Abstract

Almost every company requires hardware resources to perform necessary calculations. Nowadays, many companies suffer unnecessary losses for the reason that this hardware is used irrationally. For instance, a company might use more servers or buy more powerful equipment than it actually needs to perform the operations which the company focuses on. Our coursework project focuses on assisting the company with the optimization of purchases by obtaining the company's data using the queries to the company's time series database, reviewing the data and adjusting it for suitable visualisation and model building. After that, we make the visualisation and build the machine learning model predicting the future values for the server resources usage and clustering the servers based on the CPU, RAM and disk space usage.

Basic Terms and Definitions

- **Server** – a computer system or software application that provides functionality to other software or hardware components, often known as "clients." Hosting websites, data management, printing, emailing, and serving as a hub for the delivery of software and updates are just a few of the many functions that servers can provide.

- **Server load** – amount of work a server is performing at any one moment. The amount of CPU, memory, and disc space being utilised, together with the number of requests being handled by the server, can all be used to measure it. Because a high server load might result in sluggish performance and prolonged user response times, it is essential to monitor and manage server load to maintain ideal performance and availability.

- **Load balancing** – a technique in computer networking for distributing workloads equitably over a number of servers, network connections, CPUs, or other resources. Its goal is to avoid overloading any one resource with work, which can improve system responsiveness, performance, and dependability. Load balancing algorithms can be used to determine which resource to assign each incoming request to based on factors such as the current load on each resource, the response time, and the number of requests already being handled.

- **Figma** – a powerful cloud-based design platform. Vector networks, frames, and components are just a few of the design capabilities and tools offered by Figma.

Prototypes can also have interactivity and animations added to them. The real-time collaboration features of Figma, which enable many team members to work on a project concurrently and view each other's modifications in real-time, are among its main advantages.

- **Tableau** [1] – a modern Data Analytics and Business Intelligence platform. It allows users to create interactive and dynamic visualisations, dashboards, and reports with real-time analytics and quick responsiveness from various data sources. It provides easy access to insights and helps businesses make data-driven decisions.

- **Kibana** – a tool for data exploration and visualisation. It offers a simple user interface for developing and disseminating interactive dashboards, graphs, and charts based on Elasticsearch data. It also has machine learning capabilities, saved searches, visualisations, and other features that make it an effective tool for data analysis and discovery.

- **Grafana** – an open-source platform for data monitoring, analysis, and visualisation. It offers a unified platform for evaluating metrics, logs, and traces from diverse sources, such as databases, cloud services, and IoT (Internet of Things) gadgets. Grafana offers a variety of visualisation choices and makes it simple to create, examine, and share dynamic and configurable dashboards and alerts.

- **Prometheus** – a popular open-source time-series database and monitoring tool. It is designed to collect, store, and query metrics from various sources, such as cloud-based infrastructure, containers, and microservices. Prometheus provides a flexible and scalable solution for gathering, processing, and alerting on time-series data, making it a popular choice for monitoring dynamic and globally distributed systems. Prometheus offers a powerful query language called PromQL for working with time-series data.

Table of Contents

1. Introduction

Recently, the availability of computational resources has drastically increased. As a result, both private users and large institutions go more and more greedy for computer power. Nevertheless, as the number of servers grows, the costs of further horizontal scaling and maintenance become significant, while the reasons for such an expansion often lie in the irrational use of the existing resources. Efforts have been made by corporations to analyse the data on server usage and optimise the distribution of resources between the systems and the employees. However, in order for the rationalisation not to affect the company's performance, one should take into account the growth of server load over time. There are various approaches to the problem of predicting the need for computational resources, from basic machine learning approaches (linear predicting models) to more advanced techniques involving RL. For complex analysis, it is crucial to collect statistics on each important resource in the machine: CPU, RAM, and disk storage.

1.1. Goal of the Project

This project aims to evaluate the usage of physical servers in the customer's (JSC "NSPK") network, visualise the data and develop machine learning models to identify common patterns of server usage and predict the company's need for computational resources in the future.

1.2. Tasks

- Study and compare the existing metrics evaluating the performance of a server.
- Study the analytic tools available in the company (Grafana, Prometheus, Tableau, etc.). Determine their relevance in the context of the server load analysis.
- Collect the data from the company's servers.
- Prepare the data for analysis: normalise it and impute the missing values.
- Develop a ML model to infer the future demand for computational resources from the current usage of physical servers.
- Develop a ML model to cluster the servers and find common patterns of their usage.
- Visualise and interpret the collected data and the analytic results.

1.3. Roles of the Team Members

- **Timofey Sluev:** research on the existing approaches to the optimisation of server usage, selection of relevant data categories, collecting metrics from the servers.
- **Konstantin Shashkov:** development of the system architecture, data preprocessing: normalisation and imputation.
- **Gregory Chebotarev:** ML-based analysis: clustering of servers according to usage patterns and prediction of future demand for computational power.
- **Daria Lapko:** design, prototyping and implementation of a solution to visualise server metrics and analytic results.

2. Description of Functional and Non-Functional Requirements

2.1. Functional Requirements

1. The application is expected to collect metrics on the usage of physical resources (e.g., CPU, RAM) from the company's servers or a dedicated database.
2. The application is expected to aggregate the data from different teams across the company and its subdivisions to present an overall picture of server usage.
3. The application is expected to visualise the historical data of server usage and aggregate metrics.
4. The application is expected to cluster the company's servers according to determined usage patterns.
5. The application is expected to forecast the future usage of servers.
6. The application is expected to visualise the results of ML analysis (clustering and prediction).

2.2. Non-Functional Requirements

1. The visualisations provided by the application are expected to be interactive.

2. The application is expected to normalise and impute the collected data before analysing it.

3. The application is expected to satisfy the corporate standard of availability for business-critical systems.

4. The application is expected to be horizontally scalable.

5. The application is expected to be universally employable given similar infrastructure (availability of similar data collection, storage, processing and visualisation tools).

3. Solution Architecture

In order for the service to function well, all its parts should be properly integrated and tested. Building an efficient architecture is as important as picking the data correctly and tuning the model. Failing to do so may result in critical issues when the services are integrated, from reduced performance to data loss or corruption, which are likely to affect the results of the analysis to a great extent.

As the application operates, the data passes through 4 main stages of processing:

1. **Raw data collection:** a dedicated tool takes input metrics from the servers as they operate and store it for future use. In the case of our application, Prometheus was chosen by the company as such a monitoring system. From there, the data is retrieved with a dedicated query language (PromQL).

2. **Data preprocessing** is the stage which ensures the quality of the obtained data. This can be done either at the beginning, or the model's script, or in a separate utility. Since some preprocessed data may be used for visualisation and cached to optimise the analysis in the future, the option of a separate script is preferable.

3. **Modelling and prediction** is when the ML comes into play. The preprocessed data is read from the storage and used to train, test and further use the model. Because the data structure is consistent across all three stages, so is the retrieval process. When the model finishes working, the original data enriched with its output is stored for later use.

4. **Visualisation** may require the outputs of both preprocessing and analysis. It reads the necessary data and presents it graphically. This is the ultimate output gate of our system.

Diagram 1 presents the described architecture schematically.

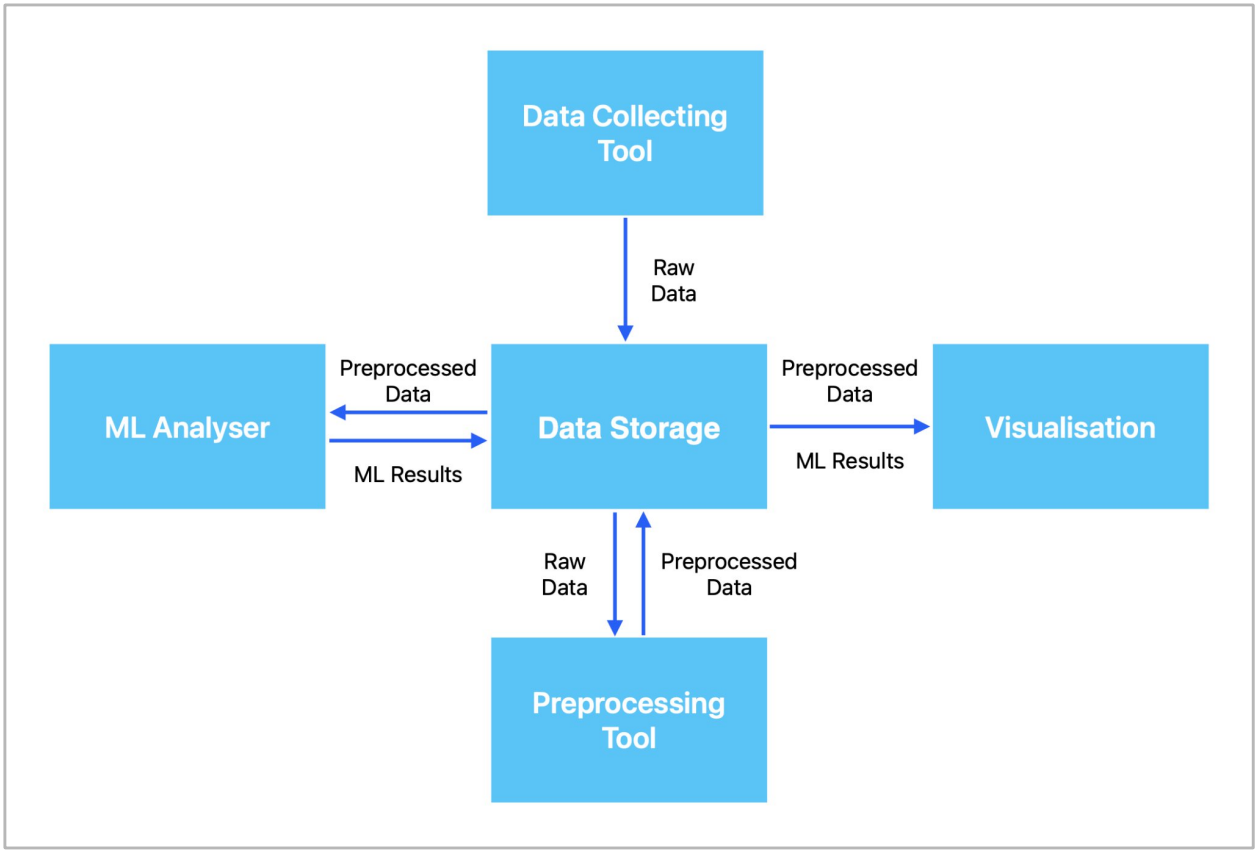

Diagram 1. Proposed Architecture of the Application.

