In [ ]:
```
from google.colab import drive
drive.mount('/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client
_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&
redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2f
www.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fau
th%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%
20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=
code

Enter your authorization code:
··········
Mounted at /drive
```

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import cm
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Futu
reWarning: pandas.util.testing is deprecated. Use the functions in the public
API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [ ]:
```
df = pd.read_csv('/drive/My Drive/newtrain.csv')
df
```

Out[ ]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category
0	2.305899e+18	0	14102922	1005	0	85f751fd	c4e18dd6	50e219e0
1	1.255358e+18	1	14102102	1005	1	d9750ee7	98572c79	f028772b
2	1.390768e+19	1	14102511	1005	0	38217daf	449497bc	f028772b
3	1.560622e+19	0	14102310	1005	0	85f751fd	c4e18dd6	50e219e0
4	8.317860e+18	0	14102611	1005	1	c63170c5	a9bba545	f028772b
...	...	...	...	...	...	...	...	...
999995	5.280819e+18	0	14102807	1005	0	85f751fd	c4e18dd6	50e219e0
999996	1.670305e+19	0	14102800	1005	0	bb4524e7	d733bbc3	28905ebd
999997	9.601175e+18	0	14102217	1005	0	85f751fd	c4e18dd6	50e219e0
999998	7.943447e+18	0	14102216	1005	1	5114c672	3f2f3819	3e814130
999999	1.342240e+19	0	14102712	1005	0	85f751fd	c4e18dd6	50e219e0

1000000 rows × 24 columns

Summerize data

In [ ]:
```
df.shape
```

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

In [ ]:
```
columns = df.keys()
columns = list(columns)
print(columns)
```

```
['id', 'click', 'hour', 'C1', 'banner_pos', 'site_id', 'site_domain', 'site_c
ategory', 'app_id', 'app_domain', 'app_category', 'device_id', 'device_ip',
'device_model', 'device_type', 'device_conn_type', 'C14', 'C15', 'C16', 'C1
7', 'C18', 'C19', 'C20', 'C21']
```

In [ ]:
```
df_dup = df.drop_duplicates()
print(df.shape, df_dup.shape)

df= df_dup
df.shape
```

```
(1000000, 24) (1000000, 24)
```

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

In [ ]:
```
df = df.drop(['id'] , axis = 1)
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000000 entries, 0 to 999999
Data columns (total 23 columns):
 #   Column            Non-Null Count    Dtype 
---  ------            --------------    ----- 
 0   click             1000000 non-null  int64 
 1   hour              1000000 non-null  int64 
 2   C1                1000000 non-null  int64 
 3   banner_pos        1000000 non-null  int64 
 4   site_id           1000000 non-null  object
 5   site_domain       1000000 non-null  object
 6   site_category     1000000 non-null  object
 7   app_id            1000000 non-null  object
 8   app_domain        1000000 non-null  object
 9   app_category      1000000 non-null  object
 10  device_id         1000000 non-null  object
 11  device_ip         1000000 non-null  object
 12  device_model      1000000 non-null  object
 13  device_type       1000000 non-null  int64 
 14  device_conn_type  1000000 non-null  int64 
 15  C14               1000000 non-null  int64 
 16  C15               1000000 non-null  int64 
 17  C16               1000000 non-null  int64 
 18  C17               1000000 non-null  int64 
 19  C18               1000000 non-null  int64 
 20  C19               1000000 non-null  int64 
 21  C20               1000000 non-null  int64 
 22  C21               1000000 non-null  int64 
dtypes: int64(14), object(9)
memory usage: 183.1+ MB
```

In [ ]:
```
df.nunique()
```

Out[ ]:
```
click                 2
hour                240
C1                    7
banner_pos            7
site_id            2632
site_domain        2855
site_category        22
app_id             3160
app_domain          194
app_category         26
device_id         150270
device_ip         554816
device_model        5175
device_type            5
device_conn_type       4
C14                2257
C15                    8
C16                    9
C17                  422
C18                    4
C19                   66
C20                  164
C21                   60
dtype: int64
```

In [ ]:
```
#missing values
df.isnull().head()
```

Out[ ]:

	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	ap
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	

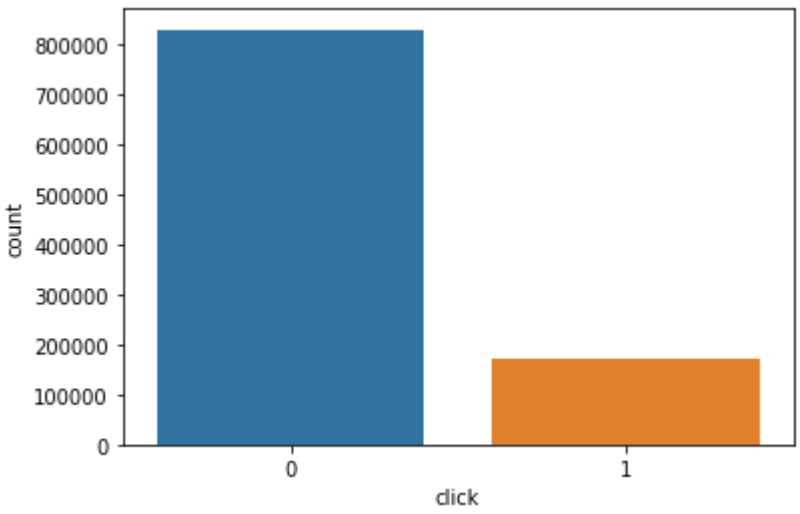
In [ ]:
```
#برای تعداد میسینگ ولیو ها
df.isnull().sum()
```

Out[ ]:
```
click              0
hour               0
C1                 0
banner_pos         0
site_id            0
site_domain        0
site_category      0
app_id             0
app_domain         0
app_category       0
device_id          0
device_ip          0
device_model       0
device_type        0
device_conn_type   0
C14                0
C15                0
C16                0
C17                0
C18                0
C19                0
C20                0
C21                0
dtype: int64
```

In [ ]:
```
print(df.click.value_counts())
sns.countplot(df.click)
```

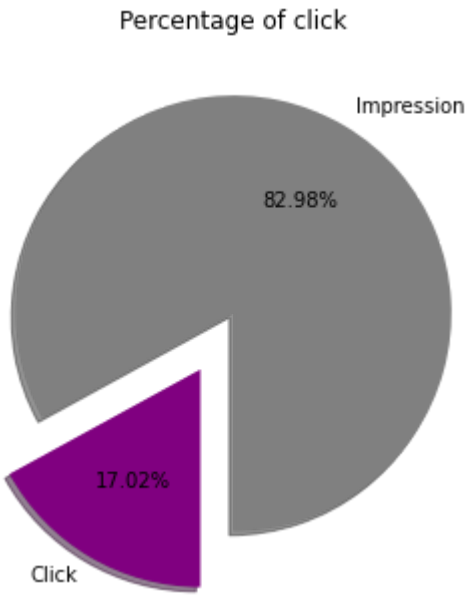
```
0    829791
1    170209
Name: click, dtype: int64
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e664100f0>
```



In [ ]:
```
from pylab import rcParams

# Data to plot
sizes = df['click'].value_counts(sort = True)
colors = ["grey","purple"]
rcParams['figure.figsize'] = 5,5 # Plot

plt.pie(sizes, explode = (0,0.3), colors=colors ,labels=["Impression", "Click"
] ,
        autopct='%.2f%%', shadow=True, startangle=270)

plt.title('Percentage of click')
```

Out[ ]:
```
Text(0.5, 1.0, 'Percentage of click')
```



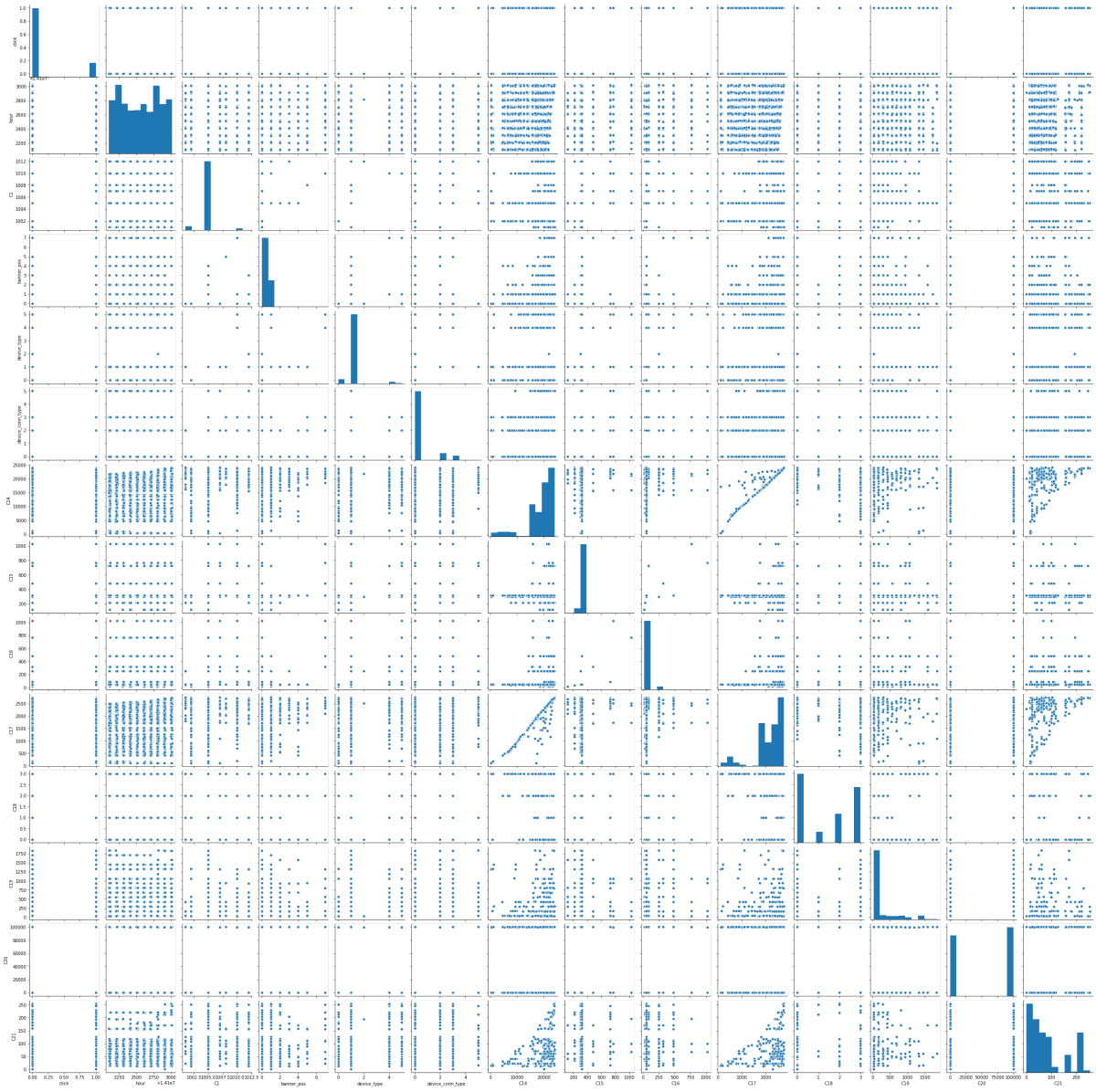
In [ ]:
```
df.describe()
```

Out[ ]:

	click	hour	C1	banner_pos	device_type	device_c
count	1000000.000000	1.000000e+06	1000000.000000	1000000.000000	1000000.000000	10000(
mean	0.170209	1.410256e+07	1004.967842	0.288050	1.015376	
std	0.375816	2.966050e+02	1.093296	0.508749	0.527228	
min	0.000000	1.410210e+07	1001.000000	0.000000	0.000000	
25%	0.000000	1.410230e+07	1005.000000	0.000000	1.000000	
50%	0.000000	1.410260e+07	1005.000000	0.000000	1.000000	
75%	0.000000	1.410281e+07	1005.000000	1.000000	1.000000	
max	1.000000	1.410302e+07	1012.000000	7.000000	5.000000	

In [ ]:
```
sns.pairplot(data=df , palette="husl")
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f1e63188ef0>
```



Hour Feature

In [ ]:
```
df['hour']=pd.to_datetime(df['hour'],format='%y%m%d%H')

print(df.hour.value_counts()/len(df))
```

```
2014-10-22 09:00:00    0.011134
2014-10-22 10:00:00    0.010964
2014-10-28 13:00:00    0.010701
2014-10-22 12:00:00    0.010225
2014-10-28 14:00:00    0.009454
                           ...
2014-10-24 19:00:00    0.000802
2014-10-24 23:00:00    0.000644
2014-10-24 20:00:00    0.000543
2014-10-24 21:00:00    0.000529
2014-10-24 22:00:00    0.000358
Name: hour, Length: 240, dtype: float64
```

In [ ]:
```
df['hour_of_day'] = df.hour.apply(lambda x: x.hour)
df.groupby('hour_of_day').agg({'click': 'count'} ).sort_values(by ='click', as
cending =[False]).style.background_gradient(cmap ='Greens')
```

Out[ ]:

hour_of_day	click
13	59050
9	56475
12	54965
14	54293
10	53271
15	51761
8	51362
16	50805
11	50654
17	50082
5	49223
4	47023
7	45696
6	43505
18	43479
3	34599
19	32593
2	30418
20	27992
21	24859
1	24474
22	22266
0	21088
23	20067

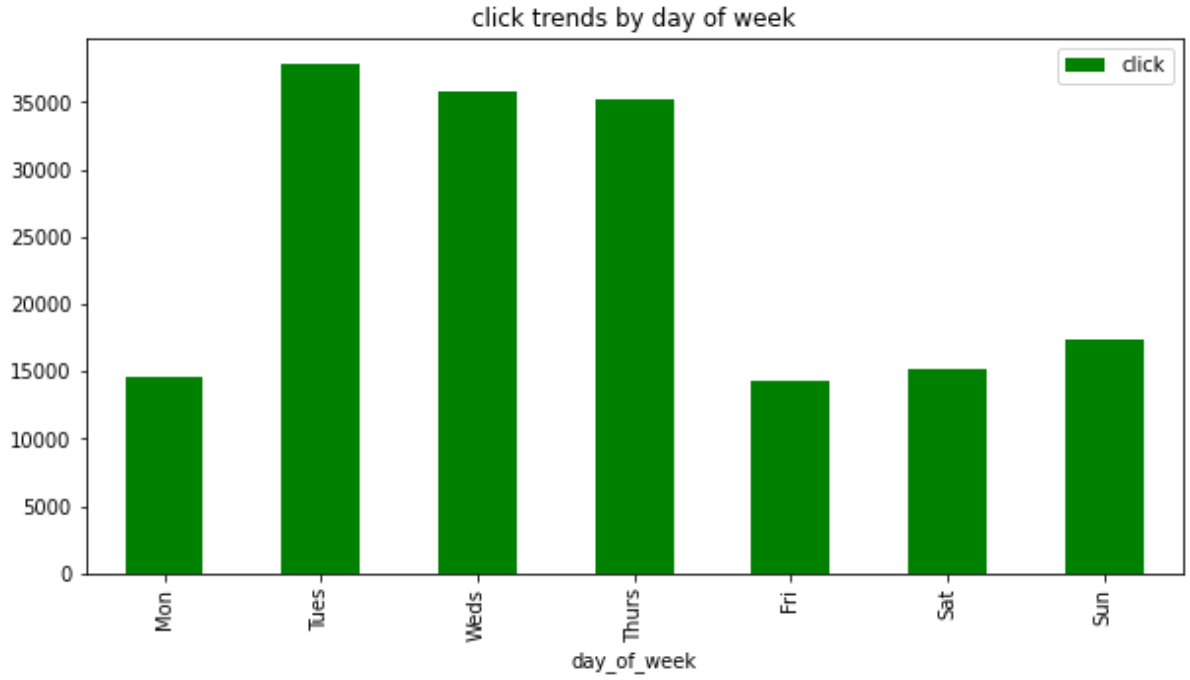
In [ ]:
```
df['day_of_month']=df.hour.dt.day
z=df.groupby(['day_of_month'])['click'].count().to_frame().reset_index().sort_
values(by = 'click', ascending =[False])
z.style.background_gradient(cmap ='Greens')
```

Out[ ]:

	day_of_month	click
1	22	132052
7	28	130755
9	30	103905
0	21	102089
2	23	96017
5	26	94937
8	29	94893
4	25	83186
3	24	82323
6	27	79843

In [ ]:
```
df['day_of_week']=df.hour.dt.dayofweek

week_days_name = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Sat
urday', 'Sunday']
df.groupby('day_of_week').agg({'click':'sum'}).reindex().plot( kind='bar',colo
r='green',figsize=(10,5))
ticks = list(range(0, 7, 1))
labels = "Mon Tues Weds Thurs Fri Sat Sun".split()
plt.xticks( ticks ,labels)
plt.title('click trends by day of week')
```

Out[ ]:
```
Text(0.5, 1.0, 'click trends by day of week')
```



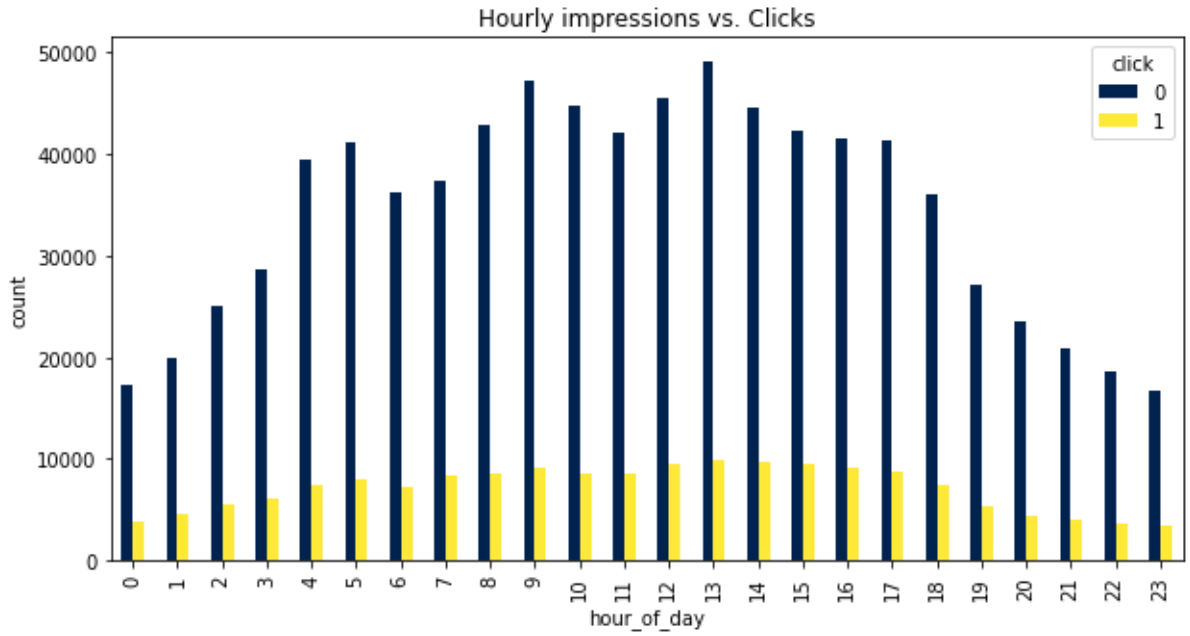
In [ ]:
```
z=df.groupby(['click','day_of_month'])['hour_of_day'].count().to_frame().reset_index()
z.style.background_gradient(cmap='Greens')
```

Out[ ]:

	click	day_of_month	hour_of_day
0	0	21	84183
1	0	22	111300
2	0	23	78492
3	0	24	68096
4	0	25	68004
5	0	26	77626
6	0	27	65210
7	0	28	110824
8	0	29	79846
9	0	30	86210
10	1	21	17906
11	1	22	20752
12	1	23	17525
13	1	24	14227
14	1	25	15182
15	1	26	17311
16	1	27	14633
17	1	28	19931
18	1	29	15047
19	1	30	17695

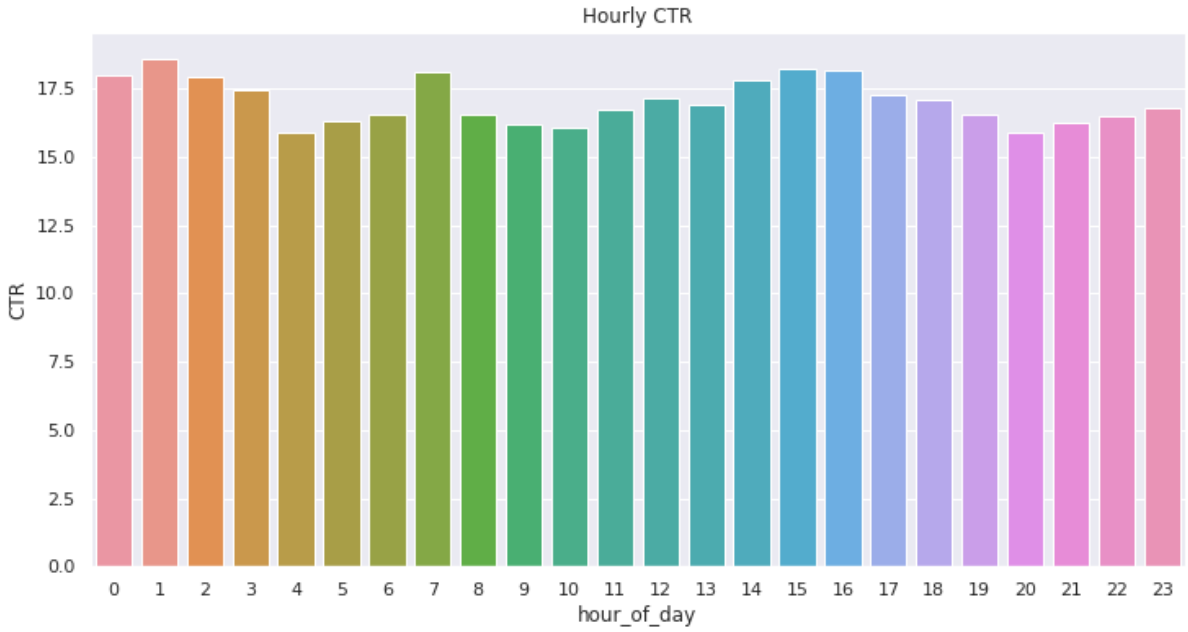
In [ ]:
```
df.groupby(['hour_of_day', 'click']).size().unstack().plot(kind='bar',cmap ='c
ividis', title="Hour of Day", figsize=(10,5))
plt.ylabel('count')
plt.title('Hourly impressions vs. Clicks')
```

Out[ ]:
```
Text(0.5, 1.0, 'Hourly impressions vs. Clicks')
```



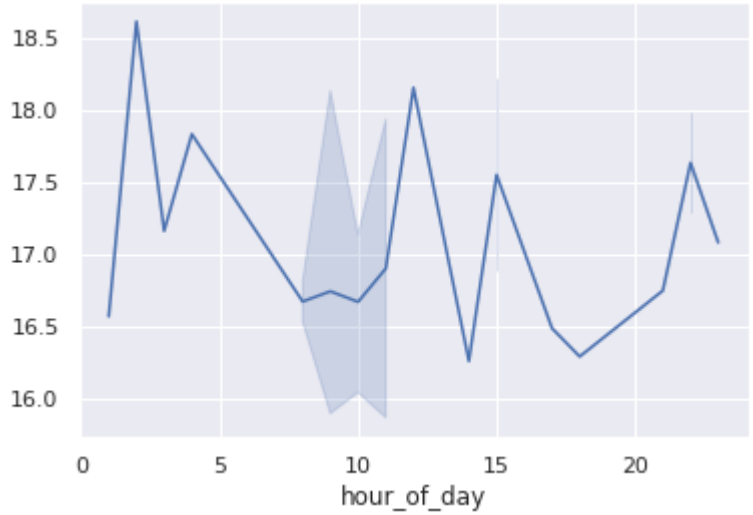
In [ ]:
```
df_click = df[df['click'] == 1]
df_hour = df[['hour_of_day','click']].groupby(['hour_of_day']).count().reset_i
ndex()
df_hour = df_hour.rename(columns={'click': 'impressions'})
df_hour['clicks'] = df_click[['hour_of_day','click']].groupby(['hour_of_day'])
.count().reset_index()['click']
df_hour['CTR'] = 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')
```

Out[ ]:
```
Text(0.5, 1.0, 'Hourly CTR')
```



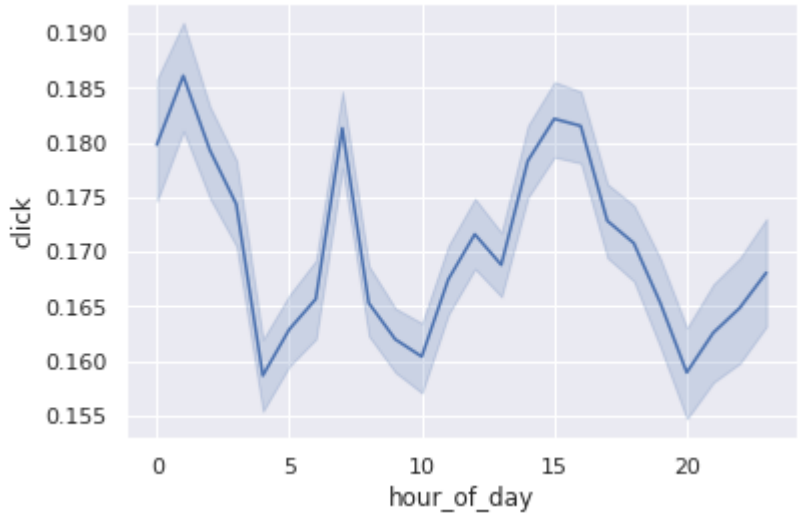
In [ ]:
```
sns.set(style="darkgrid")
sns.lineplot(x = 'hour_of_day', y =df_hour['clicks']/df_hour['impressions']*10
0  , data = df)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefe2c26978>
```



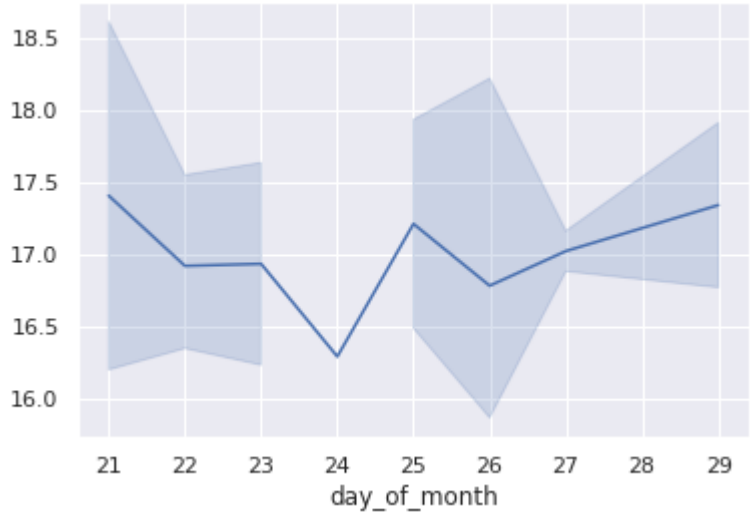
In [ ]:
```
sns.set(style="darkgrid")
sns.lineplot(x = 'hour_of_day', y ='click'  , data = df)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fefe275c5c0>



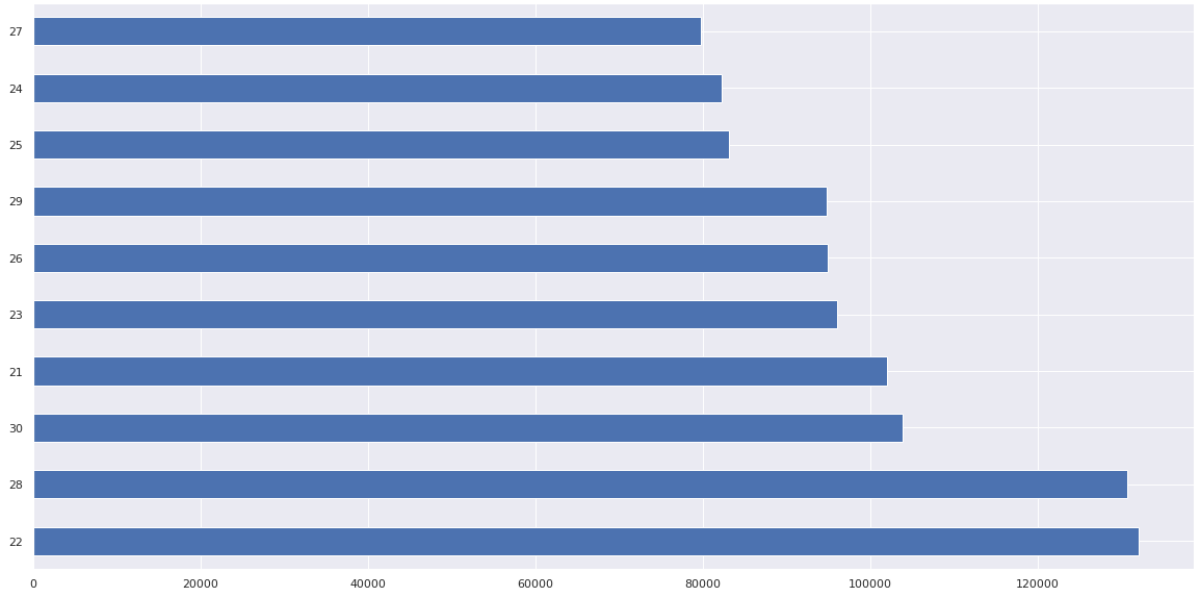
In [ ]:
```
sns.set(style="darkgrid")
sns.lineplot(x = 'day_of_month', y =df_hour['clicks']/df_hour['impressions']*1
00  , data = df)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fefe25c1978>



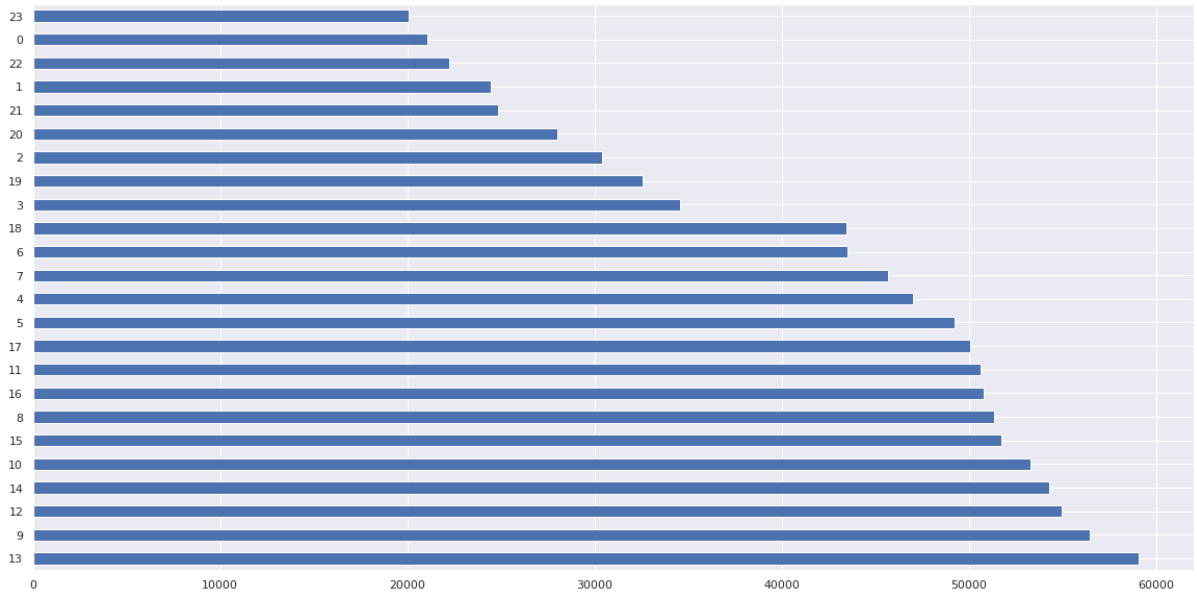
In [ ]:
```
df['day_of_month'].value_counts().head(100).plot(kind='barh', figsize=(20,10))
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefe2d274e0>
```



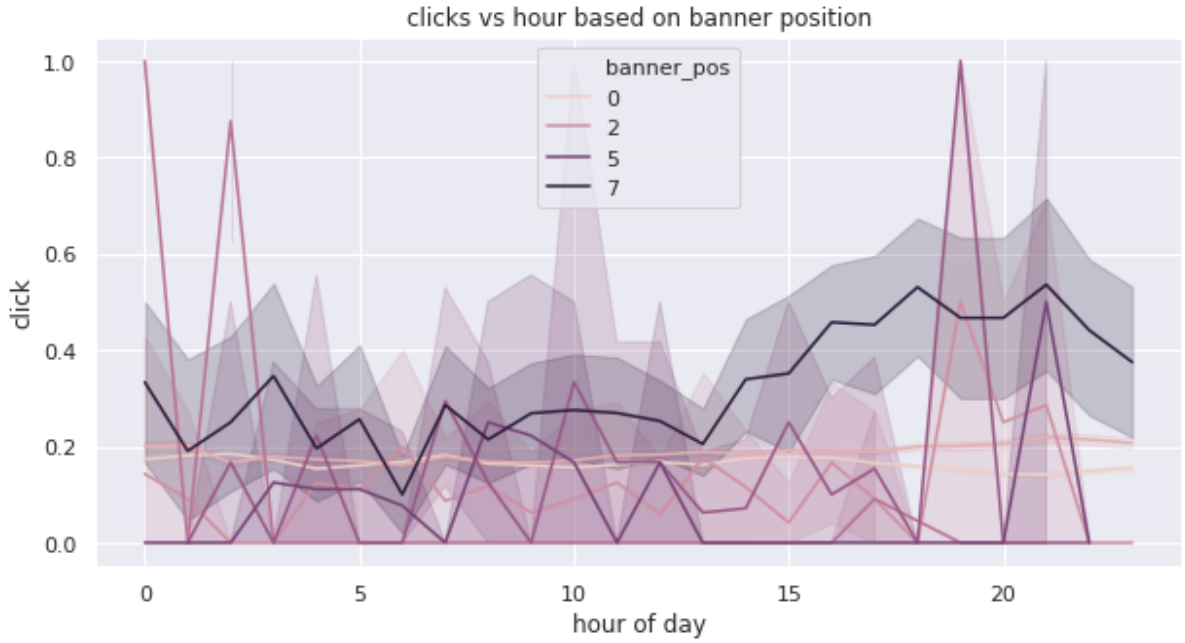
In [ ]:
```
df['hour_of_day'].value_counts().head(100).plot(kind='barh', figsize=(20,10))
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefe24a5710>
```



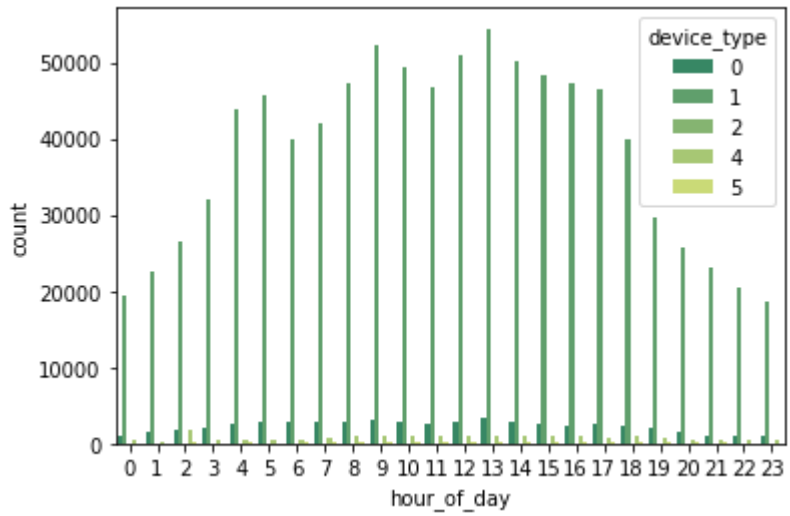
In [ ]:
```
plt.figure(figsize=(10,5))
sns.lineplot(x='hour_of_day' , y = 'click',  hue= 'banner_pos' , markers=True
, dashes=False, data = df)

plt.ylabel('click')
plt.xlabel('hour of day') ;
plt.title('clicks vs hour based on banner position')
```

Out[ ]:
```
Text(0.5, 1.0, 'clicks vs hour based on banner position')
```



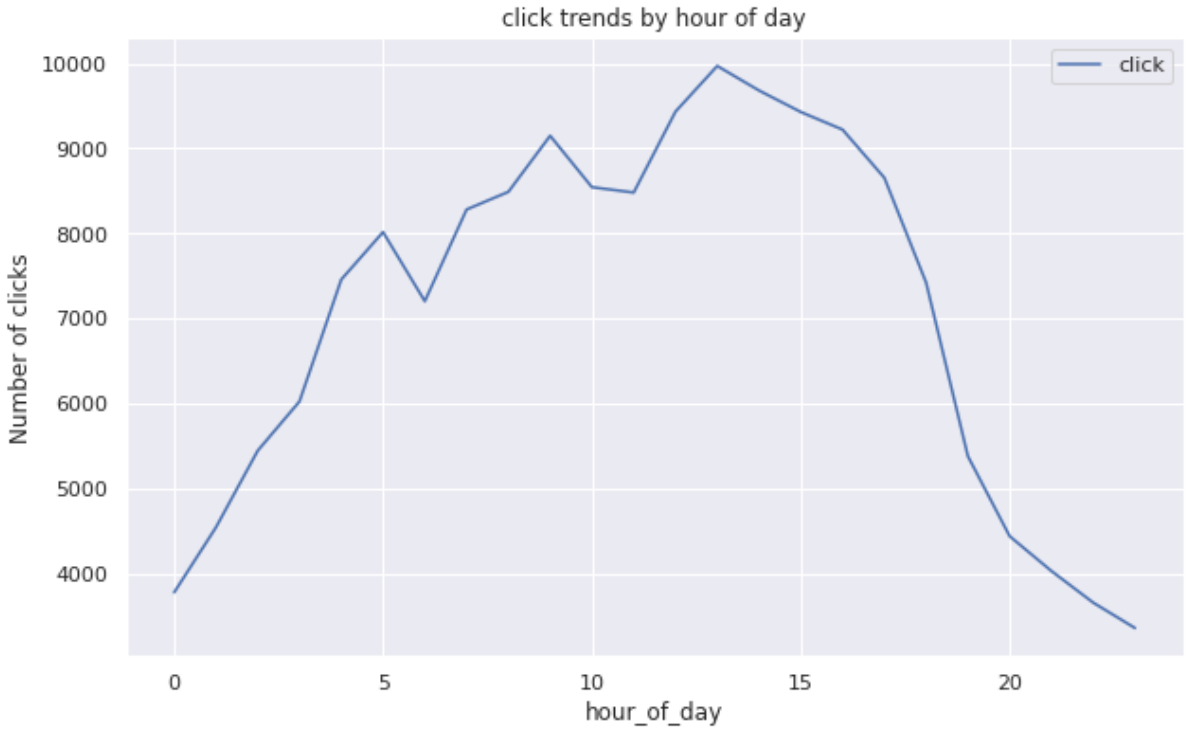
In [ ]:
```
sns.countplot(df.hour_of_day, hue=df.device_type, palette="summer" )
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabc62c1710>
```



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

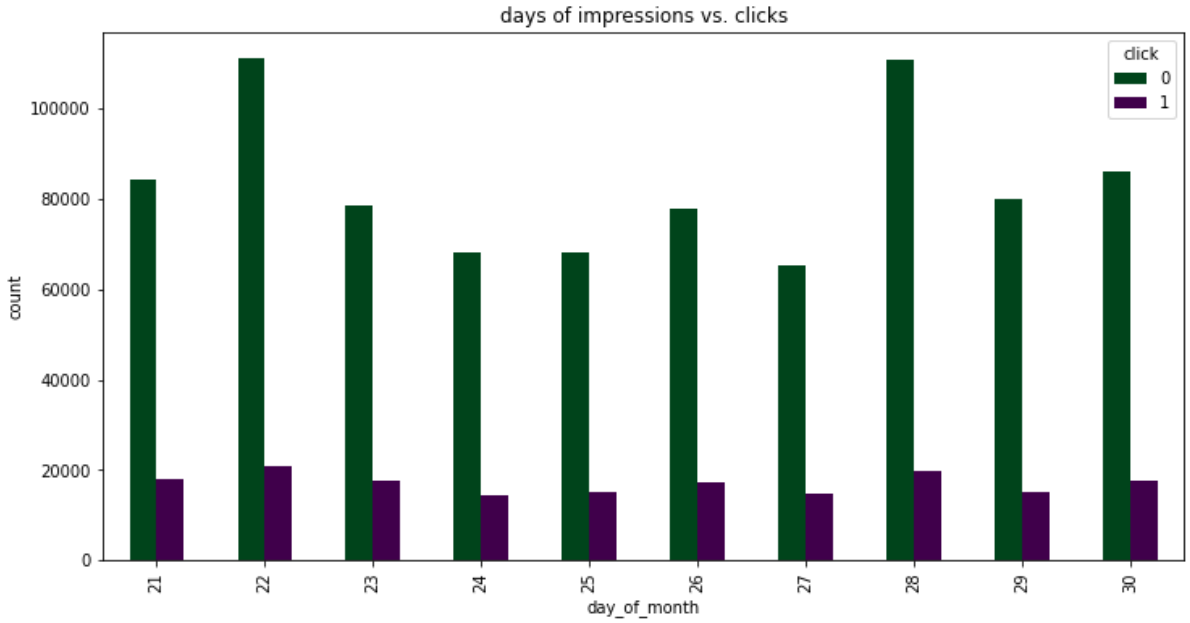
Out[ ]:
```
Text(0.5, 1.0, 'click trends by hour of day')
```



