Real Time Bidding Project

Evelyn Ding, PhD

May 11, 2021

Valassis Digital has run a digital advertising campaign for one of our clients, targeting mobile users in the Southeast. The goal of the campaign was to drive conversions - users clicking through the ad and accepting our client's offer. To execute this campaign, we built an audience of target users based on historical information about the likelihood to convert.

The goal of this project is to exploratory the distributions related to requests and the users targeted, to analyze the possibility to convert between test and control groups, to evaluate the chance to convert retargeting users, and finally to predict the convert possibility by using machine learning models.

As ever, the contents of this project are listed as below:

1 Sourcing and loading

[781]:
```
import re
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import scale, StandardScaler
from pathlib import Path
import numpy as np
import re
from sklearn.metrics import accuracy_score
import math
import imblearn
from sklearn.linear_model import LogisticRegression
import ast
```

[83]:
```
# import user_attributes.csv
df_user = pd.read_csv('./user_attributes.csv/user_attributes.csv')
df_attributes = pd.DataFrame([ast.literal_eval(i) for i in df_user['attributes'].
 ↪values])
df_user= df_user.join(df_attributes).drop(['attributes'], axis = 1)
df_user.head()
```

[83]:

	user_id	age	gender	location	test
0	00003e3b9e5336685200ae85d21b4f5e	33	F	FL	1
1	000053b1e684c9e7ea73727b2238ce18	26	M	AL	1
2	00029153d12ae1c9abe59c17ff2e0895	29	F	AR	1
3	0002ac0d783338cfeab0b2bdbd872cda	29	M	SC	0
4	0004d0b59e19461ff126e3a08a814c33	27	F	AR	1

[84]:
```
# import bid_requests.csv
df_bid = pd.read_csv('./bid_requests.csv/bid_requests.csv',␣
 ↪parse_dates=['timestamp'])
df_bid.head()
```

[84]:

	timestamp	user_id	bid	win	conversion
0	2017-01-01	be7485be5b6eb3690efcbc9e95e8f15a	0	0	0
1	2017-01-01	26c5dca2512a4c7fe8810bd04191b1b3	0	0	0
2	2017-01-01	2121376a323507c01c5e92c39ae8ccd4	0	0	0
3	2017-01-01	fa6a0925d911185338b0acc93c66dc92	0	0	0
4	2017-01-01	4299f209da83da82b711f1d631cc607b	1	0	0

[85]:
```
# combine user_attributes.csv and bid_requests.csv
df = df_user.merge(df_bid, on = 'user_id')
df = df[['timestamp', 'user_id', 'age', 'gender', 'location', 'test',  'bid',
       'win', 'conversion']]
```

```python
df.head()
```

[85]:
```
             timestamp                           user_id  age gender location  \
0 2017-01-01 13:43:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
1 2017-01-04 03:59:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
2 2017-01-04 17:41:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
3 2017-01-07 04:02:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
4 2017-01-08 09:05:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL

   test  bid  win  conversion
0     1    1    0           0
1     1    0    0           0
2     1    1    1           0
3     1    1    1           0
4     1    1    0           0
```

[86]:
```python
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 600000 entries, 0 to 599999
Data columns (total 9 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   timestamp   600000 non-null  datetime64[ns]
 1   user_id     600000 non-null  object
 2   age         600000 non-null  int64
 3   gender      600000 non-null  object
 4   location    600000 non-null  object
 5   test        600000 non-null  int64
 6   bid         600000 non-null  int64
 7   win         600000 non-null  int64
 8   conversion  600000 non-null  int64
dtypes: datetime64[ns](1), int64(5), object(3)
memory usage: 45.8+ MB
```

[86]:
```
                 age           test            bid            win  \
count  600000.000000  600000.000000  600000.000000  600000.000000
mean       25.506995       0.563155       0.499735       0.250265
std         4.599157       0.495996       0.500000       0.433166
min        18.000000       0.000000       0.000000       0.000000
25%        22.000000       0.000000       0.000000       0.000000
50%        25.000000       1.000000       0.000000       0.000000
75%        29.000000       1.000000       1.000000       1.000000
max        33.000000       1.000000       1.000000       1.000000

          conversion
count  600000.000000
```

```
mean        0.010192
std         0.100438
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
```

[9]:
```
df["age"].value_counts().sort_index(ascending=False)
```

[9]:
```
33    37981
32    36432
31    38235
30    37203
29    37268
28    37351
27    37770
26    37727
25    37893
24    37362
23    38013
22    37418
21    37943
20    37750
19    37250
18    36404
Name: age, dtype: int64
```

[10]:
```
df["gender"].value_counts()
```

[10]:
```
M    300126
F    299874
Name: gender, dtype: int64
```

[11]:
```
df["location"].value_counts()
```

[11]:
```
KY    55222
TN    55153
AR    55106
NC    55084
SC    55021
AL    54889
VA    54237
MS    54173
FL    53860
LA    53630
GA    53625
```

```
Name: location, dtype: int64
```

[12]:
```
pd.DataFrame(df["test"].value_counts())
```

[12]:

	test
1	337893
0	262107

[13]:
```
pd.DataFrame(df["bid"].value_counts()).join(pd.DataFrame(df["win"].
 ↪value_counts())).join(pd.DataFrame(df["conversion"].value_counts()))
```

[13]:

	bid	win	conversion
0	300159	449841	593885
1	299841	150159	6115

2 Cleaning, Transforming and Visualizing

2.1 EDA - Date Analysis

[337]:
```
df["timestamp"].value_counts()
```

[337]:
```
2017-01-14 22:15:00    37
2017-01-07 02:37:00    36
2017-01-20 20:45:00    34
2017-01-07 05:35:00    33
2017-01-20 03:32:00    33
                       ..
2017-01-04 06:57:00     8
2017-01-19 05:42:00     8
2017-01-23 13:19:00     8
2017-01-23 01:34:00     8
2017-01-22 07:58:00     7
Name: timestamp, Length: 32480, dtype: int64
```

[339]:
```
df["date_cat"] = df["timestamp"].astype("category")
df["date_weekday"] = df['date_cat'].dt.dayofweek
df["date_day"] = df['date_cat'].dt.day
df["date_hour"] = df['date_cat'].dt.hour
print(df.groupby("date_weekday").size())
print(df.groupby("date_day").size())
print(df.groupby("date_hour").size())
```

```
date_weekday
0      94171
1      79789
2      79914
3      80109
4      79848
```

```
5     79670
6    106499
dtype: int64
date_day
1     26379
2     26562
3     26525
4     26725
5     26541
6     26508
7     26563
8     26676
9     26465
10    26703
11    26805
12    26861
13    26694
14    26593
15    26735
16    26483
17    26561
18    26384
19    26707
20    26646
21    26514
22    26709
23    14661
dtype: int64
date_hour
0     25579
1     25626
2     25176
3     25584
4     25513
5     25469
6     25745
7     25558
8     25261
9     25530
10    25503
11    25498
12    25636
13    24812
14    24564
15    24577
16    24300
17    24340
18    24475
```

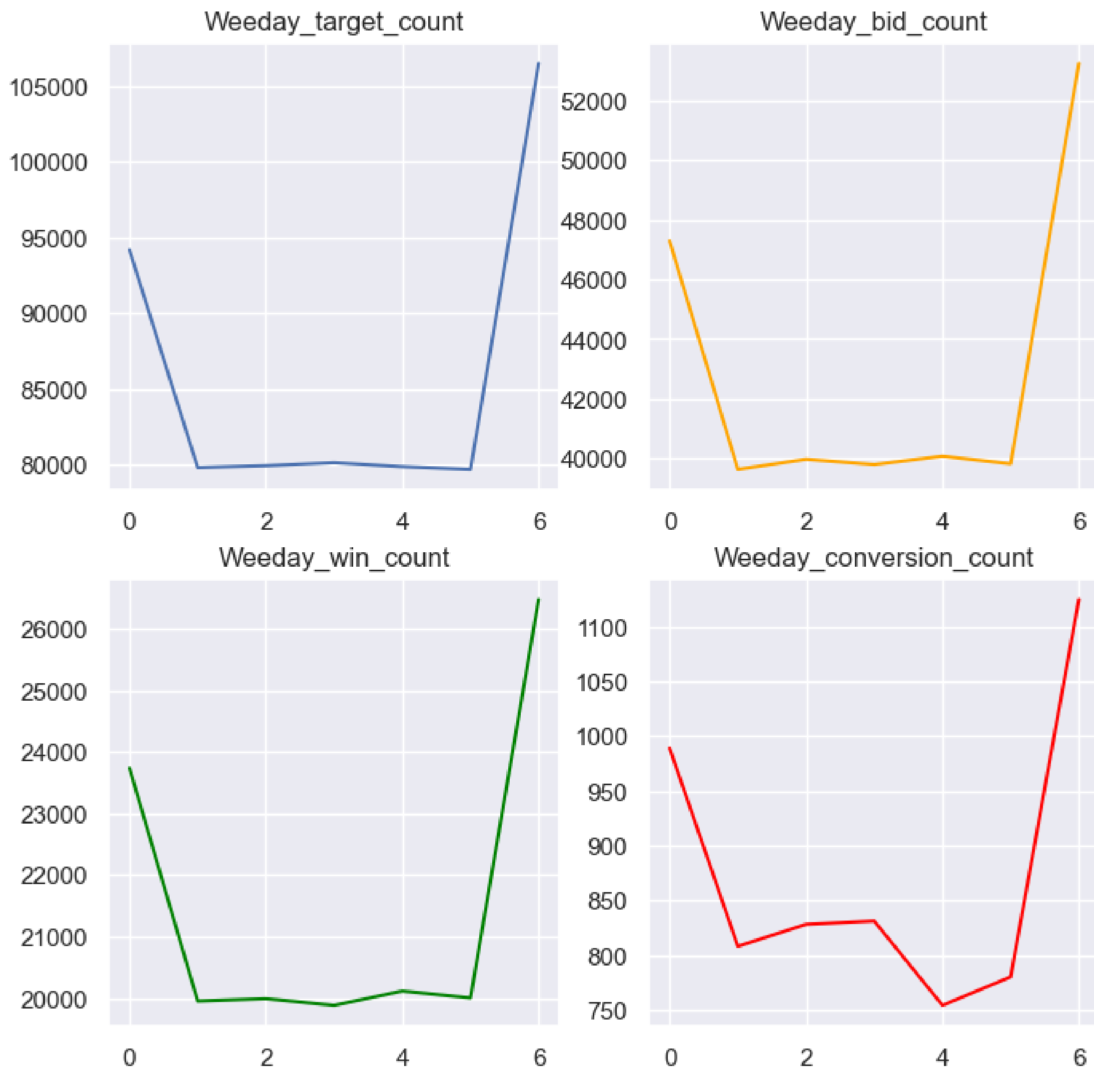
```
19    24264
20    24240
21    24287
22    24083
23    24380
dtype: int64
```

[276]:

```python
# Weekday Analysis
df_date_weekday = df.loc[:, ["date_weekday", "bid", 'win', 'conversion']].
 ↪groupby(["date_weekday"]).agg({'date_weekday': 'count', 'bid': 'sum', 'win':␣
 ↪'sum', 'conversion': 'sum'})

fig, ax = plt.subplots(figsize=(8, 8))
ax1 = plt.subplot(221)
ax1.plot(df_date_weekday.index, df_date_weekday['date_weekday'])
ax1.set_title("Weeday_target_count")

ax2 = plt.subplot(222)
ax2.plot(df_date_weekday.index, df_date_weekday['bid'], c = 'orange')
ax2.set_title("Weeday_bid_count")

ax3 = plt.subplot(223)
ax3.plot(df_date_weekday.index, df_date_weekday['win'], c = 'green')
ax3.set_title("Weeday_win_count")

ax4 = plt.subplot(224)
ax4.plot(df_date_weekday.index, df_date_weekday['conversion'], c = 'red')
ax4.set_title("Weeday_conversion_count")
```

[276]:

```
Text(0.5, 1.0, 'Weeday_conversion_count')
```



[284]:

```
# DayofMonth Analysis
df_date_day = df.loc[:, ["date_day", "bid", 'win', 'conversion']].
 ↪groupby(["date_day"]).agg({'date_day': 'count', 'bid': 'sum','win': 'sum',␣
 ↪'conversion': 'sum'})

fig, ax = plt.subplots(figsize=(8, 8))
ax1 = plt.subplot(221)
ax1.plot(df_date_day.index, df_date_day['date_day'])
ax1.set_ylim([26300, 26900])
ax1.set_title("Day_target_count")

ax2 = plt.subplot(222)
ax2.plot(df_date_day.index, df_date_day['bid'], c = 'orange')
```

```
ax2.set_ylim([13000, 13600])
ax2.set_title("Day_bid_count")

ax3 = plt.subplot(223)
ax3.plot(df_date_day.index, df_date_day['win'], c = 'green')
ax3.set_ylim([6450, 6900])
ax3.set_title("Day_win_count")

ax4 = plt.subplot(224)
ax4.plot(df_date_day.index, df_date_day['conversion'], c = 'red')
ax4.set_ylim([220, 310])
ax4.set_title("Day_conversion_count")
```

[284]: Text(0.5, 1.0, 'Day_conversion_count')

