CMP7203 – Big Data Management

Big data use in Catch the pink flamingo

Coursework technical report

Zan Zver
18133498

BIRMINGHAM CITY
University

Faculty of Computing, Engineering and the Built Environment
Birmingham City University

Contents

List of Tables

List of Figures

1 Vs in big data

Vs in big data represent challenges we are going to encounter. There are 3 main Vs (volume, variety and velocity) that are seen on the image bellow (Sagiroglu and Sinanc 2013). Over the years, new challenges have occurred with data therefore new Vs have been added (Anuradha et al. 2015).

Described bellow are main 3 Vs with 2 additional ones.

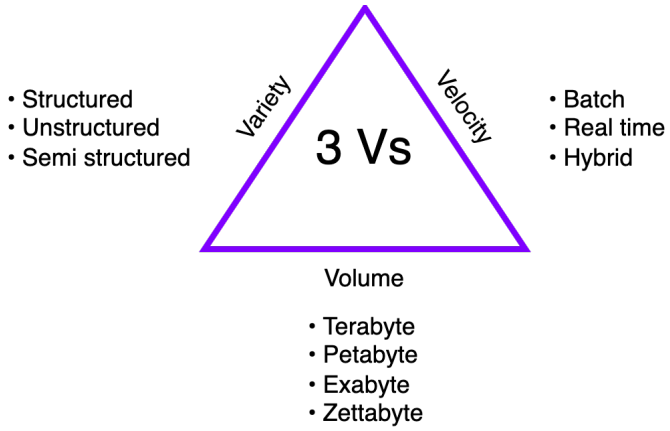

Figure 1: Vs

1.1 Volume

Volume is a representation of data coming into the computer (server). It is measured in computer storage capacity (Katal, Wazid, and Goudar 2013).

In our case, we have data at rest (it will not change). The total storage equals to:

file name	size
chat_join_team_chat.csv	82 KB
chat_leave_team_chat.csv	67 KB
chat_mention_team_chat.csv	238 KB
chat_respond_team_chat.csv	252 KB
combined-data.csv	162 KB
ad-clicks.csv	826 KB
buy-clicks.csv	135 KB
game-clicks.csv	33.8 MB
level-events.csv	43 KB
team-assignments.csv	331 KB
team.csv	8 KB
user-session.csv	492 KB
users.csv	137 KB
Total file size	36.436 MB

Table 1: Files and Sizes

In total we are dealing with 36.436 MB (or 36436 KB). This can work as a representation of big data, but in reality we would be dealing with more sophisticated files (bigger file sizes, more file types, more files, etc...)

1.2 Variety

Variety describes the kind of data we are dealing with. This can be from text, to video, sound, etc... Different files carry different burdens, for example file size. Video platform (such as YouTube) is distributing mainly video formats (Hota et al. 2018), therefore they need a different infrastructure compared to text based website (such as Stack overflow).

As seen in table above (1), we are mainly dealing with CSV files. This is represented in semi structured form since we can link items together. In the processing stage, files could be change to different format in order to save space (example: Parquett with Apache Hudi).

1.3 Velocity

Velocity represents frequency of data ingestion. How often we receive data depends on the system we are talking about (Cappa et al. 2021). It can be once a second

(video stream), every hour (record update), every 6 moths, etc... Frequency is important from computational point of view. If we only expect data every week, data pipeline will need to be executed one a week, otherwise if data is received every second, our pipeline needs to be running all the time (24/7).

In our case, we have already received data beforehand. That means we have data at rest, we are not expecting it to change. In case we would have access to live data (with an API for example), our records would be changing based on each run.

1.4 Veracity

Veracity represents how data is accurate, reliable and certain. To accomplish this, we can use different approaches (Rubin and Lukoianova 2013). If data is stored at one storage medium (SSD/HDD/USB) and that gets damaged, we could experience data loss. The best practise is to use 3-2-1 approach (Storage 2012):

- 3 copies of data
- 2 different media
- 1 copy being off-site

This can be achieved by using one of the cloud providers (AWS, Azure, Google Cloud, etc...) in order to save data across the word.

To insure quality, we would need to restrain data either at input level or at processing. If we have complete access to the data based on its life cycle, we can impose rules at the beginning of insertion (limit age, countries available, etc...) (Yu and Wen 2010). Otherwise if we don't have access to data from the start, we can apply rules in the data processing stage.

In our case, we don't need to worry about the data loss since data is hosted on GitHub (Zver n.d.). By using cloud service to host our data we don't need to worry about 3-2-1 since they are taking care of it (C. Wang et al. 2010). There are also historical versions of our files available which is beneficial. Disadvantage is that we don't control the infrastructure, meaning if GitHub (owned by Microsoft) would disappear overnight, all of the data is gone.

In order to assure data quality, we are doing data transformations in the processing stage. One example is date time. In some of the files it is saved as YYYY-mm-DD HH:MM:SS while in others it is in Unix time format. Due to benefits of Unix time (deeper dive at 9.1), all the time was converted to it. If we would have control from the beginning, small bugs like this could have been fixed at source.

1.5 Value

Value provides business insights of the data. This can be interpreted in a number of ways, such as most of the players are from US therefore lets focus our infrastructure

there, a lot of players seem to click on technology related ads lets expand that, etc....

Data can provide different value to the company. It all depends on context and analysts extracting it.

In our case, the biggest value it provides is players. With it we can see where they are from, age, platform, etc... Chat data is available as well but it doesn't provide as much details for us but for players it is a lot more important since they can communicate between each other. Value can also be added from external sources. For example, we have Twitter handles in our data. We could expand our dataset by doing text mining on each users Twitter profile in order to server them personalised advertisements. This would increase our data value and financial gain as a company.

2 Big data processing paradigms

There are different processing paradigms that were evolved over time. It all started with Batch, then Real-time was invented and most recently Hybrid. Bellow are all of them explained with two new approaches added.

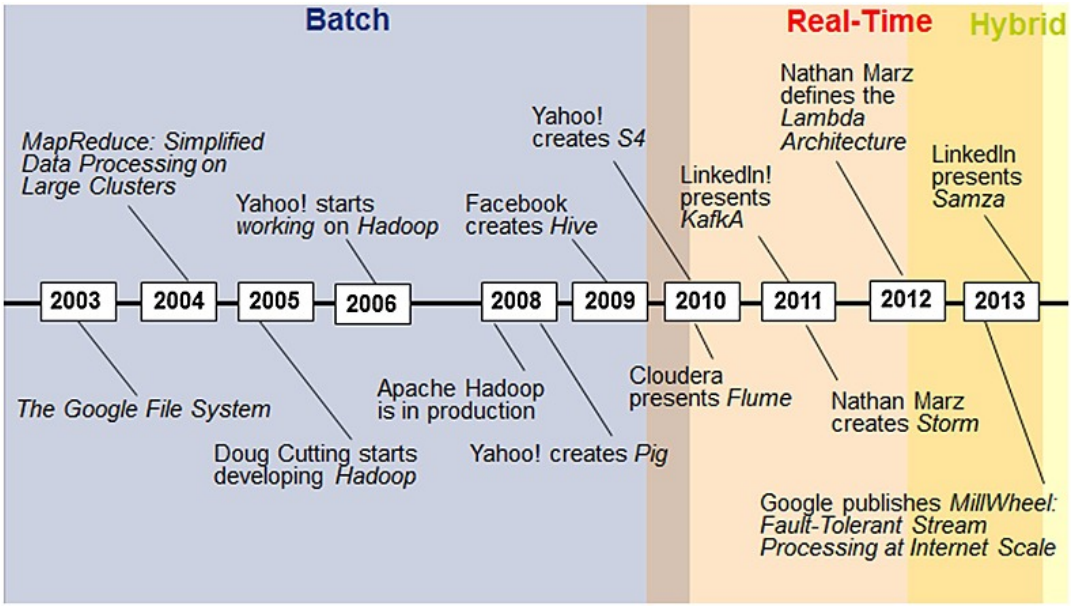

Figure 2: Paradigms over time (Casado and Younas 2015)

2.1 Batch

Batch processing is one of the earliest processes. As name suggests, in batch processing we are expecting multiple files (batch) (Amazon n.d.). Once they are received, they are being processed.

One real world example is roller coaster. In order to operate the ride in optimal capacity, we need 20 (an example number) people on it. The idea is for operator to fill the ride with 20 people. After the ride is complete, the next 20 are selected.

Similar process is in batch processing. Computer (server) waits for n number of files. After they are received, they are inserted into data pipeline in order to start the operation. Limits can be set if files are not received (but this could then be considered hybrid processing).

In our case, we are using batch processing with flamingo data. Files are downloaded from the web and once they are ready pipeline is triggered.

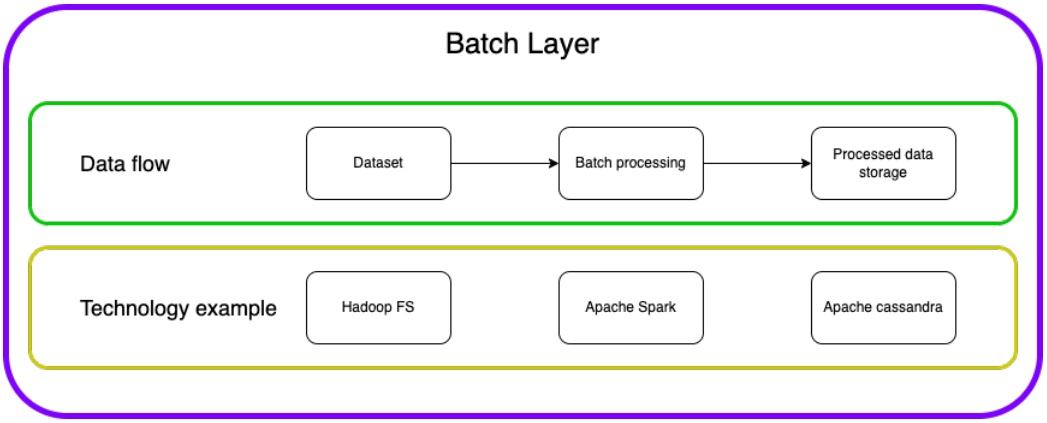

Figure 3: Batch

2.2 Real time

Real time (or stream) processing is used when there are multiple sources feeding data all the time (Wu et al. 2020).

Real world example could be water slide. There is a queue of people waiting for the ride. Operator approves each person to take a ride one after the other.

Similarly, in stream processing, computer (server) is always waiting and ready for the files to be uploaded. We can get 1 or n number of files per second, all of them will be send to our pipeline. For example, IoT sensors. They are constantly sending the data in order to extract relevant information.

In our case, if game would be played online we would use real time operation. This is due to player movements. The rough concept would be for player to make a move, that is recorded in database and with API we would constantly getting new, updated information.

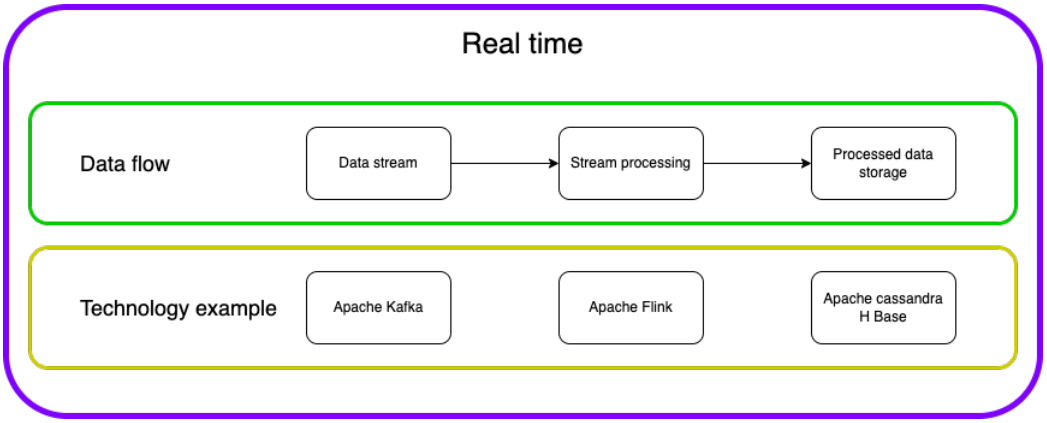

Figure 4: Real time

2.3 Hybrid

Hybrid processing is a mix of batch and real time (Lewis, Zamith, and Hermida 2013).

Lets revisit roller coaster example. In batch scenario operator was waiting for 20 people to be seated. In hybrid scenario, we are expecting 20 people to show up. If there isn't enough people, operator will just start the ride with less than 20 people. One scenario would be business hours. In the morning and evening, we don't get as many people therefore operator is running the ride in hybrid mode but thought the noon (when there are core hours) operation seems like batch (but is hybrid on a grand scale).

Now lets consider our file ingestion from batch processing. We are still expecting n number of files. If we get n number of files, we start the processing with them (batch). Otherwise if we don't get all the files, we work with the files that were received as stream.

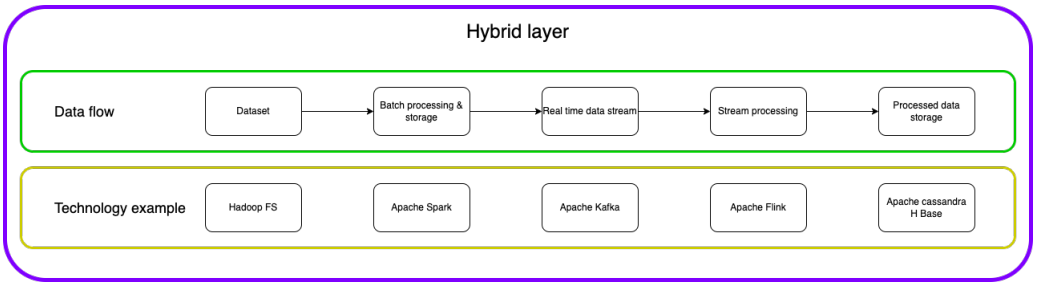

Figure 5: Hybrid layer

2.4 Other

As technology moves on, new problems and solutions are developed. Lambda and Kappa architectures are not as popular but still interesting approaches of data pipelines.

Further discussion on Lambda and Kappa can be found under 9.8 and 9.9.

3 Exploratory data analysis

Exploratory data analysis (EDA) is a process in which we familiarise ourselves with data (Tukey et al. 1977). Listed in the tree structure bellow are the files originally provided to us. Listed bellow can be seen split sections, based on the nested paths (chat, combined and flamingo data).

```
DataSet
├── dataset_attribute_desc.doc
│   └── description of the dataset
├── combined-data.csv
├── chat-data
│   ├── chat_join_team_chat.csv
│   │   └── When a user joins a team, a new record is going to be added
│   │       to this file
│   ├── chat_leave_team_chat.csv
│   │   └── When a user leaves a team, a new record is going to be added
│   │       to this file
│   ├── chat_mention_team_chat.csv
│   │   └── When a user gets a mention, a new record is going to be added
│   │       to this file
│   └── chat_respond_team_chat.csv
│       └── When a player with chatid2 responds to a post by another player
│           with chatid2, a new line is added in this file
└── flamingo-data
    ├── ad-clicks.csv
    │   └── Database of clicks on ads
    ├── buy-clicks.csv
    │   └── Database of purchases
    ├── game-clicks.csv
    │   └── A record of each click a user performed during the game
    ├── level-events.csv
    │   └── A record of each level event for a teams,lLevel events are
    │       recorded when a team ends or begins a new level
    ├── team-assignments.csv
    │   └── A record of each time a user joins a team
    ├── team.csv
    │   └── A record of each team in the game
    ├── user-session.csv
    │   └── A record of each session a user plays
    └── users.csv
        └── Database of the game users
```

Originally the following items were zipped:

- chat-data.zip

- combined-data.zip
- flamingo-data.zip

3.1 Combined data

This is a general representation of data. It combines different items from flamingo data together into one.

Attribute	Description
userId	id of the user
userSessionId	id of the session
teamLevel	level team is at
platformType	what platform user is on
count_gameclicks	number of game clicks
count_hits	number of hits
count_buyId	id of buy
avg_price	average buy price

Table 2: combined-data.csv

Discovering missing data showcases two things; missing data in count_buyId and avg_price and correlation between them. It seems like attributes are coupled together since user who buys a product gets buy ID and average price of it. Based on that logic, if item is not bought there is no ID and no price. Situation like this could be avoided by having product with id 0 and price 0 inside. This would help us expose actual missing values since at the moment we cannot confirm if missing values are intentional or by mistake.

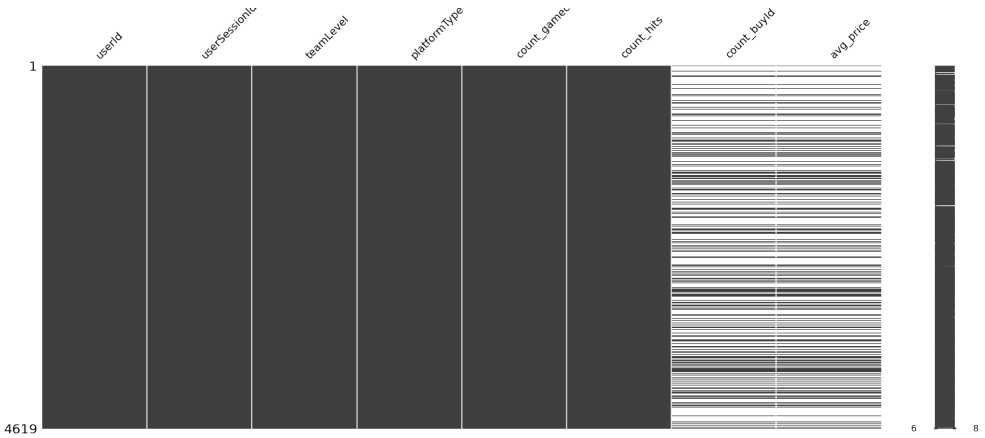

Figure 6: combinedData_msno

Knowing our audience is key thing, therefore we need to know what platform is the most used for the game. Pie chart bellow showcases that mobile platform (mainly iphone) is what majority of our players use.

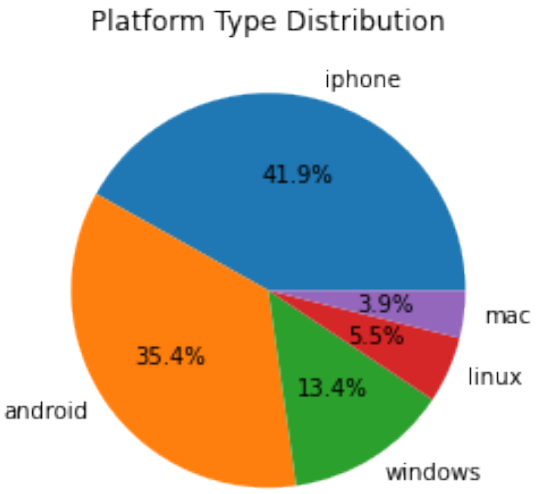

Figure 7: combinedData_correlationPlot

To understand skills, we can compare platforms between each other. Although iphone has the most game clicks (due to being the most popular) and the most hits, android seems to be fairly close to iphone. That could suggest that android players are getting more hits either due to skill or due to platform advantage.

