

DATA ANALYTICS WITH COGNOS

COVID VACCINES

PHASE-3

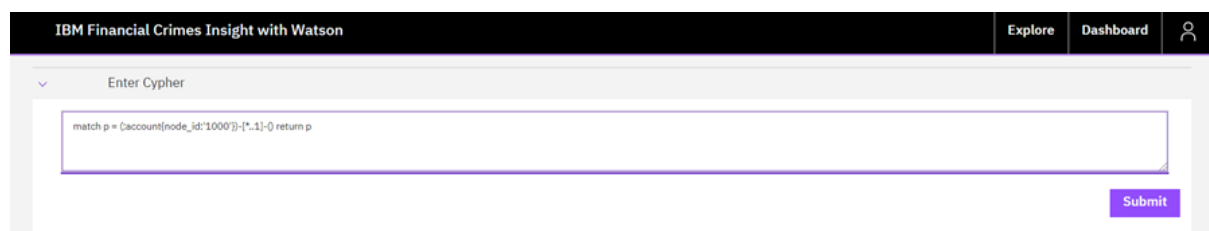
Data exploration and visualization

The Explore page allows you to try out exploratory analysis of the data and also visualize the data. This page allows you to try custom queries to help the you understand the data better and to try out detecting any new patterns. The data exploration component uses a open-source based Cypher library to explore the data.

The Explorer page allows you to enter a cypher query and run the query on the graph data. The results can be seen either as a table or network, depending on the type of query. For example, if a user wants to fetch all accounts where the credits are coming from 1000 or more accounts, the results can be viewed as a table and sort by the in-degree (number of accounts from where the money is flowing into). In another example, if a user wants to see the two-hop network of money flow of Account A, results can be viewed as network graph.

Using the Explore page in Graph Analytics

The Explore page provides a query pane for the user to enter the query. The output of the query can either be a list of nodes or a network graph.



IBM Financial Crimes Insight with Watson

Explore Dashboard

Enter Cypher

```
match p = (account{node_id:'1000'})-[*..1]-() return p
```

Submit

In queries with a network-based response, the user can further explore by clicking the nodes to expand the network.



On queries that return a list response, the user has an ability to click a node entry to explore the node's neighborhood in the graph view. The user can further apply filters available for the network graph.

IBM Financial Crimes Insight with Watson								Explore	Dashboard	
Enter Cypher										
Graph Table										
search for account id or party name										
Investigate Edit										
<p>Michael Tiongco... Anthony Parra V... Alexis Hughes</p> <p>GBP 77183.6 GBP 77183.6</p>										
Nodes										
Id	Node_id	Label	Title	Type	Account ID	Association Score	SAR			
41088	1050	account	Daisy Zamora	SAV	1050	1.00	true			
1155112	2625	account	Kellen Marchbanks	CHK	2625	1.00	true			
24616	1041	account	Brelen Mohamed	CHK	1041	1.00	true			
40964176	1000	account	Anthony Parra Velez	SAV	1000	1.00	true			
43700304	3620	account	Sylvester Union	SAV	3620	1.00	true			
12408	1021	account	Karisa Teter	CHK	1021	1.00	true			
57432	1046	account	Matenah Verzani	SAV	1046	1.00	true			
1962104	4926	account	Navinah Eidenschink	CHK	4926	1.00	true			
41832528	6543	account	Karen Harrington	SAV	6543	0.36	false			
503928	6674	account	Isaac Raps	SAV	6674	0.35	false			

Sample Cypher queries

Getting a network graph for a node:

For a given account number '1000', this query fetches all the accounts (1 hop) it has transacted with and also the accounts the first hop accounts are transacted with. In social network, this is called as "friends of friends."

```
match p = (:account{node_id:'1000'})-[*..2]-() return p
```

Getting a list of nodes based on a property value:

This query returns a list of accounts whose association_score is greater than 0.25. The same can be modified to try on other properties of the account or party and with different other conditions (equals, less then, and so on).

```
match p = (m:account) where m.association_score>0.25 return p
limit 25
```

