SURUTHI S

225229141

SMA LAB 14

Click-Through Rate Prediction

In [1]:
```
import numpy as np
import random
n = 40428967  #total number of records in the clickstream data
sample_size = 1000000
skip_values = sorted(random.sample(range(1,n), n-sample_size))
```

In [2]:
```
types_tllrain = {
    'id': np.dtype(int),
    'click': np.dtype(int),
    'hour': np.dtype(int),
    'C1': np.dtype(int),
    'banner_pos': np.dtype(int),
    'site_id': np.dtype(str),
    'site_domain': np.dtype(str),
    'site_category': np.dtype(str),
    'app_id': np.dtype(str),
    'app_domain': np.dtype(str),
    'app_category': np.dtype(str),
    'device_id': np.dtype(str),
    'device_ip': np.dtype(str),
    'device_model': np.dtype(str),
    'device_type': np.dtype(int),
    'device_conn_type': np.dtype(int),
    'C14': np.dtype(int),
    'C15': np.dtype(int),
    'C16': np.dtype(int),
    'C17': np.dtype(int),
    'C18': np.dtype(int),
    'C19': np.dtype(int),
    'C20': np.dtype(int),
    'C21':np.dtype(int)
}

types_test = {
    'id': np.dtype(int),
    'hour': np.dtype(int),
    'C1': np.dtype(int),
    'banner_pos': np.dtype(int),
    'site_id': np.dtype(str),
    'site_domain': np.dtype(str),
    'site_category': np.dtype(str),
    'app_id': np.dtype(str),
    'app_domain': np.dtype(str),
    'app_category': np.dtype(str),
    'device_id': np.dtype(str),
    'device_ip': np.dtype(str),
    'device_model': np.dtype(str),
    'device_type': np.dtype(int),
    'device_conn_type': np.dtype(int),
    'C14': np.dtype(int),
    'C15': np.dtype(int),
    'C16': np.dtype(int),
    'C17': np.dtype(int),
    'C18': np.dtype(int),
    'C19': np.dtype(int),
    'C20': np.dtype(int),
    'C21':np.dtype(int)
}
```

In [3]:
```
import pandas as pd
import gzip

parse_date = lambda val : pd.datetime.strptime(val, '%y%m%d%H')

with gzip.open('train.gz') as f:
    train = pd.read_csv(f, parse_dates = ['hour'], date_parser = parse_date, dt

train.head()
```

```
C:\Users\2mscdsa41\AppData\Local\Temp\ipykernel_16100\1409201024.py:4: Future
Warning: The pandas.datetime class is deprecated and will be removed from pan
das in a future version. Import from datetime module instead.
  parse_date = lambda val : pd.datetime.strptime(val, '%y%m%d%H')
```

Out[3]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4
3	230718000	1	2014-10-21	1002	0	84c7ba46	c4e18dd6	50e219e0	ecad2386
4	175637501	0	2014-10-21	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386

5 rows × 24 columns

In [4]:
```
train.shape
```

Out[4]:
```
(1000000, 24)
```

In [5]:
```
train.dtypes
```

Out[5]:
```
id                            int32
click                         int32
hour                 datetime64[ns]
C1                            int32
banner_pos                    int32
site_id                      object
site_domain                  object
site_category                object
app_id                       object
app_domain                   object
app_category                 object
device_id                    object
device_ip                    object
device_model                 object
device_type                   int32
device_conn_type              int32
C14                           int32
C15                           int32
C16                           int32
C17                           int32
C18                           int32
C19                           int32
C20                           int32
C21                           int32
dtype: object
```

Target feature -> click

site features -> site_id, site_domain, site_category

app feature -> app_id, app_domain, app_category

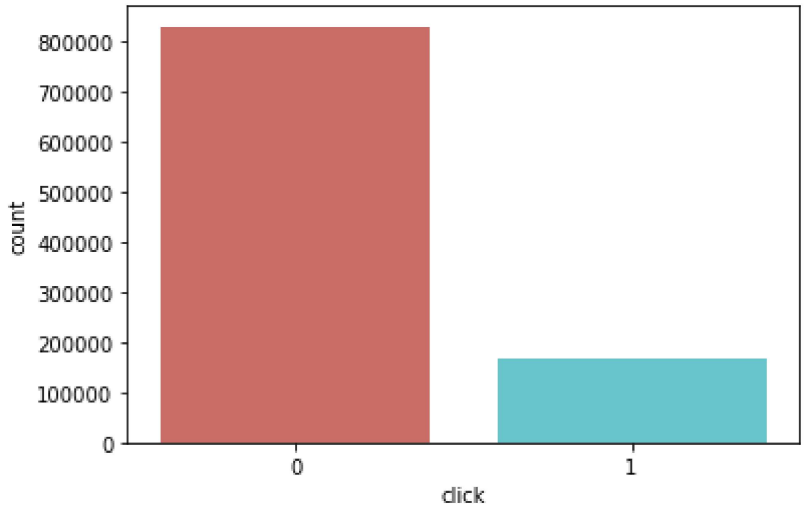
device feature -> device_id, device_ip, device_model, device_type, device_conn_type

anonymized categorical features -> C14-C21

In [6]:
```
train['click'].value_counts()
```

Out[6]:
```
0    830514
1    169486
Name: click, dtype: int64
```

In [7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(x='click',data=train, palette='hls')
plt.show();
```



In [8]:
```
train['click'].value_counts()/len(train)
```

Out[8]:
```
0    0.830514
1    0.169486
Name: click, dtype: float64
```

Click through rate is approx. 17%, and approx. 83% is not clicked.

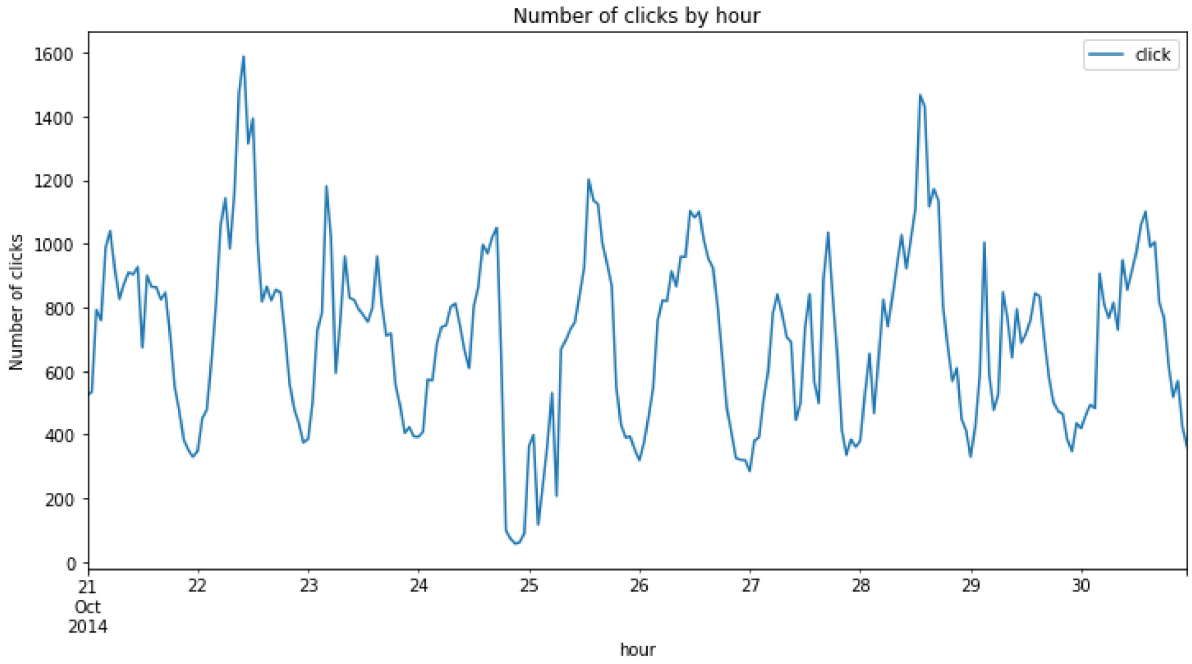
In [9]:
```
train.hour.describe()
```

