Cheng Miao
Professor: Amir Hossein Jafari
Capstone Final Project Report
November 28th, 2018

Google Analytics Customer Revenue Prediction

1. Abstract:

As the demand Deep Learning techniques increases nowadays, many problems need neural network make prediction, etc. While some of problems use traditional machine learning model, I resort to use neural networks. In this final project, I propose to learn a new Deep Learning Framework that predicts the customer's revenue based on data from the Google store. To realize this idea, I propose and implement a loss function for deep network in Pytorch. I also comprehensively studied baseline methods and discuss our qualitative result and properties with them. At the same time, the project needs to implement the LightGBM model to predict customer's revenue.

2. Introduction:

The 80/20 rule has proven true for many businesses–only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies. In this project, the challenge is to analyze a Google Merchandise Store (also known as Google Store, where Google swag is sold) customer dataset to predict revenue per customer. The outcome will be more actionable operational changes and a better use of marketing budgets for those companies who choose to use data analysis on top of GA data. The data has 50 columns, with "transaction Revenue" as the target to explore the relationship between the rest of the dataset. The dataset is obtained from Kaggle, which has factors such as Visitor ID, Country, Transaction Revenue, etc.

3. My approach:

In this section, I first clarify how I take the "Transaction Revenue" out to determine as the target value. Then working on data cleaning, a lot of data has no value, I need to drop those data and change the variable "date" into a suitable format. Then working on data exploratory analysis method to determine the relationship between "transaction revenue" verse another factor. Working on some useless data and not exiting data, then delete these useless data immediately. Using LightGBM and Random Forest to determine which factor has importance contributing for the model. I state how I learn framework, loss function and weight of the pytroch framework. I also report how I prepare dataset for training in Pytorch, make prediction on test dataset and validate on validation dataset, with some notable detail of the implementation.

4. Problem Statement:

Based on the data, use Machine Learning method LightGBM, Random Forest and Deep Learning method "Multilayer Feed-Forward Neural Network" to establish a model to predict revenue per customer. The goal is to understand the difference between Neural Networks and general machine learning (Ensemble Learning). Although both of these models could do regression, I

want to understand the advantages and disadvantages of different model, the applicable environment, etc.

5. Data Preprocessing:

1. Storing Data in JSON Format in Python

In [8]:
```
import pandas as pd
import json

# Define COLUMNS
JSON_COLUMNS= ['device', 'geoNetwork', 'totals', 'trafficSource']
print ("Starting read data")
data=pd.read_csv("/Users/richardmiao/Desktop/Capstone/all/train1.csv",sep=',',header=0,\
                 converters={column:json.loads for column in JSON_COLUMNS})
print('Load Data in Json, that takes long time')
print ("Done reading data")
```

In machine learning, we may need to input a lot of data. In order to speed up the process of reading data, we often pre-define the columns and read dataset from local directory. To save the pre-trained feature columns, we can use the JSON format. Then processing data in JSON, I did change type into multiple columns, the object of json_normalize is dict, and the object of json_loads is string.

2. Change Data Format

In [3]:
```
# change date format, convenient in statistics

data['date'] = data['date'].astype(str)
data['date'] = data['date'].apply(lambda x:x[:4]+"-"+x[4:6]+"-"+x[6:8])
data['date'] = pd.to_datetime(data['date']) # timestamp
data['month'] = data['date'].apply(lambda x:x.strftime('%Y-%m'))
data['week'] = data['date'].dt.weekday

# transactionRevenue convert
data['transactionRevenue']=data['transactionRevenue'].astype(float).fillna(0)
```

Change the string of numbers in the date column to the format of the normal reading date (ex: from 20160730 to 2016-07-30), which is convenient for future statistical work. At the same time, find the target value and convert the data format about the target value. Also converted the data type of other features for future used.

3. Drop columns

In [6]:
```
# one column has same value, so drop columns
# print those columns who contain same value.

for r in data.columns:
    a = data[r].value_counts()
    if len(a)<2:
        print(r)
        print(a)
        print('----')

# value is constant, get rid of these, I have 38 columns left.
data = data.drop(['socialEngagementType','browserSize','browserVersion','flashVersion',\
                  'language','mobileDeviceBranding','mobileDeviceInfo','mobileDeviceMarketi
                  'mobileInputSelector','operatingSystemVersion','screenResolution','screen
                  'latitude','longitude','networkLocation','visits','adwordsClickInfo.crite
```

```
socialEngagementType
Not Socially Engaged    99999
Name: socialEngagementType, dtype: int64
----
browserSize
not available in demo dataset    99999
Name: browserSize, dtype: int64
```

List the columns which has the same value “not available in demo dataset”. These values do not make sense, so those columns have been removed.

In [15]:

```
# drop missing value columns

for r in data.columns:
    a = len(data[r][pd.isnull(data[r])])/len(data)
    if a>0.8:
        print(r)
        print(a)
        print('----')

# starting drop columns
data = data.drop(['adContent','adwordsClickInfo.adNetworkType','adwordsClickInfo.gclId','adwordsClickInfo.isVideoAd
```

```
adContent
0.986749867498675
```