4. Data Collection and Processing

4.1. Discussing Prometheus Framework

The data was extracted using Prometheus time series database and PromQL query language for writing the queries to extract the data. As a part of the project, we conducted a comparison of different time-series databases and studied the concept of a database itself and conducted a comparison of different time-series database models (InfluxDB, Prometheus, etc.) and are able to make several arguments why Prometheus TSDB is the most promising choice of a database in our situation. The concept of a time-series database lies in the fact that a lot of data is stored using key-value pairs, where the key is the timestamp of the observed value. The convenience of this approach to storing the data is that the timestamps are unique. That is why a lot of NoSQL approaches to storing the time-series databases are now widely discussed. The main benefit of PromQL (in comparison, for example, with InfluxDB) lies behind the fact that Prometheus has its own Python client library, which allows us to write the queries conveniently

in a small Python script. It also helps us to quickly collect the raw data from various sources (disk, cpu, memory) to a single CSV or JSON file. This makes the further work with this data (analysis, visualisation and user interface creation, statistical model building) much easier and helps to save time by avoiding the use of external instruments.

4.2. Discussing Existing Visualisation Approaches

Another important framework we are going to talk about is Grafana. In industrial software engineering, time series visualisation is often conducted through transmitting the data to Kibana or Grafana, where the data visualisation is done on the latter side. With a little more details on these two frameworks in the corresponding part of the report, Grafana is a fork of Kibana which extends the possibilities of the data illustrations. One example of this extension lies beyond the fact that Kibana only accepts the data from the Elasticsearch (in its own data format), which would require unnecessary conversion of data. On the other hand, Grafana can read the data directly from the Prometheus database (as a Prometheus query accepted by Grafana). For these reasons we used Grafana, Prometheus queries and Python scripts to review, gather, and extract the data from the company servers.

Let us now focus on the data we are extracting and review the queries which we use to gather the data and metrics which we obtain:

4.3. Discussing Metrics and Queries

1. The first data source we are going to use is the memory usage. By memory here, we refer to random access memory, or RAM. The computer uses it to immediately access and retrieve the information for the computations it performs. It is physically stored in microchips, which require expensive materials to produce. RAM is hence one of the most expensive parts of the server, and optimising the memory usage shall rationalise the resource allocation of the company, therefore cutting costs, resulting in better economic efficiency. To monitor the RAM usage and further build a machine learning model we decided to use the ratio of the available memory to the total memory capacity of the corresponding server. To extract the data, we made a dataframe with the corresponding server, team and the server memory usage for the past 30 days (including today). The values are initially (after receiving the PromQL response) in the .json format, which we convert to csv by using pd.json_normalize(). The memory values in

each day are the ratio of daily averages, which we offset for the past values by the corresponding number of days (up to 30). The script used for obtaining the values for the last day can be observed on Fig. 1. The script for offset values can be observed on Fig. 2

2. Another important data that we are going to consider is the CPU usage. The CPU includes all the circuits which execute the commands we give to the applications, so all the calculations are done in the CPU. In our project, we made the queries which get the average time that the CPU spent **not** in the idle mode for the last 1 day (in percentage points), then offset this value for every day in the last 30 days, and obtained the data frame for the CPU workload.

3. Last but not least, we have to work on the optimization of disk usage. Here, we refer to disk space as to available permanent computer storage. Unlike RAM, it requires more time to access the information, but has a far larger capacity and does not erase the data when the server is turned off. All the operating system and other applications files are stored on disk. In our script, for the storage data the query calculates and outputs the ratio of the number of bytes available (free) in the storage to the total possible storage capacity for this hardware (also in bytes). We paid close attention to the filesystem type that the server has and specified the mounting point of the disk. In the same manner, we specify the Prometheus target URL, generate the request, receive the response and convert the data to the .csv format. Note that after receiving the final data, we group it by servers and add the obtained value to the corresponding server.

4.4. Discussing Data Problems

After collecting the data, we noticed that certain servers contain null values in certain fields. We also noticed while doing the intermediate data analysis for the past 7 days that there are no null fields. This occurs due to the fact that some teams did not set up the Prometheus database to store the data for the past 30 days, but the 7 day storage was set up for all teams.

```
# Load data on memory usage
qry = '100 * (1 - (avg_over_time(node_memory_MemAvailable_bytes{job="' + job + '"}\
    [1d]) / avg_over_time(node_memory_MemTotal_bytes{job="' + job + '"}[1d])))'
url = prometheus_url + qry
response = requests.get(url, verify=False)
data = response.json()
data = data['data']['result']
df = pd.json_normalize(data)
df['mem_today'] = df['value'].apply(lambda x: x[1])
if 'value' in df.columns:
    df = df.drop(['value'], axis=1)
    df = df.sort_values(by=['metric.instance'])
else:
    df['mem_today'] = pd.np.NaN
```

Figure 1. The memory usage for today.

```
for i in range(1, n_days):
    qry = '100 * (1 - (avg_over_time(node_memory_MemAvailable_bytes{job="' + job + '"}[1d] offset ' + str(i) + 'd)\
        / avg_over_time(node_memory_MemTotal_bytes{job="' + job + '"}[1d]  offset ' + str(i) + 'd)))'
    url = prometheus_url + qry
    response = requests.get(url, verify=False)
    data = response.json()
    data = data['data']['result']
    df_ith = pd.json_normalize(data)
    if 'value' in df_ith.columns:
        df_ith = df_ith.sort_values(by=['metric.instance'])
        df['mem_' + str(i) +'d_ago'] = df_ith['value'].apply(lambda x: x[1])
    else:
        df['mem_' + str(i) +'d_ago'] = pd.np.NaN
```

Figure 2. The memory usage for the past N days.

4.5. Discussing Results

Overall, we see that the available memory differs among teams and servers significantly, with some teams often having less than 10% available memory on average, while others have a lot more than 50% RAM usage. It is a clear sign of misuse of the available memory, which might lead to higher costs for the company and irrational budget balance. One of the ways to address this issue is looking at the number and power of the servers and redistributing them in a way which makes the resource utilisation closer to the medium value (around some fixed percentage average RAM workload). As for the CPU usage, the values are far less critical and usually tend to be around 0 - 10%. However, it doesn't mean that we can let up on optimising it. This should also be addressed, as usually CPU is the most expensive part of the server, hence a large amount of resources can be saved by optimising the purchases of CPU. Disk usage also varies between teams significantly, and this time, unlike in the case of RAM and CPU usage, the average is more likely to be above 50%, sometimes exceeding 90% space used, which suggests that here we are unlikely to really cut on the disk hardware purchases. However, the storage might still be optimised in order to allocate additional disk space and/or notify the management prior to

the moment when new disk purchases shall be required (which we shall discuss later in our machine learning modelling section).

5. Normalisation and Reconstruction of Time-Series Data

To ensure the accuracy and interpretability of analysis, complete and quality data should be considered. Unfortunately, in real life, collecting such a dataset is barely possible. Although Prometheus is a highly-automate tool collecting metrics from computers, it is configured and maintained by people. Therefore, one should bear in mind the possible consequences of human factor on the quality of the data. In the company, there are many teams of developers, each administrating their own servers. The settings for Prometheus are chosen and implemented by the product team and the responsible DevOps engineer(s). As a result, not only is the resultant data likely to be nonuniform, it also may happen to be inadequate at times. In this work, we decided not to dive deep into interaction between Prometheus and the physical servers, mostly relying on the data collected by the product teams and the CI/CD department. To prepare it for future analysis, I addressed the problems of normalisation and reconstruction that I am going to discuss further.

5.1. Data Normalisation

As I mentioned above, Prometheus is set up by people, and people tend to make mistakes. Before proceeding to inference, it is crucial to verify the adequacy of collected data. Since all three metrics considered in this work are measured in percentages, their values should fall between 0 and 100. One way to deal with the data points lying outside of these bounds is to treat them as nulls and apply one of the reconstruction techniques suggested below. Another approach is to set them equal to the closest boundary values (either 0 or 100). Having obtained the data, we observed high-magnitude negative as outliers. Since there was no reasonable interpretation for such values, and assuming the corresponding cells to contain zeros was counter-realistic given the data around, I decided to treat them as missing values.

5.2. Reconstruction of Time-Series Data

A more complicated problem to solve was the reconstruction of missing values. The issue of incomplete data arise since the tracking period for Prometheus metrics varies from team to

team. As a result, large blocks of data happened to be empty. One could have taken a straightforward approach and built the analysis on the longest period, in which complete data is available. Unfortunately, the shortest record we encountered was kept for one week only, which is not enough to make any insightful conclusions on the dynamics of server usage. Moreover, the project is intended to be scalable and suitable for regular analysis rather than a one-time glance at the data. Clearly, this requires the application to be stable in case of any changes in record timing. Hence, we decided to choose the period of analysis as an optimal trade-off between insightfulness and data availability. After a series of discussions, we agreed to use the observations within 30 calendar days back from the sampling date. This brought me to the task of reconstructing the missing values in the obtained dataset, were a few teams miss several weeks of record. Conventional methods like replacing NaNs with mean, mode or median seem ed insufficient for filling entire sections of a dataframe. Aiming to improve the quality of future analysis, I decided to address this issue with machine learning.

5.2.1. Existing Solutions

It is worth mentioning that there have already been attempts to apply methods of machine learning to multivariate data imputation. The *sklearn.impute* module suggests [2] two ready-to-use ML-based algorithms capable of reconstructing several dimensions of missing values in a dataframe.

The first option is the *IterativeImputer* class inspired by the *Multivariate Imputation by Chained Equations* algorithm, which was proposed in 2011 by Stef van Buuren and Karin Groothuis-Oudshoorn [3]. In simple terms, it iteratively regresses every feature on the others using the known data in order to later predict the missing values. Further, it may reiterate through the data frame for a specified number of times to re-impute it employing the results of the previous step. That way, it is expected to improve the accuracy of its estimation. The class allows one to specify the desired estimator for the regression [4]. By default, it uses the *BayesianRidge* model [4,5].

An alternative approach is proposed in the *KNNImputer* class, which is based on the k-Nearest Neighbours algorithm [6]. The missing value is filled with an average or a distance-weighted average of *k* nearest neighbours, where *k* is a model parameter. By default, the module uses *nan_euclidean_distances* [7] as the metric for finding neighbours. One may design their own metric if they wish to.

A clear drawback of both these solutions in the context of our project is the fact that they use the entire dataframe for imputation and do not take into account the predefined restrictions of the data. Firstly, the data frame consists of 3 different sections each representing a dedicated metric. Although there might be some correlation between the load of CPU, disk, and memory, it would be naive to assume a relationship between them. Given the various factors affecting these metrics (e.g. available physical resources, specific use of the server), one could easily find a counterexample for such a conjecture. Secondly, neither algorithm is intended for time-series data. For instance, it is natural to consider possible autocorrelation and seasonality. Generally, close-in-time observations of a similar metric on the same server appear to be the most suitable regressors for a missing value. This conclusion brought me to the idea of constructing my own data-specific autoregressive model.

5.2.2. Proposed Solution

Following the discussion above, I came up with a simple yet efficient solution for reconstructing time-series server usage data. The proposed algorithm is a modification of the classic AR model defined as follows [8]:

$$X_t = \alpha_0 + \sum_{i=1}^{n} a_i X_{t-i} + \epsilon_t \quad (1)$$

In (1), X_t stands for the value of the variable X at time t, n is the order of the model (i.e. number of past observations used for predictions), $\alpha_0 ... \alpha_n$ are the coefficients assigned to the corresponding regressors, and ϵ_t is a stationary white noise disturbance term at time t. This model is typically used to regress the future values of a time series on past observations.

The problem of data reconstruction requires a slightly different perspective on time-series inference. Instead of predicting the future values, I needed to reconstruct missing records in the past. To do so, I reformatted equation (1) to estimate older values based on more recent observations and obtained the following:

$$X_t = \alpha_0 + \sum_{i=1}^{n} a_i X_{t+i} + \epsilon_t \quad (2)$$

Next I had to choose the parameters of the model. Since only monthly data was considered, which is a pretty short term, I assumed the metric to be stationary over time. The only seasonal patterns that one could have observed on such a time horizon are weekly trends. Having that in mind, I decided to set $n = 7$. An AR(7) model regressed on a week of observations allows one

to capture the similarity of data points both on the neighbouring dates and the dates corresponding to the same weekdays. Moreover, 7 is the maximum number of regressors, for which I had a complete future record available at any data point. For simplicity of interpretation, I assumed the constant term $\alpha_0 = 0$, although the impact of the constant term on the performance of the model may be an important topic for future research. Finally, I defined X_t as the observation of a particular metric (CPU, RAM, or disk usage) on a particular server on day t. To estimate the coefficients $\alpha_1 ... \alpha_7$, I chose the classic OLS estimator [9]. Further, it may be interesting to experiment with different regression estimators and possibly improve the model's accuracy.

5.2.3. Implementation

To train the model, I selected a subset of the data, which contained the servers with a complete month-long record only. Having done a 1:3 train-test split, I constructed a separate training dataframe for every metric (3 in total). For every metric observation $X_{-7} ... X_{-29}$ of a server X, I constructed the following table:

Table 1. Structure of the training data.

7	6	5	4	3	2	1	Target
X[0]	X[-1]	X[-2]	X[-3]	X[-4]	X[-5]	X[-6]	X[-7]
X[-1]	X[-2]	X[-3]	X[-4]	X[-5]	X[-6]	X[-7]	X[-8]
…	…	…	…	…	…	…	…
X[-22]	X[-23]	X[-24]	X[-25]	X[-26]	X[-27]	X[-28]	X[-29]

Given $\alpha_0 = 0$ and $n = 7$, I obtained the following model to train:

$$X_t = \sum_{i=1}^{7} a_i X_{t+i} + \epsilon_t \qquad (3)$$

For every metric (CPU, RAM, and disk), I fit a dedicated model on the data described in *Table 1*. To estimate the regression coefficients, I used the *OLS* estimator from the *statsmodels* package [10].

To reconstruct the lost data, I iteratively estimated the missing values from the most recent to the oldest, this way always having the predictors complete for 7 days in advance. Before imputation, the given dataset is split in 3 to estimate every metric with a dedicated model.

5.2.4. Testing

To assess the performance of my model, I synthetically removed part of the data from the test set, imitating the structure of losses in the original dataset.

I decided to compare the performance of my model to the ones of the library models discussed above. To do so, I applied *IterativeImputer* and *KNNImputer* with default parameters to the complete train-and-test dataset with parts of the test data being synthetically removed as described in the previous paragraph. At this point, one should note that both of these solutions are intended to process entire dataframes without a train-test split.

To compare the performance of the models, I calculated the *sklearn Root Mean Squared Error (RMSE)*, *Mean Absolute Error (MAE)*, and *Mean Absolute Percentage Error (MAPE)* [11,12,13] for their predictions on the train set.

5.2.5. Results

Table 2 presents how the proposed model performed on the test dataset compared to the library *IterativeImputer* and *KNNImputer* with default parameters.

Table 2. Comparative performance of the proposed model, IterativeImputer, and KNNImputer.

	Proposed AR model	**IterativeImputer**	**KNNImputer**
RMSE	6,60117	9,99687	5,71086
MAE	3,60792	6,85515	2,70549
MAPE	0,46682	1,37328	0,48849

The model has performed relatively well. It has shown a notable advantage over *IterativeImputer* and outperformed *KNNImputer* in *MAPE* being slightly behind it against the other metrics. This way, it was proven efficient for reconstructing the missing values of the project's dataset.

5.2.6. Conclusion and Prospects for Future Work

In this research, I have developed a project-specific imputer of missing values based on an autoregressive linear model (AR(7)). On a synthetically degraded testing dataset, the solution

has outperformed the regression-based *IterativeImputer* from *sklearn.impute* showing the results comparable to the ones of the *sklearn.impute.KNNImputer*. Eventually, it was applied to the original incomplete dataset to reconstruct its missing values and improve the quality of the analysis discussed in the next sections of this report.

6. Machine Learning Models

My task in this project was to implement machine learning algorithms in order to get predictions about future usage of the server resources and to clusterize the data in order to find patterns in users' (teams' particularly) behaviour regarding the available resources.

The primary aim of my project is to build several different prediction and clusterization models (regression models, random forest, neural network, K-Means,...) that will allow us to make future predictions and compare their performance using predefined metrics. I decided to go for MSE (mean squared error), MAE (mean absolute error) and R-squared, as these metrics are easy to interpret while being decent estimators of the model performance.

Data preprocessing, feature selection, model training and performance evaluation, are some of the major steps in the project. By fulfilling this steps one after another, I hope to create a solid and trustworthy load prediction and clusterization models that could help the company to tackle the problem of irrational resources distribution.

6.1. Data Preprocessing, EDA

Firstly, I started with exploratory data analysis (EDA) and data preprocessing, which really stands for transforming/cleaning raw data. Since all the NA values were imputed by my teammate, I noticed that the data is not normalised as the average value for memory usage is about 28-30 %, while the average cpu usage is less than 4%. So, to normalise the data it is natural to use StandardScaler which standardises features by subtracting the mean and scaling it to unit variance. Data normalisation is crucial in machine learning as it results in better performance, more stable model and StandardScaler is optimal here because there is a small range of values for each feature.

To gain the first insight on the data I built histograms with the distribution of the main features (memory, disk and CPU usage), which are presented below:

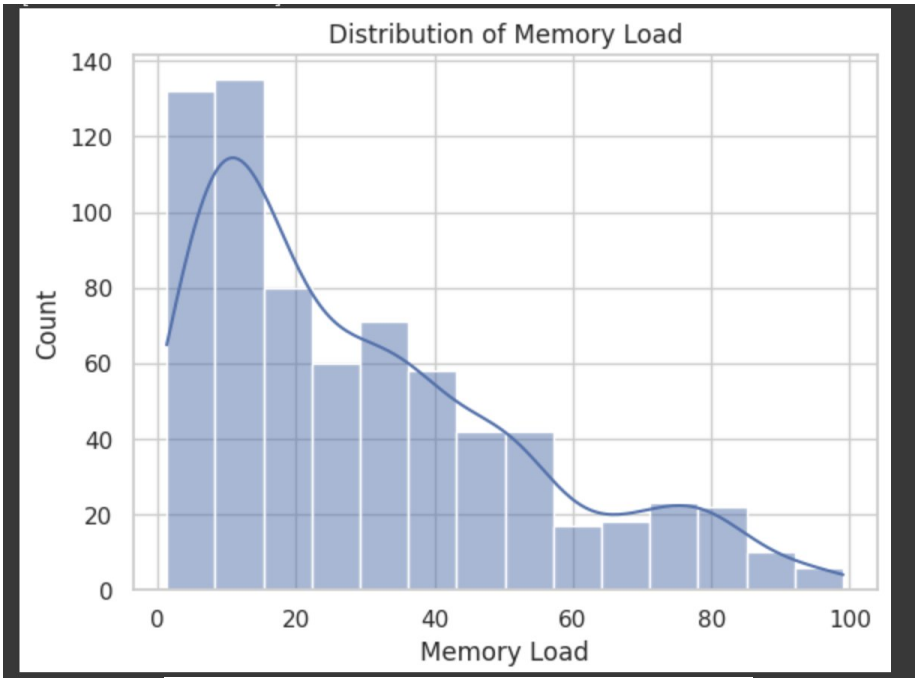

Figure 3. The distribution (histogram) of memory load.

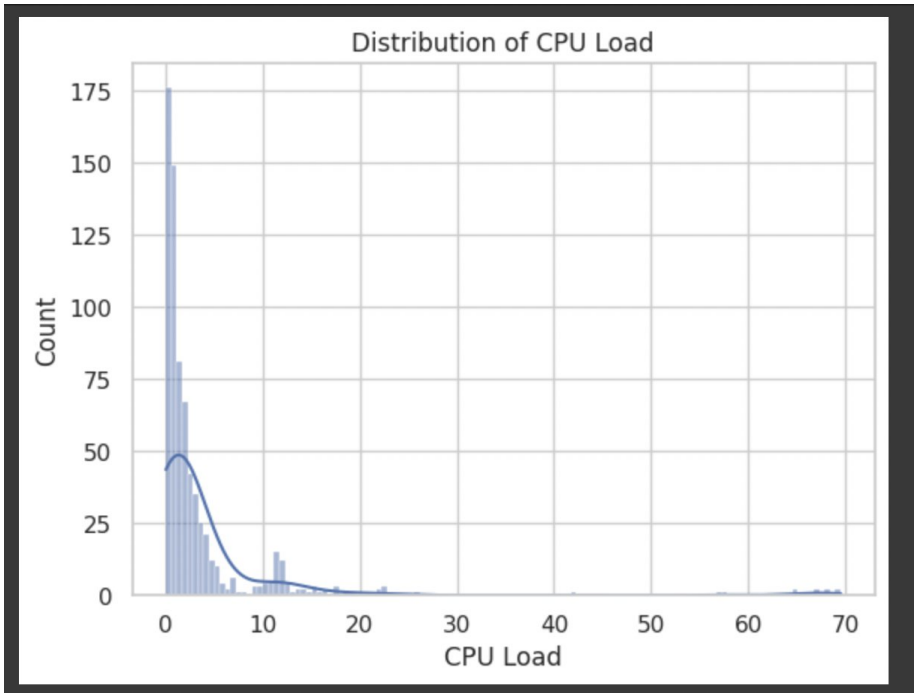

Figure 4. The distribution (histogram) of CPU load.

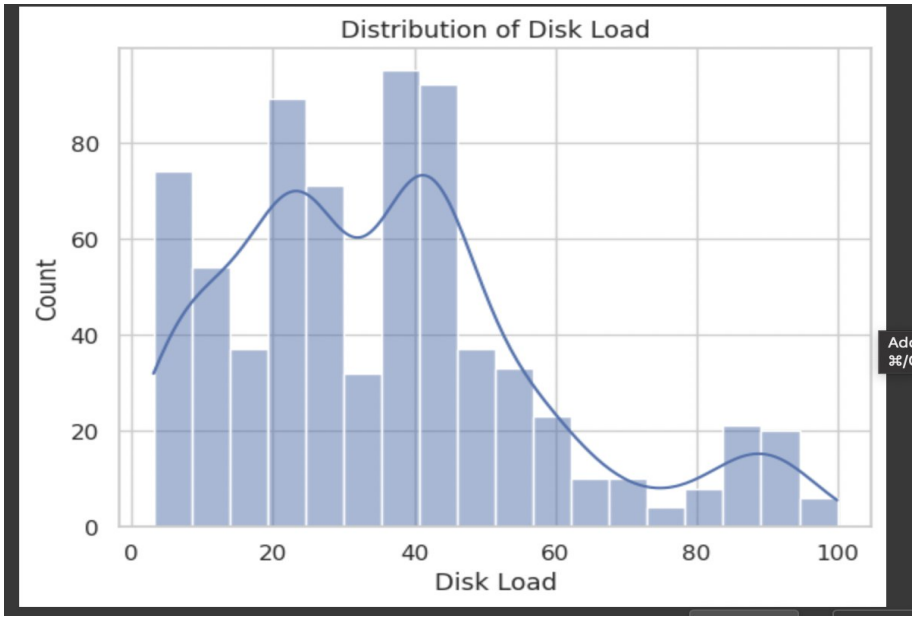

Figure 5. The distribution (histogram) of disk load.