```
C:\Users\2mscdsa41\AppData\Local\Temp\ipykernel_16100\1468264.py:1: FutureWar
ning: Treating datetime data as categorical rather than numeric in `.describe
` is deprecated and will be removed in a future version of pandas. Specify `d
atetime_is_numeric=True` to silence this warning and adopt the future behavio
r now.
  train.hour.describe()
```

Out[9]:
```
count                 1000000
unique                    240
top       2014-10-22 09:00:00
freq                    11015
first     2014-10-21 00:00:00
last      2014-10-30 23:00:00
Name: hour, dtype: object
```

The data covers 10 days of click streams data from 2014-10-21 to 2014-10-30, that is 240 hours.

In [10]:
```
train.groupby('hour').agg({'click':'sum'}).plot(figsize=(12,6))
plt.ylabel('Number of clicks')
plt.title('Number of clicks by hour');
```



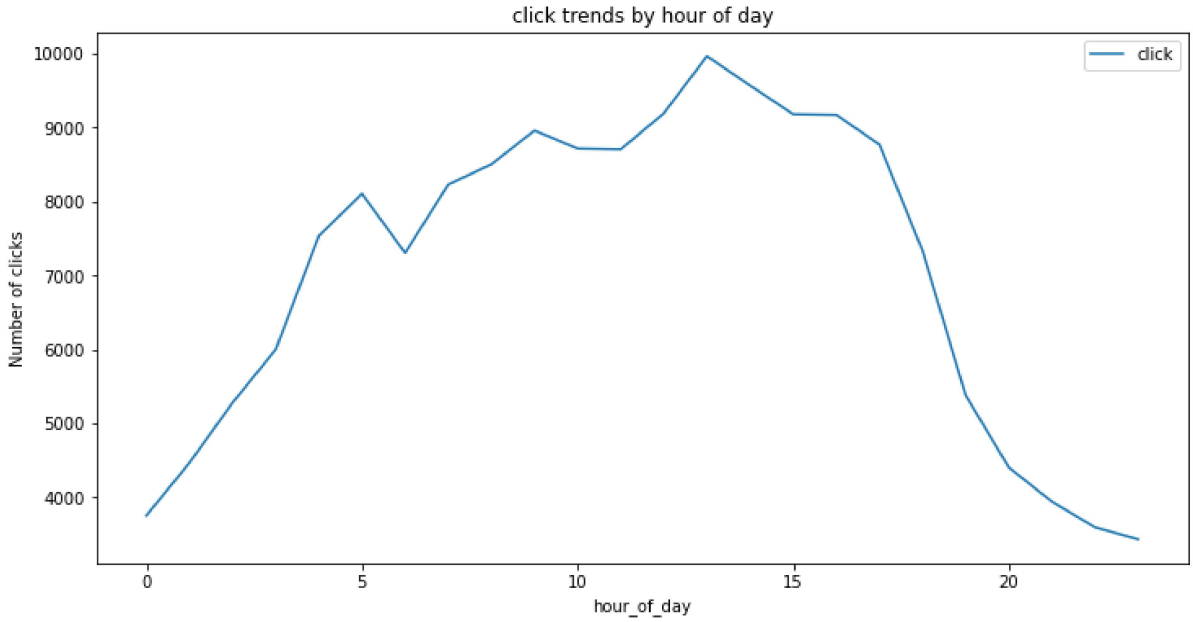
The hourly clicks pattern looks pretty similar every day. However, there were a couple of peak hours, one is sometime in the mid of the day on Oct 22, and another is sometime in the mid of the day on Oct 28. And one off-peak hour is close to mid-night on Oct 24.

Feature engineering for date time features

Hour

Extract hour from date time feature.

In [11]:
```
train['hour_of_day'] = train.hour.apply(lambda x: x.hour)
train.groupby('hour_of_day').agg({'click':'sum'}).plot(figsize=(12,6))
plt.ylabel('Number of clicks')
plt.title('click trends by hour of day');
```



In general, the highest number of clicks is at hour 13 and 14 (1pm and 2pm), and the least number of clicks is at hour 0 (mid-night). It seems a useful feature for roughly estimation.

In [12]:
```
train.head(3)
```

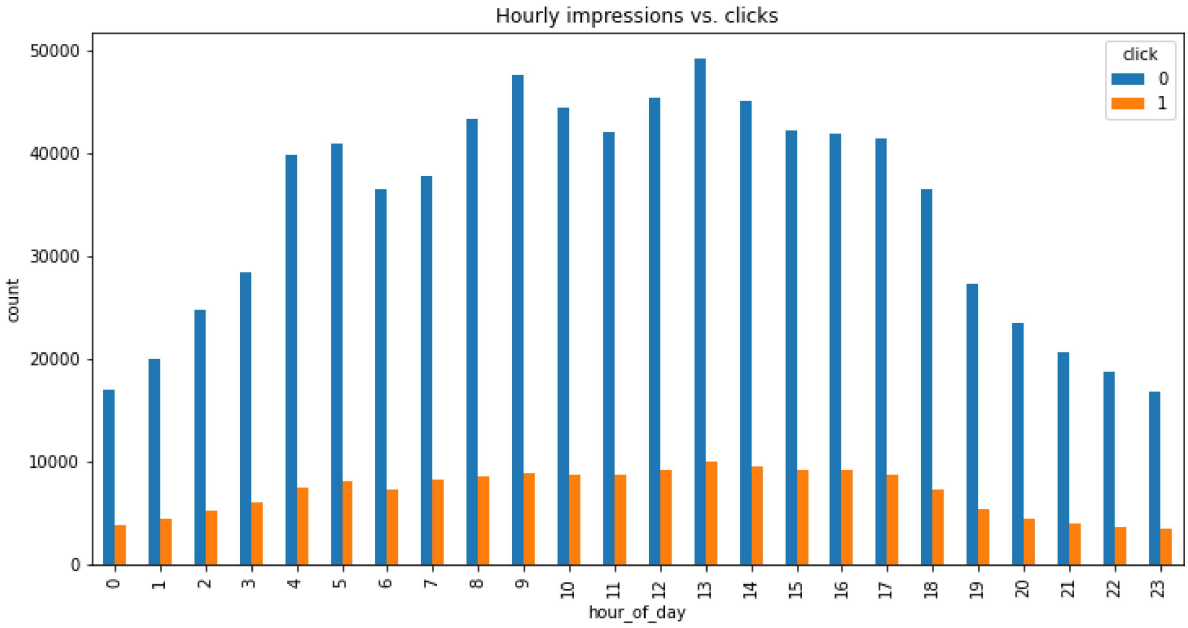
Out[12]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4

3 rows × 25 columns

Let's take impressions into consideration.

In [13]:
```
train.groupby(['hour_of_day', 'click']).size().unstack().plot(kind='bar', title
plt.ylabel('count')
plt.title('Hourly impressions vs. clicks');
```



There is nothing shocking here.

In [14]:
```
train.head(3)
```

Out[14]:

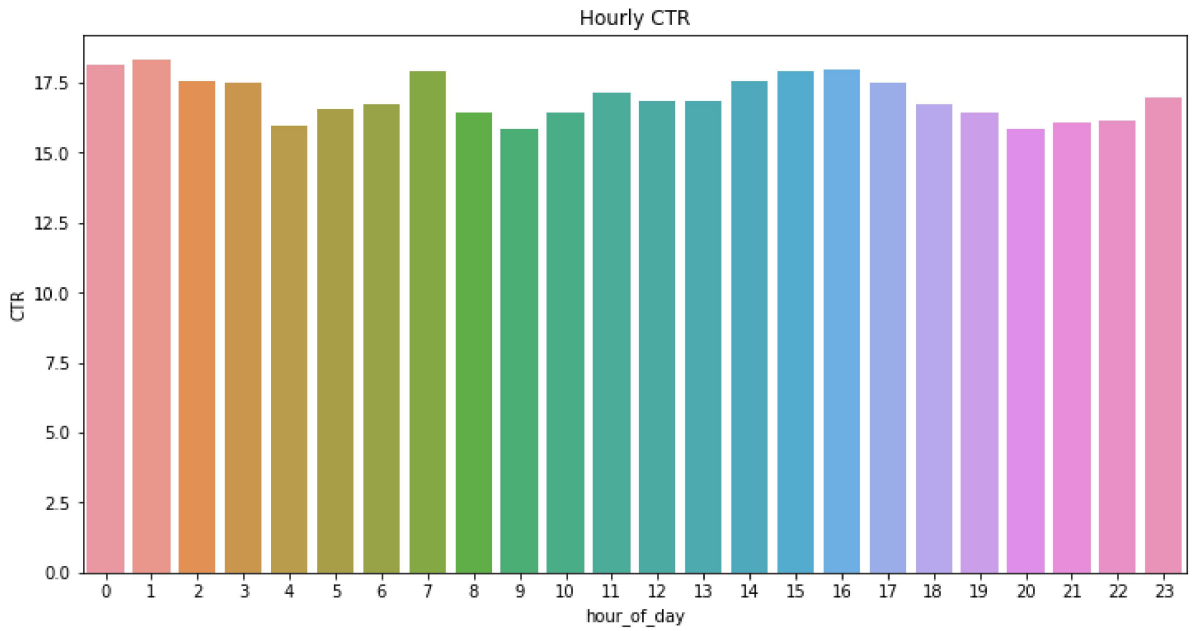
	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4

3 rows × 25 columns

Now that we have click and impression. We can calculate Click-through rate (CTR). CTR is the ratio of ad clicks to impressions. It measures the rate of clicks on each ad.

Hourly CTR

In [15]:
```
import seaborn as sns

df_click = train[train['click'] == 1]
df_hour = train[['hour_of_day','click']].groupby(['hour_of_day']).count().reset
df_hour = df_hour.rename(columns={'click': 'impressions'})
df_hour['clicks'] = df_click[['hour_of_day','click']].groupby(['hour_of_day']).
df_hour['CTR'] = df_hour['clicks']/df_hour['impressions']*100

plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='hour_of_day', data=df_hour)
plt.title('Hourly CTR');
```



One of the interesting observations here is that the highest CTR happened in the hour of mid-night, 1, 7 and 15. If you remember, around mid-night has the least number of impressions and clicks.

Day of week CTR

In [16]:
```
train.head(3)
```

Out[16]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4

3 rows × 25 columns

While Tuesdays and Wednesdays have the highest number of impressions and clicks, their CTR are among the lowest. Saturdays and Sundays enjoy the highest CTR. Apparently, people have more time to click over the weekend.

In [17]:
```
train.head(3)
```

Out[17]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4

3 rows × 25 columns

C1 feature

C1 is one of the anonymized categorical features. Although we don't know its meaning, we still want to have a look its distribution.

In [18]:
```
print(train.C1.value_counts()/len(train))
```

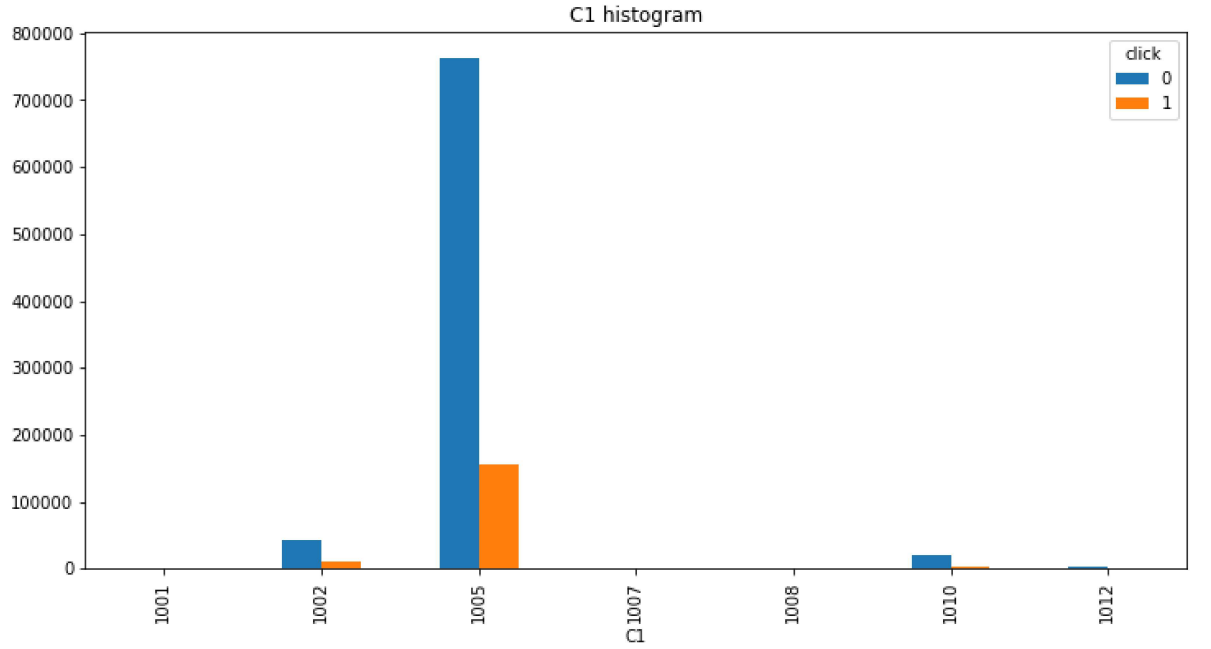
```
1005    0.918603
1002    0.054882
1010    0.022479
1012    0.002766
1007    0.000870
1001    0.000261
1008    0.000139
Name: C1, dtype: float64
```

C1 value = 1005 has the most data, almost 92%. Let's see whether we can find value of C1 indicates something about CTR.

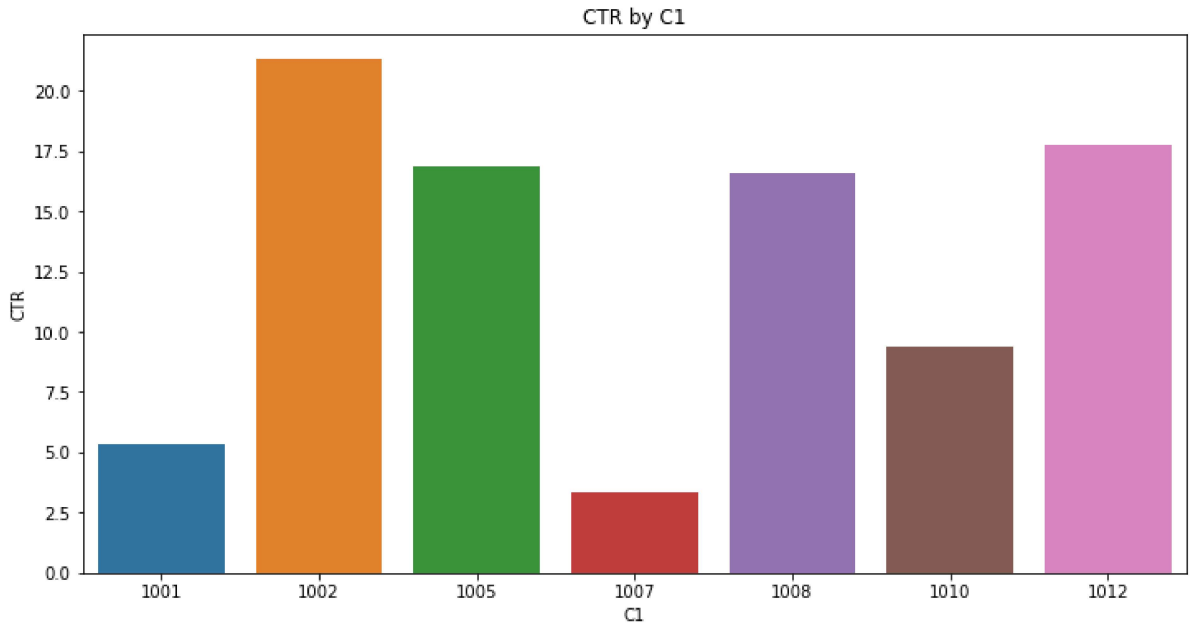
In [19]:
```
C1_values = train.C1.unique()
C1_values.sort()
ctr_avg_list=[]
for i in C1_values:
    ctr_avg=train.loc[np.where((train.C1 == i))].click.mean()
    ctr_avg_list.append(ctr_avg)
    print("for C1 value: {},  click through rate: {}".format(i,ctr_avg))
```

```
for C1 value: 1001,  click through rate: 0.05363984674329502
for C1 value: 1002,  click through rate: 0.21300244160198242
for C1 value: 1005,  click through rate: 0.16887382253269367
for C1 value: 1007,  click through rate: 0.03333333333333333
for C1 value: 1008,  click through rate: 0.16546762589928057
for C1 value: 1010,  click through rate: 0.09395435740024022
for C1 value: 1012,  click through rate: 0.17715112075198844
```

In [20]:
```
train.groupby(['C1', 'click']).size().unstack().plot(kind='bar', figsize=(12,6)
```



In [21]:
```
df_c1 = train[['C1','click']].groupby(['C1']).count().reset_index()
df_c1 = df_c1.rename(columns={'click': 'impressions'})
df_c1['clicks'] = df_click[['C1','click']].groupby(['C1']).count().reset_index(
df_c1['CTR'] = df_c1['clicks']/df_c1['impressions']*100

plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='C1', data=df_c1)
plt.title('CTR by C1');
```



The average CTR in the data is 0.17.

In [22]:
```
train['click'].mean()
```

Out[22]:
```
0.169486
```

In [23]:
```
df_c1.CTR.describe()
```

Out[23]:
```
count     7.000000
mean     12.934608
std       6.874272
min       3.333333
25%       7.379710
50%      16.546763
75%      17.301247
max      21.300244
Name: CTR, dtype: float64
```

The important C1 values and CTR pairs are:

1005: 92% of the data and 0.17 CTR

1002: 5.5% of the data and 0.21 CTR

1010: 2.3% of the data and 0.096 CTR

1002 has a much higher than average CTR, and 1010 has a much lower than average CTR, it seems these two C1 values are important for predicting CTR.

Banner position

I have heard that there are many factors that affect the performance of your banner ads, but the most influential one is the banner position. Let's see whether it is true.

In [24]:
```
print(train.banner_pos.value_counts()/len(train))
```

```
0    0.719486
1    0.278725
7    0.001071
2    0.000318
4    0.000219
5    0.000139
3    0.000042
Name: banner_pos, dtype: float64
```

In [25]:
```
banner_pos = train.banner_pos.unique()
banner_pos.sort()
ctr_avg_list=[]
for i in banner_pos:
    ctr_avg=train.loc[np.where((train.banner_pos == i))].click.mean()
    ctr_avg_list.append(ctr_avg)
    print("for banner position: {},  click through rate: {}".format(i,ctr_avg))
```

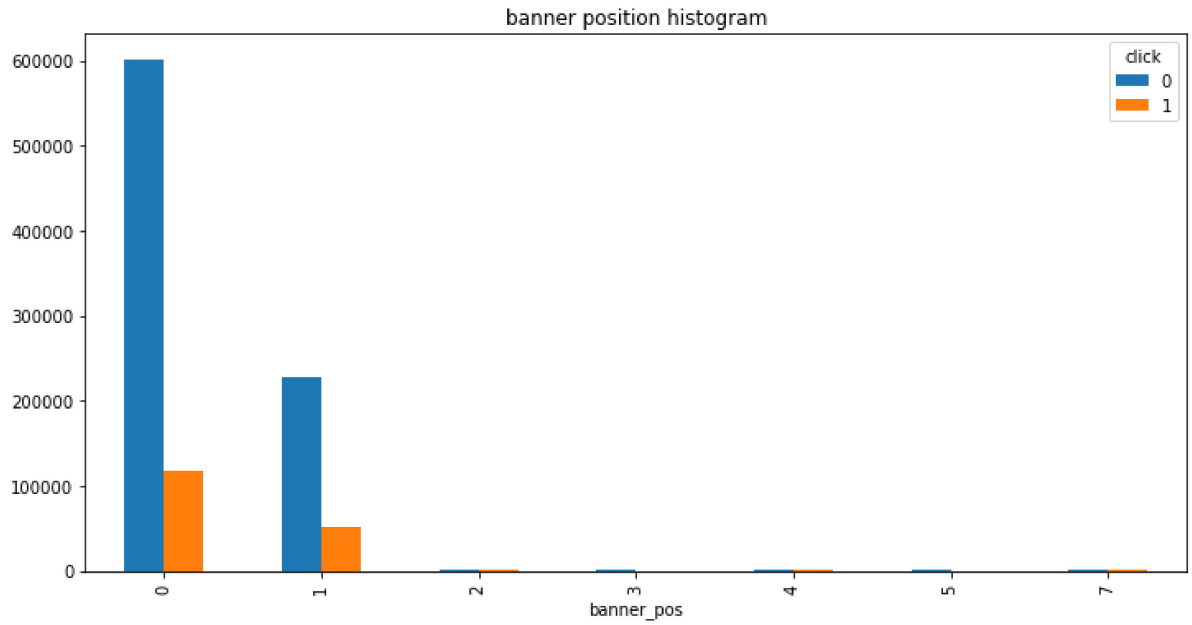
```
for banner position: 0,  click through rate: 0.16400458104813714
for banner position: 1,  click through rate: 0.18306933357251773
for banner position: 2,  click through rate: 0.11949685534591195
for banner position: 3,  click through rate: 0.11904761904761904
for banner position: 4,  click through rate: 0.1963470319634703
for banner position: 5,  click through rate: 0.16546762589928057
for banner position: 7,  click through rate: 0.32866479925303455
```

The important banner positions are:

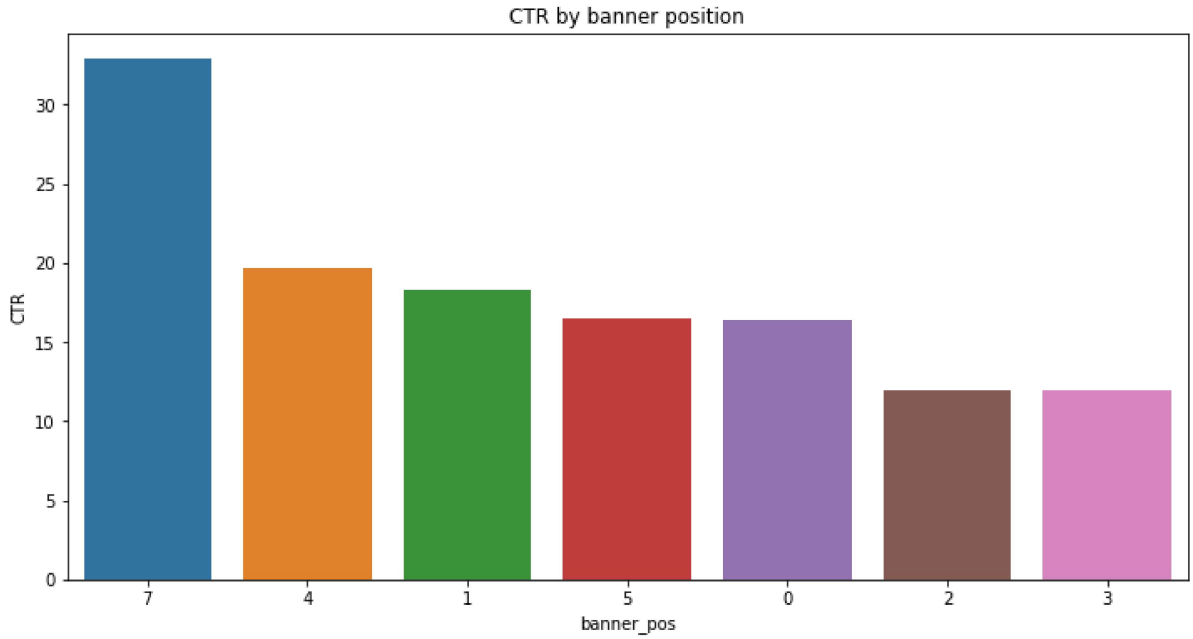
position 0: 72% of the data and 0.16 CTR

position 1: 28% of the data and 0.18 CTR

In [26]:
```
train.groupby(['banner_pos', 'click']).size().unstack().plot(kind='bar', figsiz
```



In [27]:
```
df_banner = train[['banner_pos','click']].groupby(['banner_pos']).count().reset
df_banner = df_banner.rename(columns={'click': 'impressions'})
df_banner['clicks'] = df_click[['banner_pos','click']].groupby(['banner_pos']).
df_banner['CTR'] = df_banner['clicks']/df_banner['impressions']*100
sort_banners = df_banner.sort_values(by='CTR',ascending=False)['banner_pos'].to
plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='banner_pos', data=df_banner, order=sort_banners)
plt.title('CTR by banner position');
```



In [28]:
```
df_banner.CTR.describe()
```

Out[28]:
```
count     7.000000
mean     18.229969
std       7.092686
min      11.904762
25%      14.175072
50%      16.546763
75%      18.970818
max      32.866480
Name: CTR, dtype: float64
```

Although banner position 0 and 1 have the highest number of impressions and clicks, banner 7 enjoys the highest click through rate. Increasing the number of ads placed on banner position 7 seems to be a good idea.

Site features

site id

In [29]:
```
print("There are {} sites in the data set".format(train.site_id.nunique()))
```

