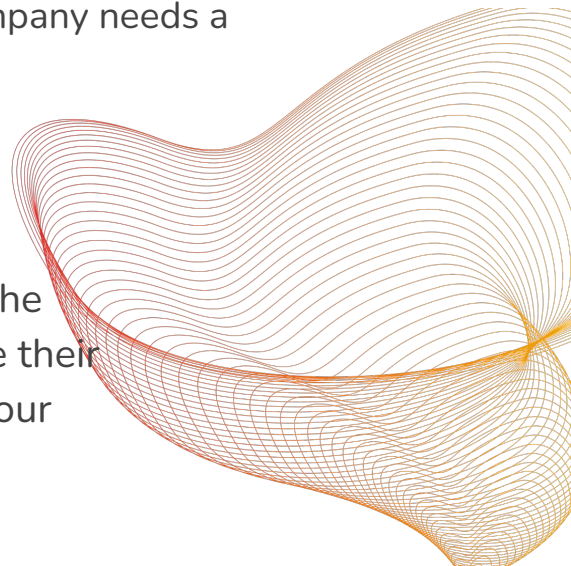


Case Study & Hands-On Project for **Global RX**

Global RX is a retail startup with no data warehouse but in Excel files.... Our task is to help them spin up a data warehouse that should be a free offering for a start. The solution should be something that can easily be automated to help the business easily gain insights into their best selling items, clients and locations. The Company needs a POC (Proof of Concept) in 2 days.

Sample Data

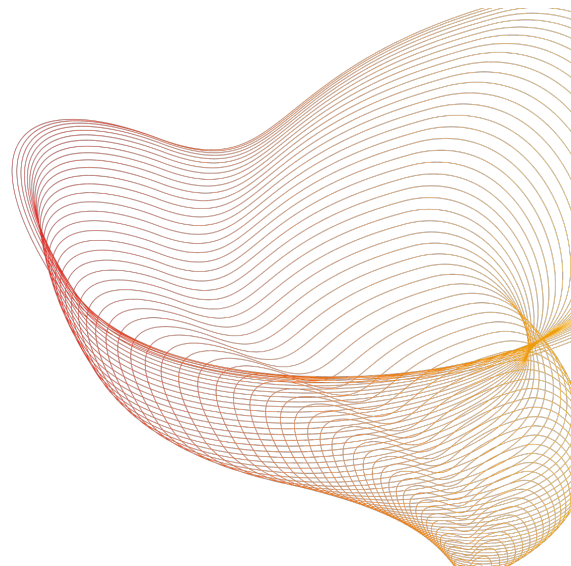
A sample dataset in the file **data/global-superstore-data.xlsx** on the **Orders** sheet, has been provided to us to help the company achieve their goal. Note that datasets can come in any form subsequently, but your logic should be easy to refactor.



Core Concepts

At the end of this case study, we should have enforced knowledge on

- 1) Problem Statement Breakdown
- 2) Proof Of Concept Planning and Architecture (Before writing Code)
- 3) Data Transformation and Modelling (SQL Python)
- 4) Data Ingestion
- 5) Functional Programming



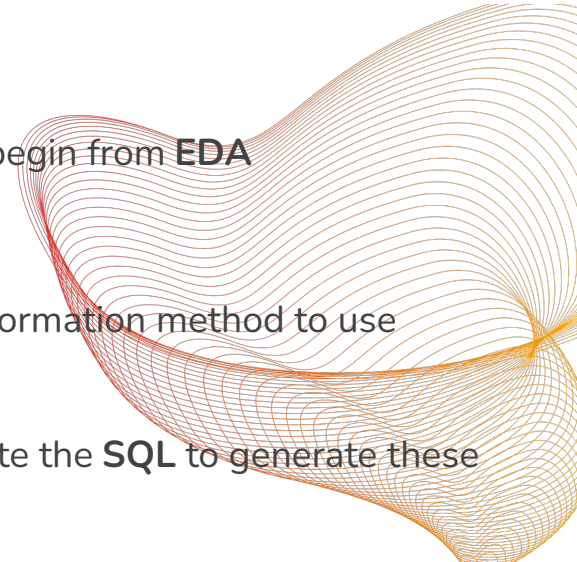
Problem Statement Breakdown

It is a good idea to solve problems by **breaking them down into tasks**. From the company's request above we can see that the deliverables include;

