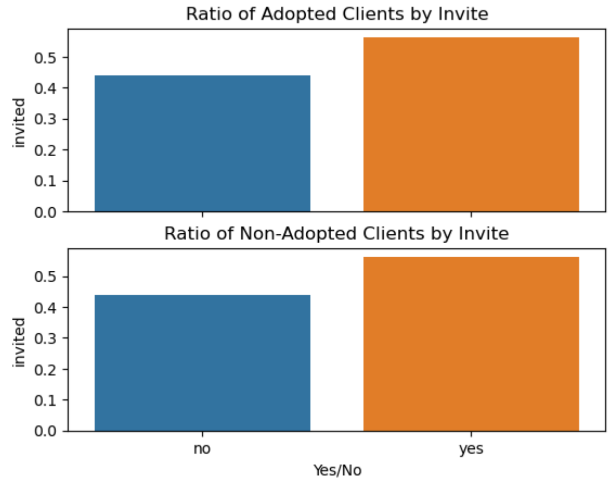
RELAX PROJECT WRITE-UP

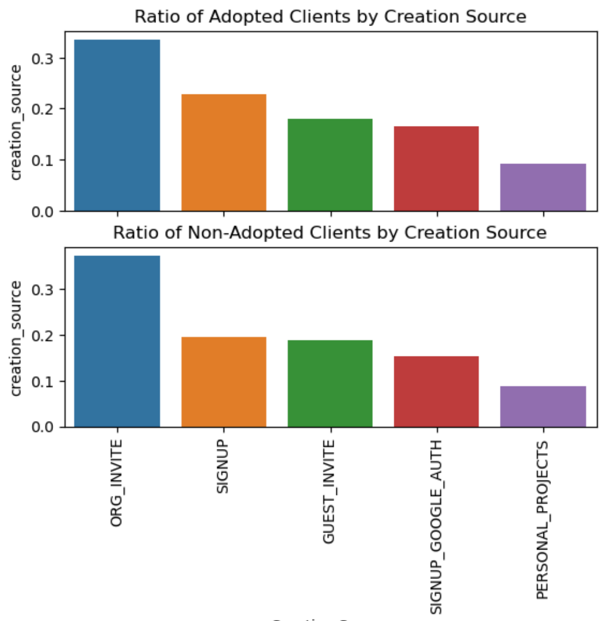
I began the project by importing the ‘.csv’ files into Pandas DataFrames for data cleaning and EDA. After performing a brief examination of the DataFrames which included a ‘.describe()’ and ‘.info()’ calls on them, I quickly found NaN values in the ‘last_session_creation_time’ and ‘invited_by_user_id’ columns. To rectify these issues I converted all columns which were clearly Datetime data into DateTimes, and assuming that the reason ‘last_session_creation_time’ had NaN values (now NaT) was due to the corresponding user not having logged in after creating their account. A quick review of the table of logins revealed this to be true. Therefore I filled those values with their corresponding DateTime in the ‘creation_time’ column. The ‘invited_by_user_id’ was much simpler: under the assumption that the NaN values were due to not having been invited I filled them with zero.

Approaching the problem as an ML Classifier issue where I would take the data to predict whether or not a user would become ‘adopted’ or not per the Relax companies standards, I converted the ‘invited_by_user_id’ to simply ‘invited’ and converted any value that wasn’t 0 to 1. I then needed to create my dependent variable dubbed ‘adopted’ by calculating which users had logged in at least 3 times within a week at any point in time. I created that column thusly:

```
users_df = users_df.merge(user_engagement_df,
               how='inner',
               left_on='object_id',
               right_on='user_id')
users_df.drop(columns=['object_id', 'visited'], inplace=True)
users_df.rename(columns={'time_stamp': 'logins'}, inplace=True)
users_df['logins'] = pd.to_datetime(users_df['logins'])
users_df.sort_values(['name','logins'], ascending=True, inplace=True)
users_df = users_df.reset_index().drop(columns='index')
adopted = []
users = users_df['name'].unique()
for user in users:
    temp_df = pd.DataFrame(users_df[users_df['name'] == user])
    temp_df = temp_df.resample(rule='W', on='logins').agg({'logins':'count'})
    temp_df.rename(columns={'logins':'login counts'}, inplace=True)
    if np.any(temp_df['login counts'] >= 3):
        adopted.append(1)
    else:
        adopted.append(0)
del temp_df
merge_dict = {users[i]: adopted[i] for i in range(len(users))}
merge_df = pd.DataFrame(list(merge_dict.items()), columns=['names', 'adopted'])
users_df = users_df.merge(merge_df, how='inner', left_on='name', right_on='names')
```