```
There are 2640 sites in the data set
```

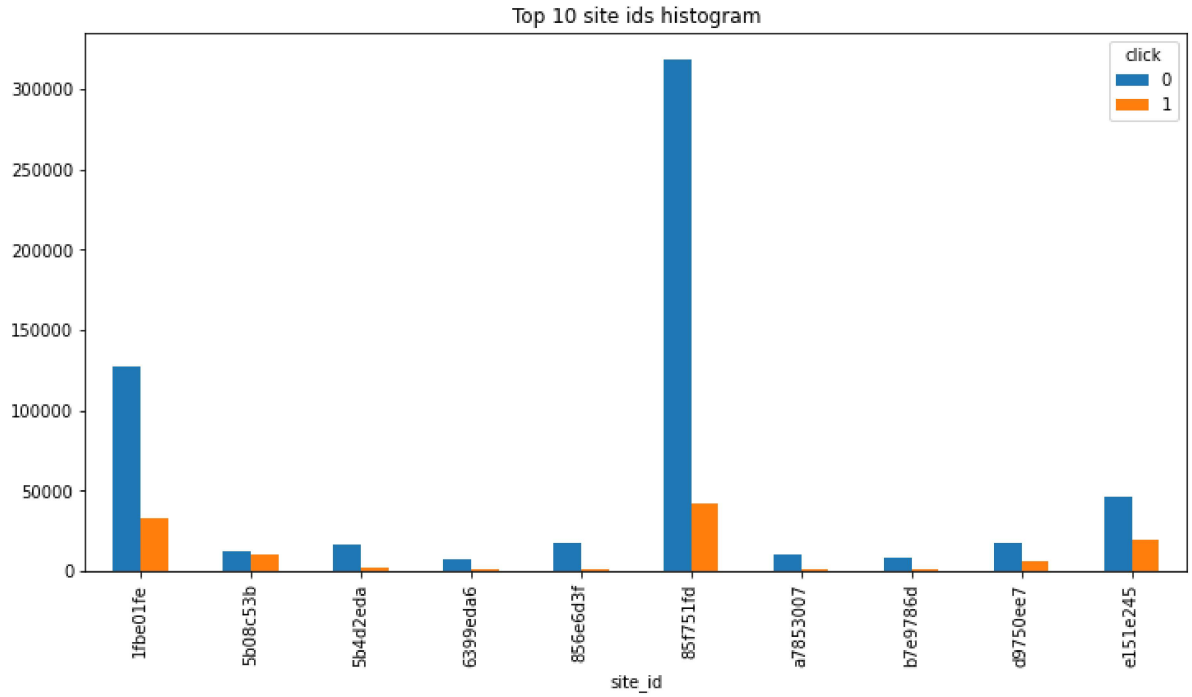
In [30]:
```
print('The top 10 site ids that have the most impressions')
print((train.site_id.value_counts()/len(train))[0:10])
```

```
The top 10 site ids that have the most impressions
85f751fd    0.361338
1fbe01fe    0.160358
e151e245    0.065364
d9750ee7    0.023719
5b08c53b    0.022490
5b4d2eda    0.018937
856e6d3f    0.018886
a7853007    0.011538
b7e9786d    0.009233
6399eda6    0.008628
Name: site_id, dtype: float64
```

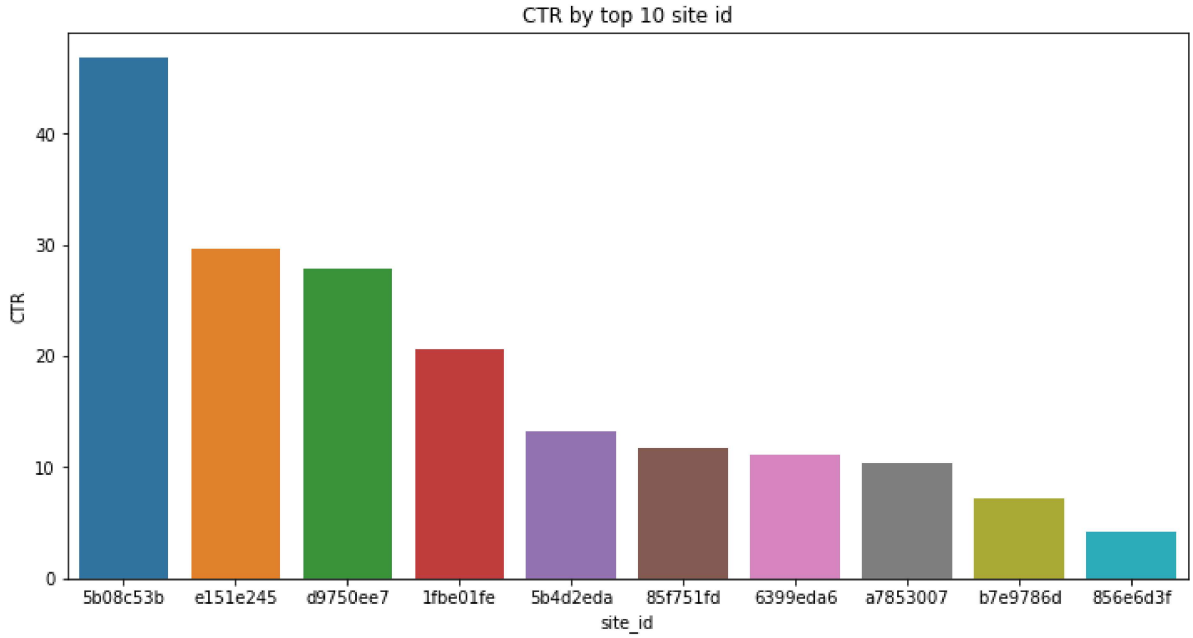
In [31]:
```
top10_ids = (train.site_id.value_counts()/len(train))[0:10].index
click_avg_list=[]

for i in top10_ids:
    click_avg=train.loc[np.where((train.site_id == i))].click.mean()
    click_avg_list.append(click_avg)
    print("for site id value: {},  click through rate: {}".format(i,click_avg))
```

```
for site id value: 85f751fd,  click through rate: 0.11748833502150341
for site id value: 1fbe01fe,  click through rate: 0.20593297496850796
for site id value: e151e245,  click through rate: 0.2967229667706995
for site id value: d9750ee7,  click through rate: 0.278763860196467
for site id value: 5b08c53b,  click through rate: 0.46745220097821255
for site id value: 5b4d2eda,  click through rate: 0.13180546021017056
for site id value: 856e6d3f,  click through rate: 0.04177697765540612
for site id value: a7853007,  click through rate: 0.1031374588316866
for site id value: b7e9786d,  click through rate: 0.07159103216722626
for site id value: 6399eda6,  click through rate: 0.11114974501622624
```

In [32]:
```
top10_sites = train[(train.site_id.isin((train.site_id.value_counts()/len(train
top10_sites_click = top10_sites[top10_sites['click'] == 1]
top10_sites.groupby(['site_id', 'click']).size().unstack().plot(kind='bar', fig
```



In [33]:
```
df_site = top10_sites[['site_id','click']].groupby(['site_id']).count().reset_i
df_site = df_site.rename(columns={'click': 'impressions'})
df_site['clicks'] = top10_sites_click[['site_id','click']].groupby(['site_id'])
df_site['CTR'] = df_site['clicks']/df_site['impressions']*100
sort_site = df_site.sort_values(by='CTR',ascending=False)['site_id'].tolist()
plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='site_id', data=df_site, order=sort_site)
plt.title('CTR by top 10 site id');
```



The site_id feature seems important because the CTR for the top 10 sites, some as high as 0.47, and some as low as 0.04, and they are significantly different from overall CTR 0.16.

site domain

In [34]:
```
print("There are {} site domains in the data set".format(train.site_domain.nuni
```

