MEF UNIVERSITY

PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES

Capstone Project

Semih Tekten

İSTANBUL, 2018

MEF UNIVERSITY

PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES

Capstone Project

Semih Tekten

Advisor: Prof. Dr. Özgür Özlük

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Predicting Facebook Ad Impressions & CPM Values
Name/Last Name of the Student: Semih Tekten
Date of Thesis Defense: ……………….

I hereby state that the graduation project prepared by Semih Tekten has been completed under my supervision. I accept this work as a "Graduation Project".

…………..
Prof. Dr. Özgür Özlük

I hereby state that I have examined this graduation project by Semih Tekten which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

…………..

Director
of
Big Data Analytics Program
Prof. Dr. Özgür Özlük

We hereby state that we have held the graduation examination of Semih Tekten and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Prof. Dr. Özgür Özlük	………………………..

ACADEMIC HONESTY PLEDGE

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name	Date	Signature
SEMİH TEKTEN	…………..	………………………..

EXECUTIVE SUMMARY

PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES

Semih Tekten

Advisor: Prof. Dr. Özgür Özlük

DECEMBER, 2018, 168 pages

It is estimated that there are more than two billion active users on Facebook as of the first quarter of 2018 and social media has tremendous opportunities for advertisers in terms of performance and measurability. However, for marketing managers, it is very difficult to manage all the campaigns on different marketing channels and optimize for better results. For that reason, Facebook Marketing Partners or other optimization solutions emerged in the ad-tech market. In order to improve existing optimization solutions in the market, ad impression costs will be predicted in this study by using different machine learning techniques and different algorithms. The main goal of this study is to generate a robust model for predicting CPM values on Facebook, and to use that model as an input for the existing optimization solution Adphorus offers for its clients. Adphorus is one of the Facebook Marketing Partners in the market.

Key Words: facebook ad delivery, cpm prediction, cpm forecasting, impression prediction, impression forecasting

ÖZET

FACEBOOK REKLAM GÖSTERİM ADET VE MALİYETLERİNİ TAHMİN ETME

Semih Tekten

Tez Danışmanı: Prof. Dr. Özgür Özlük

ARALIK, 2018, 168 sayfa

2018'in ilk çeyreği itibariyle Facebook aktif kullanıcı sayısının 2 milyarın üzerinde olduğu tahmin ediliyor ve sosyal medya reklam verenler için performans ve ölçümleme anlamında sayısız fırsatlar sunuyor. Ancak pazarlama yöneticileri için bu aynı zamanda daha iyi sonuç almak amacıyla farklı kanallarda çok sayıda kampanya yönetmek anlamına da gelmekte ve yorucu olmaktadır. Bu sebeplerle "ad-tech" sektöründe "Facebook Marketing Partners" ve optimizasyon çözümleri sunan şirketler doğmuştur. Bu çalışmada, piyasadaki optimizasyon çözümlerini geliştirmek için farklı makine öğrenim teknikleri ve algoritmaları kullanılarak Facebook reklam gösterim maliyetleri (CPM) tahmin edilecektir. Bu projenin amacı Facebook CPM değerlerini güçlü bir şekilde tahmin ederek, bu tahminleri Adphorus'un halihazırda müşterilerine sunduğu optimizasyon algoritması içerisinde kullanmaktır. Adphorus, Türkiye merkezli Facebook Marketing Partner şirketidir.

Anahtar Kelimeler: facebook reklam dağıtımı, reklam gösterimi tahminleme, reklam gösterim maliyeti tahminleme

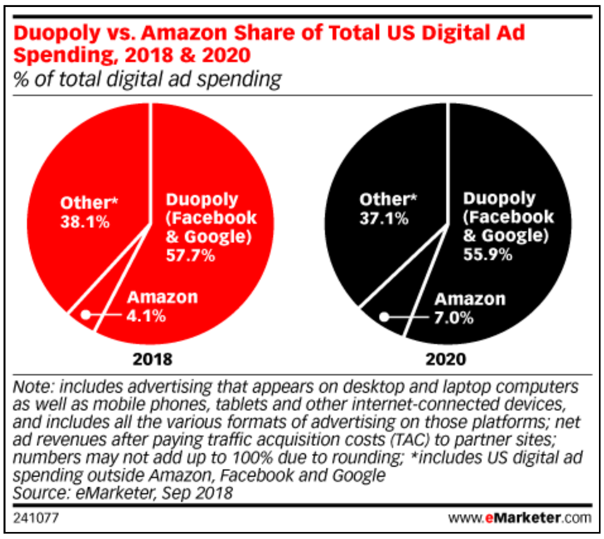
TABLE OF CONTENTS

1. INTRODUCTION

It is estimated that there are more than two billion active users on Facebook as of the first quarter of 2018 and social media has tremendous opportunities for advertisers in terms of performance and measurability. However, for marketing managers, it is very difficult to manage all the campaigns on different marketing channels and optimize for better results. For that reason, Facebook Marketing Partners or other optimization solutions emerged in ad-tech market.

Facebook Marketing Partners (FMP) are tech companies or agencies authorized by Facebook in helping advertisers to get the most efficient results from their advertisement campaigns. FMPs are experts on managing hundreds of ads, achieving more scale on campaigns, reaching new audiences, optimizing budget allocation and hitting higher metrics[1].

In ad-tech market, optimization solutions refer to controlling budgets, bids and monitoring campaigns' performance to ensure the best results on advertising spending. Best results may refer to, but are not limited to, maximizing number of desired user actions or amount of revenue. The main target of optimization solutions in the market is to improve return on ad spend while controlling advertising costs and spending. It is challenging to manage thousands of campaigns and control performance metrics at the same time. When companies do not have sufficient resources or know-how, they are able to partner up with FMPs or Google Partners and then utilize "optimization solutions" from such companies[2].



Market Share of Digital ad spending[3]

There is a clear duopoly of Facebook & Google in the digital ad spending market according to eMarketer report. Any improvement in optimization and performance increase in Google or Facebook ad delivery will have a significant impact in the market. Ad delivery is Facebook delivery system that determines to whom Facebook shows the ads, and when and where to show them. An advertiser needs to choose a target audience and an optimization event (e.g. an app install, click or purchase). Facebook tries to show ads to users in that target audience who are likely to perform that optimization event[4].

Facebook Marketing Partners or Google Partners have a huge opportunity for three reasons:

1. They can develop black-box algorithms with scarce resources without Google or Facebook needing to know the details of the algorithms.
2. Their headquarters may be anywhere in the world.
3. These companies are able to target 58% of the total digital ad spending market in spite of not being large corporations.

Optimization solutions are not only beneficial to advertiser companies, but also employees in those companies. For marketing managers, it is very difficult and tedious to manage all the campaigns on different marketing channels and optimize for better results. Managing campaigns is repetitive and requires manual work, mental capacity, number crunching and concentration. Since optimization solutions also offer automation, which reduces digital advertising efforts and labor force needed to run digital campaigns, they do reduce labor needed for managing campaigns and optimization.

By the time this project was prepared, I was working as a data analyst for Adphorus (https://www.adphorus.com/). Adphorus is an Istanbul based Facebook Marketing Partner owned by US based Sojern Inc (https://www.sojern.com/). Both companies focus on travel industry and intend to solve the problems in the travel-marketing area.

Adphorus offers its clients the tools to manage their Facebook campaigns and automated optimization solutions in order to maximize their ad spend returns. Adphorus' optimization engine, Marvin, makes necessary changes for thousands of campaigns each night. These changes may be, but are not limited to, adjusting bids and setting budgets for each campaign.

What Marvin attempts to solve is defined as an optimization problem. Given the campaign budget (Constraint), this engine targets to maximize the number of conversions. Conversion is an action that a person takes on an advertiser's website or mobile app, such as adding an item to the shopping basket, purchasing it, filling a subscription form or just visiting a product page. These actions are "desired actions" for advertisers. In other words, conversions are the actions that advertisers want their users to perform.

Advertisers may also add another constraint to Marvin, which is Target Cost per Conversion. With Target Cost per Conversion, advertisers feed Marvin how much they want to pay for a single conversion on average. The ultimate aim of Marvin is to increase the efficiency and performance of advertiser's Facebook campaigns through increasing campaign spend as much as possible while keeping the cost per conversion at or below advertiser's

Target Cost per Conversion. In other words, advertisers like to have the maximum amount of conversions with a limited budget and with a unit cost under a target threshold.

The idea and/or question behind this project can be summarized as follows:

“If we had a chance to have information regarding ad impression costs for the next day, could we improve Marvin optimization?”

Impression, sometimes called a view, an ad view or ad impression, is a term that refers to the point in which an ad is viewed once by a user, or displayed once on a web page. Impression is a common metric used by the online marketing industry. Impressions measure how often advertisers’ ads were on their target audience’s screen[5].

For the reasons explained above, ad impression costs are going to be predicted by using different machine learning techniques and different algorithms with the help of the past campaign history and Adphorus data. The main goal of this project is to generate a robust model for predicting CPM values on Facebook, and to use that model as an input for the existing optimization solution Adphorus offers for its clients.

Cost per Thousand (CPM):

> “Cost per thousand (CPM) is a marketing term used to denote the price of 1,000 advertisement impressions on one webpage. If a website publisher charges $2.00 CPM, that means an advertiser must pay $2.00 for every 1,000 impressions of its ad. The "M" in CPM represents the Roman numeral for 1,000.”[6]

The ads which will be shown on Facebook to its users are decided by auction mechanism. Advertisers are bidders and they compete for the ad impression. In other saying, Facebook does not set prices for ads on its platform. Therefore CPM prices can be considered as continuous variable such as stock prices and CPM prices always change.

In addition to CPM, advertisers monitor and consider another metric called “Cost per Thousand People Reached”. CPM counts number of impressions in the cost calculation. On the other hand, Cost per Thousand People Reached counts the number of people that saw an ad in the cost calculation. It is the average cost to reach 1,000 people[7].

2. PROBLEM DEFINITION

In this study Facebook CPM values are going to be predicted on a daily basis. There are multiple ways to define this problem. During the project, various problem definitions, different target values and features have been utilized. The reason behind the approach is to find the best results and performing model. In the end, a single problem and target has been taken into consideration.

2.1. Auto-Correlation

Before moving on with different approaches, it is necessary to explain what "predicting CPM values" mean. Users never stop visiting Facebook, there is always an opportunity for ad impressions, and advertisers (or bidders in this case) always compete for these. CPM can be considered as a continuous variable according to its definition and Facebook auction mechanism. There can be a comparison between CPM and stock prices. Both of them are continuous and highly affected from their previous value. For that reason, Auto-Correlation between CPM value and its previous values in time-series data can be assumed.

The question of "How does Auto-Correlation effect the model?" can be asked in this case. At first, CPM value was predicted using its previous values with a linear regression model. The R-Squared score, not surprisingly, was above 0.9. It can be said that the model has a huge success for predicting CPM values, but actually what was accomplished is just to prove that the target is highly auto-correlated.

If CPM is highly correlated, today's CPM value would probably be around yesterday's CPM value. Most of the time, change in CPM value is between [-0.3, +0.4], when the campaign bid and/or budget does not change more than 25%. Therefore, any prediction within that range would be reasonable and that is why high R-Squared score was obtained. The lack of the model is the “direction” of change.

If CPM prediction model was going to be used in Marvin optimization, the “error” term must be defined for Adphorus’ specific use case. It has been decided that there are two types of errors in this case:

1. The direction of CPM value (whether it will increase/decrease tomorrow)
2. The amount of change

Without considering the first error, it is natural to have a high score due to the reason explained above. The direction of change is more important than the amount of change; if it is assumed that costs will be lower tomorrow- but actually it will not be, then our bid/budget algorithm results will be suboptimal. Marvin algorithm may falsely assume that the campaign will have a lower cost, thereby spending more money on that specific campaign; yet actually what it will accomplish is to increase the costs even more by spending more on a campaign that will have higher CPM.

To sum up, it was not beneficial to target CPM values. Instead of this, ”Logarithmic CPM Change compared to yesterday” was used in the regression model. Logarithmic returns are highly used in finance. Thus, the same logic is applied in this study.

2.2. CPM Prediction as a Classification Problem

At first, the problem was defined as a “Supervised Classification Problem”. As discussed above, finding the direction of CPM change is more important and vital. For that reason, the distribution of percentage changes was divided into groups (0,-1,1,2.. etc.) and different classification algorithms were executed using these groups as target categories.

Approaching CPM value prediction as a classification problem was part of MEF Big Data Analytics Program BDA 502 Course Final Project. Here is the brief summary of classification results:

- After some trials, three categories were used:
    - If CPM will decrease more than 25% => -1
    - If CPM will stay between -25% & 25% => 0
    - If CPM will increase more than 25% => 1

- Around 60 features have been generated and then that number has been reduced to 16 in the end.

- At first, train accuracy scores were above 95% and test scores were around 60% which seemed to be a clear overfitting.

- After reducing features and dimensions, the gap between the test and train scores decreased. (Around 1-2% percent.)

- Train and test scores were reduced around 55% when the dimensions were reduced.

- Best scores were observed with Random Forest algorithm.

- After manual-parameter tuning, test accuracy score increased to 70.5%.

- CPM change with a rule-based model was predicted in order to have a benchmark and decide whether newly developed model brings an uplift in accuracy.

- Rule-based model predicted for the next day's CPM value by calculating weighted average of past 7 days' CPM values, and grouped them according to the thresholds above.

- Weighted model had an accuracy of 47%.
- Manually tuned Random Forest algorithm performed better compared to the weighted model.

All the details of classification project and the report itself can be found in Appendix A.

2.3. Predictive vs. Explanatory Models

There is a significant difference between two models: In CPM prediction as a regression problem, any forward-looking feature was not applied. However, in Classification problem for BDA 502 course, "*spend change of last day*" was used as a feature while predicting the CPM change. Normally, that information was not available when predicting CPM change of tomorrow. In that sense, the former can be categorized as a "**predictive**" model and the latter as an "**explanatory**" model.

Since CPM values are time-series data, in order to predict tomorrow's CPM value or CPM change, the available information at the time of prediction should be used. In other words, the information at point *t* should only be used, when *t + 1* is predicted. To use the information from *t + 1* at point *t* would be cheating the algorithm. Back-test results will be biased and will not be accurate. Besides, it would be impossible to make a forward-test, as the information of next day's spend change did not exist. However, insights can be generated by using *t +1* deliberately.

For the reasons above, BDA 502 CPM Change Classification Project can be considered as an **explanatory** model. The benefit of explanatory models with time-series data is to understand which forward-looking features have a correlation or impact on the target values. In this case, it has been observed that spend levels of last day have an impact on CPM values. As a result, **predictive** model with the input from **explanatory** model may be restructured in the future.

2.4. CPM Prediction as a Regression Problem

After BDA 502 Course Final Project, the problem was redefined as “Supervised Regression” in order to understand whether predicting numerical values bring an uplift or not.

Different target values were used during the project. Cost value itself, cost change in percentage and logarithmic cost change are three different target types. Combination of these three with CPM and Cost per Thousand People Reached generates six different targets for modelling:

- Cost per Thousand People Reached value (in US Dollars)
- Cost per Thousand People Reached change, compared to yesterday
- Logarithmic Cost per Thousand People Reached change, compared to yesterday
- Cost per Thousand value (in US Dollars)
- CPM change, compared to yesterday
- Logarithmic CPM change, compared to yesterday

Due to the reasons related to Auto-Correlation explained above, values in US Dollars were not used as target values. R-Squared scores were high, yet have no use for Marvin algorithm and could lead to suboptimal results.

Cost per Thousand People Reached is very similar to CPM as well as model results. Since there was no significant uplift with Cost per Thousand People Reached as a target value, CPM was used instead.

Although the results were very similar between percentage changes and logarithmic changes, the latter was chosen to be used. Logarithmic returns are highly used in finance, so the same logic was used in this project as well. Because of these reasons, the target value was “Logarithmic CPM Change compared to yesterday” in the regression model.

Having multiple target values and multiple regression models in order to find a robust model may have seemed promising, however this was not the case. From this point on, the *target* is referred to “Logarithmic CPM Change compared to yesterday”.

3. LITERATURE REVIEW

Literature related to Facebook ad delivery, campaign optimization is scarce. This can be attributed to three important factors:

- Facebook announced its Marketing Partner Program in 2015[8], which shows that optimization solutions are relatively new in the market. At the most digital marketing has a history of 30 years. There is still more time needed for more research and experimentation.

- Google is relatively old compared to Facebook and some advertisers and researchers give more credit to performance of Google Adwords. Ad optimization on Google Adwords have significantly more academic research compared to Facebook.

- Facebook does not share personal and auction level data with advertisers which makes collecting granular data, conducting experiments and optimization more difficult.

The following articles have some relation to the problem in this project.

3.1. Web-Scale Bayesian CTR Prediction for Sponsored Search Advertising[9]

The algorithm used in the paper (Graepel, Candela, Borchert & Herbrich, 2010) describes a new Bayesian online learning algorithm for binary prediction based on a generalised linear model with a probit (cumulative Gaussian) function. Estimating CPM values is crucial for Adphorus as CTR prediction in the digital marketing business. CTR is a metric calculated by “clicks/impressions” and the abbreviation of Click-Through Rate.

Estimated click-through rate plays a critical role in deciding both allocation and payments, and has significant effect on the user experience, advertiser and income of the ad marketplace (Graepel, Candela, Borchert & Herbrich, 2010, 2). Similar to the prediction of the CPM change, the learning algorithm in this paper tries to map the set of ad impressions as represented by their feature descriptions, and the intervals represent the set of possible CTRs, in other words, the probabilities of clicks.

3.2. Ad Impression Forecasting for Sponsored Search[10]

Another study (Nath, Mukherjee, Jain, Goyal & Laxman, 2013) focuses on the ad impressions forecasting in order to optimize return on investment for sponsored search advertising.

Sponsored search is a dynamic process where advertisers and ads fluctuate, and the query traffic shows seasonal or geographic variations which may cause changes in traffic volumes. As a result, advertisers get confused about how to set bid values to achieve their goals. For example, advertisers may want to maximize the number of impressions for a given budget. This confusion about bid-budget trade-off leaves advertisers with ineffective bids, they may bid either too much or too little, and sometimes end up with leftover budgets (Nath, Mukherjee, Jain, Goyal & Laxman, 2013, 943).

The model in this study addresses the problem that the majority of the existing forecasting models ignore or overfit to the query traffic information which has a crucial impact on the probability of winning an ad auction. In the project, ad auctions are holistically modelled using a Bayes net model that captures the correlation between competitors' scores and query traffic features. The reason why this model is preferred to others is that most of the auction features are categorical, thus Bayes net can be easily trained and sampled to generate artificial auctions.

Another advantage of Bayes net is that it allows for feature targeting. For example, if the advertiser has a certain geographical area to which they would like to advertise, fixing the corresponding nodes in the Bayes net makes the model be able to generate the traffic corresponding to the targeting (Nath, Mukherjee, Jain, Goyal & Laxman, 2013, 952).

3.3. Forecasting User Visits for Online Display Advertising[11]

Another study (Cetintas, Chen & Si, 2012) points out an important problem in online advertising that makes it difficult for advertisers to forecast the number of user visits for a web page. One of the major contributions of this study is the discovery of existing models addressing the same problem that have been utilized by traditional time-series forecasting techniques on historical data of user visits. In other words, previous models used a single regression model for forecasting that was based on historical data for all web pages. Existing models ignore the fact that various types of web pages and timestamps have different patterns of user visits. However, in this study, a set of probabilistic latent class models are used as the learning algorithm automatically captures the underlying user visit patterns among multiple web pages and multiple timestamps.

Compared to a single regression model, the proposed probabilistic latent class model is more flexible in identifying various latent classes for web pages and timestamps with similar user visit trends, as well as learning a separate forecasting model for each class of web pages and timestamps (Cetintas, Chen & Si, 2012). Thus, unlike the traditional regression models, the probabilistic latent class model is shown to be more capable of differentiating the importance of various types of information across different classes of web pages and timestamps.

3.4. Applying Bayesian Bandits For Solving Optimal Budget Allocation In Social Media Marketing[12]

There is limited amount of research regarding CPA value prediction in the social media marketing literature. However, a great amount of study can be found on optimal budget allocation in digital marketing. One of them focuses on the bayesian approach applied on multi-armed bandit problems for solving optimal budget allocation in social media marketing.

The Bayesian bandit method in the study (Ahonen, 2017) is actually based on the multi-armed bandit problem. Multi-armed bandit problem is a kind of sequential resource allocation problem in probability theory. There are two main challenges to this problem; exploitation and exploration. The former accounts for using resources on alternative options based on the current knowledge that gives the best results, the latter refers to seeking out for better alternatives at the risk of losing some gains from current budget allocation. Specifically, it is a sequential resource allocation problem where one tries to maximize the expected returns. It is very difficult for companies to allocate marketing budget on various advertising campaigns in real time since expected returns change in real time as well. Thus, multi-armed budget allocation decisions are usually made periodically.

Multi-armed bandit problems can be solved using sophisticated parameter tuning methods but heuristics are generally preferred (Ahonen, 2017). One of them is Bayesian statistics and it is called “random probability matching”. Random probability matching is derived from the Thompson Sampling approach to solving multi-armed bandit problems.

Facebook has launched its own automated solution for real time budget allocation for the online auction mechanism as of late 2018. This brand new algorithm is in its infancy, yet poses a great threat for Facebook Marketing Partners, like Adphorus, who are also struggling to design the best optimal budget allocation tools for online auctions.

Another point in the work of Ahonen is the modelling of CPA value change. (Cost per Conversion is also referred as Cost per Action. CPA is a commonly used abbreviation of Cost per Action in the marketing industry.) Ahonen's work on modelling CPA change is parallel with this project. However, the author keeps details of the algorithm as a trade-secret. Therefore, only a limited amount of details was mentioned in Ahonen's work.

In this study (Ahonen, 2017), an algorithm for budget constraint Bayesian bandits is formulated in order to measure what happens to CPM when the budget is changed. In the Facebook ad delivery mechanism when the budget is increased, the competing bid is also increased, thus CPM rises. This mechanism gets complicated when the advertisers raise their competing bids and then Facebook adjusts its algorithm with its own competing bid. Therefore, it can be assumed that the CPM and then CPA increase as the budget increases.

Similar to this project, Ahonen executes various algorithms to solve the optimal budget allocation problem when CPA changes as a function of budget. In each step, the learning rate is iteratively updated. Afterwards, the change in CPA of each campaign is calculated and then the CPA samples are updated accordingly. After each iteration, a new estimate of the optimal allocation is calculated with the new CPA samples and existing budget constraints. The iteration is performed until the optimal budget allocation is obtained.

The optimization algorithm is presented in the image below (Ahonen, 2017):

Algorithm 3: Iterative budget allocation

```
Calculate prior from data;
Calculate conversion rate samples;
Convert conversion rate samples to CPA samples;
Calculate budget limits;
Initialize budget allocations with current budget proportions;
while Iteration limit not reached do
    Calculate step;
    Calculate optimal budget allocation with budget limits;
    Change budget proportions towards the optimal allocation by the
    step;
    Calculate squared difference between between optimal and current
    budget proportion;
    if squared difference is low enough then
        break loop;
return Ad sets budget proportions;
```

4. DATA PREPARATION

During the project, Python 2.7 programming language and Pandas, Numpy, Scikit-learn packages have been used. For encoding 'UTF-8' has been chosen. Dataset used in the project was obtained from Adphorus & Sojern databases. 10 CSV files, summing up to 1.87GB in disk size, were read as Pandas DataFrames and merged into one single DataFrame. Each row of the dataset had values of a campaign's metrics for a specific date.

4.1. Initial Preprocessing

After generating initial DataFrame, the following methods were applied:

- Metrics related to spending were in clients' local currency. The rates table was then converted into DataFrame, and if there were any missing days in the rates, the previous day's currency value was filled with Pandas' 'forward-fill' method. Lastly, currency rates were joined to master DataFrame and metrics related to spending were converted to USD currency. The following columns were converted to USD and saved as a new column:
    - Cost per Thousand Reach
    - CPM
    - Spend
    - Target
    - Budget
    - Bid
    - Weighted CPM

- The shape of the DataFrame at the initial step was [877790, 72].

- Some of the metrics had NA values that had a meaning. Target Cost per Conversion is an example of such a case. If Target Cost per Conversion value is NA, then it means the campaign has “no target”. NA values in such metrics are filled with meaningful values. (In the case above, ‘infinity’ is the meaningful value.)

- Since excessive amount of dummy features were going to be generated, a Python function for dummy feature generation was defined. Dummy features are based on the values of the following columns:
    - Advertiser time-zone
    - Billing event
    - Optimization Goal
    - Campaign Objective
    - Promoted Object
    - Campaign Attribution Window
    - Pacing Type
    - Health Status
    - DAO
    - Dynamic Ads
    - Cost / Revenue Optimization
    - Target

- Features related to time were generated:
    - Day of week
    - Day of year
    - Month
    - Quarter

- Feature related to spending, delivery and costs were generated:
    - CTR
    - Social CTR

    - Social Reach Ratio
    - Frequency
    - Absorption rate
    - CPM
    - Cost per Thousand Reach
    - Budget Ratio
    - Spend / Budget Ratio
    - Budget / Bid ratio

- Metric columns were shifted in order to find the previous days' metrics for each campaign by applying the following steps:
    - Reindexing DataFrame rows using campaign IDs and dates
    - Finding starting day of each campaign
    - Shifting columns for each campaign
    - Reindexing master DataFrame back to original state

- Weekly weighted CPM was calculated. The more recent the date, the more weight it was given. Thus, the weights were ordered as follows with 0.36 being the most recent of the dates:
    - [0.36, 0.24, 0.16, 0.10, 0.07, 0.04, 0.03]

- Metric percentage changes compared to previous days were calculated. Metric change features would have string values of ['0vs1', '1vs2', … , '6vs7, '1vs7'] located in the end of its name while the numbers refer to the relevant day. Changes were calculated based on the following columns:
    - CTR
    - Social CTR
    - Social Reach Ratio
    - Bid
    - Spend

    - Budget
    - Absorption Rate
    - CPM
    - Frequency
    - Cost per Thousand Reach
    - Budget Ratio
    - Spend / Budget Ratio
    - Budget / Bid ratio

- New features were generated based on metric changes:
    - bidchange*spendchange_yesterday
    - bidchange*budgetchange_yesterday
    - spendchange*budgetchange_yesterday'

- Infinity values were replaced with zero in the following columns:
    - CTR
    - Social ctr
    - Social Reach Ratio

4.2. Filtering

- For continuuity, campaigns that were not observed for more than seven days adjacently were omitted. Otherwise null values would be fed to the regression algorithm which would end up with a Python error. There were 18,046 campaigns that had at least eight adjacent days in the DataFrame.

- Filters were applied to the rows. Before the filters were applied, there were 844,845 rows and 566 columns. In order to apply filters, a Python function was prepared. The following filters were applied:

    - Days should be adjacent and campaign should be running for at least 8 days. (As explained above.)
    - Campaign budget should be greater than 0.
    - CPM should be greater than 0.
    - Weighted CPM should be greater than 0.
    - Cost per Thousand Reach should be greater than 0.
    - Frequency should be greater than 0.
    - Bid should be greater than 0.
    - Spend should be greater than 0.
    - Impressions should be greater than 99. (CPM calculation should be based on significant amount of ad impressions.)
    - There should be no campaign schedule.

- In addition to the filters above, filters related to campaign structure were also applied. If there is a change in the campaign structure, Facebook ad delivery and CPM is also affected. For that reason, campaign structure changes were detected and omitted. The following filters were applied in order to detect observations with structure changes:
    - Campaign budget change should be 0.
    - Client currency should not change.
    - Client timezone should not change.
    - Billing event should not change.
    - Optimization goal should not change.
    - DAO status should not change.
    - Dynamic ad status should not change.
    - Campaign objective should not change.
    - Promoted object should not change.
    - Attribution window should not change.
    - Optimize revenue status should not change.
    - Targeting specifications should not change.
    - Target should not change.

    - Pacing should not change, and be equal to ‘standard’.
    - Health status should not change, and be equal to ‘GOOD’.
    - Campaign status should not change, and be equal to ‘ACTIVE’.

- After the steps above, there were not any rows with an NA value. The shape of the master DataFrame was 141,775 rows and 470 columns at this point..

- The columns that did not contain any information were omitted. If a column falls into the following criteria, then it is assumed that it does not contain any information:
    - It has only one unique value.
    - It contains client-specific information, like client name or user name.
    - It contains long text like a description.
    - It contains infrastructure specific value, like ID.
    - It contains values that Facebook will not share in the future.
    - It is generated during preprocessing and only used in filtering.
    - It has forward-looking values, and is not any target variable.
    - It contains date related values, and is not convenient to be fed into regression function.
    - It is an exact duplicate of another column.

- 258 columns were renamed and reordered for better comprehension.

4.3. Clustering

- Various clusters were generated from campaign structure features in order to use in the model.
- K-Means Clustering algorithm in Scikit-learn package was utilized.
- Number of clusters was chosen to be dynamic and were in a range of [2,20].
- 19 more columns that contain clusters of each observation added.

4.4. Storing the Processed Dataset

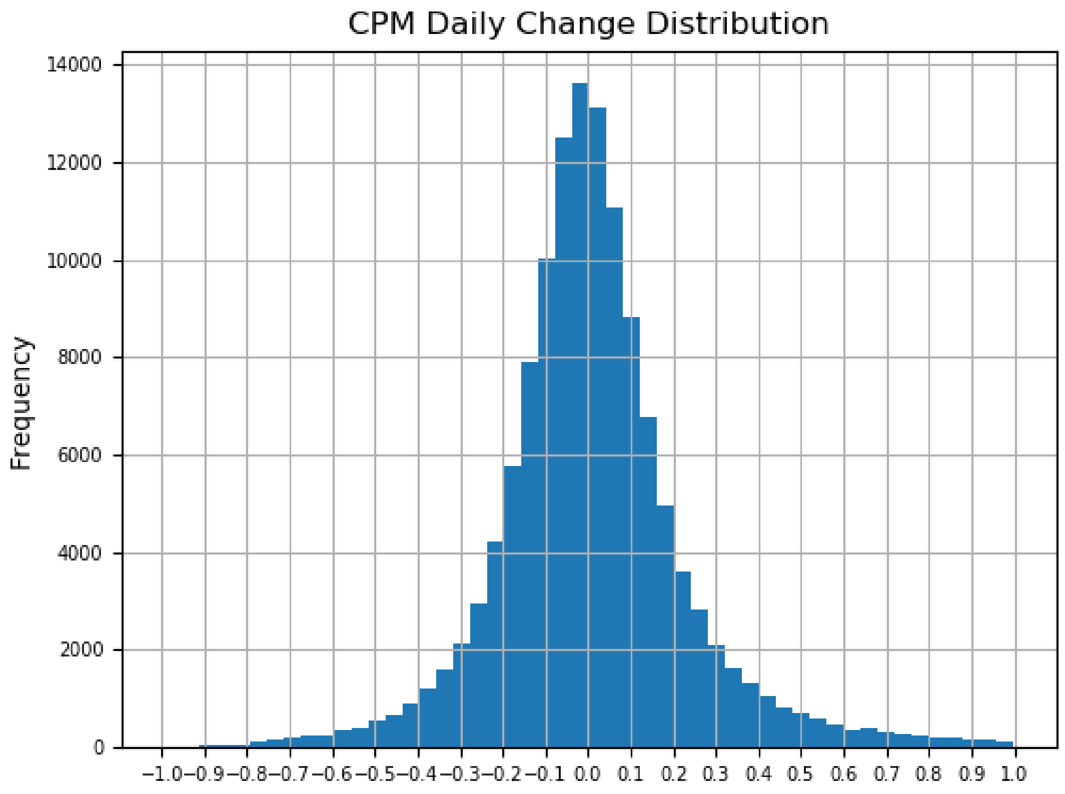
There were 141,775 rows, 277 columns at the final step. Master DataFrame was saved to disk for future use. GZIP compression method was used in order to reduce the file size.

Without utilizing Pandas built-in functions, data preprocessing script took more than six hours. The reason for this was multiple nested ‘for’ loops in the script were used. After converting these parts into Pandas built-in functions, data preprocessing duration decreased significantly to only 15 minutes, saving five hours and forty-five minutes.

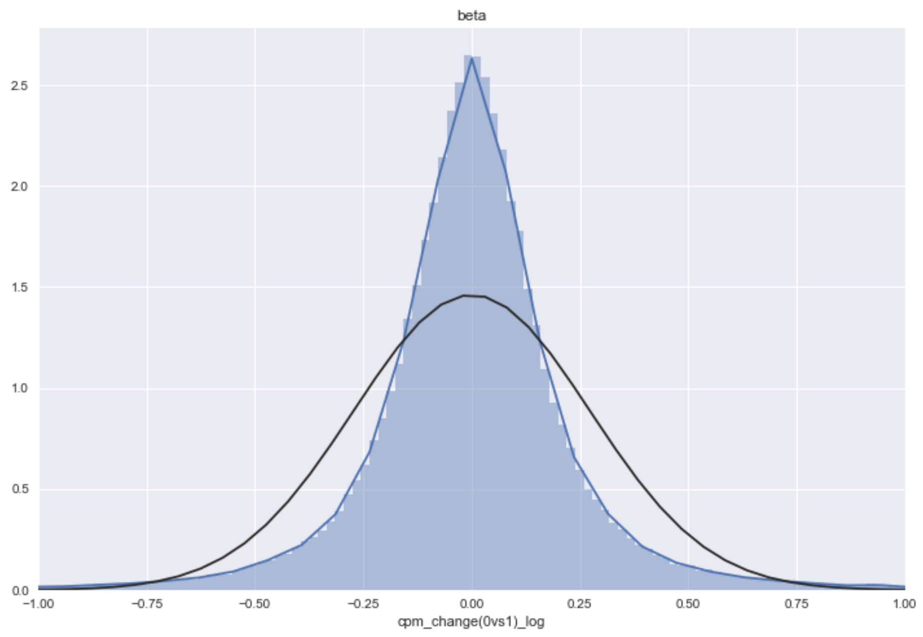
5. DATA VISUALIZATION

Visualizations in this section contain only the target and 13 features used in the last model. After data preparation and multiple trials in order to find the best performing model, a major part of the observations was filtered. Following visualizations were generated from 14 columns and 141,775 rows. **matplotlib, scipy** and **seaborn** packages were used for data visualization.

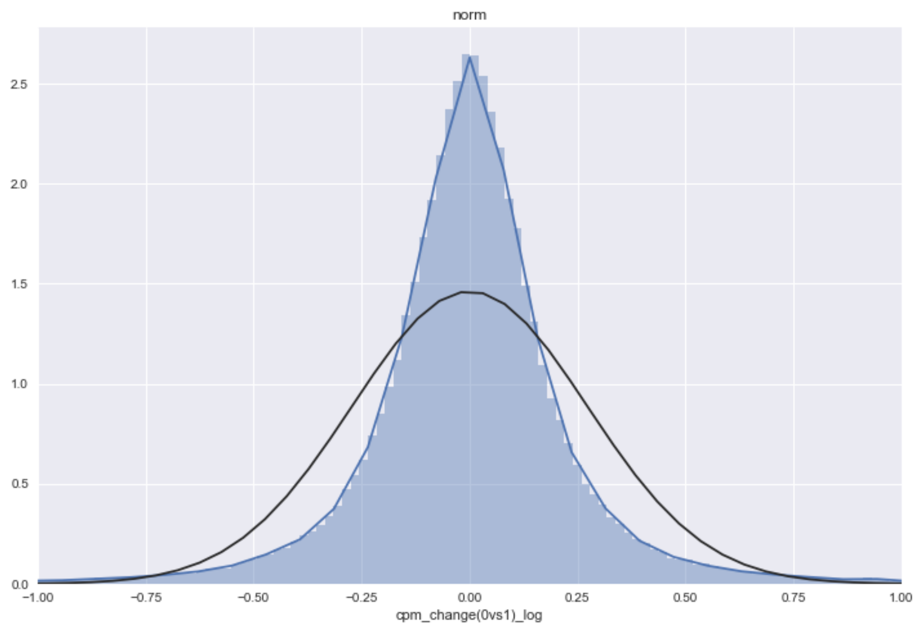
Following visualization is from MEF Big Data Analytics Program BDA 502 Course Final Project and contains histogram of CPM Change, the target variable:



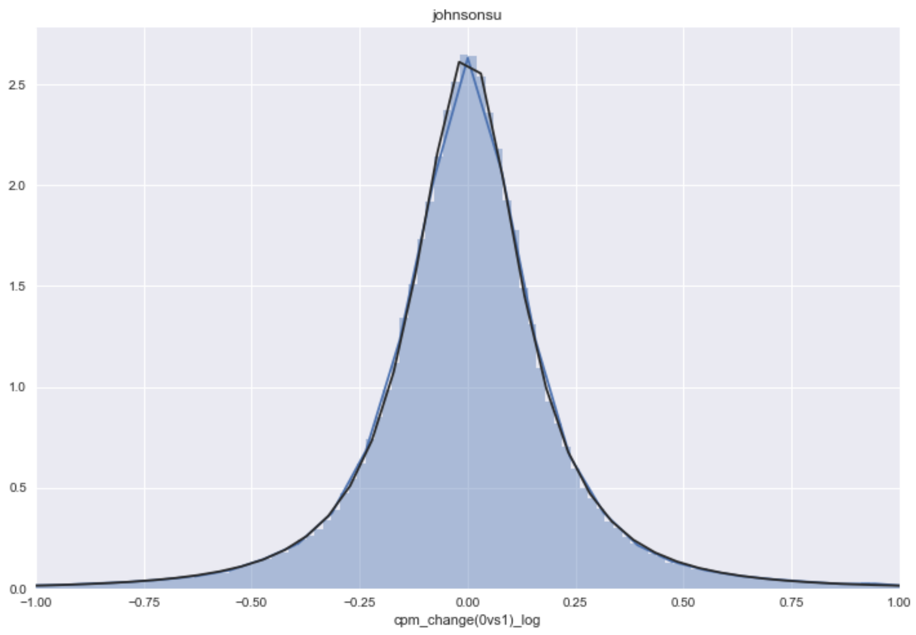
After plotting the target variable, the question of “To which statistical distribution observations in this study fit best?” can be asked. Seaborn histogram function has *fit* parameter, which accepts continuous random distributions in scipy package. 70 histograms, each containing fitted distribution from scipy.stats, were generated. 18 of them were found to fit better ocularly.



Logarithmic daily CPM change, fitted to Beta Distribution

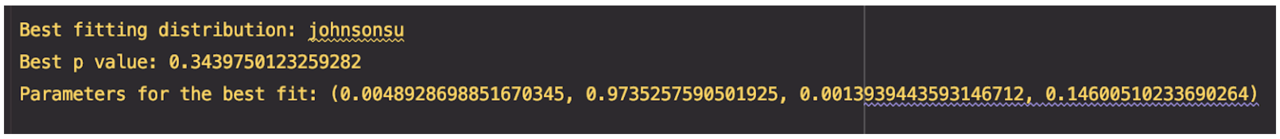


Logarithmic daily CPM change, fitted to Normal Distribution



Logarithmic daily CPM change, fitted to Johnson's SU Distribution

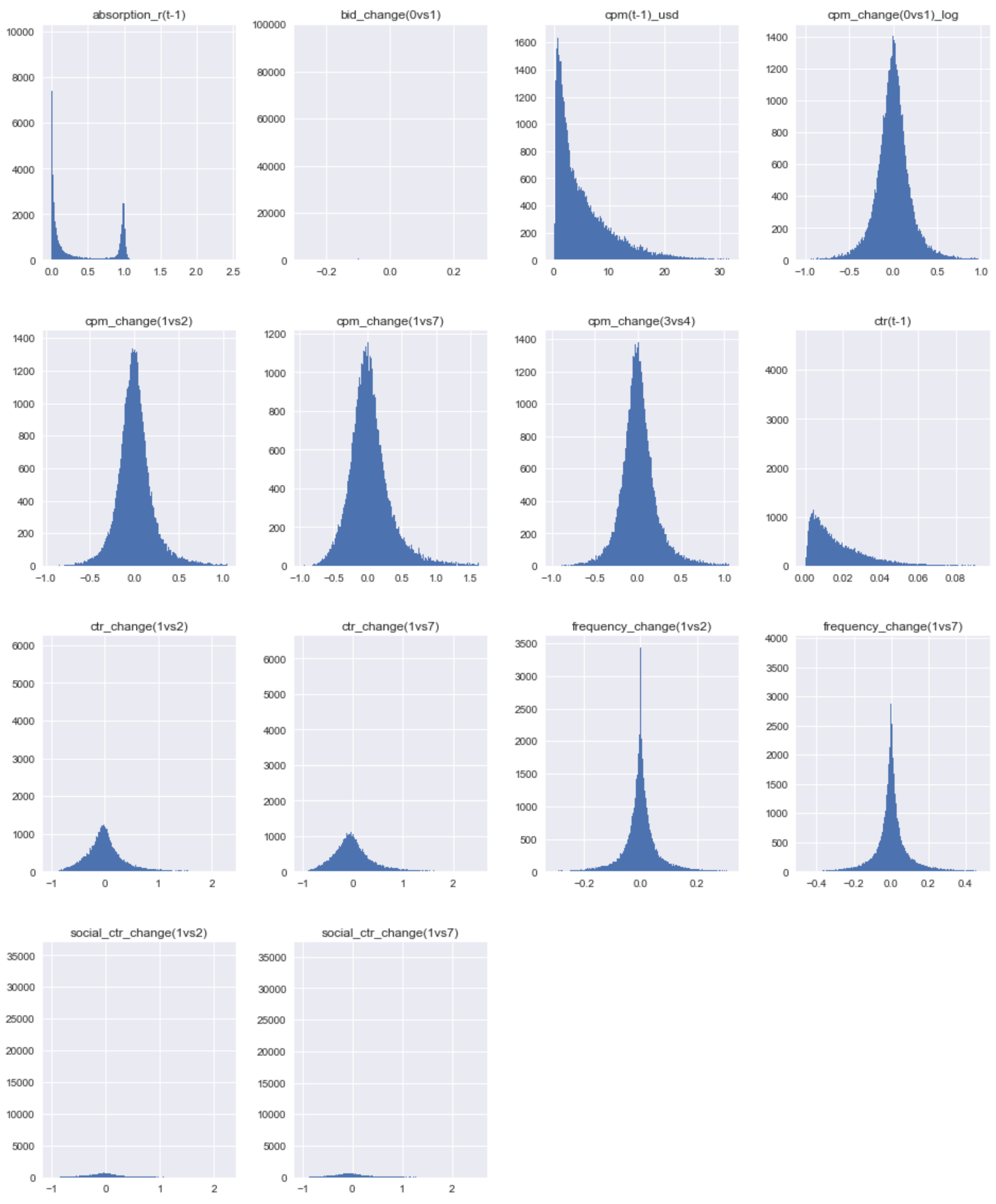
Kolmogorov-Smirnov[13] test was applied to 18 distributions in order to find which distribution fits best[14], and *Johnson's SU* distribution fit best compared to others. Supportively, seaborn histogram visualization for Jonhnson's SU distribution above is very clear and shows the resemblance.



```
Best fitting distribution: johnsonsu
Best p value: 0.3439750123259282
Parameters for the best fit: (0.0048928698851670345, 0.9735257590501925, 0.0013939443593146712, 0.14600510233690264)
```

Output of Kolmogorov-Smirnov test on the console

Following visualization shows the histogram of all variables:

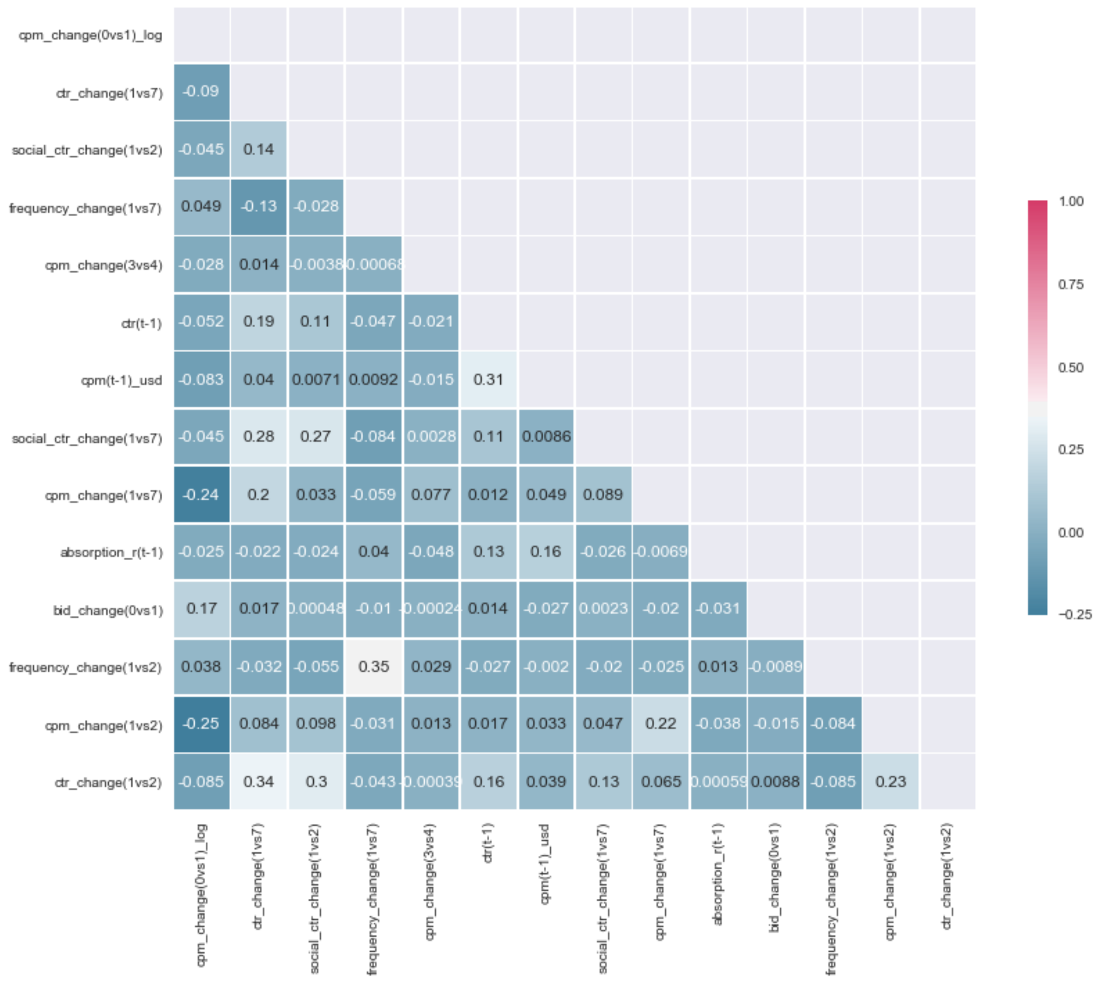


Histogram of 13 Features & Target

These 13 features were chosen as,

- They were the most correlated features with the target.
- They were not inter-correlated more than 50%.
- Using other features in addition to these 13 features did not increase performance significantly.

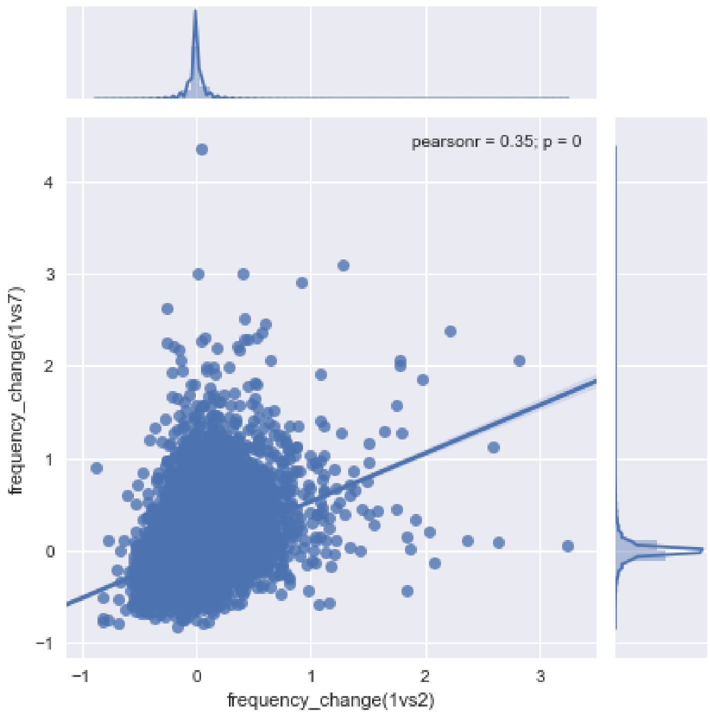
Following visualization depicts correlation heatmap of 13 features and target:

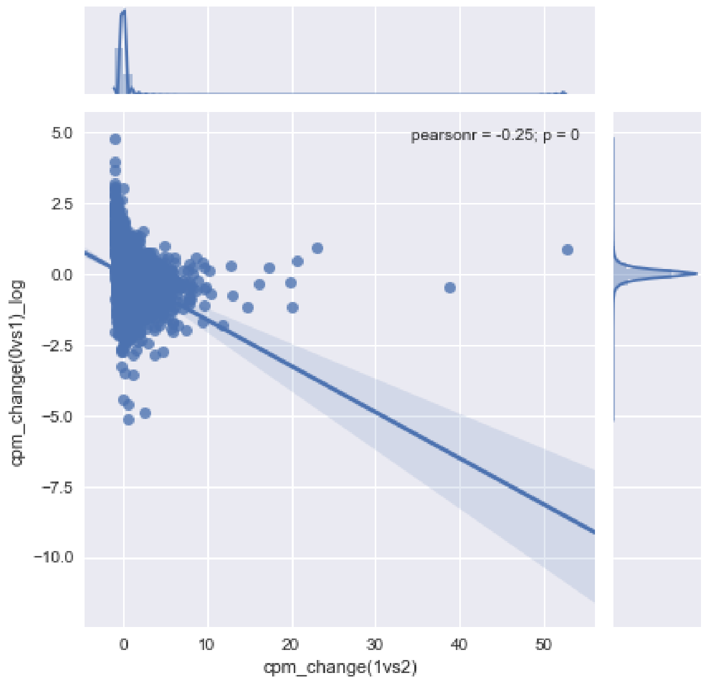


	cpm_change(0vs1)_log	ctr_change(1vs7)	social_ctr_change(1vs2)	frequency_change(1vs7)	cpm_change(3vs4)	ctr(t-1)	cpm(t-1)_usd	social_ctr_change(1vs7)	cpm_change(1vs7)	absorption_r(t-1)	bid_change(0vs1)	frequency_change(1vs2)	cpm_change(1vs2)	ctr_change(1vs2)
cpm_change(0vs1)_log														
ctr_change(1vs7)	-0.09													
social_ctr_change(1vs2)	-0.045	0.14												
frequency_change(1vs7)	0.049	-0.13	-0.028											
cpm_change(3vs4)	-0.028	0.014	-0.0038	0.00068										
ctr(t-1)	-0.052	0.19	0.11	-0.047	-0.021									
cpm(t-1)_usd	-0.083	0.04	0.0071	0.0092	-0.015	0.31								
social_ctr_change(1vs7)	-0.045	0.28	0.27	-0.084	0.0028	0.11	0.0086							
cpm_change(1vs7)	-0.24	0.2	0.033	-0.059	0.077	0.012	0.049	0.089						
absorption_r(t-1)	-0.025	-0.022	-0.024	0.04	-0.048	0.13	0.16	-0.026	-0.0069					
bid_change(0vs1)	0.17	0.017	0.00048	-0.01	-0.00024	0.014	-0.027	0.0023	-0.02	-0.031				
frequency_change(1vs2)	0.038	-0.032	-0.055	0.35	0.029	-0.027	-0.002	-0.02	-0.025	0.013	-0.0089			
cpm_change(1vs2)	-0.25	0.084	0.098	-0.031	0.013	0.017	0.033	0.047	0.22	-0.038	-0.015	-0.084		
ctr_change(1vs2)	-0.085	0.34	0.3	-0.043	-0.00039	0.16	0.039	0.13	0.065	0.00059	0.0088	-0.085	0.23	

Correlation Heatmap

Most correlated features are *frequency_change(1vs2)* and *frequency_change(1vs7)* by 0.35. Most correlated feature to target variable is *cpm_change(1vs2)* by -0.25. Following visualizations depict these feature combinations and fit a regression line.





6. FEATURE ENGINEERING & MODELLING

During feature engineering and modelling, Scikit-learn package was used. Pandas DataFrame that was used in the modelling had 141,775 observations and 277 features. Features and observations were filtered based on different criteria, model results were obtained and compared with benchmark. Benchmark values were obtained from Random Forest Regressor. By filtering features and observations, uplift in model results was sought. There were no NA value in the DataFrame.

Various target variables were used in the model. However, the results below refers to “Logarithmic CPM change, compared to yesterday” for the reasons explained. “674” was used for seed value wherever possible.

Since regression model was formed multiple times during the project, a Python function was defined. The function expects features, a single target, an algorithm to use in the model, parameters for algorithm, a seed number and a text description as arguments. Number of observations should be equal for features and the target. The function splits the observations to 10 folds for cross-validation and searches for the best parameters generating the highest R^2 using GridSearch. After that, the function fits train data using the algorithm, predicts target values using both train and test data, and generates average scores which are adjusted R^2 and Root-Mean-Square-Error (RMSE). Lastly, the model returns following dictionary:

```
to_return = {   'target':target.name,
                'avg_train_r2':np.mean(score_train_list),
                'avg_test_r2':np.mean(score_test_list),
                'avg_train_rmse':np.mean(rmse_train_list),
                'avg_test_rmse':np.mean(rmse_test_list),
                'predictions':np.concatenate(prediction_list, axis=0),
                'actuals':np.concatenate(actual_list, axis=0),
                'algorithm':algorithm,
                'params':reg.best_params_,
                'description':description,
                'Seed':seed    }
```

Scikit-learn package does not have a built-in function that calculates adjusted R^2. For that reason, another function to calculate adjusted R^2 was defined. The formula for adjusted R^2 is:

*adjusted_r = 1 - (1 - rs) * (n - 1) / (n - p - 1)*

in which,

rs: r-squared

n: number of observations

p: number of independent variables

There are multiple algorithms to form a regression model in Scikit-Learn. The regression function expects a single algorithm as an argument, however performance of various machine learning algorithms were sought. For that reason, a new modelling function was defined. The aim of the function is to try each algorithm with the regression function, collect results and report them back. Machine learning algorithms that were used in the modelling function are as follows:

- Linear Regression Algorithms
  - Linear Model
  - Ridge
  - Lasso
  - Bayesian Ridge
  - Lasso Lars
  - Lars

- Tree Regression Algorithms
  - Random Forest Regressor

- Decision Tree Regressor

Various parameters were fed into GridSearch function for tuning. Scikit-Learn documentation indicates which parameters have importance in tuning for most algorithms. Parameters for Random Forest Regressor were taken from MEF Big Data Analytics Program BDA 502 Course Final Project.

6.1. Benchmark

Benchmark scores were generated with Random Forest Regressor. The DataFrame was fed into the modelling function, and results for different "max_features" parameters were obtained. Thereafter different algorithms, filtering options were utilized and their results were compared with benchmark scores. If different scenarios brought significant uplift, then they were used in the final model.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Features: 14	0.2288	0.2948	0.2339	0.1967
Features: 25	0.2229	0.3307	0.2294	0.2267
Features: 40	0.2197	0.3495	0.2273	0.2409
Features: 80	0.2173	0.3638	0.2258	0.2513

Results for Benchmark Modelling

With 80 features, the algorithm generates the minimum error and can explain 25,13% of the distribution variance. As the number of the features increases, RMSE scores decrease and R^2 scores increase. Although the test R^2 score with 80 features is more than the score with 14 features, the latter was used in the benchmark comparison if Random Forest Regressor Algorithm was chosen. The reason is computation takes more than 30 minutes with 80 features. Following scores were obtained either with Random Forest Regressor or Linear Regression algorithm, since the other algorithms generated similar results and did not bring significant uplift. Both train and test R^2 values are adjusted in the tables that show model scores.

6.2. Filtering and Choosing Observations

By filtering observations and feeding them into the model, subset of all observations were tested if it will bring better results. Different scenarios were chosen based on business knowledge.

6.2.1. Significant Changes

If Facebook metrics change significantly, they may affect ad delivery. For that reason, various thresholds were chosen and features related to change only below that threshold were fed into the model. These features were:

- Absorption_r_change
- bid_change(0vs1)
- 'bid_change(1vs2)
- 'budget_change(0vs1)
- budget_change(1vs2)
- 'ctr_change(1vs2)
- spend_change(1vs2)

The threshold was a parameter to select rows and results were collected for following threshold values: [0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.8] Absolute changes were considered.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Threshold: 0.1	0.1244	0.1900	0.1326	-0.2968
Threshold: 0.2	0.1445	0.1932	0.1516	0.0069
Threshold: 0.3	0.1647	0.1902	0.1706	0.0541
Threshold: 0.4	0.1769	0.1960	0.1821	0.0830
Threshold: 0.5	0.1890	0.2063	0.1944	0.0990
Threshold: 0.6	0.1969	0.2204	0.2024	0.1164
Threshold: 0.7	0.2027	0.2394	0.2081	0.1355
Threshold: 0.8	0.2080	0.2660	0.2135	0.1587

Results for Significant Changes

When threshold for significant change increases, both average train & test R^2 increases. When threshold for significant change increases, RMSE also increases. All thresholds bring less RMSE, but their R^2 is also less than benchmark. Filtering for significant change will not be used in the final model, since the results were mixed.

6.2.2. Campaign Runtime

Facebook has its own ad delivery optimization so that advertisers will get the most from their Facebook marketing efforts. It is already known that Facebook requires a learning phase so that its optimization engine will have significant results. Thresholds for various campaign runtimes in days were tested. Threshold values were [0,9,13,19,29].

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Threshold: 0	0.2288	0.2948	0.2339	0.1967
Threshold: 9	0.2289	0.2982	0.2342	0.2003
Threshold: 13	0.2308	0.2998	0.2361	0.1994
Threshold: 19	0.2328	0.3047	0.2377	0.1973
Threshold: 29	0.2369	0.3079	0.2427	0.2010

Campaign Runtime in Days Test Results

Results are similar. It seems campaign runtime in days has trivial effect on target variable. Campaign runtime in days filter will not be used in the final model, since the results were insignificant

6.2.3. Budget / Bid Ratio

Bid defines how much money does an advertiser value for a single conversion on Facebook. Campaign budget divided by bid represents how many conversions an advertiser wants to have from Facebook. If Budget / Bid Ratio is low, it means that conversions expected are also low. With budget and bid so low, Facebook optimization engine may not reach a confidence level in order to find the right audience that are most likely to convert. For these reasons, conversions expected were tested in the model. Thresholds were [5,10,15,20,50]

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Threshold: 5	0.2462	0.3225	0.2507	0.2068
Threshold: 10	0.2513	0.3274	0.2569	0.2088
Threshold: 15	0.2542	0.3320	0.2599	0.2122
Threshold: 20	0.2563	0.3339	0.2625	0.2109
Threshold: 50	0.2647	0.3461	0.2719	0.2191

Test Results of Conversions Expected

Conversions expected has a trivial impact on increasing test R^2 score. If conversions expected were going to be used in the final model, half of the observations should be sacrificed for 1% increase in the score. Conversions expected will not be used in the final model.

6.2.4. Bias in Spend Change

CPM and CPA tend to be higher when budget and campaign spending increase according to literature about Facebook ad delivery. Since campaigns have different spend change levels, bias may affect model results. If number of observations in each spend change level range are similar, then it may remove the bias and increase the model performance.

Following bins were used for spend change level calculations:

[-1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1,
0,
 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

Various thresholds for maximum number of observations were set. These thresholds were [650,2000,3000,6500,9000]. That means, if a bin has more observations than the threshold, observations were dropped randomly until the bin has the maximum number of observations allowed.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Threshold: 650	0.2588	0.4259	0.2780	0.1058
Threshold: 2000	0.2525	0.3942	0.2652	0.2174
Threshold: 3000	0.2456	0.3801	0.2570	0.2262
Threshold: 6500	0.2375	0.3340	0.2458	0.2100
Threshold: 9000	0.2325	0.3213	0.2396	0.2068

Test Results of Spend Change Bias Control

When threshold decreases, train R^2 increases but the opposite holds true for test dataset which leads to clear overfitting. Converting dataset to unbiased one in terms of spend change did not help and it will not be used in the final model.

6.2.5. Bias in Bid Change

Advertisers have more chance to win more auctions and to exhaust their campaign budget, if their bid is higher. Different campaigns may have different bid changes and different number of observations in each level. The same logic behind bias control in spend change levels was applied to bid changes. Thresholds for maximum number of observations were [6500, 3000].

Following bins were used for bid change level calculations:

```
[-np.inf, -1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1,
0,
0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, np.inf]
```

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Threshold: 6500	0.2294	0.3503	0.2452	0.1818
Threshold: 3000	0.2335	0.3694	0.2534	0.1346

Test Results of Bid Change Bias Control

Test results are similar to spend change bias control. There is a clear overfitting, and results are not better than benchmark. Bias control in terms of bid change will not be used in the final model.

6.3. Filtering and Choosing Features

By filtering features and feeding them into the model, subset of all features were tested if it will bring better results. Different scenarios were chosen based on business knowledge.

6.3.1. Campaign Structure

Each campaign has a different structure on Facebook. Variables related to campaign structure are mostly categorical, and they were converted to dummy features. These dummy variables start with "_is" and they were tested whether they bring an uplift or not. In addition to campaign structure, clusters generated for each observation in preprocessing phase were also tested.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Dummy + No Clusters	0.2279	0.3004	0.2330	0.2032
Dummy + Clusters	0.2280	0.2998	0.2332	0.2021
No Dummy + No Clusters	0.2269	0.3068	0.2328	0.2087
No Dummy + Clusters	0.2269	0.3070	0.2324	0.2110

Test Results of Campaign Structure

All train and test RMSE and R^2 scores are very similar. There is 0.01 point increase in test R^2 with option four: "No Dummy + Clusters". Option four suggests not to use campaign structure features, but best clusters. By not using campaign structure features, the complexity will be decreased, also the score increases by 1 point. Only clusters will be used in the final model.

6.3.2. PCA

There were more than two hundred features, and it was questioned whether reducing dimensions with PCA increase performance of the model. Various number of components were tested in the model. Number of components were [3, 5, 10, 20]

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
n_components: 3	0.2716	0.0085	0.2652	-0.0006
n_components: 5	0.2714	0.0104	0.2651	-0.0006
n_components: 10	0.2677	0.0371	0.2649	0.0018
n_components: 20	0.2646	0.0587	0.2640	0.0076

Test Results of the model formed with PCA components

PCA did not help with the scores as clearly shown in the table. Test R^2 values are negative because of adjusted R^2 formula. Dimensionality reduction will not be used in the final model.

6.3.3. Correlation with Target

Features were selected based on their correlation with the target variable and performance results were obtained. Number of features tested were [10, 20, 50, 100, 200] The features that are most correlated to the target variable were selected.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
# of feature: 10	0.2537	0.1348	0.2474	0.1262
# of feature: 20	0.2512	0.1515	0.2460	0.1329
# of feature: 50	0.2509	0.1536	0.2461	0.1307
# of feature: 100	0.2503	0.1575	0.2562	0.0303
# of feature: 200	0.2470	0.1784	0.4794	-4.0478

Test Results of the most correlated features

Because of computational speed in the test linear regression model was used instead of Random Forest Regressor. The top 20 correlated features generated the maximum test R^2 and minimum RMSE. After 50 features test R^2 dropped significantly. Top 20 features will be used in the final model.

6.3.4. Correlation with Other Features

After selecting top 20 features from previous section, the same features were also filtered based on their correlation with each other. Features that are correlated more than 0.5 were detected, and only the features that has higher correlation with the target variable were selected. Then these features were fed into the model and compared with performance of the features decided in the former tests. After running Python script, 13 features out of 20 were detected.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
13 features	0.2535	0.1360	0.2472	0.1279
Features selected in former tests	0.2512	0.1515	0.2460	0.1329

Test Results of the most correlated features

Because of computational speed in the test linear regression model was used instead of Random Forest Regressor. Although features selected in former tests have slightly higher test R^2 score, the final model will utilize 13 features that were selected in this test. The reasons are explained below:

- Overfitting seems to be less with the features selected in this test.
- R^2 and RMSE scores are similar.
- Less features mean less complexity.

Features that were selected in this test:

- 'ctr_change(1vs7)',
- 'social_ctr_change(1vs2)',

- 'frequency_change(1vs7)',
- 'cpm_change(3vs4)',
- 'ctr(t-1)',
- 'cpm(t-1)_usd',
- 'social_ctr_change(1vs7)',
- 'cpm_change(1vs7)',
- 'absorption_r(t-1)',
- 'bid_change(0vs1)',
- 'frequency_change(1vs2)',
- 'cpm_change(1vs2)',
- 'ctr_change(1vs2)’

6.4. Outliers

Although impossible or problematic cases were filtered out in preprocessing phase, effect of removing outliers from observations was tested. z-Score and IQR were used for outlier removal. z-Score threshold was 3.0 and IQR thresholds were [0.25 - 0.75].

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
z-Score removal	0.1770	0.2206	0.1828	0.1536
IQR removal	0.1252	0.1872	0.1328	0.0771
Without outlier removal	0.2250	0.3191	0.2296	0.2415

Outlier Removal Test Results

Removing outliers surprisingly decreased test R^2 score by almost 10%. This may also show that outlier cases contribute more to CPM change. Outlier observations will not be filtered in the final model.

6.5. Scaling

Various scaling algorithms were used on the features in the hope of increasing model performance.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
MinMaxScaler	0.2535	0.1360	0.2472	0.1279
MaxAbsScaler	0.2535	0.1360	0.2472	0.1279
StandardScaler	0.2535	0.1360	0.2472	0.1279
RobustScaler	0.2535	0.1360	0.2472	0.1279
No Scaling	0.2535	0.1360	0.2472	0.1279

Feature Scaling Test Results

Feature scaling has no effect on the performance with the linear regression model.

6.6. Normalizing

Linear regression models assume the dataset is normally distributed. The effect of normalizing the features was observed.

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
With Normalizing	0.2501	0.1591	0.2443	0.1425
Without Normalizing	0.2535	0.1360	0.2472	0.1279

Normalizing Test Results

Normalizing the features increases both train and test R^2 scores and decreases RMSE. The dataset will be normalized in the final model.

6.7. Final Model

During the feature engineering and modelling phase, various assumptions and scenarios were tested against the benchmark. These results were applied to the final model and are as follows:

- About observations:
  - Observations will not be filtered based on significant changes.
  - Observations will not be filtered based on campaign runtime.
  - Observations will not be filtered based on conversions expected.
  - Observations will not be filtered based on spend changes.
  - Observations will not be filtered based on bid changes.

- About features:
  - PCA will not be utilized.
  - Outliers will not be removed.
  - Features will not be scaled.
  - Features will be normalized.
  - 13 features will be fed into the model.

After applying the above methods to the final model, the following results were obtained:

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Final Model	0.2295	0.2920	0.2350	0.2086
Benchmark with 14 features	0.2288	0.2948	0.2339	0.1967
Benchmark with 25 features	0.2229	0.3307	0.2294	0.2267
Benchmark with 40 features	0.2197	0.3495	0.2273	0.2409
Benchmark with 80 features	0.2173	0.3638	0.2258	0.2513

Final Model vs. Benchmark Scores

The final model uses Random Forest Regressor with selected 13 features in order to predict "Logarithmic CPM Change compared to yesterday". Scores with SVR algorithm produced 1% more test R^2 score, but it took several hours to generate the model. Since SVR algorithm is costly and does not bring significant uplift, Random Forest Regressor was chosen.

Random Forest Regressor with 80 features increases test R^2 score to 0.25, however as with the SVR algorithm, it is too costly to form the model. Consequently, best scores generated in this study and best parameters are mentioned below:

Case	Avg. Train RMSE	Avg. Train R^2	Avg. Test RMSE	Avg. Test R^2
Final Model	0.2295	0.2920	0.2350	0.2086

Final Model Results

```
{
        n_estimators = 50,
        max_features = 13,
        max_depth = 10,
        min_samples_split = 5,
        n_jobs = -1,
        min_samples_leaf = 20,
        oob_score = True,
        random_state = 674
}
```

Parameters used in the Random Forest Regressor

7. CONCLUSION

Facebook with more than 2 billion active users seems to continue its part in the digital ad spending duopoly. In this study, changes in Facebook CPM values were predicted and various model results were shown. Although the model results are not satisfactory, the study will be a basis for future researches. Predicting campaign CPM does not seem to be a simple task, yet the returns will cover the endeavor as an outstanding competitive advantage in the ad-tech market.

The following issues from this project may be utilized in future studies:

- Predictive model structure may use insights from explanatory models.
- In addition to CPM and Cost per Thousand People Reached, conversion numbers, Cost per Conversion and features related to conversion may be generated.
- Features related to currency and rates may be generated.
- Facebook started to share various performance metrics about campaign delivery, the dynamics of ad marketplace and their relation, which Facebook coined “Delivery Insights”. Information from Delivery Insights may be used in the future researches.
- New features related to campaign structure may be generated and fed into the model.
- New features about the audience targeted in the campaigns may be generated.

8. REFERENCES

1. Edmundson,T. & Redman, R. (2018, March 6). 4 Reasons Why You Need a Facebook Marketing Partner. Retrieved from https://steelhouse.com/social/4-reasons-need-facebook-marketing-partner/

2. Getchell, B. (2015, September 30). The Five Questions Every Company Should Ask A Facebook Marketing Partner. Retrieved from https://www.ampush.com/blog/5-questions-to-evaluate-a-potential-fmp/

3. Perrin, N. (2018, September 19). Amazon Is Now the No. 3 Digital Ad Platform in the US. Retrieved from https://www.emarketer.com/content/amazon-is-now-the-no-3-digital-ad-platform-in-the-us

4. Facebook Business. Ad delivery and optimisation. About Ad Delivery. (n.d.). Retrieved from https://www.facebook.com/business/help/1000688343301256

5. Brick Marketing. What is An Impression ?. (n.d.). Retrieved from https://www.brickmarketing.com/define-impression.htm

6. Kenton, W. (2018, May 12). What is a Cost Per Thousand - CPM. Retrieved from https://www.investopedia.com/terms/c/cpm.asp

7. Facebook Business. Analyse Results. Cost per 1,000 people reached. (n.d.). Retrieved from https://www.facebook.com/business/help/1461718327429941

8. AdParlor. (n.d). What Every Marketer Should Know About Facebook's New Marketing Partner Program. Retrieved from https://adparlor.com/blog/facebook-marketing-partner-program/

9. Graepel,T., Candela, J.Q., Borchert, T., Herbrich, R. (2010, June). Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft Bing's Search Engine. Retrieved from https://www.microsoft.com/en-us/research/wp-content/uploads/2010/06/AdPredictor-ICML-2010-final.pdf

10. Nath, A., Mukherjee, S., Jain, P., Goyal, N., Laxman,S. (2013, May). Ad Impression Forecasting for Sponsored Search. Retrieved from http://www2013.w3c.br/proceedings/p943.pdf

11. Cetintas, S., Chen, D., Si, L. (2013). Forecasting User Visits for Online Display Advertising. Information Retrieval, 16(3), pp 369–390. doi: 10.1007/s10791-012-9201-4

12. Ahonen, N.P. (2017, May 22). Applying Bayesian Bandits For Solving Optimal Budget Allocation In Social Media Marketing (Doctoral dissertation). Retrieved from https://aaltodoc.aalto.fi/bitstream/handle/123456789/26743/master_Ahonen_Niko-Petteri_2017.pdf?sequence=1&isAllowed=y

13. Kolmogorov–Smirnov Test. (1970, January 01). Retrieved from https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-32833-1_214

14. cardelling. (2016). How to find probability distribution and parameters for real data? (Python 3). [Blog comment]. Retrieved from https://stackoverflow.com/questions/37487830/how-to-find-probability-distribution-and-parameters-for-real-data-python-3

APPENDIX A: BDA 502 Final Report

Following final report was prepared for BDA 502 Course in MEF Big Data Analytics Program, held in second part of 2018 Spring Term. The lecturer was Tuna Çakar. BDA 502 Final Report is the basis for this study. The problem was defined as a classification problem, instead of regression.

BDA 502
Final Project
Semih Tekten

Introduction:

In this project I tried to forecast CPM change ranges for given Facebook ads. Let me explain the term CPM:

Cost per Thousand (CPM):
"*Cost per thousand (CPM) is a marketing term used to denote the price of 1,000 advertisement impressions on one webpage. If a website publisher charges $2.00 CPM, that means an advertiser must pay $2.00 for every 1,000 impressions of its ad. The "M" in CPM represents the Roman numeral for 1,000.*"

Source: https://www.investopedia.com/terms/c/cpm.asp

On Facebook which ads are shown to users are decided by auction mechanism. For that reason, campaign dynamics on Facebook are very dynamic and CPM values are changing frequently.

Our company, Adphorus, offers its clients a service to optimize their Facebook campaigns so that they will get maximum amount of returns from their campaigns.

Problem to Solve & Research Question:

Our optimization engine, Marvin, sets bids and budgets for every campaign each night for our clients. My object is to find a model for CPM values, so that we will enhance our optimization. We need to find CPM values for given campaign history, bid, spend and budget.

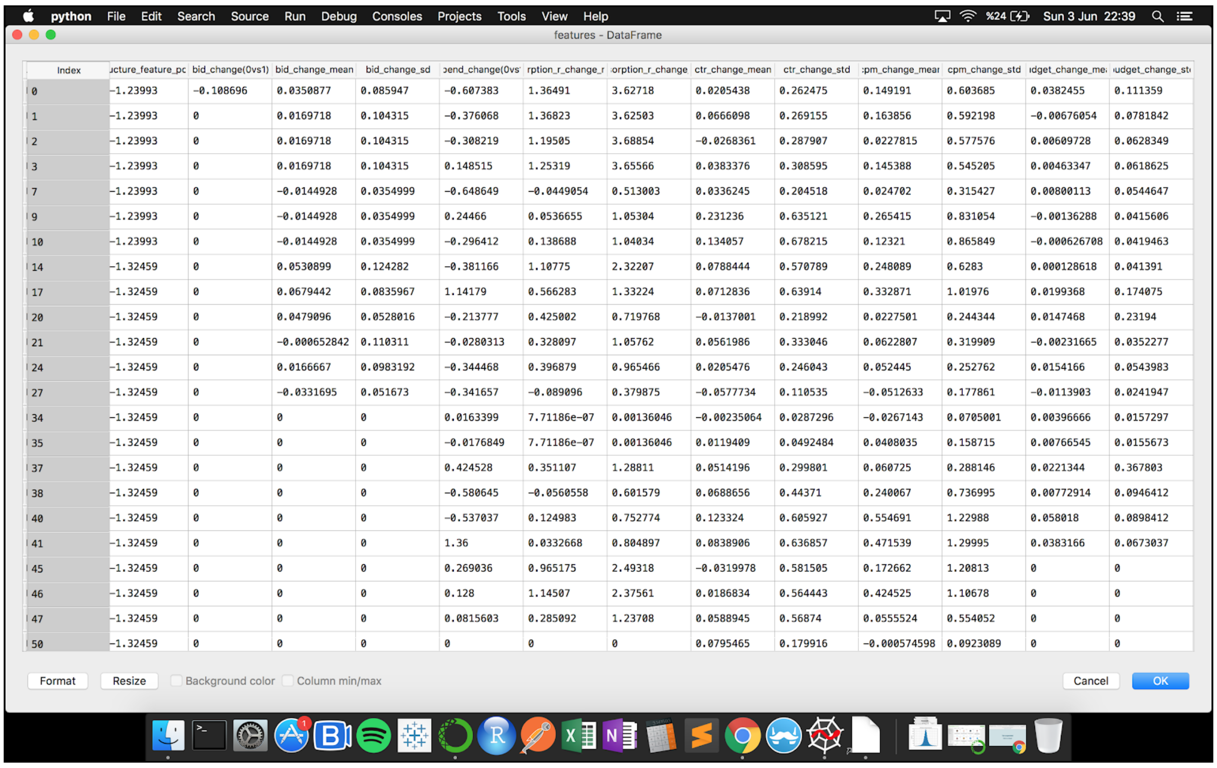
I wanted to narrow down my question to classification problem. For that reason, I set my target labels so that they will correspond to CPM change ranges. After testing for multiple times, I decided to set my labels as follows:

- If CPM will decrease more than 25% => -1
- If CPM will stay between -25% & 25% => 0
- If CPM will increase more than 25% => 1

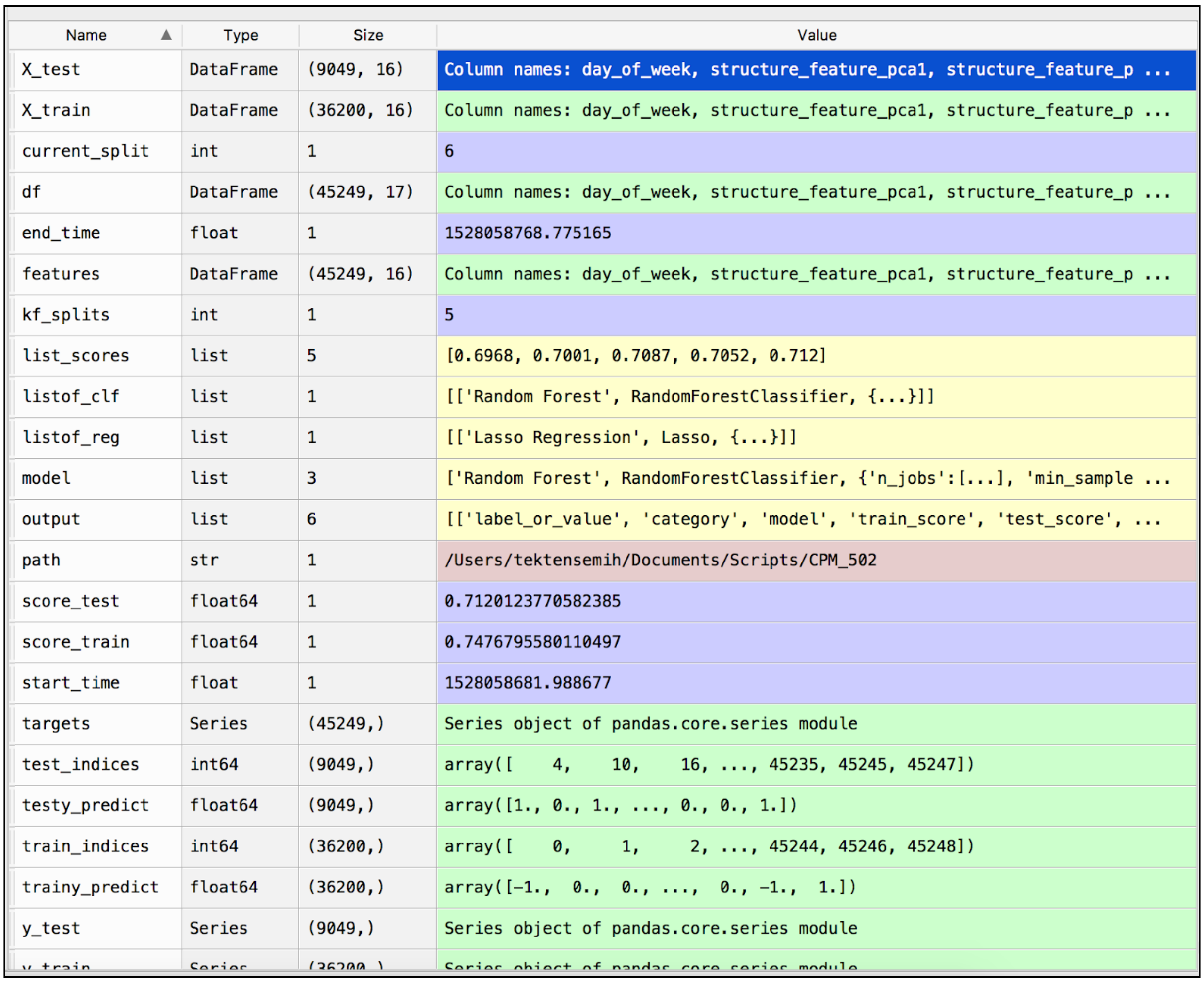
Data Preprocessing & Feature Generation:

I've collected my dataset from Adphorus Databases. Total size of CSV files is more than 1GB. There were 825233 rows and 40 columns. Corresponding files are: "2_preprocess_1.py" and "3_preprocess_2.py".

Since Adphorus is owned by an American company called Sojern, I can't share my dataset. If dataset is needed for grading purposes, I can show how my code works on my own business computer. Below you can find screenshot of my features dataframe:



Index	ucture_feature_pc	bid_change(0vs1)	bid_change_mean	bid_change_sd	pend_change(0vs	rption_r_change_r	orption_r_change	ctr_change_mean	ctr_change_std	pm_change_mear	cpm_change_std	idget_change_me	udget_change_st
0	-1.23993	-0.108696	0.0350877	0.085947	-0.607383	1.36491	3.62718	0.0205438	0.262475	0.149191	0.603685	0.0382455	0.111359
1	-1.23993	0	0.0169718	0.104315	-0.376068	1.36823	3.62503	0.0666098	0.269155	0.163856	0.592198	-0.00676054	0.0781842
2	-1.23993	0	0.0169718	0.104315	-0.308219	1.19505	3.68854	-0.0268361	0.287907	0.0227815	0.577576	0.00609728	0.0628349
3	-1.23993	0	0.0169718	0.104315	0.148515	1.25319	3.65566	0.0383376	0.308595	0.145388	0.545205	0.00463347	0.0618625
7	-1.23993	0	-0.0144928	0.0354999	-0.648649	-0.0449054	0.513003	0.0336245	0.204518	0.024702	0.315427	0.00800113	0.0544647
9	-1.23993	0	-0.0144928	0.0354999	0.24466	0.0536655	1.05304	0.231236	0.635121	0.265415	0.831054	-0.00136288	0.0415606
10	-1.23993	0	-0.0144928	0.0354999	-0.296412	0.138688	1.04034	0.134057	0.678215	0.12321	0.865849	-0.000626708	0.0419463
14	-1.32459	0	0.0530899	0.124282	-0.381166	1.10775	2.32207	0.0788444	0.570789	0.248089	0.6283	0.000128618	0.041391
17	-1.32459	0	0.0679442	0.0835967	1.14179	0.566283	1.33224	0.0712836	0.63914	0.332871	1.01976	0.0199368	0.174075
20	-1.32459	0	0.0479096	0.0528016	-0.213777	0.425002	0.719768	-0.0137001	0.218992	0.0227501	0.244344	0.0147468	0.23194
21	-1.32459	0	-0.000652842	0.110311	-0.0280313	0.328097	1.05762	0.0561986	0.333046	0.0622807	0.319909	-0.00231665	0.0352277
24	-1.32459	0	0.0166667	0.0983192	-0.344468	0.396879	0.965466	0.0205476	0.246043	0.052445	0.252762	0.0154166	0.0543983
27	-1.32459	0	-0.0331695	0.051673	-0.341657	-0.089096	0.379875	-0.0577734	0.110535	-0.0512633	0.177861	-0.0113903	0.0241947
34	-1.32459	0	0	0	0.0163399	7.71186e-07	0.00136046	-0.00235064	0.0287296	-0.0267143	0.0705001	0.00396666	0.0157297
35	-1.32459	0	0	0	-0.0176849	7.71186e-07	0.00136046	0.0119409	0.0492484	0.0408035	0.158715	0.00766545	0.0155673
37	-1.32459	0	0	0	0.424528	0.351107	1.28811	0.0514196	0.299801	0.060725	0.288146	0.0221344	0.367803
38	-1.32459	0	0	0	-0.580645	-0.0560558	0.601579	0.0688656	0.44371	0.240067	0.736995	0.00772914	0.0946412
40	-1.32459	0	0	0	-0.537037	0.124983	0.752774	0.123324	0.605927	0.554691	1.22988	0.058018	0.0898412
41	-1.32459	0	0	0	1.36	0.0332668	0.804897	0.0838906	0.636857	0.471539	1.29995	0.0383166	0.0673037
45	-1.32459	0	0	0	0.269036	0.965175	2.49318	-0.0319978	0.581505	0.172662	1.20813	0	0
46	-1.32459	0	0	0	0.128	1.14507	2.37561	0.0186834	0.564443	0.424525	1.10678	0	0
47	-1.32459	0	0	0	0.0815603	0.285092	1.23708	0.0588945	0.56874	0.0555524	0.554052	0	0
50	-1.32459	0	0	0	0	0	0	0.0795465	0.179916	-0.000574598	0.0923089	0	0



Name	Type	Size	Value
X_test	DataFrame	(9049, 16)	Column names: day_of_week, structure_feature_pca1, structure_feature_p ...
X_train	DataFrame	(36200, 16)	Column names: day_of_week, structure_feature_pca1, structure_feature_p ...
current_split	int	1	6
df	DataFrame	(45249, 17)	Column names: day_of_week, structure_feature_pca1, structure_feature_p ...
end_time	float	1	1528058768.775165
features	DataFrame	(45249, 16)	Column names: day_of_week, structure_feature_pca1, structure_feature_p ...
kf_splits	int	1	5
list_scores	list	5	[0.6968, 0.7001, 0.7087, 0.7052, 0.712]
listof_clf	list	1	[['Random Forest', RandomForestClassifier, {...}]]
listof_reg	list	1	[['Lasso Regression', Lasso, {...}]]
model	list	3	['Random Forest', RandomForestClassifier, {'n_jobs':[...], 'min_sample ...
output	list	6	[['label_or_value', 'category', 'model', 'train_score', 'test_score', ...
path	str	1	/Users/tektensemih/Documents/Scripts/CPM_502
score_test	float64	1	0.7120123770582385
score_train	float64	1	0.7476795580110497
start_time	float	1	1528058681.988677
targets	Series	(45249,)	Series object of pandas.core.series module
test_indices	int64	(9049,)	array([4, 10, 16, ..., 45235, 45245, 45247])
testy_predict	float64	(9049,)	array([1., 0., 1., ..., 0., 0., 1.])
train_indices	int64	(36200,)	array([0, 1, 2, ..., 45244, 45246, 45248])
trainy_predict	float64	(36200,)	array([-1., 0., 0., ..., 0., -1., 1.])
y_test	Series	(9049,)	Series object of pandas.core.series module
y_train	Series	(36200,)	Series object of pandas.core.series module

Data preparation part took most of my time, since I needed to calculate features for each campaign. Generating features took more than 8 processor hours. (I needed to commute with my Macbook open & on my hand. :))

I've generated around 60 features more and then reduced that number to 16 at the end of my work. (I'll explain that later.) I tried to fill NA's with meaningful values.

Features that I generated from raw datasets fall into two categories:

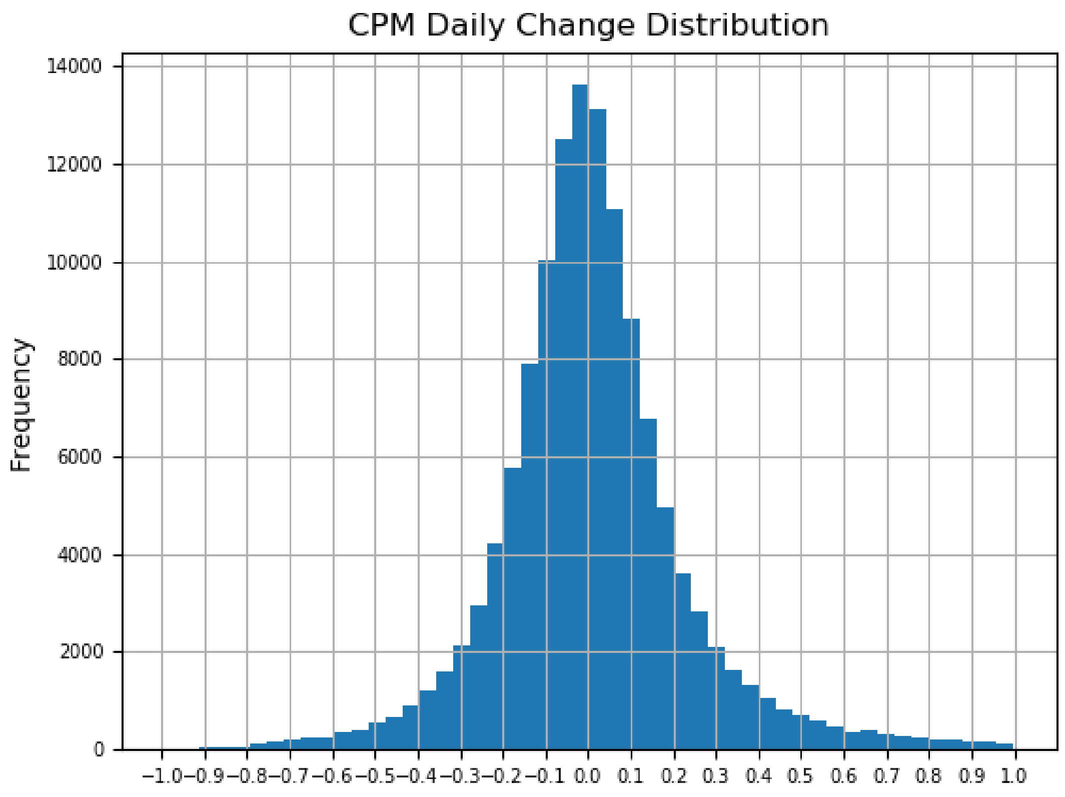
- Features related to campaign structure (Mostly binary, dummy variables)
- Features related to campaign past performance (Mostly continuous variables)

After generating features, I chose only campaigns with:

- No campaign structure changes
- Only adjacent 7 days (prevent any breaks/paused campaigns)

I used Python 2.7, since our backend developers in our company use this version instead of Python3. I also used packages that we learnt in our lectures like Scikit Learn, Numpy, Pandas. The IDE that I developed my code is Spyder. All of my environment is based on Anaconda.

Target Visualization:



Above you can find CPM daily change distribution that I generated in my code after cleaning. (Such a beauty!)

Modelling:

I've divided this part into two. Related files are: "5_featuresn_modelling.py" and "5_featuresn_modelling2.py" In the former I've used all the features that I generated without any limit and overfit my model. In the latter I've reduced number of features, but get very similar test score results but reduced overfitting. As a result, I could manage to decrease complexity in my features and improve scores.

In both of the files I've used following methods:

- I've removed outliers that have more than 3 Z-Scores.
- I've used K-Fold cross-validation and calculated average accuracy scores
- I've set different percentage changes for labels so that I could understand for which target label I got the maximum accuracy scores. (Thus, decided on 25% change.)

- For different labels, I might have unbalanced dataset. In order to prevent bias, I've dropped a random fraction of rows so that all labels are represented equally. In my experience, I observed that balance in the dataset affects the outcomes and the scores significantly. I could see 50% and both 90% accuracy for the same dataset between biased and unbiased datasets. :)

In second file I've also used following methods:

- Since campaign structures have lots of dummy binary variables, I've reduced their numbers using PCA method. I know PCA doesn't work well with binary data, but it helped me reduce dimensions.
- Former file uses campaigns' past performance metrics for each day. (t-1 to t-7). Instead of using all days' metrics individually, I've calculated standard deviation and means for related features. Results stayed same, but I've managed to reduce complexity.
- Since I've reduced dimensions, I could have a chance to observe each feature's effect on the outcome and manually optimize.

In order to have a benchmark, I've also calculated weighted averages of CPM values. My purpose was to have a very simple model to compare my results. Related file's name is: "4_simple_model_weight.py"

I've used following machine learning algorithms from Scikit Learn package:

- Random Forest
- K Nearest [Never had a chance to finish algorithm]
- R Nearest [Never had a chance to finish algorithm]
- Gaussian Naive Bayes
- Decision Tree [Never had a chance to finish algorithm]
- Neural Network
- Logistic Regression Classifier
- Support Vector [Never had a chance to finish algorithm]
- Bagging

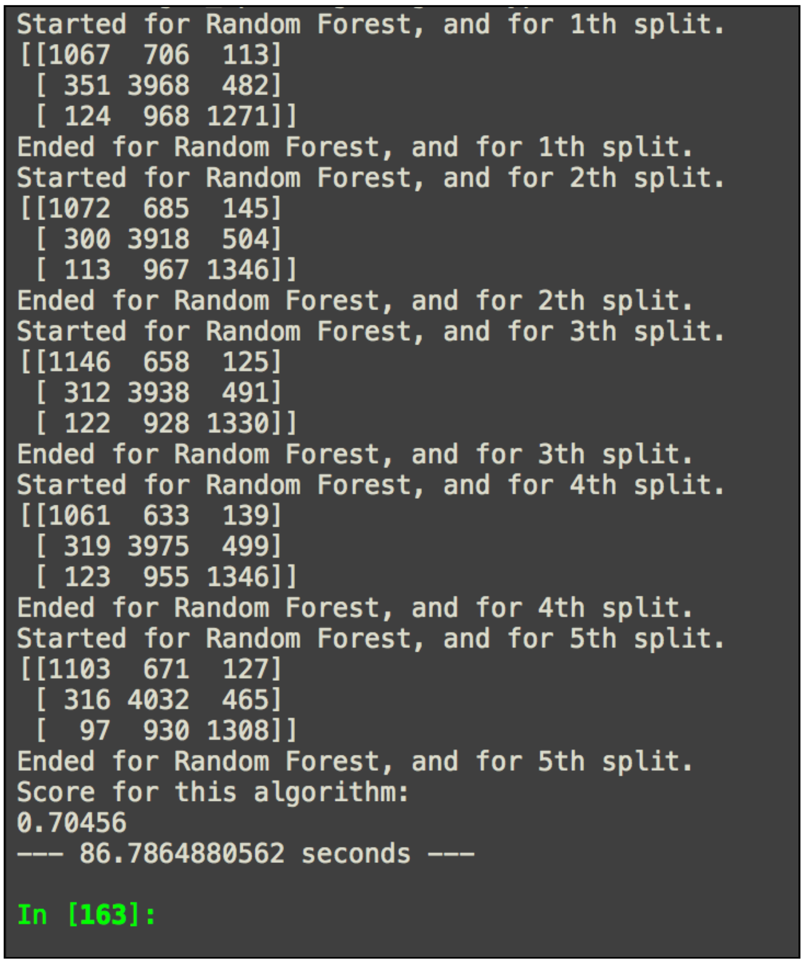
As you can see for some of the algorithms, I couldn't even manage to finish the algorithm. The reason is my computer is a little bit old and takes hours for some of the algorithms. As a result I've cancelled the execution. The same slowness goes to GridSearch. I couldn't benefit from GridSearch, since adding different parameters increased execution time significantly. That's why I tuned parameters manually.

Summary:

At first, my train accuracy scores were above 95% and test scores were around 60%. That seemed to be a clear overfitting. In order to reduce memorizing, I've reduced features and dimensions in the second file as I explained above. After that, I managed to have a small gap between test and train scores. (Around 1-2% percent.)

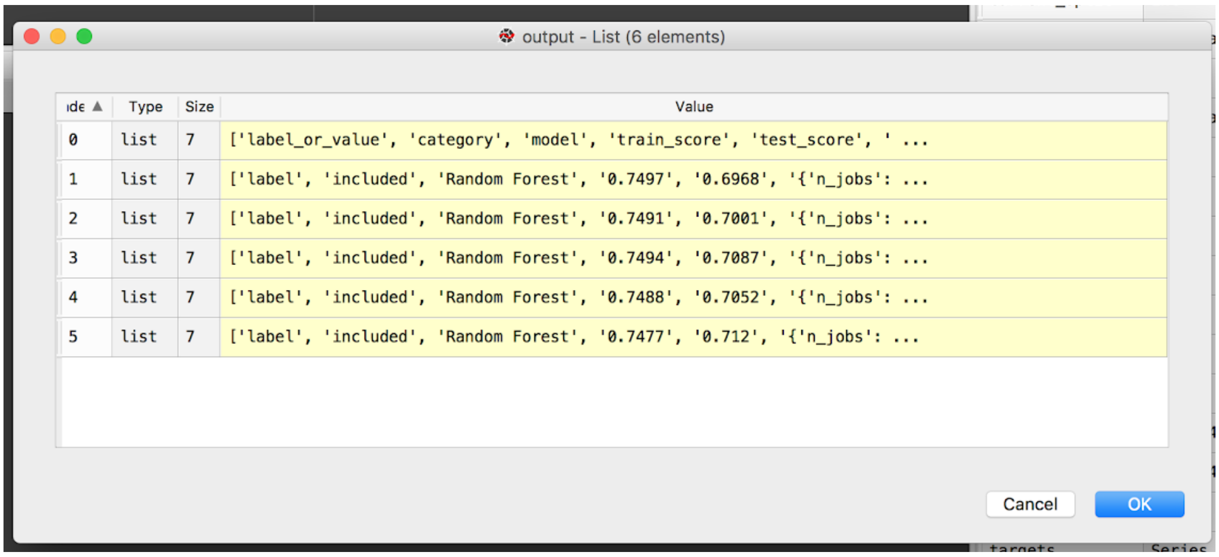
After manually optimizing and tuning parameters, I’ve decided that best algorithm is Random Forest for my case. (The speed of Random Forest has also an effect on my decision.) Train & test scores were reduced to 55% when I reduced dimensions in the second file. But after tuning parameters manually, I’ve managed to increase my test accuracy score to 70.5%. It may not seem such a big success, but since this is the first real-life problem that I try to solve using Data Science, it means a lot for me. I was expecting around 30-40% accuracy.
Lastly, I’ve mentioned a simple weighted model for benchmark above. Simple weighted model had an accuracy of 47%. It seems manually tuned Random Forest algorithm performs better compared to simple weighted model.

Following pictures were taken after executing Random Forest algorithm:



```
Started for Random Forest, and for 1th split.
[[1067  706  113]
 [ 351 3968  482]
 [ 124  968 1271]]
Ended for Random Forest, and for 1th split.
Started for Random Forest, and for 2th split.
[[1072  685  145]
 [ 300 3918  504]
 [ 113  967 1346]]
Ended for Random Forest, and for 2th split.
Started for Random Forest, and for 3th split.
[[1146  658  125]
 [ 312 3938  491]
 [ 122  928 1330]]
Ended for Random Forest, and for 3th split.
Started for Random Forest, and for 4th split.
[[1061  633  139]
 [ 319 3975  499]
 [ 123  955 1346]]
Ended for Random Forest, and for 4th split.
Started for Random Forest, and for 5th split.
[[1103  671  127]
 [ 316 4032  465]
 [  97  930 1308]]
Ended for Random Forest, and for 5th split.
Score for this algorithm:
0.70456
--- 86.7864880562 seconds ---

In [163]:
```



Index	Type	Size	Value
0	list	7	['label_or_value', 'category', 'model', 'train_score', 'test_score', ' ...
1	list	7	['label', 'included', 'Random Forest', '0.7497', '0.6968', '{'n_jobs': ...
2	list	7	['label', 'included', 'Random Forest', '0.7491', '0.7001', '{'n_jobs': ...
3	list	7	['label', 'included', 'Random Forest', '0.7494', '0.7087', '{'n_jobs': ...
4	list	7	['label', 'included', 'Random Forest', '0.7488', '0.7052', '{'n_jobs': ...
5	list	7	['label', 'included', 'Random Forest', '0.7477', '0.712', '{'n_jobs': ...

Further Developments:

New features to be considered:
- Reach metrics
- Delivery Insights metrics
- Client information
- Targeting Audience Specifications
- Suggested bids
- Market & Auction change information
- Audience overlap rates

Refine the problem as follows:
- Forecast CPM value (Regression), instead of change range (Classification)
- Find the highest & lowest values that CPM can have next day

Optimize coding & feature engineering:
- Use Pandas built-in solutions for Time Series data
- Use Pandas Rolling & Shifting methods
- Use Scikit Learn algorithms' own class_weight parameter, instead of manually dropping random fraction of rows
- Use other solutions instead of PCA in order to feed Campaign Structure into features

APPENDIX B: Presentation For MeasureCamp Istanbul

MeasureCamp is an *unconference*, and the schedule is created on the day and speakers are fellow attendees. (http://istanbul.measurecamp.org/) As a concept, unconferences are designed to encourage discussions and exchange of ideas. They provide an alternative to the traditional one-way conferences through a more collaborative social framework for knowledge sharing. Attendees are encouraged to discuss and participate in sessions, even to lead sessions themselves.

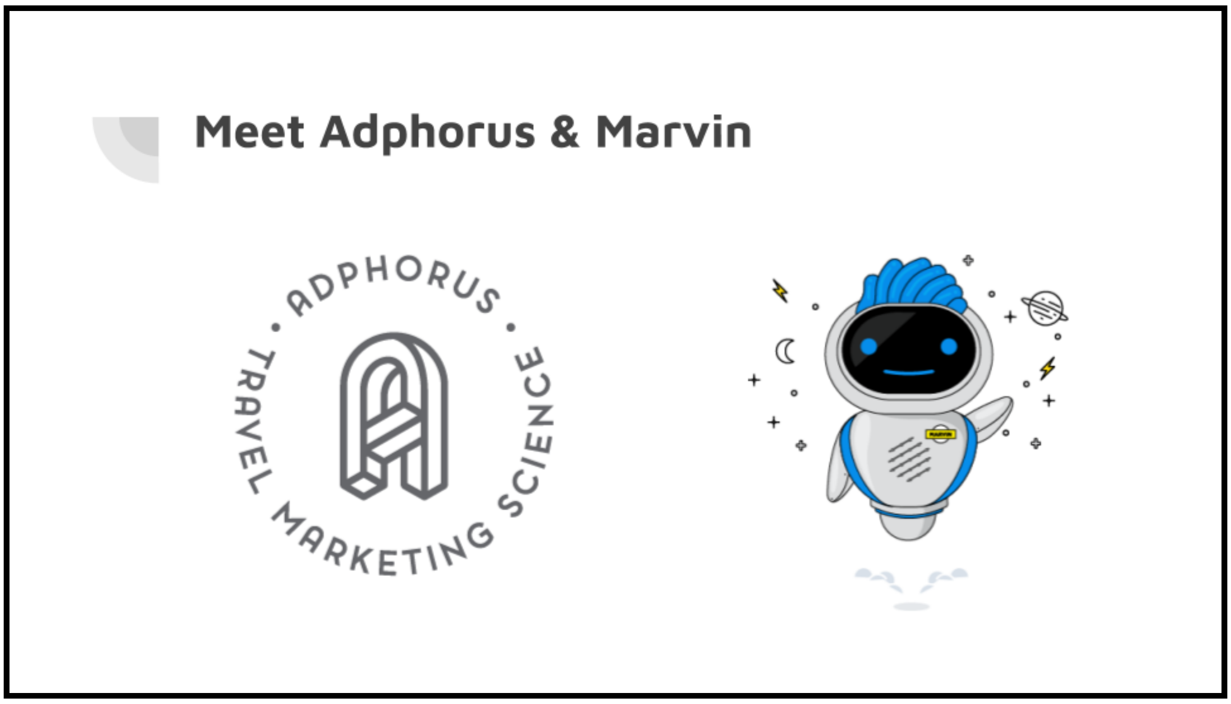
MeasureCamp in its own words:

“MeasureCamp is an open, free-to-attend unconference, different to any other web analytics conference held around the world.”

As a part of the MeasureCamp conference held on Saturday 10 Nov 2018, in Altınbaş University Campus, “Predicting Facebook Ad Impressions & CPM Values” project was presented. Attendees were mostly from marketing industry. Following images are pages from the presentation.

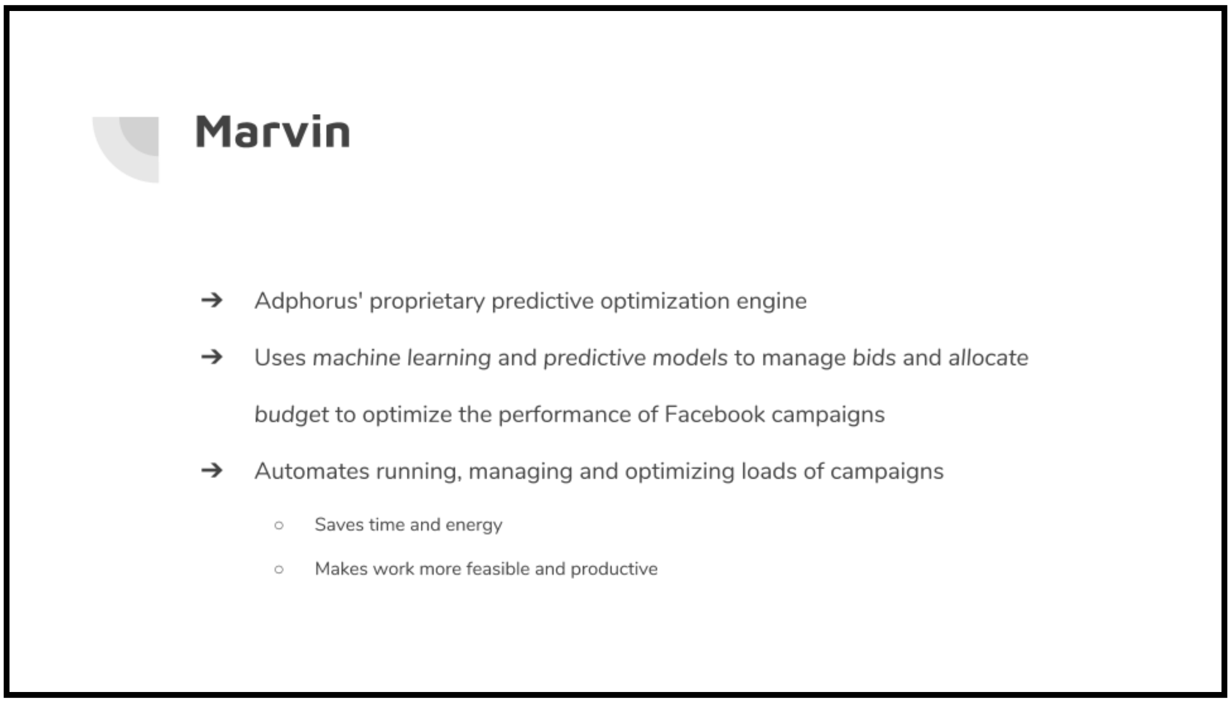
Predicting Facebook CPM

Prepared for MeasureCamp
10.11.2018



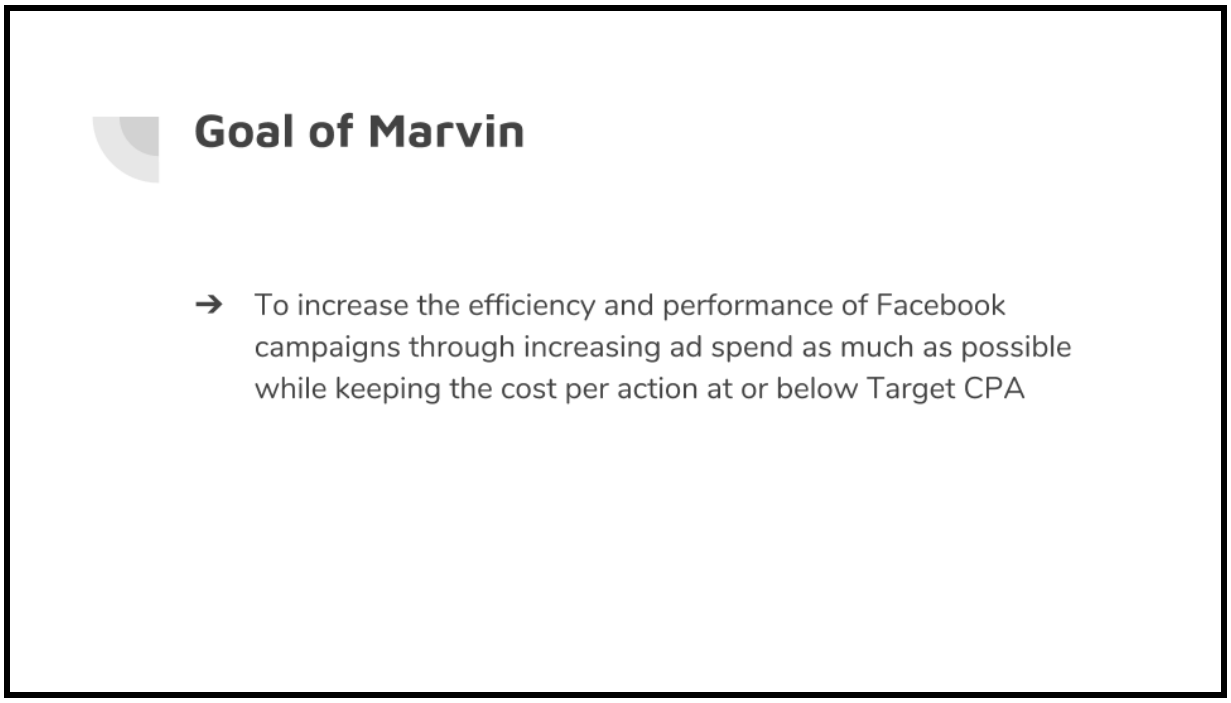
Meet Adphorus & Marvin

ADPHORUS • TRAVEL MARKETING SCIENCE •



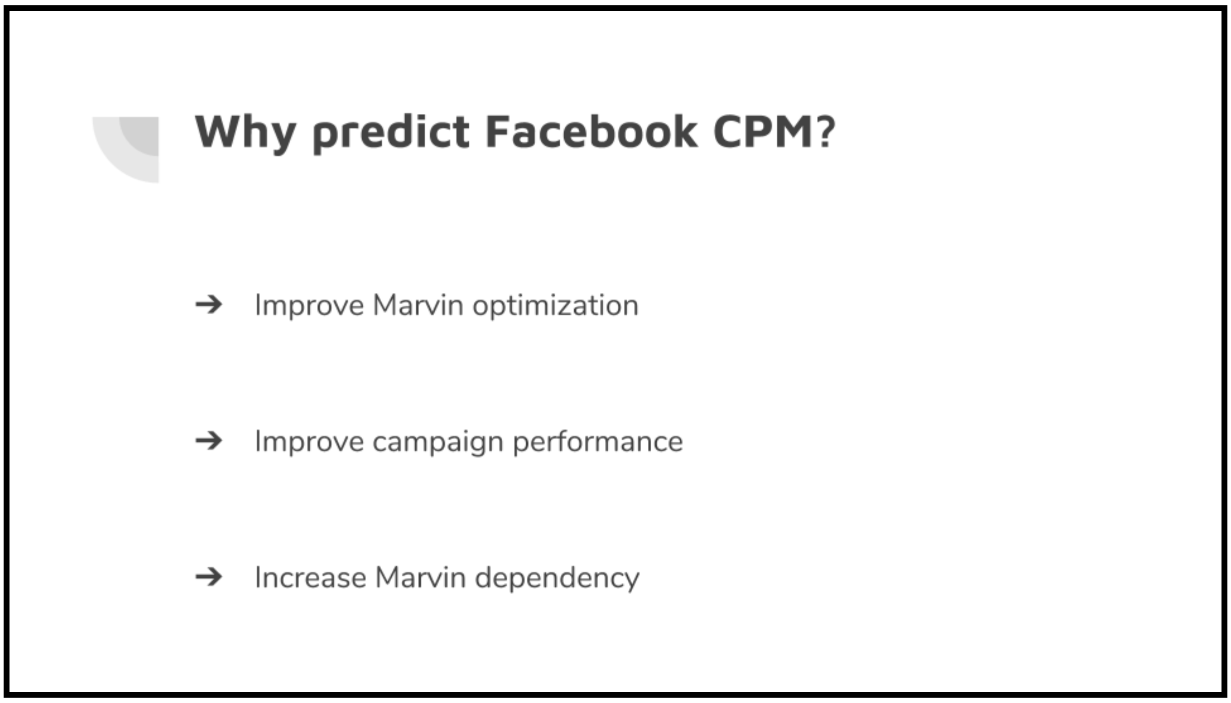
Marvin

- ➔ Adphorus' proprietary predictive optimization engine
- ➔ Uses *machine learning* and *predictive models* to manage *bids* and *allocate budget* to optimize the performance of Facebook campaigns
- ➔ Automates running, managing and optimizing loads of campaigns
  - Saves time and energy
  - Makes work more feasible and productive



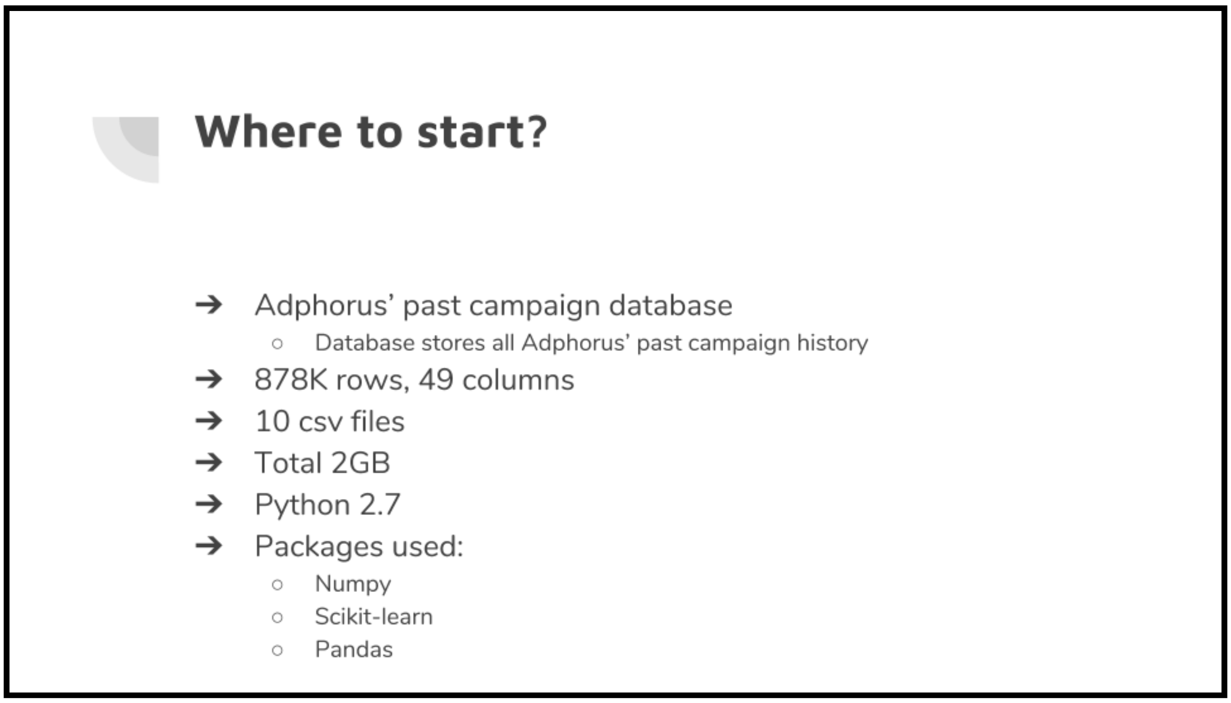
Goal of Marvin

- ➔ To increase the efficiency and performance of Facebook campaigns through increasing ad spend as much as possible while keeping the cost per action at or below Target CPA



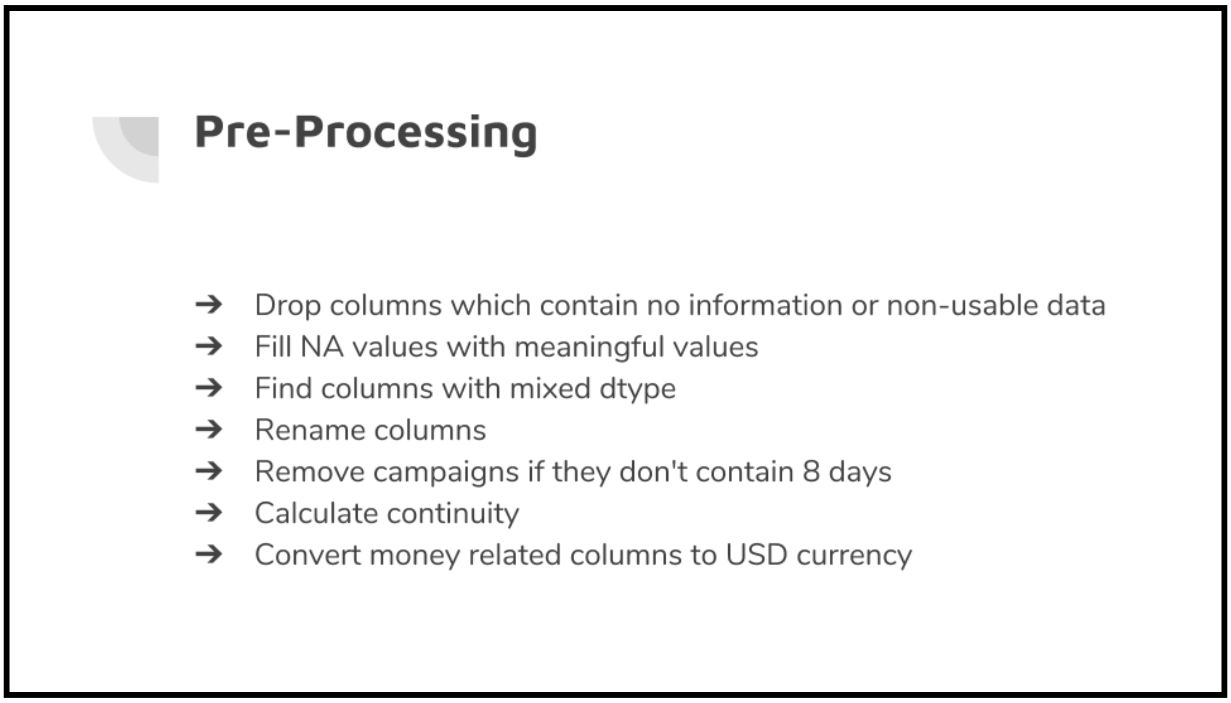
Why predict Facebook CPM?

- ➔ Improve Marvin optimization
- ➔ Improve campaign performance
- ➔ Increase Marvin dependency



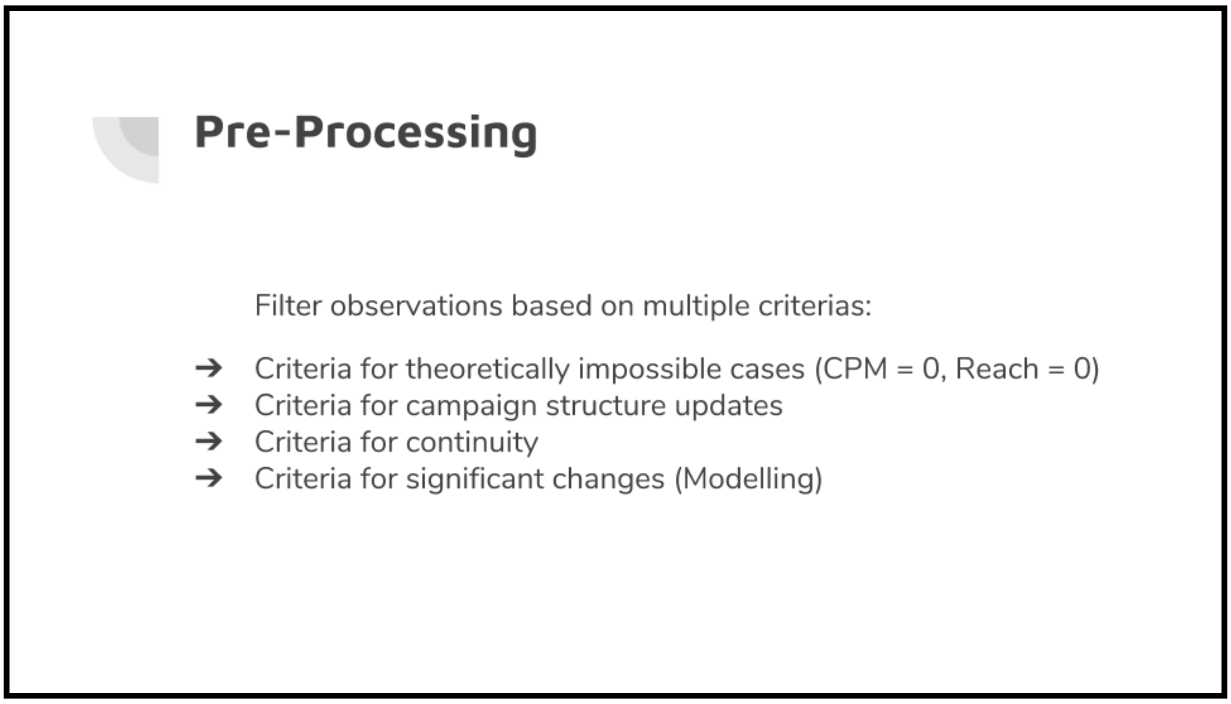
Where to start?

- ➔ Adphorus' past campaign database
  - Database stores all Adphorus' past campaign history
- ➔ 878K rows, 49 columns
- ➔ 10 csv files
- ➔ Total 2GB
- ➔ Python 2.7
- ➔ Packages used:
  - Numpy
  - Scikit-learn
  - Pandas



Pre-Processing

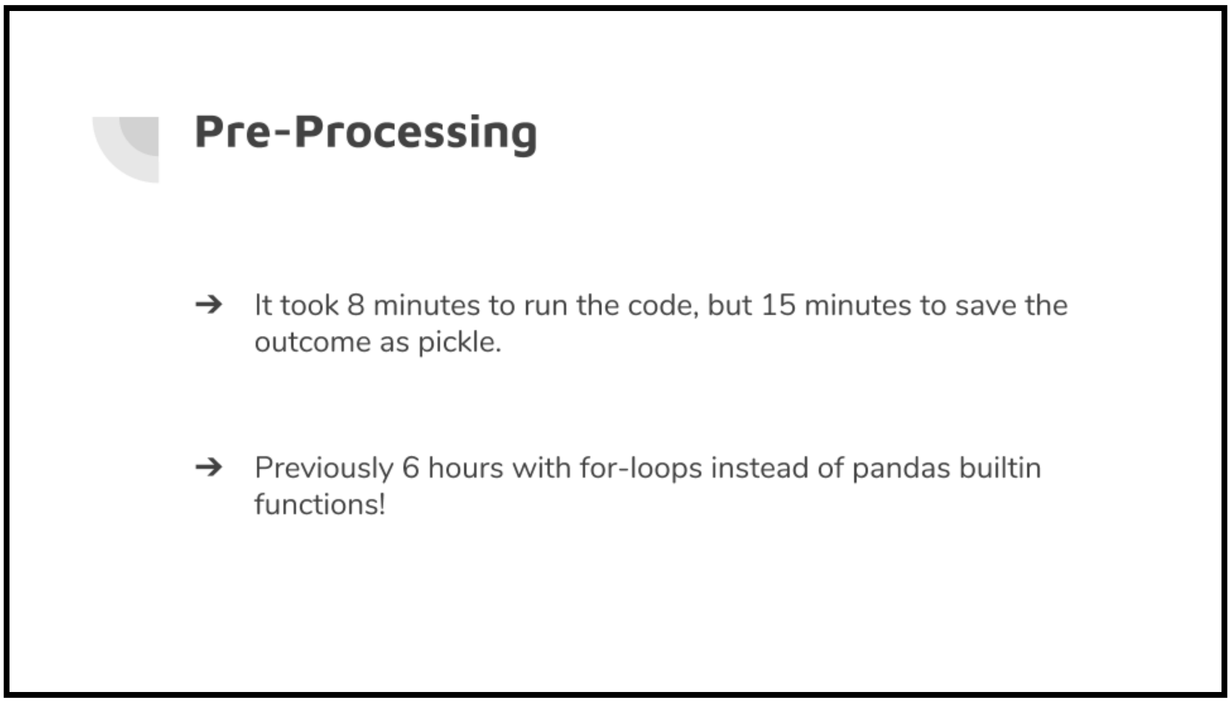
- ➔ Drop columns which contain no information or non-usable data
- ➔ Fill NA values with meaningful values
- ➔ Find columns with mixed dtype
- ➔ Rename columns
- ➔ Remove campaigns if they don't contain 8 days
- ➔ Calculate continuity
- ➔ Convert money related columns to USD currency



Pre-Processing

Filter observations based on multiple criterias:

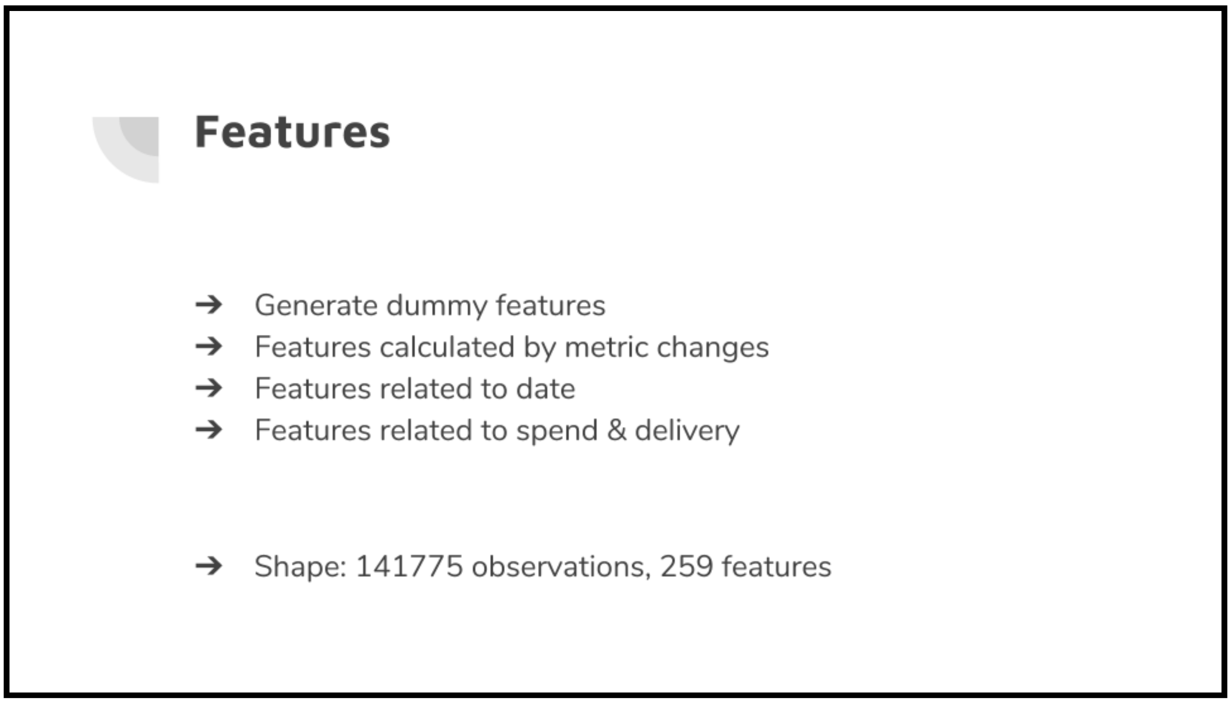
- ➔ Criteria for theoretically impossible cases (CPM = 0, Reach = 0)
- ➔ Criteria for campaign structure updates
- ➔ Criteria for continuity
- ➔ Criteria for significant changes (Modelling)



Pre-Processing

→ It took 8 minutes to run the code, but 15 minutes to save the outcome as pickle.

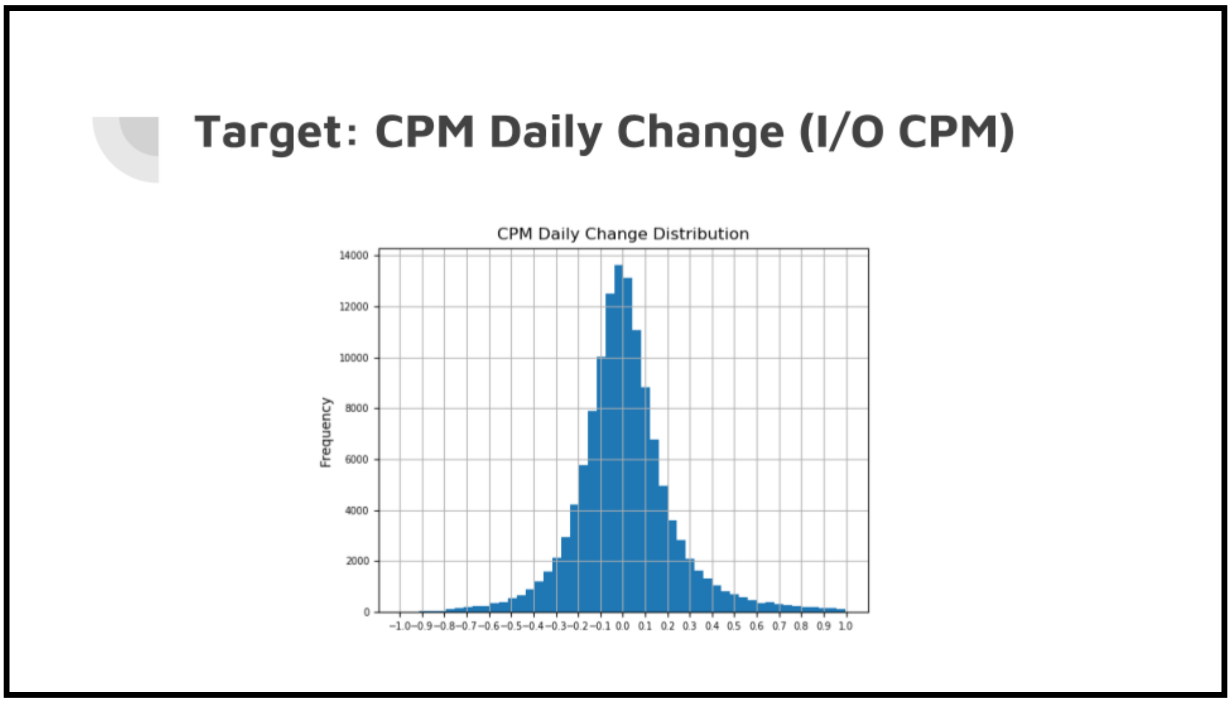
→ Previously 6 hours with for-loops instead of pandas builtin functions!



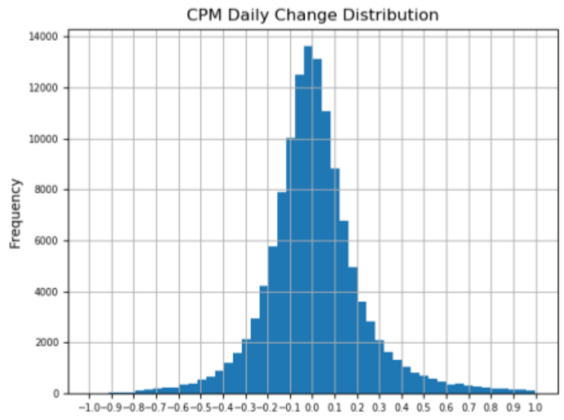
Features

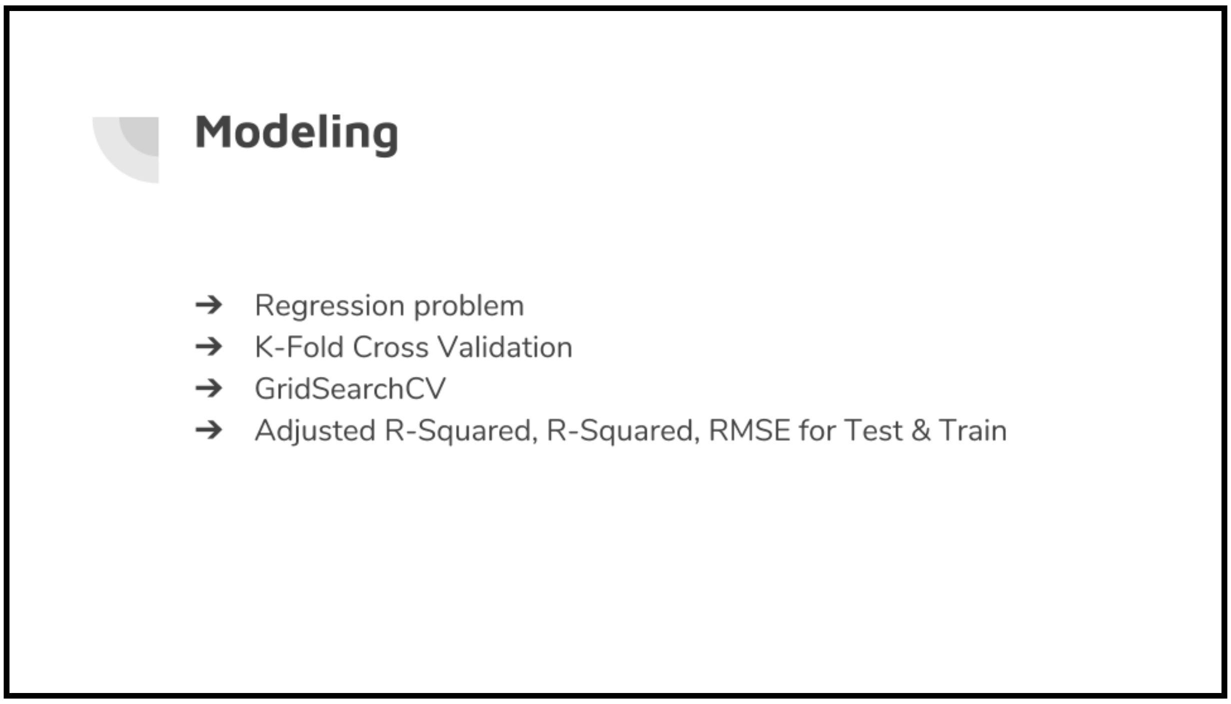
→ Generate dummy features
→ Features calculated by metric changes
→ Features related to date
→ Features related to spend & delivery

→ Shape: 141775 observations, 259 features



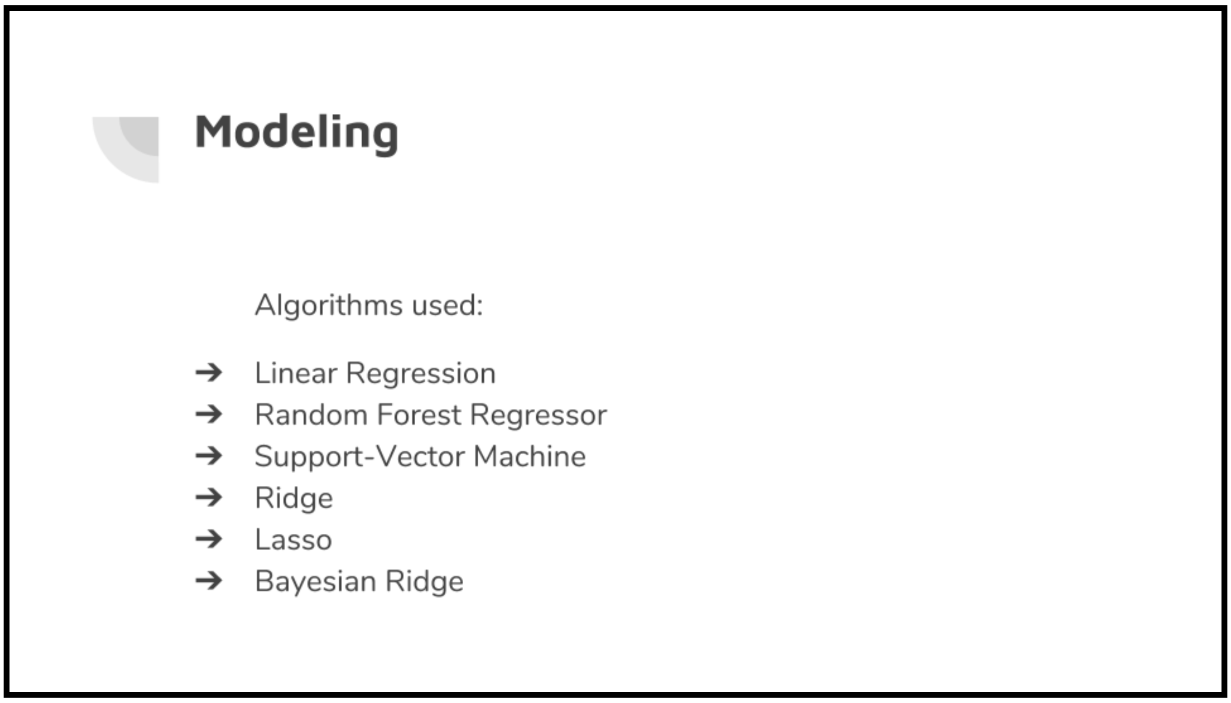
Target: CPM Daily Change (I/O CPM)





Modeling

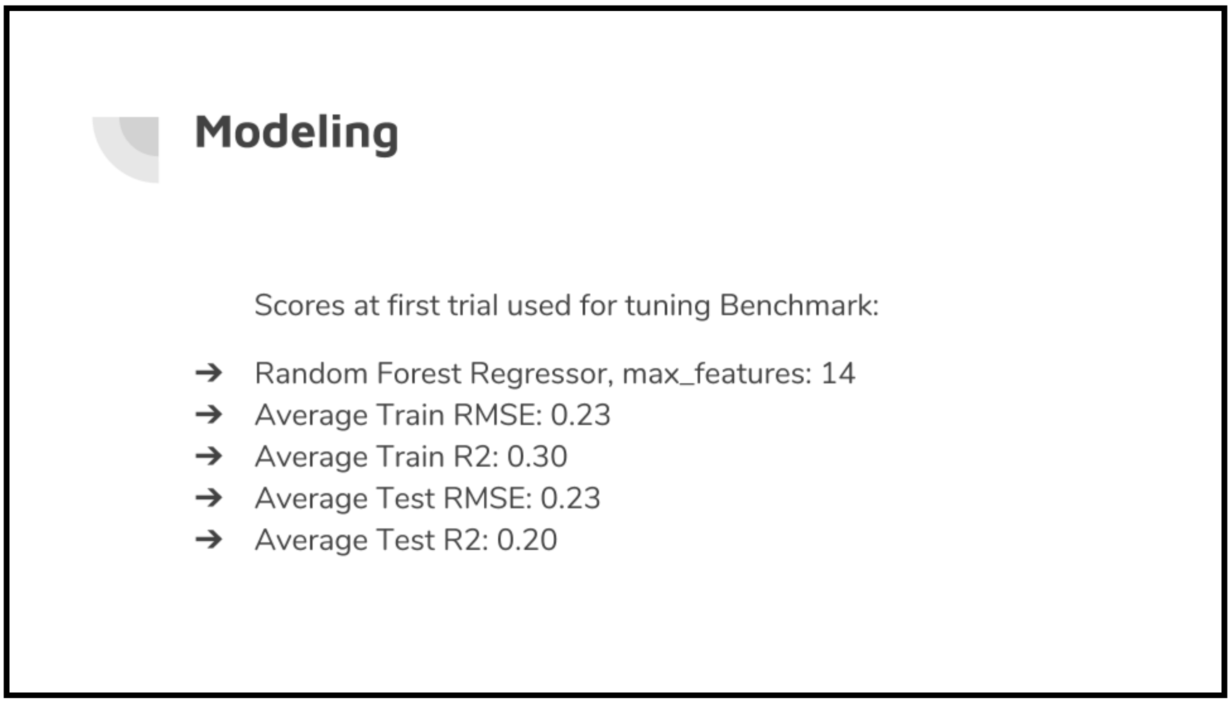
- ➔ Regression problem
- ➔ K-Fold Cross Validation
- ➔ GridSearchCV
- ➔ Adjusted R-Squared, R-Squared, RMSE for Test & Train



Modeling

Algorithms used:

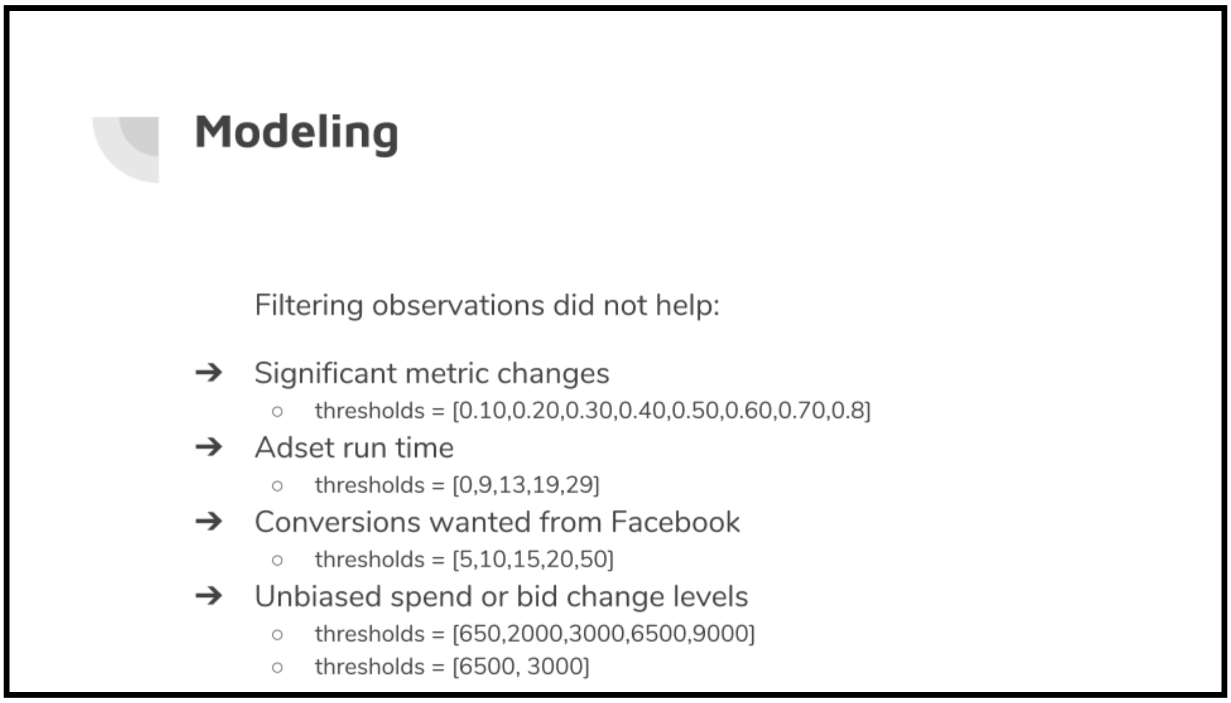
- ➔ Linear Regression
- ➔ Random Forest Regressor
- ➔ Support-Vector Machine
- ➔ Ridge
- ➔ Lasso
- ➔ Bayesian Ridge



Modeling

Scores at first trial used for tuning Benchmark:

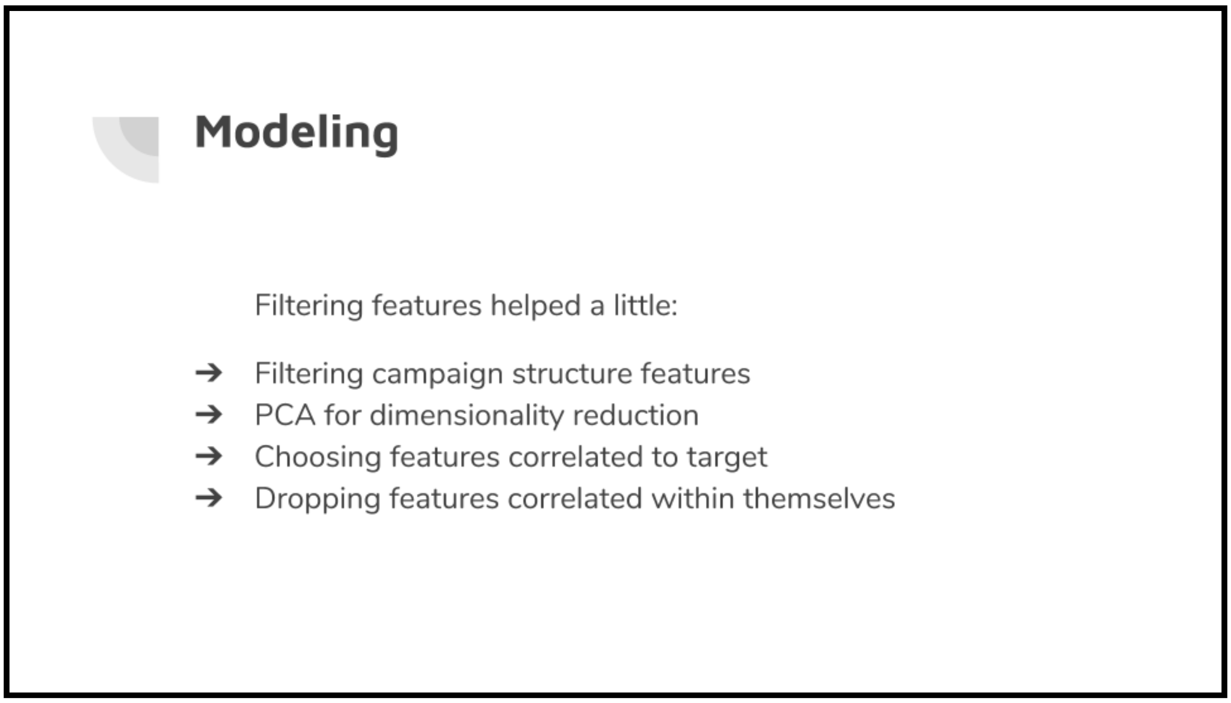
- ➔ Random Forest Regressor, max_features: 14
- ➔ Average Train RMSE: 0.23
- ➔ Average Train R2: 0.30
- ➔ Average Test RMSE: 0.23
- ➔ Average Test R2: 0.20



Modeling

Filtering observations did not help:

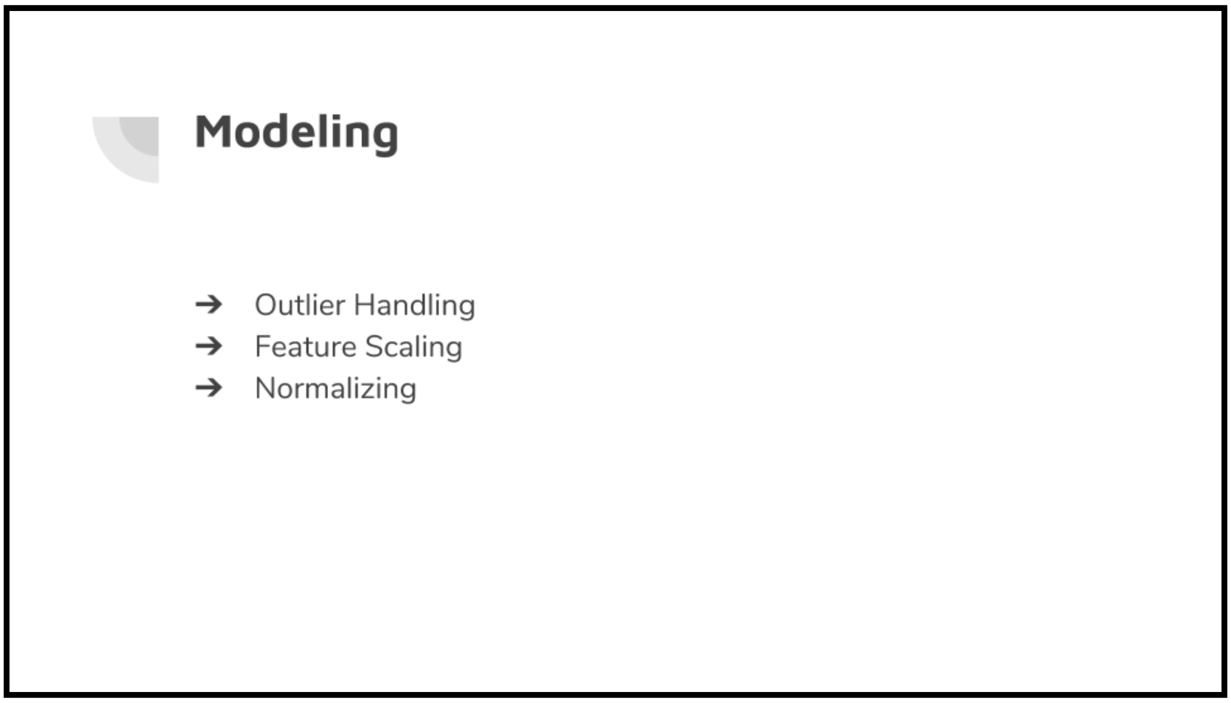
- ➔ Significant metric changes
  - ○ thresholds = [0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.8]
- ➔ Adset run time
  - ○ thresholds = [0,9,13,19,29]
- ➔ Conversions wanted from Facebook
  - ○ thresholds = [5,10,15,20,50]
- ➔ Unbiased spend or bid change levels
  - ○ thresholds = [650,2000,3000,6500,9000]
  - ○ thresholds = [6500, 3000]



Modeling

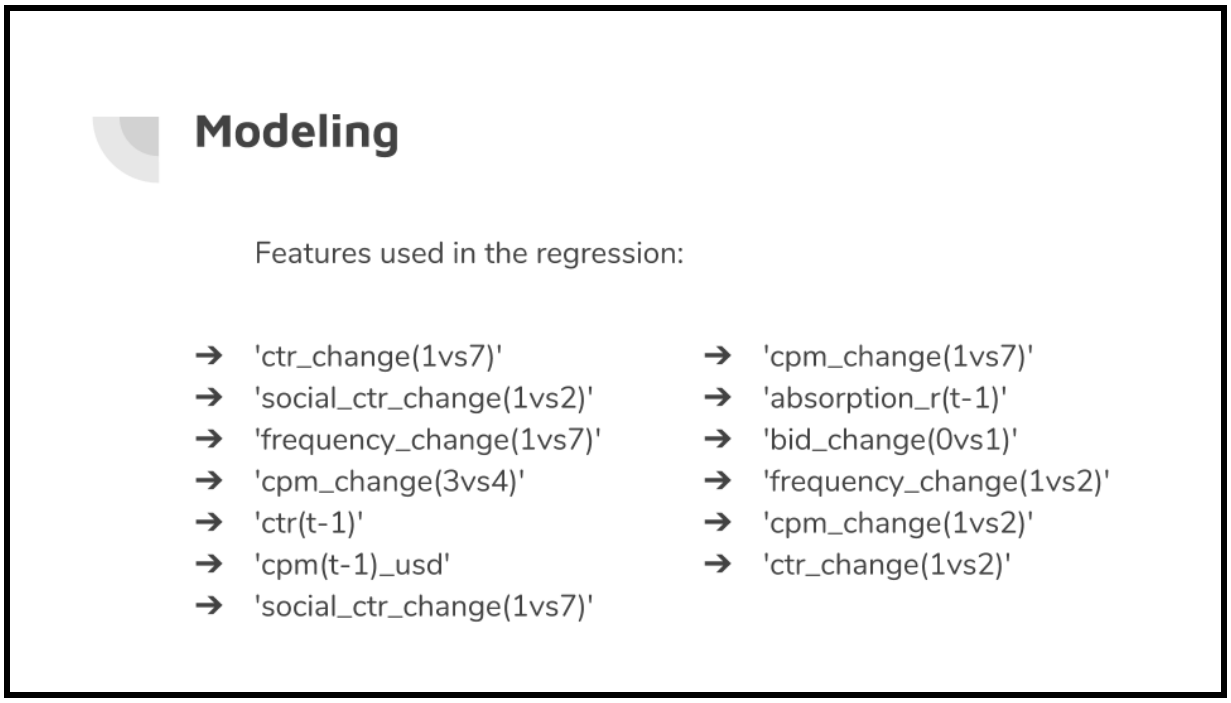
Filtering features helped a little:

- ➔ Filtering campaign structure features
- ➔ PCA for dimensionality reduction
- ➔ Choosing features correlated to target
- ➔ Dropping features correlated within themselves



Modeling

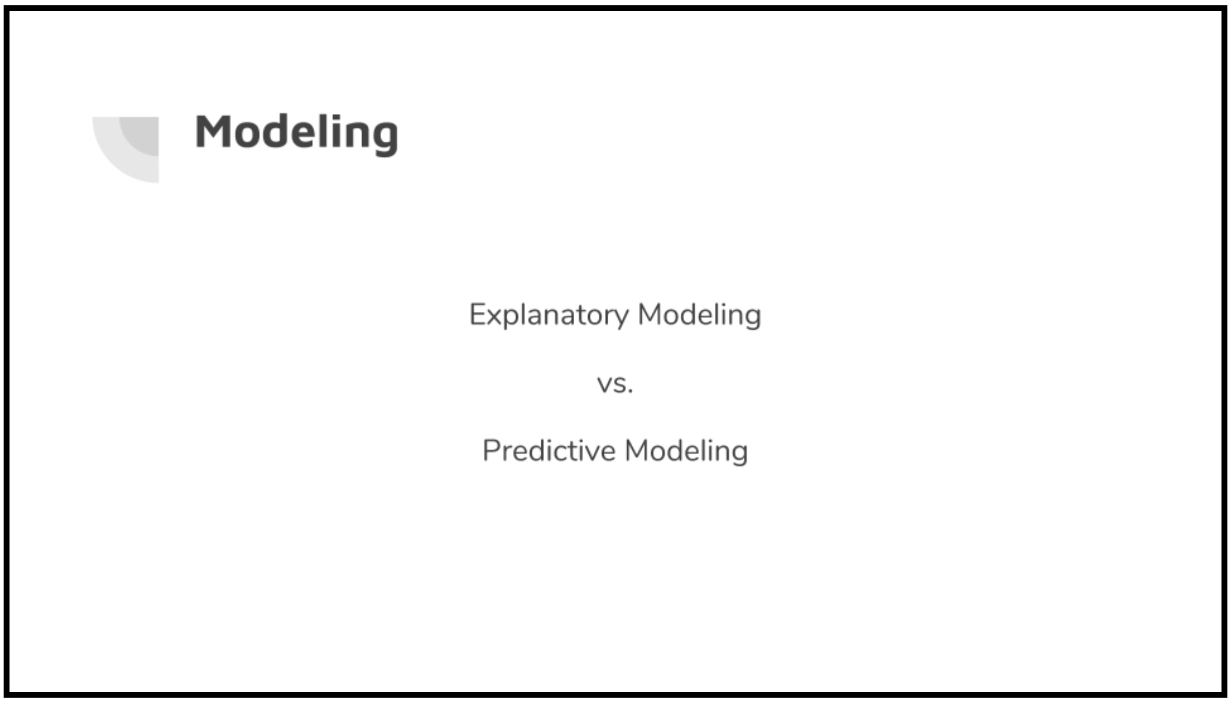
- ➔ Outlier Handling
- ➔ Feature Scaling
- ➔ Normalizing



Modeling

Features used in the regression:

- ➔ 'ctr_change(1vs7)'
- ➔ 'social_ctr_change(1vs2)'
- ➔ 'frequency_change(1vs7)'
- ➔ 'cpm_change(3vs4)'
- ➔ 'ctr(t-1)'
- ➔ 'cpm(t-1)_usd'
- ➔ 'social_ctr_change(1vs7)'
- ➔ 'cpm_change(1vs7)'
- ➔ 'absorption_r(t-1)'
- ➔ 'bid_change(0vs1)'
- ➔ 'frequency_change(1vs2)'
- ➔ 'cpm_change(1vs2)'
- ➔ 'ctr_change(1vs2)'

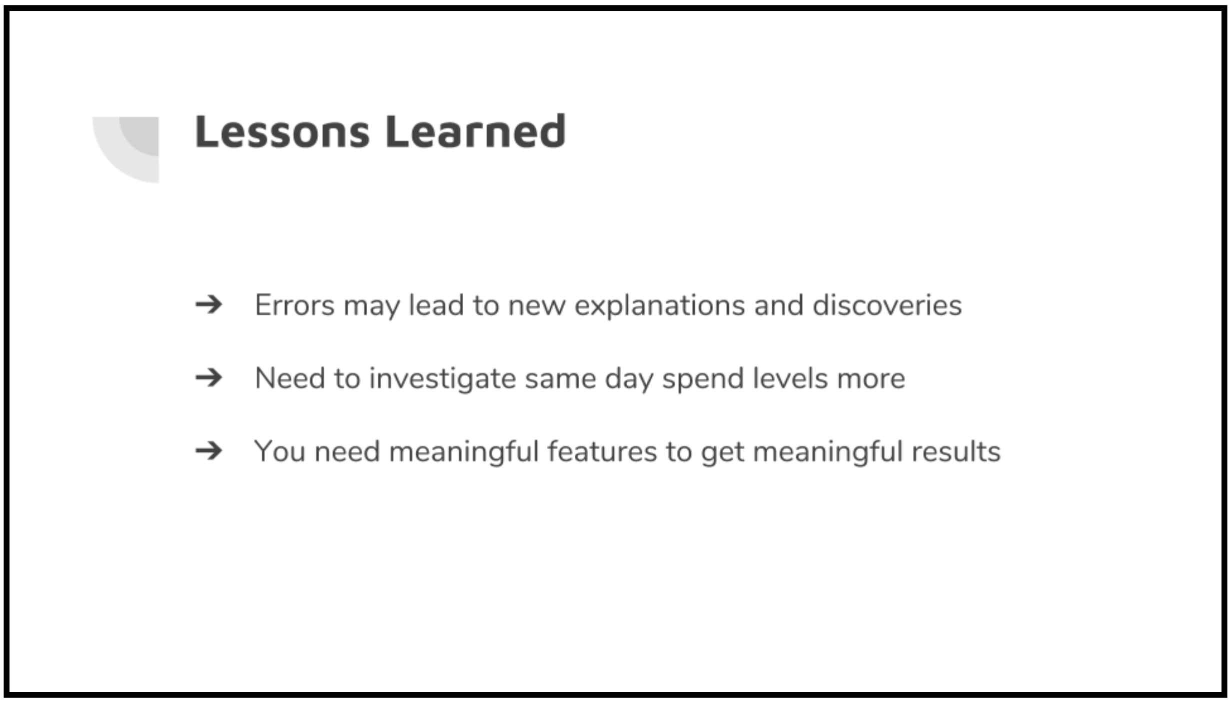


Modeling

Explanatory Modeling

vs.

Predictive Modeling



Lessons Learned

- ➔ Errors may lead to new explanations and discoveries
- ➔ Need to investigate same day spend levels more
- ➔ You need meaningful features to get meaningful results

Semih Tekten

semih@adphorus.com

BOĞAZİÇİ ÜNİVERSİTESİ
1863
MEF
UNIVERSITY

APPENDIX C: Code Base

Following Python files and codes have been used for preparing data, generating features and executing models.

preprocess.py:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Jul  8 13:42:59 2018

@author: tektensemih
"""

######################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES

######################################################################
######## PREPROCESSING ###############################################
######################################################################


# import libraries
import pandas as pd
import numpy as np
import glob, os
import time
from datetime import datetime, timedelta

# Start timer
start_time = time.time()

# Change path
path = os.path.expanduser("~/Documents/Scripts/CPM/Capstone/")
os.chdir(path)
```

```
# Read all CSV files into list
allFiles = glob.glob(path + "/raw_dataset/*.csv")
allFilesList = []

for csv_file in allFiles:
    df = pd.read_csv(csv_file,index_col=None, header=0)
    allFilesList.append(df)
del df, csv_file

# Merge dataframes
db_df = pd.concat(allFilesList)
del allFilesList

# Drop columns which contain no information or non-usable data
cols_to_drop = ['adset_budget_type',
                'brand_id',
                'brand_name',
                'campaign_auto_optimization_action',
                'campaign_custom_event_type',
                'campaign_fb_id',
                'campaign_health_details',
                'campaign_pacing_type',
                'campaign_sf_opp_id',
                'changes',
                'company_id',
                'company_name',
                'fbid',
                'meta_run_id',
                'meta_save_time',
                'nonattr_run_id',
                'nonattr_save_time',
                'predicted_attr',
                'predicted_metric',
                'predicted_revenue_metric',
                'targeting_size',
                'user_id',
                'user_name',
                'suggested_bid_min',
                'suggested_bid_median',
                'suggested_bid_max']
db_df.drop(cols_to_drop, axis=1, inplace = True)
del cols_to_drop
```

```
# Remove rows with zero reach. It's theoretically impossible.
db_df = db_df[db_df['reach']>0]

# Read rates table into dataframe & save rates to adset rows
rates = pd.read_csv(path + "/rates/rates.csv",index_col=None, header=0)
rates['date_pd'] = pd.to_datetime(rates['date'], format='%Y-%m-%d')
rates.drop('date', axis=1, inplace = True)
rates.drop_duplicates(subset=['date_pd','currency'], keep='last', inplace=True)

# Fill missing days with previous days' values
rates.set_index(['date_pd','currency'], inplace=True)
rates = rates.unstack('currency')
rates  = rates.reindex(pd.date_range('2017-01-20', '2018-07-08', freq='D')).fillna(value=None,
method='ffill', axis=0)
rates        =        rates.stack('currency').reset_index().rename(columns={'level_0':'date_pd'},
inplace=False)

# Join currency rates
db_df.rename(columns={'adaccount_currency':'currency'}, inplace=True)
db_df['date_pd'] = pd.to_datetime(db_df['day'], format='%Y-%m-%d')
db_df['date_pd_7db'] = db_df['date_pd'] - pd.Timedelta(7, unit='d')
db_df['date_pd_30db'] = db_df['date_pd'] - pd.Timedelta(30, unit='d')

db_df  =  pd.merge(db_df,  rates,  left_index=True,  how='left',  on=['date_pd',  'currency'],
suffixes=('_0db', '0db'))
db_df.rename(columns={'rate':'rate_0day'}, inplace=True)
rates.rename(columns={'date_pd':'date_pd_7db'}, inplace=True)
db_df = pd.merge(db_df, rates, left_index=True, how='left', on=['date_pd_7db', 'currency'],
suffixes=('_7db', '_7db'))
db_df.rename(columns={'rate':'rate_7day'}, inplace=True)
rates.rename(columns={'date_pd_7db':'date_pd_30db'}, inplace=True)
db_df = pd.merge(db_df, rates, left_index=True, how='left', on=['date_pd_30db', 'currency'],
suffixes=('_30db', '_30db'))
db_df.rename(columns={'rate':'rate_30day'}, inplace=True)

db_df.rename(columns={'currency':'adaccount_currency'}, inplace=True)

cols_to_drop =  ['date_pd', 'date_pd_7db', 'date_pd_30db']
db_df.drop(cols_to_drop, axis=1, inplace = True)
del cols_to_drop, rates
```

```
# Transfer db_df to raw_df
raw_df = db_df.copy()
del db_df

# Get summary info of raw_df
print(raw_df.info())

# Fill NA values with meaningful values
fillings = {
            'campaign_attribution_window':'actions',
            'campaign_marvin_target_cpa':0.,
            'campaign_marvin_target_roas':0.,
            'adset_optimization_goal':'thisvalueneverexists',
            'campaign_promoted_object':'thisvalueneverexists',
            'campaign_is_dynamic_ad':False,
            'adset_pacing_type':"['standard']",
            'campaign_health_status':'GOOD',
            'campaign_spend_cap':0,
            'campaign_schedule':'no_schedule',
            'rate_0day':-1.,
            'rate_7day':-1.,
            'rate_30day':-1.
            }

for column, fillwith in fillings.iteritems():
    raw_df[column].fillna(fillwith, inplace=True)
del fillwith

# Get summary info of raw_df, again
print(raw_df.info())

# Find columns with mixed dtype
for col in raw_df.columns:
    weird = (raw_df[[col]].applymap(type) != raw_df[[col]].iloc[0].apply(type)).any(axis=1)
    if len(raw_df[weird]) > 0:
        print(col)
del col, weird

"""
Returns:
None
So none of the columns has mixed dtype or Null value.
```

```
"""

# Function for dummy feature generating
def dummy_generator(df, new_col, base_col, searched_str):
    df[new_col] = np.where(df[base_col].str.contains(searched_str, case=False), 1, 0)

# Generate dummy features
dummyfeature_list = [   # Time zone
                        [raw_df, 'is_tz_asia', 'adaccount_timezone', 'Asia'],
                        [raw_df, 'is_tz_america', 'adaccount_timezone', 'America'],
                        [raw_df, 'is_tz_europe', 'adaccount_timezone', 'Europe'],
                        [raw_df, 'is_tz_atlantic', 'adaccount_timezone', 'Atlantic'],
                        [raw_df, 'is_tz_australia', 'adaccount_timezone', 'Australia'],
                        [raw_df, 'is_tz_pacific', 'adaccount_timezone', 'Pacific'],

                        # Billing
                        [raw_df, 'is_billing_impressions', 'adset_billing_event', 'IMPRESSIONS'],
                        [raw_df, 'is_billing_clicks', 'adset_billing_event', 'LINK_CLICKS'],
                        [raw_df, 'is_billing_likes', 'adset_billing_event', 'PAGE_LIKES'],
                        [raw_df, 'is_billing_mai', 'adset_billing_event', 'APP_INSTALLS'],
                        [raw_df, 'is_billing_video', 'adset_billing_event', 'VIDEO_VIEWS'],
                        [raw_df, 'is_billing_post', 'adset_billing_event', 'POST_ENGAGEMENT'],

                        # Optimization Goal
                        [raw_df, 'is_optgoal_appinstall', 'adset_optimization_goal', 'APP_INSTALLS'],
                        [raw_df, 'is_optgoal_eventresponse', 'adset_optimization_goal',
'EVENT_RESPONSES'],
                        [raw_df, 'is_optgoal_impression', 'adset_optimization_goal', 'IMPRESSIONS'],
                        [raw_df, 'is_optgoal_leadgeneration', 'adset_optimization_goal',
'LEAD_GENERATION'],
                        [raw_df, 'is_optgoal_click', 'adset_optimization_goal', 'LINK_CLICKS'],
                        [raw_df, 'is_optgoal_offsiteconversion', 'adset_optimization_goal',
'OFFSITE_CONVERSIONS'],
                        [raw_df, 'is_optgoal_pagelike', 'adset_optimization_goal', 'PAGE_LIKES'],
                        [raw_df, 'is_optgoal_postengagement', 'adset_optimization_goal',
'POST_ENGAGEMENT'],
                        [raw_df, 'is_optgoal_reach', 'adset_optimization_goal', 'REACH'],
                        [raw_df, 'is_optgoal_videoview', 'adset_optimization_goal', 'VIDEO_VIEWS'],

                        # Campaign Objective
                        [raw_df, 'is_campobjective_appinstall', 'campaign_objective',
'APP_INSTALLS'],
                    [raw_df, 'is_campobjective_brandawareness', 'campaign_objective',
'BRAND_AWARENESS'],
                    [raw_df, 'is_campobjective_conversion', 'campaign_objective',
'CONVERSIONS'],
                    [raw_df, 'is_campobjective_eventresponse', 'campaign_objective',
'EVENT_RESPONSES'],
                    [raw_df, 'is_campobjective_leadgeneration', 'campaign_objective',
'LEAD_GENERATION'],
        [raw_df, 'is_campobjective_click', 'campaign_objective', 'LINK_CLICKS'],
                    [raw_df, 'is_campobjective_appengagement', 'campaign_objective',
'MOBILE_APP_ENGAGEMENT'],
                    [raw_df, 'is_campobjective_mobileappinstall', 'campaign_objective',
'MOBILE_APP_INSTALLS'],
        [raw_df, 'is_campobjective_pagelike', 'campaign_objective', 'PAGE_LIKES'],
                    [raw_df, 'is_campobjective_postengagement', 'campaign_objective',
'POST_ENGAGEMENT'],
                    [raw_df, 'is_campobjective_pcs', 'campaign_objective',
'PRODUCT_CATALOG_SALES'],
        [raw_df, 'is_campobjective_reach', 'campaign_objective', 'REACH'],
                    [raw_df, 'is_campobjective_videoview', 'campaign_objective',
'VIDEO_VIEWS'],

        # Promoted Object
                    [raw_df, 'is_promotedobject_catalog', 'campaign_promoted_object',
'product_catalog_id'],
        [raw_df, 'is_promotedobject_pixel', 'campaign_promoted_object', 'pixel_id'],
        [raw_df, 'is_promotedobject_page', 'campaign_promoted_object', 'page_id'],
                [raw_df, 'is_promotedobject_objectstoreurl', 'campaign_promoted_object',
'object_store_url'],
          [raw_df, 'is_promotedobject_customeventtype', 'campaign_promoted_object',
'custom_event_type'],
                  [raw_df, 'is_promotedobject_application', 'campaign_promoted_object',
'application_id'],

        # Campaign Attribution Window
        [raw_df, 'is_attr_28dc', 'campaign_attribution_window', '28d_click'],
        [raw_df, 'is_attr_7dc', 'campaign_attribution_window', '7d_click'],
        [raw_df, 'is_attr_1dc', 'campaign_attribution_window', '1d_click'],
        [raw_df, 'is_attr_28dv', 'campaign_attribution_window', '28d_view'],
        [raw_df, 'is_attr_7dv', 'campaign_attribution_window', '7d_view'],
        [raw_df, 'is_attr_1dv', 'campaign_attribution_window', '1d_view'],
        [raw_df, 'is_attr_default', 'campaign_attribution_window', 'actions'],
                    # Adset Pacing Type
                    [raw_df, 'is_nopacing', 'adset_pacing_type', "['no_pacing']"],

                    # Campaign Health Status
                    [raw_df, 'is_camphealth_good', 'campaign_health_status', 'GOOD'],
                    [raw_df, 'is_camphealth_crit', 'campaign_health_status', 'CRITICAL'],
                  ]

for feature in dummyfeature_list:
    dummy_generator(*feature)
del feature

# Features related to date
raw_df['date_pd'] = pd.to_datetime(raw_df['day'], format='%Y-%m-%d')
raw_df['day_of_week'] = raw_df['date_pd'].dt.dayofweek
raw_df['day_of_year'] = raw_df['date_pd'].dt.dayofyear
raw_df['month'] = raw_df['date_pd'].dt.month
raw_df['quarter'] = raw_df['date_pd'].dt.quarter

# Features related to DAO, Dynamic Ads, Cost/Revenue and Target
raw_df['is_dao'] = np.where(raw_df['campaign_da_optimizer_status'] == True, 1, 0)
raw_df['is_dynamicad'] = np.where((pd.isna(raw_df['campaign_is_dynamic_ad'])) |
(raw_df['campaign_is_dynamic_ad'] == False), 0, 1)
raw_df['is_revenue'] = np.where(raw_df['campaign_optimize_revenue'] == True, 1, 0)
raw_df['is_target'] = np.where((raw_df['campaign_marvin_target_cpa'] > 0) |
(raw_df['campaign_marvin_target_roas'] > 0), 1, 0)

# Features related to spend & delivery
raw_df['ctr(t-0)'] = raw_df['clicks'] / raw_df['impressions']
raw_df['social_ctr(t-0)'] = raw_df['social_clicks'] / raw_df['impressions']
raw_df['social_reach_ratio(t-0)'] = raw_df['social_reach'] / raw_df['reach']
raw_df['frequency(t-0)'] = raw_df['impressions'] / raw_df['reach']
raw_df['absorption_r(t-0)'] = raw_df['spend'] / raw_df['adset_budget']
raw_df['cpm(t-0)'] = raw_df['spend'] / raw_df['impressions'] * 1000.
raw_df['cp_thousand_reach(t-0)'] = raw_df['spend'] / raw_df['reach'] * 1000.
raw_df['budget/camp_budget(t-0)'] = raw_df['adset_budget'] / raw_df['campaign_budget']
raw_df['spend/camp_budget(t-0)'] = raw_df['spend'] / raw_df['campaign_budget']
raw_df['budget/bid(t-0)'] = raw_df['adset_budget'] / raw_df['adset_bid_amount']

# Rename columns
raw_df.rename(columns={# Features to be shifted, not related to spend and/or currency
                'date_pd':'date(t-0)',
                'adaccount_timezone':'adaccount_timezone(t-0)',
                'adset_billing_event':'adset_billing_event(t-0)',
                'adset_optimization_goal':'adset_optimization_goal(t-0)',
                'campaign_da_optimizer_status':'campaign_da_optimizer_status(t-0)',
                'campaign_is_dynamic_ad':'campaign_is_dynamic_ad(t-0)',
                'campaign_objective':'campaign_objective(t-0)',
                'campaign_promoted_object':'campaign_promoted_object(t-0)',
                'targeting_spec':'targeting_spec(t-0)',
                'campaign_attribution_window':'campaign_attribution_window(t-0)',
                'campaign_optimize_revenue':'campaign_optimize_revenue(t-0)',
                'adset_pacing_type':'adset_pacing_type(t-0)',
                'campaign_health_status':'campaign_health_status(t-0)',
                'campaign_status':'campaign_status(t-0)',
                'adset_status':'adset_status(t-0)',
                # Features to be shifted, related to spend and/or currency
                'adset_bid_amount':'bid(t-0)',
                'spend':'spend(t-0)',
                'adset_budget':'budget(t-0)',
                'campaign_budget':'campaign_budget(t-0)',
                'adaccount_currency':'adaccount_currency(t-0)',
                'campaign_marvin_target_cpa':'campaign_marvin_target_cpa(t-0)',
                'campaign_marvin_target_roas':'campaign_marvin_target_roas(t-0)'
                }, inplace=True)

# Remove adsets if they don't contain 8 days
raw_df = raw_df.groupby('adset_id').filter(lambda x: len(x) >= 8).reset_index()
print('Count of adsets with at least 7 previous observation days:')
print(raw_df['adset_id'].nunique())

# Reindex main dataframe using adset IDs and dates
raw_df.set_index(['adset_id', 'date(t-0)'], drop=False, append=False, inplace=True,
verify_integrity=False)
raw_df.sort_index(axis=0, level=[0,1], ascending=[True,True], inplace=True)

# Find starting day of each adset
adset_min_date =
raw_df.groupby(level=0)['date(t-0)'].min().reset_index().rename(columns={'date(t-0)':'min_da
te'}, inplace=False)

# Generate function to shift columns
def shifter(df, column):
    # Column should be string
    for preday in range(1,8):
        df[column+'(t-'+str(preday)+')'] = df.groupby(level=0)[column+'(t-0)'].shift(preday)
    print('Finished shifting for column %s.') % (column)

# Shift columns for each adset
cols_to_shift                                                                          =
['bid','spend','budget','absorption_r','cpm','ctr','social_ctr','social_reach_ratio','date','campaign_
budget', \

'adaccount_currency','adaccount_timezone','adset_billing_event','adset_optimization_goal', \

'campaign_da_optimizer_status','campaign_is_dynamic_ad','campaign_objective','campaign_p
romoted_object', \
                    'targeting_spec','campaign_attribution_window','campaign_marvin_target_cpa', \
                                                                    'campaign_marvin_target_roas',
'campaign_optimize_revenue','adset_pacing_type','campaign_health_status', \
                    'campaign_status', 'adset_status', 'frequency', 'cp_thousand_reach', \
                    'budget/camp_budget', 'spend/camp_budget', 'budget/bid', \
                ]

for shiftcolumn in cols_to_shift:
    shifter(raw_df, shiftcolumn)
del shiftcolumn

# Reindex main dataframe back to integer values
raw_df.reset_index(drop=True, inplace=True)
raw_df.drop('index', axis=1, inplace = True)

# Store starting day of each adset
raw_df = pd.merge(raw_df, adset_min_date, how='left', on='adset_id')
del adset_min_date

# Calculate adset run time
raw_df['adset_run_time'] = (raw_df['date(t-0)'] - raw_df['min_date']).dt.days + 1

# Transfer raw dataframe into half-clean dataframe
half_clean_df = raw_df.copy()
del raw_df

# Calculate weekly weighted CPM
week_weights = [0.36,0.24,0.16,0.10,0.07,0.04,0.03]
```

```
half_clean_df['weighted_cpm_lastweek'] =    half_clean_df['cpm(t-7)'] * week_weights[6] + \
                                            half_clean_df['cpm(t-6)'] * week_weights[5] + \
                                            half_clean_df['cpm(t-5)'] * week_weights[4] + \
                                            half_clean_df['cpm(t-4)'] * week_weights[3] + \
                                            half_clean_df['cpm(t-3)'] * week_weights[2] + \
                                            half_clean_df['cpm(t-2)'] * week_weights[1] + \
                                            half_clean_df['cpm(t-1)'] * week_weights[0]


def change_calculator(df, column):
    for i in range(0,7):
                    df[column+'_change'+'('+str(i)+'vs'+str(i+1)+')'] = (df[column+'(t-'+str(i)+')'] -
df[column+'(t-'+str(i+1)+')']) / df[column+'(t-'+str(i+1)+')']
    df[column+'_change(1vs7)'] = (df[column+'(t-1)'] - df[column+'(t-7)']) / df[column+'(t-7)']

cols_to_calcchange = ['ctr', 'social_ctr', 'social_reach_ratio', 'bid', 'spend', \
                      'budget', 'absorption_r', 'cpm', 'frequency', 'cp_thousand_reach', \
                      'budget/camp_budget', 'spend/camp_budget', 'budget/bid', \
                      ]

for col in cols_to_calcchange:
    change_calculator(half_clean_df, col)

# Features calculated by metric changes
half_clean_df['bidchange*spendchange_yesterday'] = half_clean_df['bid_change(1vs2)'] *
half_clean_df['spend_change(1vs2)']
half_clean_df['bidchange*budgetchange_yesterday'] = half_clean_df['bid_change(1vs2)'] *
half_clean_df['budget_change(1vs2)']
half_clean_df['spendchange*budgetchange_yesterday'] =
half_clean_df['spend_change(1vs2)'] * half_clean_df['budget_change(1vs2)']

print('Finished calculating changes.')

# Replace infinity values with zero.
def infinity_fixer(df, column):
    for i in range(0,7):
        df[column+'_change'+'('+str(i)+'vs'+str(i+1)+')'].replace(np.inf, 0, inplace=True)
    df[column+'_change(1vs7)'].replace(np.inf, 0, inplace=True)

cols_to_infinityfix = ['ctr','social_ctr','social_reach_ratio',]

for col in cols_to_infinityfix:
    infinity_fixer(half_clean_df, col)

print('Finished replacing infinity values.')

# Calculate continuity
half_clean_df['date(t-0)-date(t-1)'] = (half_clean_df['date(t-0)'] - half_clean_df['date(t-1)']).dt.days
half_clean_df['date(t-1)-date(t-2)'] = (half_clean_df['date(t-1)'] - half_clean_df['date(t-2)']).dt.days
half_clean_df['date(t-2)-date(t-3)'] = (half_clean_df['date(t-2)'] - half_clean_df['date(t-3)']).dt.days
half_clean_df['date(t-3)-date(t-4)'] = (half_clean_df['date(t-3)'] - half_clean_df['date(t-4)']).dt.days
half_clean_df['date(t-4)-date(t-5)'] = (half_clean_df['date(t-4)'] - half_clean_df['date(t-5)']).dt.days
half_clean_df['date(t-5)-date(t-6)'] = (half_clean_df['date(t-5)'] - half_clean_df['date(t-6)']).dt.days
half_clean_df['date(t-6)-date(t-7)'] = (half_clean_df['date(t-6)'] - half_clean_df['date(t-7)']).dt.days

half_clean_df['campaign_budget(t-0)-campaign_budget(t-1)'] = (half_clean_df['campaign_budget(t-0)'] - half_clean_df['campaign_budget(t-1)'])
half_clean_df['campaign_budget(t-1)-campaign_budget(t-2)'] = (half_clean_df['campaign_budget(t-1)'] - half_clean_df['campaign_budget(t-2)'])
half_clean_df['campaign_budget(t-2)-campaign_budget(t-3)'] = (half_clean_df['campaign_budget(t-2)'] - half_clean_df['campaign_budget(t-3)'])
half_clean_df['campaign_budget(t-3)-campaign_budget(t-4)'] = (half_clean_df['campaign_budget(t-3)'] - half_clean_df['campaign_budget(t-4)'])
half_clean_df['campaign_budget(t-4)-campaign_budget(t-5)'] = (half_clean_df['campaign_budget(t-4)'] - half_clean_df['campaign_budget(t-5)'])
half_clean_df['campaign_budget(t-5)-campaign_budget(t-6)'] = (half_clean_df['campaign_budget(t-5)'] - half_clean_df['campaign_budget(t-6)'])
half_clean_df['campaign_budget(t-6)-campaign_budget(t-7)'] = (half_clean_df['campaign_budget(t-6)'] - half_clean_df['campaign_budget(t-7)'])

def samevalue_sevendays(df, col):
    pastseven_list = []
    for i in range(0,8):
        pastseven_list.append(col+'(t-'+str(i)+')')
    df[col+'_8values'] = df[pastseven_list].values.tolist()
    df[col+'_8uniquecount'] = df[col+'_8values'].apply(lambda cell_value: len(set(cell_value)))
```

```
samevaluecols = [
                    'adaccount_currency',
                    'adaccount_timezone',
                    'adset_billing_event',
                    'adset_optimization_goal',
                    'campaign_da_optimizer_status',
                    'campaign_is_dynamic_ad',
                    'campaign_objective',
                    'campaign_promoted_object',
                    'targeting_spec',
                    'campaign_attribution_window',
                    'campaign_marvin_target_cpa',
                    'campaign_marvin_target_roas',
                    'campaign_optimize_revenue',
                    'adset_pacing_type',
                    'campaign_health_status',
                    'campaign_status',
                    'adset_status',
                ]

for column in samevaluecols:
    samevalue_sevendays(half_clean_df, column)

print('Finished calculating continuity.')

# Convert money features to USD currency

def converter(df, col):
    for preday in range(0,8):
        df[col + '(t-' + str(preday) + ')_usd'] = df[col + '(t-' + str(preday) + ')'] / df['rate_7day']
    return df
del col

cols_to_conv = ['cp_thousand_reach','cpm','spend','campaign_marvin_target_roas', \
                'campaign_marvin_target_cpa','campaign_budget','budget','bid', \
                ]

for column in cols_to_conv:
    converter(half_clean_df, column)
del column
```

```
half_clean_df['weighted_cpm_lastweek_usd'] = half_clean_df['weighted_cpm_lastweek'] / half_clean_df['rate_7day']

print('Finished converting currency.')

# Save dataframe to disk for future use
# half_clean_df.to_pickle("df/half_clean_df.gzip", compression = 'gzip')
# It takes 8 minutes to run the code, but 15 minutes to save the outcome as pickle.
# I decided to execute filtering in this script file also.



######################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES
######################################################################
######## FILTERING ###################################################
######################################################################

# Change path

df_beingcleaned = half_clean_df.copy()
# df_beingcleaned with XXXXXXXX rows, YYYYYYY columns
del half_clean_df

col_list = list(df_beingcleaned.columns)

def value_picker(df, col, operator, val, explanation):
    start_row = df.shape[0]
    if operator == 'greater':
        df = df.loc[df[col] > val]
    elif operator == 'smaller':
        pass
    elif operator == 'greater_e':
        pass
    elif operator == 'smaller_e':
        pass
    elif operator == 'equal':
        df = df.loc[df[col] == val]
    elif operator == 'not_equal':
        pass
    end_row= df.shape[0]

    print('########################################')
    print('Column filtered: ' + str(col))
    print('Operator: ' + str(operator))
    print('Value: ' + str(val))
    print('Cause: ' + str(explanation))
    print('Current dimensions: ' + str(df.shape))
    print('Rows dropped: ' + str(start_row-end_row))
    print('########################################')
    return df

value_pick_list = [
                ['date(t-0)-date(t-1)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-1)-date(t-2)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-2)-date(t-3)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-3)-date(t-4)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-4)-date(t-5)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-5)-date(t-6)', 'equal', 1, 'Days should be adjacent.'],
                ['date(t-6)-date(t-7)', 'equal', 1, 'Days should be adjacent.'],

                    ['campaign_budget(t-0)-campaign_budget(t-1)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-1)-campaign_budget(t-2)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-2)-campaign_budget(t-3)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-3)-campaign_budget(t-4)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-4)-campaign_budget(t-5)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-5)-campaign_budget(t-6)', 'equal', 0, 'Campaign budget
change should be 0.'],
                    ['campaign_budget(t-6)-campaign_budget(t-7)', 'equal', 0, 'Campaign budget
change should be 0.'],

                ['budget(t-0)', 'greater', 0, 'Adset budget should be greater than 0.'],
                ['budget(t-1)', 'greater', 0, 'Adset budget should be greater than 0.'],
                ['budget(t-2)', 'greater', 0, 'Adset budget should be greater than 0.'],
                ['budget(t-3)', 'greater', 0, 'Adset budget should be greater than 0.'],
        ['budget(t-4)', 'greater', 0, 'Adset budget should be greater than 0.'],
        ['budget(t-5)', 'greater', 0, 'Adset budget should be greater than 0.'],
        ['budget(t-6)', 'greater', 0, 'Adset budget should be greater than 0.'],
        ['budget(t-7)', 'greater', 0, 'Adset budget should be greater than 0.'],

            ['campaign_budget(t-0)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-1)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-2)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-3)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-4)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-5)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-6)', 'greater', 0, 'Campaign budget should be greater than
0.'],
            ['campaign_budget(t-7)', 'greater', 0, 'Campaign budget should be greater than
0.'],

        ['cpm(t-0)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-1)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-2)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-3)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-4)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-5)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-6)', 'greater', 0, 'CPM should be greater than 0.'],
        ['cpm(t-7)', 'greater', 0, 'CPM should be greater than 0.'],

        ['cp_thousand_reach(t-0)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-1)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-2)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-3)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-4)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-5)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-6)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],
        ['cp_thousand_reach(t-7)', 'greater', 0, 'CP 1000 Reach should be greater than 0.'],

        ['frequency(t-0)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-1)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-2)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-3)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-4)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-5)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-6)', 'greater', 0, 'Frequency should be greater than 0.'],
        ['frequency(t-7)', 'greater', 0, 'Frequency should be greater than 0.'],

        ['bid(t-0)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-1)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-2)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-3)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-4)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-5)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-6)', 'greater', 0, 'Bid should be greater than 0.'],
        ['bid(t-7)', 'greater', 0, 'Bid should be greater than 0.'],

        ['spend(t-0)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-1)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-2)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-3)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-4)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-5)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-6)', 'greater', 0, 'Spend should be greater than 0.'],
        ['spend(t-7)', 'greater', 0, 'Spend should be greater than 0.'],

        ['adaccount_currency_8uniquecount', 'equal', 1, 'Ad account currency should not change.'],
        ['adaccount_timezone_8uniquecount', 'equal', 1, 'Ad account timezone should not change.'],
        ['adset_billing_event_8uniquecount', 'equal', 1, 'Billing event should not change.'],
            ['adset_optimization_goal_8uniquecount', 'equal', 1, 'Optimization goal should not change.'],
            ['campaign_da_optimizer_status_8uniquecount', 'equal', 1, 'DAO status should not change.'],
            ['campaign_is_dynamic_ad_8uniquecount', 'equal', 1, 'Dynamic ad status should not change.'],
            ['campaign_objective_8uniquecount', 'equal', 1, 'Campaign objective should not change.'],
            ['campaign_promoted_object_8uniquecount', 'equal', 1, 'Promoted object should not change.'],
            ['campaign_attribution_window_8uniquecount', 'equal', 1, 'Attribution window
should not change.'],
                    ['campaign_optimize_revenue_8uniquecount', 'equal', 1, 'Optimize revenue status should not change.'],
                    ['targeting_spec_8uniquecount', 'equal', 1, 'Targeting specifications should not change.'],

                    ['campaign_marvin_target_cpa_8uniquecount', 'equal', 1, 'TCPA should not change.'],
                    ['campaign_marvin_target_roas_8uniquecount', 'equal', 1, 'TROAS should not change.'],

                    ['impressions', 'greater', 99, 'Impressions should be significant.'],

                    ['adset_pacing_type_8uniquecount', 'equal', 1, 'Pacing should not change.'],
                    ['adset_pacing_type(t-0)', 'equal', "['standard']", 'Pacing should be standard.'],

                    ['campaign_health_status_8uniquecount', 'equal', 1, 'Health status should not change.'],
                    ['campaign_health_status(t-0)', 'equal', 'GOOD', 'Health status should be GOOD.'],

                    ['campaign_status_8uniquecount', 'equal', 1, 'Campaign status should not change.'],
                    ['campaign_status(t-0)', 'equal', 'ACTIVE', 'Campaign status should be ACTIVE.'],

                    ['adset_status_8uniquecount', 'equal', 1, 'Adset status should not change.'],
                    ['adset_status(t-0)', 'equal', 'ACTIVE', 'Adset status should be ACTIVE.'],

                    ['weighted_cpm_lastweek', 'greater', 0, 'Weighted CPM should be greater than 0.'],

                    ['adset_run_time', 'greater', 7, 'Adset should be running for at least 8 days.'],

                    ['campaign_schedule', 'equal', 'no_schedule', 'There should be no schedule.'],
                   ]

for val_pick in value_pick_list:
    df_beingcleaned = value_picker(df_beingcleaned, *val_pick)
del val_pick

# Drop columns that carry no information
```

```
df_beingcleaned                                                                          =
df_beingcleaned[df_beingcleaned.columns[~df_beingcleaned.columns.str.endswith('_8unique
count')]]
df_beingcleaned                                                                          =
df_beingcleaned[df_beingcleaned.columns[~df_beingcleaned.columns.str.endswith('8values')]
]

no_info_cols = []
for column in df_beingcleaned:
   if df_beingcleaned[column].nunique(dropna=True) <= 1:
      no_info_cols.append(str(df_beingcleaned[column].name))
del column

df_beingcleaned.drop(no_info_cols, axis=1, inplace=True)

# Find out columns with any NaN value
cols_with_nan  =  df_beingcleaned.columns[df_beingcleaned.isna().any()].tolist()  #  Only
change metrics have Null value.
df_beingcleaned[cols_with_nan]      =      df_beingcleaned[cols_with_nan].fillna(value=0,
inplace=False)
cols_with_nan = df_beingcleaned.columns[df_beingcleaned.isna().any()].tolist() # No metric
has any Null value.

# Drop NA rows
df_beingcleaned   =   df_beingcleaned.dropna(axis='index',   how='any',   thresh=None,
subset=None, inplace=False)
# df_beingcleaned with 141775 rows, 470 columns.

# Drop columns that have no use from now on
cols_to_drop = [
            'adaccount_currency(t-0)',
            'adaccount_currency(t-1)',
            'adaccount_currency(t-2)',
            'adaccount_currency(t-3)',
            'adaccount_currency(t-4)',
            'adaccount_currency(t-5)',
            'adaccount_currency(t-6)',
            'adaccount_currency(t-7)',
            'adaccount_id',
            'adaccount_name',
            'adaccount_timezone(t-0)',
            'adaccount_timezone(t-1)',
            'adaccount_timezone(t-2)',
            'adaccount_timezone(t-3)',
            'adaccount_timezone(t-4)',
            'adaccount_timezone(t-5)',
            'adaccount_timezone(t-6)',
            'adaccount_timezone(t-7)',
            'adset_billing_event(t-0)',
            'adset_billing_event(t-1)',
            'adset_billing_event(t-2)',
            'adset_billing_event(t-3)',
            'adset_billing_event(t-4)',
            'adset_billing_event(t-5)',
            'adset_billing_event(t-6)',
            'adset_billing_event(t-7)',
            'adset_id',
            'adset_optimization_goal(t-0)',
            'adset_optimization_goal(t-1)',
            'adset_optimization_goal(t-2)',
            'adset_optimization_goal(t-3)',
            'adset_optimization_goal(t-4)',
            'adset_optimization_goal(t-5)',
            'adset_optimization_goal(t-6)',
            'adset_optimization_goal(t-7)',
            'campaign_attribution_window(t-0)',
            'campaign_attribution_window(t-1)',
            'campaign_attribution_window(t-2)',
            'campaign_attribution_window(t-3)',
            'campaign_attribution_window(t-4)',
            'campaign_attribution_window(t-5)',
            'campaign_attribution_window(t-6)',
            'campaign_attribution_window(t-7)',
            'campaign_da_optimizer_status(t-0)',
            'campaign_da_optimizer_status(t-1)',
            'campaign_da_optimizer_status(t-2)',
            'campaign_da_optimizer_status(t-3)',
            'campaign_da_optimizer_status(t-4)',
            'campaign_da_optimizer_status(t-5)',
            'campaign_da_optimizer_status(t-6)',
            'campaign_da_optimizer_status(t-7)',
            'campaign_id',
            'campaign_is_dynamic_ad(t-0)',
            'campaign_is_dynamic_ad(t-1)',
            'campaign_is_dynamic_ad(t-2)',
            'campaign_is_dynamic_ad(t-3)',
            'campaign_is_dynamic_ad(t-4)',
            'campaign_is_dynamic_ad(t-5)',
            'campaign_is_dynamic_ad(t-6)',
            'campaign_is_dynamic_ad(t-7)',
            'campaign_marvin_target_cpa(t-0)',
            'campaign_marvin_target_cpa(t-0)_usd',
            'campaign_marvin_target_cpa(t-1)',
            'campaign_marvin_target_cpa(t-1)_usd',
            'campaign_marvin_target_cpa(t-2)',
            'campaign_marvin_target_cpa(t-2)_usd',
            'campaign_marvin_target_cpa(t-3)',
            'campaign_marvin_target_cpa(t-3)_usd',
            'campaign_marvin_target_cpa(t-4)',
            'campaign_marvin_target_cpa(t-4)_usd',
            'campaign_marvin_target_cpa(t-5)',
            'campaign_marvin_target_cpa(t-5)_usd',
            'campaign_marvin_target_cpa(t-6)',
            'campaign_marvin_target_cpa(t-6)_usd',
            'campaign_marvin_target_cpa(t-7)',
            'campaign_marvin_target_cpa(t-7)_usd',
            'campaign_marvin_target_roas(t-0)',
            'campaign_marvin_target_roas(t-0)_usd',
            'campaign_marvin_target_roas(t-1)',
            'campaign_marvin_target_roas(t-1)_usd',
            'campaign_marvin_target_roas(t-2)',
            'campaign_marvin_target_roas(t-2)_usd',
            'campaign_marvin_target_roas(t-3)',
            'campaign_marvin_target_roas(t-3)_usd',
            'campaign_marvin_target_roas(t-4)',
            'campaign_marvin_target_roas(t-4)_usd',
            'campaign_marvin_target_roas(t-5)',
            'campaign_marvin_target_roas(t-5)_usd',
            'campaign_marvin_target_roas(t-6)',
            'campaign_marvin_target_roas(t-6)_usd',
            'campaign_marvin_target_roas(t-7)',
            'campaign_marvin_target_roas(t-7)_usd',
            'campaign_objective(t-0)',
            'campaign_objective(t-1)',
            'campaign_objective(t-2)',
            'campaign_objective(t-3)',
            'campaign_objective(t-4)',
            'campaign_objective(t-5)',
            'campaign_objective(t-6)',
            'campaign_objective(t-7)',
            'campaign_optimize_revenue(t-0)',
            'campaign_optimize_revenue(t-1)',
            'campaign_optimize_revenue(t-2)',
            'campaign_optimize_revenue(t-3)',
            'campaign_optimize_revenue(t-4)',
            'campaign_optimize_revenue(t-5)',
            'campaign_optimize_revenue(t-6)',
            'campaign_optimize_revenue(t-7)',
            'campaign_promoted_object(t-0)',
            'campaign_promoted_object(t-1)',
            'campaign_promoted_object(t-2)',
            'campaign_promoted_object(t-3)',
            'campaign_promoted_object(t-4)',
            'campaign_promoted_object(t-5)',
            'campaign_promoted_object(t-6)',
            'campaign_promoted_object(t-7)',
            'conversion',
            'date(t-0)',
            'date(t-1)',
            'date(t-2)',
            'date(t-3)',
            'date(t-4)',
            'date(t-5)',
            'date(t-6)',
            'date(t-7)',
            'day',
            'min_date',
            'revenue',
            'targeting_spec(t-0)',
            'targeting_spec(t-1)',
            'targeting_spec(t-2)',
            'targeting_spec(t-3)',
            'targeting_spec(t-4)',
            'targeting_spec(t-5)',
            'targeting_spec(t-6)',
            'targeting_spec(t-7)',
            'usergroup_id',
            'usergroup_name',
            'impressions',
            'clicks',
            'reach',
            'social_clicks',
            'social_impressions',
            'social_reach',
            'total_actions',
            'unique_clicks',
            'unique_social_clicks',
            'rate_0day',
            'rate_30day',

            # Forward looking
            'absorption_r_change(0vs1)',
            'absorption_r(t-0)',
            'ctr(t-0)',
            'frequency(t-0)',
            'social_ctr(t-0)',
            'social_reach_ratio(t-0)',
            'spend/camp_budget(t-0)',
            'ctr_change(0vs1)',
            'frequency_change(0vs1)',
            'social_ctr_change(0vs1)',
            'social_reach_ratio_change(0vs1)',
            'spend_change(0vs1)',
            'spend/camp_budget_change(0vs1)',
            'spend(t-0)_usd',

            # Currency related duplicates
            'bid(t-0)',
            'bid(t-1)',
            'bid(t-2)',
            'bid(t-3)',
            'bid(t-4)',
            'bid(t-5)',
            'bid(t-6)',
            'bid(t-7)',
            'budget(t-0)',
            'budget(t-1)',
            'budget(t-2)',
            'budget(t-3)',
            'budget(t-4)',
            'budget(t-5)',
            'budget(t-6)',
            'budget(t-7)',
            'campaign_budget(t-0)',
            'campaign_budget(t-1)',
            'campaign_budget(t-2)',
            'campaign_budget(t-3)',
            'campaign_budget(t-4)',
            'campaign_budget(t-5)',
            'campaign_budget(t-6)',
            'campaign_budget(t-7)',
            'cp_thousand_reach(t-0)',
            'cp_thousand_reach(t-1)',
            'cp_thousand_reach(t-2)',
            'cp_thousand_reach(t-3)',
            'cp_thousand_reach(t-4)',
            'cp_thousand_reach(t-5)',
            'cp_thousand_reach(t-6)',
            'cp_thousand_reach(t-7)',
            'cpm(t-0)',
            'cpm(t-1)',
            'cpm(t-2)',
            'cpm(t-3)',
            'cpm(t-4)',
            'cpm(t-5)',
            'cpm(t-6)',
            'cpm(t-7)',
            'spend(t-0)',
            'spend(t-1)',
            'spend(t-2)',
            'spend(t-3)',
            'spend(t-4)',
            'spend(t-5)',
            'spend(t-6)',
            'spend(t-7)',
            'weighted_cpm_lastweek',
            ]

df_beingcleaned.drop(columns=cols_to_drop, inplace=True, errors='raise')

# Select & reorder columns
cols_order = [
            # Targets
            'cp_thousand_reach_change(0vs1)',
            'cpm_change(0vs1)',
            'cp_thousand_reach(t-0)_usd',
            'cpm(t-0)_usd',

            # Ratios
            'absorption_r(t-1)',
            'absorption_r(t-2)',
            'absorption_r(t-3)',
            'absorption_r(t-4)',
            'absorption_r(t-5)',
            'absorption_r(t-6)',
            'absorption_r(t-7)',

            'budget/bid(t-0)',
            'budget/bid(t-1)',
            'budget/bid(t-2)',
            'budget/bid(t-3)',
            'budget/bid(t-4)',
            'budget/bid(t-5)',
            'budget/bid(t-6)',
            'budget/bid(t-7)',

            'ctr(t-1)',
            'ctr(t-2)',
            'ctr(t-3)',
            'ctr(t-4)',
            'ctr(t-5)',
            'ctr(t-6)',
            'ctr(t-7)',

            'frequency(t-1)',
            'frequency(t-2)',
            'frequency(t-3)',
            'frequency(t-4)',
            'frequency(t-5)',
            'frequency(t-6)',
            'frequency(t-7)',

            'social_ctr(t-1)',
            'social_ctr(t-2)',
            'social_ctr(t-3)',
            'social_ctr(t-4)',
            'social_ctr(t-5)',
            'social_ctr(t-6)',
            'social_ctr(t-7)',

            'budget/camp_budget(t-0)',
            'budget/camp_budget(t-1)',
            'budget/camp_budget(t-2)',
            'budget/camp_budget(t-3)',
            'budget/camp_budget(t-4)',
            'budget/camp_budget(t-5)',
            'budget/camp_budget(t-6)',
            'budget/camp_budget(t-7)',

            'social_reach_ratio(t-1)',
            'social_reach_ratio(t-2)',
            'social_reach_ratio(t-3)',
            'social_reach_ratio(t-4)',
            'social_reach_ratio(t-5)',
            'social_reach_ratio(t-6)',
            'social_reach_ratio(t-7)',

            'spend/camp_budget(t-1)',
            'spend/camp_budget(t-2)',
            'spend/camp_budget(t-3)',
            'spend/camp_budget(t-4)',
            'spend/camp_budget(t-5)',
            'spend/camp_budget(t-6)',
            'spend/camp_budget(t-7)',

            # Changes
            'absorption_r_change(1vs2)',
            'absorption_r_change(1vs7)',
            'absorption_r_change(2vs3)',
            'absorption_r_change(3vs4)',
            'absorption_r_change(4vs5)',
            'absorption_r_change(5vs6)',
            'absorption_r_change(6vs7)',

            'bid_change(0vs1)',
            'bid_change(1vs2)',
            'bid_change(1vs7)',
            'bid_change(2vs3)',
            'bid_change(3vs4)',
            'bid_change(4vs5)',
            'bid_change(5vs6)',
            'bid_change(6vs7)',

            'bidchange*budgetchange_yesterday',
            'bidchange*spendchange_yesterday',
            'spendchange*budgetchange_yesterday',

            'budget_change(0vs1)',
            'budget_change(1vs2)',
            'budget_change(1vs7)',
            'budget_change(2vs3)',
            'budget_change(3vs4)',
            'budget_change(4vs5)',
            'budget_change(5vs6)',
            'budget_change(6vs7)',

            'budget/bid_change(0vs1)',
            'budget/bid_change(1vs2)',
            'budget/bid_change(1vs7)',
            'budget/bid_change(2vs3)',
            'budget/bid_change(3vs4)',
            'budget/bid_change(4vs5)',
            'budget/bid_change(5vs6)',
            'budget/bid_change(6vs7)',

            'budget/camp_budget_change(0vs1)',
            'budget/camp_budget_change(1vs2)',
            'budget/camp_budget_change(1vs7)',
            'budget/camp_budget_change(2vs3)',
            'budget/camp_budget_change(3vs4)',
            'budget/camp_budget_change(4vs5)',
            'budget/camp_budget_change(5vs6)',
            'budget/camp_budget_change(6vs7)',

            'cp_thousand_reach_change(1vs2)',
            'cp_thousand_reach_change(1vs7)',
            'cp_thousand_reach_change(2vs3)',
            'cp_thousand_reach_change(3vs4)',
            'cp_thousand_reach_change(4vs5)',
            'cp_thousand_reach_change(5vs6)',
            'cp_thousand_reach_change(6vs7)',

            'cpm_change(1vs2)',
            'cpm_change(1vs7)',
            'cpm_change(2vs3)',
            'cpm_change(3vs4)',
            'cpm_change(4vs5)',
            'cpm_change(5vs6)',
            'cpm_change(6vs7)',

            'ctr_change(1vs2)',
            'ctr_change(1vs7)',
            'ctr_change(2vs3)',
            'ctr_change(3vs4)',
            'ctr_change(4vs5)',
            'ctr_change(5vs6)',
            'ctr_change(6vs7)',

            'frequency_change(1vs2)',
            'frequency_change(1vs7)',
            'frequency_change(2vs3)',
            'frequency_change(3vs4)',
            'frequency_change(4vs5)',
            'frequency_change(5vs6)',
            'frequency_change(6vs7)',

            'social_ctr_change(1vs2)',
            'social_ctr_change(1vs7)',
            'social_ctr_change(2vs3)',
            'social_ctr_change(3vs4)',
            'social_ctr_change(4vs5)',
            'social_ctr_change(5vs6)',
            'social_ctr_change(6vs7)',

            'social_reach_ratio_change(1vs2)',
            'social_reach_ratio_change(1vs7)',
            'social_reach_ratio_change(2vs3)',
            'social_reach_ratio_change(3vs4)',
            'social_reach_ratio_change(4vs5)',
            'social_reach_ratio_change(5vs6)',
            'social_reach_ratio_change(6vs7)',

            'spend_change(1vs2)',
            'spend_change(1vs7)',
            'spend_change(2vs3)',
            'spend_change(3vs4)',
            'spend_change(4vs5)',
            'spend_change(5vs6)',
            'spend_change(6vs7)',

            'spend/camp_budget_change(1vs2)',
            'spend/camp_budget_change(1vs7)',
            'spend/camp_budget_change(2vs3)',
            'spend/camp_budget_change(3vs4)',
            'spend/camp_budget_change(4vs5)',
            'spend/camp_budget_change(5vs6)',
            'spend/camp_budget_change(6vs7)',

            # Campaign Structure
            'is_attr_1dc',
            'is_attr_1dv',
            'is_attr_28dc',
            'is_attr_7dc',
            'is_attr_7dv',
            'is_attr_default',
            'is_billing_clicks',
            'is_billing_impressions',
            'is_billing_likes',
            'is_billing_mai',
            'is_billing_video',
            'is_campobjective_appinstall',
            'is_campobjective_click',
            'is_campobjective_conversion',
            'is_campobjective_leadgeneration',
            'is_campobjective_mobileappinstall',
            'is_campobjective_pagelike',
            'is_campobjective_pcs',
            'is_campobjective_postengagement',
            'is_campobjective_reach',
            'is_campobjective_videoview',
            'is_dao',
            'is_dynamicad',
            'is_optgoal_appinstall',
            'is_optgoal_click',
            'is_optgoal_impression',
            'is_optgoal_leadgeneration',
            'is_optgoal_offsiteconversion',
            'is_optgoal_pagelike',
            'is_optgoal_postengagement',
            'is_optgoal_reach',
            'is_optgoal_videoview',
            'is_promotedobject_application',
            'is_promotedobject_catalog',
            'is_promotedobject_customeventtype',
            'is_promotedobject_objectstoreurl',
            'is_promotedobject_page',
            'is_promotedobject_pixel',
            'is_revenue',
            'is_target',
            'is_tz_america',
            'is_tz_asia',
            'is_tz_atlantic',
            'is_tz_australia',
            'is_tz_europe',

            # Currency Related
            'bid(t-0)_usd',
            'bid(t-1)_usd',
            'bid(t-2)_usd',
            'bid(t-3)_usd',
            'bid(t-4)_usd',
            'bid(t-5)_usd',
            'bid(t-6)_usd',
            'bid(t-7)_usd',

            'budget(t-0)_usd',
            'budget(t-1)_usd',
            'budget(t-2)_usd',
            'budget(t-3)_usd',
            'budget(t-4)_usd',
            'budget(t-5)_usd',
            'budget(t-6)_usd',
            'budget(t-7)_usd',
            'campaign_budget(t-0)_usd',
            'campaign_budget(t-1)_usd',
            'campaign_budget(t-2)_usd',
            'campaign_budget(t-3)_usd',
            'campaign_budget(t-4)_usd',
            'campaign_budget(t-5)_usd',
            'campaign_budget(t-6)_usd',
            'campaign_budget(t-7)_usd',

            'cp_thousand_reach(t-1)_usd',
            'cp_thousand_reach(t-2)_usd',
            'cp_thousand_reach(t-3)_usd',
            'cp_thousand_reach(t-4)_usd',
            'cp_thousand_reach(t-5)_usd',
            'cp_thousand_reach(t-6)_usd',
            'cp_thousand_reach(t-7)_usd',

            'cpm(t-1)_usd',
            'cpm(t-2)_usd',
            'cpm(t-3)_usd',
            'cpm(t-4)_usd',
            'cpm(t-5)_usd',
            'cpm(t-6)_usd',
            'cpm(t-7)_usd',

            'spend(t-1)_usd',
            'spend(t-2)_usd',
            'spend(t-3)_usd',
            'spend(t-4)_usd',
            'spend(t-5)_usd',
            'spend(t-6)_usd',
            'spend(t-7)_usd',

            'weighted_cpm_lastweek_usd',

            # Other
            'adset_run_time',
            'campaign_spend_cap',
            'day_of_week',
            'day_of_year',
            'month',
            'quarter',
            'rate_7day',
        ]

df_beingcleaned = df_beingcleaned[cols_order]

# Save dataframe to disk for future use
df_beingcleaned.to_pickle("df/df.gzip", compression = 'gzip')
del df_beingcleaned

