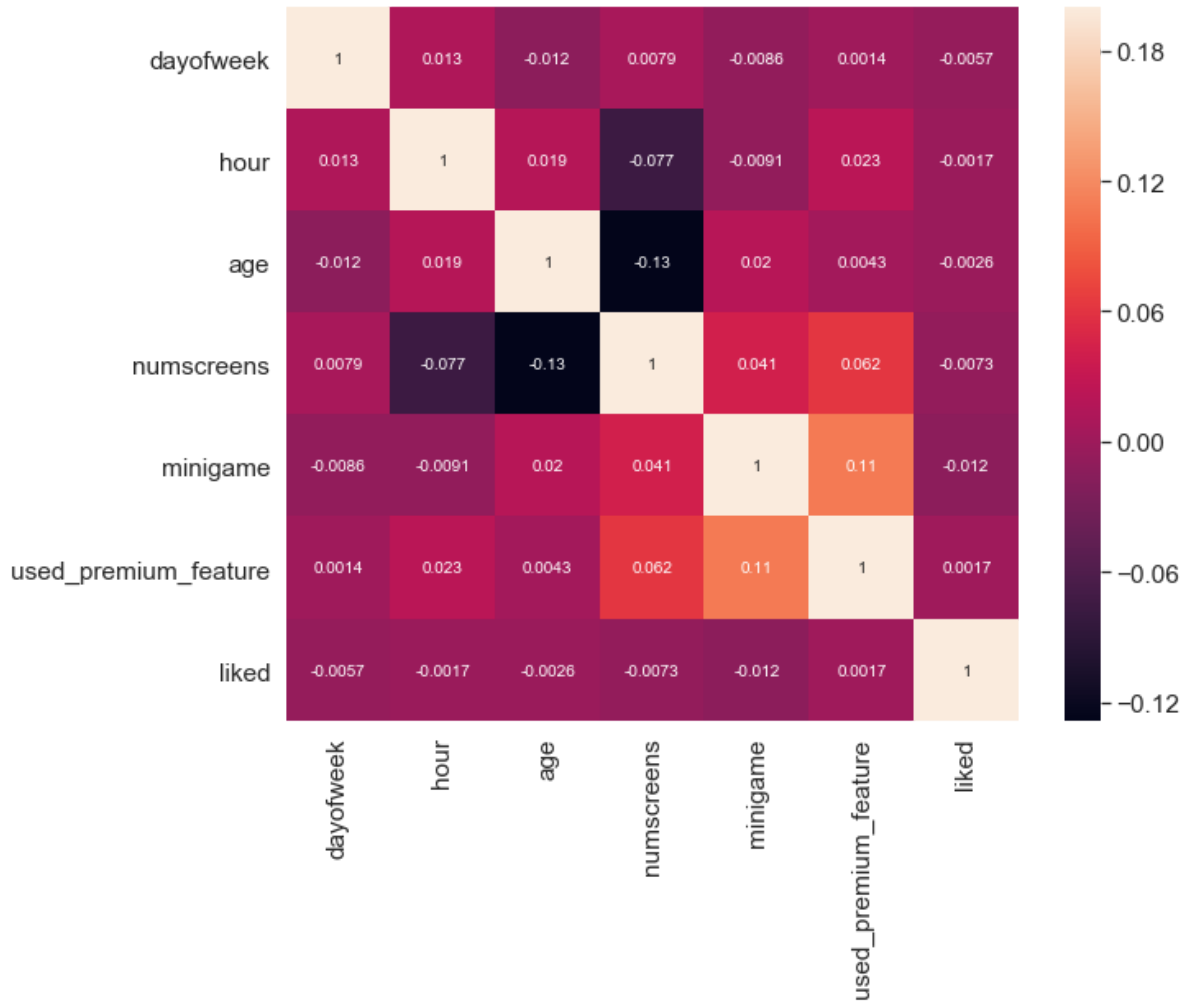
```
(50000, 7)
```

In [807]:

```
import seaborn as sn
plt.figure(figsize=(10,8))
sn.heatmap(numdata.corr(),annot=True,vmax=0.2)
```