```
There are 2874 site domains in the data set
```

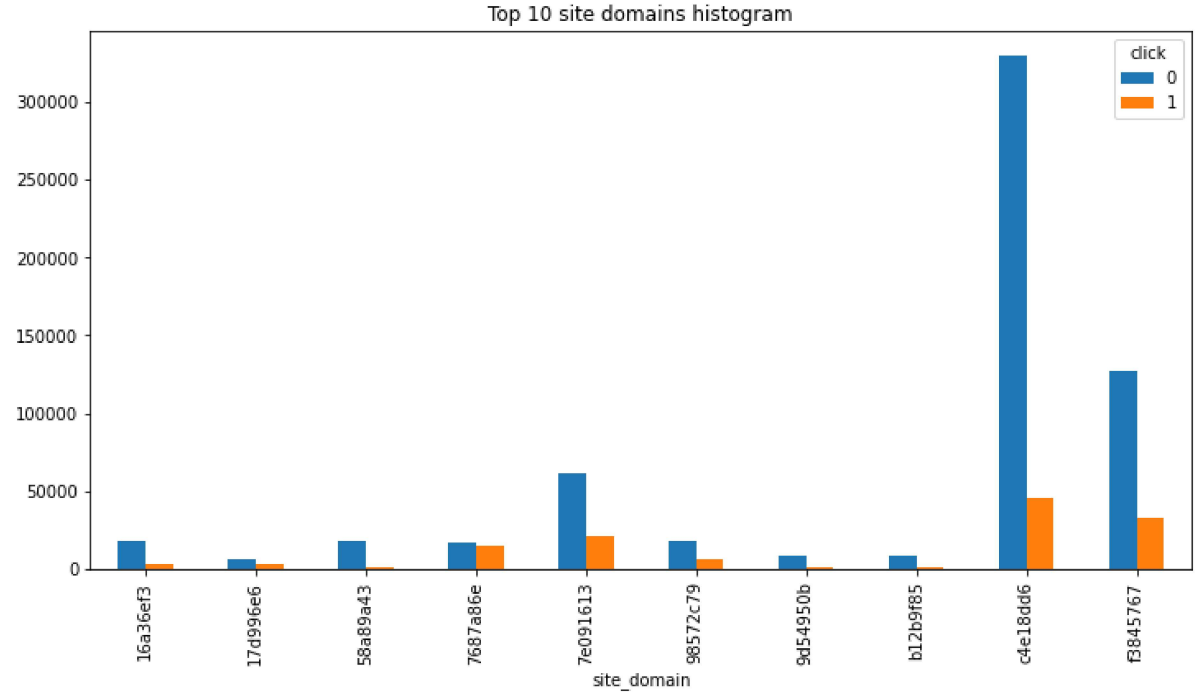
In [35]:
```
print('The top 10 site domains that have the most impressions')
print((train.site_domain.value_counts()/len(train))[0:10])
```

```
The top 10 site domains that have the most impressions
c4e18dd6    0.374576
f3845767    0.160358
7e091613    0.082492
7687a86e    0.031678
98572c79    0.024532
16a36ef3    0.020967
58a89a43    0.018886
b12b9f85    0.009350
9d54950b    0.009318
17d996e6    0.008688
Name: site_domain, dtype: float64
```

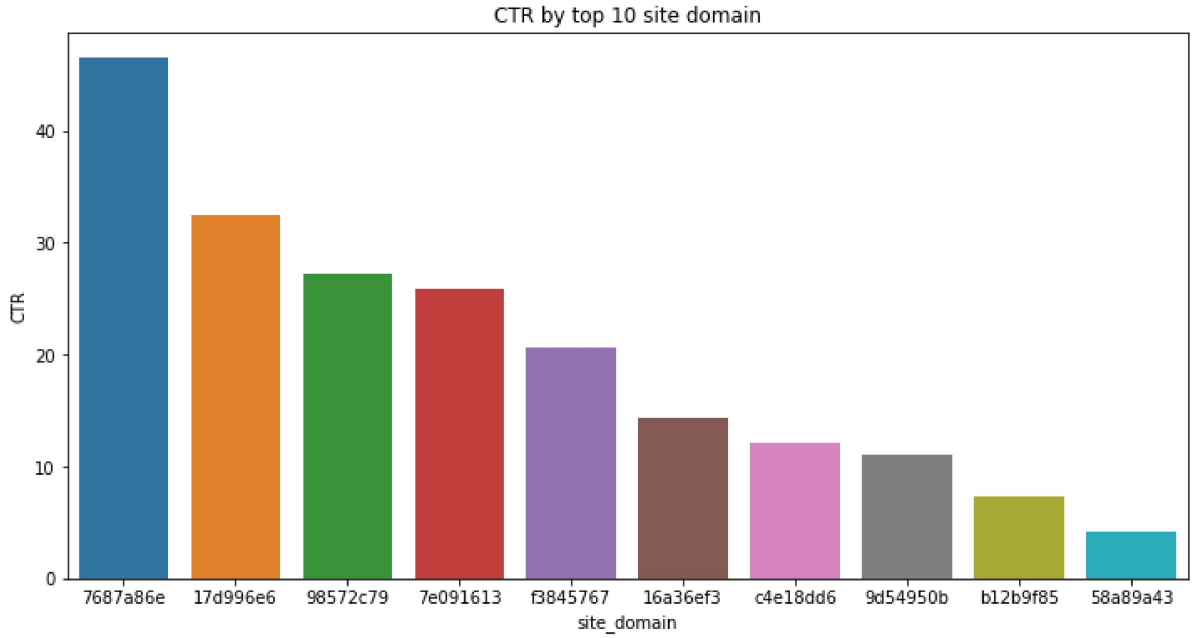
In [36]:
```
top10_domains = (train.site_domain.value_counts()/len(train))[0:10].index
click_avg_list=[]

for i in top10_domains:
    click_avg=train.loc[np.where((train.site_domain == i))].click.mean()
    click_avg_list.append(click_avg)
    print("for site domain value: {},  click through rate: {}".format(i,click_a
```

```
for site domain value: c4e18dd6,  click through rate: 0.12156945452992184
for site domain value: f3845767,  click through rate: 0.20593297496850796
for site domain value: 7e091613,  click through rate: 0.25861901760170686
for site domain value: 7687a86e,  click through rate: 0.46473893553885975
for site domain value: 98572c79,  click through rate: 0.27282732757215067
for site domain value: 16a36ef3,  click through rate: 0.14312967997329137
for site domain value: 58a89a43,  click through rate: 0.04177697765540612
for site domain value: b12b9f85,  click through rate: 0.07251336898395722
for site domain value: 9d54950b,  click through rate: 0.1102167847177506
for site domain value: 17d996e6,  click through rate: 0.32435543278084716
```

In [37]:
```
top10_domain = train[(train.site_domain.isin((train.site_domain.value_counts()/
top10_domain_click = top10_domain[top10_domain['click'] == 1]
top10_domain.groupby(['site_domain', 'click']).size().unstack().plot(kind='bar'
```



In [38]:
```
df_domain = top10_domain[['site_domain','click']].groupby(['site_domain']).cour
df_domain = df_domain.rename(columns={'click': 'impressions'})
df_domain['clicks'] = top10_domain_click[['site_domain','click']].groupby(['sit
df_domain['CTR'] = df_domain['clicks']/df_domain['impressions']*100
sort_domain = df_domain.sort_values(by='CTR',ascending=False)['site_domain'].to
plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='site_domain', data=df_domain, order=sort_domain)
plt.title('CTR by top 10 site domain');
```



Similar with the site_id feature, the site_domain feature seems important as well.

site category

In [39]:
```
print("There are {} site categories in the data set".format(train.site_category
```

```
There are 22 site categories in the data set
```

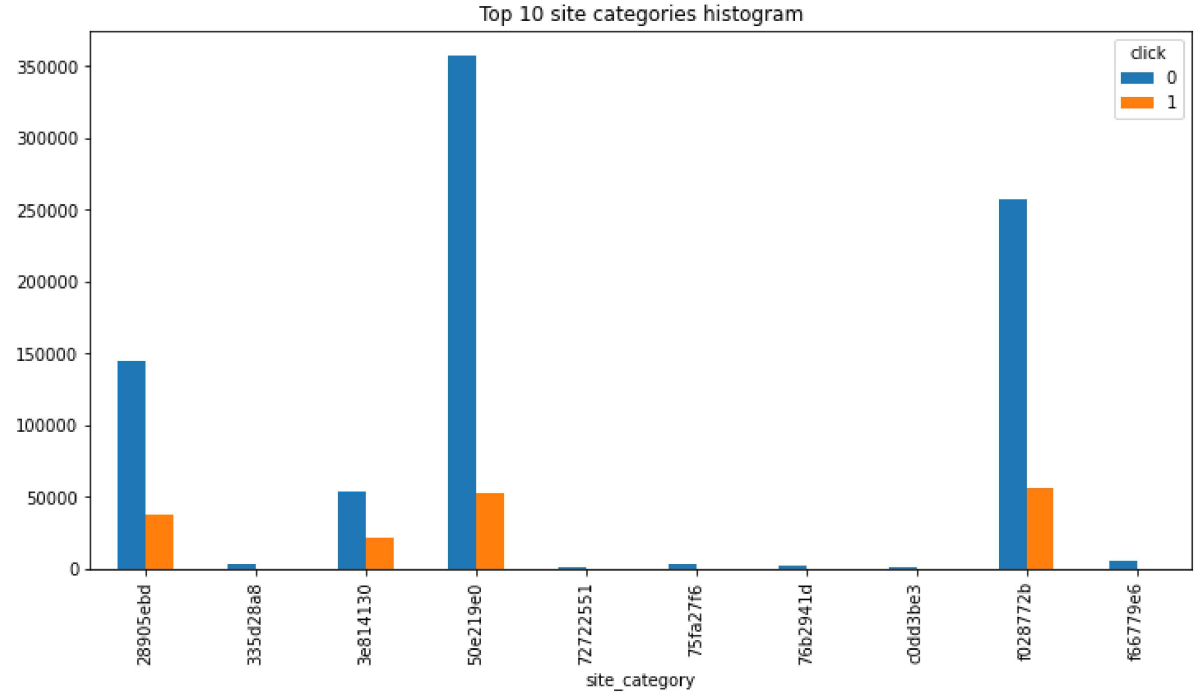
In [40]:
```
print('The top 10 site categories that have the most impressions')
print((train.site_category.value_counts()/len(train))[0:10])
```

