<table>
<tr><th colspan="2">DATA FIELDS</th><th>OBJECTIVE</th></tr>
<tr><td>FULLVISITORID</td><td>A unique identifier for each user of the Google Merchandise Store.</td><td rowspan="7">IN THIS COMPETITION, WE A'RE CHALLENGED TO ANALYZE A GOOGLE MERCHANDISE STORE (ALSO KNOWN AS GSTORE, WHERE GOOGLE SWAG IS SOLD) CUSTOMER DATASET TO PREDICT REVENUE PER CUSTOMER.</td></tr>
<tr><td>CHANNELGROUPING</td><td>The channel via which the user came to the Store.</td></tr>
<tr><td>DATE</td><td>The date on which the user visited the Store.</td></tr>
<tr><td>DEVICE</td><td>The specifications for the device used to access the Store.</td></tr>
<tr><td>GEONETWORK</td><td>This section contains information about the geography of the user.</td></tr>
<tr><td>SESSIONID</td><td>A unique identifier for this visit to the store.</td></tr>
<tr><td>SOCIALENGAGEMENTTYPE</td><td>Engagement type, either "Socially Engaged" or "Not Socially Engaged".</td></tr>
<tr><td>TOTALS</td><td>This section contains aggregate values across the session.</td><th>FILES</th></tr>
<tr><td>TRAFFICSOURCE</td><td>This section contains information about the Traffic Source from which the session originated.</td><td rowspan="4">1. Train 2.Test</td></tr>
<tr><td>VISITID</td><td>An identifier for this session. This is part of the value usually stored as the _utmb cookie. This is only unique to the user. For a completely unique ID, you should use a combination of fullVisitorId and visitId.</td></tr>
<tr><td>VISITNUMBER</td><td>The session number for this user. If this is the first session, then this is set to 1.</td></tr>
<tr><td>VISITSTARTTIME</td><td>The timestamp (expressed as POSIX time).</td></tr>
</table>

Outline

1. **Data Investigation**
   A. Count the Missing Values
   B. Datatypes wise Columns Counts
   C. Investigate the target variable distribution
   D. Investigate Statistics of Data
   E. Correlation of dataset
2. **Remove Varible That Contrain Same Class**
3. **Categorical to Numeric Variable Conversion for Model Training(Label Encoding)**
4. **Bayesian Optimization for Best Parameter Tuning**
5. **Parameter as per Understanding of Model**

Code

In [2]:

```
def load_df(csv_path='../input/train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df

print(os.listdir("../input"))
```

```
['train.csv', 'sample_submission.csv', 'test.csv']
```

In [3]:

```
%%time
train_df = load_df(nrows=900000)
train_corr = train_df.copy()
test_df = load_df("../input/test.csv", nrows=1000000)
# train_df.to_csv("500000.csv", index=False)
```

```
Loaded train.csv. Shape: (900000, 55)
Loaded test.csv. Shape: (804684, 53)
CPU times: user 4min 55s, sys: 27.1 s, total: 5min 22s
Wall time: 5min 23s
```

