



UNIVERSITY OF PELOPONNISOS

SCHOOL OF ENGINEERS

DEPARTMENT OF ELECTRICAL AND

COMPUTER ENGINEERS

MASTER THESIS

**<< GRAPH DATABASES AND THEIR APPLICATION
TO FINANCIAL PROBLEMS >>**

Nikolaos Livieratos

Registration Number: 9093202101014

SUPERVISOR: Vasilios T. Tampakas, Professor

PATRA 2023

It was approved by the three-member examination committee

Patras, Date

EVALUATION COMMITTEE

1. Vasilios Tampakas
2. Ioannis Tsaknaris
3. Sotirios Christodoulou

I certify that I am the author of this paper and that any help I had in its preparation of her is fully recognized and referred to in the work. I have also cited any sources from which I have used data, ideas or words, whether they are quoted exactly or paraphrased. Also I certify that this assignment was prepared by me personally specifically for this assignment. The approval of the thesis by the Department of Electrical and Computer Engineering University of Peloponnese does not necessarily imply acceptance of the views of the author part of the Department. This thesis is the intellectual property of the student Nikolaou Livieratou who prepared it. In the framework of the open access policy the author/creator assigns to the University Peloponnese, non-exclusive license to use the right to reproduce, adapt, public lending, presenting to the public and digitally disseminating them internationally, in electronic form and in any medium, for teaching and research purposes, free of charge and for the whole year duration of intellectual property rights. Open access to the full text for study and reading does not imply in any way the granting of intellectual property rights author/creator nor allows to reproduce, republish, copy, store, sell, commercial use, transmission, distribution, publication, performance, "downloading", "posting" (uploading), translation, modification in any

way, partial or summary of the work, without the express prior written consent of the author/creator. The author/creator retains the set of his moral and property rights.

Contents

Contents	3
Abstract.....	9
FIRST PART	11
Use Cases	11
1.Customer Experience with a 360-Degree View.....	11
1.1.1 Navigation Graph on Clients	17
1.1.2 Single-user View Prediction (SU)	17
1.1.3 Cross-user View Prediction (CU).....	18
1.1.4 Performance evaluation	19
2. Compliance Management	21
2.1 Use Case.....	21
2.1.1 Artifact 1: Building Knowledge Graph-Based on User Interaction.....	25
2.1.2 Artifact 2: Link Prediction with Neo4J Graph Machine Learning Algorithms	26
2.1.3 Executive summary of node embedding algorithm Node2Vec	30
3. Data Lineage & Metadata Management	32
3.1 Use Case.....	33
4. Financial Crime Types.....	41
4.1 STOCKS AND SECURITIES INVESTMENT FRAUD.....	41
4.2 FRAUD DETECTION AND ANTI-MONEY LAUNDERING (AML).....	42
4.3 SIM-SWAPPING AND PHISHING	45
4.4 ROMANCE FRAUD	46
4.5 RANSOMWARE.....	47

4.6 DEEPFAKES AND GPT-2	47
4.6 Use Case.....	48
5. Recommender Systems & Conversational AI	55
5.1 Use Case.....	56
5.1.1 CRS method.....	56
5.1.2 EAR method.....	60
5.1.3 CPR method.....	62
SECOND PART	69
6. Brief historical retrospective of databases	69
6.1 SQL Databases.....	70
6.2 NoSQL Databases	71
6.2.1 Key Value Stores	71
6.2.2 Document Store	72
6.2.3 Graph Database.....	72
6.2.4 Column Oriented Databases	72
6.2.5 Object Oriented Databases.....	73
6.3 Evaluation of differences between SQL and NoSQL Databases.....	73
6.3.A Scalability and performance.....	73
6.3.B Flexibility	74
6.3.C Query language	74
6.3.D Security	75
6.3.E Data management - Storage and Access	75
6.4 Graphs	76
6.4.1 <i>Types of Graph Algorithms</i>	78
THIRD PART	82
7. Platforms	82
7.1 Selecting Platform.....	82

7.2 Apache Spark.....	83
7.2.1 Spark Graph Evolution	84
7.3 Neo4j Graph Platform.....	84
8. Forming – Operating – Evaluating a methodology for AML	85
8.1 Choosing Data Set.....	85
8.2 Neo4j.....	87
8.2.1 Creating a graph data base in Neo4j	87
8.2.2 Adding files.....	88
8.2.3 Forming node alerts	89
8.2.4 Forming relationships between transactions and accounts	90
8.2.5 Forming relationships between transactions and alerts	91
8.2.6 First AML Query	92
8.2.7 Second AML Query	93
8.2.8 Third AML Query	95
8.2.9 Projection of graph in the graph catalogue of gds.library.....	96
8.2.10 DFS ALGORITHM	97
8.2.11 BFS ALGORITHM	99
8.2.12 DEGREE CENTALITY ALGORITHM	100
8.2.13 MACHINE LEARNING MODELS	100
8.2.13.A LOGISTIC REGRESSION MODEL DESCRIPTION	100
8.2.13.B RANDOM FORESTS MODEL DESCRIPTION	101
8.2.14 APPLICATION	102
8.2.14. A LOGISTIC REGRESSION MODEL	102
8.2.14.B RANDOM FORESTS MODEL.....	106
8.3 Apache Spark.....	109
8.3.1 PySpark and SparkSession.....	109
8.3.2 Forming a graph.....	111

8.3.3 Visualization of the graph.....	113
8.3.4 DFS ALGORITHM	115
8.3.5 BFS ALGORITHM	116
8.3.6 DEGREE CENTRALITY ALGORITHM.....	119
8.4 MACHINE LEARNING	120
8.4.1 LOGISTIC REGRESSION MODEL	120
8.4.2 RANDOM FORESTS MODEL	125
9. Conclusions.....	130
Bibliography	132

Figure 1 Segments,Tiles,Viewports and View	14
Figure 2 view transitions.....	15
Figure 3 Navigation Graph in Video Server for Cross-User View prediction	16
Figure 4 Navigation Graph in Clients for Single-User ViewPrediction	17
Figure 5 View Prediction with the Navigation Graph	19
Figure 6 CDFs of View Prediction Precision	20
Figure 7 Data Model in Neo4j	24
Figure 8 Frequencies of Usr Interactions	25
Figure 9 Cypher Query <<who knows who>>	26
Figure 10 Process Diagram for Antifact 2	27
Figure 11 Labelled Property Graph Visualized with neo4j Bloom(left) and Predicted Relationships with Link Prediction(right). Screenshot from neo4j Bloom of the USER-KNOWS-USER query.....	28
Figure 12 BFS and DFS search strategies from node u (k=3)	30
Figure 13 Colt conceptual system architecture	35
Figure 14 data flows from producer to a consumer	35

Figure 15 A data flow is a complex structure containing concepts and their treatments as well boundaries.....	36
Figure 16 Example network containing 9 nodes.....	36
Figure 17 Colt UI visualize and explore a network	37
Figure 18 Network tracing for System Z (with grayed/dashed segments determined to be irrelevant for the trace).....	38
Figure 19 Downstream trace result for System A.....	39
Figure 20 System W fails validation because p1 is only subset of p. System X passes validation.....	40
Figure 21 Diagram of the illegal process	42
Figure 22 Diagram of ML.....	45
Figure 23 Diagram of the SIM-Swap process.....	46
Figure 24 Diagram of romance fraud.....	47
Figure 25 System training overview	50
Figure 26 Schematic of graph(node) embedding.....	51
Figure 27 ROC curves and corresponding AUCs for all models considered.....	53
Figure 28 UMAP visualization	55
Figure 29 The conversational recommender system overview.....	57
Figure 30 The structure of the proposed Conversational Recommender System.....	58
Figure 31 The workflow of our multi-round conversational recommendation scenario	61
Figure 32 An illustration of interactive path reasoning in CPR	63
Figure 33 CPR framework overview	66
Figure 34 Success Rate * of compared methods at different turns on LastFM and Yelp (RQ1).	68
Figure 35 Graph Database on the market today.....	77
Figure 36 A high-level view of a typical graph compute engine deployment.....	78
Figure 37 Pathfinding and search algorithms	79
Figure 38 Representative centrality algorithms and the type of questions they answer	80
Figure 39 Representative community algorithms	81
Figure 40 Spark is an open-source distributed and general-purpose cluster computing framework. It includes several modules for various workloads	84

Figure 41 The Neo4j Graph Platform is built around a native graph database that supports transactional applications and graph analytics.....	85
Figure 42 Creating a database and adding APOC and GDS libraries.....	87
Figure 43 Adding csv files.....	88
Figure 44 Creating vertex alerts.....	89
Figure 45 Creating vertex accounts.....	89
Figure 46 Creating relationships transactions-accounts.....	90
Figure 47 Creating relationships transactions-alerts.....	91
Figure 48 Schema visualization of the imported data.....	91
Figure 49 First AML Query.....	92
Figure 50 Results of the First AML Query.....	93
Figure 51 Second AML Query.....	94
Figure 52 Results of Second AML Query.....	94
Figure 53 Third AML Query.....	95
Figure 54 Results of Third AML Query.....	96
Figure 55 Graph projection.....	96
Figure 56 Results of DFS Algorithm.....	98
Figure 57 Results of BFS Algorithm.....	99
Figure 58 Results of Degree Centrality Algorithm.....	100
Figure 59 Logistic Regression Model.....	103
Figure 60 Results for f1_score,accuracy_score,precision_score,recall_score and the ROC curve.....	104
Figure 61 Bar Chart for metrics.....	104
Figure 62 Random Forest Model.....	107
Figure 63 Results for f1_score,accuracy_score,precision_score,recall_score and ROC curve area.....	107
Figure 64 Bar Chart for metrics.....	108
Figure 65 Starting SparkSession.....	109
Figure 66 Import modules, define variables and checking in the HDFS directory.....	110
Figure 67 Constructing a graph.....	112
Figure 68 Visualization of 0,0004 of nodes and edges of the graph.....	114
Figure 69 DFS Algorithm in Apache Spark.....	115
Figure 70 Results of DFS Algorithm.....	115
Figure 71 BFS Algorithm and results.....	116

Figure 72 BFS Algorithm with minimum,maximum and average	117
Figure 73 Results of BFS Algorithm	118
Figure 74 DEGREE CENTRALITY ALGORITHM and results	119
Figure 75 Logistic Regression Model and metrics results.....	121
Figure 76 Bar Chart for metrics	123
Figure 77 Code for AUC-ROC.....	123
Figure 78 Result and graphical representation of AUC-ROC	124
Figure 79 Random Forests Model and metrics results.....	126
Figure 80 Bar Chart for metrics	128
Figure 81 Code for AUC-ROC.....	128
Figure 82 Result and graphical representation of AUC-ROC	129
Table 1 Selected Variables Based on Data Model1	24
Table 2 Train and Test Results of Link Prediction Model with neo4j Fast RP Algorithm.....	29
Table 3 ROC AUC and average precision(AP) results on the test data for all methods under consideration	53
Table 4 Success Rate and Average Turn	67

Abstract

Over the last 20 years Artificial Intelligence (AI) techniques have being accomplished a significant deployment and using in many aspects of human activity.

By giving a definition we could say that Artificial intelligence (AI) systems are machine-based systems with varying levels of autonomy that can, for a given set of human-defined objectives, make predictions, recommendations or decisions. AI techniques are increasingly using massive amounts of alternative data sources and data analytics referred to as 'big data'. Such data feed machine learning (ML) models which use these kind of data, to learn and improve predictability and performance automatically through experience, without being programmed to do so by humans.

The usage of such technologies offer competitive advantages for firms like improving firms' efficiency through cost reduction, enhancing productivity which drives to higher profitability, improving the quality of services and products offered to consumers.

In this thesis we are going to focus on the AI application and especially, on analytics with graphs in finance.

The financial sector is one of the most prominent, significant and prone to technological advances which increased information exchange and the necessity to assess and evaluate these large amount of data effectively.

During this master Thesis we are going to see graph use cases like Customer Experience with a 360-Degree View, Compliance Management, Data Lineage & Metadata Management, Financial Crime Types and Recommender Systems & Conversational AI.

(1)

In a next level we will dive into, how all these flooding of financial news sources reporting on companies, markets, currencies and stocks can be, conveniently, stored in a graph which can be used to drive new insights through answering complex queries using high level declarative languages, what tools are utilized, making it possible to mine data in structured representational forms for strategic decision making.

And lastly we will implement a graph and apply an evaluation in order to extract graph statistics and performing machine learning models for binary classification.

FIRST PART

Use Cases

1. Customer Experience with a 360-Degree View

One of the main goals of companies, especially in the financial sector, is to become aware of their customers, the relationships their customers hold with each other, products, relationships between different products and much more to provide customers with what they want in a accurate, effective and personalized way.

Customer expectations are rising. In today's competitive landscape, especially in the financial sector, customer service has become a significant differentiator. Companies are striving to meet these heightened expectations as part of their primary goals.

The massive use of technology have made possible the collection of large sum of information about customers, including:

- Master data — name, age, gender, address
- Transactions — purchases, types of items bought, purchase times
- Big data — call center logs, traffic lines, web click streams, SNS activities
- Predictions — classification, taste signatures (often created by different models)

(2)

All of these efforts require a 360-degree view of customers, and that's something most financial services firms simply don't have. Data tends to be locked away in silos across the organization without any way to leverage the connections between data and innovate based on those connections. Harnessing the power of connected data (i.e., data relationships) is essential to sustainable competitive advantage in today's ever-more-connected, ever-more competitive world.

But these silos of data can be logically integrated on graphs and the graph users can simply view all of the surrounding information of one entity (the customer). With graphs, decision makers can gain a more comprehensive view of their customers—the relationships the customers hold with each other, the relationships between all the purchased products, and more. Then, graph users can run algorithms to discover even more fine-grained detail about the customer.

By inspecting all these knowledge about one particular customer is important to realize the customer and to execute customer`s total analysis, to discover which predictions (usually created through machine learning such as Enhanced Customer Relations) are authentic and why.

(3) (4)

1.1 Use Case

The e-Commerce has become one of the most influential and substantial factor for the survival and prosperity of business in the contemporary global surrounding. As online competition becomes fiercer over time, online enterprises face increasingly more sophisticated e-Customers. On line enterprises have to meet e-Customers' expectations and to cause positive online shopping experience and satisfaction. Therefore, the design and construction of a 360-degree view of customers and their behavior must be prioritized, since "companies that make extensive use of customer analytics are more likely to have a considerable impact on corporate performance, outperforming its competitors".

The part '360-degree' denotes 'complete' or 'all-around', whilst the part 'view' refers to the ability to see something from a particular place or angle. Therefore, the term '360-degree view of a customer' suggests the ability to use the best available and most relevant information about each customer to enhance sales, marketing, and servicing decisions.

The technological capacity of using streaming 360-degree videos in the customer virtual navigation, during e-shopping, will open a new era in the field of customer experience. However, streaming these videos requires larger bandwidth and less latency than what is found in conventional video streaming systems. Rate adaptation of tiled videos and view prediction techniques are used to solve this problem.

A potential solution to this challenge could be use of the Navigation Graph, which models viewing behaviors in the temporal (segments) and the spatial (tiles) domains to perform the rate adaptation of tiled media associated with the view prediction. The Navigation Graph allows online enterprises to perform view prediction more easily by sharing the viewing model.

The server-client model is used. The server stores the video data as segments with multiple quality choices and uses the media presentation description file to provide video information to the clients. As the clients request the video data, the server provides them with information from the MPD file. The clients can then also request the video segments with the proper quality to allow for continuous play. The server can send the segments as soon as requests are received. The clients are responsible for rate adaptation. This allows the video to be viewed on various types of devices without changing the video. Initially, 360-degree video streaming services streamed the whole video to the viewers and the viewers extracted the viewports from the video they wanted.

The Navigation Graph is a graph that consists of Vertices (set of tiles in a segment) and Edges (transition probabilities between vertices).

360-degree videos are created by stitching together multiple videos taken by multiple outward looking cameras that capture the whole surrounding sphere. The sphere is then projected into the 2D plane to make the 360-degree video easier to process and store. Therefore, the original videos are cut into smaller videos, called tiles, and encoded independently with different qualities (representations). The encoded and stored video data are called segments, and they are delivered to clients upon request.

A cross-user learning based system (CLS) [20] gathers the users' fixation data on the server and gives more weight to the tiles with more fixations. It can also utilize prior viewers' fixation data to optimize the weighting coefficients. It also performs the clustering of the fixations to classify clients and chooses the weighting coefficients that should be used for each client. A viewport prediction is critical for lowering the required bandwidth because prediction error leads to waste bandwidth. While the viewport predictions rely solely on viewport data from the viewer him/herself, a cross-user viewport prediction could further improve the prediction accuracy since the viewport trajectory of multiple viewers could be correlated.

The Navigation Graph concept can be useful for both single-user view prediction and cross-user view prediction. Moreover, the Navigation Graph can be used to encode important trajectories in videos using a powerful computer on the video server side and can then share this information with clients.

The duration D of the segments is the basic unit of rate adaptation and is usually between 1 to 15 seconds long.

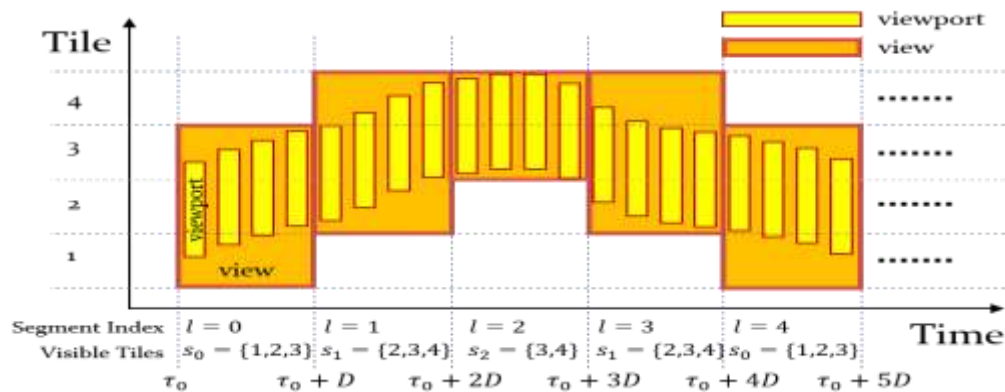


Figure 1 Segments, Tiles, Viewports and View

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

Figure 1 shows the segments, tiles, viewports and views. The tiles cut the video into smaller spatial regions, and then, the viewport can span multiple tiles within a video frame. A video segment consists of multiple consecutive frames, and viewport can change every frame. We define a "view" as the union of all visible tiles within a segment duration D . Viewport usually means the visible part of the video, but the view is defined as a set of tiles in specific segment that is used for recovering viewports in the video segment. Since the viewers move continuously, the viewport can change within the duration of a segment.

We define the view v as a tuple consisting of a segment index l and a set of visible tiles s , where s is the union of all visible tiles in the duration of a segment. The temporal variation of the view must be considered to perform the future view prediction and manage the playback buffer. Therefore, a model that describes the relationship between the views in series of segments is required. We introduce the Navigation Graph G , which is a directed graph describing view transitions.

$$G = (V, E)$$

The vertices are defined as

$$V = \{v \mid v = (l, s), l \in \{1, 2, \dots, L\} \text{ and } s \in S\},$$

Where l is a segment index, L is the number of segments in a video and S is the set of s configuring the views that clients have seen at least once. E is a set of edges connecting the vertices, where at least one transition happens.

$$E = \{(v_i, v_j) \mid v_i, v_j \in V, w(v_i, v_j) = p(v_j \mid v_i), i, j \in 1, \dots, N\},$$

Where $w(v_i, v_j)$ is a weight function. We also define the matrix E^{\wedge} which consists of the transition probabilities from one vertex to other vertices connected by edges in E

$$E^{\wedge} = R^{N \times N} = \begin{pmatrix} p(v_1|v_1), p(v_1|v_2), & \dots & , p(v_1|v_n) \\ \vdots & \ddots & \vdots \\ p(v_n|v_1), p(v_n|v_2), & \dots & , p(v_n|v_n) \end{pmatrix}$$

Where N is the number of vertices. The maximum number of vertices is $2 T \times L$, where the T is the number of tiles and L denotes the number of segments. However, N counts only the vertices that have been visited at least once. The Navigation Graph is expended whenever it encounters a new view the and the number of vertices N will range between L and $C \times L$ depending on how various the viewing patterns of clients are, where C is the number of clients.

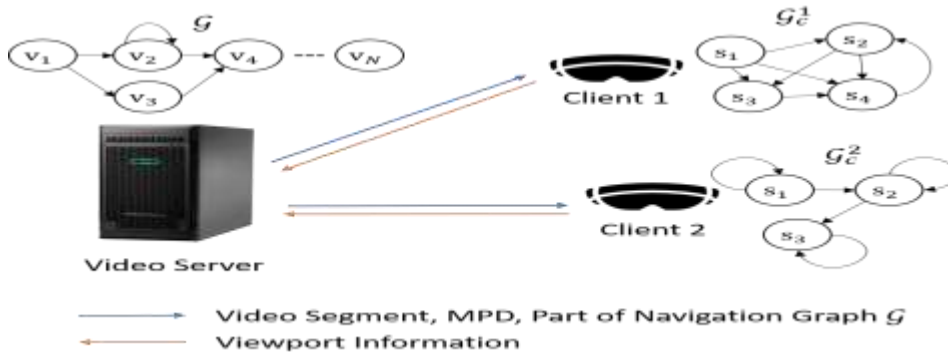


Figure 2 view transitions

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

As we can see in Figure 2 each client also has a Navigation Graph generated from their own view information. Therefore, the client's Navigation Graph learns the client's

distinctive viewing patterns and the Navigation Graph on the server learns the viewing patterns of all of the clients watching the same video.

Navigation Graph on Video Server

The video server can generate a view transition model by collecting multiple viewers' viewport feedback information.

The server compares the current view v_c received from the client with the vertices in a set V . If the same vertex exists in V that is the same as v_c , then the server only updates the edges E and the transition matrix E^{\wedge} which describes the transition probabilities. Otherwise, the Navigation Graph increases N by 1 and adds the vertex v_c into V as a v_N , and updates the edges E and the transition matrix E .

The transition probability in E^{\wedge} is updated as

$$p(v_c | v_p) = \text{number of clients moving their view from } v_p \text{ to } v_c / \text{number of clients visiting } v_p$$

Where the v_p is a prior vertex that the client had visited before she moved to the current vertex v_c .

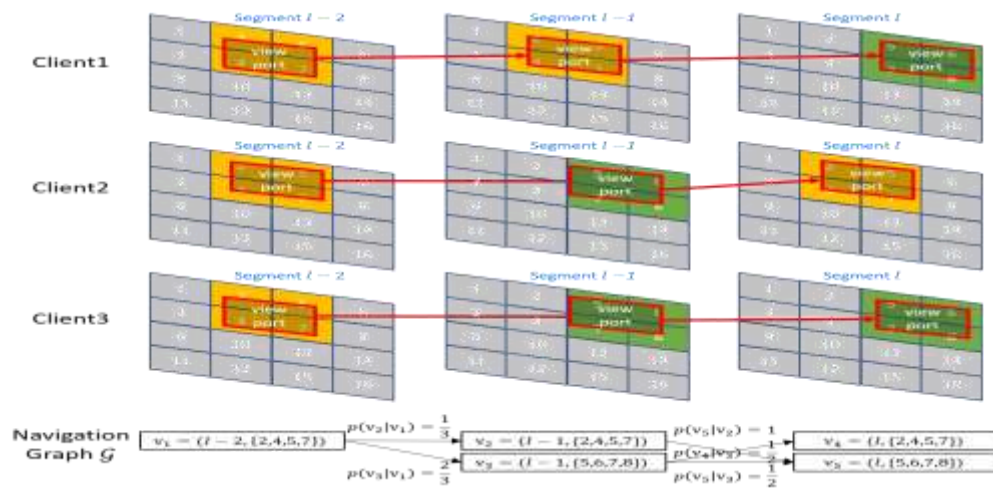


Figure 3 Navigation Graph in Video Server for Cross-User View prediction

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

Figure 3 shows a Navigation Graph made by three clients' view data.

1.1.1 Navigation Graph on Clients

The simplified Navigation Graph for clients is defined as $G_c = (S, E_c)$ where the set S consists of sets of visible tiles s that are defined in the previous section (Figure 1). All vertices $s_m \in S, m = 1, \dots, M$ are the vertices that a client has visited at least once. E_c consists of edges connecting all vertices in S and the transition probability matrix is defined as

$$E_c = R^{MXM} = \begin{pmatrix} p(s_1|s_1), p(s_1|s_2), & \dots & , p(s_1|s_m) \\ \vdots & \ddots & \vdots \\ p(s_m|s_1), p(s_m|s_2), & \dots & , p(s_m|s_m) \end{pmatrix}$$

Where M is the number of vertices and the difference between the original Navigation Graph G and the simplified version G_c is the configuration of the vertices. The vertices for the simplified Navigation Graph consist of the set of tiles S and do not include a segment index. E_c elements present the transition probabilities $p(s_c | s_p)$ from vertices s_p to vertices s_c , where

$p(s_c | s_p) = \text{number of transitions from } s_p \text{ to } s_c / \text{number of times visiting the set of visible tiles } s_p$

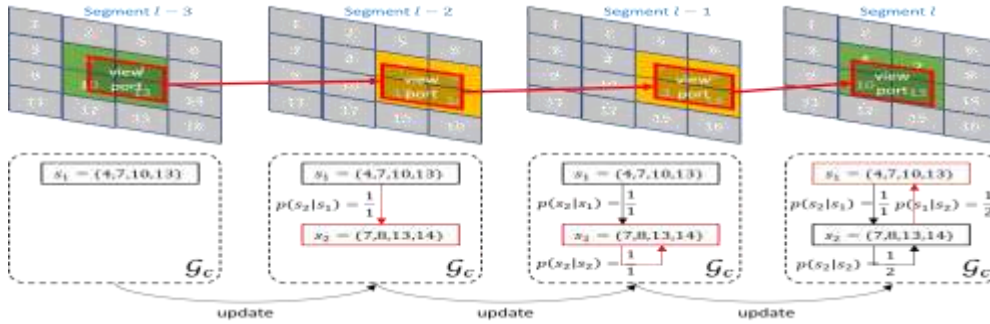


Figure 4 Navigation Graph in Clients for Single-User View Prediction

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

Figure 4 shows how the Navigation Graph is constructed by four consecutive video segments that a client has seen.

1.1.2 Single-user View Prediction (SU)

Clients can perform the prediction themselves using past view transition data encoded as a Navigation Graph G_c . The probability that a viewer will change his head position from the current set of visible tiles s_c to another set of visible tiles s_m for $1 \leq m \leq M$ is

the column vector of the E^c , which is defined as a vector $d_1 = R^{1 \times M}$ whose elements are $p(s_m | s_c)$, for $1 \leq m \leq M$. For example, E^c is updated after the video started, and the viewer is currently (segment 1) at the vertex s_1 . The first column of E^c is d_1 , which describes the probabilities that the viewer will have a set of tiles s_m , $m = 1, \dots, M$, in segment $l + 1$.

In general, we can define a k^{th} future transition probability as

$$d_k = E_c^{(k-1)} d_1$$

Since the vertices s_m consist of many tiles, the probability $p_{t,k}$ of needing a specific tile t in future segment k is given as

$$P_{t,k} = \sum d_k^m \quad \forall m, t \in s_m$$

Where the t is the index of tile and d_k^m indicates the m^{th} element of vector d_k .

We can generate a prediction matrix $P_s = R^{T \times K}$ that the user's current behavior is related to the video content, the CU will work better.

1.1.3 Cross-user View Prediction (CU)

The Navigation Graph in the media server is updated by all prior viewers' view transitions. The Navigation Graph provides the statistics for how many times other viewers move from the current view to the subsequent views. The probability that a viewer will change his view from the current view v_c to the next view v_n is the column vector of E^v , which is defined as $b_1 = R^{1 \times N}$, whose elements are $p(v_n | v_c)$, for $1 \leq n \leq N$. We can also define a k^{th} transition vector b_k , which represents the transition probabilities from the current vertex to the vertices in the future segment $l + k$; the $k = 1, \dots, K$ transition happens after the current segment l as follows:

$$b_k = E^v^{(k-1)} b_1$$

Since the vertices consist of many tiles, the probability of needing a specific tile t is given as

$$P_{t,k} = \sum b_k^n \quad \forall n, t \in v_n$$

Again, we can get a prediction matrix $P_c = R^{T \times K}$ that has elements $p_{t,k}$.

1.1.4 Performance evaluation

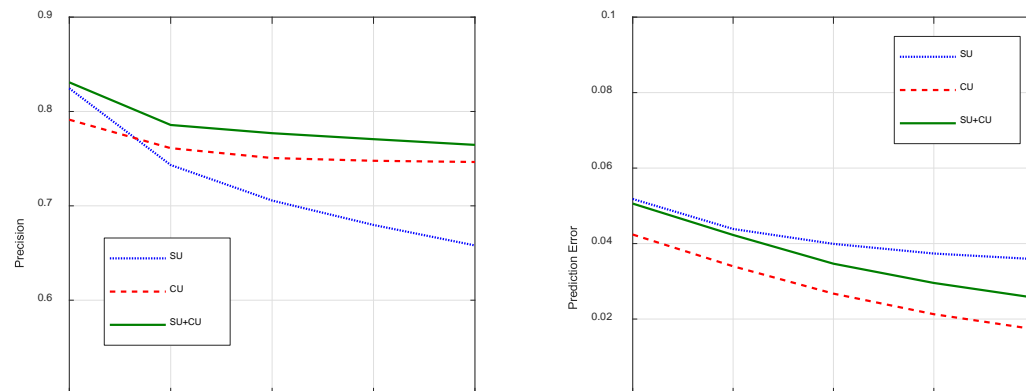


Figure 5 View Prediction with the Navigation Graph

(a) Average Precision

(b) Average Prediction Error

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

Figure 5.a shows the mean precision of view prediction result for k segments, $k = 1, \dots, 5$. SU+CU outperforms the other schemes because it can opportunistically choose the better prediction scheme.

Second, the prediction errors are also measured (Figure 5.b) to see whether the necessary tiles are requested on time. The prediction error is defined as the average of the number of tiles that has $p_{t,k} = 0$ but $g_{t,k} > 0$ over the total number of tiles needed to render the view. Therefore, it represents the percentage of visible blank areas in the view. The prediction errors under each condition are less than 6%. Our system tends to have lower prediction error for distant future prediction because the higher uncertainty of distant future prediction causes the Navigation Graph to request more redundant tiles. The prediction precision of the proposed Navigation Graph based scheme (SU+CU) scheme is compared with existing solutions, which are Linear Regression (LR), Linear Regression with Gaussian distributed error (LR-G), CLS-1, and CLS-2.

Linear Regression with Gaussian distributed error (LR-G) is probabilistic model of viewport which leverage Linear Regression (LR) with the assumption of normally distributed errors (or “Gaussian distributed errors”). The particular model simplifies the mathematics and makes the parameters in the model easier to estimate, it leads to optimal properties of the estimators (like being unbiased with minimum variance) under the Gauss-Markov theorem. (5)

CLS is a Cross user Learning based System for viewport-adaptive 360-degree video streaming aiming at improving the prediction precision for tile-based 360-degree video which uses the Client – Server model. Without user classification is referred to as CLS-1 and with user classification is referred to as CLS-2.

(6)

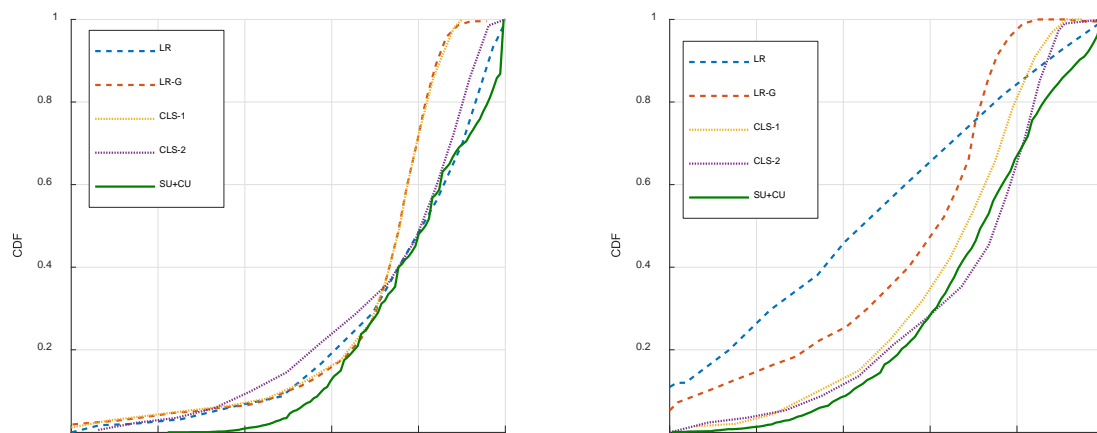


Figure 6 CDFs of View Prediction Precision

(a) View Prediction of 1 sec after current segment (b) View Prediction of 5 sec after current segment

(Navigation Graph for Tiled Media Streaming, Jounsup Park, Klara Nahrstedt, 2019)

Figure 6 shows that the proposed Navigation Graph based prediction has better precision results than CLS-2, which is a state-of-the-art method. LR works well for 1 second future view prediction, but the Navigation Graph based prediction has better precision than LR. The view prediction for 5 seconds in the future with LR does not work very well because the viewer usually does not keep the same viewing pattern for 5 seconds. The Navigation Graph again shows the best precision performance in predicting the view for 5 seconds in the future. The Navigation Graph based prediction outperforms for near future prediction and distant future prediction.

(7)

2. Compliance Management

Today enterprises do business in a rapidly changing environment and it is crucial for their prosperity to remain compliant with ever-changing and growing universe of regulations, policies and internal contracts.

Financial compliance orders companies to possess the technical ability and agility to screen, effectively, customer's economic behavior and transactions something that imposes data quality and data availability challenges. It, also, requires companies to maintain sophisticated customer screening and transaction surveillance systems that pose data quality and data availability challenges.

Current compliance systems are focusing mainly on data collection and data consolidation, leaving less time for in-depth analysis.

By using graphs is feasible to unify and interlink data from various sources and to apply complex rules and patterns for (semi-) automated compliance monitoring reaching an optimal level with the combination of contextual domain knowledge, Natural Language Processing (NLP) and Machine Learning such as regulatory compliance (8) or customer screening (9).

2.1 Use Case

The General Data Protection Regulation (GDPR) is the toughest privacy and security law in the world. Though it was drafted and passed by the European Union (EU), it imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the EU. The regulation was put into effect on May 25, 2018. The GDPR will levy harsh fines against those who violate its privacy and security standards, with penalties reaching into the tens of millions of euros and it is a very challenging task for companies to process data in a GDPR-compliant manner, extract information values, and use predictive analytics models. GDPR is laid out in 99 articles that describe its legal requirements, and 173 recitals that provide additional context and clarifications to these articles. GDPR is an expansive set of regulation that covers the entire lifecycle of personal data. As such, achieving compliance requires interfacing with infrastructure components (including compute, network, and storage systems) as well as operational components (processes, policies, and personnel)

(10)

Beekeeper AG provides a GDPR-compliant platform for employees of an enterprise in order to form a social network for communicating and finding work-related information.

Because of GDPR, analyzing the textual body of user interactions, such as posts, comments, and chats, is not allowed without consent. Therefore, it is impossible to gain insights from textual data across the spectrum of users. Exploring the network structure of employees provides data-driven insights into how information flows within an organization. Interpreting user interactions as a graph, where the nodes are users and elements, enables further extraction of user-user relationships.

The company offers a mobile application for target groups to establish a high-quality communication solution for companies whose employees have limited access to computers and laptops, such as hotel workers, construction workers, and customer service personnel. The application acts as a social networking platform that enables online collaboration and communities within the customer's organization. Due to GDRP rules, text data cannot be accessed, making text analytics for suggesting content to peer users impossible. However, anonymized metadata of user interactions can be used for analysis aiming at investigating whether modelling data in graph databases and applying graph mining can provide accurate results for extracting and predicting user relationships in enterprise social networks under the constraints of the GDPR.

Beekeeper user activity dataset considered for the period between 2019–2020, which contains 204 different user events related to user interaction in the platform such as likes of the content, reading comments, chat activity, login activity, etc.

Graph databases store data in a graph structure that consists of nodes and relationships, and both can have properties. They are considered NoSQL databases, and the information is stored in an entity-relationship model. It is the main difference between it and a relational database, as the data are not stored in tables. In the industry, graph databases are often used for fraud detection, recommendation systems, or social network modeling, to name a few examples. Neo4j has become one of the most popular and market-leading graph databases (The DB-Engines Ranking, which ranks database management systems according to their popularity, shows a positive trend for Neo4j or

Neo4j raised \$325 million at a more than \$2 billion evaluation in Series F deal led by Eurazeo, with additional capital from Alphabet's venture wing GV) (11) (12)

When considering how nodes are connected in a network, there are two common approaches. One is called a "random network," in which one node is connected to another with equal probability. The distribution of the degrees of a node follows the Poisson distribution, which means most nodes have the same number of connections. The other is a so-called scale-free network where node distribution follows the power law, meaning most nodes have fewer connections, and very few nodes have a high number of links to other nodes.

Neo4j divides graph mining algorithms into four main categories: Centrality, Community detection, Similarity, and Pathfinding.

Machine learning algorithms on graphs have attracted a lot of attention in both research and industry. There are three standard machine learning algorithms: unsupervised, supervised, and semi-supervised. They can all be applied to graphs depending on the use case. Neo4j offers in-database machine learning algorithms. The following algorithms, like Graph SAGE, Node2vec, Fast Random Projection (Fast RP), are considered node embedding algorithms, which compute a low dimensional vector representation of the nodes that can be used as features for further machine learning algorithms.

Beekeeper provided the data in a CSV file format. The agreement with Beekeeper was that, the client cannot be named, only that they are in the transportation industry. The data were pre-cleaned by Beekeeper. The dataset included ~70 million rows with a size of 13 GB of data for a period from 1 January 2019 to 28 February 2020.

The CSV file containing the data was loaded and converted using Python. The variables in Table 1 were selected to be loaded from the original dataset. Due to performance issues and hardware limitations, the number of rows in the original dataset was reduced to 10 million.

Variable Name	Description	Data Type
occured_at	The time the user interaction happened	Datetime
user_id	Anonymized ID of the user	Integer
Client	Client of user device	Integer
Path	API endpoint of interaction	String
normalized_path	Normalized API endpoint	String

Table 1 Selected Variables Based on Data Modell

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

The data went through transformations and the resulting data frame was loaded with the binary values of the interactions into Neo4j as a CSV file, using various scripts to obtain the predefined data model, which is demonstrated in Figure 7. Due to slow performance in Neo4j during the load process, 750.000 rows were used to establish the data model. The properties and data types were set with the Cypher scripts. Weights can be calculated for the relationships between users. However relationship weights are not considered because the interest of the work is to extract and predict static relationships.

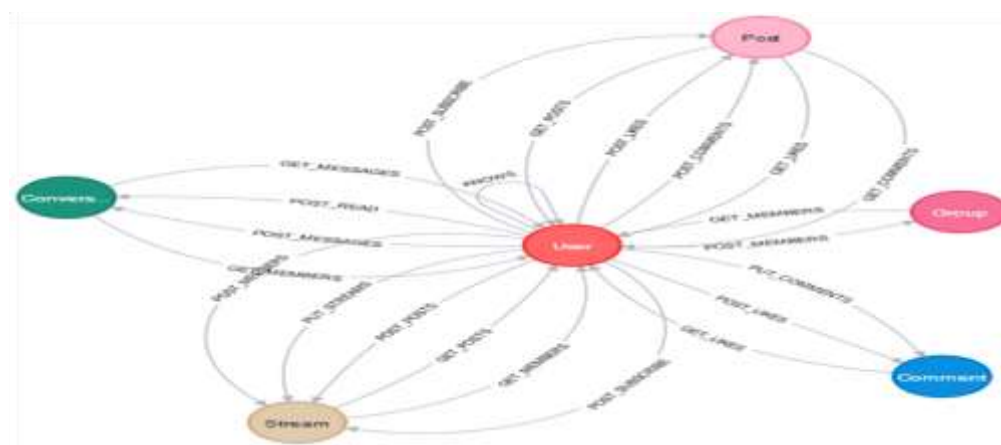


Figure 7 Data Model in Neo4j

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

Figure 8 demonstrates that the most frequent user interaction is retrieving posts, as the distribution of the user interactions and the GET_posts is the most common type in the sample. This interaction type means that a user is loading a post. The users tend to behave passively in other words, they consume information rather than actively responding, sharing, or liking content. The x-axis represents the user interaction, and the y-axis the frequency of the interaction. Interactions starting with CUMSUM and OPTIONS_read, OPTIONS_likes, and OPTIONS_like are not relevant for this research and were further ignored.

