

National Research University Higher School of Economics
Graduate School of Business
Master's Programme "Big Data Systems" ("Business Analytics and Big Data
Systems" for 1-st year students)


Project report

Realization and improvement of the insurance prolongation process
(Project title)

Performed by student

Eremin Gleb Borisovich

(student's full name)


(student's signature)

Project supervisor:


Head of XSELL₂

Gort Andrey Viktorovich

(position, full name)

10.06.2022

(date)


(signature)

Moscow 2022

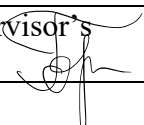
Assessment sheet

Realization and improvement of the insurance
prolongation process (project title)

Applied (project type)

Module 4 (project duration)

Project supervisor Full name, position	Head of XSELL, Gort Andrey Viktorovich
Student¹:	1
Full name	Eremin Gleb Borisovich
Master's Programme	Big Data Systems
Группа №	211

Components of the final grade ²	Grade on a 10-point scale	Notes (if necessary)
О np – Grade for the project result (product)	8	
О cn Grade for the methods and technologies used	8	
О p Grade for the implementation of the project's work	8	
О κ Grade for the developed competencies	8	
О rp Grade for the student's individual contribution to group work	–	
О kom Grade for team work	–	
О 3 Grade for the presentation, project defense		
О B3 Grade given by other project participants (peers evaluation)	–	
О c Student self-assessment	8	
Grade calculation (formula with weight coefficient)		
Final grade for the project		Project supervisor's signature 
Number of credits	3	

Date 10.06.2022

¹ For group projects, an assessment sheet is filled out for each group member

² Only necessary components are used, if some component is not used in the assessment, then a dash is put in the corresponding line

Table of contents

General description of the project.....	3
The main part.....	4
What is OSAGO and E-OSAGO?	4
The architecture of the prolongation job	4
Accumulating data in the DWH.....	5
Extracting data into the job and sending to CDXP	7
Errorhandler	7
Exponea_helper.....	8
Snowflake_helper.....	8
Main.....	8
Ensuring the daily operation of the process	10
The idea of segmenting clients.....	10
The dataset for training model	11
The implementation of the model	12
Conducting A/B testing.....	15
Conclusion.....	16

General description of the project

The project was initiated by the supervisor to get more control on the prolongation process of car insurance policies, which are also known as CTP or OSAGO. Before its implementation, all of the communications were on the side of the insurance team, which made it difficult to make any changes, as everything should have been agreed with the product team, as well as the implementation was also on their side.

The main idea of the project was to create a clear and easily configured process, which included the job, which could send communications, including calls, emails, SMS and push notifications, every day to the potential clients. At the same time, there already existed some ideas, like implementing ML model to predict the best time for communication, which could be applied in the process, so its architecture should have taken into account the possibility of a quite simple applying of additional functionality in the process.

Project initiator	Sravni OOO
Project supervisor	Gort Andrey Viktorovich
Type of the project	Applied
Where the project was realized	Sravni OOO

The main part

What is OSAGO and E-OSAGO?

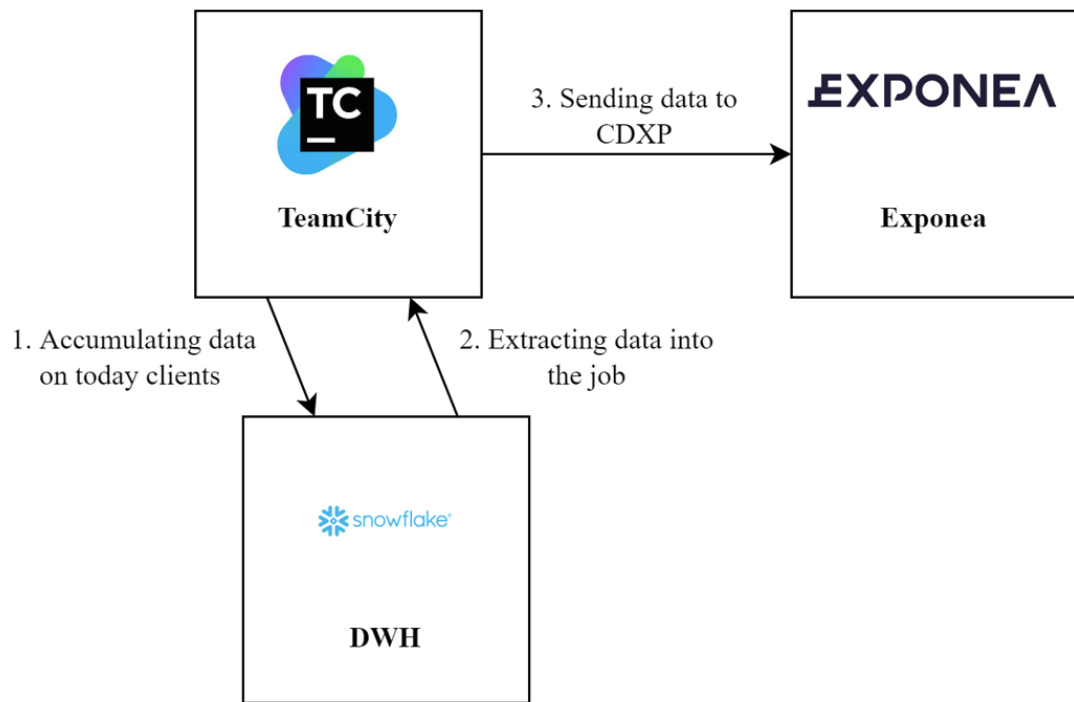
OSAGO is a mandatory insurance of motor liability in Russia. In simple words, in case of a car accident such kind of an insurance policy will cover the damage caused to the cars and the culprit will not have to reimburse it, while the victim will not have to demand funds from the guilty person on repair of the car personally. They usually are valid for one-year period, so the prolongation times comes just before its expiration time year later.