1. Data Investigation

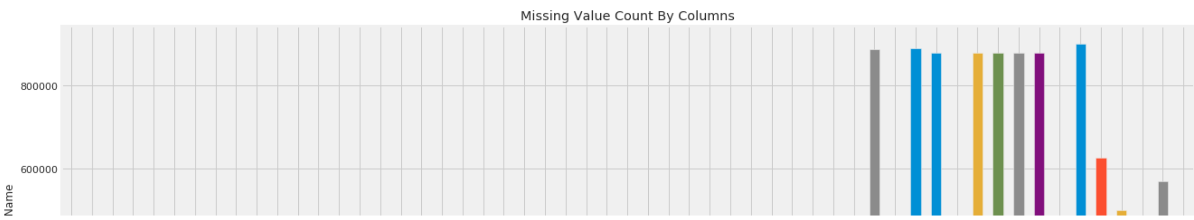
1.1 Count Missing Values

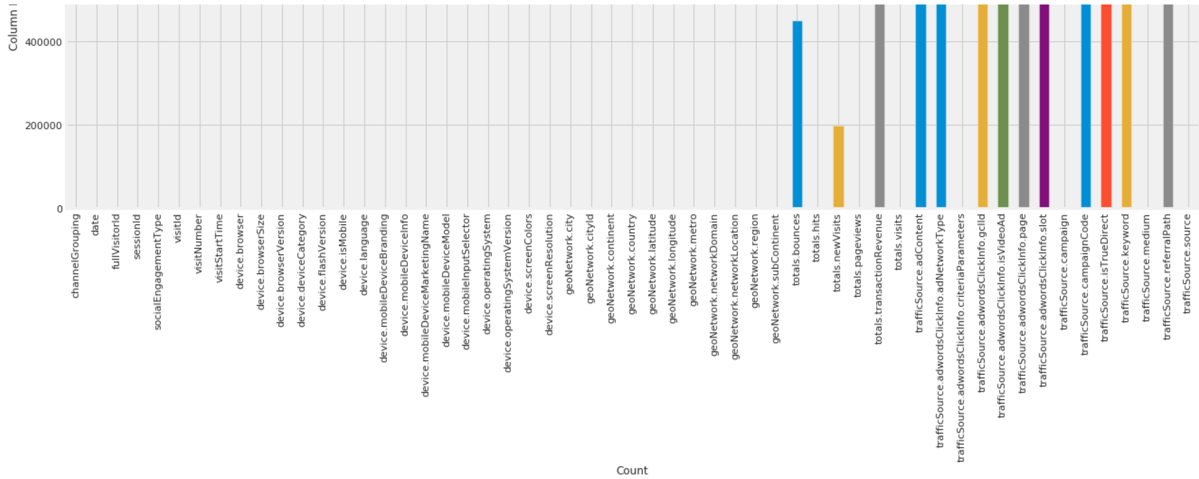
- We can see that so missing Values are to high.
- Missing data are a **common occurrence and can have a significant effect** on the conclusions that can be drawn from the **data.**

Code

Out[4]:

```
Text(0.5,1,'Missing Value Count By Columns')
```





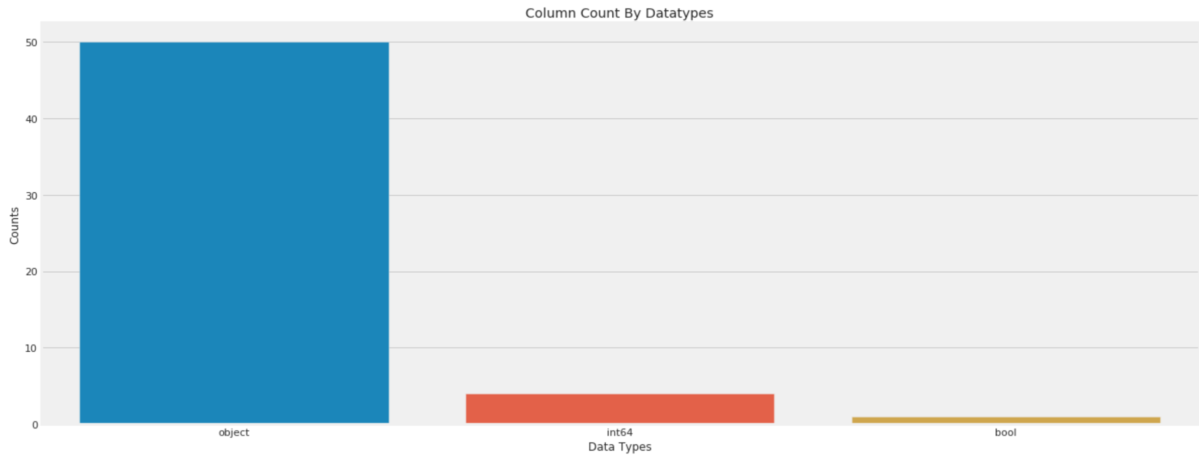
1.2 Count the Datatypes Columnwise

- We can see that most of columns datatype is **object**.
- The type of each attribute is important. Strings may need to be converted to floating point values or integers to represent categorical or ordinal values.
- we can get an idea of the types of attributes by peeking at the raw data. You can also list the data types used by the DataFrame to characterize each attribute using the dtypes property.