```
The top 10 site categories that have the most impressions
50e219e0    0.409314
f028772b    0.313158
28905ebd    0.182354
3e814130    0.075353
f66779e6    0.006096
75fa27f6    0.004036
335d28a8    0.003298
76b2941d    0.002606
c0dd3be3    0.001008
72722551    0.000702
Name: site_category, dtype: float64
```

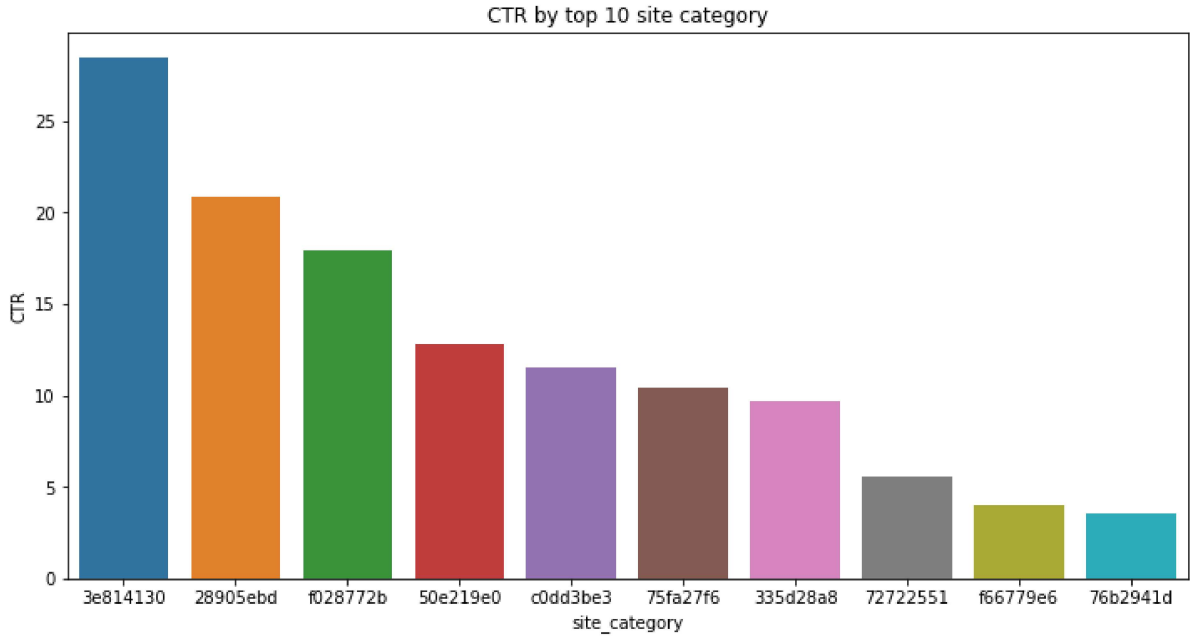
In [41]:
```
top10_categories = (train.site_category.value_counts()/len(train))[0:10].index
click_avg_list=[]

for i in top10_categories:
    click_avg=train.loc[np.where((train.site_category == i))].click.mean()
    click_avg_list.append(click_avg)
    print("for site category value: {},  click through rate: {}".format(i,click
```

```
for site category value: 50e219e0,  click through rate: 0.12764039343877806
for site category value: f028772b,  click through rate: 0.17923540193768003
for site category value: 28905ebd,  click through rate: 0.20842427366550775
for site category value: 3e814130,  click through rate: 0.2843416984061683
for site category value: f66779e6,  click through rate: 0.04019028871391076
for site category value: 75fa27f6,  click through rate: 0.10381565906838454
for site category value: 335d28a8,  click through rate: 0.09733171619163129
for site category value: 76b2941d,  click through rate: 0.03568687643898695
for site category value: c0dd3be3,  click through rate: 0.11507936507936507
for site category value: 72722551,  click through rate: 0.05555555555555555
```

In [42]:
```
top10_category = train[(train.site_category.isin((train.site_category.value_cou
top10_category_click = top10_category[top10_category['click'] == 1]
top10_category.groupby(['site_category', 'click']).size().unstack().plot(kind='
```



In [43]:
```
df_category = top10_category[['site_category','click']].groupby(['site_category
df_category = df_category.rename(columns={'click': 'impressions'})
df_category['clicks'] = top10_category_click[['site_category','click']].groupby
df_category['CTR'] = df_category['clicks']/df_category['impressions']*100
sort_category = df_category.sort_values(by='CTR',ascending=False)['site_categor
plt.figure(figsize=(12,6))
sns.barplot(y='CTR', x='site_category', data=df_category, order=sort_category)
plt.title('CTR by top 10 site category');
```



Device features

device id

In [44]:
```
print("There are {} devices in the data set".format(train.device_id.nunique()))
```

```
There are 150479 devices in the data set
```

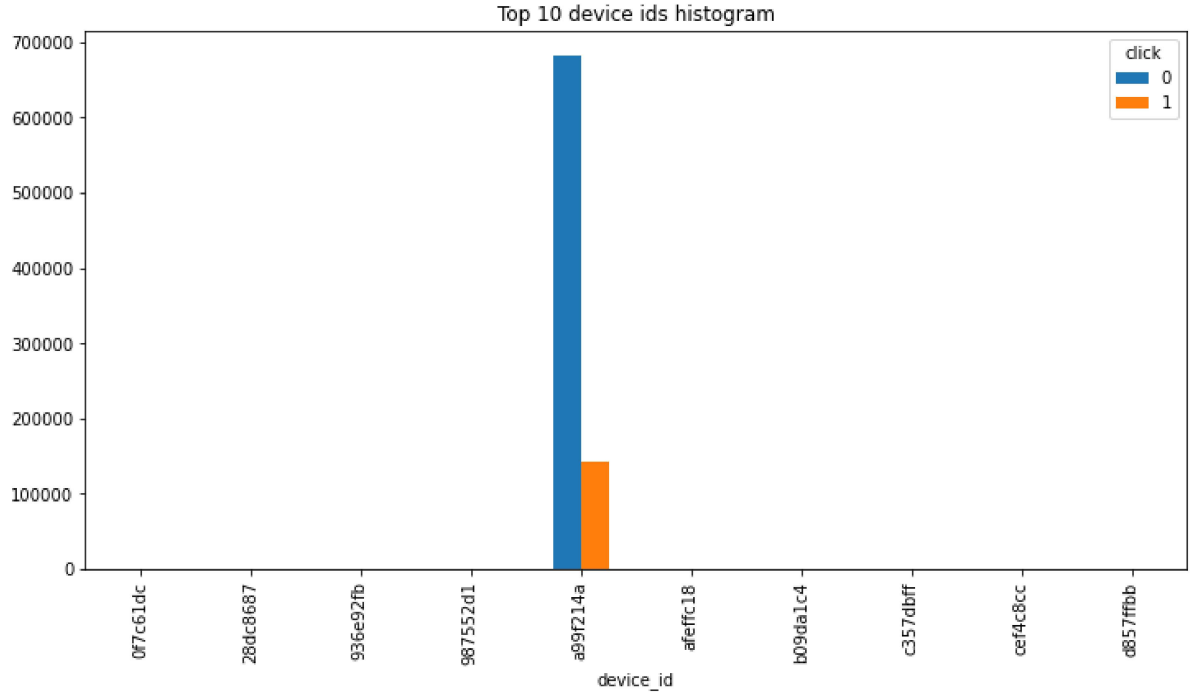
In [45]:
```
print('The top 10 devices that have the most impressions')
print((train.device_id.value_counts()/len(train))[0:10])
```

```
The top 10 devices that have the most impressions
a99f214a    0.824774
0f7c61dc    0.000552
c357dbff    0.000509
936e92fb    0.000318
afeffc18    0.000245
987552d1    0.000124
d857ffbb    0.000106
28dc8687    0.000104
b09da1c4    0.000100
cef4c8cc    0.000086
Name: device_id, dtype: float64
```

In [46]:
```
top10_devices = (train.device_id.value_counts()/len(train))[0:10].index
click_avg_list=[]

for i in top10_devices:
    click_avg=train.loc[np.where((train.device_id == i))].click.mean()
    click_avg_list.append(click_avg)
    print("for device id value: {},  click through rate: {}".format(i,click_avg
```