After that I evaluated the data finding that none of the current variables provided a clear indicator as to what factor suggested a user would become adopted or not.

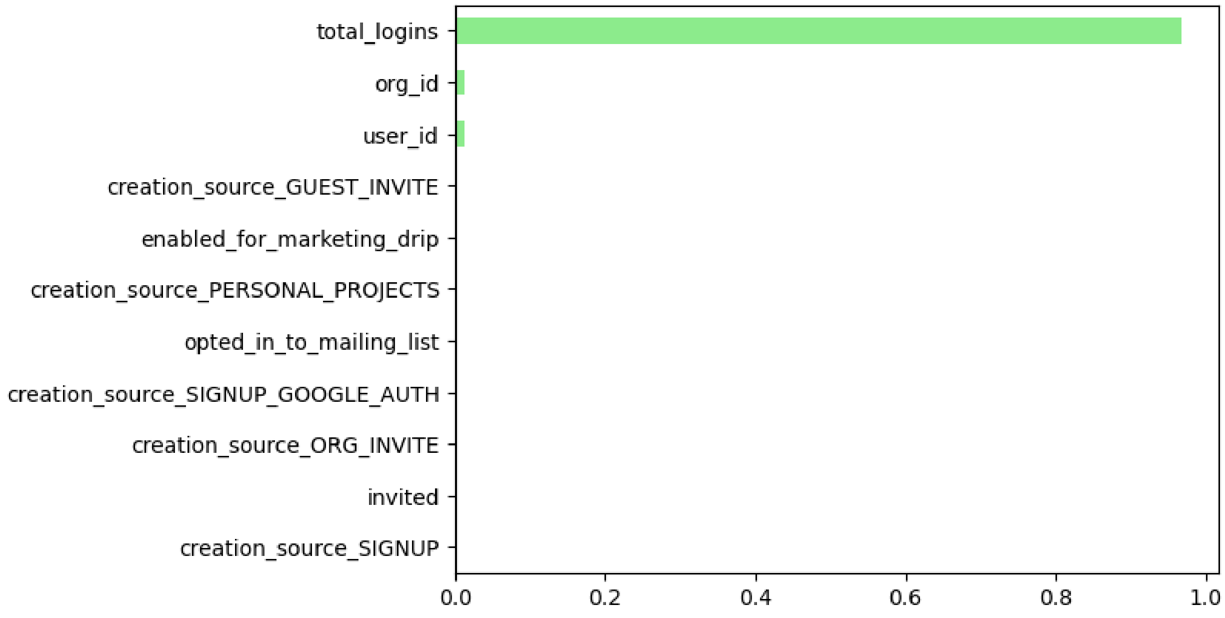




Thus I needed to feature engineer another data point: ‘total_logins’

I found this to be frustrating as the ‘total_logins’ of course would make a marvelous predictor but was really only one step away from using the ‘adopted’ column on itself as both an indicator and a predictor. (that’s a cyclical definition for you!) Simply put, yes It would make an excellent predictive indicator, but it tells absolutely nothing about what causes people to be adopted. More analysis would need to be done specifically on the individuals who regularly login and gather more habitual information from these individuals to find some kind of confluence.

The ultimate measure of what factors predict adoption was this:


total_logins
org_id
user_id
creation_source_GUEST_INVITE
enabled_for_marketing_drip
creation_source_PERSONAL_PROJECTS
opted_in_to_mailing_list
creation_source_SIGNUP_GOOGLE_AUTH
creation_source_ORG_INVITE
invited
creation_source_SIGNUP