TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Heino Sass Hallik 179636 IAIB

Konstantin Saposnitsenko 164026 IAPB

Juri Ponomarenko 164668 IAPB

DEVELOPMENT OF PIPEDRIVE SALES VELOCITY CALCULATOR

Bachelor's thesis

Supervisor: Martin Verrev
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Heino Sass Hallik 179636 IAIB

Konstantin Saposnitsenko 164026 IAPB

Juri Ponomarenko 164668 IAPB

PIPEDRIVE MÜÜGIKIIRUSE ARVUTAMISE RAKENDUSE ARENDUS

bakalaureusetöö

Juhendaja: Martin Verrev
MSc

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Authors: Heino Sass Hallik, Konstantin Saposnitsenko, Juri Ponomarenko

25.05.2020

Abstract

As part of this thesis, a web application was developed for the client, Pipedrive. The application is a sales velocity calculator. Sales velocity is a metric, which tells a salesperson how much money's worth of successful deals he makes in a given time period. The purpose of the application is to give the salesperson a better understanding of the effectiveness of his work, and to help him understand which of his sales velocity parameters could be improved.

The application uses a client-server architecture. It uses the Gin-gonic and React.js frameworks, as well as PostgreSQL database and Docker. New versions of the application are deployed using Gitlab CI/CD. End users' data is fetched from Pipedrive's public API. The application requires the user to login using OAuth.

Initially, the scope of the work included automatically making suggestions about which sales velocity parameters the salesperson should improve. A neural network based machine learning approach was used for this purpose. However, the used data set proved difficult for machine learning and the attempt was unsuccessful. One result of this work is a thorough analysis of the data set and the code for an inadequately accurate machine learning model.

The application is available at www.pipedrivesalesvelocity.tk. A Pipedrive account is required to log in.

The thesis is in English and contains 58 pages of text, 8 chapters, 33 figures and 7 tables.

Annotatsioon

Käesoleva bakalaureusetöö raames on tehtud Pipedrive jaoks müügikiiruse arvutamise rakendus. Müügikiirus on indikaator, mis annab müügiinimesele teada rahasumma, mis väärtuses ta teatud ajaperioodis edukaid müügitehinguid teeb. Eesmärk on anda müügiinimesele parema arusaamise tema töö efektiivsuse kohta ning aidata mõista, missuguseid tema müügikiiruse parameetreid oleks tark parandada.

Rakendus on valmistatud klient-server arhitektuuri järgi, kasutades Gin-gonic ning React.js raamistikke, PostgreSQL andmebaasi ning Dockerit. Uute versioonide serverisse paikamise protsess on lahendatud Gitlab CI/CD tööriistaga. Lõppkasutaja andmed saadakse Pipedrive avaliku API'ga suheldes ning sisselogimine kasutab Pipedrive OAuth'i.

Töö esialgsesse skoopi jäi lõppkasutajatele automaatsete soovituste pakkumine selle kohta, missuguseid müügikiiruse parameetreid ta võiks parandada. Selle jaoks võeti kasutusele närvivõrkudega masinõppimise meetod. Antud andmestik osutus masinõppimise jaoks keerukaks. Töö tulemusena valmis põhjalik andmete analüüs ning ebapiisava täpsusega masinõppe mudeli kood.

Rakenduse valmistamise põhilisteks raskuskohtadeks olid masinõpe ning Pipedrive avaliku APIga andmevahetus.

Valminud rakendus on kättesaadav aadressil www.pipedrivesalesvelocity.tk. Sisselogimiseks on vaja Pipedrive kontot.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 58 leheküljel, 8 peatükki, 33 joonist, 7 tabelit.

List of abbreviations and terms

ReactJS	Javascript Framework for Frontend applications
Pipeline	Set of stages that a prospect moves through, as they progress from a new lead to a customer
Kanban board	Agile project management tool
React Context	Component structure provided by the React framework, which enables sharing specific forms of data across all levels of the application
TensorFlow	Free and open-source software library for dataflow and differentiable programming across a range of tasks
Jupyter Notebook	Open document format based on JSON
Google Colab	Google research project created to help disseminate machine learning education and research
CSV	Comma-separated values (CSV) file is a delimited text file that uses a comma to separate values
Zeppelin	Web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala and more
Seaborn	Python data visualization library based on matplotlib
Pairplot	Set of graphs of pairwise relationships in a dataset
MAE	Mean Absolute Error is a measure of errors between paired observations
SPA	Single-page application. A web application or website that interacts with the web browser by dynamically rewriting the current web page, instead of the default method of the browser loading entire new pages
Gin-gonic	HTTP web framework written in Go
Golang, Go	Statically typed, compiled programming language designed at Google
DBMS	Database Management Software
DB	Database
DAO	Data Access Object. Pattern that provides an abstract interface to some

	type of database or other persistence mechanism
JSX	JavaScript XML. Syntax extension to JavaScript
Jest	JavaScript testing framework
AWS	Amazon Web Services. Cloud computing service provider
ECR	Amazon Elastic Container Registry. Managed AWS Docker registry service
EC2	Amazon Elastic Compute Cloud. Web service that provides compute capacity in the cloud
ECS	Amazon Elastic Container Service. Highly scalable, high performance container management service
S3	Amazon Simple Storage Service. Cloud-based data storage
Cloudfront	AWS content delivery network
Cloudwatch	Monitoring service for AWS cloud resources
Security Group	Virtual firewall for AWS instances

Table of contents

List of figures

List of tables

1 Introduction

Pipedrive is a cloud-based sales software company. It is the maker of the web application and mobile app Pipedrive, a sales customer relationship management (CRM) tool for salespeople in scaling companies. Pipedrive aims to improve the productivity of businesses through the use of their software.

Pipedrive has a lot of different features that all work together but are separated visually and are meant for different purposes. One of these functionalities is the Insights & Reports page which provides statistics about how logged in salesmen and other people in the same company are doing and how effective they are.

As part of this Insights page, Pipedrive wishes to let their users know how well they are currently doing as a salesperson by letting them know what their sales velocity is. Sales velocity is one of the key metrics for the salespeople to understand how successfully they are selling and also a way to compare their current results with their past results [1].

Sales Velocity (**S**) is an indicator, which is calculated based on 4 key variables: the number of opportunities in the pipeline (**A**), average deal size (**B**), win rate (**C**), and time it takes to win a sale (**D**). The following formula is used:

$$S = \frac{A * B * C}{D}$$

Equation 1 Sales velocity formula

The logical solution for this problem would have been to create the additional functionality as part of the Insights page. However, that would have required access to Pipedrive's internal API. Because only access to Pipedrive's Public API had been granted for the project, the only option left was to create a Marketplace application that Pipedrive users can add to their account.

2 Project scope

The project scope consisted of three concrete phases. In the first phase creation of an application which allows the user to calculate his sales velocity was required. After the completion of the second phase the user is able to change the values of his key variables to see how the change affects his sales velocity. The third phase was to suggest the most optimal change for increasing the user's sales velocity (Appendix 2).

The expectation of the third phase was unclear from the start. A question was posed to Pipedrive about how they expected the algorithm to suggest the most optimal change to the user. They initially suggested to look at which small changes in velocity parameters would result in the greatest change in sales velocity, and to simply suggest those changes.

It was explained to the Pipedrive mentor that it doesn't matter if you increase the parameter A, B or C by 10%, the resulting sales velocity increase would still be 10% (Equation 1). It was also explained by the team that a very effective suggestion for increasing sales velocity would simply be to lower the D parameter to have a value which approaches zero, so that the sales velocity approaches infinity. However, this is clearly not practical.

Instead, a meaningful suggestion should take into account which values are achievable in the real world for each parameter. Changes should be suggested based on where the salesman is underperforming.

Pipedrive didn't have any ideas about how exactly they wanted this problem to be approached, so they allowed the team to find the best approach themselves. It was decided to use a neural network-based machine learning approach, described in chapter 4.1.

2.1 Alternative scope

It was discussed to have an alternative plan in case the machine learning system didn't work. The Pipedrive mentors gave input that the most useful feature would be a filtering system incorporated into the application.

The system would have the possibility to select the user, the pipeline and the time period (this/last week/month/quarter) which would be displayed. Additionally, a 6-month history was added, which displays a sales velocity chart and has tooltips with all key metrics which the sales velocity is calculated from. Filtering can be seen as a simple feature, however the challenges and problems with them can be read about in chapter 5.1.2.

Fortunately, it turned out that allowing the users to view their history and compare themselves to other users can be used for the same purposes that the machine learning was meant for. By comparing themselves to others and seeing their past performance, the clients can understand which parameters they need to concentrate on.

2.2 User stories

User stories were given by the Pipedrive. Afterwards, they were discussed with the team and the following is the final user story list and visualization (Figure 1):

- As a salesperson, I want to know my sales velocity, so that I know my performance and can forecast the revenue more accurately.
- As a salesperson, I want to change the original value of key variables, so I can understand how it would affect my velocity.
- As a salesperson, I want to see how my sales velocity has changed during the last 6 months, so I can see how effective and productive I was during that period.
- As a salesperson, I want to be able to filter the results by time, pipeline and user.

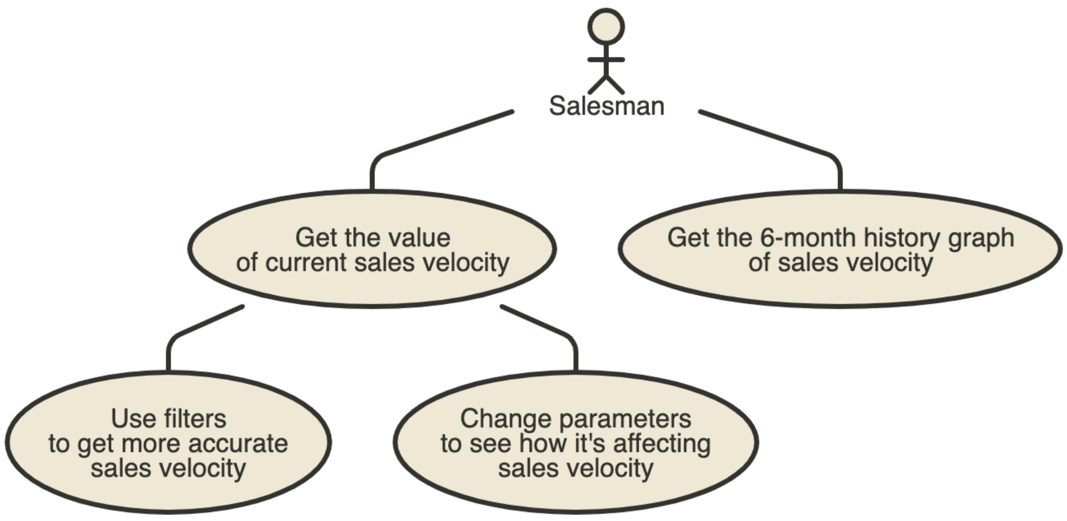

Salesman
Get the value of current sales velocity
Get the 6-month history graph of sales velocity
Use filters to get more accurate sales velocity
Change parameters to see how it's affecting sales velocity

Figure 1 User stories

3 Project description

Prior to the actual beginning of development, the project framework was set up:

- The roles and responsibilities were split.
- The rules, methodology, agreements, development guidelines and project structure were set.
- The tools and environments were set up.

It was agreed that Heino Sass will be a DevOps and backend engineer, Juri will be a frontend engineer and the team lead and Konstantin will be a full stack engineer. While doing machine learning Konstantin and Juri were performing the role of data engineers while Heino Sass assumed the role of ML engineer and data scientist

Two days were chosen for full-time work on this project (8h per day). Those days were Tuesday and Saturday for the first half of the semester. For the second half of the semester they were Wednesday and Saturday.

Prior to the emergency situation in Estonia, which started on 12th of May, work was done in Pipedrive's office, where tables with monitors and other needed equipment were provided for each team member.

As a communication channel Slack was used for internal text messaging communication and Zoom was used for video meetings. The main reasons were that every team member had some experience with those applications. Pipedrive mentors also use those, so communication with them was faster and easier for both sides.

Meetings with the Taltech mentor were organized twice a month at the beginning of the project. After COVID-19 it was decided to have meetings once a week. Meetings with Pipedrive managers were organized on-demand, when required. Once a week the progress of the team was written in Slack and it was asked whether Pipedrive mentors wanted to video conference with the team or not.

A Kanban board was used to organize tasks. Table 1 describes each stage which was on the board.

Table 1 Kanban board stages

Stage title	Description
Backlog	Tasks that had to be done later, not in the current milestone
Selected for development	Tasks that have to be done in current milestone, but not yet taken by any team member
In development	Tasks that are already taken and are being implemented
In review	Tasks are currently in review. There was a guideline that every PR should be approved by another teammate before it’s merged into the master branch
Done	Tasks that are completed

Tasks were grouped into milestones which served the role of a one sprint iteration. In the beginning, milestones were one week long. However, it became apparent that too much time was used for planning and management. Consequently, it was decided to extend the milestone duration. Therefore, in the second part of the project, milestones were 2 weeks long. After milestone planning the timeline graph was updated on the project's Wiki page (Appendix 4 - Project links).

Every working day was started with a morning standup. Backlog grooming was done at standups when needed. Retrospectives were done every two weeks.

Additionally, a daily project log was kept (Appendix 4 – Project links).

4 Project design

The high-level abstract design of the application is shown in Figure 2.

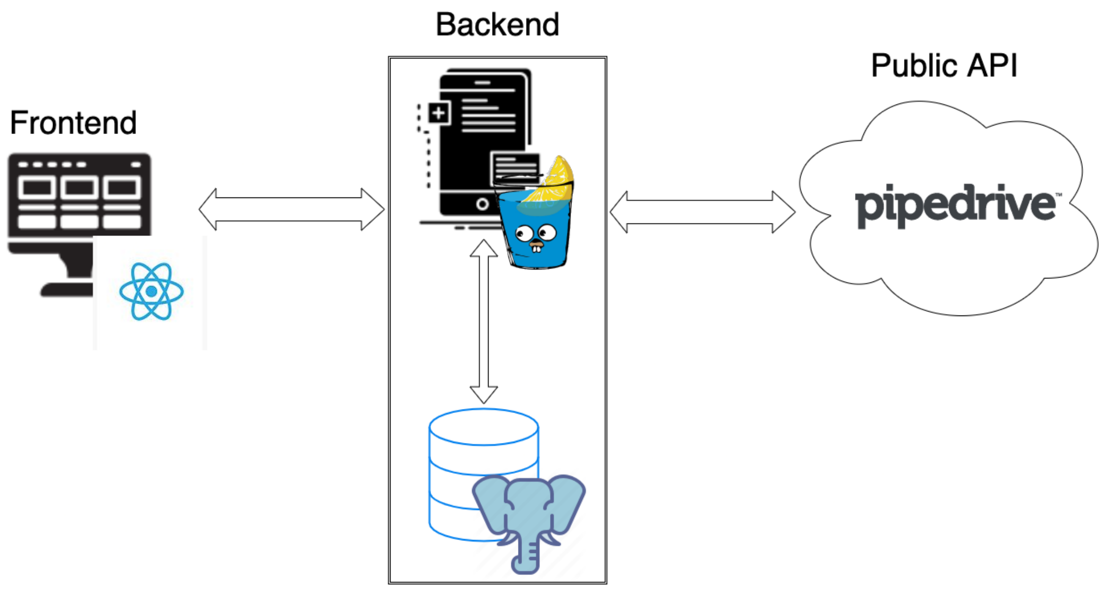


Figure 2 Application design

In the frontend ReactJS SPA (Single Page Application) is used, which sends HTTPS requests to the API. The API is a Golang application, in which the Gin-gonic REST framework is used. The Golang application stores information about users in a PostgreSQL database. It receives the data, which is needed for calculations, by querying the Pipedrive public API.

The decision of using PostgreSQL was made since it's a very popular relational DBMS (Database Management Software) [2] and each member of the team already had previous experience with it. The fact that each team member was already comfortable with the DBMS made it an easy choice. However, due to the architecture of the app and simplicity of the schema, it can be easily replaced by some other SQL database, or any other suitable data storage, such as Key-Value storages like Redis.

For the backend language the project team had a choice between Go and Node.js, which are the only two languages currently approved for making new internal services in Pipedrive.

Reasons why Go was chosen over Node.js:

- It is a statically typed language.
- It has opinionated styling rules (enforced with go fmt built-in tool). This helped save time by setting team-specific style rules, since the team consists of developers from different companies.
- It was a great opportunity to learn something new. The team had a lot of experience with Node.js, thus choosing Golang for the project was an opportunity to grow professionally and improve the authors' value on the job market.

4.1 Backend

The two key aspects of designing the backend were the database schema design and design of the backend application.

4.1.1 Database design

The choice of SQL database was partly reasoned by the fact that the plan was to add a velocity_params_record (see Figure 3) table for adding caching to the results.

