Technical Test

First and foremost, we see that there is no unique identifier for each observation nor a timestamp, we thus can't know if there might be some duplicates in the data or if two identical lines just represent two different observations that happen to be identical. With this lack of information, we choose to consider that each line represents a different observation.

With this hypothesis, we can answer the questions :

Preliminary questions:

1. **The margin being defined as (revenue - cost) / revenue, what is our global margin based on this log?**

The total margin of this log is 21081 if we consider that the margin is zero when the revenue is zero.

2. **Can you think of other interesting metrics to optimise?**

With the information at hand, we can also optimise the net profit which is the difference between the revenue and the cost. It will allow us to take into account the losses when there is no revenue which is not possible with the margin.

3. **What are the most profitable Operating Systems?**

We group the data by operating systems and we sum up the margin or profit in each group. iOS is then the most profitable O.S followed by Windows and Android respectively.

Machine Learning questions:

1. **How would you use this historical data to predict the event 'we benefit from a revenue' (ie revenue > 0) in a dataset where the revenue is not known yet?**

First we look at the quality of the data, clean it, then we extract some features and then we build a classifier that will predict from the features if the revenue is superior to zero or not.

2. **Compute the prediction accuracy of a well chosen algorithm and comment the results. Do not hesitate to describe your methodology.**

Data cleaning:

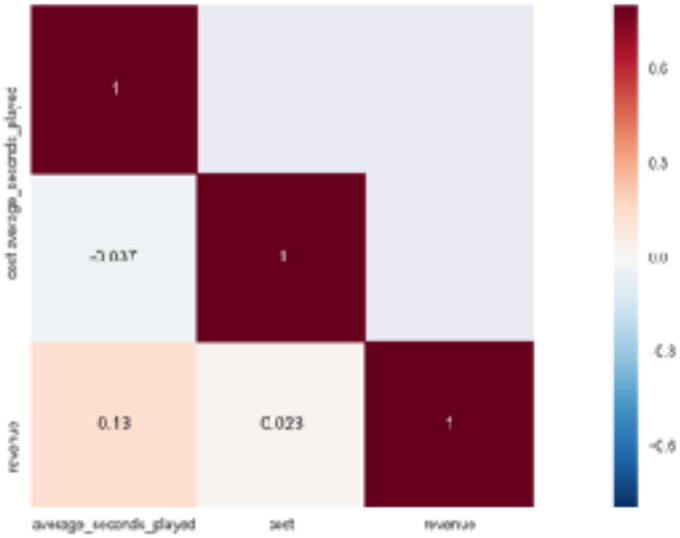
When present, the values seem consistent and realistic, the only concern for the data quality is then the absence of values.

When we look at the data, we see that the columns creative_id, cost, revenue and user_operating_system don't have null values.

The column user_device has 8 null values. If we plot those lines, we can see that for 5 of them, the corresponding operating system is BSD which is a computer O.S. We thus allocate the value 'PersonalComputer' to those lines. For the 3 remaining, the corresponding operating system is Windows which doesn't correspond to one operating system alone. With no further information, we can either leave it like this or fill it with 'PersonalComputer'.

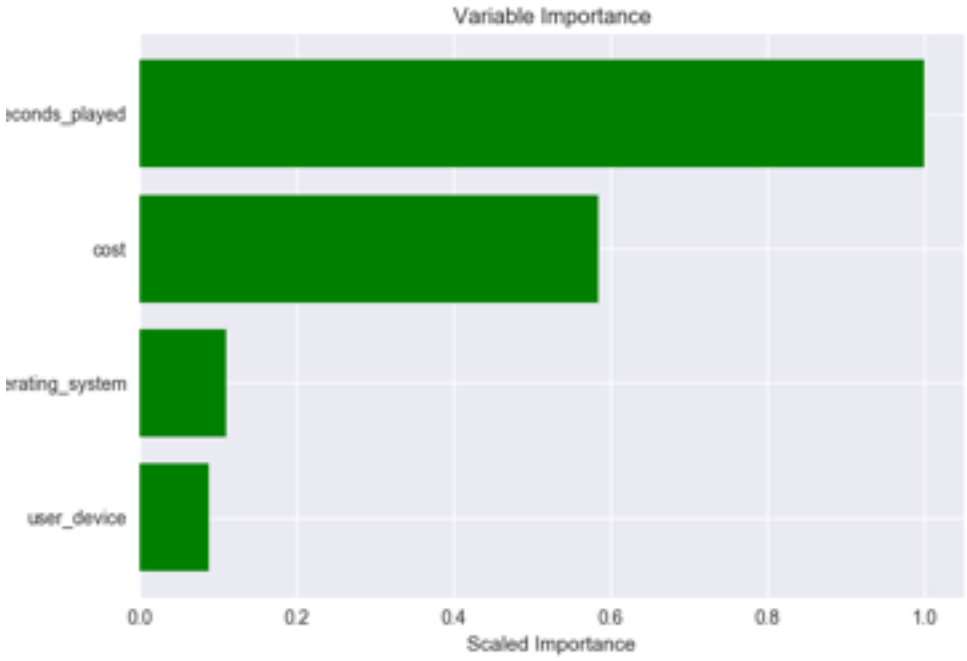
There is an 'unknown' category in the user_operating_system categorical column. With no further information, we leave it like this.

