GDDA612 – Data Transformation and Management

Assessment 2 - 18 August 2024

Table of Contents

1. Introduction

Objective

This assessment aims to evaluate proficiency in data manipulation techniques and exporting data into various formats, thereby enhancing business decision-making through insights and analysis-ready data.

Purpose

In this role, you aim to leverage the extensive dataset to drive business decisions effectively. To realize these objectives, you are tasked with manipulating and exporting datasets into diverse formats conducive to analysis and reporting. This process empowers decision-makers within the company with access to invaluable insights, fostering informed and strategic decision-making essential for supporting business expansion and development.

Task A – Data Preparation and Database Integration

A: Use Python or a similar tool, to load and analyze the structure of the selected dataset.

```
# Load the data file into a dataframe
df1 = pd.read_csv("C:/Users/patir/project_jupyter/sales_data/sales_data.csv")
df2 = pd.read_csv("C:/Users/patir/project_jupyter/customer_data/customer_data.csv")
```

Load the Dataset: You can load a dataset using `pandas` from various formats such as CSV, Excel, or SQL databases. For this example, let's assume you're working with a CSV file.

First data set: **Sales data**

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   invoice_no     99457 non-null  object
 1   customer_id    99457 non-null  object
 2   category       99457 non-null  object
 3   quantity       99457 non-null  int64
 4   price          99457 non-null  float64
 5   invoice_date   99457 non-null  object
 6   shopping_mall  99457 non-null  object
dtypes: float64(1), int64(1), object(5)
memory usage: 5.3+ MB
```

I can see all columns, types, and ranges.

```
#Dataset - Sales_data
df1
```

	invoice_no	customer_id	category	quantity	price	invoice_date	shopping_mall
0	I138884	C241288	Clothing	5	1500.40	05-08-2022	Kanyon
1	I317333	C111565	Shoes	3	1800.51	12-12-2021	Forum Istanbul
2	I127801	C266599	Clothing	1	300.08	09-11-2021	Metrocity
3	I173702	C988172	Shoes	5	3000.85	16-05-2021	Metropol AVM
4	I337046	C189076	Books	4	60.60	24-10-2021	Kanyon
...	...	...	...	...	...	...	...
99452	I219422	C441542	Souvenir	5	58.65	21-09-2022	Kanyon
99453	I325143	C569580	Food & Beverage	2	10.46	22-09-2021	Forum Istanbul
99454	I824010	C103292	Food & Beverage	2	10.46	28-03-2021	Metrocity
99455	I702964	C800631	Technology	4	4200.00	16-03-2021	Istinye Park
99456	I232867	C273973	Souvenir	3	35.19	15-10-2022	Mall of Istanbul

99457 rows × 7 columns

Second data set: **Customer data**

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   customer_id     99457 non-null  object
 1   gender          99457 non-null  object
 2   age             99338 non-null  float64
 3   payment_method  99457 non-null  object
dtypes: float64(1), object(3)
memory usage: 3.0+ MB
```

I can see all columns, types, and ranges.

```
#Dataset - Customer_data
df2
```

	customer_id	gender	age	payment_method
0	C241288	Female	28.0	Credit Card
1	C111565	Male	21.0	Debit Card
2	C266599	Male	20.0	Cash
3	C988172	Female	66.0	Credit Card
4	C189076	Female	53.0	Cash
...	...	...	...	...
99452	C441542	Female	45.0	Credit Card
99453	C569580	Male	27.0	Cash
99454	C103292	Male	63.0	Debit Card
99455	C800631	Male	56.0	Cash
99456	C273973	Female	36.0	Credit Card

99457 rows × 4 columns

```
#Merge between df1 and df2
df_merge = pd.merge(df1, df2, how='outer')
df_merge
```

	invoice_no	customer_id	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
0	I138884	C241288	Clothing	5	1500.40	05-08-2022	Kanyon	Female	28.0	Credit Card
1	I317333	C111565	Shoes	3	1800.51	12-12-2021	Forum Istanbul	Male	21.0	Debit Card
2	I127801	C266599	Clothing	1	300.08	09-11-2021	Metrocity	Male	20.0	Cash
3	I173702	C988172	Shoes	5	3000.85	16-05-2021	Metropol AVM	Female	66.0	Credit Card
4	I337046	C189076	Books	4	60.60	24-10-2021	Kanyon	Female	53.0	Cash
...	...	...	...	...	...	...	...	...	...	...
99452	I219422	C441542	Souvenir	5	58.65	21-09-2022	Kanyon	Female	45.0	Credit Card
99453	I325143	C569580	Food & Beverage	2	10.46	22-09-2021	Forum Istanbul	Male	27.0	Cash
99454	I824010	C103292	Food & Beverage	2	10.46	28-03-2021	Metrocity	Male	63.0	Debit Card
99455	I702964	C800631	Technology	4	4200.00	16-03-2021	Istinye Park	Male	56.0	Cash
99456	I232867	C273973	Souvenir	3	35.19	15-10-2022	Mall of Istanbul	Female	36.0	Credit Card