- 1) Building a Data Warehouse
- 2) Semi-Automated Process

Steps to Take

To achieve the above stated, we have to take the following steps which begin from **EDA**

- i) Load Data in to pandas using **pd.read_excel()**.
 - ii) Understand the structure of the business data (and decide what transformation method to use (**ETL/ELT**) as well as tools to implement.
 - iii) Come up with a potential logical /Visual ERD for the business and write the **SQL** to generate these tables.
- 



Data Normalization

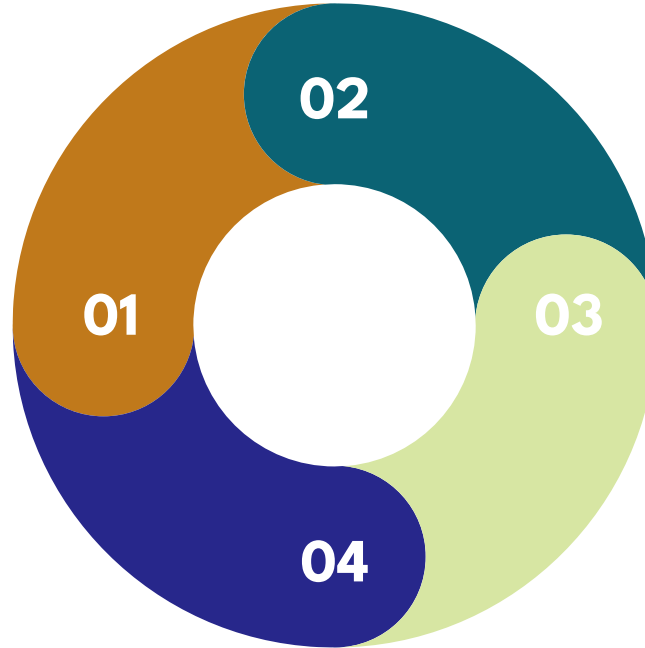
- Create a directory named `outputs/models/`. This directory should be created using the `'os'` library that was taught during the class. This would be used in the next step.
- Perform **normalization** on this dataset Using Pandas. For this task you should select a subset of the dataframe for the 3 dimensional tables and drop duplicates while the orders table (fact) should be as-is with the other columns. See slide 3. eg `products df = df[['Product ID','Product Name']].drop_duplicates(), you can also sort_values to ensure arrangement in your table.`
- Output should be **4 csv files** written to the directory (`outputs/models/`) created in the above step
- Note: during your transformation of this, change all columns names to lower cases and replace the spaces in- between with them with an underscore (Hint: Pandas has a method used to rename columns `df.rename`, but there are other ways around this)
- This is good practice to ensure compatibility of column names around data platforms



Normalized Tables

products.csv (4 columns
->
product_id,category,sub_
category,product_name)

customers.csv (2
columns ->
customer_id,customer_na
me)



locations.csv (3 columns
-> city,state,country)

orders.csv (would have
other columns
customer_id, product_id
and city)



