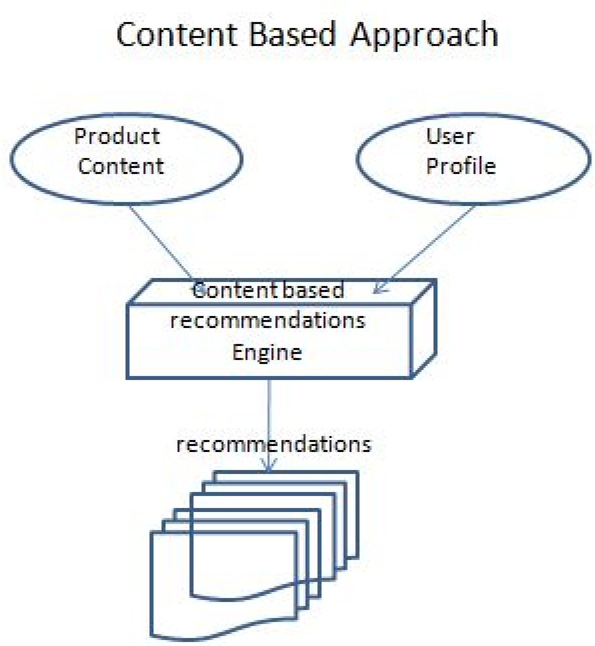
Assignment 4 - Content based filtering using Light GBM

Content based Filtering

A **Content-based recommendation system** tries to recommend items to users **based** on their profile. The user's profile revolves around that user's preferences and tastes. It is shaped **based** on user ratings, including the number of times that user has clicked on different items or perhaps even liked those items.



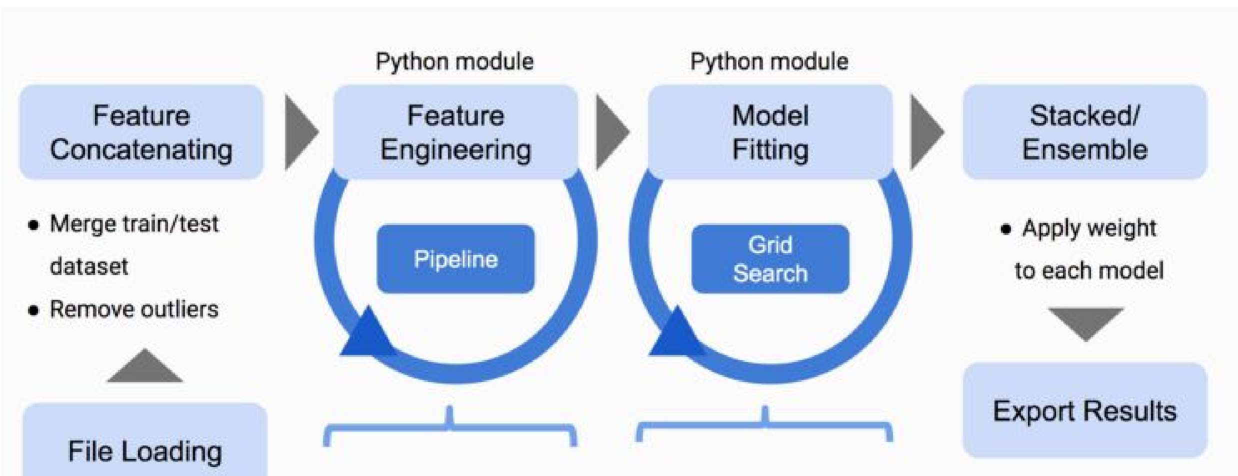
Moreover, in terms of user preference it is usually expressed by two categories:

1. **Explicit Feedback**: Explicit feedback are the ratings of movies by users on Netflix, ratings of products on Amazon. It takes into consideration the input

from the user about how they liked or disliked a product. Explicit feedback data are quantifiable.

2. **Implicit Feedback**: Implicit data is easy to collect in large quantities without any effort from the users. The goal is to convert user behavior into user preferences which indirectly reflect opinion through observing user behavior. For example, a user that bookmarked many articles by the same author probably likes that author. Implicit feedback includes the data logs such as clicks/ no clicks, purchase or no purchase, user engagement, conversion rate, etc. The data for Implicit feedback is often readily available from transaction logs.

LIGHT GBM



LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages: Faster training speed and higher efficiency. Lower memory usage. Better accuracy.

Our objective is to classify whether or not a particular user will click a products' domain based on his/her implicit actions. "Click" being our dependent variable and the other features being independent variables.

Explained below is the prototype dataset that covers 0-3 stages of a typical recommendation workflow. I.e. data preparation, model building and evaluation of the model.

Data preparation:

We start by installing the required modules and importing necessary libraries.

https://www.kaggle.com/c/avazu-ctr-prediction/data

In [159]:
```
DATA_FILE='train.gz'
all_data = pd.read_csv(DATA_FILE, sep=',', compression='gzip')
```

In [267]:
```
import featuretools as ft
import os
from tqdm import tqdm


def make_user_sample(all_data, user_ids, out_dir):
    all_data_sample = all_data[all_data["id"].isin(user_ids)]

    try:
        os.mkdir(out_dir)
    except:
        pass
    all_data_sample.to_csv(os.path.join(out_dir, "all_data.csv"), index=None)

def main():
    all_data = pd.read_csv(DATA_FILE, sep=',', compression='gzip')
    users_unique = all_data["id"].unique()
    chunksize = 500000
    part_num = 0
    partition_dir = "partitioned_data"
    try:
        os.mkdir(partition_dir)
    except:
        pass
    for i in tqdm(range(0, len(users_unique), chunksize)):
        users_keep = users_unique[i: i+chunksize]
        make_user_sample(all_data, users_keep,os.path.join(partition_dir, "part_%d" % part_num))
        part_num += 1

if __name__ == "__main__":
    main()
```

```
  0%|                                    | 0/81 [00:00<?, ?it/s]
  1%|█                                   | 1/81 [00:14<19:23, 14.55s/it]
  2%|██                                  | 2/81 [00:29<19:10, 14.56s/it]
  4%|███                                 | 3/81 [00:46<19:57, 15.36s/it]
```