99457 rows × 10 columns

Merge both data sets.

```
#Duplicate columns
duplicate_rows = df_merge.duplicated()
print("Number of duplicate rows:", duplicate_rows.sum())
```

```
Number of duplicate rows: 0
```

After the merge, I checked if I had duplicate rows.

```
# Get basic information about the dataset
print("Dataset Info:")
print(df_merge.info())

# Display summary statistics of the dataset
print("\nSummary Statistics:")
print(df_merge.describe())

# Check for missing values
print("\nMissing Values:")
print(df_merge.isnull().sum())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99457 entries, 0 to 99456
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   invoice_no      99457 non-null  object
 1   customer_id     99457 non-null  object
 2   category        99457 non-null  object
 3   quantity        99457 non-null  int64
 4   price           99457 non-null  float64
 5   invoice_date    99457 non-null  object
 6   shopping_mall   99457 non-null  object
 7   gender          99457 non-null  object
 8   age             99338 non-null  float64
 9   payment_method  99457 non-null  object
dtypes: float64(2), int64(1), object(7)
memory usage: 7.6+ MB
None
```

Results after merge.
There are 9946 rows and 9 columns in the dataset.
The data type of all columns is object.
The columns in the datasets are:

- category', 'quantity', 'price', 'invoice_date', 'shopping_mall', 'gender', 'age', 'payment_method'

```
Summary Statistics:
            quantity         price           age
count  99457.000000  99457.000000  99338.000000
mean       3.003429    689.256321     43.425859
std        1.413025    941.184567     14.989400
min        1.000000      5.230000     18.000000
25%        2.000000     45.450000     30.000000
50%        3.000000    203.300000     43.000000
75%        4.000000   1200.320000     56.000000
max        5.000000   5250.000000     69.000000
```

Descriptive statistics are a collection of quantitative measures that summarize and describe the main characteristics of a dataset.

```
# look at the of missing points in the first ten columns
df_merge.isnull().sum()
```

```
category            0
quantity            0
price               0
invoice_date        0
shopping_mall       0
gender              0
age               119
payment_method      0
dtype: int64
```

```
#input value in missing value - Age
df_merge['age'].fillna(value=df_merge['age'].median(),inplace=True)
```

There are a few missing values in the dataset. Dealing with the missing values is one of the most important parts of the data wrangling process, we must deal with the missing values to get the correct insights from the data.

I put a median of ages in my scenario.

```
#Drop column irrelevant analysis
df_merge = df_merge.drop(columns="invoice_no")
df_merge
```

	customer_id	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
0	C241288	Clothing	5	1500.40	05-08-2022	Kanyon	Female	28.0	Credit Card
1	C111565	Shoes	3	1800.51	12-12-2021	Forum Istanbul	Male	21.0	Debit Card
2	C266599	Clothing	1	300.08	09-11-2021	Metrocity	Male	20.0	Cash
3	C988172	Shoes	5	3000.85	16-05-2021	Metropol AVM	Female	66.0	Credit Card
4	C189076	Books	4	60.60	24-10-2021	Kanyon	Female	53.0	Cash
...	...	...	...	...	...	...	...	...	...
99452	C441542	Souvenir	5	58.65	21-09-2022	Kanyon	Female	45.0	Credit Card
99453	C569580	Food & Beverage	2	10.46	22-09-2021	Forum Istanbul	Male	27.0	Cash
99454	C103292	Food & Beverage	2	10.46	28-03-2021	Metrocity	Male	63.0	Debit Card
99455	C800631	Technology	4	4200.00	16-03-2021	Istinye Park	Male	56.0	Cash
99456	C273973	Souvenir	3	35.19	15-10-2022	Mall of Istanbul	Female	36.0	Credit Card

99457 rows × 9 columns

I dropped the column **'invoice no'** because I don't need to work on my analysis.

B: Utilize data analytic systems knowledge, including machine learning packages and techniques, to transform the messy dataset into a tidy data format.

```
# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Transform Gender Male =1 and Female=0
df_merge['gender']= label_encoder.fit_transform(df_merge['gender'])
df_merge['gender'].unique()
```

```
array([0, 1])
```

```
df_merge.head()
```

	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
0	Clothing	5	1500.40	05-08-2022	Kanyon	0	28.0	Credit Card
1	Shoes	3	1800.51	12-12-2021	Forum Istanbul	1	21.0	Debit Card
2	Clothing	1	300.08	09-11-2021	Metrocity	1	20.0	Cash
3	Shoes	5	3000.85	16-05-2021	Metropol AVM	0	66.0	Credit Card
4	Books	4	60.60	24-10-2021	Kanyon	0	53.0	Cash

I changed the column 'gender' for Male = 0 and Female = 1

```
#initiate model with CleanLearning
cl = CleanLearning(model, seed=42)

clean_preds = cl.predict(X_test)
print(classification_report(y_test, clean_preds))
```

```
              precision    recall  f1-score   support

           2       0.71      0.71      0.71       155
           3       0.32      0.28      0.30       151
           4       0.00      0.00      0.00       159
           5       0.44      1.00      0.62       135

    accuracy                           0.48       600
   macro avg       0.37      0.50      0.41       600
weighted avg       0.36      0.48      0.40       600
```

Group 2 is performing decently with a good balance between precision and recall. The F1-score also reflects this.

Group 3 has a low precision and recall, indicating many misclassifications. The F1-score being low reflects this poor performance.

Group 4 the model failed to classify any instance correctly. Both precision and recall are 0, which is a significant issue.

Group 5 the model identified all instances (100% recall), but there were false positives, resulting in a lower precision.

Accuracy: the model correctly classifies 48% of all instances.
Macro Average: the macro average considers each class equally, showing that the model's performance is quite low on average.
Weighted Average: accounts for the support (number of instances) in each class, showing a slight improvement over the macro average, but still indicating overall poor performance.

```python
# Calculate mean sales by product category
grouped_df = df_merge.groupby('category')['price'].mean().idxmax()

# Print mean sales by product category
print(grouped_df)
```

```
Technology
```

```python
# Calculate the percentage of male and female
gender_counts = df_merge['gender'].value_counts(normalize=True) * 100
print("Percentage of Male Customers: {:.2f}%".format(gender_counts[0]))
print("Percentage of Female Customers: {:.2f}%".format(gender_counts[1]))
```

```
Percentage of Male Customers: 59.81%
Percentage of Female Customers: 40.19%
```

```python
from scipy import stats
import statsmodels.api as sm

# Separate price by gender Male=0 and Female=1
male_price = df_merge[df_merge['gender'] == 0]['price']
female_price = df_merge[df_merge['gender'] == 1]['price']

# T-test for difference in means
t_stat_gender, p_value_gender = stats.ttest_ind(male_price, female_price)
print(f"Gender and Price:\nT-statistic: {t_stat_gender}, P-value: {p_value_gender}")
```

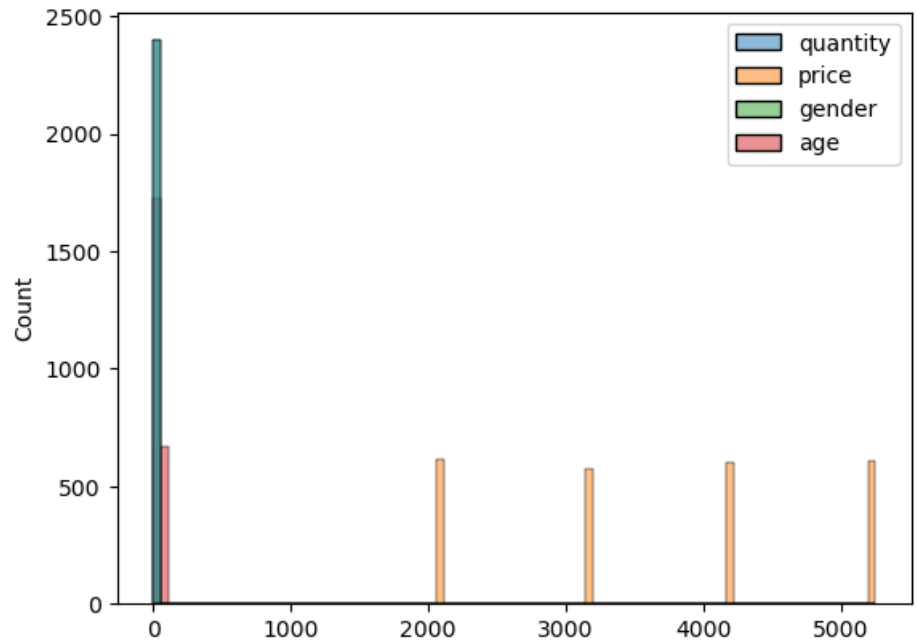
```
Gender and Price:
T-statistic: -0.45725199179880294, P-value: 0.6474909309116355
```

```python
#Filter Price, quantity, gender equal 'Male' and category equal 'Technology'
filtered_df = df_merge[(df_merge['price'] > 500.0) & (df_merge['quantity'] > 1) & (df_merge['gender'] == 0) & (df_merge['category
filtered_df
```

	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
53	Technology	4	4200.0	22-02-2022	Metrocity	0	43.0	Cash
69	Technology	5	5250.0	19-11-2021	Mall of Istanbul	0	44.0	Credit Card
90	Technology	2	2100.0	22-08-2022	Kanyon	0	43.0	Cash
122	Technology	2	2100.0	30-04-2022	Kanyon	0	64.0	Cash
139	Technology	2	2100.0	21-02-2021	Metropol AVM	0	32.0	Credit Card
...	...	...	...	...	...	...	...	...
99262	Technology	2	2100.0	04-06-2022	Kanyon	0	61.0	Credit Card
99362	Technology	2	2100.0	10-01-2021	Metrocity	0	61.0	Credit Card
99388	Technology	2	2100.0	06-12-2021	Metrocity	0	27.0	Cash
99398	Technology	2	2100.0	28-05-2021	Forum Istanbul	0	44.0	Debit Card
99424	Technology	3	3150.0	13-01-2021	Metrocity	0	34.0	Cash

```
sns.histplot(data=filtered_df)
```

```
<Axes: ylabel='Count'>
```



I checked the category 'Technology' because I got more information about who is buying, where shopping is, and age.

C: Display the initial rows of the resulting tidy dataset.

```
#Filter top 5 products
price_by_category_avg = df_merge.groupby('category')['price'].mean()
price_by_category_avg = price_by_category_avg.sort_values(ascending=False)
print("Top 5 product categories by average price:")
print(price_by_category_avg.head(5))
print("\n")
```

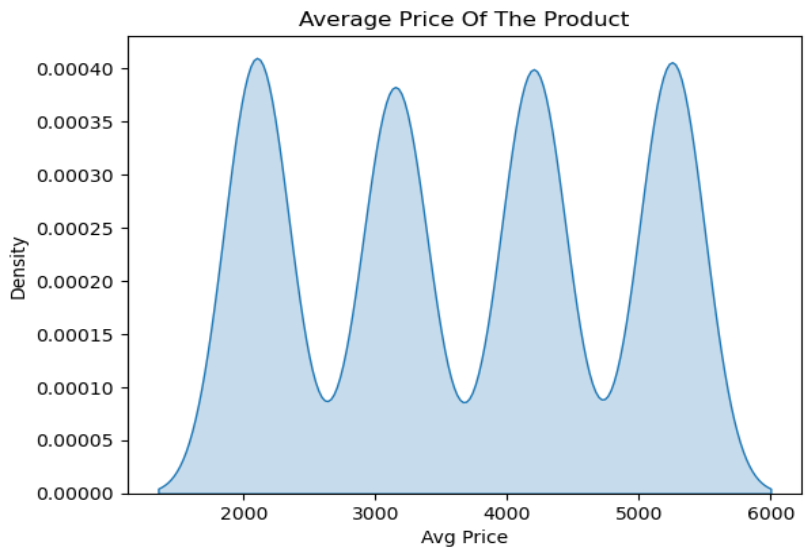
```
Top 5 product categories by average price:
category
Technology    3156.935548
Shoes         1807.388568
Clothing       901.084021
Cosmetics      122.448626
Toys           107.733185
Name: price, dtype: float64
```

There is a significant price disparity across the categories, with Technology products being considerably more expensive on average than Toys. The data suggests that Technology and Shoes are more premium markets, while Cosmetics and Toys cater to a broader range of consumers, including more budget-conscious segments.

```python
#Filter in category and price
category_price = df_merge.groupby('category')['price'].sum()
percent_price_by_category = ((category_price / category_price.sum()) * 100).sort_values(ascending=False)
percent_price_by_category = percent_price_by_category.apply(lambda x: "{:.2f}%".format(x))
print("Percentage of price by category:")
print(percent_price_by_category)
print("\n")
```

```
Percentage of price by category:
category
Clothing           45.33%
Shoes              26.46%
Technology         23.01%
Cosmetics           2.70%
Toys                1.59%
Food & Beverage     0.34%
Books               0.33%
Souvenir            0.25%
Name: price, dtype: object
```

```python
sns.kdeplot(filtered_df['price'],fill=True)
plt.title('Average Price Of The Product')
plt.xlabel('Avg Price')
plt.show()
```



D: Employ Python and similar tools to filter the tidy dataset based on specified criteria and retrieve the filtered dataset.

```python
#Filter Price, quantity, gender equal 'Male' and category equal 'Technology'
filtered_df = df_merge[(df_merge['price'] > 500.0) & (df_merge['quantity'] > 1) & (df_merge['gender'] == 0)
filtered_df
```

```
& (df_merge['gender'] == 0) & (df_merge['category'] == 'Technology')]
```

Filter

filtered_df = df_merge[(df_merge['price'] > 500.0) & (df_merge['quantity'] > 1) & (df_merge['gender'] == 0) & (df_merge['category'] == 'Technology')]

	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
53	Technology	4	4200.0	22-02-2022	Metrocity	0	43.0	Cash
69	Technology	5	5250.0	19-11-2021	Mall of Istanbul	0	44.0	Credit Card
90	Technology	2	2100.0	22-08-2022	Kanyon	0	43.0	Cash
122	Technology	2	2100.0	30-04-2022	Kanyon	0	64.0	Cash
139	Technology	2	2100.0	21-02-2021	Metropol AVM	0	32.0	Credit Card
...	...	...	...	...	...	...	...	...
99262	Technology	2	2100.0	04-06-2022	Kanyon	0	61.0	Credit Card
99362	Technology	2	2100.0	10-01-2021	Metrocity	0	61.0	Credit Card
99388	Technology	2	2100.0	06-12-2021	Metrocity	0	27.0	Cash
99398	Technology	2	2100.0	28-05-2021	Forum Istanbul	0	44.0	Debit Card
99424	Technology	3	3150.0	13-01-2021	Metrocity	0	34.0	Cash

2397 rows × 8 columns

Filter price high $500, quantity high one, and gender male and category Technology.

E:	Demonstrate proficiency in data analytics systems by establishing a connection with either an SQL or NoSQL database.

```
# Step2 Import MongoClient
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure
```

```
#Connect to MongoDB

#url = "mongodb+srv://patirangelsantos:aUdT7Jxk%40yUx2hu@cluster0.pnlxrbn.mongodb.net/"
url = "mongodb://localhost:27017/"

#client = MongoClient("mongodb+srv://patirangelsantos:aUdT7Jxk%40yUx2hu@cluster0.pnlxrbn.mongodb.net/")
client = MongoClient("mongodb://localhost:27017/")

try:
    # Create MongoClien objective
    db = client['gdda612_MongoDB']
    # Trying to connect to the server
    client.admin.command('ping')
    # Access collection
    collection = db['gdda612_MongoDB_collection_v3']
    print("successfu lconnection")
except ConnectionFailure:
    print("connection fail")
```

```
successfu lconnection
```

```
collection
```

```
Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'gdda612_MongoD
B'), 'gdda612_MongoDB_collection_v3')
```

I made connect with Mongo DB.