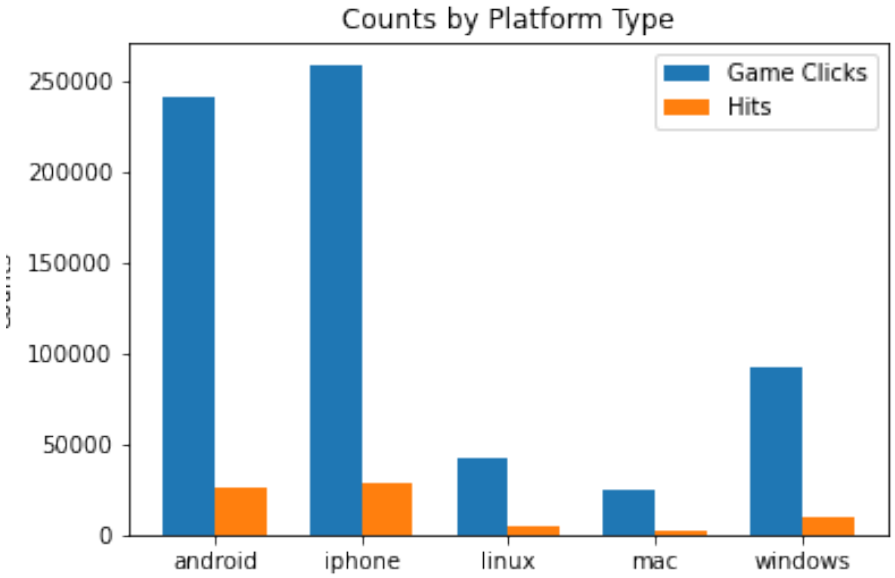

Figure 8: combinedData_multiGrap

Spending habits can tell us a lot about the user. By averaging price spent per platform we can see that iphone users spend the most, but what is surprising is mac users. They are second biggest spenders despite being the smallest platform (only 3.9%). This could lead us to promote more expensive things to mac users.

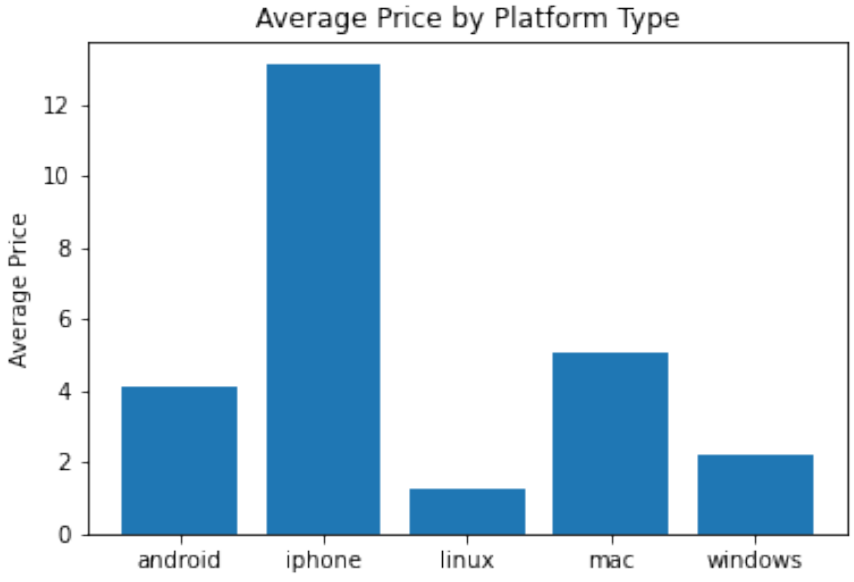

Figure 9: combinedData_priceHistogram

3.2 Chat data

Chat data is a representation between messages send by the users. The EDA for this section wouldn't make much sense, therefore it was not completed.

Reason why it doesn't make sense are attributes. Listed bellow are 6 attributes. This is all we have from the 4 files.

- userId
- teamchat_session_id
- date
- chat_item
- chatid1
- chatid1

Out of 6 attributes, 5 of them are IDs and one of them is date. Time series analysis could be done for when messages were send, but that would be it. Since majority of items are IDs, this wouldn't add much of the business value. Regardless, we can see documented attributes for each of the files bellow.

Attribute	Description
userId	ID of the user
teamchat_session_id	ID of the session that team is in
date	time of the operation

Table 3: chat_join_team_chat.csv

Attribute	Description
userId	ID of the user
teamchat_session_id	ID of the session that team is in
date	time of the operation

Table 4: chat_leave_team_chat.csv

Attribute	Description
chat_item	ID of the message that was send

user_id	ID of the user (that send the message)
date	time when message was send

Table 5: chat_mention_team_chat.csv

Attribute	Description
chatid1	ID of the message that was send
chatid2	ID of the message that got responded
date	time when chatid2 was send

Table 6: chat_respond_team_chat.csv

3.3 Flamingo data

Attribute	Description
timestamp	when ad was clicked
txId	ID of the click
userSessionId	ID of the users session who made the click
teamId	ID of the team that user is in
userId	ID of the user that made the click
adId	ID of an add that was clicked
adCategory	type of an ad that was clicked

Table 7: ad-clicks.csv

Inspecting for missing data in adClicks showcases no missing values.

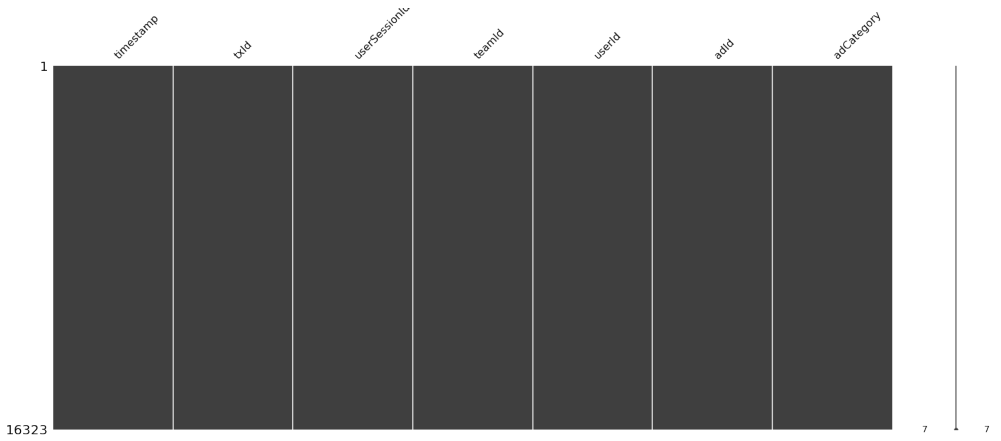

Figure 10: adClicks_msno

Figure (ID) represents decline of ad clicks over time. From business point of view, decline looks steep and fast, therefore something must have happened with the game.

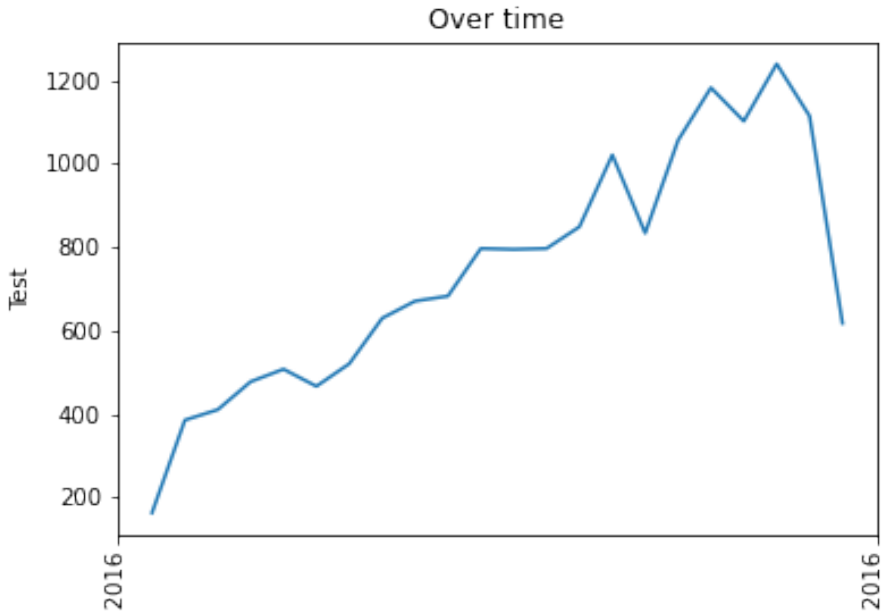

Figure 11: timeseries_adClicks

Looking at the biggest teams, we can see that team with id 64 has occurred 681 times. With knowing what top teams are doing, we can assume other teams will copy them.

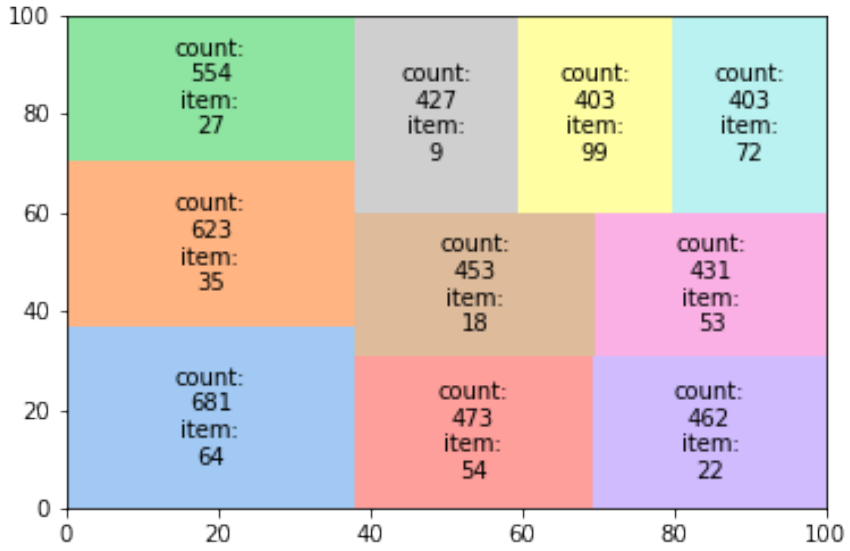

Figure 12: adClicks_tree_map

Comparing IDs of ads to the sections they are representing, we can see that computers and games have the most IDs. This would make sense since the product we are offering is computer game. Other products have on average 3 IDs meaning they aren't so popular.

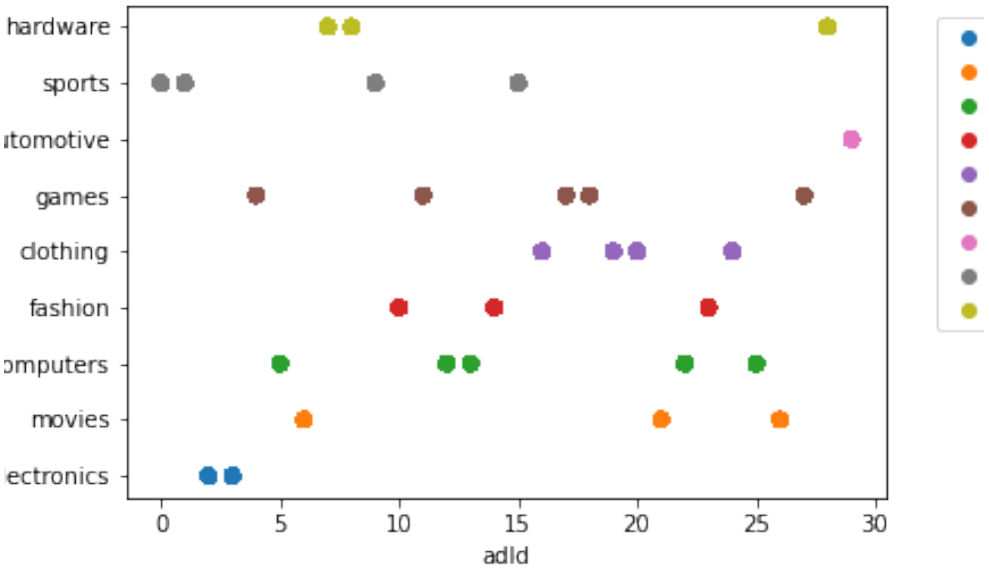

Figure 13: adClicks_create_ad

Attribute	Description
timestamp	when the purcahse was made
txId	ID of the purchase
userSessionId	ID of the user who made the purchase
team	is ID of the team that user is in
userId	is ID of the user that made the purchase
buyId	ID of purchased item
price	price of prucased item

Table 8: buy-clicks.csv

Dataset buy-clicks appears to not have any missing data.

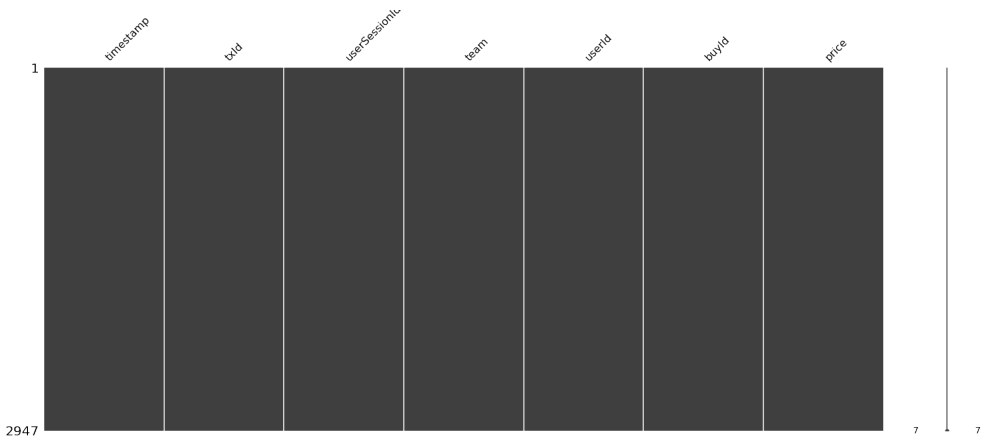

Figure 14: missingno_buyClicks

Looking over the time series, graph is similar to ad-clicks in terms of decline.

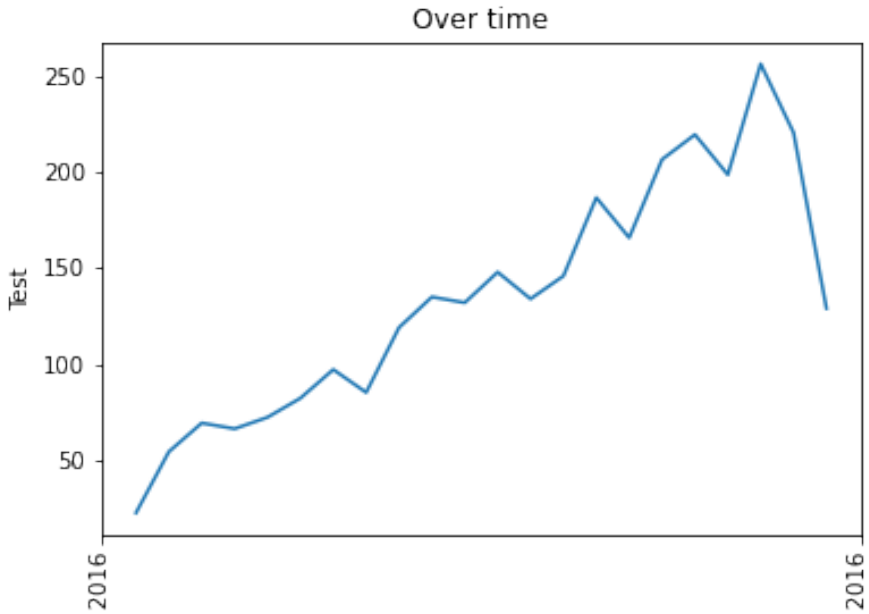

Figure 15: timeseries_buyClicks

Investigating which team has bought the most items, we can see that team with id 27 is the top spender although it is only 3rd one in terms of team members. The second biggest spender, team 64, has the most members. This would put team 27 in odd position since they appear to be the perfect team (quite big and they spend the most).

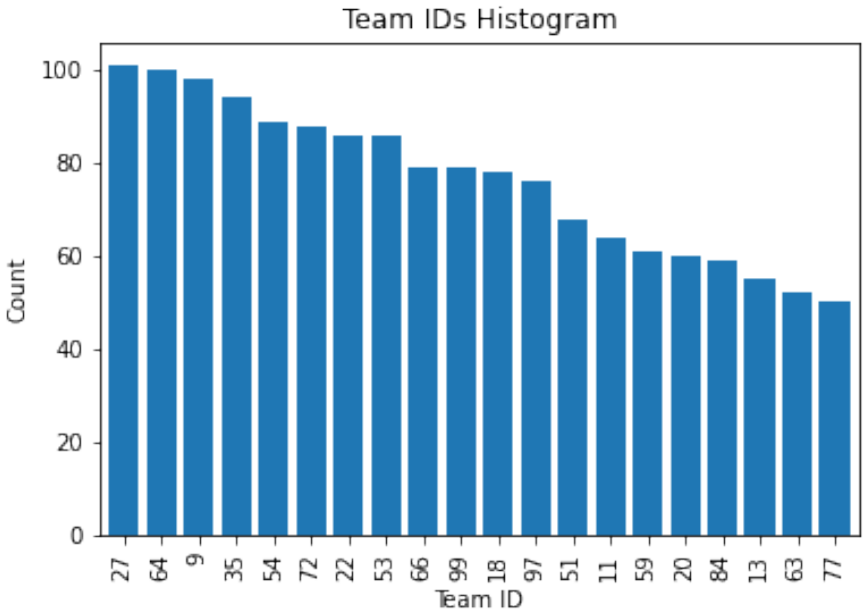

Figure 16: histogram_buyClicks

Creating a correlation chart between team, price and buy ID shows us that team is not correlated to anything. On the other hand, buy id and price seems to be heavily correlated which we have confirmed with missing data in Combined data section (Figure 6).

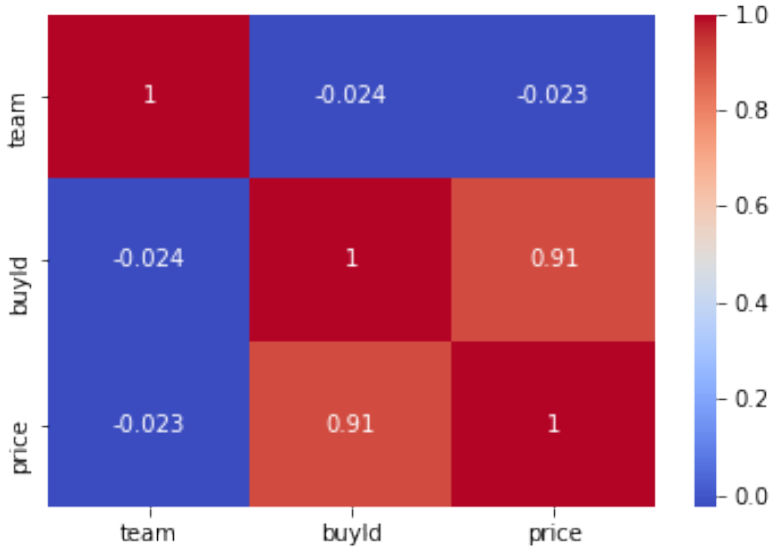

Figure 17: correlationPlot_buyClicks

Attribute	Description
timestamp	when click occurred
clickId	ID of the click
userId	ID of the user who clicked
userSessionId	ID of session that user was in
isHit	if flamingo was hit (1) or not (2)
teamId	ID of the team that user is in
teamLevel	level that team is in

Table 9: game-clicks.csv

Game clicks doesn't appear to have any missing values.

