**Script**

**NEO4j:**

Setting up our neo4j database with relations.

LOAD CSV WITH HEADERS FROM 'file:///product.csv' AS row1 FIELDTERMINATOR ';'

MERGE (p:Product {

name: row1.Name,

price: toFloat(row1.Price)

})

MERGE (c:Category {

name: row1.Category

})

MERGE (q:Brand {

name: row1.Comapny

})

MERGE (q)-[:OWNS]->(p)

MERGE (p)-[:CATEGORIZED\_AS]->(c);

LOAD CSV WITH HEADERS FROM 'file:///reviews.csv' AS row2 FIELDTERMINATOR ';'

MATCH (p:Product {name: row2.Product})

MERGE (r:Review {

userId: toInteger(row2.user\_id),

reviewText: row2.review\_text,

rating: toFloat(row2.rating),

helpfulVotes: toInteger(row2.helpful\_votes)

})

MERGE (r)-[:REVIEWS]->(p);

Example query:

MATCH (n:Review)-[r:REVIEWS]-(p:Product{name: "Soy Protein"})-[:OWNS]-(b:Brand)

RETURN n,r,p,b

MATCH (p:Product)-[r]-(c) return p, r, c

Graph projection:

CALL gds.graph.project( 'nuna', ['Product', 'Review','Category', 'Brand'], { OWNS: {}, REVIEWS:{}, CATEGORIZED\_AS:{} } );

Used to make node similairity, to get similar products recommended:

export async function getServerSideProps(context) {

const {params} = context

const {productId} = params

const res1 = await read(`

MATCH (p: Product)

WHERE ID(p) = ${productId}

RETURN p

`);

const product = JSON.stringify(res1.map((record) => record.p));

const res2 = await read(`CALL gds.alpha.nodeSimilarity.filtered.stream('nuna', {

sourceNodeFilter: ${productId}

})

YIELD node1, node2, similarity

RETURN gds.util.asNode(node2) AS Product2,

similarity

ORDER BY similarity DESCENDING, Product2 LIMIT 4`)

Used to get reviews on each product you are looking at.

export async function getServerSideProps( context ) {

const {params} = context

const {productId} = params

const res = await read(`

MATCH (n:Review)-[f:REVIEWS]-(p:Product)-[:OWNS]-(b:Brand) WHERE id(p) = ${productId} RETURN n

`);