Obviously, we see that while the memory and disk distribution is pretty diversified, the CPU distribution's 3rd quantile is approximately 5%. This gives us an idea that CPU is probably one of the most irrationally used resources. However, 3rd quantile for memory and disk distribution are roughly 43% and 45% respectively. This numbers are not that high, so our assumption about not optimal allocation of resources still persists.

Speaking about feature engineering, I tried to generate new features using tsfresh library , but unfortunately it did not lead to any increase in metric proposed for evaluation of the prediction models, so I decided to omit feature generation here.

Finally, as the dataset containes a great amount of features and new generated features did not help, using dimensionality reduction techniques can help to increase the interpretability of the data while keeping the maximum amount of information. A solution that I proposed was PCA (principal component analysis). This algorithm is quite simple, the intuition behind it is to fit the k-dimensional ellipsoid to the dataset where each axis is so called principal component (vector). To define the optimal number of components, I will use the cumulative total variance graph setting the benchmark to 0.9 of the total variance. So when the proportion of cumulative variance reaches 0.9, we take the number of components that corresponds to this point.

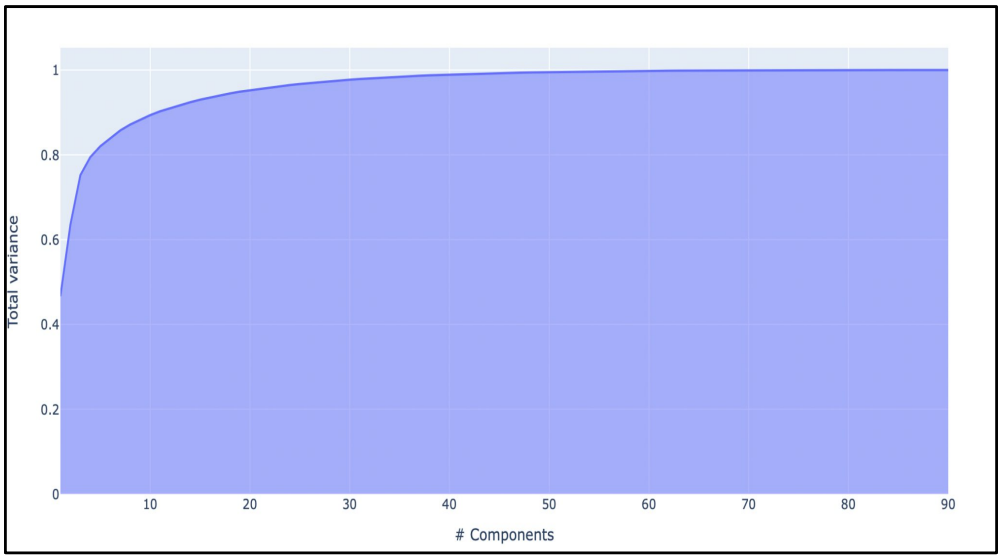

Figure 6. The cumulative total variance graph.

After building the cumulative variance plot, it is clear that 10 components explain 90% of the variance, so I will go for this number of components.

6.2. Data Clusterization

In this section of the project, I intend to create a number of clusterization models that will reveal the teams that frequently use less capacity than is actually available. At first, I chose K-Means since it's reasonably easy to implement and typically results in tighter clusters than other methods. The main idea of this technique is to recalculate cluster centroids each iteration and clusterize the points using Euclidean distance. But what value of k is usually taken? One of the most famous approaches to find the optimal value of k is the elbow method that consists of building a graph of the explained variance by y-axis and the number of clusters by x-axis. Then one should choose the point (# of clusters) where the ‘elbow’ curve becomes much smoother. In our case this number of clusters is 4. Here we can see the results of the K-Means algorithm with k = 4:

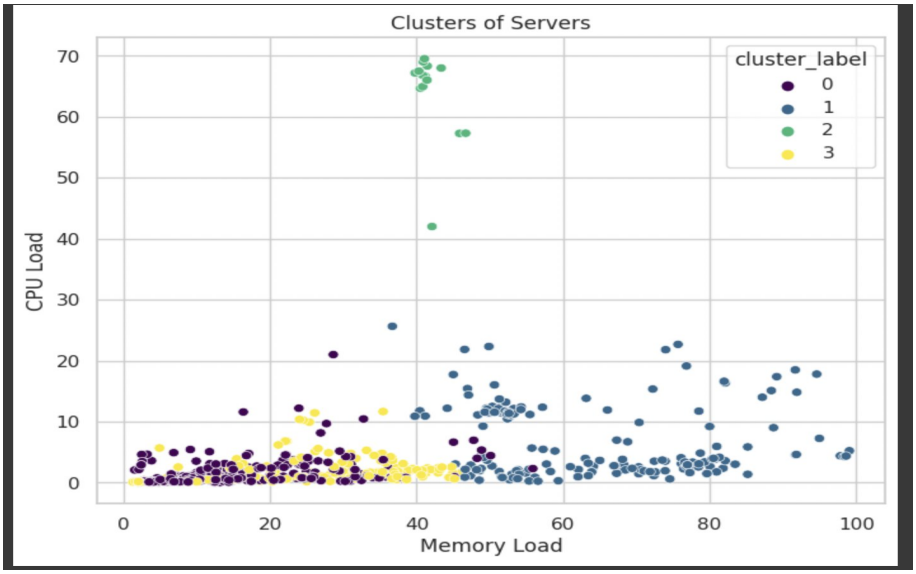

Figure 7. Results of K-Means algorithm with k = 4.

The cluster means are presented below:

cluster_label	mem_today	cpu_today	disk_today
0	14.653914	1.410722	18.734035
1	64.738746	6.671897	47.400374
2	41.879588	63.968882	84.471171
3	24.515129	1.824776	51.317636

Figure 8. The cluster means obtained from K-Means algorithm with k = 4.

Clearly, we see that K-Means reveals (fig. 8) the first (0) and the fourth (3) clusters as the ones with the lowest average resource utilization, while the second and third especially looks more promising. However, it still debatable as even the third cluster with the highest average utilization in CPU and disk still suffers from not sufficient memory utilization. So this information will help the company to identify the teams (from first and fourth clusters especially) that are likely to underuse the available capacities.

The second algorithm that was used to clusterize the data was Agglomerative Clustering which is a well-known type of hierarchical clustering where the bottom-up approach is used, i.e the similar clusters are joined greedily. To define the optimal number of clusters in this case one may use silhouette method. This method shows how each point is similar to other points in its own cluster compared to other clusters. It takes values from -1 to +1, where a higher value depicts that a point is well classified. Silhouette method revealed the k = 3 is the most optimal

number here. The results are presented below(to simplify only one graph with CPU and memory was used):

Figure 9. Results of Agglomerative Clustering algorithm with k = 3.

cluster_label	mem_today	cpu_today	disk_today
0	21.381094	1.647862	29.077782
1	61.908176	7.618253	58.760299
2	41.862487	65.657653	87.491465

Figure 10. The cluster means obtained from Agglomerative Clustering with k = 3.

Well, we see that the clusters slightly changed (fig. 10) but the general interpretation remains the same: 2 clusters seem to correspond to medium-loaded servers while 1 cluster represents low-loaded servers. The number of servers belonging to the 1st (0) cluster is about 550, which is roughly 75% of the total number of servers. What is more, several teams have more than 40 low-loaded servers (belong to the 1st cluster)

To conclude, our assumption seems valid and the company have a tendency to allocate to many servers as obviously the load in the 1st cluster is low and thus can be redistributed in order to get more free servers.

6.3. Building models to make predictions

In this part of the project my main objective was to build several machine learning models that will be capable of predicting the potential CPU, memory and disk usage. To train and evaluate the models, the data was further split into training and testing sets using 80:20 ratio. Initially, several regression models were built as the primary ones. So, four regression model were selected for evaluation: Linear Regression, Ridge Regression, Lasso Regression, ElasticNet Regression. Cross-validation was performed on the training dataset in order to select the best model based on the metrics mentioned above (MSE, MAE and R-squared).All in all, the best-performing regression models were ElasticNet (fig. 11) and Lasso, which showed similar metrics among the evaluated models. Then the models was validated using the test dataset. The evaluation metrics, including MSE, mean absolute error (MAE), and R-squared, were calculated to assess the performance of the model on the test data. Testing also revealed ElasticNet and Lasso Regressions as the best models.

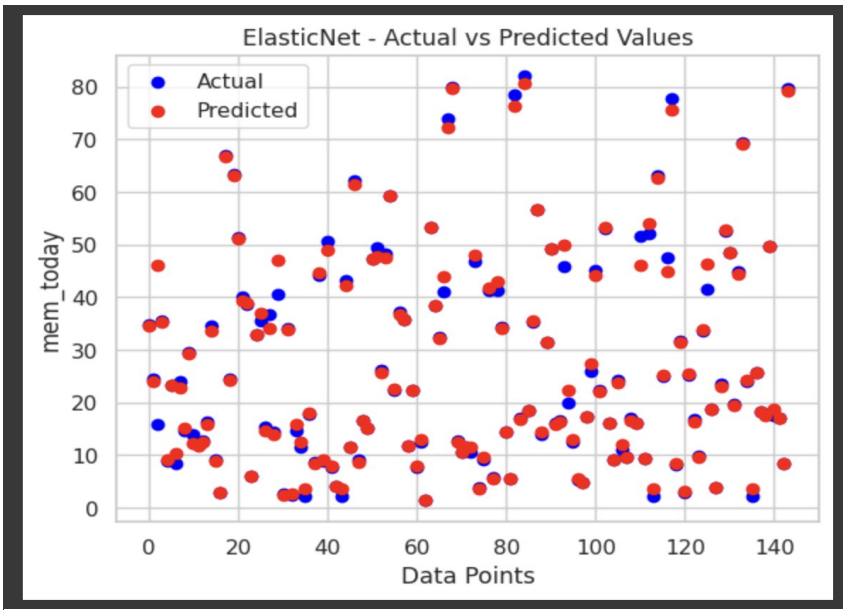

Figure 11. The graph with actual and predicted(via ElasticNet) memory usage.

However, it is crucial to compare the obtained results with several other models. For comparison I chose Random Forest, Decision Tree, Custom Neural Network and several well-known boosting algorithms (XGB, CatBoost, LGBM). This diversification of models will help us to find the model that fits the data perfectly. To select the best performing model, we compared the evaluation metrics across different models. By analyzing the performance metrics, we identified the model that achieved the lowest MSE, MAE, and highest R-squared value as the best performing model. The final results are presented below:

	Model	Mean Squared Error	Mean Absolute Error	R-squared
0	Linear Regression	19.086001	1.584879	0.964047
1	Ridge	16.681481	1.531113	0.968576
2	Lasso	10.703388	0.915361	0.979838
3	ElasticNet	10.407146	0.939078	0.980396
4	XGBRegressor	22.383498	1.454787	0.957835
5	CatBoostRegressor	20.546754	2.139720	0.961295
6	LGBMRegressor	13.233821	1.264016	0.975071
7	DecisionTreeRegressor	13.642375	1.356708	0.974301
8	RandomForestRegressor	11.668950	1.175505	0.978019
9	CustomNN	73.167465	3.947506	0.862172

Figure 12. The dataframe with the metrics for all considered models.

Even though Random Forest and Decision Tree showed decent results, our initial ElasticNet model should be chosen (fig. 12) as the final one.

6.4. Summary

Using the selected best-performing model, CPU/Memory/Disk load predictions were generated for future time periods. These predictions provide insights into the expected load levels and patterns, allowing resource managers to anticipate and allocate resources accordingly. The load predictions can aid in capacity planning, resource provisioning, and workload balancing to optimize system performance and ensure efficient resource utilization. Speaking about clusterization, the data was split into several clusters and the results showed that there are indeed several teams that have more than 40 servers with the average CPU, memory and disk load less than 30%, which clearly indicates the necessity of redistributing the load in order to have less ‘fully’ working servers.

7. Visualisation

In this project, the visualisation plays a crucial part since it allows to show the results of all workings in an easy and comprehensive way. By playing with different visualisation techniques, it is possible to make data look beautiful and understandable at the same time. Last one is especially important because the main goal is to improve a real company’s effectiveness of servers, and the presentation of our analysis and ideas should include details that are clear even to a non-IT person. Finally, the visualisation makes clear the business implications of the conducted analysis.

7.1. Prototype Creation

First step of my work was to draw the future look of the GUI (graphical user interface), so that I could understand where each element and visualisation should be so that the whole picture is holistic. For this purpose I chose the Figma platform, and the ultimate goal at this step was to make a prototype – a clickable version of the future GUI, which fully imitated its work.

After I talked to the project supervisor and understood what the interface should look like to be informative, I started creating the pages (see Appendix A). The first page (fig.15) shows the main page of the GUI. It has the total number of servers indicated at the top and a pie chart with distribution of problematic servers in the middle. By server problem here we understand exceeding the permissible limit in one of the metrics. The red zone of the pie contains servers with more than 2 occurring problems, the yellow zone – servers with 1 problem, and the green zone – with no problems. The exact number of servers falling into each category is also shown. From the main page we are able to reach 3 other pages, each related to the list of servers in each category of the pie chart. By clicking at the specific section we move to the page with either green (fig. 16), or yellow (fig. 17), or red (fig. 18) servers list. In the case of the last two, the metric(s) which caused the problem(s) to appear is/are highlighted on the right, next to the server's name. By clicking at the coloured bar, we go to the server page with related metrics (fig. 19). These are shown both as bar plots of average weekly load on each day and as line plots of historical load during last month. The final page in this line is the daily load page (fig. 20). It is opened if we click at one of the bars of the weekly bar chart, so that we choose the day that we are interested in. On this last page we can see a line graph with historical monthly load for each metric on a specific day chosen beforehand. Besides all these informative visualisations each page (except main one – there's no need in this) also has features for easier moving between them: breadcrumbs, pagination dots and scroll bar.

The complete prototype was presented to my supervisor. After fixing minor issues and getting positive feedback, I moved on to creating real data visualisations.

7.2. Visualisations in Kibana and Grafana

Two main instruments that provide a possibility to construct professional visualisations straight from the database are Kibana and Grafana. They are pretty similar in the way they work and look, the main difference is what they are used for. The purpose of Grafana is to analyse and visualise system usage indicators of processor, memory, disk and input/output. Grafana does

not allow full-text data queries. On the other hand, Kibana runs on Elasticsearch and is mainly used for analysing log messages [14]. It also presents a bigger range of possibilities to visualise data than Grafana. The last advantage of Kibana became a main reason why it was chosen to be the desired instrument.

Unfortunately, when I started working, it turned out that I did not have access to corporate data in this application, and it was not possible to solve this problem due to specific privacy rules of the company I am cooperating with. My supervisor recommended that I leave it and move to another application, where I am guaranteed to be able to make high-quality visualisations, without the need to connect to the corporate network, and it will be a worthy analogue of the previous tools.

7.3. Visualisations in Tableau

As an alternative to the aforementioned instruments, I chose Tableau. I already had an experience working with it and I considered it professional enough to make informative visualisations with a decent look. The dashboards, which can be made there out of several charts, are suitable both for company needs to obtain desired information about servers load, and for me to have as a project product. The visualisations made were mainly based on the Figma prototype, presented earlier. Besides this, thanks to the wide functionality of Tableau I was able to construct some additional plots, which were not planned to be included in the dashboard initially. This showed another advantage of working in this application.

Some preparation of data was conducted by the means of Python code and Pandas library before I started working on dashboards. The data frame I received from my group mates was unsuitable for making visualisations: it contained day offsets which cannot be processed correctly by the Tableau system. That is why I changed these offsets to real dates. I also made minimal modifications to the data table, like change of column names to more clear ones and removal of unnecessary columns.

The final product of my work on this project is a story, which is a collection of five dashboards (see Appendix B). Each dashboard represents its own topic (i.e. the graphs presented in it are put together on purpose), so it is possible to get a lot of different useful information and look at a single problem from different sides. Consider the first dashboard (fig. 21). It has three plots: pie chart, bar chart and line plot. The first one in the left upper corner shows the number of servers in each cluster (clusters were determined earlier and described in section 6.2), the

second one in the right upper corner displays average load with division by cluster and metric, and the third one, in the bottom, – average load dynamics by all three metrics throughout the observation period (around 1 month). From this dashboard it is easy to find out that more than 75% of servers belong to cluster 0, while the 2nd cluster has the smallest number of servers in it. Also, clustering successfully separated servers with low and high metrics values. The timeline showed that disk and memory load are doing generally better than CPU: they have higher load and are stationary on full range, while cpu is significantly lower and have spikes in usage on some days.

The next dashboard (fig. 22) has three big bubble graphs. They are huge due to the big number of servers being analysed. From left to right, average CPU, memory and disk load by server are shown: the bigger is the bubble, the higher is the load. Different colours indicate belonging to one of the teams. Even though this chart is quite messy, we are able to get a lot of useful information about server resources usage in each team. For example, the blue team[1] uses disk, memory and CPU resources quite uniformly among their servers, while the green team has some servers underusing memory and CPU (bubbles are very small compared to others). I presume this team does not need so many servers and those which stand idle can be removed to save energy and money.

The following three dashboards are very similar to each other. They have the same list of plots: pie chart in the right upper corner with number of servers having metric value below a threshold, bar plot in the upper centre with metric load weekly dynamics, bubble chart in the right upper corner which shows average metric load by team and box-and-whisker plot in the bottom depicting metric load dynamics throughout the observation period, and only differ by the metric analysed. The third dashboard (fig. 23) has a CPU metric in it. Pie chart shows that only a little number of servers are loaded above the low threshold, which is bad. This means the load on the servers should be redistributed in such a way as to keep at a medium level on all servers. Probably, this is a reason for the reduction in server count. The bar plot outputs quite predictable statistics: CPU load is smaller on Saturday than on the other days of week, while the smallest level is reached on Sunday, both of them being weekends. During the week, the highest CPU load is observed on Friday. Moving on to the bubble chart, we have the brown team as an absolute leader in CPU usage, while the purple and yellow teams underuse it. Lastly,

[1] Here and afterwards in this part team names were changed from the original.

on each day of the observation period CPU usage was quite low on average, with only single servers out of many showing high results.

The fourth dashboard (fig. 24) holds information about memory load, and here everything looks better than on the CPU page. Pie chart has almost half-and-half division into servers with memory load above threshold and below it. Over the weeks and last month in general, the memory usage is basically the same every day. In teams, blue and red ones seem to be underusing resources compared to the others, but the difference is not very big.

The last dashboard (fig. 25) tells us how disk load goes on, and it appears to be the most stable among all metrics. Pie chart shows that about ⅔ of all servers use disk above threshold level, which is good. Weekly load is constant every day, and similarly each day of the observation period has a common disk load average and maximum. Most of the teams have a decent load of disc, except the purple and blue.

7.4. Direct Connection between Tableau and Jupyter Notebook

As an additional task, I decided to implement a direct connection between the visualisation application, Tableau, and Jupyter notebook, where the data was modified before the visualisation step. Initially, it was necessary to export data from Jupyter as a CSV file, and then load it manually to Tableau. I proposed that it takes too much time and human actions to simply transfer a table between two apps, and so I decided to automate this process. This improvement is also significant for company needs, since it allows to combine all steps of work completed by different group members of the project into one uninterrupted pipeline, with Prometheus database as the input and database in Tableau ready to be visualised as an output.

After conducting research, I found a library that perfectly suited my needs in linking two apps: Jupytab. It allows to explore data in Tableau which is dynamically transferred from Jupyter notebook [15]. That is, every time data is changed in the notebook, Tableau will instantly update the database and alter the visualisations made.

To start with, the Jupytab was loaded to the Jupyter notebook with data frame preparation code (fig. 13). It was necessary to load both jupytab and jupytab-server libraries, since first one cares about notebook and second one allows to use server for data transmission. Then, I created an instance of a publication-ready table and assigned to it the data frame with required information about server load (fig. 14). In the next two cells I allows Tableau to retrieve data from the table

(as noted in documentation [15], the occuring error is harmless to the whole process) and the schema of all available types in the table. The last is done for troubleshooting and to monitor that data is transmitted correctly and completely to Tableau. As a last step, by the use of a special configuration file, I was able to open the port for connection and tell Tableau the path to my notebook file. At this point, the table with data is sent to the port, while I simply open the same port in Tableau and obtain the live connection to the database.

```
!pip install jupytab-server

Requirement already satisfied: jupytab-server in c

!pip install jupytab

Requirement already satisfied: jupytab in c:\users

import jupytab
```

Figure 13. Installation of jupytab and jupytab-server.

```
tables = jupytab.Tables()
tables['load'] = jupytab.DataFrameTable('Server Load', dataframe=data_vis, include_index=True)

# GET /schema
tables.render_schema()

[{"id": "load", "alias": "Server Load", "columns": [{"id": "metric_instance", "dataType": "strin

# GET /data
tables.render_data(REQUEST)

Traceback…
NameError: name 'REQUEST' is not defined

!jupytab --config=config.ini
```

Figure 14. Implementation of algorithm that sends data from Jupyter notebook directly to Tableau through jupytab.

8. Conclusion

Throughout the work, we managed to build a complete flow from raw Prometheus metrics to interactive dashboard visualising the historic usage of the company's machines and the obtained ML insights. To make it possible, we had to develop a custom Prometheus client,

resolve the issue of systematic data incompleteness, implement and tune a series of ML models, design the visualisations, and finally set up a channel to deliver our Python results to the Tableau dashboards. We found the regularities in the way the company's employees use their development stands making conclusions on how the server usage can be optimised and what expenditures on computational power the business should expect in future.

8.1. Future Improvements

Although the obtained results are already quite impressive, there is always room for improvement. For example, at its current stage, the system needs to be launched manually for the scripts to work and the results to update. Redesigning it to allow dynamic data collection and processing would give a significant boost to the quality of user experience. Data collection might be automated to receive new data every fixed period of time (for instance, gather new data every day), or when the user requires it. We can also vary the metrics (other metrics might be more illustrative or meaningful), although the provided ones show relatively good performance. In general, this work can be pipelined to arrive at better results or allow more convenient use.

As for data preprocessing, the proposed reconstruction algorithm has performed pretty well as compared to the existing generalised analogues. Nevertheless, one may notice a slight trend appearing when the data is reconstructed for a large time period. Experiments with the model's hyperparameters are likely to further improve its accuracy. Moreover, given the success of *KNNImputer* on the testing dataset, one could try to achieve even better performance tuning this package solution by, for instance, developing a custom time-based distance metric.

Concerning ML, even though several models showed great performance and produced accurate predictions, we must not stop on this. Some companies use reinforcement learning (RL) to analyse the distribution of server resources. In a reinforcement learning setup, the server is usually considered as the environment and the agent can be a software program that makes decisions about resource allocation. The agent will learn from its past decisions and adjust the strategy to optimise the reward signal over time. Clearly, this task is very challenging but the potential results obtained from the agent may outperform all current models.

The visualisation step is one that certainly can be extended. Although 5 dashboards, which are ready to this point, do reflect information about the state of server utilisation by various metrics, this may still not be enough to make a certain conclusion about each server's state. A lot of

other plots can be added, and the current controversial plots (like bubble plots on dashboard 2) can be changed into something more obvious and indicative. This way, the number of dashboards for a complete report on the servers status must be around 15. The work on this part of the project will be continued to benefit the company.

Appendixes

Appendix A. Figma Prototype Pages

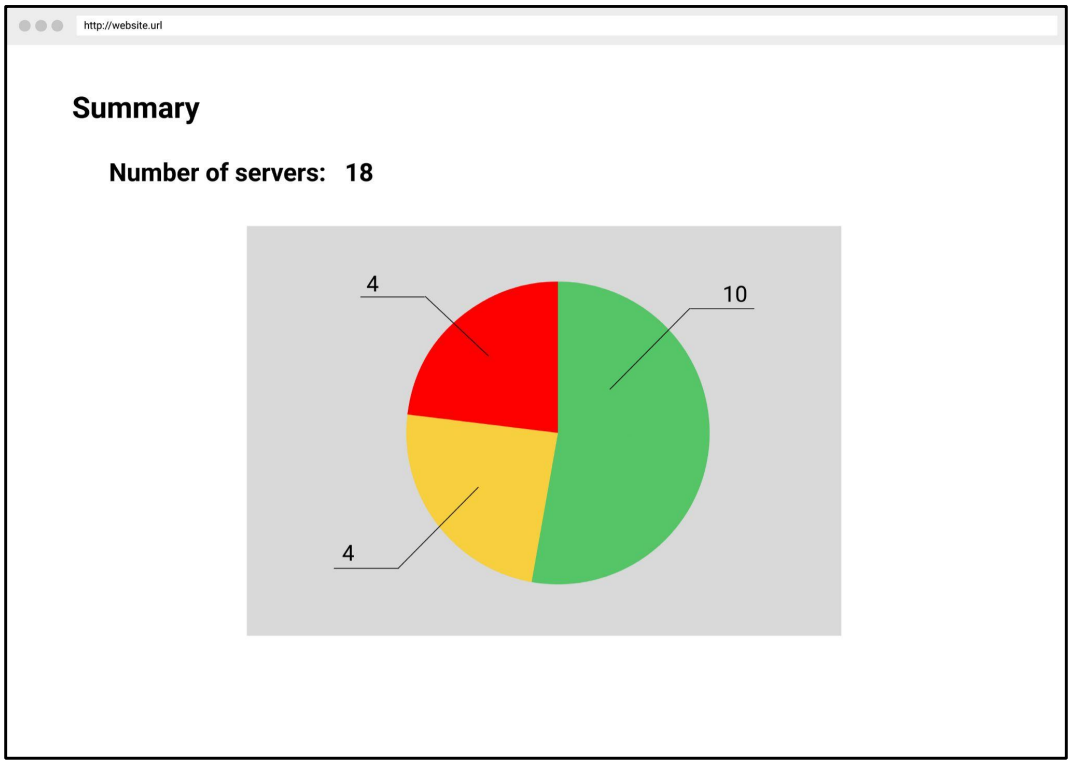

Figure 15. First page, or main page of the prototype. It contains the number of servers and pie chart with their distribution taking into account their problems.

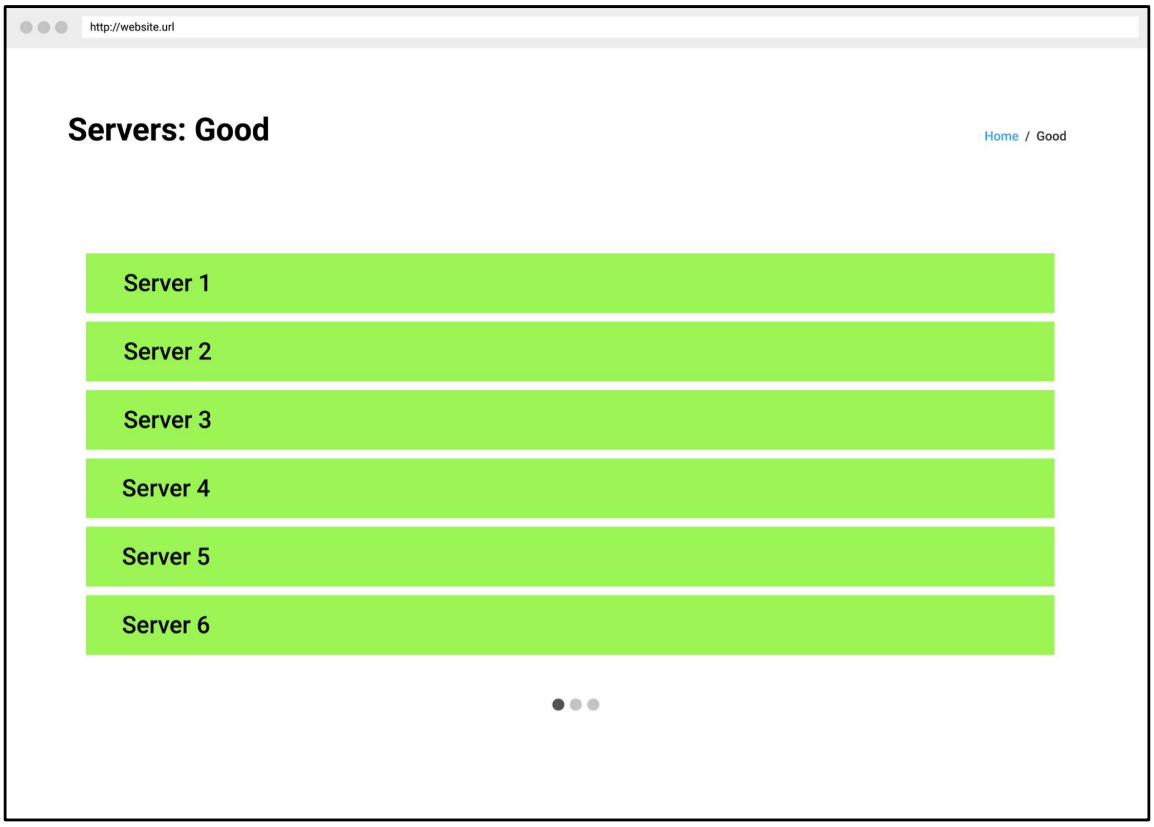

Figure 16. Page with servers coloured green. No problems found in these servers.

Figure 17. Page with servers coloured yellow. One problem found in these servers. Problem is indicated on the right of the server name.

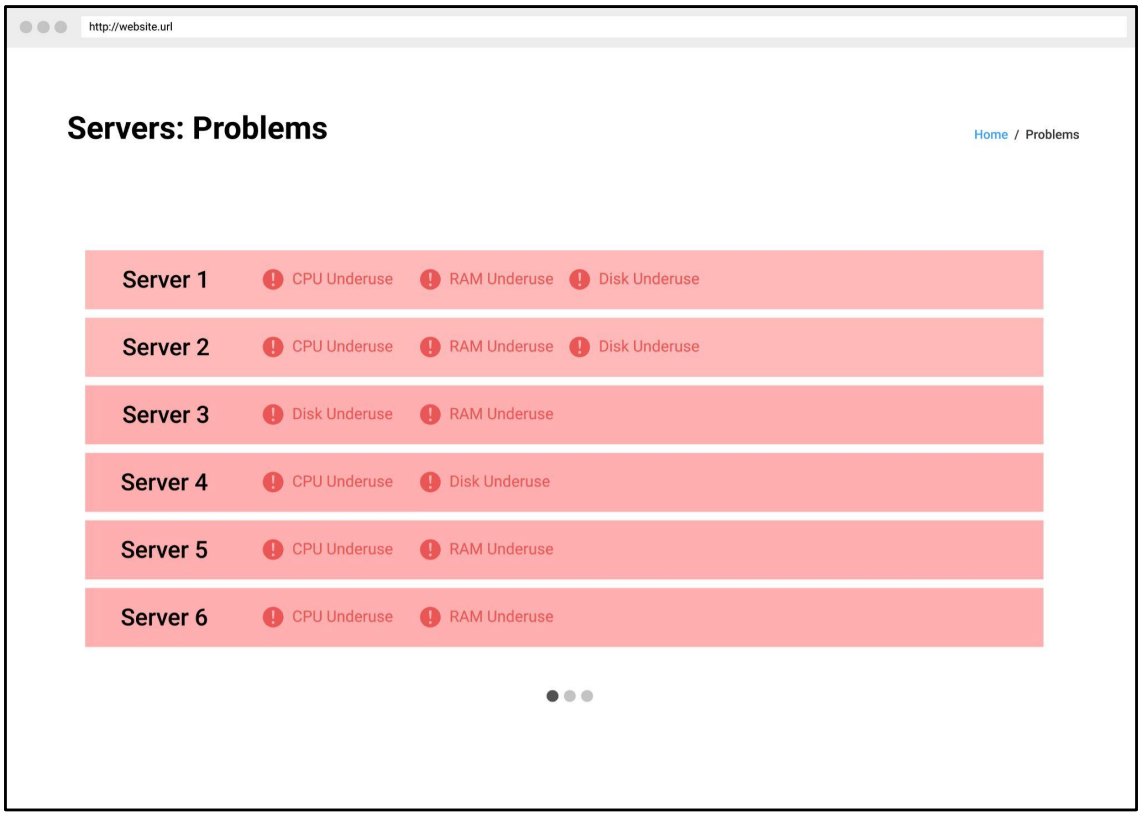

Figure 18. Page with servers coloured red. Two or more problems found in these servers. Problems are indicated on the right of the server name.