E-OSAGO or E-CTP is its electronic version, which originally could be bought at the sites of the insurance companies. There is no difference between two versions of policies, as, if client prefers, he or she can print the copy of the electronic version or just use the file with the document. At the end of 2020 the sale of E-CTP was also permitted at marketplaces, which have some advantages over many sites, as the client only needs to enter data once at the marketplace and compare prices from all available insurance companies. These marketplaces can be accessed through websites or mobile applications. As the insurance is obligatory, potential policyholders are always interested in buying the policy at the lowest possible price, and the marketplaces, such as presented at sravni.ru, seem to be the best option for them.

The architecture of the prolongation job

The prolongation job architecture, presented at picture 1, consists of several steps:

1. Accumulating data in the DWH, which is realized on the Snowflake database, about potential policyholders, including date of the policy expiration and information on the car and drivers
2. Extracting the data into the job, adding some information and transforming it into the suitable format
3. Sending the data about the client and the type of communication to customer data and experience platform, which then sends out the communications



Picture 1 The prolongation job architecture

Accumulating data in the DWH

Since the prolongation works with the existent clients, all data about them is already present in the database. Though, at this step two main problems were faced:

1. Filtering available for communications clients, taking into account the possibility of two or more insurance policies bought by one client
2. The setup of the process of adding new communications to the job without need of editing anything at the other steps

The first problem was solved with the help of the product team, who provided the full description of their tables in the database and functions to filter the clients. To exclude the duplication of communications to one client, who owns two or more policies, it was decided to take into account each client's first not prolonged policy, which allowed to make serial communication chain. In other words, if client prolongs his earliest policy, he then will receive communication on the second earliest not prolonged policy and so on.

The second problem was solved by the division of the accumulating of data process in two main steps:

1. Creating an sql view with all the data about clients available for communication, including their phones, emails, policy expiration date and etc. The example of the view structure is presented in Table 1.
2. Creating reference table with the list of communications. The example of the table is presented in Table 2.

Field	Type	Description	Example value
id	NUMBER	Unique client id	12345
phone	NUMBER	Client phone number	79164274858
policyenddate	DATE	Policy expiration date	2022-01-01

Table 1 The view structure

Field	Type	Description	Example value
communicationtype	STRING	The type of communication	email
daysbeforeexpirationdate	NUMBER	The amount of days before the expiration date, when the communication should be sent	10

Table 2 The reference table structure

That solution provided the possibility of extracting all the needed data by the joint of these two data structures. The example of the sql query is presented on Picture 2.

```
SELECT *  
FROM VIEW  
JOIN REFERENCE_TABLE ON  
    DATEDIFF(DAYS, CURRENT_DATE(), VIEW.policyenddate) =  
    REFERENCE_TABLE.daysbeforeexpirationdate
```

Picture 2 The request example

The result of the joint is saved every day in the logs table with the addition of the date of communication. That allows to get not only the general data about the communication, which was sent at particular day to the client, but to find out the everyday amounts of sent communications.

Extracting data into the job and sending to CDXP

The main part of the job was created in Python programming language, as it has ready libraries for connecting to the database, sending requests and one of the best tools for working with different data forms and formats.