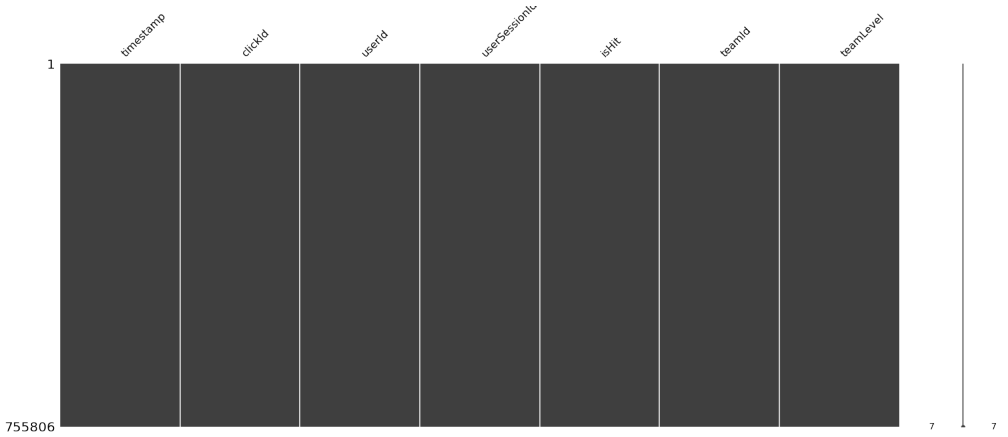

Figure 18: missingno_gameClicks

Time series for game clicks again similar to previous graphs. The only difference is the line is much steeper.

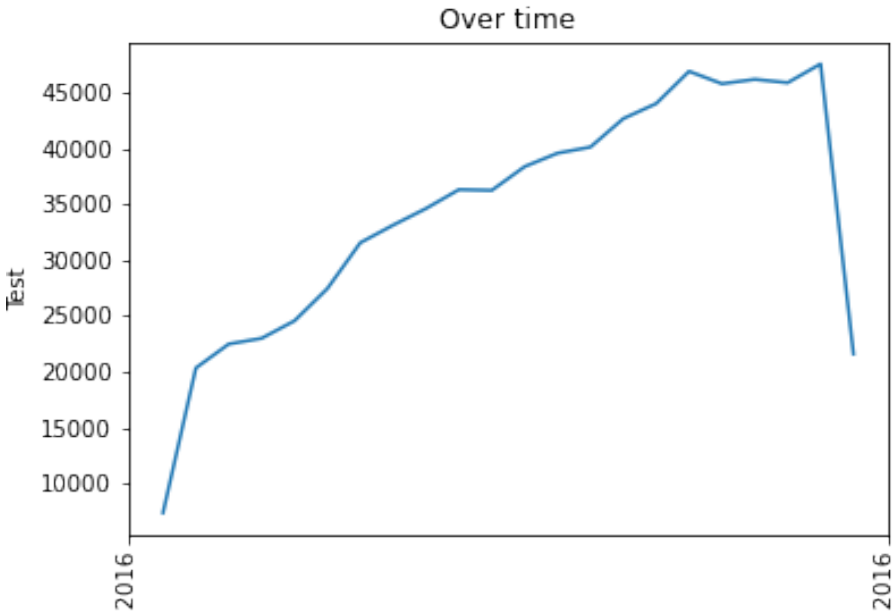

Figure 19: timeseries_gameClicks

Attribute	Description
timestamp	when the event occured
eventId	ID of the event
teamId	ID of the team
teamLevel	level that team has stared or completed
eventType	type of the event (start or end)

Table 10: level-events.csv

Level events appears not to have any missing values.

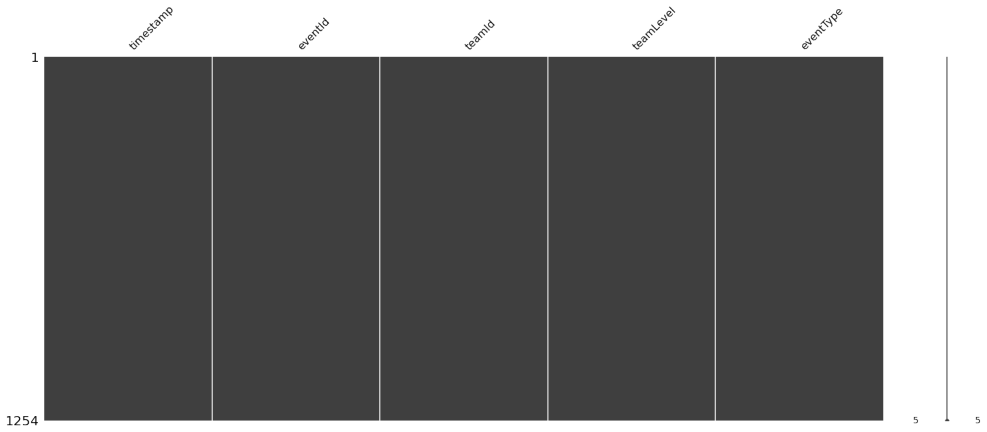

Figure 20: missingno_levelEvents

The time series for level events is unusual. There appears to be 7 spikes, which could indicate when players went up the levels and then it finishing. Other possibility is game events occurring every so often. The graph itself does look different, but the ending seems to be the same as with other graphs. Analysing the graph left to right, we can see how it goes up and down. But at the end, it goes down from the spike, when we would expect it to go back up, it went down again. This seems to confirm that something odd has happened with time series so far.

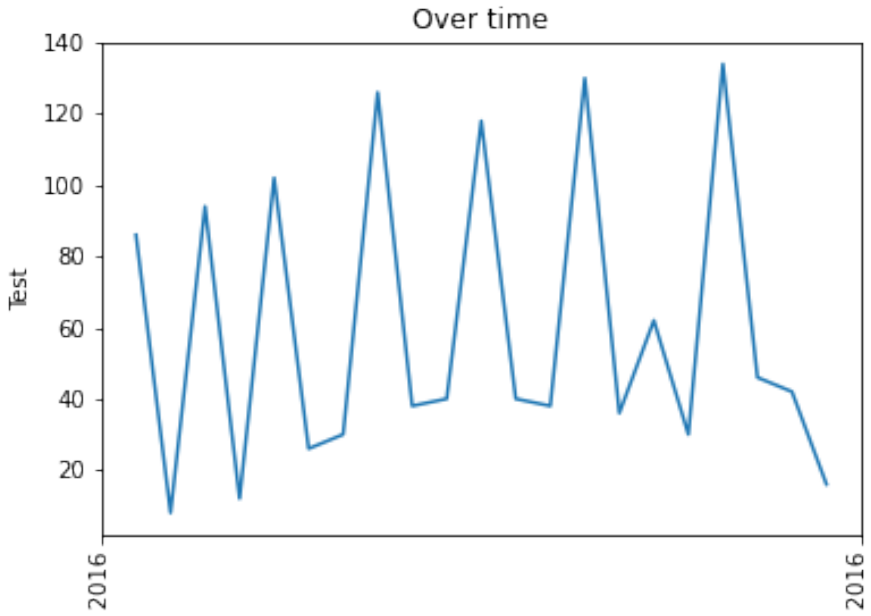

Figure 21: timeseries_levelEvents

Attribute	Description
timestamp	when the user joined the team
team	ID of the team
userId	ID of the user
assignmentId	Temp ID assigned to the user (while in the team/session)

Table 11: team-assignments.csv

Team assignments dataframe has no missing data.

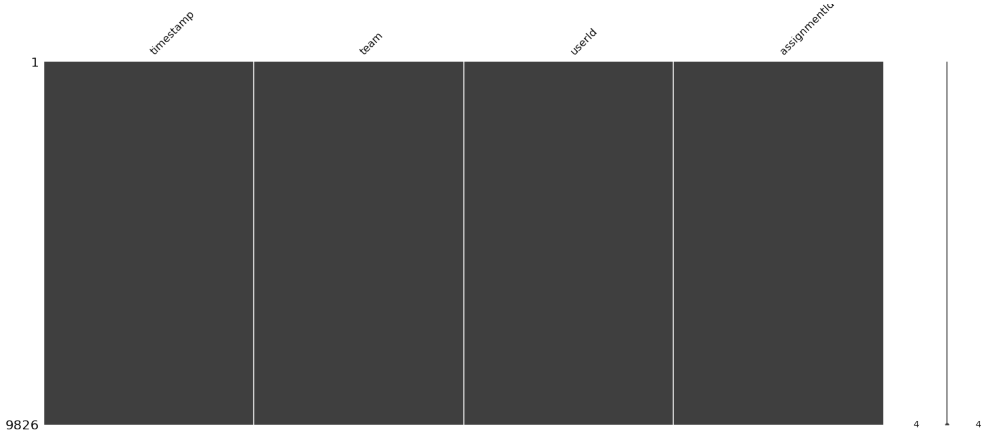

Figure 22: missingno_teamAssignments

At the beginning of the time series, we can see quite a steep decline. This was momental, since afterwards decline of team assignments is slow. At the end, we can observe a steep decline, that matches all the time sereies so far.

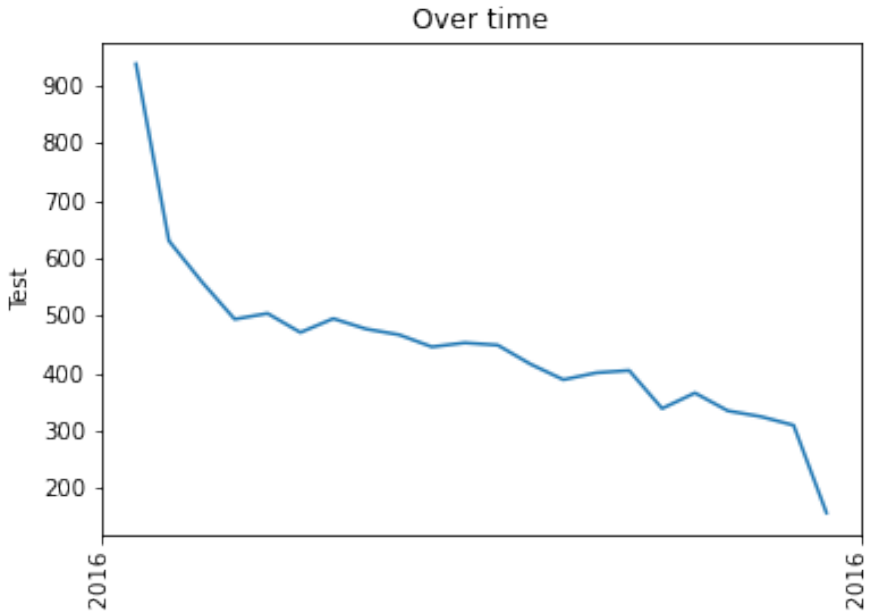

Figure 23: timeseries_teamAssignments

Attribute	Description
teamId	ID of the team
name	name of the team
teamCreationTime	time when team was created
teamEndTime	time when last member left the team
strength	how strong the team is (based on performance)
currentLevel	current level of the team

Table 12: team.csv

Team dataframe has no missing data.

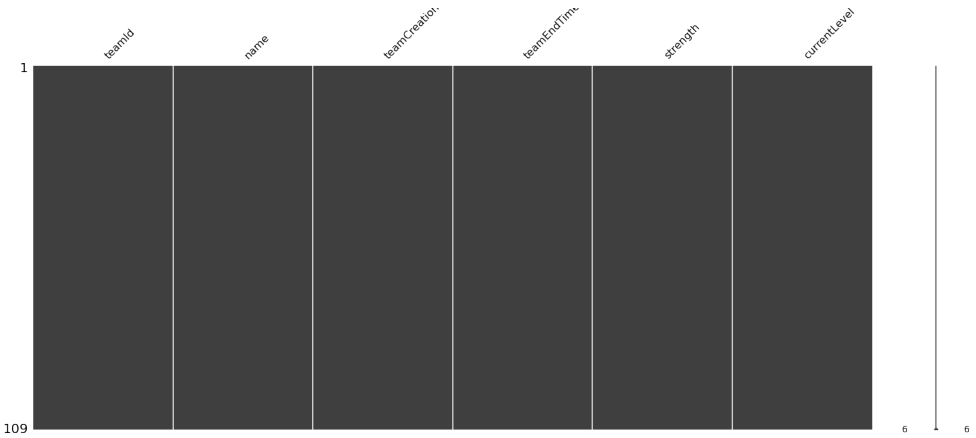

Figure 24: missingno_team

Analysing team creation time with team end time, we can see something odd. Blue line (representing team creation time) is showcasing how teams were created. Growth was slow, until it expend a lot. By that logic, there should be some teams that were ended (orange line). This can be seen from the start how teams were created and ended, but after a while there seems to be nothing regarding ending of the teams.

With this information, we can conclude two things:

- As with time series from above, this one is effected as well,
- There seems to be missing data (as we have the data but it is wrong) for team end date since it straight up ends in the middle of the graph.

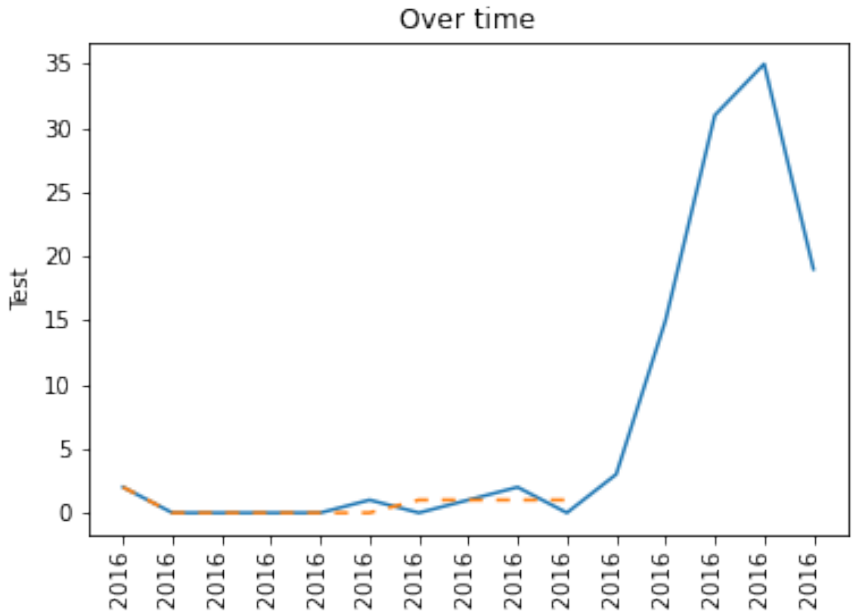

Figure 25: timeseries2_team

The strongest teams on the graph do not appear on the list of the biggest teams. What is interesting is team with id 9, they are 3rd in terms of strength (Figure 26) and in terms of spending (Figure16). This team appears to spend the most and be the strongest at the same time being 4th biggest team in terms of members (Figure 12).

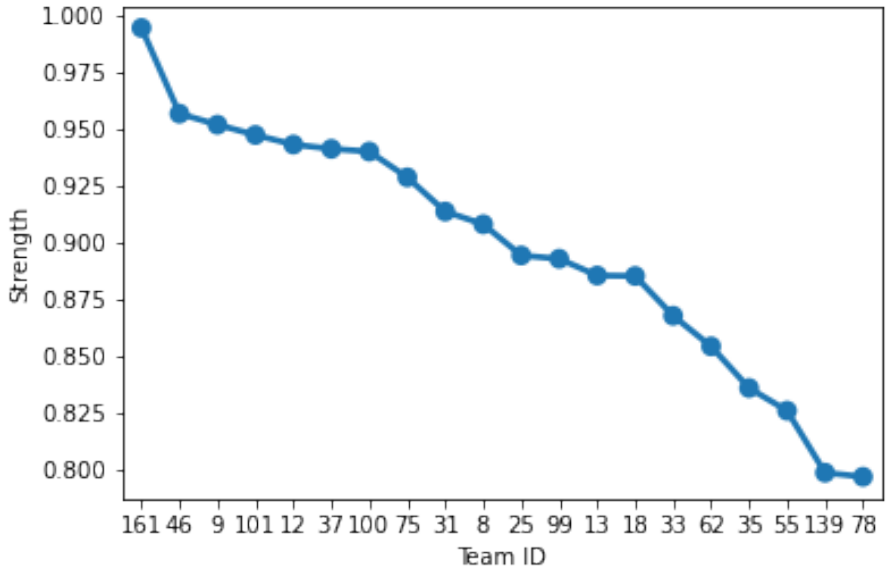

Figure 26: teamStrength_team

Teams with lower strength do not appear in the biggest team section. But 4th weakest team appears to be on the lower section on spending.

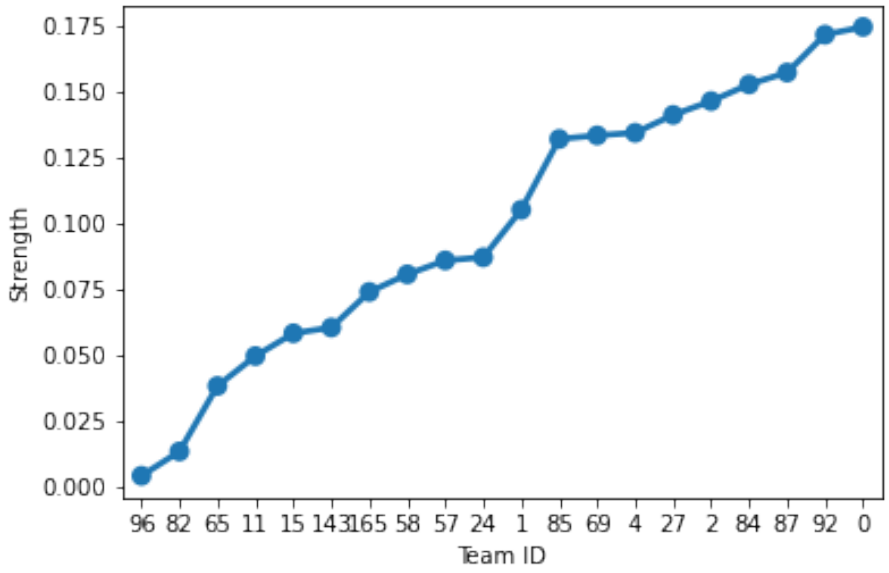

Figure 27: teamStrength2_team

Attribute	Description
timestamp	time when session started
userSessionId	ID of the session
userId	ID of the user in session
teamId	ID of the team that user is in
assignmentId	Temp ID assigned to the user (while in the team/session)
sessionType	type of the event (start or end)
teamLevel	level of the team in the current session
platformType	what device / platform is being used

Table 13: user-session.csv

User session dataframe does not have any missing values.