Code

Out[5]:

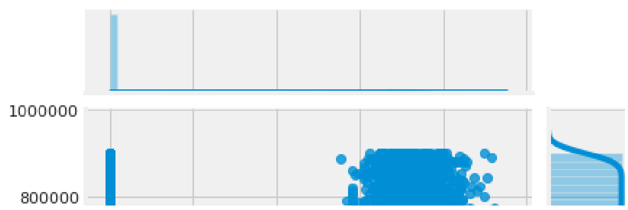
```
Text(0.5,1,'Column Count By Datatypes')
```

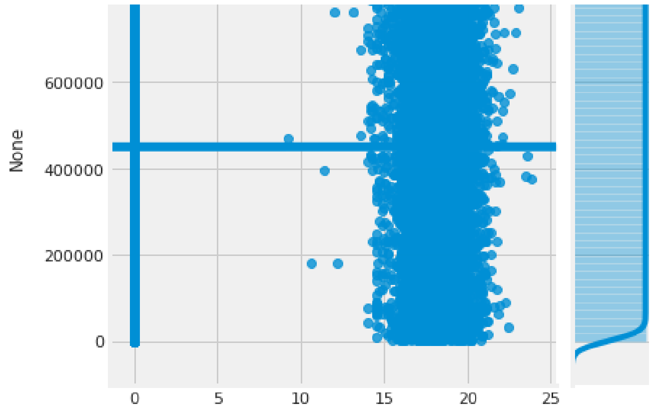


1.3 Investigate the Target Columns

Code

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a
non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` ins
tead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.arr
ay(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





1.4 Investigate the Statistics of Data

Descriptive statistics can give you great insight into the shape of each attribute. Often you can create more summaries than you have time to review. The describe() function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are:

- Count.
- Mean.
- Standard Deviation.
- Minimum Value.
- 25th Percentile.
- 50th Percentile (Median).
- 75th Percentile.
- Maximum Value.

Categorical Variable Columns Statistics

Code

Out[7]:

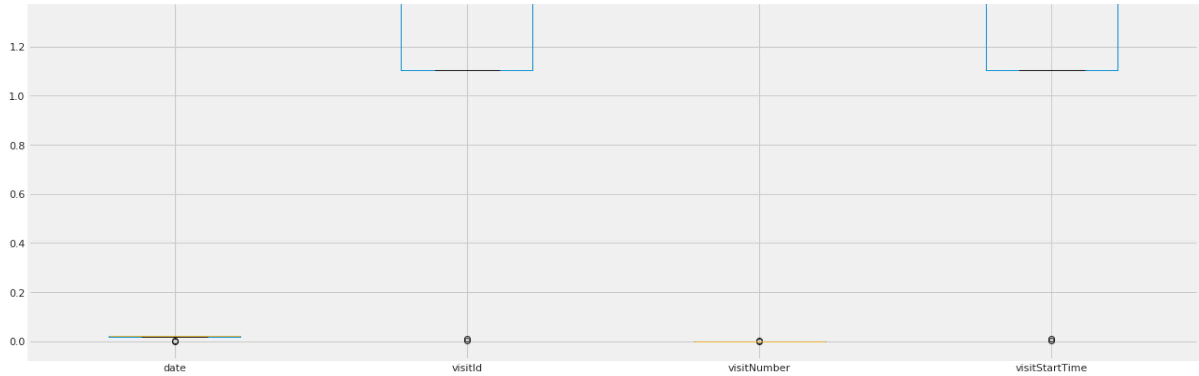
	channelGrouping	fullVisitorId	sessionId	socialEngagementType	device.brow
count	900000	900000	900000	900000	900000
unique	8	711659	899109	1	54
top	Organic Search	1957458976293878100	4373857412882577239_1474354675	Not Socially Engaged	Chrome
freq	379820	278	2	900000	617808

Numeric Statistics

Code

Out[8]:

	date	visitId	visitNumber	visitStartTime
count	9.000000e+05	9.000000e+05	900000.000000	9.000000e+05
mean	2.016587e+07	1.484991e+09	2.263774	1.484991e+09
std	4.698691e+03	9.023482e+06	9.278566	9.023482e+06
min	2.016080e+07	1.470035e+09	1.000000	1.470035e+09
25%	2.016103e+07	1.477541e+09	1.000000	1.477541e+09
50%	2.017011e+07	1.483975e+09	1.000000	1.483975e+09
75%	2.017042e+07	1.492743e+09	1.000000	1.492743e+09
max	2.017080e+07	1.501657e+09	395.000000	1.501657e+09