# End timer
end_time = time.time()

# Calculate execution time
print("--- %s seconds ---" % (end_time - start_time))
```

clustering.py:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Aug  2 01:57:14 2018

@author: tektensemih
"""

####################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES
####################################################################
########      CLUSTERING        #################################
####################################################################

# import libraries
import pandas as pd
import numpy as np
import glob, os
import time

# Scikit Learn Libraries
from sklearn.cluster import KMeans

# Start timer
start_time = time.time()

# Change path
path = os.path.expanduser('~/Documents/Scripts/CPM/Capstone/')
os.chdir(path)

df = pd.read_pickle('df/df.gzip', compression='gzip')
# df with 141775 rows, 258 columns

# Check if NaN exists
df = df.dropna(axis='index', how='any', thresh=None, subset=None, inplace=False) # No NaN
```

```
# Find campaign structure features
str_features = list(df.filter(like='is_').columns)

# K-Means Clustering
seed = 674

for n in range(1,20):
        kmeans = KMeans(random_state=seed, n_init=20, max_iter=500, n_jobs=-1,
n_clusters=n+1)
  kmeans = kmeans.fit(df[str_features])
  cluster_name = 'adset_seg_' + str(n+1)
  df[cluster_name] = kmeans.predict(df[str_features])
  print('Finished for the ' + str(n+1) + 'th algorithm.')
del n

# Save dataframe to disk for future use
df.to_pickle("df/df_clustered.gzip", compression = 'gzip')
del df

# End timer
end_time = time.time()

# Calculate execution time
print("--- %s seconds ---" % (end_time - start_time))
```

modelling.py:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 22 14:22:01 2018

@author: tektensemih
"""


######################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES
######################################################################
######## MODELLING ###################################################
######################################################################

# import libraries
import pandas as pd
import numpy as np
import glob, os
import time
from datetime import datetime, timedelta
# import matplotlib.pyplot as plt
# import seaborn as sns
from math import sqrt
from scipy import stats

# Scikit Learn Libraries
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ARDRegression
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import LassoLars
from sklearn.linear_model import Lars
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn import decomposition

# Start timer
start_time = time.time()

# Change path
# path = os.path.expanduser('~/Documents/Scripts/CPM/Capstone/')
# os.chdir(path)

df = pd.read_pickle('df/df_clustered.gzip', compression='gzip')
# df with 141775 rows, 277 columns

# Check if NaN exists
df = df.dropna(axis='index', how='any', thresh=None, subset=None, inplace=False) # No NaN

# Log transformation of CPM Change
df['cpm_change(0vs1)_log'] = np.log(df['cpm_change(0vs1)']+1)
df['cp_thousand_reach_change(0vs1)_log']                                        =
np.log(df['cp_thousand_reach_change(0vs1)']+1)

# Put Log columns to beginning
cols = df.columns.tolist()
cols.insert(2, cols.pop(cols.index('cpm_change(0vs1)_log')))
cols.insert(1, cols.pop(cols.index('cp_thousand_reach_change(0vs1)_log')))
df = df.reindex(columns = cols)

# Correlation Matrix
corr_matrix = df.corr()
corr_matrix_abs = df.corr().abs()

# Visualize Correlation Matrix
```

```
plt.matshow(corr_matrix_abs)

# Visualize CPM Change Values, if no metric has significant change
threshold = 0.25
no_sig_change = df[ (df['absorption_r_change(1vs2)'].abs() <= threshold) &
                (df['bid_change(0vs1)'].abs() <= threshold) &
                (df['bid_change(1vs2)'].abs() <= threshold) &
                (df['budget_change(0vs1)'].abs() <= threshold) &
                (df['budget_change(1vs2)'].abs() <= threshold) &
                (df['ctr_change(1vs2)'].abs() <= threshold) &
                (df['spend_change(1vs2)'].abs() <= threshold)
                ]

no_sig_change          =          no_sig_change['cpm_change(0vs1)'].sort_values(axis=0,
ascending=True).reset_index()
no_sig_change = no_sig_change['cpm_change(0vs1)']
drop_5 = int(no_sig_change.shape[0] * 0.025)
no_sig_change = no_sig_change.iloc[drop_5:]
no_sig_change = no_sig_change.head(no_sig_change.shape[0]-drop_5)
no_sig_change.hist()
del no_sig_change, drop_5, threshold

################################################################################
####
# What is CPM change range distribution, when there is no significant change?
# Without any significant changes, CPM change value ranges from -0.3 to 0.4.
################################################################################
####


############################################################
############################################################
######     REGRESSION                    ######
############################################################
############################################################

seed = 674

# Regression Function

def adjusted_rsqu(rs,n,p):
   # rs: r-squared
```

```
    # n: number of observations
    # p: number of independent variables
    print('######################')
    print('R_Squared: ' + str(rs))
    print('Observations: ' + str(n))
    print('Variables: ' + str(p))
    adjusted_r = 1-(1-rs)*(n-1)/(n-p-1)
    print('Adjusted R: ' + str(adjusted_r))
    print('######################')
    return adjusted_r

def regression(features, target, algorithm, params, seed, description):

    # K-Fold splits
    kf_splits = 10
    kf = KFold(n_splits=kf_splits, shuffle=False, random_state=seed).split(features)

    score_train_list = []
    score_test_list = []
    rmse_train_list = []
    rmse_test_list = []
    prediction_list = []
    actual_list = []

    for train_indices, test_indices in kf:
        X_train = features.iloc[train_indices]
        X_test  = features.iloc[test_indices]
        y_train = target.iloc[train_indices]
        y_test  = target.iloc[test_indices]

        reg = GridSearchCV(algorithm, params,scoring='r2')
        reg = reg.fit(X_train, y_train)

        trainy_predict = reg.predict(X_train)
        testy_predict = reg.predict(X_test)

                                                        score_train_list.append(adjusted_rsqu(r2_score(y_train,
trainy_predict),trainy_predict.size,len(features.columns)))
                                                            score_test_list.append(adjusted_rsqu(r2_score(y_test,
testy_predict),testy_predict.size,len(features.columns)))

        rmse_train_list.append(sqrt(mean_squared_error(y_train, trainy_predict)))
        rmse_test_list.append(sqrt(mean_squared_error(y_test, testy_predict)))

        prediction_list.append(testy_predict)
        actual_list.append(y_test)

    to_return = {'target':target.name,
                 'avg_train_r2':np.mean(score_train_list),
                 'avg_test_r2':np.mean(score_test_list),
                 'avg_train_rmse':np.mean(rmse_train_list),
                 'avg_test_rmse':np.mean(rmse_test_list),
                 'predictions':np.concatenate(prediction_list, axis=0),
                 'actuals':np.concatenate(actual_list, axis=0),
                 'algorithm':algorithm,
                 'params':reg.best_params_,
                 'description':description,
                 'seed':seed
                 }
    return to_return

def modelling(features, target, description, seed):
    models = [
            # Linear Regression Algorithms

            # Linear Model
            [LinearRegression(),
            {'n_jobs':[-1]}],

            # Ridge Model
            #[Ridge(random_state=seed),
            # {'alpha':[1.0]}],

            # Lasso Model
            #[Lasso(random_state=seed),
            # {alpha:[1.0]}],

            # Bayesian Ridge Model
            #[BayesianRidge(),
            # {}],

            # Lasso Lars Model
            #[LassoLars(),
            # {alpha:[0.01]}],
            # Lars Model
            #[Lars(),
            # {n_nonzero_coefs:[1]}],

            # Tree Regression Algorithms

            #[RandomForestRegressor(n_estimators = 50, max_features = 13, \
            #               max_depth = 10, min_samples_split = 5, \
            #               n_jobs = -1, min_samples_leaf = 20, \
            #               oob_score = True, random_state = seed),
            #  {},
            #  ],
            ]

        # Decision Tree Regression
        # Various parameters

    func_results = []
    for model in models:
        func_results.append(regression(features,target,model[0],model[1],seed,description))
    return func_results



# Targets & Features

features    = df[df.columns.difference(['cp_thousand_reach(t-0)_usd',
                        'cp_thousand_reach_change(0vs1)',
                        'cp_thousand_reach_change(0vs1)_log',
                        'cpm(t-0)_usd',
                        'cpm_change(0vs1)',
                        'cpm_change(0vs1)_log'
                        ])]

targets     = df[['cp_thousand_reach(t-0)_usd',
              'cp_thousand_reach_change(0vs1)',
              'cp_thousand_reach_change(0vs1)_log',
              'cpm(t-0)_usd',
              'cpm_change(0vs1)',
              'cpm_change(0vs1)_log'
              ]]
```

```
features_cols = list(features.columns)
targets_cols = list(targets.columns)

# Cases

df_backup = df.copy()

default_result = modelling(df[features_cols], df['cpm_change(0vs1)_log'], 'Default result',
seed)

"""
Scores of default dataset:

max_features: 14
Average Train RMSE: 0.22882540779754917
Average Train R2: 0.2948863064898627
Average Test RMSE: 0.23390752740619963
Average Test R2: 0.1967944827157384

max_features: 25
Average Train RMSE: 0.22293444038854077
Average Train R2: 0.3307816881070299
Average Test RMSE: 0.2294464401721604
Average Test R2: 0.22679395887852385

max_features: 40
Average Train RMSE: 0.21978561091089116
Average Train R2: 0.3495413666074792
Average Test RMSE: 0.2273921648766361
Average Test R2: 0.24095141754046895

max_features: 80
Average Train RMSE: 0.21735614876533188
Average Train R2: 0.3638359850458821
Average Test RMSE: 0.22585352408859322
Average Test R2: 0.25135911955940393


Review:
We'll continue with '14' features, considering the accuracy & computational speed.
"""
```

```
#####################################################################################
# Choose rows

# Case 1: Significant changes
# Option 1: Use no case with significant change, 'change' threshold: +/- 0.10 (Exclude CPM, CPTR change)
# Option 2: Use no case with significant change, 'change' threshold: +/- 0.20 (Exclude CPM, CPTR change)
# Option 3: Use no case with significant change, 'change' threshold: +/- 0.30 (Exclude CPM, CPTR change)
# Option 4: Use no case with significant change, 'change' threshold: +/- 0.40 (Exclude CPM, CPTR change)
# Option 5: Use no case with significant change, 'change' threshold: +/- 0.50 (Exclude CPM, CPTR change)
# Option 6: Use no case with significant change, 'change' threshold: +/- 0.60 (Exclude CPM, CPTR change)
# Option 7: Use no case with significant change, 'change' threshold: +/- 0.70 (Exclude CPM, CPTR change)
# Option 8: Use no case with significant change, 'change' threshold: +/- 0.80 (Exclude CPM, CPTR change)
# Option 9: Use all cases

results = []
df_case = df.copy()

thresholds = [0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.8]

for threshold in thresholds:
    df_case = df_case[ (df_case['absorption_r_change(1vs2)'].abs() <= threshold) &
                    (df_case['bid_change(0vs1)'].abs() <= threshold) &
                    (df_case['bid_change(1vs2)'].abs() <= threshold) &
                    (df_case['budget_change(0vs1)'].abs() <= threshold) &
                    (df_case['budget_change(1vs2)'].abs() <= threshold) &
                    (df_case['ctr_change(1vs2)'].abs() <= threshold) &
                    (df_case['spend_change(1vs2)'].abs() <= threshold)
                    ]

        results.append(modelling(df_case[features_cols], df_case['cpm_change(0vs1)_log'], 'Significant change test', seed))
    df_case = df.copy()

print 'Significant Change Test Results for target column CPM Change 0vs1 Log, Seed: 674:'
for i in range(len(results)):
    print 'For threshold: ' + str(thresholds[i])
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
Significant Change Test Results for target column CPM Change 0vs1 Log, Seed: 674:

For threshold: 0.1
Average Train RMSE: 0.12440986218790287
Average Train R2: 0.19008153087560836
Average Test RMSE: 0.13267132123307132
Average Test R2: -0.2968232613575119

For threshold: 0.2
Average Train RMSE: 0.14456060134961174
Average Train R2: 0.19322744801623606
Average Test RMSE: 0.15164783582686397
Average Test R2: 0.00690866738794278

For threshold: 0.3
Average Train RMSE: 0.16477676008243541
Average Train R2: 0.19020627869602383
Average Test RMSE: 0.17062932333323955
Average Test R2: 0.054137263786468215

For threshold: 0.4
Average Train RMSE: 0.17693001112861098
Average Train R2: 0.1960034472147269
Average Test RMSE: 0.18213284169914284
Average Test R2: 0.08300376954184555

For threshold: 0.5
Average Train RMSE: 0.18901330578140244
Average Train R2: 0.20634503386943423
Average Test RMSE: 0.19447148810831702
Average Test R2: 0.0990436607379319

For threshold: 0.6
Average Train RMSE: 0.19695027525820769
Average Train R2: 0.22042913861287278
Average Test RMSE: 0.20240130791173022
Average Test R2: 0.11649305400315746

For threshold: 0.7
Average Train RMSE: 0.2027384362601079
Average Train R2: 0.23948828371404987
Average Test RMSE: 0.2081901065883122
Average Test R2: 0.13557409595423225

For threshold: 0.8
Average Train RMSE: 0.20807937907528457
Average Train R2: 0.26603164840403953
Average Test RMSE: 0.21350303154579692
Average Test R2: 0.15874604075714846

Review:

When threshold for significant change increases, both average train & test R2 increases.
When threshold for significant change increases, RMSE also increases.
All thresholds bring less RMSE, but their R2 is also less than default.
No filtering for significant change will be used.

"""

del df_case, results


# Case 2: Adset run time
# Option 1: Adset run time above 29
# Option 2: Adset run time above 19
# Option 3: Adset run time above 13
# Option 4: Adset run time above 9
# Option 5: All cases

results = []
df_case = df.copy()
```

```
thresholds = [0,9,13,19,29]

for threshold in thresholds:
    df_case = df_case[ (df_case['adset_run_time'] >= threshold)]

     results.append(modelling(df_case[features_cols], df_case['cpm_change(0vs1)_log'], 'Adset
run time test', seed))

    df_case = df.copy()

print 'Adset Run Time Test Results for target column CPM Change 0vs1 Log, Seed: 674:'
for i in range(len(results)):
    print 'For threshold: ' + str(thresholds[i])
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
Adset Run Time Test Results for target column CPM Change 0vs1 Log, Seed: 674:

For threshold: 0
Average Train RMSE: 0.22882540779754917
Average Train R2: 0.2948863064898627
Average Test RMSE: 0.23390752740619963
Average Test R2: 0.1967944827157384

For threshold: 9
Average Train RMSE: 0.228948515586391
Average Train R2: 0.29826357309878115
Average Test RMSE: 0.23420119726105018
Average Test R2: 0.20030548457207967

For threshold: 13
Average Train RMSE: 0.23082139665679433
Average Train R2: 0.2998692677621351
Average Test RMSE: 0.23611052961041729
Average Test R2: 0.1994730390322385

For threshold: 19
Average Train RMSE: 0.23282327698686242
Average Train R2: 0.3047183690890668
Average Test RMSE: 0.23772238440339297
Average Test R2: 0.19735846130469784

For threshold: 29
Average Train RMSE: 0.2369909709563236
Average Train R2: 0.3079032611346693
Average Test RMSE: 0.2427339254985604
Average Test R2: 0.20107366361491721


Review:

Adset run time has little/no impact on both Train&Test RMSE & R2.
I decided not to include adset run time filter in my model.

"""

del df_case, results


# Case 3: Conversions wanted from Facebook
# Option 1: Use conversions wanted (Adset Budget / Adset Bid) threshold > 4
# Option 2: Use conversions wanted (Adset Budget / Adset Bid) threshold > 9
# Option 3: Use conversions wanted (Adset Budget / Adset Bid) threshold > 14
# Option 4: Use conversions wanted (Adset Budget / Adset Bid) threshold > 19
# Option 5: Use conversions wanted (Adset Budget / Adset Bid) threshold > 49
# Option 6: Use no conversions wanted information

results = []
df_case = df.copy()

thresholds = [5,10,15,20,50]

for threshold in thresholds:
    threshold = 50
    df_case = df_case[ (df_case['budget/bid(t-0)'] >= threshold)]

        results.append(modelling(df_case[features_cols],    df_case['cpm_change(0vs1)_log'],
'Conversions wanted test', seed))

    df_case = df.copy()
```

```
print 'Conversions Wanted Test Results for target column CPM Change 0vs1 Log, Seed: 674:'
for i in range(len(results)):
    print 'For threshold: ' + str(thresholds[i])
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])


"""
Conversions Wanted Test Results for target column CPM Change 0vs1 Log, Seed: 674:

For threshold: 5
Average Train RMSE: 0.24623905258377898
Average Train R2: 0.3225085117333603
Average Test RMSE: 0.250765542004317
Average Test R2: 0.2068910103153972

For threshold: 10
Average Train RMSE: 0.25130971895485205
Average Train R2: 0.3274162598713405
Average Test RMSE: 0.256995101666384
Average Test R2: 0.2088076662547167

For threshold: 15
Average Train RMSE: 0.2542120836477324
Average Train R2: 0.3320525587219921
Average Test RMSE: 0.2599277854345284
Average Test R2: 0.2122799399279673

For threshold: 20
Average Train RMSE: 0.2563430083621602
Average Train R2: 0.3339301254976247
Average Test RMSE: 0.2625540990480219
Average Test R2: 0.21095520701367892

For threshold: 50
Average Train RMSE: 0.26473659839536057
Average Train R2: 0.34614303037504357
Average Test RMSE: 0.2719576150837061
Average Test R2: 0.2191745489084341
```

```
Review:

Conversions wanted has little impact on Test R2.
If I want to use this filter, I need to drop 75K of observations, which is half of the cases, for
%1 increase in R2.
For that reason, I decided not to use Conversions wanted for filtering.

"""

del df_case, results


# Case 4: Unbiased Spend Levels
# Option 1: Calculate spend change levels with 0.1 and drop random rows in order to have
balanced spend change levels
# Option 2: Use all spend levels

results = []
df_case = df.copy()

df_case = df_case[df_case['spend_change(1vs2)'] < 1.0]

df_case['spend_change_level'] = pd.cut(df_case['spend_change(1vs2)'],
                                       bins=[-1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1,
                                             0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])

df_case['spend_change_level'].value_counts()

"""
(-0.1, 0.0]     22013
(0.0, 0.1]      20007
(-0.2, -0.1]    13268
(0.1, 0.2]      10726
(-0.3, -0.2]     9762
(-0.4, -0.3]     7856
(0.2, 0.3]       6900
(-0.5, -0.4]     5768
(0.3, 0.4]       5175
(-0.6, -0.5]     4560
(0.4, 0.5]       4127
(-0.7, -0.6]     3223
(0.5, 0.6]      3084
(0.6, 0.7]      2535
(-0.8, -0.7]     2194
(0.7, 0.8]      2146
(0.8, 0.9]      1807
(0.9, 1.0]      1429
(-0.9, -0.8]     1377
(-1.0, -0.9]      650

Action points:
1. Drop rows until each category has 650 or less observations
2. Drop rows until each category has 2000 or less observations
3. Drop rows until each category has 3000 or less observations
4. Drop rows until each category has 6500 or less observations
5. Drop rows until each category has 9000 or less observations
"""

unique_changelevel_l = list(df_case['spend_change_level'].unique())
thresholds = [650,2000,3000,6500,9000]
df_case_backup = df_case.copy()

for threshold in thresholds:
    for level in unique_changelevel_l:
        drop_count = df_case[df_case['spend_change_level'] == level].shape[0] - threshold
        if drop_count <= 0:
            continue
                                                df_case.drop(df_case[df_case['spend_change_level']      ==
level].sample(n=drop_count).index, inplace=True)

            results.append(modelling(df_case[features_cols],   df_case['cpm_change(0vs1)_log'],
'Unbiased spend level test', seed))
    df_case = df_case_backup.copy()

print 'Unbiased Spend Level Test Results for target column CPM Change 0vs1 Log, Seed:
674:'
for i in range(len(results)):
    print 'For threshold: ' + str(thresholds[i])
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])
```

```
"""
Unbiased Spend Level Test Results for target column CPM Change 0vs1 Log, Seed: 674:

For threshold: 650
Average Train RMSE: 0.2588506276101789
Average Train R2: 0.42591004402587557
Average Test RMSE: 0.2780724217405783
Average Test R2: 0.10580988671788519

For threshold: 2000
Average Train RMSE: 0.25253197474243166
Average Train R2: 0.3942124510349339
Average Test RMSE: 0.265273190140986
Average Test R2: 0.21743982321098243

For threshold: 3000
Average Train RMSE: 0.24561012309752317
Average Train R2: 0.38017422569007275
Average Test RMSE: 0.25706558601312507
Average Test R2: 0.2262279778074649

For threshold: 6500
Average Train RMSE: 0.2375299090755974
Average Train R2: 0.3340856655127354
Average Test RMSE: 0.2458517935346752
Average Test R2: 0.21004661113794731

For threshold: 9000
Average Train RMSE: 0.23251609275486912
Average Train R2: 0.32136358282631583
Average Test RMSE: 0.23969571172477688
Average Test R2: 0.20689190349952927

Review:

Converting our dataframe into an unbiased dataframe in terms of spend change(1vs2) has
little/no impact on Test R2 & RMSE.
In our classification model in BDA 502 Course, spend has an impact on CPM change.
In our case, we can't use Spend change (0vs1) since it's a forward looking feature.
For that reason, I will also try balanced bid change (0vs1) in the next section.
I won't use balanced Spend change in my model.
"""
```

```
del unique_changelevel_l, thresholds, df_case_backup, results, df_case


# Case 5: Unbiased Bid Change Levels
# Option 1: Calculate bid change levels with 0.1 and drop random rows in order to have
balanced spend change levels
# Option 2: Use all bid change levels

results = []
df_case = df.copy()

df_case['bid_change_level'] = pd.cut(df_case['bid_change(0vs1)'],
                                bins=[-np.inf, -1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1,
                                    0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, np.inf])

df_case['bid_change_level'].value_counts()

"""
(-0.1, 0.0]     116746
(-0.2, -0.1]      6320
(0.1, 0.2]        6026
(0.0, 0.1]        4908
(0.2, 0.3]        3874
(0.3, 0.4]        1496
(-0.3, -0.2]      1290
(-0.4, -0.3]       501
(0.4, 0.5]         193
(-0.5, -0.4]       192
(-0.6, -0.5]        56
(0.5, 0.6]          39
(0.6, 0.7]          36
(1.0, inf]          27
(-0.7, -0.6]        20
(0.7, 0.8]          12
(0.8, 0.9]          10
(-0.8, -0.7]        10
(0.9, 1.0]           7
(-0.9, -0.8]         7
(-1.0, -0.9]         5
(-inf, -1.0]         0
There is a clear bias towards (-0.1, 0.0].
"""

unique_changelevel_l = list(df_case['bid_change_level'].unique())
thresholds = [6500, 3000]
df_case_backup = df_case.copy()

for threshold in thresholds:
    for level in unique_changelevel_l:
        drop_count = df_case[df_case['bid_change_level'] == level].shape[0] - threshold
        if drop_count <= 0:
            continue
                                                    df_case.drop(df_case[df_case['bid_change_level'] ==
level].sample(n=drop_count).index, inplace=True)

            results.append(modelling(df_case[features_cols], df_case['cpm_change(0vs1)_log'],
'Unbiased bid change level test', seed))
    df_case = df_case_backup.copy()

print 'Unbiased Bid Change Level Test Results for target column CPM Change 0vs1 Log,
Seed: 674:'
for i in range(len(results)):
    print 'For threshold: ' + str(thresholds[i])
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
Unbiased Bid Change Level Test Results for target column CPM Change 0vs1 Log, Seed:
674:

For threshold: 6500
Average Train RMSE: 0.22941100676132428
Average Train R2: 0.350330991365721
Average Test RMSE: 0.2452606765517483
Average Test R2: 0.18181726318724836

For threshold: 3000
Average Train RMSE: 0.23357002943467015
Average Train R2: 0.369453546461928
Average Test RMSE: 0.25344862281564573
Average Test R2: 0.13463327065091346

Review:

It seems in parallel to spend change filtering, bid change filtering also has no impact on CPM change.
I won't use bid change filtering in my model.

"""

del unique_changelevel_l, thresholds, df_case_backup, results, df_case



################################################################################
# Choose columns

# Case 1: Campaign structure features
# Option 1: Use campaign structure dummy features, starting with '_is', without best segmentation
# Option 2: Use campaign structure dummy features, starting with '_is', with best segmentation
# Option 3: Don't use campaign structure dummy features, starting with '_is', without best segmentation
# Option 4: Don't use campaign structure dummy features, starting with '_is', with best segmentation

results = []
df_case = df.copy()

all_features = features_cols
str_features = [col for col in df_case if col.startswith('is_')]
seg_features = [col for col in df_case if col.startswith('adset_seg')]

corr_matrix_abs = df_case.corr().abs()
corr_matrix_abs = corr_matrix_abs[['cpm_change(0vs1)_log']]
corr_matrix_abs = corr_matrix_abs.reset_index()
corr_matrix_abs =
corr_matrix_abs[corr_matrix_abs['index'].str.contains('adset_seg')].sort_values(by='cpm_change(0vs1)_log', ascending=False)
# Adset Segmentation with 10 Clusters is the correlated segmentation feature.
# Option 1
all_features_w_str_wout_segment = list(set(all_features) - set(seg_features))

# Option 2
all_features_w_str_w_segment = list(set(all_features) - set(seg_features))
all_features_w_str_w_segment.append('adset_seg_10')

# Option 3
all_features_wout_str_wout_segment = list(set(all_features) - set(seg_features))
all_features_wout_str_wout_segment = list(set(all_features_wout_str_wout_segment) -
set(str_features))

# Option 4
all_features_wout_str_w_segment = list(set(all_features) - set(seg_features))
all_features_wout_str_w_segment = list(set(all_features_wout_str_w_segment) -
set(str_features))
all_features_wout_str_w_segment.append('adset_seg_10')

results.append(modelling(df_case[all_features_w_str_wout_segment],
                    df_case['cpm_change(0vs1)_log'],
                    'all_features_w_str_wout_segment',
                    seed))

results.append(modelling(df_case[all_features_w_str_w_segment],
                    df_case['cpm_change(0vs1)_log'],
                    'all_features_w_str_w_segment',
                    seed))

results.append(modelling(df_case[all_features_wout_str_wout_segment],
                    df_case['cpm_change(0vs1)_log'],
                    'all_features_wout_str_wout_segment',
                    seed))

results.append(modelling(df_case[all_features_wout_str_w_segment],
                    df_case['cpm_change(0vs1)_log'],
                    'all_features_wout_str_w_segment',
                    seed))

print 'Campaign Structure & Segment Information Test Results for target column CPM
Change 0vs1 Log, Seed: 674:'
for i in range(len(results)):
   print 'For option: ' + str(i+1)
```

```
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
Campaign Structure & Segment Information Test Results for target column CPM Change
0vs1 Log, Seed: 674:

Option 1: Use campaign structure dummy features, starting with '_is', without best
segmentation
Average Train RMSE: 0.22795171923374574
Average Train R2: 0.30043021439686696
Average Test RMSE: 0.23309810847575946
Average Test R2: 0.203221359084972

Option 2: Use campaign structure dummy features, starting with '_is', with best segmentation
Average Train RMSE: 0.2280545942271978
Average Train R2: 0.29982341293029563
Average Test RMSE: 0.23322790153874137
Average Test R2: 0.2021590791937312

Option 3: Don't use campaign structure dummy features, starting with '_is', without best
segmentation
Average Train RMSE: 0.22694318920666073
Average Train R2: 0.30681967024179063
Average Test RMSE: 0.23288077888245096
Average Test R2: 0.20874681207043916

Option 4: Don't use campaign structure dummy features, starting with '_is', with best
segmentation
Average Train RMSE: 0.22690727460534613
Average Train R2: 0.30705444030414153
Average Test RMSE: 0.23249544980148462
Average Test R2: 0.21107646711490896

Review:

All Train RMSE & R2 are very similar.
All Train RMSE & R2 are very similar.
There is 1 point increase in Test R2 with Option4.
Option 4 suggests not to use campaign structure features, but to use best segmentation feature.
```

```
By not using campaign structure we decrease complexity in our model and also increase the
accuracy by 1 point.
We will proceed with Option 4: all_features_wout_str_w_segment

"""

del all_features, str_features, seg_features, corr_matrix_abs, results, df_case
del all_features_w_str_w_segment, all_features_w_str_wout_segment

best_features = all_features_wout_str_w_segment[:]
del all_features_wout_str_w_segment, all_features_wout_str_wout_segment







# Case 2: PCA
# Option 1: Use PCA, if better than best columns from Case 1, n_components = 3
# Option 2: Use PCA, if better than best columns from Case 1, n_components = 5
# Option 3: Use PCA, if better than best columns from Case 1, n_components = 10
# Option 4: Use PCA, if better than best columns from Case 1, n_components = 20
# Option 5: Don't use PCA, instead use best columns from Case 1

results = []

# Option 1: PCA, n_components = 3
df_case = df.copy()
pca = decomposition.PCA(n_components=3)
pca.fit(df_case[best_features])
X = pca.transform(df_case[best_features])
df_case_pca = pd.DataFrame(X)
df_case_pca['cpm_change(0vs1)_log'] = df_case['cpm_change(0vs1)_log'].reset_index(drop =
True).copy()

results.append(modelling(df_case_pca.drop(['cpm_change(0vs1)_log'], axis=1),
                    df_case_pca['cpm_change(0vs1)_log'],
                    'PCA, n_components=3',
                    seed))
del X, df_case_pca, df_case, pca
```

```
# Option 2: PCA, n_components = 5
df_case = df.copy()
pca = decomposition.PCA(n_components=5)
pca.fit(df_case[best_features])
X = pca.transform(df_case[best_features])
df_case_pca = pd.DataFrame(X)
df_case_pca['cpm_change(0vs1)_log'] = df_case['cpm_change(0vs1)_log'].reset_index(drop =
True).copy()

results.append(modelling(df_case_pca.drop(['cpm_change(0vs1)_log'], axis=1),
                         df_case_pca['cpm_change(0vs1)_log'],
                         'PCA, n_components=5',
                         seed))
del X, df_case_pca, df_case, pca


# Option 3: PCA, n_components = 10
df_case = df.copy()
pca = decomposition.PCA(n_components=10)
pca.fit(df_case[best_features])
X = pca.transform(df_case[best_features])
df_case_pca = pd.DataFrame(X)
df_case_pca['cpm_change(0vs1)_log'] = df_case['cpm_change(0vs1)_log'].reset_index(drop =
True).copy()

results.append(modelling(df_case_pca.drop(['cpm_change(0vs1)_log'], axis=1),
                         df_case_pca['cpm_change(0vs1)_log'],
                         'PCA, n_components=10',
                         seed))
del X, df_case_pca, df_case, pca


# Option 4: PCA, n_components = 20
df_case = df.copy()
pca = decomposition.PCA(n_components=20)
pca.fit(df_case[best_features])
X = pca.transform(df_case[best_features])
df_case_pca = pd.DataFrame(X)
df_case_pca['cpm_change(0vs1)_log'] = df_case['cpm_change(0vs1)_log'].reset_index(drop =
True).copy()

results.append(modelling(df_case_pca.drop(['cpm_change(0vs1)_log'], axis=1),
                    df_case_pca['cpm_change(0vs1)_log'],
                    'PCA, n_components=20',
                    seed))
del X, df_case_pca, df_case, pca


# Option 5: No PCA, instead best_features
df_case = df.copy()
results.append(modelling(df_case[best_features],
                    df_case['cpm_change(0vs1)_log'],
                    'Best Features, without PCA',
                    seed))
del df_case


print 'PCA Test Results for target column CPM Change 0vs1 Log, Seed: 674:'
for i in range(len(results)):
   print 'For option: ' + str(i+1)
   print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
   print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
   print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
   print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
PCA Test Results for target column CPM Change 0vs1 Log, Seed: 674:

For option: 1, n_components: 3
Average Train RMSE: 0.27166416635077134
Average Train R2: 0.00858998073446926
Average Test RMSE: 0.2652159952912888
Average Test R2: -0.0006940098167486974

For option: 2, n_components: 5
Average Train RMSE: 0.2714088629922835
Average Train R2: 0.010447269384867397
Average Test RMSE: 0.2651886394452006
Average Test R2: -0.0006835313126161591

For option: 3, n_components: 10
Average Train RMSE: 0.2677190537102853
Average Train R2: 0.03716663353250475
Average Test RMSE: 0.2649224939094136
Average Test R2: 0.0018782163614055381

For option: 4, n_components: 20
Average Train RMSE: 0.2646851336392976
Average Train R2: 0.058757115614457386
Average Test RMSE: 0.26405469560859934
Average Test R2: 0.007626193139637738

For option: 5, No PCA
Average Train RMSE: 0.22387304477967457
Average Train R2: 0.32545440408390897
Average Test RMSE: 0.23026933020187643
Average Test R2: 0.22585182230712633


Review:
PCA did not help improving R2.

"""





# Case 3: Features correlated to target
# Option 1: Choose top 10 features, correlated to target
# Option 2: Choose top 20 features, correlated to target
# Option 3: Choose top 50 features, correlated to target
# Option 4: Choose top 100 features, correlated to target
# Option 5: Choose top 200 features, correlated to target
# Option 6: Choose all features, regardless of correlation


results = []
df_case = df.copy()
corr_table = df_case.corr().ix['cpm_change(0vs1)_log', :-1]
corr_table = corr_table.abs().sort_values(axis=0, ascending=False)

corr_table = corr_table.drop(labels=['cp_thousand_reach(t-0)_usd',
                                     'cp_thousand_reach_change(0vs1)',
                                     'cp_thousand_reach_change(0vs1)_log',
                                    'cpm(t-0)_usd',
                                    'cpm_change(0vs1)',
                                    'cpm_change(0vs1)_log'])

# Option 1: Choose top 10 features, correlated to target
corr_features = list(corr_table.index[range(0,10)])
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         '10 correlated features',
                         seed))

# Option 2: Choose top 20 features, correlated to target
corr_features = list(corr_table.index[range(0,20)])
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         '20 correlated features',
                         seed))

# Option 3: Choose top 50 features, correlated to target
corr_features = list(corr_table.index[range(0,50)])
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         '50 correlated features',
                         seed))

# Option 4: Choose top 100 features, correlated to target
corr_features = list(corr_table.index[range(0,100)])
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         '100 correlated features',
                         seed))

# Option 5: Choose top 200 features, correlated to target
corr_features = list(corr_table.index[range(0,200)])
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         '200 correlated features',
                         seed))

# Option 6: Choose all features, regardless of correlation
corr_features = list(corr_table.index[range(0,len(corr_table))])
results.append(modelling(df_case[corr_features],
                    df_case['cpm_change(0vs1)_log'],
                    '200 correlated features',
                    seed))


print 'Number of features used in Linear Regression, decided by correlation to target, Seed:
674:'
for i in range(len(results)):
    print 'For option: ' + str(i+1)
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])


"""
Number of features used in Linear Regression, decided by correlation to target, Seed: 674:

For option: 1
Average Train RMSE: 0.25375969732516157
Average Train R2: 0.1348723841573915
Average Test RMSE: 0.24749586273522994
Average Test R2: 0.12626433541067952

For option: 2
Average Train RMSE: 0.25128357592559447
Average Train R2: 0.15158465175272334
Average Test RMSE: 0.2460733419295122
Average Test R2: 0.1329372189228998

For option: 3
Average Train RMSE: 0.2509490433788367
Average Train R2: 0.15364742482237728
Average Test RMSE: 0.24612720160776416
Average Test R2: 0.13071400860009197

For option: 4
Average Train RMSE: 0.2503219194947005
Average Train R2: 0.1575418295507221
Average Test RMSE: 0.2562521818873632
Average Test R2: 0.030388450352519258
```

```
For option: 5
Average Train RMSE: 0.247090694488108
Average Train R2: 0.17849608714200074
Average Test RMSE: 0.4794876785297985
Average Test R2: -4.047825260493367

For option: 6
Average Train RMSE: 0.24681855482206067
Average Train R2: 0.17984423254182408
Average Test RMSE: 0.653816316483946
Average Test R2: -11.411021251202612

Review:

Using top 20 correlated features generated the maximum Test R2 and minimum RMSE.
After 50 features, Test R2 dropped significantly.
We will continue our modelling with top 20 best_features.

