CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

GOOGLE ANALYTICS TECHNICAL REPORT

Minya Na, Yuankun Huang,
Siyu Zhang, Jingru Tang,
Yulin Liu, Elizabeth Zhu

1. Introduction

1.1 Background

Google Analytics is a web analytics service offered by Google that tracks and reports website traffic, currently as a platform serves small business owners.

Small business owners who are always keen to gain insights about their customers, but due to the scale of economics, they have limited resources to comprehensively identify high-value customers. Currently, small business owners rely heavily on Google Analytics as a strong tool to passively record and understand customers' behaviors and patterns.

Thus it is important for Google Analytics to offer small business owners with a sophisticated revenue prediction model so they can identify potential customers who will likely make the great purchases in the near future.

1.2 Goal Statement

For Google Analytics:

- Develop a reliable and transferrable model
- Offer better products and services to clients
- Attract more small business clients to refine the analytics ecosystem

For Small business owners:

- Identify High-Valued Customers
- Understand their demographic profile and purchase behaviors
- Adjust marketing and operational strategy accordingly

2. Analysis

2.1 Data Cleaning and Transformation

By observing the dataset, we noticed that dataset contains a few JSON columns that needs to get expanded, such as device, geoNetwork, and trafficSource.

Firstly, we transformed some columns into specific data types based on the needs. For example, we transformed visitId into character, date into DateTime, but we skipped hits and customDimensions columns, which are not quite relevant in this scenario.

After flattening, the JSON columns, which gave us around 85 columns.

Then we started to process the flattened columns based on our understanding of column features and business needs. We segment the columns into three types: categorical data, columns need to be dropped, and numerical data, whose types are misread as objects.

```
#transform categorical data into numerical data
#columns that need to be transformed into numerical ones
columns_cate = ['channelGrouping','device_browser','deviceCategory', \
                'device_operatingSystem','device_isMobile','geoNetwork_city','geoNetwork_continent','geoNetwork_country'
                'geoNetwork_metro','geoNetwork_networkDomain','geoNetwork_region','geoNetwork_subContinent', \
                'trafficSource_adContent','trafficSource_adwordsClickInfo.adNetworkType','trafficSource_adwordsClickInfo
                'trafficSource_adwordsClickInfo.isVideoAd','trafficSource_adwordsClickInfo.slot', \
                'trafficSource_campaign','trafficSource_isTrueDirect','trafficSource_keyword', \
                'trafficSource_medium','trafficSource_referralPath','trafficSource_source']
columns_delete = ['device_browserSize','device_browserVersion','flashVersion','language', \
                  'device_mobileDeviceBranding','device_mobileDeviceInfo','device_mobileDeviceMarketingName', \
                  'device_mobileDeviceModel','device_mobileInputSelector','device_operatingSystemVersion', \
                  'device_screenColors','device_screenResolution','geoNetwork_cityId','geoNetwork_latitude', \
                  'geoNetwork_longitude','geoNetwork_networkLocation','trafficSource_adwordsClickInfo.criteriaParameters
columns_like_num_cate = ['totals_bounces','totals_hits','totals_newVisits','totals_pageviews', \
                         'totals_sessionQualityDim','totals_timeOnSite','totals_totalTransactionRevenue', \
                         'totals_transactionRevenue','totals_transactions','totals_visits','trafficSource_adwordsClickIn
```

We detected that about 17 columns that have the same values across all rows, the same data will have the same role on each customer, such data would not affect our predictions and therefore we dropped them.

For categorical data, we transformed those into numbers using label encoder. For numerical data, we changed the type of values into numbers using to_numeric function. For columns needed to be dropped, we deleted those columns.

```
#convert those categorical data into numbers
for col in columns_cate:
    print(col)
    lbl = preprocessing.LabelEncoder()
    lbl.fit(list(sample_final1[col].values.astype('str')))
    sample_final1[col] = lbl.transform(list(sample_final1[col].values.astype('str')))
```

```
channelGrouping
device_browser
deviceCategory
device_operatingSystem
device_isMobile
geoNetwork_city
geoNetwork_continent
geoNetwork_country
geoNetwork_metro
geoNetwork_networkDomain
geoNetwork_region
geoNetwork_subContinent
trafficSource_adContent
trafficSource_adwordsClickInfo.adNetworkType
trafficSource_adwordsClickInfo.gclId
trafficSource_adwordsClickInfo.isVideoAd
trafficSource_adwordsClickInfo.slot
trafficSource_campaign
trafficSource_isTrueDirect
trafficSource_keyword
trafficSource_medium
trafficSource_referralPath
trafficSource_source
```

```
sample_final1[columns_like_num_cate].applymap(lambda x: pd.to_numeric(x, errors='ignore'))
```

We have also figured out that there are no NAs in categorical data and on the contrary, there are many NAs in numerical data. Therefore, we replace all NAs with number ‘0’.

```
#for numerical data, change NA into 0
sample_final1[columns_like_num_cate] = sample_final1[columns_like_num_cate].fillna(0)
```

We did the same data cleaning and transformation for testing data. Up to now, all the data are transformed into numerical values, and therefore we can do feature engineering as following.

2.2 Feature Selection

To include only features that can optimize final predicting results in our model, as stated before, firstly we manually dropped columns with constant value across all rows. Then we tried three models -- lightGBM, XGBoost, Random Forest to help rank the most important features. Besides we also manually went through all selected features from the model to confirm if they intuitively make sense to retain from a business perspective. For example, residential city can impact purchase power, etc.