Out[807]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16e53be80>
```



In [808]:

```
data.dtypes
```

Out[808]:

```
user                     int64
first_open              object
dayofweek                int64
hour                     int64
age                      int64
screen_list             object
numscreens               int64
minigame                 int64
used_premium_feature     int64
enrolled                 int64
enrolled_date           object
liked                    int64
dtype: object
```

In [809]:

```
data['first_open'].head(2)
```

Out[809]:

```
0    2012-12-27 02:14:51.273
1    2012-12-02 01:16:00.905
Name: first_open, dtype: object
```

In [810]:

```
data['first_open'] = [parser.parse(i) for i in data['first_open']]
```

In [811]:

```
data['first_open'].head(2)  # we just converted it to date type
```

Out[811]:

```
0   2012-12-27 02:14:51.273
1   2012-12-02 01:16:00.905
Name: first_open, dtype: datetime64[ns]
```

In [812]:

```
data['enrolled_date'].head()
```

Out[812]:

```
0                       NaN
1                       NaN
2                       NaN
3    2013-07-05 16:11:49.513
4    2013-02-26 18:56:37.841
Name: enrolled_date, dtype: object
```

In [813]:

```
# some are string and some are NaN hence transformation should be conditional
data['enrolled_date'] = [parser.parse(i) if isinstance(i, str) else i for i in data[
```

In [814]:

```
data['enrolled_date'].head(2)
```

Out[814]:

```
0   NaT
1   NaT
Name: enrolled_date, dtype: datetime64[ns]
```

In [815]:

```
# Lets calculate how much time does the customers took to enroll
# enrolled date - first open
data['difference'] = (data.enrolled_date - data.first_open)
```

In [816]:

```
data['difference'].head()
```

Out[816]:

```
0                NaT
1                NaT
2                NaT
3   00:03:03.159000
4   00:05:49.180000
Name: difference, dtype: timedelta64[ns]
```

In [817]:

```
data['difference']=data['difference'].astype('timedelta64[h]')
# converted to number of Hours
```

In [818]:

```
data['difference'].tail()
```

Out[818]:

```
49995    0.0
49996    NaN
49997    NaN
49998    0.0
49999    NaN
Name: difference, dtype: float64
```

In [819]:

```
np.sum(data.isnull())
```

Out[819]:

```
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
difference              18926
dtype: int64
```

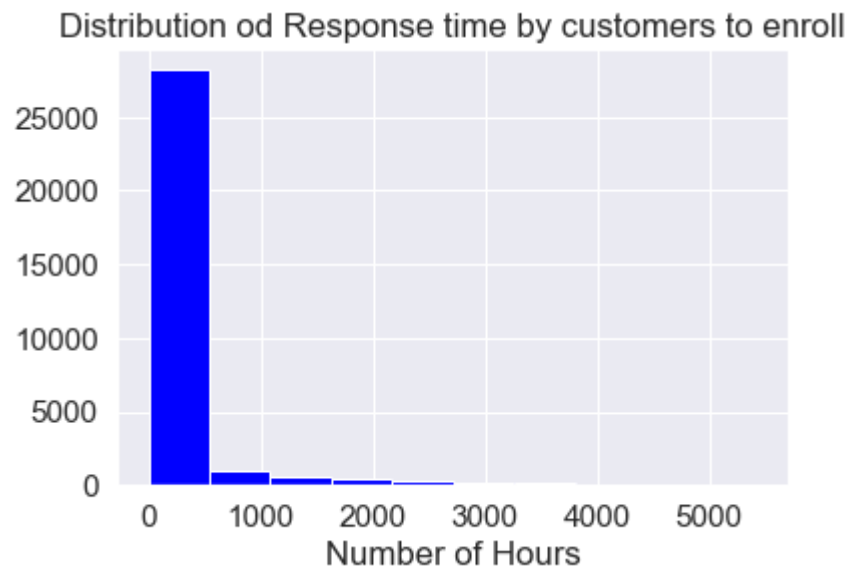
In [820]:

```
# out of 50000 18926 people did not enroll yet.
# we also need to check within what time people enrolled to our app
plt.hist(data['difference'], color='blue')
plt.title("Distribution of Response time by customers to enroll")
plt.xlabel("Number of Hours")
```

```
/usr/local/lib/python3.7/site-packages/numpy/lib/histograms.py:824: Ru
ntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/usr/local/lib/python3.7/site-packages/numpy/lib/histograms.py:825: Ru
ntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

Out[820]:

```
Text(0.5, 0, 'Number of Hours')
```

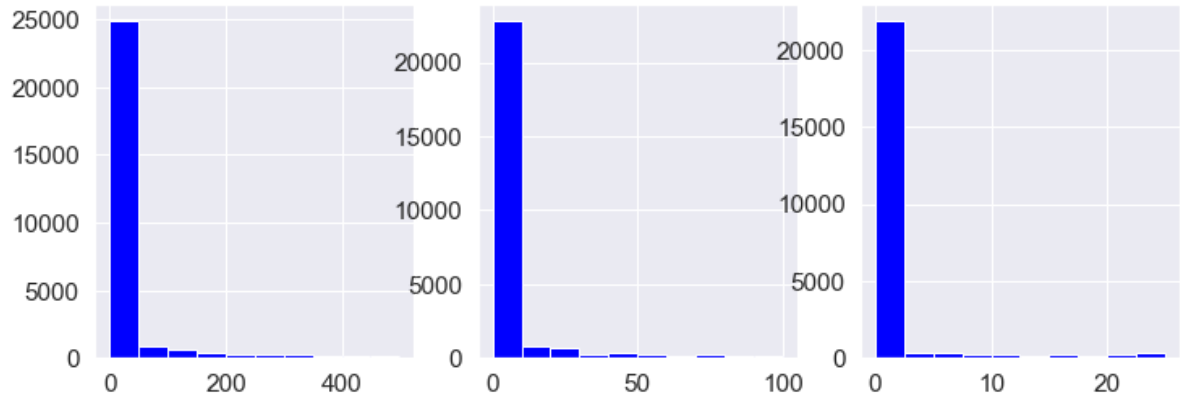


In [821]:

```
plt.figure(figsize=(12,4))
plt.subplot(1,3,1)
plt.hist(data['difference'], color='blue', range=[0,500])
plt.subplot(1,3,2)
plt.hist(data['difference'], color='blue', range=[0,100])
plt.subplot(1,3,3)
plt.hist(data['difference'], color='blue', range=[0,25])
```

Out[821]:

```
(array([21847.,   366.,   371.,   209.,   248.,   144.,   230.,   13
3.,
          222.,   299.]),
 array([ 0. ,  2.5,  5. ,  7.5, 10. , 12.5, 15. , 17.5, 20. , 22.5, 2
5. ]),
 <a list of 10 Patch objects>)
```



In [822]:

```
#majority of people enrolled in the app within 5 hours of first time usage
data[data['difference'] > 48 ].dropna().shape
```

Out[822]:

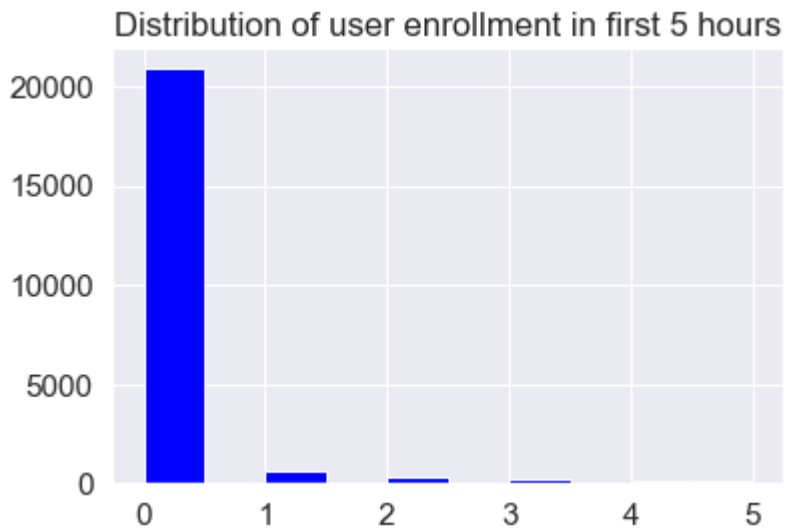
```
(6224, 13)
```

In [823]:

```
plt.title("Distribution of user enrollment in first 5 hours")
plt.hist(data['difference'], color='blue', range=[0,5])
```

Out[823]:

```
(array([20885.,     0.,   625.,     0.,   337.,     0.,   207.,
0.,
          159.,   129.]),
 array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ]),
 <a list of 10 Patch objects>)
```



3 Data Cleaning

In [824]:

```
data.loc[data['difference']>48,'enrolled'] = 0
# for greater than 48 hours (2 days) all enrolled data is considered not enrolled
```

In [825]:

```
topscreendata=pd.read_csv('P39-CS3-Data/top_screens.csv').top_screens
topscreendata.head()
```

Out[825]:

```
0               Loan2
1            location
2        Institutions
3    Credit3Container
4         VerifyPhone
Name: top_screens, dtype: object
```

In [ ]:

```

```

In [ ]:

```

```

In [827]:

```
# Eaerlier we saw the number of screens are highly correlated to the decision of en
# now we want to consider the screen names that user used.
# We only want to consider first 50 or Top screens
```

In [828]:

```
# Adding a comma at the end of screenlist column, this will helpin replacing the te
data['screen_list'] = data.screen_list.astype(str)+','
```

In [829]:

```
for eachscreen in topscreendata:
    data[eachscreen] = data.screen_list.str.contains(eachscreen).astype(int)
    data['screen_list'] = data.screen_list.str.replace(eachscreen+",","")
```

In [830]:

```
data.shape
```

Out[830]:

```
(50000, 71)
```

In [831]:

```
# There are multiple screens which are of similar type and they are highly correlat
# we want them to group into single column.
```

In [832]:

```
savings_screens = ["Saving1",
                   "Saving2",
                   "Saving2Amount",
                   "Saving4",
                   "Saving5",
                   "Saving6",
                   "Saving7",
                   "Saving8",
                   "Saving9",
                   "Saving10"]
cm_screens = ["Credit1",
              "Credit2",
              "Credit3",
              "Credit3Container",
              "Credit3Dashboard"]
cc_screens = ["CC1",
              "CC1Category",
              "CC3"]

loan_screens = ["Loan",
              "Loan2",
              "Loan3",
              "Loan4"]
```

In [833]:

```
data['savingcount'] = data[savings_screens].sum(axis=1)
data['cm_screenscount'] = data[cm_screens].sum(axis=1)
data['cc_screenscount'] = data[cc_screens].sum(axis=1)
data['loan_screenscount'] = data[loan_screens].sum(axis=1)
```

In [834]:

```
data = data.drop(savings_screens,axis=1)
data = data.drop(cm_screens,axis=1)
data = data.drop(cc_screens,axis=1)
data = data.drop(loan_screens,axis=1)
data = data.drop(['screen_list'],axis=1)
```

In [835]:

```
data.head()
```

Out[835]:

	user	first_open	dayofweek	hour	age	numscreens	minigame	used_premium_feature
0	235136	2012-12-27 02:14:51.273	3	2	23	15	0	0
1	333588	2012-12-02 01:16:00.905	6	1	24	13	0	0
2	254414	2013-03-19 19:19:09.157	1	19	23	3	0	1
3	234192	2013-07-05 16:08:46.354	4	16	28	40	0	0
4	51549	2013-02-26 18:50:48.661	1	18	31	32	0	0