The biggest concern is the high number of null values in the average_seconds_played column. Even if it is intentional because it indicates that we don't know the habit of the user it still makes it difficult to build a model with a feature having only one third of none null values.



The figure above shows us the correlation between the quantitative values, the correlation scores are not that important.
We can first see how a model performs with this limitation.

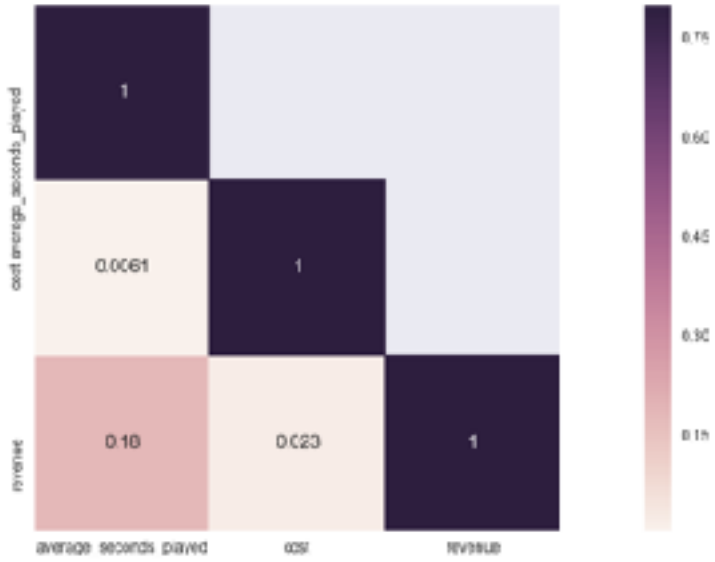
First modelling:
Features: user_operating_system, user_device (cleaned), cost, average_seconds_played (raw).
Label: isRevenue (boolean True if revenue>0 False otherwise)
Model : Since there are some categorical features, we would choose a tree-based model because it can handle those kind of features better. As a start we train a simple tree on a training set with a cross validation to avoid over-fitting and use a test sample to look at the performances. It is not that good so we can either fine-tune the tree or use an ensemble method such as a random forest model. The latter was tested with 200 trees and 15 cross validation folds and even if the performance was increased, it is still not that great (we have a AUC score of 0.6).



The most important feature in the random tree model is the average_seconds_played then comes the cost then the operating system and finally the user device.

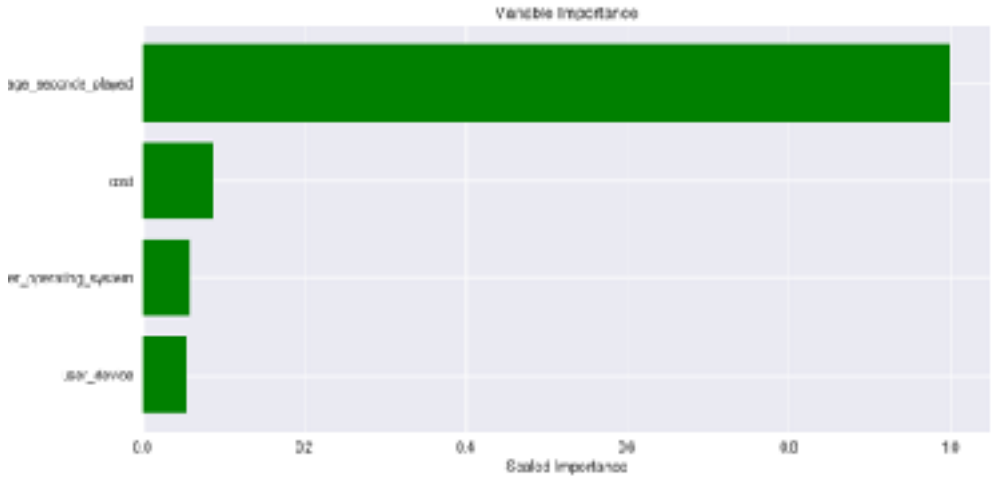
To improve the accuracy of the model we can either fine-tune this ensemble model or try to work on the missing values to improve the modelling. Since the average_seconds_played feature is that important in the model and the accuracy really below expectations, it is worth reworking this field.