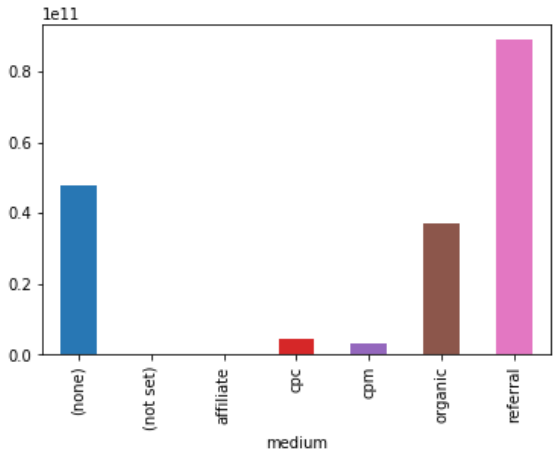
First write a for loop to list the value of the data which is the null, and then set a value, list these features, and then drop the list columns. Finally clean out the data for future use.

```
data.head()
```

	channelGrouping	date	fullVisitorId	sessionId	visitId	visitNumber	visitStartTime	browser	deviceCategory	isMobile	mobile
0	Organic Search	2016-09-02	1.131660e+18	1131660440785968503_1472830385	1472830385	1	1472830385	Chrome	desktop	False	
1	Organic Search	2016-09-02	3.773060e+17	377306020877927890_1472880147	1472880147	1	1472880147	Firefox	desktop	False	
2	Organic Search	2016-09-02	3.895550e+18	3895546263509774583_1472865386	1472865386	1	1472865386	Chrome	desktop	False	
3	Organic Search	2016-09-02	4.763450e+18	4763447161404445595_1472881213	1472881213	1	1472881213	UC Browser	desktop	False	
4	Organic Search	2016-09-02	2.729440e+16	27294437909732085_1472822600	1472822600	2	1472822600	Chrome	mobile	True	

Till now, there are only 32 features left.

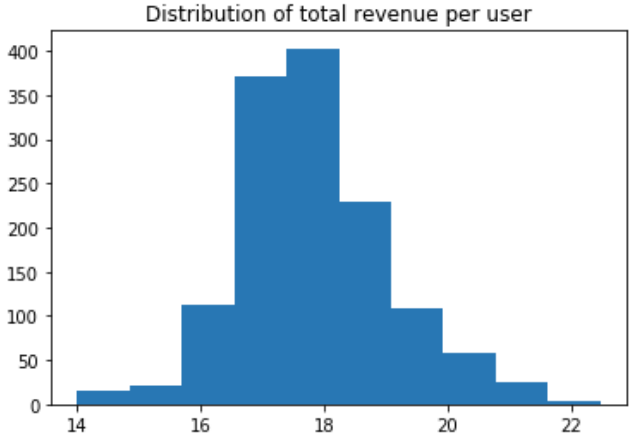
6. Exploratory Data Analysis



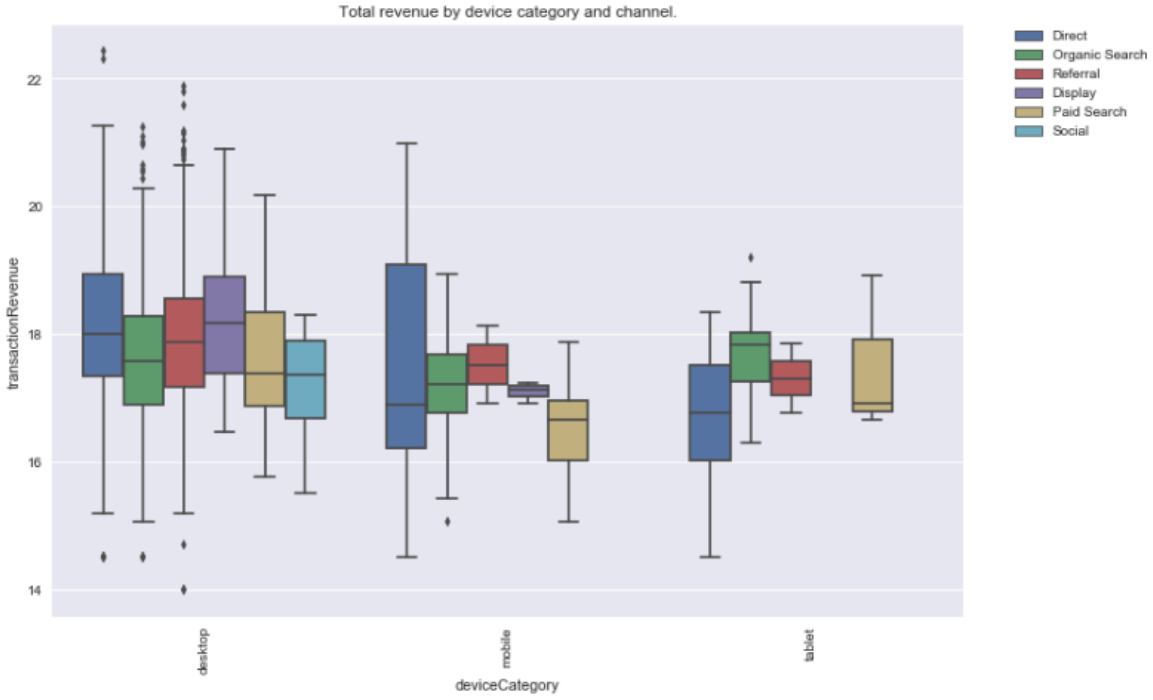

```
month
2016-08    3.114720e+10
2016-09    4.477520e+09
2016-10    5.961130e+09
2016-11    3.443621e+10
2016-12    2.612220e+10
2017-01    8.254440e+09
2017-02    9.046760e+09
2017-03    4.573590e+09
2017-04    4.063360e+09
2017-05    1.615542e+10
2017-06    2.939814e+10
2017-08    8.304940e+09
Name: transactionRevenue, dtype: float64
.........................................
week
0    2.257004e+10
1    3.929821e+10
2    3.274166e+10
3    1.966774e+10
4    5.708245e+10
5    7.979720e+09
6    2.601090e+09
Name: transactionRevenue, dtype: float64
```

The “traffic source” statistics are shown above. The main sources of traffic source are Organic Search, Social, Direct and Referral. At the same time, Desktop devices have an absolute advantage, and non-mobile counts is significantly larger than mobile. The main sources of traffic source are the United States, Asia, Europe. In the media, the traffic source of Organic