5 rows × 52 columns

In [836]:

```
data.describe()
```

Out[836]:

	user	dayofweek	hour	age	numscreens	minigame	use
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	
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	

8 rows × 50 columns

In [837]:

```
data.columns
```

Out[837]:

```
Index(['user', 'first_open', 'dayofweek', 'hour', 'age', 'numscreens',
       'minigame', 'used_premium_feature', 'enrolled', 'enrolled_dat
e',
       'liked', 'difference', 'location', 'Institutions', 'VerifyPhon
e',
       'BankVerification', 'VerifyDateOfBirth', 'ProfilePage', 'Verify
Country',
       'Cycle', 'idscreen', 'Splash', 'RewardsContainer', 'EditProfil
e',
       'Finances', 'Alerts', 'Leaderboard', 'VerifyMobile', 'VerifyHou
sing',
       'RewardDetail', 'VerifyHousingAmount', 'ProfileMaritalStatus',
       'ProfileChildren ', 'ProfileEducation', 'ProfileEducationMajo
r',
       'Rewards', 'AccountView', 'VerifyAnnualIncome', 'VerifyIncomeTy
pe',
       'ProfileJobTitle', 'Login', 'ProfileEmploymentLength', 'WebVie
w',
       'SecurityModal', 'ResendToken', 'TransactionList', 'NetworkFail
ure',
       'ListPicker', 'savingcount', 'cm_screenscount', 'cc_screenscoun
t',
       'loan_screenscount'],
      dtype='object')
```

In [838]:

```
data.to_csv('CleanAppData.csv',index=False)
```

In [839]:

```
!ls
```

```
CleanAppData.csv
P39-CS3-Data
P39-CS3-Python-Code
P39-CS3-Python-Code.zip
Predict Users more likely to pay subscription.ipynb
Untitled.ipynb
```

4 Data Preprocessing

In [840]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [841]:

```
data = data.dropna()
response = data['enrolled']
data = data.drop(['enrolled','first_open','enrolled_date'],axis=1)
```

In [842]:

```
X_train,X_test,y_train,y_test = train_test_split(data,response,test_size=0.4,random_
```

In [843]:

```
# we need to get rid of userID before proceeeding further
userXtrain = X_train['user']
userXtest = X_test['user']
X_train = X_train.drop(['user'],axis=1)
X_test = X_test.drop(['user'],axis=1)
# we retained user id so as to compare post analysis to which user should we target
```

In [844]:

```
sc = StandardScaler()
X_train_new = pd.DataFrame(sc.fit_transform(X_train))
X_test_new = pd.DataFrame(sc.transform(X_test))
```

In [845]:

```
# We retreive the columnnames and indexes

X_train_new.columns = X_train.columns.values
X_test_new.columns = X_test.columns.values
X_train_new.index = X_train.index.values
X_test_new.index = X_test.index.values

X_train = X_train_new
del X_train_new
X_test = X_test_new
del X_test_new
```

5 Model Building

In [846]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='l1')
```

In [847]:

```
#np.sum(X_train.isnull())
```

In [848]:

```
clf.fit(X_train,y_train)
ypred = clf.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,ypred))
print(confusion_matrix(y_test,ypred))
```

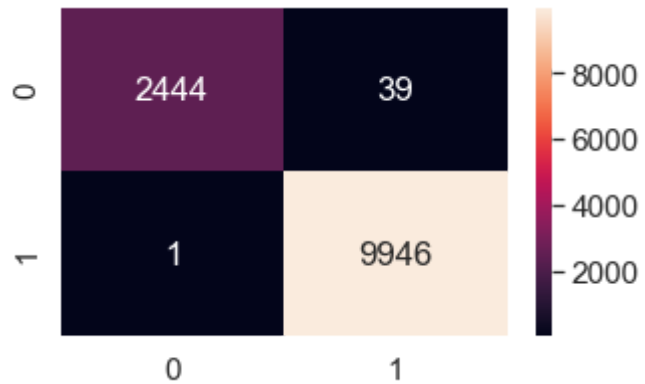
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2483
           1       1.00      1.00      1.00      9947

    accuracy                           1.00     12430
   macro avg       1.00      0.99      0.99     12430
weighted avg       1.00      1.00      1.00     12430

[[2444   39]
 [   1 9946]]
```