In [273]:

```python
all_data = pd.read_csv('partitioned_data/part_0/all_data.csv')
```

In [275]:

```python
pd.set_option('display.float_format', '{:.0f}'.format)
all_data.head(5)
```

Out[275]:

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	...	device_type	device_conn_t
0	1000009418151094400	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	
1	10000169349117863936	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	
2	10000371904215119872	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	
3	10000640724480839680	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	1	
4	10000679056417042432	0	14102100	1005	1	fe8cc448	9166c161	0569f928	ecad2386	7801e8d9	...	1	

5 rows × 24 columns

Converting randomly initiated values to readable values so that the data is easy to comprehend for the user.

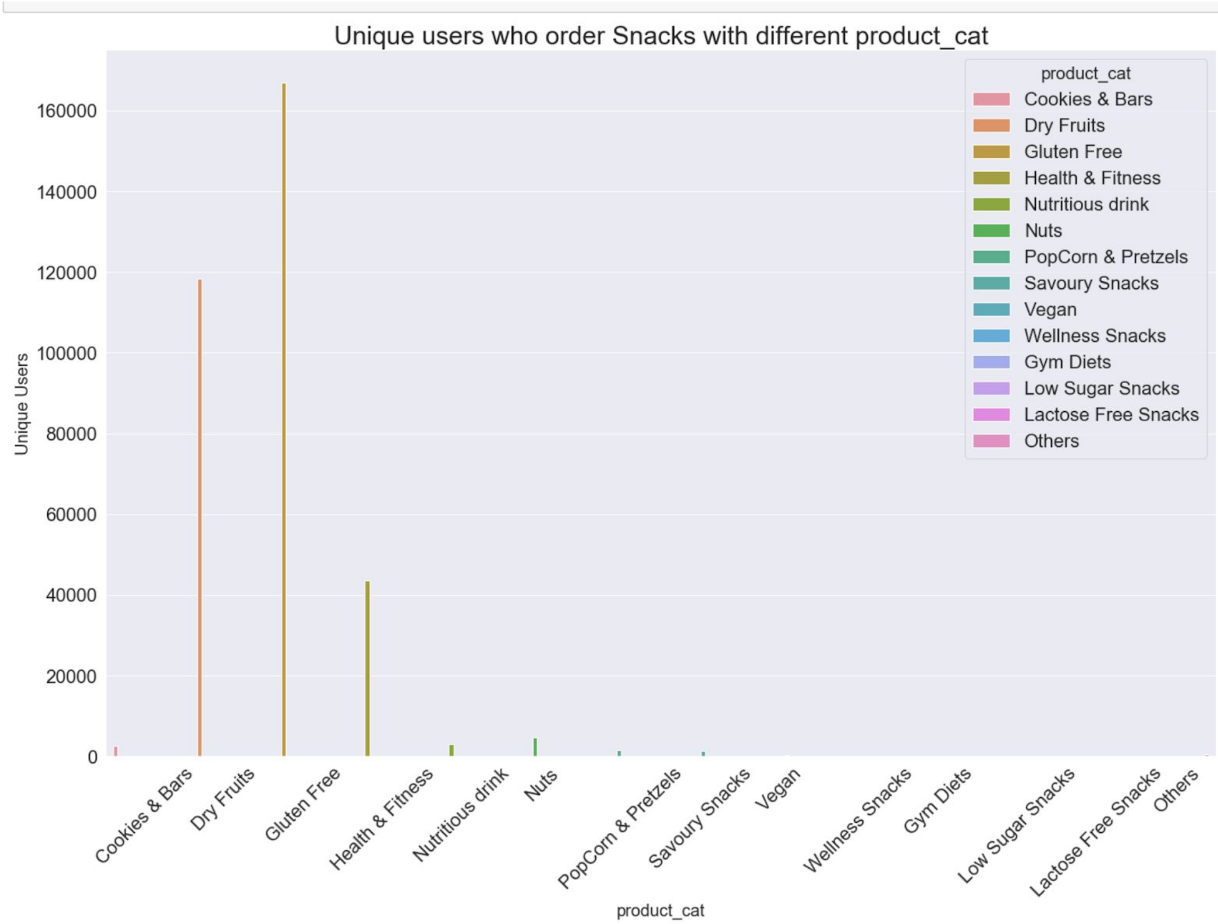
In [137]:

```python
def conditions(s):
    if (s['site_category'] == '28905ebd') :
        return 'Chips & Crackers'
    elif (s['site_category'] == '0569f928'):
        return 'Cookies & Bars'
    elif (s['site_category'] == 'f028772b'):
        return 'Dry Fruits'
    elif (s['site_category'] == '50e219e0'):
        return 'Gluten Free'
    elif (s['site_category'] == '3e814130'):
        return 'Health & Fitness'
    elif (s['site_category'] == '76b2941d'):
        return 'Natural Snacks'
    elif (s['site_category'] == 'f66779e6'):
        return 'Nutritious drink'
    elif (s['site_category'] == '335d28a8'):
        return 'Nuts'
    elif (s['site_category'] == '72722551'):
        return 'PopCorn & Pretzels'
    elif (s['site_category'] == '75fa27f6'):
        return 'Savoury Snacks'
    elif (s['site_category'] == '110ab22d'):
        return 'Snack mixes'
    elif (s['site_category'] == 'c0dd3be3'):
        return 'Vegan'
    elif (s['site_category'] == 'bcf865d9'):
        return 'Wellness Snacks'
    elif (s['site_category'] == 'a818d37a'):
        return 'Gym Diets'
    elif (s['site_category'] == '42a36e14'):
        return 'Low Sugar Snacks'
    elif (s['site_category'] == 'e787de0e'):
        return 'Lactose Free Snacks'
    elif (s['site_category'] == '5378d028'):
        return 'Dairy Products'
    else:
        return 'Others'
```