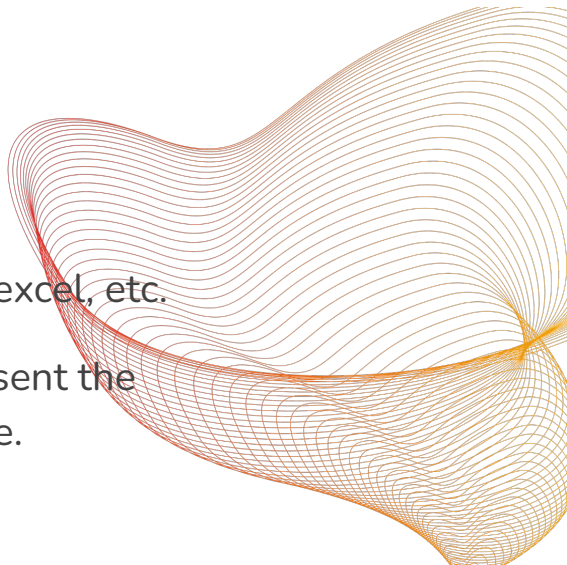
Materializing Tables in Postgres

The next step would be to materialize these tables into a Postgres database using Pandas. At this point, we should have 4 tables in our database based on the business structure. We would use the Pandas built in method to enable us WRITE DATA to our data warehouse.

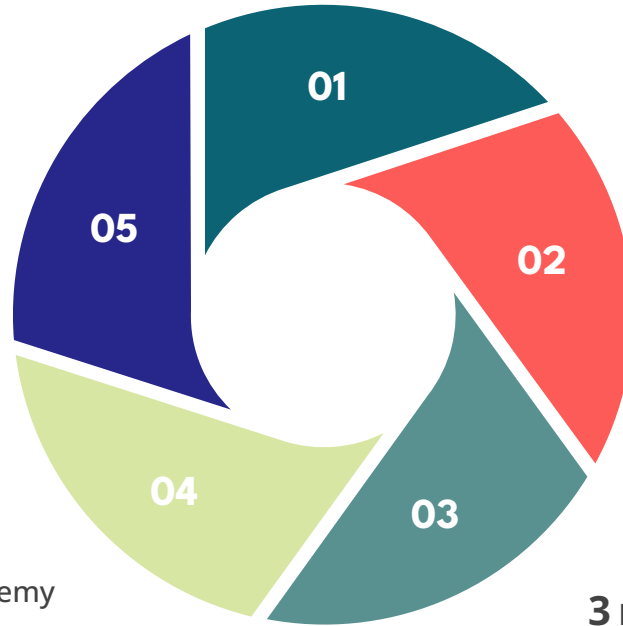
This is done as follows;

```
data = pd.read_csv("file_path") #note that this can also be read_excel, etc.
```

```
data.to_sql(destination_table_name,conn) conn here would represent the  
database connection, with which we are connecting to our database.
```



Approach



1

Create a Free Postgres DB in the cloud.

2 Recommended:

elephantsql.com free plan
(<https://www.elephantsql.com/plans.html>), get the url of your db instance (and supply it to the engine).

3 Example Resource:

https://www.skytowner.com/explore/exporting_pandas_dataframe_to_postgresql_table

5 Replace Postgres with postgresql in the database url to avoid errors

4 - pip install the sqlalchemy and psycopg2 libraries

Functional Programming

To make our process faster, We would write a **function** that takes our now transformed data and loads it to the data warehouse.

Functions help us to abstract repetitive logic and ensure DRY practice is maintained in our code.

*We would first go ahead and execute some DROP table statements in our warehouse as follows; **DROP TABLE products**, we can also decide to **ALTER** our tables to add Timestamp Fields;*

Example

```
def load_to_db(file_path, table_name):  
    try:  
        engine = create_engine(#####)  
        conn = engine.connect()  
  
        data = ##read file_path  
        df.to_sql(table_name, conn)  
        ## add a print statement to show if this  
        process was successful. Check your db to see if it was successful.  
    except Exception as e:  
        raise e
```





Deliverables

- We should have a final python script and a couple of SQL scripts that would be automated later on to solve the business problem for Global RX
 - We would also have a basic diagram showing our workflow process. Which would be useful in explaining our POC to the management/ engineering leadership of Global RX.
- 