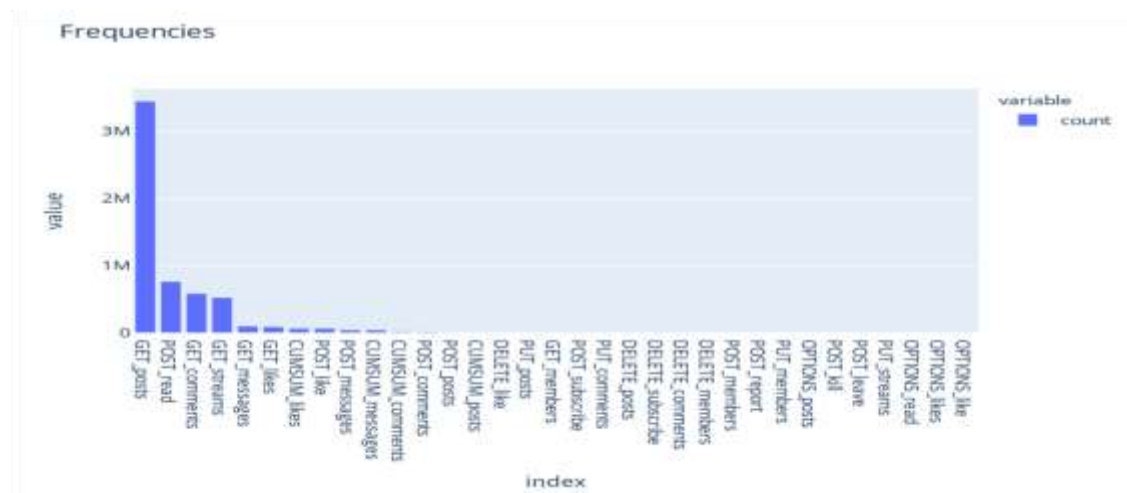


Figure 8 Frequencies of User Interactions

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

2.1.1 Artifact 1: Building Knowledge Graph-Based on User Interaction

This artifact aims to extract the current relationships of users—who knows whom—within Neo4j. The who knows whom type of relationship is not explicitly present on the Beekeeper platform, and second, there is no baseline data to confirm the relationships. Due to the limitations of the GDPR, we do not attempt to label the relationship as professional or private between the users. Therefore, only the chat interactions were used. The user node represents the user, the conversation represents the chat, and the three interactions represent what the user exactly did. The first

interaction with the chat item is always GET messages. A query was designed (Figure 9) that resulted in a new relationship, KNOWS, which was written into the database.

```
// Artifact_1_User_Knows_User_Query  
  
MATCH      (u1:User)<-[GET_MESSAGES]-(Conversation)-  
[:GET_MESSAGES]->(u2:User)  
  
WITH u1, u2, Conversation  
  
WHERE NOT u1.ser_id = u2.user_id  
  
MATCH (u1)-[k: KNOWS]-(u2) RETURN u1, u2, count(k) as weight;
```

Figure 9 Cypher Query <<who knows who>>

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

2.1.2 Artifact 2: Link Prediction with Neo4J Graph Machine Learning Algorithms

The designed artifact is used to predict user relationships by applying transductive learning and fitting logistic regression to predict probabilities for the user relationship. For prototyping, we used Neo4j's Graph Data Science Library version 1.7 algorithms. Artifact 2 preserves the graph structure of the data. Training and testing of the models were executed internally in the database, i.e., every transaction happens in the database, and no external algorithms or programming languages were used. The design of the artifact can be divided into eight main steps.

Figure 10 demonstrates the process steps described for prototype Artifact 2. The predicted positive relationships are written back into the database. These can be visualized with Neo4j Bloom or downloaded in a JSON or CSV format. The topN parameter defines the top 30 relationships, and the threshold determines the probability score above which the relationships are returned, in this case, 0.45. However, both topN and the threshold are arbitrary.

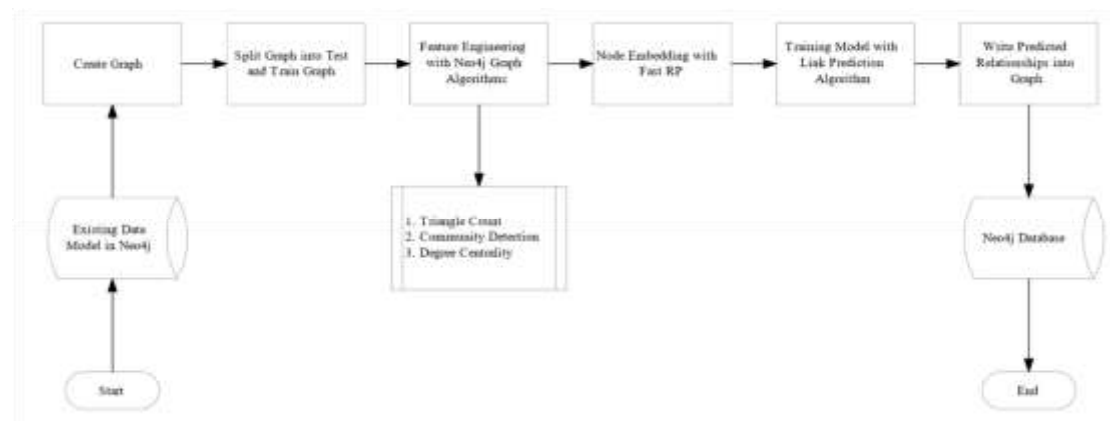


Figure 10 Process Diagram for Artifact 2

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

In the above mentioned diagram, we engineer a feature by using degree centrality. The algorithm uses Freeman’s formula to calculate the centrality of a graph. A single node in an undirected graph, is the number of neighbors with which there is a connection. The following equation is used to calculate the degree of nodes, where p_k is a single node and $a(p_i, p_k) = 1$ if and only if p_i and p_k are connected by a line, 0 otherwise.

$$C_D(p_k) = \sum_{i=1} a(p_i, p_k)$$

The equation means the higher the number, the more connections a node has. A value of 0 means, the node is isolated from the network.

We apply a node embedding algorithm, Fast Random Projection (Fast RP) algorithm which is 4000× faster in computational times than other state-of-the-art algorithms, such as Node2Vec* or Deep Walk. The above mentioned algorithm extracts the nodes with their features as vectors, optimizes the similarity matrix and utilizes a very sparse random. These algorithms perform two significant actions: first, the network is constructed as a similarity matrix, and second, a dimension reduction technique is applied. The extracted vectors will be used as numerical features to train the link prediction algorithms.

The link prediction algorithm is applied using the node embeddings. The link prediction algorithm fits a logistic function and sigmoid function with values between 0 and 1. Logistic regression predicts the probability of the label for a given input variable. The threshold is 0.45 for a provided label, i.e., if the probability is less than 0.45, the label

is predicted to be 0. Otherwise, it is expected to be 1. Then the predicted positive relationships are written back into the database.

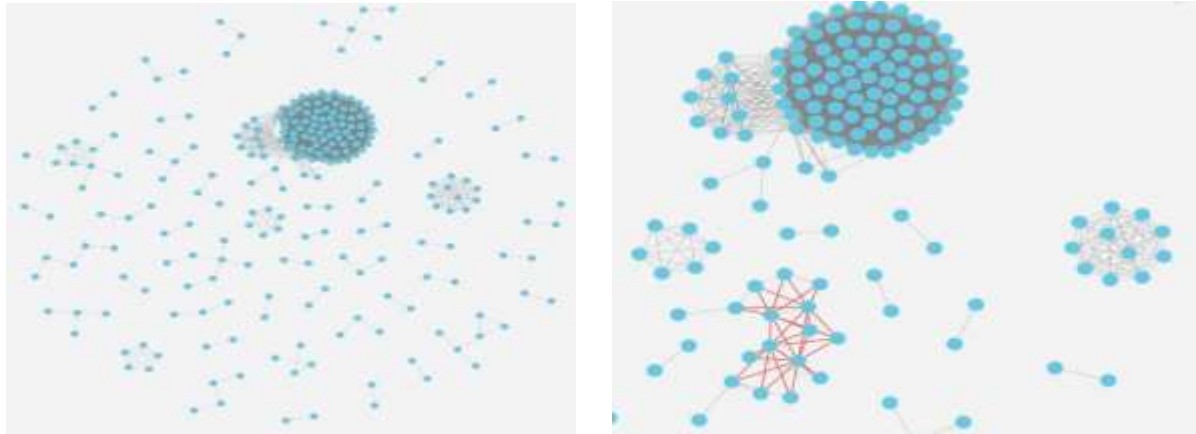


Figure 11 Labelled Property Graph Visualized with neo4j Bloom(left) and Predicted Relationships with Link Prediction(right). Screenshot from neo4j Bloom of the USER-KNOWS-USER query.

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

The second resulting artifact (Figure 11) is a predictive machine learning model using Neo4j in databased algorithms. The database contains 2819 user nodes, which were used to build the model. Artifact 2 is a probabilistic model providing the 30 highest predicted probabilities. They are above the defined threshold of 0.45. In other words, the red lines in Figure 11 represent predicted relationships where the probability value is above 0.45, indicating that users know each other, and these are written as different relationships in the graph.

Beekeeper was asked to provide feedback on the relationships sampled from Artifact 1, and the prediction results for Artifact 2. The validation sample included 3373 users, including the user pairs, depending on the artifact, the probability score, and predicted classes.

The precision-recall area under the curve (AUCPR) displays the precision and recall variety at different thresholds, and the goal is to have the area under the PR curve maximized, which represents a good classifier model. Precision (P) is defined as the number of true positives (TP) over the number of true positives plus the number of false positives (FP):

$$P = TP / TP + FP$$

Recall (R) is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (Fn):

$$R = TP / TP + FN$$

The so-called baseline average precision mean score (AP) has a value of 0.5, and the perfect classifier 1. Under 0.5, the classifier is not considered performant. The AP is calculated with the following equation:

$$AP = \sum_n (R_n - R_{n-1})P_n$$

The sample included 300 users for Artifact 1, which was sent to Beekeeper to validate the user relationships, whether the users knew each other or not. The validation by Beekeeper confirmed that out of 300 relationships, 278 exist. Twenty two relationships could not be found in the system.

Table 2 represents the results of the Link Prediction algorithm extracted from the Neo4j Desktop. TrainGraphScore here means the AUCPR score.

Model Name	Parameter Value			
	Winning		Model	
	TrainGraphScore		TestGraphScore	
	Max Epochs	Penalty		
myModel	1000	0.5	0.352	0.344

Table 2 Train and Test Results of Link Prediction Model with neo4j Fast RP Algorithm

(GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper, Rita Korányi , José A. Mancera * and Michael Kaufmann, 2022)

The test graph scores 0.344, indicating that the model performs moderately poorly overall, but it is considered acceptable due to the absence of the original features and although the predictive models have an average low AUCPR score, they are performing

well from a business domain perspective, as all the relationships exist, and there are no unknown labels.

GDPR data has consistently added value to the Beekeeper business model since the company guarantees total privacy to its users. However, it also represents a limitation to improving user experience through simple recommendations or analyzing user behavior for the purpose of understanding the final users better because the company cannot openly analyze the user data without their consent.

(13)

2.1.3 Executive summary of node embedding algorithm Node2Vec

Node2vec is a semi-supervised algorithm for scalable feature learning in networks. A flexible notion of a node's network neighborhood is established and with the use of random walk approach is possible to generate (sample) network neighborhoods for nodes, enabling node2vec to learn representations that organize nodes based on their network roles and/or communities they belong to. Experiments demonstrate that node2vec outperforms state-of-the-art methods by up to 26.7% on multi-label classification and up to 12.6% on link prediction.

Generally, there are two extreme sampling strategies for generating neighborhood set(s) NS of k nodes:

- Breadth-first Sampling (BFS) The neighborhood NS is restricted to nodes which are immediate neighbors of the source. For example, in Figure 1 for a neighborhood of size $k = 3$, BFS samples nodes s_1, s_2, s_3 .

- Depth-first Sampling (DFS) The neighborhood consists of nodes sequentially sampled at increasing distances from the source node. In Figure 1, DFS samples s_4, s_5, s_6 .

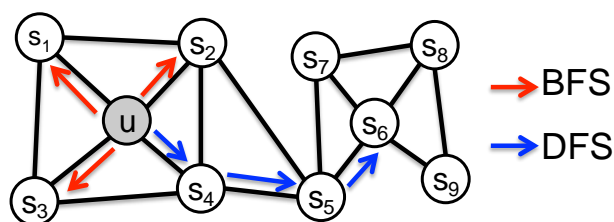


Figure 12 BFS and DFS search strategies from node u ($k=3$)

(node2vec: Scalable Feature Learning for Networks, Aditya Grover, Jure Leskovec, 2016)

The neighborhoods sampled by BFS lead to embedding's that correspond closely to structural equivalence. By restricting search to nearby nodes, BFS achieves this characterization and obtains a microscopic view of the neighborhood of every node.

In DFS, the sampled nodes more accurately reflect a macro-view of the neighborhood which is essential in inferring communities based on homophily.

Adopting a flexible sampling strategy which allows a smooth interpolation between BFS and DFS is feasible to develop a flexible biased random walk procedure that can explore neighborhoods in a BFS as well as DFS way.

Formally, given a source node u , we simulate a random walk of fixed length l . Let c_i denote the i th node in the walk, starting with $c_0 = u$. Nodes c_i are generated by the following distribution:

$$P(c_i = x | c_{i-1} = v) = \left\{ \frac{\pi_{vx}}{z} \text{ if } (v, x) \in E, 0 \text{ otherwise} \right\}$$

We define a 2nd order random walk with two parameters p and q which guide the walk and set the unnormalized transition probability to $\pi_{v,x} = \alpha_{p,q}(t, x) \cdot w_{v,x}$, where

and $d_{t,x}$ denotes the shortest path distance between nodes t and x . Note that $d_{t,x}$ must be one of $\{0, 1, 2\}$, and hence, the two parameters are necessary and sufficient to guide the walk having the privilege of being computationally efficient in terms of both space and time requirements.

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Parameters p and q control how fast the walk explores and leaves the neighborhood of starting node u . In particular, the parameters allow our search procedure to (approximately) interpolate between BFS and DFS and thereby reflect an affinity for different notions of node equivalences.

The node2vec algorithm

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d , Walks per

```

node  $r$ , Walk length  $l$ , Context size  $k$ , Return  $p$ , In-out  $q$ )
 $\pi = \text{PreprocessModifiedWeights}(G, p, q)$ 
 $G^0 = (V, E, \pi)$ 
Initialize  $walks$  to Empty for  $iter = 1$  to  $r$  do
for all nodes  $u \in V$  do
     $walk = \text{node2vecWalk}(G^0, u, l)$ 
    Append  $walk$  to  $walks$ 
 $f = \text{StochasticGradientDescent}(k, d, walks)$ 
return  $f$ 

```

```

node2vecWalk (Graph  $G^0 = (V, E, \pi)$ , Start node  $u$ , Length  $l$ )
    Initialize  $walk$  to [ $u$ ]
    for  $walk\_iter = 1$  to  $l$  do
 $curr = walk[-1]$ 
 $V_{curr} = \text{GetNeighbors}(curr, G^0)$ 
 $s = \text{AliasSample}(V_{curr}, \pi)$ 
    Append  $s$  to  $walk$ 
return  $walk$ 

```

The three phases of node2vec, i.e., preprocessing to compute transition probabilities, random walk simulations and optimization using SGD, are executed sequentially. Each phase is parallelizable and executed asynchronously, contributing to the overall scalability of node2vec.

(14)

3. Data Lineage & Metadata Management

Data lineage represents a detailed map of all direct and indirect dependencies between data entities in the environment ,in other words, what sources the data comes from,

where it is flowing to in the environment, and—last but not least—what happens to it along the way. A distinction is often made between horizontal and vertical lineage. Horizontal lineage describes a physical lineage through a data warehouse: from a landing area, through a staging area, via a core area into an outbound layer. A vertical lineage, on the other hand, focuses on the design of a database. It includes a business model, also known as a conceptual data model. This model is an abstract business view of the data. This serves as a design layer for a physical implementation in the form of a physical data model, which is a depiction of the physical artifacts on the database. These models are often directly linked, or are linked through a logical model, forming a bridge between business and IT. Horizontal and vertical lineage are therefore different concepts and demand an entirely different approach to the documentation of the data lineage. A vertical lineage revolves around the design process and thus relates abstract objects, such as a business partner, to physical implementation, such as a part of a star schema. A horizontal lineage shows the actual data flow and contains information about where the data enters the system, flows through the system, and is consumed.

(15)

In addition, the main issue is that complex data on the one hand, is difficult to be understood by human logical ability and on the other hand, do not explain their journey. The pure positive value of it is proportionate to its interpretability to stakeholders so data must be described with relevant metadata and be organized in a way that reflects its meaning and fitness to use.

The combination of detailed metadata and relationships between data lifecycle phases results in a semantic data layer. A semantic data layer (also known as data fabric) enables both data experts and business stakeholders to take advantage of any data asset to which they have access. Knowledge Graph, which is a collection of interlinked descriptions of concepts, entities, relationships and events, provides a semantic layer with full view of the data lifecycle all the way from the business to the most technical component.

3.1 Use Case

GE Capital is the financial arm of the General Electric Company which primary focus is on leasing and lending in the aviation, healthcare, and energy sectors where GE

manufactures and sells industrial equipment. To identify prospects, track customers, and manage the overall risk exposure of the portfolio, GE Capital utilizes data from a wide range of internal and external data sources, storing this data across over 2,000 internal repositories.

The GE Capital sub-businesses operated in silos, independently collecting and managing the data relevant to their products and customers. Due to the disjoint nature of this approach, it was difficult if not impossible to get a complete picture of where all of GE Capital's data resided, or how it flowed from system to system. This severely impeded the ability to perform key data management activities, for example, to identify duplicate data or processes, or find the point of origination of a particular data element.

In order to be able to capture the cohesive view of data flows and to interactively query and perform ad hoc reporting for internal information sharing and oversight GE Capital built a Concept Lineage Tool ('Colt') to address this challenge. With Colt, users can capture information about different types of systems and metadata about the data flowing between them, and answer questions about the lineage of those datasets. This information is made available for interactive exploration in the form of a directed graph (data flow network) where nodes represent data producers and/or consumers (hereafter referred to more generally as 'data stores'), and edges represent data flows between those data stores. This way data flows can be characterized by using an extensible set of hierarchical taxonomies, including many potential combinations of businesses, products and concepts associated with data flowing between systems possessing the ability to view the entire network, view only those systems upstream and/or downstream of a single system or collection of systems, and/or view the network filtered by terms selected from the data flow taxonomies.

Colt was designed and built to present complete metadata describing data flowing from one system to another. As such, the data is characterized in terms of a collection of taxonomies, including the data contents ("concepts") and the context in which this data exists ("boundaries"). It maintains lineage information at the concept level, with metadata describing flows in the context of concepts, businesses, and products. Their hierarchical organization enables rich queries—a key feature which is not readily available in other graph-based provenance models.

The system based on the conceptual architecture shown in Figure 13 where the data storage layer is comprised of a semantic triple store.

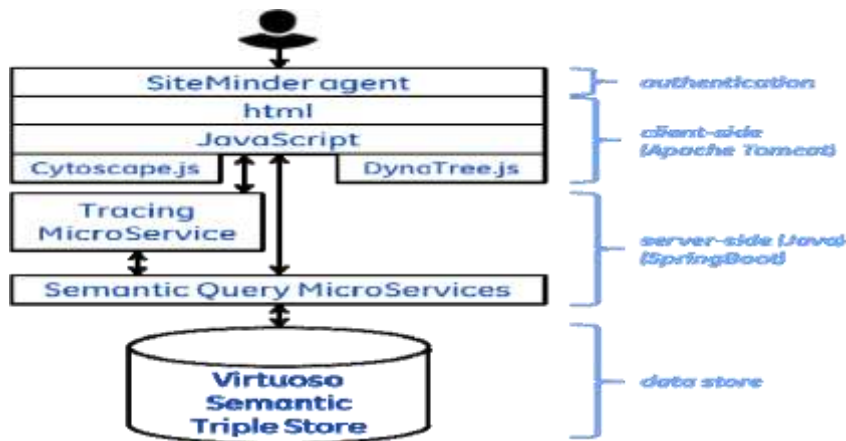


Figure 13 Colt conceptual system architecture

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

The use of semantic technologies was made because a store with intrinsic recursive graph query capabilities enables the efficient network traversal functionality something which is required in Colt and most important semantic models enable a high level of expressivity for representing the hierarchical data flow metadata in an easily extensible format.

A semantic model was built to represent data systems and data flows between them. Figure 14 is a simplified depiction of how the model represents a producer system sending data to a consumer system. The semantic model captures metadata about the nodes, including a node name, type and a brief description.



Figure 14 data flows from producer to a consumer

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

The data flow itself is modeled as a potentially complex combination of concepts and boundaries, as shown in Figure 15, with the asterisks representing one-to-many relationships. A data flow is associated with one or more structures called boundary groups, which in turn may contain one or more concepts and treatments and boundary sets. Each boundary set is a combination of boundaries (business and product boundaries, at present).

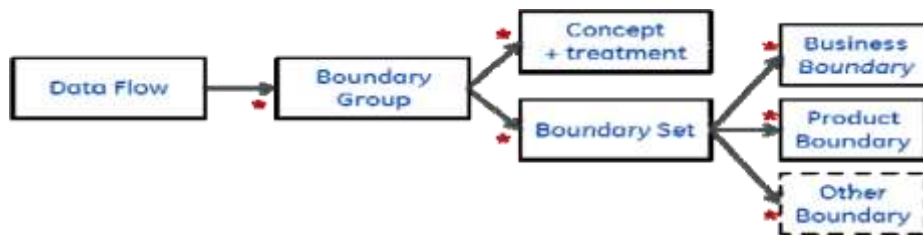


Figure 15 A data flow is a complex structure containing concepts and their treatments as well boundaries

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

A web-based user interface allows users to create, manage, view, and interactively explore the lineage network. Each node in the visualized network represents a GE Capital data system (or a relevant system external to GE Capital). Each edge in the graph represents the data flow from one system to another. Figure 16 shows a sample network wherein seven systems are sending data to a system called System H, which in turn sends data to System I. Note that this is a small example—the application is capable of displaying large networks with thousands of nodes and edges.

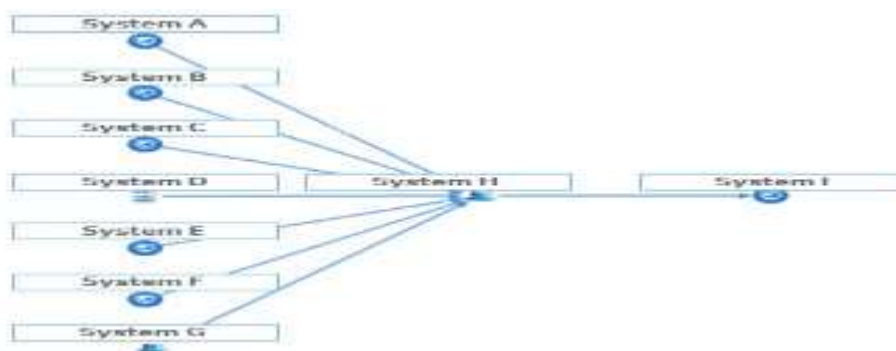


Figure 16 Example network containing 9 nodes

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

The user may explore the network in several ways, via the interface shown in Figure 17. Users may choose to view the entire network of data systems and data flows. Alternatively, the user may select one or more systems and choose to display all systems that feed the selected systems (“upstream” systems) and/or consume from the selected systems (“downstream” systems). At any point, a user may click on an individual system in the network and expand the graph to include systems upstream and/or downstream of the selected node. The user may choose to display the concepts or boundaries along the edges, as well.

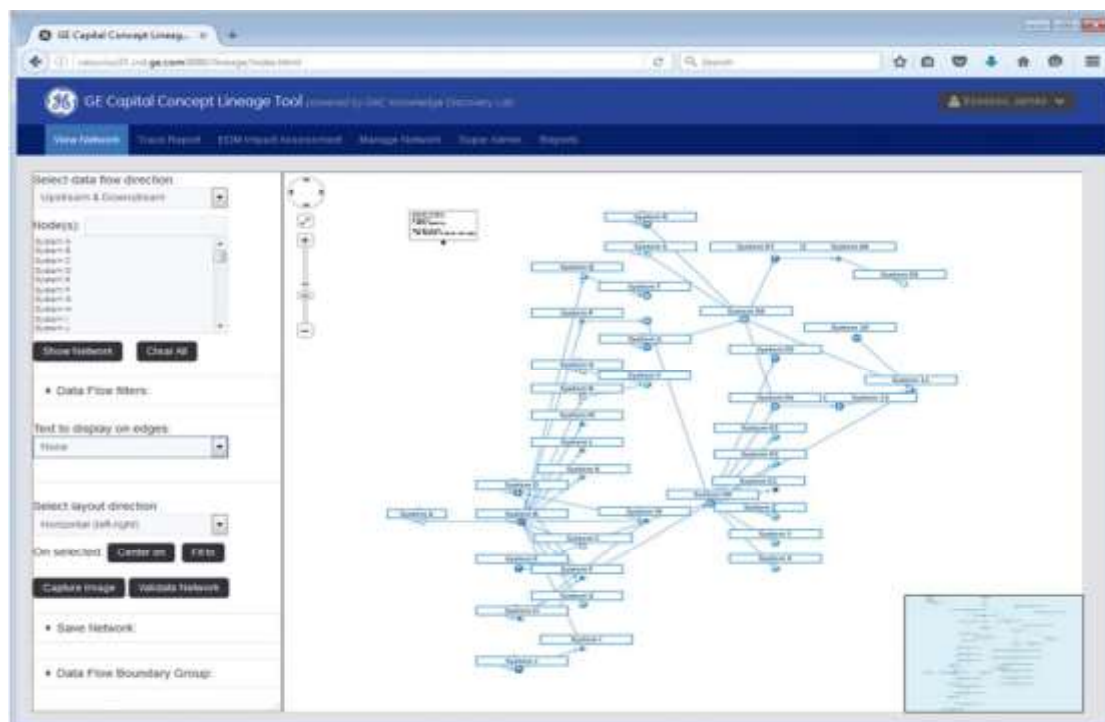


Figure 17 Colt UI visualize and explore a network

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

GE Capital users need the ability to trace data concepts from a given point in the network upstream to its points of origin (typically its point of creation), or downstream to its final points of destination for impact assessment.

For example as we can see in Figure 18 an upstream trace is performed on System Z. The first system to be considered is System Y. Next, Systems W and X are considered, and so forth. When tracing the data, the algorithm must not only consider the concept, but also the boundaries of interest. In this figure, concepts, businesses, and products are represented in the form {c,b,p}, since the metadata to be traced is defined by the triples

{concept, business, product}. When tracing upward from System Y, note that System W is determined to be included in the trace (since it sends the data of interest {c,b,p}), but System X is excluded from the trace because it sends product q instead of product p, and thus is not relevant to this trace. The tracing routine is recursive, with each iteration dependent on the results from previous iterations. Each iteration evaluates a single edge and determines if its data flows are relevant to the trace based on the data that has been passed through the previous nodes.

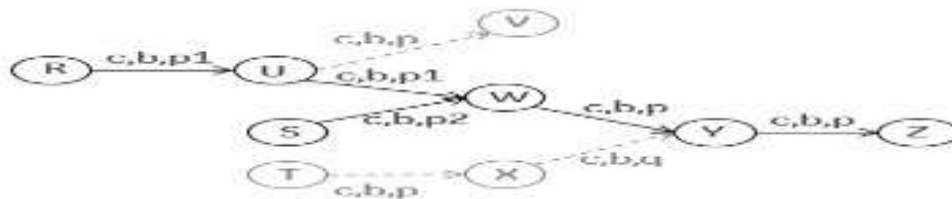


Figure 18 Network tracing for System Z (with grayed/dashed segments determined to be irrelevant for the trace)

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

The evaluation of each edge can be expressed by the set equation:

$$S_{A1 \rightarrow A2} \cap S_{A2 \rightarrow T} \neq \emptyset \rightarrow \text{keep}$$

Where:

$$S_{A2 \rightarrow T} = S_{A2 \rightarrow A3} \cap S_{A3 \rightarrow A4} \cap \dots \cap S_{An \rightarrow T}$$

and $S_{Ax \rightarrow Ay}$ = set of triples sent from node Ax to node Ay .

The algorithm used for the network tracing is as follows:

Identify concepts/boundaries feeding target system T

Convert all concepts/boundaries to concept-business-product triples at the lowest hierarchy level

Get all systems that feed data to node T, concatenate with name of node T to form a node trace path, and push those trace paths onto empty trace stack

Add T to traced network

While trace stack is not empty:

Pop trace path (“A”) from stack (will take the form

A1:::A2:::.....:T)

If path has a loop with one or more nodes between (e.g., A:::B:::C:::B:::D repeats node B) discard path and return to (a) (to avoid recursive loops in paths)

Query for concept-business-product triples that flow from the single-node hop from A1 to A2 in path

Determine whether there is an intersection in the {c,b,p}

between $A1 \square A2$ and $A2 \square T$ paths

If YES, add A1 to traced network

Store the resulting intersection of {c,b,p} triples that flow from A1 all the way to T

Get names of all systems that feed data to node A1, concatenate with previous node path and push those trace paths onto stack (e.g., A0:::A1:::A2:::.....:T)

If NO, continue to evaluate next entry on stack

(16)

By using the above mentioned algorithm a user can quickly determine the origination points for data feeding a particular system that intersect with the destination's concepts and boundaries. Figure 19 shows a sample trace result, for System A traced downstream.

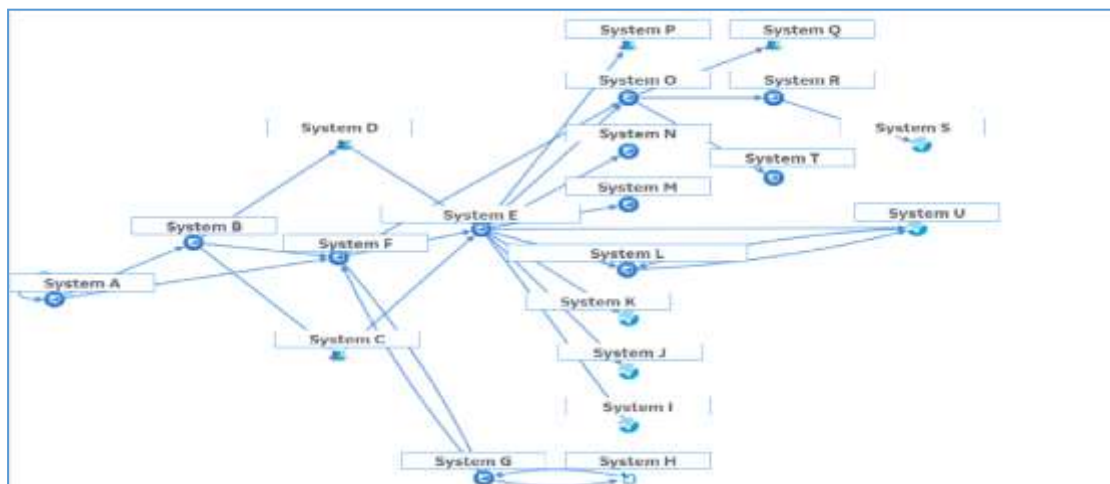


Figure 19 Downstream trace result for System A

(Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis, Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar)

Moreover GE Capital needs to perform validation checks to see whether the network is incomplete or incorrect. The validation routine works on a per-system basis, determining if the data that a given system sends is possible given the data that the system creates or receives. Specifically, a system will fail the validation routine if it is found to be sending data that it is neither creating nor receiving.

As an example, think (Figure 20), assuming that product p is comprised of subproducts $p1$ and $p2$. System W will fail validation, because it only receives subproduct $p1$, which is not enough for it to send product p . On the other hand, System X will pass validation because it receives subproducts $p1$ and $p2$, which are sufficient to product p .



Figure 20 System W fails validation because $p1$ is onlt subset of p . System X passes validation.

Validation algorithm

For each node N to validate:

Retrieve all lowest-level concept-business-product triples sent by node N

Retrieve all lowest-level concept-business-product triples created or received by node N

Perform an intersection of the triples sent and received and subtract the intersection from the triples sent by N

If the set of remaining triples sent is not empty, node N fails validation, else passes

(16)

Colt has been used extensively by GE Capital since it was first put into production in June 2015. At present, it contains a network of 2,200+ nodes and 1,047 edges (data

flows), which represent over 47,000 unique concept-business-product combinations sent between data systems. Colt currently contains 64 concepts, 226 business boundaries, and 125 product boundaries, each organized into 2- or 3-tier hierarchies. The tool's user base is currently at 50 and growing.

Overall, GE Capital estimates that Colt will result in a 50% reduction in time and cost of the requirement-building phases of complex IT projects, leading to millions in productivity savings alone. These estimated savings are attributable to Colt's ability to handle complex queries about data movement across the entirety of GE Capital, enabling analysts to efficiently answer regulatory and operational inquiries without having to elicit and fuse this information from a variety of personnel and other potentially non-computable sources.

(16)

4. Financial Crime Types

Financial crime, or economic crime, is defined by Europol as “illegal acts committed by an individual or a group of individuals to obtain a financial or professional advantage. The principal motive in such crimes is economic gain”.

There are numerous methods being deployed by criminals to attack financial institutions, corporations, public agencies, and individuals of the public.

(17)

4.1 STOCKS AND SECURITIES INVESTMENT FRAUD

The stock market and financial securities allow people to invest their money with the ambition of making a positive return based on either performing research, or just a hunch. However, it is known that a proportion of the market participants cheat, and by doing so make huge profits at the cost of institutional and retail investors. Catching these fraudulent actors is not easy, and typically requires a large workforce to gather evidence over a long period of time, particularly in cases of insider trading.

Market manipulation is considered an act of selling or buying a financial security with the objective of purposefully manipulating the price of the underlying asset or security. Illegal insider trading is when ‘insiders’, or people who are privy to private and non-

public company material use that information ahead of its public dissemination to benefit monetarily. This includes not only the act of trading on securities, but also the leaking of non-public information to third parties.

FIGURE 21. Diagram of the illegal insider trading process. Phase 1: Insiders with access or knowledge of nonpublic information about a company which will impact the share price. Phase 2 (Option 1): The insiders then buy or sell stock depending on underlying information. Phase 2 (Option 2): The insiders can share this information to a group of other malicious traders for monetary kickback or future insider information in return. Phase 3: The consortium or individual insiders reap the monetary benefits of the stock movements after nonpublic material is released to the wider investor public.



Figure 21 Diagram of the illegal process

(Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape, JACK NICHOLLS , ADITYA KUPPA , AND NHIEN-AN LE-KHAC , (Member, IEEE), 2021)

4.2 FRAUD DETECTION AND ANTI-MONEY LAUNDERING (AML)

Money laundering is a method used by criminals and people in possession of ‘dirty’ or illegally obtained funds through criminal activity to transform the money to a ‘clean’ or legitimate state in the eyes of the law and governments.

In the money laundering procedure we can distinguish three stages. The first is placement which is the process of depositing criminal money into the financial system. The second is layering which is moving the money within the financial system through complex webs of transactions with the goal of obfuscation. Layering is typically performed through offshore companies. Finally, integration is the criminal money being absorbed or blended into the real economy, through investments like real estate, stock purchases, and luxury items.

Financial Institutions have continued to heavily invest in suspicious transaction monitoring solutions that target specific transaction behaviors primarily through rules based triggers. These types of traditional transaction monitoring systems can process millions of transactions daily but are often hamstrung by a lack of flexibility in rule construction and can easily make use of rules that have become irrelevant or ineffective as criminals adopt new techniques. As a result over 95% of system-generated alerts from transaction monitoring systems are reported to be false positives. This staggering statistic reflects just how difficult it has become for banks to predict money laundering, as well as has become to predict as well as hinting at the inefficiencies in AML operations that exist at present.

Modern bank data infrastructure is built primarily using relational databases. While the reliance of storing data in rows and columns and linking data through primary and foreign keys has made data easier to manage, model and visualize, it has its limitations. Relational databases are often slow and rigid, requiring analysts to join multiple tables to produce consolidated views. AML investigations are hindered as data must be consolidated to determine payment trails and client networks across various datasets.

In order to refine TM and reduce false negatives, technology firms are promoting graphical databases and graphs that are designed to align to the nature of money laundering – networks and relationships.

In contrast, graph databases and the use of graph data models are entity orientated rather than table orientated and therefore, allow for the modelling of numerous relationships between entities (accounts, clients, customer details), as we would consider them in real-life. Knowledge graphs (how data is modelled in a graph database) are built using “triples” or “triplets”, consisting of nodes (the subject and the object) and edges (the relationship between the subject and object). These models can be analyzed using

mathematical graph theory and graph algorithms to uncover relational insights. To allow for additional dimensionality (and better machine readability), graph databases make use of taxonomies and ontologies. Taxonomies include names for objects and fundamental relationships, while ontologies define the types of nodes and relationships, classify concepts into meaningful categories, ascribe attributes to nodes and edges and define possible relationships between nodes (much like schemas). Using an ontology as a framework and data together you create a graph which empowered by machine learning and reasoning capabilities allow companies to better identify fraudulent patterns by traversing many hops on very large amounts of interconnected data in real-time.

This results in a polyhierarchical model, where entities are categorized multiple times to produce a graph-view. This is one of the key advantages of graph technology in that the data is represented in a way that makes sense to the human brain and is considered a close representation of how we would “whiteboard” these data landscapes during planning sessions. Furthermore, graphs do not rely on joins that can become tedious to code as well as constrain memory and CPU resources to execute. From an AML perspective, this is a problem as investigations are often slowed by the process of consolidating disparate datasets when examining suspicious transaction events. Graph technology allows for considerably faster ad-hoc querying to identify relationships that would often be unattainable through SQL and relationship databases.

FIGURE 22. Diagram of money laundering example. Phase 1: Person has money (typically cash) from the proceeds of criminal activities and places the money into the financial system through bank deposits. This can be done through a business front that is cash heavy (i.e., food business), Phase 2: Placement—Transactions are performed with shell companies to obfuscate origin. Phase 3: Layering—Offshore company can return the money to the original criminal through loan-back schemes to the cash heavy food business fronting as an investor. Phase 4: Integration—The criminal proceeds are now integrated into the economy and laundered.



Figure 22 Diagram of ML

(Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape, JACK NICHOLLS , ADITYA KUPPA , AND NHIEN-AN LE-KHAC , (Member, IEEE), 2021)

4.3 SIM-SWAPPING AND PHISHING

SIM-Swapping is an attack which allows a cybercriminal to gain unauthorized control of a wireless customer's mobile phone number. This gives an attacker access to the SMS-based text messages which enable resetting of account passwords on websites that rely on the security of a mobile phone number. A successful SIM-Swap attack requires a malicious actor to have the target's phone number, and depending on what account they wish to access, their email, as well. The attackers will either contact a victim's service provider and imitate the victim in order to transfer the phone number to a new SIM card, or the attackers have cooperating employees of a service provider which will allow them to an easier route of access. Once the attacker has access to the victim's phone number on their own SIM, they can extract SMS messages, including One Time Passwords sent by financial services such as Coin base. There are multiple aspects including phishing and social engineering which surround SIM-Swapping, but the main motivation for committing the act has been for the financial gain of the attacker.

Phishing is considered a social engineering technique with the interest of luring victims to unwillingly hand over their personal information including passwords, email addresses, phone numbers, addresses, usernames and financial information. There are multiple aspects including phishing which surround SIM-Swapping, but the main motivation for committing the act has been for the financial gain of the attacker.

FIGURE 23. Diagram of the SIM-Swap process. Phase 1: Attacker accesses victim's account credentials and mobile numbers. Phase 2: Attacker manipulates the service provider to perform the SIM-Swap with the victim's mobile number. Phase 3: Using newly gained access, attacker can now use account credentials to initiate a login attempt to a financial account. Phase 4: A One Time Password is sent from the financial service provider to the victim's mobile number. Phase 5: The victim's financial account is accessed, and funds are moved and laundered.

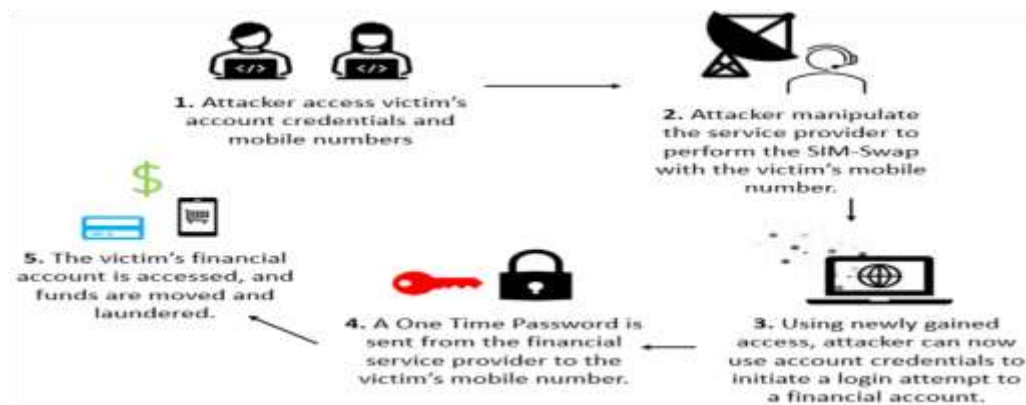


Figure 23 Diagram of the SIM-Swap process

(Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape, JACK NICHOLLS , ADITYA KUPPA , AND NHIEN-AN LE-KHAC , (Member, IEEE), 2021)

4.4 ROMANCE FRAUD

Romance fraud as defined by the FBI as a scam that occurs when a criminal adopts a fake online identity to gain a victim's affection and trust. The scammers will use that trust to build up an illusion of romance or close relationship and manipulate victims with the ambition of illegal financial gain. This fraud has seen a rise in popularity with scammers particularly through the global lockdown due to COVID-19 with reports of up to 20% increase in bank transfer fraud linked to romance scams in 2020 when compared with 2019. Romance fraud has reportedly been responsible for the theft of over \$362 million US dollars alone in 2018. Not only are victims scammed from their own money but can be used as money laundering mules unassumingly by being asked to transfer received money from the criminal to various accounts the criminal will instruct.

FIGURE 24. Diagram of romance fraud example. Phase 1: Genuine users search through online dating sites for matching profiles interested in starting a relationship. Phase 2: A match is made with a profile that appears genuine. Conversation usually attempts to build up trust and romance without physically meeting the person over several months. Phase 3: Behind the account is a fraudster. These are skilled at manipulating genuine users by portraying characters. They use fake profiles with stole profile photos or mimicked identities. Phase 4 (Option 1): Victims can be used unwittingly as launderers, cleaning criminals proceeds thinking they are doing favors to their potential love interests by performing illegal transactions. Phase 4 (Option 2):

Over time the fraudsters 'borrow' or are gifted funds from their victims. False promises are used such as money tied in investments or incoming inheritance to which they will repay their victims.