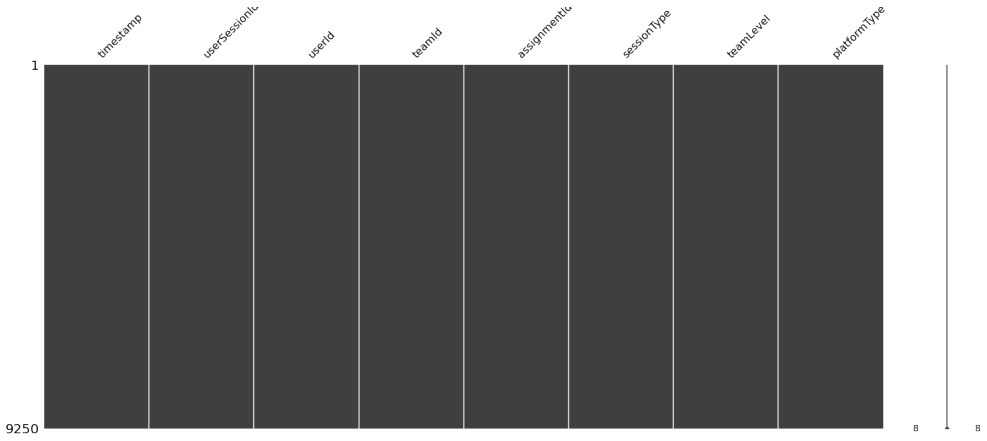

Figure 28: missingno_userSession

Time series for user session appears to be spiky (simmilar to levelEvents, Figure 21). It ends with the same fall as other time series events.

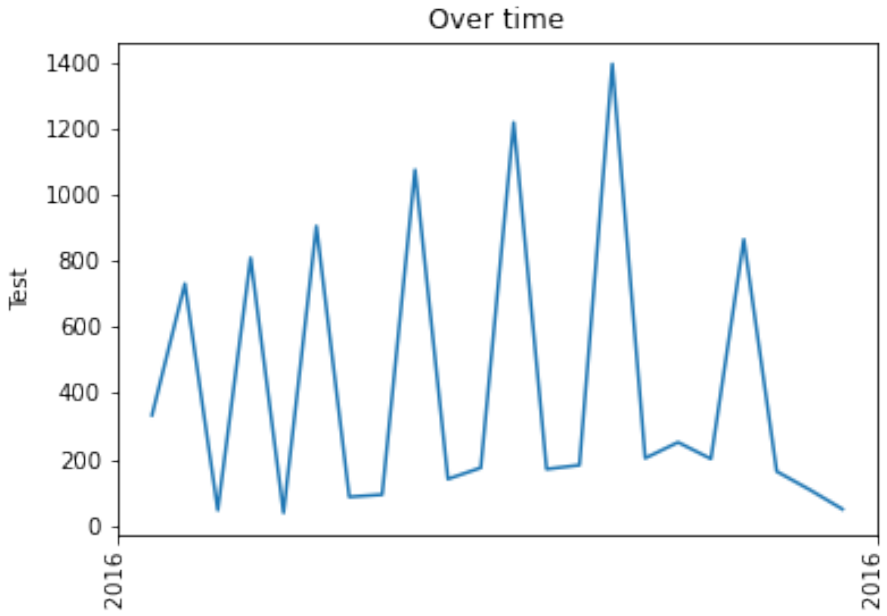

Figure 29: timeseries_userSession

If we compare start and end of sessions for platforms, we can see that users tend not to switch platforms while the session is running. This could be one of the game limitations, meaning if you would rejoin from other platform, your session would expire.

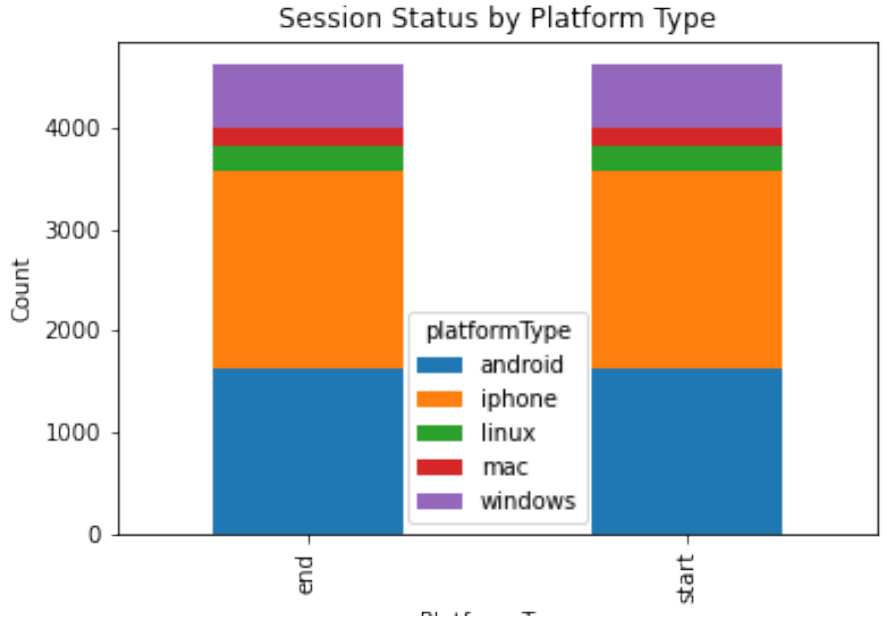

Figure 30: histogram_userSession

Attribute	Description
timestamp	time when first game starts
userId	ID of the user
nick	nickname chosen by the user
twitter	twitter handle of the user
dob	date of birth of the user
country	two letter country code of the user

Table 14: user.csv

Data frame users is the first one (in this path) to have missing data. The data that is missing is in country and it is a small amount.

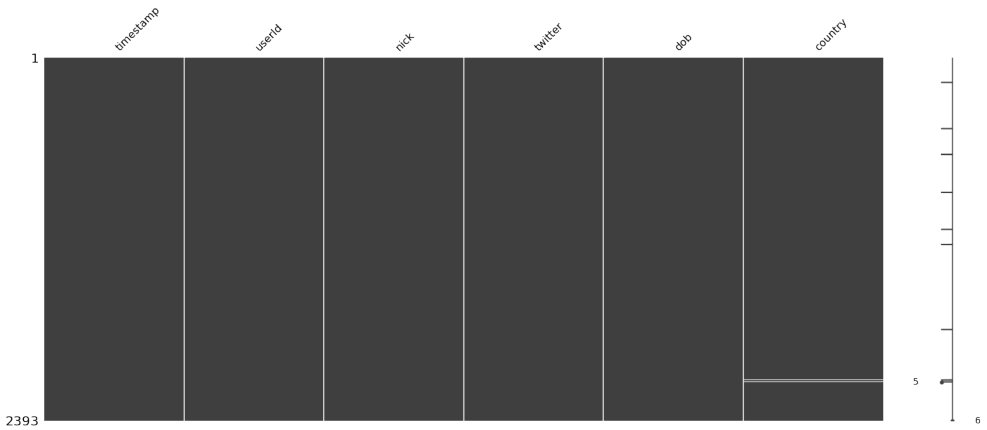

Figure 31: missingno_users

The demographic of our users seem to be on the younger side. The majority of players seem to be born after 1970.

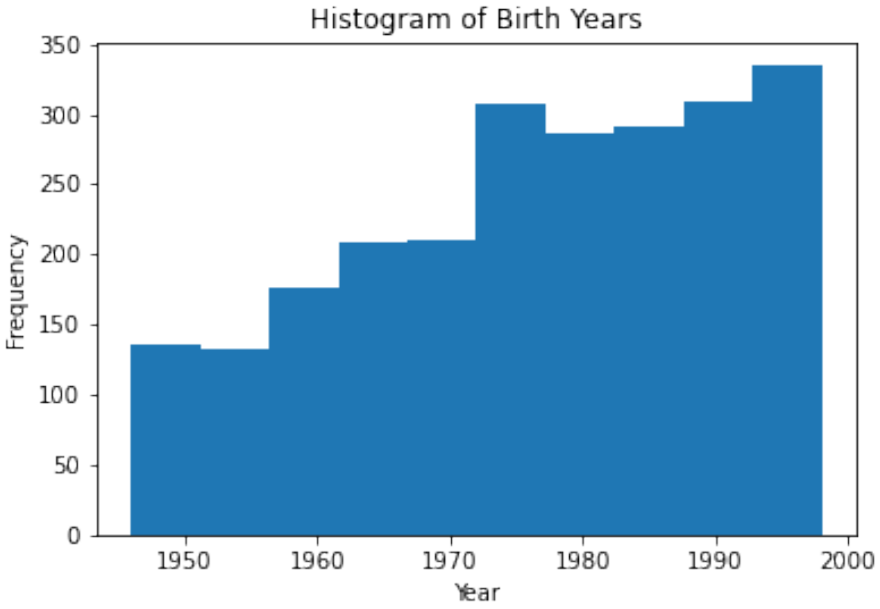

Figure 32: histogram_userSession

Geographically looking there seems to be players from all around the globe. This is good in terms of different markets and bad in terms of computational overhead (since we need computers in all of the regions for optimal performance).

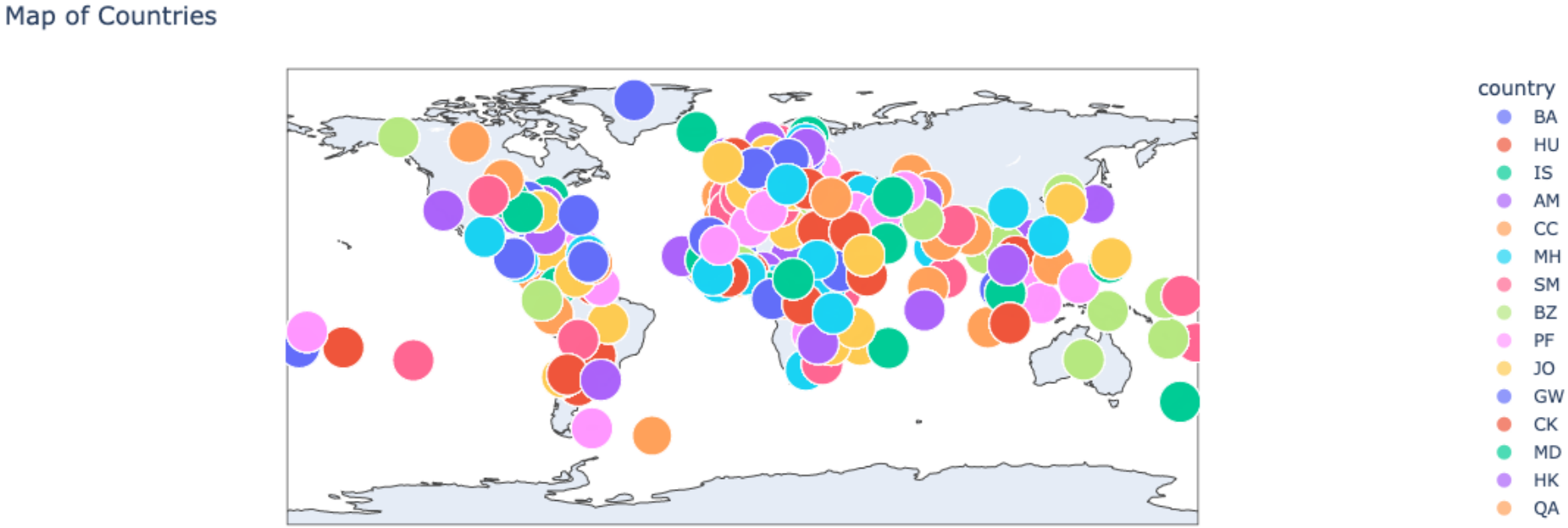

Figure 33: map

4 Classification results on the proposed data set

4.1 About classification

Classification is a process in machine learning where we categorise data (Kotsiantis, Zaharakis, and Pintelas 2006). This is used in daily bases in our lives (example: email filter, spam or not).

Figure bellow showcases how people are divided into sick and healthy. Section 9.3 uses this as an example in order to go over 4 (main) steps that are needed.

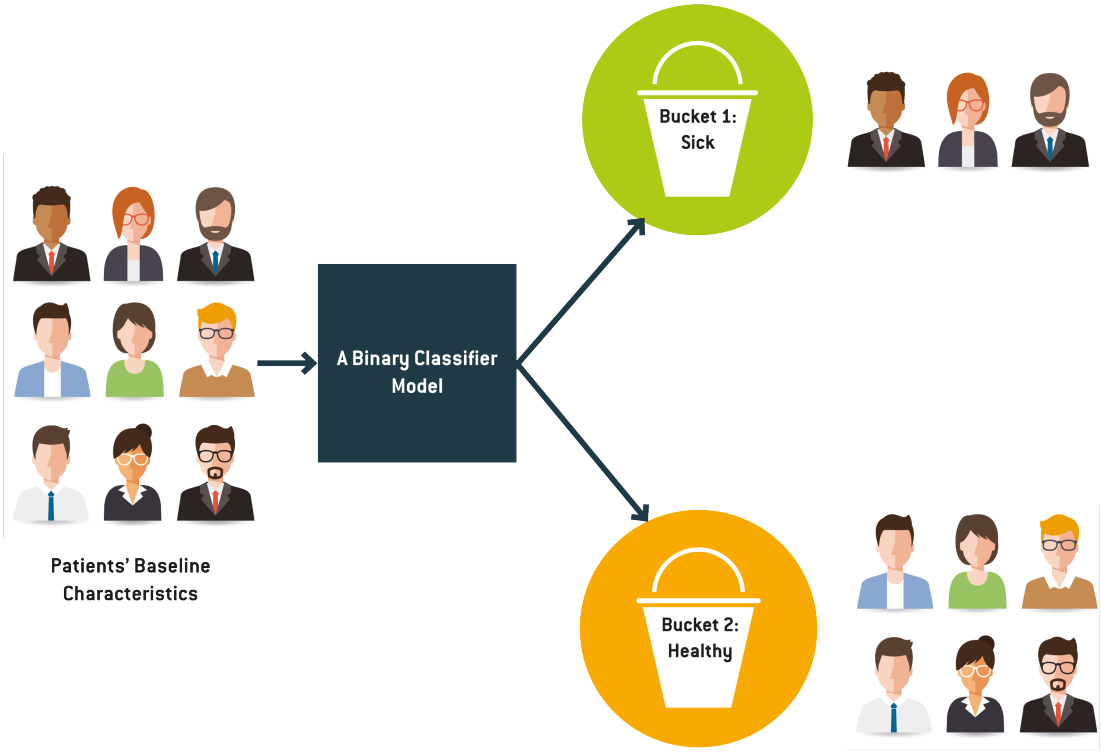

Figure 34: Classification showcase (S-cubed n.d.)

Code bellow showcase how the following steps were achieved. It looks similar since the idea was for code to be interchangeable regardless of the model.

As mentioned, we first split the data. Split is 80% training and 20% testing. For reproducibility reasons, seed is set to specific number (42). In our case, x (train/test) are the features that we use for prediction and y (train/test) being the feature we are trying to predict (what platform user is on).

```
def split_data(data_frame):
    indexer = StringIndexer(inputCol="platformType",
                            outputCol="label")
    data_frame = indexer.fit(data_frame).transform(data_frame)

    split_ratio = [0.8, 0.2]
    seed = 42
    train_data, test_data = data_frame.randomSplit(split_ratio,
                                                   seed=seed)

    x_train = train_data.select("platformType").toPandas()
    y_train = train_data.select("label").toPandas()

    x_test = test_data.select("platformType").toPandas()
    y_test = test_data.select("label").toPandas()

    return x_train, x_test, y_train, y_test
```

Listing 1: Split data function

4.2 Decision tree

4.2.1 About

Decision trees are one of the older concepts of predicting (Friedl and Brodley 1997). The idea is to determine if something goes on the left or right path. Example image bellow shows how decisions are made with it.

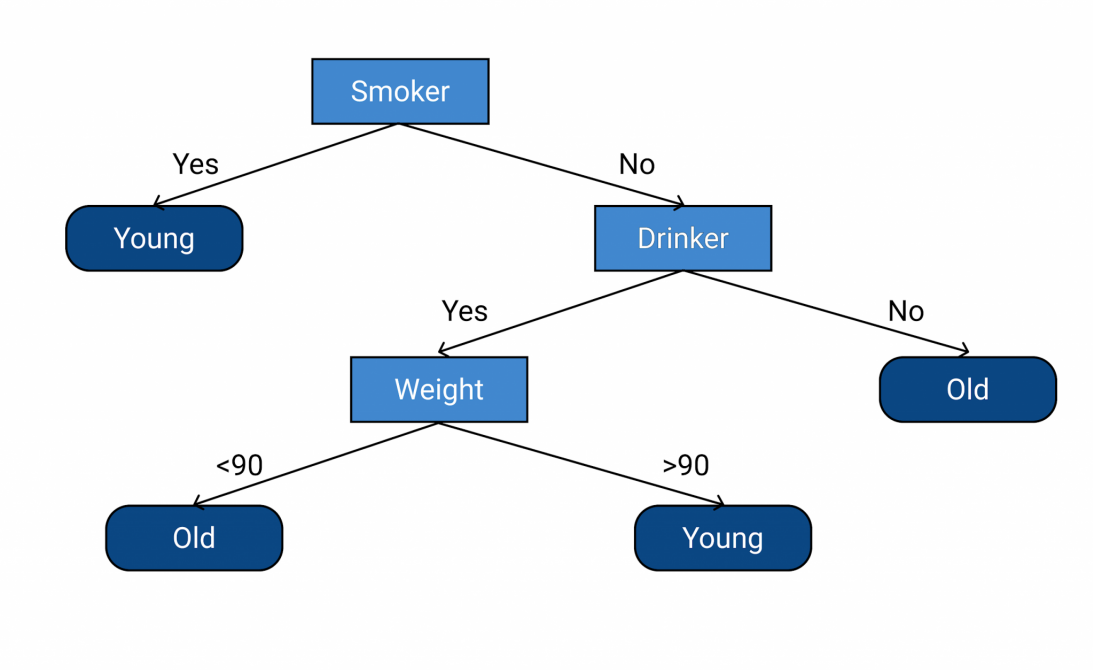

Figure 35: Decision tree showcase (Upgrad n.d.)

In our case we have used MLlib library from PySpark in order to use one of preexisting models.