Figure 19. Page with server metrics. It shows history of last month and average load by weekday for every metric.

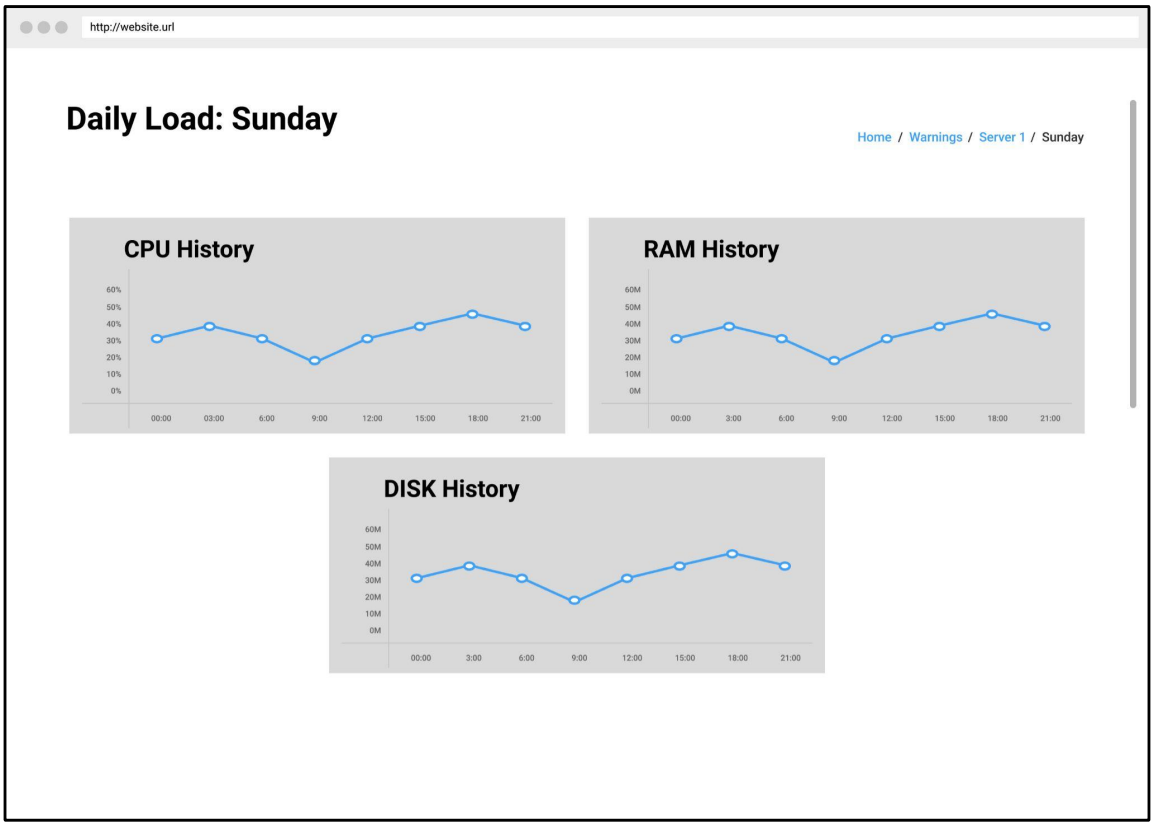

Figure 20. Page with daily load for every metric on a specific server.

Appendix B. Tableau Dashboards

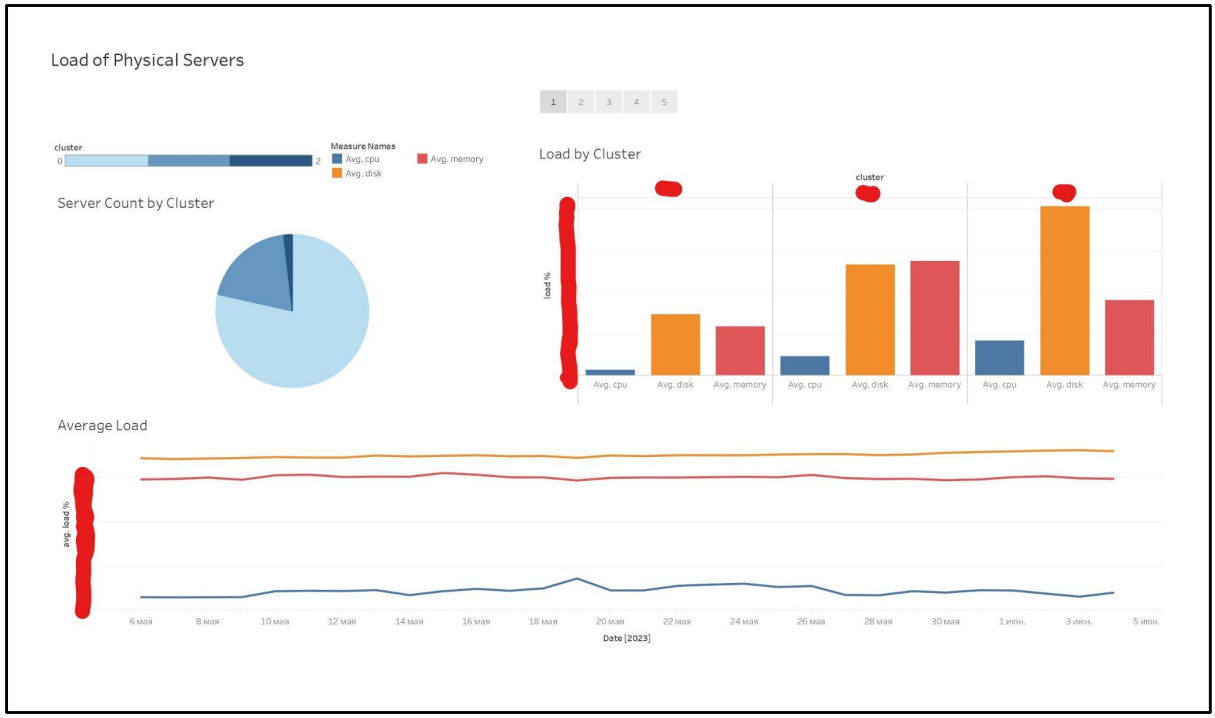

Figure 21. Dashboard 1. Server count by cluster, average load by cluster and metric, average load dynamics throughout the observation period.

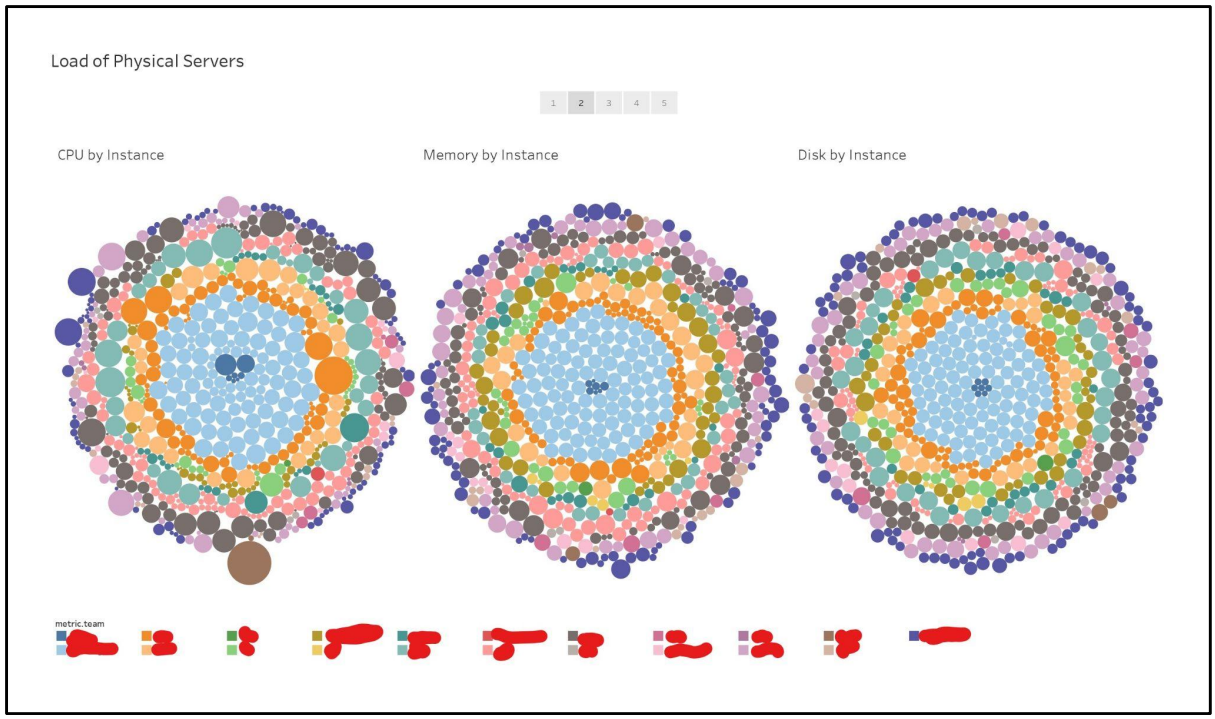

Figure 22. Dashboard 2. Average CPU, memory and disk load by server.