```
for device id value: a99f214a,  click through rate: 0.17383913653922164
for device id value: 0f7c61dc,  click through rate: 0.7373188405797102
for device id value: c357dbff,  click through rate: 0.6463654223968566
for device id value: 936e92fb,  click through rate: 0.059748427672955975
for device id value: afeffc18,  click through rate: 0.24081632653061225
for device id value: 987552d1,  click through rate: 0.0
for device id value: d857ffbb,  click through rate: 0.25471698113207547
for device id value: 28dc8687,  click through rate: 0.0
for device id value: b09da1c4,  click through rate: 0.12
for device id value: cef4c8cc,  click through rate: 0.16279069767441862
```

In [47]:
```
top10_device = train[(train.device_id.isin((train.device_id.value_counts()/len(
top10_device_click = top10_device[top10_device['click'] == 1]
top10_device.groupby(['device_id', 'click']).size().unstack().plot(kind='bar',
```



You will see that most of device_id is a99f214a : Approx. 83% of the data, and the second major device_id is only 0.05% of the data. And there are some extremely high CTR here with device id at 0f7c61dc.

device ip

Device ip is more of a users ip address, so, there are a lot of them.

In [48]:
```
print("There are {} device ips in the data set".format(train.device_ip.nunique(
print("There are {} device types in the data set".format(train.device_type.nuni
print("There are {} device models in the data set".format(train.device_model.nu
print("There are {} device cnn types in the data set".format(train.device_conn_
```

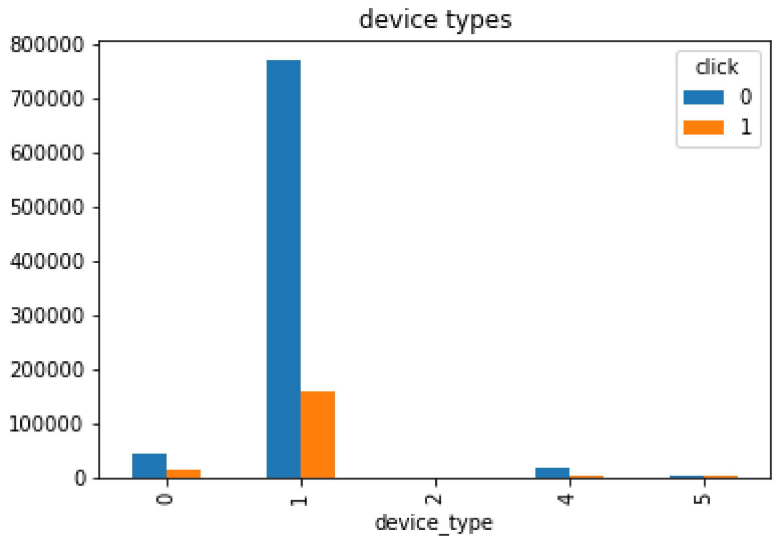
```
There are 555448 device ips in the data set
There are 5 device types in the data set
There are 5156 device models in the data set
There are 4 device cnn types in the data set
```

device type

In [49]:
```
print('The impressions by device types')
print((train.device_type.value_counts()/len(train)))
```

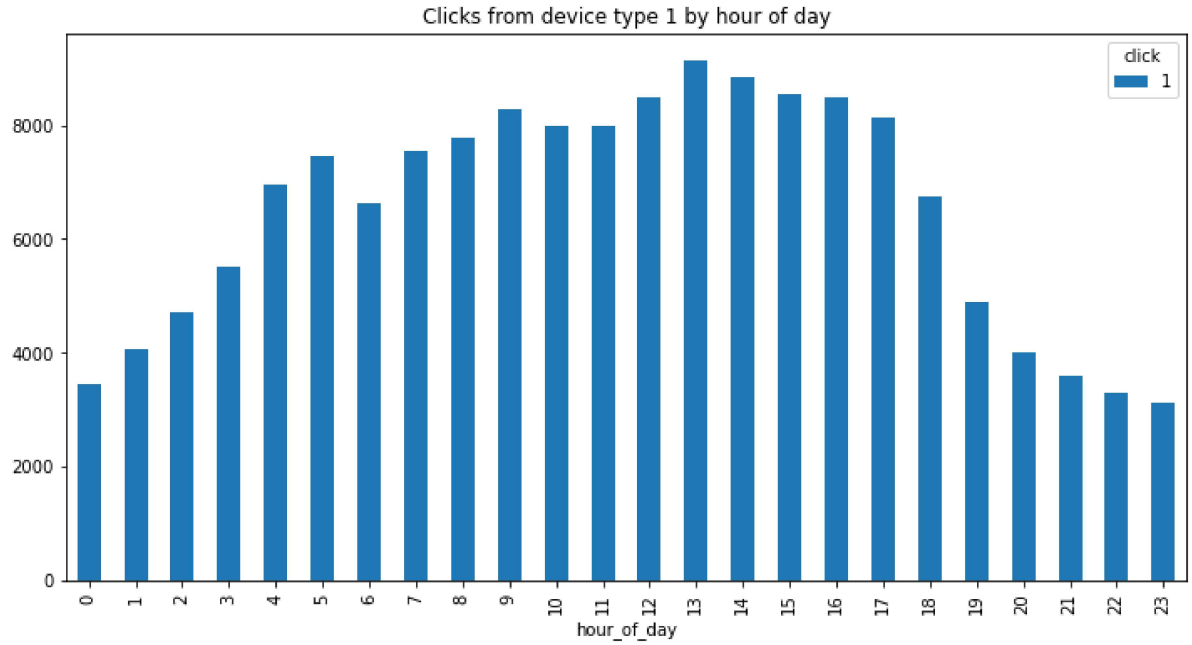
```
The impressions by device types
1    0.922638
0    0.054882
4    0.019312
5    0.003167
2    0.000001
Name: device_type, dtype: float64
```

In [50]:
```
train[['device_type','click']].groupby(['device_type','click']).size().unstack(
```



Device type 1 gets the most impressions and clicks. And the other device types only get the minimum impressions and clicks. We may want to look in more details about device type 1.

In [51]:
```
df_click[df_click['device_type']==1].groupby(['hour_of_day', 'click']).size().u
```



As expected, most clicks happened during the business hours from device type 1. device type is definitely an important feature.

In [52]:
```
device_type_click = df_click.groupby('device_type').agg({'click':'sum'}).reset_
device_type_impression = train.groupby('device_type').agg({'click':'count'}).re
merged_device_type = pd.merge(left = device_type_click , right = device_type_in
```

In [53]:
```
merged_device_type['CTR'] = merged_device_type['click'] / merged_device_type['i
```

In [54]:
```
merged_device_type
```

Out[54]:

	device_type	click	impressions	CTR
0	0	11690	54882	21.300244
1	1	155684	922638	16.873790
2	4	1826	19312	9.455261
3	5	286	3167	9.030628

The highest CTR comes from device type 0.

app features

In [55]:
```
print("There are {} apps in the data set".format(train.app_id.nunique()))
print("There are {} app domains in the data set".format(train.app_domain.nuniqu
print("There are {} app categories in the data set".format(train.app_category.r
```

```
There are 3156 apps in the data set
There are 196 app domains in the data set
There are 27 app categories in the data set
```

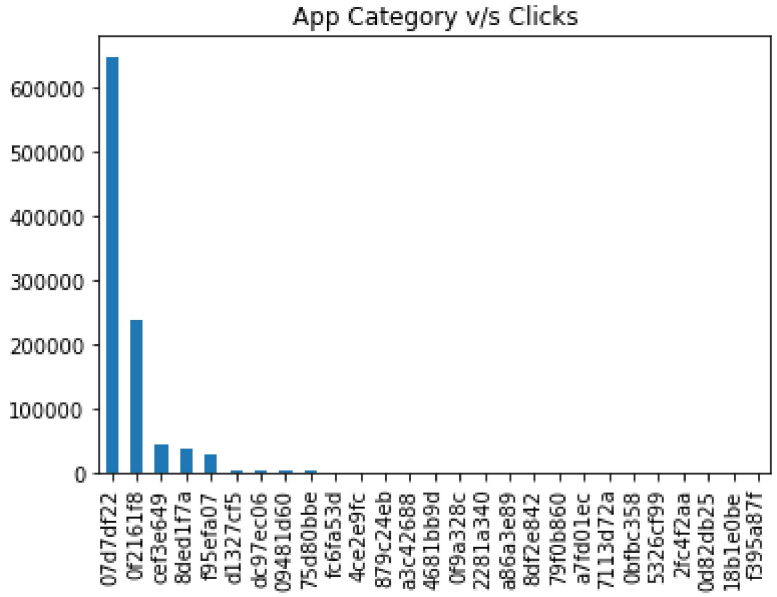
Looks like app category something worth to explore.

In [56]:
```
print('The impressions by app categories')
print((train.app_category.value_counts()/len(train)))
```

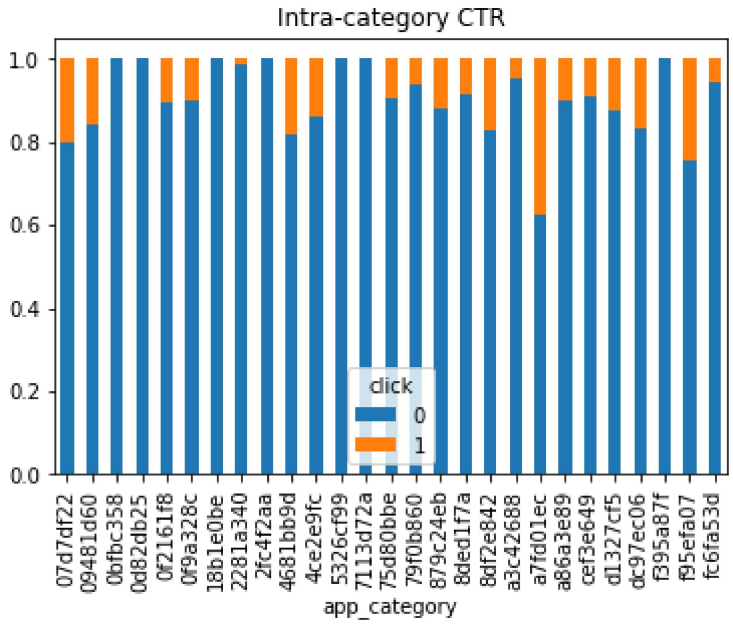
```
The impressions by app categories
07d7df22    0.646927
0f2161f8    0.236926
cef3e649    0.042625
8ded1f7a    0.036239
f95efa07    0.028330
d1327cf5    0.003026
dc97ec06    0.001393
09481d60    0.001335
75d80bbe    0.001021
fc6fa53d    0.000586
4ce2e9fc    0.000509
879c24eb    0.000307
a3c42688    0.000264
4681bb9d    0.000153
0f9a328c    0.000139
2281a340    0.000067
a86a3e89    0.000059
8df2e842    0.000041
79f0b860    0.000016
a7fd01ec    0.000008
7113d72a    0.000008
0bfbc358    0.000007
5326cf99    0.000006
2fc4f2aa    0.000004
0d82db25    0.000002
18b1e0be    0.000001
f395a87f    0.000001
Name: app_category, dtype: float64
```

In [57]:
```
train['app_category'].value_counts().plot(kind='bar', title='App Category v/s C
```

Out[57]:
```
<AxesSubplot:title={'center':'App Category v/s Clicks'}>
```



In [58]:
```
train_app_category = train.groupby(['app_category', 'click']).size().unstack()
train_app_category.div(train_app_category.sum(axis=1), axis=0).plot(kind='bar',
```

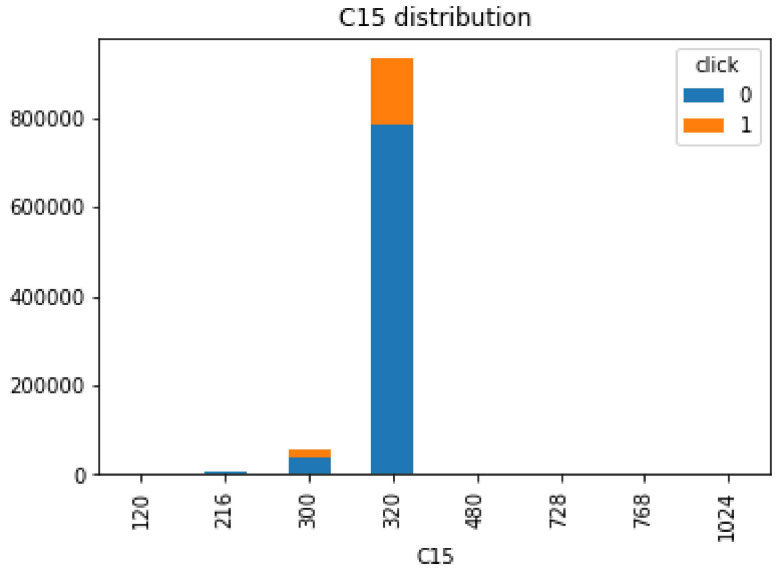


C14 - C21 features

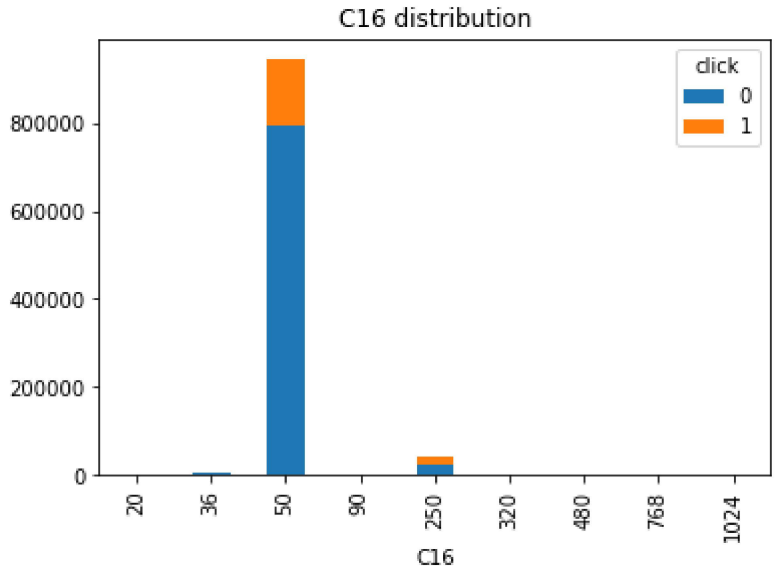
In [59]:
```
print("There are {} C14 in the data set".format(train.C14.nunique()))
print("There are {} C15 in the data set".format(train.C15.nunique()))
print("There are {} C16 in the data set".format(train.C16.nunique()))
print("There are {} C17 in the data set".format(train.C17.nunique()))
print("There are {} C18 in the data set".format(train.C18.nunique()))
print("There are {} C19 in the data set".format(train.C19.nunique()))
print("There are {} C20 in the data set".format(train.C20.nunique()))
```