4.2.2 Implementation

In order to create the model, we need to pass in the training data to the function. Since data isn't numeric, we need to overcome the problem by vectorizing it (use numeric representation). To improve the score of the model, we can fine tune it. One of the ways to do so for decision tree is to fine tunde the grid and create coss-validation. At the end we fit the model and return it.

```
def build_decision_tree_model(x_train, y_train):
    train_data = spark.createDataFrame
                (pd.concat([x_train, y_train], axis=1))

    platform_indexer = StringIndexer(inputCol="platformType",
                                     outputCol="platformIndex")
    train_data = platform_indexer.fit(train_data).transform(train_data)

    assembler = VectorAssembler(inputCols=["platformIndex"],
                                outputCol="features")
    train_data = assembler.transform(train_data)

    dt = DecisionTreeClassifier(featuresCol="features", labelCol="label")

    paramGrid = ParamGridBuilder() \
        .addGrid(dt.maxDepth, [2, 4, 6]) \
        .addGrid(dt.minInstancesPerNode, [1, 2, 4]) \
        .build()

    evaluator = MulticlassClassificationEvaluator(labelCol="label",
                predictionCol="prediction", metricName="accuracy")
    crossval = CrossValidator(estimator=dt, estimatorParamMaps=paramGrid,
                              evaluator=evaluator, numFolds=5)

    cvModel = crossval.fit(train_data)

    return cvModel.bestModel
```

Listing 2: Decision tree model function

4.2.3 Results

At the end, we need to evaluate the model in order to get the results. We pass in our model, test data and location where we want results to be saved. The function (that is split into parts 1-4) then evaluates the model and results are available to us.

```
def evaluate_model(model, x_test, y_test, file_Path):
    test_data = spark.createDataFrame
                (pd.concat([x_test, y_test], axis=1))

    platform_indexer = StringIndexer(inputCol="platformType",
                                     outputCol="platformIndex")
    test_data = platform_indexer.fit(test_data).transform(test_data)

    assembler = VectorAssembler(inputCols=["platformIndex"],
                                outputCol="features")
    test_data = assembler.transform(test_data)

    predictions = model.transform(test_data)

    evaluator = MulticlassClassificationEvaluator(labelCol="label",
                                        predictionCol="prediction")

    class_labels = test_data.select("label").distinct()
                    .rdd.flatMap(lambda x: x).collect()
    metrics = {}
```

Listing 3: Evaluate model model function -part 1

```
    for label in class_labels:
        evaluator.setMetricName("accuracy")
        evaluator.setMetricLabel(label)
        accuracy = evaluator.evaluate(predictions)

        evaluator.setMetricName("weightedPrecision")
        evaluator.setMetricLabel(label)
        precision = evaluator.evaluate(predictions)

        evaluator.setMetricName("weightedRecall")
        evaluator.setMetricLabel(label)
        recall = evaluator.evaluate(predictions)

        evaluator.setMetricName("weightedFMeasure")
        evaluator.setMetricLabel(label)
        f1_score = evaluator.evaluate(predictions)
```

Listing 4: Evaluate model model function -part 2

```
        metrics[label] = {"accuracy": accuracy, "precision": precision,
                          "recall": recall, "f1-score": f1_score}

    predictionAndLabels = predictions.select("prediction", "label").rdd
    multiclass_metrics = MulticlassMetrics(predictionAndLabels)
    confusion_matrix = multiclass_metrics.confusionMatrix().toArray()

    label_counts = predictionAndLabels.map(lambda x: (x[1], 1))
                   .reduceByKey(lambda x, y: x + y).collectAsMap()
    support = {label: label_counts.get(label, 0) for label in class_labels}

    metrics_table = pd.DataFrame.from_dict(metrics, orient="index")
    print("Metrics per Class:")
    print(metrics_table)

    support_table = pd.DataFrame.from_dict
                    (support, orient="index", columns=["Support"])
    print("Support:")
    print(support_table)

    fig, ax = plt.subplots()
    im = ax.imshow(confusion_matrix, cmap="Blues")

    tick_labels = np.arange(len(class_labels))
    ax.set_xticks(tick_labels)
    ax.set_yticks(tick_labels)
    ax.set_xticklabels(class_labels, rotation=45)
    ax.set_yticklabels(class_labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
```

Listing 5: Evaluate model model function -part 3

```
cbar = ax.figure.colorbar(im, ax=ax)
cbar.ax.set_ylabel("Count", rotation=-90, va="bottom")

for i in range(len(class_labels)):
    for j in range(len(class_labels)):
        text = ax.text(j, i, int(confusion_matrix[i, j]),
                       ha="center", va="center", color="w")

plt.title("Confusion Matrix")

metrics_table.to_csv(file_Path + "/metrics.csv")
plt.savefig(file_Path + "/confusion_matrix.png")

plt.close()
```

Listing 6: Evaluate model model function -part 4

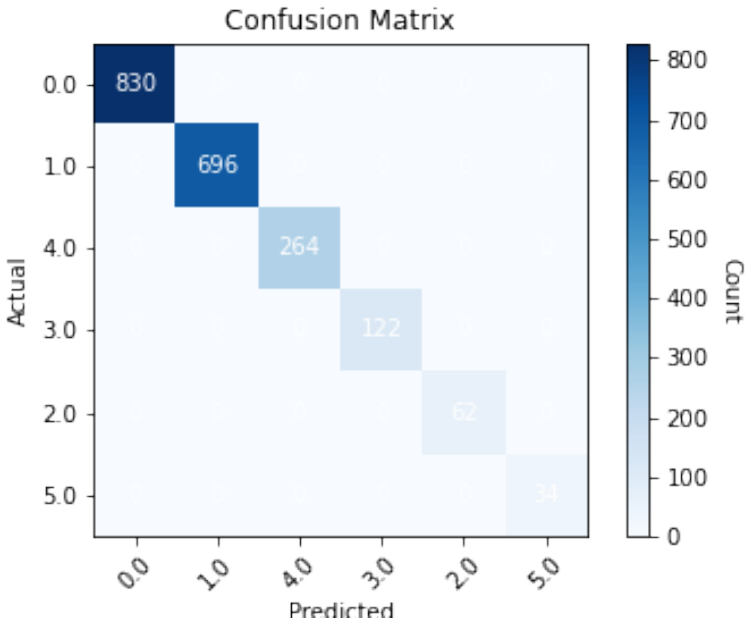

Figure 36: Decision tree confusion matrix

4.3 SVM

4.3.1 About

SVM (Support Vector Machines) works by, simply put, dividing the data into categories (H. Wang and Hu 2005). This is done with "drawing the line" on the grid.

Example bellow showcases how two classes are divided by the model.

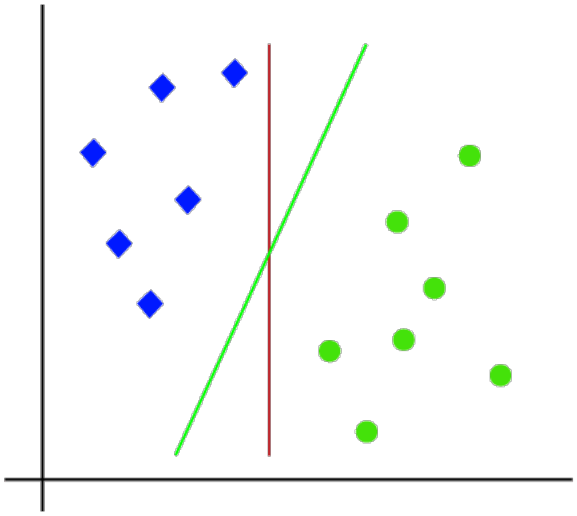

Figure 37: SVM example (Rocketloop n.d.)

As with decision tree, we have used same library in order to get SVM model.

4.3.2 Implementation

Implementation is similar to decision tree. We are expecting training data in the function, select platform type as what to predict and vectorise he data. SVM can be optimised as well, grid was used (same as with decision trees) and cross validation was applied as well. At the end, SVM model is returned.

```
def build_svm_model(x_train, y_train):
    train_data = spark.createDataFrame
                 (pd.concat([x_train, y_train], axis=1))

    platform_indexer = StringIndexer(inputCol="platformType",
                                     outputCol="indexedLabel")
    train_data = platform_indexer.fit(train_data).transform(train_data)

    assembler = VectorAssembler(inputCols=["indexedLabel"],
                                outputCol="features")
    train_data = assembler.transform(train_data)

    svm = LinearSVC(featuresCol="features", labelCol="indexedLabel")

    paramGrid = ParamGridBuilder() \
        .addGrid(svm.maxIter, [10, 100]) \
        .addGrid(svm.regParam, [0.1, 0.01]) \
        .build()

    ovr = OneVsRest(classifier=svm)

    evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel",
                predictionCol="prediction", metricName="accuracy")
    crossval = CrossValidator(estimator=ovr, estimatorParamMaps=paramGrid,
                              evaluator=evaluator, numFolds=5)

    cvModel = crossval.fit(train_data)

    return cvModel.bestModel
```

Listing 7: SVM model function

4.3.3 Results

Evaluate model (same function as with decision trees) was used in order to measure performace of the SVM. Model, test data and path where we want to save results is being passed to the function.

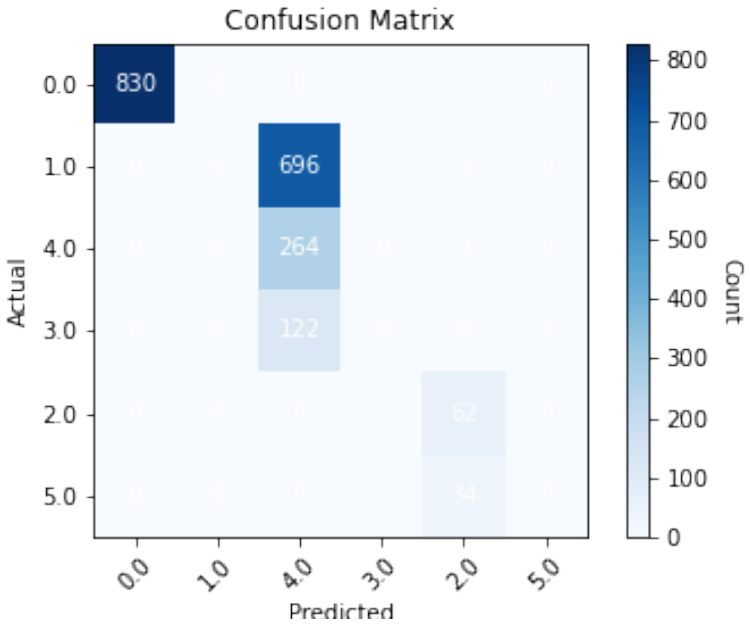

Figure 38: SVM onfusion matrix

4.4 Execute classification code

Now that we have the models ready, we can call the code bellow in order to split the data and build models.

```
x_train, x_test, y_train, y_test = split_data(mega_dataframe)

dt_model = build_decision_tree_model(x_train, y_train)
svm_model = build_svm_model(x_train, y_train)

evaluate_model(dt_model, x_test, y_test,
file_Path = file_paths_dict["classification"] + "DecisionTree")

evaluate_model(svm_model, x_test, y_test,
file_Path = file_paths_dict["classification"] + "SVM")
```

Listing 8: Execute the classification functions

Comparing classifications, we can see that our models did not perform the best. Based on the EDA, we know that mobile platform (iphone and android) are 80% of the devices used. Since our data is unbalanced, this can be a factor why results are skewed towards one end.

5 Clustering results on the proposed data set

5.1 About Clustering

Clustering is a process of grouping items together based on their distance (Likas, Vlassis, and Verbeek 2003). Lets take the image from bellow as an example. We have 3 classes (red, green and blue). Model calculates the distances between items (nodes) and classifies them based on how close are they between each other.

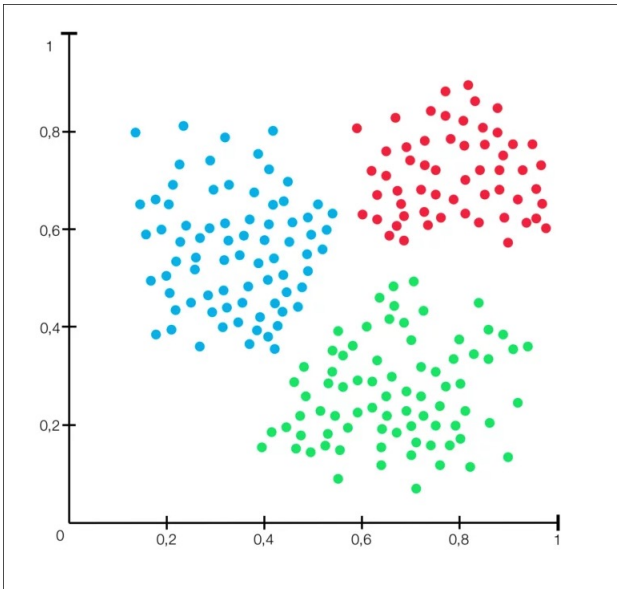

Figure 39: Clustering (Rocketloop n.d.)

Further explanation can be seen at 9.4.
Code bellow showcases how we split the data. It is similar to classification.

```
def train_test_split(dataframe, test_ratio=0.2):
    dataframe = dataframe.withColumn('strength', col('strength').cast('double'))

    train_df, test_df = dataframe.randomSplit
                        ([1 - test_ratio, test_ratio], seed=42)

    return train_df, test_df
```

Listing 9: Split data function

5.2 K-means

5.2.1 About

K-means is a clustering algorithm (Hartigan and Wong 1979). It is one of the more popular algorithms used for clustering. How it usually works is by us providing k (number of clusters) to the algorithm. Our model will then calculate the distance (there are different calculations: Euclidean, Manhattan, etc...) between nodes. Afterwards we can see the ideal number and chose it as our number of clusters.

5.2.2 Implementation

In order to create the model, we pass in number of clusters, training and testing data to the function. In a similar way as with classification, the model is build in order to classify team strength. At the end, model is fitted and predictions are returned.

MLlib library from PySpark was used for kmeans model.

```
def create_kmeans_model(train_df, test_df, k):
    assembler = VectorAssembler(inputCols=['strength'], outputCol='features')

    x_train = assembler.transform(train_df).select('features')
    x_test = assembler.transform(test_df).select('features')

    kmeans = KMeans(k=k, seed=42)

    model = kmeans.fit(x_train)

    predictions = model.transform(x_test)

    return predictions
```

Listing 10: Create kmeans model

5.2.3 Results

In order to evaluate the model, train, test, max k (max number of clusters) are passed to the function along with where to save the results. In our case, we are using silhouette score to measure cluster performance. In the function bellow, we are creating scores (from create model function) for different number of clusters. At the end we append score to the list which is then used in order to plot the silhouette score.

```
def evaluate_kmeans_model(train_df, test_df, max_k, file_Path):
    silhouette_scores = []

    for k in range(2, max_k + 1):
        predictions = create_kmeans_model(train_df, test_df, k)

        evaluator = ClusteringEvaluator()

        silhouette = evaluator.evaluate(predictions)

        silhouette_scores.append(silhouette)

    plt.plot(range(2, max_k + 1), silhouette_scores, marker='o')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Score vs. Number of Clusters')

    plt.savefig(file_Path)

    plt.close()
```

Listing 11: Evaluate kmeans model

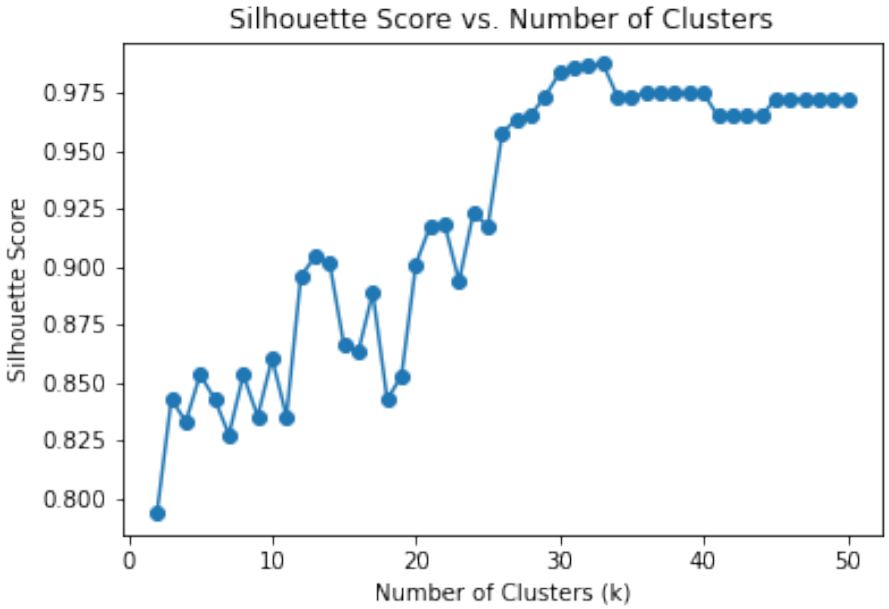

Figure 40: Silhouette k-means

5.3 GMM

5.3.1 About

Gaussian Mixture Model (or GMM) is clustering algorithm (Reynolds et al. 2009) that assumes data is generated from a mixture of Gaussian distributions. Model is trained by expectation maximization based algorithm. Underneath we are trying to predict what is the likelihood of an item to appear in our data.

Pyspark MLlib library was used for implementation of the model as well.

5.3.2 Implementation

Implementation is the same as with k-means. The one difference is that we are using different model.

```
def create_gmm_model(train_df, test_df, k):
    assembler = VectorAssembler(inputCols=['strength'], outputCol='features')

    x_train = assembler.transform(train_df).select('features')

    x_test = assembler.transform(test_df).select('features')

    gmm = GaussianMixture(k=k, seed=42)

    model = gmm.fit(x_train)

    predictions = model.transform(x_test)

    return predictions
```

Listing 12: Create GMM model

5.3.3 Results

Results for GMM are calculated the same way as for k-means. The only difference is the function that we are using in order to generate the model.

```
def evaluate_gmm_model(train_df, test_df, max_k, file_Path):
    silhouette_scores = []

    for k in range(2, max_k + 1):
        predictions = create_gmm_model(train_df, test_df, k)

        evaluator = ClusteringEvaluator()

        silhouette = evaluator.evaluate(predictions)

        silhouette_scores.append(silhouette)

    plt.plot(range(2, max_k + 1), silhouette_scores, marker='o')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Silhouette Score')
    plt.title('Silhouette Score vs. Number of Clusters')

    plt.savefig(file_Path)

    plt.close()
```

Listing 13: Evaluate gmm model

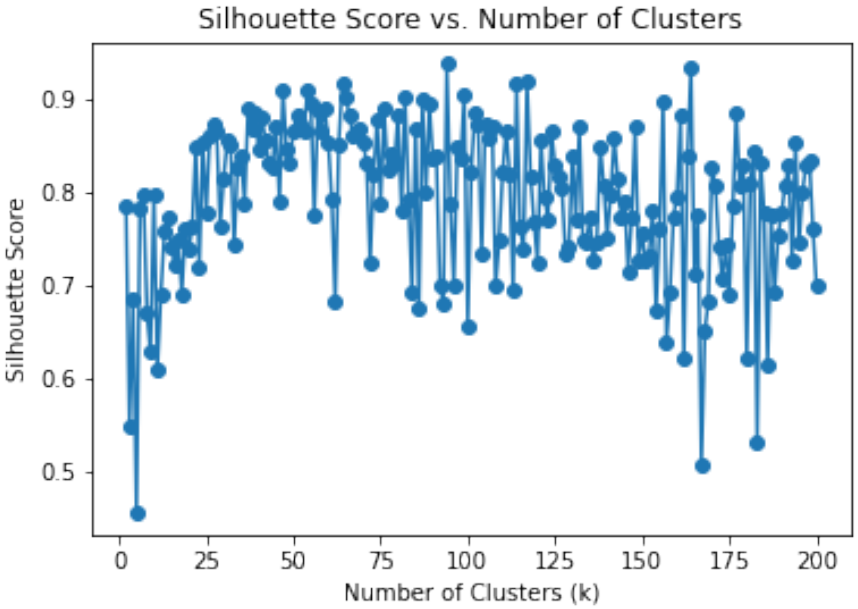

Figure 41: Silhouette GMM

5.4 Execute clustering code

Code bellow is the main part that calls the functions in order to generate the models and evaluates the models.

As seen in results section, or models performed quite well. K-means seems to be the most efficient with about 30-34 clusters while GNN seems to be happy with 100 distributions.

```
train_df, test_df = train_test_split(mega_dataframe)
train_df, test_df = train_test_split(mega_dataframe)

evaluate_kmeans_model(train_df, test_df, 50,
file_Path = file_paths_dict["clustering"] + "kmenas")

evaluate_gmm_model(train_df, test_df, 200,
file_Path = file_paths_dict["clustering"] + "gmm")
```

Listing 14: Execute the classification functions

6 Graph analysis

Graph databases are part of NoSQL (also known as NotOnlySQL). They don't represent tables and relations.

