פרויקט מבוא למדעי הנתונים

יובל גולדנשטיין - 311552368
ישי שפירא - 203016217
נדב שטרן - 203016100

חיזוי לחיצות על פרסומות באפליקציות

1 Data Loading:

1 1. Data Loading

```
#upload data
df = pd.read_csv('ctr_dataset_train.csv')
df = pd.DataFrame(df)
print('df.columns:',df.columns)
print('df.shape:',df.shape)
```

```
df.columns: Index(['op_id', 'timestamp', 'state', 'user_isp', 'app_id', 'app_cat', 'banner_pos', 'manufacturer', 'device_mod
el', 'device_version', 'device_height', 'device_width', 'resolution', 'clicked'], dtype='object')
df.shape: (2519056, 14)
```

```
df.describe(include='all')
```

[3]:

	op_id	timestamp	state	user_isp	app_id	app_cat	banner_pos	manufacturer	device_model	device_version	de
count	2519056	2519056.000000	2519056	2519024	2519056	2519056	2519056	1748114	1748210	2519056	2
unique	2519056	nan	51	3316	14315	46	9	251	2256	16	
top	b685b440-d40e-4c98-afe4-60a937033894	nan	Texas	T-Mobile USA	mc8dtso	unknown	top-left	Samsung	LM-X210(G)	7.0	
freq	1	nan	327255	589526	43385	596348	304202	690996	88524	867074	

Notice the unique row.
We have a number of columns with high cardinality.
Also a column(resolution) with only 1 unique value - will remove.
The Data types need to be fixed.

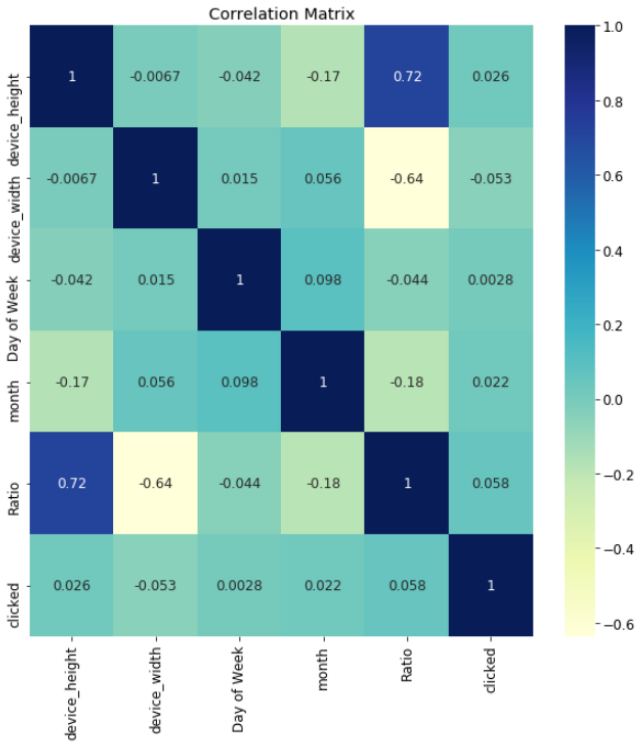
2 Data Exploration:

We have missing value.

```
df.isnull().sum()
```

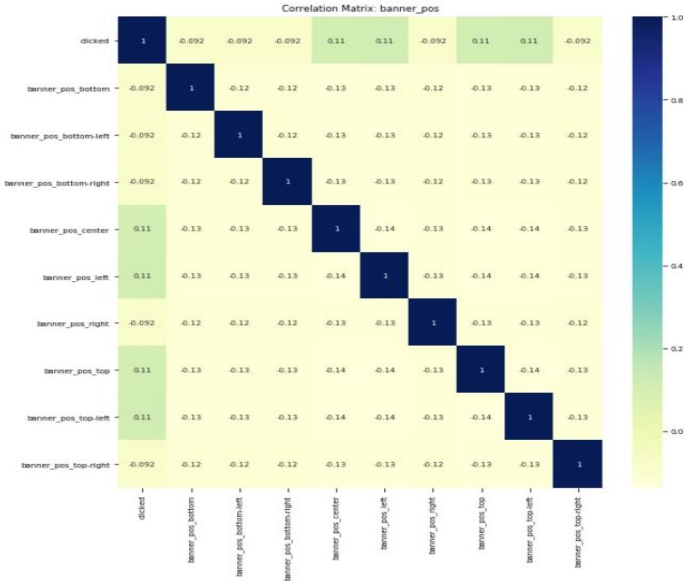
```
op_id                0
timestamp            0
state                0
user_isp            32
app_id               0
app_cat              0
banner_pos           0
manufacturer    770942
device_model    770846
device_version       0
device_height        0
device_width         0
resolution           0
clicked              0
dtype: int64
```