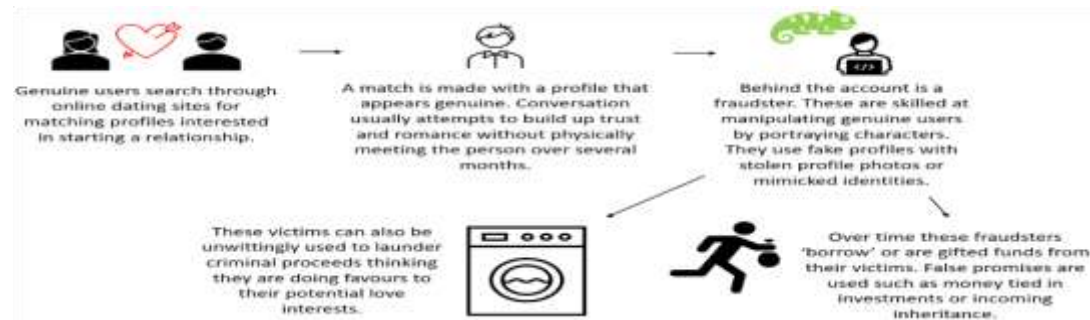


Figure 24 Diagram of romance fraud

(Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape, JACK NICHOLLS , ADITYA KUPPA , AND NHIEN-AN LE-KHAC , (Member, IEEE), 2021)

4.5 RANSOMWARE

Ransomware is a form of malware that has the ability to encrypt victim's computer systems and digital information, prohibiting access to it until a ransom is paid to the attackers. Malware is malicious software, it is created with an intent for criminality to gain access undetected into the computer systems of its victims. There are various forms of malware including Trojan horses, rootkits, and viruses. Typical payment demanded by the criminals is in the form of cryptocurrency due to the anonymity surrounding the owner of wallets.

(17)

4.6 DEEPPAKES AND GPT-2

Deep fakes and advanced chat bots like GPT-2 are capable of spoofing and manipulating staff at all levels of an organization. Deep fakes are not only audio manipulation but also visual. Deep fake programs are capable of creating completely fictitious identities of individuals. Websites such as <<Person Does Not Exist. Accessed: Aug. 26, 2021. [Online]. Available: <https://thispersondoesnotexist.com>>> uses a Generative Adversarial Network to create a 'person' or even generate modified images of a person without their consent. These images can also be used in online

profiles that can spoof genuine users of websites such as dating sites or social media vendor sites.

GPT-2, an open-AI chat bot which is trained to predict the next word in a sentence and has shown it can produce human-like passages of text such as news articles. GPT2 has been used to create false reviews for vendor websites such as Amazon. False reviews have the ability of fooling genuine customers into transacting with either illegitimate suppliers or low-quality goods manufacturers or damaging a rival business's total review score and reputation. Manually doing this across vendor websites is a method known as "crowdturfing" and is considered an attack on online review systems. A developed AI method by << D. I. Adelani, H. Mai, F. Fang, H. H. Nguyen, J. Yamagishi, and I. Echizen, "Generating sentiment-preserving fake online reviews using neural language models and their human- and machine-based detection," Jul. 2019, arXiv:1907.09177 >> implemented the GPT-2 system to create a bundle of false reviews which were not distinguishable against genuine reviews.

(17)

4.6 Use Case

Global financial crime volume was estimated to be around 1.4-3.5 trillion USD per year according to the latest industry reports, having negative effects on individuals and financial institutions as well as systemic effects such as negative consequences on a countries welfare through macroeconomic performance. Money laundering is estimated to be around 2-5% of the global GDP (up to 1.87 trillion EU), a large percentage of which is not detected.

(18)

Financial Institutions in order to be compliant with the strict rules about AML employ compliance experts that investigate suspicious activities alerted, usually, through a rule-based system following a procedure that can take several days to complete, culminating in a decision of flagging as suspicious activity or not. When the former is identified, a suspicious activity report must be filed and delivered to a regulatory institution that proceeds with due action. It is a process that requires the filtering of large bulk of transactions into a smaller set of abnormal interactions that can be used to justify suspicious activity.

With the intention to moderate all the above mentioned cumbersome and complex tasks we present LaundroGraph, a novel fully self-supervised approach leveraging Graph Neural Networks (GNNs) to encode representations of customers and transactions within the context of AML reviewing. The network of financial interactions is represented as a directed bipartite customer-transaction graph with the GNN trained through a link prediction task between pairs of customer and transaction nodes, essentially corresponding to an anomaly prediction task providing an analyst a starting point of potentially suspicious movements and alleviating the effort required to filter the bulk of transactions.

The goal is, for this mechanism, to be concluded within a larger system for AML reviewing that handles the necessary workload of assessment creation. Within this system, these insights will be digested and provided in an easy-to-understand manner through tailor-made visualizations for AML as soon as the investigation starts.

We make use of a directed bipartite graph $G = (V, E)$, with $V = C \cup T$ denoting the set of customer (C) and transaction (T) nodes, and $E = I \cup O$ denoting the set of edges between them, where O represents outgoing transactions of the form $C \rightarrow T$, and I represents incoming transactions of the form $T \rightarrow C$. Each node type is associated with a feature vector $f_c \in R^{dc}$ and $f_t \in R^{dt}$, respectively representing the customer and transaction node feature vectors. Customer features, which we refer to as profiles, characterize the customers' transactional behavior within time windows of different granularities, plus other relevant attributes about the customer, while transaction features contain information about the transaction itself. Customer nodes are connected to all transactions in which they are involved, and transaction nodes are connected to their source and destination customer. As such, each customer has as many edges as transactions performed in that time period and each transactions has, at most, two edges: one incoming and one outgoing. A simplified illustration of this graph can be visualized in Figure 25.

Figure 25: Proposed system training overview. Outgoing transactions are represented with filled arrows, and incoming transactions with dashed arrows. First, the bipartite graph is built from a dataset comprised of raw transactions. Then, positive pairs (green) and negative pairs (red) together with their K -hop subgraphs ($K = 2$ in the figure) are extracted and their embedding's obtained through the encoder. Finally, the decoder uses the aforementioned embeddings to generate the prediction for each sampled edge.

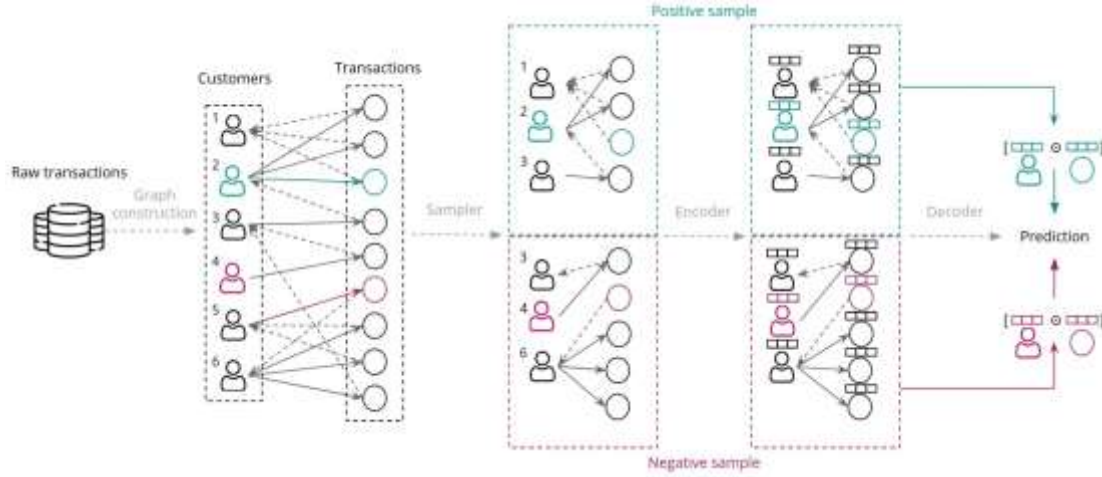


Figure 25 System training overview

(LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering, Mário Cardoso, Pedro Saleiro, Pedro Bizarro, 2022)

Preliminaries. The objective is to jointly learn an encoder $E(X, A) \rightarrow \mathbb{R}^{N_c \times d_c} \times \mathbb{R}^{N_t \times d_t}$ and a decoder $D(z_c, z_t) \rightarrow \mathbb{R}^1$. The encoder receives a node feature matrix $X : \mathbb{R}^{N_c \times d_c} \times \mathbb{R}^{N_t \times d_t}$ and an adjacency matrix $A : \mathbb{R}^{N_c \times N_t} \times \mathbb{R}^{N_t \times N_c}$ and produces a set of embedding's $Z = [z_c^i, z_t^j], \forall i \in \{0, \dots, N_c\}, j \in \{0, \dots, N_t\}$, with each embedding $z_c^i \in \mathbb{R}^{d_c}$ and $z_t^j \in \mathbb{R}^{d_t}$ denoting the representations for each customer node i and transaction node j , respectively. The decoder receives a pair of customer-transaction embedding's (z_c, z_t) , and outputs the likelihood of that transaction existing for that customer.

By using the term customer-transaction embedding's we imply the possibility of presenting nodes that live in a sparse, high-dimensional non-Euclidean space to a low-dimensional space (continuous dense vectors).

Fig. 26. Schematic of graph (node) embedding. For a simple graph $G(V, E)$ consisting of a node set V and an edge set E , using a graph embedding model f , different nodes (e.g., v_1 and v_2) from the original graph in a high-dimensional irregular domain can be

mapped into a latent low-dimensional space as a L-dimensional dense and continuous

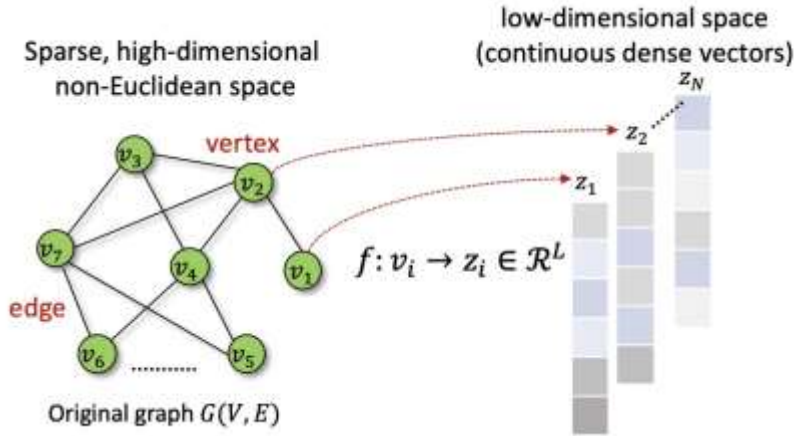


Figure 26 Schematic of graph(node) embedding

vector z_i , $L \ll |V|$.

(19)

The node structure property can be also preserved in the latent space, i.e., similar nodes in the original space will be close to each other in the latent space. Moreover, the obtained latent variables z_i , $i \in V$ (i.e., features) can be readily used for diverse downstream graph analytic tasks. (19)

Let \odot denote the Hadamard product and σ the sigmoid nonlinearity. The decoder is comprised of a simple feed-forward, and the prediction for an edge with customer node c and transaction node t is defined as follows:

$$\hat{y}_{c,t} = \sigma(W[z_c \odot z_t])$$

A single decoder is used to predict both incoming and outgoing transactions and the anomaly score is defined as $1 - \hat{y}_{c,t}$.

Algorithm 1 LaundroGraph forward propagation algorithm

Input: Graph G ; number of layers L ; neighborhood sampler N ;

mini-batch size B ; edge sampling function S ; edge direction D

$Ep : (c_1, t_1), \dots, (c_B, t_B) \leftarrow$ select B edges from G in direction D

$En : (\tilde{c}_1, \tilde{t}_1), \dots, (\tilde{c}_B, \tilde{t}_B) \leftarrow S(G) \triangleleft$ Sample random c and t as

non-edges $E \leftarrow Ep \cup En$

if $D ==$ outgoing then

$G \leftarrow G \setminus (c \rightarrow t), \forall t \in E \triangleleft$ Delete real outgoing edges

else

$G \leftarrow G \setminus (t \rightarrow c), \forall t \in E \triangleleft$ Delete real incoming edges

end if

$z_{ci}^0, z_{ti}^0 \leftarrow f_{ci}, f_{ti}, \forall (ci, ti) \in (N_{(c)} \cup c, N_{(t)} \cup t), \forall (c, t) \in E \triangleleft$

Input to the first layer is the raw features of all required nodes

for $l \in 1, \dots, L$ do

for $(c, t) \in E$ do

$z_c^l \leftarrow \text{Convolve}(\{z_{ci}^{l-1}, \forall c_i \in N_{(c)} \cup c\}) \triangleleft$ Encode nodes

$z_t^l \leftarrow \text{Convolve}(\{z_{ti}^{l-1}, \forall t_i \in N_{(t)} \cup t\}) \triangleleft$ Encode nodes

end for

end for

$y_{c,t}^{\wedge} \leftarrow \sigma(W[z_c^L \odot z_t^L]), \forall (c, t) \in E \triangleleft$ Decoder edge prediction

(20)

This usual behavior is dictated by the input graph G , and is leveraged by the decoder to classify new transactions entering the system with the goal to identify anomalous transactions within the context of a customer's usual behavior.

A real-world banking dataset is used in our experiments performing experimental analysis with popular models of an MLP and LightGBM which predict the existence of an edge, given only the raw features of the source customer, destination customer and transaction.

Apart from the above mentioned models we experiment with another popular self-supervised GNN objective, namely the Deep Graph Infomax (DGI) objective.

.

Method	AUC	AP
--------	-----	----

MLP	77.26	82.45
LightGBM	82.58	89.02
DGI	85.87	84.06
LaundroGraph _{SAGE}	89.97	93.17
LaundroGraph _{GIN}	90.24	93.82
LaundroGraph _{GAT}	94.83	95.22

Table 3 ROC AUC and average precision(AP) results on the test data for all methods under consideration

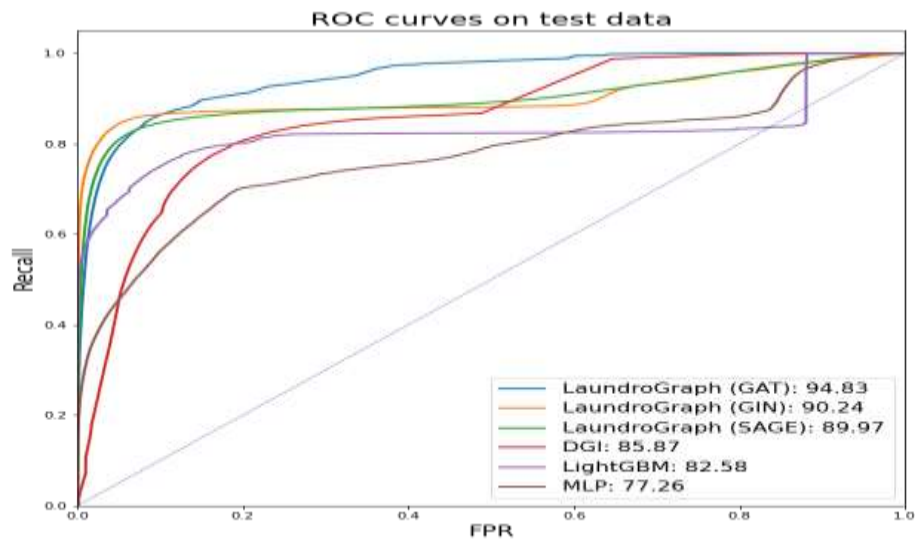


Figure 27 ROC curves and corresponding AUCs for all models considered

(LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering, Mário Cardoso, Pedro Saleiro, Pedro Bizarro, 2022)

By considering table 3 we can see the MLP and LightGBM models that count upon raw features manage to achieve quite competitive results but all graph-based baselines achieved superior performance, showing the importance of leveraging the structural information provided by the underlying graph.

Figure 27 shows the ROC curves of all the methods reported in Table 1. The ROC curves show the trade-off between recall and specificity. Moreover, the area under the curve (AUC) can be seen as a measure of separability, representing how much a model is capable of distinguishing between classes. From observing Figure 27, we verify that all graph-based models consistently outperform the models relying exclusively on raw features. In particular, for very low false positive rates (FPRs), all graph-based variants trained directly on the link prediction task already achieve a recall of $> 80\%$, whereas the MLP and DGI models achieve a recall of 40% or below, with the LightGBM model being a middle-ground between them at $\sim 60\%$ recall. As the FPR increases, the DGI model approaches the performance of the remaining graph-based models, while the MLP and LightGBM models continue to achieve consistently inferior results.

Figure 28: UMAP visualization of the customer embeddings produced by LaundroGraph *GAT* (left), together with corresponding cosine similarity heatmaps (right), for 6 sampled customers across 3 snapshots of data. Colors represent the different customers. On the left plot, the UMAP embeddings are shown, with each customer providing 3 points, one for each snapshot (18 points in total), connected through a dashed line of the same color. On the right plot, the cosine similarities on the original embedding space are shown, for each customer and snapshot.

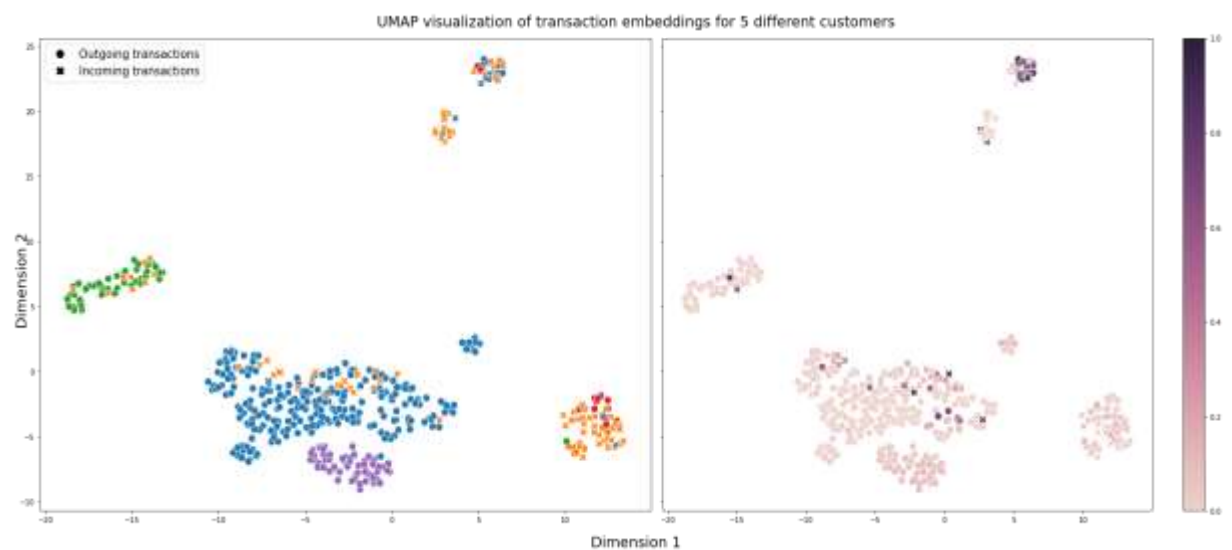


Figure 28 UMAP visualization

(LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering, Mário Cardoso, Pedro Saleiro, Pedro Bizarro, 2022)

In figure 28 we can see a plot of the UMAP embeddings for the transactions of 5 different and randomly sampled customers with more than 10 transactions. The marker represents the direction of each transaction, with "o" representing outgoing transactions, and "x" representing incoming transactions. On the left side of the figure transactions are colored according to their customer, and on the right side transactions are colored according to their anomaly score. For example the green customer, all outgoing transactions except one were received by the same counterpart, resulting in the left-most green cluster. The remaining outgoing transaction can be seen farther away, near the right-most cluster.

Another interesting case is the purple customer where the cluster represents interactions with several different counterparts whose behavior is very similar. More specifically, almost all counterparts only received transactions from the purple customer. From the right side of the figure we can observe that, generally, transactions farther away from their respective non-anomalous clusters (i.e., the "expected" behavior) usually have a higher anomaly score. This can be observed, for example, with the anomalous cluster at the top, and with the scattered incoming transactions from the orange customer.

(20)

5. Recommender Systems & Conversational AI

Different from traditional one-shot recommendation systems conversational recommender systems (CRS) obtain users' interests through multi-turn conversation, and make recommendations with responses. Typical CRS consists of two parts: recommender and response generation. The recommender aims to understand users' dynamic preference from contextual utterances to find the items matching the preference best. Subsequently, the response generation aims to generate appropriate sentences asking for more information or exhibiting the recommended items and related explanation. Recommender and response generation are expected to be mutually

beneficial. However, contextual utterances are usually insufficient to understand users' preference.

In order to provide recommendations that match user's intent, preferences and interest, a recommender system needs to know about the context of conversation and must be capable of not only looking at structural similarity between items but also their semantic similarity. For example the word "balance" might convey different meanings when used in a financial conversation as opposed to a medical context.

A Knowledge Graph is built as a large semantic network of entities and their attributes. Therefore, it allows finding the best matching entities based on semantic similarities between the entities. Knowledge Graphs also allow enriching the context of data by incorporating domain-specific knowledge vocabularies, taxonomies or ontologies. Knowledge Graphs provide a personalized experience which enables conversational banking tools to interact, more effectively, with customers on their financial needs.

5.1 Use Case

As a use case we exhibit Conversational Path Reasoning (CPR), a generic method that models conversational recommendation as an interactive path reasoning problem on a knowledge graph. It walks through the attribute vertices by following user feedback, utilizing the user preferred attributes in an explicit way. By leveraging the knowledge graph structure, CPR is able to eliminate many irrelevant candidate attributes, leading to better chance of finding user preferred attributes. With the purpose of showing how CPR works we make use of a simple, yet effective, application named SCPR (Simple CPR). During the presentation of the above mentioned method and before we examine its effectiveness comparing with others like CRS and EAR methods we will provide a brief but comprehensive description of them.

5.1.1 CRS method

It is a conversational system that tries to collect user preferences by asking questions in the form of semi-structured query with facet (attribute)-value pairs. Once enough user preference is collected, a set of machine actions tailored for recommendation agents is introduced and a deep policy network is trained, to decide which action (i.e. asking for the value of a facet or making a recommendation) the agent should take at each step in

order to make personalized recommendations to the user. A main concern is to avoid overwhelming users with too many facet (attribute)-value pair options per conversation so a faceted search engine selects a small set of facet (attribute)-value pairs based on the context and asking user to provide information about his preferences such as "What's the color you like?", "Which brand you prefer?", "Do you like small size, middle size or large size.

The particular system has three major components. First, a natural language understanding (NLU) module for analyzing each user utterance, keeping track of the user's dialogue history and constantly updating the user's intention. This NLU module focuses on extracting item specific meta data. Second, a dialogue management (DM) module that decides which action to take given the current state. This DM module has an action space defined specifically for this task. The third component is a natural language generation module to generate response to the user. This conceptual construction provides the capacity to build a conversational search and recommender system that can decide when and how to gather information from users and make recommendations based on a user's past purchasing history and context information in the current session.

We make use of Deep Reinforcement Learning which is based in deep neural networks and has been applied for better sequential decision making in many domains. As above mentioned our framework has three components: a belief tracker, a recommender system and a policy network.

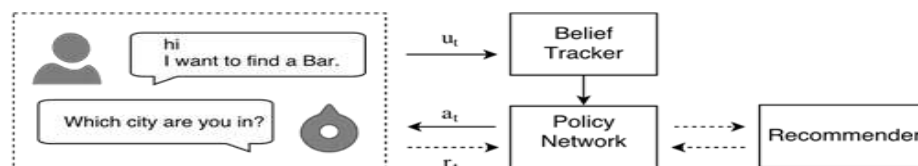


Figure 29 The conversational recommender system overview

(Conversational Recommender System, Yueming Sun, Yi Zhang, 2018)

Figure 29 presents the overview of our proposed framework. At a time step in the dialogue, the user utters "I want to find a Bar". The framework calls the belief tracker

to convert the utterance into a vector representation or “belief”; then the belief is sent to the policy network to make a decision. For example, the policy network may decide to request the city information next. Then the agent may respond with “Which city are you in?”, and gets a reward, which is used to train the policy. A different decision is to make a recommendation. Then the agent calls the recommender system to get a list of items personalized for the user.

In Figure 30 the structure of the proposed conversational recommender model in the bottom part is the belief tracker, the top left part is the recommendation model, and the top right part is the deep policy network.

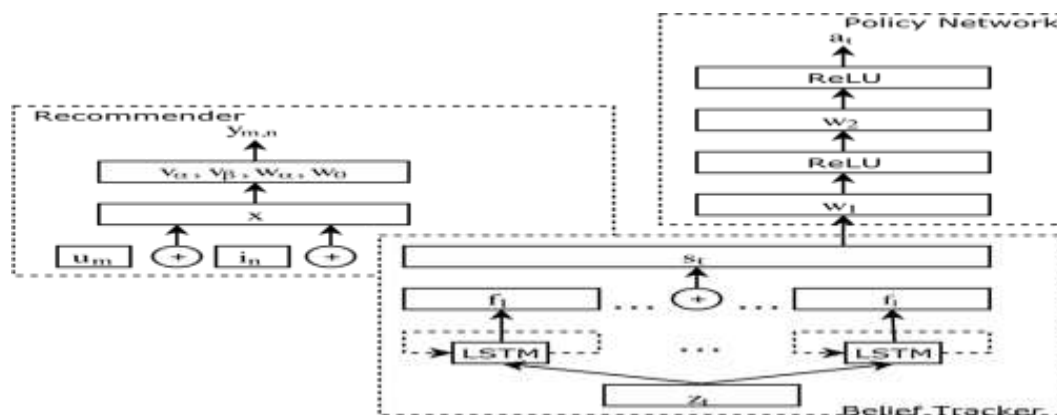


Figure 30 The structure of the proposed Conversational Recommender System

(Conversational Recommender System, Yueming Sun, Yi Zhang, 2018)

Belief Tracker module extracts facet-value pairs from user utterances during the conversation, and maintain the facet-value pairs as the memory state (i.e. user query) of the agent. The product facet (or attribute, metadata) f along with its specific value v is a facet-value pair (f, v) . Each facet-value pair represents a constraint on the items. For example, (color, red) is a facet-value pair which constrains that the items need to be red in color. As we can see in Figure 30 the belief tracker takes the current and the past user utterances as the input, and outputs a probability distribution across all the possible values of a facet at the current time point. The dialogue system’s belief of the session is constituted by the predicted values of different facets.

As the conversational system interacts with the users, at certain round, the conversational system can decide to make a recommendation based on its current belief of the user’s information need, which is interpreted as the dialogue state. The structure of recommendation model is shown in the upper left part of Figure 30. Let U denote

the users and I the items. For M users and N items in the dataset, the users and items are represented as the sets: $\{u_1, u_2, \dots, u_M\}$ and $\{i_1, i_2, \dots, i_N\}$. The input feature x is the concatenation of the 1-hot encoded user/item vector, where the only element that is not zero in the vector corresponds to the index of the encoded info, and the dialogue belief:

$$x = u_m \oplus i_n \oplus s_t$$

$$u_m = \{0, 0, \dots, 1, \dots, 0\}, \text{with } 1 \text{ at the } m^{\text{th}} \text{ element}$$

$$i_n = \{0, 0, \dots, 1, \dots, 0\}, \text{with } 1 \text{ at the } n^{\text{th}} \text{ element}$$

where m and n denotes that i_n is rated by the u_m .

The output $y_{m,n}$ can be either a rating score for the explicit feedback or a 0-1 scalar for the implicit feedback.

Then we portray the deep policy network where its structure is shown in the upper right part in Figure 30. The reinforcement learning has the basic components of state S , action A , reward R and policy $\pi(a|s)$.

State: The state s_t is the current description of the environment from the viewpoint of the agent. In our case, it is the description of the conversation context, which is the belief tracker's output, $s_t = \{f_1 \oplus f_2 \dots \oplus f_t\}$.

Action: An action a_t is the decision the agent needs to make at time step t . Here we have mainly two kinds of actions. One is to request the value of a facet, which is further divided into l actions $\{a_1, a_2, \dots, a_t\}$, one per each facet. The other is to make a personalized recommendation a_{rec} , in which case the recommendation module described above would be called.

Reward: The reward is the benefit/penalty the agent gets from interacting with its environment. At each turn, according to the current state s_t , the agent selects an action a_t following the policy, and it gets an immediate reward r_t , denoting how good the current decision is. The state s_t transits to a new state s' .

Policy: This is the target the model tries to learn. Usually denoted as $\pi(a_t | s_t)$, the policy represents the score, such as the probability, of taking action a_t when the agent is in state s_t .

(21)

5.1.2 EAR method

The aforementioned method is a multi-round CRS model which in each round, is allowed to choose two types of actions — either explicitly asking whether a user likes a certain item attribute or recommending a list of items. In a session, the model may alternate between these actions multiple times, with the goal of finding desirable items while minimizing the number of interactions. It is made of three stages to better converse with users.

(a) Estimation, which builds predictive models to estimate user preference on both items and item attributes. In particular we train a factorization machine (FM) using user profiles and item attributes as input features. Our Estimation stage builds in two novel advances: 1) the joint optimization of FM on the two tasks of item prediction and attribute prediction exerting positive influence on EAR, where the first directly contributes to success rate of recommendation, and the second guides the CC to choose better attributes to ask users so as to shorten the conversation. 2) the adaptive training of conversation data with online user feedback on attributes.

(b) Action, which learns a dialogue policy to determine whether to ask attributes or recommend items, based on Estimation stage and conversation history. Our focus is on conversational recommendation strategy, as opposed to fluent dialogue (the language part) we use templates as wrappers to handle user utterances and system response generation. We train a policy network with reinforcement learning, optimizing the reward of shorter turns and successful recommendations based on the FM’s estimation of user preferred items and attributes, and the dialogue history. In EAR, we design four kinds of rewards, namely: (1) r_{suc} , a strongly positive reward when the recommendation is successful, (2) r_{ask} , a positive reward when the user gives positive feedback on the asked attribute, (3) r_{quit} , a strongly negative reward if the user quits the conversation, (4) r_{prev} , a slightly negative reward on every turn to discourage overly lengthy conversations. The intermediate reward r_t at turn t is the sum of the above four rewards, $r_t = r_{suc} + r_{ask} + r_{quit} + r_{prev}$. We denote the policy network as $\pi(a^t | s^t)$, which returns the probability of taking action a^t given the state s^t

(c) Reflection, which updates the recommender model when a user rejects the recommendations made by the Action stage and adapts the CRS model with user’s online feedback. Specifically, when a user rejects the recommended items, we construct

new training triplets by treating the items as negative instances and update the FM in an online manner.

In Figure 31 in the workflow of our multi-round conversational recommendation scenario, the system may recommend items multiple times, and the conversation ends only if the user accepts the recommendation or chooses to quit.

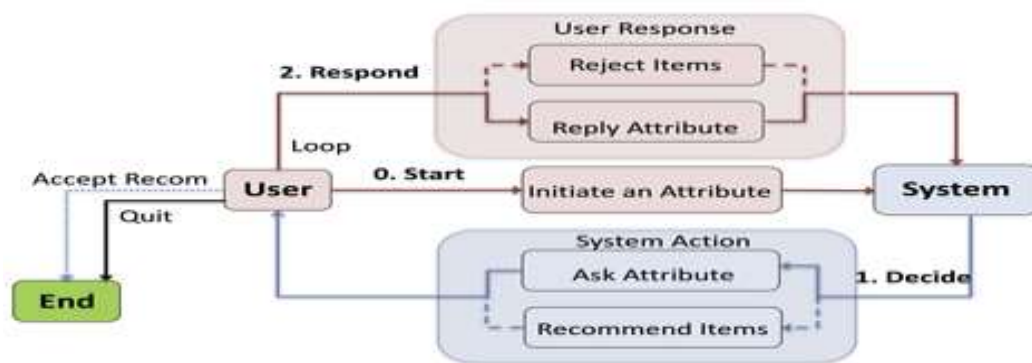


Figure 31 The workflow of our multi-round conversational recommendation scenario

(Estimation–Action–Reflection: Towards Deep Interaction Between Conversational and Recommender Systems, Wenqiang Lei , Xiangnan He , Yisong Miao , Qingyun Wu , Richang Hong , Min-Yen Kan, Tat-Seng Chua, 2020)

Let $u \in U$ denote a user u from the user set U and $v \in V$ denote an item v from the item set V . Each item v is associated with a set of attributes P_v which describe its properties, such as music genre “classical” or “jazz” for songs or tags such as “nightlife”, “serving burgers”, or “serving wines” for businesses. We denote the set of all attributes as P and use p to denote a specific attribute. A CRS session is started with u ’s specification of a preferred attribute p^0 , then the CRS filters out candidate items that contain the preferred attribute p^0 .

Then in each turn t ($t = 1, 2, \dots, T$; T denotes the last turn of the session), the CRS needs to choose an action: recommend or ask:

If the action is recommend, we denote the recommended item list $V^t \subset V$ and the action as a_{rec} . Then the user examines whether V^t contains his desired item. If the feedback is

positive, this session succeeds and can be terminated. Otherwise, we mark V^t as rejected and move to the next round.

If the action is ask (where the asked attribute is denoted as $p^t \in P$ and the action as $a_{ask}(p^t)$), the user states whether he prefers items that contain the attribute p^t or not. If the feedback is positive, we add p^t into P_u to denote the preferred attributes the user in the current session. Otherwise, we mark p^t as rejected; regardless of rejection or not, we move to the next turn.

This whole procedure forms an interaction loop (Figure 31) where the CRS model may ask zero too many questions before making recommendations. A session finishes whether a user accepts the recommendations or leaves due to his impatience.

(22)

5.1.3 CPR method

In this method we model conversational recommendation as the process of finding a path in user-item-attribute knowledge graph interactively. Figure 32 shows an illustrative example. The vertices in the right graph represent users, items and attributes as well as other relevant entities. An edge between two vertices represent their relation, for example, a user item edge indicates that the user has interacted with the item, and a user attribute edge indicates that the user has affirmed an attribute in a conversation session. A conversation session in our CPR is expressed as a walking in the knowledge graph. It starts from the user vertex, and travels in the graph with the goal to reach one or multiple item vertices the user likes as the destination. Note that the walking is navigated by users through conversation. This means, at each step, a system needs to interact with the user to find out which vertex to go and takes actions according to user's response.

In Figure 32 an illustration of interactive path reasoning in CPR. As the convention of this paper, light orange, light blue, and light gold vertices represents the user, attribute and items respectively. For example, the artist Michael Jackson is an item and the attributes are rock, dance etc.

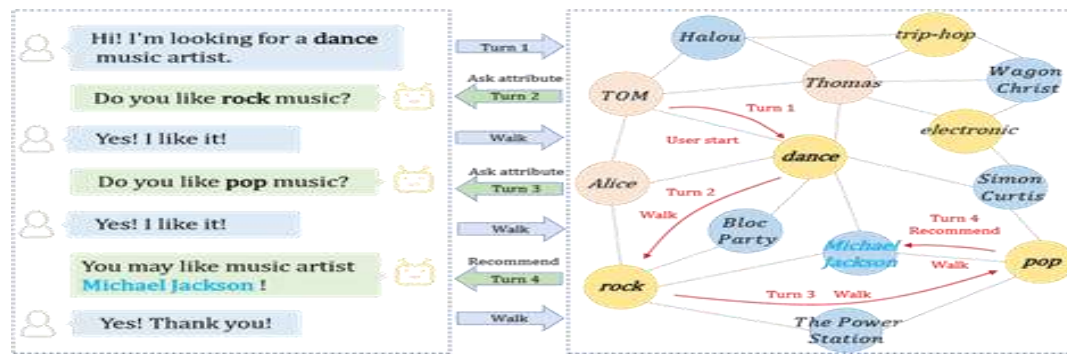


Figure 32 An illustration of interactive path reasoning in CPR

(Interactive Path Reasoning on Graph for Conversational Recommendation, Wenqiang Lei¹, Gangyi Zhang, Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, Tat-Seng Chua, 2020)

The immediate gain of such a method is that it models conversational recommendation as an interactive path reasoning problem on the graph with each step confirmed by the user, limits the candidate attributes to the adjacent attributes of the current vertex resulting in the reduction of the candidate space and offer an aesthetically appealing framework which demonstrates the natural combination and mutual promotion of conversation system and recommendation system.

In the course of the validation procedure about the effectiveness of CPR method we make use of multi-round conversational recommendation (MCR) scenario conducting experiments on the Yelp and Last FM datasets, comparing SCPR with CRS and EAR methods which also use the information of user, item and attribute but does not use knowledge graph. In MCR scenario, as it is the most realistic setting in research, the system is free to ask attributes or make recommendation multiple times.

Specifically, an item v is associated with a set of attributes P_v . The attributes broadly cover various descriptions as long as it can describe certain properties of an item. A conversation session starts on the user side, which initializes the attribute p_0 by specifying an attribute the user likes (e.g., I like some dance music). Next, the CRS is free to ask his preference on an attribute selected from the candidate attribute set P_{cand} or recommend items from the candidate item set V_{cand} . Then, the user needs to give feedback accordingly, either accepting or rejecting them. The CRS makes use of such feedback from the user — if the user accepts the asked attribute, the CRS puts it in the preferred attribute set P_u and removes it from P_{cand} . Then the CRS updates V_{cand} to $V_{cand} \cap V_p$, representing the items containing all attribute confirmed by the user in

the session. V_p denote the items containing the attribute p . If he rejects the asked attribute, the CRS removes it from P_{cand} . Based on the updated sets, the CRS takes the next action, i.e., asking or recommending, and repeats the above process. The conversation session ends until the CRS hits the user preferred items or reaches the maximum number of turns T . This process is detailed in Algorithm 1.

Table 1: Main notations used in the paper

u, v, p	User, item, and attribute
P	An active attribute path in the graph
aa_t	An adjacent attribute of the attribute p_t
AA_t	The set of adjacent attributes of the attribute p_t
P_u	The set of attributes confirmed by u in a session
P_{cand}	The set of candidate attributes
V_p	The set of items that contain the attribute p
V_{cand}	The set of candidate items
a	The action of CPR, either a_{ask} or a_{rec}

Algorithm 1 The MCR Scenario

Input: user u , all attributes P , all items V , the number of items

to recommend k , the maximum number of turns T ;

Output: recommendation result: success or fail;

- 1: User u specifies an attribute p_0 ;
- 2: Update: $P_u = \{p_0\}$; $P_{cand} = P \setminus p_0$; $V_{cand} = V_{p_0}$
- 3: for turn $t = 1, 2, 3 \dots T$ do
- 4: Select an action a
- 5: if $a == a_{ask}$ then
- 6: Select the top attribute p from P_{cand}
- 7: if u accepts p_t then


```

8:         Update:  $P_u = P_u \cup p$ ;  $V_{cand} = V_{cand} \cap V_p$ 
9:     Update:  $P_{cand} = P_{cand} \setminus p$ 
10:  else [  $a == a_{rec}$  ]
11:     Select the top-k items  $V_k$  from  $V_{cand}$ 
12:     if User accepts  $V_k$  then
13:         Recommendation succeeds; Exit.
14:     else [User rejects  $V_k$  ]
15:         Update:  $V_{cand} = V_{cand} \setminus V_k$ 
16: Recommendation fails; Exit

```

(23)

A graph uses vertices to represent entities and edges to represent the relationships between entities. Specifically, a graph G is defined as a set of triplets $\{(h,r,t)\}$, indicating a certain relation r exists between the head entity h and the tail entity t . The relations between each types of entities can vary a lot depending on specific datasets. CPR maintains an active path P , comprising the attributes confirmed by a user (i.e., all attributes in P_u) in the chronological order, and exploring on the graph for the next adjacent attribute vertex to walk. CPR does not visit the attributes that have been visited before and does not perform the walking over all types of vertices. As a result it emphasizes the importance of the attributes as explicit reasons for recommendation and it makes the walking process more concise, eliminating the uncertainty in an unnecessarily long reasoning path which might lead to error.

In Figure 33 CPR framework overview. It starts from the user u_0 and walks over adjacent attributes, forming a path (the red arrows) and eventually leading to the desired item. The policy network (left side) determines whether to ask an attribute or recommend items in a turn. Two reasoning functions f and π score attributes and items, respectively.

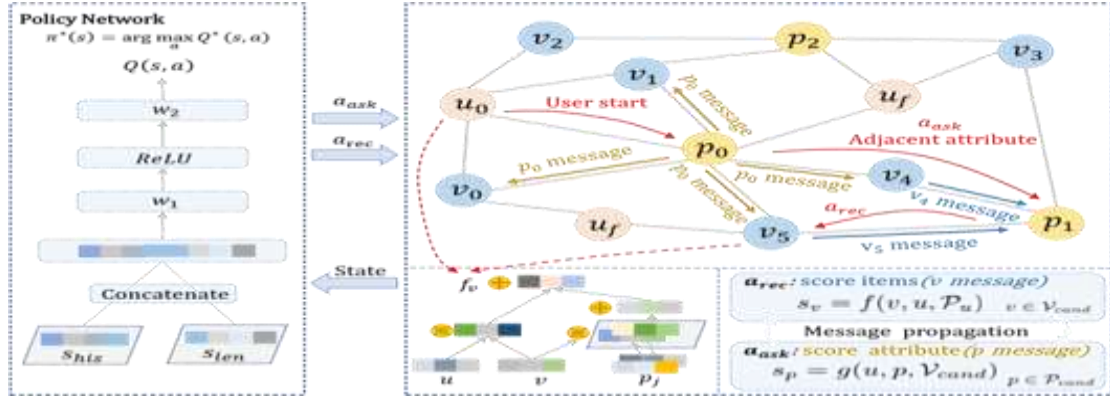


Figure 33 CPR framework overview

(Interactive Path Reasoning on Graph for Conversational Recommendation, Wenqiang Lei¹, Gangyi Zhang, Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, Tat-Seng Chua, 2020)

In Figure 33 when a user initializes his preferred attribute p_0 (i.e., $P = p_0$), the CPR propagates messages from p_0 to its directly connected items (i.e., v_0, v_1, v_4, v_5) to score these items. The scoring function for each item is $s_v = f(v, u, P_u)$ where s_v is a scalar indicating the recommendation score of item v in the current conversation session, and P_u denotes the attributes confirmed by u in the session. Then the candidate items in turn propagate messages to the candidate attributes (the light blue arrows) with each candidate attribute scoring function ($p \in P_{cand}$) to be $s_p = g(u, p, V_{cand})$

Then the output action space of the policy function contains two choices: a_{ask} or a_{rec} , indicating whether to perform top-k recommendations or to ask an attribute in this turn. If the decision is a_{ask} , we directly take highest-scored attribute from P_{cand} , where the score is s_p . Otherwise, we recommend top-k items from V_{cand} according to the score of s_v and the transition step will be triggered after the user confirms an asked attribute p_t .

We make use of two datasets LastFM and Yelp, splitting each one of them for training, validation and testing in a ratio of 7:1.5:1.5 and the rewards to train the policy network are: $r_{rec_suc}=1, r_{rec_fail}=-0.1, r_{ask_suc}=0.01, r_{ask_fail}=-0.1, r_{quit}=-0.3$

For our validation experiment we use, except CRM method, four additional models: Max Entropy. This method follows a rule-based protocol to ask and recommend. When asking question, it always chooses an attribute with the maximum entropy within the current candidate item set.

Abs Greedy. This method serves as a baseline where the model only recommends items and updated itself, until it finally makes successful recommendation.

CRM. This is a CRS model which records user’s preference into a belief tracker, and uses reinforcement learning (RL) to find the policy to interact with the user.

EAR. This is the state-of-the-art method on MCR setting and proposed a three stage solution called Estimation–Action– Reflection which emphasizes on the interaction between conversation component and recommendation component.

We use success rate (SR@t) to measure the cumulative ratio of successful recommendation by turn t. We also use average turns (AT) to record the average number of turns for all session (if a session still fails in the last turn T, we count the turn for that session as T). Therefore, the higher SR@t indicates a higher performance at a specific turn t, while the lower AT means an overall higher efficiency

In Table 4 the Success Rate @ 15 and Average Turn. Bold number represents the improvement of SCPR over existing models is statistically significant ($p < 0.01$) (RQ1)

	LastFM		Yelp	
	SR@15	AT	SR@15	AT
Abs Greedy	0.222	13.48	0.264	12.57
Max Entropy	0.283	13.91	0.921	6.59
CRM	0.325	13.75	0.923	6.25
EAR	0.429	12.88	0.967	5.74
SCPR	0.465	12.86	0.973	5.67

Table 4 Success Rate and Average Turn

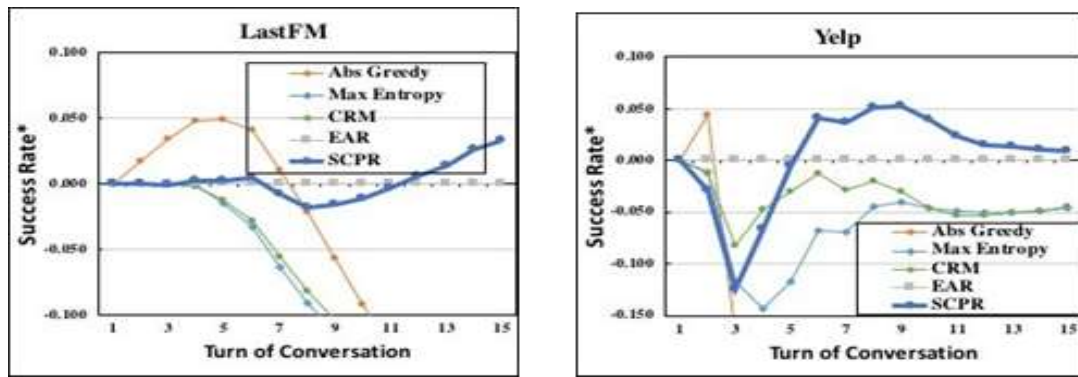


Figure 34 Success Rate * of compared methods at different turns on LastFM and Yelp (RQ1).

(Interactive Path Reasoning on Graph for Conversational Recommendation, Wenqiang Lei¹, Gangyi Zhang, Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, Tat-Seng Chua, 2020)

By noticing table 2 SCPR model achieves significantly higher SR and less AT than the rest models, demonstrating SCPR method's superior performance in usage. It utilizes the graph adjacent attribute constraint to extinguish many irrelevant attributes to ask, something that becomes especially helpful when there are a large number of attribute and has a more dedicated RL model with smaller action space. Looking Figure 3 CRM, in Yelp dataset, may outperform SCPR on first few rounds, but it falls behind in future rounds. Abs Greedy can achieve the best results on the first few turns but immerses in further turns.

(23)

SECOND PART

6. Brief historical retrospective of databases

The term database appeared in the early 1960s and was used to denote the collections of data managed by large time-sharing computing systems, replacing the term <<integrated data processing>> that had been in use since the previous decade. In 1970 E. F. Codd, with his pioneering work, founded the relational database model which was going to change immediately and dramatically the way of managing and usability of databases. By the use of the above mentioned term we mean a collection of logically related items together with their description, designed to meet the information needs of an organization. Databases therefore make it possible to organize and store data in the computer, enabling this way, the processing and extraction of the desired information.

The model of a database describes the structure of the database and how it can be used, including the basic operations of retrieving and updating data.

Data models can be categorized, according to the form of concepts they contain for description, to:

- The high-level or conceptual models contain a level of abstraction that approximates the way a layman perceives the data
- The low-level or physical data models describe how data is stored in the computer.
- The logical or representational data models contain concepts of a medium level of abstraction, so that they are not too far from the way the user thinks, nor from the way are they stored in the computer.

Historically, three basic logical database models have been proposed. The first is the hierarchical model which considers records as nodes in trees. The tree offers the prioritization of data required by the specific model. The second is the network model which is an improved version of the hierarchical one. The main difference with the first one is that there is no restriction that each record has only one parent. Each record can have one or more parents. In this way, a network is created that connects the database records. The latter is the relational model which represents the database as an unordered

collection of relations where each relation is an unordered collection of tuples and each tuple is an unordered collection of attributes or predicates.

The main feature of the 80s was the dominance of relational database management systems, something that continues to this day. (24)

With the advent of the new millennium and the subsequent development of the Internet which resulted in the genesis of Big Data, existing databases have been forced to deal with new issues of managing and effectively exploiting this kind of data.

6.1 SQL Databases

Every relational database management system consists of a storage and a management component. The storage component stores both data and the relationships between pieces of information in tables. In addition to tables with user data from various applications, it contains the predefined system tables necessary for database operation. These contain descriptive information and can be queried but not manipulated by users. The management component's most important part is the relational data definition, selection, and manipulation language SQL. This component also contains service functions for data restoration after errors, for data protection, and for backup. (25). The two most extensively used relational databases are MySQL and Oracle. MySQL is more popular with the websites. It is a light weight system which is extremely fast but Oracle is majorly used in case of large database requirement like Banking, Insurance, ERP and finance companies. It is used to solve complex problems and supports large OLTP environments. (26). These databases have to be refined periodically to remove any kind of redundant, inconsistent or dirty data so as to perform effectively, also their data structure follow the ACID properties (Atomicity, Consistency, Isolation, Durability) and use vertical scalability, which means that when the volume of data is being expanded, there could be expand just the storage capacity and computing power of existing node, for example, the capacity of CPU, the RAM and the SSD of the database server.

With the explosion of data volume, SQL-based data querying lose efficiency, and in particular, managing larger databases has become a major challenge. In addition, relational databases exhibit a variety of limitations in meeting the recent Big Data analytics requirement in businesses. While clusters-based architecture has emerged as a solution for large databases, SQL is not designed to suit clusters and this miss match

has led to think of alternate solutions. There are miss matches between persistent data model and in memory data structures, and servers based on SQL standards are now prone to memory footprint, security risks and performance issues. (27) (28)

6.2 NoSQL Databases

The term NoSQL was first used in 1998 for a database that (although relational) did not have an SQL interface. It became of growing importance during the 2000s, especially with the rapid expansion of the internet. The growing popularity of global web services saw an increase in the use of web-scale databases, since there was a need for data management systems that could handle the enormous amounts of data (sometimes in the petabyte range and up) generated by web services. Although NoSQL primarily reads as databases that provide no SQL access, the acronym is commonly defined as “not only SQL”. (Book, SQL & NoSQL Databases, 2019, Andreas Meier Michael Kaufmann) The major challenge with the growing data is its no uniformity. Due to this problem, in recent years, a nonrelation database is needed to scale the growing need of industry and at the same time, must be highly efficient. This gave rise to NoSQL databases which are highly scalable, efficient and can store large amount of data. To deal with this non-uniformity of data a fresh thought was given to the storage of data, leading to the creation of NoSQL. They do not follow the general table/row/column approach which is practiced by all RDBMSs. NoSQLs are primarily called distributed or non-relational databases. They support horizontal scalability, so to scale number of servers are increased rather than upgrading hardware of the system which happens in RDBMS where vertical scalability is performed. (28)

Non-relational databases may primarily be classified on the basis of way of organizing data as follows:

6.2.1 Key Value Stores

It allows the app-developer to store schema-less data. This data consists of a key which is represented by a string and the actual data which is the value in key-value pair. The data can be any primitive of programming language, which may be a string, an integer or an array or it can be an object. Thus it loosens the requirement of formatted data for storage, eliminating the need for fixed data model. Key-value model is a schema-less database which is implemented using a hash table where keys are stored as indexes and a pointer that holds the actual data. This structure creates the 'key-value' pair as the

model itself is named. The hash tables are suitable for lookups for simple or complex values in extremely large datasets. The data in the key-value database are stored in the form of rows as structured data but also can be stored as JSON or some other self-describing data format as semi structured data.

6.2.2 Document Store

Document Store, also commonly known as “Document Oriented Database”, is basically a computer program used for storing, retrieving, updating data stored in database. The underlying storage structure used in such databases is a ‘document’. Each document is represented by a unique key which is a string (URI or path). An API or a query language is provided for fast retrieval of documents on the basis of its content. The document based model perfectly handles all types of data including structured, semi-structured and unstructured data. The documents in a collection should be similar, but a document can contain attributes that are not necessarily need to have other documents in that collection.

6.2.3 Graph Database

Graph databases are schema-less databases which use graph data structures along with nodes, edges and certain properties to represent data. Nodes may represent entities like people, business or any other item similar to what objects represent in any programming language. Properties designate any pertinent information related to nodes. On the other hand; edges relate a node to other node or a node to some property. One can obtain some meaningful pattern or behavior after studying the interconnection between all three viz. nodes, properties and edges. This model can support complex data queries for a relatively short period of time, also can support ACID properties and the rollback feature which ensures the consistency of data. This type of database is used when the importance is given on the relationships between data than the data itself.

6.2.4 Column Oriented Databases

Column Store Databases, unlike Row Databases, store their data in the form of columns. It serializes all the values of one column together and so on. Column-oriented databases are comparatively efficient than row oriented one’s when new values for a column are entered for all rows at once as column data can be written efficiently and replace old data without altering any other columns for the rows. Column-oriented

model is a wider concept of key value architecture, organized by columns. This model is a composite approach to relational databases and key-value model schema. The data are stored in column families and rows. Every row has a row-key and a row may contain many columns. A row can have a different number of columns and in case of nested columns inside another column, those columns are called super columns. This database type works very well with complex datasets as a result of its scalability.

6.2.5 Object Oriented Databases

Object Oriented Databases also commonly known as OODBMS), is a database system. It stores its data in the form of objects. This feature supports inheritance and hence reusability similar as in object oriented programming. Object oriented database can be considered as a combination of object oriented programming (OOP) and database principles. Object data store offers all the features of OOP such as data encapsulation, polymorphism and inheritance. The class, objects, and class attributes in such databases are comparable to a table, tuple and columns in a tuple in RDBMS respectively. Each object has an object identifier which can be used to uniquely represent that object

(26) (29) (30)

6.3 Evaluation of differences between SQL and NoSQL Databases

6.3.A Scalability and performance

On the one hand Relational databases (SQL databases) use vertical scalability, which means that when the volume of data is being expanded, there could be expand just the storage capacity and computing power of existing node, for example, the capacity of CPU, the RAM and the SSD of the database server. This kind of scalability is expensive because of grater hardware failure risk, hardware costs in means of future upgradability (hardware became older and the support is less, vendors may have some requests, hardware and software limitations, etc.), so the overall implementation cost will increase with data growth. On the other hand NoSQL databases use horizontal scalability which means that when the volume of data is rapidly growing and the volume of data is large the system expand by adding more nodes for data storage and processing power. By following this procedure, the horizontal scalability of the system is a cheaper solution than the vertical scalability. Inherently the NoSQL databases support the auto – sharding feature by distributing data on different servers, which

increases the performance of the database. NoSQL databases are, therefore, fit in the current world, where outward scalability is replacing upward scalability. Additionally, NoSQL databases are used for handling big data applications, which RDBMSs cannot manage (31)

6.3.B Flexibility

The SQL databases have a static database schema that should be pre-defined before data injection and should support structured data. If there is a need to change the schema, with pre-existing data, there is a huge problem and a modification of the database schema or tables should be considered precisely, because that modification can cause service failure, decrease performance, or may need maintenance and further investments to modify application modules. While on the other hand, NoSQL databases have a dynamic schema and not necessarily need to be pre-defined. NoSQL databases can easily accommodate changes in data type / structure due to its dynamic schema design. The NoSQL databases because of their data modeling are used for agile and scalable environments which will be continuously developing and evolving. One more important point is that SQL databases handle just well – structured data but NoSQL databases handle every kind of data including their well – structured, semi – structured and unstructured data.

6.3.C Query language

Relational databases use a standard query language known as Structured Query Language (SQL). This query language is a powerful one and can handle complex queries through a standardized interface. SQL databases have portability since SQL is compatible with a broad range of computer programs and can be used for quick communication with other databases. On the other side, the NoSQL databases do not have a standardized language to query and manage data. However, every NoSQL database management system vendor has created their own query language but there is a lack of creating complex queries such as aggregation on NoSQL databases. Many NoSQL systems do not provide join operation as part of their query language, so the joins need to be implemented on the application side. There is a need for a common query language like SQL which can be used for all NOSQL databases.

6.3.D Security

The security is an important issue for a DBMS. The relational databases have very secure mechanisms which ensure the security of the services and support parameterized queries and prepared statements to prevent SQL injection attacks. By separating SQL code from user-supplied input, databases can protect against malicious SQL commands.

Since the feature of sharding is considered the key to success of NoSQL databases by distributing data over servers this probably has impact in data security as the most difficult challenge for NoSQL databases. There is a concern, how the confidentiality, privacy and the security of the data are guaranteed from these systems. Most of the NoSQL databases do not have secure client-server communication and do not provide these mechanisms that can ensure security. There are some key factors that should be considered when dealing with the security of databases. Those factors are authentication, access control, secure configurations, data encryption, and auditing. To ensure the authentication, authorization, and auditing there should be external methods to perform the operation and should be implemented based on the NoSQL database used. It is the same way in defining the access control of the users, some of the NoSQL databases provide access control from the system, but some of them do not ensure this kind of mechanism and need to implement it from the third party. NoSQL databases are less mature compared to RDBMSs, which have been existing for an extended period, thus becoming more stable and richly functional.

6.3.E Data management - Storage and Access

In relational databases, data stored are highly normalized and very clean. The data redundancy is avoided in a remarkable way using normalization by slicing data in small logical tables and preventing duplication. In this way, happens the improvement and usage of storage in a reasonable manner.

NoSQL database are stored in collections without relationships and normalization between each other so this could contain data redundancy. NoSQL databases practice the data replication of the database between clustered servers, in order to prevent data loss and to guarantee the security of data. The replication process is done in two ways: master-slave and master-master. Master-slave replication allows the slave to take a copy of the data just for read, while the master holds the permission to write and read the data, so this way guarantees the consistency of data. While master – master

replication allows reads and writes to any of the copies and this may lose the consistency. In the relational databases, a replication is when the whole database is replicated in every site of the distributed system and as a result the availability of data is improved but the performance of the database operations will be decreased obviously. Most of the Non-Relational databases are open source software and though well appreciated, it compromises in reliability as nobody is responsible in times of failures. Many Non-relational databases provide BASE properties and sacrifice conventional ACID properties as a step to increase performance. This could mean than non-relational databases compromise on consistency within the database.

(29)

6.4 Graphs

Formally, a graph is just a collection of *vertices* and *edges*—or, in less intimidating language, a set of *nodes* and the *relationships* that connect them. Graphs represent entities as nodes and the ways in which those entities relate to the world as relationships. This general-purpose, expressive structure allows us to model all kinds of scenarios, from the construction of a space rocket, to a system of roads, and from the supply chain or provenance of foodstuff, to medical history for populations, and beyond.

A *graph database management system* (henceforth, a *graph database*) is an online database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. Graph databases are generally built for use with transactional (OLTP) systems. Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind.

There are two properties of graph databases we should consider when investigating graph database technologies:

The underlying storage

Some graph databases use *native graph storage* that is optimized and designed for storing and managing graphs. Not all graph database technologies use native graph storage, however. Some serialize the graph data into a relational database, an object-oriented database, or some other general-purpose data store.

The processing engine

Some definitions require that a graph database use *index-free adjacency*, meaning that connected nodes physically “point” to each other in the database. Here we take a slightly broader view: any database that from the user’s perspective *behaves* like a graph database (i.e., exposes a graph data model through CRUD operations) qualifies as a graph database. We do acknowledge, however, the significant performance advantages of index-free adjacency, and therefore use the term *native graph processing* to describe graph databases that leverage index-free adjacency.

Relationships are first-class citizens of the graph data model. This is not the case in other database management systems, where we have to infer connections between entities using things like foreign keys or out-of-band processing such as map-reduce. By assembling the simple abstractions of nodes and relationships into connected structures, graph databases enable us to build arbitrarily sophisticated models that map closely to our problem domain. The resulting models are simpler and at the same time more expressive than those produced using traditional relational databases and the other NOSQL (Not Only SQL) stores. (32)

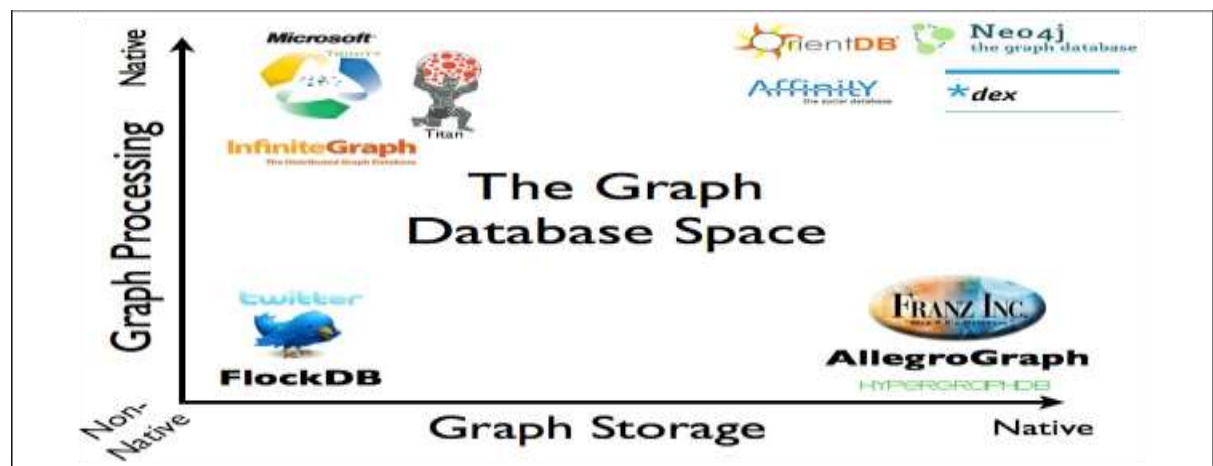


Figure 35 Graph Database on the market today

(Ian Robinson, Jim Webber & Emil Eifrem. *Graph Databases*. Menlo Park, California : O'REILLY, 2013)

A *graph compute engine* is a technology that enables global graph computational algorithms to be run against large datasets. Graph compute engines are designed to do things like identify clusters in your data, or answer questions such as, “how many relationships, on average, does everyone in a social network have?”

Because of their emphasis on global queries, graph compute engines are normally optimized for scanning and processing large amounts of information in batches, and in that respect they are similar to other batch analysis technologies, such as data mining and OLAP, in use in the relational world. Whereas some graph compute engines include a graph storage layer, others (and arguably most) concern themselves strictly with processing data that is fed in from an external source, and then returning the results for storage elsewhere.

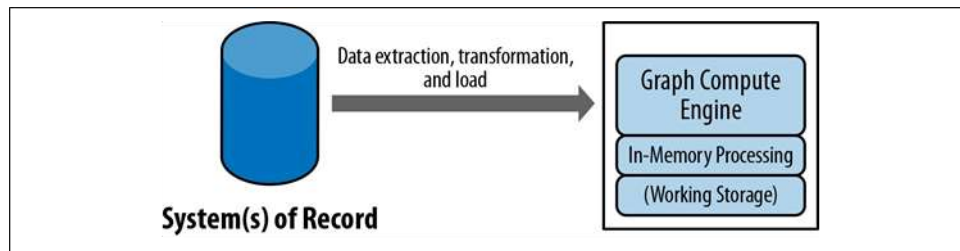


Figure 36 A high-level view of a typical graph compute engine deployment

(Ian Robinson, Jim Webber & Emil Eifrem. *Graph Databases*. Menlo Park, California : O'REILLY, 2013)

A variety of different types of graph compute engines exist. Most notably there are in memory/ single machine graph compute engines like Cassovary and distributed graph compute engines like Pegasus or Giraph. Most distributed graph compute engines are based on the Pregel white paper, authored by Google, which describes the graph compute engine Google uses to rank pages.

(32) (33) (34)

6.4.1 Types of Graph Algorithms

6.4.1.1 Pathfinding and Graph Search Algorithms

Graph search algorithms explore a graph either for general discovery or explicit search. These algorithms carve paths through the graph, but there is no expectation that those paths are computationally optimal. We will cover Breadth First Search and Depth First Search because they are fundamental for traversing a graph and are often a required first step for many other types of analysis. Pathfinding algorithms build on top of graph search algorithms and explore routes between nodes, starting at one node and traversing through relationships until the destination has been reached. These algorithms are used

to identify optimal routes through a graph for uses such as logistics planning, least cost call or IP routing, and gaming simulation. Specifically, the pathfinding algorithms we'll cover are:

- Shortest Path, with two useful variations (A* and Yen's): finding the shortest path or paths between two chosen nodes
- All Pairs Shortest Path and Single Source Shortest Path: for finding the shortest paths between all pairs or from a chosen node to all others
- Minimum Spanning Tree: for finding a connected tree structure with the smallest cost for visiting all nodes from a chosen node
- Random Walk: because it's a useful preprocessing/sampling step for machine learning workflows and other graph algorithms.

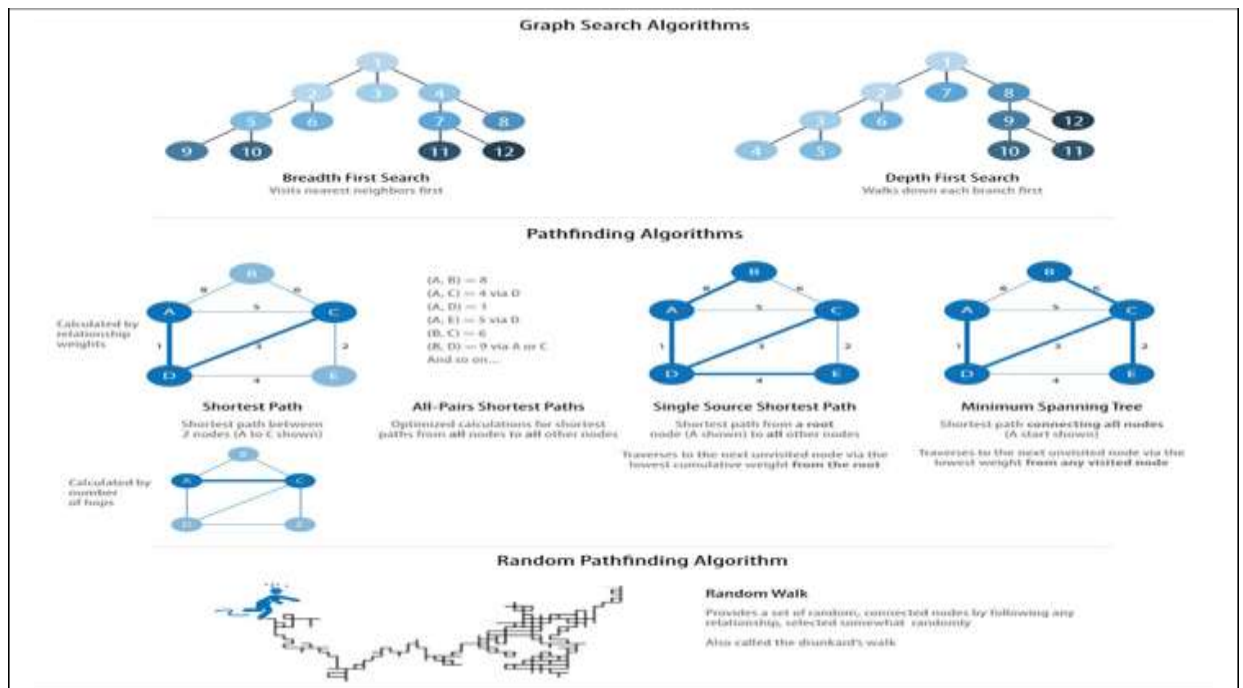


Figure 37 Pathfinding and search algorithms

(35) (36)

6.4.1.2 Centrality Algorithms

Centrality algorithms are used to understand the roles of particular nodes in a graph and their impact on that network. They're useful because they identify the most important nodes and help us understand group dynamics such as credibility, accessibility, the speed at which things spread, and bridges between groups. Although many of these

algorithms were invented for social network analysis, they have since found uses in a variety of industries and fields. We'll cover the following algorithms:

Degree Centrality as a baseline metric of connectedness

Closeness Centrality for measuring how central a node is to the group, including two variations for disconnected groups

Betweenness Centrality for finding control points, including an alternative for approximation

PageRank for understanding the overall influence, including a popular option for personalization

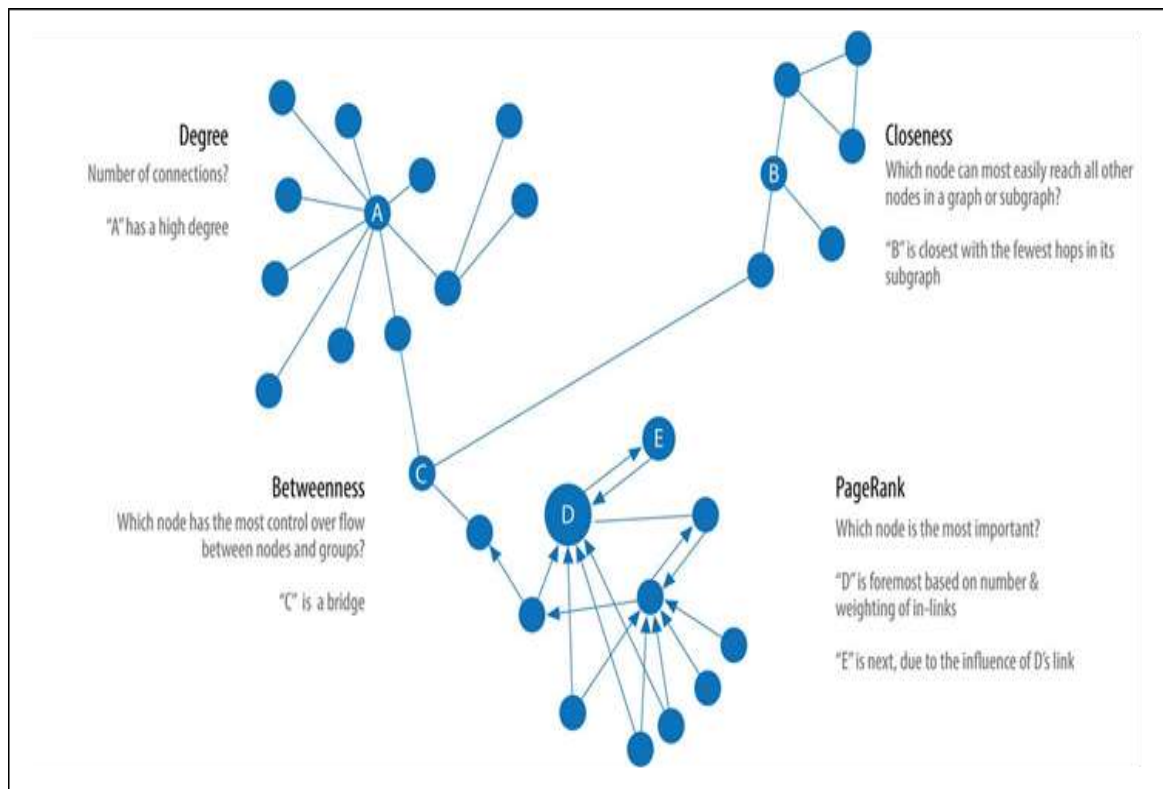


Figure 38 Representative centrality algorithms and the type of questions they answer

(35) (37)

6.4.1.3 Community Detection Algorithms

Community formation is common in all types of networks, and identifying them is essential for evaluating group behavior and emergent phenomena. The general principle in finding communities is that its members will have more relationships within the group than with nodes outside their group. Identifying these related sets reveals

clusters of nodes, isolated groups, and network structure. This information helps infer similar behavior or preferences of peer groups, estimate resiliency, find nested relationships, and prepare data for other analyses. Community detection algorithms are also commonly used to produce network visualization for general inspection. We'll provide details on the most representative community detection algorithms:

- Triangle Count and Clustering Coefficient for overall relationship density
- Strongly Connected Components and Connected Components for finding connected clusters
- Label Propagation for quickly inferring groups based on node labels
- Louvain Modularity for looking at grouping quality and hierarchies

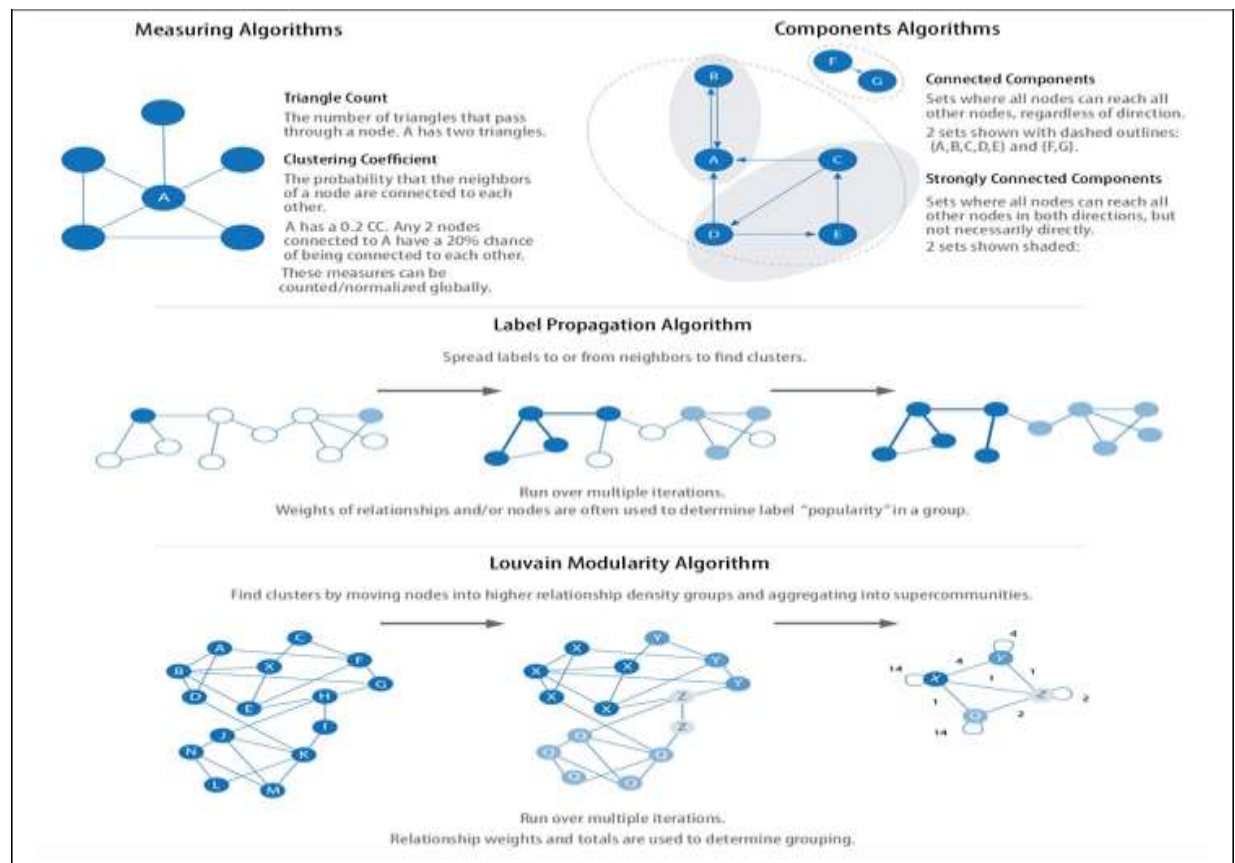


Figure 39 Representative community algorithms

(35) (38)

THIRD PART

7. Platforms

Traditionally there was a separation between graph compute engines and graph databases, which required users to move their data depending on their process needs:

Graph compute engines

These are read-only, nontransactional engines that focus on efficient execution of iterative graph analytics and queries of the whole graph. Graph compute engines support different definition To address the requirements of graph processing, several platforms have emerged. and processing paradigms for graph algorithms, like node-centric (e.g., Pregel, Gather-Apply-Scatter) or MapReduce-based approaches (e.g., PACT). Examples of such engines are Giraph, GraphLab, Graph-Engine, and Apache Spark.

Graph databases

From a transactional background, these focus on fast writes and reads using smaller queries that generally touch a small fraction of a graph. Their strengths are in operational robustness and high concurrent scalability for many users.

7.1 Selecting Platform

Choosing a production platform involves many considerations, such as the type of analysis to be run, performance needs, the existing environment, and team preferences. We use Apache Spark and Neo4j to showcase graph algorithms in this book because they both offer unique advantages.

Spark is an example of a scale-out and node-centric graph compute engine. Its popular computing framework and libraries support a variety of data science workflows. Spark may be the right platform when our:

- Algorithms are fundamentally parallelizable or partitionable.
- Algorithm workflows need “multilingual” operations in multiple tools and languages.
- Analysis can be run offline in batch mode.
- Graph analysis is on data not transformed into a graph format.

- Team needs and has the expertise to code and implement their own algorithms.
- Team uses graph algorithms infrequently.
- Team prefers to keep all data and analysis within the Hadoop ecosystem.

The Neo4j Graph Platform is an example of a tightly integrated graph database and algorithm-centric processing, optimized for graphs. It is popular for building graphbased applications and includes a graph algorithms library tuned for its native graph database. Neo4j may be the right platform when our:

- Algorithms are more iterative and require good memory locality.
- Algorithms and results are performance sensitive.
- Graph analysis is on complex graph data and/or requires deep path traversal.
- Analysis/results are integrated with transactional workloads.
- Results are used to enrich an existing graph.
- Team needs to integrate with graph-based visualization tools.
- Team prefers prepackaged and supported algorithms.

Finally, some organizations use both Neo4j and Spark for graph processing: Spark for the high-level filtering and preprocessing of massive datasets and data integration, and Neo4j for more specific processing and integration with graph-based applications.

7.2 Apache Spark

Apache Spark (henceforth just Spark) is an analytics engine for large-scale data processing. It uses a table abstraction called a DataFrame to represent and process data in rows of named and typed columns. The platform integrates diverse data sources and supports languages such as Scala, Python, and R. Spark supports various analytics libraries, as shown in Figure 40. Its memory-based system operates by using efficiently distributed compute graphs.

GraphFrames is a graph processing library for Spark that succeeded GraphX in 2016, although it is separate from the core Apache Spark. GraphFrames is based on GraphX, but uses DataFrames as its underlying data structure. GraphFrames has support for the Java, Scala, and Python programming languages. In spring 2019, the “Spark Graph: Property Graphs, Cypher Queries, and Algorithms” proposal was accepted (see “Spark Graph Evolution” on page 33). We expect this to bring a number of graph features using the DataFrame framework and Cypher query language into the core Spark project.

However, in this book our examples will be based on the Python API (PySpark) because of its current popularity with Spark data scientists.

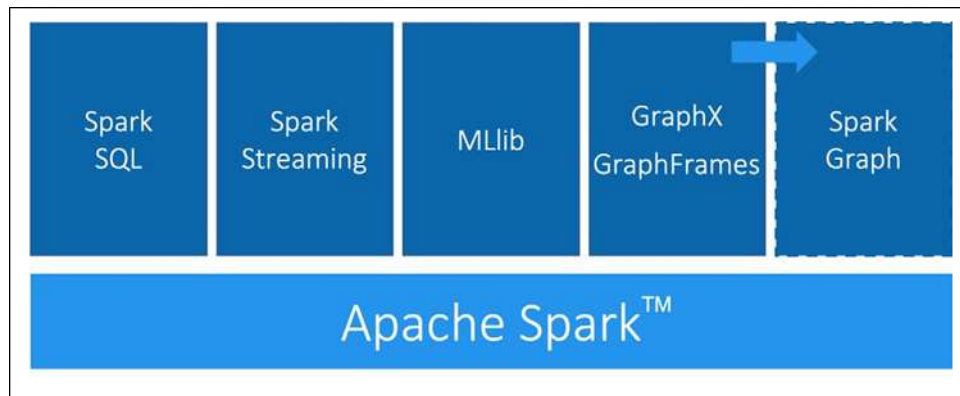


Figure 40 Spark is an open-source distributed and general-purpose clustercomputing framework. It includes several modules for various workloads

(35) (39) (40) (41)

7.2.1 Spark Graph Evolution

The Spark Graph project is a joint initiative from Apache project contributors in Databricks and Neo4j to bring support for DataFrames, Cypher, and DataFramesbased algorithms into the core Apache Spark project as part of the 3.0 release.

Cypher started as a declarative graph query language implemented in Neo4j, but through the openCypher project it's now used by multiple database vendors and an opensource project, Cypher for Apache Spark (CAPS).

In the very near future, we look forward to using CAPS to load and project graph data as an integrated part of the Spark platform. We'll publish Cypher examples after the Spark Graph project is implemented.

This development does not impact the algorithms covered in this book but may add new options to how procedures are called. The underlying data model, concepts, and computation of graph algorithms will remain the same.

7.3 Neo4j Graph Platform

The Neo4j Graph Platform supports transactional processing and analytical processing of graph data. It includes graph storage and compute with data management and analytics tooling. The set of integrated tools sits on top of a common protocol, API, and

query language (Cypher) to provide effective access for different uses, as shown in Figure 41.

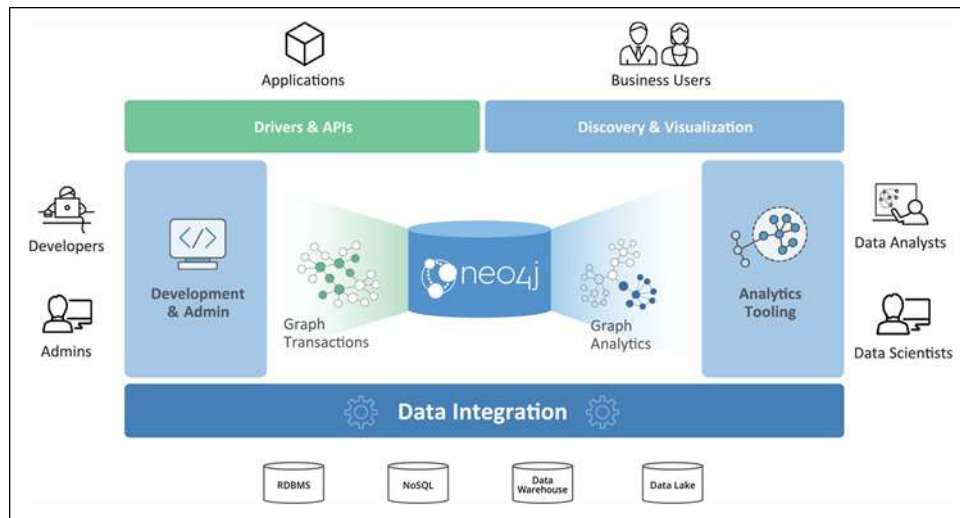


Figure 41 The Neo4j Graph Platform is built around a native graph database that supports transactional applications and graph analytics

The graph algorithm library includes parallel versions of algorithms supporting graph analytics and machine learning workflows. The algorithms are executed on top of a task-based parallel computation framework and are optimized for the Neo4j platform. For different graph sizes there are internal implementations that scale up to tens of billions of nodes and relationships.

Results can be streamed to the client as a tuples stream and tabular results can be used as a driving table for further processing. Results can also be optionally written back to the database efficiently as node properties or relationship types.

(35) (42) (43)

8. Forming – Operating – Evaluating a methodology for AML

8.1 Choosing Data Set

In this master thesis is used as **IBM AMLSim Example Dataset** which is located in kaggle platform.