There are different NoSQL databases (MariaDB, Cassandra, MarkLogic, etc...), each one is focusing on its own solution. MariaDB is document oriented, but in our case we are using Neo4j (Miller 2013) which is graph database. Instead of tables and relations, we have nodes and edges. Another change is language. Queries are not written/executed in SQL but with Cypher.

Setup of the database can be seen at 9.5 and data used in 9.6.

6.1 Graph 1

First graph database is showcasing relationships between chat join, leave, mention and respond. By that we can see what chat sessions seems to be the most active.

How data was inserted can be seen in Python code bellow.

```
def create_user_sessions(data_chat_join_team_chat,
                         data_chat_leave_team_chat,
                         data_chat_mention_team_chat,
                         data_chat_respond_team_chat):
    uri, user, password = get_creds(0)
    driver = GraphDatabase.driver(uri, auth=(user, password))

    create_user_query = "CREATE (:User {id: $user_id})"
    create_teamchat_session_query =
        "CREATE (:TeamchatSession {id: $teamchat_session_id, date: $date})"
    create_chat_item_query = "CREATE (:ChatItem {id: $chat_item})"
    create_chat_relation_query =
        "MATCH (c1), (c2) WHERE
            c1.id = $chatid1 AND
            c2.id = $chatid2 CREATE (c1)-[:RESPONDS_TO]->(c2)"
    create_mention_relation_query =
        "MATCH (u), (c) WHERE
            u.id = $user_id AND
            c.id = $chat_item CREATE (u)-[:MENTIONS]->(c)"
    create_join_relation_query =
        "MATCH (u), (t) WHERE
            u.id = $user_id AND
            t.id = $teamchat_session_id CREATE (u)-[:JOINS]->(t)"
    create_leave_relation_query =
        "MATCH (u), (t) WHERE
            u.id = $user_id AND
            t.id = $teamchat_session_id CREATE (u)-[:LEAVES]->(t)"
```

Listing 15: User sessions -part 1

```
queries = [
    (create_user_query,
        data_chat_join_team_chat.select("user_id")
        .distinct()),
    (create_teamchat_session_query,
        data_chat_join_team_chat
        .select("teamchat_session_id", "date")
        .distinct()),
    (create_chat_item_query,
        data_chat_mention_team_chat
        .select("chat_item")
        .distinct()),
    (create_chat_relation_query,
        data_chat_respond_team_chat
        .select("chatid1", "chatid2")),
    (create_mention_relation_query,
        data_chat_mention_team_chat
        .select("user_id", "chat_item")),
    (create_join_relation_query,
        data_chat_join_team_chat
        .select("user_id", "teamchat_session_id")),
    (create_leave_relation_query,
        data_chat_leave_team_chat
        .select("user_id", "teamchat_session_id"))
]

with driver.session() as session:
    for query, data in queries:
        for row in data.collect():
            session.run(query, **row.asDict())
```

Listing 16: User sessions -part 2

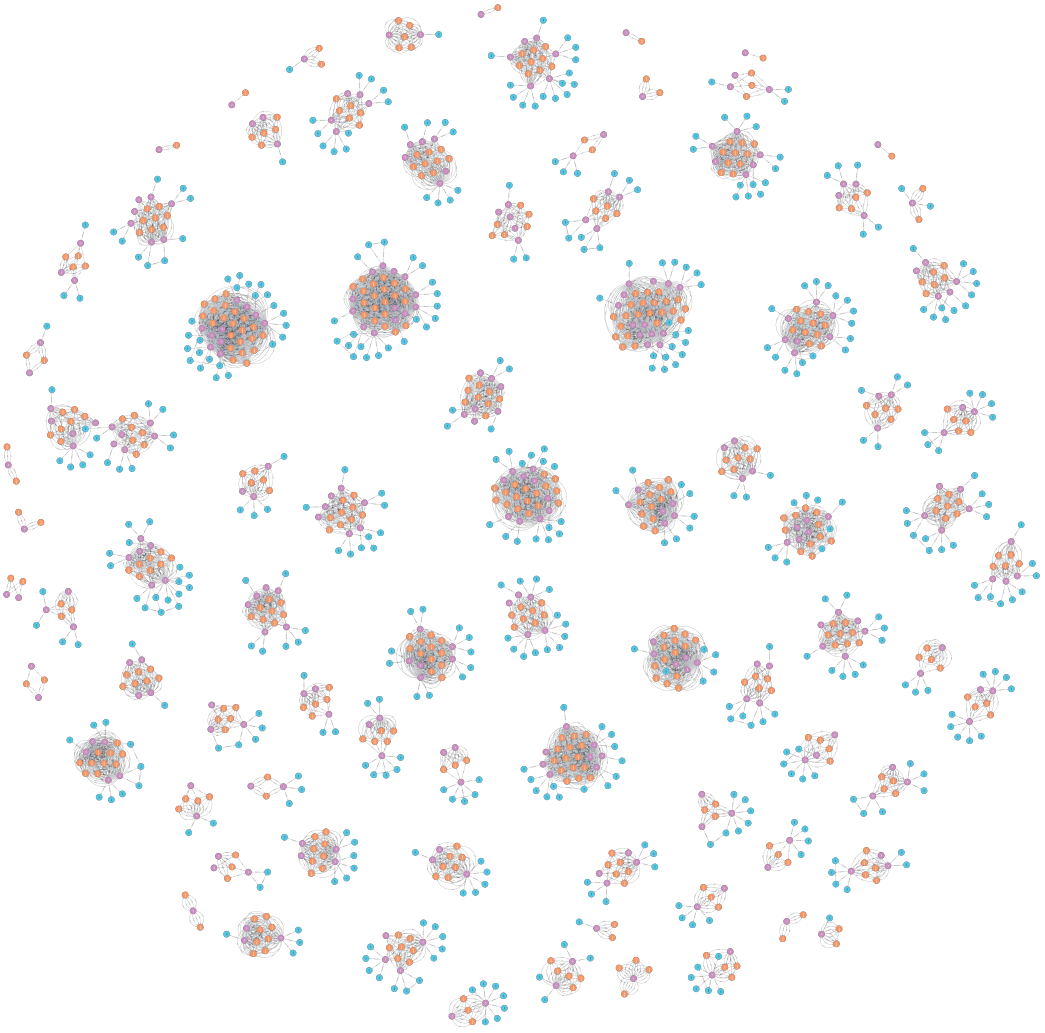

Figure 42: Graph 1

If we want to filter the nodes by the most mentioned, we can use Cypher code from bellow. This will showcase top mentioned nodes and their relations.

```
MATCH (n)-[r]->()
WHERE TYPE(r) IN ['RESPONDS_TO', 'MENTIONS', 'JOINS', 'LEAVES']
WITH n, COUNT(*) AS mentionsCount
ORDER BY mentionsCount DESC
LIMIT 10
MATCH (n)-[r]->(m)
RETURN n, r, m
```

Listing 17: Cypher filter 1

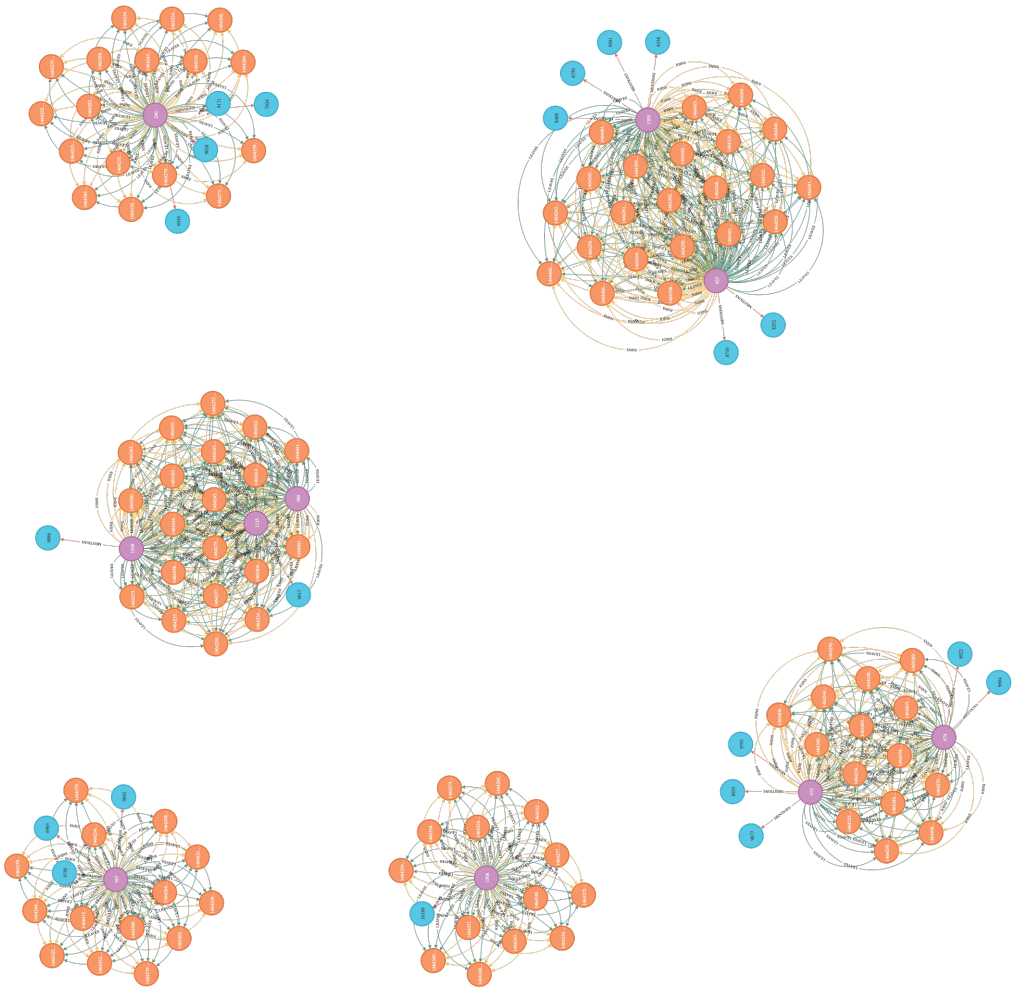

Figure 43: Graph 1 filtered

6.2 Graph 2

Messages between users are not as important to us as they are between the users. What is important to us is reasoning for n number of messages. If users are constantly interacting, it means that they are satisfied and would gladly return.

Function bellow showcases how data was inserted.

```
def create_msg_between_users(data_chat_join_team_chat,
                             data_chat_leave_team_chat,
                             data_chat_mention_team_chat,
                             data_chat_respond_team_chat):
    uri, user, password = get_creds(1)
    driver = GraphDatabase.driver(uri, auth=(user, password))

    create_user_query = "MERGE (:User {id: $user_id})"
    create_teamchat_session_query =
        "MERGE (:TeamchatSession {id: $teamchat_session_id, date: $date})"
    create_chat_item_query = "MERGE (:ChatItem {id: $chat_item})"
    create_chat_relation_query = """
        MATCH (c1), (c2)
        WHERE c1.id = $chatid1 AND c2.id = $chatid2
        AND c1.user_id <> c2.user_id
        CREATE (c1)-[:RESPONDS_TO]->(c2)
    """
    create_mention_relation_query =
        "MATCH (u), (c) WHERE
            u.id = $user_id AND
            c.id = $chat_item CREATE (u)-[:MENTIONS]->(c)"
    create_join_relation_query =
        "MATCH (u), (t) WHERE
            u.id = $user_id AND
            t.id = $teamchat_session_id CREATE (u)-[:JOINS]->(t)"
    create_leave_relation_query =
        "MATCH (u), (t) WHERE
            u.id = $user_id AND
            t.id = $teamchat_session_id CREATE (u)-[:LEAVES]->(t)"
    create_send_relation_query =
        "MATCH (u), (c), (t) WHERE
            u.id = $user_id AND c.id = $chat_item AND
            t.id = $teamchat_session_id CREATE (u)-[:SENDS]->(c)-[:IN]->(t)"
```

Listing 18: Messages between users -part 1

```
queries = [
    (create_user_query,
        data_chat_join_team_chat
        .select("user_id").distinct()),
    (create_teamchat_session_query,
        data_chat_join_team_chat
        .select("teamchat_session_id", "date").distinct()),
    (create_chat_item_query,
        data_chat_mention_team_chat
        .select("chat_item").distinct()),
    (create_chat_relation_query,
        data_chat_respond_team_chat
        .select("chatid1", "chatid2")),
    (create_mention_relation_query,
        data_chat_mention_team_chat
        .select("user_id", "chat_item")),
    (create_join_relation_query,
        data_chat_join_team_chat
        .select("user_id", "teamchat_session_id")),
    (create_leave_relation_query,
        data_chat_leave_team_chat
        .select("user_id", "teamchat_session_id")),
    (create_send_relation_query,
        data_chat_mention_team_chat
        .join(data_chat_join_team_chat["user_id"])
        .join(data_chat_leave_team_chat, ["user_id", "teamchat_session_id"])
        .select("user_id", "chat_item", "teamchat_session_id"))
]

with driver.session() as session:
    for query, data in queries:
        for row in data.collect():
            session.run(query, **row.asDict())
```

Listing 19: Messages between users -part 2

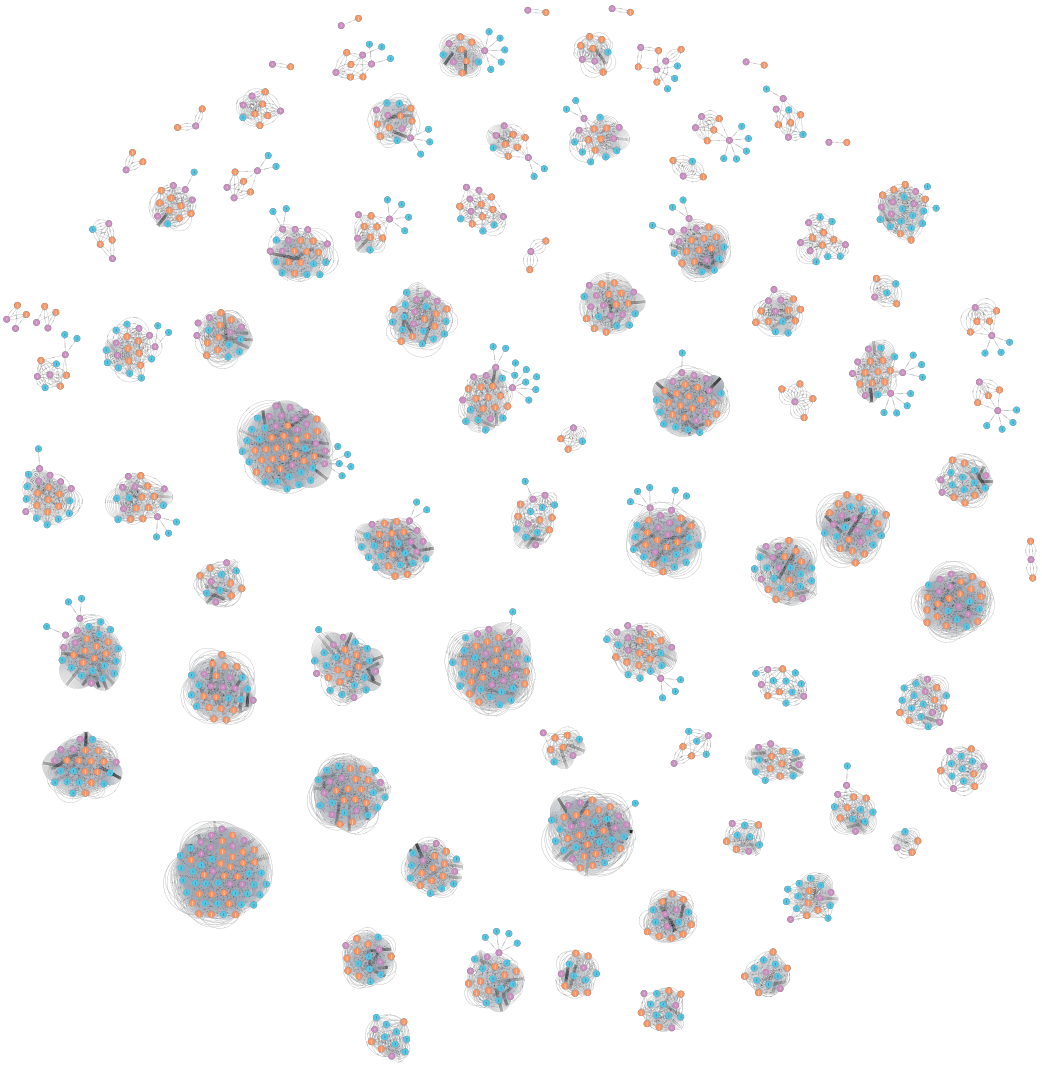

Figure 44: Graph 2

If we want to see the most messages send, we can execute the code bellow in order to see what users seem to be the most active.

```
MATCH (n)-[r:SENDS]->()
WITH n, COUNT(r) AS sendsCount
ORDER BY sendsCount DESC
LIMIT 10
MATCH (n)-[rel]->(m)
RETURN n, rel, m
```

Listing 20: Cypher filter 2

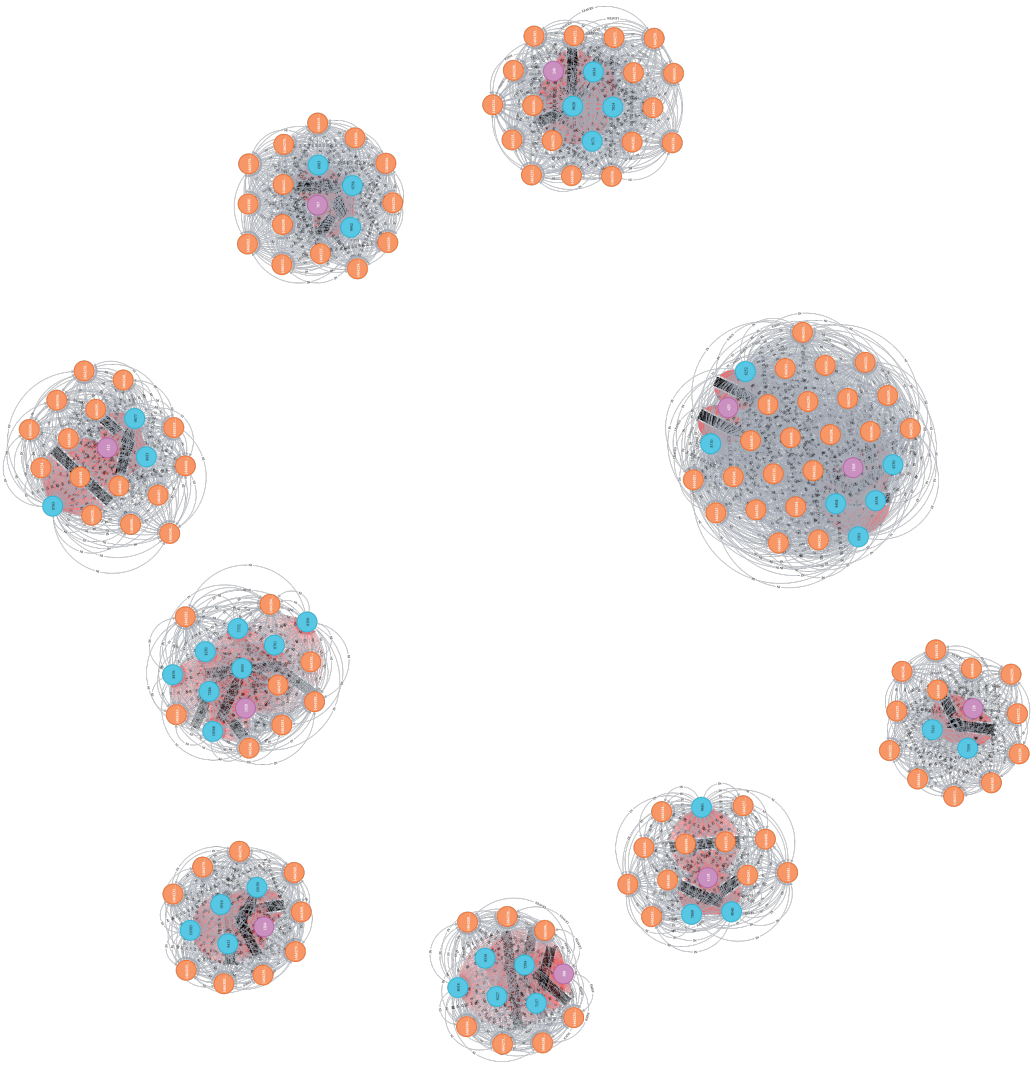

Figure 45: Cypher filter 2

6.3 Graph 3

This graph is a representation of all the data. The idea is to have one big graph that we could fitler information from.

