Network Data

authors: *Dominika Rzepka (271301), Mateusz Guściora (228884)*
subject: *Modelling and Analysis of Web-based Systems [Lab. Tuesday 15:15]*

Raport Structure:

1. General information, Project pipeline, Program structure
2. First look on data
3. Loading data
4. Cleaning data
5. Analysis of data
6. Scaling data
7. Predictions, evaluation
8. Experiments
9. Conclusions

General information

The project involves the analysis of network log data obtained from RIPE Atlas measurements. The data, stored in a JSON file named RIPE-Atlas-measurement-50728410 is loaded into the program and merged for further analysis. The Python-based program consists of modules including loading, cleaning, analyzing, and prediction, which are executed sequentially. The data preprocessing steps involve cleaning and handling duplicates, missing, outlying data and scaling the data. The analysis phase focuses on exploring the remaining data and extracting valuable insights, while the prediction phase involves building and evaluating models. The program structure includes separate folders for analysis and evaluation, where outputs such as charts, tables, models, and plots are saved. The data provides log entries for HTTP measurements.

DataSet: Data of network logs in JSON file named "*RIPE-Atlas-measurement-50728410*"
Environment: Python 3.9 - libraries: pandas, numpy, sklearn, matplotlib, seaborn, json.
Visual studio code, excel and csv extensions, Excel, Orange tool.

Project pipeline

First look on the data → Loading datasets into program and merging into new → Preprocessing - cleaning part → Analyzing remaining data → Preprocessing - scaling/standardizing/normalizing → Prediction (building a model and evaluating) → Experiments → Formalize Conclusions.

Program structure

A program written in Python is divided into modules that are run through the 'main.py' file. This file runs consecutively files that are put into the src folder, that is 'loading', 'cleaning', 'analyzing', 'prediction'. Each module is a functionality running after the previous one. Data derived from functionalities that can benefit in analysis and the output of this analysis (charts,

tables, minimum values, maximum values, averages, medians…) are saved in the ‘analysis’ folder. Output of prediction and evaluation is saved in the ‘evaluation’ folder. The program is very flexible in terms of a prediction module where users can select models, attributes etc. Program structure is represented below.

- Program - “network logs loading-cleaning-analyzing-predicting”
    - RIPE-Atlas-measurments-50728410.json
    - RIPE-Atlas.csv
    - main.py
    - src folder
        - __init__.py
        - ex3_loading.py
        - ex3_cleaning.py
        - ex3_analyzing.py
        - ex3_prediction
    - analysis folder
        - charts, tables, minimum values, maximum values, averages, medians…
    - evaluation folder
        - models, evaluations, plots
    - readMe, requirements, .gitignore

First look on data

Log entries for HTTP measurements conducted using RIPE Atlas. RIPE Atlas is a global network measurement infrastructure that allows users to conduct various network measurements from different vantage points around the world. Each row in the data corresponds to a specific measurement, capturing attributes such as firewall details, source and destination addresses, measurement results, timestamps, and more. The version that is in project is: Version 4460 HTTP it produces JSON file with the result of (source: https://atlas.ripe.net/docs/apis/result-format/):

- "from" -- IP address of the probe as known by controller (string)
- "msm_id" -- measurement identifier (int)
- "prb_id" -- source probe ID (int)
- "result" -- results of query (array)

Each element is an associative array consisting of:

- "af" -- address family, 4 or 6 (integer)
- "bsize" -- size of body in octets (int)
- "dnserr" -- [optional] DNS resolution failed (string) (not include in the data)
- "dst_addr" -- target address (string)
- "err" -- [optional] other failure (string) (include in the data)
- "header" -- [optional] elements are strings. The last string can be empty to indicate the end of enders or end with "[...]" to indicate truncation (array) (not include in the data)
- "hsize" -- header size in octets (int)

- "method" -- "GET", "HEAD", or "POST" (string)
- "res" -- HTTP result code (int)
- "rt" -- time to execute request excluding DNS (float)
- "src_addr" -- source address used by probe (string)
- "subid" -- [optional] sequence number of this result within a group of results, when the 'all' option is used without the 'combine' option (int) (not include in the data)
- "submax" -- [optional] total number of results within a group (int) (not include in the data)
- "time" -- [optional] Unix timestamp, when the 'all' option is used with the 'combine' option (int) (not include in the data)
- "ver" -- major, minor version of http server (string)

- "timestamp", "stored_timestamp"-- Unix timestamp (int)
- "lts" -- last time synchronized . How long ago (in seconds) the clock of the probe was found to be in sync with that of a controller. The value -1 is used to indicate that the probe does not know whether it is in sync (int)
- "type" -- "http" (string)
- "uri" -- request uri (string)

There are other elements that are included, such as:

- "fw": identifies the firmware version used by the probe that generated that result
- "mver": that specifies the version of the measurement code. This field has the format "x.y.z", where the "x" field is that major version, which changes when the measurement results are incompatible with the previous version. The "y" field is the minor version, and changes when new fields are added, but otherwise old parsers can still parse measurement results. Finally, the "z" field specifies that the measurement code changed, but the output format is still the same. This happens when only (minor) bugs are fixed and no new features are added.

loading and first look on data

	method	af	dst_addr	src_addr	rt	res	ver	...	prb_id	timestamp	msm_name	from	type	group_id	stored_timestamp
0	GET	4	185.229.88.69	185.229.88.69	1.439729	200.0	1.1	...	7183	1678215442	HTTPGet	185.229.88.69	http	50728408	1678215556
1	GET	4	185.229.88.69	172.31.0.48	26.646395	200.0	1.1	...	1000566	1678217243	HTTPGet	35.156.148.12	http	50728408	1678217292
2	GET	4	185.229.88.69	109.48.204.209	124.118691	200.0	1.1	...	1001377	1678217239	HTTPGet	109.48.204.209	http	50728408	1678217329
3	GET	4	185.229.88.69	95.92.246.110	125.328691	200.0	1.1	...	1001722	1678217245	HTTPGet	95.92.246.110	http	50728408	1678217300
4	GET	4	185.229.88.69	95.95.32.172	119.390321	200.0	1.1	...	1001843	1678217241	HTTPGet	95.95.32.172	http	50728408	1678217304

Figure: First look on the data in python program

Loading Data

In this phase, the program efficiently handles the data loading process. The data is extracted from a JSON file named "*RIPE-Atlas-measurement-50728410*" and transformed into a CSV format. The implementation utilizes popular Python libraries, such as pandas and json, to accomplish tasks like loading the JSON file, handling nested columns, converting the data into a dataframe, and finally saving it as a CSV file. The resulting data frame is then seamlessly passed on to the subsequent module of the program. To provide an overview of the data, the program generates basic statistics about the dataframe. Currently, the data frame consists of 262,672 rows, each representing a unique measurement, and 22 columns, each representing an attribute associated with the network log data. This efficient data loading process ensures the smooth flow of information for further analysis and processing.

The provided data structure represents a dataframe consisting of various columns with their data type format in pandas libraries, including *method (*object*), af (*integer*), dst_addr*

*(*object*), src_addr (*object*), rt (*float*), res (*float*), ver (*object*), hsize (*float*), bsize (*float*), err (object), fw (*integer*), mver (*object*), lts (*integer*), uri (*object*), msm_id (*integer*), prb_id (*integer*), timestamp (*integer*), msm_name (*object*), from (*object*), type (*object*), group_id (*integer*), and stored_timestamp (*integer*).*

Cleaning data

In this section, the main part of the preprocessing phase takes place. It is respectively handling duplicates, handling missing data and handling outliers. Cleaning data was performed using the pandas library. As a result of this phase, the program outputs information about duplicates, missing data, outliers, what attributes were considered for outliers and produces csv for cleaned data frame which is used in next steps. Preprocessing phase had two approaches to the problem - analyzing data. First consideration was about getting rid of data that were missing or was considered a large outlier value in order to make regression prediction for target 'timestamp'. Although, due to the fact that results were poor, it was decided to do regression on target 'lts' attribute. The second consideration was about keeping missing data, except 'err' missing values, which were imputed, and outlying data to analyze and predict error information that is in the '*err*' column.

First consideration included diagnosing duplicates, missing data, and outliers. Then these values were either dropped or imputed. Below there is a description of the diagnosis and actions that were taken.

Handling duplicates required no action, as it was checked and printed that there are no duplicate rows. Lots of missing data were handled one by one method, this means that each attribute was considered separately. Handling missing data was performed after diagnosing the real situation of missing data, which is presented in the table below. As a result, none of the missing data was left in the data frame.

Column name	**No. of NaN**
fw	0
mver	3844
lts	0
err	260722
method	0
af	0
dst_addr	0
src_addr	1592
rt	1950
res	1950
ver	1950
hsize	1950
bsize	1950
uri	0
msm_id	0
prb_id	0
timestamp	0
msm_name	0
from	45
type	0
group_id	0
stored_timestamp	0

Table: Numbers of missing data in each column

As a result of this diagnosis, the following actions, concerning missing data, were made:

- Column ‘err’ missing values were filled with ‘no_error’, because missing data (lack of information) indicates no losing connection, therefore no error.
- Missing values from *‘rt’, ‘res’, ‘ver’, ‘hsize’, ‘bsize’* that is equal number of 1950 were dropped
- Missing values from the *‘mver’* column were dropped.
- Due to the above actions, the missing data in the ‘*src_addr’* and *‘from’* columns has also been dropped.