In [246]:

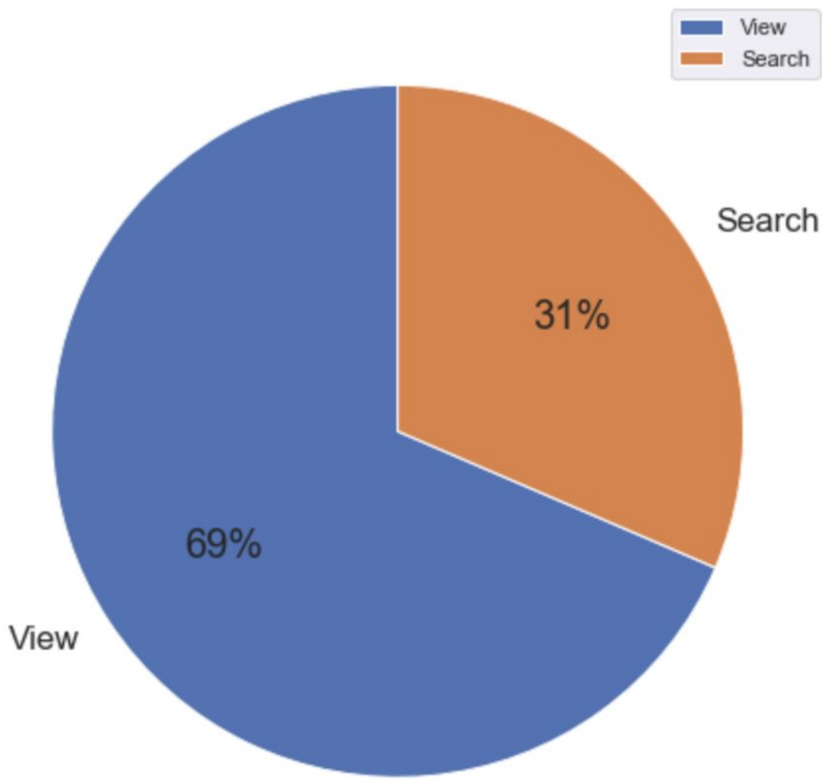
```
def sites(s):
    if (s['site_category'] == '28905ebd') :
        return 'Facebook'
    elif (s['site_category'] == '0569f928'):
        return 'Google'
    elif (s['site_category'] == 'f028772b'):
        return 'Ask.com'
    elif (s['site_category'] == '50e219e0'):
        return 'Yahoo'
    elif (s['site_category'] == '3e814130'):
        return 'Bing'
    elif (s['site_category'] == '76b2941d'):
        return 'Facebook'
    elif (s['site_category'] == 'f66779e6'):
        return 'Google'
    elif (s['site_category'] == '335d28a8'):
        return 'Ask.com'
    elif (s['site_category'] == '72722551'):
        return 'Yahoo'
    elif (s['site_category'] == '75fa27f6'):
        return 'Bing'
    elif (s['site_category'] == '110ab22d'):
        return 'Facebook'
    elif (s['site_category'] == 'c0dd3be3'):
        return 'Google'
    elif (s['site_category'] == 'bcf865d9'):
        return 'Ask.com'
    elif (s['site_category'] == 'a818d37a'):
        return 'Yahoo'
    elif (s['site_category'] == '42a36e14'):
        return 'Bing'
    elif (s['site_category'] == 'e787de0e'):
        return 'Ask.com'
    elif (s['site_category'] == '5378d028'):
        return 'Facebook'
    else:
        return 'Others'
```

Unique users who order Snacks with different product_cat



An overview of users against the action(search/view):