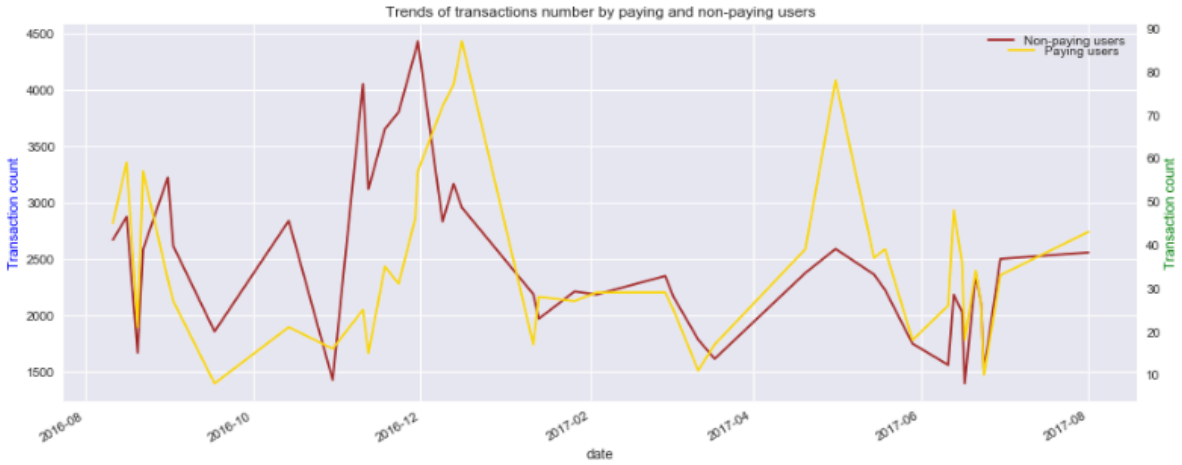
and Referral are pretty large. In a week, Friday and Saturday have the least number of sources during the week.



it is more useful to see a distribution of total revenue per user, and we could see the total revenue per user focus on 16 to 19 presenting a normal distribution.

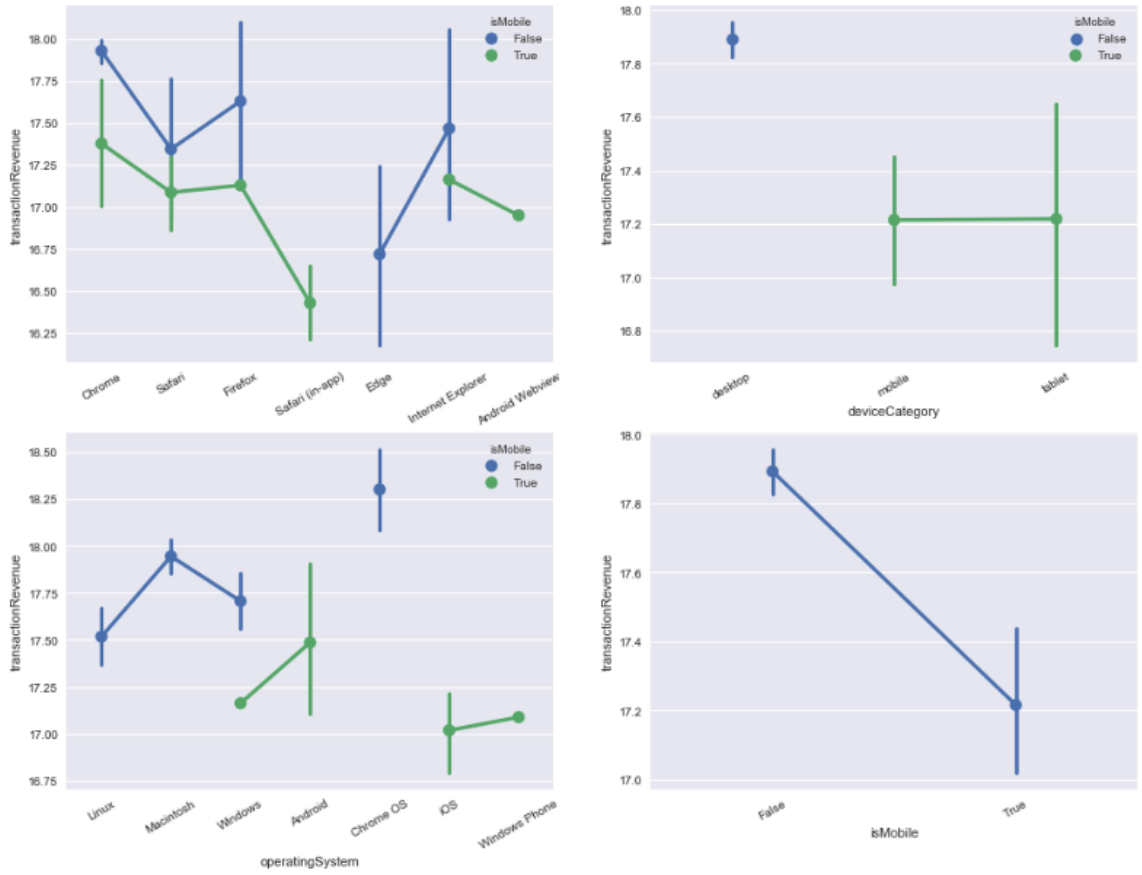


We can see that revenue comes mostly from desktops. Social, Affiliates and others aren't as profitable as other channels.