Followings are important features selected by the three models:

```
# Split the train dataset into development and valid based on time
dev_df = clean[clean['date']<= '2017-05-31']
val_df = clean[clean['date']> '2017-05-31']
dev_y = np.log1p(dev_df["totals_transactionRevenue"].values)
val_y = np.log1p(val_df["totals_transactionRevenue"].values)

dev_X = dev_df[columns_cate + columns_num]
val_X = val_df[columns_cate + columns_num]
test_X = test[columns_cate + columns_num]
```

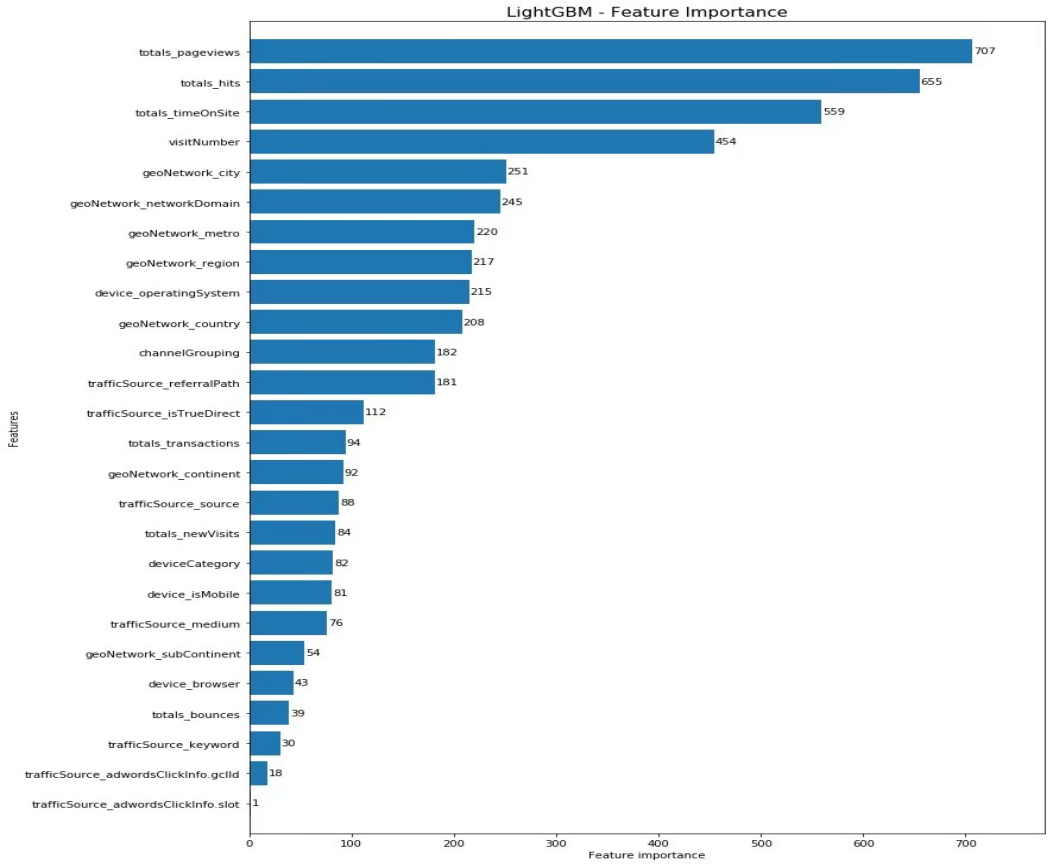
- LightGB

```
def run_lgb(train_X, train_y, val_X, val_y, test_X):
    params = {
        "objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 30,
        "min_child_samples" : 100,
        "learning_rate" : 0.1,
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.5,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    model = lgb.train(params, lgtrain, 1000, valid_sets=[lgval], early_stopping_rounds=100, verbose_eval=100)

    pred_test_y = model.predict(test_X, num_iteration=model.best_iteration)
    pred_val_y = model.predict(val_X, num_iteration=model.best_iteration)
    return pred_test_y, model, pred_val_y

# Training the model #
pred_test, model, pred_val = run_lgb(dev_X, dev_y, val_X, val_y, test_X)
```

```
from sklearn import metrics
pred_val[pred_val<0] = 0
val_pred_df = pd.DataFrame({"fullVisitorId_str":val_df["fullVisitorId_str"].values})
val_pred_df["transactionRevenue"] = val_df["totals_transactionRevenue"].values
val_pred_df["PredictedRevenue"] = np.expm1(pred_val)
#print(np.sqrt(metrics.mean_squared_error(np.log1p(val_pred_df["transactionRevenue"].values), np.log1p(val_pred_df["P
val_pred_df = val_pred_df.groupby("fullVisitorId_str")["transactionRevenue", "PredictedRevenue"].sum().reset_index()
print(np.sqrt(metrics.mean_squared_error(np.log1p(val_pred_df["transactionRevenue"].values), np.log1p(val_pred_df["Pr
```

```
sub_df = pd.DataFrame({"fullVisitorId_str":test_id})
pred_test[pred_test<0] = 0
sub_df["PredictedLogRevenue"] = np.expm1(pred_test)
sub_df = sub_df.groupby("fullVisitorId_str")["PredictedLogRevenue"].sum().reset_index()
sub_df.columns = ["fullVisitorId_str", "PredictedLogRevenue"]
sub_df["PredictedLogRevenue"] = np.log1p(sub_df["PredictedLogRevenue"])
sub_df.to_csv("baseline_lgb.csv", index=False)
```

```
fig, ax = plt.subplots(figsize=(12,18))
lgb.plot_importance(model, max_num_features=50, height=0.8, ax=ax)
ax.grid(False)
plt.title("LightGBM - Feature Importance", fontsize=15)
plt.show()
```


