INTERNATIONAL
HELLENIC
UNIVERSITY

Programmatic Automation and Yield Optimization on the Ad Exchange

Galinos Giaglis

SID: 3305150010

Supervisor: Prof. Christos Tjortjis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in E-Business, Innovation & Entrepreneurship (Part-time)

DECEMBER 2018

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in eBusiness, Innovation & Entrepreneurship at the International Hellenic University.

The total media ad spending worldwide will rise to 7.4%, to $628 Bn in 2018, according to an eMarketer report, while the digital media (digital advertising) itself will account for almost 45% of the investments made, partly thanks to the global ecommerce shifting sizeable amounts of budgets from the TV ecosystem to the Digital placements. Marketer projections put digital advertising to a valuation of $129 Bn by 2021, making the digital advertising sector one of the largest in the non-tangible products market and one of the most intriguing to further investigate, automate and invest in. The advertising ecosystem is currently comprised of thousands of intermediary entities between an advertiser and a publisher; the two most essential aspects of this market. Unfortunately, the chain between the advertiser and the publisher is not a straight line and is usually filled with the intermediaries that in some cases provide excellent value and in other cases just intervene with the price and misalign the information between an advertiser and a publisher. The misinformation caused by the intermediaries along with the many different subchannels of digital advertising that exist in the market, e.g. direct buys, programmatic buys, performance buys, currently affect the way ads are being bought online. The purpose of this dissertation is to investigate the actions needed and create an algorithm that sets a (economic) yield management strategy, directly setting the price that they sell their ads on the global exchange.

This goal was successfully achieved by creating a software that works in four steps. First collects historical data from websites, their ad placements and the ad vendors’ reporting tool, clears the redundant data, analyzes all possible variables affecting the price of the ad and deciding which truly affect the price and then using information around these variables makes suggestions for higher or lower price that can consecutively lead to higher revenue for a publisher.

I would like to acknowledge my supervisor Dr. Christos Tjortjis for his valuable help and support in all stages of this Dissertation.

Galinos Giaglis
7 December 2018

Contents

1 Introduction

A publisher may decide to sell advertising space (ad inventory) online in two different and distinct ways. The first way it is by signing an Insertion Order (IO) with the Advertiser. In this IO the advertiser will buy a predefined amount of Ad Impressions (a term used for the quantity of advertising) at a predefined price in a predefined position on the website. This process is called “Direct Sell” or “Reservation Sell” and is considered the most sacred form of advertisement as each IO is as important as a contract between the advertiser and the publisher. When a publisher can’t sell all of the ad inventory, one uses “Ad Exchanges”, i.e Google, Appnexus, Rubicon Project, Index Exchange among others, which have several thousand buyers connected to them and act as an intermediary between their buyers and the publishers. By having such a large volume of advertisers, they have the ability to fill any gaps that exist from the publisher’ inability to sell all of the ad inventory directly. This type of selling the **remnant inventory** is called “Programmatic Sell” and is the focus of investigation for this dissertation.

In our dissertation we will investigate the pricing strategies a publisher can set by utilizing a handful of variables that affect the price an ad can be sold at. Some of these variables are the size of the ad (e.g. 300x250, 300x600 etc.), the device type (e.g. Desktop, Mobile, Tablet, Connected TV), the transparency of the advertiser (e.g. Branded or Anonymous) among others. Our focus will be to identify all affecting variables, separate the most relevant, use the pertinent information to predict a higher revenue generating price and apply that strategy starting the cycle again the next day.

To be more specific, in this dissertation we present a time series algorithmic procedure on how to forecast floor pricing values of Google Ad Manager pricing rules, in order to achieve revenue increment (maximization) based on historical data. We describe our dataset extensively so that we can acquire information on the behavior of floor values along with the rest of the features and especially the ones highly correlated to the Ad eCPM. For each set of features that affect the Ad eCPM value, we tried several models in an iterative way in our effort to find the best fit to our dataset. These models were checked according to a set of statistical metrics and the results were visualized. Finally, for each set of features along with the model that fit best, we forecasted the values of Ad eCPM which indicated the optimal Floor value for each pricing rule. Furthermore, we are in the phase of developing a neural network based on the proposed modeling approach that would programmatically learn, decide and set of the optimal Floor value of each pricing rule.

The first chapter of this dissertation includes the introduction where a brief walk through the algorithmic procedure that was developed is presented. In addition, a literature review is provided aside with the proposed approach definition. It focuses on the advantages that this approach has, including general information about the software developed

In the second chapter, an example dataset is analyzed and decomposed in order to understand the given dimensions and metrics that will try to fit in the modeling phase of the solution.

In the third chapter, the methodology of the analysis is explained including the five different models specified. In this section, each model of the analysis which concluded in using the Seasonal ARIMA process is explained with its weaknesses in comparison to the one produced by this process.

Next in the fourth chapter, the results and several outcomes of the analysis are shown. This section presents the results by visualizing the model fittings and the forecasting values aside with the original data. In addition, this section shows the final results after applying the software analysis and modeling to the real-time website pricing rules.

In the fifth chapter, the instructions of how to use the developed software are explained by giving a variety of ways to run it.

In the sixth chapter, the concluded results are shown with the suggestions for improvement.

The last chapter includes the full source code of the application in an ipython notebook format that ran on a specific website, over a time period of three months. The resulting revenue uplift is still under monitoring because of the exogenous factors that may apply to this optimization problem.

1.1 Literature overview

To the best of our knowledge, researchers' interest is focused on real time bidding (Wush Chi-Hsuan Wu, 2015), (Weinan Zhang, 2016), (Jun Wang, 2016) techniques which require data mining provided by scripting tools or custom pieces of code that reside on the client's website. These pieces of code are responsible for tracking features and metrics in order to understand and deliver statistical values and probabilistic models to solve the revenue optimization problem. Such solutions provide a state-of-the-art approach, in an academic interest perspective, but lack on application in real-life websites. Their major drawback rises when the owner of the website declines to provide access for equivalent research. Moreover, several solutions need to retain a waiting state of the ad unit in order to get the best revenue for each ad impression. This fact causes the "line-item to be auctioned" to expire, because of the time margin expiration leading to the impressions being lost.

Surveys like (Shuai Yuan, 2014) also state clearly how most of the website analysis are based on cookie manipulation and information retrieval in order to analyze efficiently online user characteristics so as to target respectively their ad campaigns.

1.2 Proposed approach definition

The proposed approach is trying to achieve a solution through experimentation with historical data retrieved from the ad vendor manager tool (Google Ad Manager) and not from the website directly. This gives us the opportunity to follow several dimensions and extract information based on a set of metrics which eventually provide the ability to forecast floor pricing values and thus optimize our revenue.

In addition, from a scientific point of view, we state that our algorithmic procedure tries to solve a time related problem, thus it requires a time series problem solving approach. This differentiates our solution from a regular regression problem solution in 2 ways:

1. It is ***time dependent***. So, the basic assumption of a linear regression model, that the observations are independent, doesn't hold in this case.

2. Along with an increasing or decreasing trend, most time series have some form of ***seasonality trends***, i.e. variations specific to a particular time frame. For example, if we visualize Ad Impressions over time, we will invariably find lower values in the weekdays rather than the weekend which depicts the fact that more people tend to browse their favorite content sites on the weekend and by that, the number of page requests increase which consequently increases the ad impressions.

Also, the resulted pricing rules are provided to the Google Ad Manager tool and processed for a period of three days over real-life websites in order to acquire feedback about the model parameters and the resulting revenue gain.

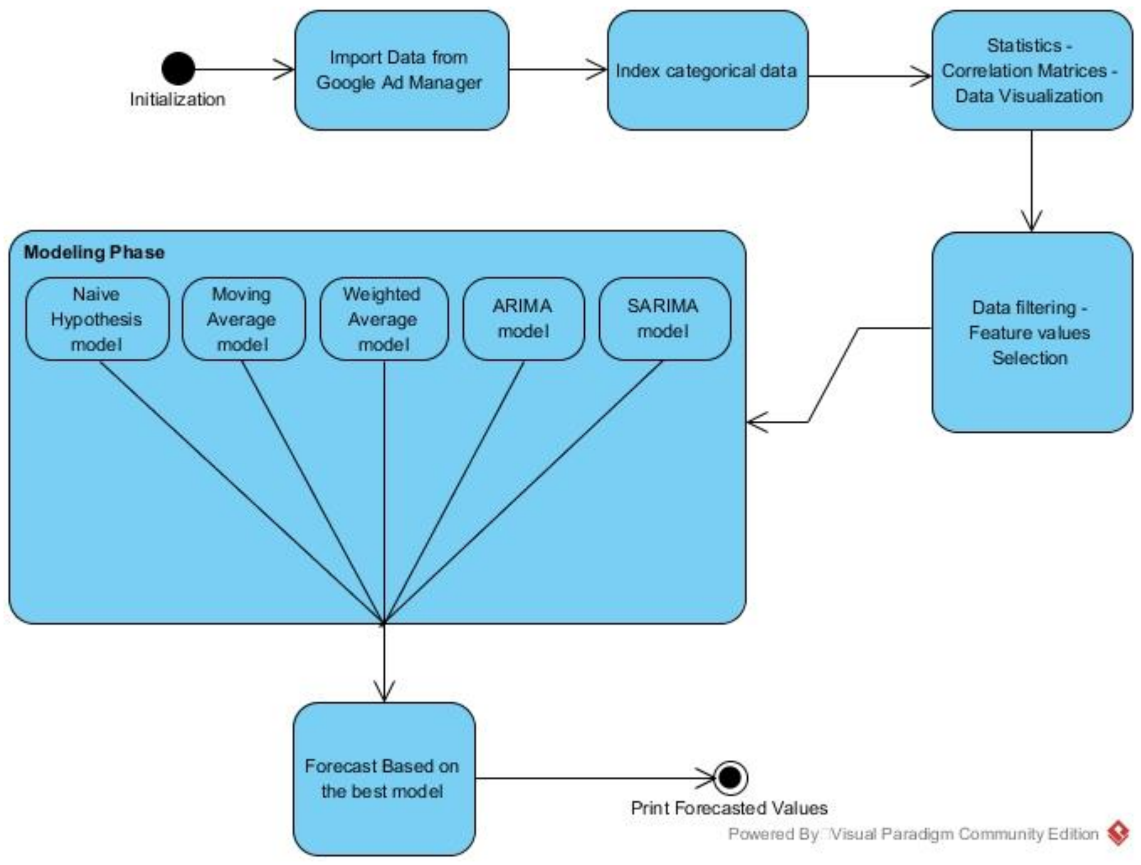

Initialization
Import Data from Google Ad Manager
Index categorical data
Statistics - Correlation Matrices - Data Visualization
Data filtering - Feature values Selection
Modeling Phase
Naive Hypothesis model
Moving Average model
Weighted Average model
ARIMA model
SARIMA model
Forecast Based on the best model
Print Forecasted Values
Powered By Visual Paradigm Community Edition

Figure 1: Flowchart diagram

Following the above assumptions, we propose an algorithmic procedure consisted of six steps as shown in Figure 1.

To start with, we acquire a historical query from the Google Ad Manager reporting tool containing specific dimensions and metrics as features of our dataset. These features will be manipulated and finally exposed to the modeling functions.

To derive numerical results from all the features gathered in the previous step, there has to follow an indexing step, where all the features that are not numerical will be transformed into numerical ones. Then the data set is ready to be studied through different statistical functions so that certain properties will be revealed. Among them are the correlation factors of the features such as inventory size, device category and branding type which will reveal association rules between them and will be used as filtering factors.

In the third step, the features that have the strongest correlation factors are chosen and specifically those that correlate best with Ad eCPM. Certain visualizations are provided to give the researcher a hint of the distribution of the data set values itself, along with histograms showing how close to the normal distribution these measures are. All features compete at a correlation matrix having a score that indicates the correlation between their values, e.g., inventory size ‘*300x250*’ is strongly correlated with mobile devices and especially with smart phones while their associated branding type is ‘branded.’

While reaching step four, our data set is being filtered by the feature values that scored best and indicated from the third step. Each set of these pairs ‘*feature – value*’ will be modeled independently and a different model will be trained by the software. The final set of feature values plus the labeled values is acquired for the modeling step aside with the aggregation information to form a per day value of Ad eCPM.

The fifth step is the modeling phase of this procedure and consists of six modeling solutions. Starting with a naïve hypothesis model, this procedure presents a baseline model. Next approaches are more sophisticated introducing a moving average model and as an extension of a weighted average model. A holt-winter exponential smoothing model is also demonstrated. At last an ARIMA model and a Seasonal ARIMA model present the best approach to this case study. Each of them are tested against several statistical measures and conclusively picked the best parameter initialization for them.

As a final step, this procedure uses the final model is selected on the modeling phaseand forecasts data for a period of time ahead. The resulting outcomes are visualized and demonstrated respectively along with evidence of correctness.

The proposed procedure by this dissertation unfortunately has also its drawbacks, with the most important to be introduced by the Google Ad Manager reporting tool which does not provide data with a sampling rate less than a day e.g. hourly or per minute. Thus, the presented procedure can forecast Ad eCPM values at a minimum per day time period basis. In case of a larger time period selected e.g. per week, a larger dataset should be acquired in order to limit the forecasting error inside the proposed probability confidence interval.

Moreover, exogenous factors that will influence Ad eCPM values in the 24 hours’ time frame cannot be introduced into our model because of the lack of relevant information by the vendors’ reporting tool.

1.3 Terms definition

The full meanings of the abbreviations used in this dissertation are:

Ad Manager – The Google tool used to deliver the ads on the page. It is essential an Ad Server or an intermediary delivering the ads from the publisher to the user

Ad Impressions – Absolute number that shows the amount of advertising ads shown

Ad Requests – Absolute number that shows the number an ad was requested (but not always shown)

Fill Rate or Coverage – The ratio of the Ad Impressions Served / the Ad Requests sent

Ad eCPM–The price for 1,000 Ad Impressions

CPM = Cost per Mile, the cost for 1,000 units

Revenue – Ad eCPM * (Ad Impressions/1000)

Ad Inventory – The available inventory the publisher can sell

Direct Sell – When a publisher sells his inventory directly to the advertiser

Programmatic Sell – When a publisher offers his remnant inventory in the Ad Exchanges

Branding Type – When the advertiser wishes to disclose her name and landing page URL. Two types of branding types exist, Branded (full disclosure) and Anonymous (no disclosure)

Floor Value – the minimum ad eCPM price required for an ad to be eligible for display in a website.

ACF – Autocorrelation function

PACF – Partial Autocorrelation function

AR – Autoregressive

MA – Moving Average

ARIMA – Autoregressive Integrated Moving Average

SARIMA – Seasonal Autoregressive Integrated Moving Average

AIC – Akaike Information Criterion

BIC – Bayesian Information Criterion

2 Data

2.1 Getting the Dataset

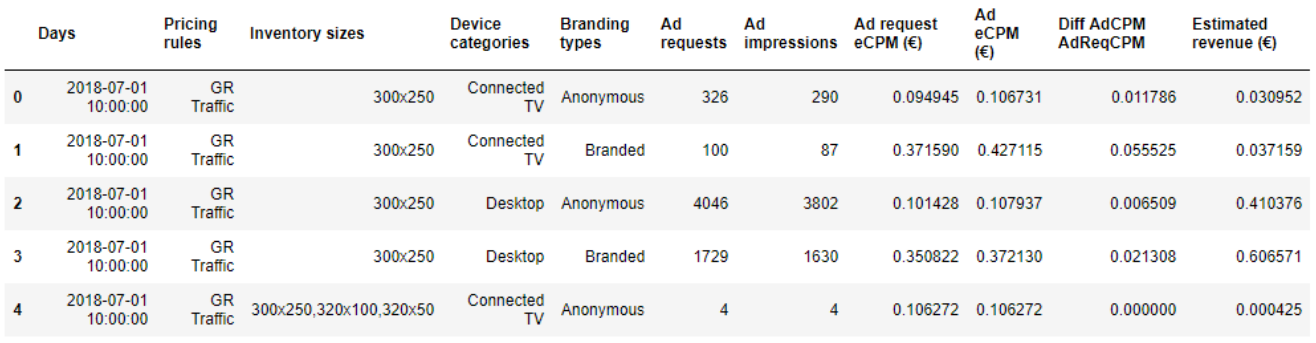
The required dataset was, at first, generated by using Google Ad Manager reporting tool where reports have been generated manually in Microsoft Excel sheets for each website that would be examined. After the finalization of this developer tool, the extraction of each dataset is going to be programmatically generated through the API of Google Ad Manager reporting tool.