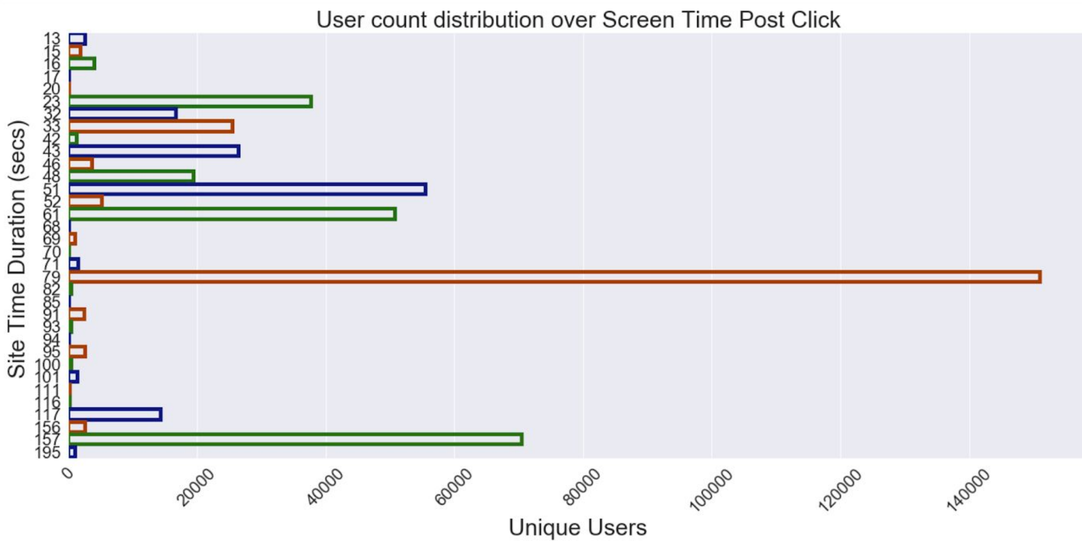
```
mpl.rcParams['font.size'] = 20.0
labels = ['View','Search']
plt.figure(figsize = (8, 8))
sizes = pd.value_counts(all_data.Action)
patches, texts, autotexts = plt.pie(sizes,
                                    labels=labels, autopct='%.0f%%',
                                    shadow=False, radius=1,startangle=90)
for t in texts:
    t.set_size('smaller')
plt.legend()
plt.show()
```



User count distribution over Screen Time Post Click:

In [294]:

```
plt.figure(figsize=(20,10))
sns.set(font_scale=2)
sns.countplot(y='C21',data=all_data,facecolor=(0,0,0,0),linewidth=5,edgecolor=sns.color_palette('dark',3))
sns.set(style="darkgrid")
plt.xlabel('Unique Users',fontsize=30)
plt.ylabel('Site Time Duration (secs) ',fontsize=30)
plt.xticks(rotation='45')
plt.title('User count distribution over Screen Time Post Click',fontsize=30)
plt.tight_layout()
```



Model Building:

Then hyper-parameter values are set to obtain the best results from our lightgbm classifier.

```python
MAX_LEAF = 64
MIN_DATA = 20
NUM_OF_TREES = 100
TREE_LEARNING_RATE = 0.15
EARLY_STOPPING_ROUNDS = 20
METRIC = "auc"
SIZE = "sample"
```

```python
params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'num_class': 1,
    'objective': "binary",
    'metric': METRIC,
    'num_leaves': MAX_LEAF,
    'min_data': MIN_DATA,
    'boost_from_average': True,
    #set it according to your cpu cores.
    'num_threads': 20,
    'feature_fraction': 0.8,
    'learning_rate': TREE_LEARNING_RATE,
}
```

After doing so, we try to understand our randomly generated sample dataset and the different data types present in it.

	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	...	C15	C16	C17	C18	C19	C20
0	1000009418151094400	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	320	50	1722	0	35	-1
1	10000169349117863936	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	320	50	1722	0	35	100084
2	10000371904215119872	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	320	50	1722	0	35	100084
3	10000640724480839680	0	14102100	1005	0	1fbe01fe	f3845767	28905ebd	ecad2386	7801e8d9	...	320	50	1722	0	35	100084
4	10000679056417042432	0	14102100	1005	1	fe8cc448	9166c161	0569f928	ecad2386	7801e8d9	...	320	50	2161	0	35	-1
5	10000720757801103360	0	14102100	1005	0	d6137915	bb1ef334	f028772b	ecad2386	7801e8d9	...	320	50	1899	0	431	100077
6	10000724729988544512	0	14102100	1005	0	8fda644b	25d4cfcd	f028772b	ecad2386	7801e8d9	...	320	50	2333	0	39	-1

We then split our cleaned dataset into a train, test and validation part to check the models performance on each one.

```python
# split data to 3 sets
length = len(all_data)
train_data = all_data.loc[:0.8*length-1]
valid_data = all_data.loc[0.8*length:0.9*length-1]
test_data = all_data.loc[0.9*length:]
```

OrdinalEncoding is also implemented to deal with the various categories within our features.

```python
nume_cols = ['Action','id','hour','banner_pos','device_type','device_conn_type','C14','C15', 'C16','C17','C18','C19','C20','C
cate_cols=['Site Name','site_id','site_domain','site_category','product_cat','app_id','app_domain','app_category','device_id'
label_col='click'
ord_encoder = ce.ordinal.OrdinalEncoder(cols=cate_cols)

def encode_csv(df, encoder, label_col, typ='fit'):
    if typ == 'fit':
        df = encoder.fit_transform(df)
    else:
        df = encoder.transform(df)
    y = df[label_col].values
    del df[label_col]
    return df, y

train_x, train_y = encode_csv(train_data, ord_encoder, label_col)
valid_x, valid_y = encode_csv(valid_data, ord_encoder, label_col, 'transform')
test_x, test_y = encode_csv(test_data, ord_encoder, label_col, 'transform')

print('Train Data Shape: X: {trn_x_shape}; Y: {trn_y_shape}.\nValid Data Shape: X: {vld_x_shape}; Y: {vld_y_shape}.\nTest Dat
      .format(trn_x_shape=train_x.shape,
              trn_y_shape=train_y.shape,
              vld_x_shape=valid_x.shape,
              vld_y_shape=valid_y.shape,
              tst_x_shape=test_x.shape,
              tst_y_shape=test_y.shape,))
train_x.head()
```

```
Train Data Shape: X: (400000, 26); Y: (400000,).
Valid Data Shape: X: (50000, 26); Y: (50000,).
Test Data Shape: X: (50000, 26); Y: (50000,).
```