Handling Outliers was performed after handling missing data. Then diagnosing the number of outliers and related attributes was performed. For detecting outliers, the Z score was provided. As a result of this diagnosis, remaining outliers were dropped. Below there are context numbers.

- Number of outliers: 4963.
- Outliers percentage: 1 %
- Attributes considered for outlier detection: *'fw', 'hsize', 'lts', 'rt'.*

As a result of these actions, the data frame consists of 252 608 rows, each representing a unique measurement, and 22 columns, each representing an attribute associated with the network log data.

Second consideration was about preparing data for prediction of errors. To this target, it was decided to not get rid of missing or outlying data, so the number of rows is equal to the original one. Nevertheless, diagnosing duplicates, missing data, and outliers was also conducted. Instead of dropping this data, missing data were imputed with either ‘*NaN*’ or 0 in format float or int depending on attribute. Missing values in ‘err’, which were the majority, were replaced by ‘no_error’ string values. Outlying values were not dropped as well. Additionally, ‘err’ values were grouped into 7 classes: *'connect: Connection refused', 'connect: Host is unreachable', 'connect: Network unreachable','connect: No route to host', 'connect: timeout', 'no_error', 'timeout reading'.*

Analysing data

In this section it is discussed analysis of data frames prepared firstly for regression prediction of target ‘timestamp’ and ‘lts’ and secondly for classification prediction where the target of prediction are values in the ‘err’ column. Due to that fact, analysis with visualization needed to be performed on two slightly different data frames, which is more detailed described in sections ‘cleaning data’ and ‘loading data’. Heatmaps, histograms, boxplots and basic statistical calculations were plotted and briefly described.

Analysis for first consideration for prediction of ‘timestamp’ or ‘lts’ is presented below with plots and brief description. New data set is named ‘cleaned_networkdata’. Analysis was visualized and based on heat maps to investigate correlation, histograms to investigate distribution and boxplots to analyze mean and outliers. Boxplots were plotted for attributes that their plotting were meaningful, that means attributes that were numeric and had more than few instances but not a very large number of instances. Next, analysis of basic statistical calculations was conducted.

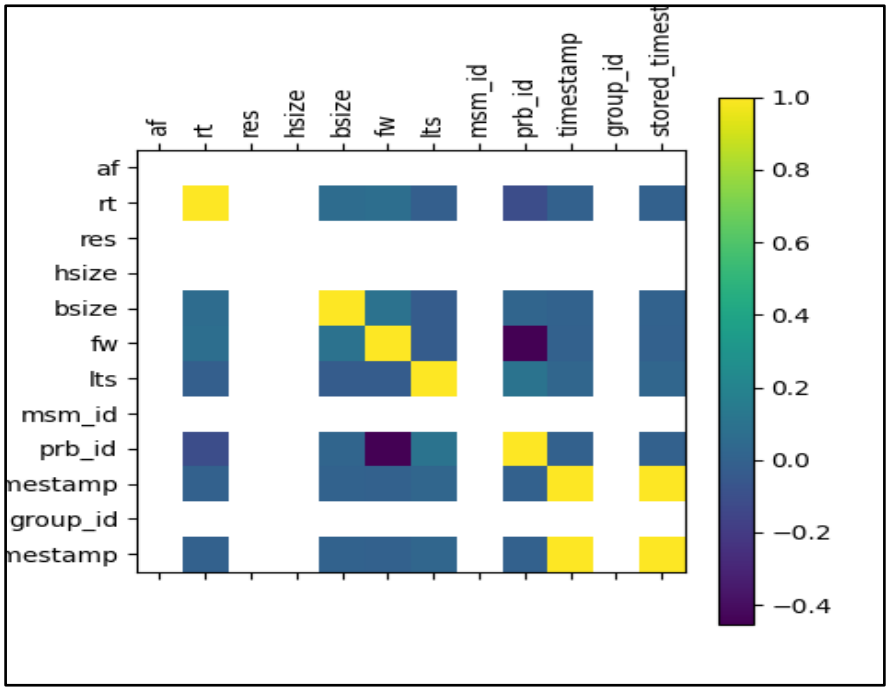


Figure: Heatmap of attributes, dropped data

Some of the parameters have no influence on each other. This is because the columns have only one value, which is the same on every other parameter. These columns could be deleted, and the value should be written down for feature reminder.

There is also a high correlation between “fw” and “prb_id” as it can be seen on heatmap. In some cases, the source probe ID may be used as a unique identifier for a particular physical or virtual probe device within the network. This ID may not directly indicate the firmware version, but can be used to associate the collected data with a specific probe instance. On the other hand, the firmware version represents the software or firmware installed on the probe device, which determines its functionality, capabilities, and behaviour. The firmware version can be used to track the software updates, bug fixes, and feature enhancements applied to the probe. In certain systems, there may be a mapping or relationship between the source probe ID and the firmware version used by that probe. This mapping allows administrators or analysts to identify the firmware version associated with a particular probe based on its unique ID.

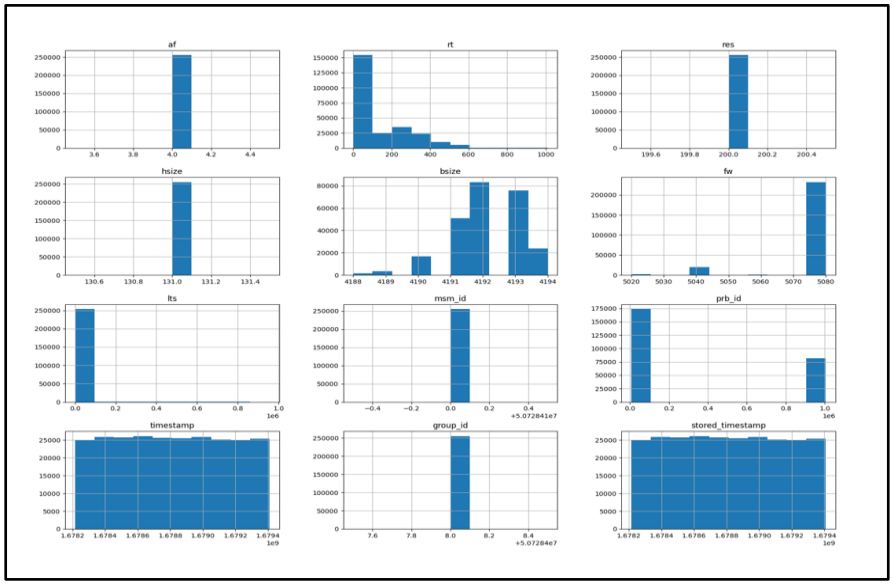


Figure: Histograms of attributes distribution, dropped data

In the histogram, there is a clear view of what was written in previous sentences. The columns: “af”, “res”, “hsize”, “msm_id” and “group_id” have one and the same value. The widest are “timestamp” and “stored_timestamp”.