Though the acquisition of the dataset came straight from the vendor's tool, the selection of the dimensions and measures that would successfully provide a well-formed and useful piece of data was rather difficult.

2.2 Describing the Dataset

Each row of the dataset consisted, as shown in Table 1, of four categorical fields (*Pricing rules, Inventory sizes, Device categories, and Branding types*) which represent the features, a date-time field which will be used later in the procedure as an index field and five numerical fields which represent the measures of each row.

Table 1: Data Sample before transformations



	Days	Pricing rules	Inventory sizes	Device categories	Branding types	Ad requests	Ad impressions	Ad request eCPM (€)	Ad eCPM (€)	Diff AdCPM AdReqCPM	Estimated revenue (€)
0	2018-07-01 10:00:00	GR Traffic	300x250	Connected TV	Anonymous	326	290	0.094945	0.106731	0.011786	0.030952
1	2018-07-01 10:00:00	GR Traffic	300x250	Connected TV	Branded	100	87	0.371590	0.427115	0.055525	0.037159
2	2018-07-01 10:00:00	GR Traffic	300x250	Desktop	Anonymous	4046	3802	0.101428	0.107937	0.006509	0.410376
3	2018-07-01 10:00:00	GR Traffic	300x250	Desktop	Branded	1729	1630	0.350822	0.372130	0.021308	0.606571
4	2018-07-01 10:00:00	GR Traffic	300x250,320x100,320x50	Connected TV	Anonymous	4	4	0.106272	0.106272	0.000000	0.000425

Table 1 displays a raw sample of the data set provided from the reporting tool of Google Ad Manager before applying any transformation to the original data. It's worth mentioning that the information of the floor value of each pricing rule contained in the data set was obtained by a historical record of the website that was analyzed which was held in an external spreadsheet.

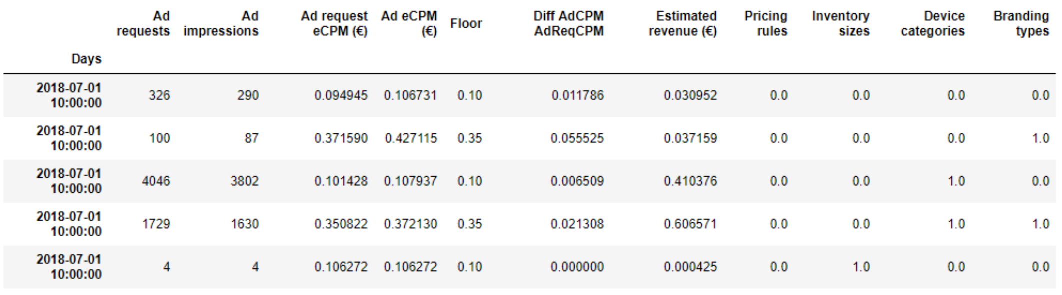
This difficulty was introduced by Google Ad Manager tool, which doesn't support the historical tracking of these values, but it can only provide the latest value used.

To support the proposed procedure, there has been an informal sub-step between the first step of importing data and the second step of indexing the categorical features, according to which the fields containing the information for date, pricing rule name and floor price value are matched between the two data frames.

In the indexing step of the procedure, the original data are transformed by an ordinal encoder transformation function, which is implemented in the Scikit–Learn toolkit (Scikit-Learn, 2018), to their numerical values respectively. This estimator transforms each categorical feature to one new feature of integer values starting from 0 to the number of distinct categories – 1. Such an integer representation can be used to convert categorical features to integer codes because their ordering is irrelevant to the information that they provide.

Table 2 shows a sample of the data set after the OrdinalEncoder transformation took place.

Table 2: Transformed sample of the data set through OrdinalEncoder



	Ad requests	Ad impressions	Ad request eCPM (€)	Ad eCPM (€)	Floor	Diff AdCPM AdReqCPM	Estimated revenue (€)	Pricing rules	Inventory sizes	Device categories	Branding types
Days											
2018-07-01 10:00:00	326	290	0.094945	0.106731	0.10	0.011786	0.030952	0.0	0.0	0.0	0.0
2018-07-01 10:00:00	100	87	0.371590	0.427115	0.35	0.055525	0.037159	0.0	0.0	0.0	1.0
2018-07-01 10:00:00	4046	3802	0.101428	0.107937	0.10	0.006509	0.410376	0.0	0.0	1.0	0.0
2018-07-01 10:00:00	1729	1630	0.350822	0.372130	0.35	0.021308	0.606571	0.0	0.0	1.0	1.0
2018-07-01 10:00:00	4	4	0.106272	0.106272	0.10	0.000000	0.000425	0.0	1.0	0.0	0.0

In order to get an insight of the given data set, the third step is dedicated to visualize the data distribution over all the features, categorical and numerical, against the Ad eCPM values.

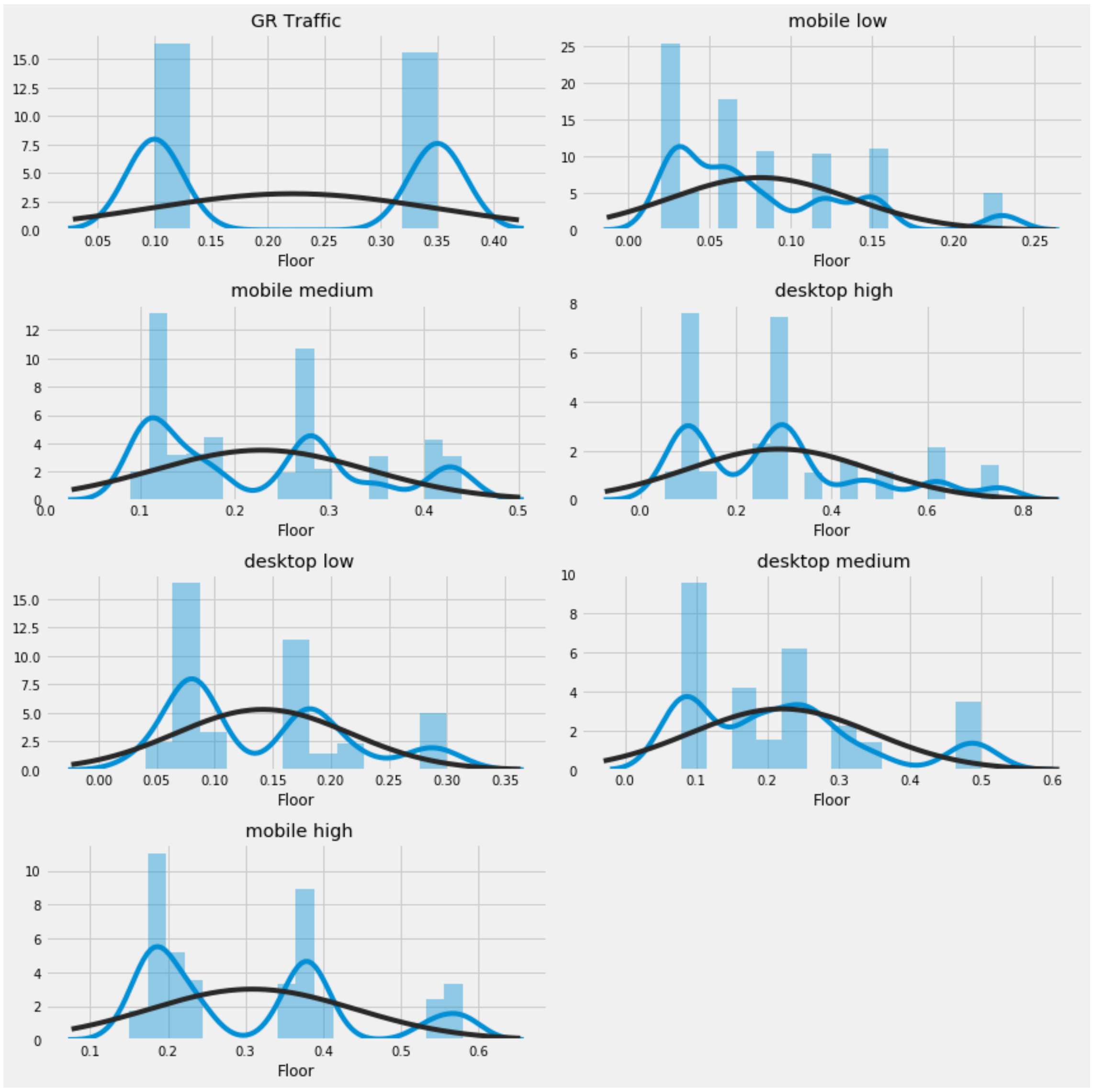


Figure 2: Floor pricing value distribution (blue line) with histogram aside with normal distribution (black line)

Figure 2 represents the way that floor price values are distributed in each of the pricing rules specified for a certain website.

The following three figures give a detailed overview of how eCPM is distributed against the categorical features used in the data set.

Figure 3 shows that most of the generally well-paid ads are promoted by branding type '*Branded*' which is the most profitable of them regarding revenue.

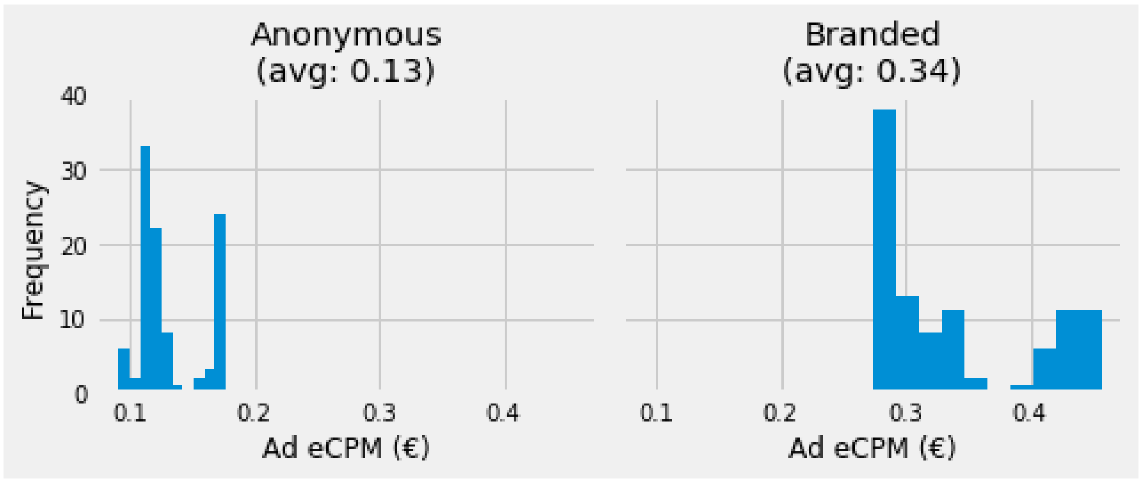


Figure 3: Branding Type histogram

Figure 4 represents the statistical analysis of the Device categories field of the data set, which shows that the best paid in average Ad eCPM are the ads that were displayed on a Desktop device while High-end mobile devices and Tablets share a rather large number of impressions.

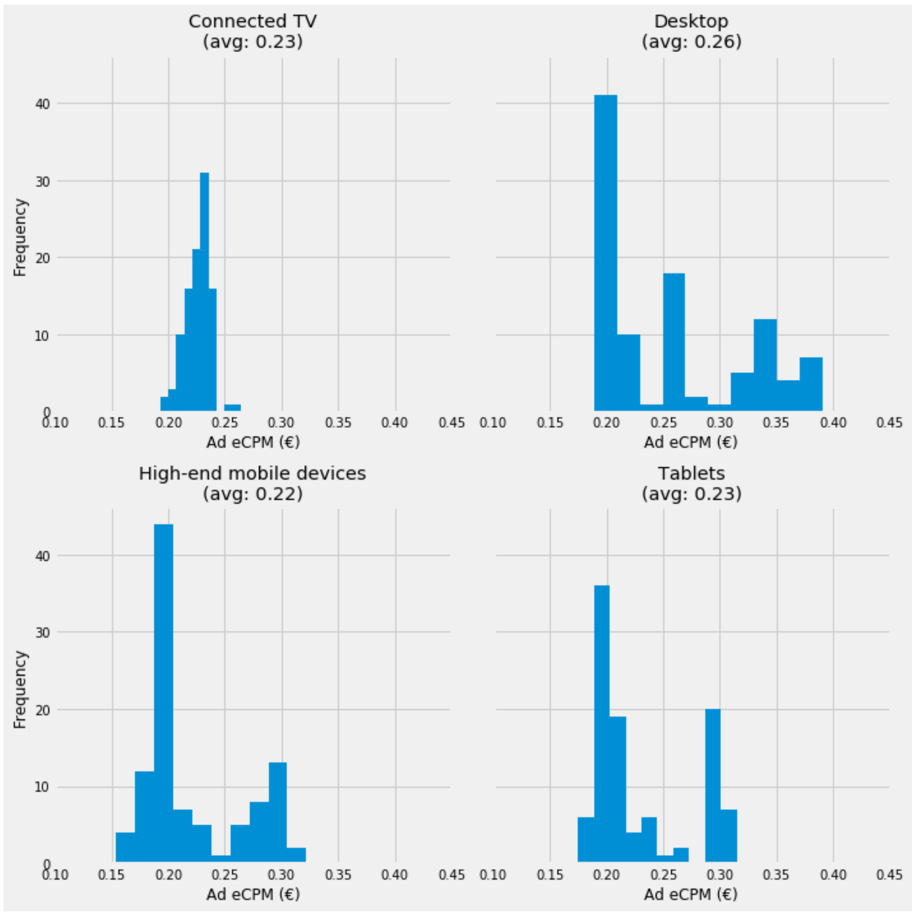


Figure 4:Device Categories histogram

Moreover,

Figure 5 gives a detailed visual estimation of the importance of each inventory size and how it is being distributed to the different values of Ad eCPM. In the example presented, the most profitable size is ‘*300x250,320x100,320x50*’ is presented with an average value of *0.28*, even though some of these sizes take part in other categories as well with lower average values.

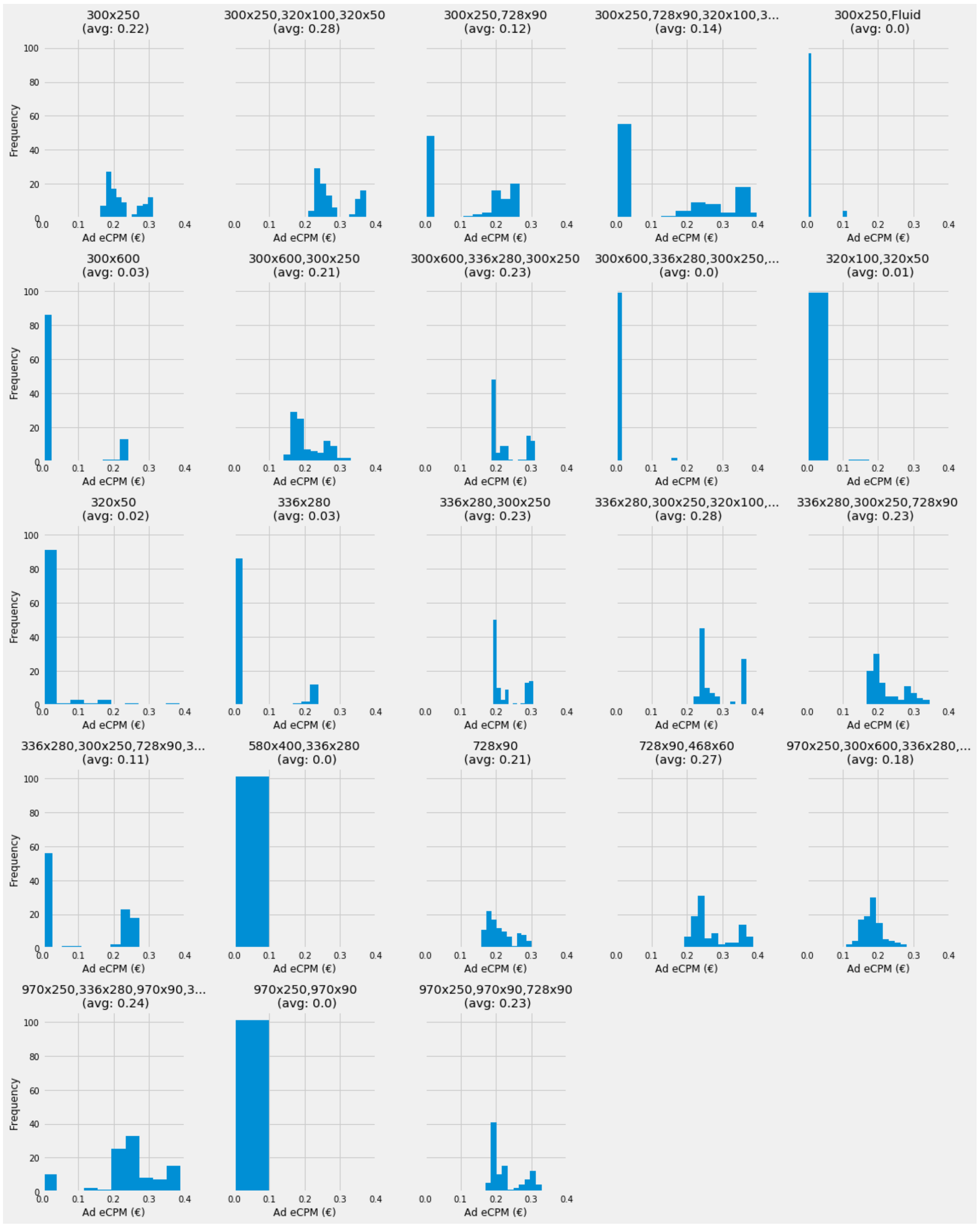


Figure 5: InventorySize Histogram

2.3 Categorical feature analysis

Supporting the observation that all the features are categorical, such as Pricing rules, Inventory sizes, Device Categories, and Branding types the proposed procedure indexes these features to their ordinal representation through the appropriate transformation and gives an integer value respectively. The rest data in the data set are numerical values that can participate in our modeling process as they are. Thus there is no need for extra preprocessing of the data.

In Figure 6 the categorical indexing is being displayed with the discrete values on the x-axis vs. the different features on the y-axis and the different pricing rules as the color coding.

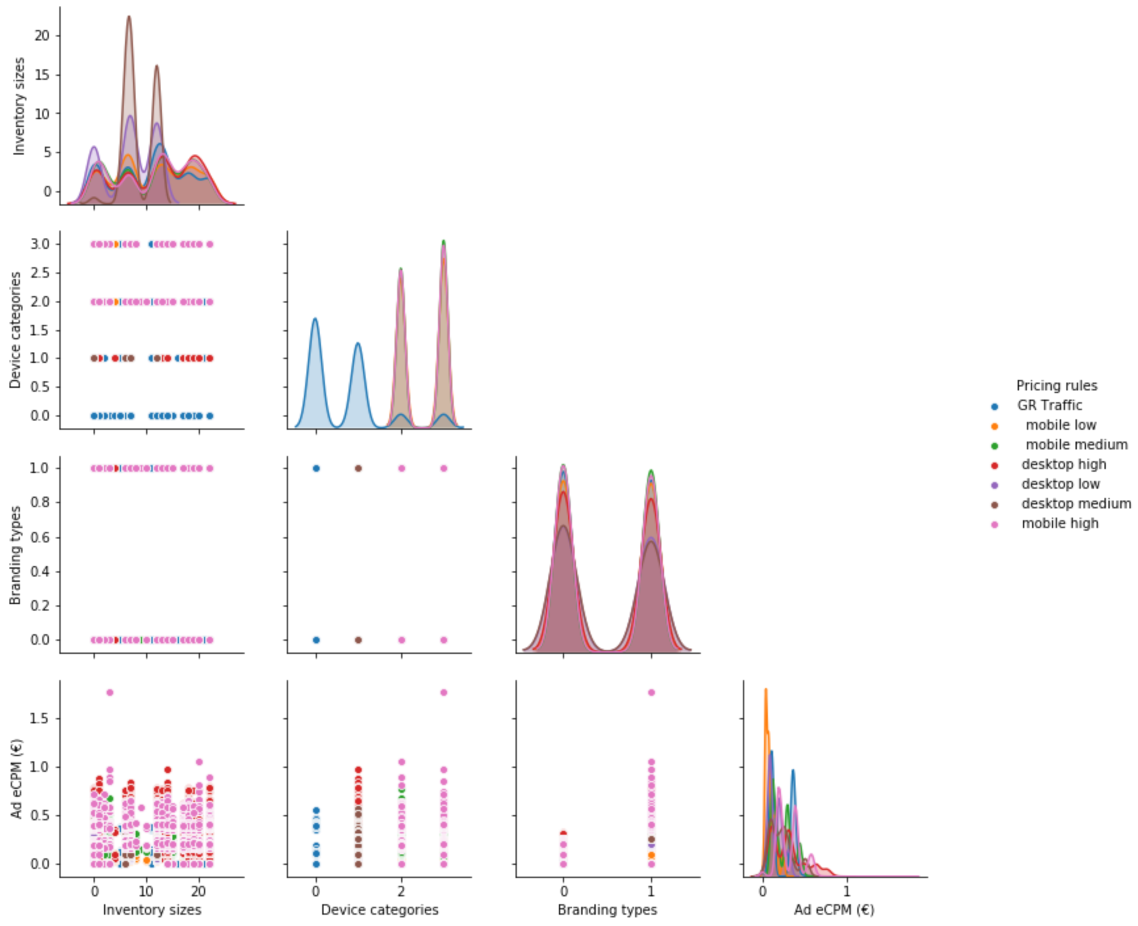


Figure 6: Categorical features distribution against Ad eCPM

2.4 Numerical measures analysis

In contrast with the categorical features, the numerical measures are imported as provided by the Google Ad Manager reporting tool.

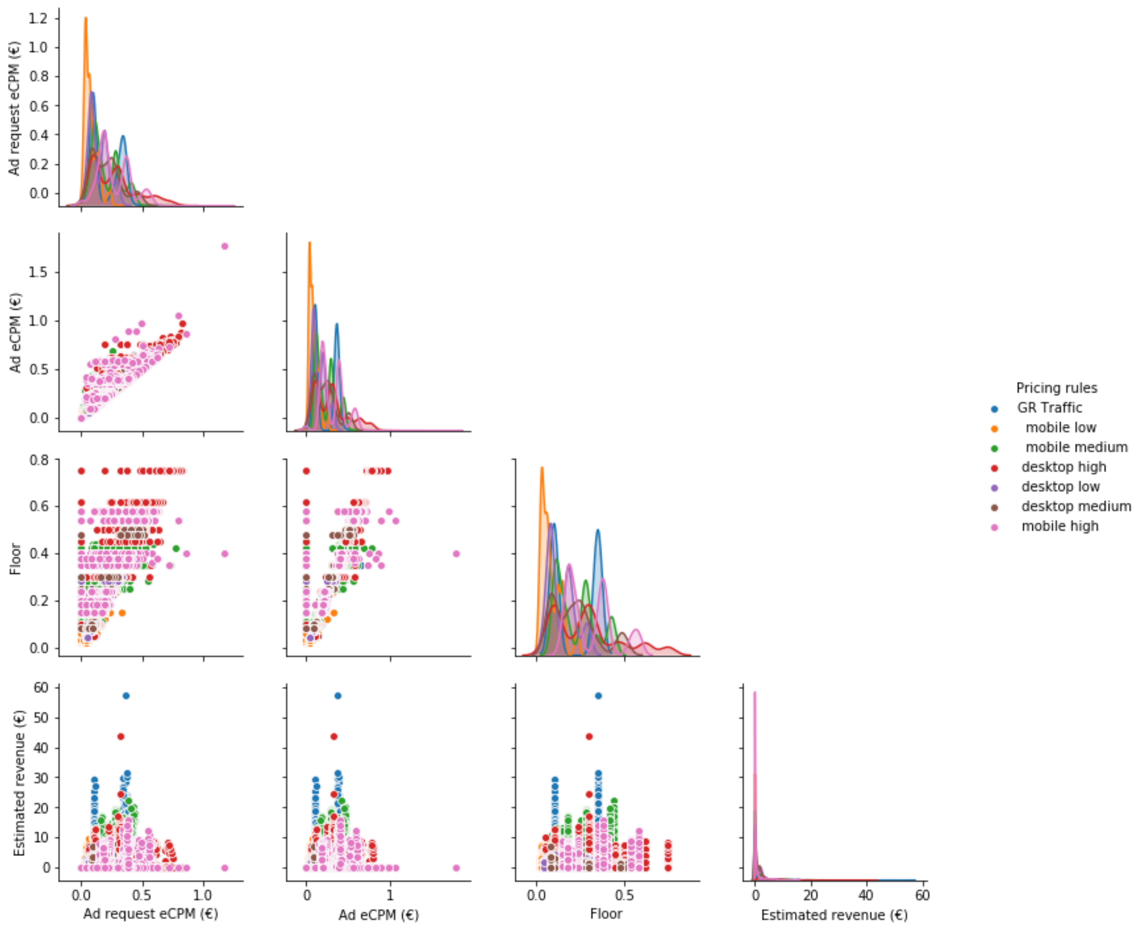


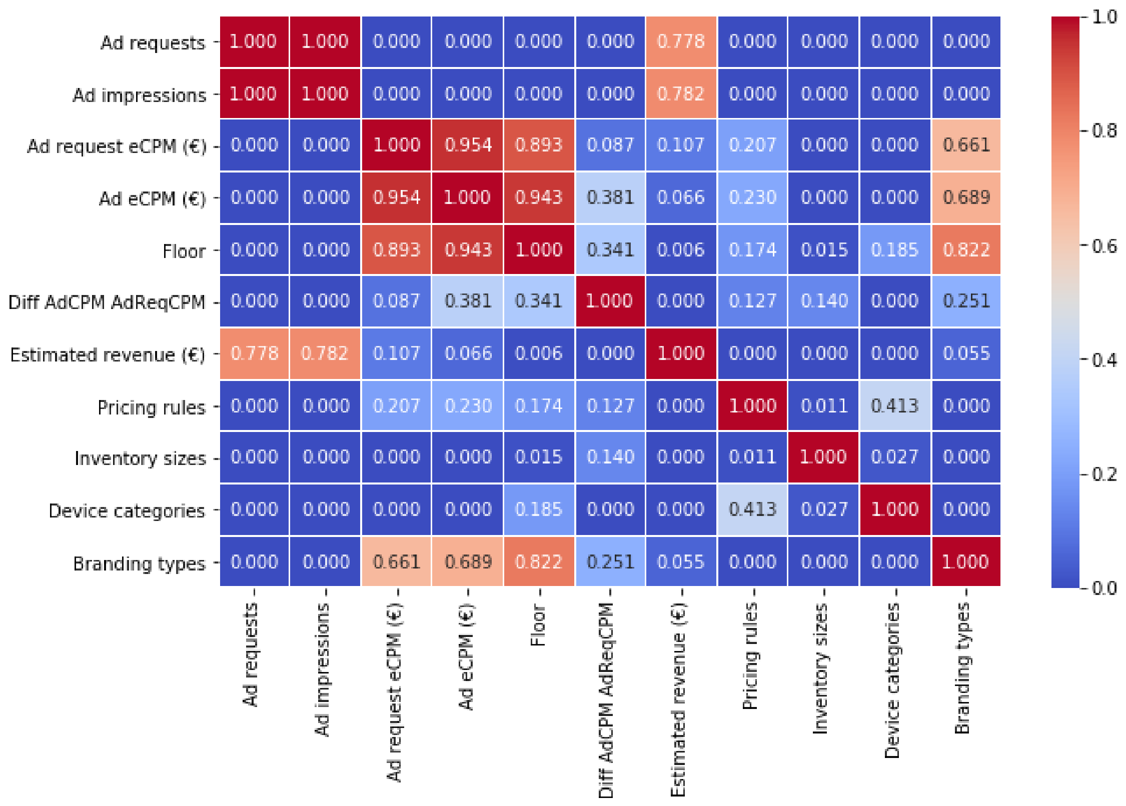
Figure 7: Numerical features distribution
(*Ad eCPM*, *Ad request eCPM*, *Estimated revenue*, and *Floor values*)

Although it is mandatory to scale all continuous numeric input features so that not a single feature influences the model performance, we chose not to standardize our data set because the only features that could interfere with our modeling estimation are Ad requests and Ad Impressions.

Moreover, after the step of categorical features encoding, the data set visualization provides the capability to extract linearity between features to combine them in new ones properly. In this use case, the procedure showed a linear correlation between *Ad Requests – Revenue* and *Ad Impressions – Revenue* which can be explained because the Revenue value results from the multiplication of Ad Impressions and a value called *Close CPM* which depicts the actual payable value of each impression. This multiplication gives an estimate of the revenue gained which on average is provided by Ad eCPM.

Also, the Ad Impressions measure can be described by a factor of Ad Request's value. This linearity between Ad requests, Ad Impressions, and Estimated revenue also denotes that these measures won't provide any information gain to the modeling procedure and their values will affect the revenue directly and not the Ad eCPM value.

As we can see both in Figure 7 and Figure 8, there is a strong statistical relationship which is measured by Pearson correlation factor (0.943 or 94.3%) between Ad eCPM and Floor value of each pricing rule which is the key property of our research.



	Ad requests	Ad impressions	Ad request eCPM (€)	Ad eCPM (€)	Floor	Diff AdCPM AdReqCPM	Estimated revenue (€)	Pricing rules	Inventory sizes	Device categories	Branding types
Ad requests	1.000	1.000	0.000	0.000	0.000	0.000	0.778	0.000	0.000	0.000	0.000
Ad impressions	1.000	1.000	0.000	0.000	0.000	0.000	0.782	0.000	0.000	0.000	0.000
Ad request eCPM (€)	0.000	0.000	1.000	0.954	0.893	0.087	0.107	0.207	0.000	0.000	0.661
Ad eCPM (€)	0.000	0.000	0.954	1.000	0.943	0.381	0.066	0.230	0.000	0.000	0.689
Floor	0.000	0.000	0.893	0.943	1.000	0.341	0.006	0.174	0.015	0.185	0.822
Diff AdCPM AdReqCPM	0.000	0.000	0.087	0.381	0.341	1.000	0.000	0.127	0.140	0.000	0.251
Estimated revenue (€)	0.778	0.782	0.107	0.066	0.006	0.000	1.000	0.000	0.000	0.000	0.055
Pricing rules	0.000	0.000	0.207	0.230	0.174	0.127	0.000	1.000	0.011	0.413	0.000
Inventory sizes	0.000	0.000	0.000	0.000	0.015	0.140	0.000	0.011	1.000	0.027	0.000
Device categories	0.000	0.000	0.000	0.000	0.185	0.000	0.000	0.413	0.027	1.000	0.000
Branding types	0.000	0.000	0.661	0.689	0.822	0.251	0.055	0.000	0.000	0.000	1.000

Figure 8: Correlation Diagram against all dimensions

Based on this working hypothesis this algorithm provides a forecasting model of the Ad eCPM against other dimensions introduced by our data set, to achieve increased revenue with the appropriate adjustment of the Floor value of each pricing rule on a per-site basis.

3 Methodology

3.1 Description

As a starting step on the modeling phase of this algorithmic procedure, a correlation matrix – diagram as shown in Figure 9 is created, in order to model the Ad eCPM behavior. This matrix consists of Pearson correlation factor values against the information gained about Ad eCPM values.

Subsequently, feature sets that have strong correlation factors over the three major categories: Branding types – Device Categories – Inventory Sizes are defined. Each set consists of a tuple of three integer values, e.g. (1, 1, 7) which indicates that there is a strong relationship among Branded type equal to 1 – ‘*Branded*’, Device category equals to 1 – ‘*Desktop*’ and Inventory size equals 7 – ‘*300x600, 336x280, 300x250*’.

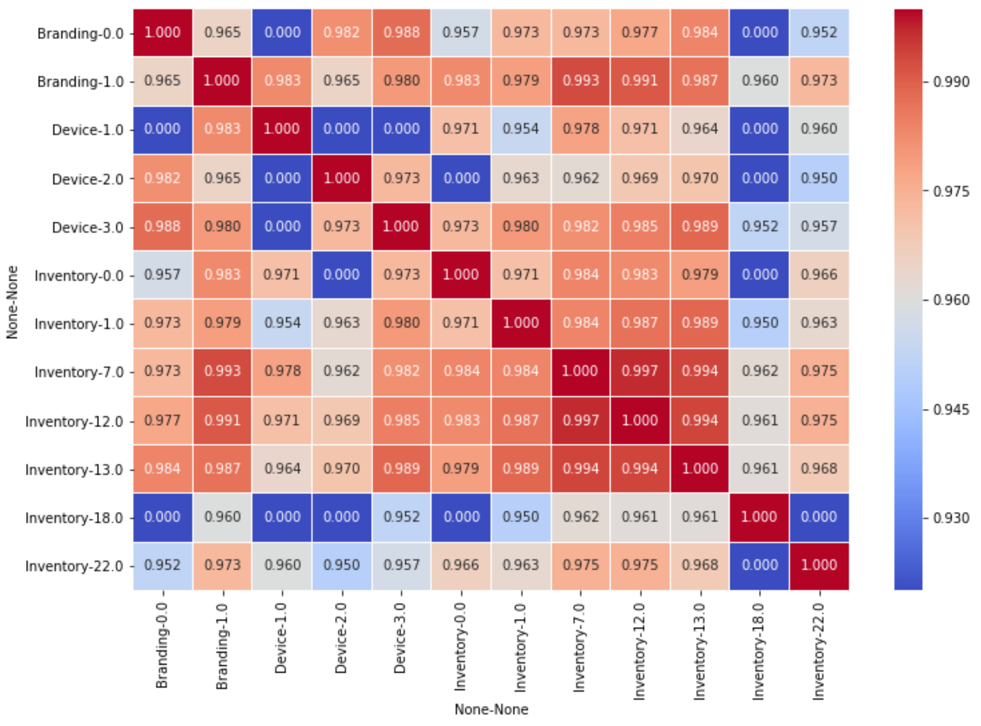


Figure 9: Correlation diagram based on average Ad eCPM values against all dimensions of (Branding type, Device Category, Inventory Size)

Each of these sets underwent to a testing phase of different modeling schemes so that the best model fitting would be acquired.

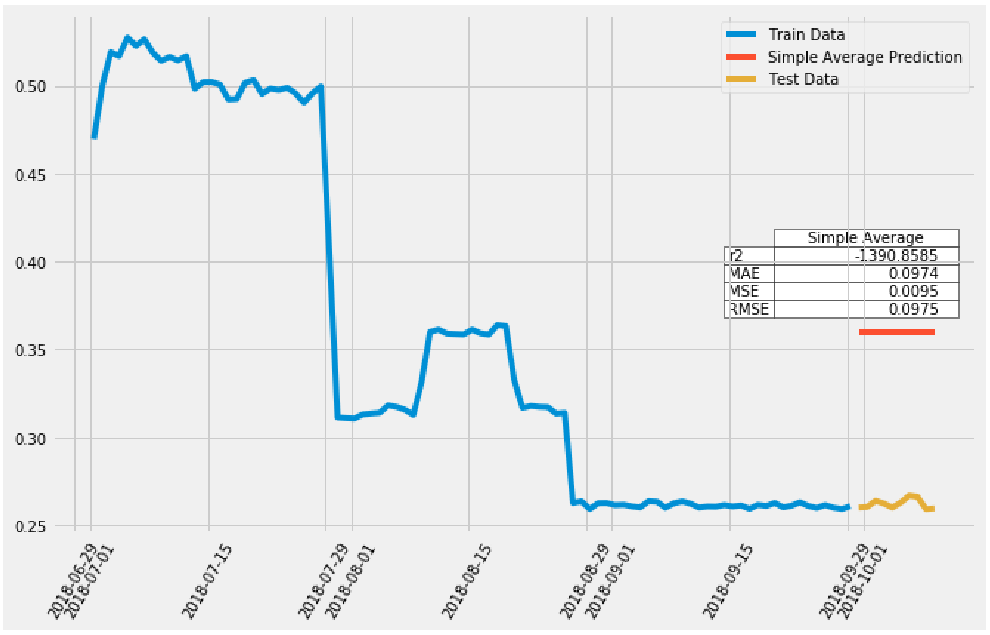
3.2 Model Specification

To achieve the best fitting for the proposed model to the specified data set, five different modeling approaches were used, starting from a naïve hypothesis as our baseline model to more sophisticated models created by ARIMA and Seasonal ARIMA processes.

3.2.1 Naive hypothesis – Simple Average

A model like $\hat{y}_t = y_{t-1}$ is a great baseline for any time series prediction, as it relies to just the previous value of our data series and thus the error that will be introduced is rather small. Moreover, financial related series are likely to depend on the previous time period value and as forecasting steps increase, the error margin increase too. To overcome this problem, we, assumed that the future value of our variable depends on the average of its k previous values and k equals to the length of the time series. Such forecasting technique which forecasts the expected value equal to the average of all previously observed points is called Simple Average technique.

$$\hat{y}_t = \frac{1}{k}\sum_{n=1}^{k} y_{t-n}$$



	Simple Average
r2	-1390.8585
MAE	0.0974
MSE	0.0095
RMSE	0.0975

Figure 10: Simple average modeling

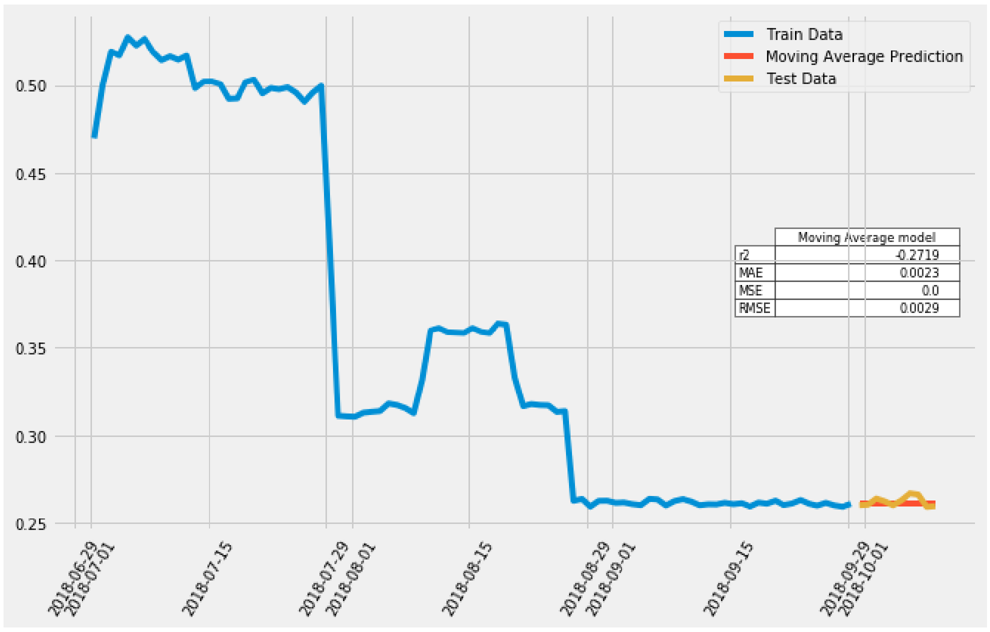
Although this approach gives a prediction with a reasonable error for the first forecasting step, most of the time, if it is used to forecast more than one step ahead it will lead to a rather large amount of errors in the last forecasting value.

3.2.2 Moving average

To provide a more accurate forecasting value, we used a simple moving average model, which forecasts the next value(s) in a time series based on the average of a fixed finite number p of the previous values. Thus, for all $i > p$

$$\hat{y}_t = \frac{1}{p} \sum_{n=1}^{p} y_{t-n}$$

This approach also incorporates the seasonal flow of the values of the time series, but again it only gives a rather rough prediction when we increment the forecasting steps.



	Moving Average model
r2	-0.2719
MAE	0.0023
MSE	0.0
RMSE	0.0029

Figure 11: Moving average example modeling

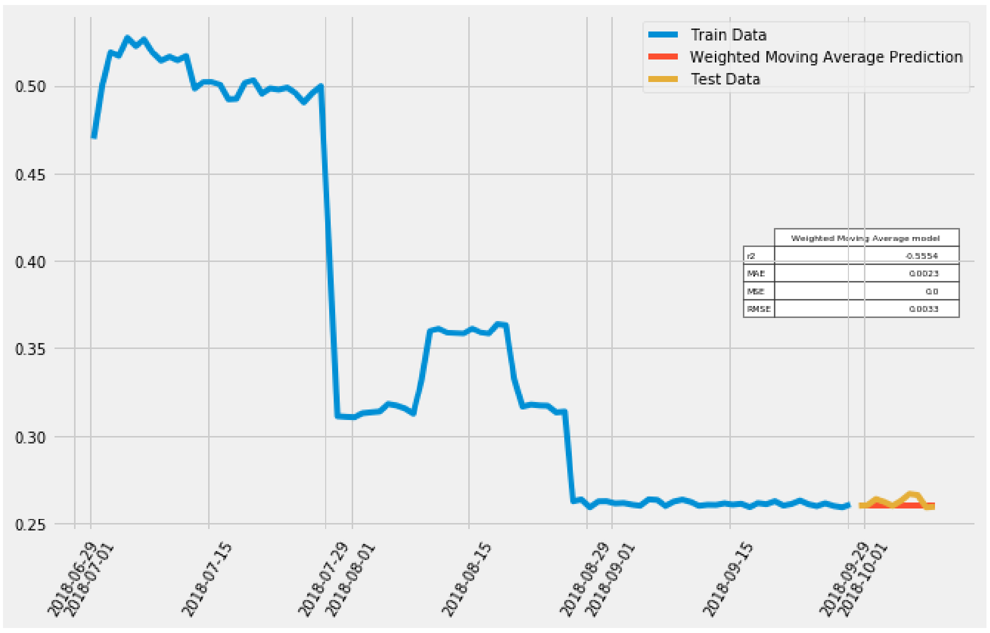
As shown in the above Figure 11 the moving average modeling method fits better the training dataset because it incorporates the seasonal trend of the data. The basic weakness of this modeling approach is that if the train data set has few data points; the error produced increments rapidly as forecasting steps increase.

3.2.3 Weighted moving average

As a next modeling approach, we chose a weighted moving average model which uses for the forecasting step value a different way of weighting the past observations and sum up to the weight value to one. The larger weights often assigned to the more recent observations promoting the corresponding values.

$$\hat{y}_t = \sum_{n=1}^{k} w_n y_{t+1-n}$$

In the above equation w_n are the weights for each of the previous values and k is the number of them to consider in the sum.



	Weighted Moving Average model
r2	-0.5554
MAE	0.0023
MSE	0.0
RMSE	0.0033

Figure 12: Weighted moving average example modeling

This modeling approach has the advantage that if the autocorrelation factor of the data set to be fit depends over the k lag values, the predicted values approximate better the testing data set. Given this attribute, this approach also has its weakness because the weights array has a finite number of elements.

3.2.4 ARIMA (p, d, q)

Autoregressive Integrated Moving Average (ARIMA) models consist of 3 basic steps. The Auto regression part AR (p) where p is the order of the AR model, the Integration I (d) part and the Moving average MA (q) part. While exponential smoothing models were based on a description of trend and seasonality in the data, ARIMA models aim to describe the correlations in the data with each other.

$$\Phi(B)(1-B)^d y_t = c + \theta(B)\varepsilon_t$$

Whereε_t is a white noise process with mean zero and varianceσ^2, B is the backshift operator, and $\Phi(z)$ and $\theta(z)$ are polynomials of order p and q respectively.If$c \neq 0$ ther e is an implied polynomial of order $d$in the forecast function.

Figure 13 shows exactly the model predictions over the train data and the accuracy of each ARIMA model of order p and q respectively.