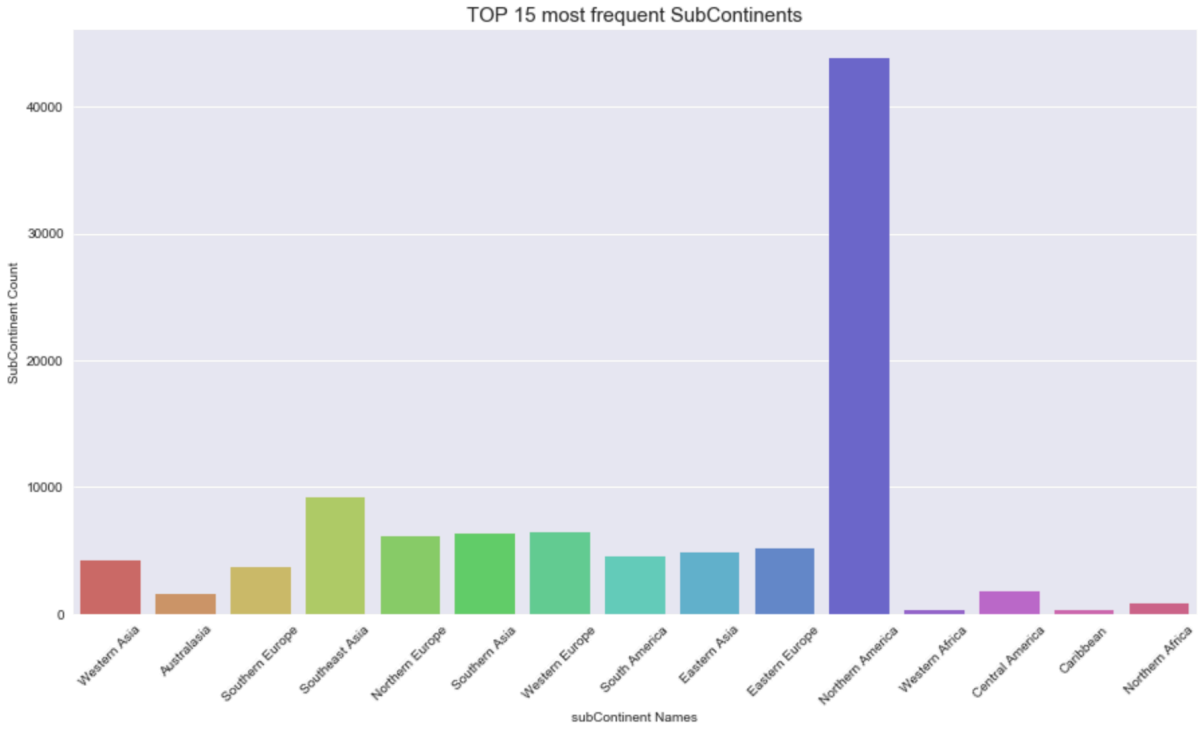


It is that trends of non-paying users and paying users of paid transactions are almost similar. It is worth noticing that there are several periods when the number of non-paying users was significantly higher that the number of paying users, but it didn't influence total sums.