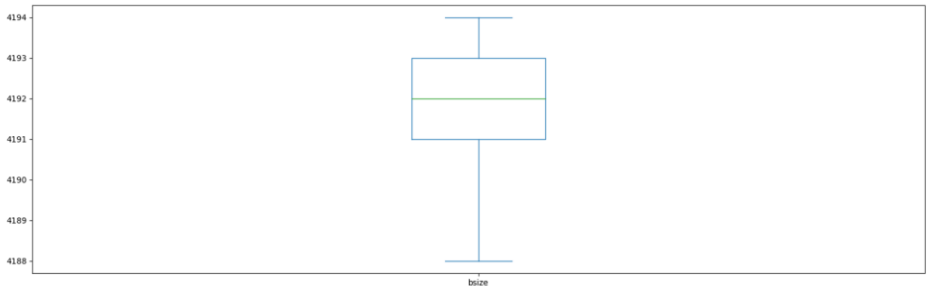


Figure: Boxplot for ‘bsize’ attribute

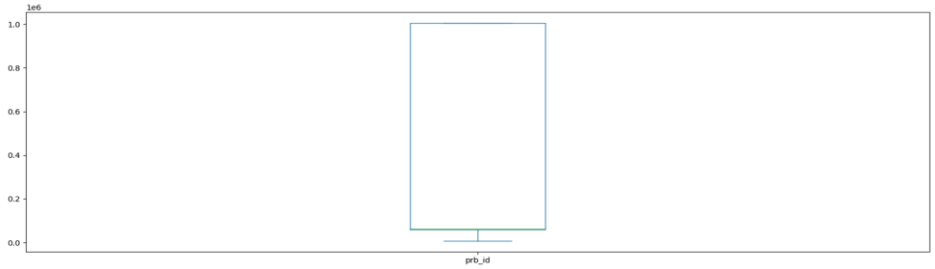


Figure: Boxplot for ‘prb_id’ attribute

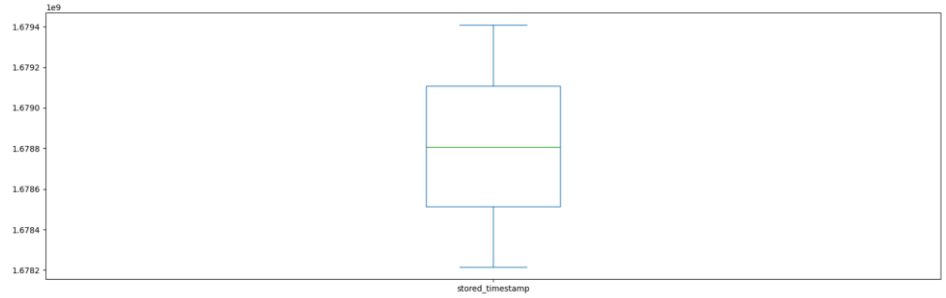


Figure: Boxplot for ‘stored_timestamp’ attribute

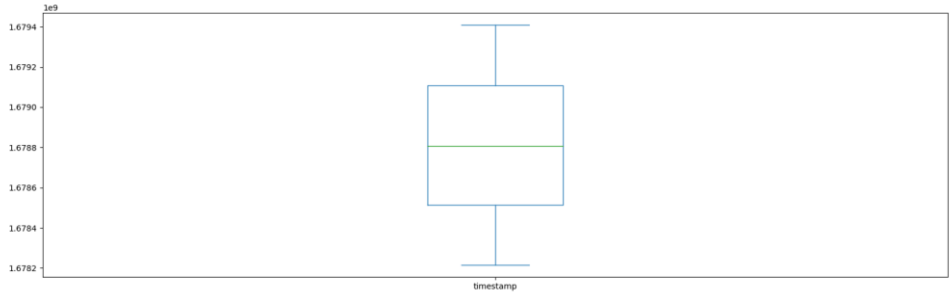


Figure: Boxplot for 'timestamp' attribute

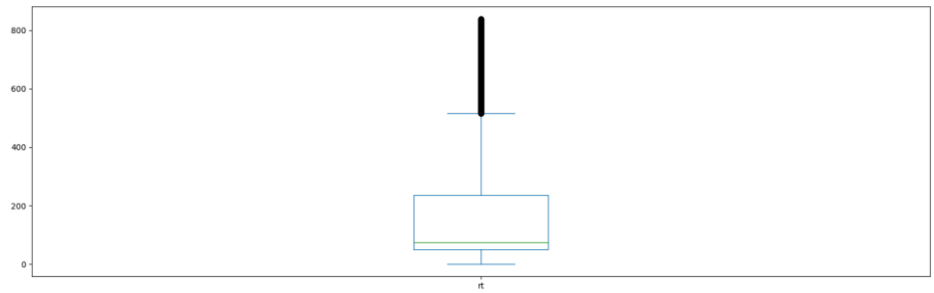


Figure: Boxplot for 'rt' attribute

In most of the boxplots we cannot see any of the outliers. They are visible only in "rt".

	af	rt	res	hsize	bsize	fw	lts	msm_id	prb_id	timestamp	group_id	stored_timestamp
count	255444	255444	255444	255444	255444	255444	255444	255444	255444	255444	255444	255444
mean	4	149,7409	200	131	4192,094	5076,295	2632,634	50728410	357010,9	1678810749	50728408	1678810902
std	0	146,7292	0	0	1,149928	11,90463	37494,55	0	444054,9	342845,8653	0	342826,2557
min	4	0,129394	200	131	4188	5020	-1	50728410	7183	1678215442	50728408	1678215556
25%	4	49,654	200	131	4191	5080	7	50728410	60347	1678514238	50728408	1678514340
50%	4	74,89202	200	131	4192	5080	24	50728410	61994	1678807646	50728408	1678807786
75%	4	236,807	200	131	4193	5080	52	50728410	1004345	1679106443	50728408	1679107147
max	4	1010,501	200	131	4194	5080	955421	50728410	1005475	1679408848	50728408	1679409033

Figure: Basic statistical calculation, dropped data

In the table, the one value columns are clearly visible. They have 0 in the standard variation, and all the other fields (except count) are the same. The columns are: "af", "res", "hsize", "msm_id" and "group_id". Similarity can be seen in the "timestamp" and "stored_timestamp" There is only a small difference which can be caused by slow transfer. The difference is only 153 in mean value,114 in max value and 185 in max value. A source probe ("prb_id") is a device or component that collects data or information from a particular source or location within a network. The source probe ID is used to differentiate and track data collected from different probes within the system. In this dataset, the mean value is 357 010,9; min value is 7 183 and max value is 1 005 475. Time to execute requests excluding DNS ("rt") has the lowest values of all the data. It starts at approximately 0,129 (min) and ends at 1010,501 (max). The mean value is something between 50% and 70% which is 149,7409. Most of the firmware version is ("fw") 5080 which is seen in all the percentiles, but true values are between, 5020 and 5080. The mean is also next to the percentile value.

Analysis for second consideration for prediction of '*err*' attribute is presented below with plots and brief description. New data set is named 'cleaned_networkdata'. Analysis was visualized and based on heat maps to investigate correlation and histograms to investigate distribution. Next, analysis of basic statistical calculations was conducted.

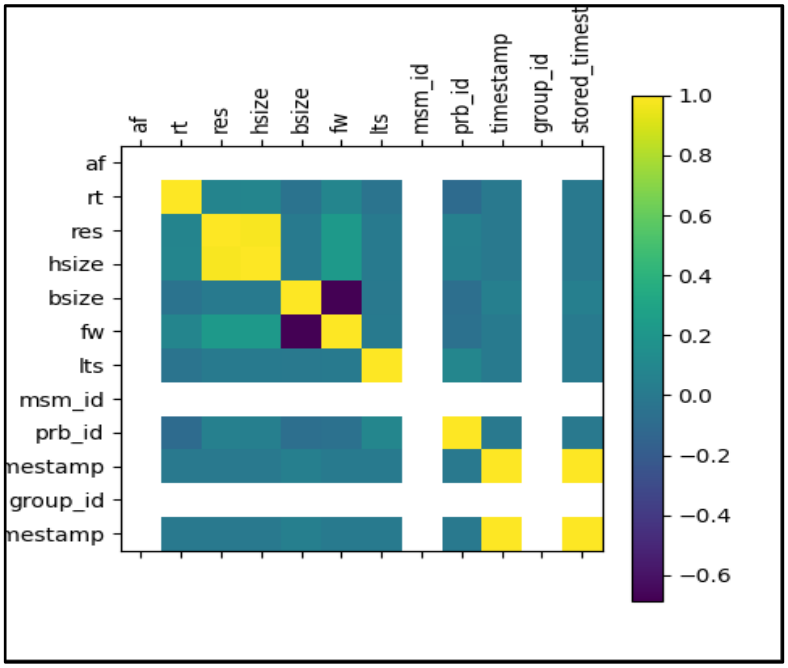


Figure: Heatmap of attributes for Http network_logs data

In here there is no strong connection except of "timestamp" and "stored_timestamp" as well as "fw" and "bsize". As in previous, in here there are also one value elements: "af", "msm_id" and "group_id".

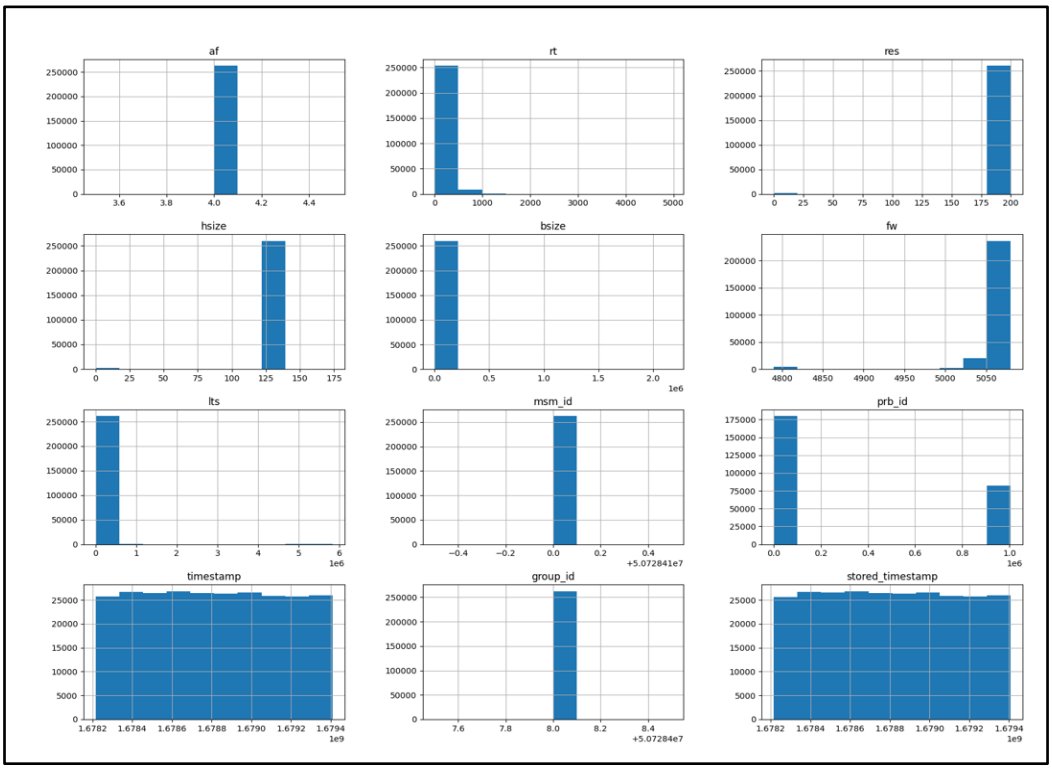


Figure: Histograms of attributes distribution

There is a proof in the histogram for one value columns. Also, “bsize” looks like one of them, but in the next step of analysis there will be shown that is not true. The same is in “lts”, “hsize” and “res” columns.