Corr Plot of the numeric data:



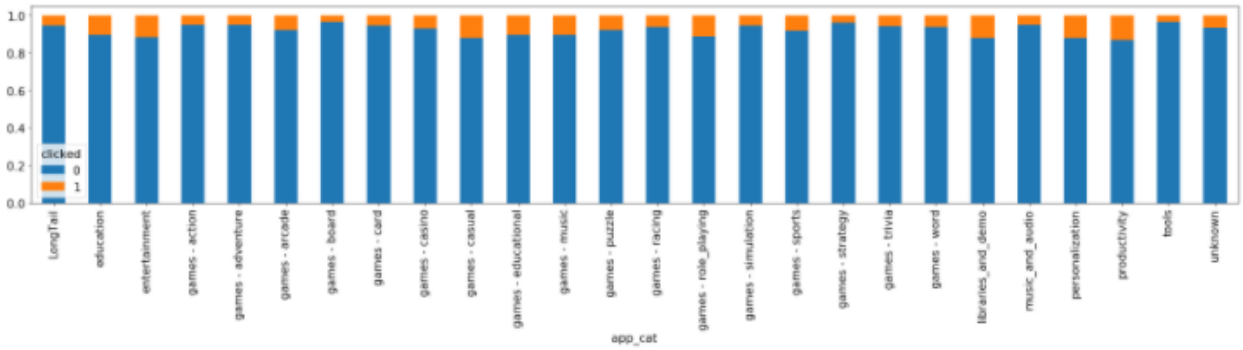
This corr plot contains columns that we added so you may not recognize them yet:)

There is high correlation as expected between Ratio, height and Ratio,width.
The rest of the params are close to 0 therefore have a low pearson correlation.
Most of the data is categorical - once we do onehotencoding to the features we reduce the cardinality for we can check for further potential correlations.
For example banner_pos: no significant corr with clicked. Similarly the other columns.

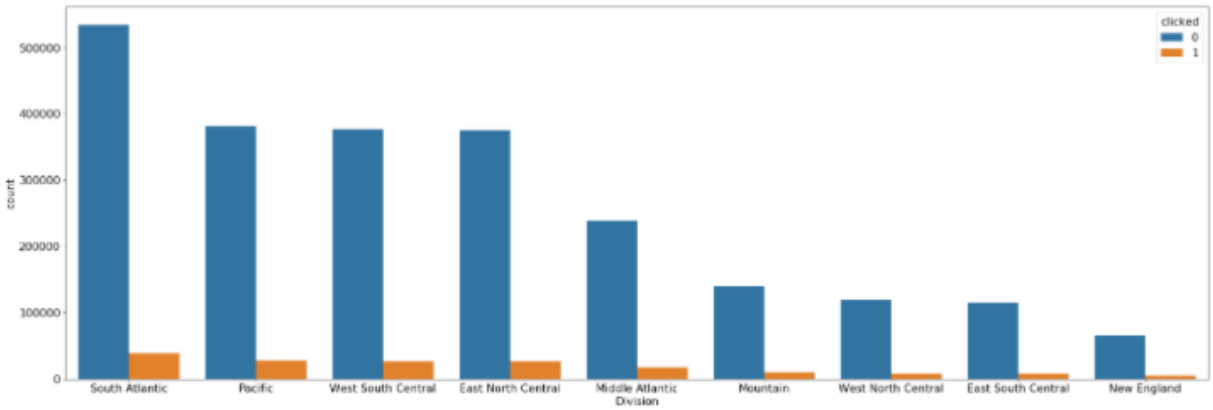


Cardinality reduction and plots:

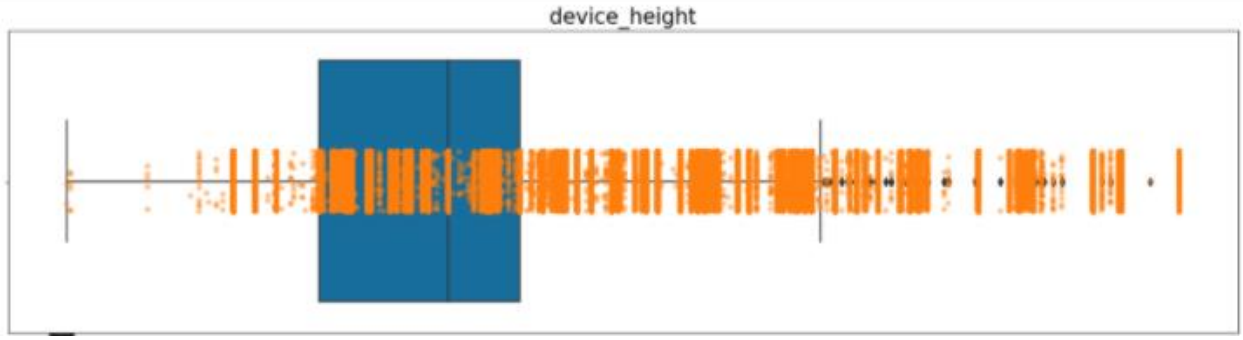
For each column with high cardinality we plotted 2 main graphs.
A 100% fill bar chart split by the label providing **relative** insights.



And a regular bar chart split by label providing **size** insights



For the numeric data with ran boxplots:
Identifying of outliers and the 'normal' behaviour - also showing which are clicked and which are not.



In a large number of columns we have high cardinality.
We reduced the number of unique values based on the ratio between the number of rows and number of unique values.
For example: in manufacturer

~90% of the categories are responsible for ~0.4% of the data -> we will combine them all to one category, 'LongTail'

```
]:  #get all the values that have a small number of uses in order to coombine
    x = df['manufacturer'][df['manufacturer'].notna()==True].value_counts().sort_values(ascending=False)
    x = x[df['manufacturer'][df['manufacturer'].notna()==True].value_counts()<1000]
    #Locate the values in the combine list and change them to longTail
    df.loc[df['manufacturer'].isin(x.keys()) , "manufacturer"] = 'LongTail'
    df['manufacturer'][df['manufacturer'] =='LongTail']
    df['manufacturer'] = df['manufacturer'].astype('category')
```

Did the same for the rest of the column with high cardinality. Significantly reducing it to a manageable number of categories while barely changing 1% of the data.

Removing columns:

- Based on the correlation and meaning we removed height and width and left Ratio.
- Resolution - removing this column as it has one unique value therefore has 0 cardinality and will not provide any information.
- Op_id - removing this as it had too high cardinality and provides no additional information - one per row.
- Removed Timestamp once we created Day of week, weekend and Month.
- Device_model and app_id have very high cardinality - removed as well(checked scores with and without,they did not provide insights).