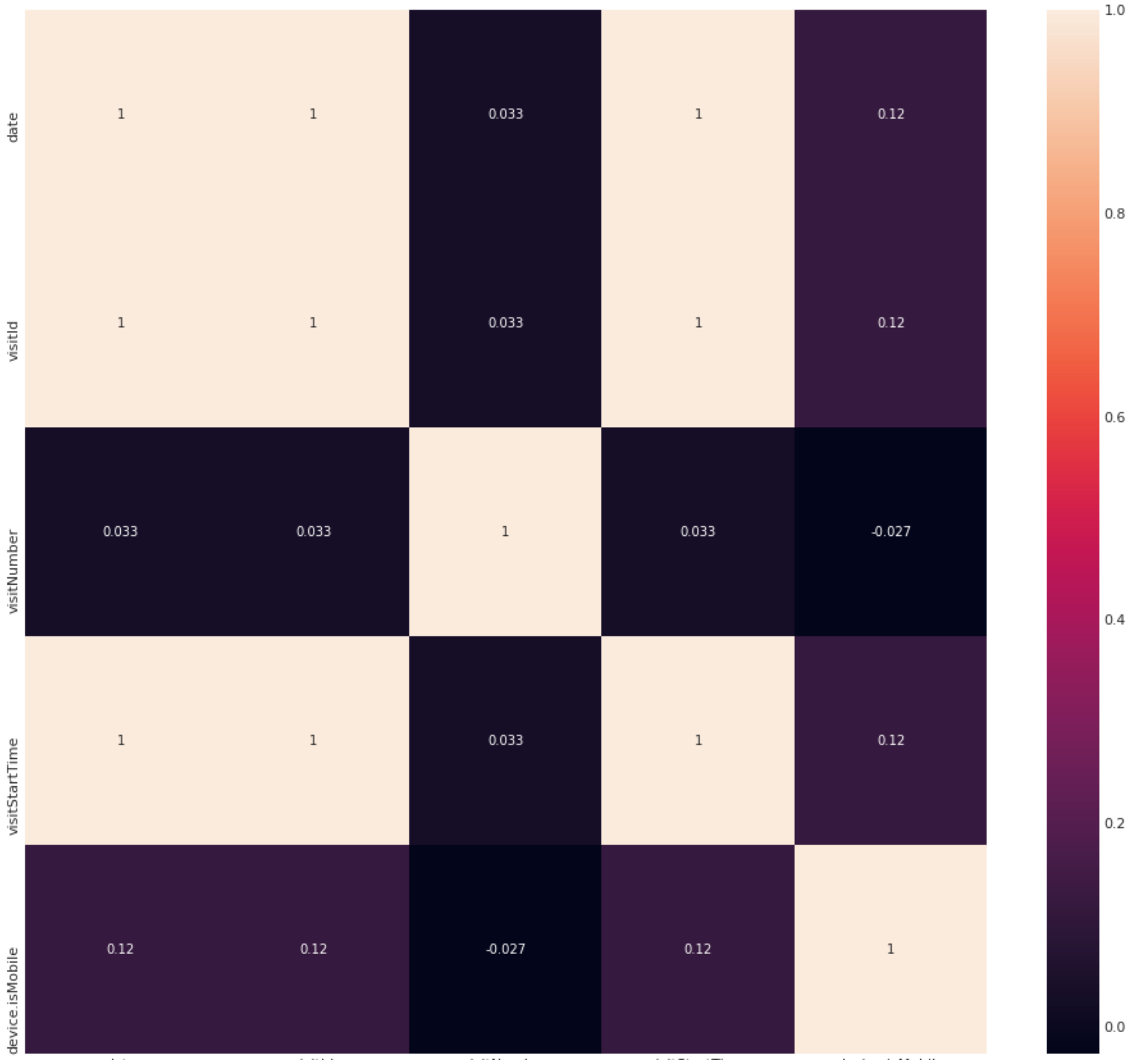
1.5 Correlation of dataset

- only for numeric variable

Code

Out[9]:

	date	visitId	visitNumber	visitStartTime	device.isMobile
date	1.0	1.0	0.033	1.0	0.12
visitId	1.0	1.0	0.033	1.0	0.12
visitNumber	0.033	0.033	1.0	0.033	-0.027
visitStartTime	1.0	1.0	0.033	1.0	0.12
device.isMobile	0.12	0.12	-0.027	0.12	1.0



date visitId visitNumber visitStartTime device.isMobile

2. Remove Variable That Contain Same Class

In [10]:
```
columns = [col for col in train_df.columns if train_df[col].nunique() > 1]
train_df = train_df[columns]
test_df = test_df[columns]
```

3. Categorical to Numeric Variable Conversion for Model Training(Label Encoding)

In [11]:
```
trn_len = train_df.shape[0]
merged_df = pd.concat([train_df, test_df])

for col in merged_df.columns:
    if col in ['fullVisitorId']: continue
    if merged_df[col].dtypes == object or merged_df[col].dtypes == bool:
        merged_df[col], indexer = pd.factorize(merged_df[col])

train_df = merged_df[:trn_len]
test_df = merged_df[trn_len:]
```

In [12]:
```
#train_df["fullVisitorId"] = train_df.fullVisitorId.astype(float)
#test_df["fullVisitorId"] = test_df["fullVisitorId"].astype(float)
```

4. Bayesian Optimization for Parameter tuning

- I tried this on **Regression Problem** but error showing me that **IT WILL APPLY ONLY BINARY AND MULTICLASS**.
- **Ideas are welcome**

In [13]:
```
# from bayes_opt import BayesianOptimization
# def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=5, random_seed=42, n_est
imators=10000, learning_rate=0.001, output_process=False):
#     # prepare data
#     train_data = lgb.Dataset(data=X, label=y)
#     # parameters
#     def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, lambda_l1, lambda_l
2, min_split_gain, min_child_weight):
#         params = {'application':'regression_l2','num_iterations': n_estimators, 'learning_rat
e':learning_rate, 'early_stopping_round':100, 'metric':'auc'}
#         params["num_leaves"] = int(round(num_leaves))
#         params['feature_fraction'] = max(min(feature_fraction, 1), 0)
#         params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
#         params['max_depth'] = int(round(max_depth))
#         params['lambda_l1'] = max(lambda_l1, 0)
#         params['lambda_l2'] = max(lambda_l2, 0)
#         params['min_split_gain'] = min_split_gain
#         params['min_child_weight'] = min_child_weight
# #         params['min_data_in_leaf'] = min_leaf_node
# #         params["min_child_samples"] = min_child_samples
#         cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=Tru
```

```
e, verbose_eval =200, metrics=['auc'])
#         return max(cv_result['auc-mean'])
#     # range
#     lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (250, 350),
#                                             'feature_fraction': (0.1, 0.99),
#                                             'bagging_fraction': (0.8, 1),
#                                             'max_depth': (5, 8.99),
#                                             'lambda_l1': (0, 5),
#                                             'lambda_l2': (0, 3),
#                                             'min_split_gain': (0.001, 0.1),
#                                             'min_child_weight': (5, 50),
# #                                             'min_data_in_leaf' : (35,45),
# #                                             'min_child_samples' : (20,30)
#                                            }, random_state=42)
#     # optimize
#     lgbBO.maximize(init_points=init_round, n_iter=opt_round)

#     # output optimization process
#     if output_process==True: lgbBO.points_to_csv("bayes_opt_result.csv")

#     # return best parameters
#     return lgbBO.res['max']['max_params']

# opt_params = bayes_parameter_opt_lgb(train_df, target, init_round=5, opt_round=10, n_folds=3)
```