Filtering a network based on an edge property:

This query helps the user to filter the relationships based on property of the relation. In this example, it is transaction. The query fetches all transactions of the account '1000' whose transaction amount is greater than \$100.

```
match p = (:account{node_id:'1000'})-[t*..1]-() where all(x in t
where x.transaction_amount > 100.00) return p
```

Getting super nodes:

This query returns the nodes as a list where the number of out nodes are greater than 10.

```
match (n)-[:transaction]-(outNode) WITH n, count(outNode.node_id)
AS outNodes WHERE outNodes >= 10 RETURN n, outNodes LIMIT 10
```

Temporal order of transactions between 2 nodes in 3 hops:

This query detects transaction flow from one account to another with up to three intermediate accounts in temporal order (A > B > C > D).

```
match (a:account{node_id:'6506'}) match
(b:account{node_id:'1001'}) match p = (a)-[t1:transaction]->()-
[t2:transaction]->()-[t3:transaction]->(b) where
t1.transaction_datetime < t2.transaction_datetime and
t2.transaction_datetime < t3.transaction_datetime return p limit 1
```

Cycles for a node:

This cypher detects circular money flow involving a given account/party. For example, it is considered circular money flow if A transfers money to B, B transfers money to C, and then C transfers money back to A.

```
match (m1:account{node_id:'1055'})-[t]->(m2:account) match
cyclePath=(m2)-[*..4]->(m1) RETURN m1, t,
nodes(cyclePath),relationships(cyclePath) as cycles limit 1
```

Fan-In nodes:

This cypher detects if there is bunch of small transactions to one particular account whose aggregated credit total is more than a given transaction value. For example, if ten people send money to Person A within a given time, and if the aggregated value is more the given amount, it is considered Fan-in. This is useful to detect where people get multiple credits in small chunks.

```
match(fanIn:account)<-[:transaction]-(b:account) with fanIn,
count(b) as inNodes order by inNodes desc limit 10
match(a:account)<-[t:transaction]-(fanIn) return
fanIn,sum(t.transaction_amount) as totalDebit,inNodes order by
inNodes desc
```

Fan-Out nodes:

This cypher detects if the person A sends money to multiple people within a time period and if the aggregated debit is more than the given value. This is useful to detect suspicious money flow to many accounts where the actual beneficiary may be the same person.

```
match(fanOut:account)-[:transaction]->(b:account) with fanOut,
count(b) as outNodes order by outNodes desc limit 10
match(fanOut)<-[t:transaction]-(a:account) return
fanOut,sum(t.transaction_amount) as totalCredit,outNodes order by
outNodes desc
```

Supervised learning regression

Supervised learning regression is a type of machine learning algorithm used to predict continuous numerical values based on labeled training data. It involves training a model on input-output pairs, where the input is a set of features and the output is a continuous target variable.

Here are some commonly used supervised learning regression algorithms:

Linear Regression: Fits a linear equation to the data by minimizing the sum of squared differences between the predicted and actual values.

Decision Trees: Builds a tree-like model where each internal node represents a feature, each branch represents a decision rule, and each leaf node represents the predicted value.

Random Forest: An ensemble method that combines multiple decision trees to make predictions. It reduces overfitting and improves accuracy.

Support Vector Regression (SVR): A regression version of Support Vector Machines (SVM) that finds a hyperplane in a higher-dimensional space to minimize the error between predicted and actual values.

Gradient Boosting Regression: Builds an ensemble of weak prediction models in a sequential manner, where each new model corrects the mistakes made by the previous models.

When using supervised learning regression, it is important to preprocess the data, split it into training and testing sets, select appropriate features, tune hyperparameters, and evaluate the model's performance using metrics such as mean squared error (MSE) or R-squared.

Keep in mind that the choice of algorithm depends on the specific problem and characteristics of the data.

THAK YOU

SUBMITTED BY: GANGAPATNAM SASIDEEKSHATH

au723921104015

gangapatnamsasideekshath2003@gmail.com