The data is highly imbalanced.
We have ~7% positive values.
We used SMOTE to balance the data as we have mostly categorical data Adasyn didn't work well with this DF.
In addition we ran Scale_Pos_weight in the model to balance the data.
Using the cost sensitive method.
Bottom line we had a combination of cost sensitive in the model and SMOTE on the data.
In Smote we balanced to a fifth of the positive number. (sampling_strategy=0.2)

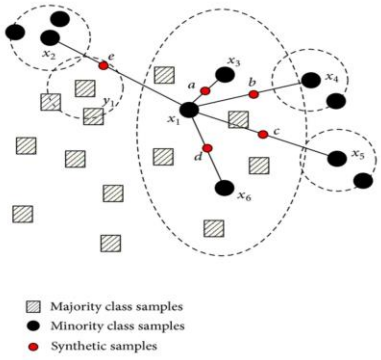
When comparing results with and without balancing we see improvements in all 3 models:

Model	With Smote	accuracy	auc	precision	precision_weighted	recall	recall_weighted	f1	f1_weighted
XGB	No	0.78165	0.76831	0.20437	0.92416	0.75285	0.78165	0.32147	0.83221
LGBM	No	0.80513	0.74124	0.20822	0.91933	0.66730	0.80513	0.31740	0.84771
ANN	No	0.93221	0.50032	0.5	0.90295	0.00069	0.93221	0.00139	0.89960
XGB	Yes	0.83404	0.72877	0.23079	0.91644	0.60674	0.83404	0.33438	0.86598
LGBM	Yes	0.83901	0.70888	0.22442	0.91371	0.55831	0.83901	0.32015	0.86874
ANN	Yes	0.89556	0.58955	0.23766	0.89496	0.23472	0.89556	0.23618	0.89526

SMOTE:
An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.
Specifically, a random example from the minority class is first chosen. Then k of the nearest

neighbors for that example are found. A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

The approach is effective because new synthetic examples from the minority class are created that are plausible, that is, are relatively close in feature space to existing examples from the minority class.
A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes.



3 Missing values:

We identified we have missing values in 3 columns.

```
df.isnull().sum()
```

```
timestamp              0
state                  0
user_isp              32
app_id                 0
app_cat                0
banner_pos             0
manufacturer      770942
device_model      770846
device_version         0
device_height          0
device_width           0
clicked                0
dtype: int64
```

Filling NA for Manufacturer based on the assumption that we will have unique manufacturer per combination below

df["combined"] = df["device_version"].astype(str).str.cat(df[['device_height','device_width']].astype(str), sep="_")

75091 missing values can be accurately populated!

We used the mapping we created with the combination column and filled over 75K missing values with accurate values.
For the rest we filled them with the value -'unknown'.
For user_isp we we have only 32 missing values - filled with the most common category.

For device model the combine column did not have as many rows to fill therefore we skipped the process for this column and filled it with ‘Unknown’ where we had NA.

```
def MissingData(df):
    df['user_isp'].fillna(df['user_isp'].value_counts().index[0], inplace=True)
    df['device_model'].fillna('Unknown', inplace=True)
    df['manufacturer'].fillna('Unknown', inplace=True)
    return df
```

After runnings:

```
df.isnull().sum()
```

```
timestamp         0
state             0
user_isp          0
app_id            0
app_cat           0
banner_pos        0
manufacturer      0
device_model      0
device_version    0
device_height     0
device_width      0
clicked           0
combined          0
Day of Week       0
weekend           0
month             0
Ratio             0
Region            0
Division          0
dtype: int64
```

4 Feature engineering:

We added a number of columns.

Ratio - the ratio between height and width.(f['Ratio'] = df['device_height'] / df['device_width']) This is one of the top values according to shap feature importance. This column provides the connection between the 2 values which makes sense will have a larger impact than just height / width on its own.

Day of week - day of week from date time. Monday to sunday. Assuming some days have higher ctr.
Month - month from date time - perhap there is an improvement we can identify on a month level(holidays/summer vacation)
Is weekend - is saturday or sunday. - assuming more people use apps in the weekend/have time to look at ads.
Region - joined a state data set to derive more info from the state column. In this case the region of the state. A way to reduce cardinality.
Division - the division of the state. - a way to reduce cardinality in State.