5.Parameter as per Understanding of Model

In [14]:

```
param = {'num_leaves':257,
         'min_data_in_leaf': 20,
         'objective':'regression',
         'max_depth': 8,
         'learning_rate':0.0001,
         "min_child_samples":20,
         "boosting":"rf",
         "feature_fraction":0.95,
         "bagging_freq":1,
         "bagging_fraction":0.85 ,
         "bagging_seed": 32,
         "metric": 'rmse',
         "lambda_l1": 0.89,
         "verbosity": 1,
         "reg_alpha": 1,
         "reg_lambda": 1,
         "subsample" : 0.82,
         "colsample_bytree" : 0.83,
         "subsample_freq ": 5}


# {"objective" : "regression", "metric" : "rmse", "max_depth": 8, "min_child_samples": 20, "reg_a
lpha": 1, "reg_lambda": 1,
#          "num_leaves" : 257, "learning_rate" : 0.01, "subsample" : 0.8, "colsample_bytree" : 0.
8, "subsample_freq ": 5}
```

6.Model Training With KFold Cross Validation

In [15]:

```
folds = KFold(n_splits=6, shuffle=True, random_state=42)
```

```
oof = np.zeros(len(train_df))
predictions = np.zeros(len(test_df))
start = time.time()
features = list(train_df.columns)
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train_df.values, target.values)):
    trn_data = lgb.Dataset(train_df.iloc[trn_idx].values, label=target.iloc[trn_idx].values)
    val_data = lgb.Dataset(train_df.iloc[val_idx].values, label=target.iloc[val_idx].values)

    num_round = 10000
    clf = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_eval
=100, early_stopping_rounds = 100)
    oof[val_idx] = clf.predict(train_df.iloc[val_idx].values, num_iteration=clf.best_iteration)

    fold_importance_df = pd.DataFrame()
    fold_importance_df["feature"] = features
    fold_importance_df["importance"] = clf.feature_importance()
    fold_importance_df["fold"] = fold_ + 1
    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

    predictions += clf.predict(test_df.values, num_iteration=clf.best_iteration) / folds.n_spli
ts
```

```
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.64578        valid_1's rmse: 1.67745
Early stopping, best iteration is:
[32]    training's rmse: 1.64523        valid_1's rmse: 1.67646
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.64659        valid_1's rmse: 1.67019
Early stopping, best iteration is:
[32]    training's rmse: 1.64622        valid_1's rmse: 1.66988
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.64675        valid_1's rmse: 1.64918
Early stopping, best iteration is:
[32]    training's rmse: 1.64609        valid_1's rmse: 1.64779
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.6436 valid_1's rmse: 1.6873
Early stopping, best iteration is:
[50]    training's rmse: 1.64331        valid_1's rmse: 1.68646
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.64393        valid_1's rmse: 1.67154
Early stopping, best iteration is:
[54]    training's rmse: 1.64335        valid_1's rmse: 1.67149
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 1.64772        valid_1's rmse: 1.65283
Early stopping, best iteration is:
[36]    training's rmse: 1.64724        valid_1's rmse: 1.65305
```

7.Best CV Score Return By Model

In [16]:

```
print("CV score: {:<8.5f}".format(mean_squared_error(oof, target)**0.5))
```

```
CV score: 1.66758
```