	af	rt	res	hsize	bsize	fw	lts	msm_id	prb_id	timestamp	group_id	stored_timestamp
count	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000	262671,0000
mean	4,0000	152,0260	198,5160	130,1357	12283,5102	5072,1418	15754,6536	50728410,0000	351253,1353	1678810700,8181	50728408,0000	1678810873,5180
std	0,0000	170,6795	17,1638	11,4551	91837,6933	36,3408	265334,1762	0,0000	442736,8742	342745,6770	0,0000	342724,3863
min	4,0000	0,0000	0,0000	0,0000	0,0000	4790,0000	-1,0000	50728410,0000	620,0000	1678215442,0000	50728408,0000	1678215556,0000
25%	4,0000	48,8446	200,0000	131,0000	4191,0000	5080,0000	7,0000	50728410,0000	60262,0000	1678514238,0000	50728408,0000	1678514343,0000
50%	4,0000	74,3647	200,0000	131,0000	4192,0000	5080,0000	25,0000	50728410,0000	61866,0000	1678807647,0000	50728408,0000	1678807799,0000
75%	4,0000	235,7775	200,0000	131,0000	4193,0000	5080,0000	53,0000	50728410,0000	1004304,0000	1679106440,0000	50728408,0000	1679107076,0000
max	4,0000	4960,3086	200,0000	174,0000	2171974,0000	5080,0000	5829334,0000	50728410,0000	1005475,0000	1679408848,0000	50728408,0000	1679409036,0000

Figure: Basic statistical calculation

The min value of “stored_timestamp” is the same as in the previous dataset. Other values are a little bit different. On the other hand, “timestamp” ranges are exactly the same, but other values are a little bit different. “prb_id” starts at a bigger value, but the mean value is lower, as well as std. The same can be observed in ”fw”. Time to execute requests excluding DNS (“rt”) is much bigger than in the previous table. Here it starts from 0, and ends with approximately 4960,31. The mean value is a little bit higher (152,0260) than previous which was 149,7409. In this case, “res” is a non-one value. It oscillates between 0 and 200, which can be caused by missing values, in this case 0. Based on the mean value, there can be a conclusion that there is a small amount of missing values. Also, “hsize” is not one value any more. Minimum value is 0 and maximum value is 174. Nevertheless, in most cases 131 is the chosen value seen in percentiles, so the same as in the previous table. “lts” value is very similar to the previous one in min and percentile values. Mean value is higher, as well as std.

Scaling data

The next step and the last step of the preprocessing phase was to scale numeric data to provide more readable data into machine learning algorithms. Two scaling techniques were proposed: standardization using the function ‘*StandardScaler*’ from the ‘sklearn’ library; and normalization - ‘*MinMaxScaler*’ from the same library. The purpose is that, later, the model will improve its capability to discover patterns and results that potentially have more impact on the predicted value. In this case, the chosen features for this function were: almost all the numeric data was normalized to exclude columns: *'mver', 'timestamp', 'uri', 'stored_timestamp'*. Only numeric data was scaled. It is up to the user to select which technique wants to use on data.

Predicting data and evaluating results

After the preprocessing phase, it is possible to make predictions on this data. Choosing a model and building it depends on our chosen Target - Predicting value. It can be a regression problem or classification problem. It was established to do (separately) prediction of attributes (targets):

- Target *'timestamp'* and *'lts'* for first consideration relating regression problem
- Target *'err'* attribute for consideration relating to classification problem

The Prediction module has two functions. For the regression problem and for classification problem. These functions build models, train data and then calculate evaluation metrics and create plots. For regression and classification different models, different evaluation metrics and different plots were used. Different attributes acting as predictors were chosen depending on the experiment. These attributes were either

- all attributes
- 5 attributes experiment: either [*'bsize', 'uri', 'from', 'stored_timestamp', 'lts']* or *['bsize','uri','from','prb_id','stored_timestamp']* depending on target.

Machine learning models used in this module were: decision tree, gradient boosting and random forest both on regression and classification problem and linear regression on regression problem. Prediction and evaluation was done in python using supporting libraries. The last experiment was done on preprocessed datasets in the Orange tool. Evaluation metrics for the regression problem were: mean squared error, root mean square error and R2 (r-squared). The evaluation of the model's performance was visually presented using a scatter plot of the predicted values versus the actual values. Evaluation metrics for the classification problem were: accuracy, precision, recall, F1 score. The evaluation of the model's performance was visually presented using both a confusion matrix and an ROC curve. The results are presented and described in the ‘Experiments’ section.

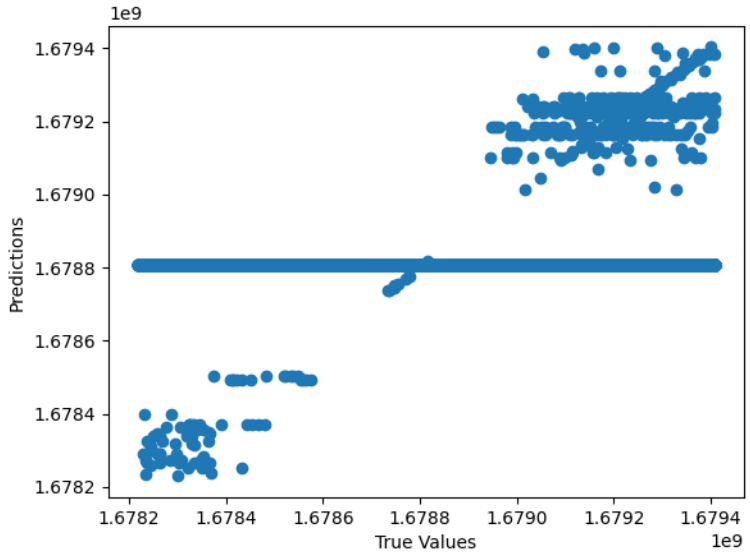
Experiments

Experiments were conducted for different datasets, for different algorithms, for different attributes acting as predictors and for different targets. Overall model performance was very good, for some models it was perfect. Although overfitting of the model is possible, a more probable option is that models were recognizing patterns and relating very well to many dependencies within attributes and were able to classify and make regression almost perfectly. Similar results across different models and on testing data would confirm this option. Preprocessing phase was performed in two ways, therefore experiments were dependent on that fact. Experiments included prediction on two different targets, therefore different preprocessed data, on different machine learning models and on different numbers of attributes. Additionally, prediction was performed on another software tool - Orange.

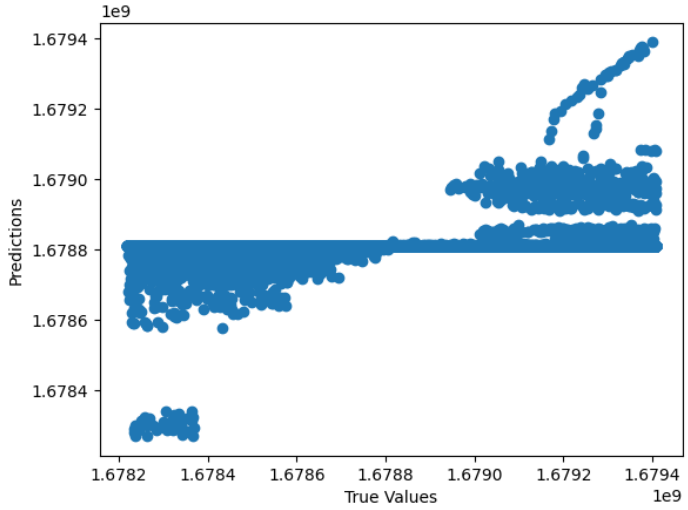
Experiment - 'timestamp'

This experiment involves preprocessing phase, taking first consideration of handling data. Target attribute was 'timestamp'. Regression prediction took place on all attributes and 5 attributes on different models.