In [ ]:
```
print (df.groupby(['day_of_month', 'click']).size().unstack())
df.groupby(['day_of_month', 'click']).size().unstack().plot(kind='bar', cmap=
'PRGn_r', title="Hour of Day", figsize=(12,6))
plt.ylabel('count')
plt.title('days of impressions vs. clicks')
```

```
click              0      1
day_of_month
21             84183  17906
22            111300  20752
23             78492  17525
24             68096  14227
25             68004  15182
26             77626  17311
27             65210  14633
28            110824  19931
29             79846  15047
30             86210  17695
```

Out[ ]: Text(0.5, 1.0, 'days of impressions vs. clicks')



In [ ]:
```
z=df.groupby(['click' , 'day_of_month'])['banner_pos'].count().to_frame().reset_index()
z.style.background_gradient(cmap='Reds')
```

Out[ ]:

	click	day_of_month	banner_pos
0	0	21	84183
1	0	22	111300
2	0	23	78492
3	0	24	68096
4	0	25	68004
5	0	26	77626
6	0	27	65210
7	0	28	110824
8	0	29	79846
9	0	30	86210
10	1	21	17906
11	1	22	20752
12	1	23	17525
13	1	24	14227
14	1	25	15182
15	1	26	17311
16	1	27	14633
17	1	28	19931
18	1	29	15047
19	1	30	17695

drop some features

In [ ]:
```
df = df.drop(['hour' , 'day_of_week', 'id'] ,axis = 1)
df.head()
```

Out[ ]:

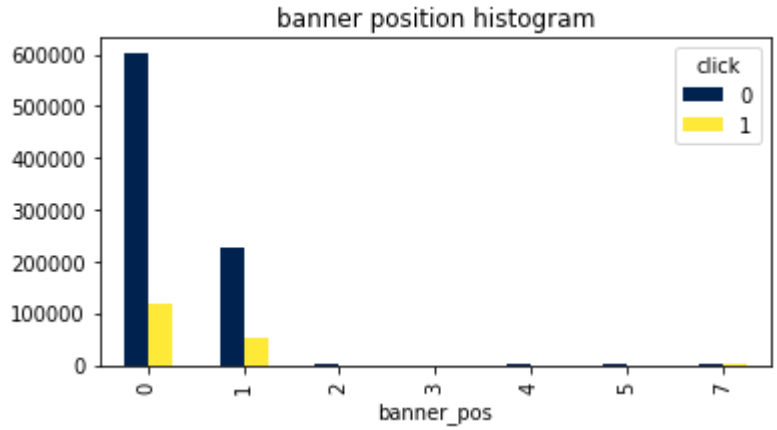
	click	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_c
0	0	1005	0	85f751fd	c4e18dd6	50e219e0	febd1138	82e27996	
1	1	1005	1	d9750ee7	98572c79	f028772b	ecad2386	7801e8d9	0
2	1	1005	0	38217daf	449497bc	f028772b	ecad2386	7801e8d9	0
3	0	1005	0	85f751fd	c4e18dd6	50e219e0	685d1c4c	2347f47a	8
4	0	1005	1	c63170c5	a9bba545	f028772b	ecad2386	7801e8d9	0

Banner Position Feature

In [ ]:
```
print (df.groupby(['banner_pos','click']).size().unstack())
df.groupby(['banner_pos','click']).size().unstack().plot(kind='bar', figsize=(
6,3), title='banner position histogram' ,cmap = 'cividis')
```

```
click            0       1
banner_pos
0           601575  118729
1           226806   51044
2              310      37
3               43       9
4              156      23
5              108      13
7              793     354
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f785eabe208>



In [ ]:
```
q = df.groupby('banner_pos').agg({ 'hour_of_day':np.size , 'click' :np.mean})
q.style.background_gradient(cmap='Reds')
```

Out[ ]:

banner_pos	hour_of_day	click
0	720304	0.164832
1	277850	0.183711
2	347	0.106628
3	52	0.173077
4	179	0.128492
5	121	0.107438
7	1147	0.308631

In [ ]:
```
z=df.groupby(['banner_pos', 'device_type'])['click'].count().to_frame().reset_
index()
z.style.background_gradient(cmap='Reds')
```

Out[ ]:

	banner_pos	device_type	click
0	0	0	54816
1	0	1	665477
2	0	2	1
3	0	5	10
4	1	1	256678
5	1	4	18154
6	1	5	3018
7	2	1	347
8	3	1	52
9	4	1	179
10	5	1	121
11	7	4	971
12	7	5	176

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

```
0    0.720304
1    0.277850
7    0.001147
2    0.000347
4    0.000179
5    0.000121
3    0.000052
Name: banner_pos, dtype: float64
```

Device Features

In [ ]:
```
z=df.groupby(['day_of_month','device_type'])['click'].count().to_frame().reset
_index()
z.style.background_gradient(cmap='Reds')
```

Out[ ]:

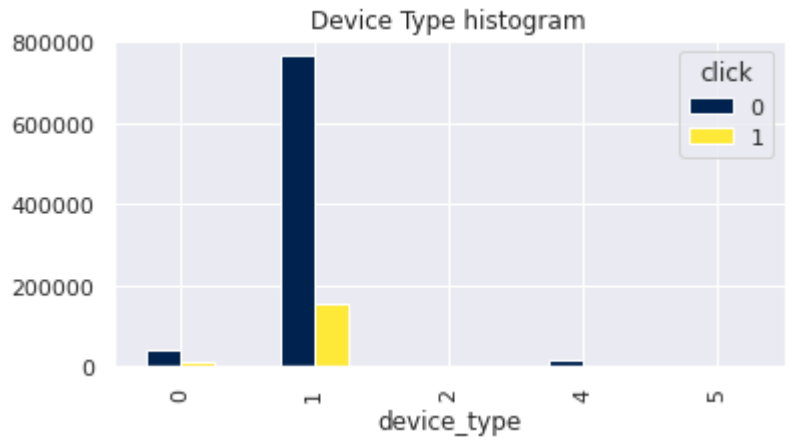
	day_of_month	device_type	click
0	21	0	4778
1	21	1	94921
2	21	4	1976
3	21	5	318
4	22	0	5752
5	22	1	123936
6	22	4	2006
7	22	5	283
8	23	0	5144
9	23	1	88876
10	23	4	1680
11	23	5	238
12	24	0	3710
13	24	1	76491
14	24	4	1657
15	24	5	362
16	25	0	4501
17	25	1	76205
18	25	4	1867
19	25	5	433
20	26	0	5463
21	26	1	87491
22	26	4	1511
23	26	5	267
24	27	0	8809
25	27	1	69219
26	27	4	1380
27	27	5	221
28	28	0	6526
29	28	1	120837
30	28	2	1
31	28	4	2764
32	28	5	392
33	29	0	5533
34	29	1	86397

	day_of_month	device_type	click
35	29	4	2412
36	29	5	336
37	30	0	4600
38	30	1	98129
39	30	4	901
40	30	5	178

In [ ]:
```
print (df.groupby(['device_type', 'click']).size().unstack())
df.groupby(['device_type', 'click']).size().unstack().plot(kind='bar', figsize
=(6,3), title='Device Type histogram' ,cmap = 'cividis')
```

```
click              0         1
device_type
0            43149.0   11667.0
1           766148.0  156354.0
2                1.0       NaN
4            16590.0    1564.0
5             2803.0     225.0
```

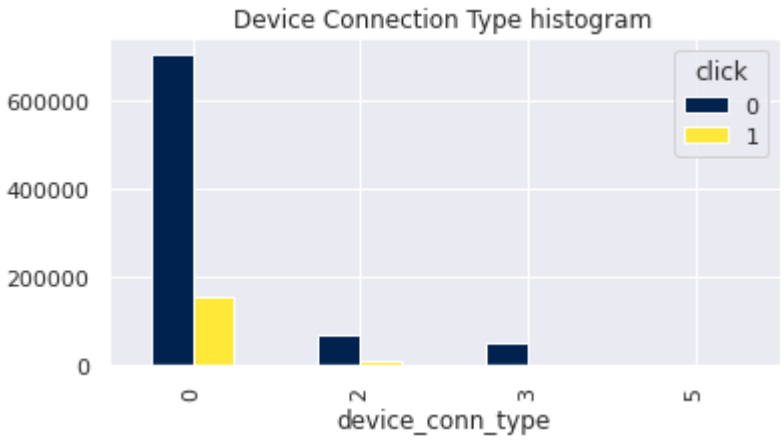
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1970f5cb00>



In [ ]:
```
print (df.groupby(['device_conn_type', 'click']).size().unstack())
df.groupby(['device_conn_type', 'click']).size().unstack().plot(kind='bar', fi
gsize=(6,3), title='Device Connection Type histogram' ,cmap = 'cividis')
```

```
click                 0       1
device_conn_type
0                705442  156777
2                 70779   10629
3                 51439    2377
5                  1031      27
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1970f574a8>



In [ ]:
```
print('device_type : ','\n',df.device_type.value_counts()/len(df))
print ('')
print('device_conn_type : ','\n',df.device_conn_type.value_counts()/len(df))
print('')
print('device model : ' , '\n' , df.device_model.value_counts()/len(df))
print('')
print('device id : ' , '\n' , df.device_id.value_counts()/len(df))
```

```
device_type :
 1    0.923887
0    0.054898
4    0.018181
5    0.003033
2    0.000001
Name: device_type, dtype: float64

device_conn_type :
 0    0.863513
2    0.081530
3    0.053897
5    0.001060
Name: device_conn_type, dtype: float64

device model :
 8a4875bd    0.060874
1f0bc64f    0.035352
d787e91b    0.034582
76dc4769    0.018870
be6db1d7    0.018511
              ...
e4492f1e    0.000001
11814293    0.000001
789c278b    0.000001
f89edea8    0.000001
4813b338    0.000001
Name: device_model, Length: 5143, dtype: float64

device id :
 a99f214a    0.826375
0f7c61dc    0.000532
c357dbff    0.000436
936e92fb    0.000336
afeffc18    0.000269
              ...
6c8a7a15    0.000001
d8cd8312    0.000001
808a632b    0.000001
90d43be1    0.000001
90402c85    0.000001
Name: device_id, Length: 149047, dtype: float64
```

C Featrues

(C1,C14,...C21)

In [ ]:
```
print('C1 : ','\n',df.C1.value_counts()/len(df))
print('')
print('C14 : ','\n',df.C14.value_counts()/len(df))
print('')
print('C15 : ','\n' , df.C15.value_counts()/len(df))
print('')
print('C16 : ','\n',df.C16.value_counts()/len(df))
print('')
print('C17 : ','\n',df.C17.value_counts()/len(df))
print('')
print('C18 : ','\n' , df.C18.value_counts()/len(df))
print('')
print('C19 : ','\n',df.C19.value_counts()/len(df))
print('')
print('C20 : ','\n',df.C20.value_counts()/len(df))
print('')
print('C21 : ','\n' , df.C21.value_counts()/len(df))
```

```
C1 :
 1005    0.939964
1002    0.056088
1012    0.002801
1007    0.000902
1001    0.000245
Name: C1, dtype: float64

C14 :
 4687     0.023863
21611    0.022763
21191    0.019358
21189    0.019336
19771    0.018302
           ...
23188    0.000001
18601    0.000001
23409    0.000001
18465    0.000001
24052    0.000001
Name: C14, Length: 2147, dtype: float64

C15 :
 320     0.932147
300     0.059351
216     0.007509
728     0.000867
120     0.000080
480     0.000030
768     0.000008
1024    0.000008
Name: C15, dtype: float64

C16 :
 50      0.943804
250     0.046007
36      0.007509
480     0.001686
90      0.000867
20      0.000080
320     0.000030
768     0.000008
1024    0.000008
Name: C16, dtype: float64

C17 :
 1722    0.109042
2424    0.038694
2227    0.036773
1800    0.029938
423     0.023863
          ...
2568    0.000001
2511    0.000001
644     0.000001
2509    0.000001
2737    0.000001
```

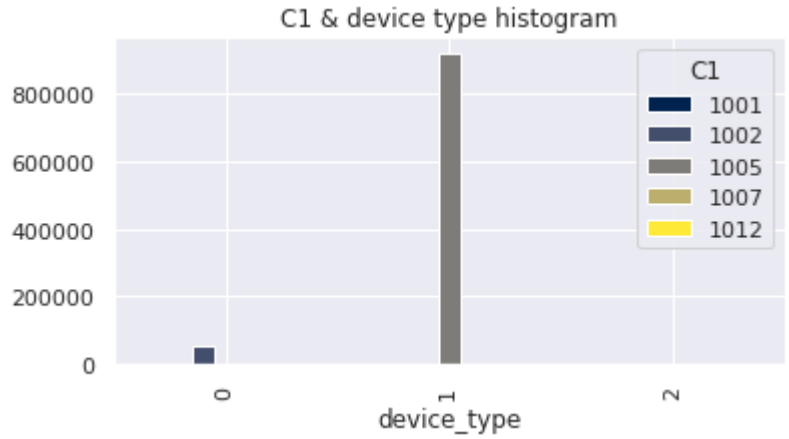
```
Name: C17, Length: 411, dtype: float64

C18 :
 0    0.414844
3    0.337404
2    0.179002
1    0.068750
Name: C18, dtype: float64

C19 :
 35      0.298614
39      0.221131
167     0.079538
161     0.039928
47      0.034378
          ...
1583    0.000092
290     0.000086
683     0.000019
1447    0.000002
553     0.000001
Name: C19, Length: 63, dtype: float64

C20 :
 -1         0.464346
 100084    0.060683
 100148    0.045260
 100111    0.043081
 100077    0.039388
             ...
 100186    0.000004
 100157    0.000002
 100098    0.000001
 100246    0.000001
 100078    0.000001
Name: C20, Length: 163, dtype: float64

C21 :
 23     0.223225
221    0.127764
79     0.111523
48     0.054092
71     0.053359
61     0.051459
157    0.046648
32     0.044942
33     0.037995
52     0.029845
42     0.024230
51     0.021612
15     0.019089
212    0.015665
43     0.014866
229    0.010372
117    0.010332
13     0.009222
16     0.008955
```

```
159    0.007384
95     0.007054
156    0.006721
68     0.005933
46     0.005883
246    0.004949
17     0.004227
69     0.004086
76     0.003467
111    0.003414
90     0.003292
91     0.003215
70     0.003181
171    0.002892
110    0.002093
253    0.001947
112    0.001796
82     0.001787
100    0.001684
178    0.001239
35     0.001195
182    0.001164
101    0.001085
108    0.000808
204    0.000673
94     0.000557
251    0.000521
116    0.000415
20     0.000356
194    0.000339
102    0.000241
93     0.000232
104    0.000223
163    0.000167
177    0.000145
126    0.000133
255    0.000126
1      0.000073
219    0.000051
195    0.000045
85     0.000009
Name: C21, dtype: float64
```

In [ ]:
```
print (df.groupby(['device_type','C1']).size().unstack())
df.groupby(['device_type','C1']).size().unstack().plot(kind='bar', figsize=(6,
3), title='C1 & device type histogram' ,cmap = 'cividis')
```

```
C1             1001     1002      1005   1007    1012
device_type
0               NaN  54816.0       NaN    NaN     NaN
1             239.0      NaN  918645.0  882.0  2736.0
2               NaN      NaN       NaN    NaN     1.0
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1970e87d30>



In [ ]:
```
z=df.groupby(['device_type','C1'])['click'].count().to_frame().reset_index()
z.style.background_gradient(cmap='Reds')
```

Out[ ]:

	device_type	C1	click
0	0	1002	54816
1	1	1001	239
2	1	1005	918645
3	1	1007	882
4	1	1012	2736
5	2	1012	1

In [ ]:
```
df.drop(['C20'], axis= 1)
```

Out[ ]:

	click	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain
0	0	1005	0	85f751fd	c4e18dd6	50e219e0	febd1138	82e27996
1	1	1005	1	d9750ee7	98572c79	f028772b	ecad2386	7801e8d9
2	1	1005	0	38217daf	449497bc	f028772b	ecad2386	7801e8d9
3	0	1005	0	85f751fd	c4e18dd6	50e219e0	685d1c4c	2347f47a
4	0	1005	1	c63170c5	a9bba545	f028772b	ecad2386	7801e8d9
...	...	...	...	...	...	...	...	...
999995	0	1005	0	85f751fd	c4e18dd6	50e219e0	f0d41ff1	2347f47a
999996	0	1005	0	bb4524e7	d733bbc3	28905ebd	ecad2386	7801e8d9
999997	0	1005	0	85f751fd	c4e18dd6	50e219e0	de97da65	33da2e74
999998	0	1005	1	5114c672	3f2f3819	3e814130	ecad2386	7801e8d9
999999	0	1005	0	85f751fd	c4e18dd6	50e219e0	e2fcccd2	5c5a694b

1000000 rows × 23 columns

Hash Function

In [ ]:
```
 numerical_features = df.select_dtypes(include=['int64','float64']).drop(['cli
ck'], axis=1).columns
categorical_features = df.select_dtypes(include=['object']).columns
```

In [ ]:
```
print('numerical features are : {0} \n  categorical features are : {1}'.format
([numerical_features],[categorical_features]))
```

```
numerical features are : [Index(['C1', 'banner_pos', 'device_type', 'device_c
onn_type', 'C14', 'C15',
       'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'hour_of_day',
       'day_of_month'],
      dtype='object')]
  categorical features are : [Index(['site_id', 'site_domain', 'site_categor
y', 'app_id', 'app_domain',
       'app_category', 'device_id', 'device_ip', 'device_model'],
      dtype='object')]
```

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

    object_columns = self.columns
    object_types = self.dtypes
    col_suffix = '_int'
    for index in range(0,len(object_columns)):
        if object_types[index] == object :
            self[object_columns[index]+col_suffix] = self[object_columns[index
]].map( lambda  x: hash(x))
            self.drop([object_columns[index]],inplace=True,axis=1)
    return self

df = convert_obj_to_int(df)
```

In [ ]:
```
df = convert_obj_to_int(df% (10**6))
```

In [ ]:
```
df.head()
```

Out[ ]:

	click	C1	banner_pos	device_type	device_conn_type	C14	C15	C16	C17	C18	C19
0	0	1005	0	1	0	21611	320	50	2480	3	297
1	1	1005	1	1	0	17753	320	50	1993	2	1063
2	1	1005	0	1	0	20345	300	250	2331	2	39
3	0	1005	0	1	3	15705	320	50	1722	0	35
4	0	1005	1	1	0	19772	320	50	2227	0	935

In [ ]:
```
df.dtypes
```

Out[ ]:
```
click                int64
C1                   int64
banner_pos           int64
device_type          int64
device_conn_type     int64
C14                  int64
C15                  int64
C16                  int64
C17                  int64
C18                  int64
C19                  int64
C20                  int64
C21                  int64
hour_of_day          int64
day_of_month         int64
site_id_int          int64
site_domain_int      int64
site_category_int    int64
app_id_int           int64
app_domain_int       int64
app_category_int     int64
device_id_int        int64
device_ip_int        int64
device_model_int     int64
dtype: object
```

outliers

In [ ]:
```
x = df.drop(['click'], axis=1)
y = df.click
```

In [ ]:
```
x.shape
```

Out[ ]:
```
(1000000, 23)
```

In [ ]:
```
from sklearn.neighbors import LocalOutlierFactor
```

In [ ]:
```
lof =LocalOutlierFactor(n_neighbors = 20,metric='manhattan')
```

In [ ]:
```
res = lof.fit_predict(x)
```

In [ ]:
```
scores = lof.negative_outlier_factor_
print(-scores)
```

```
[0.96325377 0.96433199 1.02535761 ... 0.9582447  0.97159752 0.98418808]
```

In [ ]:
```
x = x[res != -1]

y = y[res != -1]
```

In [ ]:
```
print(x.shape , y.shape)
```

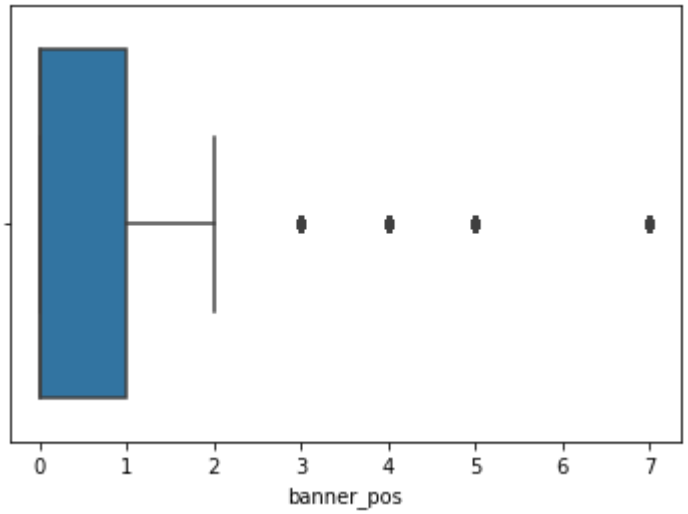
```
(903405, 23) (903405,)
```

Outliers with boxplot

In [ ]:
```

```

In [ ]:
```
sns.boxplot(x=df['banner_pos'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f28555f21d0>
```



In [ ]:
```
df_o =df[ ((df['banner_pos'] - df['banner_pos'].mean() )/
           df['banner_pos'].std()).abs() < 3.8]

print(df.shape, df_o.shape)
```

In [ ]:
```
df = df_o
```

In [ ]:
```
df.shape
```

In [ ]:
```
sns.boxplot(x=df['device_type'])
```

In [ ]:
```
df_o =df[ ((df['device_type'] - df['device_type'].mean() )/
           df['device_type'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

In [ ]:
```
sns.boxplot(x=df['C1'])
```

In [ ]:
```
df_o =df[ ((df['C1'] - df['C1'].mean() )/
           df['C1'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

In [ ]:
```
sns.boxplot(x=df['C14'])
```

In [ ]:
```
df_o =df[ ((df['C14'] - df['C14'].mean() )/
           df['C14'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

In [ ]:
```
sns.boxplot(x=df['C15'])
```

In [ ]:
```
df_o =df[ ((df['C15'] - df['C15'].mean() )/
           df['C15'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

In [ ]:
```
sns.boxplot(x=df['C19'])
```

In [ ]:
```
df_o =df[ ((df['C19'] - df['C19'].mean() )/
           df['C19'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

In [ ]:
```
sns.boxplot(x=df['device_conn_type'])
```

In [ ]:
```
df_o =df[ ((df['device_conn_type'] - df['device_conn_type'].mean() )/
           df['device_conn_type'].std()).abs() < 3.8]
print(df_o.shape , df.shape)
```

In [ ]:
```
df = df_o
df.shape
```

feature selection

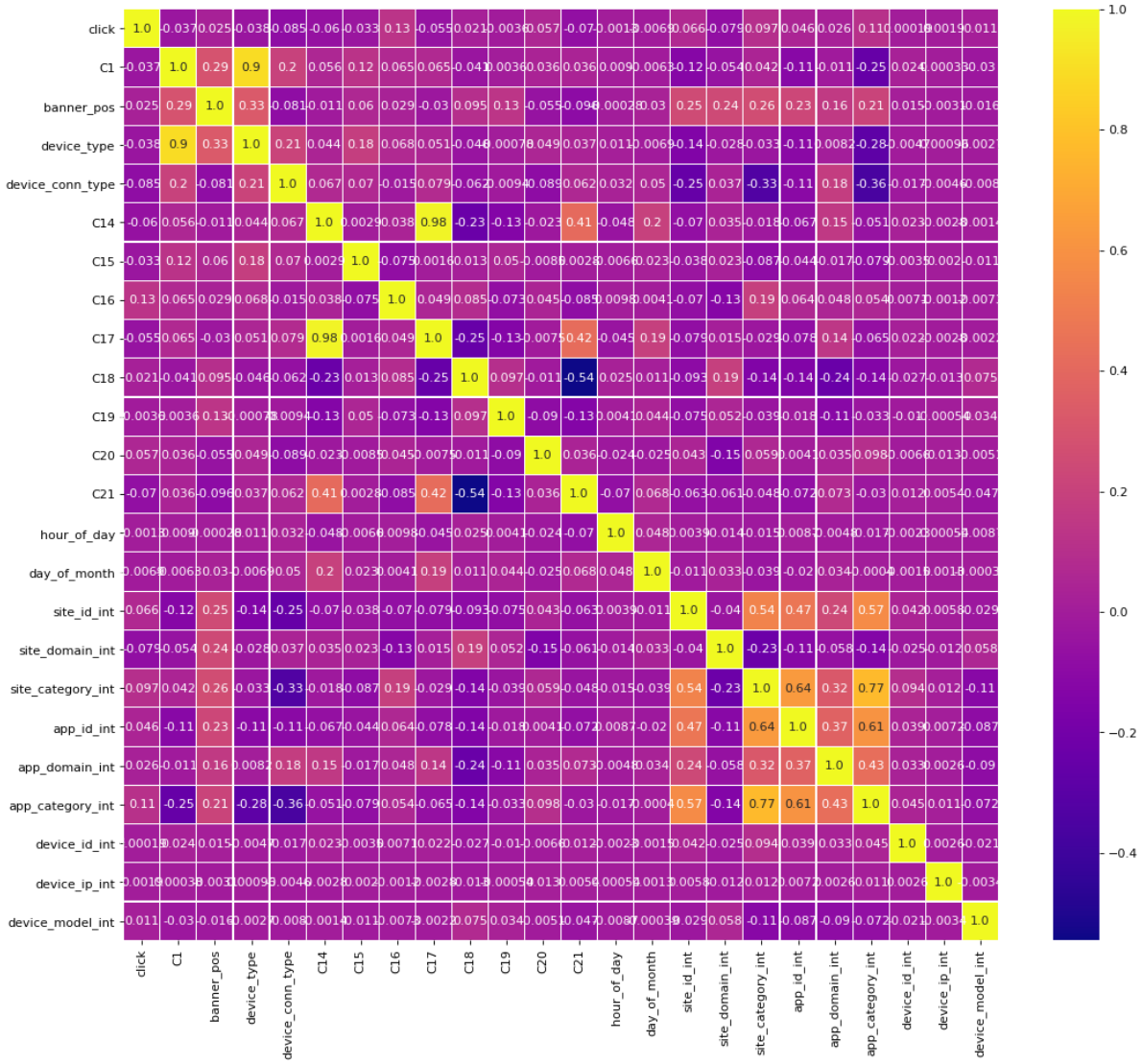
In [ ]:
```
corr = df.corr()
corr
```

Out[ ]:

	click	C1	banner_pos	device_type	device_conn_type	C14
click	1.000000	-0.037348	0.024850	-0.038206	-0.084622	-0.060058
C1	-0.037348	1.000000	0.290193	0.895021	0.198065	0.055954
banner_pos	0.024850	0.290193	1.000000	0.325933	-0.080710	-0.011075
device_type	-0.038206	0.895021	0.325933	1.000000	0.210882	0.044243
device_conn_type	-0.084622	0.198065	-0.080710	0.210882	1.000000	0.067049
C14	-0.060058	0.055954	-0.011075	0.044243	0.067049	1.000000
C15	-0.032623	0.121532	0.059785	0.182330	0.069694	0.002866
C16	0.129680	0.065115	0.029307	0.068111	-0.015454	0.038278
C17	-0.055493	0.064911	-0.030104	0.050698	0.079294	0.976696
C18	0.021024	-0.040874	0.095107	-0.045738	-0.062387	-0.231457
C19	-0.003612	0.003640	0.133628	-0.000784	-0.009374	-0.133615
C20	0.056693	0.036167	-0.054677	0.049210	-0.088933	-0.022967
C21	-0.070129	0.036354	-0.095782	0.036638	0.062464	0.409866
hour_of_day	-0.001296	0.008996	-0.000278	0.011468	0.032295	-0.047851
day_of_month	-0.006854	-0.006330	0.029692	-0.006885	0.050176	0.197096
site_id_int	0.066491	-0.123886	0.247540	-0.142523	-0.250194	-0.069944
site_domain_int	-0.079436	-0.053657	0.239949	-0.028215	0.037344	0.035251
site_category_int	0.097461	0.041598	0.260033	-0.033356	-0.331484	-0.017524
app_id_int	0.045502	-0.111652	0.233108	-0.107358	-0.106938	-0.067464
app_domain_int	0.026043	-0.010763	0.159200	0.008171	0.176448	0.151016
app_category_int	0.109595	-0.251712	0.211665	-0.280892	-0.362352	-0.050788
device_id_int	0.000193	0.024295	0.014722	-0.004740	-0.017341	0.023052
device_ip_int	0.001919	0.000331	-0.003118	0.000958	-0.004619	-0.002821
device_model_int	0.010740	-0.030007	-0.016199	-0.002736	-0.008024	-0.001423

In [ ]:
```
plt.figure(figsize=(16,14), dpi= 80)
sns.heatmap(corr, annot=True , fmt=".2" ,linewidths=.1, cmap='plasma')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb127f66fd0>