Getting Region and Division of the states

```
#upload data
df_states_Region = pd.read_csv('UsStateToRegion.csv')
df_states_Region = pd.DataFrame(df_states_Region)
df_states_Region=df_states_Region.rename(columns={"State": "state"
df_states_Region
```

44]:

	state	State Code	Region	Division
0	Alaska	AK	West	Pacific
1	Alabama	AL	South	East South Central
2	Arkansas	AR	South	West South Central
3	Arizona	AZ	West	Mountain

5 Normalization:

We normalized the numeric features with the rescaling approach, we normalized them to be between zero and one, using the function below:

```
# normalizing feature values from zero to one
def NormalizingFromZeroToOne(column_):
    Max= column_.max()
    Min= column_.min()
    MinMax= Max-Min
    res = (column_-Min)/(MinMax)
    return res.astype('float16')
```

We have chosen this normalization approach because one of the models we tested is an artificial neural network (ANN) and in this kind of models the weights are calculated wrong if there are features in different scales.

We also encoded the categorical features to “one hot” in two of the models we tested (ANN and XGBOOST) because they don’t support categorical features.
We did it using the function below:

```
def GetOneHotEnc(df_, column_):
    enc = OneHotEncoder(handle_unknown='ignore')
    matrix = enc.fit_transform(df_[column_].to_frame())
    column_names = enc.get_feature_names([column_])
    res = pd.DataFrame(matrix.todense(), columns= column_names).astype('int16')
    return res
```

* We cast all the features to be “float16” and “int16” in order to improve computing performance.

6 Training:

Train, val and test split

We split the train file data into 3 parts.
Train, test and validation.

```
# split the data to train(70%), val(15%) and test(15%)
x_train_val_xgb, x_test_xgb, y_train_val_xgb, y_test_xgb =
train_test_split(data_xgb, target_xgb, test_size=0.15, random_state=RANDOM_STATE)

x_train_xgb, x_val_xgb, y_train_xgb, y_val_xgb =
train_test_split(x_train_val_xgb, y_train_val_xgb, test_size=0.20, random_state=RANDOM_STATE)
```

We trained 3 models:

- XGBOOST
- LGBM
- ANN

For each model we ran a random search.
In addition we added the weighted scores as the data is highly imbalanced.
Overall the accuracy is relatively high, though is an imbalance data set this is an insignificant measure.

The best results for XGB:

```
{'accuracy': 0.8340392156862745,
 'auc': 0.7287718244766394,
 'boosting': 'gbtree',
 'f1': 0.3343818810946839,
 'f1_weighted': 0.8659827225391126,
 'learning_rate': 0.1632972863901326,
 'max_depth': 6,
 'n_estimators': 214,
 'objective': 'binary:logistic',
 'precision': 0.23078593139383413,
 'precision_weighted': 0.9164401381219319,
 'recall': 0.6067351598173516,
 'recall_weighted': 0.8340392156862745,
 'reg_lambda': 0.8033674434562963}
```


The weighted scores are high >80% for recall precision and F1.
Our target mesure F1 has a best score of 86.6%.in a highly imbalance data set this is great.
Our Recall 83.4% - so we predict over 83% of the actual clicked values correctly.
Our Precision 91.6% - out of all our positive predictions we are correct more that 91% of the time. Very little noise.

The best results for LGBM:


```
{'accuracy': 0.8390117647058823,
 'auc': 0.7088834060192376,
 'boosting': 'dart',
 'f1': 0.32015103338632755,
 'f1_weighted': 0.8687367182603136,
 'learning_rate': 0.0725834727902253,
 'max_depth': 23,
 'metric': 'auc',
 'n_estimators': 653,
 'num_leaves': 50,
 'precision': 0.22442014348401476,
 'precision_weighted': 0.9137096634334414,
 'recall': 0.5583087853058395,
 'recall_weighted': 0.8390117647058823}
```


The weighted scores are high >83% for recall precision and F1.
Our target mesure F1 has a best score of 86.9%.in a highly imbalance data set this is great.
Our Recall 84% - so we predict over 83% of the actual clicked values correctly.
Our Precision 91.4% - out of all our positive predictions we are correct more that 91% of the time. Very little noise.