The whole Python job consists of several files:

1. Errorhandler.py – The specially created class, which inherits the Exception class, to handle errors, while sending the users to CDXP
2. Exponea_helper.py – The file with the functions, which are used to establish connection to the Exponea API
3. Snowflake_helper.py – The file with the functions, which are to establish connection to the Snowflake API
4. Main.py – The main code of the job, which is regularly started by the Teamcity

Errorhandler

In this module the custom exception class is defined, so the errors, which can happen during the process of sending queries to Exponea. The main goal of this class is to give an easy-to-use instrument, which was implemented with its function, to get all needed information on the type of exception - the code of the answer – and the client, which was failed to send, to add the data about the failure in the log table.

Exponea_helper

This module implements the function, which allows sending contacts to CDXP. The function uses the Requests module to send a post-type request to Exponea API, which is containing data on the user and communication in the json. For that purpose, the json data is passed as the argument of the function.

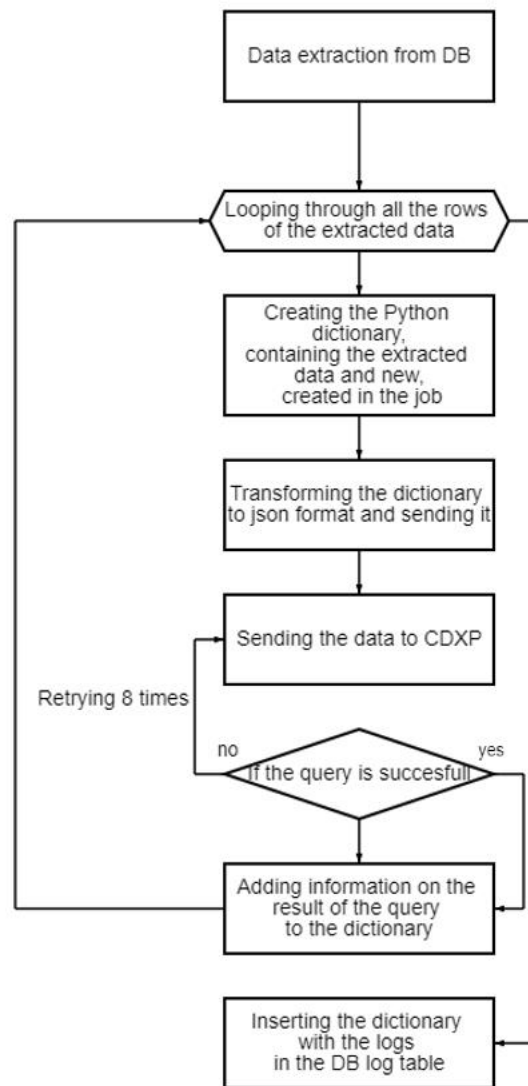
In the case of error code answer of the API the custom exception is raised. After it happens, with the use of the Backoff module, the function retries sending the contact at the maximum for 8 times. If the error is still raised, then the data about failure is added in the log table, and function starts trying to send a new contact.

Snowflake_helper

To extract data from the database to the job buffer the snowflake.sqlalchemy and the snowflake.connector modules were used to create connection to the database, with the use of the link to the database itself, username and password. For this particular task, a new special account was created for security reasons, so all the triggered queries could be marked as done by the job.

Main

This is the file of the job, which contents the main function of the job, which triggers the whole process. The block diagram of the process is presented on Picture 3.



Picture 3 The block diagram of the process

The snowflake_helper and Pandas modules were used for sending queries and receiving the answer. Then, based on the received data, information on the client and the communication was transformed in json format. Some of the fields of the json are added directly in the job, so the processes, that are working in the database keep unchanged, which benefits the overall stability of the whole process. Then, the data is put into the Python dictionary data type, which was then transformed into the json format with the use of the Json module. The final json type data is sent to the CDXP and the result is written in the log table.

Ensuring the daily operation of the process

Since the process was meant to be daily, it was necessary to ensure its automatic start every day. For that purpose, Teamcity was used. The teamcity is a build management and continuous integration server, which allows to run multiple builds and tests under different platforms and environments simultaneously. It provides wide opportunities to configure the build, including its launch time, the notifier system about the possible errors of execution, which can send messages on the email or in slack, and has a ready solution for copying of the repository from the Github into the build.

Each build on Teamcity can consist of several steps, which can run bash scripts or repository files independently, which allows adding new functionality into process without the fear of ruining it, as the error caused by one of the steps will not ruin the whole process. Moreover, Teamcity allows triggering builds not only based on the schedule, but also on the result of another build. That allows adding new steps to the job without even editing the existent ones, to achieve this goal another build can be configured and added to the original “chain” of builds.

In this job two main builds were configured. The first one was all about accumulating data on the database side – inserting the data about today clients in the tables. That build was configured to start daily and retry with deleting the inserted data, if any error happens during the process. The second one was configured to start just after the end of the first one, and launched the Python job itself.