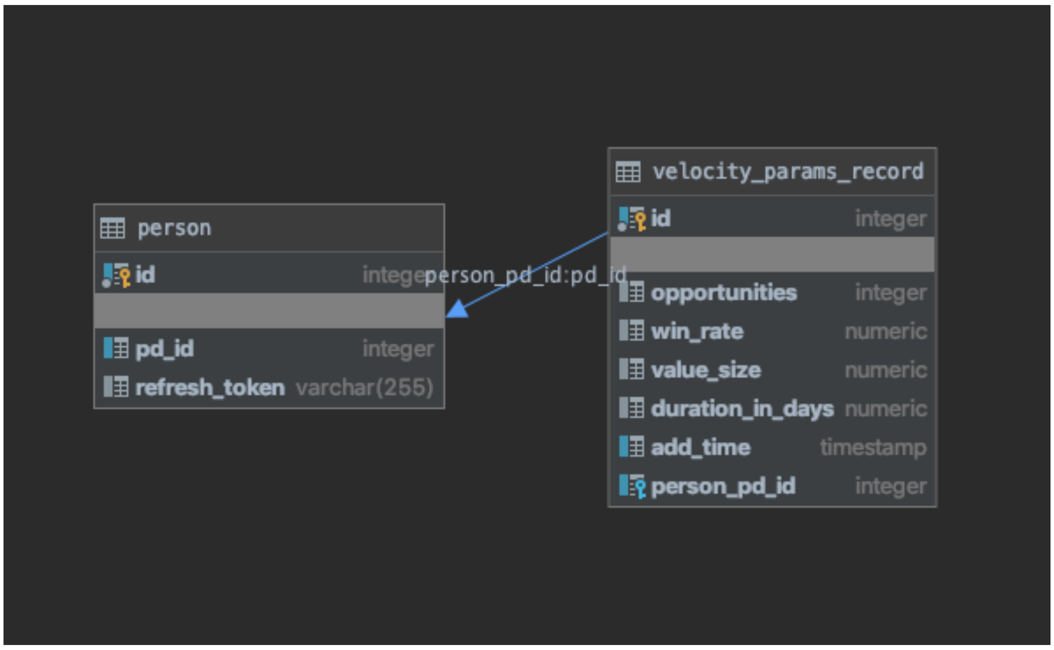


Figure 3 Initial database schema

It was considered both useful for performance and for analyzing the data for the machine learning part (since it was unknown initially if access to Pipedrive's data warehouse would be provided). For data analysis purposes it was also planned to have a company table which would contain useful metadata for predictions, such as the sales team size or the region of the company.

However, after time filtering appeared in the scope, caching became less relevant. It was initially not clear which filters were required to be implemented, and therefore how the caching should be implemented. Therefore, development of the caching functionality was delayed. Later, caching was removed from the scope of the project since it was not a high priority feature for Pipedrive.

Furthermore, access to Pipedrive's data warehouse was given for data analysis and machine learning purposes, so velocity parameters storage in the DB (database) became completely redundant.

Currently the database is used only for storing refresh tokens for the users.

4.1.2 Backend application design

A layered architecture (shown on Figure 4) with clear separation of concerns is used in the backend.

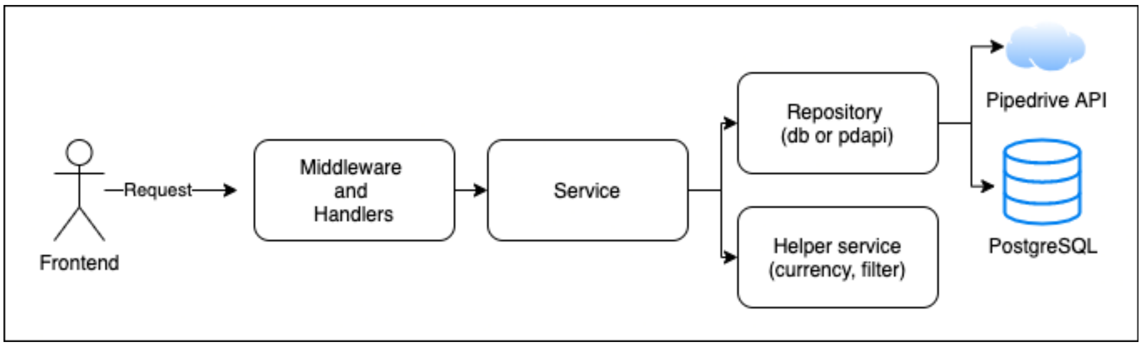


Figure 4 Backend architecture

The backend project has the directory structure described in Table 2.

Table 2 Backend application folder structure

Folder name	Purpose
controllers	Route handlers that extract parameters from request. After the requests are passed to services, the handler decides how to forward the service's response back to the client
middleware	CORS and authorization (Figure 7 in chapter 5.1.1) middleware
services	Folder containing packages with services. Services are packages that do most of the logic of the app, using pdapi and db packages
db	Containing DB migrations, DB instance initialization, and DAOs
pdapi	Wrappers for working with Pipedrive public API. In Figure 5 there are mappings between Pipedrive response data and Go structures
mocks	Auto-generated mocks done by make (Unix terminal command) generate
types	Data structures used across different packages
util	Helper functions used across different packages

dev-setup	Containing pre-commit scripts which should be installed in the developer’s machine
vendor	Folder with go modules (dependencies), which is used by the Docker container running an app

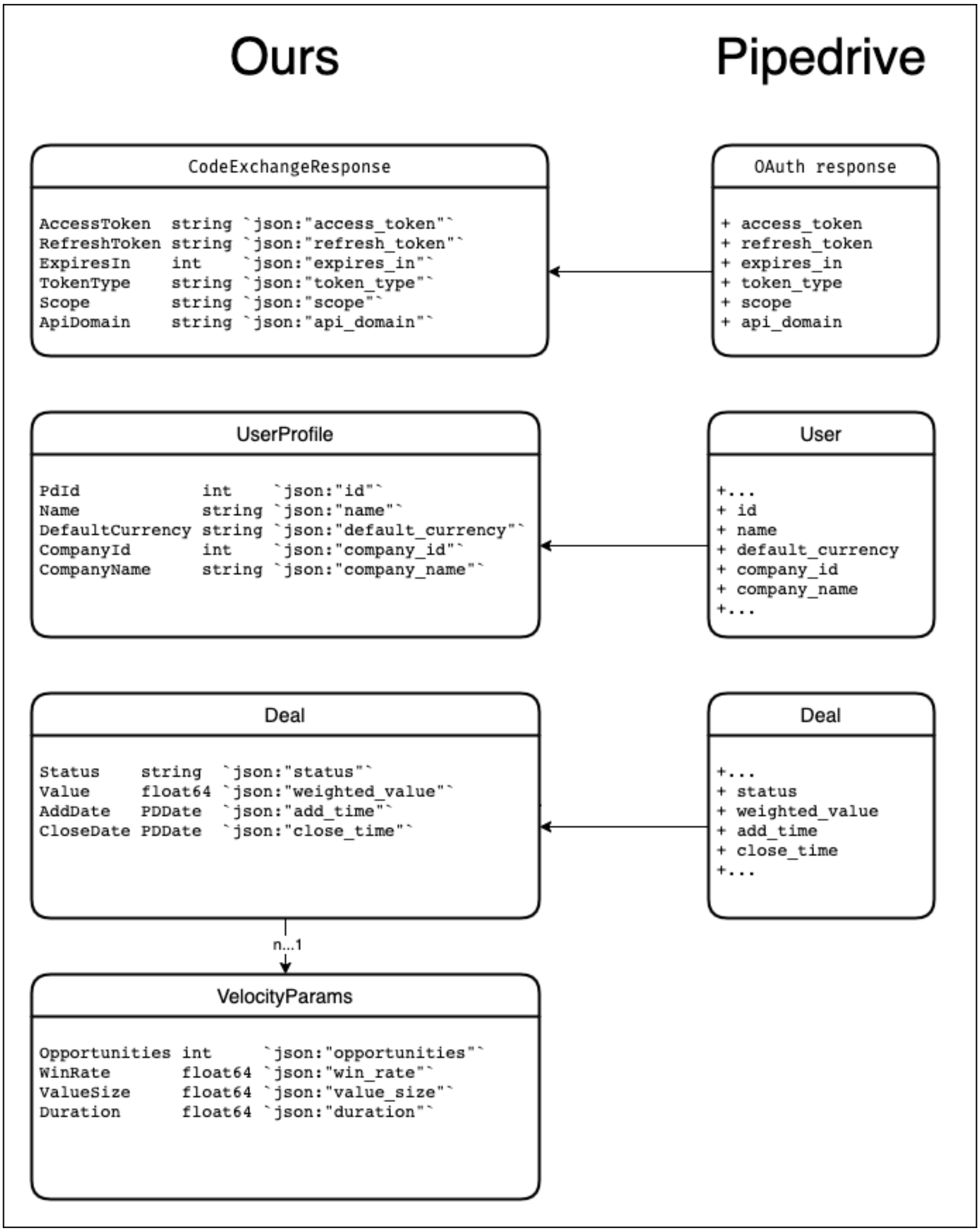
Also, a makefile with helper commands is present in the project. Table 3 contains the most relevant helper commands in the makefile:

Table 3 Most often used CLI commands

Command	Purpose
make setup install	Installs linter and prehooks
make setup install	Makes tidying of modules and generates the vendor folder, which is described above
make docker-prepare	Generates mocks and places them to the mocks folder

The code is covered by unit tests with coverage of around 80%. Functional tests, however, aren't present, since most of them would require valid access tokens, so trying to write those would not be worth the effort.

A custom open-source docker image that supports hot reloading is used [2]. Hot reloading makes it extremely practical to use the same environment as in production, while developing new functionality locally.



```
AccessToken  string `json:"access_token"`
RefreshToken string `json:"refresh_token"`
ExpiresIn    int    `json:"expires_in"`
TokenType    string `json:"token_type"`
Scope        string `json:"scope"`
ApiDomain    string `json:"api_domain"`
```

```
+ access_token
+ refresh_token
+ expires_in
+ token_type
+ scope
+ api_domain
```

```
PdId            int    `json:"id"`
Name            string `json:"name"`
DefaultCurrency string `json:"default_currency"`
CompanyId       int    `json:"company_id"`
CompanyName     string `json:"company_name"`
```

```
+...
+ id
+ name
+ default_currency
+ company_id
+ company_name
+...
```

```
Status    string  `json:"status"`
Value     float64 `json:"weighted_value"`
AddDate   PDDate  `json:"add_time"`
CloseDate PDDate  `json:"close_time"`
```

```
+...
+ status
+ weighted_value
+ add_time
+ close_time
+...
```

```
Opportunities int     `json:"opportunities"`
WinRate       float64 `json:"win_rate"`
ValueSize     float64 `json:"value_size"`
Duration      float64 `json:"duration"`
```

Figure 5 Data diagram

4.2 CI/CD integration

To ease the deployment process, it was decided to use Gitlab CI/CD. Having a CI/CD solution is useful for the project because deploying the application consists of many time-consuming steps, such as installing packages, building, testing, etc. Automating the process saves a significant amount of time. Additionally, any updates to the master branch will be immediately deployed to production, making it easy to check that the code, which works in a development environment, also works in production.

A preproduction environment could have been also set up to test that everything works before deploying the changes to production. That way any bugs will be spotted before the clients see them in production. In this project it was chosen not to do that, since it is not expected that there will be any future development of this app after it is added to the marketplace. Instead Pipedrive has said that they would like to integrate the functionality into the "Insights" view where a Pipedrive-specific CI/CD process would be used.

4.3 Frontend

The main technology for the frontend is ReactJS. It was an obvious choice, since it's the main frontend framework used in Pipedrive. Since the application code is intended to be integrated into Pipedrive's web application it was the only option.

For faster bootstrapping of the frontend Create React App was used - "an officially supported way to create single-page React applications. It offers a modern build setup with no configuration" [3]. It was also decided to use Typescript to improve code readability and ease of development by using optional types that Typescript provides. For writing CSS, the project team agreed to use the CSS modules approach, which is one of the defaults from the Create React App template. The benefit of this approach is that it is not necessary to worry about CSS class name overlapping anymore, because during compilation a unique hash is generated and added to the class name, thus avoiding any overlapping class definitions. Other than that, the CSS modules approach uses the classical CSS syntax.

4.3.1 Code design

Design principles that are promoted by ReactJS's community and documentation were followed when designing the Frontend part of the application. Although those principles might be interpreted in different ways and are not too strict, the team tried to follow the best practices while writing the Frontend code. The main pattern of React is creating small and reusable components that are responsible for a certain part of application. In this project the Functional components and React Hooks approach are used, where each component is just a function that returns a piece of the DOM. As it was decided to work with functional components, React Hooks are a great option for managing state, application context, and side-effects. One functional component is essentially a function written in JSX (JavaScript XML) syntax, which returns a small standalone piece of the application.

4.3.2 User interface design

The team did not include any designers. It was suggested by Pipedrive that UI designs could be asked from Pipedrive designers if necessary. However, it would have been excessive as this application has only 2 pages without a complex user interface. The first UI variations were created on paper and discussed inside the team. Further changes in UI and UX were done in the following manner:

- Design the idea, create a mock-up in HTML.
- Discuss it in a weekly meeting with the client, Pipedrive.
- Work on implementation.

Design patterns such as color scheme, spacings, fonts, font sizes etc. were chosen in such a way that it would look similar to Pipedrive's main web application. It helps to create the feeling that the Sales Velocity Calculator is a part of Pipedrive.

4.3.3 Testing

The frontend only has two main pages, a login page and a home page. Therefore, it can be tested manually without a strong need for automated tests. Nevertheless, it was decided to cover basic functionality and verify components' correct rendering with unit and smoke tests. The smoke tests just try to render the main components and check if this operation succeeded. This way it is ensured that the component will render correctly for the client and will not throw an error. Unit testing for components was also added to

check that all the elements that the user needs to interact with are rendered and visible. Unit testing was also used for testing simple JS functions.
For testing the frontend, Jest and React Testing Library were used. Jest is mainly used for assertions and basic testing functionality. React Testing Library is used for React components testing, which basically provides a convenient way of testing DOM nodes. That way what users see and what they interact with is tested, not the code itself.

5 Project contents

The following is the detailed description of the project implementation.

5.1 Backend

An authorized user (see chapter 5.1.1) has access to the endpoints described in Table 4.

Table 4 Accessible endpoints for authorized user

Endpoint	Parameters	Description
GET/filters/pipelines	-	Returns all pipelines that the user has access to. After that user can use their IDs in calculator requests
GET/filters/users	-	Same as pipelines, but returns list of users
GET/calculator/velocity_history	pipeline: int user: int	Returns array of sales velocity params for past 6 months
GET/calculator/velocity_params	pipeline: int user: int interval: Enum<String> {this_month, last_month, this_week, last_week, this_quarter, last_quarter and 6_months}	Returns sales velocity params using provided filters
GET/user/me	-	Returns information about the user, such as name and default currency

5.1.1 Auth

The first request a user makes after OAuth is *POST/send_code* [5].

Then the user's row in the database (storing his refresh token) is created or updated. Finally backend responds with the access token to the user.

From time to time, a client will call *GET/refresh_token* which will get the user's refresh token from the database and will use it to refresh the user's access token and return it to the user.

Every request (except *POST/sendCode*) should contain the AUTHORIZATION header with an access token. This access token will be checked in the auth middleware by fetching the user profile from the Pipedrive API. If fetching was successful, the middleware passes the request further. If it is not successful, which in most cases means that access token is expired, the backend will return a 401 to the client that will force the client to log out and do OAuth again.

You can see the authorization and corresponding middleware sequence diagrams on Figures 6 and 7.