Data insertion is showcased with the function bellow.

```
def create_mega_graph(mega_dataframe):
    uri, user, password = get_creds(2)
    driver = GraphDatabase.driver(uri, auth=(user, password))

    create_user_query = "MERGE (:User {id: $userId})"
    create_country_query = "MERGE (:Country {name: $country})"
    create_team_query = "MERGE (:Team {id: $teamId, name: $name})"
    create_platform_query = "MERGE (:PlatformType {name: $platformType})"
    create_ad_query = "MERGE (:Ad {id: $adId})"
    create_category_query = "MERGE (:AdCategory {name: $adCategory})"
    create_country_user_rel_query =
        "MATCH (u:User), (c:Country) WHERE
            u.id = $userId AND
            c.name = $country CREATE (u)-[:BELONGS_TO_COUNTRY]->(c)"
    create_team_user_rel_query =
        "MATCH (u:User), (t:Team) WHERE
            u.id = $userId AND
            t.id = $teamId CREATE (u)-[:MEMBER_OF_TEAM {strength: $strength}]->(t)
    create_platform_user_rel_query =
        "MATCH (u:User), (p:PlatformType) WHERE
            u.id = $userId AND
            p.name = $platformType CREATE (u)-[:USES_PLATFORM]->(p)"
    create_ad_user_rel_query =
        "MATCH (u:User), (a:Ad) WHERE
            u.id = $userId AND
            a.id = $adId CREATE (u)-[:VIEWED_AD]->(a)"
    create_category_ad_rel_query =
        "MATCH (a:Ad), (c:AdCategory) WHERE
            a.id = $adId AND
            c.name = $adCategory CREATE (a)-[:BELONGS_TO_CATEGORY]->(c)"
```

Listing 21: Mega graph -part 1

```
queries = [
    (create_user_query,
        mega_dataframe.select("userId")
        .distinct()),
    (create_country_query,
        mega_dataframe.select("country")
        .distinct()),
    (create_team_query,
        mega_dataframe
        .select("teamId", "name")
        .distinct()),
    (create_platform_query,
        mega_dataframe
        .select("platformType")
        .distinct()),
    (create_ad_query,
        mega_dataframe
        .select("adId")
        .distinct()),
    (create_category_query,
        mega_dataframe
        .select("adCategory")
        .distinct()),
    (create_country_user_rel_query,
        mega_dataframe
        .select("userId", "country")),
    (create_team_user_rel_query,
        mega_dataframe
        .select("userId", "teamId", "strength", "name")),
    (create_platform_user_rel_query,
        mega_dataframe
        .select("userId", "platformType")),
    (create_ad_user_rel_query,
        mega_dataframe
        .select("userId", "adId")),
    (create_category_ad_rel_query,
        mega_dataframe
        .select("adId", "adCategory"))
]

with driver.session() as session:
    for query, data in queries:
        for row in data.collect():
            session.run(query, **row.asDict())
```


Listing 22: Mega graph -part 2

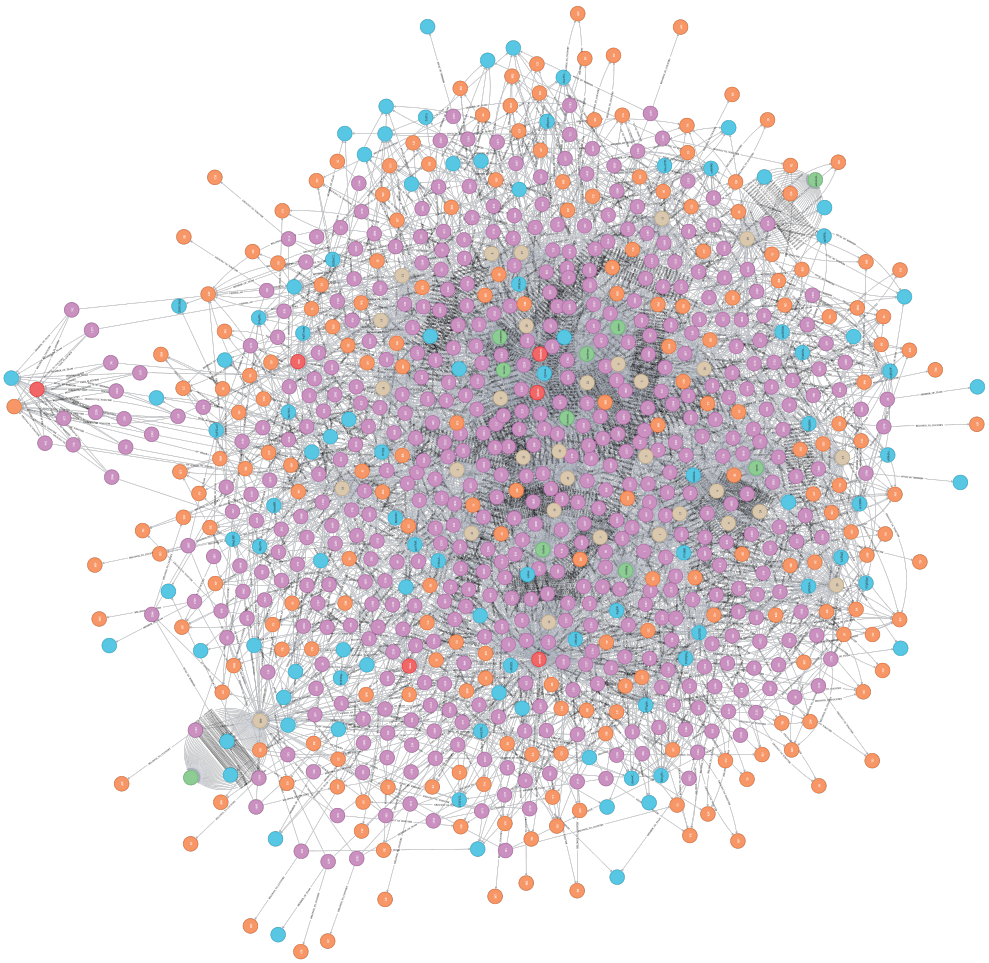

Figure 46: Graph 3

We can filter out couturiers with most of the users.

```
MATCH (n)-[r:BELONGS_TO_COUNTRY]->()
WITH n, COUNT(r) AS belongsToCountryCount
ORDER BY belongsToCountryCount DESC
LIMIT 10
MATCH (n)-[rel]->(m)
RETURN n, rel, m
```

Listing 23: Cypher filter 3

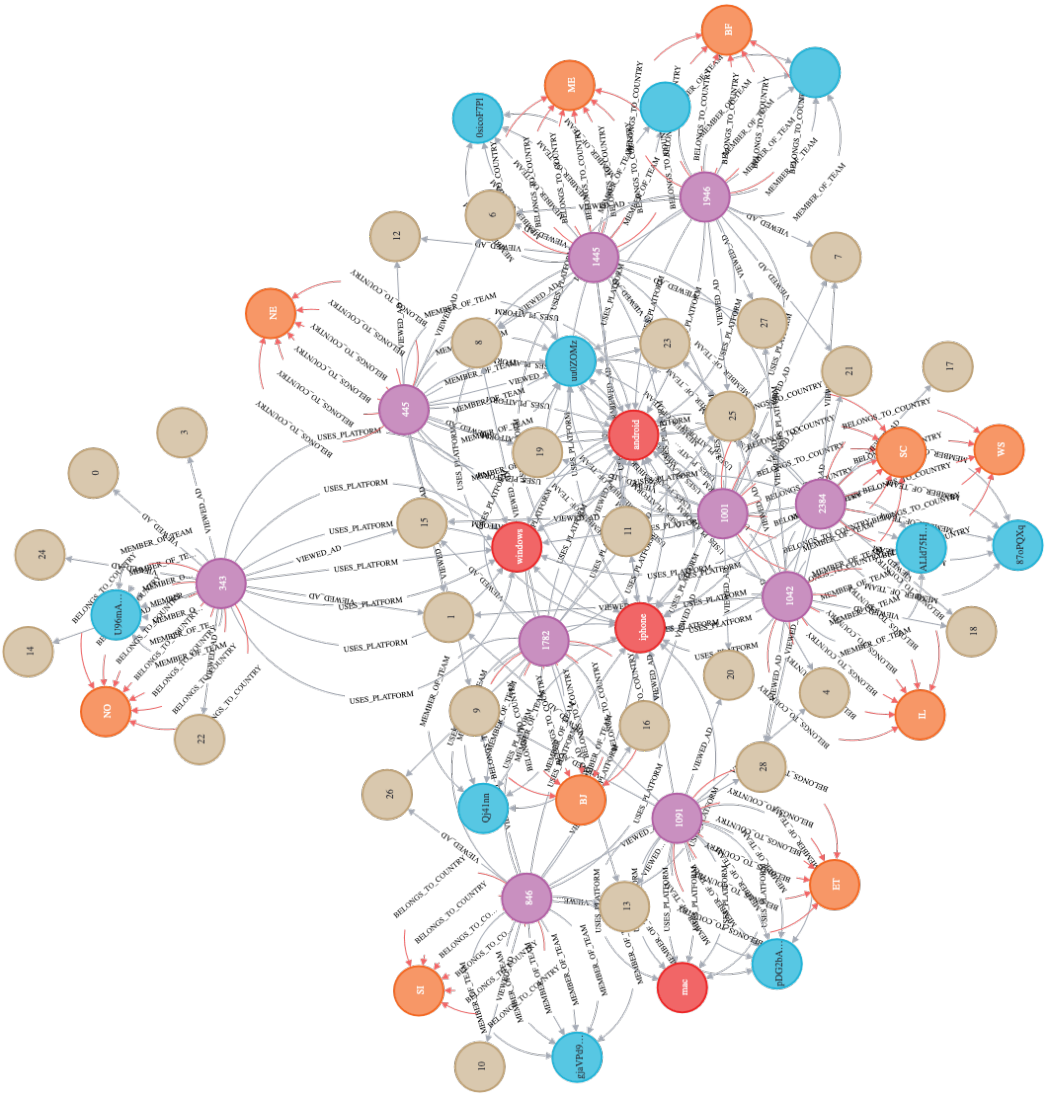

Figure 47: Graph 3 filtered

6.4 Graph 4

The final graph showcases how users are connected to ads. From marketing perspective, this could be the most important part. For example, we could see what advertisements are popular per country.

Insertion of the data is demonstrated with the function bellow.

```
def create_ad_graph(ad_dataframe):
    uri, user, password = get_creds(3)
    driver = GraphDatabase.driver(uri, auth=(user, password))

    create_user_query = "MERGE (:User {id: $userId})"
    create_country_query = "MERGE (:Country {name: $country})"
    create_team_query =
        "MERGE (:Team {id: $teamId, name: $team, price: $price})"
    create_ad_query = "MERGE (:Ad {id: $adId, category: $adCategory})"
    create_user_country_relation_query =
        "MATCH (u:User), (c:Country) WHERE
            u.id = $userId AND
            c.name = $country CREATE (u)-[:LIVES_IN]->(c)"
    create_user_team_relation_query =
        "MATCH (u:User), (t:Team) WHERE
            u.id = $userId AND
            t.id = $teamId CREATE (u)-[:SUPPORTS]->(t)"
    create_team_ad_relation_query =
        "MATCH (t:Team), (a:Ad) WHERE
            t.id = $teamId AND
            a.id = $adId CREATE (t)-[:SHOWS]->(a)"
```

Listing 24: Advertisements graph -part 1

```
queries = [
    (create_user_query,
        ad_dataframe
        .select("userId")
        .distinct()),
    (create_country_query,
        ad_dataframe
        .select("country")
        .distinct()),
    (create_team_query,
        ad_dataframe
        .select("teamId", "team", "price")
        .distinct()),
    (create_ad_query,
        ad_dataframe
        .select("adId", "adCategory")
        .distinct()),
    (create_user_country_relation_query,
        ad_dataframe
        .select("userId", "country")),
    (create_user_team_relation_query,
        ad_dataframe
        .select("userId", "teamId")),
    (create_team_ad_relation_query,
        ad_dataframe
        .select("teamId", "adId"))
]

with driver.session() as session:
    for query, data in queries:
        for row in data.collect():
            session.run(query, **row.asDict())
```

Listing 25: advertisements graph -part 2

Filter that was applied in this case was top teams (based on users) and country they are in. With this information, query can be extended in order to see what advertisements are popular in top teams.

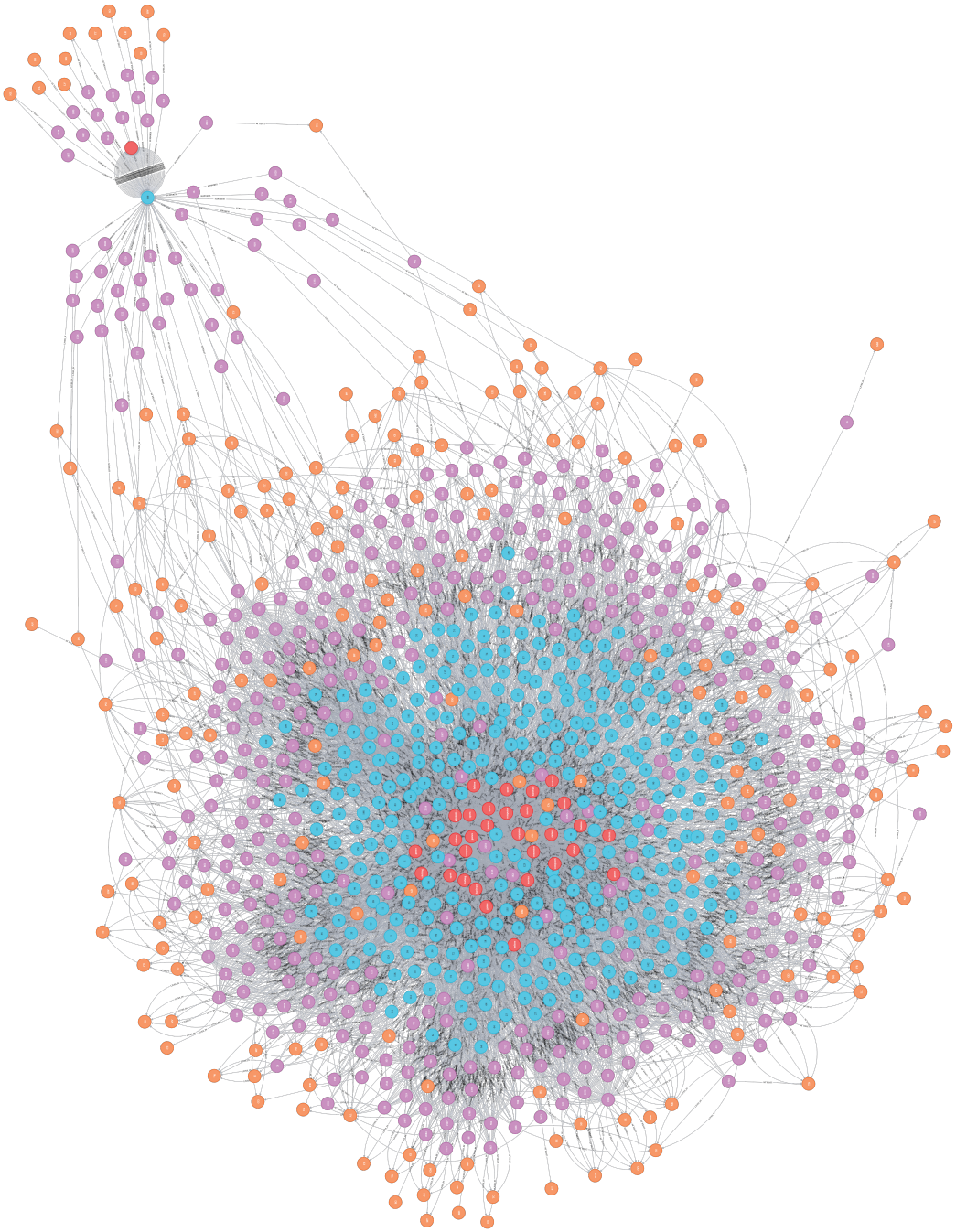

Figure 48: Graph 4

```
MATCH (n)
WHERE (n)-[:LIVES_IN|:SUPPORTS]->()
WITH n, SIZE((n)<-[:LIVES_IN|:SUPPORTS]-()) AS incomingCount
ORDER BY incomingCount DESC
LIMIT 10
MATCH (n)-[rel]->(m)
RETURN n, rel, m
```

Listing 26: Cypher filter 4

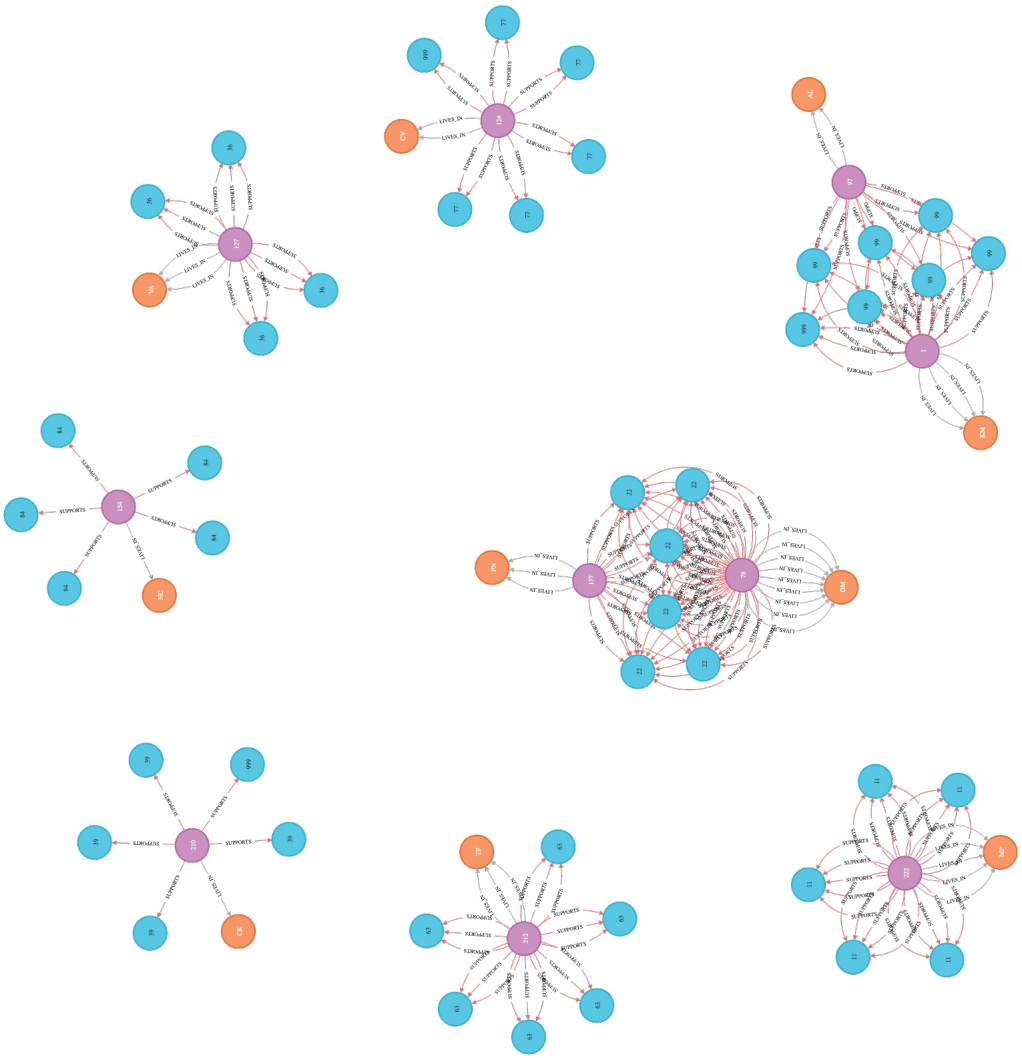

Figure 49: Graph filtered 4

7 Data ethics

When working with data, we need to keep in mind that it originated from somewhere or from someone. In our case, we would have two sensitive attributes, country and Twitter name. User could be harassed on their personal twitter based on the game they have played.