In [849]:

```
cm=confusion_matrix(y_test,ypred)
df_cm = pd.DataFrame(cm, index = (0, 1), columns = (0, 1))
plt.figure(figsize = (5,3))
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True, fmt='g')
```

Out[849]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bcccc88>
```



In [850]:

```
# tring K fold cross validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = clf,X=X_train,y=y_train,cv=10)
print("CV Accuracy: %0.2f with +/- %0.3f" % (accuracies.mean(), accuracies.std() * 2
```

```
CV Accuracy: 1.00 with +/- 0.003
```

In [851]:

```
coeficients = pd.DataFrame({'features': data.drop(columns='user').columns,
                            'Coefficients': np.transpose(clf.coef_).flatten()})
coeficients.head()
```

Out[851]:

	features	Coefficients
0	dayofweek	0.059027
1	hour	-0.008862
2	age	-0.045675
3	numscreens	0.000000
4	minigame	-0.069372

In [852]:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [856]:

```
from sklearn.model_selection import GridSearchCV
penalty = ['l1','l2']
C = [0.001, 0.01, 0.1, 1, 10]
parameters=dict(C=C,penalty=penalty)
gridSearch = GridSearchCV(estimator=clf,
                          param_grid=parameters,
                          scoring="accuracy",
                          cv=10,
                          n_jobs=-1)
```

In [857]:

```
import time
tstart=time.time()
gridSearch=gridSearch.fit(X_train,y_train)
tend=time.time()
print("It took %0.2f seconds"%(tend-tstart))
```

```
It took 65.29 seconds
```

In [858]:

```
bestaccuracy = gridSearch.best_score_
bestparam = gridSearch.best_params_
bestaccuracy, bestparam
```

Out[858]:

```
(0.9980154473288994, {'C': 10, 'penalty': 'l1'})
```

In [869]:

```
finalresult = pd.concat([X_test,userXtest],axis=1)
finalresult['predicted_reach'] = ypred
finalresult = finalresult[['user','predicted_reach']]
```

In [866]:

```
finalresult
```

Out[866]:

	user	predicted_reach
31569	48571	1
37260	360791	1
40921	251974	1
7351	107014	0
31204	131222	1
21167	366147	1
19942	341789	0
44349	319984	0
44409	207198	0
24703	139513	1
26954	168538	1
27154	171753	0
16748	8961	1
25021	237559	1
26247	356839	1
39235	221276	0
33141	275975	0
41495	373525	1
23203	96642	1
35892	296390	0
30961	67668	1
24259	297410	1
27204	78811	1
4311	8396	1
27782	65598	1
24416	255288	1
21465	40127	1
26091	183313	0
2874	173989	1
36034	28409	0
...	...	...
5953	268101	1
47533	211817	1

	user	predicted_reach
39192	302612	1
17628	372347	1
17322	111790	1
35747	76094	1
12589	248109	1
2913	142541	1
33791	54486	1
3798	175437	1
24650	369842	1
9732	298442	0
42089	176742	1
40275	112423	1
6904	248918	1
27051	182585	1
47901	360959	1
25575	195588	0
21223	233661	1
8854	32793	1
33286	234080	0
21136	322013	1
20155	26747	1
44005	342893	1
39238	284515	0
9341	208274	0
16732	294050	1
17877	82765	1
23156	369509	1
25002	91102	1

12430 rows × 2 columns

In [ ]: