

kfzteile24 Data Engineer Test

[Git repo link](#)

Sr.	Content
1.	Given data
2.	Task
3.	Assumptions
4.	Conditions
5.	Workflow
6.	Folder Structure
7.	How to reproduce the project
8.	SQL Queries

Given data

Imagine you are working with a large dataset in a CSV file that contains information about sales transactions for the last month.

The Pricing team has provided `sales_data.csv` CSV file that contains all the information.

Each row in the CSV file represents a single transaction, and the columns include information such as

"TransactionID," "ProductID," "Quantity," "SalePrice" and "PurchasePrice"

Task

1. Clean and analyze the Dataset:

a. Cleaning and transformations:

- Convert the data to numeric data type
- Drop the duplicate rows
- Drop the rows with null values
- Drop the rows with negative values of SalePrice and PurchasePrice
- Check the quantity is greater than 0 and drop the rows if quantity is less than or equal to 0
- Convert TransactionID and ProductId to int data type
- Save the cleaned and transformed dataframe as csv file sales_data_cleaned.csv.

b. Analysis of the relation between Sale Price and Quantity

- i. As the quantity of the product increases, the unit sale price decreases.
- ii. After the certain threshold amount of the quantity, the unit sale price remains the same.(eg. For the quantity greater than or equal to 11, the unit sale price remains the same)

ProductID=1119 screenshot			ProductID=1112 screenshot		
ProductID	Quantity	SalePrice	ProductID	Quantity	SalePrice
1119	1	31.2	1112	1	7.47
1119	3	29.19	1112	2	7.23
1119	5	27.18	1112	3	6.99
1119	6	26.17	1112	4	6.75
1119	8	24.16	1112	5	6.51
1119	9	23.15	1112	6	6.27
1119	10	22.14	1112	7	6.03
1119	11	21.14	1112	8	5.78
1119	12	21.14	1112	9	5.54
1119	13	21.14	1112	10	5.3
1119	14	21.14	1112	11	5.06
1119	15	21.14	1112	12	5.06
1119	16	21.14	1112	13	5.06
1119	17	21.14	1112	14	5.06
1119	18	21.14	1112	15	5.06
1119	19	21.14	1112	16	5.06
1119	20	21.14	1112	17	5.06
1119	21	21.14	1112	18	5.06
1119	23	21.14	1112	19	5.06
1119	24	21.14			
1119	25	21.14			
1119	18493899	23.15			

2. Store data into database

- a. The python script transfer_sales_data.py reads sales_data_cleaned.csv using csv python library.
- b. The database connection and data insertion operation is done using psycopg2 python library.

3. Implement Process Data Python Function

- a. The python script app.py implements process_data() function.
- b. This function uses the database connection defined in the class called ProcessCSVFile of transfer_sales_data.py script.
- c. Then we execute the SQL queries and collect the result and return them as a tuple.

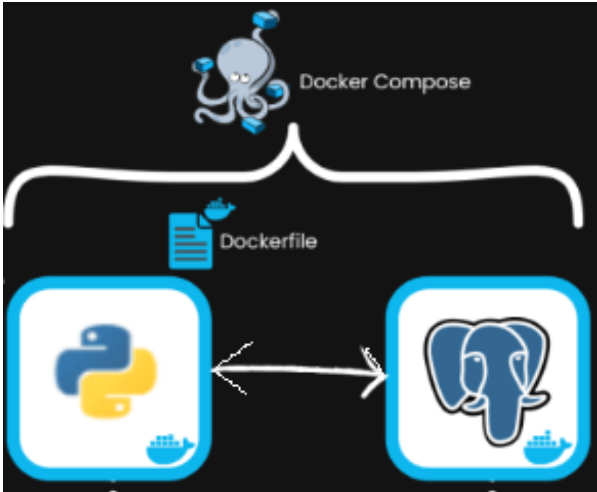
Assumptions

1. We are using PostgreSQL as a database.
2. By default, the program uses the sales_data_cleaned.csv which is the output after cleaning the provided sales_data.csv file.
3. In the third part of the task, the requirement states to read the data from the database, however if we take a look into the input parameter of the function, the function takes csv_file_path. As per my understanding, this function has to interact with the database only without needing the csv file. This is because we are already cleaning the data and storing it into the database in the previous steps.

Conditions

1. Round floating results to 2 decimal places
2. Docker compose file to run PostgreSQL
3. The Python function `process_data()` reads the data from database and returns a tuple containing:
 - a. A dictionary where the keys are transaction IDs, and values are the total profit for each transaction.
 - b. A dictionary where the keys are product IDs, and values are the total profit for each product.
 - c. The product IDs of the 2 top-selling products.

Workflow

Workflow diagram	Workflow steps
	<ol style="list-style-type: none">1 Build docker-compose file2 Create Network3 Create container db4 Create container k24_assignment-app-15 db connection6 Python scripts execution

Folder Structure

Main folder	scripts folder
<p>PC > New Volume (D:) > K24_Assignment</p> <p>Name ^</p> <ul style="list-style-type: none">scripts.envdata_analysis.ipynbdocker-composeDockerfileinitREADME.mdrequirementssales_datasales_data_cleanedstart	<ul style="list-style-type: none">appconstantstesttransfer_sales_data

File	File description
.env	File stores the credentials of PostgreSQL
data_analysis.ipynb	Jupyter notebook for cleaning the data (pandas library used)
docker-compose	File that defines and runs the containers for postgres and python
Dockerfile	File contains a set of instructions to run the python application
init.sql	As soon as the database is initialized, the database table called sales is created. This table definition is written as a SQL query in this file.
requirements.txt	List of Python libraries used
start.sh	Shell script containing commands to run python files

sales_data.csv	The provided CSV file
sales_data_cleaned.csv	Output cleaned csv file generated by data_analysis.ipynb
Python scripts	
transfer_sales_data.py	Python script to upload csv file to PostgreSQL
app.py	Python script to do data aggregation
constants.py	Python script to define the constants value
test.py	Unit test python script

How to reproduce the project

Step 1: Download and extract the project zip file

Step 2: Start the docker daemon

Step 3: Go to the project folder and open the command prompt

Step 4: Run the following command:

docker-compose up

```
db | 2024-04-14 13:38:52.349 UTC [60] LOG: database system was shut down at 2024-04-14 13:38:52 UTC
app-1 | 2024-04-14 13:38:52.376 UTC [1] LOG: database system is ready to accept connections
app-1 | Starting the execution of transfer_sales_data.py...
app-1 | Database connection is successful...
app-1 | Starting the execution of app.py...
app-1 | Database connection is successful...
app-1 | View created successfully...
app-1 | ({1: 45.0, 2: 25.0, 3: 20.0, 4: 25.0, 5: 60.0}, {101: 70.0, 102: 45.0, 103: 60.0}, [102, 103])
```

Step 5: Run the following command:

docker-compose down

```
[+] Running 3/3
  Container k24-data-engineering-challenge-app-1 Removed
  Container db Removed
  Network k24-data-engineering-challenge_default Removed
```

[Git repo link](#)

SQL Queries

- (1) Create a view with fields transaction_id, product_id, quantity, sale_price, purchase_price and one calculated field called profit

```
view_query = """create view cte as
(select *, quantity*(sale_price-purchase_price) as profit
from public.sales) """
```

- (2) Calculate the total profit for each transaction (this query uses cte view)**

```
trans_profit_query = """select transaction_id, round(cast(sum(profit) as numeric), 2)
from cte
group by transaction_id
having sum(profit) > 0.0
order by sum(profit) desc"""
```

- (3) Calculate the total profit for each product (this query uses cte view)**

```
prod_profit_query = """select product_id, round(cast(sum(profit) as numeric), 2)
from cte
group by product_id
having sum(profit) > 0.0
order by sum(profit) desc"""
```

- (4) Identify the top 2 selling products based on the ****total quantity******

```
top_prod_query = """select x.product_id from
(select product_id, sum(quantity), dense_rank() over(order by sum(quantity) desc) as r
from public.sales
group by product_id
order by sum(quantity) desc) x
where x.r < 3"""
```