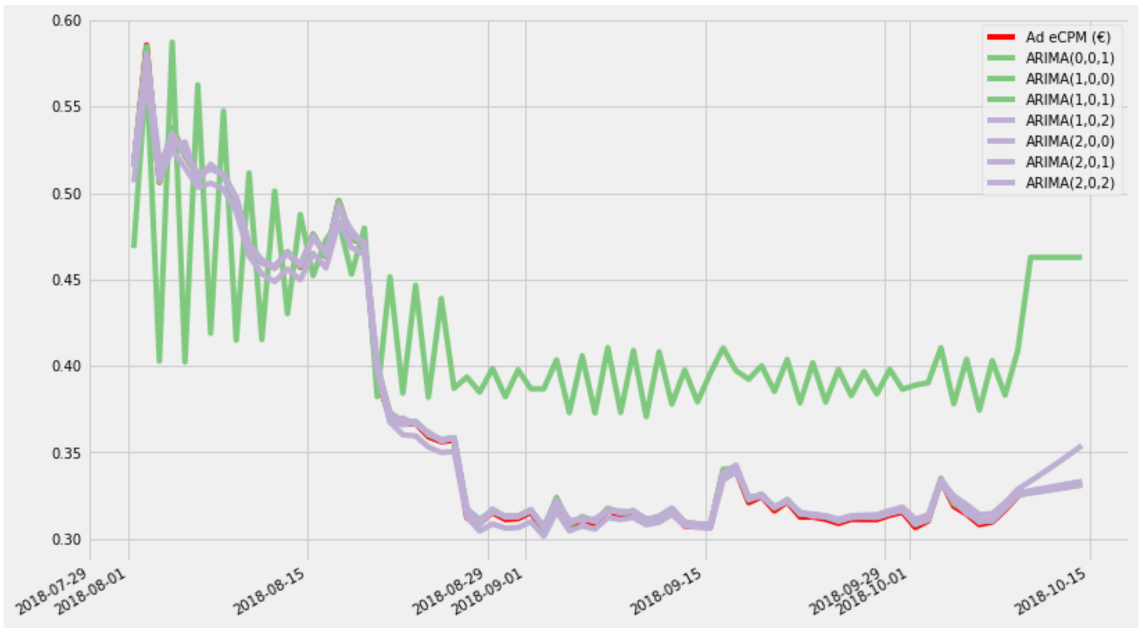


Figure 13: Different parameter values in ARIMA (p, d, q) along with the original data

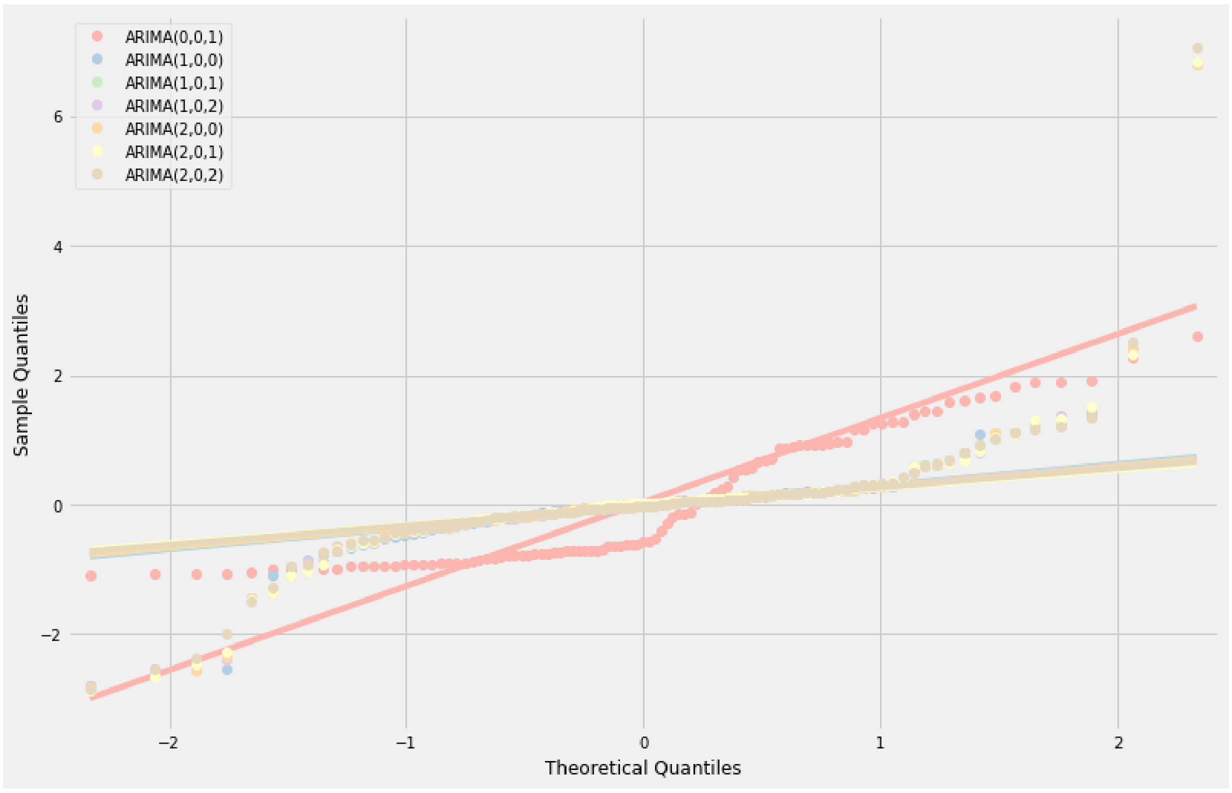


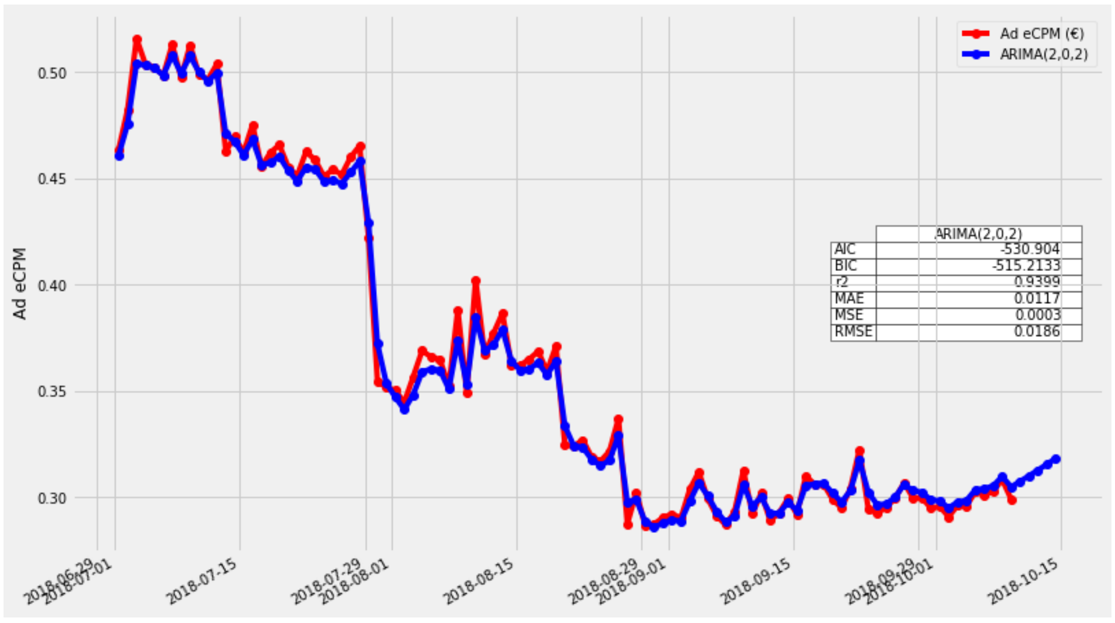
Figure 14: Quantiles convergence plots

In addition, in Figure 14 we present the plots that provide us information about the approximated values for the ARIMA modeling approach as described by *Algorithm 1*.

All of the above modeling schemes are all examined through statistical measures of error and information retrieval. One of them is the Mean Square Error rate (MSE), and another is the r^2 measure which provides the metric of the accuracy for the examined model.

The Akaike information criterion (AIC) (Bozdogan, 1987) is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection. The advantages are that it is valid for both nested and non-nested models, it can compare models with different error distribution and finally, it can avoid multiple testing issues.

Some weakness of the AIC is that it cannot be used to compare models of different data sets. Thus, the selected model with the lowest AIC is only valid and better than another for the specific data set.



	ARIMA(2,0,2)
AIC	-530.904
BIC	-515.2133
r2	0.9399
MAE	0.0117
MSE	0.0003
RMSE	0.0186

Figure 15 ARIMA Model fitting

As an example of the ARIMA model fitting Figure 15 show the resulting visualization for the predicted data denoted by the blue line against the original data set. As an extension of the information that describes best the model Table 3 presents thoroughly all the parameters required to define this model fitting.

Table 3: ARIMA (2, 0, 2) model summary

```
====================================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------
const                  0.3418      0.017     19.827      0.000       0.308       0.376
ar.L1.Ad eCPM (€)      1.9554      0.024     80.301      0.000       1.908       2.003
ar.L2.Ad eCPM (€)     -0.9576      0.025    -38.471      0.000      -1.006      -0.909
ma.L1.Ad eCPM (€)     -0.6248      0.096     -6.480      0.000      -0.814      -0.436
ma.L2.Ad eCPM (€)     -0.3752      0.093     -4.014      0.000      -0.558      -0.192
------------------------------------------------------------------------------------
```

3.2.5 Seasonal ARIMA (SARIMA)

An improvement over ARIMA is Seasonal ARIMA which takes into account the seasonality of dataset. As shown in Figure 16 the original data are decomposed into a series of trend and seasonality. This decomposition is required to determine the seasonality factor which will be used by the equation of the appropriate model.

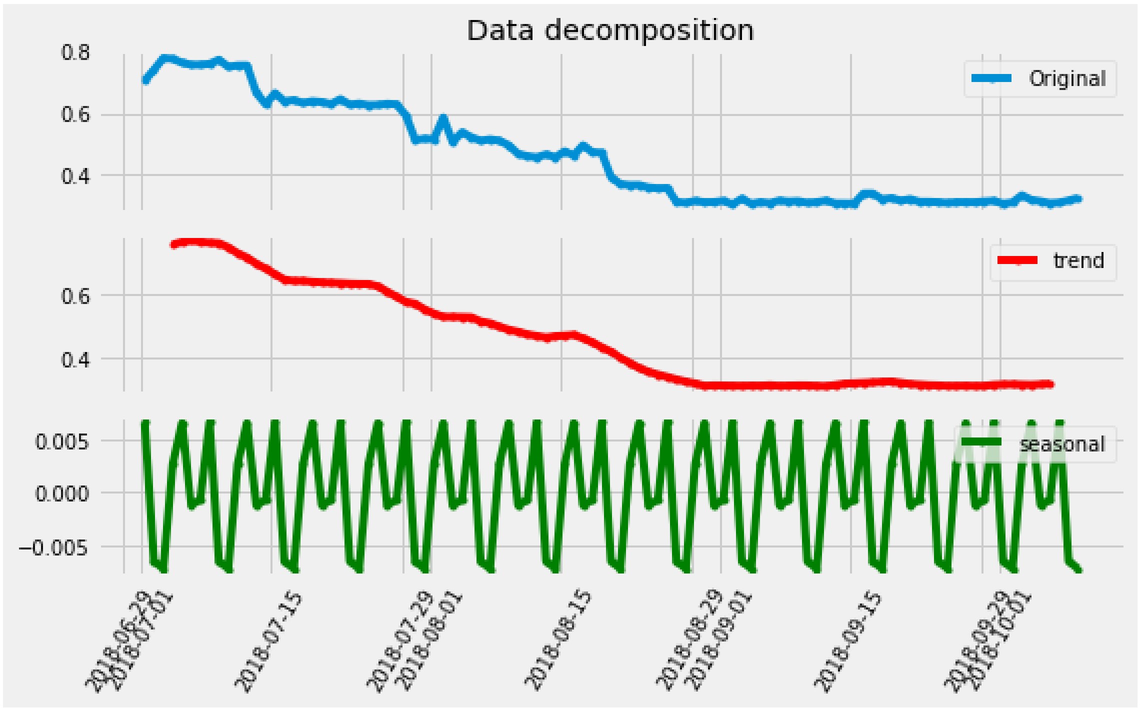


Figure 16: Data decomposition showing trend and seasonality

First of all, this class of models was introduced by Box and Jenkins (1976) and offers a good representation of many seasonal series that we find in practice and in simplified form is written as the ARIMA model $(P, D, Q)_m \times (p, d, q)$ where the period of the seasonal series is$m$.

$$\Phi(B^m)\varphi(B)(1 - B^m)^D(1 - B)^d y_t = c + \Theta(B^m)\theta(B)\varepsilon_t$$

$\Phi(z)$ and $\Theta(z)$ are polynomials, each containing no roots inside the unit circle, representing the seasonal AR operator of order P and the seasonal moving average MA operator of order Q respectively. If$c \neq 0$ the implied polynomial is of order $d + D$ in the forecasting function.

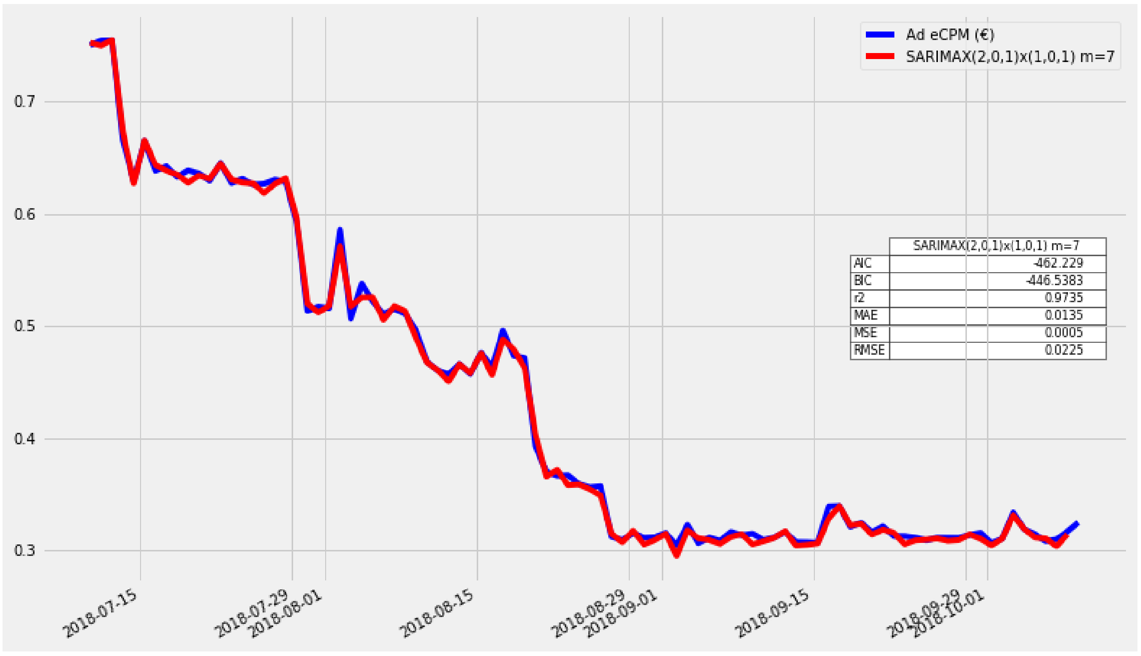
ARIMA and SARIMA forecasting is selecting an appropriate model order, that is the valuesp, q, P, Q, D, d. If d and D are known, we can select the orders p, q, P and Q via an information criterion such as theAIC:

$$AIC = -2 \, log(L) + 2(p + q + P + Q + k)$$

Where$k = 1$ if $c \neq 0$ and 0 otherwise, and L is the maximized likelihood of the model fitted to the differenced data$(1 - B^m)^D (1 - B)^d y_t$.

As already mentioned, the goal again is to select the model that minimizes the AIC amongst all of the models that are appropriate for the data.

The AIC (K.P. Burnham, 2004) also provides a method for selecting between the additive and multiplicative error models. The point forecasts from the two models are identical so that standard forecast accuracy measures such as the MSE or mean absolute percentage error (MAPE) are unable to select between the error types. The AIC is able to select between the error types because it is based on likelihood rather than one-step forecasts.



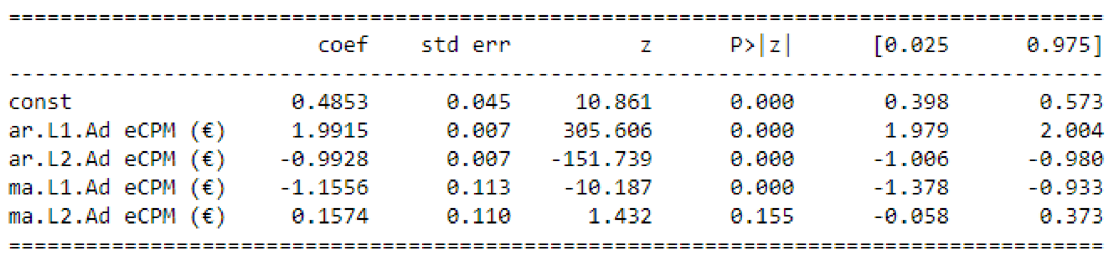
	SARIMAX(2,0,1)x(1,0,1) m=7
AIC	-462.229
BIC	-446.5383
r2	0.9735
MAE	0.0135
MSE	0.0005
RMSE	0.0225

Figure 17 SARIMA model fitting

Again as an example of the SARIMA process of model fitting, Figure 17 visualizes the way that predictions are fit over the original data set, while

Table 4 describe the coefficients of the model function.