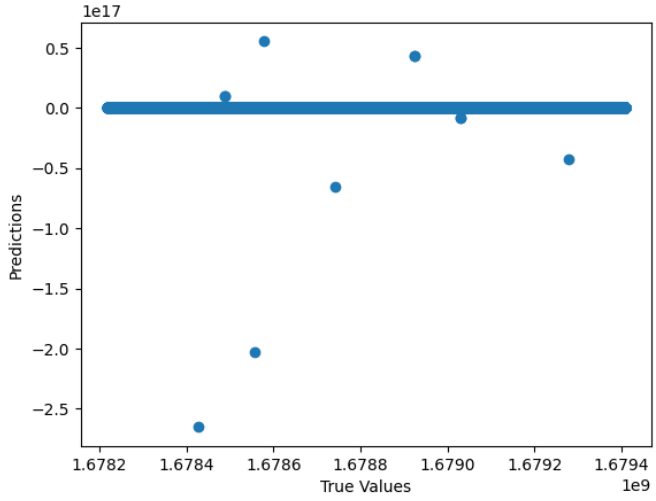
model decision tree regressor - all attributes (*all attributes besides 'stored_timestamp')	MSE: 115703480053.97255 RMSE: 340152.14251 R-squared: 0.01242



model random forest regressor - all attributes	MSE: 115020097950.33893 RMSE: 339146.1306728103 R-squared: 0.018257351897982943



model logistic regression - all attributes	MSE: 2.4664484432625875e+30 RMSE: 1570493057374844.8 R-squared: -2.105212627397676e+19

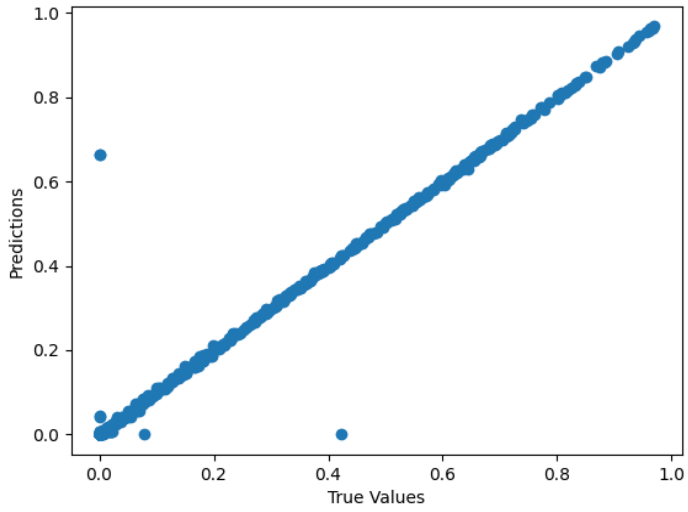


Based on these results, both the decision tree regressor and random forest regressor models have low R-squared values, indicating poor fit to the data. The logistic regression model performs even worse, with extremely high MSE, RMSE, and negative R-squared values. It suggests that these models are not suitable for accurately predicting the target variable based on the given attributes. Further exploration and consideration of alternative models or feature engineering may be necessary. Therefore, it was decided to not do prediction on a smaller number of attributes.

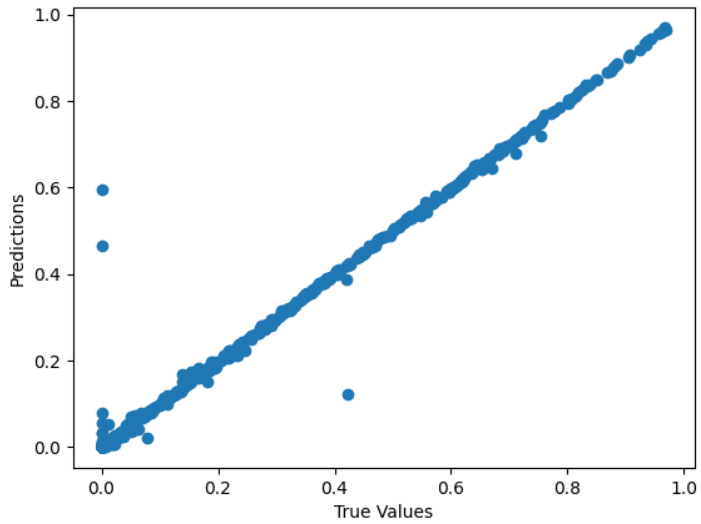
Experiment - ‘lts’

This experiment involves preprocessing phase, taking first consideration of handling data. But having poor results of prediction on timestamp target, it was decided to change to ‘lts’. Regression prediction took place on all attributes and 5 attributes on different models.

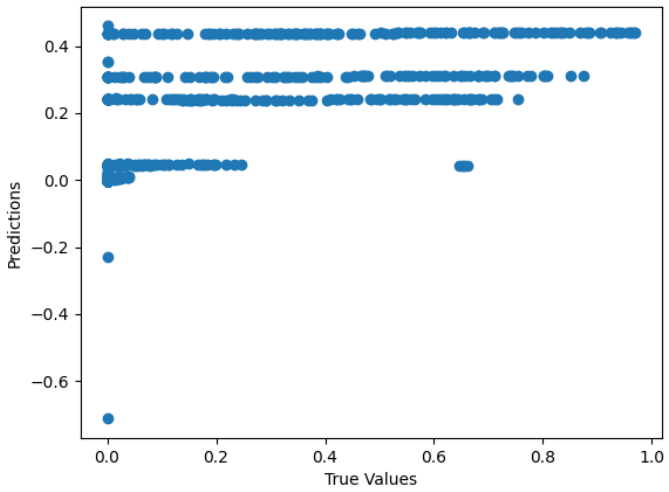
model decision tree regressor - all attributes.	MSE: 2.13648459120945e-05 RMSE: 0.0046 R-squared: 0.9876



model random forest regressor	MSE: 1.3739870877004891e-05 RMSE: 0.0037 R-squared: 0.9921



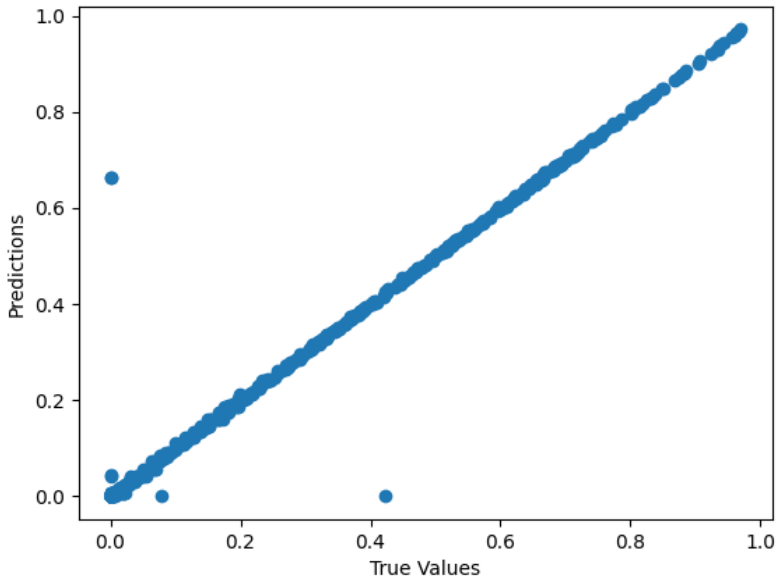
model logistic regression - all attributes	MSE: 0.0007 RMSE: 0.0256 R-squared: 0.6212



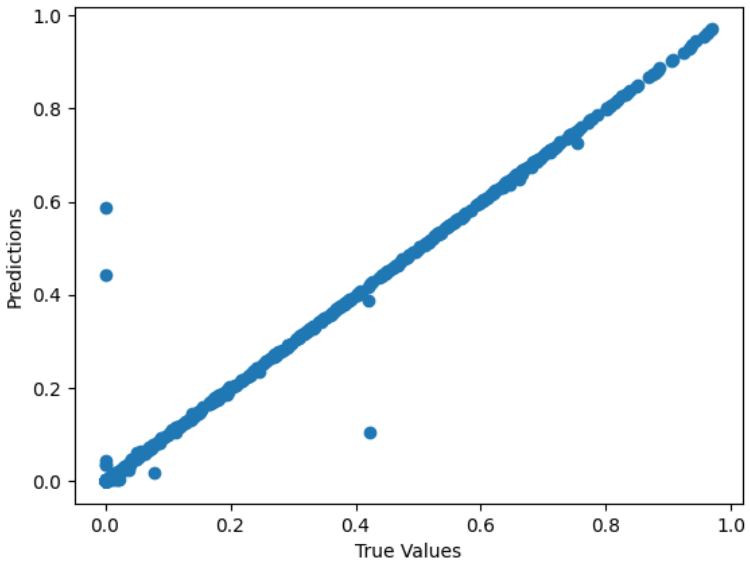
Testing all attributes all results appeared very good indicating small error and high R-squared value. Based on the results, the random forest regressor outperformed both the decision tree regressor and logistic regression models in terms of predictive accuracy. It achieved the lowest MSE and RMSE values, indicating better precision in estimating the target variable. Additionally, the random forest regressor also obtained the highest R-squared value, suggesting a stronger overall fit to the data. Therefore, the random forest model appears to be the most effective in capturing the underlying patterns in the dataset and making accurate predictions.