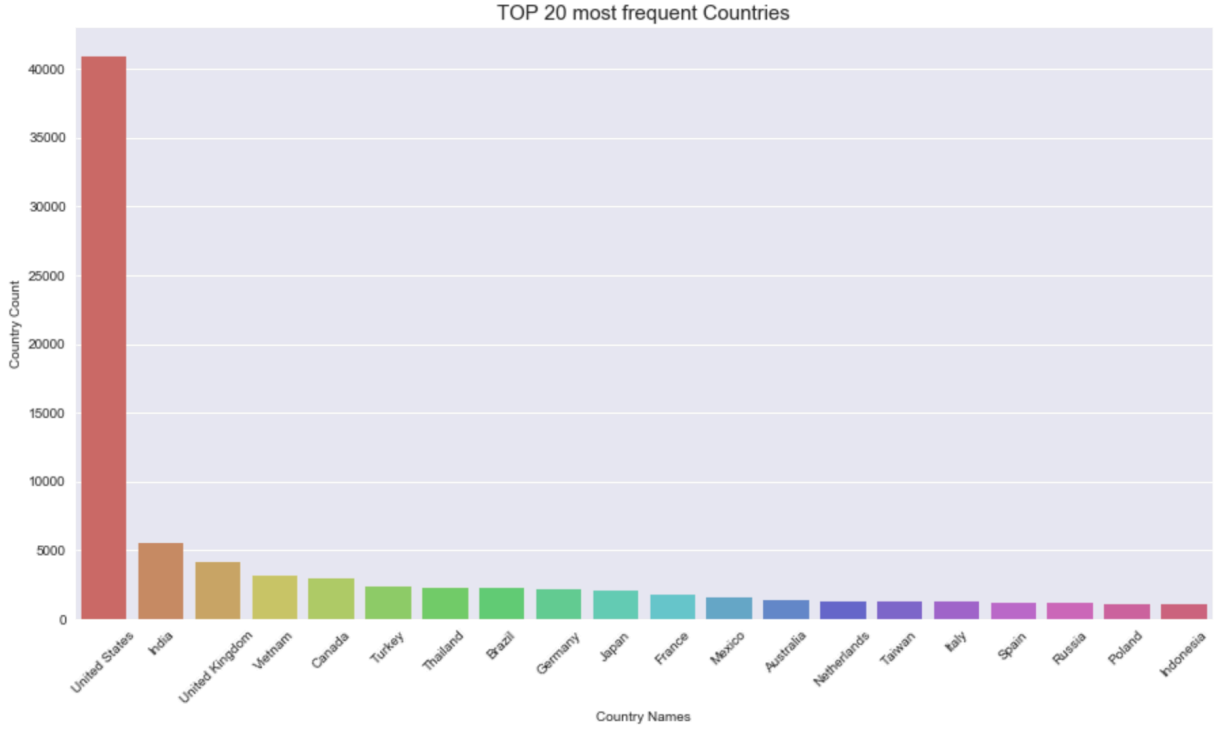
```
Mean revenue per transaction
```



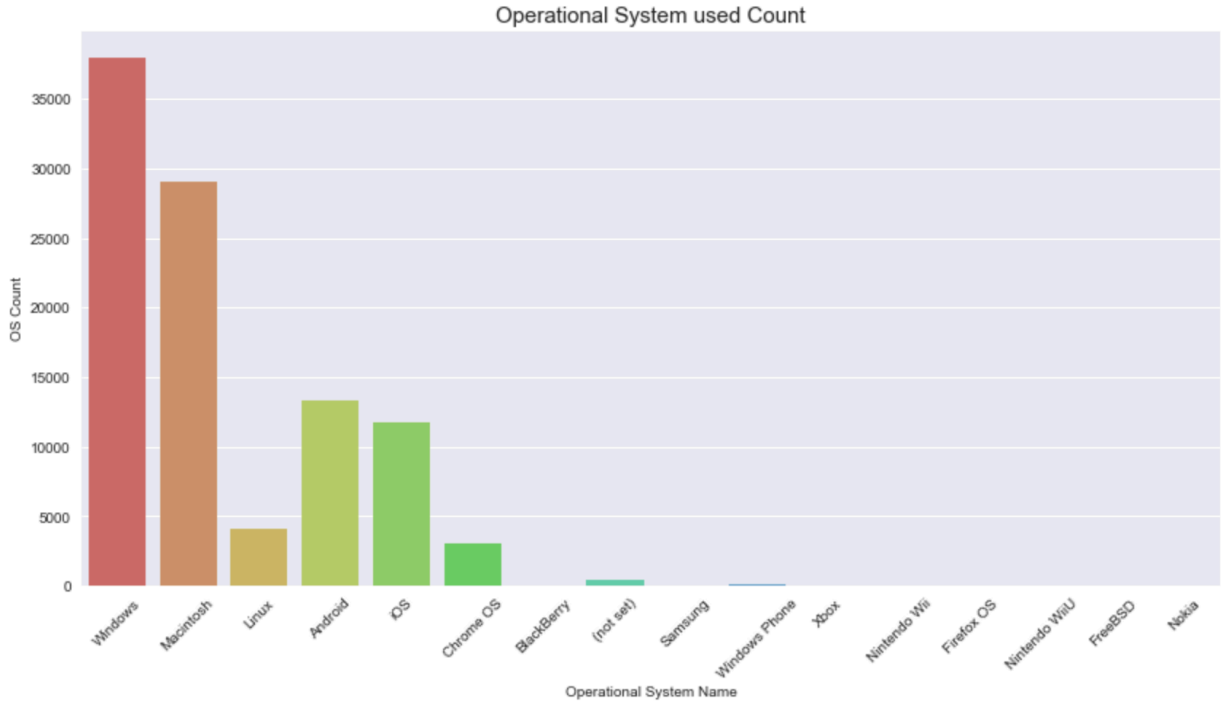
The top-left graph shows devices on Chrome, Safari and Firefox could bring profit. And left corner graph looks that devices on Chrome OS and Macs bring most profit. In the top-right graph, it looks mobile and tablet could bring most profit.



This graph shows the top 15 most frequent Sub Continents, as we could see, the Northern America has the most frequencies in the world, Southeast Asia, Western Europe, northern Europe, Southern Asia, South America and Eastern Asia seems relatively close. The rest of sub continents are also relatively close. Only North America stands out.



The above graph shows the top 20 most frequent countries. The United States stands out in the world and has the highest frequency. The remaining 19 countries are relatively close, indicating that the United States is the main consumer country and can bring more profits.



This graph shows the operational system used count bar chart. Windows and Mac operational system there are the top two operational system, others are relatively small, but Android and IOS mobile phones also have a small proportion. Indicating that most people are based on Desktop platforms, mobile phones and other platform Still a minority. The desktop platform brings more profit.

7. Model

1. LighGBM (Light Gradient Boost Machine)

The reason for choosing this model is that Kaggle competition requires this. Through research, LightGBM (Light Gradient Boosting Machine) is a framework for implementing GBDT (Gradient Boost Decision Tree) algorithm, which supports efficient parallel training and has the following advantages: faster training speed, lower memory consumption, better accuracy, distributed support, and fast processing of large dataset.

In [29]:
```
X=data_
y=data["transactionRevenue"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
```

First determine my data frame, import “train_test_split” function to split the data, and divide it into train, test, and validation. First, the data set is split into 80% for train, 20% for test, and then the 80% train is divided into 75% train, 25% validation. Train test split will avoid over fitting.

```
estimator = lgb.LGBMRegressor(num_leaves=31)

param_grid = {
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [10, 20, 40],
}

gbm = GridSearchCV(estimator, param_grid, cv=3)
gbm.fit(X_train, y_train)

print('Best parameters found by grid search are:', gbm.best_params_)
```

```
Starting predicting...
The rmse of prediction is: 2.039396145191303
Feature importances: [127, 76, 25, 12, 0, 44, 36, 18, 14, 38, 14, 136]
Starting training with custom eval function...
[1]     valid_0's l2: 4.38725   valid_0's RMSLE: 0.382336
Training until validation scores don't improve for 5 rounds.
[2]     valid_0's l2: 4.34299   valid_0's RMSLE: 0.378514
[3]     valid_0's l2: 4.30754   valid_0's RMSLE: 0.377066
[4]     valid_0's l2: 4.27729   valid_0's RMSLE: 0.37687
[5]     valid_0's l2: 4.2536    valid_0's RMSLE: 0.377813
[6]     valid_0's l2: 4.23376   valid_0's RMSLE: 0.379385
[7]     valid_0's l2: 4.21987   valid_0's RMSLE: 0.381486
[8]     valid_0's l2: 4.20675   valid_0's RMSLE: 0.383569
[9]     valid_0's l2: 4.19588   valid_0's RMSLE: 0.385673
Early stopping, best iteration is:
[4]     valid_0's l2: 4.27729   valid_0's RMSLE: 0.37687
Starting predicting...
The rmsle of prediction is: 0.3768701275205428
Best parameters found by grid search are: {'learning_rate': 0.1, 'n_estimators': 20}
```

```
gbm=lgb.LGBMRegressor(num_leaves=31,
                      learning_rate=0.1,
                      n_estimators=20)

gbm.fit(X_train, y_train,
        eval_set=[(X_test, y_test)],
        eval_metric='l1',
        early_stopping_rounds=5)

print('Starting predicting...')
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration_)
```