Table 4: Model coefficients description table



```
=====================================================================================
                          coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
const                   0.4853      0.045     10.861      0.000       0.398       0.573
ar.L1.Ad eCPM (€)       1.9915      0.007    305.606      0.000       1.979       2.004
ar.L2.Ad eCPM (€)      -0.9928      0.007   -151.739      0.000      -1.006      -0.980
ma.L1.Ad eCPM (€)      -1.1556      0.113    -10.187      0.000      -1.378      -0.933
ma.L2.Ad eCPM (€)       0.1574      0.110      1.432      0.155      -0.058       0.373
=====================================================================================
```

As an example of $\text{SARIMA}(2,0,1)_7(1,0,1)$ the above equations evaluate to:

$$(1 - B^7)^0(1 - B)^0 = 1$$

$$\Phi(B^m) = \Phi(B^7) = 1 - \Phi_1 B^7$$

$$\varphi(B) = 1 - \varphi_1 B$$

$$\Theta(B^m) = \Theta(B^7) = 1$$

$$\theta(B) = 1 + \theta_1 B$$

Forming the final equation of the model which is:

$$\Phi(B^m)\varphi(B)y_t = c + \ \theta(B)\varepsilon_t$$

$$(1 - \Phi_1 B^7)(1 - \varphi_1 B)y_t = c + \ (1 + \theta_1 B)\varepsilon_t$$

$$(1 - \varphi_1 B - \Phi_1 B^7 + \Phi_1\varphi_1 B^8)y_t = c + \ (1 + \theta_1 B)\varepsilon_t$$

$$y_t - \varphi_1 y_{t-1} - \Phi_1 y_{t-7} + \Phi_1\varphi_1 y_{t-8} = c + \ \varepsilon_t + \theta_1\varepsilon_{t-1}$$

$$y_t = \varphi_1 y_{t-1} + \Phi_1 y_{t-7} - \Phi_1\varphi_1 y_{t-8} + c + \ \varepsilon_t + \theta_1\varepsilon_{t-1}$$

And finally, the forecasting function will be:

$$y_{t+1} = \varphi_1 y_t + \Phi_1 y_{t-6} - \Phi_1\varphi_1 y_{t-7} + c + \theta_1\varepsilon_t$$

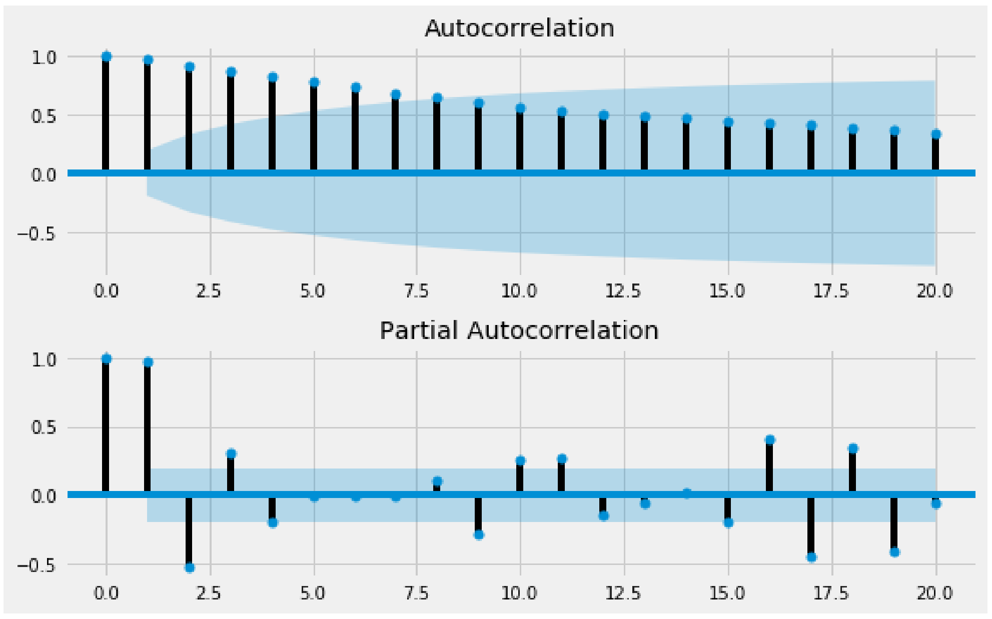


Figure 18: Autocorrelation and partial autocorrelation plotting of the time series original data

Algorithm 1 Determine appropriate model order p, q, P, Q, D, d for SARIMA modeling approach.

For each high correlated feature list of (Branding type BT, Inventory size IS, Device category DC):

Construct Autocorrelation, and partial autocorrelation plots to heuristic determine an approximation of $p_0, q_0, P_0, Q_0, D_0, d_0$

Based on the previous step we calculate the AIC information criterion for $i \epsilon [p_0 - 2, p_0 + 2]$, $k \epsilon [q_0 - 2, q_0 + 2]$, $l \epsilon [P_0 - 2, P_0 + 2]$, $j \epsilon [Q_0 - 2, Q_0 + 2]$ and $D_0, d_0 \epsilon \{0, 1\}$, where $l,\ j \leq p_0$, q_0 respectively.

Finally, we choose p, q, P, Q, D, d that evaluate as $minAIC$ from the derived table.

4 Results

To test the software and the methodology that was created, all models have been applied on two different pricing rules over a specific website and the results of each analysis present revenue increase after providing the forecasted floor pricing values as new rules at Google Ad Manager pricing rules editing tool.

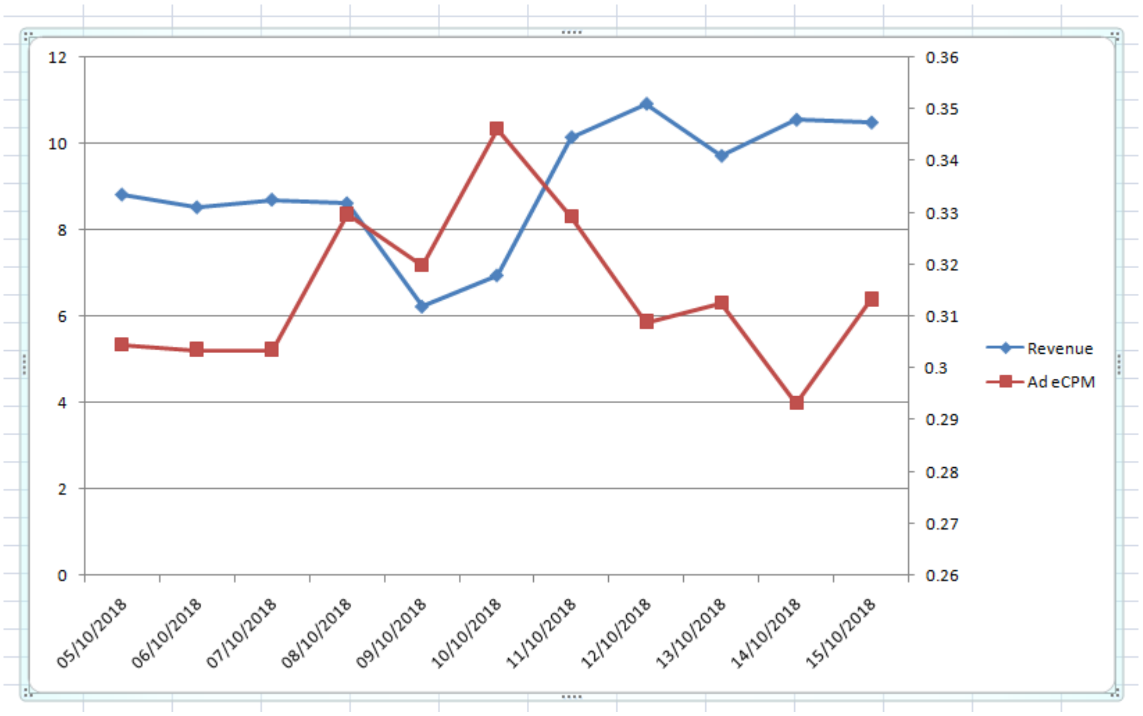


Figure 19: Forecasted values of Ad eCPM and their revenue

Figure 19 shows the revenue increment after applying the forecasted Ad eCPM values for 11 consequent days on the pricing rule floor price. Also, Table 5 presents the revenue uplift estimation for these days.

Table 5: Estimated revenue uplift case 1

Dates	Estimated Revenue
05-10-2018	8.82 €
06-10-2018	8.53 €
07-10-2018	8.70 €
08-10-2018	8.62 €
09-10-2018	6.23 €
10-10-2018	6.96 €
11-10-2018	10.14 €
12-10-2018	10.91 €
13-10-2018	9.71 €
14-10-2018	10.55 €
15-10-2018	10.48 €

This revenue uplift was achieved at first by keeping the number of impressions nearly constant with load balancing of the available traffic of the specific ad units that are being examined. Only 20% of the real traffic appeared at the examined ad units were under the forecasting process.

Moreover, we acquired the resulting forecasted Ad eCPM values by the SARIMA process as depicted in Figure 17. Those values were applied as floor values for the specific pricing rule, and the resulting revenue increment was reported back as feedback from the Google Ad Manager reporting tool. This procedure of course must be at a constant level and should have daily feedback of the forecasting error so as to eliminate the case like Figure 20.

In Figure 20 on the other hand, shows that in this case although the forecasted Ad eCPM values presented uplift on the revenue for six days, the model error rate increased rapidly, and a re-evaluation should be instantiated. This fact led to a decrease in the estimated revenue which after a re-training and evaluation of the new forecasting values showed an increasing trend. Again, the uplift of the estimated revenue was increasing roughly higher than 10% for these six days as shown in Table 6.

Table 6: Estimated revenue uplift case 2

Dates	Estimated Revenue
05-10-2018	9.78 €
06-10-2018	10.28 €
07-10-2018	8.18 €
08-10-2018	7.77 €
09-10-2018	8.76 €
10-10-2018	10.90 €

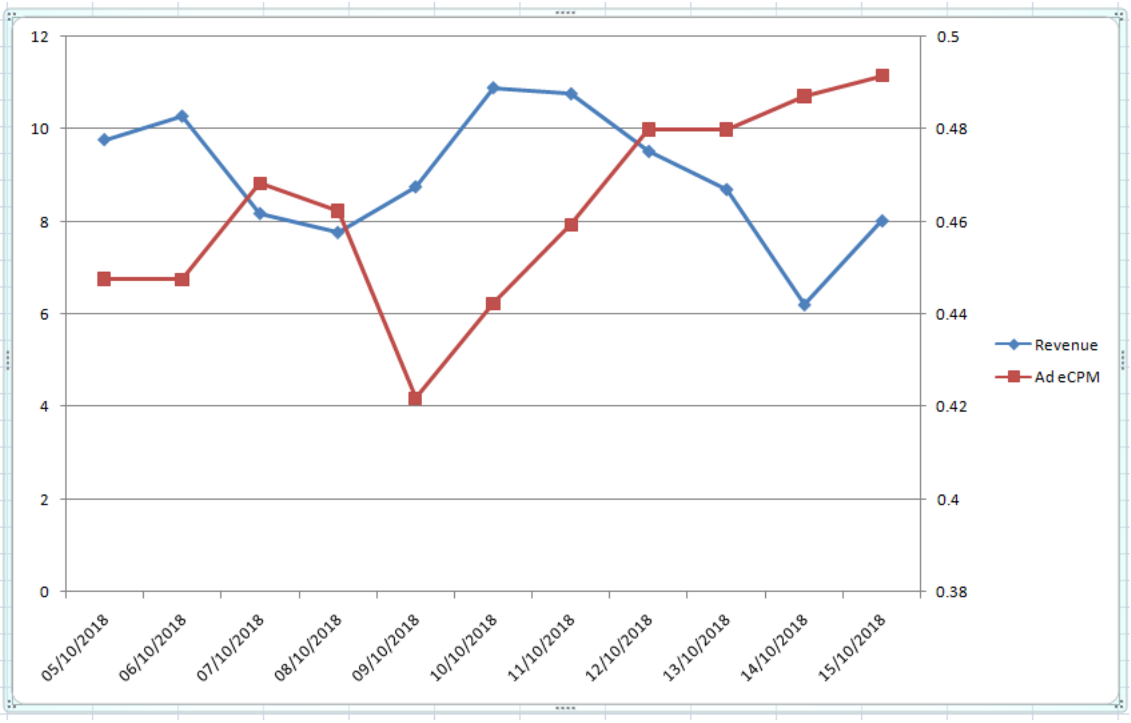


Figure 20: Forecasted values of Ad eCPM and their revenue

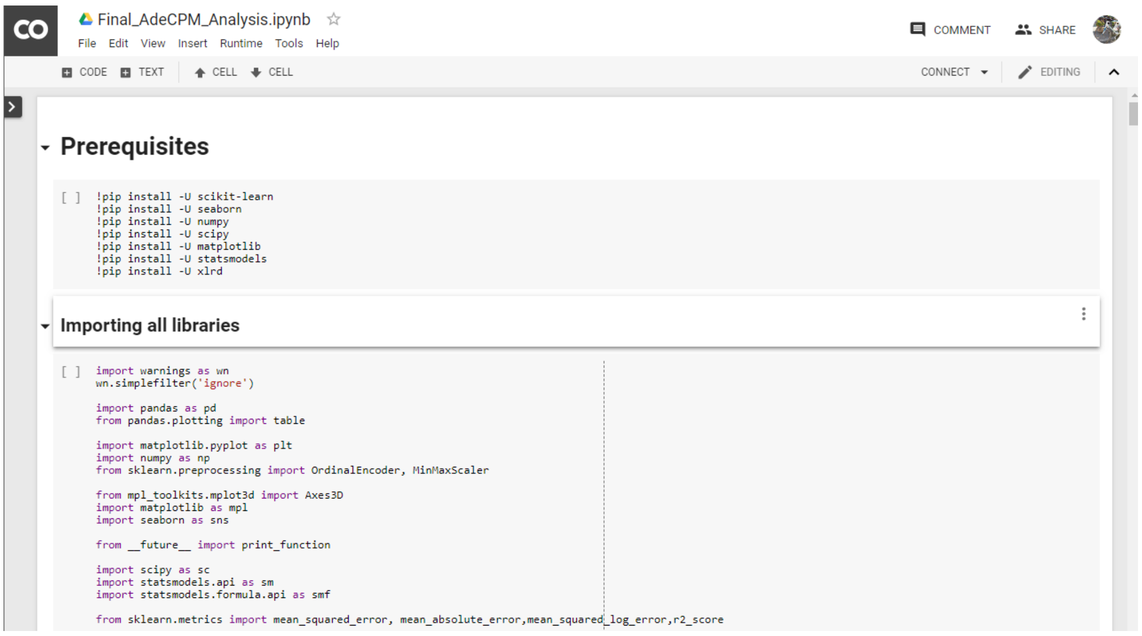
5 Software Usage

The software that has been developed is made on the Python 3.7.0 programming language. It has been used Jupyter notebook server for developing the test modeling approach locally as shown in Figure 22, while the final tool was deployed on a Google Colaboratory notebook.

Jupyter notebook environment installed as a local server can be obtained, is supported and fully documented on (Jupyter notebook environment , 2018) as shown in Figure 23.

Google Collaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud as displayed in Figure 21.

The advantage of Collaboratory over Jupyter notebook local server is that any machine learning and neural network algorithms can be accelerated using Google's Cloud computing engine and by using powerful graphic card GPUs or TensorFlow Processing Units - TPUs while the local server is limited to the computational resources provided by the personal computer.