In [ ]:
```
from sklearn.feature_selection import SelectFromModel
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.pipeline import FeatureUnion
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [ ]:
```
model = LogisticRegression()
combined_features = FeatureUnion([('select_best', SelectKBest(score_func=chi2,
k=10)),(('RFE', RFE(model , 8))) ])

x_features = combined_features.fit(x, y).transform(x)

print("Combined space has", x_features.shape[1], "features")
```

```
Combined space has 18 features
```

In [ ]:
```
x = x_features
```

In [ ]:
```
x.shape
```

Out[ ]:
```
(903405, 18)
```

In [ ]:
```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.01, rando
m_state=1)

x_train, x_val, y_train, y_val  = train_test_split(x_train, y_train, train_siz
e=0.99, random_state=1)
  # 0.25 x 0.8 = 0.2

print('valid', x_val.shape, y_val.shape)
print('Train', x_train.shape, y_train.shape)
print('Test', x_test.shape, y_test.shape)
```

```
valid (8944, 18) (8944,)
Train (885426, 18) (885426,)
Test (9035, 18) (9035,)
```

Oversampling data

In [1]:
```
pip install imblearn
```

```
Note: you may need to restart the kernel to use updated packages.

'D:\machinelearning' is not recognized as an internal or external command,
operable program or batch file.
```

In [ ]:
```
from imblearn.over_sampling import SMOTE
```

In [ ]:
```
y_train.value_counts()
```

Out[ ]:
```
0    732430
1    152996
Name: click, dtype: int64
```

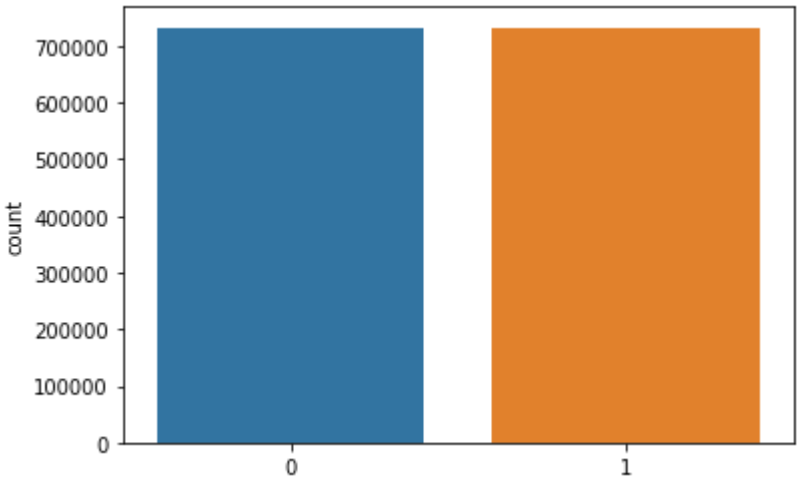
In [ ]:
```
smt = SMOTE()
x_train , y_train = smt.fit_sample(x_train , y_train)
```

In [ ]:
```
np.bincount(y_train)
```

Out[ ]:
```
array([732430, 732430])
```

In [ ]:
```
sns.countplot(y_train)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1251f1048>
```



In [ ]:
```
x_train.shape
```

Out[ ]:
```
(1464860, 18)
```

Data Transformation

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

scaler = scaler.fit(x_train)
rescaled = scaler.transform(x_train)
```

In [ ]:
```
rescaled
```

Out[ ]:
```
array([[-0.57677806, -0.28319613,  1.01650776, ..., -0.53383958,
         0.00342917,  0.59616586],
       [ 0.96933254, -0.28319613, -0.98367141, ...,  3.12003152,
        -0.39130516,  0.59616586],
       [ 1.02632187, -0.28319613,  1.01650776, ..., -0.49990269,
        -0.81527907,  0.12055094],
       ...,
       [ 0.52021661, -0.28319613,  1.01650776, ..., -0.42637278,
         0.17886665, -1.86254663],
       [-0.57557828, -0.28319613,  1.01650776, ..., -0.53383958,
         0.00342917,  0.59616586],
       [-2.40503584, -0.28319613,  1.01650776, ..., -0.52252728,
        -0.68370096,  0.59616586]])
```

In [ ]:

spot check classification algorithms

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score , average_precision_score , precisi
on_recall_curve , f1_score ,roc_curve ,auc
```

In [ ]:
```
#LogisticRegression
lr_c=LogisticRegression(random_state=0)
lr_c.fit(x_train,y_train)
lr_pred=lr_c.predict(x_val)
lr_cm=confusion_matrix(y_val,lr_pred)
lr_ac=accuracy_score(y_val, lr_pred)
```

In [ ]:
```
#GradientBoostingClassifier
GB_c=GradientBoostingClassifier(random_state=0)
GB_c.fit(x_train,y_train)
GB_pred=GB_c.predict(x_val)
GB_cm=confusion_matrix(y_val,GB_pred)
GB_ac=accuracy_score(y_val, GB_pred)
```

In [ ]:
```
#Bayes
gaussian=GaussianNB()
gaussian.fit(x_train,y_train)
bayes_pred=gaussian.predict(x_val)
bayes_cm=confusion_matrix(y_val,bayes_pred)
bayes_ac=accuracy_score(bayes_pred,y_val)
```

In [ ]:
```
#RandomForest
rdf_c=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=
0)
rdf_c.fit(x_train,y_train)
rdf_pred=rdf_c.predict(x_val)
rdf_cm=confusion_matrix(y_val,rdf_pred)
rdf_ac=accuracy_score(rdf_pred,y_val)
```

In [ ]:
```
# DecisionTree Classifier
dtree_c=DecisionTreeClassifier(criterion='entropy',random_state=0)
dtree_c.fit(x_train,y_train)
dtree_pred=dtree_c.predict(x_val)
dtree_cm=confusion_matrix(y_val,dtree_pred)
dtree_ac=accuracy_score(dtree_pred,y_val)
```

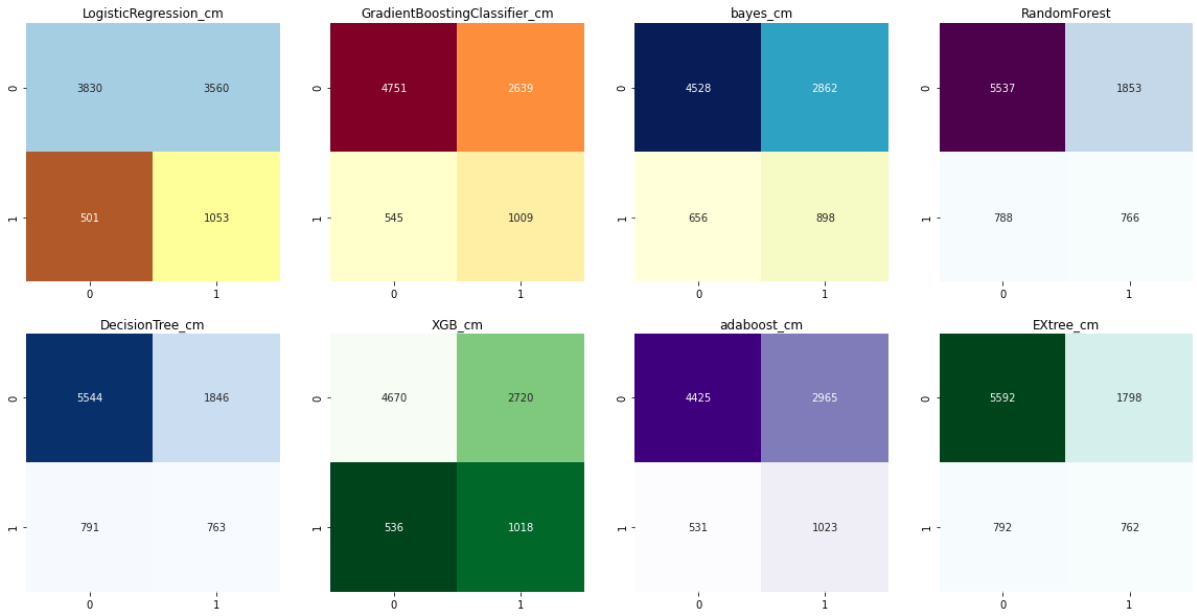
In [ ]:
```
#XGBoost
XGB=XGBClassifier()
XGB.fit(x_train,y_train)
XGB_pred=XGB.predict(x_val)
XGB_cm=confusion_matrix(y_val,XGB_pred)
XGB_ac=accuracy_score(XGB_pred,y_val)
```

In [ ]:
```
#AdaBoost

adaboost=AdaBoostClassifier()
adaboost.fit(x_train,y_train)
adaboost_pred=adaboost.predict(x_val)
adaboost_cm=confusion_matrix(y_val,adaboost_pred)
adaboost_ac=accuracy_score(adaboost_pred,y_val)
```

In [ ]:
```
#ExtraTreesClassifier
EXtree=ExtraTreesClassifier(criterion='entropy',random_state=0)
EXtree.fit(x_train,y_train)
EXtree_pred=EXtree.predict(x_val)
EXtree_cm=confusion_matrix(y_val,EXtree_pred)
EXtree_ac=accuracy_score(EXtree_pred,y_val)
```

In [ ]:
```
plt.figure(figsize=(20,10))
target_names = ["impression", "click"]

plt.subplot(2,4,1)
plt.title("LogisticRegression_cm")
sns.heatmap(lr_cm,annot=True,cmap="Paired_r",fmt="d",cbar=False)


plt.subplot(2,4,2)
plt.title("GradientBoostingClassifier_cm")
sns.heatmap(GB_cm,annot=True,cmap="YlOrRd",fmt="d",cbar=False)


plt.subplot(2,4,3)
plt.title("bayes_cm")
sns.heatmap(bayes_cm,annot=True,cmap="YlGnBu",fmt="d",cbar=False)

plt.subplot(2,4,4)
plt.title("RandomForest")
sns.heatmap(rdf_cm,annot=True,cmap="BuPu",fmt="d",cbar=False)


plt.subplot(2,4,5)
plt.title("DecisionTree_cm")
sns.heatmap(dtree_cm,annot=True,cmap="Blues",fmt="d",cbar=False)


plt.subplot(2,4,6)
plt.title("XGB_cm")
sns.heatmap(XGB_cm,annot=True,cmap="Greens_r",fmt="d",cbar=False)


plt.subplot(2,4,7)
plt.title("adaboost_cm")
sns.heatmap(adaboost_cm,annot=True,cmap="Purples",fmt="d",cbar=False)


plt.subplot(2,4,8)
plt.title("EXtree_cm")
sns.heatmap(EXtree_cm,annot=True,cmap="BuGn",fmt="d",cbar=False)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb119b3de48>