Here I used grid search to select the appropriate model parameters. Here the learning rate is 0.1 and the n_estimators selection is 20. Then bring this parameter into the LGBM Regressor to train my model. The rest of the parameter selection is based on some references and other competitors' codes.

```
Did not meet early stopping. Best iteration is:
[18]    valid_0's l2: 4.15914   valid_0's l1: 0.462884
Starting predicting...
The rmse of prediction is: 2.039396145191303
```

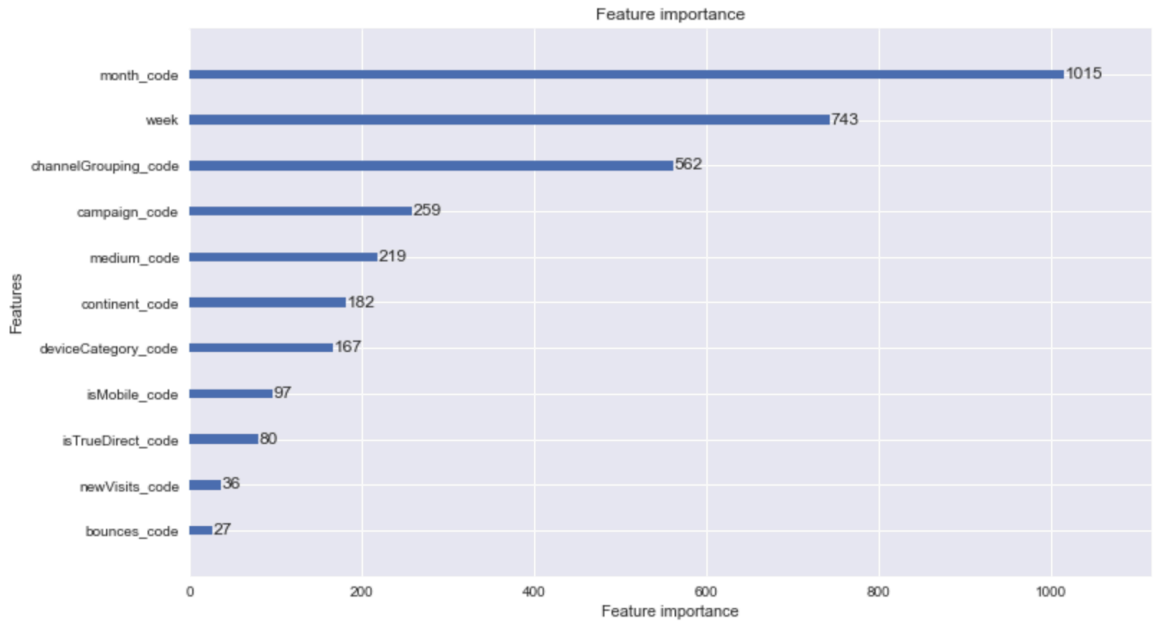
figure 1

```
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.0096 valid_1's rmse: 2.0525
Early stopping, best iteration is:
[59]    training's rmse: 2.01491        valid_1's rmse: 2.05153
```

figure 2

Figure 1 is the result of my own work based on the reference LightGBM model, and Figure 2 is the code belonging to other competitors of Kaggle. As can be seen, the two RMSEs on validation dataset are very close. By comparing my model with other competitors' models, although the resulting RMSE is similar, other competitors model have more choices in parameter selection. After I plan to add more parameters into grid search method, the grid search selection parameter will take more time. However, there is no results are obtained, so I decided to keep the original parameter selection will continue my work.

```
1 lgb.plot_importance(model, figsize=(12, 7))
2 plt.show()
```