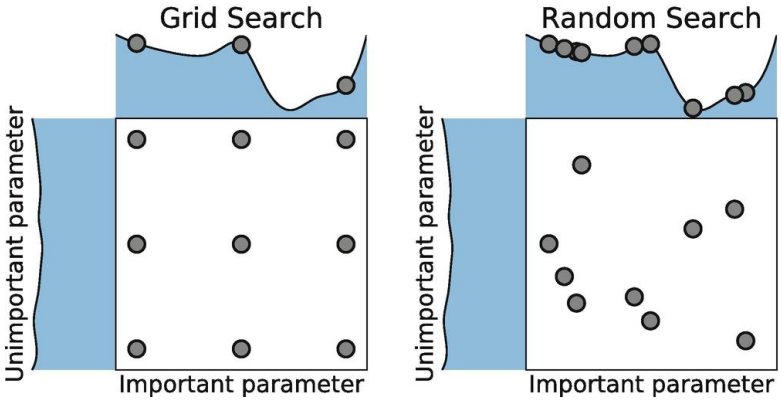
The best results for ANN:

```
{'# of layers': 3,
 '# of units per layer': 115,
 'Activation function': 'tanh',
 'Dropout': 0.07891923161534266,
 'Optimizer': 'Adadelta',
 'accuracy': 0.8955588235294117,
 'auc': 0.5895476055058965,
 'f1': 0.23617982361798237,
 'f1_weighted': 0.8952569815537927,
 'precision': 0.23766233766233766,
 'precision_weighted': 0.8949567724694513,
 'recall': 0.23471569046601112,
 'recall_weighted': 0.8955588235294117}
```

The weighted scores are high >89% for recall precision and F1.
Our target mesure F1 has a best score of 89.5%.in a highly imbalance data set this is great.
Our Recall 89.5% - so we predict over 89% of the actual clicked values correctly.
Our Precision 89.5% - out of all our positive predictions we are correct more that 89% of the time. Very little noise.

Hyperparameters tuning

We implemented random search with 30 iterations in each model, the reason we chose to random search over grid search is because we wanted to tune as many parameters as we can and it's more feasible in random search, the picture below explains pretty well the main advantage of random search over grid search:



In each model there are dozens of hyperparameters that can be tuned, we have investigated each model and focus on the most important parameters, see optimal parameters for each model in the previous section above.
We measured the model performance in each iteration by the F1 score which is:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

XGBOOST:

We have tuned the below parameters:

- Objective function - we checked 3 objective functions that fit to our case - binary classification: 'Binary:logistic', 'binary:logitraw' and 'binary:hinge'
- N_estimators - the number of boosting iterations
- Max_depth - Maximum depth of a tree
- Learning_rate - Step size shrinkage used in update to prevents overfitting
- Reg_lambda - L2 regularization term on weights
-

LGBM:

We have tuned the below parameters:

- Metric - we checked 3 metrics to be evaluated on the evaluation set: 'binary_logloss', 'binary_error' and 'auc'
- Boosting - we checked two boosters that relevant to binary classification: 'dart' and 'goss'
- N_estimators - the number of boosting iterations
- Max_depth - Maximum depth of a tree
- Learning_rate - Step size shrinkage used in update to prevents overfitting
- Num_leaves - max number of leaves in one tree

ANN:

We have tuned the below parameters:

- Optimizer - optimizers are the extended class, which include added information to train a specific model, we checked the following optimizer: 'SGD', 'Adam', 'Adamax', 'Adagrad', 'Adadelta', 'RMSprop' and 'Nadam'
- Activation_function - the activation function of a node defines the output of that node given an input or set of inputs we checked the following Activation functions: 'relu', 'linear', 'selu' and 'tanh'
- Num_layers - number of hidden layers in the network
- Num_units - number of neurons in each layer
- Dropout - useful for regularizing, number between zero and one, probability to prune unuseful neurons

The chosen Model:

The model we chose to move forward with is XGB.
As it had the highest unweighted scores for F1.
As we don't know the measure most important to you, we will go with the harsh assumption.