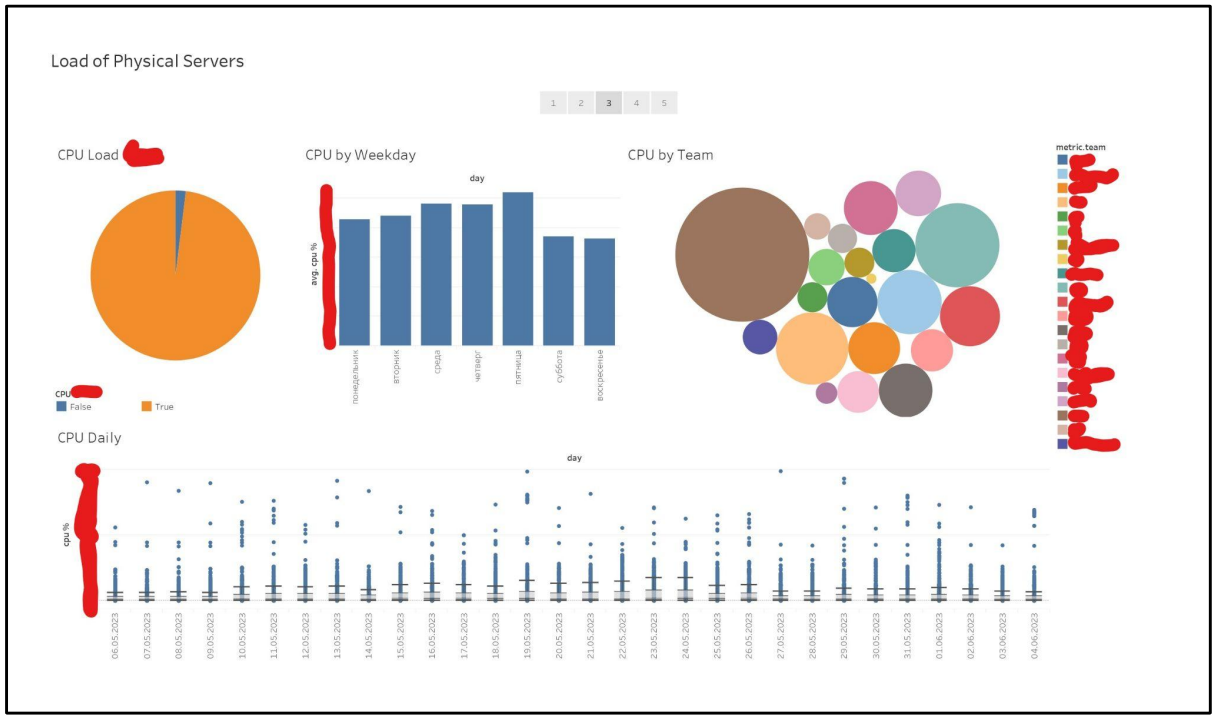

Figure 23. Number of servers using CPU below a threshold, CPU load weekly dynamics, average CPU load by team, CPU load dynamics throughout the observation period.

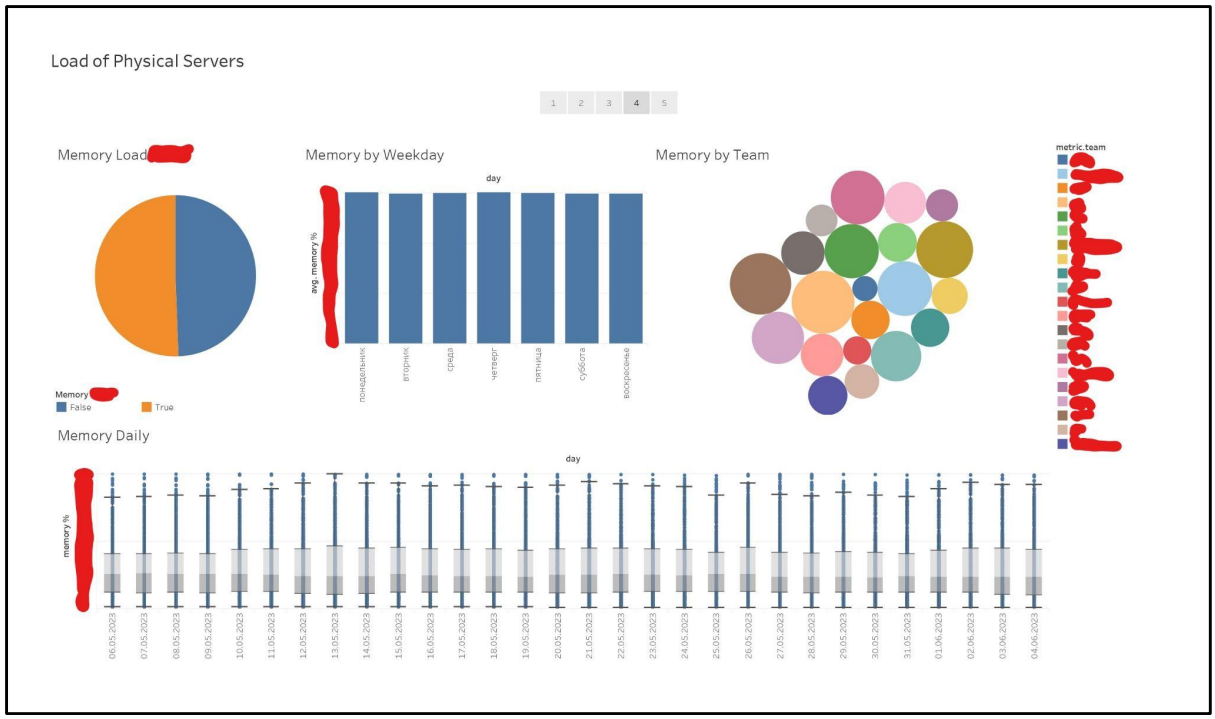

Figure 24. Number of servers using memory below a threshold, memory load weekly dynamics, average memory load by team, memory load dynamics throughout the observation period.

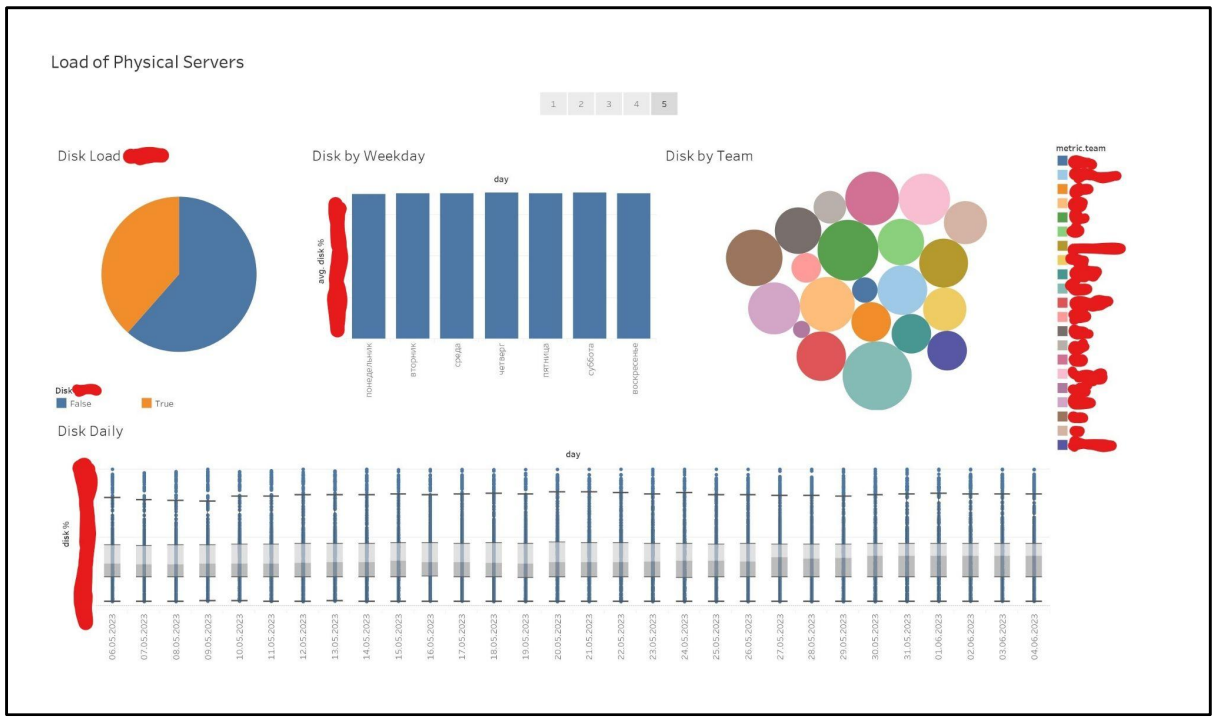

Figure 25. Number of servers using disk space below a threshold, disk load weekly dynamics, average disk load by team, disk load dynamics throughout the observation period.

References

1. Samuel, Nicholas. “What Is Tableau? - A Comprehensive Guide.” *Hevo*, 11 Nov. 2021, hevodata.com/learn/what-is-tableau/.

2. “6.4. Imputation of Missing Values.” *Scikit Learn*, 2023, scikit-learn.org/stable/modules/impute.html#impute. Accessed 9 June 2023.

3. Buuren, Stef Van, and Karin Groothuis-Oudshoorn. “**Mice**: Multivariate Imputation by Chained Equations in*r*.” *Journal of Statistical Software*, vol. 45, no. 3, 2011, https://doi.org/10.18637/jss.v045.i03.

4. “Sklearn.Impute.IterativeImputer.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html#sklearn.impute.IterativeImputer. Accessed 9 June 2023.

5. “Sklearn.Linear_model.Bayesianridge.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html#sklearn.linear_model.BayesianRidge. Accessed 9 June 2023.

6. “Sklearn.Impute.KNNImputer.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer. Accessed 9 June 2023.

7. “Sklearn.Metrics.Pairwise.Nan_euclidean_distances.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.nan_euclidean_distances.html. Accessed 9 June 2023.

8. “The Elements of Statistical Learning: Data Mining, Inference, and Prediction” - Trevor Hastie, Robert Tibshirani, Jerome Friedman, p. 206 https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12.pdf

9. “The Elements of Statistical Learning: Data Mining, Inference, and Prediction” - Trevor Hastie, Robert Tibshirani, Jerome Friedman, p. 11 – 18 https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12.pdf

10. “Statsmodels.Regression.Linear_model.Ols¶.” *Statsmodels.Regression.Linear_model.OLS - Statsmodels 0.15.0 (+14)*,

www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html. Accessed 9 June 2023.

11. “SKLEARN.METRICS.MEAN_SQUARED_ERROR.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_erro. Accessed 9 June 2023.

12. “SKLEARN.METRICS.MEAN_ABSOLUTE_ERROR.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error. Accessed 9 June 2023.

13. “SKLEARN.METRICS.MEAN_ABSOLUTE_PERCENTAGE_ERROR.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_percentage_error.html#sklearn.metrics.mean_absolute_percentage_error. Accessed 9 June 2023.

14. “Sravnenie Mezhdu Grafanoj i Kibanoj.” *Sravnenie Mezhdu Grafanoj i Kibanoj - Russkie Blogi*, https://russianblogs.com/article/4657389642/.

15. Tribondeau, Brian. “CFMTech/Jupytab: Display in Tableau Data from Jupyter Notebooks.” *GitHub*, 2023, github.com/CFMTech/Jupytab.

16. Prioglio, Charly. “Wireframe Component Library.” *WCL - Figma*, Nov. 2018, https://www.figma.com/file/gOZcXeqIvibFX26rd0bMTw/Wireframe-Component-Library-v1.0-(Copy)?type=design&node-id=805%3A0&t=MXpNXLCnyf6gqFco-1

17. Feature/Variable importance after a PCA analysis - https://stackoverflow.com/questions/50796024/feature-variable-importance-after-a-pca-analysis

18. “An Introduction to Statistical Learning with Applications in R” - Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, p. 59 – 103 https://www.statlearning.com

19. “The Elements of Statistical Learning: Data Mining, Inference, and Prediction” - Trevor Hastie, Robert Tibshirani, Jerome Friedman, p. 501 – 553 https://hastie.su.domains/ElemStatLearn/printings/ESLII_print12.pdf

20. Bader, Andreas, Oliver Kopp, and Michael Falkenthal. "Survey and comparison of open source time series databases." *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband* (2017).