LightGBM - Feature Importance
totals_pageviews 707
totals_hits 655
totals_timeOnSite 559
visitNumber 454
geoNetwork_city 251
geoNetwork_networkDomain 245
geoNetwork_metro 220
geoNetwork_region 217
device_operatingSystem 215
geoNetwork_country 208
channelGrouping 182
trafficSource_referralPath 181
trafficSource_isTrueDirect 112
totals_transactions 94
geoNetwork_continent 92
trafficSource_source 88
totals_newVisits 84
deviceCategory 82
device_isMobile 81
trafficSource_medium 76
geoNetwork_subContinent 54
device_browser 43
totals_bounces 39
trafficSource_keyword 30
trafficSource_adwordsClickInfo.gclId 18
trafficSource_adwordsClickInfo.slot 1
Features
0
100
200
300
400
500
600
700
Feature importance

- XGBoost

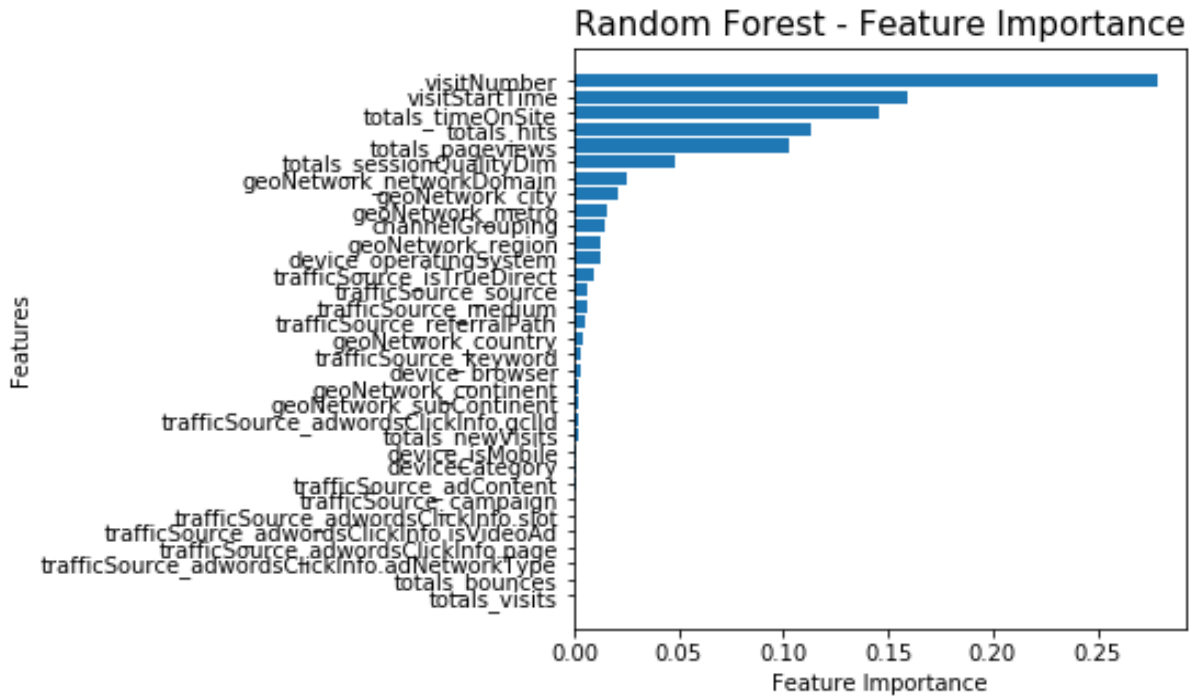
```
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import XGBRegressor
import xgboost as xgb
from xgboost import plot_importance
import matplotlib.pyplot as plt
import pandas as pd
```

```
data = train_df
x = data.iloc[:,0:56]
y = data['totals_transactionRevenue']
y[y.isnull()] = 0

x = x[['channelGrouping',
       'visitNumber', 'visitStartTime',
     'device_browser',
     'device_operatingSystem',
     'device_isMobile',  'deviceCategory',
     'geoNetwork_continent', 'geoNetwork_subContinent',
     'geoNetwork_country', 'geoNetwork_region', 'geoNetwork_metro',
     'geoNetwork_city', 'geoNetwork_networkDomain',
     'trafficSource_campaign',
     'trafficSource_source', 'trafficSource_medium',
     'trafficSource_keyword', 'trafficSource_referralPath',
     'trafficSource_isTrueDirect', 'trafficSource_adContent',
     'trafficSource_adwordsClickInfo.page',
     'trafficSource_adwordsClickInfo.slot',
     'trafficSource_adwordsClickInfo.gclId',
     'trafficSource_adwordsClickInfo.adNetworkType',
     'trafficSource_adwordsClickInfo.isVideoAd', 'totals_visits',
     'totals_hits', 'totals_pageviews', 'totals_bounces',
     'totals_newVisits', 'totals_sessionQualityDim',
     'totals_timeOnSite' ]]
x[x.isnull()] = 0
```

```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1, max_depth = 5,
                          alpha = 10, n_estimators = 10)
xg_reg.fit(x, y)
import matplotlib.pyplot as plt
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [5, 5]
plt.title('Xgboost - Feature Importance', fontsize = 15)
```