The idea of segmenting clients

The idea of segregating prolongation users on the groups by their preferable communications time came from the analytics of the current prolongation process. From the aggregated data it was seen, that the users can be split into three groups by the amount of days of their policies prolongation before the end of the policy. It was decided to change the communication politics, based on the belonging of the user to one of these groups.

There were several reasons for this:

1. To save money on excess communications, which are sent too early to the clients, who tend to prolong their policies just before the deadline
2. To avoid annoying clients with excess communications, when they already remember about the ending of their policies, but are not eager to prolong the policy at this time and would prefer waiting for the deadline to come closer

It was decided to split the clients in three groups:

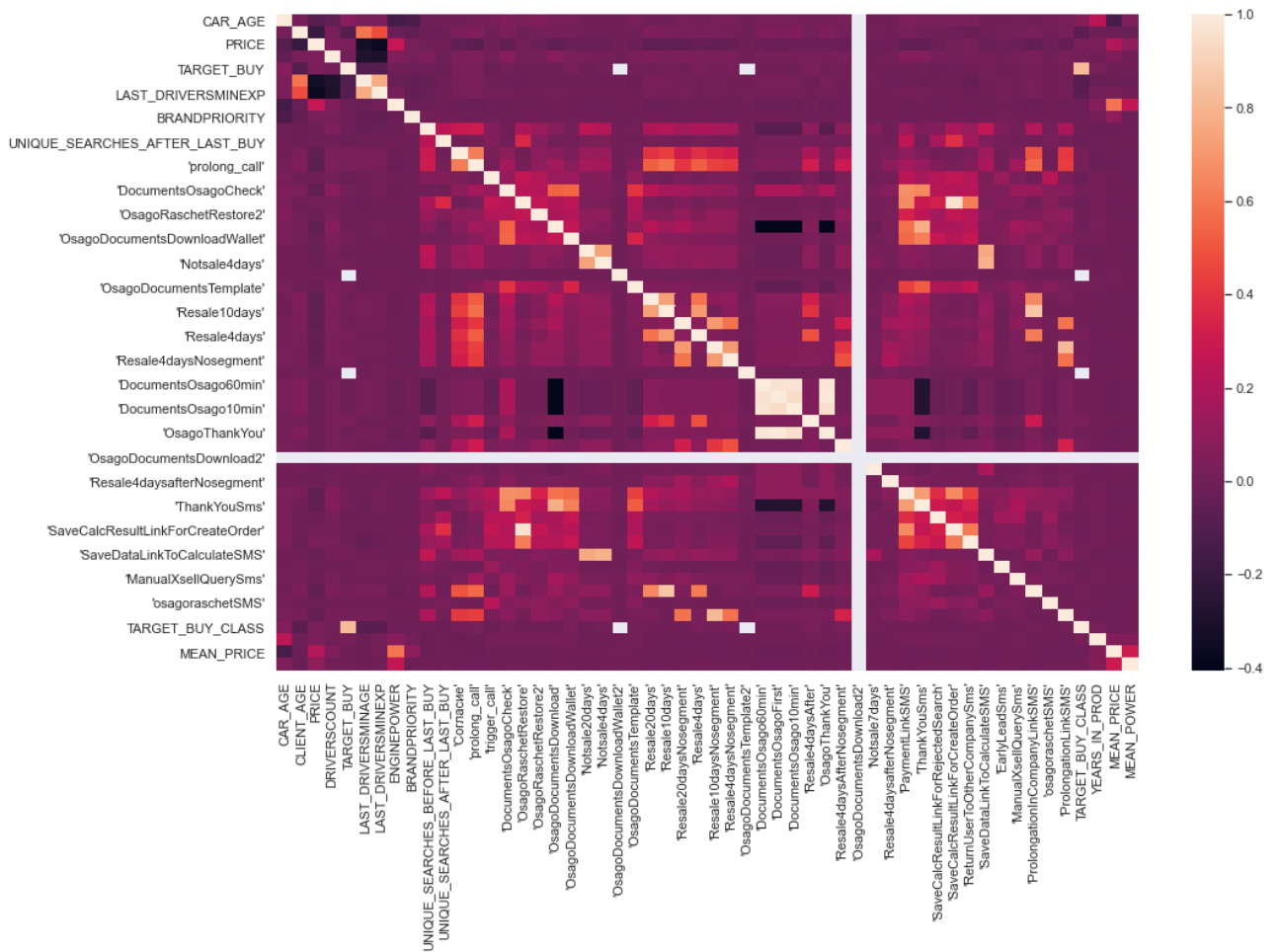
1. Those, who prefer buying policy early before the deadline. These clients were marked with 0 in the class column.
2. Those, who prefer buying policy just a week before the deadline. These clients were marked with 1 in the class column.
3. Those, who prefer buying policy after the deadline. These clients were marked with 2 in the class column.