Then we run our model on the train, validation and test datasets to obtain a AUC value of .77 on an average.

```
lgb_train = lgb.Dataset(train_x, train_y.reshape(-1), params=params, categorical_feature=cate_cols)
lgb_valid = lgb.Dataset(valid_x, valid_y.reshape(-1), reference=lgb_train, categorical_feature=cate_cols)
lgb_test = lgb.Dataset(test_x, test_y.reshape(-1), reference=lgb_train, categorical_feature=cate_cols)
lgb_model = lgb.train(params,
                      lgb_train,
                      num_boost_round=NUM_OF_TREES,
                      early_stopping_rounds=EARLY_STOPPING_ROUNDS,
                      valid_sets=lgb_valid,
                      categorical_feature=cate_cols)
```

```
[1]	valid_0's auc: 0.743381
Training until validation scores don't improve for 20 rounds.
[2]	valid_0's auc: 0.75896
[3]	valid_0's auc: 0.762684
[4]	valid_0's auc: 0.763949
[5]	valid_0's auc: 0.766595
[6]	valid_0's auc: 0.767837
[7]	valid_0's auc: 0.769191
[8]	valid_0's auc: 0.769651
[9]	valid_0's auc: 0.770165
[10]	valid_0's auc: 0.771039
[11]	valid_0's auc: 0.772212
[12]	valid_0's auc: 0.772686
[13]	valid_0's auc: 0.773054
[14]	valid_0's auc: 0.773809
[15]	valid_0's auc: 0.774236
[16]	valid_0's auc: 0.774422
[17]	valid_0's auc: 0.774793
[18]	valid_0's auc: 0.775481
[19]	valid_0's auc: 0.775551
[20]	valid_0's auc: 0.775533
```

Evaluation:

We then test the evaluation of the model on the test data. We attain an AUC score of .71 on the test data which is pretty reasonable as we got a score of 0.72 on validation dataset and .70 training data

```
In [300]:  test_preds = lgb_model.predict(test_x)
           auc = roc_auc_score(np.asarray(test_y.reshape(-1)), np.asarray(test_preds))
           logloss = log_loss(np.asarray(test_y.reshape(-1)), np.asarray(test_preds), eps=1e-12)
           res_basic = {"auc": auc, "logloss": logloss}
           print(res_basic)
           pm.record("res_basic", res_basic)

           {'auc': 0.7473918241332705, 'logloss': 0.3905822684604061}
```

Optimization:

Now that we have evaluated our model, we will try to optimize it using NumEncoder(). After running NumEncoder() we can verify that there isn't much difference in the results of AUC in regards with test data.

```
In [301]:  label_col = 'click'
           num_encoder = lgb_utils.NumEncoder(cate_cols, nume_cols, label_col)
           train_x, train_y = num_encoder.fit_transform(train_data)
           valid_x, valid_y = num_encoder.transform(valid_data)
           test_x, test_y = num_encoder.transform(test_data)
           del num_encoder
           print('Train Data Shape: X: {trn_x_shape}; Y: {trn_y_shape}.\nValid Data Shape: X: {vld_x_shape}; Y: {vld_y_shape}.\nTest Dat
                 .format(trn_x_shape=train_x.shape,
                         trn_y_shape=train_y.shape,
                         vld_x_shape=valid_x.shape,
                         vld_y_shape=valid_y.shape,
                         tst_x_shape=test_x.shape,
                         tst_y_shape=test_y.shape,))
```

```
  9%| | 1/11 [00:02<00:22,  2.24s/it]
 18%| | 2/11 [00:03<00:17,  1.91s/it]
 27%| | 3/11 [00:04<00:14,  1.78s/it]
 36%| | 4/11 [00:06<00:11,  1.61s/it]
 45%| | 5/11 [00:07<00:09,  1.64s/it]
 55%| | 6/11 [00:09<00:07,  1.53s/it]
 64%| | 7/11 [00:10<00:05,  1.38s/it]
 73%| | 8/11 [00:11<00:03,  1.28s/it]
 82%| | 9/11 [00:12<00:02,  1.19s/it]
 91%| | 10/11 [00:13<00:01,  1.17s/it]
100%| | 11/11 [00:14<00:00,  1.29s/it]

  0%| | 0/36 [00:00<?, ?it/s]
  3%| | 1/36 [00:00<00:06,  5.03it/s]
  6%| | 2/36 [00:00<00:07,  4.69it/s]
  8%| | 3/36 [00:00<00:07,  4.33it/s]
 11%| | 4/36 [00:00<00:06,  4.85it/s]
 14%| | 5/36 [00:01<00:05,  5.23it/s]
 17%| | 6/36 [00:01<00:05,  5.49it/s]
 19%| | 7/36 [00:01<00:05,  5.60it/s]
```