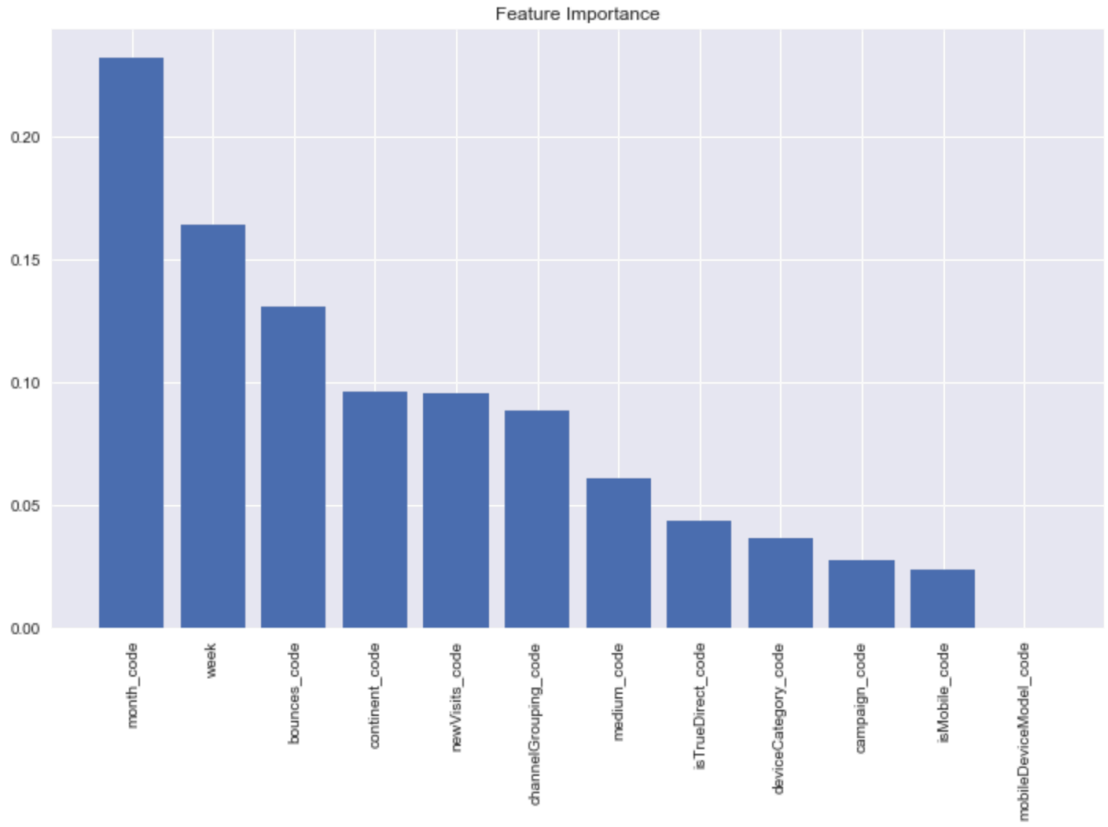


This figure shows the importance of these 12 features. From the important to the secondary, I can see that month is the most important, and mobile device model is the most unimportant.

2. Random Forest

The random forest what I used here for my capstone project is to verify that the RMSE and feature importance obtained by using Light GBM are justified.

```
prediction: [0.97870904 0.         1.71891184 ... 0.         0.         0.        ]
RMSE 2.097921376837861
```



From these two figures we could see the RMSE is still close, the errors between those two RMSE is pretty small 1.9%. And also, in feature importance plot, the feature importance is also relatively consistent, but some of features are of different. Because these are two different models, although it is an Ensemble Learning model, each model may have a number of different node splits.

8. Principal Component Analysis (PCA)

Use PCA to make the data on dimensionality reduction. In order to have the smallest error and extract the main features. Principal component analysis is a dimension reduction method is divided into three steps: calculating the covariance matrix and decomposing the covariance matrix. Finally, selecting the eigenvector corresponding to the eigenvalue with the largest absolute value of the eigenvalue as the transformation matrix, and reducing the original data.

```
1  # reduce dimension of features
2  from sklearn.decomposition import PCA
3
4  pca=PCA(n_components=0.95) #n_components<=n_classes-1
5
6  pca_data=pca.fit_transform(data_standardized)
7  print (pca_data[1])
```

```
[-1.21675246  0.32847542  1.82301643 -1.77005558  0.85666901  0.72287247
 -0.52267115  1.12312211]
```

As shown in the figure above, the data has been reduced from 12 features to an array of 8 columns.

9. Data Standardized – Reduce Loss

```
from sklearn import preprocessing
data_standardized=preprocessing.scale(data_)
print("done")
```

Data standardized and normalization. There is a difference between data normalization is a typical practice of data standardization, that is, the data is uniformly mapped to the interval [0,1]. The standardization of data refers to scaling the data to make it fall into a specific interval.

10. Multilayer Feed-Forward Neural Network

1. Prepare Data for Pytorch Framework used

```
import torch
import torch.nn.functional as F  # implementation activation function
import matplotlib.pyplot as plt
from torch.autograd import Variable

X=pca_data
y=data["transactionRevenue"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
```

```
#Convert data for pytorch training more easy
X_train=torch.from_numpy(np.array(X_train))
y_train=(torch.from_numpy(np.array(y_train))).view(59999,1)
X, y = Variable(X_train,requires_grad=True), Variable(y_train,requires_grad=True)
```

```
# convert data type float
x=X.float()
y=y.float()
```

Re-split the data after dimensionality reduction into train, test, and validation dataset. And then is to convert the data into a torch format suitable for the Pytorch framework, in order to work on training and make prediction. Before inputting to the neural network as input, I have to convert the data to float.

2. Define Model

Here I define the Neural Network model refers to a lot of research online, which will be listed in the reference. Since input data is reduce dimension, so that n_feature should be 8, 9 is hidden layer. This parameter can be conditional. The

```
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as tf


# define model
class net(nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(net, self).__init__()
        #self.poly = nn.Linear(6,1)
#         self.l1 = torch.nn.Linear(8, 6)
#         self.l2 = torch.nn.Linear(6, 5)
#         self.l3 = torch.nn.Linear(5, 4)
#         self.l4 = torch.nn.Linear(4, 3)
#         self.l5 = torch.nn.Linear(3, 2)
#         self.l6 = torch.nn.Linear(2, 1)
        self.hidden = torch.nn.Linear(8, 9)
        self.predict = torch.nn.Linear(9, 1)


    def forward(self, x):
        x = tf.relu(self.hidden(x))  #  function (linear value of hidden layer)
        x = self.predict(x)   # output
        return x
```

hidden layer can be adjusted from 1 to 100. Due to its limited capacity, it is temporarily set to 9. The output is 1 because I only have one output (prediction)