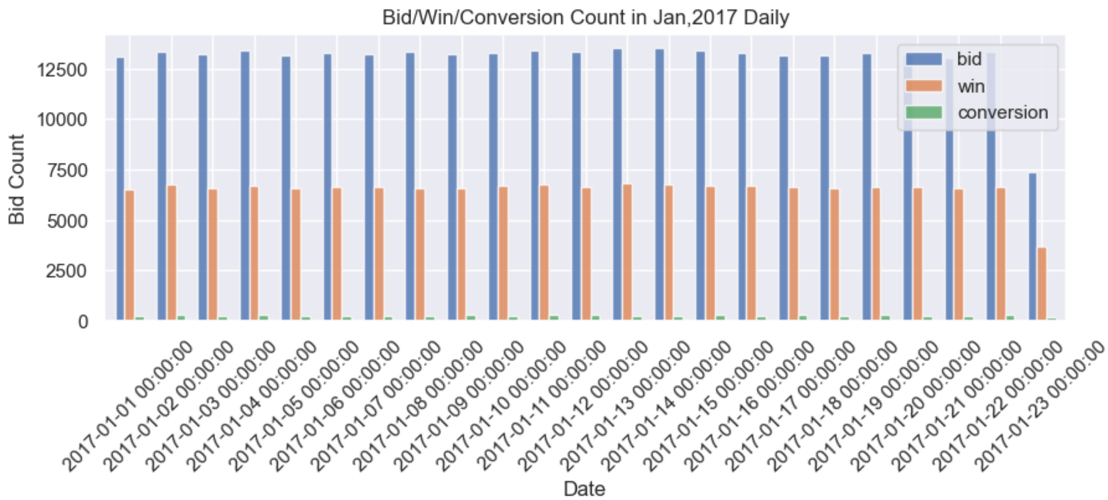
[285]:
```python
# HourofDay Analysis
df_date_hour = df.loc[:, ["date_hour", "bid", 'win', 'conversion']].
 ↪groupby(["date_hour"]).agg({'date_hour': 'count', 'bid': 'sum', 'win': 'sum',␣
 ↪'conversion': 'sum'})

fig, ax = plt.subplots(figsize=(8, 8))
ax1 = plt.subplot(221)
ax1.plot(df_date_hour.index, df_date_hour['date_hour'])
# ax1.set_ylim([26300, 27000])
ax1.set_title("Hour_target_count")

ax2 = plt.subplot(222)
ax2.plot(df_date_hour.index, df_date_hour['bid'], c = 'orange')
# ax2.set_ylim([13000, 13600])
ax2.set_title("Hour_bid_count")

ax3 = plt.subplot(223)
ax3.plot(df_date_hour.index, df_date_hour['win'], c = 'green')
# ax3.set_ylim([220, 310])
ax3.set_title("Hour_win_count")

ax4 = plt.subplot(224)
ax4.plot(df_date_hour.index, df_date_hour['conversion'], c = 'red')
# ax3.set_ylim([220, 310])
ax4.set_title("Hour_conversion_count")
```

[285]:
```
Text(0.5, 1.0, 'Hour_conversion_count')
```

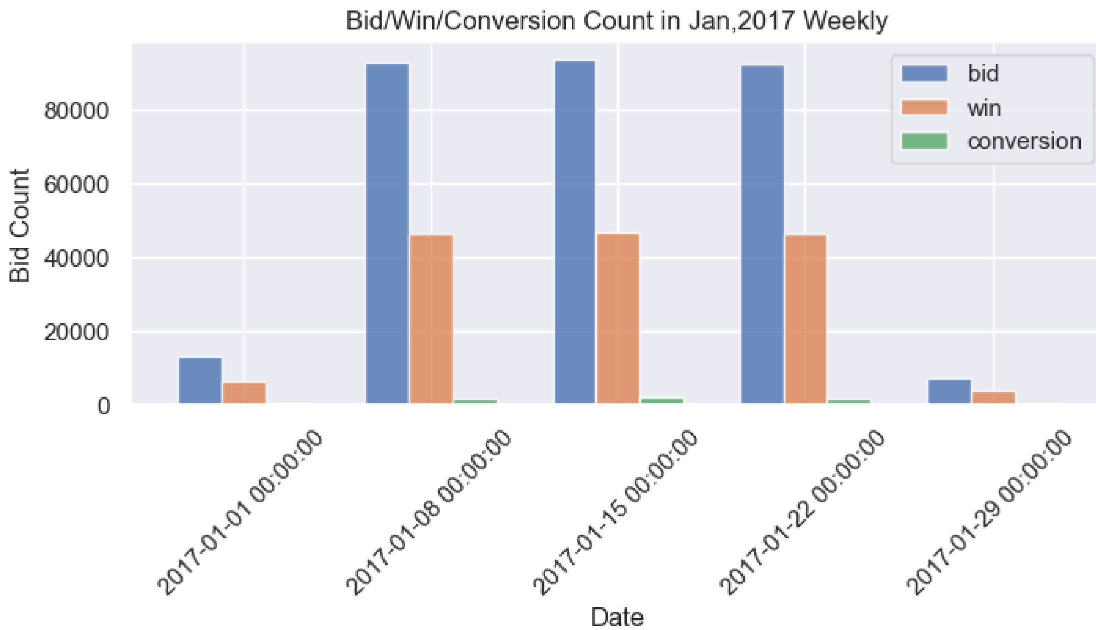
[515]:

```
df0 = df.resample('D', on='timestamp').sum()
df0 = df0.reset_index()

# Create figure and plot space
fig, ax = plt.subplots(figsize=(10, 3))

# Add x-axis and y-axis
ax = df0.plot('timestamp', ["bid", 'win', 'conversion'], kind="bar", ax = ax,␣
↪width=0.7, alpha = 0.8)

# Set title and labels for axes
ax.set(xlabel="Date",
       ylabel="Bid Count",
```

```
       title="Bid/Win/Conversion Count in Jan,2017 Daily")

# Rotate tick marks on x-axis
plt.setp(ax.get_xticklabels(), rotation=45)

plt.show()
```



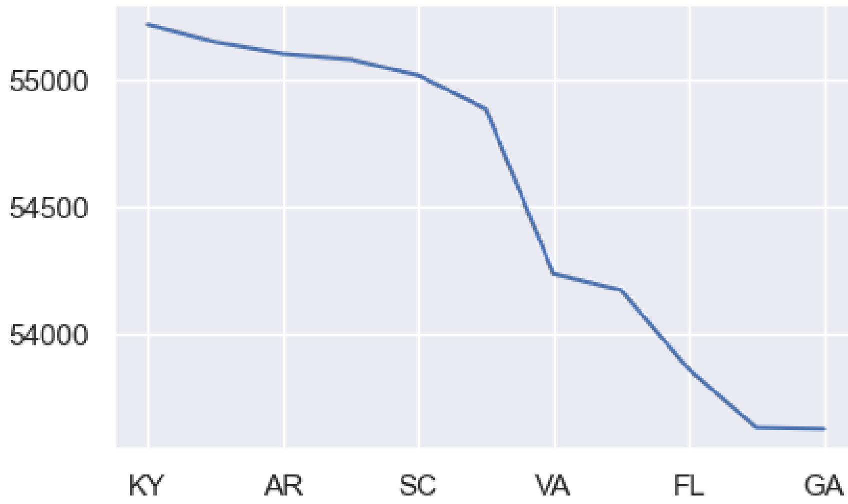
[348]:
```
df1 = df.resample('W', on='timestamp').sum()
df1 = df1.reset_index()

# Create figure and plot space
fig, ax = plt.subplots(figsize=(8, 3))

# Add x-axis and y-axis
ax = df1.plot('timestamp', ["bid", 'win', 'conversion'], kind="bar", ax = ax,␣
 ↪width=0.7, alpha = 0.8)
# ax.bar(df1.index.values, df1["bid"], color='blue', width=2.8)

# Set title and labels for axes
ax.set(xlabel="Date",
       ylabel="Bid Count",
       title="Bid/Win/Conversion Count in Jan,2017 Weekly")

# Rotate tick marks on x-axis
plt.setp(ax.get_xticklabels(), rotation=45)

plt.show()
```



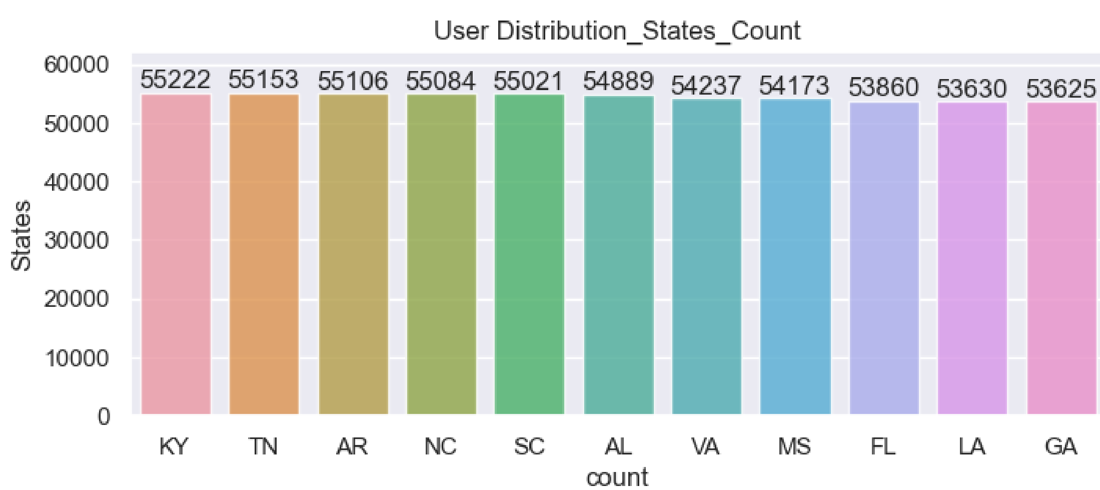
Section 2.1
Summary 1: On weekly basis, the counts of taget users/bid/win/conversion, every Monday and Sunday are showing the highest rate.
Summary 2: On daily basis, the counts of taget users/bid, in the middle of the month between 10~15th, are showing the highest rate.
Summary 3: On hourly basis, the counts of taget users/bid/win/conversion, every day in the monring before 12pm, are showing the highest rate. Only the conversion rate are slightly higher between 13:00 ~ 19:00 in the afternoon/evening.

2.2 EDA - Geographic Analysis

[143]:
```
fig.set_size_inches(5, 3)
df["location"].value_counts().plot()
plt.show()
```



[293]:
```
state_count = df["location"].value_counts()
plt.figure(figsize=(8,3))
g = sns.barplot(state_count.index, state_count.values, alpha=0.8)
for p in g.patches:
    g.annotate("%.0f" % p.get_height(), (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
    ha='center', va='center', xytext=(0, 5), textcoords='offset points') ␣
 ↪#vertical bars
plt.title('User Distribution_States_Count')
plt.ylabel('States', fontsize=12)
plt.xlabel('count', fontsize=12)
g.set_ylim(0, 62000)
plt.show()
```



[16]:
```python
# Code Block
def stateNames(stateAbbreviation):
    states = {
            'AK': 'Alaska',
            'AL': 'Alabama',
            'AR': 'Arkansas',
            'AS': 'American Samoa',
            'AZ': 'Arizona',
            'CA': 'California',
            'CO': 'Colorado',
            'CT': 'Connecticut',
            'DC': 'District of Columbia',
            'DE': 'Delaware',
            'FL': 'Florida',
            'GA': 'Georgia',
            'GU': 'Guam',
            'HI': 'Hawaii',
            'IA': 'Iowa',
            'ID': 'Idaho',
            'IL': 'Illinois',
            'IN': 'Indiana',
            'KS': 'Kansas',
            'KY': 'Kentucky',
            'LA': 'Louisiana',
            'MA': 'Massachusetts',
            'MD': 'Maryland',
            'ME': 'Maine',
            'MI': 'Michigan',
            'MN': 'Minnesota',
            'MO': 'Missouri',
            'MP': 'Northern Mariana Islands',
            'MS': 'Mississippi',
            'MT': 'Montana',
            'NA': 'National',
            'NC': 'North Carolina',
            'ND': 'North Dakota',
            'NE': 'Nebraska',
            'NH': 'New Hampshire',
            'NJ': 'New Jersey',
            'NM': 'New Mexico',
            'NV': 'Nevada',
            'NY': 'New York',
            'OH': 'Ohio',
            'OK': 'Oklahoma',
            'OR': 'Oregon',
```

```
            'PA': 'Pennsylvania',
            'PR': 'Puerto Rico',
            'RI': 'Rhode Island',
            'SC': 'South Carolina',
            'SD': 'South Dakota',
            'TN': 'Tennessee',
            'TX': 'Texas',
            'UT': 'Utah',
            'VA': 'Virginia',
            'VI': 'Virgin Islands',
            'VT': 'Vermont',
            'WA': 'Washington',
            'WI': 'Wisconsin',
            'WV': 'West Virginia',
            'WY': 'Wyoming'
    }
    if stateAbbreviation is not None:
        if stateAbbreviation in states:
            return states[stateAbbreviation]
        else:
            return None
    else:
        return None

# Convert State Abbreviation
state_list = list(df["location"].value_counts().index)
state_list = [stateNames(state) for state in state_list]
```

[123]:
```
import matplotlib.pyplot as plt
# import geopandas
import math
#import os

import os
import conda
#os.environ["PROJ_LIB"] = r'C:\ProgramData\Anaconda3\Library\share\proj'

conda_file_dir = conda.__file__
conda_dir = conda_file_dir.split('lib')[0]
proj_lib = os.path.join(os.path.join(os.path.join(conda_dir, 'Library'),␣
 ↪'share'), 'proj')

os.environ["PROJ_LIB"] = proj_lib

from mpl_toolkits.basemap import Basemap
from geopy.geocoders import Nominatim
```

```
states = [['Kentucky',55222],
          ['Tennessee',55153],
          ['Arkansas',55106],
          ['North Carolina',55084],
          ['South Carolina',55021],
          ['Alabama',54889],
          ['Virginia',54237],
          ['Mississippi',54173],
          ['Florida',53860],
          ['Louisiana',53630],
          ['Georgia',53625]]

map = Basemap(llcrnrlon=-119,llcrnrlat=22,urcrnrlon=-64,urcrnrlat=49,
        projection='lcc',lat_1=32,lat_2=45,lon_0=-95)

# load the shapefile, use the name 'states'
plt.figure(figsize=(8,3))
map.readshapefile('st99_d00', name='states', drawbounds=True)

# Get the location of each city and plot it
geolocator = Nominatim(user_agent="http")
for (state,count) in states:
    loc = geolocator.geocode(state)
    x, y = map(loc.longitude, loc.latitude)
    map.plot(x,y,marker='o',color='Red',markersize=int((count-50000)/500))
plt.show()
```

[18]:
```
# Relationship User_count with State Population
states_url = 'https://simple.wikipedia.org/wiki/List_of_U.S._states'
usa_states = pd.read_html(states_url)
```

```
usa_states = usa_states[0]
usa_states.head()
```

[18]:

```
  Name &postal abbs. [1]                          Unnamed: 2_level_0  \
  Name &postal abbs. [1] Name &postal abbs. [1].1 Unnamed: 2_level_1
0                Alabama                      NaN                 AL
1                 Alaska                      NaN                 AK
2                Arizona                      NaN                 AZ
3               Arkansas                      NaN                 AR
4             California                      NaN                 CA

        Cities                              Established[A] Population[B][3]  \
       Capital Largest (by population)[5] Established[A] Population[B][3]
0   Montgomery                  Birmingham   Dec 14, 1819          4903185
1       Juneau                   Anchorage    Jan 3, 1959           731545
2      Phoenix                     Phoenix   Feb 14, 1912          7278717
3  Little Rock                 Little Rock   Jun 15, 1836          3017804
4   Sacramento                 Los Angeles    Sep 9, 1850         39512223

  Total area[4]          Land area[4]          Water area[4]          \
            mi2      km2          mi2      km2           mi2     km2
0         52420   135767        50645   131171          1775    4597
1        665384  1723337       570641  1477953         94743  245384
2        113990   295234       113594   294207           396    1026
3         53179   137732        52035   134771          1143    2961
4        163695   423967       155779   403466          7916   20501

  Numberof Reps.
  Numberof Reps.
0              7
1              1
2              9
3              4
4             53
```

[19]:

```
usa_states_sub = usa_states.iloc[:, [0,6]].copy()
usa_states_sub.columns = ['state', 'state_population']
# usa_states_sub['state'] = usa_states_sub['state'].str.replace('[C]','',␣
 ↪regex=True)
usa_states_sub.head()
```

[19]:

```
        state  state_population
0     Alabama           4903185
1      Alaska            731545
2     Arizona           7278717
3    Arkansas           3017804
4  California          39512223
```

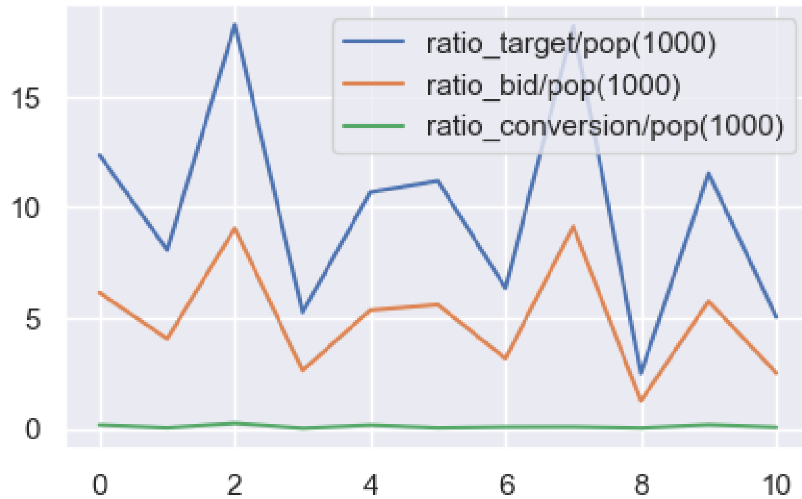
[62]:
```python
import warnings
warnings.filterwarnings('ignore')

df_location_pop = pd.DataFrame(df["location"].value_counts())
df_location_pop.columns = ['count']
state_list = list(df["location"].value_counts().index)
df_location_pop['state'] = [stateNames(state) for state in state_list]
df_location_pop = df_location_pop.reset_index()
df_location_pop = df_location_pop.merge(usa_states_sub, on = 'state', how =␣
↪'left')
df_location_pop.loc[df_location_pop['state'] == 'Kentucky', 'state_population']␣
↪= 4467673
df_location_pop.loc[df_location_pop['state'] == 'Virginia', 'state_population']␣
↪= 8535519
df_location_pop['state_population'] = df_location_pop['state_population'].
↪astype(int)
df_location_pop['ratio_target/pop(1000)'] = df_location_pop['count']/
↪df_location_pop['state_population'] * 1000

state_count_request = pd.DataFrame(df["location"][df['bid'] == 1].value_counts())
state_count_request = state_count_request.reset_index()
state_count_request.columns = ['index', 'bid_count']
state_count_conversion = pd.DataFrame(df["location"][df['conversion'] == 1].
↪value_counts())
state_count_conversion = state_count_conversion.reset_index()
state_count_conversion.columns = ['index', 'conversion_count']

df_location_pop = df_location_pop.merge(state_count_request, on = 'index', how =␣
↪'left')
df_location_pop = df_location_pop.merge(state_count_conversion, on = 'index',␣
↪how = 'left')
df_location_pop = df_location_pop[['index', 'count', 'bid_count',␣
↪'conversion_count', 'state', 'state_population',
       'ratio_target/pop(1000)']]
df_location_pop['ratio_bid/pop(1000)'] = df_location_pop['bid_count']/
↪df_location_pop['state_population'] * 1000
df_location_pop['ratio_conversion/pop(1000)'] =␣
↪df_location_pop['conversion_count']/df_location_pop['state_population'] * 1000
df_location_pop
```

[62]:
```
  index  count  bid_count  conversion_count           state  \
0    KY  55222      27433               755        Kentucky
1    TN  55153      27741               274       Tennessee
2    AR  55106      27339               756        Arkansas
3    NC  55084      27540               193  North Carolina
4    SC  55021      27584               830  South Carolina
```

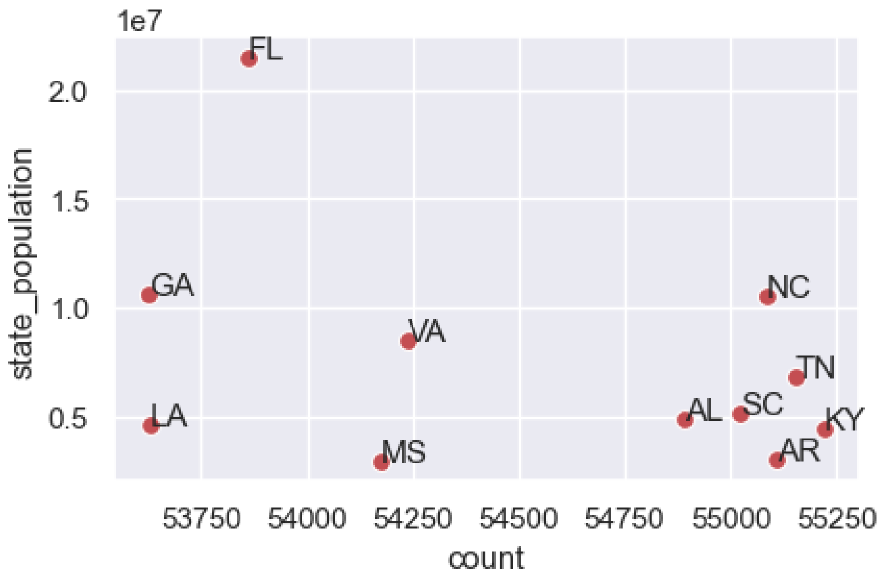
```
5     AL  54889     27476                184        Alabama
6     VA  54237     27067                671       Virginia
7     MS  54173     27202                252    Mississippi
8     FL  53860     26941                682        Florida
9     LA  53630     26779                830      Louisiana
10    GA  53625     26739                688        Georgia

    state_population  ratio_target/pop(1000)  ratio_bid/pop(1000)  \
0            4467673               12.360350             6.140333
1            6829174                8.076087             4.062131
2            3017804               18.260298             9.059236
3           10488084                5.252056             2.625837
4            5148714               10.686358             5.357454
5            4903185               11.194560             5.603705
6            8535519                6.354271             3.171102
7            2976149               18.202382             9.139999
8           21477737                2.507713             1.254369
9            4648794               11.536325             5.760419
10          10617423                5.050661             2.518408

    ratio_conversion/pop(1000)
0                     0.168992
1                     0.040122
2                     0.250513
3                     0.018402
4                     0.161205
5                     0.037527
6                     0.078613
7                     0.084673
8                     0.031754
9                     0.178541
10                    0.064799
```

[341]:
```
df_location_pop[['index','ratio_target/pop(1000)', 'ratio_bid/pop(1000)',␣
↪'ratio_conversion/pop(1000)']].plot()
```

[341]: <matplotlib.axes._subplots.AxesSubplot at 0x19e1b214be0>



[347]:
```
# Plot Scatter Plot between User Count and State Population
sns.set()
sns.scatterplot(data=df_location_pop, x="count", y="state_population", s=50,␣
 ↪color = 'r')
plt.grid()
def label_point(x, y, val, ax):
    a = pd.concat({'x': x, 'y': y, 'val': val}, axis=1)
    for i, point in a.iterrows():
        ax.text(point['x']+.05, point['y'], str(point['val']))

label_point(df_location_pop["count"], df_location_pop["state_population"],␣
 ↪df_location_pop["index"], plt.gca())
plt.grid()
```

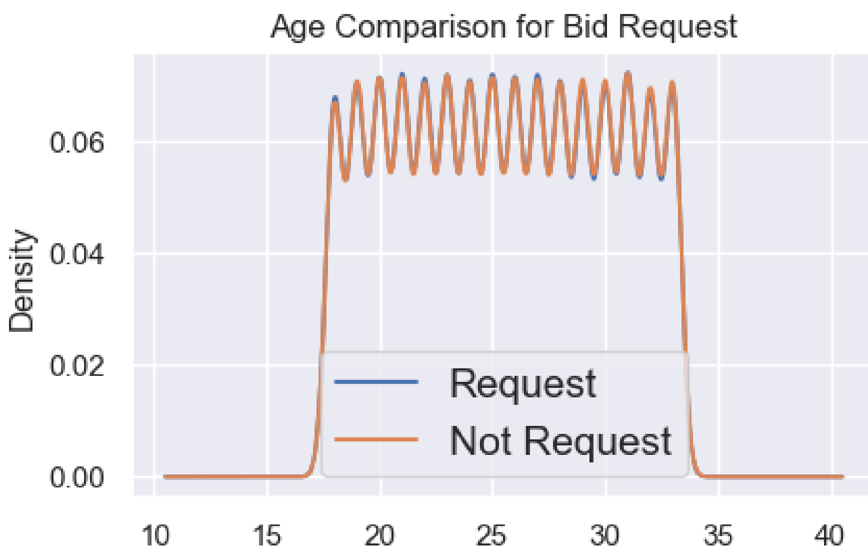


Section 2.2 - Summary: The ratio of targeted people/bid/conversion per 1000 population of Arkansas is the highest while Florida has the lowest ratios.

2.3 Further EDA - General Distribution Analysis for Targeted Users

2.3.1 Age Comparison

[751]:
```
# Visualization Age Distribution
plt.figure(figsize=(5,3))
df_request = df[df['bid'] == 1]
plt.title('Age Comparison for Bid Request')
# plt.axis([0, 1, 0, 4.5])
df['age'][df['bid'] == 1].plot.kde(label='Request')
df['age'][df['bid'] == 0].plot.kde(label='Not Request')
plt.legend(prop={'size': 15})
plt.grid()
plt.show()
```

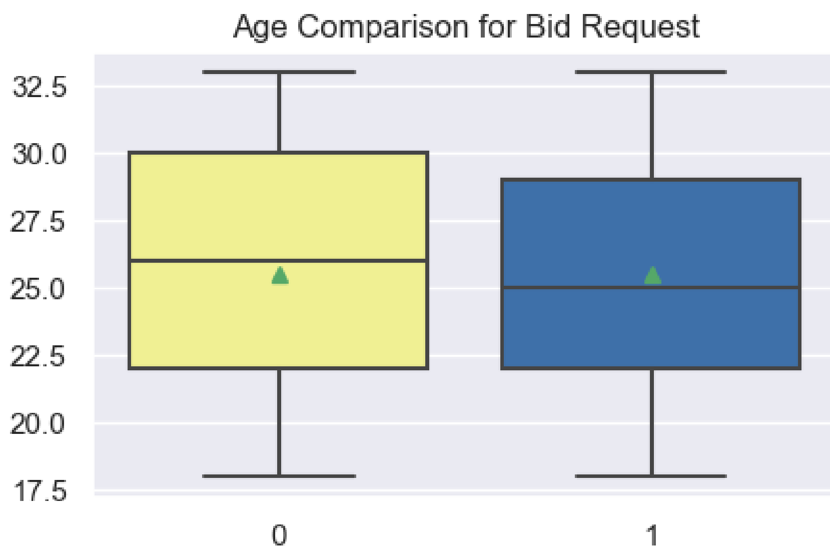


[389]:
```
print("Average Age for Users Not Request:", df['age'][df['bid'] == 0].mean())
print("Average Age for Users Request:", df['age'][df['bid'] == 1].mean())
```

```
Average Age for Users Not Request: 25.519138190092583
Average Age for Users Request: 25.494838931300254
```

[351]:
```
# Age Comparison
# sns.axlabel(xlabel=df['bid'], ylabel="Age", fontsize=16)
sns.boxplot(data=[df['age'][df['bid'] == 0], df['age'][df['bid'] == 1]],
    palette=[sns.xkcd_rgb["pale yellow"], sns.xkcd_rgb["medium blue"]],
    showmeans=True)
# sns.swarmplot(data=[df['age'][df['bid'] == 0], df['age'][df['bid'] == 1]],␣
↪size=6, edgecolor="black", linewidth=.9)
plt.title("Age Comparison for Bid Request")
```

[351]: Text(0.5, 1.0, 'Age Comparison for Bid Request')



For Question 1:
Summary 1: The users at younger age group, has higher chance to open websites.

2.3.2 Gender Comparison

[363]:
```
gender_count_nrequest = df['gender'][df['bid'] == 0].value_counts()
gender_count_request = df['gender'][df['bid'] == 1].value_counts()

# Investigation on Gender Comparison Not Request & Request
fig, axes = plt.subplots(1, 2, figsize=(10, 3), sharey=True)

# Investigation on Gender for Not Request
g = gender_count_nrequest.plot(ax=axes[0], kind='bar', title='Gender␣
↪Proportion_Not Request', alpha = 0.5)
for p in g.patches:
    g.annotate("%.0f" % p.get_height(), (p.get_x() + p.get_width() / 2., p.
↪get_height()),
    ha='center', va='center', xytext=(0, 10), textcoords='offset points') ␣
↪#vertical bars
g.set_ylim(0, 165000)

# Investigation on Gender for Request
g = gender_count_request.plot(ax=axes[1], kind='bar', title='Gender␣
↪Proportion_Request', color = 'g', alpha = 0.5)

for p in g.patches:
    g.annotate("%.0f" % p.get_height(), (p.get_x() + p.get_width() / 2., p.
↪get_height()),
```

```
    ha='center', va='center', xytext=(0, 10), textcoords='offset points') ␣
 ↪#vertical bars
g.set_ylim(0, 165000)
```

[363]: (0.0, 165000.0)

[364]:
```
gender_count_nrequest = pd.DataFrame(gender_count_nrequest)
gender_count_nrequest.columns = ['Count_Not Request']
gender_count_request = pd.DataFrame(gender_count_request)
gender_count_request.columns = ['Count_Request']
gender_count_combine = gender_count_nrequest.join(gender_count_request)
gender_count_combine['Request_Percent'] = gender_count_combine['Count_Request']/␣
 ↪\
                                          (gender_count_combine['Count_Request']+ \
                                           gender_count_combine['Count_Not␣
 ↪Request'])
gender_count_combine.sort_values('Request_Percent', ascending = False)
```

[364]:

	Count_Not Request	Count_Request	Request_Percent
M	149816	150310	0.500823
F	150343	149531	0.498646

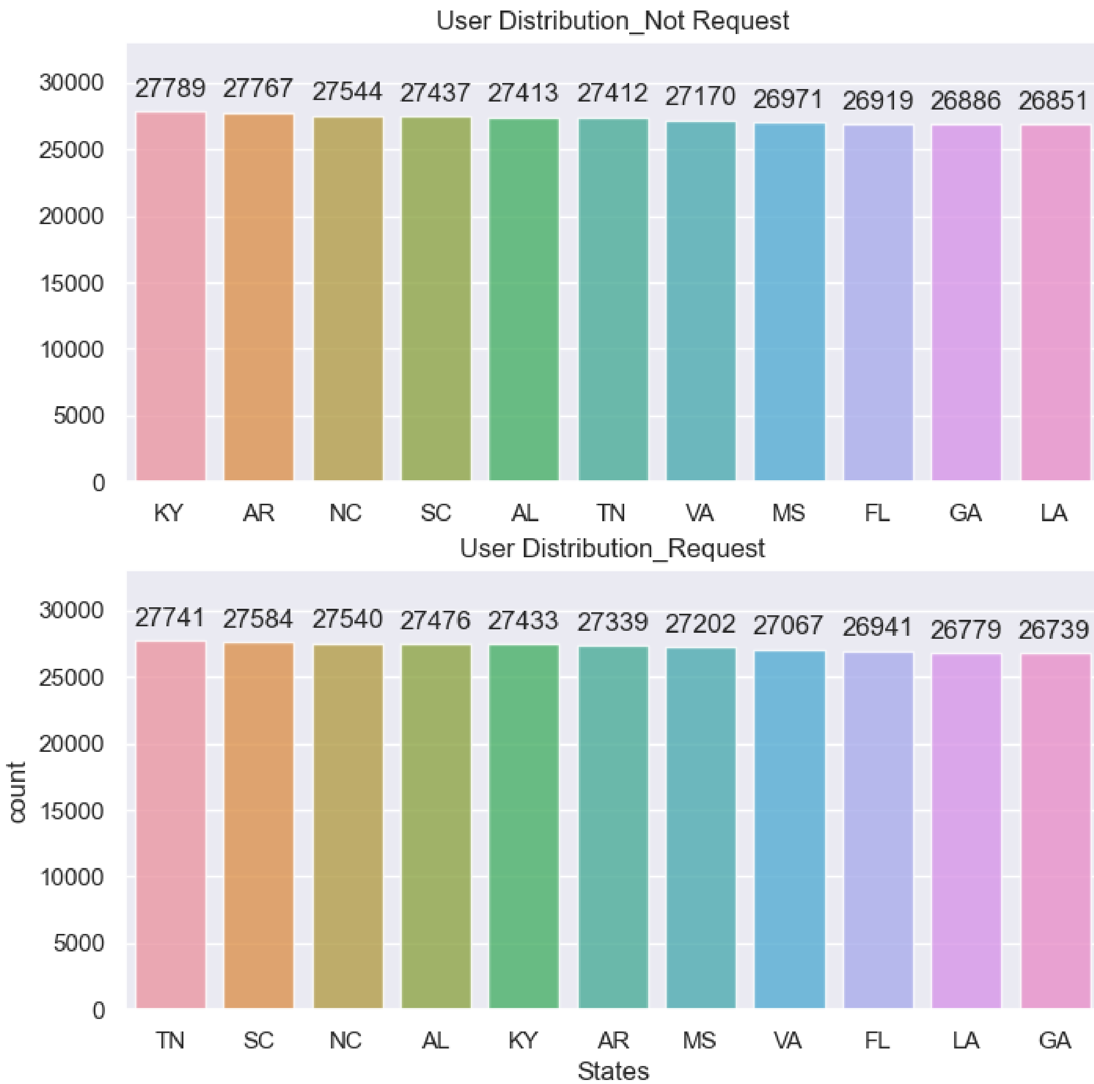
For Question 1:
Summary 2: There is no distinct preference differentiated from genders in targeted group to open websites.
Summary 3: The female group is slightly higher chance to open websites than male group.

2.3.3 Location Comparison

[149]:
```
# Location Analysis for Users Comparison Not Request & Request
fig, axes = plt.subplots(2, 1, figsize=(8, 8), sharey=True)

# Location Analysis for Users Not Request
state_count_nrequest = df["location"][df['bid'] == 0].value_counts()
```

```
g = sns.barplot(ax=axes[0], x = state_count_nrequest.index, y =␣
 ↪state_count_nrequest.values, alpha=0.8)
axes[0].set_title('User Distribution_Not Request')
plt.xlabel('States', fontsize=12)
plt.ylabel('count', fontsize=12)
for p in g.patches:
    g.annotate("%.0f" % p.get_height(), (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
    ha='center', va='center', xytext=(0, 10), textcoords='offset points') ␣
 ↪#vertical bars
g.set_ylim(0, 33000)

# Location Analysis for Users Request
state_count_request = df["location"][df['bid'] == 1].value_counts()
g = sns.barplot(ax=axes[1], x = state_count_request.index, y =␣
 ↪state_count_request.values, alpha=0.8)
axes[1].set_title('User Distribution_Request')
plt.xlabel('States', fontsize=12)
plt.ylabel('count', fontsize=12)
for p in g.patches:
    g.annotate("%.0f" % p.get_height(), (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
    ha='center', va='center', xytext=(0, 10), textcoords='offset points') ␣
 ↪#vertical bars
g.set_ylim(0, 33000)
```

```
[149]: (0.0, 33000.0)
```



[359]:

```
df_state_count_nrequest = pd.DataFrame(state_count_nrequest)
df_state_count_nrequest.columns = ['Count_Not Request']
df_state_count_request = pd.DataFrame(state_count_request)
df_state_count_request.columns = ['Count_Request']
df_state_count_combine = df_state_count_nrequest.join(df_state_count_request)
df_state_count_combine['Request_Percent'] = df_state_count_combine['Count_Request']/ \
    (df_state_count_combine['Count_Request']+ \
                                         df_state_count_combine['Count_Not Request'])
df_state_count_combine.sort_values('Request_Percent', ascending = False)
```

[359]:

	Count_Not Request	Count_Request	Request_Percent
TN	27412	27741	0.502983
MS	26971	27202	0.502132
SC	27437	27584	0.501336
AL	27413	27476	0.500574
FL	26919	26941	0.500204
NC	27544	27540	0.499964
LA	26851	26779	0.499329
VA	27170	27067	0.499050
GA	26886	26739	0.498629
KY	27789	27433	0.496777
AR	27767	27339	0.496117

For Question 1:
Summary 4: There is no distinct difference from states in targeted group to open websites.
Summary 5: Target Users located in Tennessee state is slightly higher chance to open websites, while users in Arizona is lower chance.

2.3.4 Bid Request Ratio - Overall

[294]:
```
# Chance to send bid request
df[df['bid'] == 1].shape[0]/df.shape[0]
```

[294]: 0.499735

For Question 1:
Summary 6: Overall 49% of the targeted group is to open websites.

2.4 Further EDA - Convert Effectiveness Analysis - Control/Test Groups

[753]:
```
df_location_0 = pd.DataFrame(df[df['test'] == 0].groupby("location")['test'].
 ↪count())
df_location_0.columns = ["test_0"]
df_location_1 = pd.DataFrame(df[df['test'] == 1].groupby("location")['test'].
 ↪count())
df_location_1.columns = ["test_1"]
df_location = df_location_0.join(df_location_1)
df_location = df_location.reindex(index=df["location"].value_counts().index)
df_location.head()
```

[753]:

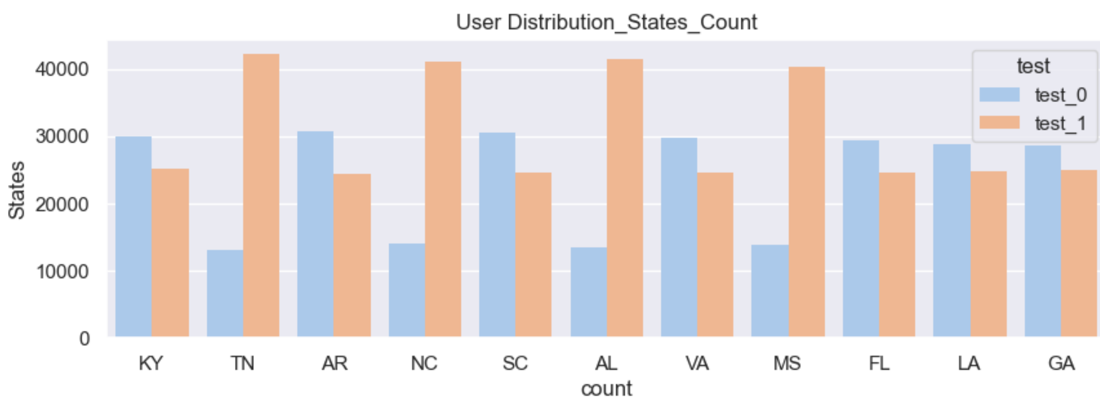
	test_0	test_1
KY	30010	25212
TN	12994	42159
AR	30787	24319
NC	14118	40966
SC	30446	24575

[754]:
```
df_location = df_location.reset_index()
df_location_melt = pd.melt(df_location, id_vars="index", var_name="test",␣
↪value_name="count")
df_location_melt.head()
```

[754]:

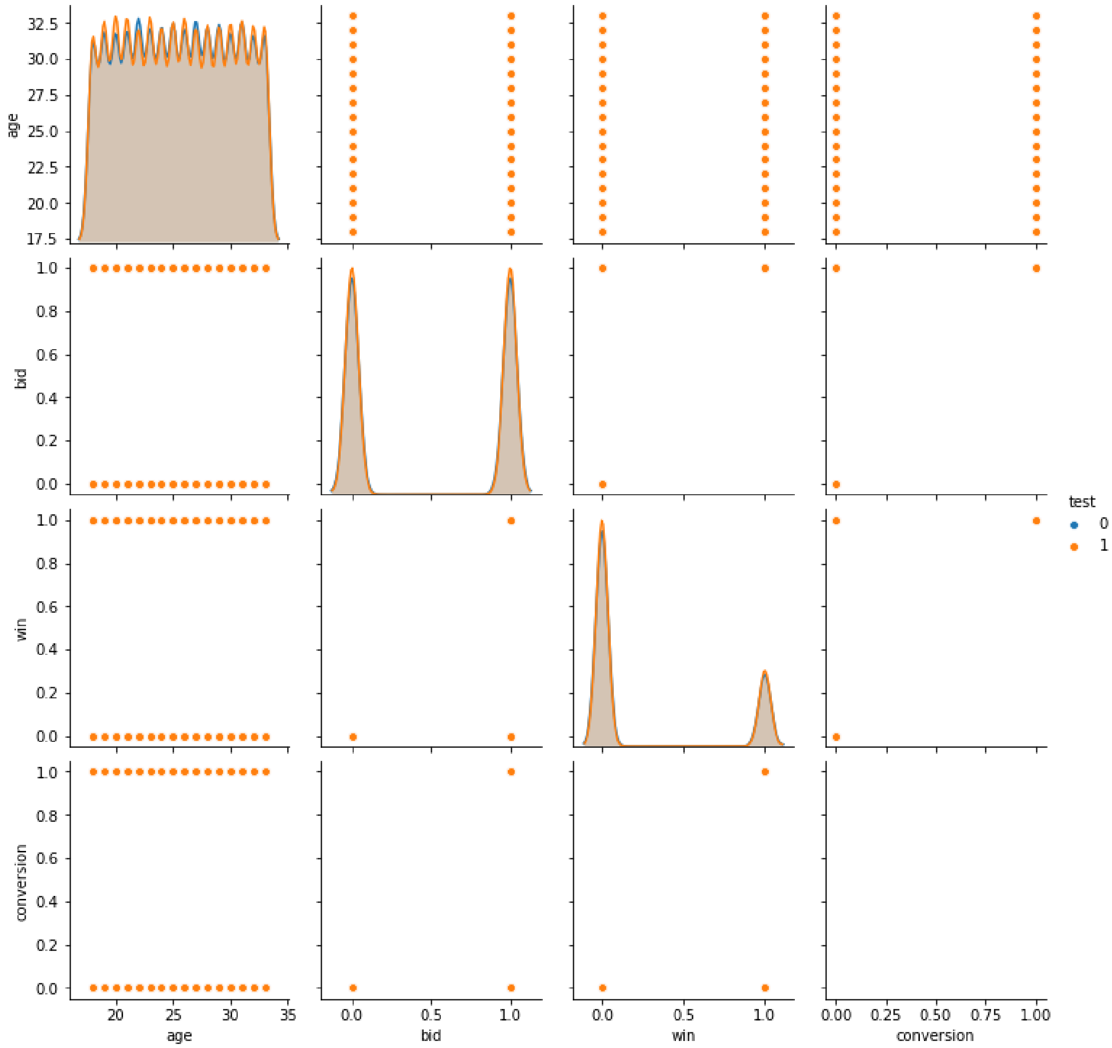
	index	test	count
0	KY	test_0	30010
1	TN	test_0	12994
2	AR	test_0	30787
3	NC	test_0	14118
4	SC	test_0	30446

[755]:
```
plt.figure(figsize=(10,3))
sns.barplot(df_location_melt["index"], df_location_melt["count"].values,
            hue = df_location_melt["test"], alpha=1, palette= "pastel")
plt.title('User Distribution_States_Count')
plt.ylabel('States', fontsize=12)
plt.xlabel('count', fontsize=12)
plt.show()
```



[206]:
```
# Normalized and visulization
# Normalization metric1 ~ metric9
import warnings
warnings.filterwarnings('ignore')

sns.pairplot(df, hue='test', height=2.5)
```

[206]: <seaborn.axisgrid.PairGrid at 0x23cd4a8bee0>



[756]:
```
# Pivot Table to Seperate on Test Group 0, 1
df_pivot = pd.pivot_table(df, values=['conversion', 'bid', 'win'],␣
↪index='user_id', columns='test', aggfunc={'conversion': np.sum,
                                'user_id': 'count', 'bid': np.sum, 'win': np.sum}).
↪sort_values(by=[('conversion', 1), ('user_id', 1)], ascending=False)
df_pivot.columns = ['sum_bid_test0', 'sum_bid_test1', 'sum_conversion_test0',␣
↪'sum_conversion_test1',
                      'count_user_id_test0', 'count_user_id_test1',␣
↪'sum_win_test0', 'sum_win_test1']
print(df_pivot.head())
```

```
                                  sum_bid_test0  sum_bid_test1  \
user_id
54f07ff91a4441b2de263b955ab8a6a0            NaN           18.0
5206920a9e23ad2413f349c1390b748e            NaN           20.0
```

```
7879f860900cb332158c7d086275b70d          NaN          18.0
d6cd1caa0374b57ed756e93e72d9b496          NaN          13.0
8b542d5e68088d7345fc31d92c8013db          NaN          11.0

                                  sum_conversion_test0  sum_conversion_test1  \
user_id
54f07ff91a4441b2de263b955ab8a6a0                   NaN                   5.0
5206920a9e23ad2413f349c1390b748e                   NaN                   3.0
7879f860900cb332158c7d086275b70d                   NaN                   3.0
d6cd1caa0374b57ed756e93e72d9b496                   NaN                   3.0
8b542d5e68088d7345fc31d92c8013db                   NaN                   3.0

                                  count_user_id_test0  count_user_id_test1  \
user_id
54f07ff91a4441b2de263b955ab8a6a0                  NaN                 27.0
5206920a9e23ad2413f349c1390b748e                  NaN                 32.0
7879f860900cb332158c7d086275b70d                  NaN                 30.0
d6cd1caa0374b57ed756e93e72d9b496                  NaN                 28.0
8b542d5e68088d7345fc31d92c8013db                  NaN                 26.0

                                  sum_win_test0  sum_win_test1
user_id
54f07ff91a4441b2de263b955ab8a6a0            NaN           15.0
5206920a9e23ad2413f349c1390b748e            NaN           12.0
7879f860900cb332158c7d086275b70d            NaN           10.0
d6cd1caa0374b57ed756e93e72d9b496            NaN           10.0
8b542d5e68088d7345fc31d92c8013db            NaN           10.0
```

[757]:

```
# Pivot Table for Test 0
df_pivot_control = df_pivot[['sum_bid_test0', 'sum_win_test0',␣
 ↪'sum_conversion_test0', 'count_user_id_test0']]. \
                             dropna().sort_values(by='sum_conversion_test0',␣
 ↪ascending=False)
df_pivot_control['ratio_c/b'] = (df_pivot_control['sum_conversion_test0']/
 ↪df_pivot_control['sum_bid_test0']).round(decimals=2)
df_pivot_control['ratio_w/b'] = (df_pivot_control['sum_win_test0']/
 ↪df_pivot_control['sum_bid_test0']).round(decimals=2)
print("df_pivot_control", df_pivot_control.head())
print(df_pivot_control.describe())

#Pivot Table for Test 1
df_pivot_test = df_pivot[['sum_bid_test1', 'sum_win_test1',␣
 ↪'sum_conversion_test1', 'count_user_id_test1']]. \
                             dropna().sort_values(by='sum_conversion_test1',␣
 ↪ascending=False)
df_pivot_test['ratio_c/b'] = (df_pivot_test['sum_conversion_test1']/
 ↪df_pivot_test['sum_bid_test1']).round(decimals=2)
```

```python
df_pivot_test['ratio_w/b'] = (df_pivot_test['sum_win_test1']/
 ↪df_pivot_test['sum_bid_test1']).round(decimals=2)
print("df_pivot_test", df_pivot_test.head())
print(df_pivot_test.describe())
```

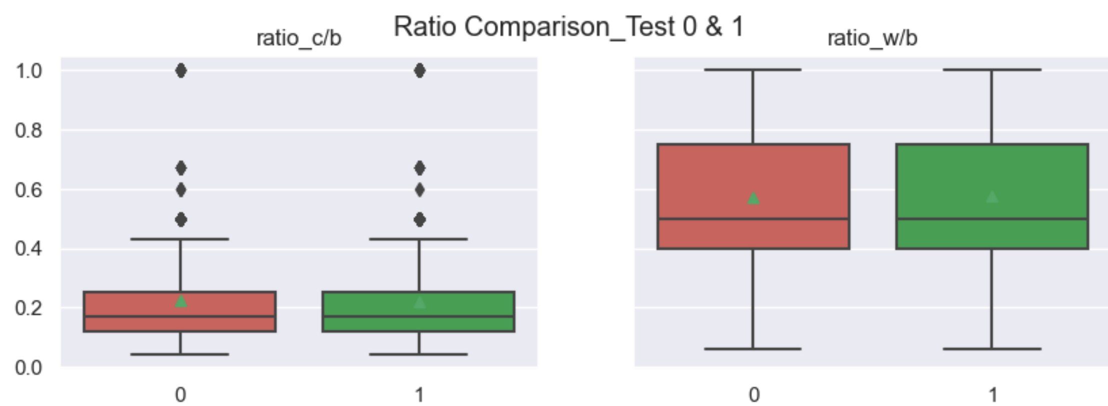
```
df_pivot_control                                   sum_bid_test0  sum_win_test0
\
user_id
e879bd196b9b3e4db974b8716c6f896e           22.0           17.0
9570efef719d705326f0ff817ef084e6           13.0            9.0
bb55408ead5dcb75a28618dbc998a0a9            9.0            7.0
2bc33f317d4f25b10e2a2a55392b11cb           11.0            9.0
797134c3e42371bb4979a462eb2f042a           13.0           10.0

                                  sum_conversion_test0  count_user_id_test0  \
user_id
e879bd196b9b3e4db974b8716c6f896e                   5.0                 32.0
9570efef719d705326f0ff817ef084e6                   4.0                 18.0
bb55408ead5dcb75a28618dbc998a0a9                   3.0                 15.0
2bc33f317d4f25b10e2a2a55392b11cb                   3.0                 25.0
797134c3e42371bb4979a462eb2f042a                   3.0                 30.0

                                  ratio_c/b  ratio_w/b
user_id
e879bd196b9b3e4db974b8716c6f896e       0.23       0.77
9570efef719d705326f0ff817ef084e6       0.31       0.69
bb55408ead5dcb75a28618dbc998a0a9       0.33       0.78
2bc33f317d4f25b10e2a2a55392b11cb       0.27       0.82
797134c3e42371bb4979a462eb2f042a       0.23       0.77
       sum_bid_test0  sum_win_test0  sum_conversion_test0  \
count   27608.000000   27608.000000          27608.000000
mean        4.744820       2.374819              0.098631
std         3.256593       2.025353              0.327467
min         0.000000       0.000000              0.000000
25%         2.000000       1.000000              0.000000
50%         4.000000       2.000000              0.000000
75%         6.000000       3.000000              0.000000
max        28.000000      18.000000              5.000000

       count_user_id_test0     ratio_c/b     ratio_w/b
count         27608.000000  26808.000000  26808.000000
mean              9.493879      0.021056      0.500328
std               5.760776      0.084210      0.295857
min               1.000000      0.000000      0.000000
25%               5.000000      0.000000      0.330000
50%               8.000000      0.000000      0.500000
75%              12.000000      0.000000      0.670000
max              48.000000      1.000000      1.000000
```

```
df_pivot_test                                 sum_bid_test1  sum_win_test1  \
user_id
54f07ff91a4441b2de263b955ab8a6a0           18.0           15.0
03994ecbaa046cb05a924a0a72aa6913           11.0            4.0
5206920a9e23ad2413f349c1390b748e           20.0           12.0
40001eb613ebc80a610670c0187b0153            3.0            3.0
7d40f910bbd18715587677b383d11dbe            7.0            5.0

                                  sum_conversion_test1  count_user_id_test1  \
user_id
54f07ff91a4441b2de263b955ab8a6a0                   5.0                 27.0
03994ecbaa046cb05a924a0a72aa6913                   3.0                 16.0
5206920a9e23ad2413f349c1390b748e                   3.0                 32.0
40001eb613ebc80a610670c0187b0153                   3.0                  6.0
7d40f910bbd18715587677b383d11dbe                   3.0                 10.0

                                  ratio_c/b  ratio_w/b
user_id
54f07ff91a4441b2de263b955ab8a6a0       0.28       0.83
03994ecbaa046cb05a924a0a72aa6913       0.27       0.36
5206920a9e23ad2413f349c1390b748e       0.15       0.60
40001eb613ebc80a610670c0187b0153       1.00       1.00
7d40f910bbd18715587677b383d11dbe       0.43       0.71
       sum_bid_test1  sum_win_test1  sum_conversion_test1  \
count   35529.000000   35529.000000          35529.000000
mean        4.752343       2.381013              0.095471
std         3.254852       2.026724              0.321933
min         0.000000       0.000000              0.000000
25%         2.000000       1.000000              0.000000
50%         4.000000       2.000000              0.000000
75%         6.000000       3.000000              0.000000
max        26.000000      16.000000              5.000000

       count_user_id_test1     ratio_c/b     ratio_w/b
count         35529.000000  34497.000000  34497.000000
mean              9.510344      0.019727      0.501375
std               5.757558      0.079539      0.297046
min               1.000000      0.000000      0.000000
25%               5.000000      0.000000      0.330000
50%               8.000000      0.000000      0.500000
75%              12.000000      0.000000      0.670000
max              50.000000      1.000000      1.000000
```

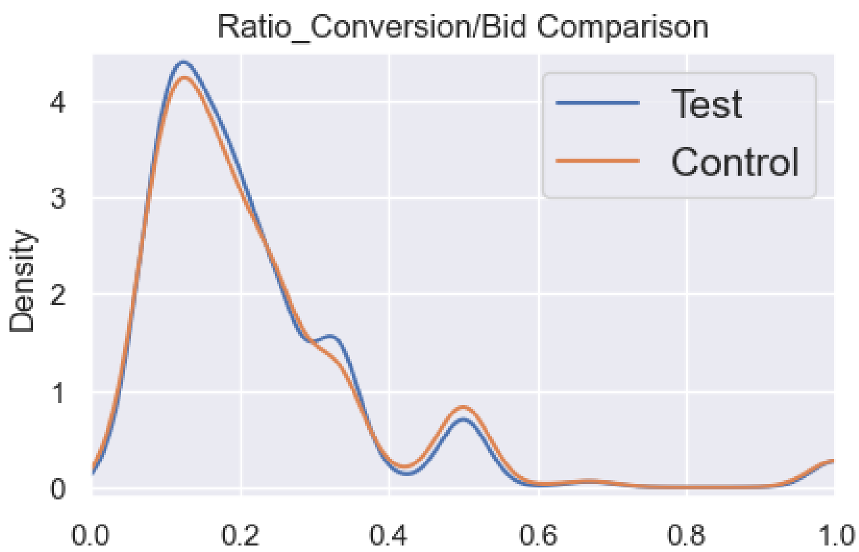
[373]:
```
# Visualization Boxplot Ratio_Conversion/Bid & Ratio_Win/Bid
fig, axes = plt.subplots(1, 2, figsize=(10, 3), sharey=True)
fig.suptitle('Ratio Comparison_Test 0 & 1')
```

```
sns.boxplot(ax=axes[0],
    data=[df_pivot_control[df_pivot_control["ratio_c/b"]>0.01]["ratio_c/b"],␣
 ↪df_pivot_test[df_pivot_test["ratio_c/b"]>0.01]["ratio_c/b"]],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
axes[0].set_title("ratio_c/b")

sns.boxplot(ax=axes[1],
    data=[df_pivot_control[df_pivot_control["ratio_w/b"]>0.01]["ratio_w/b"],␣
 ↪df_pivot_test[df_pivot_test["ratio_w/b"]>0.01]["ratio_w/b"]],
    palette=[sns.xkcd_rgb["pale red"], sns.xkcd_rgb["medium green"]],
    showmeans=True,
)
axes[1].set_title("ratio_w/b")
```

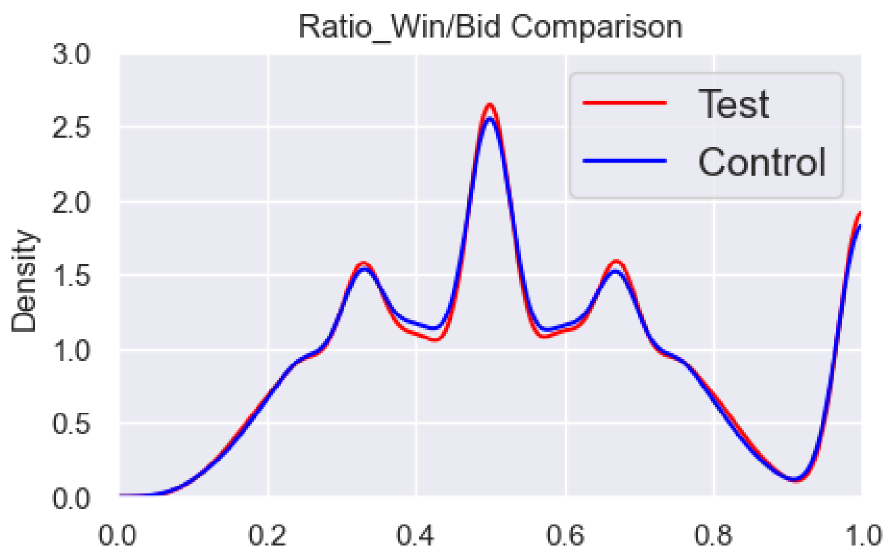
[373]: Text(0.5, 1.0, 'ratio_w/b')



[376]:
```
# Visualization Ratio_Conversion/Bid
plt.title('Ratio_Conversion/Bid Comparison')
# df_pivot_test["ratio_c/b"][df_pivot_test["ratio_c/b"]>0.01].hist(alpha=0.5,␣
 ↪edgecolor = 'black',linewidth=1, label='Test')
# df_pivot_control["ratio_c/b"][df_pivot_control["ratio_c/b"]>0.01].hist(alpha=0.
 ↪5, edgecolor = 'black',linewidth=1, label='Control')
plt.axis([0, 1, -0.1, 4.5])
df_pivot_test["ratio_c/b"][df_pivot_test["ratio_c/b"]>0.01].plot.
 ↪kde(label='Test')
df_pivot_control["ratio_c/b"][df_pivot_control["ratio_c/b"]>0.01].plot.
 ↪kde(label='Control')
plt.legend(prop={'size': 15})
plt.show()
```

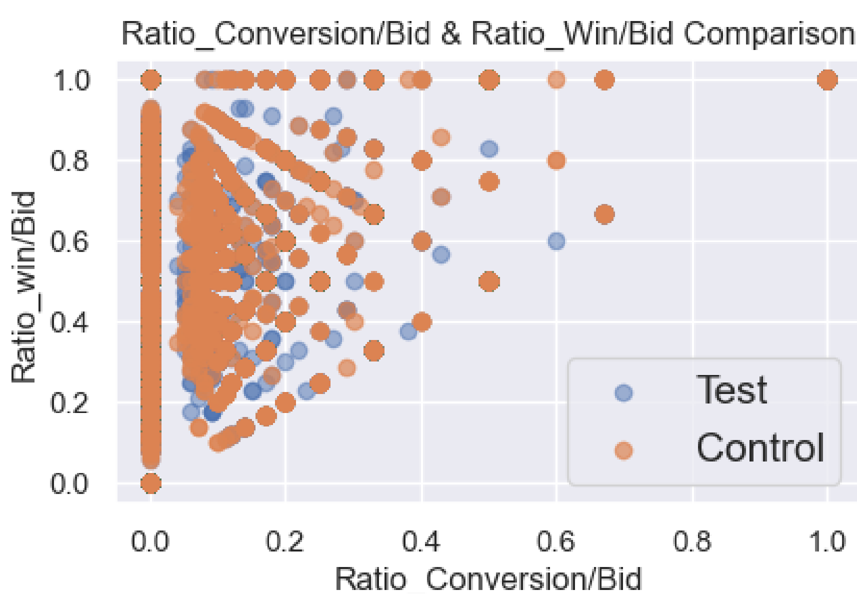


[378]:
```
#Visualization Ratio_Win/Bid
plt.title('Ratio_Win/Bid Comparison')
# df_pivot_test["ratio_c/b"][df_pivot_test["ratio_c/b"]>0.01].hist(alpha=0.5,␣
↪edgecolor = 'black',linewidth=1, label='Test')
# df_pivot_control["ratio_c/b"][df_pivot_control["ratio_c/b"]>0.01].hist(alpha=0.
↪5, edgecolor = 'black',linewidth=1, label='Control')
plt.axis([0, 1, 0, 3])
df_pivot_test["ratio_w/b"][df_pivot_test["ratio_w/b"]>0.01].plot.kde(color =␣
↪'red', label='Test')
df_pivot_control["ratio_w/b"][df_pivot_control["ratio_w/b"]>0.01].plot.kde(color␣
↪= 'blue', label='Control')
plt.legend(prop={'size': 15})
plt.show()
```



[381]:

```
#Visualization Ratio_Conversion/Bid and Ratio_Win/Bid
plt.title('Ratio_Conversion/Bid & Ratio_Win/Bid Comparison')
plt.scatter(x = df_pivot_test["ratio_c/b"], y = df_pivot_test["ratio_w/b"],␣
 ↪alpha = 0.5, label='Test')
plt.scatter(x = df_pivot_control["ratio_c/b"], y = df_pivot_control["ratio_w/
 ↪b"], alpha = 0.7, label='Control')
plt.xlabel("Ratio_Conversion/Bid")
plt.ylabel("Ratio_win/Bid")
plt.legend(prop={'size': 15})
plt.show()
```



For Question 2:
Summary 1: The average of ratio_conversion/bid of the control group is 0.021016, which is greater than test group 0.019727. It shows that the users in the control group are slightly easier to convert.
Summary 2: The average of ratio_win/bid of the test group is 0.501375, which is greater than control group 0.500328. It shows that the users in the test group are more chance to win.

2.5 Further EDA - Retargeting Analysis

[444]:
```
# Analysis the Users with conversion rate more than 1
df_retarget = df.loc[:, ["user_id", "bid", 'win', 'conversion']].
 ↪groupby(["user_id"]).agg({'conversion': np.sum,
                           'user_id': 'count', 'bid': np.sum, 'win': np.sum})
df_retarget = df_retarget[["user_id", "bid", 'win', 'conversion']]
df_retarget.columns = ['count', 'bid', 'win', 'conversion']
df_retarget = df_retarget.sort_values('conversion', ascending=False)
df_retarget_filter = df_retarget[df_retarget['conversion'] > 1]
df_retarget_filter_valuecount = df_retarget_filter['conversion'].value_counts().
 ↪sort_values(ascending=False)
print(df_retarget_filter_valuecount)
df_retarget_filter
```

```
2    423
3     37
5      2
4      1
Name: conversion, dtype: int64
```

[444]:

	count	bid	win	conversion
user_id				
e879bd196b9b3e4db974b8716c6f896e	32	22	17	5
54f07ff91a4441b2de263b955ab8a6a0	27	18	15	5
9570efef719d705326f0ff817ef084e6	18	13	9	4
ec151b6ecbb40275f4ac68bc99635554	21	12	8	3
0e4c11f657de720a1b7aeb04e2ba810e	20	11	7	3
...	...	...	...	...
f0b875eb6cff6fd5f491e6b6521c7510	9	5	4	2
3d79ce2378da226ffbcf28b59647431d	10	4	2	2
a588b762d68fe60225d3de3c647a52b9	23	16	6	2
1501b0c827e8fd20504d9eef796bb530	12	9	7	2
2c08e0bc5a77c595c881d7b8a189d05a	7	5	2	2

[463 rows x 4 columns]

[489]:

```
retarget_list = list(df_retarget_filter.index)
df_ori = df[df['user_id'].isin(retarget_list)]
df_ori
```

[489]:

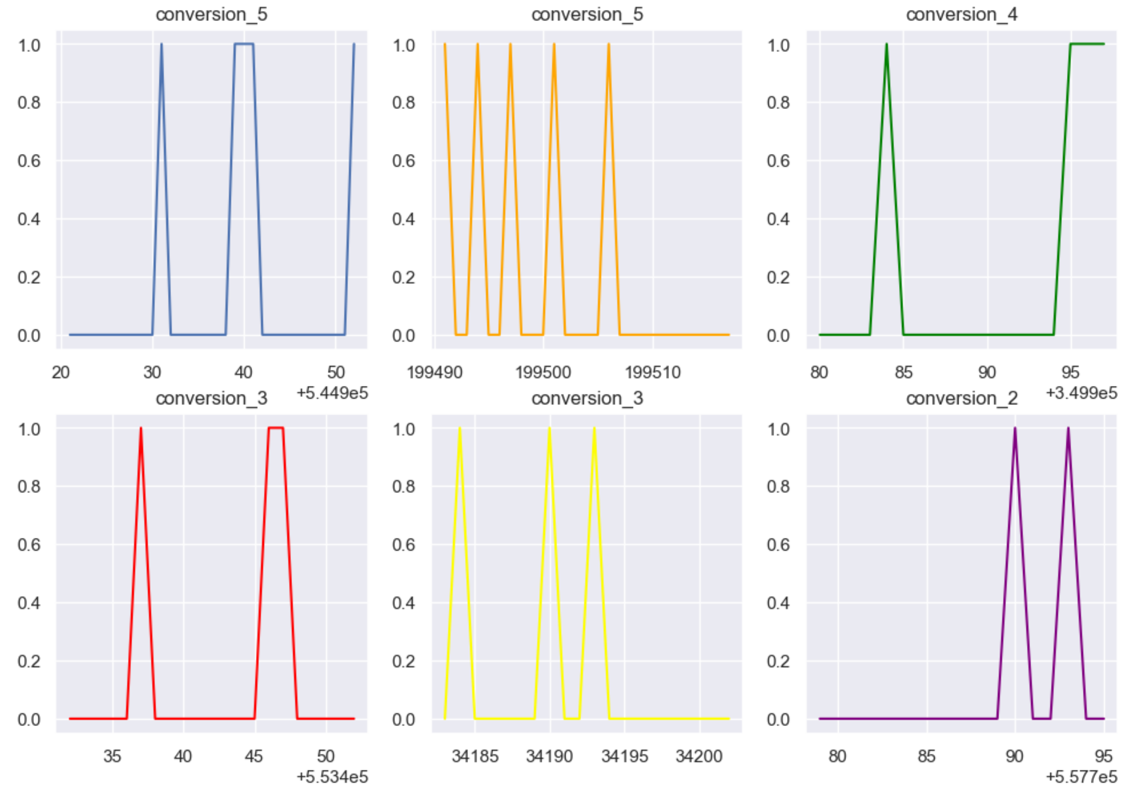
```
                  timestamp                           user_id  age gender  \
1044    2017-01-02 08:19:00  0070d23b06b1486a538c0eaa45dd167a   28      F
1045    2017-01-03 06:11:00  0070d23b06b1486a538c0eaa45dd167a   28      F
1046    2017-01-03 07:27:00  0070d23b06b1486a538c0eaa45dd167a   28      F
1047    2017-01-03 13:55:00  0070d23b06b1486a538c0eaa45dd167a   28      F
1048    2017-01-03 21:53:00  0070d23b06b1486a538c0eaa45dd167a   28      F
...                     ...                               ...  ...    ...
599566  2017-01-15 00:39:00  ffcf3174be79160cd4f3eb50bc76d034   24      F
599567  2017-01-15 16:19:00  ffcf3174be79160cd4f3eb50bc76d034   24      F
599568  2017-01-20 09:18:00  ffcf3174be79160cd4f3eb50bc76d034   24      F
599569  2017-01-21 23:56:00  ffcf3174be79160cd4f3eb50bc76d034   24      F
599570  2017-01-23 00:16:00  ffcf3174be79160cd4f3eb50bc76d034   24      F

       location  test  bid  win  conversion            date_cat  date_weekday  \
1044         AR     0    1    1           0 2017-01-02 08:19:00             0
1045         AR     0    0    0           0 2017-01-03 06:11:00             1
1046         AR     0    1    1           0 2017-01-03 07:27:00             1
1047         AR     0    0    0           0 2017-01-03 13:55:00             1
1048         AR     0    0    0           0 2017-01-03 21:53:00             1
...         ...   ...  ...  ...         ...                 ...           ...
599566       FL     0    1    1           1 2017-01-15 00:39:00             6
599567       FL     0    1    0           0 2017-01-15 16:19:00             6
599568       FL     0    1    1           0 2017-01-20 09:18:00             4
599569       FL     0    0    0           0 2017-01-21 23:56:00             5
599570       FL     0    1    1           1 2017-01-23 00:16:00             0

        date_day  date_hour
1044           2          8
1045           3          6
1046           3          7
1047           3         13
1048           3         21
...          ...        ...
599566        15          0
599567        15         16
599568        20          9
599569        21         23
599570        23          0

[7461 rows x 13 columns]
```

[488]:

```
fig, ax = plt.subplots(figsize=(12, 8))
ax1 = plt.subplot(231)
```

```
ax1.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 5].index[0]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 5].
 ↪index[0]]['conversion'])
ax1.set_title("conversion_5")

ax2 = plt.subplot(232)
ax2.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 5].index[1]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 5].
 ↪index[1]]['conversion'], c = 'orange')
ax2.set_title("conversion_5")

ax3 = plt.subplot(233)
ax3.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 4].index[0]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 4].
 ↪index[0]]['conversion'], c = 'green')
ax3.set_title("conversion_4")

ax4 = plt.subplot(234)
ax4.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 3].index[0]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 3].
 ↪index[0]]['conversion'], c = 'red')
ax4.set_title("conversion_3")

ax5 = plt.subplot(235)
ax5.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 3].index[1]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 3].
 ↪index[1]]['conversion'], c = 'yellow')
ax5.set_title("conversion_3")

ax6 = plt.subplot(236)
ax6.plot(df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 2].index[0]].index,
         df_ori[df_ori['user_id'] ==␣
 ↪df_retarget_filter[df_retarget_filter['conversion'] == 2].
 ↪index[0]]['conversion'], c = 'purple')
ax6.set_title("conversion_2")
```

[488]: Text(0.5, 1.0, 'conversion_2')



[514]:
```
# Ratio to convert the user more than 1
df_retarget_filter_1 = df_retarget[df_retarget['conversion'] >= 1]
ratio_retarget_morethan2 = df_retarget_filter.shape[0]/df_retarget_filter_1.
  ↪shape[0]
print('ratio_retarget_morethan2:')
print(df_retarget_filter_1.shape)
print(df_retarget_filter.shape)
print('ratio_retarget_morethan2', ratio_retarget_morethan2)

ratio_retarget_random = df_retarget_filter_1.shape[0]/df_retarget.shape[0]
print('ratio_retarget_random:')
print(df_retarget.shape)
print(df_retarget_filter_1.shape)
print('ratio_retarget_random', ratio_retarget_random)

# ratio_retarget_perbid = df_retarget_filter.shape[0]/
  ↪df_retarget[df_retarget['bid'] != 0].shape[0]
# print(df_retarget[df_retarget['bid'] != 0].shape)
# print(df_retarget_filter.shape)
```

```
# print('ratio_retarget_perbid', ratio_retarget_perbid)
```

```
ratio_retarget_morethan2:
(5607, 4)
(463, 4)
ratio_retarget_morethan2 0.08257535223827359
ratio_retarget_random:
(63137, 4)
(5607, 4)
ratio_retarget_random 0.08880688027622471
```

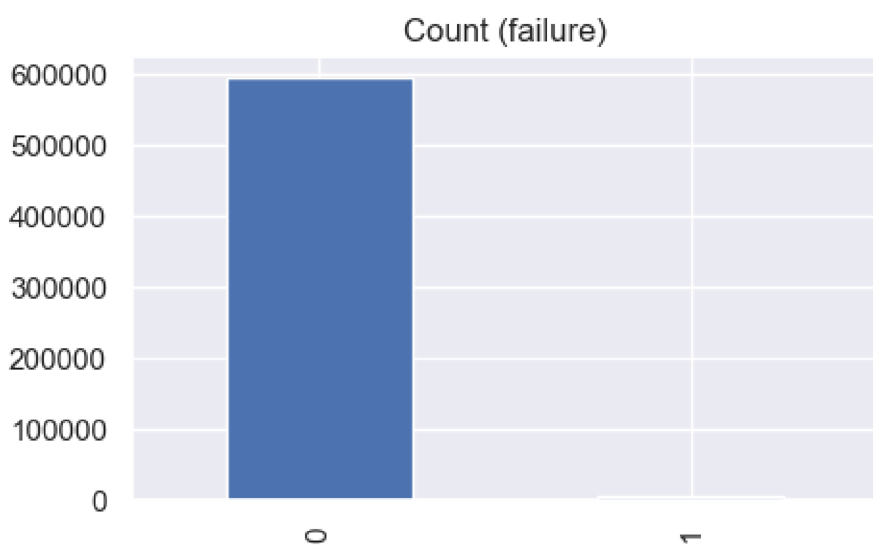
For Question 3:
Summary 1: There are total 463 users converted more than once, which takes 8.25% of total previous converted user. If not regarting the previous converted user, the ratio is about 8.89% to convert new users on a random basis. Therefore the percentage is not higher to retarget the previous converted users.

3 Modelling

3.1 Train/test split

[518]:
```
# Investigation
target_count = df.conversion.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (failure)');
```

```
Class 0: 593885
Class 1: 6115
Proportion: 97.12 : 1
```



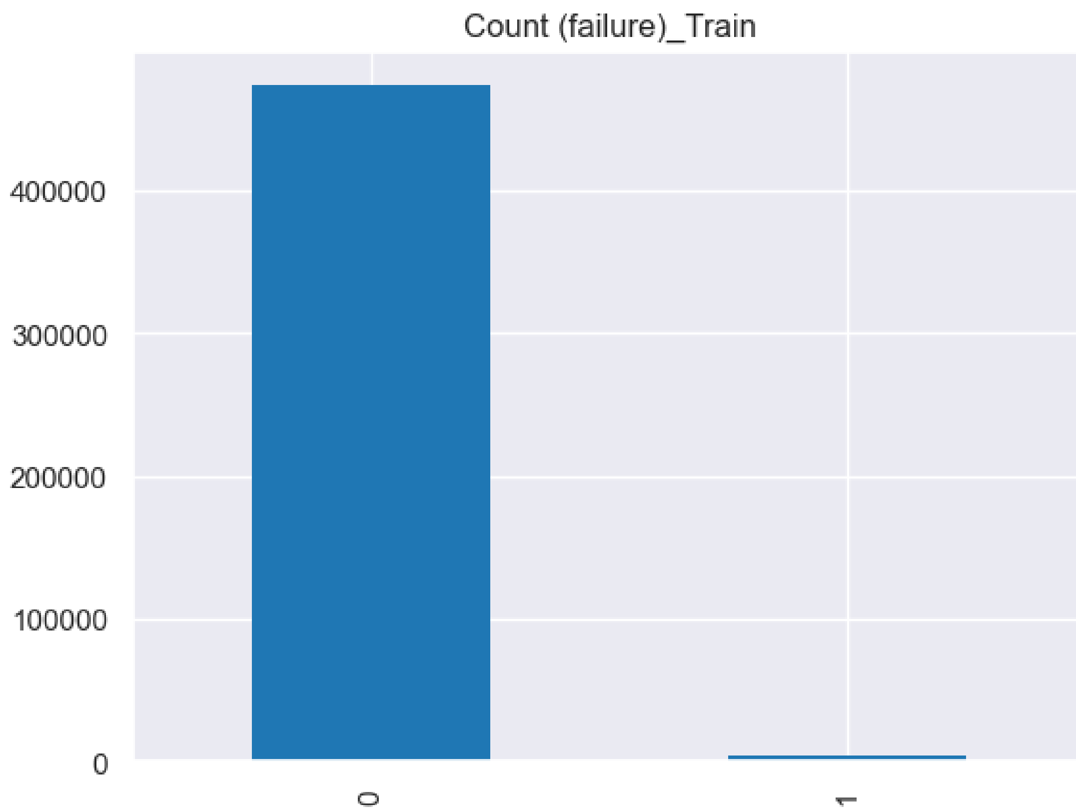
[519]:
```
df.head()
```

[519]:
```
            timestamp                          user_id  age gender location  \
0 2017-01-01 13:43:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
1 2017-01-04 03:59:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
2 2017-01-04 17:41:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
3 2017-01-07 04:02:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL
4 2017-01-08 09:05:00  00003e3b9e5336685200ae85d21b4f5e   33      F       FL

   test  bid  win  conversion            date_cat  date_weekday  date_day  \
0     1    1    0           0 2017-01-01 13:43:00             6         1
1     1    0    0           0 2017-01-04 03:59:00             2         4
2     1    1    1           0 2017-01-04 17:41:00             2         4
3     1    1    1           0 2017-01-07 04:02:00             5         7
4     1    1    0           0 2017-01-08 09:05:00             6         8

   date_hour
0         13
1          3
2         17
3          4
4          9
```

[773]:
```
# GroupShuffleSplit
from sklearn.model_selection import GroupShuffleSplit

train_inds, test_inds = next(GroupShuffleSplit(test_size=.20, n_splits=2,␣
 ↪random_state = 7).split(df[['age', 'gender',
       'location', 'test', 'bid', 'win', 'date_weekday', 'date_day',␣
 ↪'date_hour']], groups=df['user_id']))

train_X = df.iloc[train_inds][['age', 'gender', 'location', 'test', 'bid',␣
 ↪'win', 'date_weekday', 'date_day', 'date_hour']]

train_y = df.iloc[train_inds]['conversion']

test_X = df.iloc[test_inds][['age', 'gender', 'location', 'test', 'bid', 'win',␣
 ↪'date_weekday', 'date_day', 'date_hour']]

test_y = df.iloc[test_inds]['conversion']

# train_y = train_y.astype('int')
# test_y = test_y.astype('int')
```

```python
# Investigation
target_count = train_y.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (failure)_Train')
plt.grid()
```

```
Class 0: 473924
Class 1: 4923
Proportion: 96.27 : 1
```



[771]:
```python
# Random Train_Test_Split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df[['age', 'gender',␣
 ↪'location', 'test', 'bid', 'win', 'date_weekday', 'date_day',
                                                        'date_hour']].values, df.
 ↪conversion.values, test_size=0.2, random_state = 7)
```

```
X_train = pd.DataFrame(X_train, columns = ['age', 'gender', 'location', 'test',␣
 ↪'bid', 'win', 'date_weekday', 'date_day',
                                                         'date_hour'])

X_test = pd.DataFrame(X_test, columns = ['age', 'gender', 'location', 'test',␣
 ↪'bid', 'win', 'date_weekday', 'date_day',                                     ␣
 ↪                'date_hour'])

y_train = y_train.astype('int')
y_test = y_test.astype('int')

y_train = pd.Series(y_train)
y_test = pd.Series(y_test)

# y_train.name = "failure"
# y_test.name = "failure"

# Investigation
target_count = y_train.value_counts()
print('Class 0:', target_count[0])
print('Class 1:', target_count[1])
print('Proportion:', round(target_count[0] / target_count[1], 2), ': 1')

target_count.plot(kind='bar', title='Count (failure)_Train')
plt.grid()
```

```
Class 0: 475134
Class 1: 4866
Proportion: 97.64 : 1
```

3.2 Imbalanced Process - Upsampling

[774]:
```
# train_X = X_train
# train_y = y_train
# test_X = X_test
# test_y = y_test

# LabelEncoder for train_X
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
train_X['location_encoder'] = le.fit_transform(train_X['location'])
train_X['gender_encoder'] = le.fit_transform(train_X['gender'])
train_X = train_X.drop(['location','gender'], axis=1)
```

[775]:
```
# LabelEncoder for test_X
test_X['location_encoder'] = le.fit_transform(test_X['location'])
test_X['gender_encoder'] = le.fit_transform(test_X['gender'])
test_X = test_X.drop(['location','gender'], axis=1)
```

[776]:
```
import imblearn
from collections import Counter

X = train_X
y = train_y

counter = Counter(y)
print(counter)

def plot_2d_space(X, y, label='Classes'):
    colors = ['#1F77B4', '#FF7F0E']
    markers = ['o', 's']
    for l, c, m in zip(np.unique(y), colors, markers):
        plt.scatter(
            X[y==l, 0],
            X[y==l, 1],
            c=c, label=l, marker=m
        )
    plt.title(label)
    plt.legend(loc='upper right')
    plt.show()
```

```
Counter({0: 473924, 1: 4923})
```

[738]:
```
# Method 1 - Upsampling all - Random
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()
X_ros, y_ros = ros.fit_resample(X, y)
train_resample= X_ros.join(y_ros, how='left')

print(X_ros.shape[0] - X.shape[0], 'new random picked points')

# plot_2d_space(X_ros, y_ros, 'Random over-sampling')

train_resample_X = X_ros[['age', 'test', 'bid', 'win', 'date_weekday',␣
 ↪'date_day', 'date_hour',
       'location_encoder', 'gender_encoder']]
train_resample_y = y_ros
```

```
470268 new random picked points
```

[777]:
```
# Method 2 - Upsampling all - SMOTE
y = pd.DataFrame(train_y)
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy=1, random_state=7)
X_sm, y_sm = smote.fit_resample(X, y)
train_resample= X_sm.join(y_sm, how='left')
```

```
counter = Counter(y_sm)
print(counter)
# plot_2d_space(X_sm, y_sm, 'SMOTE over-sampling')

train_resample_X = X_sm[['age', 'test', 'bid', 'win', 'date_weekday',␣
 ↪'date_day', 'date_hour',
       'location_encoder', 'gender_encoder']]
train_resample_y = y_sm
```

```
Counter({'conversion': 1})
```