The dataset for training model

Let us introduce the DND term, which defines the day of the end of the policy.

At the first sight it was decided to check whether the amount of days before DND in the prolongation year, had a correlation with the amount of days before DND in the year, when the policy was first bought on the site. It was found, that there was some correlation, but the results of the implemented A/B testing were not as satisfying, as they expected to be. So, it was decided to build a ML model to predict the class of the potential customer, which could use the amount of days as one of the features.

The dataset for the model was collected from the available data on previous communications with the client and client's previous policy purchase. Their total amount was around 24. To train the model, the month with the biggest percent of prolongation and in which the communication policy was not yet configured by our team, so its influence on the target class would be the lowest, was taken. That allowed, to get information on both the purchase a year ago, and that was done this year and is considered to be the prolongation one. The correlation matrix of the used features is presented in the Picture 4.



Picture 4 The correlation heatmap of the features

The initial distribution of classes in the assembled dataset was the following:

- Class 0 – 25%
- Class 1 – 65%
- Class 2 – 10%

The classes were marked based on the amount of days before DND, considering the prolongation purchase.

The implementation of the model

Before training the model, the data was firstly prepared. All the missing values were filled in with the median values, where possible, while the others were deleted from the initial dataset.

The trained model is based on the Catboost Classifier due to several reasons:

1. No need of pre-engineering the categorical features, which can just be transferred to the training function as the `cat_features` parameter
2. The possibility of passing the class weights directly in the function
3. One of the best results among the other algorithms, as it provides the improved accuracy by reducing the overfitting

As a result of training and implementing the model, and taking in concern the unbalanced distribution of users among the classes in the dataset, the following results were received:

Class	F1-score
0	0.43
1	0.84
2	0.68

With the overall accuracy of the model of 0.6634917198831042.

The top 10 most important features are presented on Picture 5, and their description is presented in the table below.

	Feature Id	Importances
0	DIFF_BUY_BEFORE	20.720048204810
1	COMPANYNAME	12.342544010035
2	CAR_AGE	9.737628954649
3	LOCATION	6.709734439185
4	LAST_DRIVERSMINAGE	5.773320455038
5	LAST_DRIVERSKBM	5.448737203065
6	CLIENT_AGE	4.720656913094
7	COMPANYID	3.967880332035
8	LAST_CAMPAIGNMEDIUM	3.072206672707
9	UNIQUE_SEARCHES_AFTER_LAST_BUY	2.700935497658
10	LAST_DRIVERSMINEXP	2.681296061968

Picture 5 The top 10 most important features

Feature	Short description
Diff_buy_before	The amount of days before DND, considering the purchase a year ago
Companyname	The name of the insurance company, where the policy was initially bought
Car_age	The age of the car
Location	The location of the insurer
Last_driversminage	The minimum age of the driver, who is inscribed in the policy
Last_driverskbm	The overall KBM – the accident-free driving ratio
Client_age	The age of the insurer who buys the policy
Companyid	The id of the company, which produced the car
Last_campaignmedium	The utm_medium of the last purchase on the site
Unique_searches_after_last_buy	The amount of calculations, made by the insurer after the purchase of the policy on the site
Last_driversminexp	The minimum experience among the drivers inscribed in the policy

Conducting A/B testing

As the prolongation base is known in advance, the users can be marked by the model just before the first communication. To implement this, a special table in the database was created with two columns – the unique identifier of the policy to be prolonged, and the number of the class, and a new step was added to the Teamcity build, which extracts the data about the clients and runs the Python script, which implies model on each client, returning their class.

The model was tested on 10% of the communication flow, which was segmented by the last number of the phone number of the client. It showed better results in terms of the percentage of consents to the calculation on the site after call communications.

Conclusion

The implemented prolongation job is working stable for several months right now. Its architecture allows adding new steps without the concern of ruining the whole process. These new steps can be added even by the other people, who can use the documentation. That allowed to make the job be available to be supported by anyone.

The implemented ML algorithm had a positive impact on the customers reaction to the communications, and the conversion rate into policy prolongations at the same time. It is still used to mark the clients, in order to improve their experience with the platform.

This report presents the process of the development and implementing both the prolongation job and the classifier. The results with the figures and graphs are under NDA and cannot be shared. Thought, the positive impact on the whole prolongation process can be indicated.