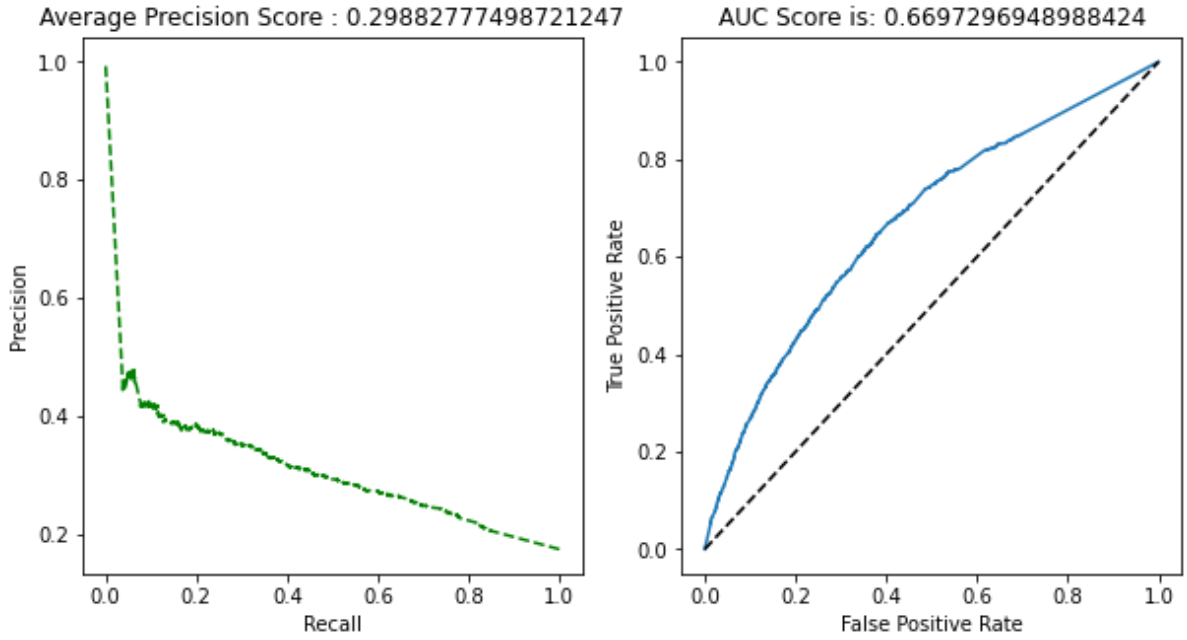
In [ ]:
```
print('LogisticRegression_accuracy:\t',lr_ac)
print('GradientBoostingClassifier:\t',GB_ac)
print('Bayes_accuracy:\t\t\t',bayes_ac)
print('RandomForest_accuracy:\t\t',rdf_ac)
print('DecisionTree_accuracy:\t\t',dtree_ac)
print('XGB_accuracy:\t\t\t',XGB_ac)
print('AdaBoost_accuracy:\t\t',adaboost_ac)
print('ExtraTreesClassifier:\t\t' , EXtree_ac)
```

```
LogisticRegression_accuracy:    0.5459525939177102
GradientBoostingClassifier:     0.6440071556350626
Bayes_accuracy:                 0.6066636851520573
RandomForest_accuracy:          0.7047182468694096
DecisionTree_accuracy:          0.7051654740608229
XGB_accuracy:                   0.6359570661896243
AdaBoost_accuracy:              0.6091234347048301
ExtraTreesClassifier:           0.7104203935599285
```

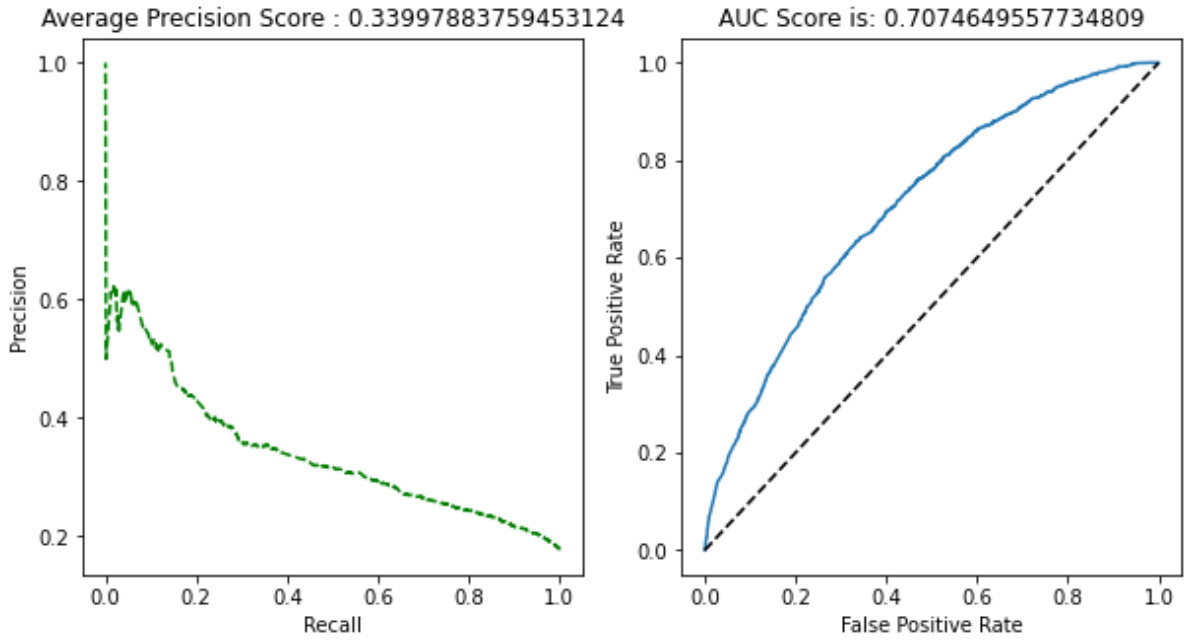
In [ ]:
```
def plotting(true,pred):
    fig,ax=plt.subplots(1,2,figsize=(10,5))

    precision,recall,threshold = precision_recall_curve(true,pred[:,1])
    ax[0].plot(recall,precision,'g--')
    ax[0].set_xlabel('Recall')
    ax[0].set_ylabel('Precision')
    ax[0].set_title("Average Precision Score : {}".format(average_precision_sc
ore(true,pred[:,1])))


    fpr,tpr,threshold = roc_curve(true,pred[:,1])
    ax[1].plot(fpr,tpr)
    ax[1].set_title("AUC Score is: {}".format(auc(fpr,tpr)))
    ax[1].plot([0,1],[0,1],'k--')
    ax[1].set_xlabel('False Positive Rate')
    ax[1].set_ylabel('True Positive Rate')
```

In [ ]:
```
plt.figure()
plotting(y_val ,rdf_c.predict_proba(x_val))


plt.figure()
plotting(y_val ,XGB.predict_proba(x_val))
```

```
<Figure size 432x288 with 0 Axes>
```

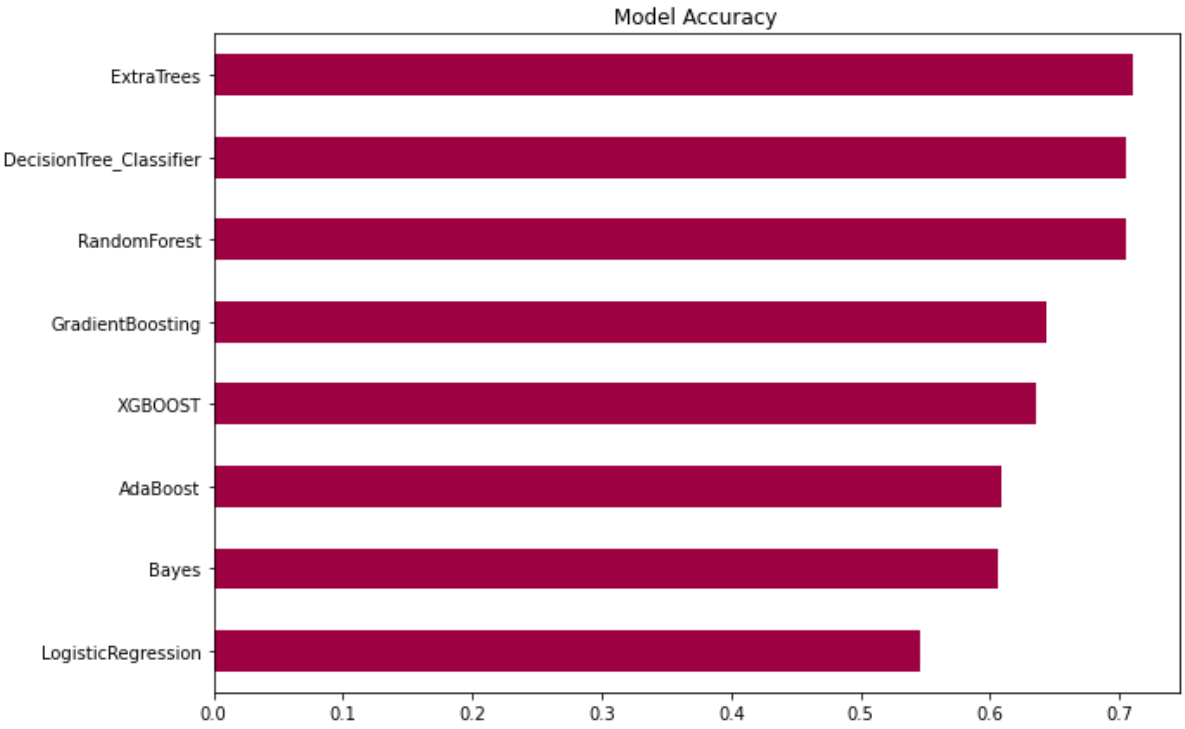



```
<Figure size 432x288 with 0 Axes>
```



In [ ]:
```
model_accuracy = pd.Series(data=[lr_ac, GB_ac,bayes_ac,rdf_ac,dtree_ac,XGB_ac,
adaboost_ac, EXtree_ac],
                index=['LogisticRegression','GradientBoosting','Bayes',
                                        'RandomForest','DecisionTree_Classifier'
,'XGBOOST' , 'AdaBoost', 'ExtraTrees' ])
fig= plt.figure(figsize=(10,7))
model_accuracy.sort_values().plot.barh(cmap = 'Spectral')
plt.title('Model Accuracy')
```

Out[ ]: Text(0.5, 1.0, 'Model Accuracy')



In [ ]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
recall_score, precision_recall_curve


print('precision_score of LR: ',  precision_score(lr_pred, y_val))
print('precision_score of GB: ',  precision_score(GB_pred, y_val))
print('precision_score of gaussian: ',  precision_score(bayes_pred, y_val))
print('precision_score of random forest : ',  precision_score(rdf_pred, y_val
))
print('precision_score of DTree: ',  precision_score(dtree_pred, y_val))
print('precision_score of XGB: ',  precision_score(XGB_pred, y_val))
print('precision_score of adaboost: ',  precision_score(adaboost_pred, y_val))
print('precision_score of EXtree: ',  precision_score(EXtree_pred, y_val))
```

```
precision_score of LR:  0.6776061776061776
precision_score of GB:  0.6492921492921493
precision_score of gaussian:  0.5778635778635779
precision_score of random forest :  0.4929214929214929
precision_score of DTree:  0.49099099099099097
precision_score of XGB:  0.6550836550836551
precision_score of adaboost:  0.6583011583011583
precision_score of EXtree:  0.49034749034749037
```

In [ ]:
```
print('recall_score of LR: ',  recall_score(lr_pred, y_val))
print('recall_score of GB: ',  recall_score(GB_pred, y_val))
print('recall_score of gaussian: ',  recall_score(bayes_pred, y_val))
print('recall_score of random forest : ',  recall_score(rdf_pred, y_val))
print('recall_score of DTree: ',  recall_score(dtree_pred, y_val))
print('recall_score of XGB: ',  recall_score(XGB_pred, y_val))
print('recall_score of adaboost: ',  recall_score(adaboost_pred, y_val))
print('recall_score of EXtree: ',  recall_score(EXtree_pred, y_val))
```

```
recall_score of LR:  0.22826793843485801
recall_score of GB:  0.27658991228070173
recall_score of gaussian:  0.23882978723404255
recall_score of random forest :  0.2924780450553646
recall_score of DTree:  0.29244921425833653
recall_score of XGB:  0.2723381487426431
recall_score of adaboost:  0.2565195586760281
recall_score of EXtree:  0.29765625
```

In [ ]:
```
print('f1_score of LR: ',  f1_score(lr_pred, y_val))
print('f1_score of GB: ',  f1_score(GB_pred, y_val))
print('f1_score of gaussian: ',  f1_score(bayes_pred, y_val))
print('f1_score of random forest : ',  f1_score(rdf_pred, y_val))
print('f1_score of DTree: ',  f1_score(dtree_pred, y_val))
print('f1_score of XGB: ',  f1_score(XGB_pred, y_val))
print('f1_score of adaboost: ',  f1_score(adaboost_pred, y_val))
print('f1_score of EXtree: ',  f1_score(EXtree_pred, y_val))
```

```
f1_score of LR:  0.341495054321388
f1_score of GB:  0.38792772010765086
f1_score of gaussian:  0.3379751599548363
f1_score of random forest :  0.3671219745986101
f1_score of DTree:  0.36656257506605816
f1_score of XGB:  0.38473167044595613
f1_score of adaboost:  0.3691808011548178
f1_score of EXtree:  0.37044239183276617
```

GridSearch

In [ ]:
```
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
rf = RandomForestClassifier()
n_estimators = [500,1000,2000]
max_depth = [3,5,8,10]
min_samples_split = [3,5]
min_samples_leaf = [1]

hyperF = dict (n_estimators = n_estimators, max_depth = max_depth,
               min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf)

gridF = GridSearchCV(rf , hyperF, cv = 3, verbose = 1)
bestF = gridF.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
```

In [ ]:
```
gridF.best_params_
```

In [ ]:
```
gridF.best_score_
```

Out[ ]:
```
0.824533983796571
```

In [ ]:
```
bm = gridF.best_model_
```

model

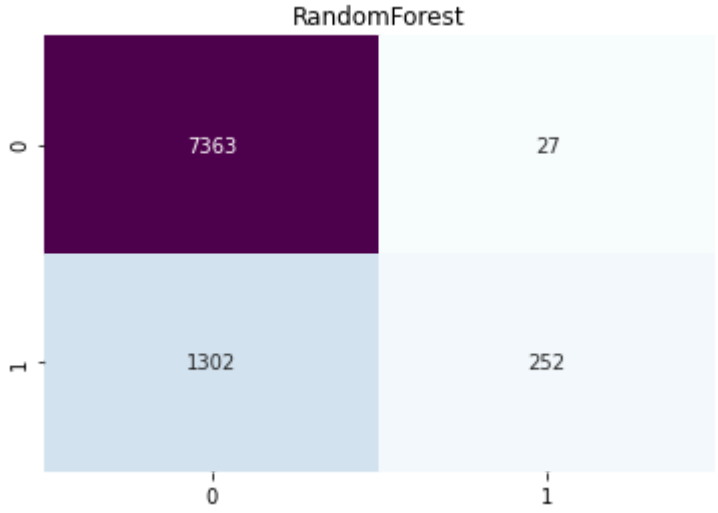
In [ ]:
```
model = RandomForestClassifier(n_estimators = 1000, max_depth = 11,
               min_samples_split = 3,
              min_samples_leaf = 1)
model.fit(x_val, y_val)

model_pred=model.predict(x_val)
model_cm=confusion_matrix(y_val,model_pred)
model_ac=accuracy_score(model_pred,y_val)
```

In [ ]:
```
print('RandomForest_accuracy:\t\t',model_ac)

print('recall_score : ',  recall_score(model_pred, y_val))

print('precision_score : ',  precision_score(model_pred, y_val))

print('f1_score : ',  f1_score(model_pred, y_val))
```

```
RandomForest_accuracy:          0.8514087656529516
recall_score :  0.9032258064516129
precision_score :  0.16216216216216217
f1_score :  0.2749590834697218
```

In [ ]:
```
plt.title("RandomForest")
sns.heatmap(model_cm,annot=True,cmap="BuPu",fmt="d",cbar=False)
```

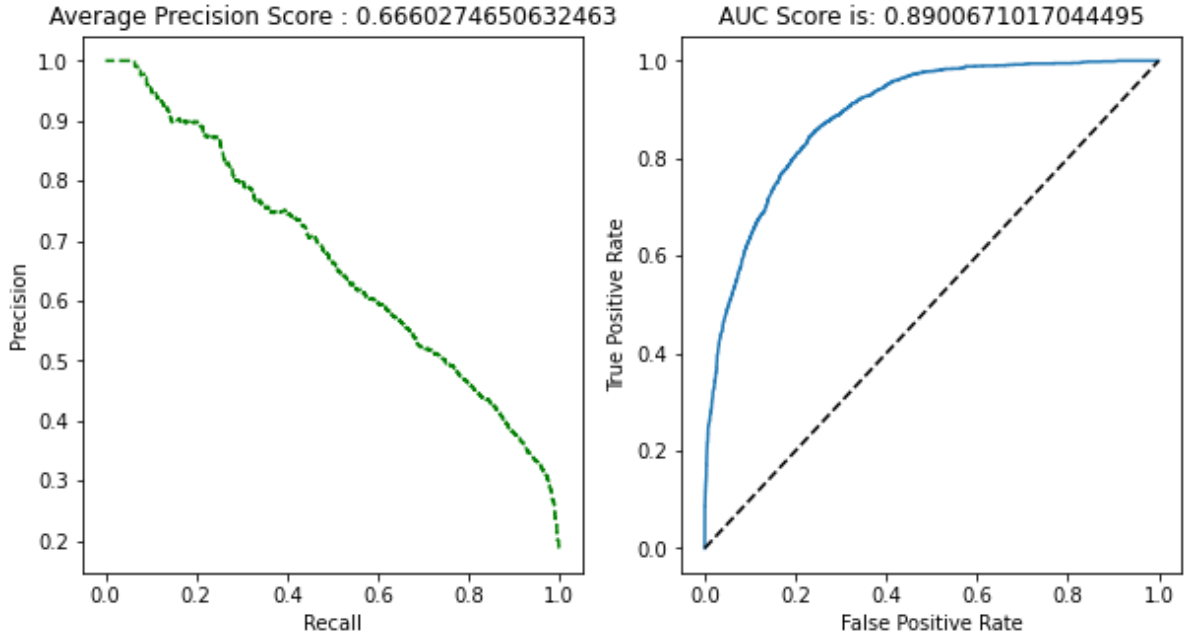
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1193c1e10>
```



In [ ]:
```
def plotting(true,pred):
    fig,ax=plt.subplots(1,2,figsize=(10,5))

    precision,recall,threshold = precision_recall_curve(true,pred[:,1])
    ax[0].plot(recall,precision,'g--')
    ax[0].set_xlabel('Recall')
    ax[0].set_ylabel('Precision')
    ax[0].set_title("Average Precision Score : {}".format(average_precision_sc
ore(true,pred[:,1])))


    fpr,tpr,threshold = roc_curve(true,pred[:,1])
    ax[1].plot(fpr,tpr)
    ax[1].set_title("AUC Score is: {}".format(auc(fpr,tpr)))
    ax[1].plot([0,1],[0,1],'k--')
    ax[1].set_xlabel('False Positive Rate')
    ax[1].set_ylabel('True Positive Rate')
```

In [ ]:
```
plt.figure()
plotting(y_val ,model.predict_proba(x_val))
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]:
```
accuracy = model.score(x_test, y_test)
print("Accuracy is %.2f %%" %(accuracy * 100))
```

```
Accuracy is 81.91 %
```

In [ ]:
```
pred_classes = model.predict(x_test)

print("Predicted classes:")
print(pred_classes)
print("Actual classes:")
print(y_test)
```

```
Predicted classes:
[0 0 0 ... 0 0 0]
Actual classes:
216768    0
326753    0
772476    0
897893    0
209569    0
         ..
470414    0
852658    0
136703    0
387169    0
230451    0
Name: click, Length: 9035, dtype: int64
```

In [ ]:
```
from sklearn.metrics import roc_auc_score
```

In [ ]:
```
y_pred = model.predict(x_test)
print("Roc_auc_score: ",roc_auc_score(y_test,y_pred)*100,"%")
```

```
Roc_auc_score:  52.60822515829305 %
```

model 1

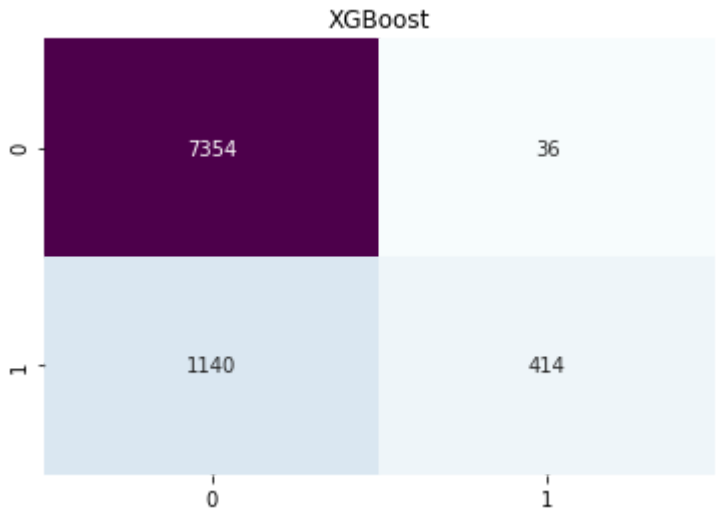
In [ ]:
```
model1 = XGBClassifier(min_child_weight=1,eta=0.1 , max_depth= 10 , n_estimato
rs= 2000 ,
                       booster= 'gbtree' ,learning_rate=0.01 , gamma= 1 )
model1.fit(x_val, y_val)

model1_pred=model1.predict(x_val)
model1_cm=confusion_matrix(y_val,model1_pred)
model1_ac=accuracy_score(model1_pred,y_val)
```

In [ ]:
```
print('XGB_accuracy:\t\t',model1_ac)

print('recall_score : ',  recall_score(model1_pred, y_val))

print('precision_score : ',  precision_score(model1_pred, y_val))

print('f1_score : ',  f1_score(model1_pred, y_val))
```

```
XGB_accuracy:           0.868515205724508
recall_score :  0.92
precision_score :  0.26640926640926643
f1_score :  0.4131736526946108
```

In [ ]:
```
plt.title("XGBoost")
sns.heatmap(model1_cm,annot=True,cmap="BuPu",fmt="d",cbar=False)
```

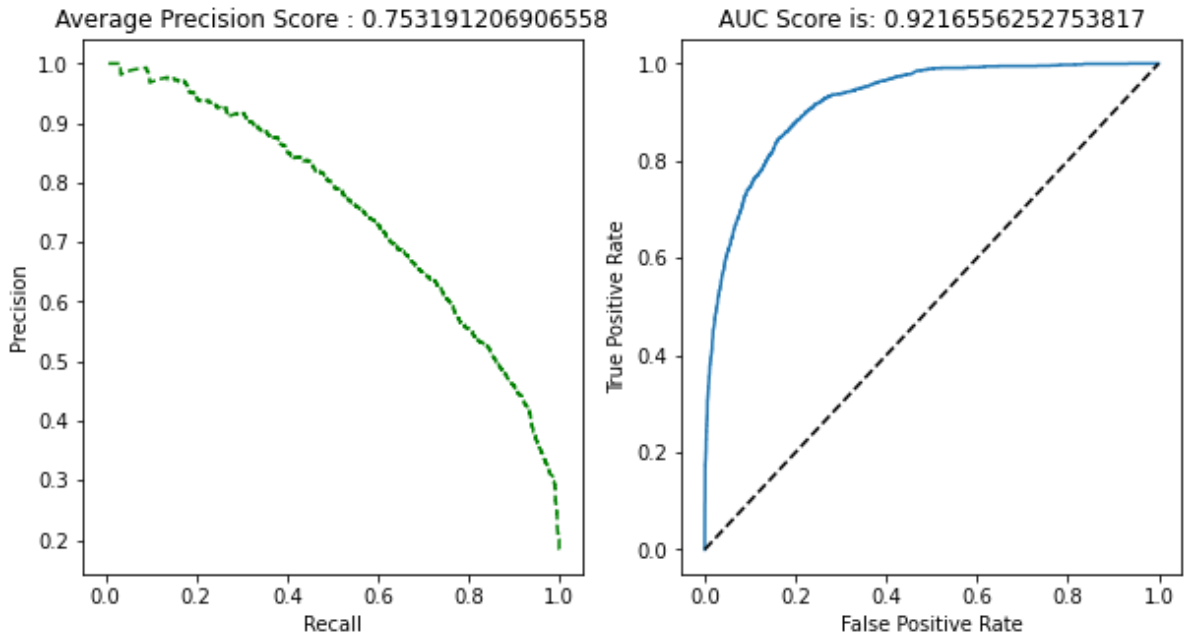
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb1115541d0>
```



In [ ]:
```
def plotting(true,pred):
    fig,ax=plt.subplots(1,2,figsize=(10,5))

    precision,recall,threshold = precision_recall_curve(true,pred[:,1])
    ax[0].plot(recall,precision,'g--')
    ax[0].set_xlabel('Recall')
    ax[0].set_ylabel('Precision')
    ax[0].set_title("Average Precision Score : {}".format(average_precision_sc
ore(true,pred[:,1])))


    fpr,tpr,threshold = roc_curve(true,pred[:,1])
    ax[1].plot(fpr,tpr)
    ax[1].set_title("AUC Score is: {}".format(auc(fpr,tpr)))
    ax[1].plot([0,1],[0,1],'k--')
    ax[1].set_xlabel('False Positive Rate')
    ax[1].set_ylabel('True Positive Rate')
```

In [ ]:
```
plt.figure()
plotting(y_val ,model1.predict_proba(x_val))
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]:

In [ ]:
```
accuracy = model1.score(x_test, y_test)
print("Accuracy is %.2f %%" %(accuracy * 100))
```

```
Accuracy is 81.54 %
```

In [ ]:
```
pred_classes = model1.predict(x_test)

print("Predicted classes:")
print(pred_classes)
print("Actual classes:")
print(y_test)
```

```
Predicted classes:
[0 0 0 ... 0 0 0]
Actual classes:
216768    0
326753    0
772476    0
897893    0
209569    0
         ..
470414    0
852658    0
136703    0
387169    0
230451    0
Name: click, Length: 9035, dtype: int64
```

In [ ]:
```
from sklearn.metrics import roc_auc_score
```

In [ ]:
```
y_pred = model1.predict(x_test)
print("Roc_auc_score: ",roc_auc_score(y_test,y_pred)*100,"%")
```

```
Roc_auc_score:  53.306846791891196 %
```

GRIDSEARCH XGB

In [ ]:
```
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
XGB = XGBClassifier()
n_estimators = [500,1000,2000]
min_child_weight=[0,1,2]
eta =[0.1, 0.3]
max_depth = [5,8,10]
learning_rate=[0.05 , 0.1 , 0.2]
gamma= [0 ,1 ,2,5]


hyperF = dict (n_estimators = n_estimators, max_depth = max_depth,
               min_child_weight = min_child_weight,
              eta = eta ,learning_rate =learning_rate , gamma = gamma , )

gridF = GridSearchCV(XGB , hyperF, cv = 3, verbose = 1)
bestF = gridF.fit(x_train, y_train)
```

In [ ]:
```
gridF.best_params_
```

In [ ]:
```
gridF.best_score_
```

In [ ]:
```
bm = gridF.best_model_
```

In [ ]:
```

```

save & load model

In [ ]:
```
# save the model to disk with pickle
from pickle import dump

filename = 'finalized_RF_model.sav'
dump(model, open(filename, 'wb'))
```

In [ ]:
```
# load the model from disk with pickle
from pickle import load

loaded_model = load(open(filename, 'rb'))
result = loaded_model.score(x_test, y_test)
print(result)
```

```
0.819147758716104
```

In [ ]:
```
# save the model to disk with pickle
from pickle import dump

filename = 'finalized_XGB_model.sav'
dump(model1, open(filename, 'wb'))
```

In [ ]:
```
# load the model from disk with pickle
from pickle import load

loaded_model1 = load(open(filename, 'rb'))
result = loaded_model1.score(x_test, y_test)
print(result)
```

```
0.8153846153846154
```

In [ ]:
```

```