7 XAI:

What is shap:

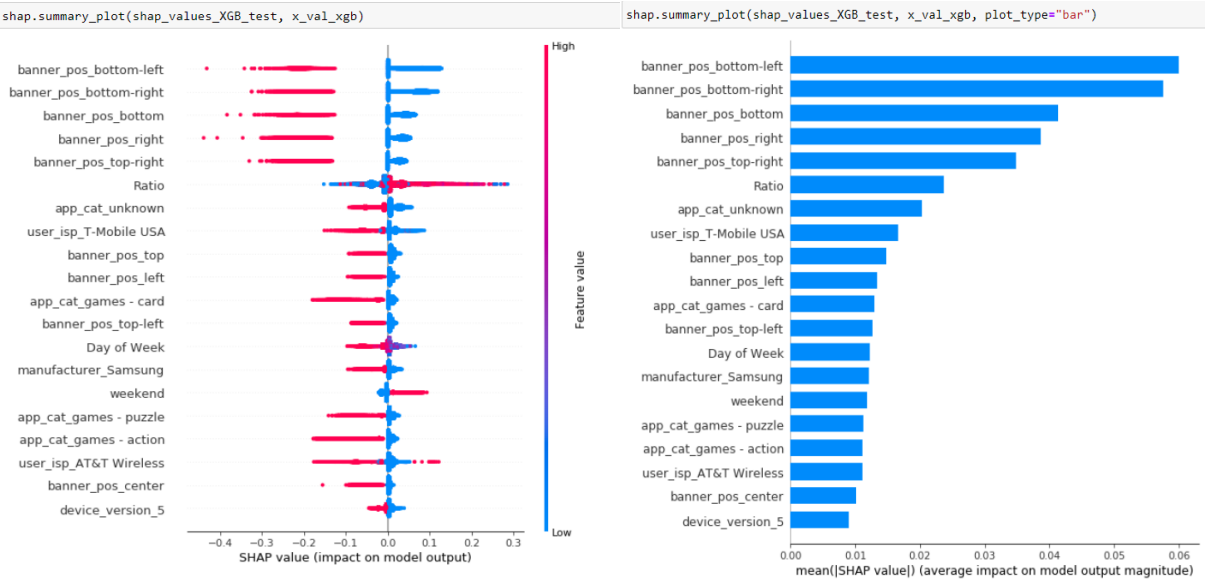
SHAP is a method in XAI. It is based on Game Theory, more specifically cooperative game theory. the player(fields) in this game are working together to achieve a common goal. The purpose of this method is to measure each player's contribution to the team's outcome.

This is done using the shapley value, a value that has a number of axioms, when these axioms are met we have what's called the fairness value. This is the most fair value to provide each player based on the contribution. The outcome here is finding the most valuable features for making a decision. both per instance and on average for all the data

Global interpretability

The summary plot displays the aggregated importance of each feature for all the data. So in shap we calculate the feature importance per row and bottom up display the overall average importance in the data to provide Global interpretability. Global interpretability enables us to identify the most important features and for high/low values how they impart the shapley value.

We can use this plot to identify interesting groups on the edges of the column plots and see easily how the values of this data impact the outcome.



Local interpretability

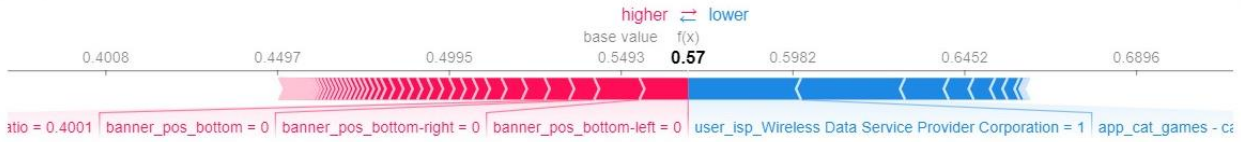
In these plots we will learn what most impacted this specific prediction to be click or not.