8. Feature Importance

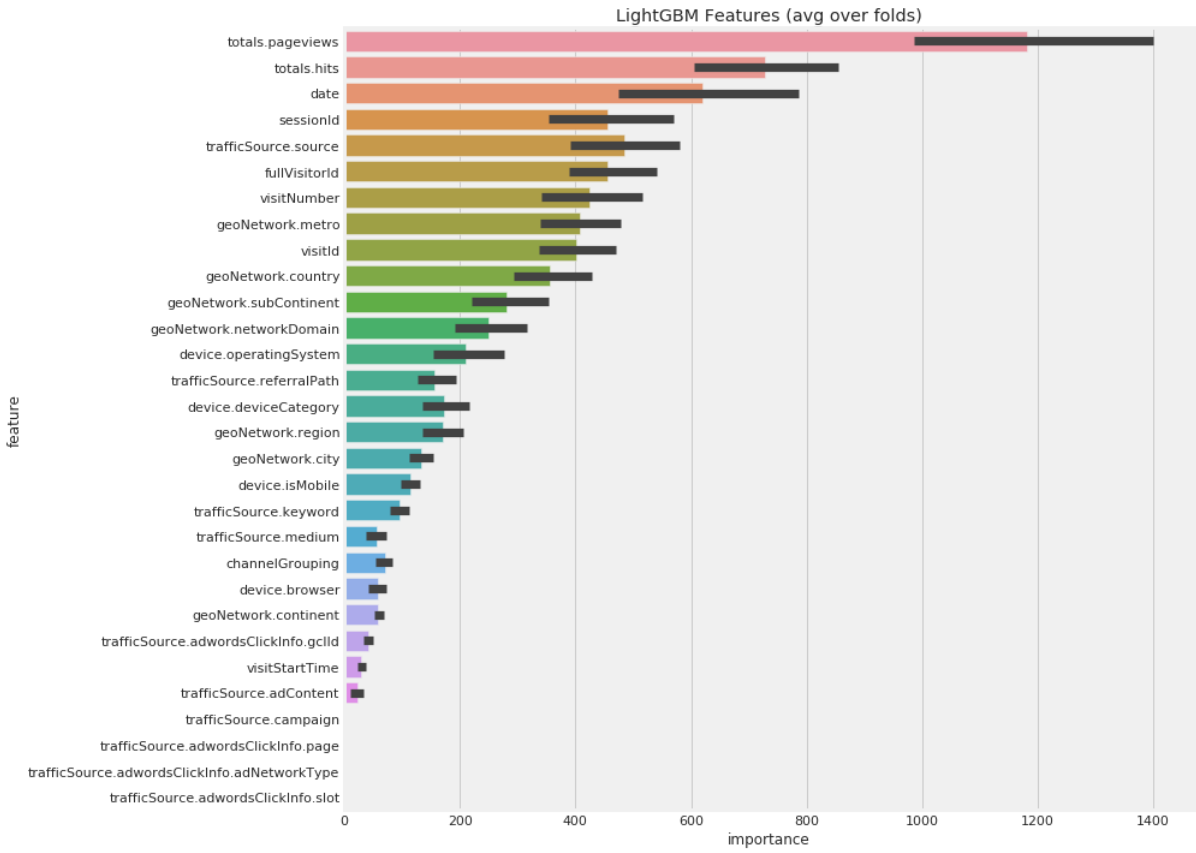
In [17]:

```
cols = feature_importance_df[["feature", "importance"]].groupby("feature").mean().sort_values(
    by="importance", ascending=False)[:1000].index

best_features = feature_importance_df.loc[feature_importance_df.feature.isin(cols)]

plt.figure(figsize=(14,10))
sns.barplot(x="importance", y="feature", data=best_features.sort_values(by="importance", ascend
ing=False))
plt.title('LightGBM Features (avg over folds)')
plt.tight_layout()
plt.savefig('lgbm_importances.png')
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-t
uple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `a
rr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, whic
h will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



9.Final Submission

In [18]:

```
submission = test_df[['fullVisitorId']].copy()
submission.loc[:, 'PredictedLogRevenue'] = predictions
grouped_test = submission[['fullVisitorId', 'PredictedLogRevenue']].groupby('fullVisitorId').su
m().reset_index()
grouped_test.to_csv('submit.csv',index=False)
```

In [ ]:

Did you find this Kernel useful?
Show your appreciation with an upvote

33

Comments (6)

All Comments

Sort by Hotness

Please sign in to leave a comment.

Djaballah • Posted on Latest Version • 3 days ago • Options

1

As a beginner this kernel really helps. Thanks for your effort.

Here is my first Analysis on kaggle link, I would be happy if you share any feedback on my work.

Ho-Youn • Posted on Latest Version • 3 days ago • Options

1

Nice and Awesome kernel!

Paresh Bhatia • Posted on Latest Version • 4 days ago • Options

1

Nice kernel Ashish! Can below link help for bayesian optimisation ? https://www.kaggle.com/baghern/bayesian-optimization-of-ridge-model-0-478

Misha Lisovyi • Posted on Version 2 • 6 days ago • Options

0

Something went wrong with variable names?

AshishPatel Kernel Author • Posted on Version 2 • 6 days ago • Options

0

Yes we will try to make it clear.thanks for attention.

5 days ago

This Comment was deleted.

Similar Kernels

Titanic Survival Prediction End To End ML Pipeline

EDA And Basic Model For Housing Prices

Let's Discover More About The Olympic Games!

Exploratory Analysis - GA Customer Revenue

A Comprehensive Guide To Titanic Machine Learning