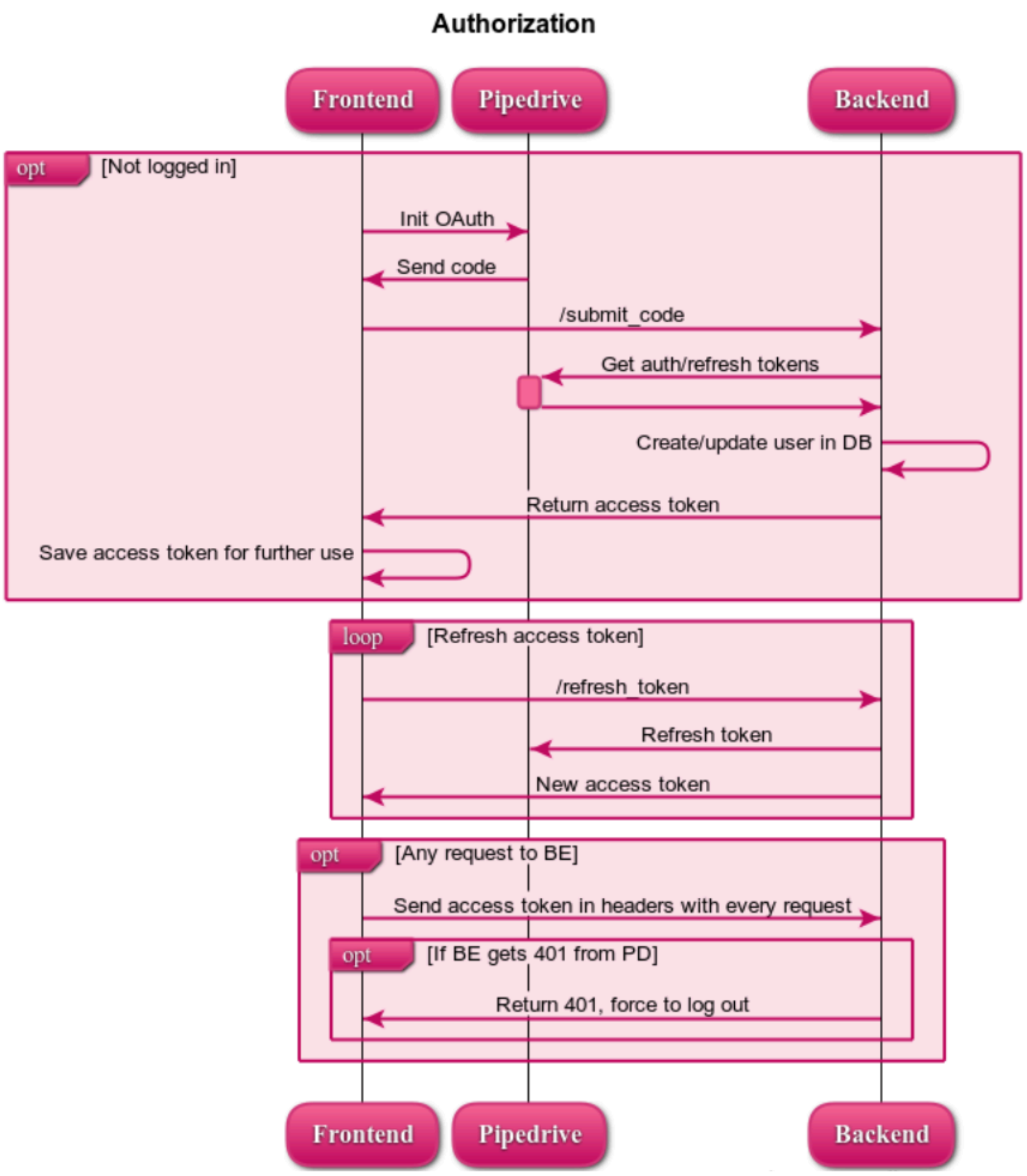


Figure 6 Authorization sequence diagram

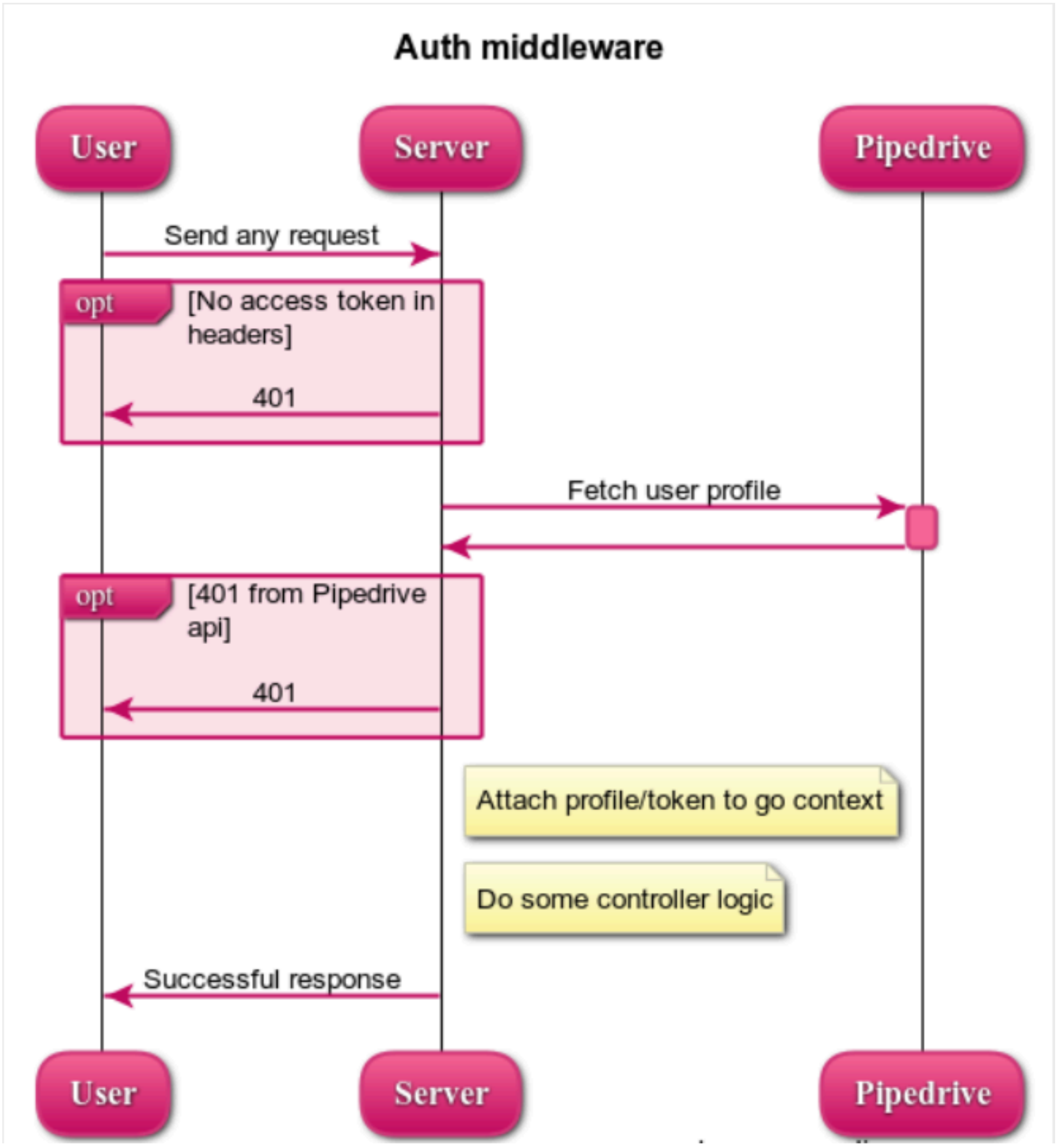


Figure 7 Authorization middleware sequence diagram

5.1.2 Filtering

Filtering data was one of the main architectural problems on the backend. In the beginning, it was discovered that a Pipedrive filter can be applied when making a call to the "/deals" Pipedrive API endpoint by including a filter ID with the request. Filters in Pipedrive are separate and powerful entities that can combine any conditions using different logical operations, data fields and helper functions/variables (as relative dates which are used in the application for filtering). However, it has one obvious drawback in the case of Sales Velocity Calculator - namely all created filters will be visible to the user in their Pipedrive user interface. So it could not be afforded to just create a separate filter for every combination of a pipeline, user and period.

Considering this fact, and after discussion with Pipedrive Product Managers, it was decided to create 7 filters (6 relative dates + half-year period for the history functionality) for each user (which are also only visible to him). Pipeline and user filtering happen on the application side.

In the last iteration, the team was notified that it is needed to make a slight change in the calculation process which has brought another problem - the application side had to have some kind of time filtering on it as well. After implementing time helpers for relative periods, Pipedrive filters have become redundant in terms of functionality - users still get the same result with or without it. However, it has performance value in case of long-lived companies, so it was not removed.

A sequence diagram with detailed explanations about how Pipedrive filtering creation happens is shown on Figure 8.

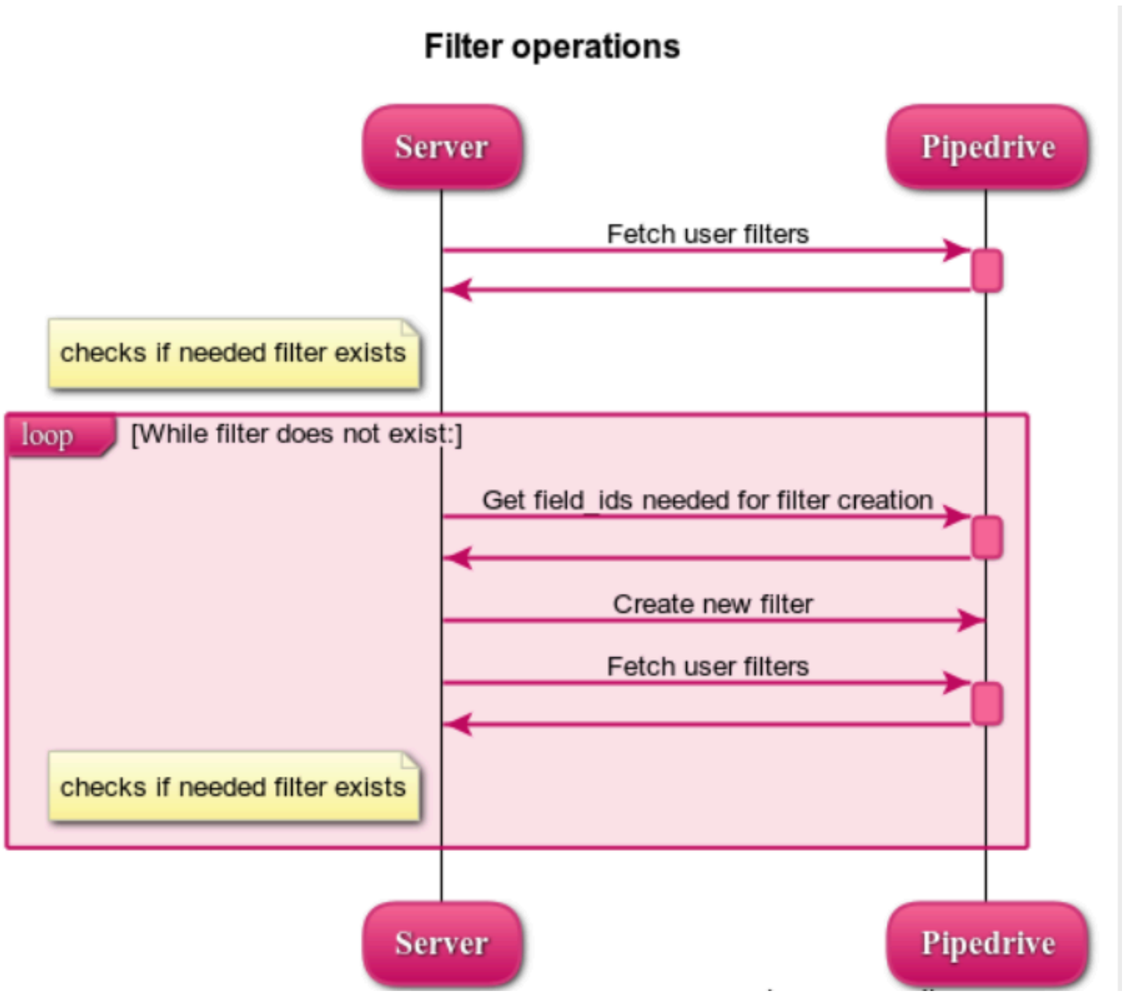

Filter operations
Server
Pipedrive
Fetch user filters
checks if needed filter exists
loop
[While filter does not exist:]
Get field_ids needed for filter creation
Create new filter
Fetch user filters
checks if needed filter exists
Server
Pipedrive

Figure 8 Pipedrive filter creation sequence diagram

5.2 Frontend

The final frontend part contains 2 main pages - Login and Home. Which page to render is decided by whether the user is logged in or not. On first landing it is checked whether the access token is present in browser local storage. If the token is present, the validity of the token is checked by making a request to the backend. If the token is not valid, then the user should first login before using the application. In case the token is valid, the user is redirected to the home page, otherwise the user is asked to log in to the application by redirecting him to Pipedrive's OAuth page. Right after login, when the user lands on the Home page, information about the user is queried, including available filters for this specific user, and save those to the context of the application. React Context[1] is used because 2 different parts of the Home page use this information, the calculator and the graph. Thus there is no point in passing this information separately. Component specific data is fetched from the backend when the component is loaded. For example, when loading HistoryContainer, which is responsible for showing the graph, */calculator/velocity_history* is queried and when loading the calculator component - InputContainer */calculator/velocity_params* is queried.

All the requests to the backend should have an *Authorization* header which contains the valid access token. If the token is not specified or status 401 (Unauthorized) is received then the user is being redirected to the Login page.

5.2.1 State management

In this project 2 types of state management are used. The first one is the state for the component if this is a complex component with a lot of different states. The second one is the global application state. In both cases there is no need to use separate libraries like Redux (Predictable State Container for JS Apps) as ReactJS allows to implement the same approach using React Context and useReducer hook. UseReducer hook allows to specify a separate reducer and initial state and gives the current state of the application and a dispatch function. By passing the state object and the dispatch function via context is it possible to change the global state of the application from any component and also receive this change made by other components.

5.2.2 Typings

As Typescript was chosen, the project has a separate folder for typings - *types*. Types are used to specify different object structures that are used around the project. One of the common use cases for types in the project is action payload definitions for state reducers and app state type definitions.

[1] https://reactjs.org/docs/context.html

5.2.3 Environment configuration

In this project only 2 environments are present: development and production. For making this possible *config.ts* was made which has different values for development and production environments. The selection is happening based on the node environment (*process.env.NODE_ENV* variable). This approach separates the configuration from the implementation logic, so the developer does not need to worry about specifying the right value for the different environments every time and can just use *config.apiUrl* variable for example.

5.3 Machine learning

The following is a detailed explanation of the research and development which was undertaken in an effort to use machine learning to give improvement suggestions to users.

5.3.1 The idea

Let's assume there is a large dataset of clients. In the dataset there are the four parameters for calculating sales velocity. Additionally, there is useful metadata, such as the industry, company size, geographical location, etc.

It is possible to train a machine learning algorithm to predict one of the four parameters based on the other parameters, given that the training data includes those parameters.

The predicted parameter may be viewed as the expected value of that parameter, or in other words, the normal value a client might achieve, given his other parameters. It is important to note that the sales velocity should not be used for predicting the parameter, because otherwise it would be a simple calculation.

If the real value of the parameter is lower than what is predicted, then the client can be given feedback that the parameter could be improved. If the real life value is higher than what is predicted, then feedback can be given that he or she is doing well.

This method can be repeated for all parameters, and thereby find out where the client is successful and which parameters could be improved.

5.3.2 Method

For predicting the values, a neural network would be created with one input layer, one middle layer and an output layer. The output layer would have one neuron, which will be the predicted value.

The calculations are handled by TensorFlow. The code is based on TensorFlow's basic regression learning tutorial [4].

Google Colab was used for running the code. It was an extremely convenient option because the required libraries are immediately available there. It also made it possible to share the code and cooperate more quickly and efficiently. The document is a Jupyter notebook, which is hosted on Google Colab. A copy of the Jupyter notebook has been included with the application's source code.

The training data is stored in a CSV (comma-separated values) file, which must be uploaded to the user's Google Drive. The CSV has been exported from Pipedrive's data warehouse.

5.3.3 Data gathering

All the data was taken from Pipedrive's data warehouse. This was the easiest way to gather all the needed data for the analysis from one place. For data gathering standard SQL queries were used. Pipedrive's data warehouse uses the tool Zeppelin which uses a specific SQL syntax - Hive SQL.

Table 5 describes the data fields which were imported from Pipedrive's data warehouse. These parameters are all considered to be "main" parameters and will be referred to as such in the future. The only exception is the sales velocity, which is not considered to be a main parameter.

Table 5 Main sales velocity parameters' explanations

Opp	The number of deals with the status "won" or "lost". In other words, the number of all closed deals.
Value	The average value of selected deals, in euros. Deal values in other currencies were converted to euros using the

	exchange rate of the day when the deal was closed.
Period	The average amount of time it takes to close a deal, in seconds.
win_rate	The number of won deals divided by the number of closed deals.
vp_result	The resulting sales velocity, in euros per day.

Table 6 describes data fields which were imported from Pipedrive's data warehouse. These parameters are all considered to be "extra" parameters and will be referred to as such in the future.

Table 6 Extra sales velocity parameters' explanations

country	The country of the user. It is an ISO 3166 two character country code.
tier_code	The tier code of the user's Pipedrive subscription. It has four possible values, ranging from "silver" to "diamond".
industry	The industry of the user.
company_size	The size of the company the user is working in, as a category.
Region	The region of the user. Either Europe or the United States.

As an effort to only gather relevant data, a number of filters were used to get rid of unwanted data points. Only the following deals were included:

- Deals that are already closed - either won or lost state.
- Deals that have a value bigger than 0.
- Deals from companies that are on the "paying" plan and were created after 2016.

These filters are relevant for a set of reasons:

- Analysing only closed deals was opportune, because open deals did not have any value in terms of training. This is because the predicted sales velocity parameters were based on closed deals.

- Deals with value 0 or a negative value have very specific use cases and should not be considered when calculating sales velocity.
- Companies created after the year 2016 were used because of Pipedrive's instructions to use only a smaller subset of the total dataset. Therefore this was a way to reduce the size of the dataset and simultaneously be able to work with more recent data.
- Only companies on the "paying" plan were used as they are more likely to be active users of Pipedrive.

The amount of data in Pipedrive's data warehouse was more than what was needed for machine learning. This was also one of the requirements from Pipedrive - not to use all the data. Therefore, only 20% of data was used, even before the application of additional filters. After the application of filters, the total amount of data points was 6092.

All the currencies were converted to euros for ease of analysis. For currency conversion an exchange rate of the two currencies at the time the deal was closed was used.

All the exported data was exported without any user- or company keys as this data should not be traceable back to specific users. This was also one of the requirements from the Pipedrive's data protection department.

In the first iterations of data gathering and analysis, Zeppelin was used for creating scatter plots and histograms. However, because Zeppelin executes queries from the web interface and, in this case, on a big dataset, it was performing really slowly. One query could take up to 3-4 minutes. Zeppelin also had a learning curve and it turned out to be much easier to try out different ideas in Python instead. All this meant Zeppelin was better used only for checking basic ideas and exporting data. Further analysis was done in Python.

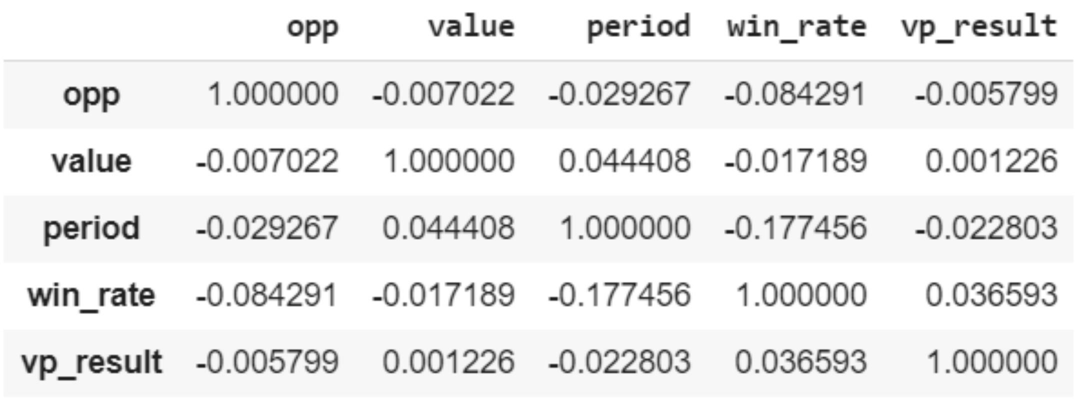
5.3.4 Data analysis introduction

Before attempting to build a machine learning model, it is extremely useful to have a thorough analysis of the training data. That way more informed decisions can be about the learning process.

Therefore, possible correlations between the learning parameters were examined.

5.3.5 Correlation coefficients

Figure 9 shows Pearson correlation coefficients, which were calculated for all the parameters which are not class-based.



	opp	value	period	win_rate	vp_result
opp	1.000000	-0.007022	-0.029267	-0.084291	-0.005799
value	-0.007022	1.000000	0.044408	-0.017189	0.001226
period	-0.029267	0.044408	1.000000	-0.177456	-0.022803
win_rate	-0.084291	-0.017189	-0.177456	1.000000	0.036593
vp_result	-0.005799	0.001226	-0.022803	0.036593	1.000000

Figure 9 Sales velocity correlation coefficients

It would be expected to see a fairly strong linear correlation between the main parameters and the sales velocity, because the sales velocity is calculated based on the main parameters. However, this is not the case, and the coefficients show almost no linear correlations.

The reason for this may be that different combinations of parameters can result in the same sales velocity, and correlations are nonlinear.

For example, if the value parameter is 10000 euros, then that can result in a very large or very small sales velocity, depending on the other parameters. Although intuitively it might seem that larger values should correspond to larger sales velocities, the data turns out to be more chaotic.

The strongest linear correlation is between the win rate and the period.

5.3.6 Main parameters' scatter plots

While correlation coefficients didn't show any large linear correlations between parameters, it was hoped that scatter plots would reveal any nonlinear correlations.

The plots were constructed using the Seaborn python library. On the top-left to bottom-right diagonal of the graphs are histograms of the parameters. All the other graphs are scatter plots, which show what the pairwise values were for each data point.

Figure 10 shows a pairplot between the main parameters and the sales velocity. 10% of the total amount of exported data points, 609 data points, were used for easier readability of the data. Unless stated otherwise, the same will be done in future scatter plots as well.

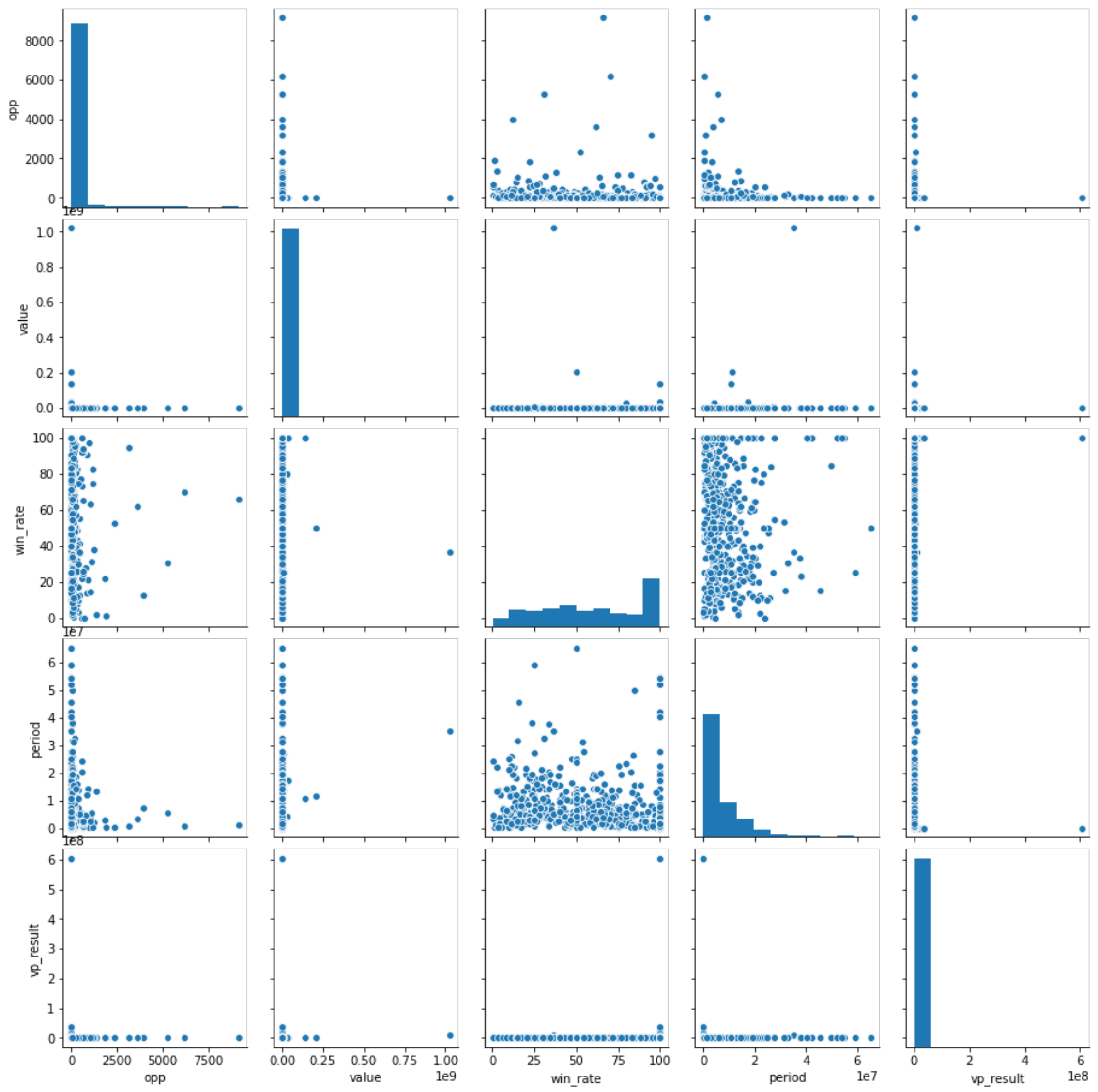


Figure 10 Scatter plot between main parameters and sales velocity

At first glance, the results seem quite messy. The win rate graphs don't seem to show any meaningful correlations, and many plots (like *value* and *vp_result*) are rendered almost unreadable due to some extremely high outlier values.

Therefore, extremely large values were filtered out, which were outliers in the general data set (over the 99th percentile). The cut-off points can be seen in Table 7.

Table 7 Sales velocity filtered 99th percentile values

sales velocity	over 169403 euros per day
Value	over 753651 euros
opportunities	over 618
Period	over 23220796 seconds, or 268 days

As a result of this filtering, the amount of data points was reduced to 583. Figure 11 shows the updated pairplot.

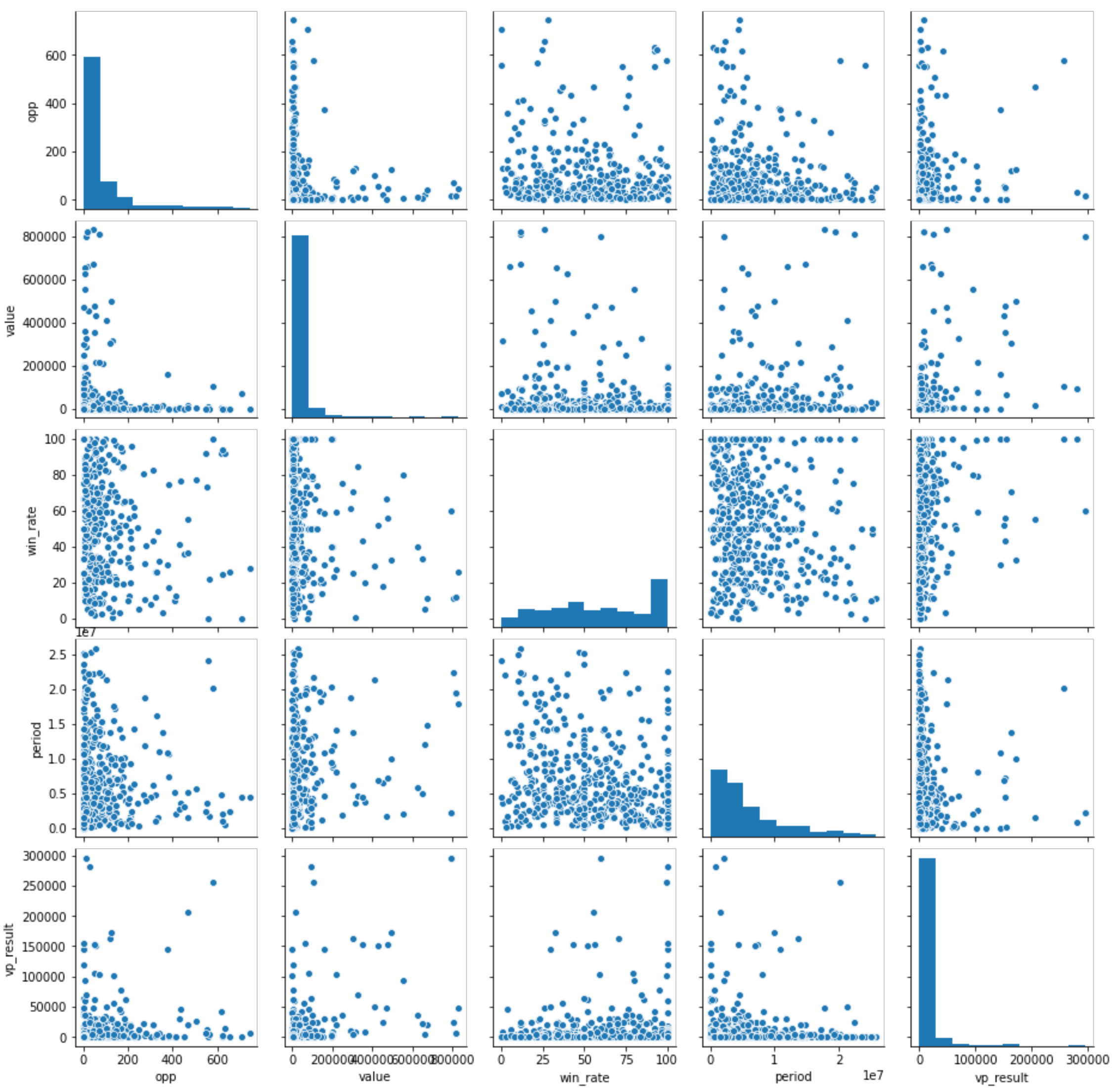


Figure 11 Scatter plot between main parameters and sales velocity with filtered data

The data still doesn't show anything too promising. However, some observations can be made:

- There exist many users who have a 100% win rate (19.5% of total data points).
- When the deal size is high, then the number of opportunities tends to be low, and vice versa.
- High values in the period tend to be accompanied by somewhat lower values in the sales velocity.
- High sales velocities more commonly occur when the number of opportunities is low and vice versa. However, the correlation is not as strong as the correlation between the deal size and the number of opportunities.
- High sales velocities more commonly occur when the period is low.

5.3.7 Main parameter correlation conclusions

The following conclusions were made:

- There are some nonlinear correlations in the data between certain parameters.
- A surprisingly high amount of users have a near 100% win rate.

More research is needed to determine if and to what extent the data is inaccurate because of the fact that people use Pipedrive in unexpected ways. For example, it is not known if the win rate might be influenced by people who delete (or simply do not mark down) unsuccessfully closed deals.

5.3.8 Problems with near 100% win rates

Roughly 20% of data points have a win rate of over (or equal to) 99%. From those users' data set, the median number of opportunities is 2.

Figure 12 is a histogram of users with over (or equal to) 99% win rate, zoomed in to under 100 opportunities. The x axis represents the number of opportunities and the y axis is the fraction of total users who have that amount of opportunities.

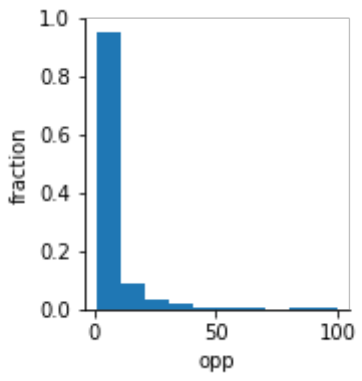


Figure 12 Histogram of users' opportunities with an above 99% win rate

It's visible that most users with very high win rates have a very low amount of opportunities. The question arises, if users with a low amount of opportunities should even be included in the data set.

Additionally, the reason it was necessary to zoom in on the histogram is because there are also companies with thousands of opportunities, who also have 100% win rates. More research is required to identify if such a high win rate is standard for their industry. Perhaps it’s possible that those users are deleting their failed deals, or simply neglecting to write them down.

5.3.9 Problems with extreme sales velocities

The highest sales velocity in the data set is 19014754040 (nineteen billion) euros per day. This seems infeasible.

Analyzing the top 1 percentile of the data in terms of sales velocity, it turns out most of these users (82%) have a 100% win rate. 61% of these users also have 4 or fewer opportunities, while only 20% of the overall data set has 4 or fewer opportunities. These users are also characterized by very low sales periods and high value deals. The median sales period is 58.6 seconds, or roughly one minute, while the median value of a deal is 26409 euros.

In conclusion, the data set has problematic users who close their deals immediately, thereby creating an unrealistically high sales velocity.

5.3.10 Problems with inconsistent user behaviour

Pipedrive is a platform where users are free to mark down their deals however they like. By marking down their deals in different ways, users can artificially increase or decrease their sales velocity (as evidenced by the high win rates and low sales periods of some users, which results in a high sales velocity).

Given this fact, there is a risk of telling a salesman that he is underperforming, while in reality, he is simply marking down his deals correctly, while others are not doing so.

A possible solution to this problem is to only take into account users from the same company when making predictions. In that case each company would have to make their users mark down their deals in the same way.

5.3.11 Extra parameters’ correlation to sales velocity

The goal was to use the extra parameters to aid in the learning process. Therefore, it was attempted to see if there is a connection between the extra parameters and the sales velocity to gain a better understanding of how the sales velocity is affected by the extra parameters. Another hope was to spot potential problems, similar to the 100% win rates in section 5.3.8.

An integer was assigned to each class for the *country* parameter, because there are too many countries to display them as labels. Once again, the extremely high sales velocities in the top 1st percentile had to be filtered out to be able to read the scatter plot. As with all scatter plots, only 10% of the total data set was included to improve readability. Scatter plots between the extra parameters and the sales velocity were created.

Figure 13 shows scatter plots between *company_size, region* and *vp_result*. The purpose of the scatter plots was to find noticeable differences in sales velocity between different classes of extra parameters. This would indicate how the sales velocity correlates to the different classes.

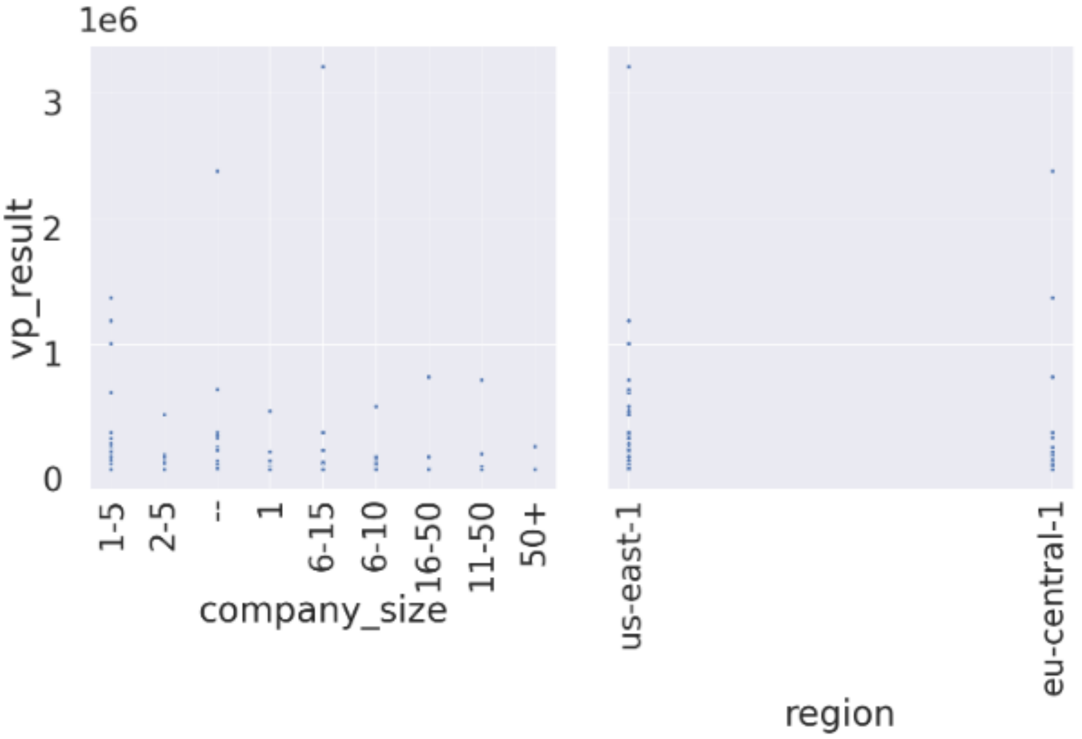


Figure 13 Scatter plots between extra parameters and sales velocity

Different company sizes have differences in sales velocities. Companies with the size "2-5" achieve lower sales velocities than companies with no specified size ("—"). Although companies with the size "1-5" have the most amount of data points with a sales velocity of over 1 million euros per day, then they are by far the largest group (Appendix 1 – company size). Assuming there is a small probability for any individual data point to be an outlier who has a high sales velocity, it is no surprise that the largest group would have the most outliers. The difference between the "2-5" and "—" group is more interesting because the "—" group is not bigger than the "2-5" group yet they have more data points with a high sales velocity.

It's also visible that the United States region seems to have a few more users with high sales velocities than the European region. However, there is still significant overlap between the two.

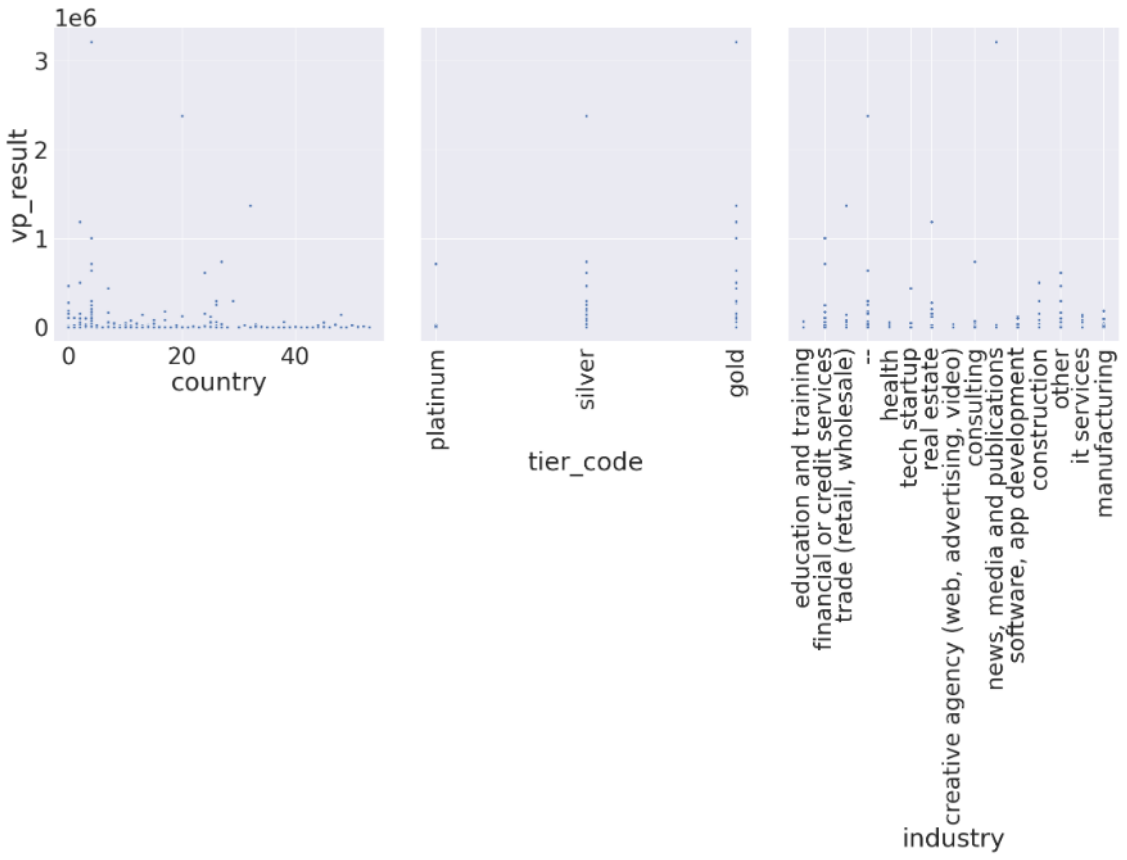


Figure 14 Scatter plots between extra parameters and sales velocity

Figure 14 shows scatter plots between the sales velocity and the remaining three extra parameters. The first easy-to-spot pattern seems to be that some countries have higher sales velocities than others. There are also differences in sales velocities between different industries.

Conclusions cannot be made about “platinum” and “diamond” tier codes because there are too few of them in the data set. There seems to be a lot of overlap between the “silver” and “gold” tier code users in terms of their sales velocities.

Deciding to investigate the correlation between countries and sales velocities further, a pairplot of the two parameters was created.

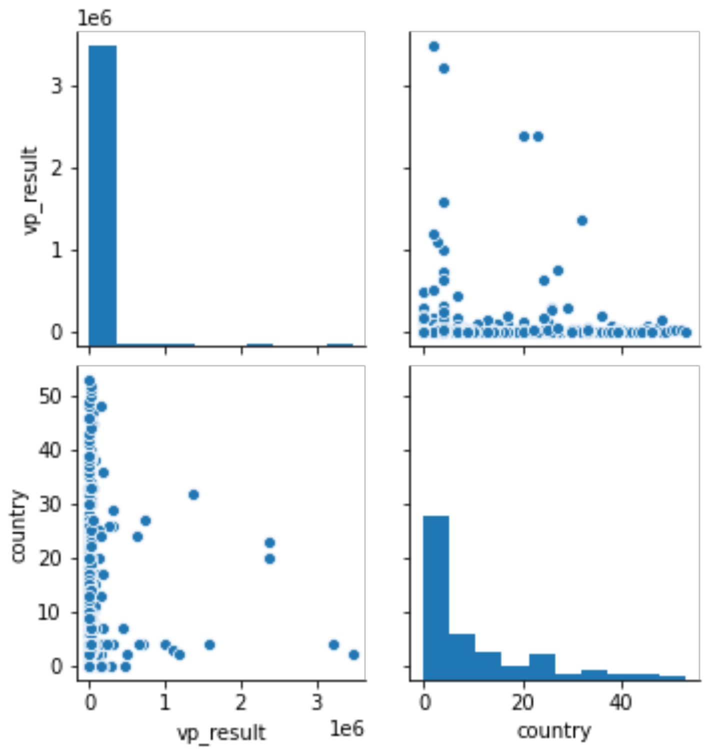


Figure 15 Pairplot between the country and sales velocity parameters

Figure 15 shows that some countries have many people who have high sales velocities, while others have none.

However, it should be noted that the countries with the most data points are, in this case, generally located to the left of the upper right graph. Therefore, if there is some low probability of a country having a person with a very high sales velocity, then it is no surprise that the countries on the right of the graph might have none (due to a small number of data points).

5.3.12 Extra parameters' correlation to sales velocity conclusion

The extra parameters do seem to have some correlation to the sales velocity. Different countries, industries and company sizes have different sales velocities. However, there is overlap and the correlations aren't as strong as it could have been hoped.

However, whether the extra parameters will be included in the training process will be determined by how strong their correlations with the main parameters are.

5.3.13 Extra parameters' correlation to main parameters

Because the goal is to predict main parameters, it should be analysed how the extra parameters correlate with the main parameters if it's desired to use the extra parameters to aid in the learning process.

The "country" parameter was converted to an integer, since there are too many labels and they're too small to read otherwise. 10% of the dataset was used for creating the graphs.

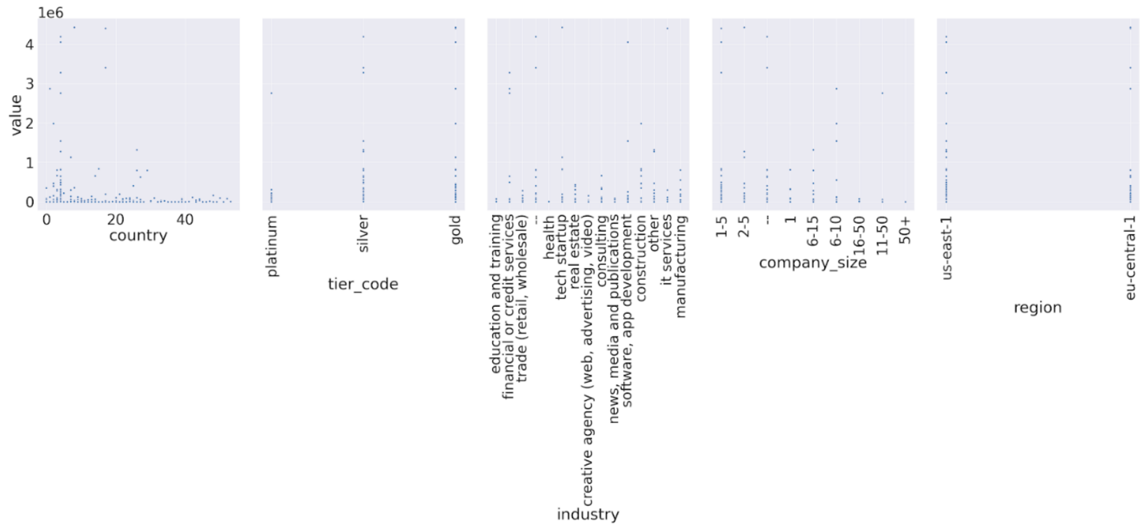


Figure 16 Pairplot between "value" and "industry" parameters

Observations about the “value” parameter were derived from Figure 16:

- Companies with the “platinum” tier code have fewer total high value deals. But also, from the histogram (Appendix 1 – tier code) it’s visible that there are far fewer companies with the “platinum” tier code.
- Some countries make more high value deals than others. Also, comparing the graph to the histogram (Appendix 1 - country), it can be seen that the countries located at the right of the graph, which have fewer data points, also have fewer high value deals.
- There are some noticeable differences in industry deal sizes.
- There is only a small difference between regions.

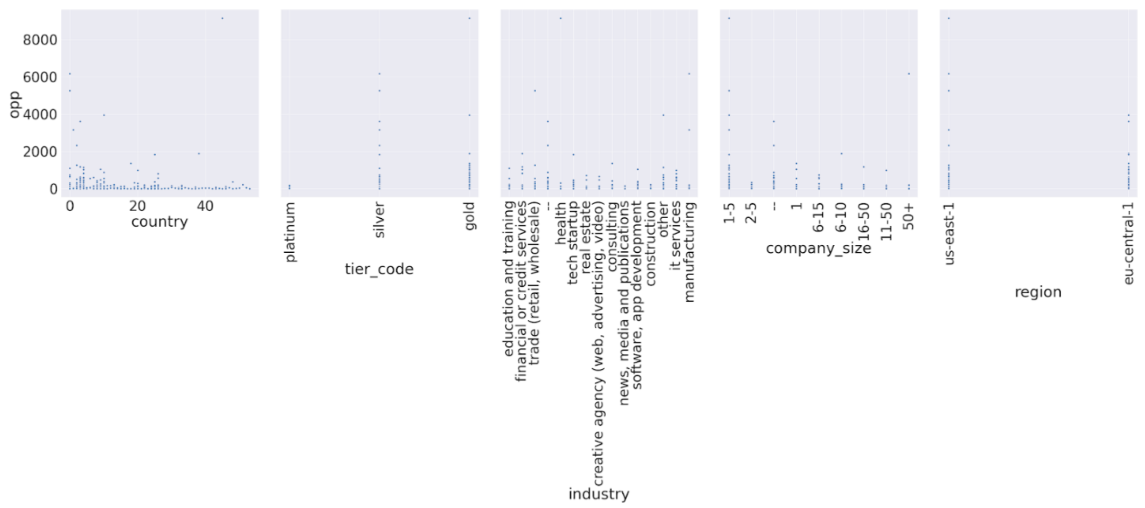


Figure 17 Pairplot between "opp" and "industry" parameters

Observations about the "opp" parameter were derived from Figure 17:

- There are differences between company sizes. For example, people in companies with the size "2-5" seem to have fewer high values in the opportunity parameter than people in companies with sizes "1-5" and "—".
- There are differences between countries.
- There are no real differences between regions.

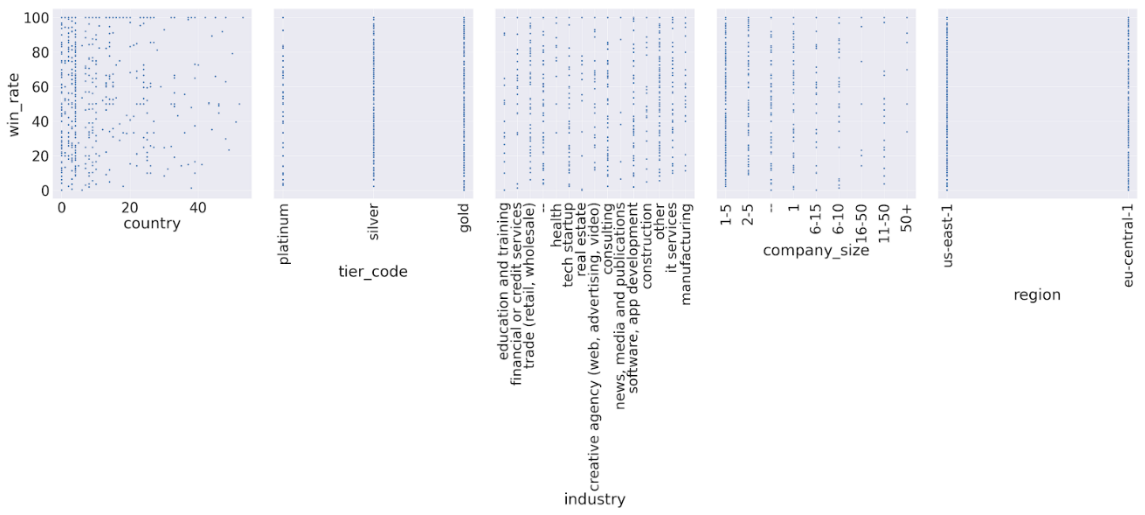


Figure 18 Pairplot between "win_rate" and "industry" parameters

Observations about the "win_rate" parameter were derived from Figure 18:

- The extra parameters do not seem to be correlated to the win rate.

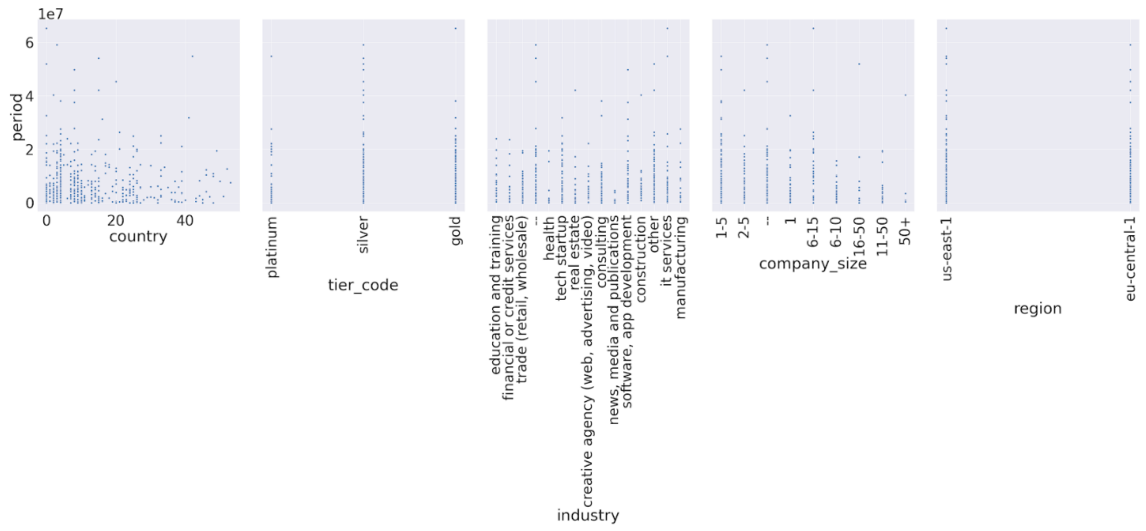


Figure 19 Pairplot between "period" and "industry" parameters

Observations about the "period" parameter were derived from Figure 19:

- There is much less correlation between extra parameters and the *period* parameter, than between extra parameters and the *value* and *opp* parameters.
- There does not seem to be any correlation to the *region* and *tier_code* parameters.

5.3.14 Extra parameters' correlation to main parameters conclusion

It was concluded that the extra parameters are mainly useful when predicting the *opp* and *value* main parameters. They are of lesser usefulness when predicting the *period* and *win_rate* parameters.

The region should not be used for predictions. The most useful extra parameters seem to be the industry and the size of the company.

5.3.15 Initial training results with only main parameters

It was attempted to train the neural network to predict the *value* parameter. Only the main parameters were used for training.

After 1000 epochs of training, there was no noticeable improvement in the training result's MAE (Mean Absolute Error). The straight line in Figure 20 is the training MAE. The dotted line represents the MAE for the validation set.

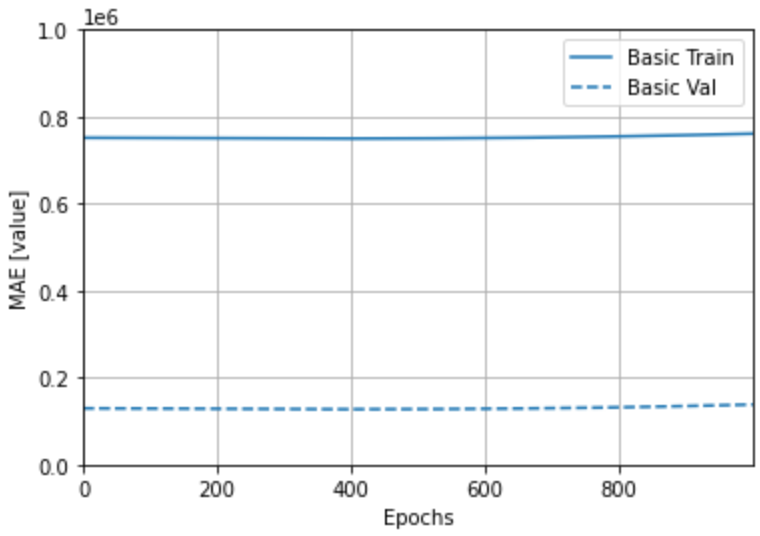


Figure 20 Initial training with main parameters MAE

The final testing set mean absolute error was 172731 euros.

Figure 21 shows how predictions correlate with actual true results. Ideally the graph should be a diagonal line, where the predictions match the true values as closely as possible.

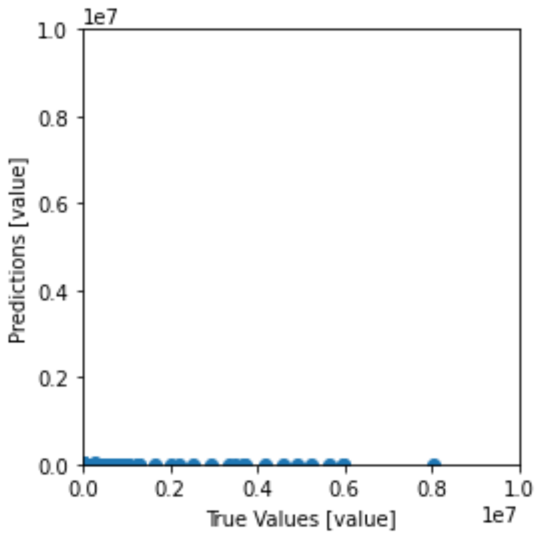


Figure 21 Initial training with main parameters prediction and true value comparison

The predictions are consistently low, even if the true values are high. Figure 22 is the same graph, zoomed in:

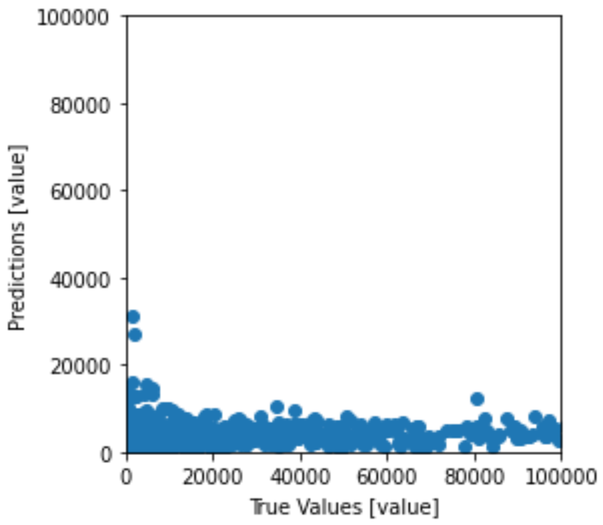


Figure 22 Initial training with main parameters prediction and true value zoomed in comparison

Most of the predictions fall between 0 and 20000 euros. The training was entirely insufficient.

5.3.16 Initial training results with extra parameters

For training, the extra parameters *industry* and *company_size* were included. The parameters *tier_code* and *region* were not used because the data analysis didn't show them to be promising candidates. The parameter *country* was not included due to overfitting concerns.

The class-based data was split into separate columns, where a "0" indicates the row does not belong to that class, and a "1" indicates the row does belong to that class.

As seen from Figure 23, after 1000 epochs of training, there was no noticeable improvement in the training results (Mean Absolute Error).

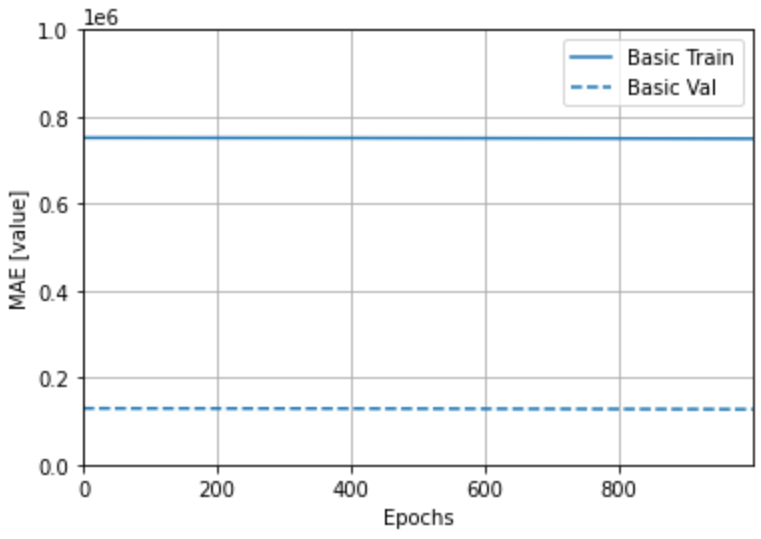


Figure 23 Initial training with extra parameters MAE

The testing set mean absolute error was 184829 euros. The results were worse on average this time.

Figure 24 shows how predictions correlate with actual true results. Ideally the graph should be a diagonal line, where the predictions match the true values as closely as possible.

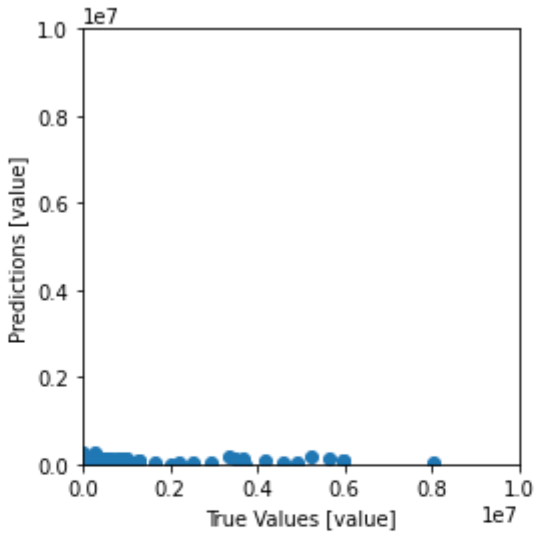


Figure 24 Initial training with extra parameters prediction and true value comparison

Figure 24 shows that predictions are consistently low, even if the true values are high.

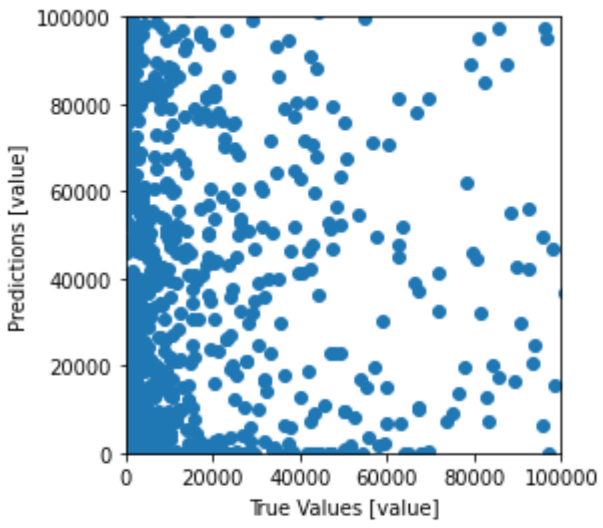


Figure 25 Initial training with extra parameters prediction and true value zoomed in comparison

Figure 25 shows that this time much more variability was seen in the predictions than when the training was done with only main parameters. The training results were worsened by the inclusion of the extra parameters.

5.3.17 Data filtering of low opportunities

It was attempted to solve some of the problems highlighted in the machine learning "Problems" sections 5.3.8 and 5.3.9. This was attempted by filtering out data points which are not adequate for training. The first thing to be filtered was data points with a low amount of opportunities.

Keeping data points with a low amount of opportunities was undesirable because:

- It is desirable for the training and predictions to be based on the parameters of long-term, active users.
- Users with a low amount of opportunities in the dataset have more "anomalies", such as 100% win rates and extreme sales velocities (Figure 26).

Originally, ~20% of data points in the total data set had a 100% win rate. It was also discovered that the vast majority of users with a 100% win rate have a low amount of opportunities.

Figure 26 shows how the number of data points with a 100% win rate goes down as the number of opportunities is increased. The number of data points is a percentage of the total amount of data points. The result is obtained by dividing the number of data points, which have a 100% win rate and meet the minimum opportunities criteria, with the total number of data points, which meet the minimum opportunities criteria.

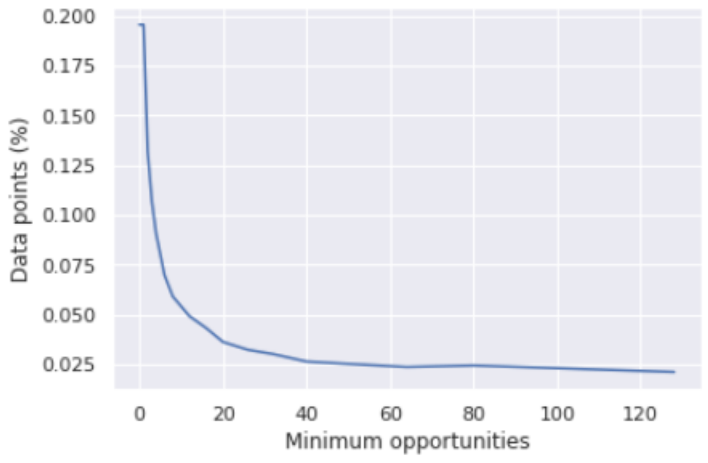


Figure 26 Minimum allowed opportunities and percentage of data points with 100% win rate relationship

From the plot it can be seen that setting the minimum opportunities to around 40 would substantially lower the amount of data points with a 100% win rate (from ~20% to ~2,5%). However, it would also cut out very many data points in the data set.

Figure 27 shows how the number of data points falls when imposing a minimum required number of opportunities:

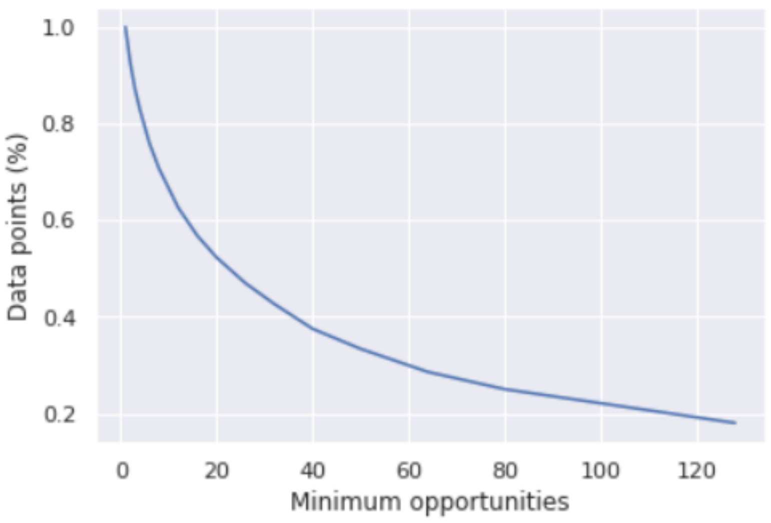


Figure 27 Percentage of data points which meet minimum opportunities restriction

It can be seen that setting a restriction to have a minimum amount of 40 opportunities would keep less than 40% of the original data points.

Therefore, the minimum amount of opportunities was chosen to be 20, which keeps ~52% of the original data points, and lowers the amount of data points with a 100% win rate to 3.6%.

It must however be conceded that this sort of filtering may also filter out people with very large deal sizes. This is because a correlation between high value deals and a low amount of opportunities was identified in section 5.3.6.

5.3.18 Data filtering using a combination of sales period and opportunities

It was visible from the data that a large amount of high sales velocity data points are simply caused by users who have only a few low value sales, which were closed quickly. These data points should be removed.

Instead of getting rid of all people with a low sales period, it was decided to use a combination of the sales period and the number of opportunities for the filtering. This is because it was identified that the users with extremely high sales velocities and low sales periods will usually also have a low amount of opportunities. Therefore, such a combined filter will be an efficient way to remove these anomalous data points.

It was decided to use a time-based metric. The combined duration of all sales should exceed 8 hours (1 workday). Such a number will make sure the non-anomalous data points don't get filtered out, because active users should easily be able to exceed one workday's worth of deals made. Additionally, it isn't so important which exact number is used as the basis for filtering, as long as it is within a reasonable timeframe. For example, if 24 hours is chosen instead of 8 hours, then in the dataset, only one extra data point was filtered as a result of the change.

The filter is calculated as follows:
$opp * period < workday$
Where:

- opp - number of deal opportunities the user has had.
- period- average sales duration in seconds.
- workday - number of seconds in a workday (8 hours)

The results of the filtering were very positive. Only ~2.7% of data points were filtered, but that included a great number of anomalous data points. The start of the sales velocity's 99th percentile in the original was ~13,5 million euros per day, while in the version with the filters applied, it's only ~80000 euros per day, a massive decrease.

Also, it can be verified that mainly the anomalous data points were filtered out, because looking at the filtered data, the median amount of opportunities is 1, the median sales period is 2 minutes, the median win rate is 100% and the median sales velocity is over 2 million euros per day.

5.3.19 Considerations for filtering out all data points with a 100% win rate

In the data set there are data points with thousands of deals, with a win rate of 100%. While it is tempting to simply filter out all those users, it is not actually known why these users have such a high win rate. Although setting a required minimum amount of opportunities removed most of these 100% win rate users, there still remained some who had thousands of opportunities and a 100% win rate.

As a follow-up to this data analysis it would be enlightening to interview some of these users and find out why their win rate is so high.

5.3.20 Data filtering conclusion

In the end, both filters were used - the low opportunities filter and the combined filter between the opportunities and the sales period.

The combined filter is good because it filters out anomalous users with a high sales velocity and a low sales period very well.