```
There are 2237 C14 in the data set
There are 8 C15 in the data set
There are 9 C16 in the data set
There are 420 C17 in the data set
There are 4 C18 in the data set
There are 66 C19 in the data set
There are 165 C20 in the data set
```

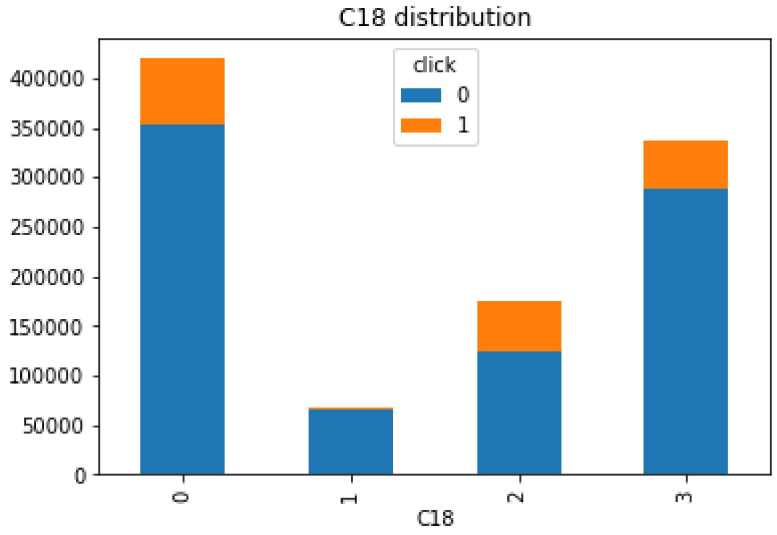
In [60]:
```
train.groupby(['C15', 'click']).size().unstack().plot(kind='bar', stacked=True,
```



In [61]:
```
train.groupby(['C16', 'click']).size().unstack().plot(kind='bar', stacked=True,
```



In [62]:
```
train.groupby(['C18', 'click']).size().unstack().plot(kind='bar', stacked=True,
```



In [63]:
```
train.head(3)
```

Out[63]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id
0	-34680422	0	2014-10-21	1005	0	d6137915	bb1ef334	f028772b	ecad2386
1	-1562553648	1	2014-10-21	1005	0	4dd0a958	79cf0c8d	f028772b	ecad2386
2	1315205890	0	2014-10-21	1002	0	85f751fd	c4e18dd6	50e219e0	a37bf1e4

3 rows × 25 columns

In [64]:
```
def convert_obj_to_int(self):

    object_list_columns = self.columns
    object_list_dtypes = self.dtypes
    new_col_suffix = '_int'
    for index in range(0,len(object_list_columns)):
        if object_list_dtypes[index] == object :
            self[object_list_columns[index]+new_col_suffix] = self[object_list_
            self.drop([object_list_columns[index]],inplace=True,axis=1)
    return self
train = convert_obj_to_int(train)
```

In [65]:
```
train.head(3)
```

Out[65]:

	id	click	hour	C1	banner_pos	device_type	device_conn_type	C14	C15	C16
0	-34680422	0	2014-10-21	1005	0	1	0	19771	320	50
1	-1562553648	1	2014-10-21	1005	0	1	0	20366	320	50
2	1315205890	0	2014-10-21	1002	0	0	0	21691	320	50

3 rows × 25 columns

In [66]:
```
train.drop('hour', axis=1, inplace=True)
```

In [67]:
```
train.drop('id', axis=1, inplace=True)
```

In [68]:
```
# pip install lightgbm
```

In [75]:
```
import lightgbm as lgb
import xgboost as xgb
```

In [70]:
```
import lightgbm as lgb
X_train = train.loc[:, train.columns != 'click']
y_target = train.click.values
#create lightgbm dataset
msk = np.random.rand(len(X_train)) < 0.8
lgb_train = lgb.Dataset(X_train[msk], y_target[msk])
lgb_eval = lgb.Dataset(X_train[~msk], y_target[~msk], reference=lgb_train)
```

In [72]:
```
# specify your configurations as a dict
params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': { 'binary_logloss'},
    'num_leaves': 31, # defauly leaves(31) amount for each tree
    'learning_rate': 0.08,
    'feature_fraction': 0.7, # will select 70% features before training each tr
    'bagging_fraction': 0.3, #feature_fraction, but this will random select par
    'bagging_freq': 5, #  perform bagging at every 5 iteration
    'verbose': 0
}

print('Start training...')
# train
gbm = lgb.train(params,
                lgb_train,
                num_boost_round=4000,
                valid_sets=lgb_eval)
```

```
Start training...
```

In [73]:
```
print(gbm.best_score)
print(gbm.best_iteration)
```

```
defaultdict(<class 'collections.OrderedDict'>, {'valid_0': OrderedDict([('bin
ary_logloss', 0.4029198012044491)])})
0
```

In [76]:
```
from operator import itemgetter
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import roc_auc_score

def run_default_test(train, test, features, target, random_state=0):
    eta = 0.1
    max_depth = 5
    subsample = 0.8
    colsample_bytree = 0.8
    print('XGBoost params. ETA: {}, MAX_DEPTH: {}, SUBSAMPLE: {}, COLSAMPLE_BY_
    params = {
        "objective": "binary:logistic",
        "booster" : "gbtree",
        "eval_metric": "logloss",
        "eta": eta,
        "max_depth": max_depth,
        "subsample": subsample,
        "colsample_bytree": colsample_bytree,
        "silent": 1,
        "seed": random_state
    }
    num_boost_round = 260
    early_stopping_rounds = 20
    test_size = 0.2

    X_train, X_valid = train_test_split(train, test_size=test_size, random_stat
    y_train = X_train[target]
    y_valid = X_valid[target]
    dtrain = xgb.DMatrix(X_train[features], y_train)
    dvalid = xgb.DMatrix(X_valid[features], y_valid)
    watchlist = [(dtrain, 'train'), (dvalid, 'eval')]
    gbm = xgb.train(params, dtrain, num_boost_round, evals=watchlist, early_sto
```

In [78]:
```
features = ['C1', 'banner_pos', 'device_type', 'device_conn_type', 'C14',
       'C15', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'hour_of_day',
       'site_id_int', 'site_domain_int', 'site_category_int', 'app_id_int',
       'app_domain_int', 'app_category_int', 'device_id_int', 'device_ip_int',
       'device_model_int', 'day_of_week_int']
run_default_test(train, y_target, features, 'click')
```

```
XGBoost params. ETA: 0.1, MAX_DEPTH: 5, SUBSAMPLE: 0.8, COLSAMPLE_BY_TREE:
0.8
```