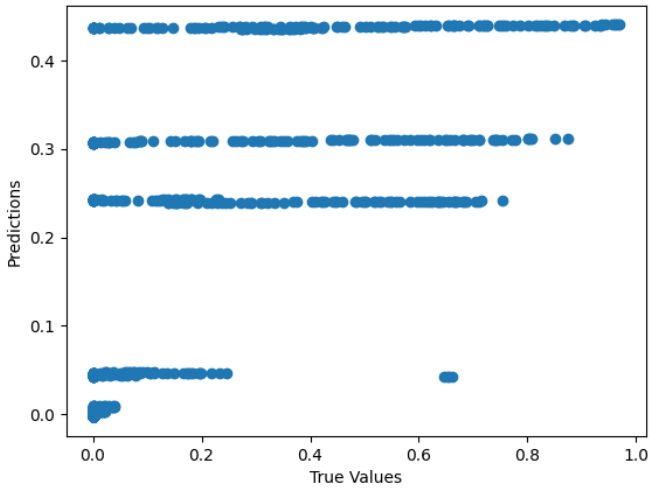
Next data classification was tested for 5 attributes, predictors: 'bsize', 'uri', 'from', 'prb_id', 'stored_timestamp'.

model decision tree regressor	MSE: 2.1344700158859934e-05 RMSE: 0.0046 R-squared: 0.9877



model random forest regressor - regression	MSE: 1.295780696668795e-05 RMSE: 0.0036 R-squared: 0.9925



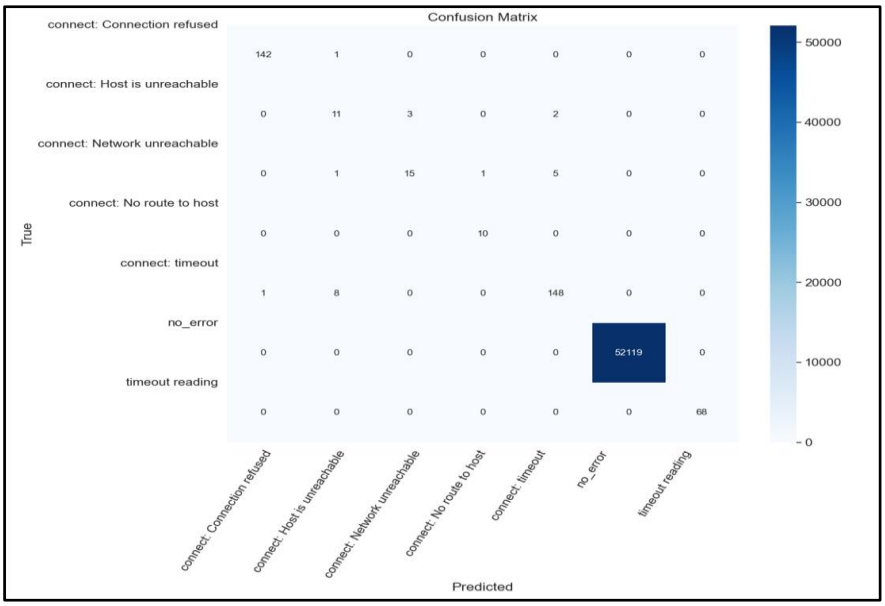
model logistic regression - regression	MSE: 0.0006 RMSE: 0.0252 R-squared: 0.6329
	

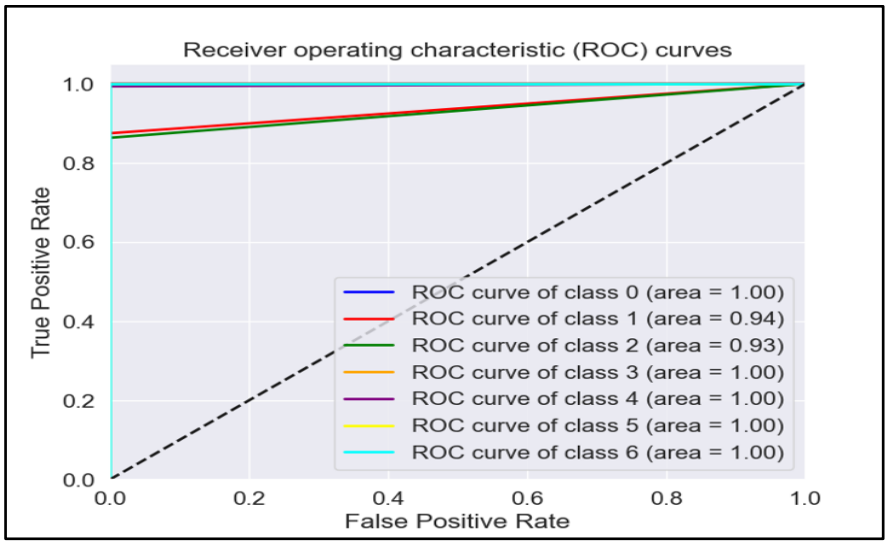
When using only five attributes, the random forest regressor still outperformed the decision tree regressor and logistic regression models. It achieved the lowest MSE and RMSE values, indicating better precision in estimating the target variable. The random forest regressor also obtained the highest R-squared value, indicating a stronger overall fit to the data. Therefore, even with a reduced number of attributes, the random forest model remains the most effective in capturing the underlying patterns and making accurate predictions.

Experiment - Target ‘err’

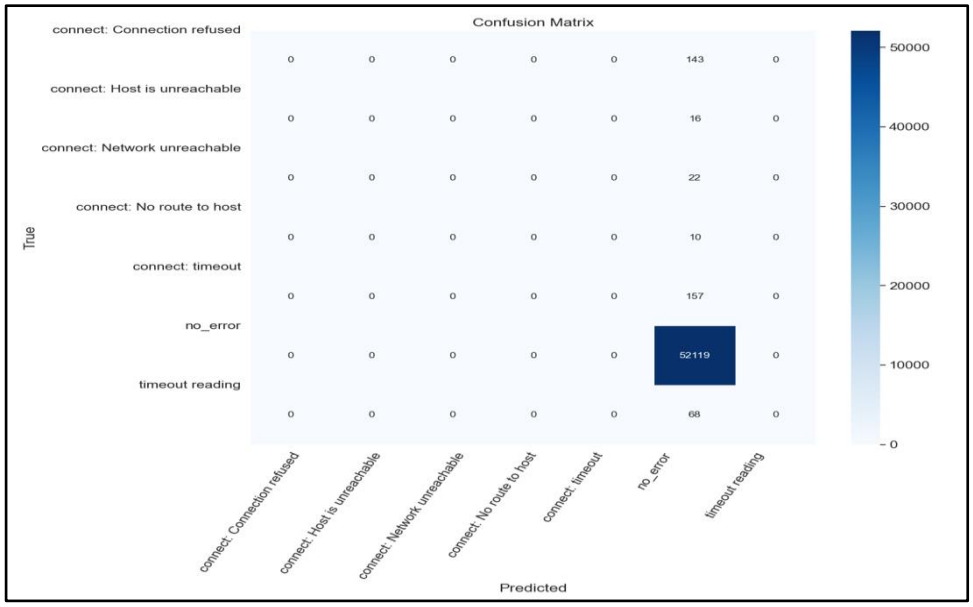
This experiment involves preprocessing phase taking second consideration of handling data. Experiment with all data, but missing data replace ‘*NaN*’ or ‘0’ and prediction for ‘err’ classification problem. Data was scaled. Target ‘err’ column gives information about occurring errors or no errors at all. Seven classes were considered: 'connect: Connection refused', 'connect: Host is unreachable', 'connect: Network unreachable', 'connect: No route to host', 'connect: timeout', 'no_error', 'timeout reading'.

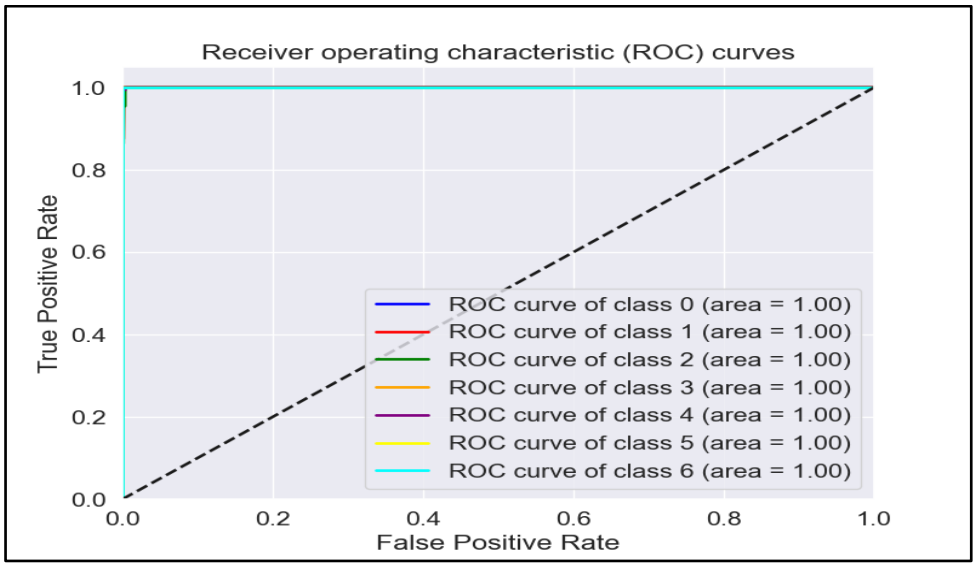
model - decision tree classifier - all attributes	Accuracy: 0.99958 Precision: 0.99961 Recall: 0.99958 F1 Score: 0.99959





model - RandomForestClassifier- all attributes	Accuracy: 0.99208 Precision: 0.98423 Recall: 0.99208 F1 Score: 0.98814

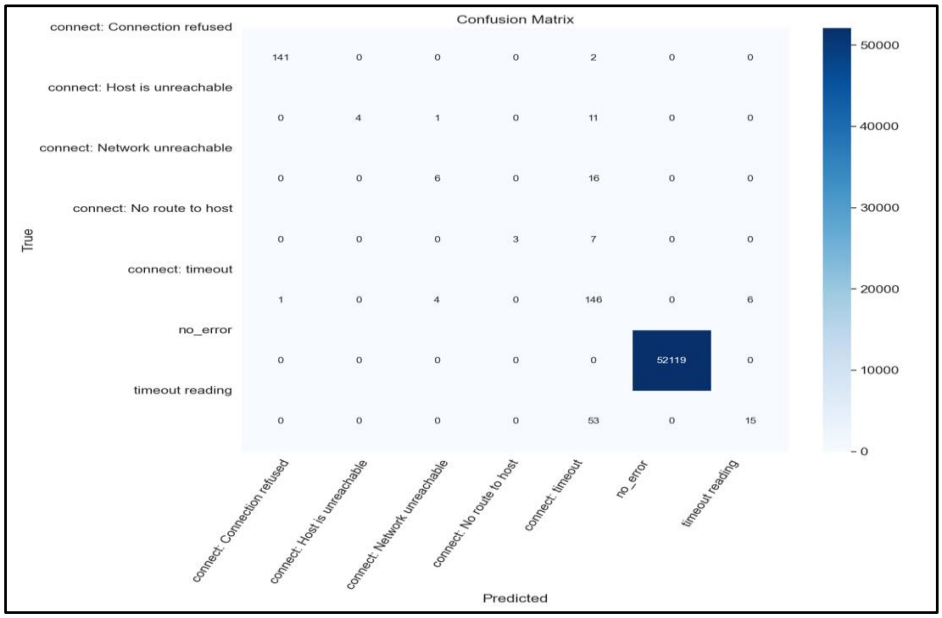


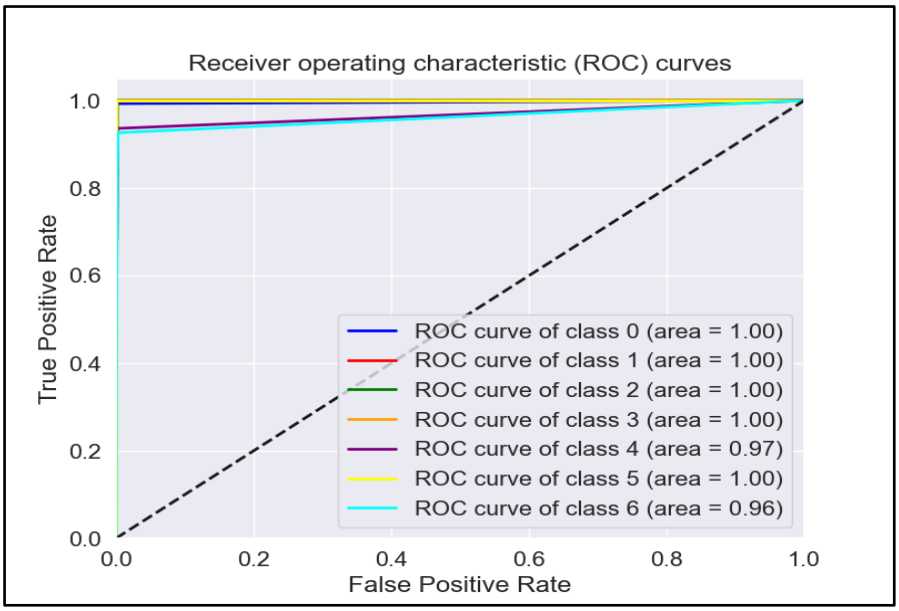


Results were very good for prediction of 'lts' on the test set. Models predicted almost 100% of data correctly. Its almost perfect accuracy and precision can be due to imbalanced data, because almost all data belongs to the class - no error. Second thing can be that imputed values can bias prediction. Nevertheless, it is highly probable that the classifiers found patterns, etc. Further experiments in this matter can potentially expand insights. In the case of a model for random forest classifier, results were perfect, that is 1. Heatmap and ROC curve were plotted.

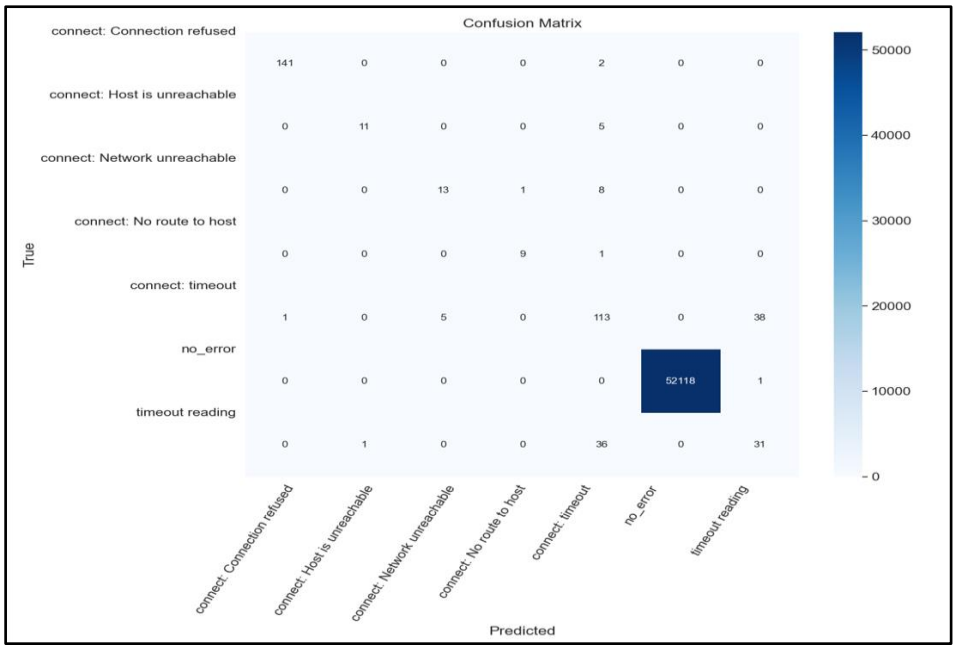
Next data classification was tested for 5 attributes, predictors: 'bsize', 'uri', 'from', 'stored_timestamp', 'lts'].

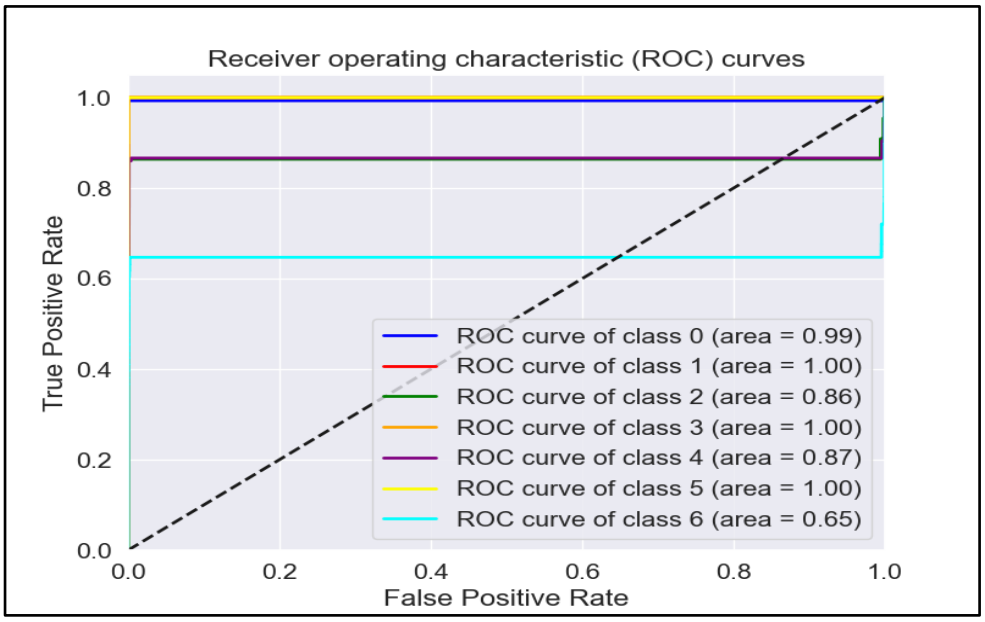
decision tree classifier - smaller number of attributes- 5 attributes	Results: Accuracy: 0.99808 Precision: 0.99829 Recall: 0.99808 F1 Score: 0.99780