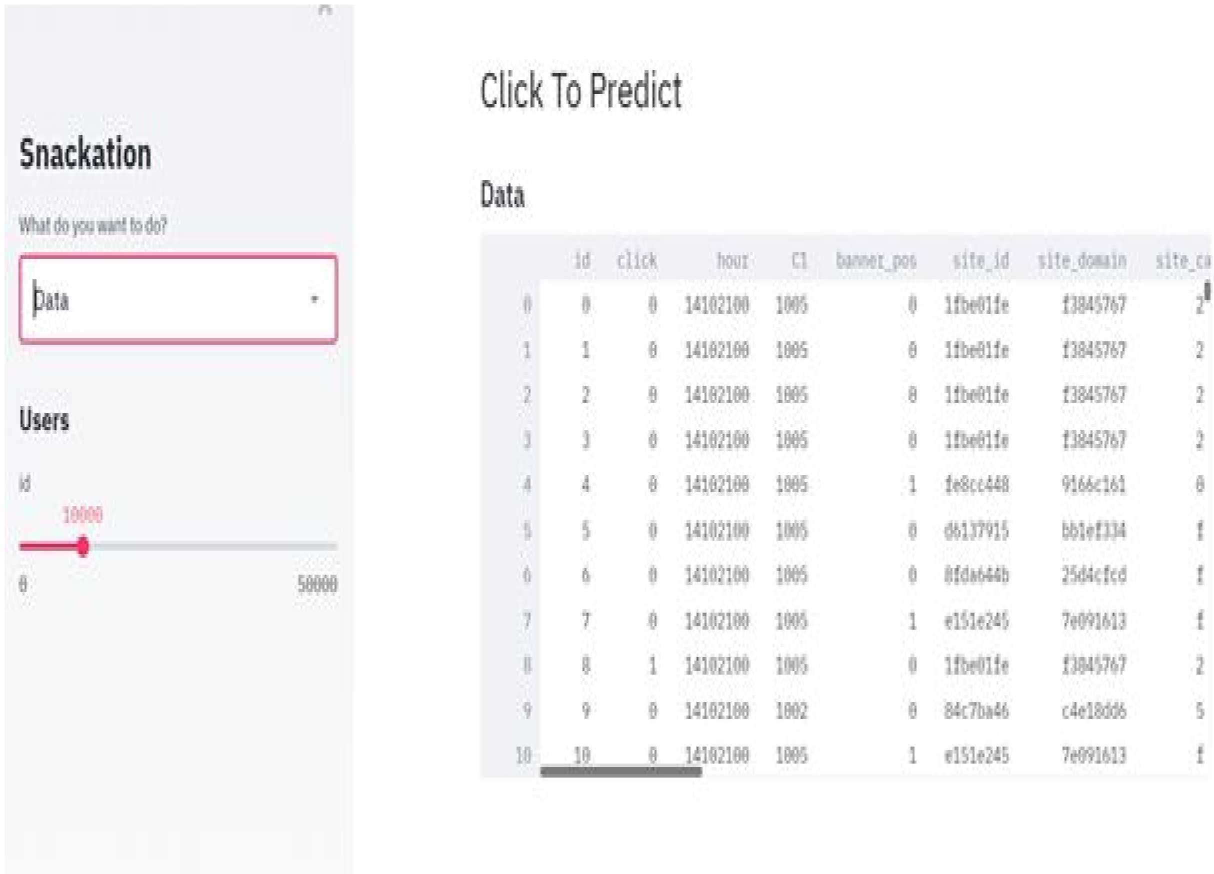
```
In [303]:  test_preds = lgb_model.predict(test_x)
           auc = roc_auc_score(np.asarray(test_y.reshape(-1)), np.asarray(test_preds))
           logloss = log_loss(np.asarray(test_y.reshape(-1)), np.asarray(test_preds), eps=1e-12)
           res_optim = {"auc": auc, "logloss": logloss}
           print(res_optim)
           pm.record("res_optim", res_optim)

           {'auc': 0.7469770267979412, 'logloss': 0.390092487563415}
```

STREAMLIT

Streamlit is an open source app framework which helps us to build beautiful web apps for machine learning and data science. To start with Streamlit, we installed it using command - **pip install streamlit,** import it, write the code and run the script with Streamlit run file.py.

Below is the implementation of streamlit with respect to our dataset.