The AMLSim project is intended to provide a multi-agent based simulator that generates synthetic banking transaction data together with a set of known money

laundering patterns - mainly for the purpose of testing machine learning models and graph algorithms.

This dataset is an example dataset generated from IBM AMLSim.

Content

There are 3 datasets mentioned here: alerts, transactions and accounts.

1. Accounts dataset: Contains the information about all the bank accounts whose transactions are monitored.
2. Alerts dataset: Contains the transactions which triggered an alert according to AML guidelines.
3. Transactions dataset: Contains the list of all the transactions with information about sender and receiver accounts.

8.2 Neo4j

8.2.1 Creating a graph data base in Neo4j

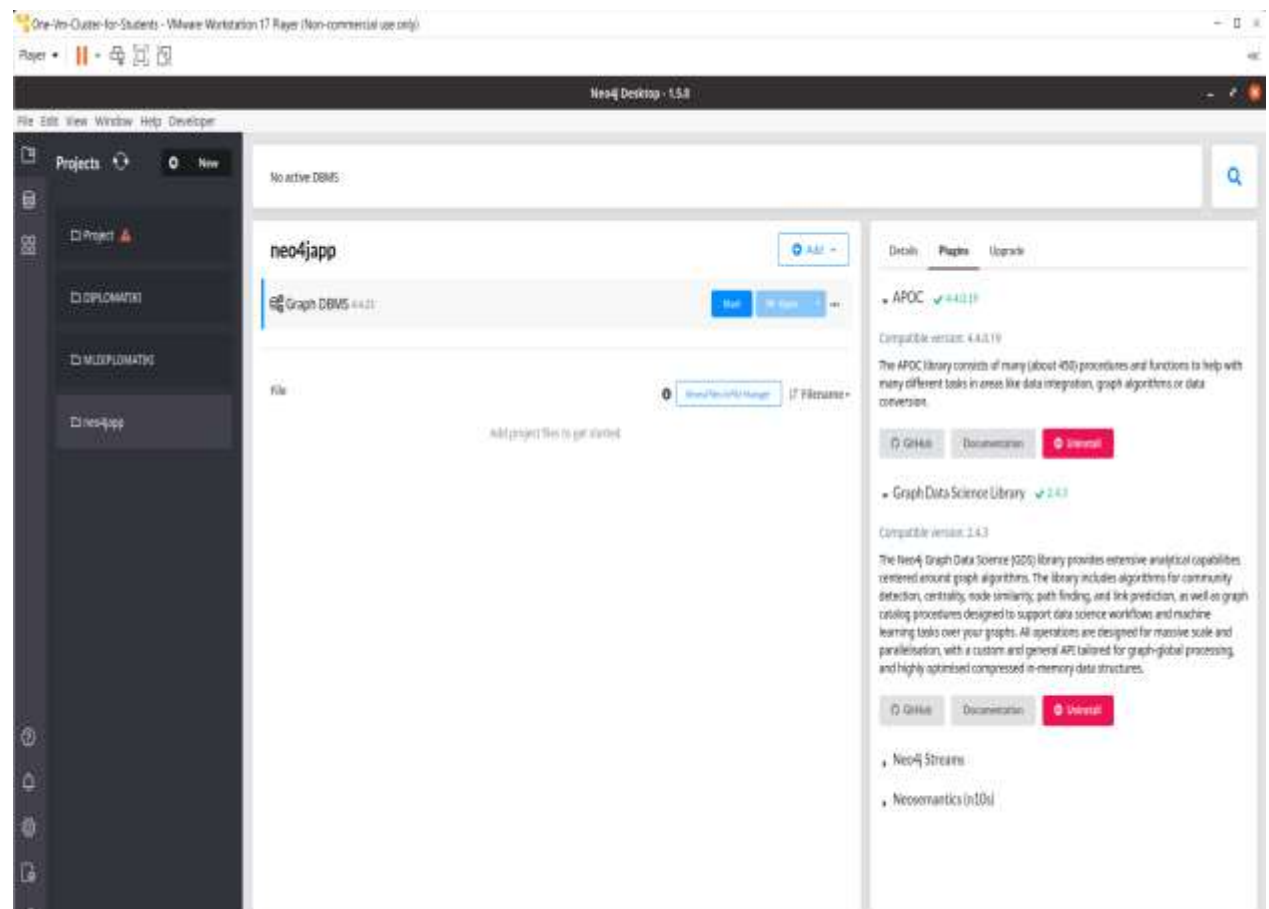


Figure 42 Creating a database and adding APOC and GDS libraries

The APOC library consists of many (about 450) procedures and functions to help with many different tasks in areas like data integration, graph algorithms or data conversion.

The Neo4j Graph Data Science (GDS) library provides extensive analytical capabilities centered around graph algorithms. The library includes algorithms for community detection, centrality, node similarity, path finding, and link prediction, as well as graph catalog procedures designed to support data science workflows and machine learning tasks over your graphs. All operations are designed for massive scale and parallelization, with a custom and general API tailored for graph-global processing, and highly optimized compressed in-memory data structures.

Parameter settings

```
dbms.directories.import=import
```

```
dbms.security.allow_csv_import_from_file_urls=true
```

dbms.security.procedures.unrestricted=jwt.security.*,apoc.*,gds.*,dbms.components.*

8.2.2 Adding files

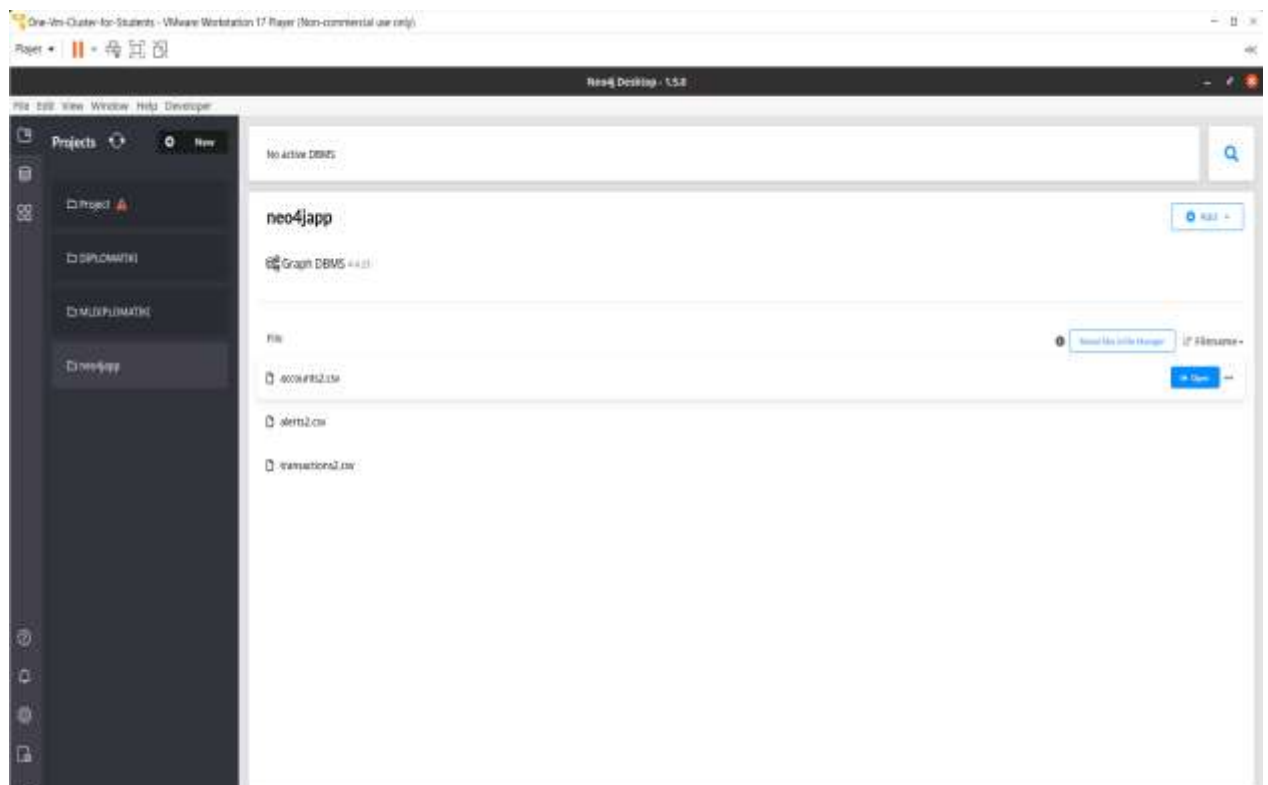


Figure 43 Adding csv files

8.2.3 Forming node alerts

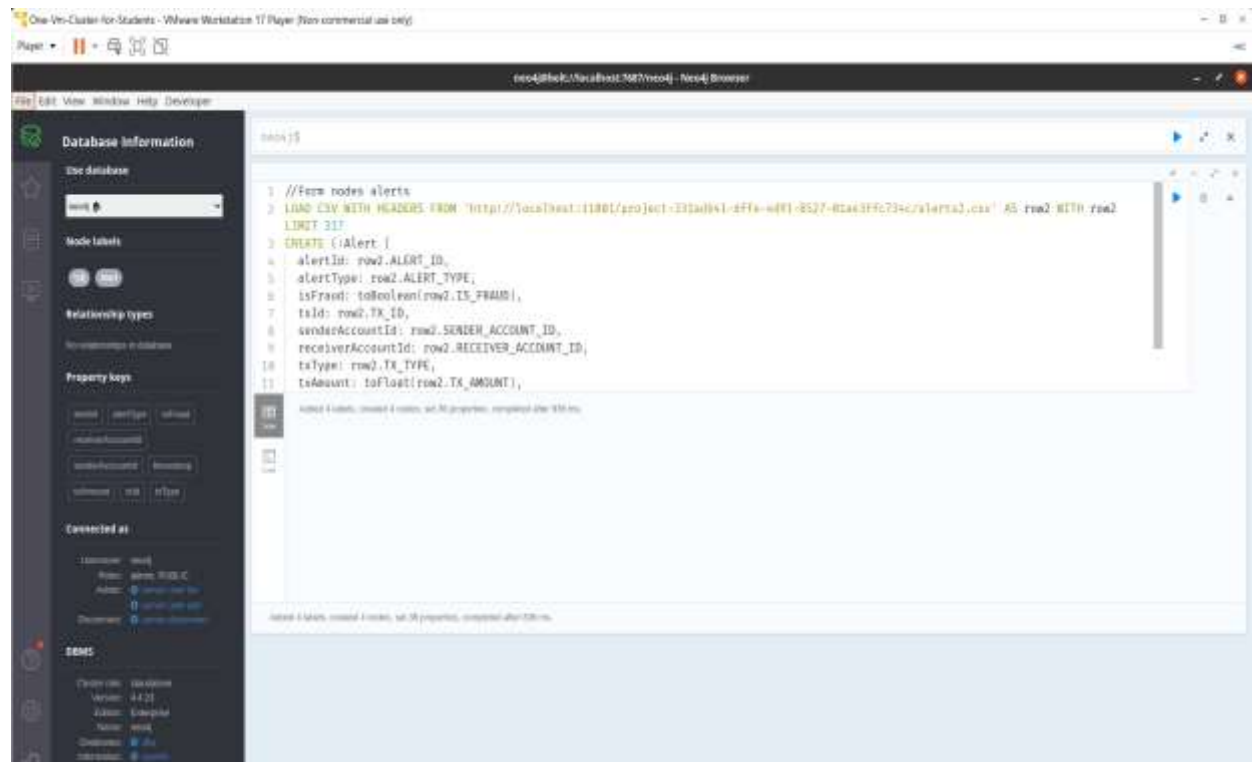


Figure 44 Creating vertex alerts

7.2.4 Forming node accounts

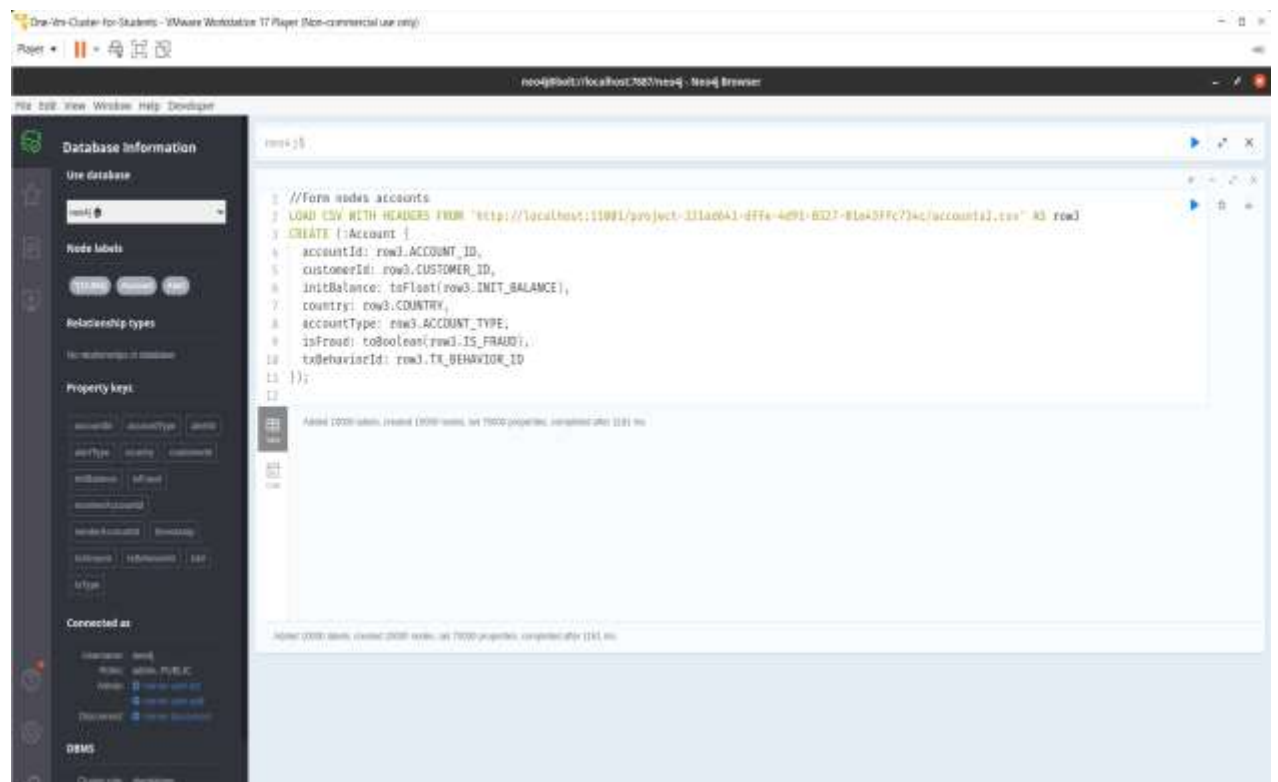


Figure 45 Creating vertex accounts

8.2.4 Forming relationships between transactions and accounts

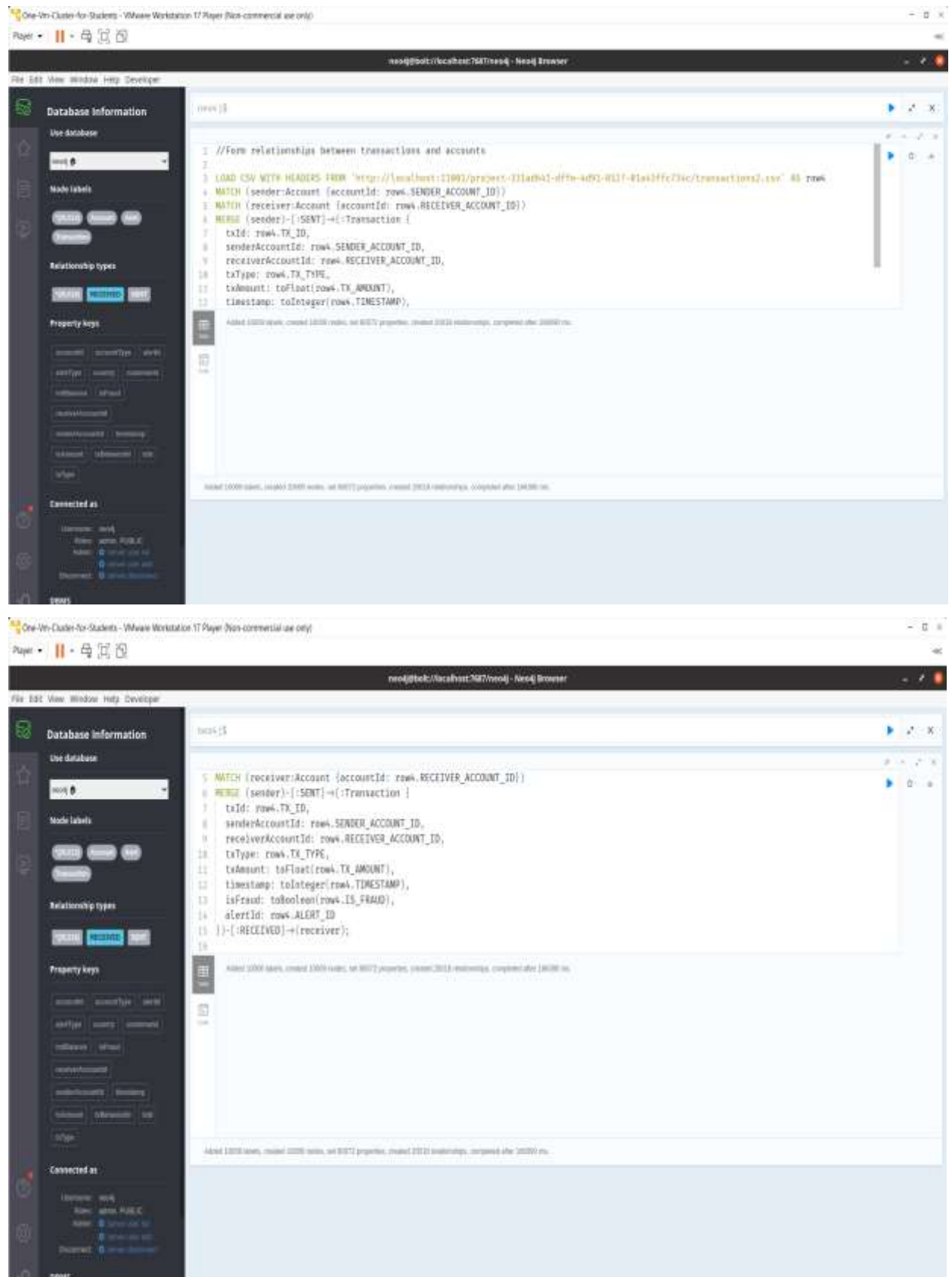


Figure 46 Creating relationships transactions-accounts

8.2.5 Forming relationships between transactions and alerts

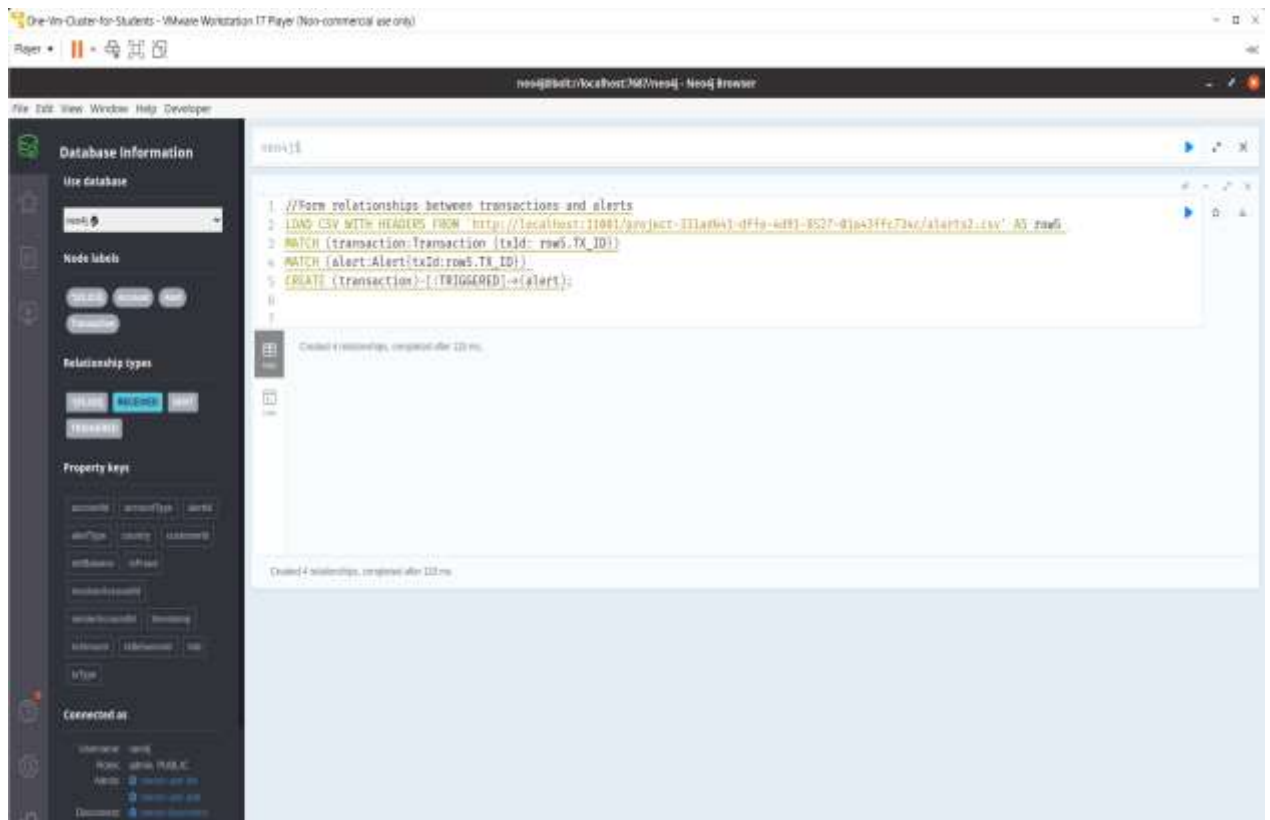


Figure 47 Creating relationships transactions-alerts

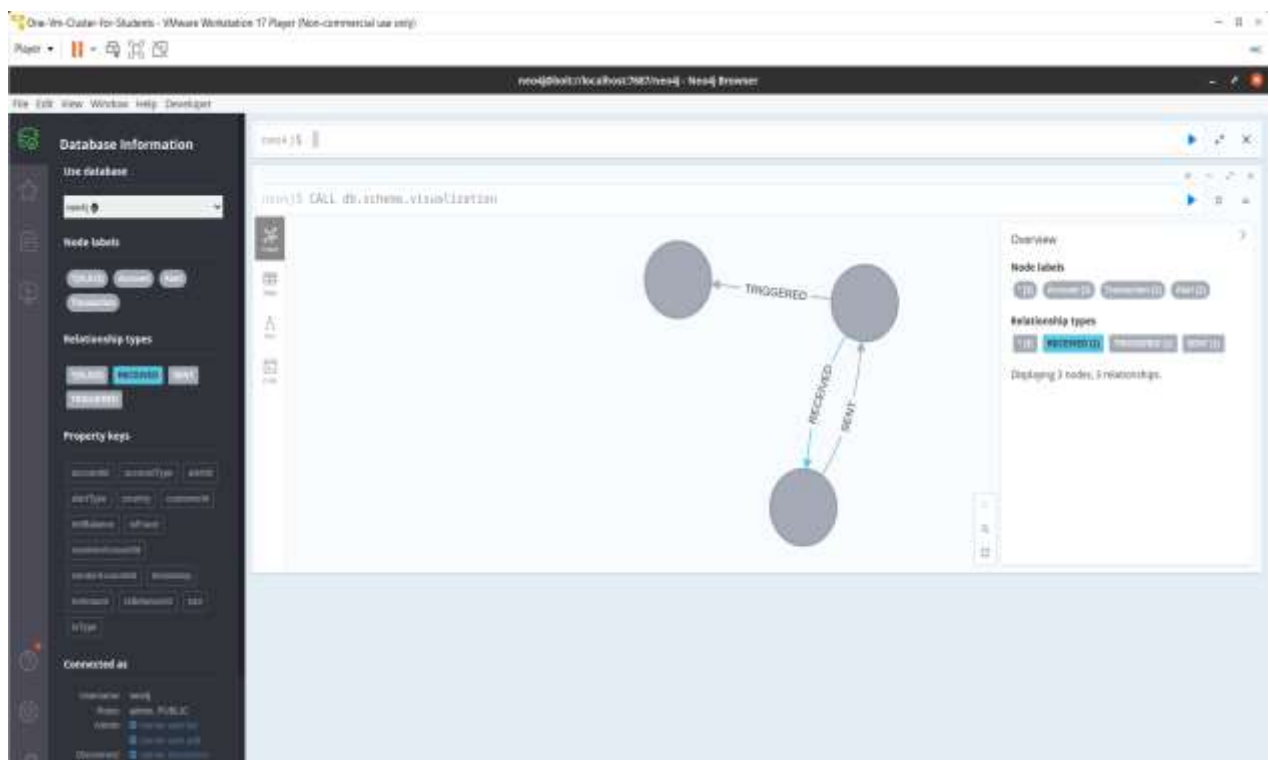
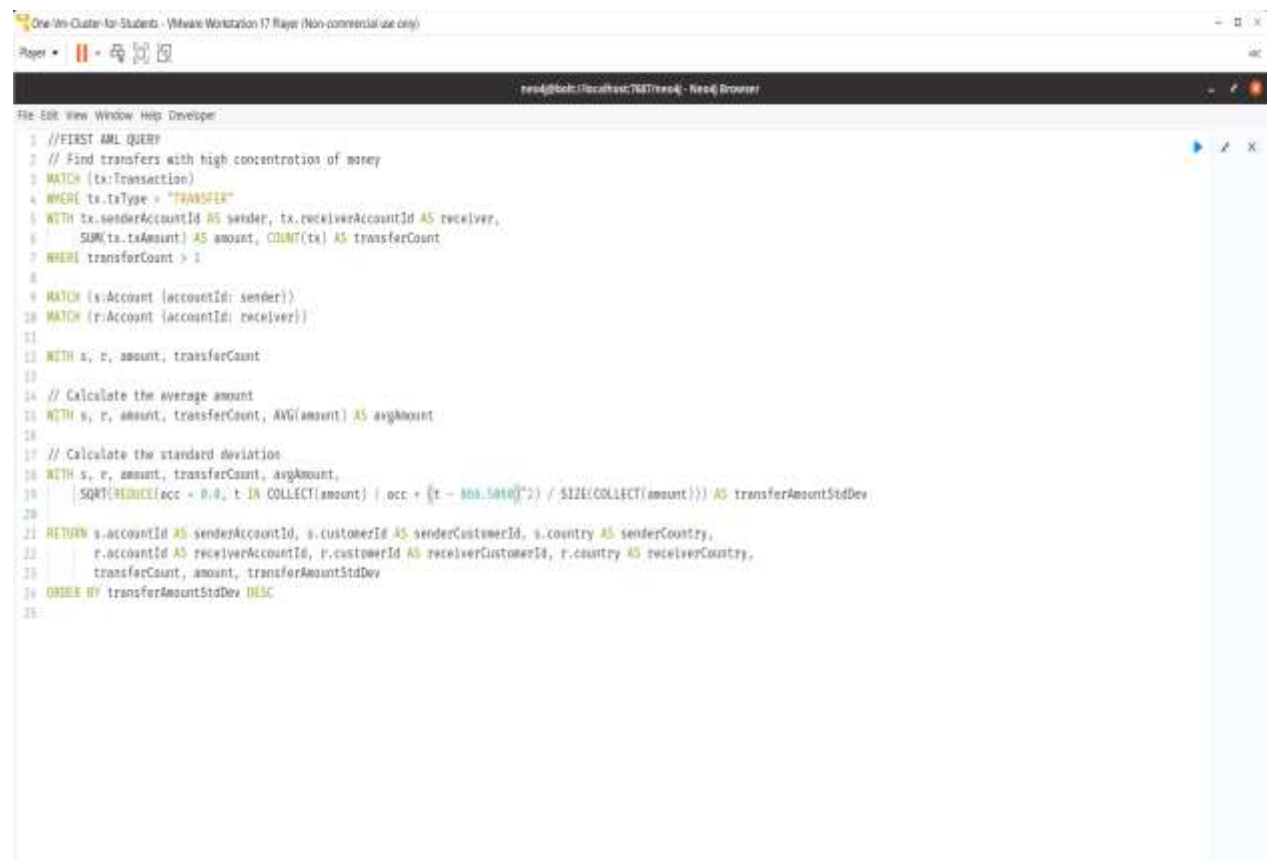


Figure 48 Schema visualization of the imported data

8.2.6 First AML Query



```

1 //FIRST AML QUERY
2 // Find transfers with high concentration of money
3 MATCH (tx:Transaction)
4 WHERE tx.txType = "TRANSFER"
5 WITH tx.senderAccountId AS sender, tx.receiverAccountId AS receiver,
6     SUM(tx.txAmount) AS amount, COUNT(tx) AS transferCount
7 WHERE transferCount > 1
8
9 MATCH (s:Account {accountId: sender})
10 MATCH (r:Account {accountId: receiver})
11
12 WITH s, r, amount, transferCount
13
14 // Calculate the average amount
15 WITH s, r, amount, transferCount, AVG(amount) AS avgAmount
16
17 // Calculate the standard deviation
18 WITH s, r, amount, transferCount, avgAmount,
19     SQRT(REDUCE(acc = 0.0, t IN COLLECT(amount) | acc + ((t - avgAmount)^2) / SIZE(COLLECT(amount))) AS transferAmountStdDev
20
21 RETURN s.accountId AS senderAccountId, s.customerId AS senderCustomerId, s.country AS senderCountry,
22     r.accountId AS receiverAccountId, r.customerId AS receiverCustomerId, r.country AS receiverCountry,
23     transferCount, amount, transferAmountStdDev
24 ORDER BY transferAmountStdDev DESC
25

```

Figure 49 First AML Query

The query matches the sender and receiver accounts and returns their account IDs, customer IDs, countries, transfer count, total transfer amount, and transfer amount standard deviation, ordered by the transfer amount standard deviation in descending order. We are using standard deviation as a mean to spot suspicious transactions.

Outliers are extreme values that differ significantly from other data points in a dataset. They can have a big impact on statistical analyses and skew the results of any hypothesis tests. When outliers are present in a data set, they significantly affect the standard deviation. Outliers tend to increase the standard deviation because they are far from the mean, making the sum of the squared deviations larger. Standard deviation is sensitive to outliers. A single outlier can raise the standard deviation and in turn, distort the picture of spread. For data with approximately the same mean, the greater the spread, the greater the standard deviation.

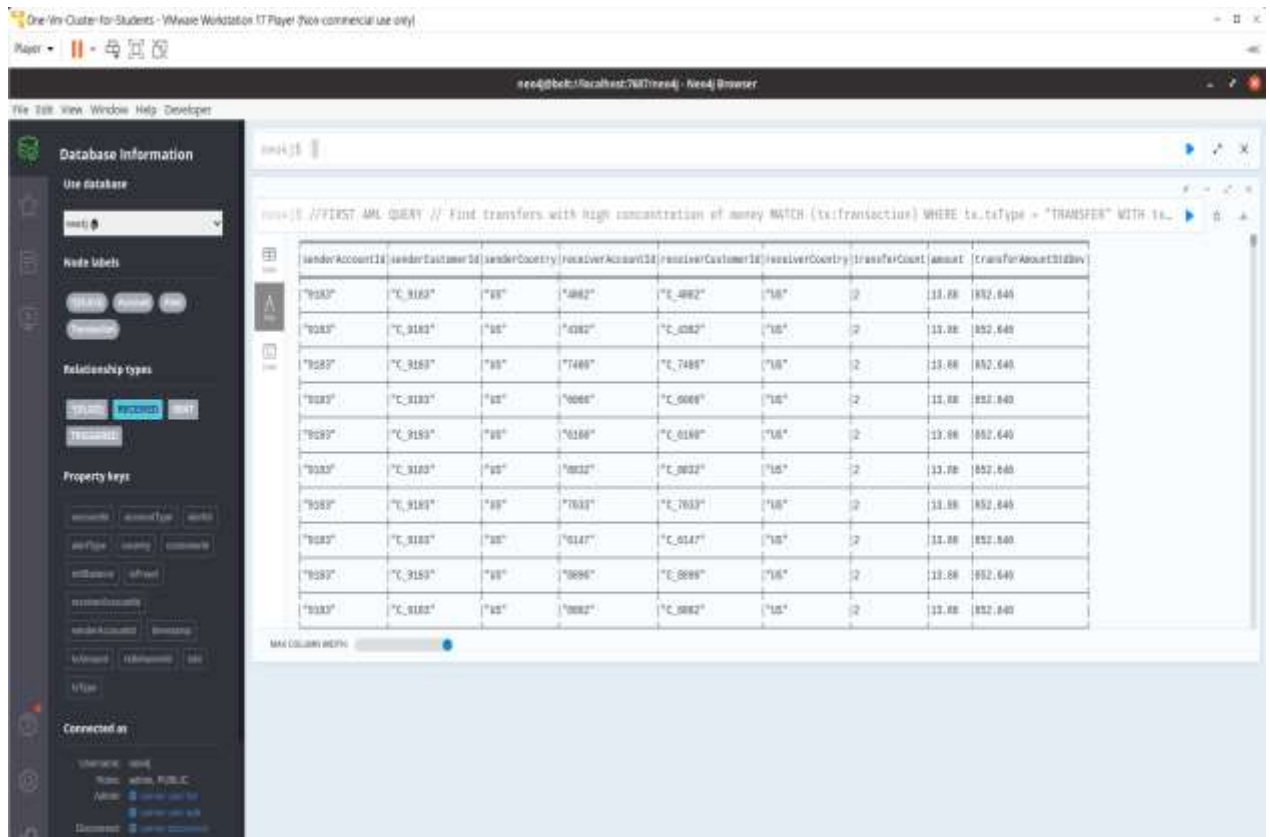


Figure 50 Results of the First AML Query

We are going to use the first pair with senderAccountId “9183” and receiverAccountId “4062” to examine their connection by using graph algorithms as they have the highest score in transferAmountStdDev with value 852.646

8.2.7 Second AML Query



Figure 51 Second AML Query

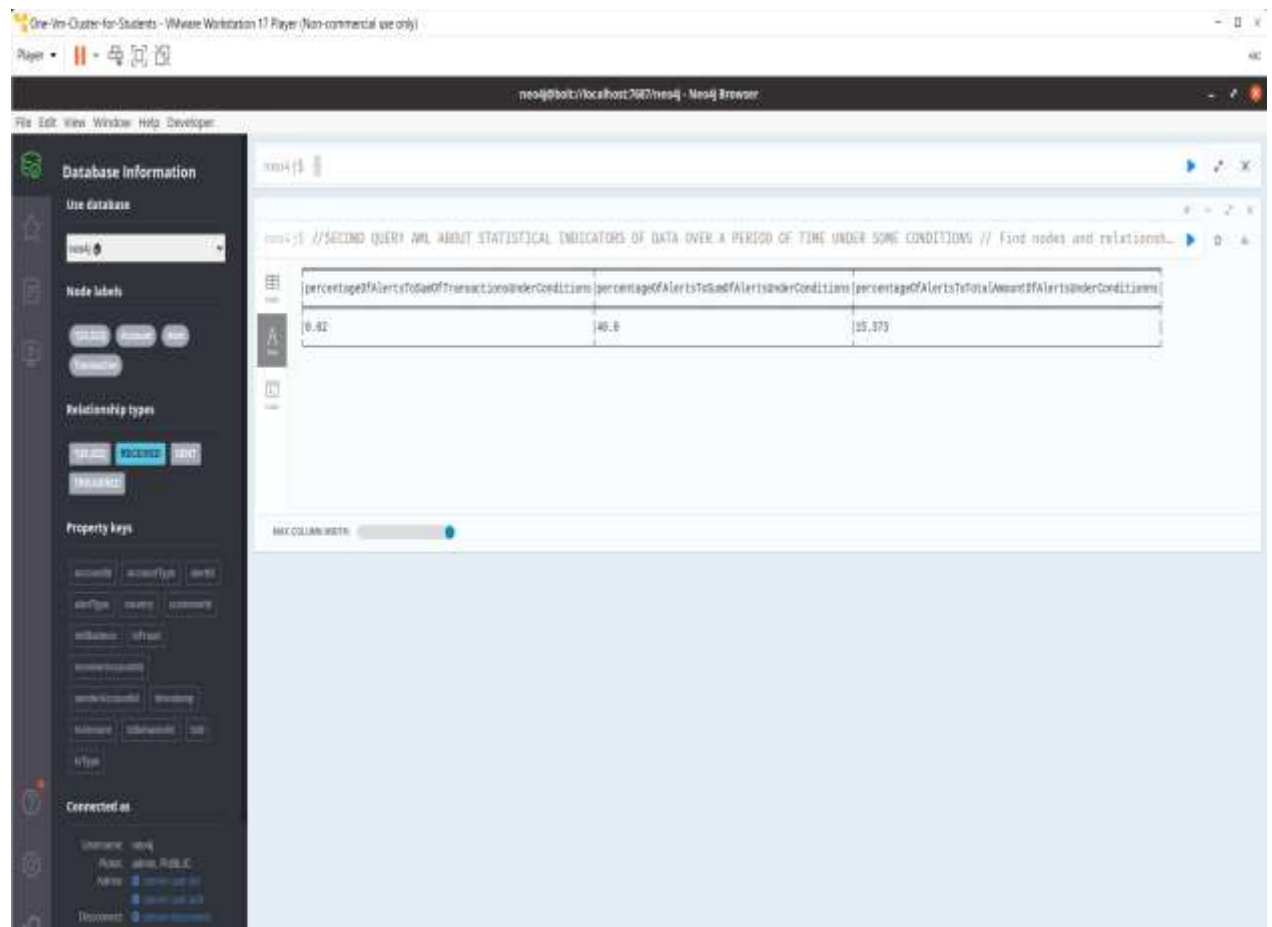


Figure 52 Results of Second AML Query

The first column shows that a very small percentage (0.02%) of all transactions under these conditions triggered an alert. The second column shows that 40% of all alerts under these conditions were triggered by transactions that meet these conditions. The third column shows that 15.373% of the total amount of alerts under these conditions were triggered by transactions that meet these conditions.

8.2.8 Third AML Query

```

1 //THIRD AML QUERY ABOUT STRUCTURING CASH
2
3 MATCH (a:Account)-[:SENT]->(t:Transaction)-[:RECEIVED]->(b:Account)
4 WHERE t.txType = 'TRANSFER'
5 WITH a, b, t
6 ORDER BY t.txAmount ASC
7 // Collect transaction amounts and timestamps into lists
8 WITH a, b, COLLECT(t.txAmount) AS amounts, COLLECT(t.timestamp) AS timestamps
9 // Define the window size and step
10 WITH a, b, amounts, timestamps, 5 AS windowSize, 1 AS step
11 // Unwind the indices based on the window size
12 UNWIND RANGE(0, SIZE(amounts) - windowSize) AS i
13 // Create rolling window of transactions
14 WITH a, b, amounts[i..i+windowSize-1] AS windowAmounts, timestamps[i..i+windowSize-1] AS windowTimestamps
15 WITH a, b, windowAmounts, windowTimestamps,
16   apoc.coll.sum(windowAmounts) AS windowSum,
17   apoc.coll.max(windowAmounts) AS windowMax,
18   apoc.coll.min(windowAmounts) AS windowMin,
19   apoc.coll.avg(windowAmounts) AS windowAvg,
20   apoc.coll.max(windowTimestamps) AS windowMaxTimestamp,
21   apoc.coll.min(windowTimestamps) AS windowMinTimestamp
22 // Filter windows based on conditions
23 WHERE windowSum > 100
24 AND windowMax > 150
25 AND windowMin < 400
26 //Return suspicious transactions with account and transaction details
27 RETURN a.accountId AS senderAccountId, b.accountId AS receiverAccountId,
28   windowSum AS totalAmountInWindow, windowMax AS maxAmountInWindow, windowMin AS minAmountInWindow, windowAvg AS avgAmountInWindow,
29   windowMaxTimestamp AS maxTimestampInWindow, windowMinTimestamp AS minTimestampInWindow
30

```

Figure 53 Third AML Query

	senderAccountId	receiverAccountId	totalAmountInWindow	maxAmountInWindow	minAmountInWindow	avgAmountInWindow	maxTimestampInWindow	minTimestampInWindow
1	"1023"	"1101"	300.04	300.23	300.23	300.21	10	0
2	"1023"	"1101"	301.17	301.19	301.19	301.19	10	0
3	"1020"	"1040"	322.2	322.3	322.3	322.3	14	3
4	"1040"	"1020"	324.86	324.86	324.86	324.86	14	4
5	"1040"	"10"	324.44	324.44	324.44	324.44	14	4
6	"1023"	"1001"	326.13	326.13	326.13	326.13	10	0
7	"1023"	"1002"	326.13	326.13	326.13	326.13	10	0
8	"1023"	"1003"	326.13	326.13	326.13	326.13	10	0
9	"1023"	"1101"	326.13	326.13	326.13	326.13	10	0
10	"1023"	"1004"	326.13	326.13	326.13	326.13	10	0
11	"1023"	"1005"	326.13	326.13	326.13	326.13	10	0
12	"1023"	"1006"	326.13	326.13	326.13	326.13	10	0

Figure 54 Results of Third AML Query

The query searches for transactions of type “TRANSFER” and then creates a rolling window of transactions based on a specified window size and step. Within each window, the query calculates various statistical indicators, such as the sum, maximum, minimum, and average of the transaction amounts, as well as the maximum and minimum timestamps. The query then filters the windows based on certain conditions to identify potentially suspicious activity. Finally, the query returns information about the suspicious transactions, including the sender and receiver account IDs, the total, maximum, minimum, and average amounts in the window, and the maximum and minimum timestamps in the window.

8.2.9 Projection of graph in the graph catalogue of gds.library

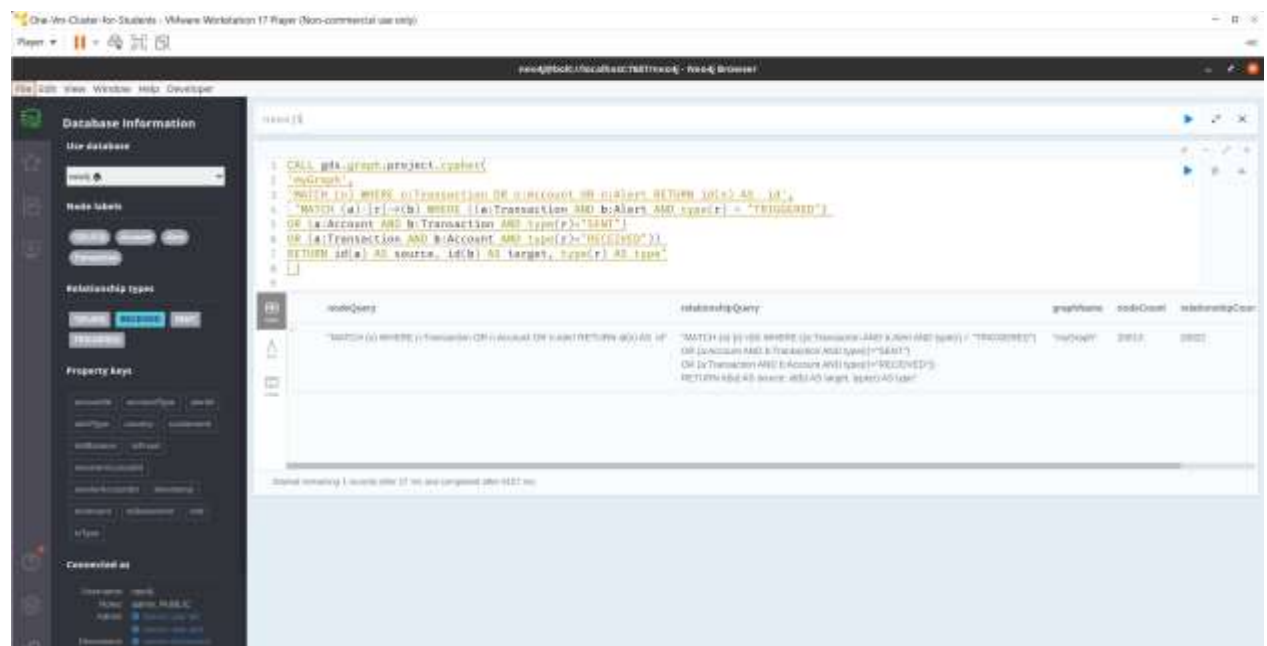


Figure 55 Graph projection

A named graph projection called myGraph is created. This procedure allows the creation of a graph projection, using Cypher queries to define the nodes and relationships that should be included in the graph. Once the named graph projection has been created, it can be used as input for various GDS algorithms or for further analysis using other GDS procedures. The GDS library provides a wide range of graph algorithms and analytics that can be used to gain insights from your data, such as community detection, centrality measures, pathfinding, and similarity algorithms.

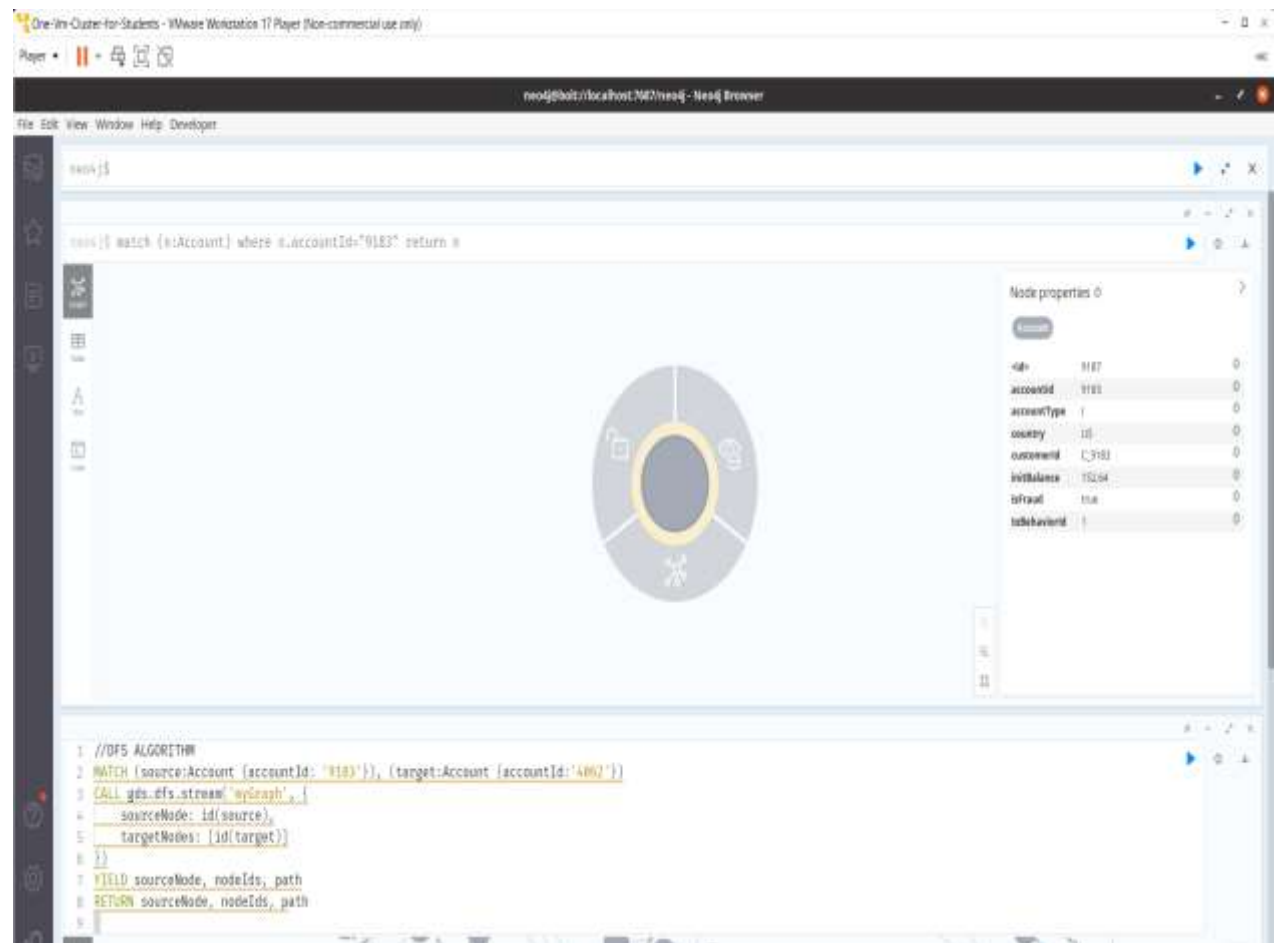


Figure 56 Results of DFS Algorithm

The algorithm found a path between the source node with accountId: '9183' and the target node with accountId: '4062'. The nodeIds field lists all the nodes that were visited during the traversal, and the path field contains a path object representing the traversal. The source node with accountId : '9183' has id: '9187'.

8.2.12 DEGREE CENTRALITY ALGORITHM

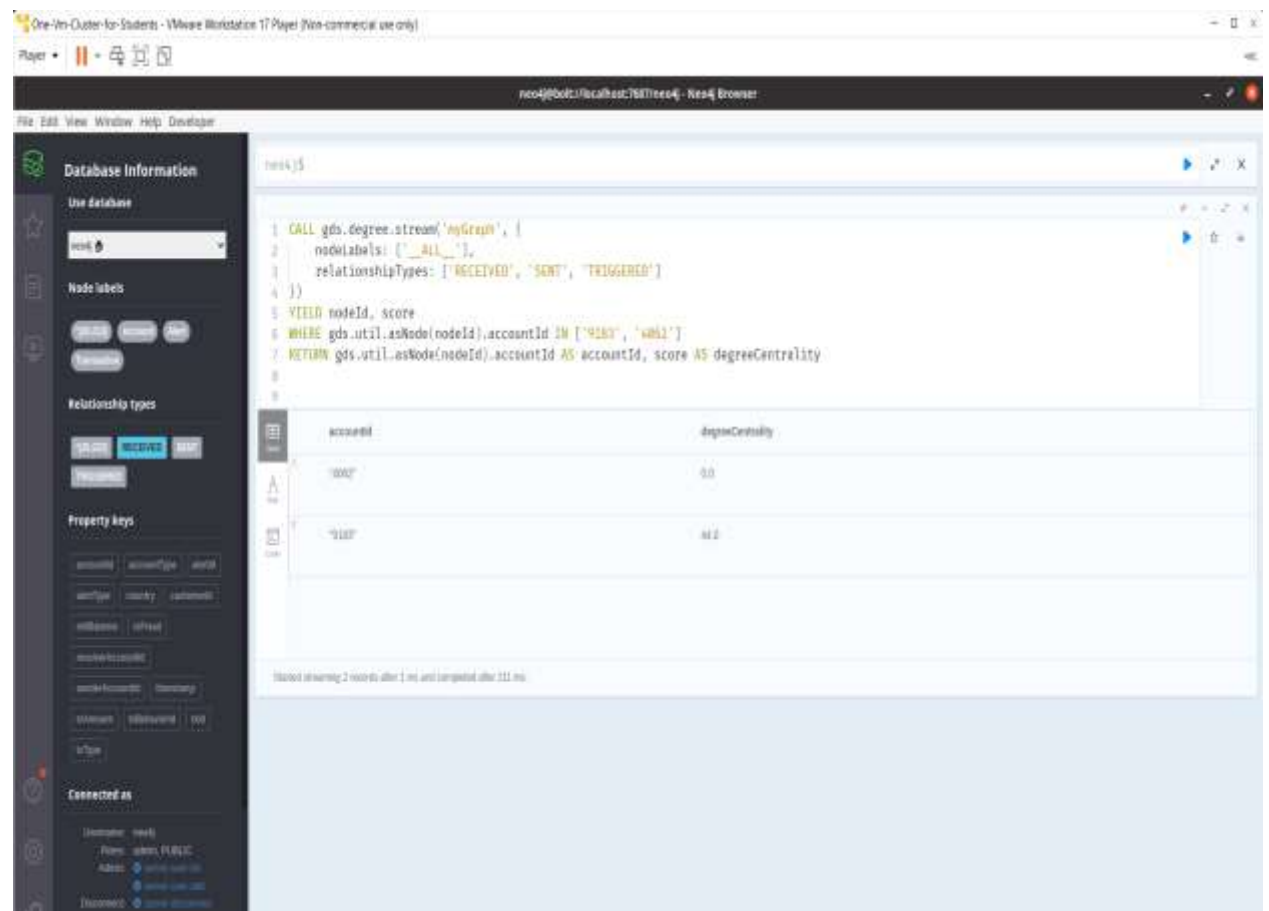


Figure 58 Results of Degree Centrality Algorithm

This algorithm shows the degree centrality of nodes in a graph. Degree centrality measures the number of incoming or outgoing (or both) relationships from a node, depending on the orientation of a relationship projection. The algorithm can be applied to either weighted or unweighted graphs, and can be used to find popular nodes within a graph.

8.2.13 MACHINE LEARNING MODELS

8.2.13.A LOGISTIC REGRESSION MODEL DESCRIPTION

Logistic regression is a statistical analysis method that constructs a statistical model to describe the relationship between a binary or dichotomous (yes/no type) outcome (dependent or response variable) and a set of independent predictor or explanatory variables. Regression modeling is a popular and useful approach in statistics that is used to explore and describe the relationship between an outcome or dependent/response variable and a set of independent predictors. Logistic regression is concerned with the

special situation in regression modeling, where the outcome is of a binary or dichotomous (yes/no) nature. Linear regression, where the outcome is continuous, cannot be used for binary outcomes because the probabilistic distribution of a binary or dichotomous variable is very different from that of a continuous variable (e.g., in the former case, the variance is usually a function of the mean, which is not the case for the latter). In addition, modeling a binary outcome entails modeling the probability of that event, which cannot be negative – a restriction that does not apply to linear regression.

The logistic regression model uses a logit link function to model the probability of a binary event. Suppose our binary outcome or “event” is Y , which can only be 0 (“No”) or 1 (“Yes”). Interpretation of the regression coefficients from a logistic regression model entails exponentiating these coefficients so that they can be expressed in terms of odds ratios. (44) (45)

8.2.13.B RANDOM FORESTS MODEL DESCRIPTION

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

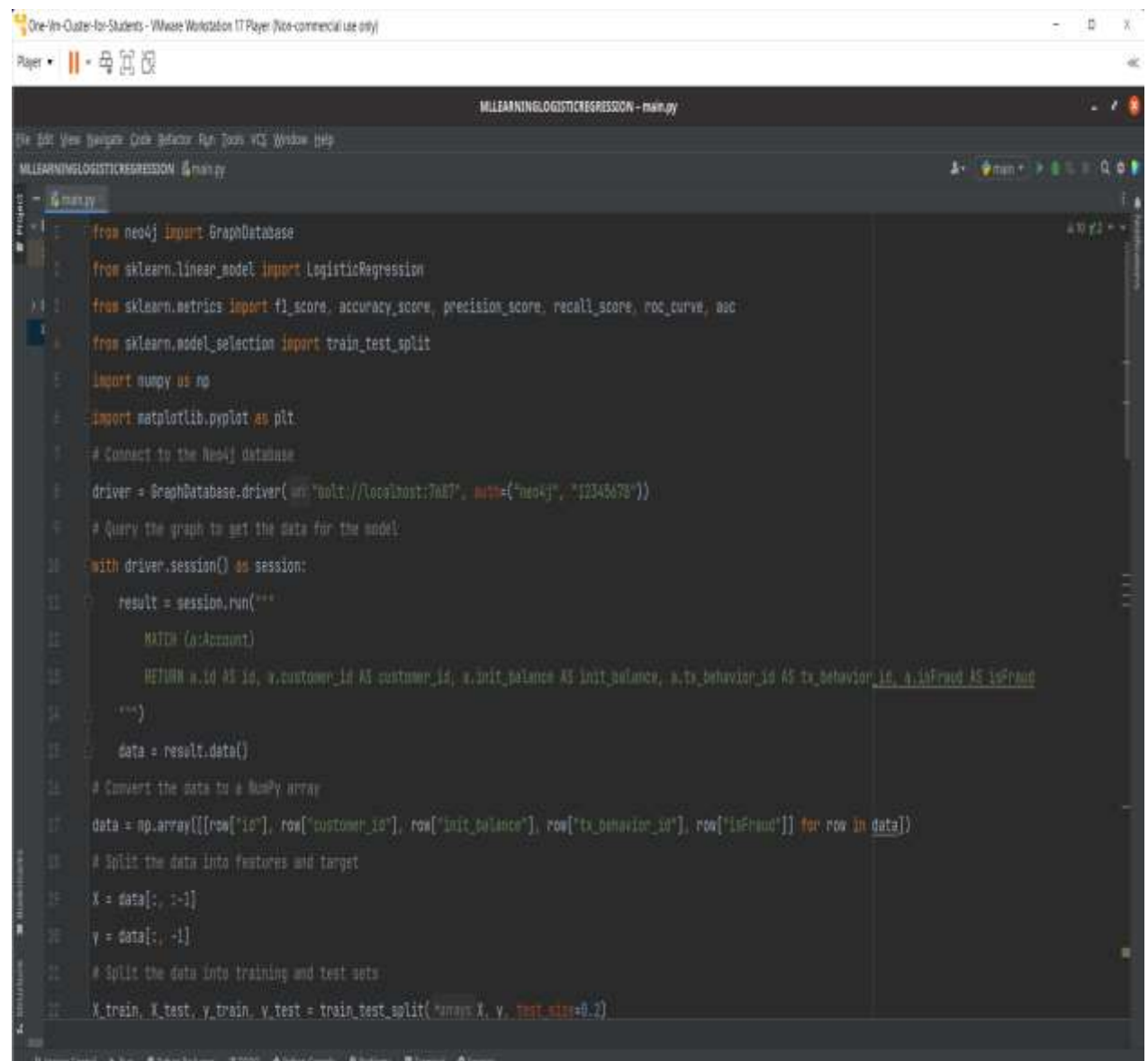
(46)(47)(48)

8.2.14 APPLICATION

8.2.14. A LOGISTIC REGRESSION MODEL

In order to perform the above mentioned model we are going to create a new graph database consisting of nodes with label account and convert the string values of features into numerical.

Additionally the GraphDatabase.driver method is used to create a driver object, which is then used to establish a connection to the database. The session.run method is used to execute a Cypher query on the database and retrieve the data. The data is then processed and used to train a machine learning model.



```
from neo4j import GraphDatabase
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_curve, auc
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Connect to the Neo4j database
driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "12345678"))

# Query the graph to get the data for the model
with driver.session() as session:
    result = session.run("""
        MATCH (a:Account)
        RETURN a.id AS id, a.customer_id AS customer_id, a.init_balance AS init_balance, a.tx_behavior_id AS tx_behavior_id, a.isFraud AS isFraud
    """)
    data = result.data()

# Convert the data to a NumPy array
data = np.array([[row["id"], row["customer_id"], row["init_balance"], row["tx_behavior_id"], row["isFraud"]] for row in data])

# Split the data into features and target
X = data[:, :-1]
y = data[:, -1]

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```



```

11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
12 # Create and train the logistic regression model
13 model = LogisticRegression()
14 model.fit(X_train, y_train)
15 # Make predictions on the test set
16 y_pred = model.predict(X_test)
17 y_pred_proba = model.predict_proba(X_test)[:, 1]
18 # Evaluate the performance of the model on the test set
19 f1 = f1_score(y_test, y_pred, average='weighted')
20 accuracy = accuracy_score(y_test, y_pred)
21 precision = precision_score(y_test, y_pred)
22 recall = recall_score(y_test, y_pred)
23 print('f1_score:', f1)
24 print('accuracy_score:', accuracy)
25 print('precision_score:', precision)
26 print('recall_score:', recall)
27 # Compute false positive rate, true positive rate, and thresholds
28 fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
29 # Compute AUC-ROC
30 roc_auc = auc(fpr, tpr)
31 # Create ROC curve plot
32 plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')

```

```

33 roc_auc = auc(fpr, tpr)
34 # Create ROC curve plot
35 plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
36 plt.plot([0, 1], [0, 1], linestyle='--')
37 plt.xlim([0.0, 1.0])
38 plt.ylim([0.0, 1.0])
39 plt.xlabel('False Positive Rate')
40 plt.ylabel('True Positive Rate')
41 plt.title('Receiver operating characteristic')
42 plt.legend(loc='lower right')
43 plt.show()
44 # Data for bar chart
45 metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
46 values = [accuracy, precision, recall, f1]
47 # Create bar chart
48 plt.bar(metrics, values)
49 # Add title and labels
50 plt.title('Model Performance')
51 plt.xlabel('Metrics')
52 plt.ylabel('Values')
53 # Show chart
54 plt.show()

```

Figure 59 Logistic Regression Model

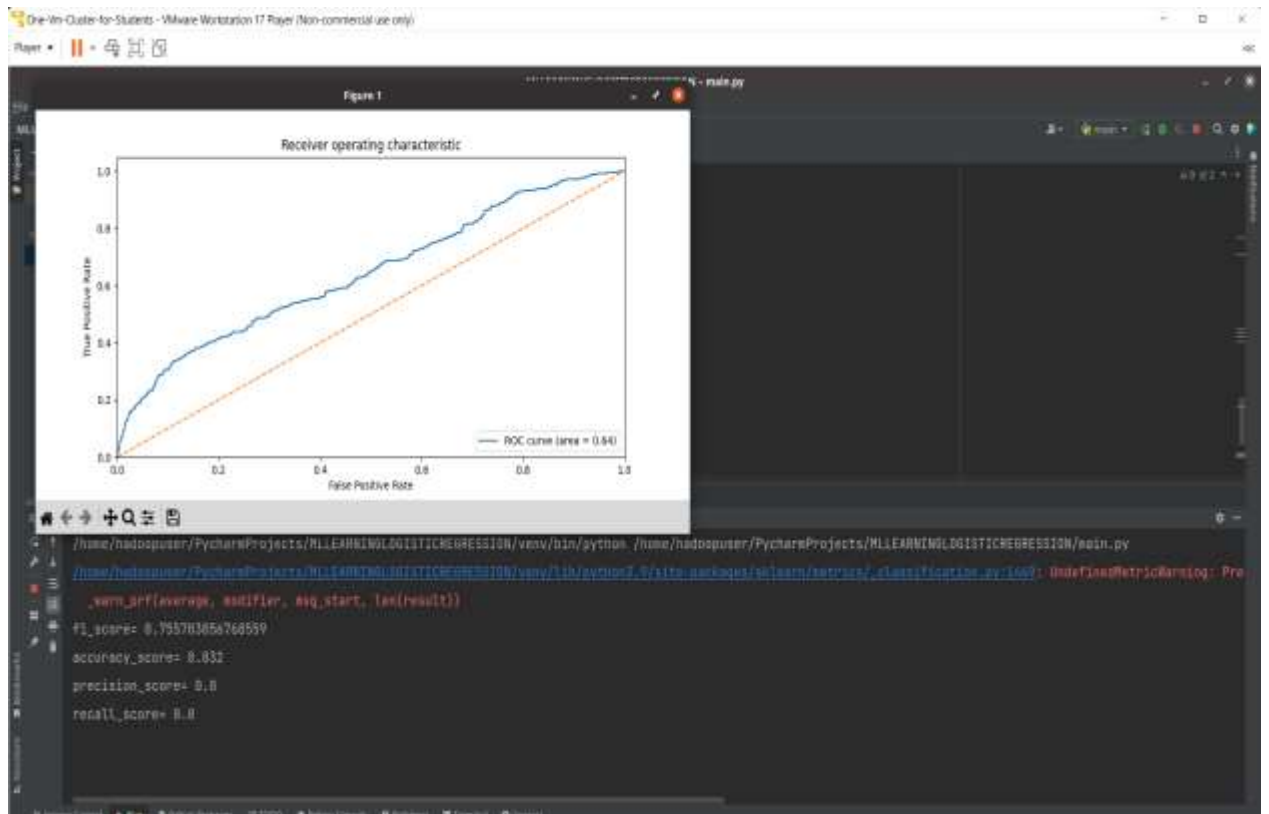


Figure 60 Results for *f1_score*, *accuracy_score*, *precision_score*, *recall_score* and the ROC curve

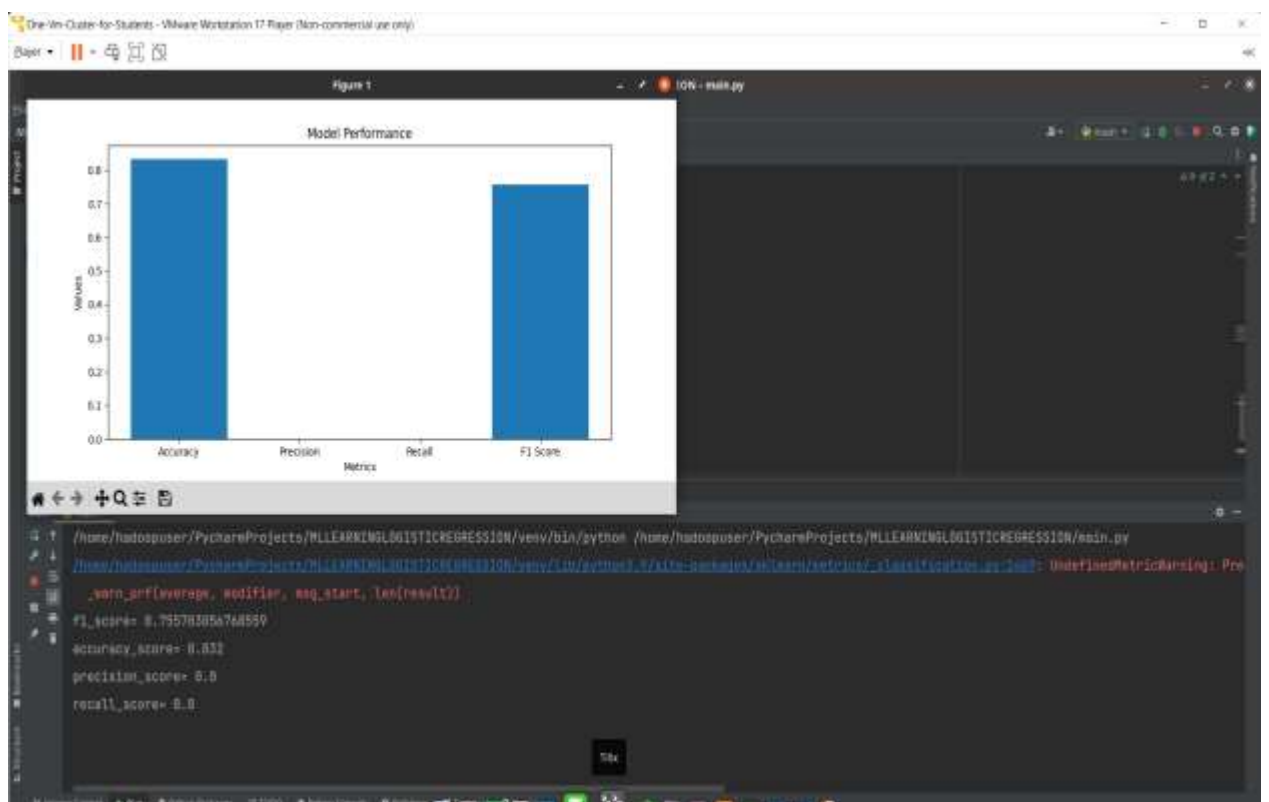


Figure 61 Bar Chart for metrics

- **Accuracy** measures the proportion of correct predictions made by the model. The accuracy score is 0.832, which means that the model correctly predicted whether an account is fraudulent or not in 83.2% of the cases.
- **Precision** measures the proportion of true positive predictions among all positive predictions made by the model. The precision score is 0.0, which means that none of the accounts predicted as fraudulent by the model were actually fraudulent.
- **Recall** measures the proportion of true positive predictions among all actual positive cases. The recall score is 0.0, which means that the model failed to identify any of the fraudulent accounts.
- **F1 score** is the harmonic mean of precision and recall and provides a balanced measure of the model's performance. The F1 score is 0.7557 but it could be improved.

The area under the ROC curve (AUC-ROC) is a measure of the model's ability to distinguish between the two classes. An AUC-ROC of 1.0 indicates a perfect classifier, while an AUC-ROC of 0.5 indicates a random classifier. Our logistic regression model achieved an ROC curve area of 0.64. This means that the model has some ability to distinguish between fraudulent and non-fraudulent accounts, but it is not perfect.

(49) (50) (51)

8.2.14.B RANDOM FORESTS MODEL

```

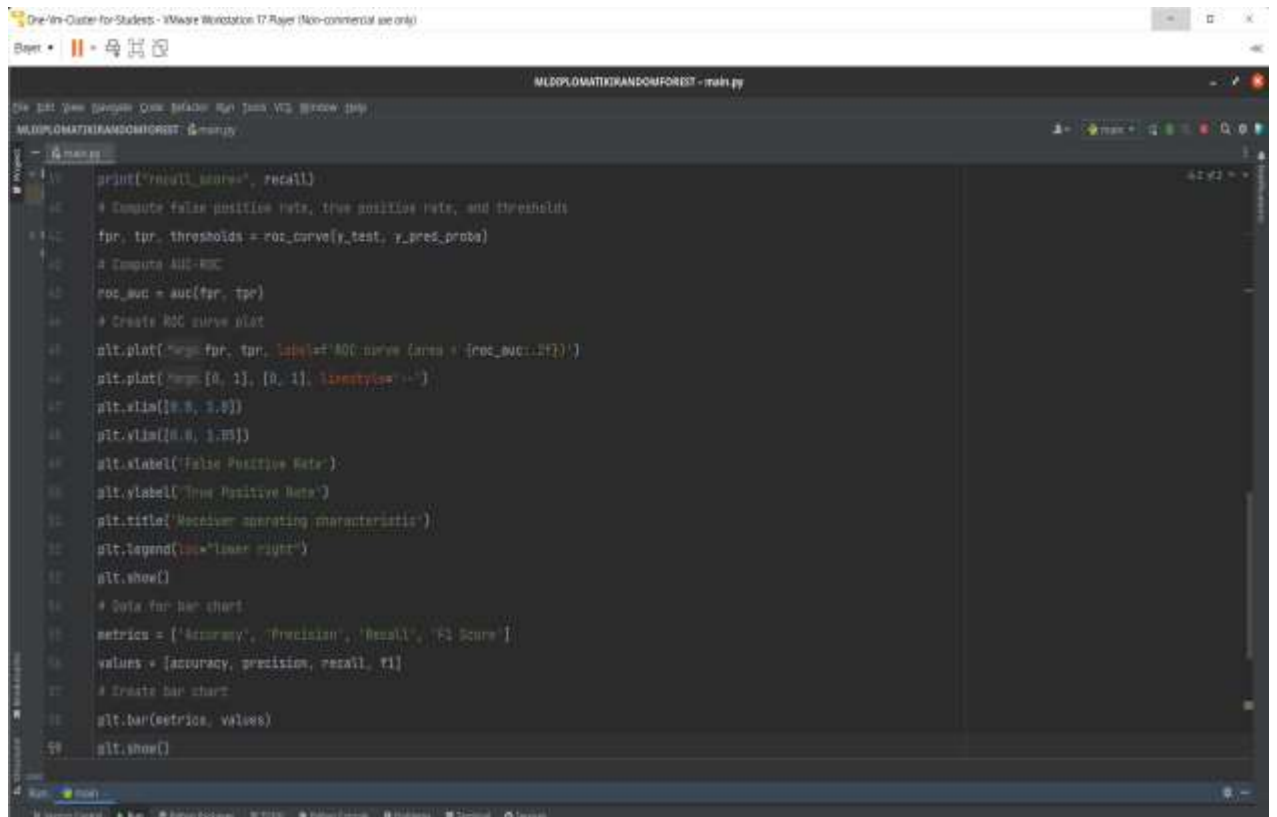
1 # RANDOM FOREST MODEL
2 from neo4j import GraphDatabase
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_curve, auc
5 from sklearn.model_selection import train_test_split
6 import numpy as np
7 import matplotlib.pyplot as plt
8 # Connect to the Neo4j database
9 driver = GraphDatabase.driver(uri="bolt://localhost:7687", auth=("neo4j", "12345678"))
10 # Query the graph to get the data for the model
11 with driver.session() as session:
12     result = session.run("""
13         MATCH (n:Account)
14         RETURN n.id AS id, n.customer_id AS customer_id, n.init_balance AS init_balance, n.tx_behavior_id AS tx_behavior_id, n.is_fraud AS is_fraud
15     """)
16     data = result.data()
17 # Convert the data to a NumPy array
18 data = np.array(
19     [[row["id"], row["customer_id"], row["init_balance"], row["tx_behavior_id"], row["is_fraud"]] for row in data])
20 # Split the data into features and target
21 X = data[:, :-1]

```

```

22 [[row["id"], row["customer_id"], row["init_balance"], row["tx_behavior_id"], row["is_fraud"]] for row in data])
23 # Split the data into features and target
24 X = data[:, :-1]
25 y = data[:, -1]
26 # Split the data into training and test sets
27 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
28 # Create and train the random forest model
29 model = RandomForestClassifier()
30 model.fit(X_train, y_train)
31 # Make predictions on the test set
32 y_pred = model.predict(X_test)
33 y_pred_proba = model.predict_proba(X_test)[:, 1]
34 # Evaluate the performance of the model on the test set
35 f1 = f1_score(y_test, y_pred, average="weighted")
36 accuracy = accuracy_score(y_test, y_pred)
37 precision = precision_score(y_test, y_pred)
38 recall = recall_score(y_test, y_pred)
39 print("f1 score=", f1)
40 print("accuracy score=", accuracy)
41 print("precision score=", precision)
42 print("recall score=", recall)

```



```
print("recall score", recall)
# Compute false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Compute AUC-ROC
roc_auc = auc(fpr, tpr)
# Create ROC curve plot
plt.plot(fpr, tpr, label='ROC curve (area = %f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
# Data for bar chart
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
values = [accuracy, precision, recall, f1]
# Create bar chart
plt.bar(metrics, values)
plt.show()
```

Figure 62 Random Forest Model

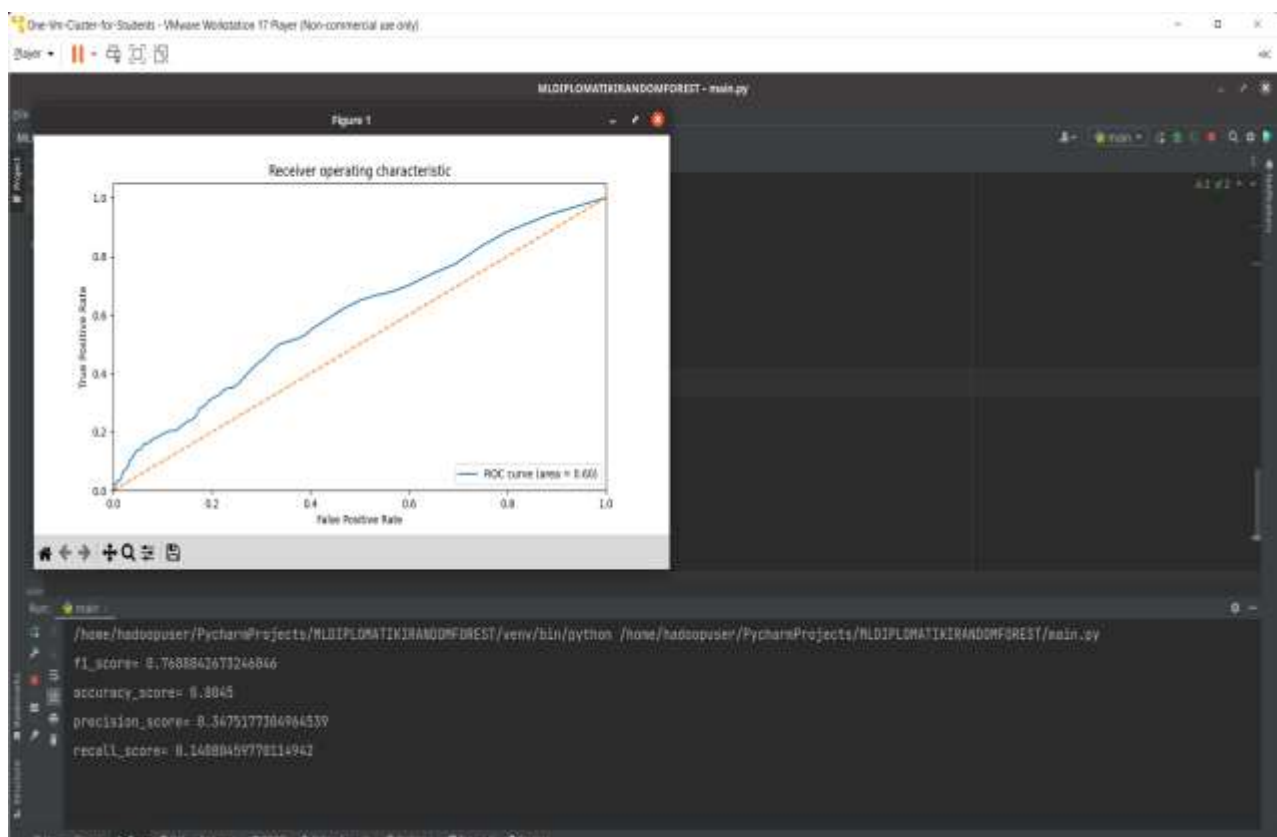


Figure 63 Results for f1_score, accuracy_score, precision_score, recall_score and ROC curve area

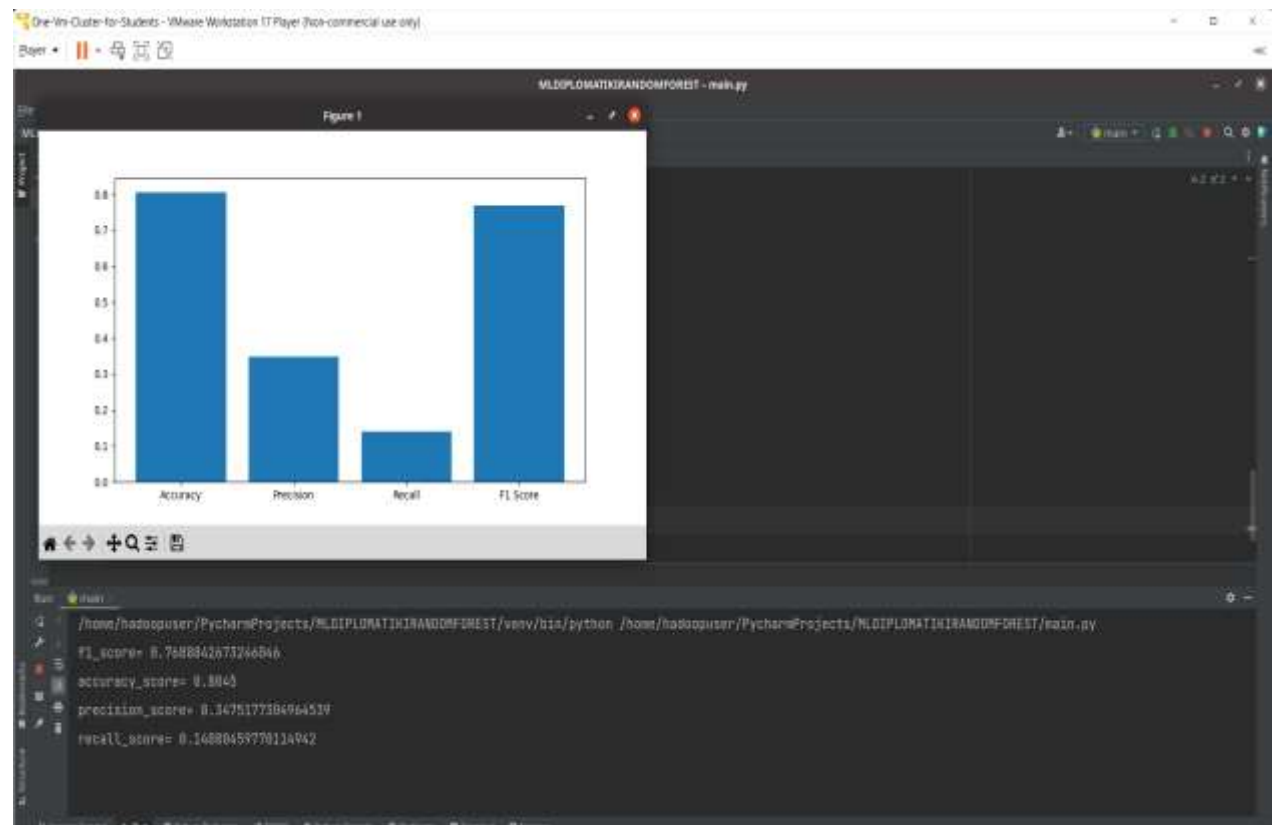


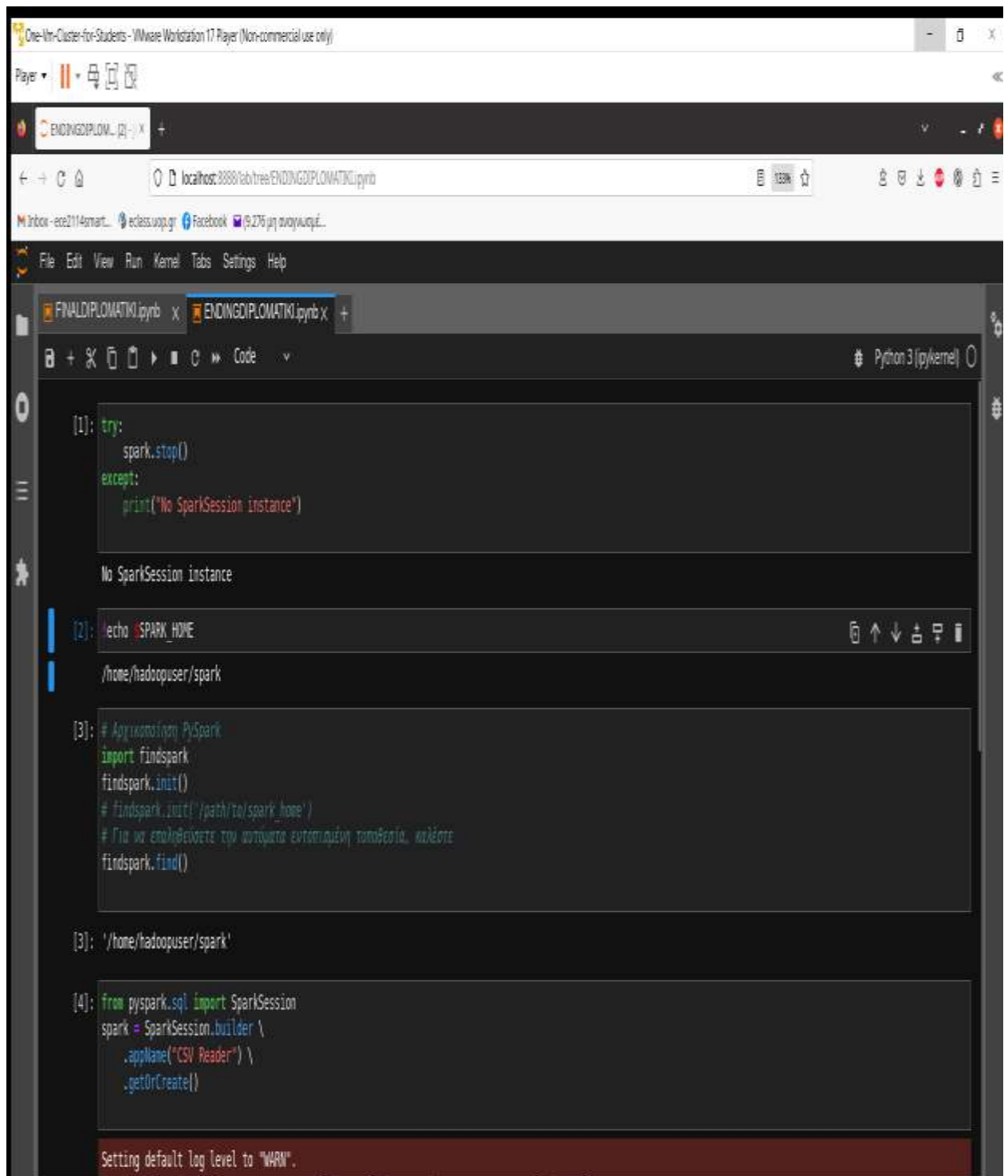
Figure 64 Bar Chart for metrics

The model achieved an accuracy of 0.8045 and an F1 score of 0.7689, which are decent results. The precision score is 0.3475, which means that about 34.75% of the accounts predicted as fraudulent by the model were actually fraudulent. The recall score is 0.1408, which means that the model was able to identify about 14.08% of the fraudulent accounts.

The random forest model achieved an ROC curve area of 0.60. The model shows some ability to distinguish between fraudulent and non-fraudulent accounts, but it is not perfect. An AUC-ROC of 0.60 is considered to be fair, but there is still room for improvement.

8.3 Apache Spark

8.3.1 PySpark and SparkSession



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[1]: try:
      spark.stop()
    except:
      print("No SparkSession instance")
```

Output: No SparkSession instance

```
[2]: !echo $SPARK_HOME
```

Output: /home/hadoopuser/spark

```
[3]: # Αρχικοποίηση PySpark
import findspark
findspark.init()
# findspark.init('/path/to/spark_home')
# Για να εστιάσετε την αντομια εντοπισμένη τοποθεσία, καλέστε
findspark.find()
```

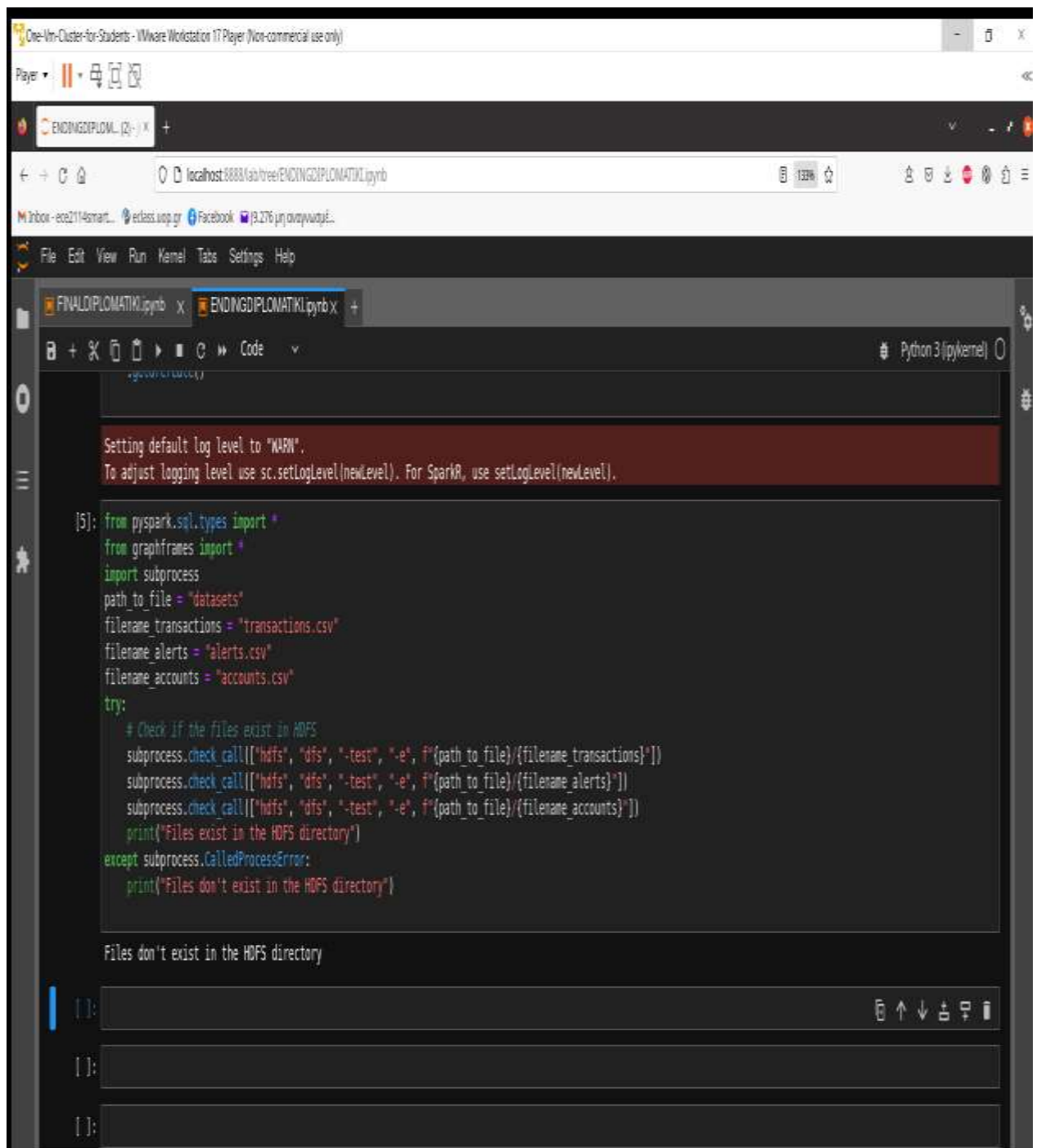
Output: '/home/hadoopuser/spark'

```
[4]: from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("CSV Reader") \
    .getOrCreate()
```

Output: Setting default log level to "WARN".