Note: Used linear regression, instead of Random Forest.
"""

# Check if top 20 correlated feature is a subset of best_features
set(corr_features).issubset(best_features)

"""
What are the top 20 most correlated features?
'cpm_change(1vs2)',
'cpm_change(1vs7)',
'cp_thousand_reach_change(1vs2)',
'cp_thousand_reach_change(1vs7)',
'bid_change(0vs1)',
'ctr_change(1vs7)',
'ctr_change(1vs2)',
'cpm(t-1)_usd',
'cp_thousand_reach(t-1)_usd',
'ctr(t-1)',
'frequency_change(1vs7)',
'social_ctr_change(1vs2)',
'social_ctr_change(1vs7)',
'weighted_cpm_lastweek_usd',
'social_ctr(t-1)',
'frequency_change(1vs2)',
'cpm(t-2)_usd',
'cpm_change(3vs4)',
'cp_thousand_reach_change(3vs4)',
'absorption_r(t-1)'

"""

del df_case, corr_features, corr_table









# Case 4: Features correlated within themselves
# Option 1: Drop correlated features
# Option 2: Do not drop correlated features
# Option 3: Choose all feaures, regardless of correlation

results = []
df_case = df.copy()

corr_table = df_case.corr().ix['cpm_change(0vs1)_log', :-1]
corr_table = corr_table.abs().sort_values(axis=0, ascending=False)

corr_table = corr_table.drop(labels=['cp_thousand_reach(t-0)_usd',
                              'cp_thousand_reach_change(0vs1)',
                              'cp_thousand_reach_change(0vs1)_log',
                              'cpm(t-0)_usd',
                              'cpm_change(0vs1)',
                              'cpm_change(0vs1)_log'])

corr_features = list(corr_table.index[range(0,20)])

df_corr = df_case[corr_features].corr().abs()
df_corr = pd.melt(df_corr.reset_index(), id_vars = ['index'], var_name = ['var_2'], value_name
= 'corr')
df_corr = df_corr.sort_values(by='corr', ascending=False)
df_corr = df_corr[df_corr['corr'] < 1.0]
df_corr = df_corr[df_corr['corr'] > 0.50]
df_corr = df_corr.drop_duplicates(subset=['corr'], keep='first')

"""
                              index                       var_2
153      weighted_cpm_lastweek_usd                  cpm(t-1)_usd
276                  cpm(t-2)_usd      weighted_cpm_lastweek_usd
156                  cpm(t-2)_usd                  cpm(t-1)_usd
167                  cpm(t-1)_usd     cp_thousand_reach(t-1)_usd
268     cp_thousand_reach(t-1)_usd      weighted_cpm_lastweek_usd
176                  cpm(t-2)_usd     cp_thousand_reach(t-1)_usd
Only one of them should stay.

Stay: cpm(t-1)_usd

Drop:
cp_thousand_reach(t-1)_usd
weighted_cpm_lastweek_usd
cpm(t-2)_usd
---
2    cp_thousand_reach_change(1vs2)              cpm_change(1vs2)
Only one of them should stay.

Stay: cpm_change(1vs2)
Drop: cp_thousand_reach_change(1vs2)
---
358  cp_thousand_reach_change(3vs4)              cpm_change(3vs4)
Only one of them should stay.

Stay: cpm_change(3vs4)
Drop: cp_thousand_reach_change(3vs4)
---
61   cpm_change(1vs7)  cp_thousand_reach_change(1vs7)
Stay: cpm_change(1vs7)
Drop: cp_thousand_reach_change(1vs7)
---
289                     ctr(t-1)               social_ctr(t-1)
ctr(t-1) should stay, since Facebook will deprecate social_ctr metric.

"""
```

```
# Choose correlated features to drop
corr_drop = ['cp_thousand_reach(t-1)_usd',
         'weighted_cpm_lastweek_usd',
         'cpm(t-2)_usd',
         'cp_thousand_reach_change(1vs2)',
         'cp_thousand_reach_change(3vs4)',
         'cp_thousand_reach_change(1vs7)',
         'social_ctr(t-1)']

# Recalculate correlations between features after dropping correlated features
df_case = df.copy()

corr_features = list(corr_table.index[range(0,20)])
corr_features = list(set(corr_features) - set(corr_drop))

df_corr = df_case[corr_features].corr().abs()
df_corr = pd.melt(df_corr.reset_index(), id_vars = ['index'], var_name = ['var_2'], value_name
= 'corr')
df_corr = df_corr.sort_values(by='corr', ascending=False)
df_corr = df_corr[df_corr['corr'] < 1.0]
df_corr = df_corr[df_corr['corr'] > 0.50]
df_corr = df_corr.drop_duplicates(subset=['corr'], keep='first')

"""
Review:

13 features are not correlated between themselves and most correlated with Target.
"""

# Option 1: Drop correlated features
results.append(modelling(df_case[corr_features],
                df_case['cpm_change(0vs1)_log'],
                'Drop correlated features',
                seed))

# Option 2: Do not drop correlated features, choose top 20
corr_features = corr_features + corr_drop
results.append(modelling(df_case[corr_features],
                df_case['cpm_change(0vs1)_log'],
                'Do not drop correlated features',
                seed))
```

```
# Option 3: Choose all feaures, regardless of correlation
results.append(modelling(df_case[best_features],
                         df_case['cpm_change(0vs1)_log'],
                         'All best features',
                         seed))


print 'Dropping correlated features vs. using all best features, Seed: 674:'
for i in range(len(results)):
    print 'For option: ' + str(i+1)
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])

"""
Dropping correlated features vs. using all best features, Seed: 674:

For option: 1
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032526

For option: 2
Average Train RMSE: 0.25128357592559447
Average Train R2: 0.15158465175272337
Average Test RMSE: 0.2460733419295122
Average Test R2: 0.13293721892289975

For option: 3
Average Train RMSE: 0.24711139688621436
Average Train R2: 0.1782983506424713
Average Test RMSE: 0.28742430655811124
Average Test R2: -0.478683778687537


Review:

We are going to use first 13 features, since their RMSE & R2 are higher than best_features.

Features that are going to used in the regression:
```

```
'ctr_change(1vs7)',
'social_ctr_change(1vs2)',
'frequency_change(1vs7)',
'cpm_change(3vs4)',
'ctr(t-1)',
'cpm(t-1)_usd',
'social_ctr_change(1vs7)',
'cpm_change(1vs7)',
'absorption_r(t-1)',
'bid_change(0vs1)',
'frequency_change(1vs2)',
'cpm_change(1vs2)',
'ctr_change(1vs2)'

"""

corr_features = list(set(corr_features) - set(corr_drop))
corr_features_target = corr_features[:]
corr_features_target.append('cpm_change(0vs1)_log')
del df_corr, df_case, corr_drop, corr_table








# Case 4: Outlier Handling
# Option 1: Remove outliers - z-Score
# Option 2: Remove outliers - IQR-Score
# Option 3: Do-not Remove outliers

results = []

# Option 1: Remove outliers - z-Score
df_case = df.copy()
df_case = df_case[corr_features_target]
z = np.abs(stats.zscore(df_case))
df_case = df_case[(z < 3).all(axis=1)]
```

```
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'Drop outliers based on z > 3',
                         seed))

# Option 2: Remove outliers - IQR-Score
df_case = df.copy()
df_case = df_case[corr_features_target]

Q1 = df_case.quantile(0.25)
Q3 = df_case.quantile(0.75)
IQR = Q3 - Q1

df_case = df_case[~((df_case < (Q1 - 1.5 * IQR)) | (df_case > (Q3 + 1.5 *
IQR))).any(axis=1)]

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'Drop outliers based on IQR Score',
                         seed))


# Option 3: Do-not Remove outliers
df_case = df.copy()
df_case = df_case[corr_features_target]

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'Do not drop outliers',
                         seed))




print 'Removal of outliers, Seed: 674:'
for i in range(len(results)):
   print 'For option: ' + str(i+1)
   print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
   print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
   print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
   print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])
```

```
"""
Removal of outliers, Seed: 674:

For option: 1
Average Train RMSE: 0.1770845351511044
Average Train R2: 0.220689187786859
Average Test RMSE: 0.18282674543035077
Average Test R2: 0.15363320618084936

For option: 2
Average Train RMSE: 0.1252493166093948
Average Train R2: 0.1872774194187321
Average Test RMSE: 0.13285313494029521
Average Test R2: 0.07718340672003085

For option: 3
Average Train RMSE: 0.2250880776254756
Average Train R2: 0.31919483121555614
Average Test RMSE: 0.22961616489415362
Average Test R2: 0.2415293015929217

Review:

Since we already cleaned our data, removing the outliers decreased our score.
So we won't remove outliers in this case.

Note: Outlier cases may also contribute more to CPM change.
"""

del z, df_case, IQR, Q3, Q1





# Case 5: Feature Scaling
# Option 1: MinMaxScaler
# Option 2: MaxAbsScaler
# Option 3: StandardScaler
# Option 4: RobustScaler
# Option 5: No Scaling

results = []

# Option 1: MinMaxScaler
df_case = df.copy()
df_case = df_case[corr_features_target]
scaler = MinMaxScaler()
df_case[corr_features] = scaler.fit_transform(df_case[corr_features])

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'MinMaxScaler',
                         seed))

# Option 2: MaxAbsScaler
df_case = df.copy()
df_case = df_case[corr_features_target]
scaler = MaxAbsScaler()
df_case[corr_features] = scaler.fit_transform(df_case[corr_features])

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'MaxAbsScaler',
                         seed))

# Option 3: StandardScaler
df_case = df.copy()
df_case = df_case[corr_features_target]
scaler = StandardScaler()
df_case[corr_features] = scaler.fit_transform(df_case[corr_features])

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'StandardScaler',
                         seed))

# Option 4: RobustScaler
df_case = df.copy()
df_case = df_case[corr_features_target]
scaler = RobustScaler()
df_case[corr_features] = scaler.fit_transform(df_case[corr_features])
```

```
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'RobustScaler',
                         seed))


# Option 5: No Scaling
df_case = df.copy()
df_case = df_case[corr_features_target]

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'No Scaling',
                         seed))

print 'Scaling vs. No-Scaling, Seed: 674:'
for i in range(len(results)):
    print 'For option: ' + str(i+1)
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])



"""
Scaling vs. No-Scaling, Seed: 674:

For option: 1
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032523

For option: 2
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032515

For option: 3
```

```
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.24727066586800212
Average Test R2: 0.127991081120325

For option: 4
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.13603742214000816
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032526

For option: 5
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032526

Review:

Scaling has no effect on the performance.

"""
```

```
# Case 6: Normalizing
# Option 1: Normalizer
# Option 2: No-Normalizer

results = []

# Option 1: Normalizer
df_case = df.copy()
df_case = df_case[corr_features_target]
scaler = Normalizer()
df_case[corr_features] = scaler.fit_transform(df_case[corr_features])
```

```
results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'Normalizer',
                         seed))

# Option 2: No Normalizer
df_case = df.copy()
df_case = df_case[corr_features_target]

results.append(modelling(df_case[corr_features],
                         df_case['cpm_change(0vs1)_log'],
                         'No Normalizer',
                         seed))

print 'Normalizer vs. No-Normalizer, Seed: 674:'
for i in range(len(results)):
    print 'For option: ' + str(i+1)
    print 'Average Train RMSE: ' + str(results[i][0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[i][0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[i][0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[i][0]['avg_test_r2'])


"""
Normalizer vs. No-Normalizer, Seed: 674:

Normalizer vs. No-Normalizer, Seed: 674:
For option: 1
Average Train RMSE: 0.2501655642972066
Average Train R2: 0.1591278015943398
Average Test RMSE: 0.2443652197705811
Average Test R2: 0.1425599275042521

For option: 2
Average Train RMSE: 0.2535864868677269
Average Train R2: 0.1360374221400082
Average Test RMSE: 0.2472706658680021
Average Test R2: 0.12799108112032526


Review:
```

```
Normalizer has improved the performance.
We will use Normalizer.
"""



"""
Decisions so far:

- If we use Random Forest, max_features: 14

Choose rows:

- No filtering for significant change will be used.
- I decided not to include adset run time filter in my model.
- I decided not to use Conversions wanted for filtering.
- I won't use balanced Spend change in my model.
- I won't use bid change filtering in my model.

Choose columns:
- PCA did not help improving R2.
- So we won't remove outliers in this case.
- Scaling has no effect on the performance.
- We will use Normalizer.
- 13 features are not correlated between themselves and most correlated with Target.

Features that are going to used in the regression:
'ctr_change(1vs7)',
'social_ctr_change(1vs2)',
'frequency_change(1vs7)',
'cpm_change(3vs4)',
'ctr(t-1)',
'cpm(t-1)_usd',
'social_ctr_change(1vs7)',
'cpm_change(1vs7)',
'absorption_r(t-1)',
'bid_change(0vs1)',
'frequency_change(1vs2)',
'cpm_change(1vs2)',
'ctr_change(1vs2)'
"""
```

```
# Visualize predictions and actual values

def visualizer(actuals, predictions)
    X = actuals
    Y = predictions
    plt.axhline(y=0, c='red')
    plt.axvline(x=0, c='red')
    plt.scatter(X, Y, c='blue')
    plt.ylabel('Prediction')
    plt.xlabel('Actual')
    # plt.ylim([0, 10])
    # plt.xlim([0, 10])
     # plttext = 'Regression: ' + 'MSE:' + str(round(mean_squared_error(X,Y),4)) + ', R2:' +
str(round(result['avg_test_score'],4))
    plt.title(plttext, loc='left')
    plt.plot( [-1,1],[-1,1], c = 'red')
    plt.show()

# Array of Errors
errors = result['predictions'] - result['actuals']
plt.hist(errors)
plt.show()
```

try_models.py:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

#######################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES
######################################################################
######## MODELLING ###################################################
######################################################################

# import libraries
import pandas as pd
import numpy as np
import glob, os
import time
from datetime import datetime, timedelta
# import matplotlib.pyplot as plt
# import seaborn as sns
from math import sqrt
from scipy import stats

# Scikit Learn Libraries
from sklearn.linear_model import LinearRegression
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ARDRegression
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import HuberRegressor
from sklearn.linear_model import LassoLars
from sklearn.linear_model import Lars

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn import decomposition
```

```
# Change path
# path = os.path.expanduser('~/Documents/Scripts/CPM/Capstone/')
# os.chdir(path)

df = pd.read_pickle('df/df_clustered.gzip', compression='gzip')
# df with 141775 rows, 277 columns

# Check if NaN exists
df = df.dropna(axis='index', how='any', thresh=None, subset=None, inplace=False) # No NaN

# Log transformation of CPM Change
df['cpm_change(0vs1)_log'] = np.log(df['cpm_change(0vs1)']+1)
df['cp_thousand_reach_change(0vs1)_log']                                                    =
np.log(df['cp_thousand_reach_change(0vs1)']+1)

# Put Log columns to beginning
cols = df.columns.tolist()
cols.insert(2, cols.pop(cols.index('cpm_change(0vs1)_log')))
cols.insert(1, cols.pop(cols.index('cp_thousand_reach_change(0vs1)_log')))
df = df.reindex(columns = cols)

# Seed
seed = 674

# Regression Function

def adjusted_rsqu(rs, n, p):
    # rs: r-squared
    # n: number of observations
    # p: number of independent variables
    print('######################')
    print('R_Squared: ' + str(rs))
    print('Observations: ' + str(n))
    print('Variables: ' + str(p))
    adjusted_r = 1 - (1 - rs) * (n - 1) / (n - p - 1)
    print('Adjusted R: ' + str(adjusted_r))
    print('######################')
    return adjusted_r


def regression(features, target, algorithm, params, seed, description):
    # K-Fold splits
```

```
kf_splits = 10
kf = KFold(n_splits=kf_splits, shuffle=False, random_state=seed).split(
    features)

score_train_list = []
score_test_list = []
rmse_train_list = []
rmse_test_list = []
prediction_list = []
actual_list = []

for train_indices, test_indices in kf:
    X_train = features.iloc[train_indices]
    X_test = features.iloc[test_indices]
    y_train = target.iloc[train_indices]
    y_test = target.iloc[test_indices]

    reg = GridSearchCV(algorithm, params, scoring='r2')
    reg = reg.fit(X_train, y_train)

    trainy_predict = reg.predict(X_train)
    testy_predict = reg.predict(X_test)

    score_train_list.append(
        adjusted_rsqu(r2_score(y_train, trainy_predict),
                      trainy_predict.size, len(features.columns)))
    score_test_list.append(
        adjusted_rsqu(r2_score(y_test, testy_predict), testy_predict.size,
                      len(features.columns)))

    rmse_train_list.append(
        sqrt(mean_squared_error(y_train, trainy_predict)))
    rmse_test_list.append(sqrt(mean_squared_error(y_test, testy_predict)))

    prediction_list.append(testy_predict)
    actual_list.append(y_test)

to_return = {'target': target.name,
             'avg_train_r2': np.mean(score_train_list),
             'avg_test_r2': np.mean(score_test_list),
             'avg_train_rmse': np.mean(rmse_train_list),
             'avg_test_rmse': np.mean(rmse_test_list),
            'predictions': np.concatenate(prediction_list, axis=0),
            'actuals': np.concatenate(actual_list, axis=0),
            'algorithm': algorithm,
            'params': reg.best_params_,
            'description': description,
            'seed': seed,
            }
    return to_return


def modelling(features, target, description, seed):
    models = [
        # Linear Regression Algorithms

        # Linear Model
        #[LinearRegression(),
        #{'n_jobs': [-1]}],

        # Ridge Model
        #[Ridge(random_state=seed),
        #{'alpha':[1.0]}],

        # Lasso Model
        #[Lasso(random_state=seed),
        #{alpha:[1.0]}],

        # Bayesian Ridge Model
        #[BayesianRidge(),
        #{}],

        # Lasso Lars Model
        #[LassoLars(),
        #{alpha:[0.01]}],

        # Lars Model
        #[Lars(),
        #{n_nonzero_coefs:[1]}],

        # Tree Regression Algorithms

        # [RandomForestRegressor(n_estimators = 50, max_features = 13, \
        #                max_depth = 10, min_samples_split = 5, \
        #                n_jobs = -1, min_samples_leaf = 20, \
        #                oob_score = True, random_state = seed),
        #  {},
        #  ],

        # Support Vector Machine

        # Linear SVM
        #[LinearSVR(random_state=seed),
        #{'epsilon':[0.01, 0.05, 0.10, 0.20, 0.50]}],

        # SVR
        [SVR(kernel='rbf', gamma=0.7, tol=0.01, C=1.0, epsilon=0.2, shrinking=True),
        {}],

        # SGD Regressor
        #[SGDRegressor(max_iter=1000, random_state=seed),
        #{'loss':['squared_loss','huber','epsilon_insensitive','squared_epsilon_insensitive'],
        # 'penalty': ['none','l2','l1','elasticnet'],
        # 'alpha':[0.0001, 0.001, 0.01, 0.1],
        # 'l1_ratio':[0, 0.15, 0.25, 0.50, 0.75, 1],
        # 'learning_rate':['constant','optimal','invscaling','adaptive']
        # }
        # ],
        ]

    # Decision Tree Regression
    # Various parameters

    func_results = []
    for model in models:
        func_results.append(
            regression(features, target, model[0], model[1], seed,
                       description))
    return func_results

# Backup

df_backup = df.copy()

"""
Decisions so far:
```

```
Choose columns:
- We will use Normalizer.

- 13 features are not correlated between themselves and most correlated with Target.

Features that are going to used in the regression:
'ctr_change(1vs7)',
'social_ctr_change(1vs2)',
'frequency_change(1vs7)',
'cpm_change(3vs4)',
'ctr(t-1)',
'cpm(t-1)_usd',
'social_ctr_change(1vs7)',
'cpm_change(1vs7)',
'absorption_r(t-1)',
'bid_change(0vs1)',
'frequency_change(1vs2)',
'cpm_change(1vs2)',
'ctr_change(1vs2)'
"""

# Targets & Features

features = df[['ctr_change(1vs7)',
               'social_ctr_change(1vs2)',
               'frequency_change(1vs7)',
               'cpm_change(3vs4)',
               'ctr(t-1)',
               'cpm(t-1)_usd',
               'social_ctr_change(1vs7)',
               'cpm_change(1vs7)',
               'absorption_r(t-1)',
               'bid_change(0vs1)',
               'frequency_change(1vs2)',
               'cpm_change(1vs2)',
               'ctr_change(1vs2)']]

features_cols = list(features.columns.values)

target = df['cpm_change(0vs1)_log']
```

```
# Normalize
scaler = Normalizer()
features[features_cols] = scaler.fit_transform(features[features_cols])




# features = features.head(10000)
# target = target.head(10000)

# Results
results = modelling(features, target, 'Default result', seed)

print 'Model results, Seed: 674:'
for i in range(len(results)):
    print 'Average Train RMSE: ' + str(results[0]['avg_train_rmse'])
    print 'Average Train R2: ' + str(results[0]['avg_train_r2'])
    print 'Average Test RMSE: ' + str(results[0]['avg_test_rmse'])
    print 'Average Test R2: ' + str(results[0]['avg_test_r2'])
    print 'Best parameters: ' + str(results[0]['params'])

"""

SVR, without parameters, Seed: 674:
Average Train RMSE: 0.24755326224685845
Average Train R2: 0.17660233557677757
Average Test RMSE: 0.24173441733436665
Average Test R2: 0.1632642786693974

SVR, kernel: 'linear','rbf','poly','sigmoid', first 10K rows, Seed: 674:
Average Train RMSE: 0.38049688498782036
Average Train R2: 0.20608181464875214
Average Test RMSE: 0.3773640732427654
Average Test R2: 0.18695326143427282
Best parameters: {'kernel': 'rbf'}

SVR, gamma: 0.00001, 0.0001, 0.001, 0.01, 0.1, 0.2, 1., 10., 100., 1000. , first 10K rows,
Seed: 674:
Average Train RMSE: 0.36135035387196146
Average Train R2: 0.28399033480160074
Average Test RMSE: 0.37143241660080567
Average Test R2: 0.21197443031255872
Best parameters: {'gamma': 1.0}

SVR, gamma: 0.5, 0.75, 1., 2. , first 10K rows, Seed: 674:
Average Train RMSE: 0.3663782734950667
Average Train R2: 0.26387188562026154
Average Test RMSE: 0.3714634353481324
Average Test R2: 0.21179737218413877
Best parameters: {'gamma': 0.75}

SVR, gamma: [0.6, 0.7, 0.8] , first 10K rows, Seed: 674:
Average Train RMSE: 0.36589779892142016
Average Train R2: 0.2658376531134338
Average Test RMSE: 0.3712833815267862
Average Test R2: 0.21257766140926143
Best parameters: {'gamma': 0.7}

SVR, tol: [0.00001, 0.0001, 0.001, 0.01, 0.1, 1., 2.] , first 10K rows, Seed: 674:
Average Train RMSE: 0.36568931477172634
Average Train R2: 0.26668675056532265
Average Test RMSE: 0.3710303504080824
Average Test R2: 0.21363303041279186
Best parameters: {'tol': 0.01}

SVR, C: [0.0001, 0.001, 0.01, 0.1, 1., 2., 10., 100., 1000.] , first 10K rows, Seed: 674:
Average Train RMSE: 0.36558924430185563
Average Train R2: 0.2670853029820186
Average Test RMSE: 0.37106946820233366
Average Test R2: 0.21346576116078433
Best parameters: {'C': 1.0}

SVR, 'C':[0.75, 1., 1.25] , first 10K rows, Seed: 674:
Average Train RMSE: 0.36636869431959707
Average Train R2: 0.2639512047890122
Average Test RMSE: 0.371184202003894
Average Test R2: 0.21296504449337492
Best parameters: {'C': 1.0}

SVR, 'epsilon':[0.0001, 0.001, 0.01, 0.1, 1., 2., 10., 100., 1000.] , first 10K rows, Seed: 674:
Average Train RMSE: 0.36567345405498586
Average Train R2: 0.2667472537398233
Average Test RMSE: 0.3711548987414313
Average Test R2: 0.21310871540339732
Best parameters: {'epsilon': 0.1}

SVR, 'epsilon':[0.05, 0.1, 0.2] , first 10K rows, Seed: 674:
Average Train RMSE: 0.3653082429957336
Average Train R2: 0.26820798927772377
Average Test RMSE: 0.3716927303356901
Average Test R2: 0.21088131272894745
Best parameters: {'epsilon': 0.2}

SVR, 'epsilon':[0.1, 0.2, 0.3, 0.4, 0.5] , first 10K rows, Seed: 674:
Average Train RMSE: 0.3652581486436769
Average Train R2: 0.26840899319253025
Average Test RMSE: 0.3716331236389133
Average Test R2: 0.21113034365175878
Best parameters: {'epsilon': 0.2}

***

SVR, best parameters: kernel='rbf', gamma=0.7, tol=0.01, C=1.0, epsilon=0.2,
shrinking=True, Seed: 674:
Average Train RMSE: 0.24179369245332621
Average Train R2: 0.21444941808891502
Average Test RMSE: 0.23833223194944816
Average Test R2: 0.18464607282979686
"""
```

visualize.py:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-

###########################################################################
# CAPSTONE PROJECT, MEF BIG DATA ANALYTICS MASTER PROGRAM
# 2017 - 2018
# SEMIH TEKTEN
# PREDICTING FACEBOOK AD IMPRESSIONS & CPM VALUES
```

```
######################################################################
######## VISUALIZATION ###############################################
######################################################################

# import libraries
import pandas as pd
import numpy as np
import os

import matplotlib.pyplot as plt
import seaborn as sns
from math import sqrt
from scipy import stats

# Change path
# path = os.path.expanduser('~/Documents/Scripts/CPM/Capstone/')
# os.chdir(path)

df_vis = pd.read_pickle('df_vis.gzip', compression='gzip')
# df with 141775 rows, 277 columns

# Check if NaN exists
df_vis = df_vis.dropna(axis='index', how='any', thresh=None, subset=None, inplace=False) #
No NaN

# Seed
seed = 674

# Backup
df_vis_backup = df_vis.copy()


# Features & Target as String List
feature_list = list(df_vis.columns)
feature_list.remove('cpm_change(0vs1)_log')
target = 'cpm_change(0vs1)_log'


"""
Decisions so far:

Choose columns:
```

```
- We will use Normalizer.

- 13 features are not correlated between themselves and most correlated with Target.

Features that are going to used in the regression:
'ctr_change(1vs7)',
'social_ctr_change(1vs2)',
'frequency_change(1vs7)',
'cpm_change(3vs4)',
'ctr(t-1)',
'cpm(t-1)_usd',
'social_ctr_change(1vs7)',
'cpm_change(1vs7)',
'absorption_r(t-1)',
'bid_change(0vs1)',
'frequency_change(1vs2)',
'cpm_change(1vs2)',
'ctr_change(1vs2)'
"""


# Diagonal correlation matrix (Target vs features)

# Compute the correlation matrix
corr = df_vis.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(13, 11))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 0, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1.0,annot= True,
            square=True, linewidths=.7, cbar_kws={"shrink": .5})

# Histogram of all features, removing outliers with quantiles
Q1 = df_vis.quantile(0.10)
```

```
Q3 = df_vis.quantile(0.90)
IQR = Q3 - Q1
df_new = df_vis[~((df_vis < (Q1 - 1.5 * IQR)) | (df_vis > (Q3 + 1.5 * IQR))).any(axis=1)]
df_new.hist(figsize=(16, 20), bins=500, xlabelsize=10, ylabelsize=10)

# Pairplot to show the relationships between the features and the target
sns.pairplot(data=df_vis,x_vars=feature_list,y_vars=target)

# Distribution of the target feature 'cpm_change(0vs1)_log
dists = [stats.alpha, stats.anglit, stats.arcsine, stats.argus, stats.beta, stats.betaprime, stats.bradford, stats.burr, stats.burr12,
                 stats.cauchy, stats.chi, stats.chi2, stats.cosine, stats.crystalball, stats.dgamma, stats.dweibull, stats.erlang, stats.expon,
           stats.exponnorm, stats.exponweib, stats.exponpow, stats.f, stats.fatiguelife, stats.fisk, stats.foldcauchy, stats.foldnorm,
                stats.frechet_r, stats.frechet_l, stats.genlogistic, stats.gennorm, stats.genpareto, stats.genexpon, stats.genextreme,
             stats.gausshyper, stats.gamma, stats.gengamma, stats.genhalflogistic, stats.gilbrat, stats.gompertz, stats.gumbel_r, stats.gumbel_l,
           stats.halfcauchy, stats.halflogistic, stats.halfnorm, stats.halfgennorm, stats.hypsecant, stats.invgamma, stats.invgauss,
        stats.invweibull, stats.johnsonsb, stats.johnsonsu]

dists_two = [stats.kappa4, stats.kappa3, stats.kstwobign, stats.laplace,
              stats.levy, stats.levy_l, stats.logistic, stats.loggamma, stats.loglaplace, stats.lognorm, stats.lomax,
           stats.maxwell, stats.mielke, stats.nakagami, stats.ncx2, stats.nct, stats.norm,
                stats.pearson3, stats.powerlaw, stats.powerlognorm, stats.powernorm, stats.rdist, stats.reciprocal, stats.rayleigh,
             stats.rice, stats.recipinvgauss, stats.semicircular, stats.skewnorm, stats.t, stats.trapz, stats.triang, stats.truncexpon,
                 stats.truncnorm, stats.uniform, stats.vonmises, stats.vonmises_line, stats.wald, stats.weibull_min,
           stats.weibull_max, stats.wrapcauchy]

for dist in dists_two:
   print("Started for: " + dist.name)
   sns.set()
   plt.figure(figsize=(12,8))
   plt.xlim(-1, 1)
   dist_plot = sns.distplot(df_vis['cpm_change(0vs1)_log'], hist=True, bins=500, fit=dist).set_title(dist.name)
```

```
    fig = dist_plot.get_figure()
    file_name_w_path = 'fit_dists/' + dist.name + '_dist.png'
    fig.savefig(file_name_w_path)
    print("Ended for: " + dist.name)


def get_best_distribution(data):
                                                                                    #
https://stackoverflow.com/questions/37487830/how-to-find-probability-distribution-and-para
meters-for-real-data-python-3
        dist_names  =  ["norm",  "beta",  "burr",  "burr12",  "cauchy",  "crystalball",  "dgamma",
"dweibull", "exponnorm",  "fisk", "foldcauchy",
                              "genlogistic", "gennorm", "hypsecant", "johnsonsu", "laplace", "logistic",
"loglaplace", "nct", "t"]
    dist_results = []
    params = {}
    for dist_name in dist_names:
        dist = getattr(stats, dist_name)
        param = dist.fit(data)

        params[dist_name] = param
        # Applying the Kolmogorov-Smirnov test
        D, p = stats.kstest(data, dist_name, args=param)
        print("p value for "+dist_name+" = "+str(p))
        dist_results.append((dist_name, p))

    # select the best fitted distribution
    best_dist, best_p = (max(dist_results, key=lambda item: item[1]))
    # store the name of the best fit and its p value

    print("Best fitting distribution: "+str(best_dist))
    print("Best p value: "+ str(best_p))
    print("Parameters for the best fit: "+ str(params[best_dist]))

    return best_dist, best_p, params[best_dist]

get_best_distribution(df_vis['cpm_change(0vs1)_log'])

"""
p value for norm = 0.0
p value for beta = 0.0
p value for burr = 1.0376459810115412e-273
```

```
p value for burr12 = 9.036728654232608e-290
p value for cauchy = 2.0304468915802398e-164
p value for crystalball = 0.0
p value for dgamma = 7.24762276632266e-51
p value for dweibull = 6.693533713472155e-38
p value for exponnorm = 0.0
p value for fisk = 2.9586554796192175e-298
p value for foldcauchy = 2.819678021874141e-193
p value for genlogistic = 2.6265027555286723e-263
p value for gennorm = 4.2739427863644434e-30
p value for hypsecant = 9.5534317451281e-137
p value for johnsonsu = 0.3439750123259282
p value for laplace = 1.559500338283894e-59
p value for logistic = 1.9380053713260365e-271
p value for loglaplace = 5.516238303027958e-60
p value for nct = 0.08494587876072554
p value for t = 0.05107259990987811

Best fitting distribution: johnsonsu
Best p value: 0.3439750123259282
Parameters for the best fit: (0.0048928698851670345, 0.9735257590501925,
0.0013939443593146712, 0.14600510233690264)

('johnsonsu',
 0.3439750123259282,
 (0.0048928698851670345,
  0.9735257590501925,
  0.0013939443593146712,
  0.14600510233690264))
"""


#  Distribution of features that are most correlated with the target.
# cpm_change(1vs7)
# cpm_change(1vs2)

sns.set()
plt.figure(figsize=(12,8))
ax = sns.distplot(df_vis['cpm_change(1vs2)'])

plt.figure(figsize=(12,8))
ax = sns.distplot(df_vis['cpm_change(1vs7)'])
```

```
#  Distribution of features that are most correlated with each other.
# frequency_change(1vs2)
# frequency_change(1vs7)

sns.set()
plt.figure(figsize=(12,8))
ax = sns.distplot(df_vis['frequency_change(1vs2)'])

plt.figure(figsize=(12,8))
ax = sns.distplot(df_vis['frequency_change(1vs7)'])

# Jointplot & Regression
sns.set()
sns.jointplot("cpm_change(1vs2)", "cpm_change(0vs1)_log", data=df_vis, kind='reg')

sns.set()
sns.jointplot("frequency_change(1vs2)", "frequency_change(1vs7)", data=df_vis, kind='reg')

# Violinplot
sns.violinplot(data=df_vis, palette="Set3", bw=.2, cut=1, linewidth=1)
```