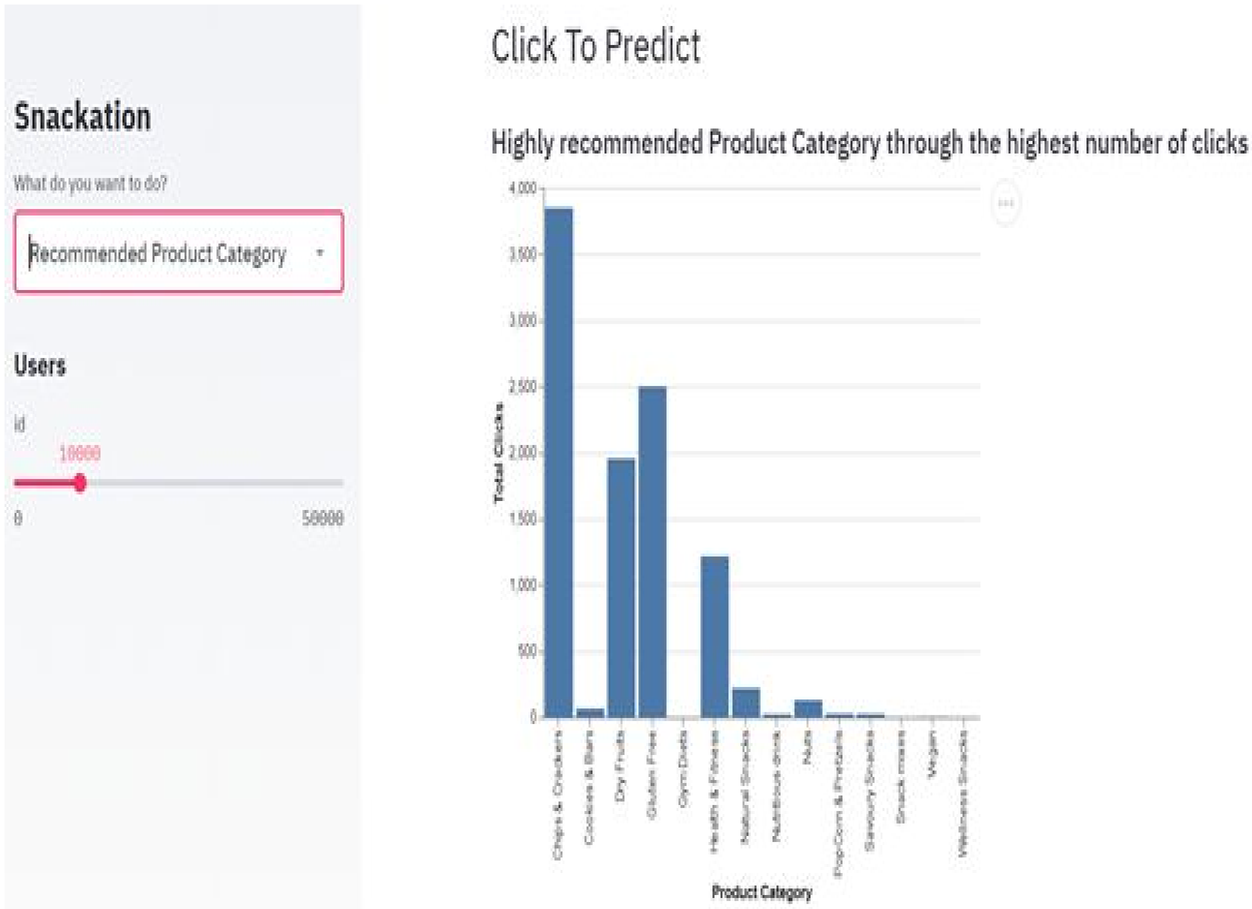
1. Data



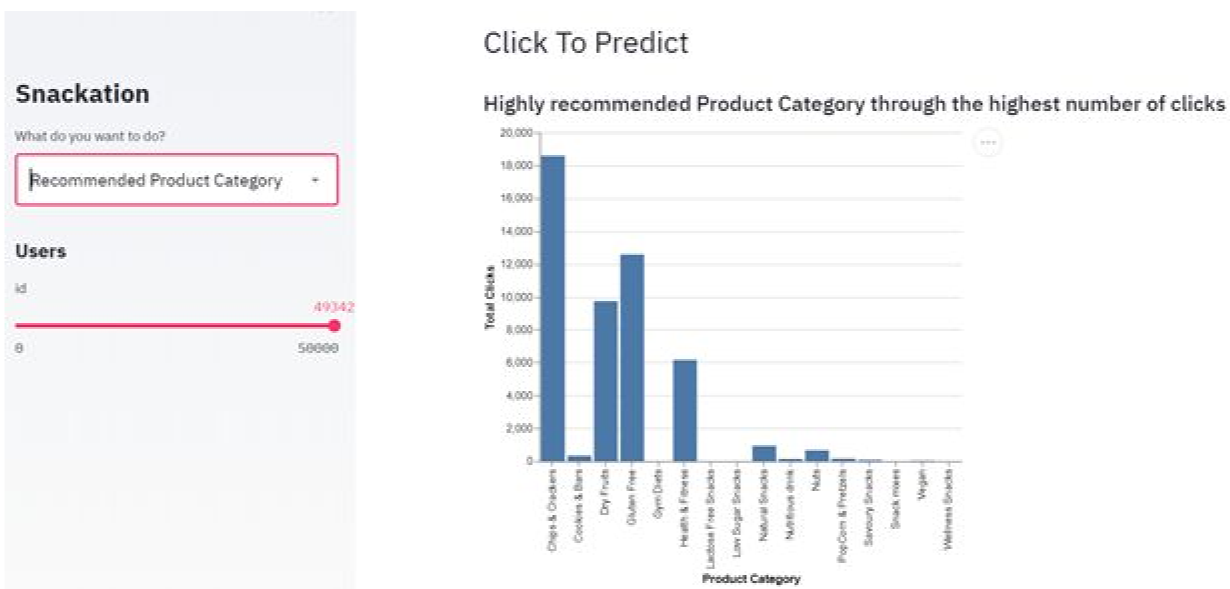
	id	click	hour	C1	banner_pos	site_id	site_domain	site_ca
0	0	0	14102100	1005	0	1fbe01fe	f3845767	2
1	1	0	14102100	1005	0	1fbe01fe	f3845767	2
2	2	0	14102100	1005	0	1fbe01fe	f3845767	2
3	3	0	14102100	1005	0	1fbe01fe	f3845767	2
4	4	0	14102100	1005	1	fe8cc448	9166c161	0
5	5	0	14102100	1005	0	d6137915	bb1ef334	f
6	6	0	14102100	1005	0	8fda644b	25d4cfcd	f
7	7	0	14102100	1005	1	e151e245	7e091613	f
8	8	1	14102100	1005	0	1fbe01fe	f3845767	2
9	9	0	14102100	1002	0	84c7ba46	c4e18dd6	5
10	10	0	14102100	1005	1	e151e245	7e091613	f

2. Recommended product category - Highly recommended product category through the number of clicks from the users(user interaction). Chips & Crackers is the first most clicked product from the users followed by Gluten-free, dry fruits and Health & Fitness products for the following set of users based on their clicks into their respective websites.