Figure 65 Starting SparkSession

Initialize PySpark and create a SparkSession



```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

[5]: from pyspark.sql.types import *
from graphframes import *
import subprocess
path_to_file = "datasets"
filename_transactions = "transactions.csv"
filename_alerts = "alerts.csv"
filename_accounts = "accounts.csv"
try:
    # Check if the files exist in HDFS
    subprocess.check_call(["hdfs", "dfs", "-test", "-e", f"{path_to_file}/{filename_transactions}"])
    subprocess.check_call(["hdfs", "dfs", "-test", "-e", f"{path_to_file}/{filename_alerts}"])
    subprocess.check_call(["hdfs", "dfs", "-test", "-e", f"{path_to_file}/{filename_accounts}"])
    print("Files exist in the HDFS directory")
except subprocess.CalledProcessError:
    print("Files don't exist in the HDFS directory")

Files don't exist in the HDFS directory
```

Figure 66 Import modules, define variables and checking in the HDFS directory

We import some modules and define some variables for your file paths and names. Then, we use the sub process module to execute some HDFS commands to check if the files exist in the HDFS directory.

8.3.2 Forming a graph

```

(6): # Create Function for Graph
from pyspark.sql import SparkSession
from graphframes import GraphFrame
from pyspark.sql.functions import lit, when
# Define the paths to the CSV files using the file:// URI scheme
transactions_path = "file:///home/hadoopuser/Desktop/transactions1.csv"
alerts_path = "file:///home/hadoopuser/Desktop/alerts1.csv"
accounts_path = "file:///home/hadoopuser/Desktop/accounts1.csv"
def create_graph(transactions_path, alerts_path, accounts_path):
    # Create a SparkSession
    spark = SparkSession.builder.appName("CSV Reader").getOrCreate()
    # Read the CSV files using the SparkSession
    transactions = spark.read.csv(transactions_path, header=True, inferSchema=True)
    alerts = spark.read.csv(alerts_path, header=True, inferSchema=True)
    accounts = spark.read.csv(accounts_path, header=True, inferSchema=True)
    # Create new IS_FRAUD columns with boolean values
    transactions = transactions.withColumn("IS_FRAUD", when(transactions["IS_FRAUD"] == "true", True).otherwise(False))
    accounts = accounts.withColumn("IS_FRAUD", when(accounts["IS_FRAUD"] == "true", True).otherwise(False))
    # Create vertices DataFrame
    accounts_vertices = accounts.selectExpr("ACCOUNT_ID as id", "CUSTOMER_ID", "INIT_BALANCE", "COUNTRY", "ACCOUNT_TYPE", "IS_FRAUD", "TX_BEHAVIOR_ID")
    accounts_vertices = accounts_vertices.withColumn("ALERT_ID", lit(None))
    transactions_vertices = transactions.selectExpr("TX_ID as id", "SENDER_ACCOUNT_ID", "RECEIVER_ACCOUNT_ID", "TX_TYPE", "TX_AMOUNT", "TIMESTAMP", "IS_FRAUD as
transactions_vertices = transactions_vertices.select([c if c in transactions_vertices.columns else lit(None).alias(c) for c in accounts_vertices.columns])
    alerts_vertices = alerts.selectExpr("ALERT_ID as id")
    alerts_vertices = alerts_vertices.select([c if c in alerts_vertices.columns else lit(None).alias(c) for c in accounts_vertices.columns])
    vertices = accounts_vertices.union(transactions_vertices).union(alerts_vertices)
    # Create edges DataFrame
    sent_edges = transactions.selectExpr("SENDER_ACCOUNT_ID as src", "TX_ID as dst")
    received_edges = transactions.selectExpr("TX_ID as src", "RECEIVER_ACCOUNT_ID as dst")
    triggered_edges = alerts.selectExpr("TX_ID as src", "ALERT_ID as dst")

```



```

transactions_vertices = transactions.selectExpr("TX_ID as id", "SENDER_ACCOUNT_ID", "RECEIVER_ACCOUNT_ID", "TX_TYPE", "TX_AMOUNT", "TIMESTAMP", "IS_FRAUD as is_fraud")
transactions_vertices = transactions_vertices.select([c if c in transactions_vertices.columns else lit(None).alias(c) for c in accounts_vertices.columns])
alerts_vertices = alerts.selectExpr("ALERT_ID as id")
alerts_vertices = alerts_vertices.select([c if c in alerts_vertices.columns else lit(None).alias(c) for c in accounts_vertices.columns])
vertices = accounts_vertices.union(transactions_vertices).union(alerts_vertices)
# Create edges DataFrame
sent_edges = transactions.selectExpr("SENDER_ACCOUNT_ID as src", "TX_ID as dst")
received_edges = transactions.selectExpr("TX_ID as src", "RECEIVER_ACCOUNT_ID as dst")
triggered_edges = alerts.selectExpr("TX_ID as src", "ALERT_ID as dst")
edges = sent_edges.union(received_edges).union(triggered_edges)
# Create a GraphFrame
g = GraphFrame(vertices, edges)
return g

```

```

[7]: #call create_graph function
g=create_graph(transactions_path, alerts_path, accounts_path)

[]:

[]:

[]:

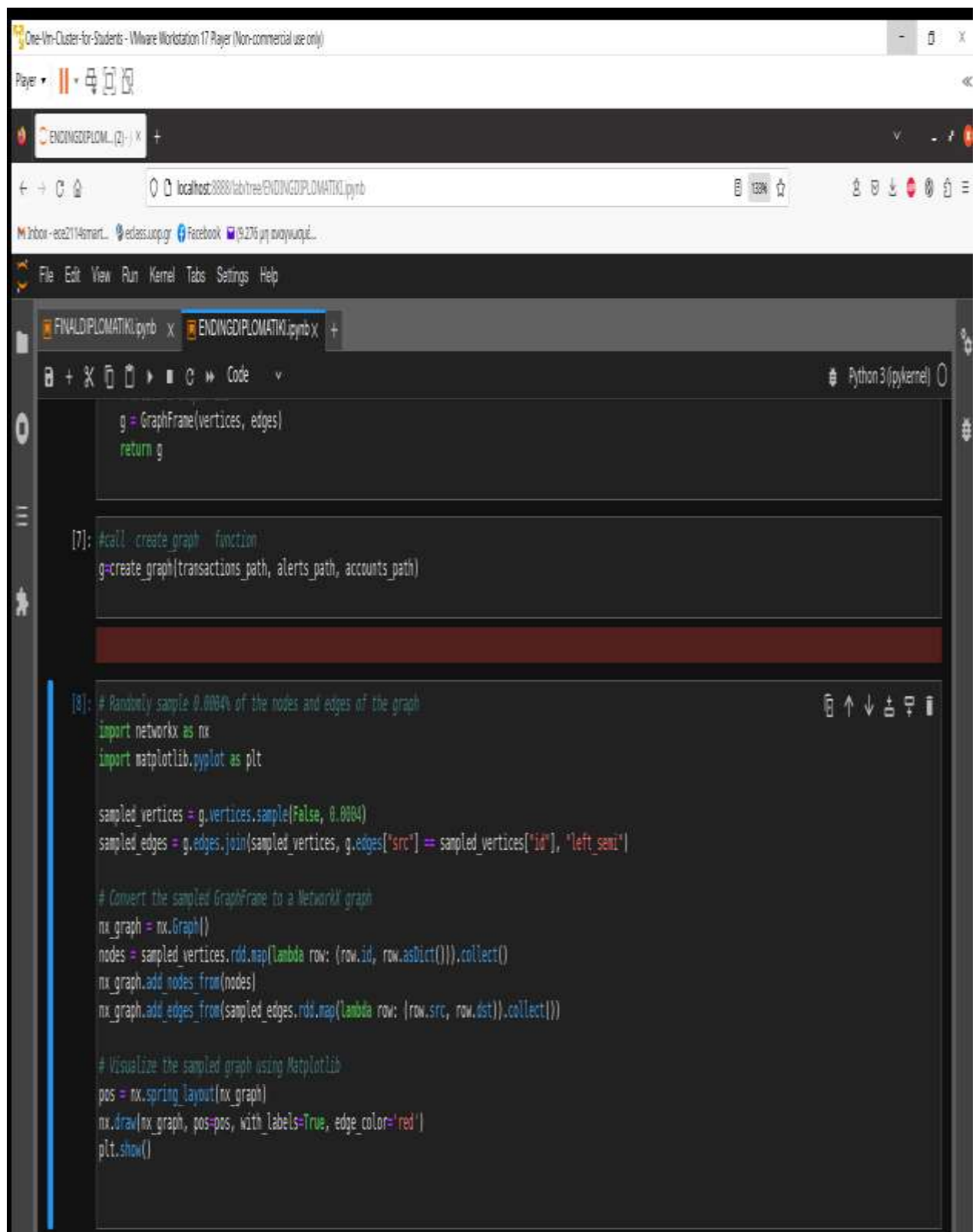
[]:

```

Figure 67 Constructing a graph

We form a Python function that creates a graph using the GraphFrame library from three CSV files: transactions_path, alerts_path, and accounts_path. The function reads the CSV files using the SparkSession object and creates a GraphFrame object from the data. The graph has vertices representing accounts, transactions, and alerts, and edges representing the relationships between them. Then we call this function and the GraphFrame object is being assigned to the variable g.

8.3.3 Visualization of the graph



The screenshot shows a Jupyter Notebook running in a browser. The notebook contains the following code and output:

```
g = GraphFrame(vertices, edges)
return g
```

[7]: #call create_graph function
g=create_graph(transactions_path, alerts_path, accounts_path)

[8]: # Randomly sample 0.0004% of the nodes and edges of the graph
import networkx as nx
import matplotlib.pyplot as plt

```
sampled_vertices = g.vertices.sample(False, 0.0004)
sampled_edges = g.edges.join(sampled_vertices, g.edges["src"] == sampled_vertices["id"], "left_semi")
```

```
# Convert the sampled GraphFrame to a NetworkX graph
nx_graph = nx.Graph()
nodes = sampled_vertices.rdd.map(lambda row: (row.id, row.asDict())).collect()
nx_graph.add_nodes_from(nodes)
nx_graph.add_edges_from(sampled_edges.rdd.map(lambda row: (row.src, row.dst)).collect())
```

```
# Visualize the sampled graph using Matplotlib
pos = nx.spring_layout(nx_graph)
nx.draw(nx_graph, pos=pos, with_labels=True, edge_color='red')
plt.show()
```

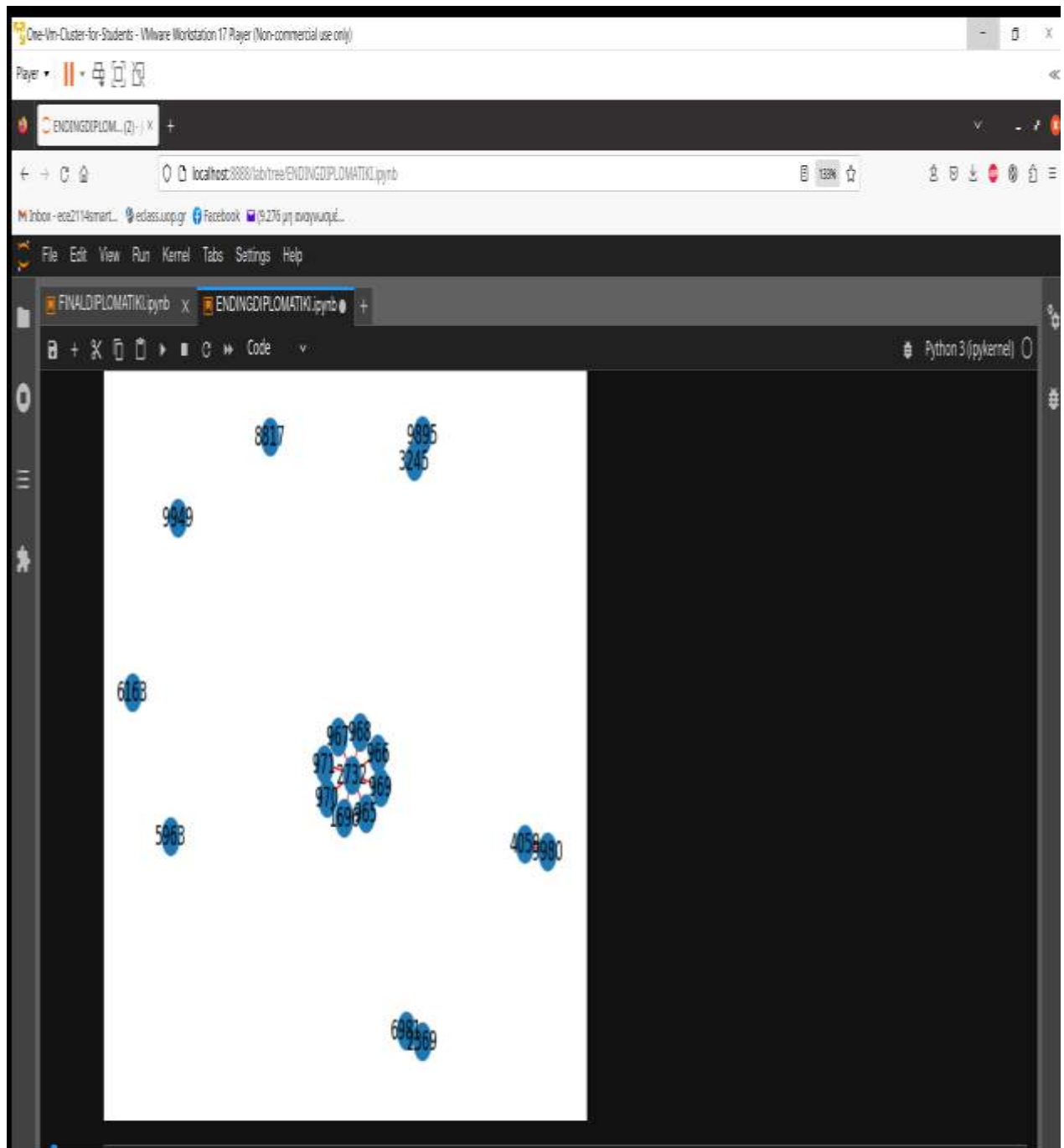


Figure 68 Visualization of 0,0004 of nodes and edges of the graph

We randomly sample 0.0004% of its vertices and edges to create a smaller graph. The sampled graph is then converted to a NetworkX graph object and visualized using the Matplotlib library. The visualization shows the nodes and edges of the sampled graph, with the nodes labeled by their IDs and the edges colored in red.

There are four paths from vertex 9183 to vertex 4062 following the specified edge sequence. Each row in the results represents one path, with columns a, b, and c representing the vertices along the path, and columns e and e2 representing the edges between them.

8.3.5 BFS ALGORITHM

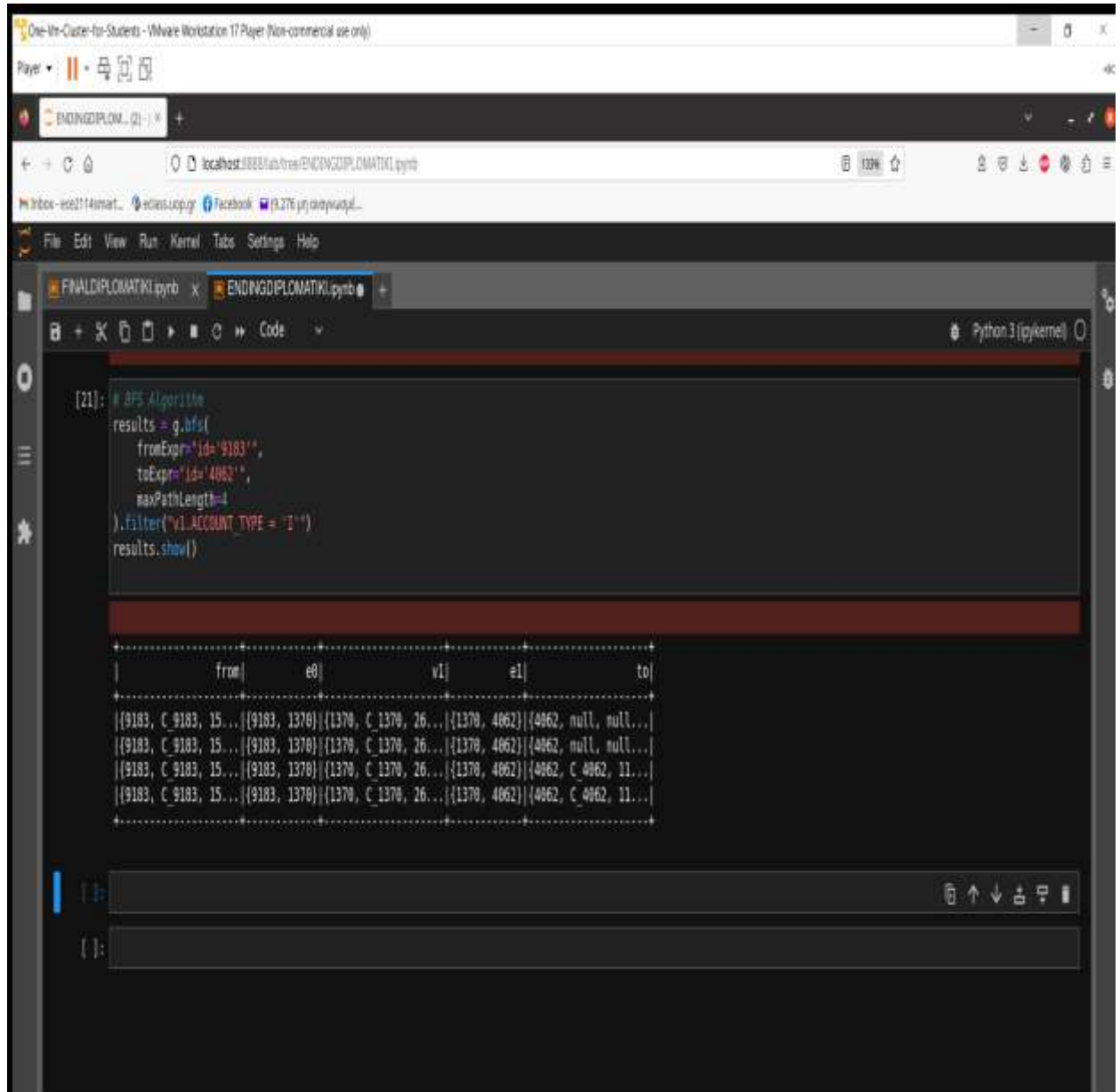
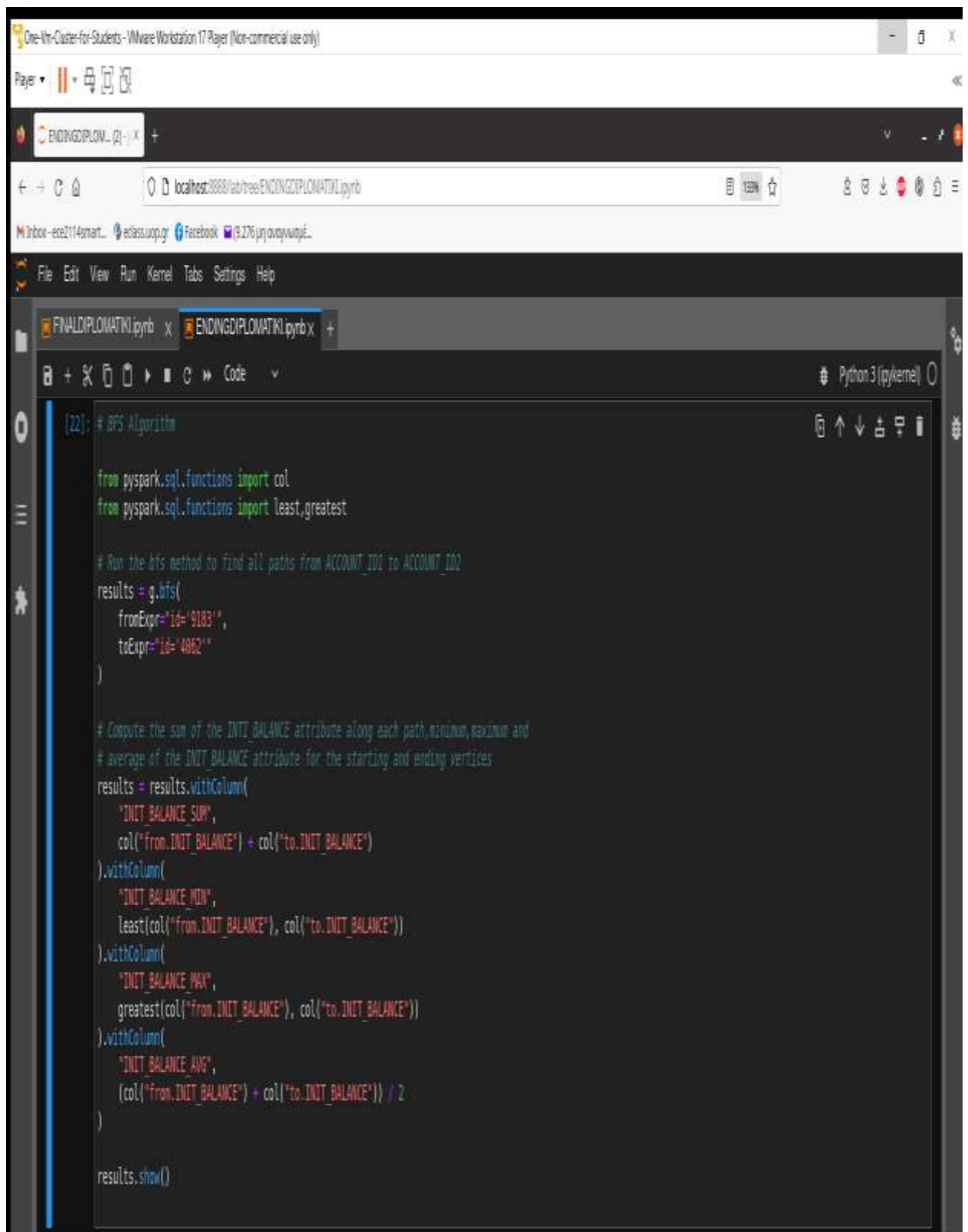


Figure 71 BFS Algorithm and results

As we can observe the results of BFS algorithm is the same with the results of DFS Algorithm.



```
[22]: # BFS Algorithm

from pyspark.sql.functions import col
from pyspark.sql.functions import least, greatest

# Run the bfs method to find all paths from ACCOUNT_ID1 to ACCOUNT_ID2
results = g.bfs(
    fromExpr="id='9183'",
    toExpr="id='4862'"
)

# Compute the sum of the INIT_BALANCE attribute along each path, minimum, maximum and
# average of the INIT_BALANCE attribute for the starting and ending vertices
results = results.withColumn(
    "INIT_BALANCE_SUM",
    col("from.INIT_BALANCE") + col("to.INIT_BALANCE")
).withColumn(
    "INIT_BALANCE_MIN",
    least(col("from.INIT_BALANCE"), col("to.INIT_BALANCE"))
).withColumn(
    "INIT_BALANCE_MAX",
    greatest(col("from.INIT_BALANCE"), col("to.INIT_BALANCE"))
).withColumn(
    "INIT_BALANCE_AVG",
    (col("from.INIT_BALANCE") + col("to.INIT_BALANCE")) / 2
)

results.show()
```

Figure 72 BFS Algorithm with minimum, maximum and average

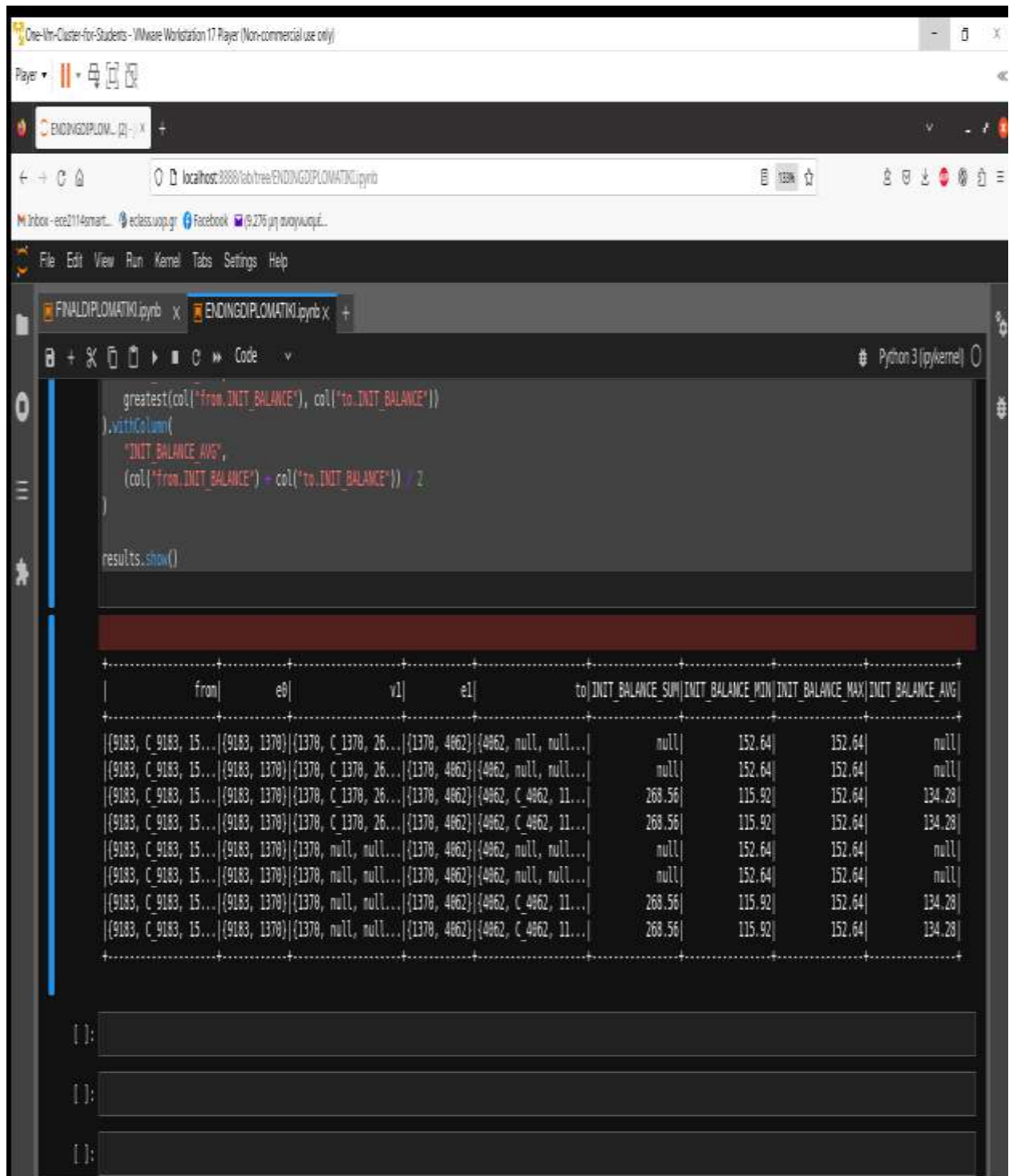


Figure 73 Results of BFS Algorithm

It appears that there are eight paths from vertex 9183 to vertex 4062. Each row in the results represents one path, with columns from, v1, and to representing the vertices along the path, and columns e0 and e1 representing the edges between them. Also we can see the sum, minimum, maximum, and average of the INIT_BALANCE attribute for the starting and ending vertices along each path.

8.3.6 DEGREE CENTRALITY ALGORITHM

The screenshot shows a Jupyter Notebook interface with a Python kernel. The code in the cell is as follows:

```
[23]: # DEGREE ALGORITHM
from graphframes import GraphFrame

# Calculate the degree of each vertex
degree = g.degrees

# Filter the results to only include vertices with ids 9183 and 4062
degree = degree.filter(degree.id.isin(['9183', '4062']))

# Show the results
degree.show()
```

The output of the code is a table with two columns: 'id' and 'degree'. The results are as follows:

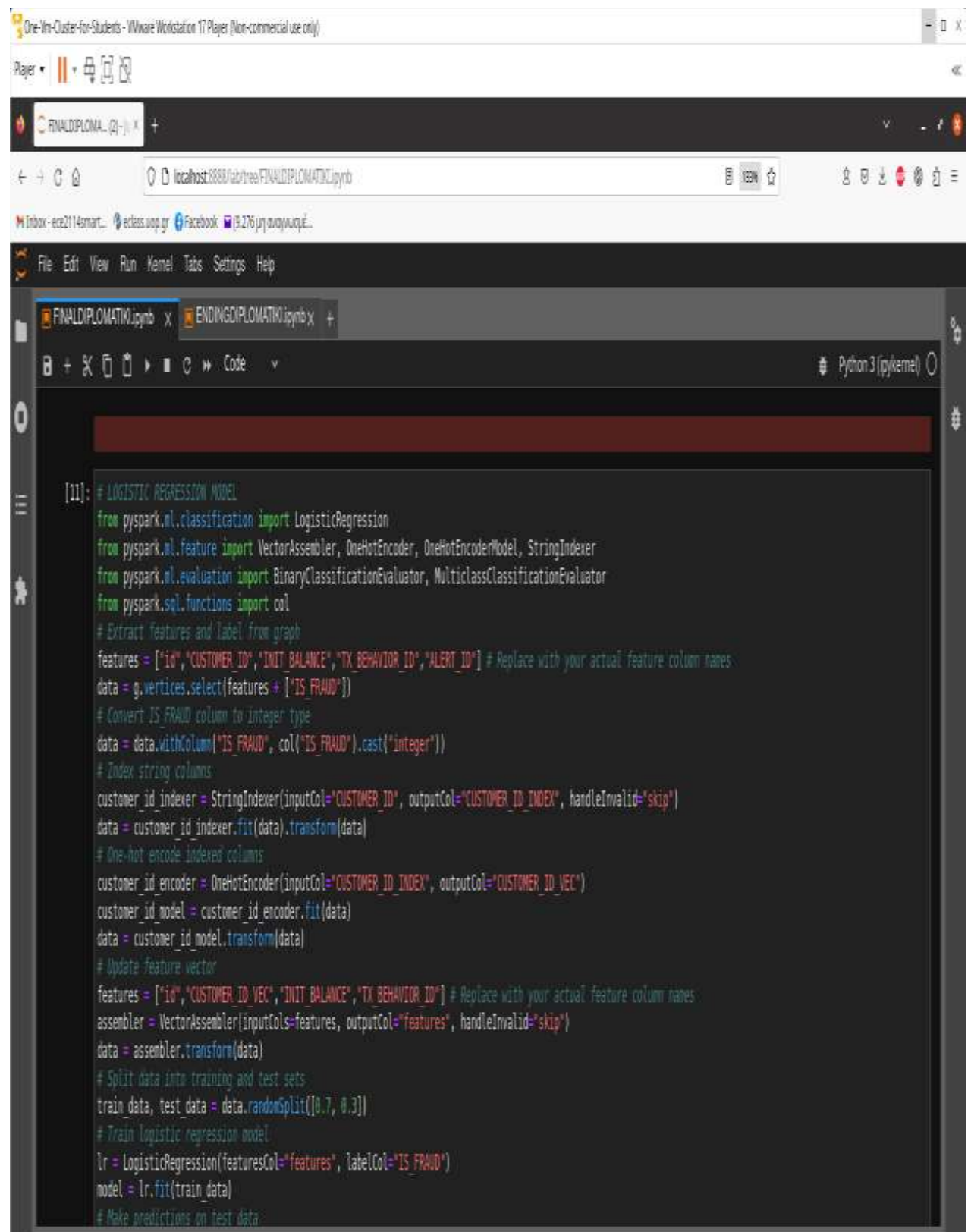
id	degree
4062	3
9183	23

Figure 74 DEGREE CENTRALITY ALGORITHM and results

Here we calculate the degree of each vertex in a GraphFrame object g using the degrees property. The degree of a vertex is the number of edges connected to it. The results are then filtered to only include vertices with IDs '9183' and '4062'.