Our example above is a minor thing, compared to bigger data exploits that have happened. One of the biggest is Cambridge analytica (Berghel 2018), where data from Facebook was used in order to effect election.

Further discussion can be found in 9.7.

8 Conclusion

We interact with big data in our daily bases, as users or as researchers. Simple example would be IoT in the office. As we come to the office, thermostat is set, badge is scanned, parking spot is taken, etc... this is all of the data that we can use. We can measure when do users come in most often and set thermostat at that time for example.

With data we got, connections were made between users and their game scores. It would be better if data representation would be a bit larger. With larger dataset we could measure performance per quarter, do users tend to communicate more or less, are there users who often message each other, etc... At the moment data seems to be a bit of a stopping point not just due to size but also quality.

At the end of the day, a representation of big data was accomplished. Core concepts were introduced, developed and showcased.

Data is the new oil, is said by many people. It seems to be true. Data production is not stopping, we are generating more and more data daily. Now we need to ask ourselves, what to do with the data?

Whole project can be found on GitHub (Zver n.d.) with instructions on how to reproduce it.

9 Appendixes

9.1 A0

Unix time measures time passed since January 1, 1970 (00:00:00) UTC (Ritchie and Thompson 1978). With it, we have a ground truth, meaning that regardless of time zone, we can always calculate correct time.

Since we have players all around the globe, the time was converted into Unix time. Another problem that was solved was transition from Pandas date time into PySpark date time. Originally, transformation was not possible, but when changed to Unix time, data type is integer, therefore there were no problems.

Unix time has one drawback. It is called bug 2038. Originally the code was written for 32 bit programs, meaning that anything after 2038 (or before 1970) is not representable. Now days the computers are mostly 64 bit, therefore problem has been avoided. In order to maximise the compatibility, the team end date was set to 2038 (last possible date) instead of year 9999.

Note that change was only needed for flamingo data. Chat data seems to already be in Unix time, therefore noting was needed there.

```
def convert_date_time_to_unix(data_frame):
    data_frame = pd.to_datetime(data_frame, format='%Y-%m-%d %H:%M:%S')
    data_frame = data_frame.astype(int) // 10**9
    return data_frame

def convert_date_to_unix(data_frame):
    data_frame = pd.to_datetime(data_frame, format='%Y-%m-%d %H:%M:%S')
    data_frame = data_frame.astype(int) // 10**9
    return data_frame

data_frames = ["ad_clicks_pd", "buy_clicks_pd", "game_clicks_pd",
               "level_events_pd", "team_assignments_pd",
               "team_pd", "user_session_pd", "users_pd"]
for df_name in data_frames:
    if df_name == "team_pd":
        data_loaded[df_name]["teamEndTime"] =
            data_loaded[df_name]["teamEndTime"]
            .replace('9999-12-31 23:59:59', "2038-01-19 03:14:07")
        data_loaded[df_name]["teamEndTime"] =
            convert_date_time_to_unix(data_loaded[df_name]["teamEndTime"])
        data_loaded[df_name]["teamCreationTime"] =
            convert_date_time_to_unix(data_loaded[df_name]["teamCreationTime"])
    else:
        data_loaded[df_name]["timestamp"] =
            convert_date_time_to_unix(data_loaded[df_name]["timestamp"])

data_loaded["users_pd"]["dob"] =
        convert_date_time_to_unix(data_loaded["users_pd"]["dob"])
```

Listing 27: Transform date to Unix time

9.2 A1

Docker is a containerisation system (Docker 2020). It is similar to virtualisation, but it is more efficient due to sharing of resources (especially by sharing kernel).

In our case, we have used docker-compose, which is markup language, in order to define our 4 environments. All of the environments were then executed at the same time.

Note that there was a limit set for RAM. Developer system was lacking resources, therefore it needed to be capped at around 3-4GB of ram per container.

9.3 A2

First we need to split the data. Split is done by training and testing. In some cases, we add third step, evaluation. More sophisticated techniques could be used (example: k-fold)

Next we clean training data. In this case we make sure that there aren't any outliers, missing data or anything that could represent a problem. EDA cold help us in this case to confirm if the data looks alright. Testing data is not touched (or analysed). Data from real world is dirty therefore we need to know how it would preform against it. Another reason is bias. If we interact with the data, we could introduce human factor (bias) to it.

Now we can train the model. Training data is passed to the model (decision tree or SVM in our case) with the attribute that we want to predict (example: is a person sick or not).

At the end we need to evaluate the model against testing (raw) data. It is common to see confusion matrix, f1 score, accuracy, etc... since only one result is not the best representation of the model. The best practice is evaluation of the model over time which can be only done once model is deployed.

9.4 A3

Steps of building the model are similar to classification. The main difference is that we are using clustering algorithms which have different parameters. One of the parameters is number of classes. This depends on the model and data, we can say that we have 3 classes at all the times. Other approach is for model to figure out how many classes should be on its own.

9.5 A4

To setup Neo4j, we are using Docker in order to have the local environment. Docker compose code bellow showcases setup for one database, the rest (3) of them have same setup (only name is different).

```yaml
version: '3'
name: neo4j_nodes
services:
  neo4j_0:
    image: neo4j:3.5
    restart: unless-stopped
    ports:
      - 7470:7474
      - 7680:7687
    volumes:
      - ./neo4j_0/conf:/conf
      - ./neo4j_0/data:/data
      - ./neo4j_0/import:/import
      - ./neo4j_0/logs:/logs
      - ./neo4j_0/plugins:/plugins
    environment:
      - NEO4J_AUTH=neo4j/password
      # Raise memory limits
      - NEO4J_dbms_memory_pagecache_size=3G
      - NEO4J_dbms.memory.heap.initial_size=3G
      - NEO4J_dbms_memory_heap_max__size=3G
```

Listing 28: Example Neo4j docker compose

In order for data to be inserted into the database, get_creds function is called. It creates a connection that is passed back. Every database has its own port, therefore this needs to be specified with the input.

This setup is not recommended for the production (since it is not secure), but it works for demonstration purposes.

```python
def get_creds(port_end):
    return f"bolt://localhost:768{port_end}", "neo4j", "password"
```

Listing 29: Credentials

9.6 A5

Two additional data frames are created in order to fill databases. One is created in order to be a collection of all the data (Mega frame) and another focuses on only advertisement data (Ads frame).

Code bellow showcases how PySpark SQL can be used in order to join the data, filter and clean together.

```
mega_dataframe = data_users
  .select("userId","nick","twitter",
        from_unixtime("dob", "yyyy-MM-dd").alias("dob"),"country")\
  .join(data_user_session
        .select("userId","teamId","platformType"), ["userId"], 'right')\
  .join(data_team
        .select("teamId","name","strength"), ["teamId"], 'left')\
  .join(data_ad_clicks
        .select("userId","adId","adCategory"), ["userId"], 'full')\
  .dropDuplicates()\
  .fillna({"strength":0})

mega_dataframe = mega_dataframe.select(
*[ F.when(F.col(column).isNull(),'')
.otherwise(F.col(column))
.alias(column)
for column in mega_dataframe.columns])
```

Listing 30: Mega frame

```
ad_dataframe = data_ad_clicks
  .select("teamId","userId","adId","adCategory")\
  .join(data_buy_clicks.select("team","userId","price"), ["userId"], 'full')\
  .join(data_users.select("userId","country"), ["userId"], 'full')\
  .dropDuplicates()\
  .fillna({
            "teamId":0,
            "adId":999,
            "team":999,
            "price":0
            })

ad_dataframe = ad_dataframe.select(
*[ F.when(F.col(column).isNull(),'')
.otherwise(F.col(column))
.alias(column)
for column in ad_dataframe.columns])
```

Listing 31: Ads frame

9.7 A6

Due to situations listed above, we need to consider key items when handling the data:

- privacy
  - Users need to be able to see and remove all their data upon request. GDPR (Voigt and Von dem Bussche 2017) is a great example of how users data can be protected
- honesty
  - Use of cookies or tracking the user should be easily identifiable and configurable (option to turn off) (Englehardt et al. 2015)
  - Data breaches are not 100% avoidable, therefore if anything happens it needs to be transparent with the users (Sen and Borle 2015)
  - If users data is collected (sold), they need to be aware of it
- security
  - sensitive data needs to be encrypted in case of the data breach (Smid and Branstad 1988)

  - when dealing with sensitive data, employees need to have proper clearance and security checks done. The best approach would be to use least privilege rule, meaning that data would be on need to know bases. Since this could represent a problem for development environment, that can have a copy of production but with fake data instead

- moral
  - when working with data, we need to know when something could hurt someone. As mentioned above, the twitter data can be used for malicious purpose. Due to situations like this, we need to ask ourselves, what does it bring to the company? It can help with advertisements but it can also help with harassment. In this case, if we need Twitter data, we can store it but not make it visible to other players.

9.8 A7

Lambda is part of hybrid architecture, that means it combines batch and real time processing. Since both processes are used, it is common to split them apart. We would have batch processing for bigger files and real time (or speed) for time intense applications (Lin 2017).

One real world example would be airline customers. We can split them into priority and none priority. Both achieve the same goal, just one type of customers will be served first.

In our example, we could use Lambda architecture. For example, game data (how players are moving) needs to be processed in real time. But chat data can be processed as a batch. Reason for this is chat data doesn't provide as much value to us as game data.

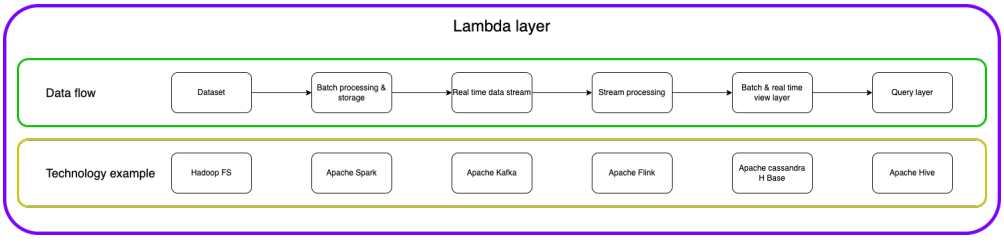

Figure 50: Lambda

9.9 A8

Kappa architecture is similar to Lambda but it has one big difference: it excludes batch processing. Arguably, it can be called another version of real time processing. If there is a batch of files send to it, they are handled as a stream of data (Feick, Kleer, and Kohn 2018).

Since Kappa is similar to real time processing, lets reuse roller coaster example. In batch processing, we would wait for 20 people to fill the ride before operator would start the ride. In Kappa, people would still queue for the ride but as soon as the ride is available all of the people who are there would be ready to ride. In other words, our roller coaster would operate real time with the goal of less stopping regardless how many people are on the ride.

As mentioned with the real time processing, this is something that could be used with flamingo data. For example, instead of waiting for conversation between players to finish, they can directly be processed.

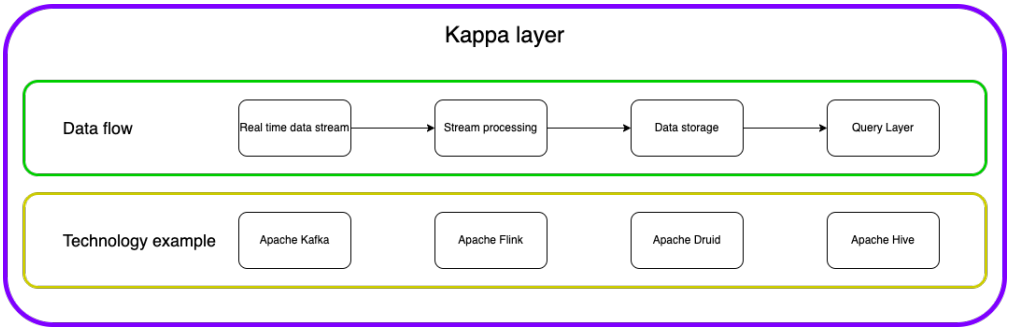

Figure 51: Kappa

10 References

References

Amazon (n.d.). *Data processing*. Last accessed 18 May 2023. URL: https://docs.aws.amazon.com/whitepapers/latest/data-warehousing-on-aws/data-processing.html.

Anuradha, J et al. (2015). "A brief introduction on Big Data 5Vs characteristics and Hadoop technology". In: *Procedia computer science* 48, pp. 319–324.

Berghel, Hal (2018). "Malice domestic: The Cambridge analytica dystopia". In: *Computer* 51.05, pp. 84–89.

Cappa, Francesco et al. (2021). "Big data for creating and capturing value in the digitalized environment: unpacking the effects of volume, variety, and veracity on firm performance". In: *Journal of Product Innovation Management* 38.1, pp. 49–67.

Casado, Rubén and Muhammad Younas (2015). "Emerging trends and technologies in big data processing". In: *Concurrency and Computation: Practice and Experience* 27.8, pp. 2078–2091.

Docker, Inc (2020). "Docker". In: *línea].[Junio de 2017]. Disponible en: https://www.docker. com/what-docker*.

Englehardt, Steven et al. (2015). "Cookies that give you away: The surveillance implications of web tracking". In: *Proceedings of the 24th International Conference on World Wide Web*, pp. 289–299.

Feick, Martin, Niko Kleer, and Marek Kohn (2018). "Fundamentals of real-time data processing architectures lambda and kappa". In: *SKILL 2018-Studierendenkonferenz Informatik*.

Friedl, Mark A and Carla E Brodley (1997). "Decision tree classification of land cover from remotely sensed data". In: *Remote sensing of environment* 61.3, pp. 399–409.

Hartigan, John A and Manchek A Wong (1979). "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the royal statistical society. series c (applied statistics)* 28.1, pp. 100–108.

Hota, Soma et al. (2018). "Big data analysis on youtube using hadoop and mapreduce". In: *International Journal of Computer Engineering In Research Trends* 5, pp. 98–104.

Katal, Avita, Mohammad Wazid, and Rayan H Goudar (2013). "Big data: issues, challenges, tools and good practices". In: *2013 Sixth international conference on contemporary computing (IC3)*. IEEE, pp. 404–409.

Kotsiantis, Sotiris B, Ioannis D Zaharakis, and Panayiotis E Pintelas (2006). "Machine learning: a review of classification and combining techniques". In: *Artificial Intelligence Review* 26.3, pp. 159–190.

Lewis, Seth C, Rodrigo Zamith, and Alfred Hermida (2013). "Content analysis in an era of big data: A hybrid approach to computational and manual methods". In: *Journal of broadcasting & electronic media* 57.1, pp. 34–52.

Likas, Aristidis, Nikos Vlassis, and Jakob J Verbeek (2003). "The global k-means clustering algorithm". In: *Pattern recognition* 36.2, pp. 451–461.

Lin, Jimmy (2017). "The lambda and the kappa". In: *IEEE Internet Computing* 21.05, pp. 60–66.

Miller, Justin J (2013). "Graph database applications and concepts with Neo4j". In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*. Vol. 2324. 36.

Reynolds, Douglas A et al. (2009). "Gaussian mixture models." In: *Encyclopedia of biometrics* 741.659-663.

Ritchie, Dennis M and Ken Thompson (1978). "The UNIX time-sharing system". In: *Bell System Technical Journal* 57.6, pp. 1905–1929.

Rocketloop (n.d.). *Clustering with Machine Learning*. Last accessed 18 May 2023. URL: https://rocketloop.de/en/blog/clustering-machine-learning-comprehensive-guide/.

Rubin, Victoria and Tatiana Lukoianova (2013). "Veracity roadmap: Is big data objective, truthful and credible?" In: *Advances in Classification Research Online* 24.1, p. 4.

S-cubed (n.d.). *Statistics and Machine Learning*. Last accessed 18 May 2023. URL: https://www.s-cubed-global.com/biometrics/statistics-and-machine-learning.

Sagiroglu, Seref and Duygu Sinanc (2013). "Big data: A review". In: *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, pp. 42–47.

Sen, Ravi and Sharad Borle (2015). "Estimating the contextual risk of data breach: An empirical approach". In: *Journal of Management Information Systems* 32.2, pp. 314–341.

Smid, Miles E and Dennis K Branstad (1988). "Data encryption standard: past and future". In: *Proceedings of the IEEE* 76.5, pp. 550–559.

Storage, Remote Backup–Cloud (2012). "Data Backup Options". In.

Tukey, John W et al. (1977). *Exploratory data analysis*. Vol. 2. Reading, MA.

Upgrad (n.d.). *Decision Tree in Machine Learning*. Last accessed 18 May 2023. URL: https://www.upgrad.com/blog/decision-tree-in-machine-learning/.

Voigt, Paul and Axel Von dem Bussche (2017). "The eu general data protection regulation (gdpr)". In: *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10.3152676, pp. 10–5555.

Wang, Cong et al. (2010). "Toward publicly auditable secure cloud data storage services". In: *IEEE network* 24.4, pp. 19–24.

Wang, Haifeng and Dejin Hu (2005). "Comparison of SVM and LS-SVM for regression". In: *2005 International conference on neural networks and brain*. Vol. 1. IEEE, pp. 279–283.

Wu, Han et al. (2020). "A reactive batching strategy of apache kafka for reliable stream processing in real-time". In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, pp. 207–217.

Yu, Xiaojun and Qiaoyan Wen (2010). "A view about cloud data security from data life cycle". In: *2010 international conference on computational intelligence and software engineering*. IEEE, pp. 1–4.

Zver, Zan (n.d.). *GitHub code*. Last accessed 18 May 2023. URL: https://github.com/ZanZver/BigDataManagement.