Filling the missing values of the average_seconds_played field:
This task can be a prediction exercice in it self i.e. build a predictive model out of the observations where this field is not null and predict the missing values with this model.
Or we can group the observations by the values of other fields and use the mean of the non-null average_seconds_played values to fill for the missing values in those groups.
For the grouping task, we can use the user_device and user_operating_system categorical fields as well as bins of revenue and/or cost. There are still some (~20) values missing after applying this method.



After that we see that the correlation between the revenue and the average_seconds_played increased which is to be expected since we filled the missing values by considering the ones taken by the revenue field.

Second modelling:
Features: user_operating_system, user_device (cleaned), cost, average_seconds_played (filled).
Label: isRevenue (boolean True if revenue>0 False otherwise)
We use the same random forest model with 200 trees and 15 cross validation folds and this time we have a AUC score of 0.94 which is a lot better.



Depending on the aim of the prediction (is it more important to predict correctly the observations that will generate revenue or those that will not ?), we can also look at the precision, recall and f1 score as accuracy metrics.
On the figure above, we can see that by filling the missing values in the average_seconds_played field, we increase the importance of this variable in the model.
We can now further fine-tune the model if we want by doing a grid search on the different parameters of the model, for example the maximum depth of a tree, the minimal number of sample in a node to allow a split or the number of trees to consider.
The grid search wasn't done in this case because of a technical problem: we used the h2o library to build a random tree model (which directly works with categorical data in contrary to the scikit-learn random forest implementation) and in the actual version of h2o and python, the grid search produces an error (H2OServerError: HTTP 500 Server Error).

Another possibility is to use a boosting model that is said to perform better than random forests but is slower to train and has more parameters to tune so it was tested.

python code: (3.X version)

```
import numpy as np
import pandas as pd
import seaborn as sn
import h2o
import matplotlib.pyplot as plt
from h2o.estimators.random_forest import H2ORandomForestEstimator

data = pd.read_csv('            .csv')
print(data.user_operating_system.unique())
print(data.user_device.unique())
print(data[data[['user_device']].isnull().any(axis=1)])
data['isRevenue']=data['revenue']>0
data['roundRevenue']=round(data['revenue'], 2)
data.user_device.fillna("PersonalComputer", inplace=True)
print(data.count())
print(data[data[['average_seconds_played']].isnull().any(axis=1)])

corrMatt = data[["average_seconds_played","cost","revenue"]].corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig,ax= plt.subplots()
fig.set_size_inches(20,10)
sn.heatmap(corrMatt, mask=mask,vmax=.8, square=True,annot=True)
def computeMargin(df):
    if df['revenue']==0:
        return 0
    else:
        return (df['revenue']-df['cost'])/df['revenue']
data['margin'] = data.apply(computeMargin,axis=1)
print("The total margin is "+str(data['margin'].sum()))
OS_revenue = data.groupby('user_operating_system').margin.sum()
print(OS_revenue)
profit = data.groupby('user_operating_system').revenue.sum()-
data.groupby('user_operating_system').cost.sum()
```

```
#%% Modeling
#initiate a h2o cluster and transforme the data into a h2o-specific frame.
h2o.init(max_mem_size = "2G", nthreads=-1)
col_types=['numeric','categorical','categorical','numeric','numeric','numeric','categorical','numeric','n
umeric']#
h2ofr = h2o.H2OFrame(data,column_types=col_types)
#split the data into training, testing and validation sets.
splits = h2ofr.split_frame(ratios=[0.7, 0.15], seed=1)
train = splits[0]
valid = splits[1]
test = splits[2]

#set the columns that while be used as features and the ones we want to predict.
y = 'isRevenue'
x = list(h2ofr.columns)
x.remove(y)
for z in ['creative_id','revenue','roundRevenue','margin']:
    x.remove(z)
#x.remove('average_seconds_played')

#train a random forest regressor on the training set.
RF = H2ORandomForestEstimator(ntrees=200,seed=1,nfolds=15)
RF.train(x=x, y=y, training_frame=train)


#look at the performances of the trained random forest model on the test set.
RF_perf = RF.model_performance(test)
print(RF_perf)

#plot a figure of the variable importances in the trained model.
fig, ax = plt.subplots()
variables = RF._model_json['output']['variable_importances']['variable']
y_pos = np.arange(len(variables))
scaled_importance = RF._model_json['output']['variable_importances']['scaled_importance']
ax.barh(y_pos, scaled_importance, align='center', color='green', ecolor='black')
ax.set_yticks(y_pos)
ax.set_yticklabels(variables)
ax.invert_yaxis()
ax.set_xlabel('Scaled Importance')
ax.set_title('Variable Importance')
plt.show()

#print in the console the varable importances.
print(RF._model_json['output']['variable_importances'].as_data_frame())

#h2o.cluster().shutdown()
```