Providing us with insight on how the model works, and the ability to share this information with the users to try and prevent/improve the process. In addition the business expert can provide insights on whether or not a specific feature should not be so impactful or vise versa.

We randomly selected 3 rows in the data

```
#create 3 random numbers to have a force plot on
values = np.random.uniform(low=1, high=len(shap_values_XGB_test), size=(3,)).astype('int64')
print(values)
```

```
shap.force_plot(explainerXGB.expected_value,shap_values_XGB_test[values[0],:], x_val_xgb.iloc[values[0],:],link='logit')
```



What we learn from this chart is that the most impactful features are:

Banner_pos is not one of:bottom-left , bottom-right, bottom and the Rasio with a value of 0.4001 have high shapley scores and push the prediction up towards being clicked on.

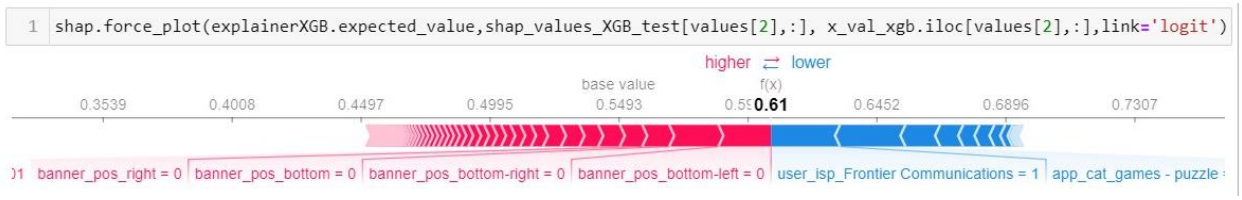
On the other end(in blue)user_isp = wireless Data Service Provider Corporation in the main value pushing the overall output value down.

```
shap.force_plot(explainerXGB.expected_value,shap_values_XGB_test[values[1],:], x_val_xgb.iloc[values[1],:],link='logit')
```

What we learn from this chart is that the most impactful features are(on the blue side):

banner_pos = bottom-right ,Rasio with a value of 0.4846 and User_isp = Verizon are the main value pushing the overall output value down to its very low score.

```
shap.force_plot(explainerXGB.expected_value,shap_values_XGB_test[values[2],:], x_val_xgb.iloc[values[2],:],link='logit')
```



What we learn from this chart is that the most impactful features are:

Banner_pos is not one of:bottom-left ,bottom-right, bottom,right. Have high shapley scores and push the prediction up towards being clicked on.

On the other end (in blue) user_isp = Frontier Communications and app_cat = Puzzle are the main value pushing the overall output value down.

8 Inference:

In order to create the predictions on the test file, we imported the train and test files separately, added an indicator column called “train_test”, in the train df we imputed this column with “train” and in the test df imputed this column with “test” and then we concatenated these two dataframes to one df.
We implemented again all the EDA processes on the merged df, splitted the merged df to train and test by the “train_test” column, resampled the train data with SMOTE and fit the best model with the train data.
Finally, we predicted the results on the test file and exported the results to “output_14.txt”.

9 Bonus:

An issue that might accor is the fact that the technology changes.
Meaning the phone sizes change, the companies that are manufacturing them are changing and the version becomes irrelevant after some time.
It might even be that the use of cellphones will change/stop at some point in time.
The model will stop providing predictions with any value when the real world data has drifted to entirely different values and meanings.
Especially in the world of advertisement, where there is a need to update the data in relatively small increments as trends change very quickly.

A potential solution to this would be having an automated monitoring system to check for data drift and alert the DS if there is a drift of over X% change.
Then have the DS update the model.
A more interesting approach would be to classify the data in a more generic manner.
For example - top manufacturer, max size min size and between.
In other words use the normalized values but also in the categorical values based on frequency of the values.
Make sure to test the model on data up to a certain point in time and evaluate on the rest, simulating the future. If you have 2 years of data and want to be confident in your results for up to 6 months into the future you can split the train test based on the date, first 1.5 years in the train the rest test - keeping chronological order.