CO
Final_AdeCPM_Analysis.ipynb
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
CONNECT
EDITING

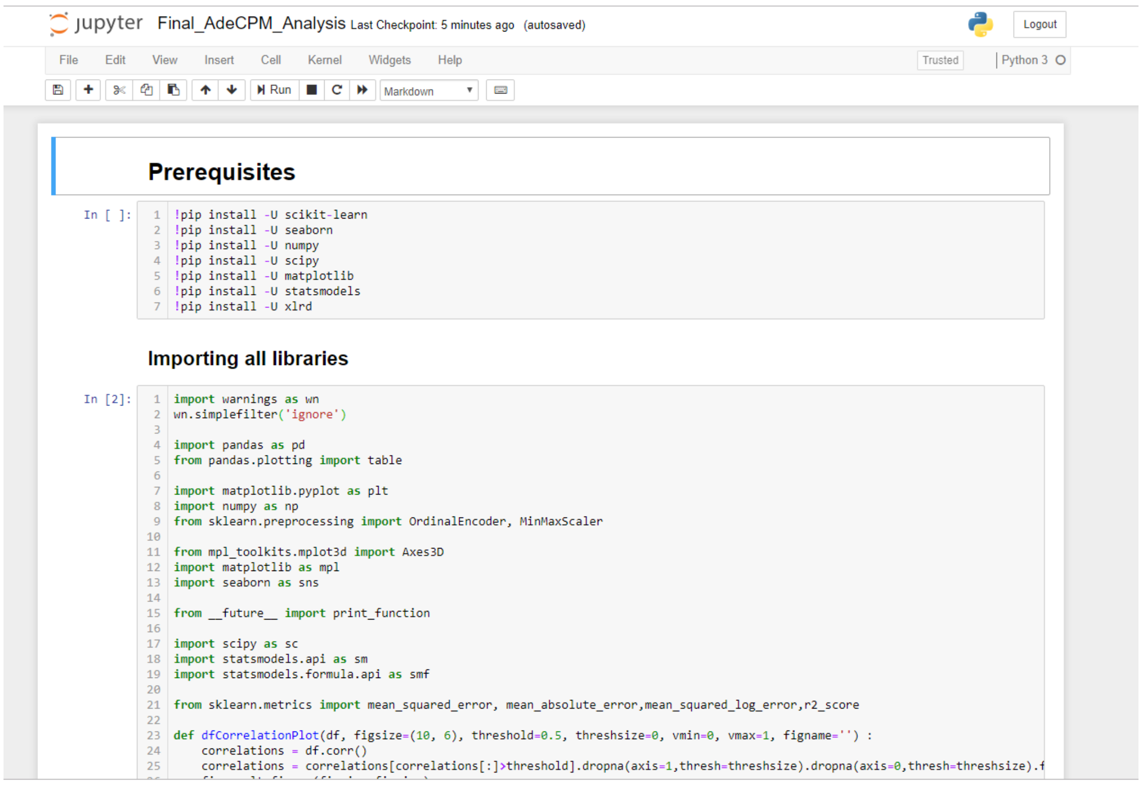
Prerequisites

```
[ ] !pip install -U scikit-learn
    !pip install -U seaborn
    !pip install -U numpy
    !pip install -U scipy
    !pip install -U matplotlib
    !pip install -U statsmodels
    !pip install -U xlrd
```

Importing all libraries

```
[ ] import warnings as wn
    wn.simplefilter('ignore')

    import pandas as pd
    from pandas.plotting import table

    import matplotlib.pyplot as plt
    import numpy as np
    from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler

    from mpl_toolkits.mplot3d import Axes3D
    import matplotlib as mpl
    import seaborn as sns

    from __future__ import print_function

    import scipy as sc
    import statsmodels.api as sm
    import statsmodels.formula.api as smf

    from sklearn.metrics import mean_squared_error, mean_absolute_error,mean_squared_log_error,r2_score
```

Figure 21: Google Collaboratory online environment


jupyter Final_AdeCPM_Analysis Last Checkpoint: 5 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Markdown

Prerequisites

In [ ]:
```
!pip install -U scikit-learn
!pip install -U seaborn
!pip install -U numpy
!pip install -U scipy
!pip install -U matplotlib
!pip install -U statsmodels
!pip install -U xlrd
```

Importing all libraries

In [2]:
```
import warnings as wn
wn.simplefilter('ignore')

import pandas as pd
from pandas.plotting import table

import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, MinMaxScaler

from mpl_toolkits.mplot3d import Axes3D
import matplotlib as mpl
import seaborn as sns

from __future__ import print_function

import scipy as sc
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn.metrics import mean_squared_error, mean_absolute_error,mean_squared_log_error,r2_score

def dfCorrelationPlot(df, figsize=(10, 6), threshold=0.5, threshsize=0, vmin=0, vmax=1, figname='') :
    correlations = df.corr()
    correlations = correlations[correlations[:]>threshold].dropna(axis=1,thresh=threshsize).dropna(axis=0,thresh=threshsize).f
```

Figure 22: Jupyter local notebook environment

In order to address the final implementation of the proposed software solution, certain popular libraries and toolboxes of Python were used such as (Numpy, 2018), (SciPy, 2018), (Pandas, 2018) and (Scikit-Learn, 2018) combined with two basic visualization libraries (MatplotLib, 2018) and (Seaborn, 2018).

For the statistical models, as well as for conducting statistical tests, and statistical data exploration such as ARIMA and SARIMA processes Statsmodels (Seabold, 2010) module has been used.

To visualize any interactivity, we also used (Bokeh, 2018) visualization library which supports such actions over plotted data.

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and

Figure 23: Jupyter environment website

6 Conclusion

The problem this dissertation tried to solve was to create a time series algorithmic procedure such as to forecast floor pricing values of Google Ad Manager pricing rules, to achieve revenue increment (maximization) based on historical data. At first, the data was provided through a programmatic procedure based on the API of Google Ad Manager reporting tool and offered a dataset with several pricing rules over several websites with the same or different ad unit dimensions. The algorithm managed to work in multiple instances and is now becoming a pillar of how the Yield Management sector for publishers. Utilizing the important variables that affect pricing and trying to leverage the maximum available amount of advertising money is something all publishers need to do. With this algorithm and way of operating the way for maximum efficiency is being paved.

6.1 Future Steps

To make this procedure more independent and accurate on the forecasting values that are provided, we propose a deployment on a neural network environment where the decision and the application of each pricing value would be instant by the Artificial Intelligent algorithm. These kinds of algorithms eliminate the problem of real-time feedback and provide more efficient ways to evaluate forecasting models. Moreover, the ability to introduce external parameters other than the actual ad parameters, e.g., weather conditions, sudden viral news, acts of God is also something that we should heavily take into account. Since we have moved towards the first step of actually incorporating data to actively affect our decision to set a price, anything that is relevant to the ad will now become an important factor.

6.2 Weaknesses

The proposed approach also suffers of some weak points that focus on the feedback scheme that should instantiate a re-evaluation process phase as soon as the error rate of the forecasting values goes over a threshold value. Moreover, the time margin being only up to a day is not sufficient to go into a marginal analysis

7 References

Bokeh. (2018, December 1). Retrieved from Bokeh: https://bokeh.pydata.org/en/latest/

Bozdogan, H. (1987). Model Selection and Akaike's Information Criterion (AIC): The General Theory and Its Analytical Extensions. *Psychometrika* , pp. 345–370.

Jun Wang, S. Y. (2016). Real-Time Bidding based Display Advertising: Mechanisms and Algorithms. *ECIR* .

Jupyter notebook environment . (2018, December 1). Retrieved from Jupyter notebook environment : https://jupyter.org/

K.P. Burnham, D. A. (2004). Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological methods and research* , pp. 261–304.

Kan Ren, W. Z. (2018, April). Bidding Machine: Learning to Bid for Directly Optimizing Profits in Display Advertising. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING* .

MatplotLib. (2018, December 1). Retrieved from MatplotLib: https://matplotlib.org/

Numpy. (2018, December 1). Retrieved from Numpy 1.15.4: http://www.numpy.org/

Pandas. (2018, December 1). Retrieved from Pandas: http://pandas.pydata.org/

Scikit-Learn. (2018, December 1). Retrieved from Scikit-Learn: https://scikit-learn.org

SciPy. (2018, December 1). Retrieved from SciPy: https://scipy.org/

Seabold, S. a. (2010). Statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference* .

Seaborn. (2018, December 1). Retrieved from Seaborn: https://seaborn.pydata.org

Shuai Yuan, J. W. (2014). A Survey on Real Time Bidding Advertising. *IEEE ADKDD* .

Weinan Zhang, J. X. (2016). Learning, Prediction and Optimisation in RTB Display Advertising. *CIKM* .

Wush Chi-Hsuan Wu, M.-Y. Y.-S. (2015, August). Predicting Winning Price in Real Time Bidding with Censored Data. *ACM* .

8 Appendices

8.1 Software code

```
# -*- coding: utf-8 -*-
"""Final_AdeCPM_Analysis.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1IuW6sKwpsGmyCSiDxD-fEoKAflENOOkL

#Prerequisites
"""

!pip install -U scikit-learn
!pip install -U seaborn
!pip install -U numpy
!pip install -U scipy
!pip install -U matplotlib
!pip install -U statsmodels
!pip install -U xlrd

"""##Importing all libraries"""

import warnings as wn
wn.simplefilter('ignore')

import pandas as pd
frompandas.plotting import table

importmatplotlib.pyplot as plt
importnumpy as np
```

```
fromsklearn.preprocessing import OrdinalEncoder, MinMaxScaler

from mpl_toolkits.mplot3d import Axes3D
importmatplotlib as mpl
importseaborn as sns

from __future__ import print_function

importscipy as sc
importstatsmodels.api as sm
importstatsmodels.formula.api as smf

fromsklearn.metrics import mean_squared_error,
mean_absolute_error,mean_squared_log_error,r2_score

defdfCorrelationPlot(df, figsize=(10, 6), threshold=0.5, threshsize=0, vmin=0, vmax=1, figname='') :
correlations = df.corr()
correlations =
correlations[correlations[:]>threshold].dropna(axis=1,thresh=threshsize).dropna(axis=0,thresh=threshsiz
e).fillna(0)
fig = plt.figure(figsize=figsize)
ax = fig.add_subplot(111)
hm = sns.heatmap(round(correlations,3), annot=True, ax=ax, cmap="coolwarm",fmt='.3f',
linewidths=.005, vmin=vmin, vmax=vmax)
if ~(figname=='') :
plt.savefig(figname)
plt.show()
return correlations

"""# Data Import

## Original Data

Importing Data from Excel spreadsheet.
"""

originalDF = pd.read_excel('adwords_new_clean.xlsx', sheet_name='Data', index_col=None)
```

```
prDF = pd.read_excel('pronews_price_floors.xlsx', sheet_name='DATA', index_col=[0,1,2])
originalDF = originalDF.join(prDF, on=['Pricing rules', 'Days', 'Branding types'])
print(originalDF.columns)
originalDF.describe()

"""Some sample data rows."""

originalDF.head()

"""## Encoding - Label indexing"""

feature_names = ['Pricing rules', 'Inventory sizes', 'Device categories', 'Branding types']
featuresDF = originalDF[feature_names]
OrdinalENC = OrdinalEncoder()
OrdinalENC.fit(featuresDF)
i=0
for cat in OrdinalENC.categories_:
print(feature_names[i], ' -> ', cat)
    i+=1
featuresDFindexed    =    pd.DataFrame(OrdinalENC.transform(featuresDF),    columns=feature_names,
index=None)
df = originalDF[['Days', 'Ad requests', 'Ad impressions',
                 'Ad request eCPM (β,¬)', 'Ad eCPM (β,¬)', 'Floor',
                 'Diff AdCPMAdReqCPM', 'Estimated revenue (β,¬)']]
df[feature_names] = featuresDFindexed
df = df.set_index('Days')

"""And the resulted dataframe."""

df.head()

"""## Scaling

Scaling features that are too large to manipulate aside others.
"""

MinMaxscaler = MinMaxScaler()
```

```
df[['Ad requests','Ad impressions']] = MinMaxscaler.fit_transform(df[['Ad requests','Ad impressions']])
df.head()

"""## Basic Data-Set Plotting

### 1. Floor value distribution per Pricing Rule
"""

fromscipy.stats import norm
fig = plt.figure(figsize=(12, 6))
for i in range(1, len(OrdinalENC.categories_[0])+1):
ax = fig.add_subplot(2, 4, i)
    ax.set_title(OrdinalENC.categories_[0][i-1].replace('pronews.gr', ''))
sns.distplot(df[df['Pricing rules']==i-1]['Floor'], ax=ax, fit=norm)
plt.tight_layout()
plt.show()

"""### 2. Numerical features distribution"""

plt.figure(figsize=(12, 12))
g = sns.pairplot(df, hue='Pricing rules', vars=['Ad requests', 'Ad impressions', 'Ad eCPM (β,¬)', 'Floor',
'Estimated revenue (β,¬)'])
for i, j in zip(*np.triu_indices_from(g.axes, 1)):
g.axes[i, j].set_visible(False)

replacements = OrdinalENC.categories_[0]
for i in range(len(g.fig.get_children()[-1].texts)):
label = int(float(g.fig.get_children()[-1].texts[i].get_text()))
    g.fig.get_children()[-1].texts[i].set_text(replacements[label].replace('pronews.gr ', ''))
g.fig.get_children()[-1].set_bbox_to_anchor((1.1, 0.5, 0, 0))
plt.show()

"""### 3. Categorical features distribution"""

plt.figure(figsize=(12, 12))
g = sns.pairplot(df, hue='Pricing rules', vars=['Inventory sizes', 'Device categories', 'Branding types', 'Ad
eCPM (β,¬)'])
```

```
for i, j in zip(*np.triu_indices_from(g.axes, 1)):
g.axes[i, j].set_visible(False)

replacements = OrdinalENC.categories_[0]
for i in range(len(g.fig.get_children()[-1].texts)):
label = int(float(g.fig.get_children()[-1].texts[i].get_text()))
   g.fig.get_children()[-1].texts[i].set_text(replacements[label].replace('pronews.gr ', ''))
g.fig.get_children()[-1].set_bbox_to_anchor((1.1, 0.5, 0, 0))
plt.show()

"""## Correlations"""

dfCorrelationPlot(df,threshold=0.0,threshsize=0,figname='CorrelationPlotDF.pdf')

"""## Group by Inventory sizes vs Ad eCPM

Histogram plots that show the distribution of data against inventory sizes according to the Ad eCPM values.
"""

grpDaysInventory = df.reset_index().groupby(['Days','Inventory sizes'], as_index=False)['Ad eCPM (β,¬)']
grpDaysInventory.aggregate(np.average)
DaysInventoryDF = grpDaysInventory.aggregate(np.average)[['Days', 'Inventory sizes','AdeCPM (β,¬)']]
DaysInventoryDFpivot = DaysInventoryDF.pivot(index='Days', columns='Inventory sizes', values='Ad eCPM (β,¬)').fillna(0)
axes = DaysInventoryDFpivot.hist(figsize=(16,20), layout=(5,5), sharey=True, sharex=True)
inventENC = OrdinalENC.categories_[1]
for i in range(axes.shape[0]):
for ax in axes[i]:
try:
num = int(float(ax.title.get_text()))
iflen(inventENC[num])>24:
inv = inventENC[num][:24]+'...'
else:
inv = inventENC[num]
```

```
ax.set_title(inv+'\n (avg: '+str(round(DaysInventoryDFpivot[[num]].mean()[num], 2))+')')
ax.set_xlabel('Ad eCPM (β,¬)')
ax.set_xlim(0,0.4)
ax.set_ylabel('Frequency')
except:
continue
plt.show()

"""## Group by Device categories vs Ad eCPM"""

grpDaysDevice = df.reset_index().groupby(['Days','Device categories'], as_index=False)['Ad eCPM (β,¬)']
DaysDeviceDF = grpDaysDevice.aggregate(np.average)[['Days', 'Device categories','AdeCPM (β,¬)']]
DaysDeviceDFpivot = DaysDeviceDF.pivot(index='Days', columns='Device categories', values='Ad eCPM (β,¬)').fillna(0)
axes = DaysDeviceDFpivot.hist(figsize=(9,8), layout=(2,2), sharey=True, sharex=True)
devicesENC = OrdinalENC.categories_[2]
for i in range(axes.shape[0]):
for ax in axes[i]:
num = int(float(ax.title.get_text()))
ax.set_title(devicesENC[num]+' (avg: '+str(round(DaysDeviceDFpivot[[num]].mean()[num], 2))+')')
ax.set_xlabel('Ad eCPM (β,¬)')
ax.set_xlim(0.15,0.4)
ax.set_ylabel('Frequency')
plt.show()

"""## Group by Pricing rules vs Ad eCPM"""

grpDaysRules = df.reset_index().groupby(['Days','Pricing rules'], as_index=False)['Ad eCPM (β,¬)']
DaysRulesDF = grpDaysRules.aggregate(np.average)[['Days', 'Pricing rules','AdeCPM (β,¬)']]
DaysRulesDFpivot = DaysRulesDF.pivot(index='Days', columns='Pricing rules', values='Ad eCPM (β,¬)').fillna(0)
axes = DaysRulesDFpivot.hist(figsize=(12,9), layout=(3,3), sharey=True, sharex=True)
devicesENC = OrdinalENC.categories_[0]
for i in range(axes.shape[0]):
for ax in axes[i]:
try:
```

```
num = int(float(ax.title.get_text()))
ax.set_title(devicesENC[num].replace('pronews.gr ', '')+
                        ' (avg: '+str(round(DaysRulesDFpivot[[num]].mean()[num], 2))+')')
ax.set_xlabel('Ad eCPM (β‚¬)')
ax.set_xlim(0,0.6)
ax.set_ylabel('Frequency')
except:
continue
plt.show()

"""## Group by Branding types vs Ad eCPM"""

grpDaysBranding = df.reset_index().groupby(['Days','Branding types'], as_index=False)['Ad eCPM
(β‚¬)']
DaysBrandingDF = grpDaysBranding.aggregate(np.average)[['Days', 'Branding types','AdeCPM (β‚¬)']]
DaysBrandingDFpivot = DaysBrandingDF.pivot(index='Days', columns='Branding types', values='Ad
eCPM (β‚¬)').fillna(0)
axes = DaysBrandingDFpivot.hist(figsize=(10,4), layout=(1,2), sharey=True, sharex=True)
brandENC = OrdinalENC.categories_[3]
for i in range(axes.shape[0]):
for ax in axes[i]:
try:
num = int(float(ax.title.get_text()))
ax.set_title(brandENC[num]+' (avg: '+str(round(DaysBrandingDFpivot[[num]].mean()[num], 2))+')')
ax.set_xlabel('Ad eCPM (β‚¬)')
ax.set_ylabel('Frequency')
except:
continue
plt.show()

"""# Correlations Ad eCPM over Branding, Devices and Inventories

##Correlation Matrix

Threshold value 0.95 and binding length at least 7
"""