The low opportunities filter is much less precise. It filters out a large percentage of all data points and at least partially overlaps with the combined filter. However, due to the low quality of the data (low activity users with abnormal win rates), it was appropriate to use this filter as well.

5.3.21 Filtered dataset description

The filters outlined in the previous section were applied and the dataset was increased to roughly 15000 data points.

The addition of that many new data points also brought some new extreme sales velocities. The new highest sales velocity is in the magnitude of 10^{13}(euros per day). For reference, the net worth of the wealthiest man in 2020 (Jeff Bezos) is $1.49 * 10^{11}$ USD [5].

These extremely sales velocities were filtered out. The arbitrary upper limit was set to be 10 million euros per day. This filtered out 50 of the ~15000 total data points.

5.3.22 Training results with main parameters

Unfortunately, it seems there still was no learning progress throughout the epochs. The number of epochs was limited to 100 because with 1000 epochs the MAE stayed the same (Figure 28).

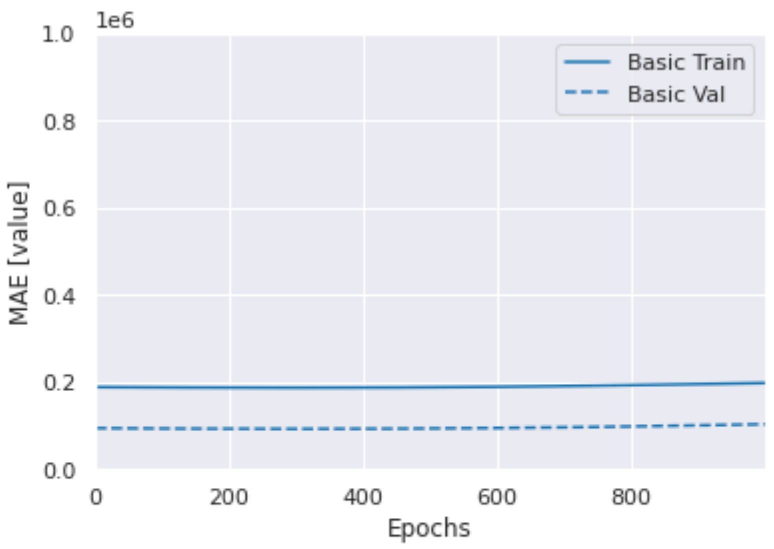


Figure 28 Training result MAE with main parameters

The testing set mean absolute error was 152425.11 euros per day.

As can be seen on Figure 29 and Figure 30, the neural network seems to stay within a certain range. The two graphs are the same, but the second graph is zoomed in 10 times.

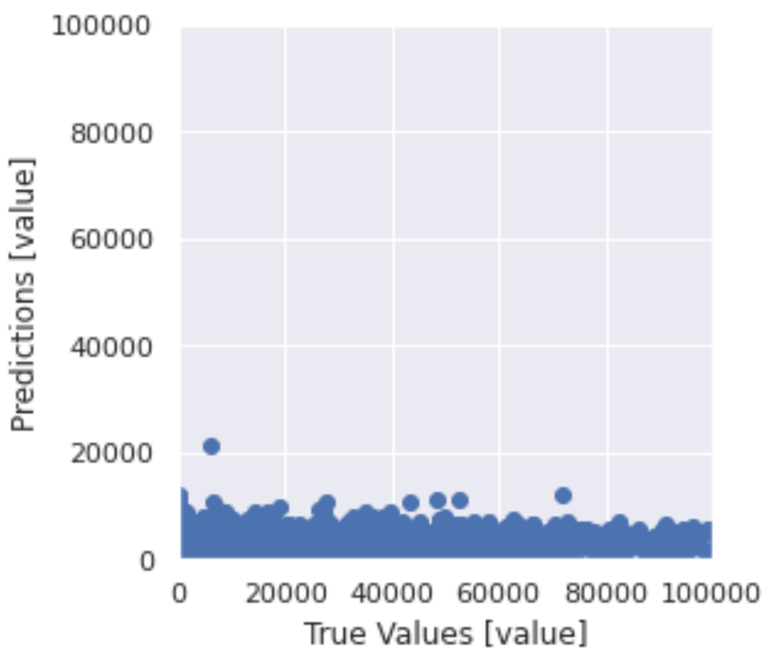


Figure 29 Training with main parameters prediction and true value comparison

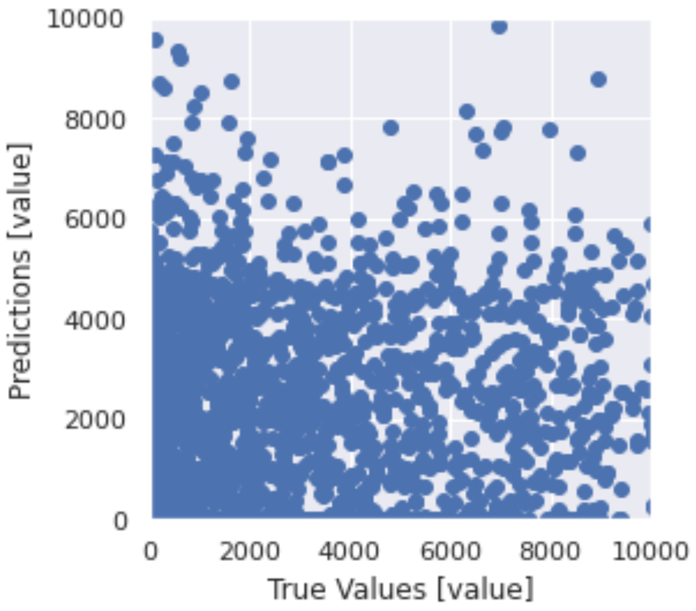


Figure 30 Training with main parameters prediction and true value zoomed in comparison

5.3.23 Training results with extra parameters

Once again, the extra parameters *industry* and *company_size* were included. Figure 31 hows degradation in the mean absolute error metric over the epochs while the loss function MSE (mean squared error) didn't improve significantly. Therefore the number of total epochs was cut down to 100 from the initial 1000.

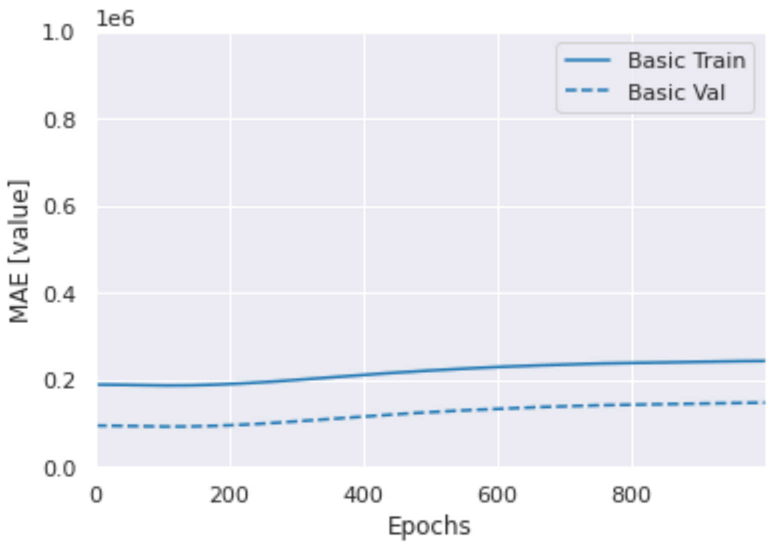


Figure 31 Training result MAE with extra parameters

The testing set mean absolute error was 213272.31 euros per day.

Once again, the results were chaotic when the extra parameters were included (Figure 32). The model has a tendency to predict high values when the true value is low (Figure 33).

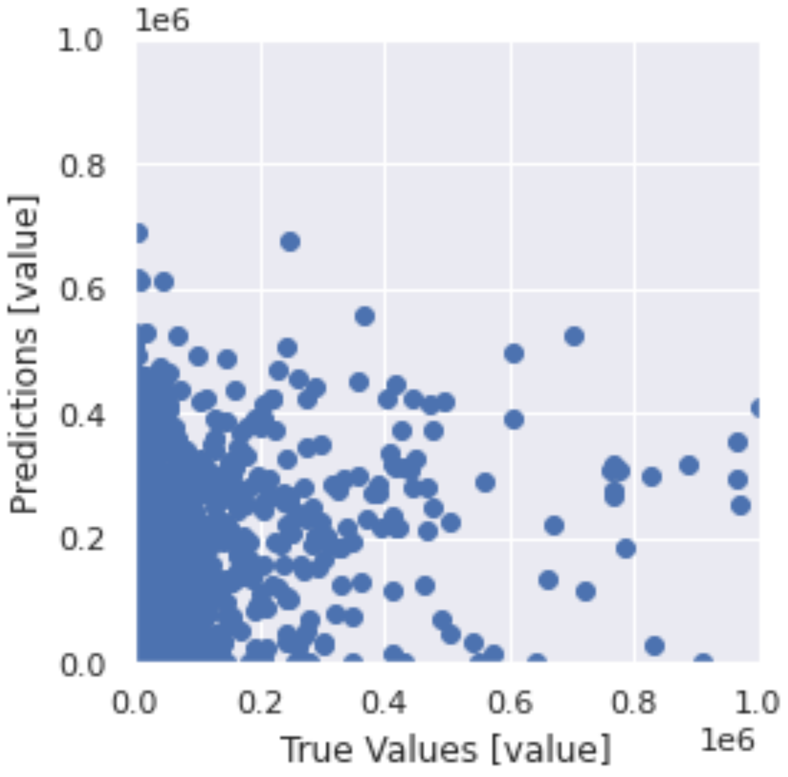


Figure 32 Training with extra parameters prediction and true value comparison

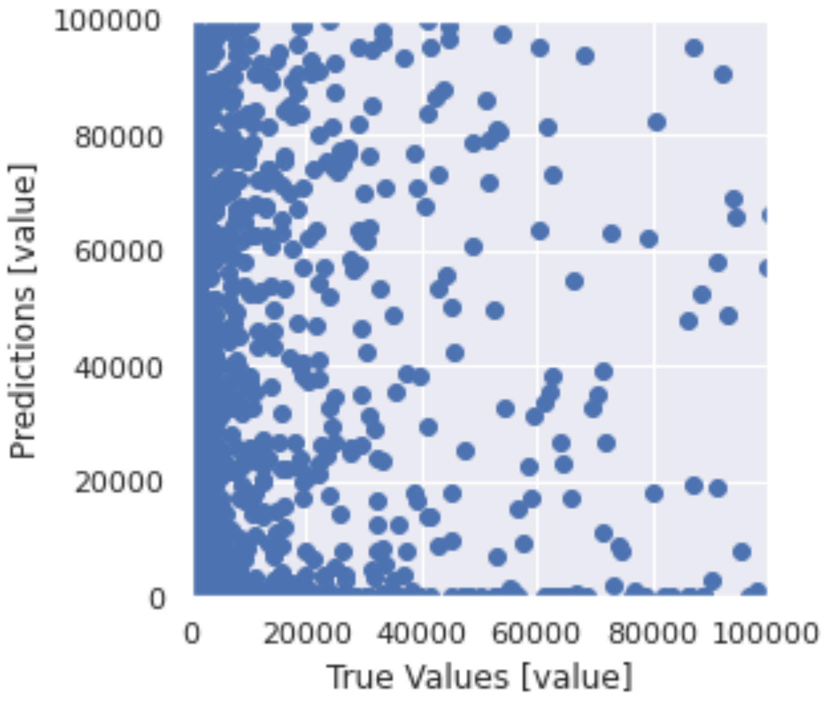


Figure 33 Training with extra parameters prediction and true value zoomed in comparison

The neural network was not able to sufficiently learn how to predict the main parameter, “value”, based on the filtered dataset.

5.3.24 Machine learning conclusion

Although the data analysis revealed some interesting statistics and correlations, the dataset turned out to be too difficult for learning. And although many problematic data points from were successfully identified and removed from the dataset, it did not significantly increase the ability to predict values.

6 Results and validation

The result is a ready and deployed web application available directly via URL or from the Pipedrive Marketplace with the links specified in Appendix 4 – Project links.

The main validation objective of the project is to verify Pipedrive's satisfaction with the results. For collecting more feedback, a post in the Pipedrive community[1] was created together with the form where users could provide feedback on how useful the application is. The created post and form did not receive much attention.

Therefore, it was decided to ask for the feedback from Pipedrive product managers, because product managers are the ones who understand the value of the application from the users' perspective. For this feedback to be more valuable and diverse product managers from different areas were asked.

The collected feedback from Pipedrive product managers is mainly positive (Appendix 3). However, even from this relatively small feedback it is noticeable that there is room for user experience improvements. For example, Vincet Jyrwa did not clearly understand why the parameters are changeable. Also, he got confused by the fact that initial user filter is not set, in reality it defaults to logged in user. Ashwin Kumar expressed the opinion that weekly history chart would be a nice improvement to the current chart functionality.

As already mentioned, overall the feedback is very positive, which shows that the results of this project are already capable of bringing value to Pipedrive's customers. Also, from the product managers' comments it can be seen that they definitely would like this application to be integrated in the main Pipedrive application.

[1] https://community.pipedrive.com/post/5eb9515c87702e3f63f010e3

7 Comments

In this section an overview is provided about the technologies that were replaced during the project development or that could be improved in the future development.

7.1 Amazon Web Services

Initially, Amazon Web Services (AWS) was used for hosting. It seemed to be an attractive option, since AWS has a free tier which was expected to entirely fulfill demands of this project. Additionally, one of project members (Heino Sass Hallik) already had previous experience with the platform.

It was decided to incorporate many of AWS's free services into the stack:

- Elastic Container Repository (ECR) for storing docker images of the backend
- Elastic Container Service (ECS)
    - For detecting and automatically deploying new ECR images in EC2 (Elastic Compute Cloud) containers
    - For automatically growing or shrinking the number of containers based on the amount of traffic
    - For automatically assigning or unassigning the Elastic Load Balancer to EC2 containers
- Amazon RDS for hosting the database
- Simple Storage Service (S3) for hosting the frontend
- Cloudfront for providing worldwide fast access to the frontend
- Cloudwatch for monitoring

However, although powerful, AWS proved to be a poor choice due to the cost of the service and the complexity of the required setup.

In theory, setting everything up is easy, especially with previous experience. In practice, however, it can be a painful and prolonged process.

All the setup in AWS is built to be "secure by default". Meaning, nothing can connect to anything by default, and it is up to the administrator to allow those connections using

Security Groups. And if something isn’t connecting, even though the Security Group configuration seems to indicate that it should, then debugging the issue can be extremely time-consuming and difficult when there is no error message other than a “502 Bad Gateway”.

The second issue with configuring AWS is that often once a service or a configuration is created, and the administrator realizes he has made a mistake, then there is often no way to change it other than to delete it and start over from the beginning.

The third and final issue with AWS is that there is no preset free tier configuration. Meaning, if the administrator wants to avoid getting billed, then he must very carefully follow the documentation to see what falls under the free tier and what doesn’t. Even after the team followed the documentation to the best of their abilities, the project team was still billed for RDS usage at the end of the month, making it not worth the hassle.

All in all, moving to TalTech’s servers and redoing everything from scratch was most likely faster than trying to resolve the AWS issues, even though the AWS configuration was already almost complete. In the future, it will be wise to avoid using AWS. Instead it’s preferable to use a simpler platform, unless there is a specific need for the scaling functionality.

7.2 SQL Database

SQL database could be changed for any type of key-value database, such as Redis, since only hashmap-like structure has to be stored on the server and long data retention is not needed. The long retention is not needed as if a user is not using the app for more than 1 hour, his access token will expire anyway. Also, it should be mentioned that any data loss at any moment of the time will only impact the user as a "logout" action and he will just need to reauthenticate. However it does not seem very beneficial from the performance perspective to change DBMS at the present moment, considering that performance won will not be noticeable and at the same time it limits possibilities of further feature development.

8 Conclusion, future development

In conclusion, the project team is quite pleased that Pipedrive is happy with the work done. Although the machine learning approach didn’t work out, the team was still able to deliver value to the customer by integrating the sales velocity history and filters into the web app.

It is very likely that sales velocity as an indicator will be soon integrated into Pipedrive’s “Insights” view. There is also hope that the machine learning data analysis is useful to Pipedrive and that Pipedrive will be able to use the analysis to attempt a different approach to machine learning solutions.

Another avenue for future research is investigating the data points which have thousands of deals and a 100% win rate to find out why those Pipedrive customers do not lose any deals.

In terms of the basic functionality of the application, the history graph can be improved. A few ideas for history improvements:

- Increase the maximum duration that can be chosen
- The time scale could be chosen. Currently the graph has only information by month, it would be great if users could see the daily or weekly change, for example.
- Ideally, different things could be stacked onto one graph. For example, logged in client could see a few users’ or pipelines’ parameters on the same graph simultaneously.

References

[1] D. Daly, „How to Grow Sales with the Sales Velocity Equation,“ Salesforce, 22 January 2014. [Web content]. Available: https://www.salesforce.com/blog/2014/01/sales-velocity-equation-gp.html. [Accessed on 23 May 2020].