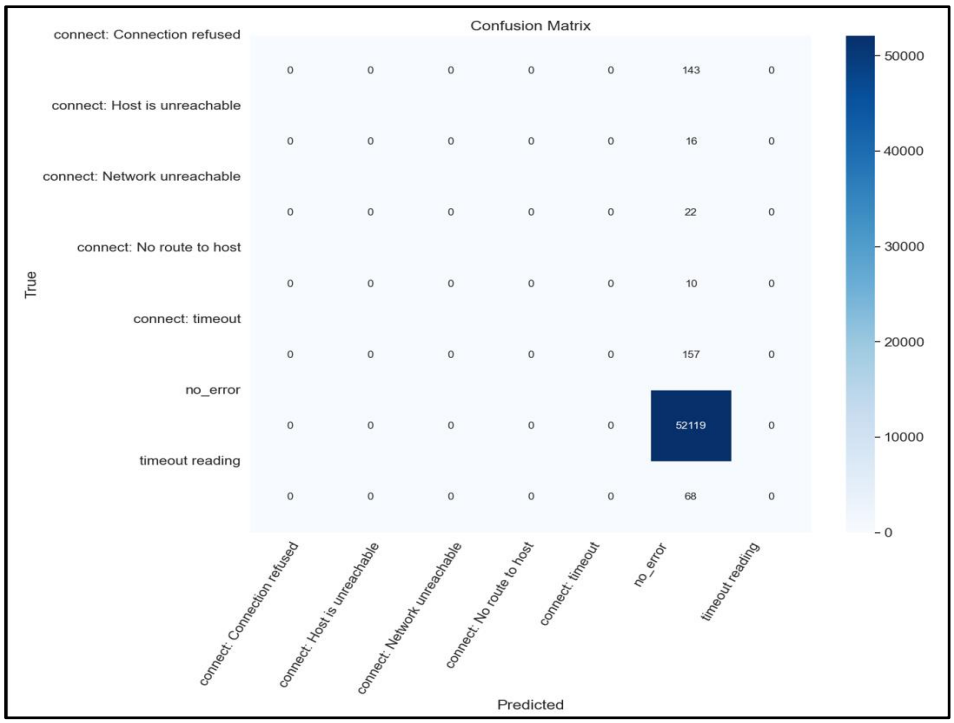


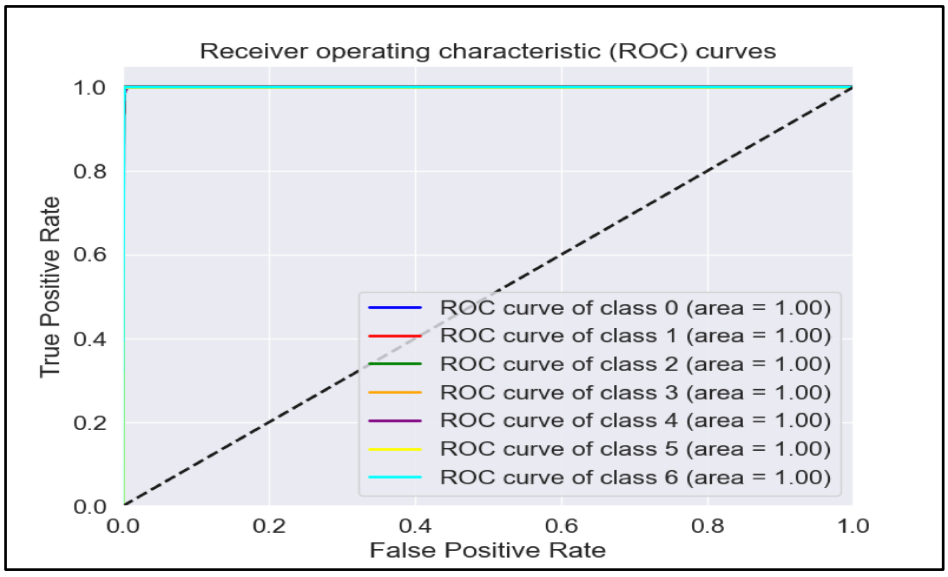
GradientBoostingClassifier - smaller number of attributes- 5 attributes	Accuracy: 0.99812 Precision: 0.99816 Recall: 0.99812 F1 Score: 0.99813





RandomForestClassifier- smaller number of attributes- 5 attributes	Accuracy: 0.99208 Precision: 0.98423 Recall: 0.99208 F1 Score: 0.98814





The same pattern of results can be observed in prediction for smaller numbers of attributes. Almost perfect results were obtained and in case of random forest, perfect that is accuracy equal to 1. Similar potential bias can be partially reasonable for overfitting.

Experiment - Orange tool

Experiment was also conducted in another software - Orange tool. Models built were linear regression, gradient boosting and random forest.

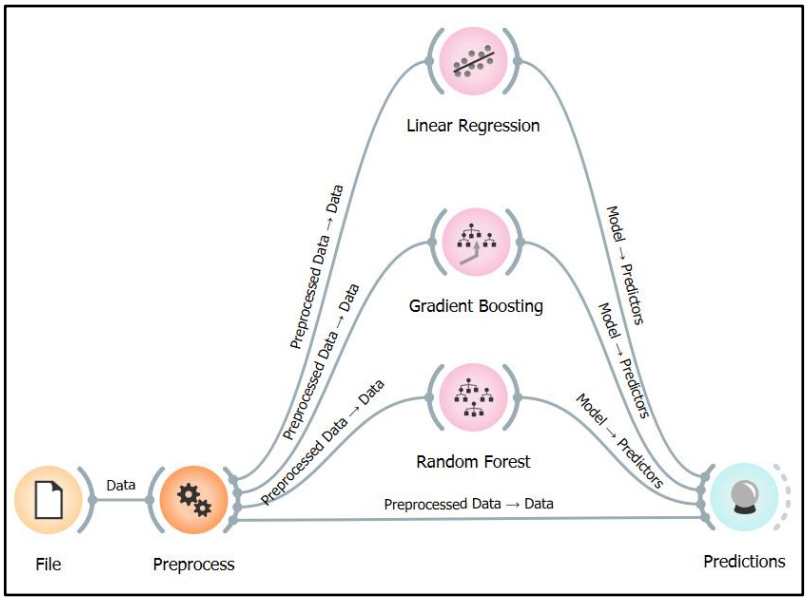


Figure: Prediction and preprocessing pipeline in Orange tool

Created pipeline in the Orange tool was saved and is visible in figure above. Experiment was conducted on one target - 'lts' and one dataset without cleaning. Also, to conduct prediction different models were used: Gradient Boosting, Random Forest and Linear Regression.

Model	MSE	RMSE	MAE	R2
Linear Regression	69432216166.036	263499.936	35336.435	0.014
Gradient Boosting	703226036.969	26518.409	2754.781	0.990
Random Forest	103241864.017	10160.800	151.427	0.999

Figure: Results of classification for target 'lts' in Orange tool

As it is seen above, the Linear Regression guessed only a few examples. This may be caused by not cleaning the data in this experiment. Other models were almost perfect. In this experiment, linear regression was the fastest model, but based on guesses it made, it is the worst predictor for this dataset. Gradient boosting was the slowest, but R2 score is almost ideal. The random forest did the most impressive job with better time as gradient boosting. The predictions were 99.9%.

Conclusions

In conclusion, the project focused on the modeling and analysis of web-based systems using network log data obtained from RIPE Atlas measurements. The Python-based program followed a well-defined pipeline and program structure to handle the data effectively. The data preprocessing steps included loading the JSON file, cleaning the data by handling duplicates, missing values, and outliers, and scaling the data for analysis. The analysis phase involved exploring the remaining data, visualizing it through heatmaps, histograms, and boxplots, and performing basic statistical calculations. The prediction phase aimed at building models and evaluating them. The program provided insights into the correlation between attributes, distribution of data, and outliers. The analysis allowed for a better understanding of the dataset and facilitated prediction tasks. Overall, the project demonstrated the importance of proper data preprocessing and analysis techniques in modeling and analyzing web-based systems. The prediction models developed for response time demonstrated high accuracy, suggesting that the selected features provided valuable information for predicting response time. Although the prediction models for ‘timestamp’ attribute didn't perform as well, they still showed some degree of predictability.