- Random Forest

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators = 100,
                           n_jobs = -1,
                           oob_score = True,
                           bootstrap = True,
                           random_state = 42)

rf.fit(x, y)

sort_zip = sorted(zip(x.columns, rf.feature_importances_),key= lambda t: t[1])
fig, ax = plt.subplots()
bar_plot = plt.barh([val[0] for val in sort_zip], [val[1] for val in sort_zip], align='center')
plt.title('Random Forest - Feature Importance', fontsize = 15)
```



According to above demonstration, the most important features are totals_pageviews, totals_hits, totals_timeOnSite, visitNumber, channel grouping, and session quality dimensions.

Those results are aligned with our domain knowledge. People who usually view the pages, hit the tags, and spend more time on the website have a higher tendency to spend money on the website.

We fitted our model firstly based on the ranking results from the above algorithms. Below are the measurement results based on the ranking features we selected:

- device_operatingSystem
- geoNetwork_networkDomain
- trafficSource_source
- channelGrouping
- device_browser

- geoNetwork_subContinent
- geoNetwork_country
- totals_pageviews
- device_isMobile
- trafficSource_isTrueDirect
- totals_bounces
- totals_hits
- visitNumber
- geoNetwork_region
- geoNetwork_metro
- geoNetwork_city
- geoNetwork_networkDomain

Below are the performance results based on the above features. However, when we fitted the data for scoring, this model doesn't give us a better scoring for the kaggle leaderboard.

```
mean_squared_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.6738925981947226
```

```
mean_absolute_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.5886683357195858
```

```
accuracy_score(y_test.loc[:,"will_return"], lgb_model.predict(X_test))
```

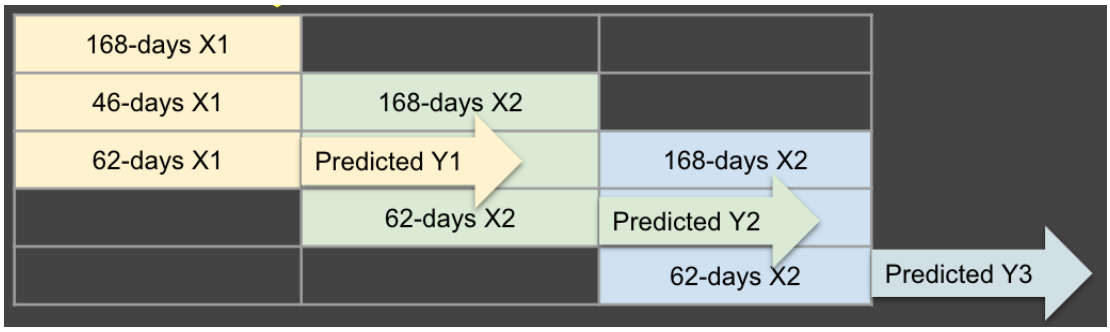
```
0.9934117915615132
```

Refer to Features Coding for Further Details

Algorithms would only give ranking on selected features, which is a subjective choice across different business users. it is also important to incorporate some business insights for features to be included for model training. As a result, apart from the features rankings provided by the algorithms, we also aggregate some features based on business knowledge: such as that different demographic people have different spending amount, the time gap between people first time saw the ads and the last time would also affect people's decisions. We will explain this part in detail in part.

2.3.Data split by time frame

The testing set given includes data from 2018-05-01 to 2018-10-15 and our prediction on whether will make purchase or not (classification) is based on whether customers that appear in the testing set will return to store after a 46-day gap. Thus, with the hope to better mimic the predicting situation, we decided to split the training dataset into several similar time frames (168 days with target variable summarized from days later on). Since the training data contains user transactions from 2016-08-01 to 2018-04-30 (637 days), we split it into three independent subset and perform feature aggregation within each segment, the splitting process is illustrated in the chart below:




```
# date_time_str = '2018-06-29 08:15:27.243860' (2016-08-01 to 2018-04-30)
def split_by_date(df, start_date_str):
    df['date'] = pd.to_datetime(df['date'])

    start_date_X = datetime.datetime.strptime(start_date_str, '%Y-%m-%d')
    end_date_X = start_date_X + datetime.timedelta(days=168)

    start_date_y = end_date_X + datetime.timedelta(days=46)
    end_date_y = start_date_y + datetime.timedelta(days=62)

    mask_X = (df['date'] >= start_date_X) & (df['date'] < end_date_X)
    mask_y = (df['date'] >= start_date_y) & (df['date'] < end_date_y)

    X = df.loc[mask_X]
    y = df.loc[mask_y]

    return X, y
```

```
X1, y1 = split_by_date(data, '2016-08-01')
X2, y2 = split_by_date(data, '2017-01-16')
X3, y3 = split_by_date(data, '2017-07-03')
X4, y4 = split_by_date(data, '2017-12-18')
```

By slicing the data into the designed 3 windows, we can make fully utilization of training datasets while avoiding using the same data multiple times to prevent our models from overfitting.

2.4 Feature Aggregation

Currently each row of the data represented each headcount/visit, however, in the final prediction we suppose to predict unique customer's spending. Therefore, we did feature aggregation based on fullVisitorID.

Apart from that, we aggregated the categorical data with most frequent ones, given as we talked about previously, transferring visit to visitor and showing how much they will spend in the future is the ultimate goal. Therefore, it is necessary for us to select the most frequent categorical data as value. In addition, we aggregated binary data with mean, and numerical data with sum, min, max, median, mean and standard deviation.