```

```
newDF = pd.concat([DaysBrandingDFpivot, DaysDeviceDFpivot, DaysInventoryDFpivot],
axis=1, keys=['Branding','Device','Inventory'])
resCorrelations = dfCorrelationPlot(newDF,
threshold=0.95,
threshsize=7,
figsize=(12,8), vmin=0.92,
figname='CorrelationPlotBrandingDeviceInventory_0.95_7.pdf')

"""##Correlated categorical dimensions
High scored features identification
"""

resCorr = resCorrelations.reset_index()
for i in range(OrdinalENC.categories_[2].shape[0]):
   r = resCorr
try:
      r = r[(r['level_1']==i) & (r['level_0']=='Device')]
      Branding                                                                          =
int(r.Branding[r.Branding==np.max(r.Branding.as_matrix())].dropna(axis=1).columns[0])
inventory = int(r.Inventory[r.Inventory==np.max(r.Inventory.as_matrix())].dropna(axis=1).columns[0])
print(OrdinalENC.categories_[2][i])
print('\t'+str(Branding)+':'+OrdinalENC.categories_[3][Branding],
str(inventory)+':'+OrdinalENC.categories_[1][inventory])
except:
continue
print('=======================================')
for i in range(OrdinalENC.categories_[1].shape[0]):
   r = resCorr
try:
      r = r[(r['level_1']==i) & (r['level_0']=='Inventory')]
      Branding                                                                          =
int(r.Branding[r.Branding==np.max(r.Branding.as_matrix())].dropna(axis=1).columns[0])
      Device = int(r.Device[r.Device==np.max(r.Device.as_matrix())].dropna(axis=1).columns[0])
print(i, OrdinalENC.categories_[1][i])
print('\t'+str(Branding)+':'+OrdinalENC.categories_[3][Branding],
str(Device)+':'+OrdinalENC.categories_[2][Device])
except:
```

```
continue

"""#Model fitting phase ARIMA proccess

##Analytical AIC criterion matrix

Analytical estimate of the Akaike information criterion (AIC) along with visualization of Q-Q plots per model parameters.
"""

cond = (df['Branding types']==1) &(df['Device categories']==1) &(df['Inventory sizes']==7)
data = df.loc[cond,['Ad eCPM (β,¬)']]
grpData = pd.DataFrame(data.groupby(['Days'], as_index=True)['Ad eCPM (β,¬)'].agg(np.average))
all_days = pd.date_range(grpData.index.min(), grpData.index.max(), freq='D')
grpData = grpData.reindex(all_days)
grpData = grpData.fillna(grpData.rolling(4,min_periods=1).mean())
grpData['Lag_1']=grpData - grpData.shift(1)
grpData['Lag_2']=grpData['Ad eCPM (β,¬)'] - grpData['Ad eCPM (β,¬)'].shift(2)
grpData['Lag_3']=grpData['Ad eCPM (β,¬)'] - grpData['Ad eCPM (β,¬)'].shift(3)
grpData['UP_DOWN']=(grpData['Lag_1']>=0).astype(int)
grpData = grpData.fillna(0)

maxp=3
maxd=0
maxq=2

aic_full = pd.DataFrame(np.zeros((maxp+1,maxq+1), dtype=float))
models = pd.DataFrame(np.zeros((maxp+1,maxq+1), dtype=object))

fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)

for p in np.arange(0,maxp+1):
for q in np.arange(0,maxq+1):
models.iloc[p,q] = sm.tsa.ARIMA(grpData['Ad eCPM (β,¬)'], order=(p,0,q))
try:
models.iloc[p,q] = models.iloc[p,q].fit(disp=False)
```

```
aic_full.iloc[p,q] = models.iloc[p,q].aic
        fig = sm.qqplot(models.iloc[p,q].resid, line='q', ax=ax, fit=True, label='Model ('+str(p)+','+str(q)+')')
except:
aic_full.iloc[p,q] = 0.0
print(aic_full)

colormap = plt.cm.Pastel1
colors = [colormap(i) for i in np.linspace(0, 1, (maxp+1)*(maxq+1)*2)]
fori,j in enumerate(ax.lines):
j.set_color(colors[i])
plt.legend(loc='best')
plt.show()

"""##Visualization of the models described in the previous step"""

fromdatetime import datetime as dt
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
grpData.loc[dt.strptime('2018-08-01 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM (β,¬)']].plot(ax=ax)

d = 0

for p in np.arange(0,maxp+1):
for q in np.arange(0,maxq+1):
try:
        predictions = models.iloc[p,q].predict(start=dt.strptime('2018-08-01 10:00:00','%Y-%m-%d %H:%M:%S'),
end=dt.strptime('2018-10-10 10:00:00','%Y-%m-%d %H:%M:%S'),
dynamic=False)
predictions.shift(-1).plot(ax=ax, label='ARIMA('+str(p)+','+str(d)+','+str(q)+')')
except:
continue

colormap = plt.cm.Accent
colors = [colormap(i) for i in np.linspace(0, 1, (maxp+1)*(maxq+1)*2)]
```

```
fori,j in enumerate(ax.lines):
ifj.get_label()=='Ad eCPM (β,¬)':
j.set_color('r')
else:
j.set_color(colors[i])
plt.legend(loc='best')
plt.show()

"""#Best model fit

Visualization of the predictions made by the model scored highest in the previous phase.
"""

fromdatetime import datetime as dt
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
grpData.loc[dt.strptime('2018-07-10 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM
(β,¬)']].plot(ax=ax, color='r', marker='o')

p = 2
d = 0
q = 0
model = models.iloc[p,q]
predictions = model.predict(start=dt.strptime('2018-07-10 10:00:00','%Y-%m-%d %H:%M:%S'),
end=dt.strptime('2018-10-31 10:00:00','%Y-%m-%d %H:%M:%S'),
dynamic=False)

predictions.shift(-1).plot(ax=ax, label='ARIMA('+str(p)+','+str(d)+','+str(q)+')', color='b', marker='o')

test = grpData.loc[dt.strptime('2018-07-10 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM (β,¬)']]
fromsklearn.metrics import mean_squared_error,
mean_absolute_error,mean_squared_log_error,r2_score

tbl = pd.DataFrame({
'AIC' :models.iloc[p,q].aic,
'BIC' :models.iloc[p,q].bic,
'r2' : r2_score(test, predictions.iloc[:92]),
'MAE' :mean_absolute_error(test, predictions.iloc[:92]),
'MSE' :mean_squared_error(test, predictions.iloc[:92]),
'RMSE' :np.sqrt(mean_squared_error(test, predictions.iloc[:92]))
}, index=['ARIMA('+str(p)+','+str(d)+','+str(q)+')'])

table(ax, np.round(tbl.T, 4), loc='center right', colWidths=[0.2, 0.2])

plt.legend(loc='best')
plt.show()

"""##Model Summary"""

print(model.summary2())

"""##Forecasted values

The forecasted values of the previous model.
"""

print('Forecasts')
print(predictions.iloc[92:])

"""#Seasonal ARIMA proccess

##Data decomposition

Decomposing data to show thier trend, seasonality
"""

fromstatsmodels.tsa.seasonal import seasonal_decompose
cond = (df['Branding types']==1) &(df['Device categories']==1) &(df['Inventory sizes']==7)
data = df.loc[cond,['Ad eCPM (β,¬)']]
grpData = pd.DataFrame(data.groupby(['Days'], as_index=True)['Ad eCPM (β,¬)'].agg(np.average))
all_days = pd.date_range(grpData.index.min(), grpData.index.max(), freq='D')
grpData = grpData.reindex(all_days)
grpData = grpData.fillna(grpData.rolling(4,min_periods=1).mean())
```

```
ts = grpData.loc[dt.strptime('2018-07-01 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM (β,¬)']]
decomp = seasonal_decompose(ts)

tr = decomp.trend
ses = decomp.seasonal

plt.figure(figsize=(12,8))
plt.subplot(311)
plt.plot(ts, label='Original', marker='o')
plt.legend(loc='best')
plt.subplot(312)
plt.plot(tr, label='trend', marker='o', color='r')
plt.legend(loc='best')
plt.subplot(313)
plt.plot(ses, label='seasonal', marker='o', color='g')
plt.legend(loc='best')
plt.show()

"""##ACF and PACF plots

Also show their autocorrelation and partial autocorrelation plots.
"""

fig = plt.figure(figsize=(12,7))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts, lags=40, ax=ax2)

"""## SARIMA model fitting"""

p = 3
d = 0
q = 1
sesP = 1
sesD = 0
sesQ = 0
```

```
ses = 2

nm = sm.tsa.statespace.SARIMAX(grpData['Ad eCPM (β,¬)'], order=(p,d,q),
seasonal_order=(sesP,sesD,sesQ,ses)).fit( disp=False)
predictions = nm.predict(start='2018-07-10 10:00:00', end='2018-10-09 10:00:00', dynamic=False)
print(nm.summary())

"""##Visaulizing data"""

fig, ax = plt.subplots(figsize=(12,8))
grpData.loc[dt.strptime('2018-07-10 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM
(β,¬)']].plot(ax=ax,color='b',
label='Original Observations')
predictions.shift(-1).plot(ax=ax,
label='SARIMAX('+str(p)+','+str(d)+','+str(q)+')x('+str(sesP)+','+str(sesD)+','+str(sesQ)+')
m='+str(ses),color='r')

test = grpData.loc[dt.strptime('2018-07-10 10:00:00','%Y-%m-%d %H:%M:%S'):,['Ad eCPM (β,¬)']]

tbl = pd.DataFrame({
'AIC' :nm.aic,
'BIC' :nm.bic,
'r2' : r2_score(test, predictions),
'MAE' :mean_absolute_error(test, predictions),
'MSE' :mean_squared_error(test, predictions),
'RMSE' :np.sqrt(mean_squared_error(test, predictions))
}, index=['Model('+str(p)+','+str(q)+')'])

table(ax, np.round(tbl.T, 4), loc='center right', colWidths=[0.2, 0.2])

plt.legend(loc='best')
plt.show()

"""##Forecasting 7 steps ahead"""

print(nm.forecast(steps=7))

```

```
"""##Plotting residuals"""

residuals = pd.DataFrame(nm.resid, columns=['Residuals'])
ax = residuals.plot.hist(figsize=(10,6))
residuals.Residuals.plot.kde(ax=ax, secondary_y=True)
importscipy.stats as sc
x=np.arange(-0.15,0.15,0.001)
plt.plot(x,sc.norm.pdf(x,0,0.0265),label='N(0,1)')
plt.legend(loc='upper left')
plt.show()

"""#Last method on modeling data

##Exponential Smoothing

Specifically Holt - Winters method on Exponential Smoothing
"""

fromstatsmodels.tsa.holtwinters import ExponentialSmoothing
Holtdf = grpData['Ad eCPM (β,¬)']
split = round(len(grpData['Ad eCPM (β,¬)'])*0.8)
train, test = Holtdf.iloc[:split], Holtdf.iloc[split:]
model = ExponentialSmoothing(train, seasonal_periods=9, seasonal="mul").fit()
pred = model.predict(start=test.index[0], end=test.index[-1])

fig, ax = plt.subplots(figsize=(10,6))
plt.plot(train.index, train, label='Train')
plt.plot(test.index, test, label='Test')
plt.plot(pred.index, pred, label='Holt-Winters')

tbl = pd.DataFrame({
'AIC' :model.aic,
'BIC' :model.bic,
'r2' : r2_score(test, pred),
'MAE' :mean_absolute_error(test, pred),
'MSE' :mean_squared_error(test, pred),
'RMSE' :np.sqrt(mean_squared_error(test, pred))
}, index=['Holt-Winters'])

table(ax, np.round(tbl.T, 4), loc='center right', colWidths=[0.2, 0.2])
plt.legend(loc='best')
plt.show()

"""##Residuals of Holt - Winters method"""

residuals = pd.DataFrame(model.resid, columns=['Residuals'])
ax = residuals.plot.hist()
residuals.Residuals.plot.kde(ax=ax, secondary_y=True, color='b',label='kde')

importscipy.stats as sc
x=np.arange(-0.15,0.15,0.001)
plt.plot(x,sc.norm.pdf(x,0,0.02),label='N(0,1)')

plt.legend(loc='upper left')
plt.show()
```