[2] solid IT, „DB-Engines Ranking - popularity ranking of database management systems,“ solid IT, 2020. [Web content]. Available: https://db-engines.com/en/ranking. [Accessed on 23 May 2020].

[3] V. Zapolski, „gomon-docker,“ 9 April 2020. [Web content]. Available: https://github.com/vadzappa/gomon-docker. [Accessed on 23 May 2020].

[4] Facebook, „Getting Started,“ Facebook, 13 February 2020. [Web content]. Available: https://create-react-app.dev/docs/getting-started. [Accessed on 23 May 2020].

[5] Pipedrive, „OAuth authorization,“ Pipedrive, [Web content]. Available: https://pipedrive.readme.io/docs/marketplace-oauth-authorization. [Accessed on 24 May 2020].

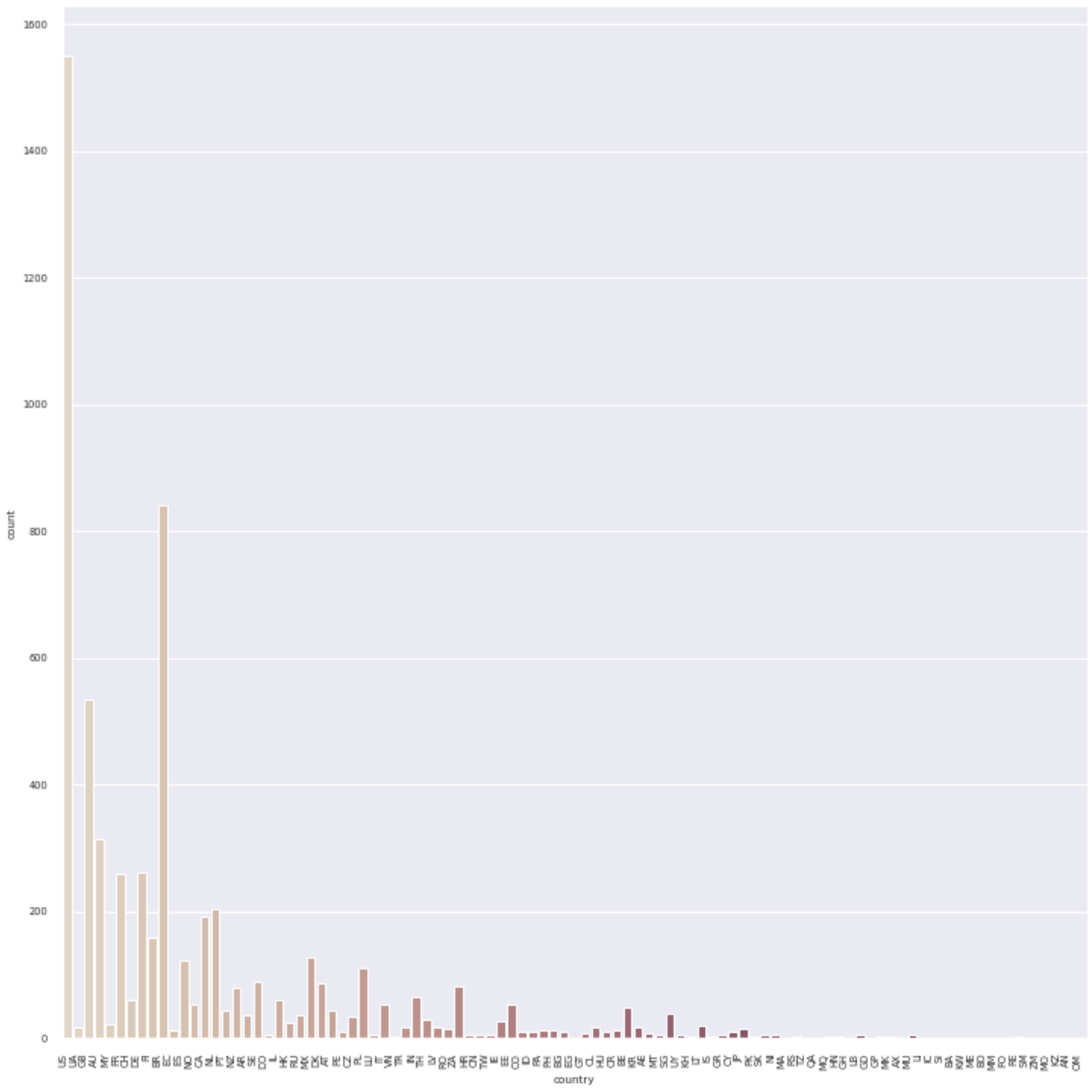
[6] Google, „Basic regression: Predict fuel efficiency,“ Google, 12 May 2020. [Web content]. Available: https://www.tensorflow.org/tutorials/keras/regression. [Accessed on 20 May 2020].

[7] Forbes, „Jeff Bezos,“ Forbes, 2020. [Web content]. Available: https://www.forbes.com/profile/jeff-bezos/#1a7910121b23. [Accessed on 20 May 2020].

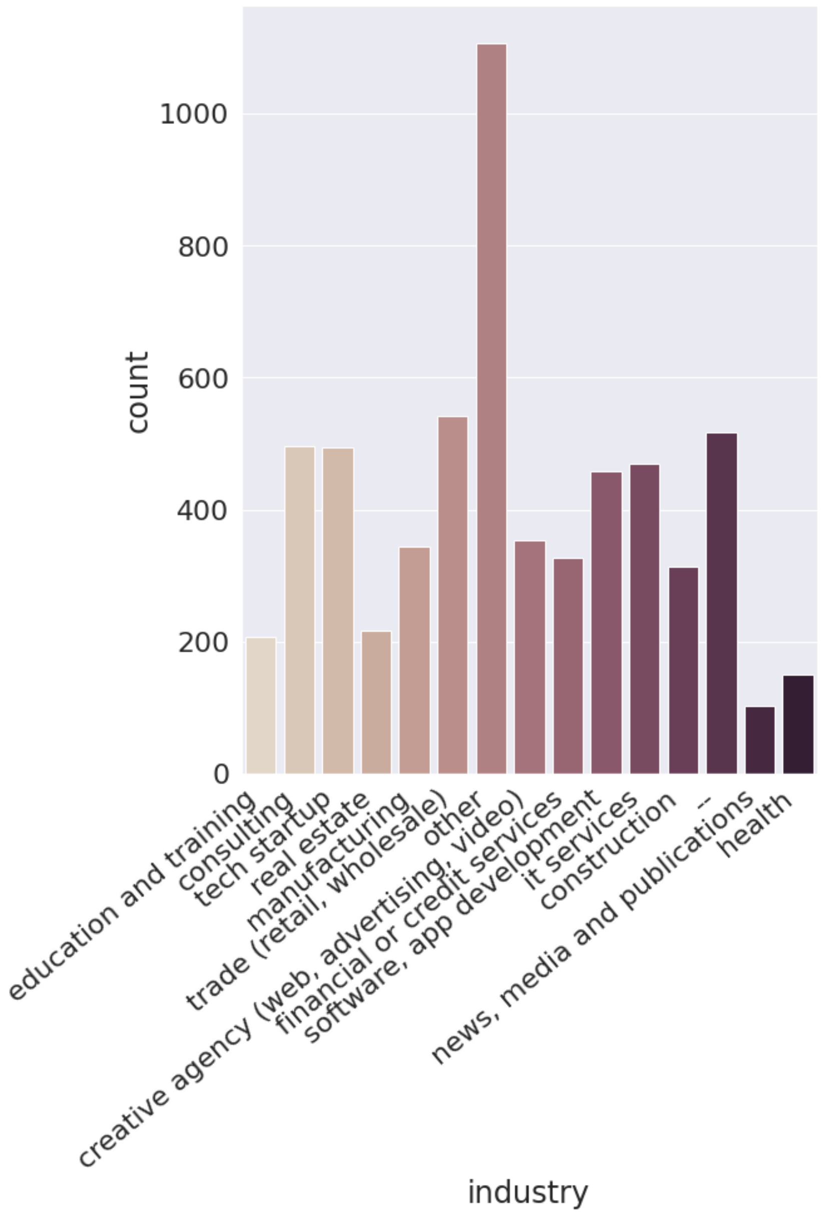
Appendix 1 – Histograms of extra parameters

Histograms of extra parameters were created to give extra context to machine learning's data analysis.

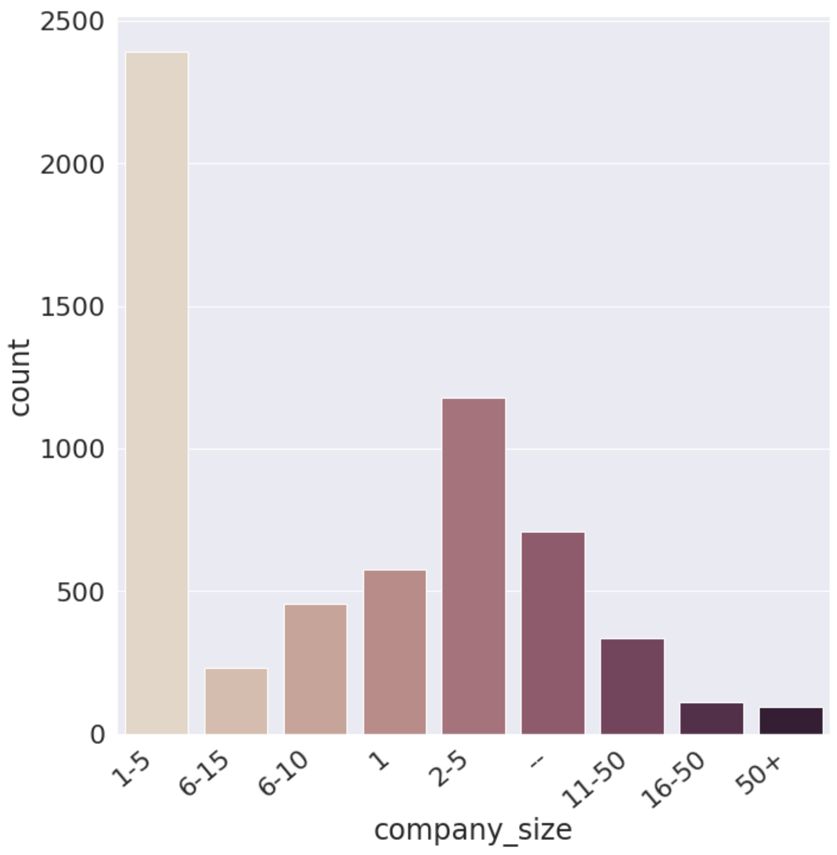
Country histogram



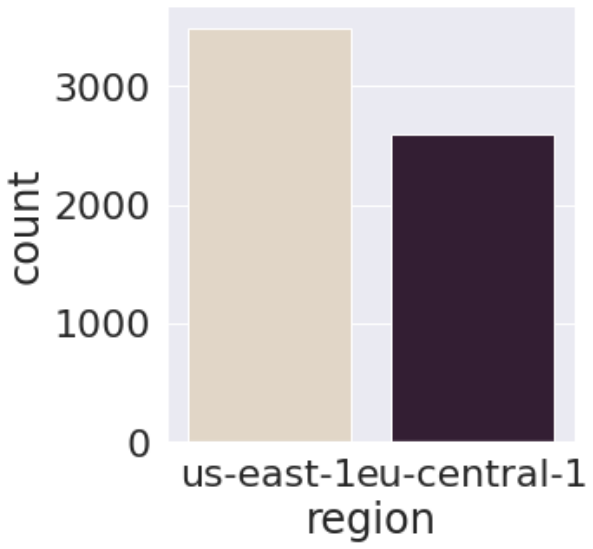
Industry histogram



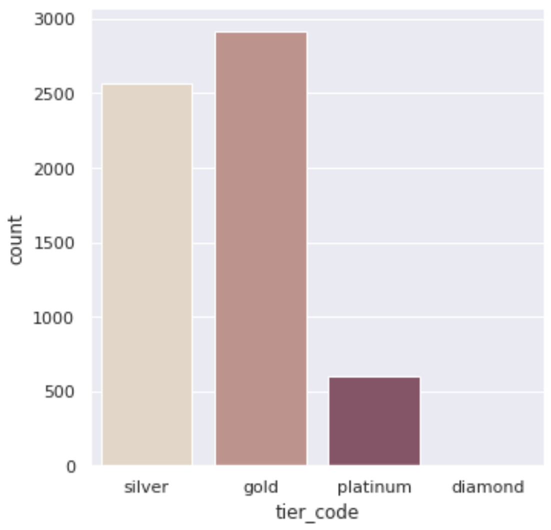
Company size histogram



Region histogram



Tier code histogram



Appendix 2 – Initial project description

TalTech Bachelor Project Description

V1 (MVP) - Calculate sales velocity

As a salesperson, I want to calculate my sales velocity, so I know my performance and can forecast the revenue more accurately.

- Sales Velocity is calculated based on 4 key variables: the number of opportunities in the pipeline (A) , average deal size (B), win rate (C) , and time it takes to win a sale (D). Formula = (A*B*C)/D. See more here[1].
- All 4 key variables are available in Pipedrive.

V2 - Change the original value of key variables

As a salesperson, I want to change the original value of key variables, so I can understand how it would affect my velocity.

- For example, my original win rate is 7%(C). As a user, I want to change it to 9%, so I can see how that would affect my sales velocity.

V3 - Suggest the most optimal change for velocity increase

As a salesperson, I want to know the most optimal change I can do in order to increase my sales velocity.

- For example, the system determines that reducing the winning time only by 3 days would increase the sales velocity by 30%.

[1] https://blog.marketo.com/2017/11/sales-velocity-critical-sales-metric-team-probably-isnt-measuring.html

Appendix 3 – Feedback from Pipedrive product managers

Seda Sahradyan, Product Manager (Progress, Insights) – “The main task was to build a web app that calculates the sales velocity based on Pipedrive users’ data. Thanks to the completed project, Pipedrive customers can already start benefiting from it. Sales Velocity is a very powerful metric in the Sales industry as it combines different important sales aspects (win rate, # of opportunities, average deal value, sales cycle duration) into one and shows the overall productivity of the team/organization. The main question it answers is - how fast is a company making money (dollars per day). Now, Pipedrive users can track that number without a need to export data to spreadsheets and calculate it manually. Moreover, users can also change the existing values to see the impact of each parameter to the whole productivity. It can help the customers to identity the bottleneck of their sales process and make an improvement for a higher efficiency. Thanks to the additional feature, it is also possible to monitor how velocity changes over time which gives insight if a company is doing worse or better compared to the previous period. Simple and clear user interface helps to navigate fast and keep the focus on the app’s main point.

Future plans - as Pipedrive is building its new customizable reporting tool which combines all the sales metrics into one, then the plan is to bring in the sales velocity there as well so that users can get a full overview of sales performance from one view. Also, the vision is to proactively tell customers how they can improve, and which are the weak aspects of their sales they should focus on.”

Ashwin Kumar, Product Manager (Engagement) – “Pipedrive is a tool for salespeople to attain sales craftsmanship. To do this it's imperative for successful sales executives to have a revenue target and systematically work towards it. Their sole focus should be on hitting their targets and not on anything else. There is the need; however, for sales executives to check if they are on track to achieve their revenue goals. Instead of de-focusing sales executives to calculate their performance, sales velocity calculator steps in. It's a quick and easy app that plugs into a user’s Pipedrive account and gives him/her a simple one page answer to the question - am I on track to achieve my sales targets? Instead of having sales teams pull out their spreadsheets, update their deal data there, and then calculate if they're on track, sales velocity calculator gives them this ability with just a few clicks. It's quick to implement, and easy to use. The only improvement needed, if at all, is that it should allow for users to have access to weekly sales velocity so that they have a better understanding of their performance.”

Vincet Jyrwa, Product Manager (Scheduler, Activities, Calendar) –

“Quick feedback:

- Maybe makes sense to add an option “all users” to the user selection field
- Also, the win rate/opportunities/etc. fields are editable, but I don’t have to edit them, right?
- I don’t see anything on the chart because unfortunately I haven’t been marking deals as won before, only tested it out just now.

Otherwise, this is a cool project! I see potential of having this inside the app somehow, especially on the Deal Forecast page”

Appendix 4 – Project links

Item	Link
Source code	https://gitlab.cs.ttu.ee/heisas/pipedrive-sales-velocity-bachelors
Deployed Application	https://www.pipedrivesalesvelocity.tk/
Pipedrive Marketplace link to Application	https://marketplace.pipedrive.com/app/sales-velocity-calculator/d0cc8349139b7ef3
Project Wiki	https://gitlab.cs.ttu.ee/heisas/pipedrive-sales-velocity-bachelors/-/wikis/home
Project Log	https://gitlab.cs.ttu.ee/heisas/pipedrive-sales-velocity-bachelors/-/wikis/Project-log
Project Issue tracker	https://gitlab.cs.ttu.ee/heisas/pipedrive-sales-velocity-bachelors/-/boards/466