```
def feature_engineer(df):
    date_format = '%Y-%m-%d'
    pd.to_datetime(df['date'], format=date_format)
    df1 = df.groupby('fullVisitorId_str').agg({'date':['min','max'],
                                        'tf_mindate': 'min',
                                        'tf_maxdate': 'max',
                                        'channelGrouping': 'max',
                                        'visitNumber': ['max', 'count'],
                                        'device_browser': 'max',
                                        'device_operatingSystem': 'max',
                                        'deviceCategory': 'max',
                                        'geoNetwork_continent': 'max',
                                        'geoNetwork_subContinent': 'max',
                                        'geoNetwork_country': 'max',
                                        'geoNetwork_region': 'max',
                                        'geoNetwork_metro': 'max',
                                        'geoNetwork_city': 'max',
                                        'geoNetwork_networkDomain': 'max',
                                        'trafficSource_source': 'max',
                                        'trafficSource_medium': 'max',
                                        'trafficSource_adwordsClickInfo.isVideoAd': 'mean',
                                        'device_isMobile': 'mean',
                                        'trafficSource_isTrueDirect': 'mean',
                                        'totals_bounces':'sum',
                                        'totals_hits': ['sum', 'mean', 'min', 'max', 'median', 'std'],
                                        'totals_pageviews': ['sum', 'mean', 'min', 'max', 'median', 'std'],
                                        'totals_transactionRevenue': 'sum',
                                        'totals_transactions': 'sum'})
    return df1
```

```
df1 = feature_engineer(df)
```

```
date_format = '%Y-%m-%d'
df1['first_ses_from_the_period_start'] = pd.to_datetime(df1['date']['min'], format=date_format) \
                                       - pd.to_datetime(df1['tf_mindate']['min'], format=date_format)
df1['first_ses_from_the_period_start'] = df1['first_ses_from_the_period_start'].astype('timedelta64[D]')
df1['last_ses_from_the_period_end'] = pd.to_datetime(df1['tf_maxdate']['max'], format=date_format) \
                                    - pd.to_datetime(df1['date']['max'], format=date_format)
df1['last_ses_from_the_period_end'] = df1['last_ses_from_the_period_end'].astype('timedelta64[D]')
df1['interval_dates'] = pd.to_datetime(df1['date']['max'], format=date_format) \
                      - pd.to_datetime(df1['date']['min'], format=date_format)
df1['interval_dates'] = df1['interval_dates'].astype('timedelta64[D]')
df1 = df1.iloc[:,4:]
```

```
df2 = df.groupby('fullVisitorId_str')['date'].nunique().rename('unique_date_num')
input_test = pd.merge(df1, df2, on = 'fullVisitorId_str')
```

We aggregated features based on the importance we get from the above feature selection graphs, below are details about final 37 features we selected:

1	first_ses_from_the_period_start	The first day of each period minus the first day of the whole dataset
2	last_ses_from_the_period_end	The last day of the whole dataset minus the last day of each period
3	interval_dates	The last day of each period minus the first day of each period
4	unique_date_num	The number of unique visit days of each customer within each time period
5	channelGrouping_max	The most frequent channel via which the user

		came to the Store.
6	visitNumber_max	Maximum session number of each user
7	visitNumber_count	Unique session number of each user
8	browser_max	The most frequent browser each user used
9	operatingSystem_max	The most frequent operating system each user used
10	Category_max	The most frequent device category each user used
11	continent_max	The continent each user lived most of the time
12	subContinent_max	The subcontinent each user lived most of the time
13	country_max	The country each user lived most of the time
14	region_max	The region each user lived most of the time
15	metro_max	The metro each user lived most of the time
16	city_max	The city each user lived most of the time
17	networkDomain_max	The network domain each user lived most of the time
18	source_max	The most frequent traffic source each user browsed
19	medium_max	The most frequent medium each user browsed
20	adwordsClickInfo.isVideoAd_mean	The most frequent traffic source each user browsed
21	isMobile_mean	The mean of isMobile
22	isTrueDirect_mean	The mean of isTrueDirect
23	bounces_sum	The sum of bounces
24	hits_sum	The sum of hits
25	hits_max	The maximum of hits

26	hits_mean	The mean of hits
27	hits_min	The minimum of hits
28	hits_median	The median of hits
29	hits_sd	The standard deviation of hits
30	pageviews_sum	The sum of pageviews
31	pageviews_max	The maximum of pageviews
32	pageviews_mean	The mean of pageviews
33	pageviews_min	The minimum of pageviews
34	pageviews_median	The medium of pageviews
35	pageviews_sd	The standard deviation of pageviews
36	transactionRevenue_sum	The sum of transaction revenue
37	transactions_sum	The sum of how many times the user had the transaction

2.5 Model Selection & Hyperparameter Tuning

The original dataset contains imbalanced data, as approximately 80% of visitors made no purchase and the prediction on revenue can only rely on the 20% revenue records. We tried two different ways to design our predicting model.

The first one tried to better cater to business needs, and we decided to conduct both classification and regression in the data mining process to take care of the imbalanced dataset so that business owners can not only have predicting spendings but also whether customer will make a purchase decision, so that they can develop different marketing promotions accordingly. For the classification part of our first model, we identified those customers who paid second visits within the 62-days time frame after their first visit in the 168-days time frame and labeled them as ‘1’ in flag column and those who did not show up again labeled as ‘0’ to predict whether they will make a purchase or not.

Then we removed the rows with ‘0’ in their labels and instead only conduct regression model on those customers we predicted to make purchase and predict their purchase amount.

The second model is simply designed to predict future revenue without classification. The drawback of this model is the lack of business insights from business owners’ standpoints.

Regarding the two model designs, we tried two ensemble models, LightGBM and XGboost, for both classification and regression to predict revenue.

2.5.1 Classification and Regression Model

2.5.1.1 Classification

LightGBM is a gradient boosting framework that uses tree-based learning algorithms.

It is designed to be distributed and efficient with advantages such as faster-training speed and higher efficiency, lower memory usage, better accuracy, support of parallel and GPU learning, and can handle large-scale data.
(From LightGBM documentation: https://lightgbm.readthedocs.io/en/latest/index.html)
We used grid search for hyperparameter tuning and got tuned values for each parameter as shown here:

max_bin	256
learning_rate	[0.01,0.02]
num_leaves	[5,10,15]
bagging_fraction	[0.8,0.9,1]
feature_fraction	[0.9,1]
min_data	[1,2]
bagging_freq	[0,1]

The best parameters given by Grid Search is:

```
(LGBMClassifier(bagging_fraction=0.8, bagging_freq=0, boosting_type='gbdt',
                class_weight=None, colsample_bytree=1.0, feature_fraction=0.9,
                importance_type='split', learning_rate=0.02, max_bin=256,
                max_depth=-1, metric='binary_logloss', min_child_samples=20,
                min_child_weight=0.001, min_data=2, min_split_gain=0.0,
                n_estimators=100, n_jobs=-1, num_leaves=15, objective='binary',
                random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
                subsample=1.0, subsample_for_bin=200000, subsample_freq=0),
 0.9934785552112811,
 {'bagging_fraction': 0.8,
  'bagging_freq': 0,
  'feature_fraction': 0.9,
  'learning_rate': 0.02,
  'max_bin': 256,
  'min_data': 2,
  'num_leaves': 15})
```

Belows are the performance measurement for the LightGBM:

```
[84] from sklearn.metrics import accuracy_score
     lgb_model = lgb.LGBMClassifier(objective = 'binary',metric = 'binary_logloss',bagging_fraction=0.9,\
                                    bagging_freq=1,feature_fraction=0.8,learning_rate=0.01,max_bin=256,min_data=1,num_leaves= 15)
     lgb_model.fit(X_train,y_train.loc[:,"will_return"])
     #mean_squared_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
     accuracy_score(y_test.loc[:,"will_return"], lgb_model.predict(X_test))
```

```
0.9944588468012434
```

```
[85] from sklearn.metrics import precision_recall_fscore_support
     precision_recall_fscore_support(y_test.loc[:,"will_return"], lgb_model.predict(X_test),average='weighted')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Prec
  'precision', 'predicted', average, warn_for)
(0.9889483979812589, 0.9944588468012434, 0.9916959676228526, None)
```

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the gradient boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

We used the grid search method again for hyperparameter tuning, and the results are:

min_child_weight	[1, 5, 10]
gamma	[1, 1.5, 2]
subsample	[0.6, 0.8, 1.0]
colsample_bytree	[0.6, 0.8, 1.0]
max_depth	[3, 4, 5]

The result indicated the best combination of hyperparameter is:
{'subsample': 0.6, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 1.5, 'colsample_bytree': 0.6}.
Code for grid search is as follows:

```
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [1, 1.5, 2],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
```

```
xgb_c = XGBClassifier(learning_rate=0.02, n_estimators=600, objective='binary:logistic',
                    silent=True, nthread=1)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

```
from datetime import datetime
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec, 2)))
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV


folds = 3
param_comb = 5

skf = StratifiedKFold(n_splits=folds, shuffle = True, random_state = 1001)

random_search = RandomizedSearchCV(xgb_c, param_distributions=params, n_iter=param_comb, scoring='roc_auc', \
                                   n_jobs=4, cv=skf.split(x_train, y_train_return), verbose=3, random_state=1001 )

# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(x_train, y_train_return)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  15 out of  15 | elapsed: 118.9min finished

 Time taken: 2 hours 13 minutes and 26.89 seconds.
```

```
print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)
print('\n Best normalized gini score for %d-fold search with %d parameter combinations:' % (folds, param_comb))
print(random_search.best_score_ * 2 - 1)
print('\n Best hyperparameters:')
print(random_search.best_params_)
```

```
 Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6, gamma=1.5,
              learning_rate=0.02, max_delta_step=0, max_depth=5,
              min_child_weight=5, missing=None, n_estimators=600, n_jobs=1,
              nthread=1, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=0.6, verbosity=1)

 Best normalized gini score for 3-fold search with 5 parameter combinations:
0.76289582232577

 Best hyperparameters:
{'subsample': 0.6, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 1.5, 'colsample_bytree': 0.6}
```

Then, we used the tuned model to make predictions. The accuracy of the model is 99.34%.

```
[87] from sklearn.metrics import accuracy_score
     accuracy_score(y_test_return, y_test_predict_classifier)
```

```
0.9934244733871889
```

```
[88] from sklearn.metrics import precision_recall_fscore_support
     precision_recall_fscore_support(y_test_return, y_test_predict_classifier,average='weighted')
```

```
(0.9913169066716786, 0.9934244733871889, 0.9903305096170357, None)
```

Then, we used the best hyperparameters in our model to make predictions on the testing data:

```
# final classification model
xgb_c = XGBClassifier(learning_rate=0.02, n_estimators=600, objective='binary:logistic',
                      silent=True, nthread=1, subsample= 0.6, min_child_weight= 5, max_depth= 5, gamma= 1.5, colsample_bytree= 0.6)
```

```
xgb_c.fit(x_train, y_train_return)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.6, gamma=1.5,
              learning_rate=0.02, max_delta_step=0, max_depth=5,
              min_child_weight=5, missing=None, n_estimators=600, n_jobs=1,
              nthread=1, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=True, subsample=0.6, verbosity=1)
```

```
# Prediction on validation set
y_test_predict_classifier = xgb_c.predict(x_test)
```