3. Main function: criterion, optimizer, prediction

```
net = net(n_feature=8,n_hidden=9,n_output=1)

if __name__ == '__main__':
    #W_target = torch.FloatTensor([0.5, 3, 2.4]).unsqueeze(1)
    #b_target = torch.FloatTensor([0.9])

#     if torch.cuda.is_available():
#         model = poly_model().cuda()
#     else:
#         model = poly_model()

# define criterion and optimizer
    criterion = torch.nn.MSELoss(size_average=False)

    optimizer = torch.optim.SGD(net.parameters(), lr=0.0001)
#x=torch.optim.SparseAdam
#epoch = 0
# after done define model, starting training
for t in range(500):
    prediction = net(x)     # give model training dataset: x, output the predict value

    #output = model(x)
    loss = criterion(prediction, y)# calcuate the loss between prediction and y
    print_loss = loss.item()

    optimizer.zero_grad()   #  clear out the last step the remaining value of parameters
    #loss.backward()         #  loss backward, calculate updated value
    loss.backward
    optimizer.step()
# output loss
print (loss)
```

```
/anaconda3/lib/python3.6/site-packages/torch/nn/functional.py:52: UserWarning: size_average and
eprecated, please use reduction='sum' instead.
  warnings.warn(warning.format(ret))
```

```
tensor(269253.0938, grad_fn=<SumBackward0>)
```

Since I tried many times and borrowed a lot of open source resources, I set the criteria (loss function) and optimizer. The learning rate was set to 0.0001, and this value can be adjusted. Next is the training model. After 500 cycles in a for loop, I can get the loss value and prediction. The loss here is greater than 269,000, which is a very large loss value, indicating that the model may not be very good, or the dataset is not suitable for neural networks.

Next, I continued to try to adjust the loss. I want to make the loss smaller. I re-define epoch=2000, train data again, and then observe the loss value, which is still very large. The code and output value as a graph are shown below.

```
epochs = 2000
optimizer = torch.optim.SparseAdam(net.parameters(), lr=0.0001)
for epoch in range(epochs):

    epoch +=1
    #increase the number of epochs by 1 every time
    #inputs = Variable(torch.from_numpy(x_train))
    #labels = Variable(torch.from_numpy(y_correct))

    #clear grads as discussed in prev post
    optimizer.zero_grad()
    #forward to get predicted values
    outputs = net(x)
    loss = criterion(outputs, y)
    loss.backward()# back props
    #optimizer.step()# update the parameters
print('epoch {}, loss {}'.format(epoch,loss.data[0]))
```

```
epoch 2000, loss 269253.09375
```

11. Prediction

LightGBM:

```
In [24]:  1  print('LightGBM results:',y_pred)
```

```
LightGBM results: [0.39616228 0.16217122 0.53179822 ... 0.16217122 0.21975678 0.16217122]
```

Random Forest:

```
 7  print('prediction:',rf_pred_test)
 8  print('RMSE',np.sqrt(metrics.mean_squared_error(y_test,rf_pred_test)))
```

```
prediction: [1.07289045 0.         2.00610394 ... 0.         0.         0.        ]
```

Multilayer Feed-forward Neural Network:

```
1  print ('MLP results:',prediction)
```

```
MLP results: tensor([[ 0.2365],
        [ 0.3104],
        [-0.1188],
        ...,
        [ 0.0150],
        [ 0.0648],
        [ 0.0783]], grad_fn=<ThAddmmBackward>)
```

Light GBM prediction results: the prediction is following the officially suggest logic: calculating logarithm of predicted revenues. The rest of the prediction are also understandable.

12. Conclusion:

LightGBM: the framework of distributed gradient boost framework based on decision tree algorithm. Using the histogram algorithm, which takes up less memory and has less complexity in data separation. It uses a leaf-wise growth strategy to find a leaf with the highest split gain from all current leaves, then split. Keep cycling.

Random Forest: it performs well and has high precision on large dataset; it is not easy to have over-fitting; it can get the order of importance of features; it can process both discrete data and continuous data; it does not need to be normalized.

MLP: multi-layer feed forward neural network: The network essentially implements a mapping function from input to output, and mathematical theory has proven to have the ability to implement any complex nonlinear mapping. This makes it particularly suitable for solving complex problems with internal mechanisms.
However, the learning speed of this algorithm is kind of slow. Because this algorithm is essentially a gradient descent method, the optimized function is very complicated. Network training is more likely to fail.

A lot of knowledge can be learned through this project and research. First, after seeing MLP training, the loss is particularly large. So, I learn from online source is to normalize the data or reduce dimension. Or we could the framework configuration is not good enough. I try to adjust optimizer but did not achieve my expectation. It is also possible that my dataset is not suitable for doing neural networks, so I totally understand why Kaggle Competition required to do LightGBM. The hidden layer is adjusted according to the situation, so that the loss could

be reduced, and the accuracy of the model is improved. I also learned the new algorithm which is LightGBM, learning that this model can handle many complex datasets. I have a deeper understanding of Random Forests.

13. Advice to Companies:

1. Customer spending is concentrated on non-mobile platforms, but mobile platforms have great potential.
2. With the rapid development of mobile phones, users will increase their consumption on mobile platforms.
3. Due to the significant number of users in North America, some applications in other regions can be added.

14. Reference

Data: https://www.kaggle.com/c/ga-customer-revenue-prediction/data

Code source:

1. LightGBM Partial EDA Plots, Kaggle competition, https://www.kaggle.com/sammir/gstore-data-preparation-analysis-random-forest/notebook
2. GitHub source, https://github.com/Microsoft/LightGBM/blob/master/examples/python-guide/sklearn_example.py
3. LightGBM second try model kernel 25, Kaggle competition, https://www.kaggle.com/shivamb/exploratory-analysis-ga-customer-revenue
4. Main Model MLP- Feed Forward Neural Network, https://github.com/MorvanZhou/PyTorch-Tutorial
5. Linear Regression Pytorch, https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2
6. MLP, https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
7. MLP Pytorch Framework, Amir Jafari, https://github.com/amir-jafari/Deep-Learning/blob/master/Pytorch_/2-nn_module/3_nn_optim.py
8. Tune Hyper-parameters, https://blog.csdn.net/liuweiyuxiang/article/details/84556275
9. MLP basic knowledge, https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
10. MLP Feed forward NN in Chinese, https://blog.csdn.net/yjbuaa/article/details/81714103