[785]:
```
# Method 3 - Upsampling on target user basis
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler

y = pd.DataFrame(train_y, columns = ['conversion'])

user_name = df.loc[train_X.index, 'user_id'].unique()

df_over_sampling_X = pd.DataFrame(columns=['age', 'test', 'bid', 'win',␣
 ↪'date_weekday', 'date_day', 'date_hour',
       'location_encoder', 'gender_encoder'])
df_over_sampling_y = pd.DataFrame(columns=['conversion'])
n_new_rows = 0

for user in user_name:
    data = train_X.loc[df[df["user_id"] == user].index, :]
    data_y = y.loc[df[df["user_id"] == user].index, 'conversion']
    X_single = data[['age', 'test', 'bid', 'win', 'date_weekday', 'date_day',␣
 ↪'date_hour',
       'location_encoder', 'gender_encoder']]
    y_single = pd.DataFrame(data_y, columns = ['conversion'])
    if len(data_y.unique()) == 2:
        ros = RandomOverSampler()
        X_ros, y_ros = ros.fit_resample(X_single, y_single)
        n_new_rows += X_ros.shape[0] - X_single.shape[0]

        df_over_sampling_X = pd.concat([df_over_sampling_X, X_ros])
        df_over_sampling_y = pd.concat([df_over_sampling_y, y_ros])
    else:
        df_over_sampling_X = pd.concat([df_over_sampling_X, X_single])
        df_over_sampling_y = pd.concat([df_over_sampling_y, y_single])

print("n_new_rows", n_new_rows)
train_resample= pd.concat([df_over_sampling_X,df_over_sampling_y], axis = 1)
```

```
[723]: print("number of new rows create: ", df_over_sampling_y['conversion'].
     ↪value_counts()[1] - y['conversion'].value_counts()[1])
train_resample_X_method3 = df_over_sampling_X[['age', 'test', 'bid', 'win',␣
     ↪'date_weekday', 'date_day', 'date_hour',
       'location_encoder', 'gender_encoder']]
train_resample_y_method3 = df_over_sampling_y.astype('int')
```

```
number of new rows create:  48868
```

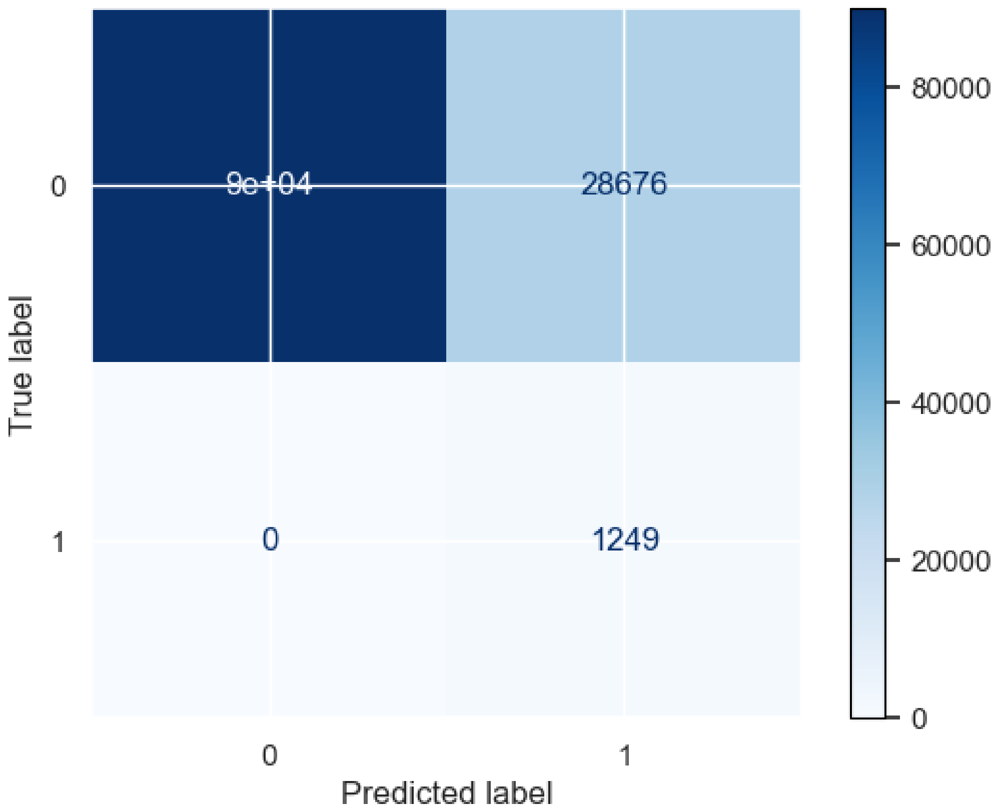
3.3 Logistic Regression Model

```
[724]: import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (2, 2)
plt.style.use('default')
```

```
[725]: # Logistic Regression Model - Shuffle

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score,␣
     ↪matthews_corrcoef
from sklearn.metrics import precision_recall_curve, roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

LR = LogisticRegression(max_iter=1000, class_weight='balanced')
# Fit the model on the trainng data.
LR.fit(train_resample_X_method3, train_resample_y_method3.values.ravel())

# Print the accuracy, precision, recall from the testing data.
print("Accuracy: ", accuracy_score(test_y, LR.predict(test_X)))
print("Precision: ", precision_score(test_y, LR.predict(test_X)))
print("Recall: ", recall_score(test_y, LR.predict(test_X)))
print("F1 score: ", f1_score(test_y, LR.predict(test_X)))
print("MCC score: ", matthews_corrcoef(test_y, LR.predict(test_X)))
print(classification_report(test_y, LR.predict(test_X), target_names=['Class 0␣
     ↪Fail to Convert:', 'Class 1 Suceed to Convert:']))

# Plot Confusion Matrix
print("Confusion matrix: ", confusion_matrix(test_y, LR.predict(test_X)))
cm=confusion_matrix(test_y, LR.predict(test_X))
plot_confusion_matrix(LR, test_X, test_y, cmap=plt.cm.Blues)
plt.show()
```

```
Accuracy:  0.7610333333333333
Precision:  0.04173767752715121
```

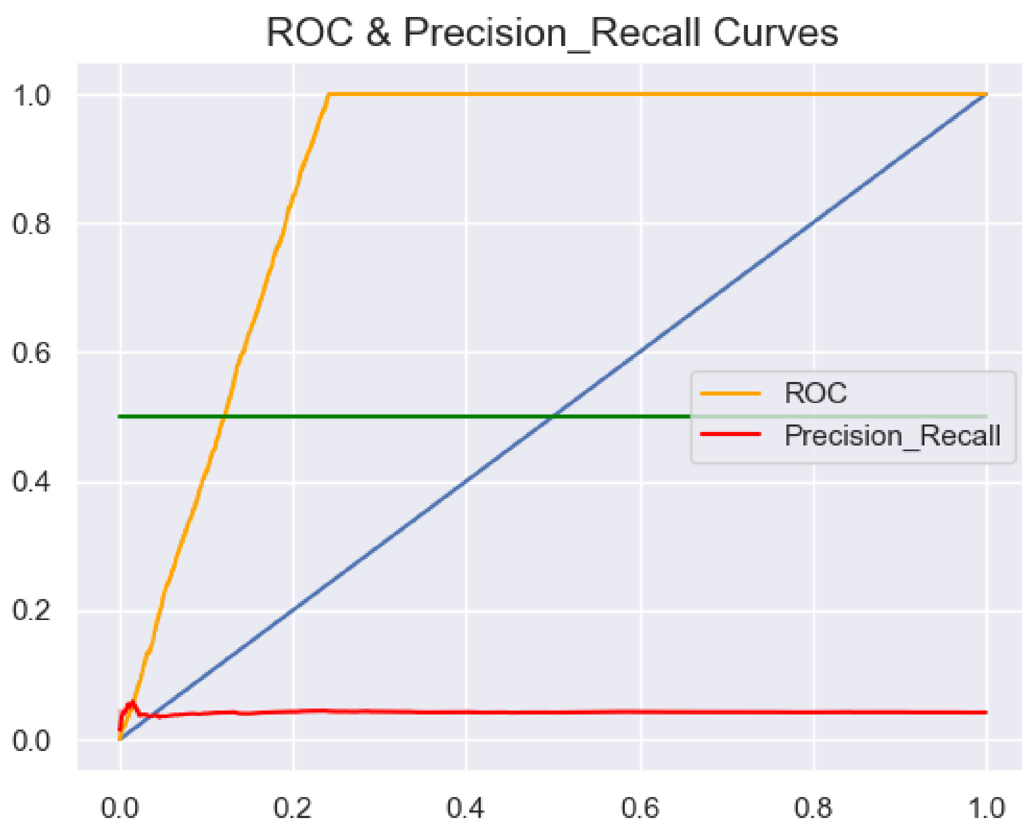
```
Recall:  1.0
F1 score:  0.08013087829601591
MCC score:  0.1779293684678036
                            precision    recall  f1-score   support

  Class 0 Fail to Convert:       1.00      0.76      0.86    118751
Class 1 Suceed to Convert:       0.04      1.00      0.08      1249

                 accuracy                           0.76    120000
                macro avg       0.52      0.88      0.47    120000
             weighted avg       0.99      0.76      0.85    120000

Confusion matrix:  [[90075 28676]
 [    0  1249]]
```



[743]:
```
# Plot ROC Curve - Shuffle
probs = LR.predict_proba(test_X)
probs = probs[:,1]
fpr, tpr, thresholds = roc_curve(test_y, probs)
sns.lineplot([0,1], [0, 1])
plt = sns.lineplot(fpr, tpr, legend='full', label=str("ROC"), c = 'orange')
```

```
plt.set_title("ROC & Precision_Recall Curves", fontsize=15)
auc_score = roc_auc_score(test_y, probs)
print("AUC_ROC: %.3f" % auc_score)

# Plot Precision/Recall Curve
probs = LR.predict_proba(test_X)
probs = probs[:,1]
precision, recall, thresholds = precision_recall_curve(test_y, probs)
sns.lineplot([0,1], [0.5, 0.5], c = 'green')
plt = sns.lineplot(recall, precision, legend='full',␣
 ↪label=str("Precision_Recall"), c = 'red')
pr_auc_score = auc(recall, precision)
print("AUC_PR: %.3f" % pr_auc_score)
```

```
AUC_ROC: 0.880
AUC_PR: 0.042
```

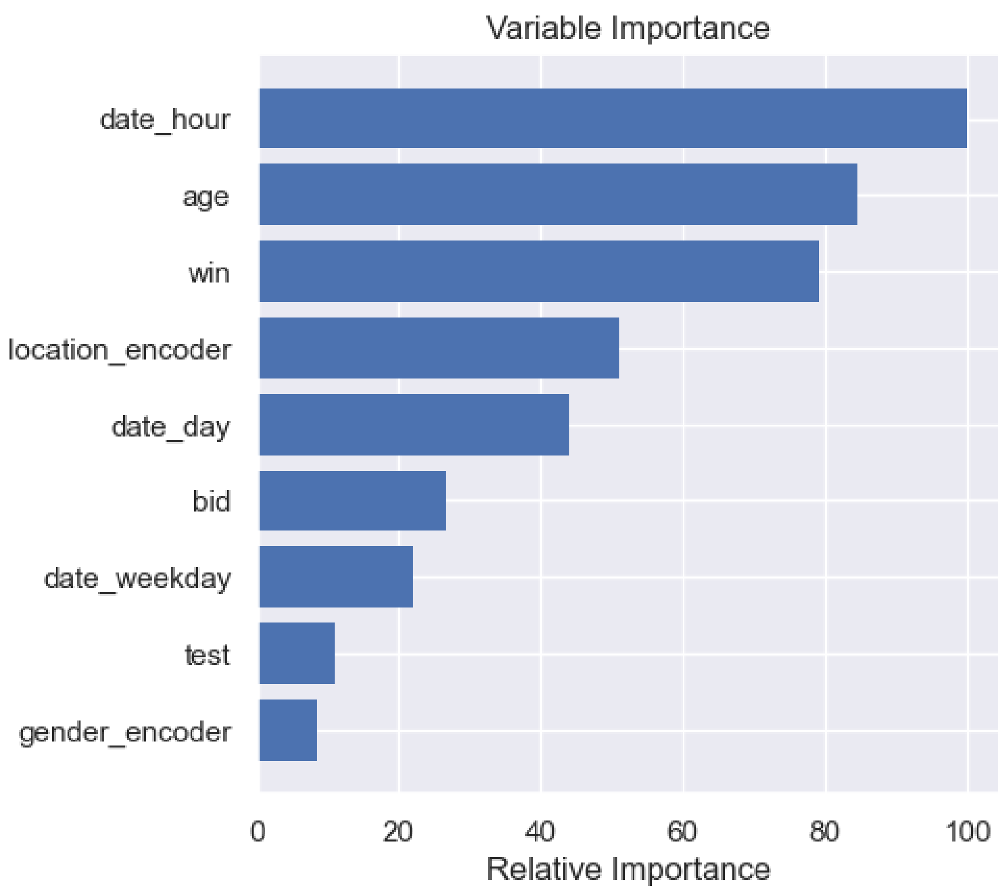


Section 3.3 - Summary: Based on the confusion matrix, the false postive is still high. Further analysis of re-sampling, test/train split will need to investigate further.

3.4 Random Forest Model

[780]:
```
import matplotlib.pyplot as plt

feature_importance = RF.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())[:30]
sorted_idx = np.argsort(feature_importance)[:30]

pos = np.arange(sorted_idx.shape[0]) + .5
print(pos.size)
sorted_idx.size
plt.figure(figsize=(5,5))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```

9



Section 3.4:
Summary: The day of hour rate, the age and the win rate are the top 3 critical features to impact final conversion rate.

4 Evaluating and Concluding

For Question 1:
Summary 1: The users at younger age group, has higher chance to open websites.
Summary 2: There is no distinct preference differentiated from genders in targeted group to open websites.
Summary 3: The female group is slightly higher chance to open websites than male group.
Summary 4: There is no distinct difference from states in targeted group to open websites.
Summary 5: Target Users located in Tennessee state is slightly higher chance to open websites, while users in Arizona is lower chance.
Summary 6: Overall 49% of the targeted group is to open websites.

For Question 2:
Summary 1: The average of ratio_conversion/bid of the control group is 0.021016, which is greater than test group 0.019727. It shows that the users in the control group are slightly easier to convert.
Summary 2: The average of ratio_win/bid of the test group is 0.501375, which is greater than the control group with 0.500328. It shows that the users in the test group are more chance to win.

For Question 3:
Summary 1: There are total 463 users converted more than once, which takes 8.25% of total previous converted user. If not regarting the previous converted user, the ratio is about 8.89% to convert new users on a random basis. Therefore the percentage is not higher to retarget the previous converted users.