(please check the 'GoogleAnalytics.ipynb document for detailed codes)

2.5.1.2 Regression

LightGBM

We used grid-search to tune the hyperparameter of the model. Hyperparameter we tune and potential values are as follows:

max_bin	256
learning_rate	[0.01,0.02]
num_leaves	[5,10,15]
bagging_fraction	[0.8,0.9,1]
feature_fraction	[0.9,1]
min_data	[1,2]
bagging_freq	[0,1]

The best parameters given by grid search are shown below:

```
(LGBMRegressor(bagging_fraction=0.8, bagging_freq=1, boosting_type='gbdt',
               class_weight=None, colsample_bytree=1.0, feature_fraction=0.9,
               importance_type='split', learning_rate=0.02, max_bin=256,
               max_depth=-1, metric='rmse', min_child_samples=20,
               min_child_weight=0.001, min_data=2, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=15,
               objective='regression', random_state=None, reg_alpha=0.0,
               reg_lambda=0.0, silent=True, subsample=1.0,
               subsample_for_bin=200000, subsample_freq=0),
 0.11703032705631884,
 {'bagging_fraction': 0.8,
  'bagging_freq': 1,
  'feature_fraction': 0.9,
  'learning_rate': 0.02,
  'max_bin': 256,
  'min_data': 2,
  'num_leaves': 15})
```

Belows are the performance measurement for the LightGBM:

```
[82] import lightgbm as lgb
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import KFold

     from sklearn.metrics import mean_squared_error
     lgb_model_re = lgb.LGBMRegressor(objective = 'regression', metric = 'rmse',bagging_fraction = 0.8,\
       bagging_freq = 1,feature_fraction = 0.9,learning_rate =  0.02,max_bin =  256,min_data=  2,num_leaves= 15)
     lgb_model_re.fit(X_train_reg,y_train_reg.loc[:,"how_much"])
     mean_squared_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.737362072365456
```

```
[83] from sklearn.metrics import mean_absolute_error
     #lgb_model_re = lgb.LGBMRegressor(objective = 'regression',metric = 'rmse', bagging_fraction = 0.8, \
     #                                 bagging_freq = 0, feature_fraction = 0.9, learning_rate = 0.02, max_bin = 256,\
     #                                 num_leaves = 10, min_data = 2)
     #lgb_model_re.fit(X_train_reg,y_train_reg.loc[:,"how_much"])
     mean_absolute_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.6133837212470491
```

XGBoost

We used the grid search method again for hyperparameter tuning, and the results are:

learning_rate	[0.03, 0.05, 0.07]
max_depth	[5, 6, 7]

The result of the grid search indicates the best combination of hyperparameter: {'subsample': 0.7, 'silent': 1, 'objective': 'reg:linear', 'nthread': 4, 'n_estimators': 500, 'min_child_weight': 4, 'max_depth': 5, 'learning_rate': 0.03, 'colsample_bytree': 0.7}.

```
xgb_r = XGBRegressor()
parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
              'objective':['reg:linear'],
              'learning_rate': [.03, 0.05, .07], #so called `eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [4],
              'silent': [1],
              'subsample': [0.7],
              'colsample_bytree': [0.7],
              'n_estimators': [500]}
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
xgb_grid = RandomizedSearchCV(xgb_r,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_grid.fit(x_train,
         y_train_value)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

```
{'subsample': 0.7, 'silent': 1, 'objective': 'reg:linear', 'nthread': 4, 'n_estimators': 500, 'min_child_weight': 4, 'max_depth': 5, 'learning_rate': 0.03,
```

Then we used the best hyperparameters in our model to do prediction:

```
# final regression model
xgb_r = XGBRegressor(subsample =  0.7, silent = 1, objective = 'reg:linear', nthread = 4, n_estimators = 500, min_child_weight = 4, max_depth = 5, l
```

```
xgb_r.fit(x_train, y_train_value)
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, gamma=0,
             importance_type='gain', learning_rate=0.03, max_delta_step=0,
             max_depth=5, min_child_weight=4, missing=None, n_estimators=500,
             n_jobs=1, nthread=4, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=1,
             subsample=0.7, verbosity=1)
```

```
# predict revenue based on validation set
predict_revenue = xgb_r.predict(x_test)
```

```
# prediction on real test data
predict_revenue_final= xgb_r.predict(test_df)
```

```
predict_revenue_final = pd.DataFrame(predict_revenue_final)
```

Belows are the performance measurement for the XGBoost:

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test_value, predict_revenue)
```

```
15.203778437305166
```

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test_value, predict_revenue)
```

```
1.7798873164489573
```

(please check the 'GoogleAnalytics.ipynb document for detailed codes)

2.5.2 Only Regression

LightGBM

We also tried to predict revenue without doing classification.

For LightBGM, we used dgrid search for hyperparameter tuning:

```
[27] gridSearchCV_reg.best_estimator_, gridSearchCV_reg.best_score_, gridSearchCV_reg.best_params_
```

```
(LGBMRegressor(bagging_fraction=0.8, bagging_freq=1, boosting_type='gbdt',
               class_weight=None, colsample_bytree=1.0, feature_fraction=0.9,
               importance_type='split', learning_rate=0.02, max_bin=256,
               max_depth=-1, metric='rmse', min_child_samples=20,
               min_child_weight=0.001, min_data=2, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=10,
               objective='regression', random_state=None, reg_alpha=0.0,
               reg_lambda=0.0, silent=True, subsample=1.0,
               subsample_for_bin=200000, subsample_freq=0),
 0.02371773303210371,
 {'bagging_fraction': 0.8,
  'bagging_freq': 1,
  'feature_fraction': 0.9,
  'learning_rate': 0.02,
  'max_bin': 256,
  'min_data': 2,
  'num_leaves': 10})
```

The performance measurements are as listed below:

```
[28] import lightgbm as lgb
     from sklearn.model_selection import GridSearchCV
     from sklearn.model_selection import KFold

     from sklearn.metrics import mean_squared_error
     lgb_model_re = lgb.LGBMRegressor(objective = 'regression', metric = 'rmse',bagging_fraction = 0.8,\
       bagging_freq = 1,feature_fraction = 0.9,learning_rate =  0.02,max_bin =  256,min_data=  2,num_leaves= 10)
     lgb_model_re.fit(X_train,y_train.loc[:,"how_much"])
     mean_squared_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.0788821686817371
```

```
[29] from sklearn.metrics import mean_absolute_error
     #lgb_model_re = lgb.LGBMRegressor(objective = 'regression',metric = 'rmse', bagging_fraction = 0.8, \
     #                                 bagging_freq = 0, feature_fraction = 0.9, learning_rate = 0.02, max_bin = 256,\
     #                                 num_leaves = 10, min_data = 2)
     #lgb_model_re.fit(X_train_reg,y_train_reg.loc[:,"how_much"])
     mean_absolute_error(y_test.loc[:,"how_much"], lgb_model_re.predict(X_test))
```

```
0.010238043481574881
```

XGBoost

We also tried to predict revenue without doing classification.

For XGBoost, we used dgrid search for hyperparameter tuning:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
xgb_grid = RandomizedSearchCV(xgb_r,
                        parameters,
                        cv = 2,
                        n_jobs = 5,
                        verbose=True)

xgb_grid.fit(x_train,
        y_train_value)

print(xgb_grid.best_score_)
print(xgb_grid.best_params_)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:266: UserWarning: The total space of parameters 9 is smaller than n_iter=10. Runn
  % (grid_size, self.n_iter, grid_size), UserWarning)
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  18 out of  18 | elapsed: 116.1min finished
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated and will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
0.006701823721211836
{'subsample': 0.7, 'silent': 1, 'objective': 'reg:linear', 'nthread': 4, 'n_estimators': 500, 'min_child_weight': 4, 'max_depth': 5, 'learning_rate': 0.03,
```

The performance measurement are as below:

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test_value, predict_revenue)
```

```
0.11697325301397242
```

```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test_value, predict_revenue)
```

```
0.013101233577934614
```

2.5.3 Final Model Scoring

We used the regression-only model for scoring on Kaggle leaderboard, the score on Kaggle for the model is as follows.

Interestingly, Kaggle gave a better score on regression models. It may be the reason that for ground truth value provided by Kaggle, there are numbers like 0.000034 which is very small. On the contrary, our classification and regression model would only give 0 to those values because of less likelihood of making purchases. The differences between such values render lower scoring for our classification and regression model .

However, from a business point of value, we think it provides more value to small business owner by providing them a model that can do two functions simultaneously, so that they can make general marketing promotions based on classification results: whether they would buy or not, and specific marketing segmentations based on regression results: the spending by each customer.

3. Interpretation

3.1 Metrics Interpretation

Below are the model performance metrics.
When using both classification and regression model, the results are as below:

	Classification		Regression	
	Accuracy	F1- score	MSE	MAE
XGBoost	0.9934	0.9903	15.2037	1.7798
LightGBM	0.9945	0.9917	0.7373	0.6133

When using only regression model, the results are as below:

	Regression	
	MSE	MAE
XGBoost	0.1169	0.0131
LightGBM	0.078	0.012

We noticed that, for both two models, even though lightGBM has better MSE and MAE performance, however, the scoring by Kaggle is higher for the XGBoost results. We inferred that LightGBM may have overfitting issues for the regression modeling. All in all, these two learning algorithms are quite compatible in overall performance.

3.2 Business Values
Based on the above metric results, we have proved the exceptionally high performance on the selected models.

With our model implemented, we will be able to help small business owners who operate on Google Analytics platform understand their customers and predict future revenue. Essentially Google Analytics will be able to verify the concepts of ‘small business laboratory’ and attract more small business owners to join the Google Analytics platform and refine the analytics ecosystem.

4. Future Improvement

4.1 Limitations

The imbalanced data present a challenge, as about 80% of the customers actually spend nothing and the revenue comes from only the remaining 20%. To address this issue, we implemented classification and regression in the modeling process. The result improved but it took a longer time to run. Thus, we decide to leave it to the small business owners to decide if they want to trade off efficiency with accuracy.

The feature selection process is very subjective, some of the features are chosen based on our domain knowledge, which can vary from person to person, business to business.

Also, the data itself is in time series format, but we didn't actually take time into account when making predictions. Instead, we only use timestamps as a way to label and split the dataset. Thus, we might want to try adding time into the model for further explorations.

4.2 Future Improvement

The most important improvement is to get our model compatible with streaming data. Currently we train our model based on static data retrieved from a fixed time frame, but in reality data is updated every second. It is not very efficient to train the model again and again once new data comes in, but we only want to make prediction on the new part and update our existing results.

If that is viable, the next step is to detect decaying model. Streaming data will vary from day to day, which means that the good model today might not perform as well tomorrow. Thus, we need to develop certain metrics to track model performance and warn us once it decays. Then we can modify our model and ensure that our model remain well functional.