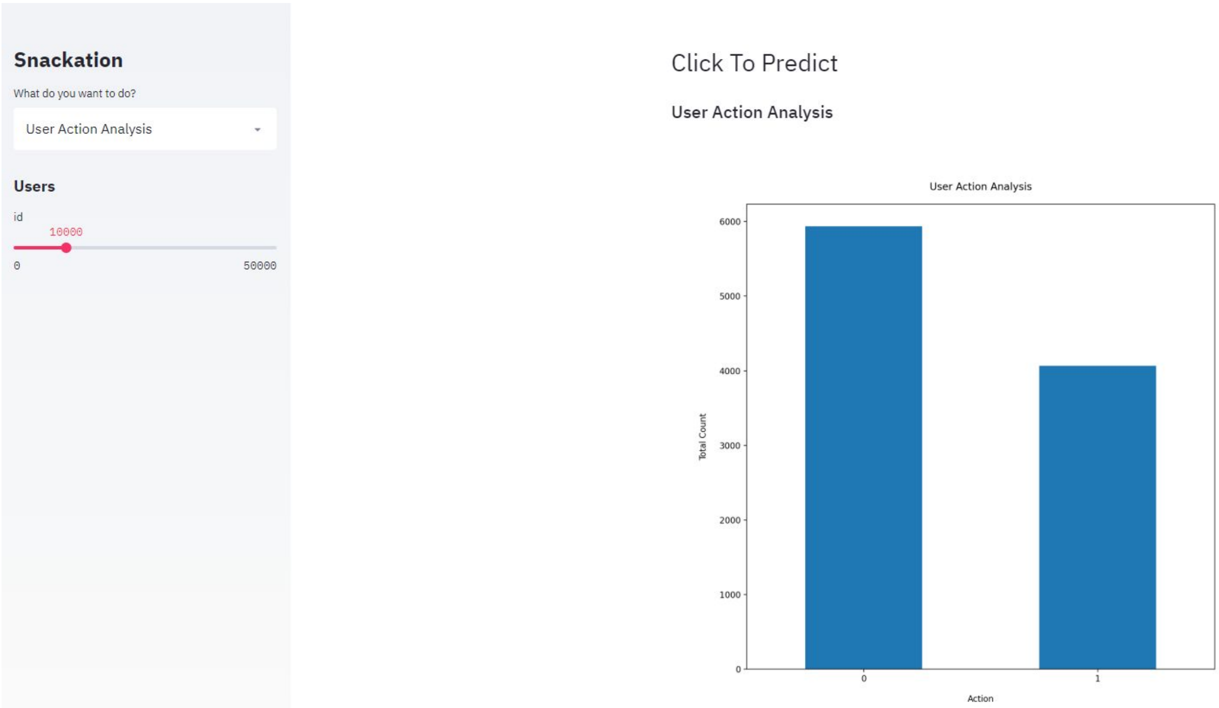
For 10,000 users,



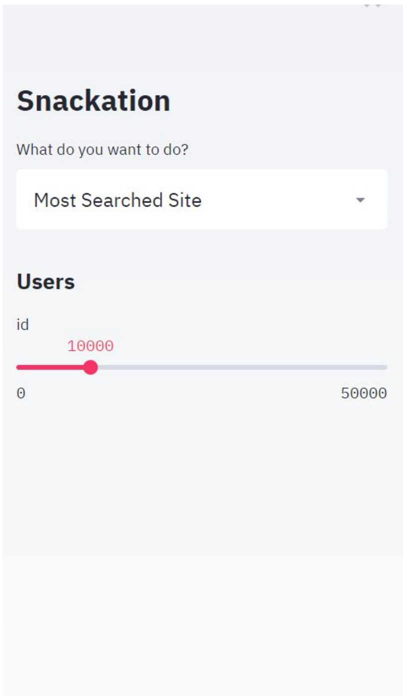
For 50,000 users,

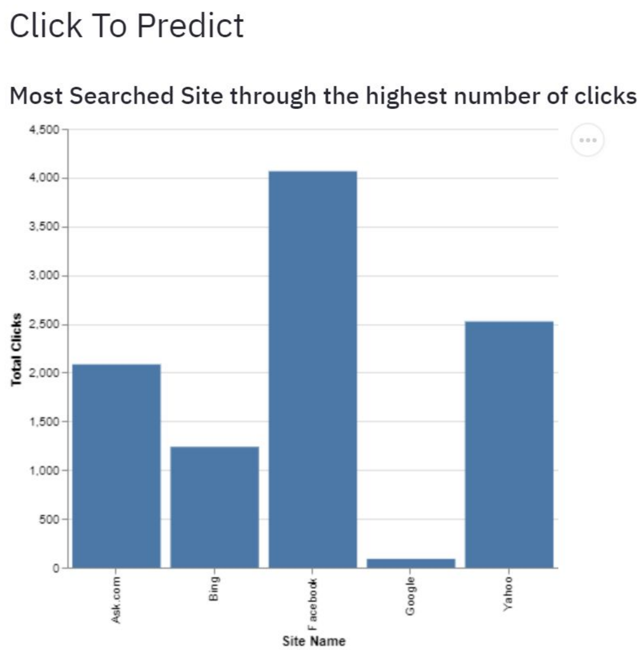


3. User Action Analysis - In the user action analysis, close to 60% of the users directly click into the article and 40% of the users use search engines to find the product category.



4. Most Searched Sites - Amongst the Searches that came in, the highest logging came in from Facebook, meaning that currently the Snackaction's main channel of Search is Facebook followed by yahoo, ask.com, Bing and Google.





References

1. https://medium.com/@teddywang0202/implicit-feedback-recommendation-system-i-intro-and-datasets-eda-eda16764602a
2. https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26
3. https://www.slideshare.net/MiguelFierro1/recommenders-repository-deep-dive
4. http://manishbarnwal.com/blog/2018/09/27/types_data_recommender_system/
5. https://www.kaggle.com/c/avazu-ctr-prediction/data