8.4 MACHINE LEARNING

8.4.1 LOGISTIC REGRESSION MODEL



```
[11]: # LOGISTIC REGRESSION MODEL
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, OneHotEncoderModel, StringIndexer
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
from pyspark.sql.functions import col
# Extract features and label from graph
features = ["id", "CUSTOMER_ID", "INIT_BALANCE", "TX_BEHAVIOR_ID", "ALERT_ID"] # Replace with your actual feature column names
data = g.vertices.select(features + ["IS_FRAUD"])
# Convert IS_FRAUD column to integer type
data = data.withColumn("IS_FRAUD", col("IS_FRAUD").cast("integer"))
# Index string columns
customer_id_indexer = StringIndexer(inputCol="CUSTOMER_ID", outputCol="CUSTOMER_ID_INDEX", handleInvalid="skip")
data = customer_id_indexer.fit(data).transform(data)
# One-hot encode indexed columns
customer_id_encoder = OneHotEncoder(inputCol="CUSTOMER_ID_INDEX", outputCol="CUSTOMER_ID_VEC")
customer_id_model = customer_id_encoder.fit(data)
data = customer_id_model.transform(data)
# Update feature vector
features = ["id", "CUSTOMER_ID_VEC", "INIT_BALANCE", "TX_BEHAVIOR_ID"] # Replace with your actual feature column names
assembler = VectorAssembler(inputCols=features, outputCol="features", handleInvalid="skip")
data = assembler.transform(data)
# Split data into training and test sets
train_data, test_data = data.randomSplit([0.7, 0.3])
# Train logistic regression model
lr = LogisticRegression(featuresCol="features", labelCol="IS_FRAUD")
model = lr.fit(train_data)
# Make predictions on test data
```

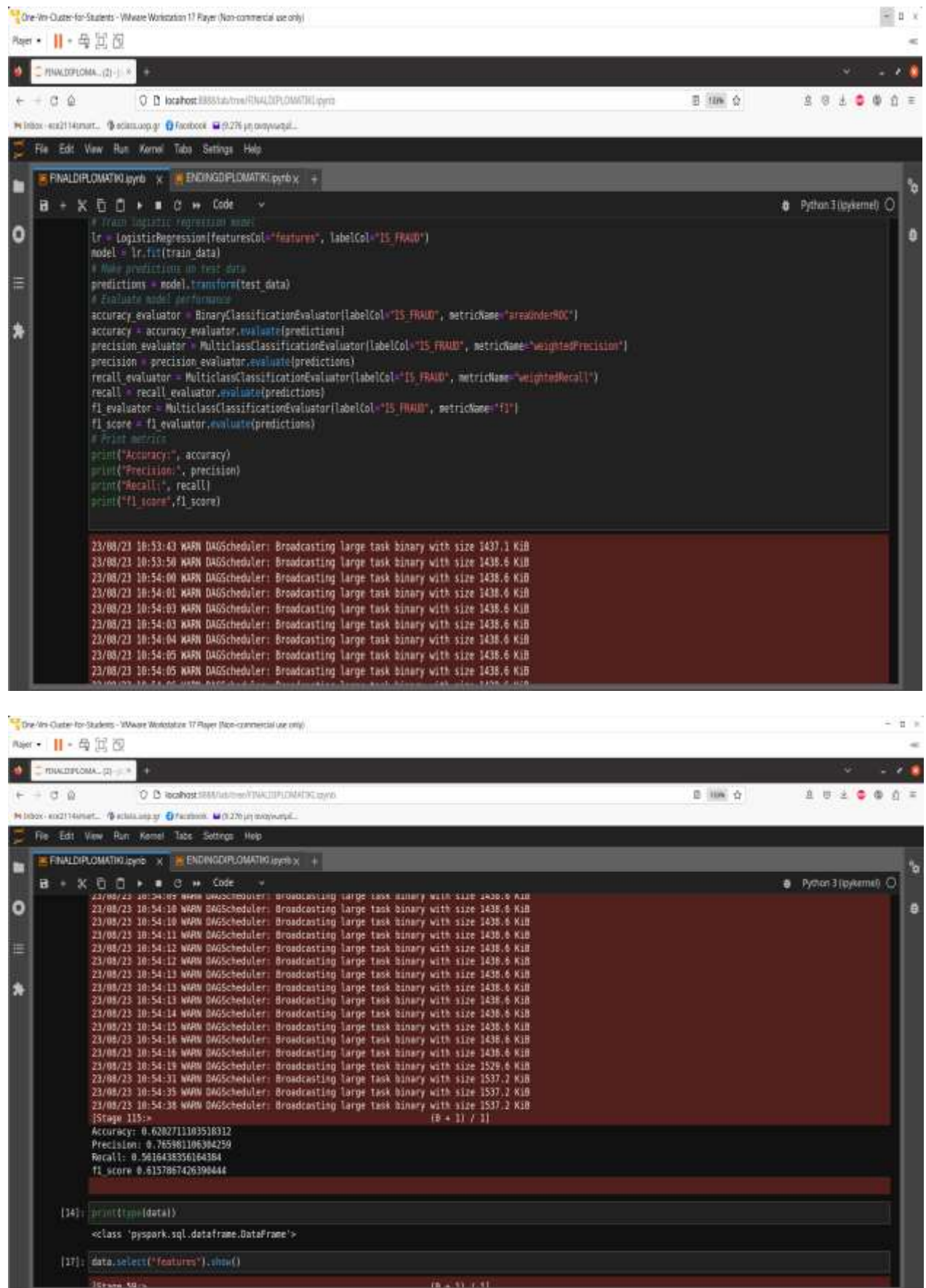
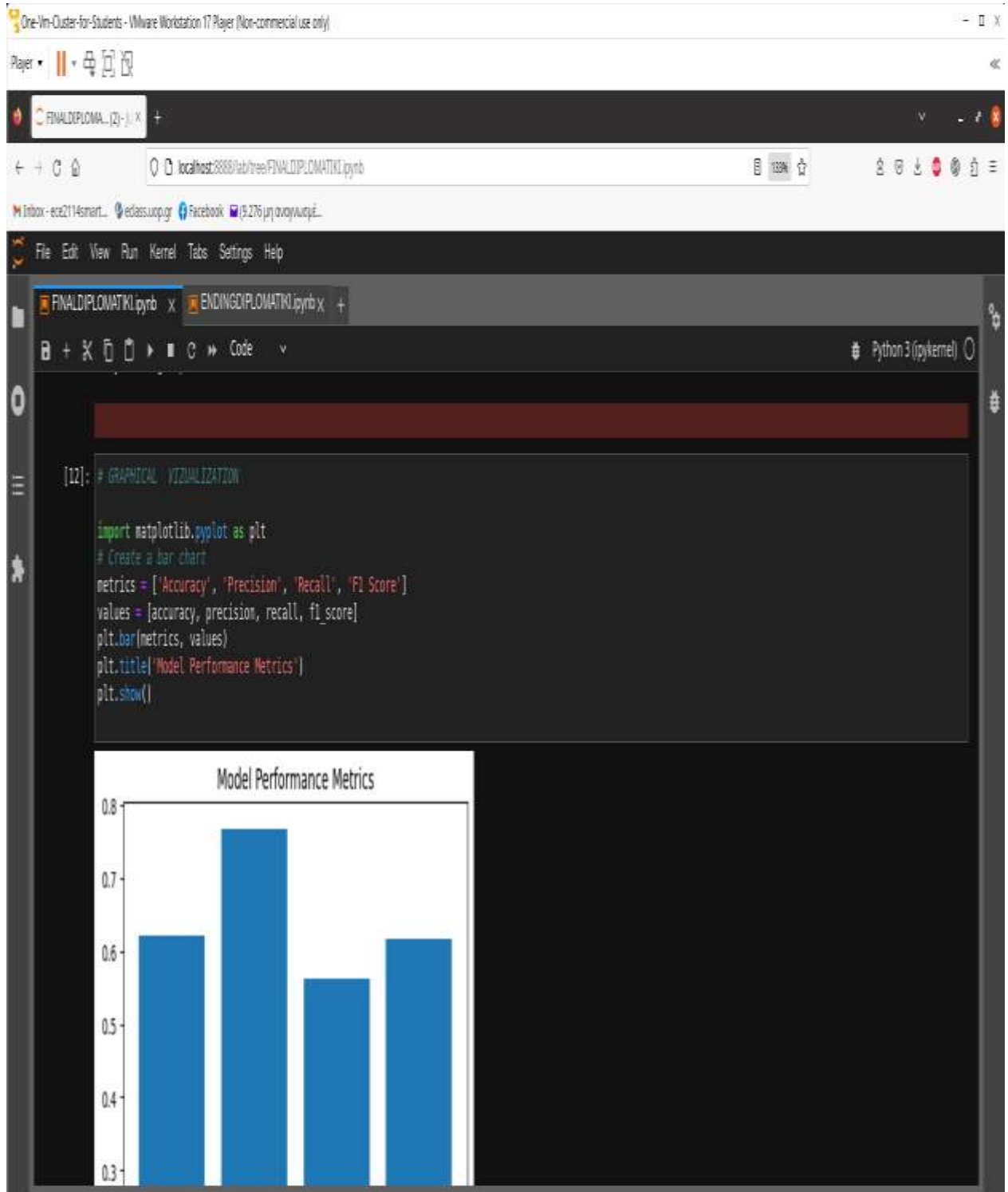



Figure 75 Logistic Regression Model and metrics results

According to our model the accuracy of 0.62 means that 62% of the predictions made by the model were correct, the precision of 0.77 means that 77% of the instances predicted as fraud by the model were actually fraud, the recall of 0.56 means that the model correctly identified 56% of all fraud instances and the f1 score of 0.62 indicates that the model has a good balance between precision and recall.



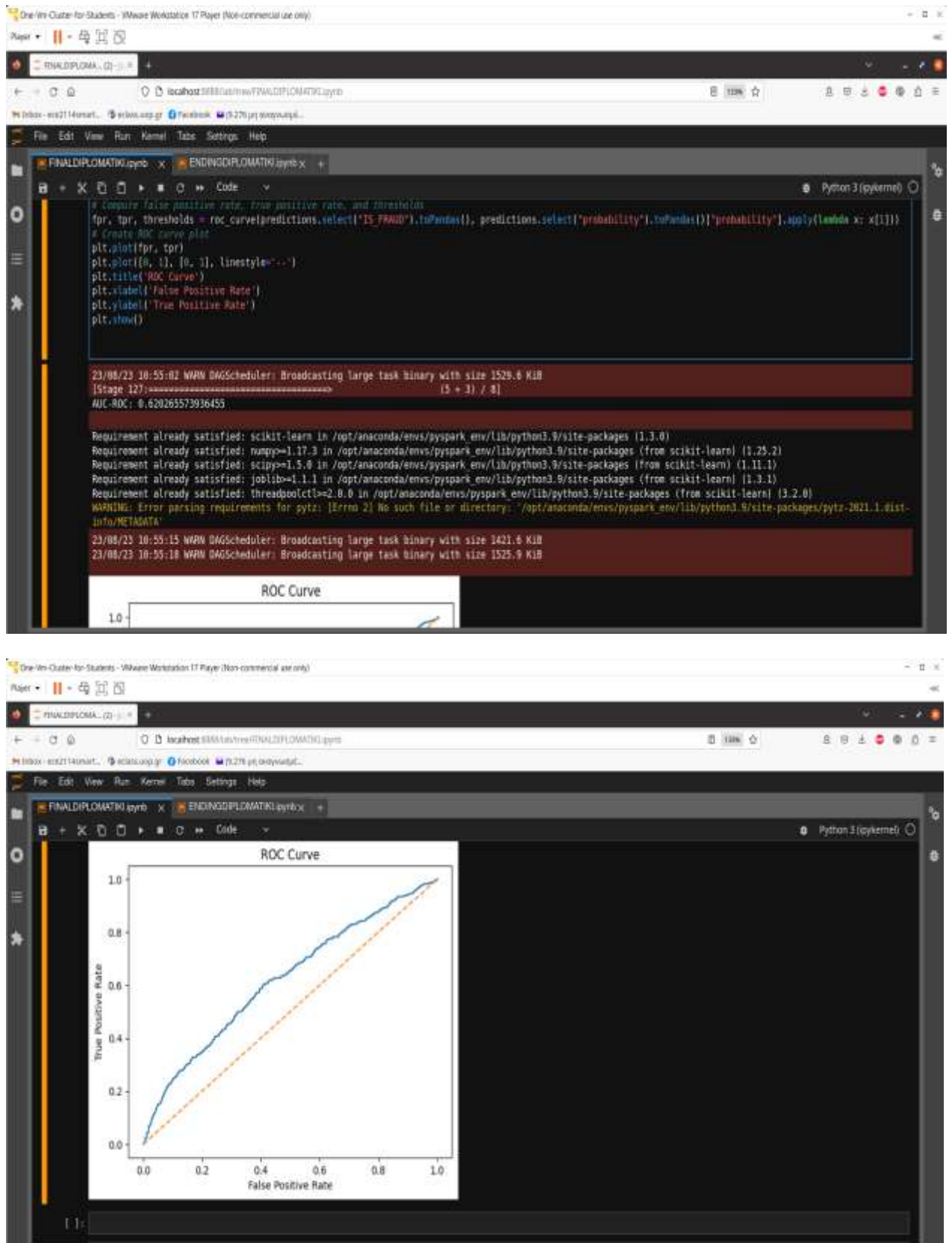


Figure 78 Result and graphical representation of AUC-ROC

According to the result AUC-ROC: 0.620265573936455 our model has moderate predictive power.

8.4.2 RANDOM FORESTS MODEL

```

[14]: # RANDOM FORESTS MODEL
from pyspark.sql import SparkSession
from graphframes import GraphFrame
from pyspark.sql.functions import lit, when, col
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, OneHotEncoderModel, StringIndexer
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
# Extract features and label from graph
features = ["id", "CUSTOMER_ID", "INIT_BALANCE", "TX_BEHAVIOR_ID", "ALERT_ID"] # Replace with your actual feature column names
data = g.vertices.select(features + ["IS_FRAUD"])
# Convert IS_FRAUD column to integer type
data = data.withColumn("IS_FRAUD", col("IS_FRAUD").cast("integer"))
# Index string columns
customer_id_indexer = StringIndexer(inputCol="CUSTOMER_ID", outputCol="CUSTOMER_ID_INDEX", handleInvalid="skip")
data = customer_id_indexer.fit(data).transform(data)
# One-hot encode indexed columns
customer_id_encoder = OneHotEncoder(inputCol="CUSTOMER_ID_INDEX", outputCol="CUSTOMER_ID_VEC")
customer_id_model = customer_id_encoder.fit(data)
data = customer_id_model.transform(data)
# Update feature vector
features = ["id", "CUSTOMER_ID_VEC", "INIT_BALANCE", "TX_BEHAVIOR_ID"] # Replace with your actual feature column names
assembler = VectorAssembler(inputCols=features, outputCol="features", handleInvalid="skip")
data = assembler.transform(data)
# Split data into training and test sets
train_data, test_data = data.randomSplit([0.7, 0.3])
# Train random forest model
rf = RandomForestClassifier(featuresCol="features", labelCol="IS_FRAUD")
model = rf.fit(train_data)
# Make predictions on test data
    
```

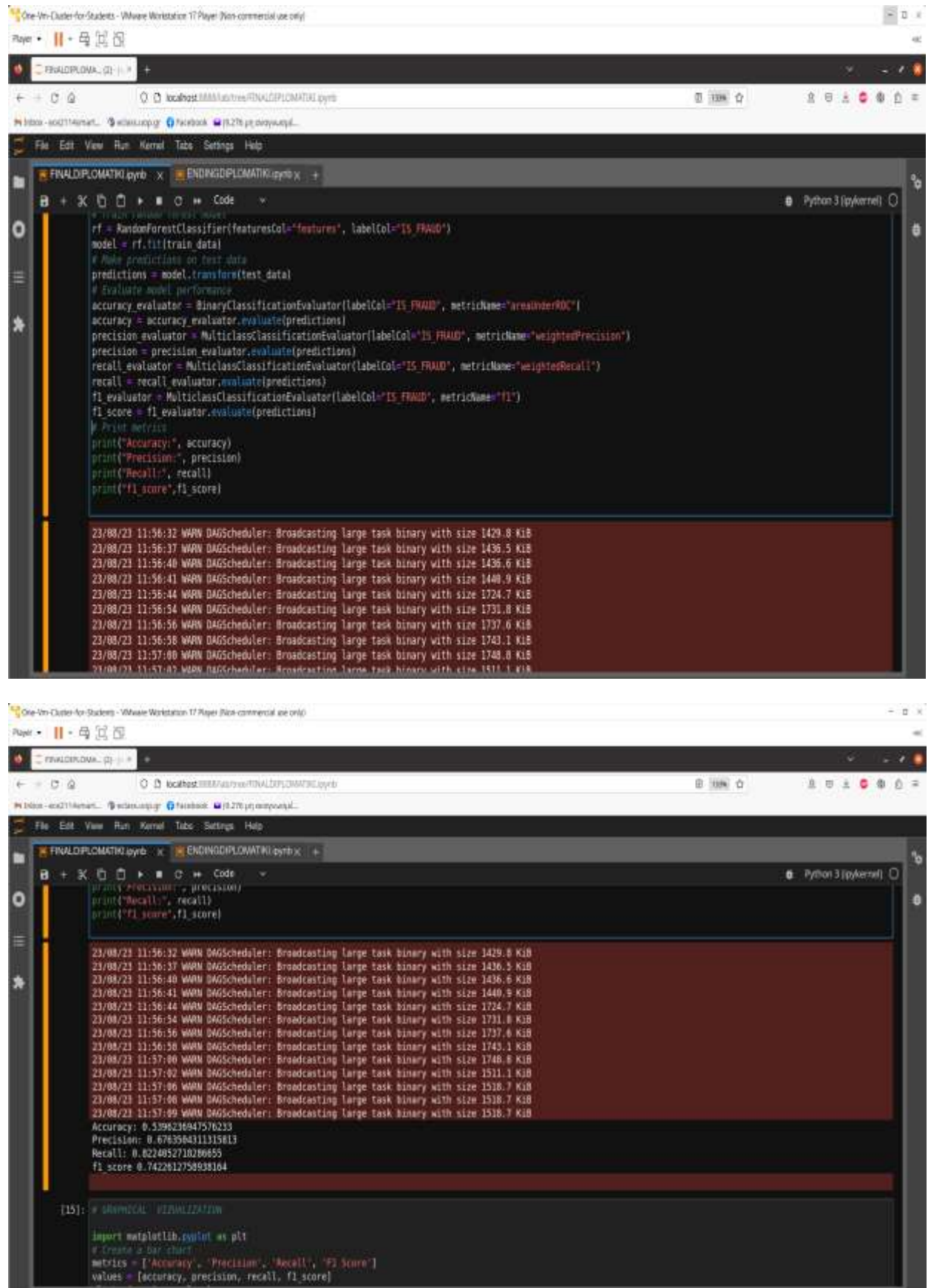
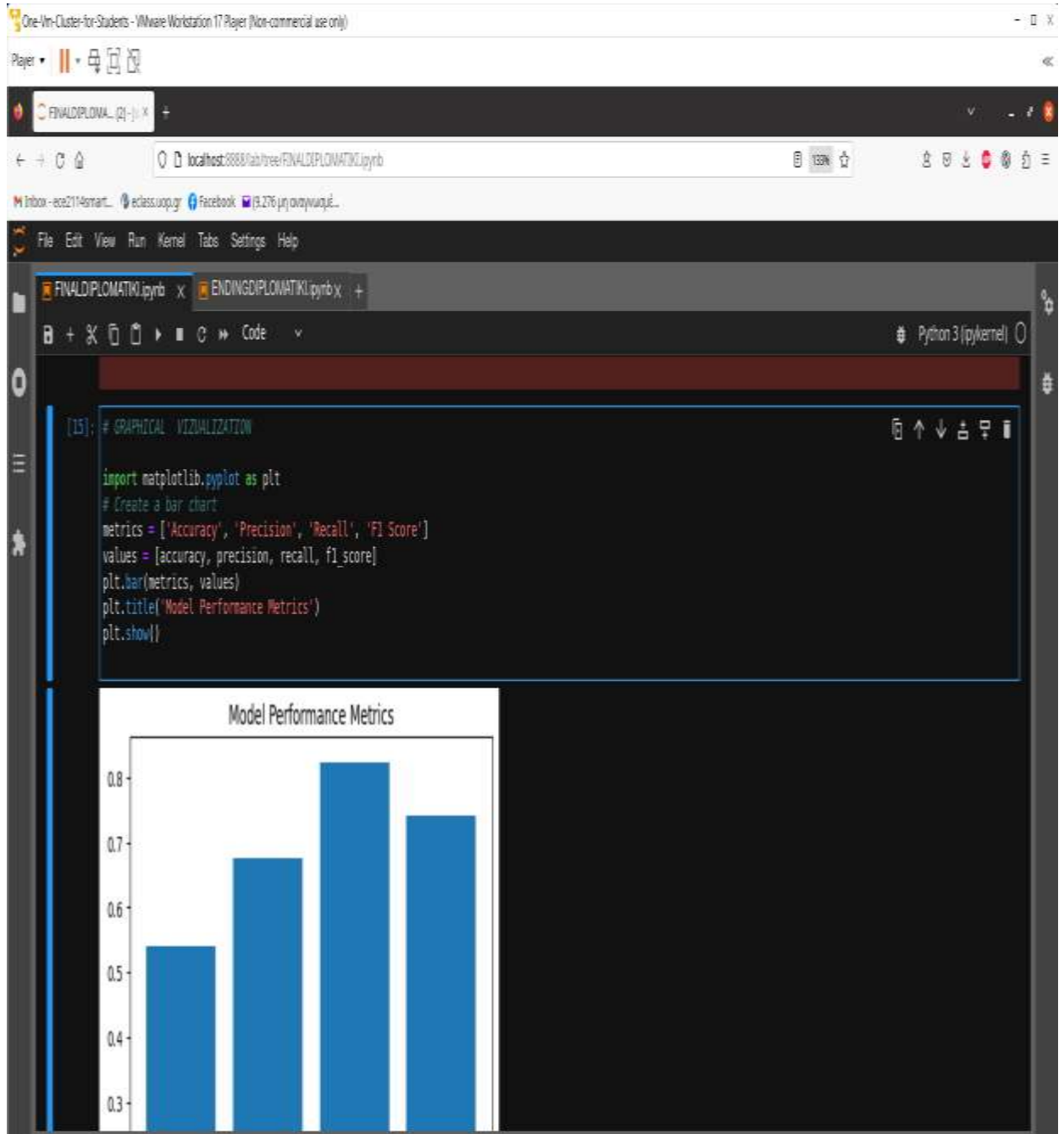


Figure 79 Random Forests Model and metrics results

Master Thesis: << The use of Graph Databases in Financial Problems >>

According to the above mentioned results of four metrics the accuracy of 0.54 means that 54% of the predictions made by the model were correct, the precision of 0.68 means that 68% of the instances predicted as fraud by the model were actually fraud, the recall of 0.82 means that the model correctly identified 82% of all fraud instances and the f1 score of 0.74 indicates that the model has a good balance between precision and recall.



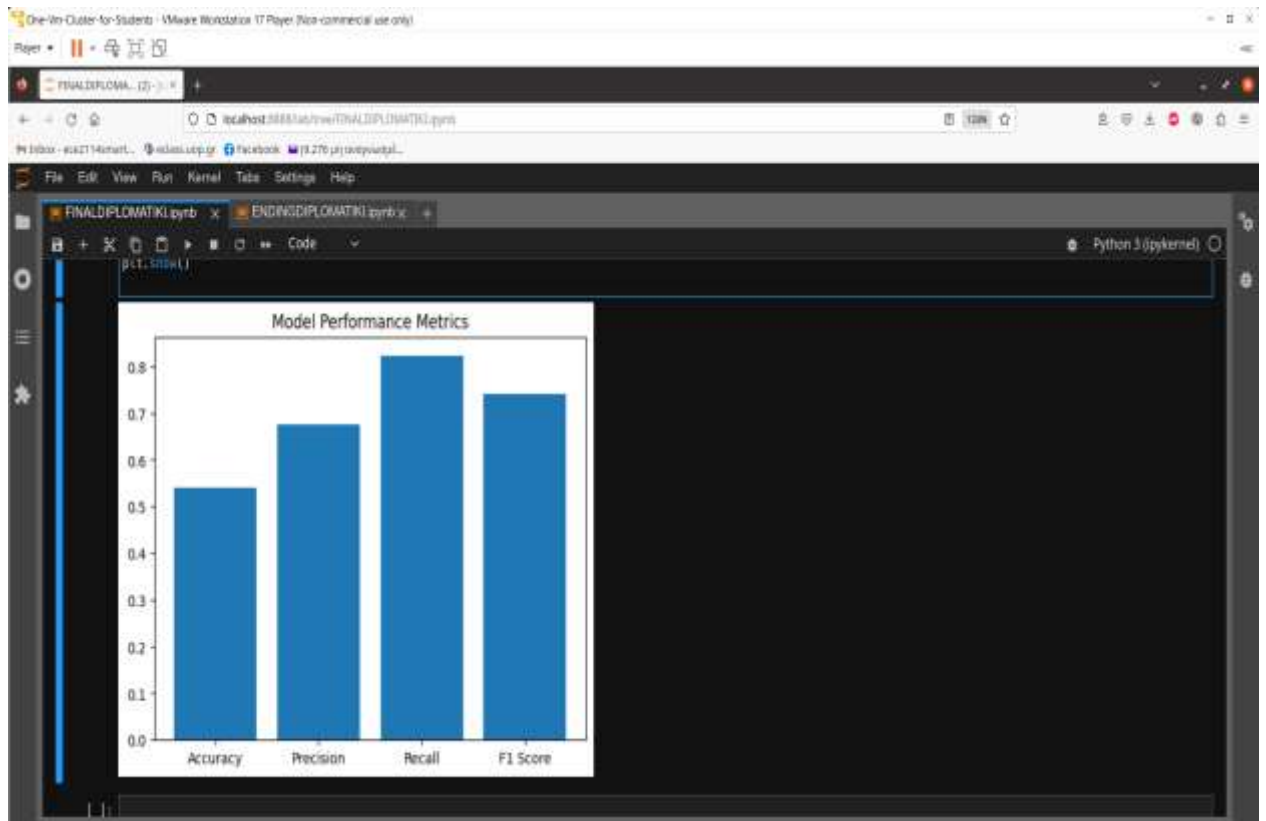


Figure 80 Bar Chart for metrics

```
[16]: from pyspark.ml.evaluation import BinaryClassificationEvaluator
# Compute AUC-ROC
evaluator = BinaryClassificationEvaluator(labelCol='IS_FINAL', metricName='areaUnderROC')
auc_roc = evaluator.evaluate(predictions)
# Print AUC-ROC
print("AUC-ROC:", auc_roc)
pip install scikit-learn
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
# Compute false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(predictions.select('IS_FINAL').toPandas(), predictions.select('probability').apply(lambda x: x[1]))
# Create ROC curve plot
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()

23/08/23 17:13:02 WARN DAGScheduler: Broadcasting large task binary with size 1511.1 KiB

AUC-ROC: 0.5396236947576233
Requirement already satisfied: scikit-learn in /opt/anaconda/envs/pyspark_env/lib/python3.9/site-packages (1.3.0)
Requirement already satisfied: numpy<=1.17.3 in /opt/anaconda/envs/pyspark_env/lib/python3.9/site-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy<=1.5.0 in /opt/anaconda/envs/pyspark_env/lib/python3.9/site-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib<=1.1.1 in /opt/anaconda/envs/pyspark_env/lib/python3.9/site-packages (from scikit-learn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda/envs/pyspark_env/lib/python3.9/site-packages (from scikit-learn) (3.2.0)
```

Figure 81 Code for AUC-ROC

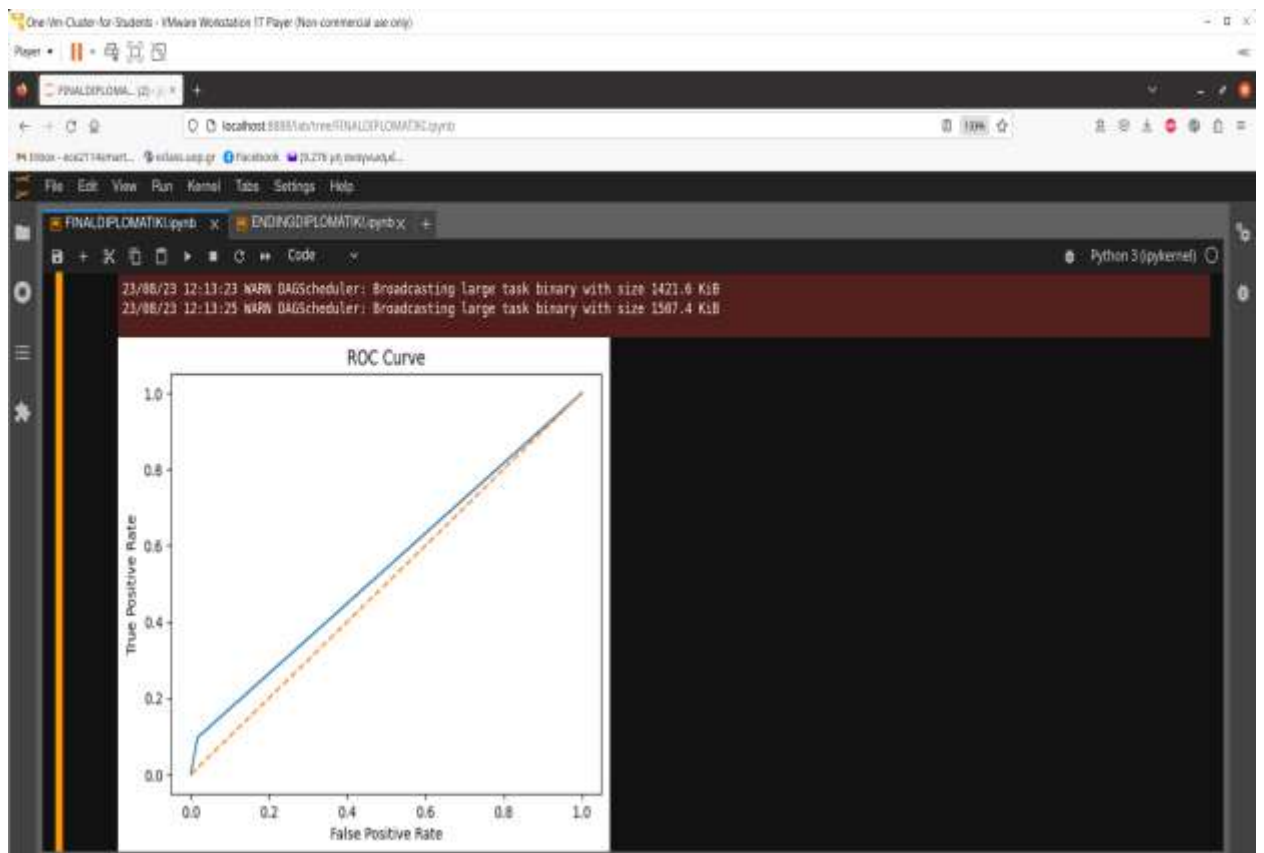
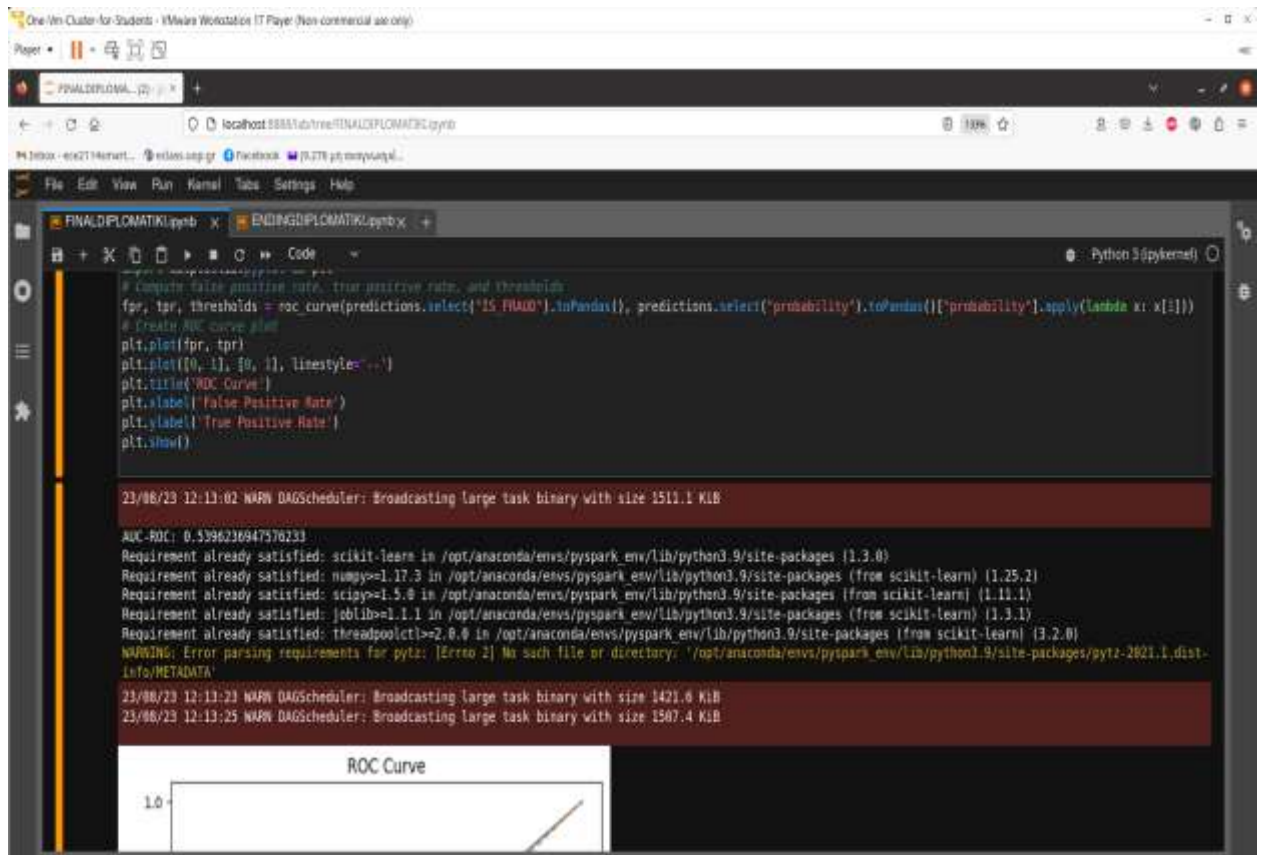


Figure 82 Result and graphical representation of AUC-ROC

The AUC-ROC of our model is 0.54, which means that the model has moderate predictive power.

9. Conclusions

In the first part of this master thesis we examined the range use of graph databases in the financial sector, one of the most promising area from the scope of technology. The strength of a database can be measured using four principal factors: **Integrity, performance, efficiency** and **scalability**. The data query ought to become quicker and simpler – the main purpose of graph databases can be roughly summarized in this way. Where relational databases reach their capacity limits, the graph-based model is particularly agile, because complexity and the quantity of data don't negatively influence the query process in this model.

Also, with the graph database model, **real facts can be stored in a natural way**. The structure used is very similar to human thinking, and this is why the links are so clear to human perception.

Graph databases are not a complete solution, though. They are **limited**, for example, where **scalability** is concerned. As they are principally designed for one-tier architecture, growth represents a (mathematical) challenge. Plus, there is still no uniform query language.

One of the difficulties we came across, was the lack of banking data about their transactions over a period of time. So, as a result we were forced to resort to synthetic banking transaction data.

Next we provide a brief historical review about SQL and NoSQL databases and continue by proposing an AML methodology which is divided into three stages:

Firstly we make use of standard deviation or a rolling window of transactions based on a specified window size and step with the purpose of forming a list of suspicious transactions without based on rules, like prior AML procedures, by combining it with Neo4j database and Cypher.

Secondly, as examined case, we choose the suspicious transaction with the highest standard deviation score and by performing graph algorithms like DFS, BFD or DEGREE CENTRALITY we examine the nodes and relationships that included in the structure of this transaction with the use of Neo4j and Apache Spark.

The results show that Neo4j, in comparison to Apache Spark, provide a better and more understandable and detailed graphical visualization of the transaction construction but in Apache Spark the CENTRALITY DEGREE ALGORITHM provides a more enlightening result.

Lastly we train and test for prediction two machine learning models LOGISTIC REGRESSION and RANDOM FORESTS in Neo4j and in Apache Spark.

In Apache Spark, both models succeeded a better overall scoring about accuracy_score, precision_score, recall_score and f1_score than in Neo4j something that reversed concerning the AUC-COV score.

To sum up, this master thesis has demonstrated the potential of graph databases in the financial sector, particularly in the field of anti-money laundering. By leveraging the natural and intuitive structure of graph databases, combined with advanced machine learning techniques, it is possible to develop sophisticated methodologies for detecting suspicious transactions. The results of this study highlight the strengths and limitations of different technologies, including Neo4j, Cypher, Graphs, Apache Spark, and Python, and provide valuable insights into their potential applications in the financial sector. Future research could build on these findings by exploring new approaches to scalability and developing more advanced query languages for graph databases. Ultimately, this thesis represents a small contribution to the field of financial technology and has the potential to drive innovation and progress in the industry.

Bibliography

1. **Deloitte**. Knowledge Graphs for Financial Services.
2. **ORACLE**. 17 Use Cases for Graph Databases and Graph Analytics . 2021.
3. *GRAPH DATABASE MODELING OF A 360-DEGREE E-CUSTOMER VIEW IN B2C E-COMMERCE*. **Ilija Hristoski¹, Tome Dimovski**. 2021. International May Conference on Strategic Management.
4. **Springer US, US Bureau of Labor Statistics (BLS),Zippia**. Machine Learning in Finance: 10 Applications and Use Cases. *PLATFORM/Coursera*. 2023.
5. **L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo**. *360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming*. 2017.
6. *CLS: A Cross-user Learning Based System for Improving QoE in 360-degree Video Adaptive Streaming*. **L. Xie, X. Zhang, and Z. Guo**. NY USA : 26th ACM International Conference on Multimedia (MM '18). ACM, New York,, 2018.
7. **Nahrstedt, Jounsup Park Klara**. Navigation Graph for Tiled Media Streaming. *Session 1D: Live Multimedia Applications & Streaming*. 2019.
8. **Kumar, Anjani**. AI and ML in Financial Services Compliance Management: Use Cases for FIs. 2018.
9. **Platform/Coursera**. Machine Learning in Finance: 10 Applications and Use Cases. 2023.
10. **Aashaka Shah, Vinay Banakar, Supreeth Shastri, Melissa Wasserman, Vijay Chidambaram**. Analyzing the Impact of GDPR on Storage Systems.
11. **DB-ENGINES**. *DB-Engines Ranking - Trend of Neo4j Popularity / Ranking Neo4j > Trend*.
12. **Crichton, Danny**. *Neo4j raises \$325M as graph-based data analysis takes hold in enterprise*. 2021.
13. **Rita Korányi, José A. Mancera, Michael Kaufmann**. GDPR-Compliant Social Network Link Prediction in a Graph DBMS: The Case of Know-How Development at Beekeeper. *knowledge*. 2022.

14. **Aditya Grover, Jure Leskovec.** *node2vec: Scalable Feature Learning for Networks.* 2016.
15. **Stegmann, Volker Liermann & Claus.** *THE DIGITAL JOURNEY OF BANKING AND INSURANCE.* 2021. p. 39.
16. **Kareem S. Aggour, Jenny Weisenberg Williams, Justin McHugh, Vijay S. Kumar.** *Colt: Concept Lineage Tool for Data Flow Metadata Capture and Analysis.*
17. **JACK NICHOLLS, ADITYA KUPPA , AND NHIEN-AN LE-KHAC , (Member, IEEE).** *Financial Cybercrime: A Comprehensive Survey of Deep Learning Approaches to Tackle the Evolving Financial Crime Landscape.* 2021.
18. **Eren Kurshan, Hongda Shen.** *Graph Computing for Financial Crime and Fraud Detection: Trends, Challenges and Outlook.*
19. **XU, MENGJIA.** *UNDERSTANDING GRAPH EMBEDDING METHODS AND THEIR APPLICATIONS.*
20. **Mário Cardoso, Pedro Saleiro, Pedro Bizarro.** *LaundroGraph: Self-Supervised Graph Representation Learning for Anti-Money Laundering.* 2022.
21. **Yueming Sun, Yi Zhang.** *Conversational Recommender System.* 2018.
22. **Wenqiang Lei, Xiangnan He , Yisong Miao , Qingyun Wu , Richang Hong , Min-Yen Kan, Tat-Seng Chua.** *Estimation–Action–Reflection: Towards Deep Interaction Between Conversational and Recommender Systems.* 2020.
23. **Wenqiang Lei¹, Gangyi Zhang , Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, Tat-Seng Chua.** *Interactive Path Reasoning on Graph for Conversational Recommendation.* 2020.
24. **VASILIOS TAMPAKAS.** *Introduction to Databases.* PATRA GREECE : GOTSIS, 2021.
25. **Andreas Meier, Michael Kaufmann.** *SQL & NoSQL Databases.* Fribourg, Switzerland : Springer Vieweg, 2019.
26. **Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, Dishant Gosain.** *A Survey and Comparison of Relational and Non-Relational Database.* *International Journal of Engineering Research & Technology (IJERT).* 2012.

27. **Sitalakshmi Venkatraman, Kiran Fahd, Samuel Kaspi, Ramanathan Venkatraman.** SQL Versus NoSQL Movement with Big Data Analytics. *I.J. Information Technology and Computer Science*. December 2016.
28. **Adity Gupta, Swati Tyagi, Nupur Panwar, Shelly Sachdeva.** NoSQL Databases: Critical Analysis and Comparison. *Upaang Saxena Minjar Cloud Services Private Limited*. October 2017.
29. **Kosovare Sahatqija, Jaumin Ajdari, Xhemal Zenuni, Bujar Raufi, Florije Ismaili.** *Comparison between relational and NOSQL databases*. Tetovo : Contemporary Sciences and Technologies, South East European University, 2018.
30. **Ameya Nayak, Anil Poriya, Dikshay Poojary.** Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems (IJ AIS)*. March 2013.
31. **Sudhakar, Kalyan.** Difference between SQL and NoSQL Databases. *International Journal of Management, IT & Engineering*. June 2018.
32. **Ian Robinson, Jim Webber & Emil Eifrem.** *Graph Databases*. Menlo Park, California : O'REILLY, 2013.
33. **Negro, Alessandro.** *Graph-Powered Machine Learning*. August 2021. 9781617295645.
34. **Tang, Jiliang.** *Deep Learning on Graphs*. s.l. : Cambridge University Press, 2021. 9781108924184.
35. **Hodler, Mark Needham & Amy E.** *Graph Algorithms*. Booz Allen Hamilton : O'REILLY, 2019.
36. **See, Steve Mussmann and Abi.** *Algorithms Graph Search*.
37. **Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters & Dagmar Tenfelde-Podehl.** *Algorithms for Centrality Indices*.
38. **Bhatia, Ruchi Mittal & M. P. S.** *Classification and Comparative Evaluation of Community Detection Algorithms*.
39. **Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia.** *Learning Spark: Lightning-Fast Big Data Analysis*.
40. **Aven, Jeffrey.** *Apache Spark in 24 Hours, Sams Teach Yourself*. 2016.

41. **Warren, Holden Karau and Rachel.** *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark* .
42. **Lyon, William.** *Fullstack GraphQL Applications with React, Node.js, and Neo4j*. s.l. : Manning.
43. **Anthapu, Ravindranatha.** *Graph Data Processing with Cypher*. s.l. : Packt.
44. **Das, Abhik.** *Encyclopedia of Quality of Life and Well-Being Research*,. Rockville, MD, USA : F. Maggino, 2021.
45. **Klein, David G. Kleinbaum and Mitchel.** *Logistic Regression: A Self-Learning Text* .
46. **Wikipedia.** *Random Forests*.
47. **Louppe, Gilles.** *UNDERSTANDING RANDOM FORESTS*.
48. **Hartshorn, Scott.** *Machine Learning With Random Forests And Decision*. s.l. : Kindle Edition.
49. **Zheng, Alice.** *Evaluating Machine Learning Models*. s.l. : O'Reilly Media, 2015.
50. **Burkov, Andriy.** *The Hundred-Page Machine Learning Book*.
51. **Trevor Hastie, Robert Tibshirani, and Jerome Friedman.** *The Elements of Statistical Learning*.
52. **Theobald, Oliver.** *Machine Learning for Absolute Beginners*.
53. **Burkov, Andriy.** *The Hundred-Page Machine Learning Book*.
54. **Scifo, Estelle.** *Graph Data Science with Neo4j*. s.l. : Packt.
55. **Frisendal, Thomas.** *Visual Design of GraphQL Data: A Practical Introduction with Legacy Data and Neo4j*. s.l. : Apress.
56. **Perrin, Jean-Georges.** *Spark in Action*. 2020.
57. **Nabi, Zubair.** *Pro Spark Streaming: The Zen of Real-Time Analytics Using Apache Spark*. 2016.
58. **Martin, Daniel Jurafsky & James H.** *Speech and Language Processing* .
59. **Murphy, Kevin P.** *Machine Learning: A Probabilistic Perspective*.

Master Thesis: << The use of Graph Databases in Financial Problems >>

60. Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters & Dagmar Tenfelde-Podeh. *Algorithms for Centrality Indices.*