F: Import the dataset into a table or collection within the database.

```
df_merge.reset_index(inplace=True)
df_dict = df_merge.to_dict('records')
df_dict
```

```
[{'index': 0,
  'category': 'Clothing',
  'quantity': 5,
  'price': 1500.4,
  'invoice_date': '05-08-2022',
  'shopping_mall': 'Kanyon',
  'gender': 'Female',
  'age': 28.0,
  'payment_method': 'Credit Card'},
 {'index': 1,
  'category': 'Shoes',
  'quantity': 3,
  'price': 1800.51,
  'invoice_date': '12-12-2021',
  'shopping_mall': 'Forum Istanbul',
  'gender': 'Male',
  'age': 21.0,
  'payment_method': 'Debit Card'},
 {'index': 2,
```

```
collection.insert_many(df_dict)
```

```
InsertManyResult([ObjectId('6696fdf65952294976a000b2'), ObjectId('6696fdf65952294976a000b3'), ObjectId('6696fdf65952294976a000b4'), ObjectId('6696fdf65952294976a000b5'), ObjectId('6696fdf65952294976a000b6'), ObjectId('6696fdf65952294976a000b7'), ObjectId('6696fdf65952294976a000b8'), ObjectId('6696fdf65952294976a000b9'), ObjectId('6696fdf65952294976a000ba'), ObjectId('6696fdf65952294976a000bb'), ObjectId('6696fdf65952294976a000bc'), ObjectId('6696fdf65952294976a000bd'), ObjectId('6696fdf65952294976a000be'), ObjectId('6696fdf65952294976a000bf'), ObjectId('6696fdf65952294976a000c0'), ObjectId('6696fdf65952294976a000c1'), ObjectId('6696fdf65952294976a000c2'), ObjectId('6696fdf65952294976a000c3'), ObjectId('6696fdf65952294976a000c4'), ObjectId('6696fdf65952294976a000c5'), ObjectId('6696fdf65952294976a000c6'), ObjectId('6696fdf65952294976a000c7'), ObjectId('6696fdf65952294976a000c8'), ObjectId('6696fdf65952294976a000c9'), ObjectId('6696fdf65952294976a000ca'), ObjectId('6696fdf65952294976a000cb'), ObjectId('6696fdf65952294976a000cc'), ObjectId('6696fdf65952294976a000cd'), ObjectId('6696fdf65952294976a000ce'), ObjectId('6696fdf65952294976a000cf'), ObjectId('6696fdf65952294976a000d0'), ObjectId('6696fdf65952294976a000d1'), ObjectId('6696fdf65952294976a000d2'), ObjectId('6696fdf65952294976a000d3'), ObjectId('6696fdf65952294976a000d4'), ObjectId('6696fdf65952294976a000d5'), ObjectId('6696fdf65952294976a000d6'), ObjectId('6696fdf65952294976a000d7'), ObjectId('6696fdf65952294976a000d8'), ObjectId('6696fdf65952294976a000d9'), ObjectId('6696fdf65952294976a000da'), ObjectId('6696fdf65952294976a000db'), ObjectId('6696fdf65952294976a000dc'), ObjectId('6696fdf65952294976a000dd'), ObjectId('6696fdf65952294976a000de'), ObjectId('6696fdf65952294976a000df'), ObjectId('6696fdf65952294976a000e0'), ObjectId('6696fdf65952294976a000e1'), ObjectId('6696fdf65952294976a000e2'), ObjectId('6696fdf65952294976a000e3'), ObjectId('6696fdf65952294976a000e4'), ObjectId('6696fdf65952294976a000e5'), ObjectId('6696fdf65952294976a000e6'), ObjectId('6696fdf65952294976a000e7'), ObjectId('6696fdf65952294976a000e8'), ObjectId('6696fdf65952294976a000e9'), ObjectId('6696fdf65952294976a000ea'), ObjectId('6696fdf65952294976a000eb'), ObjectId('6696fdf65952294976a000ec'), ObjectId('6696fdf65952294976a000ed'), ObjectId('6696fdf65952294976a000ee'), ObjectId('6696fdf65952294976a000ef'), O
```

Save in Mongo DB my collection.

G: Retrieve and display records or documents from the table or collection.

```
for doc in collection.find():
    print(doc)
```

{'_id': ObjectId('66a839e7c68814fa3834cd48'), 'index': 0, 'category': 'Clothing', 'quantity': 5, 'price': 170.0, 'invoice_date': '05-08-2022', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 28.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd49'), 'index': 1, 'category': 'Shoes', 'quantity': 3, 'price': 1800.51, 'invoice_date': '12-12-2021', 'shopping_mall': 'Forum Istanbul', 'gender': 'Male', 'age': 21.0, 'payment_method': 'Debit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd4a'), 'index': 2, 'category': 'Clothing', 'quantity': 1, 'price': 300.08, 'invoice_date': '09-11-2021', 'shopping_mall': 'Metrocity', 'gender': 'Male', 'age': 20.0, 'payment_method': 'Cash'}
{'_id': ObjectId('66a839e7c68814fa3834cd4b'), 'index': 3, 'category': 'Shoes', 'quantity': 5, 'price': 3000.85, 'invoice_date': '16-05-2021', 'shopping_mall': 'Metropol AVM', 'gender': 'Female', 'age': 66.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd4c'), 'index': 4, 'category': 'Books', 'quantity': 4, 'price': 60.6, 'invoice_date': '24-10-2021', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 53.0, 'payment_method': 'Cash'}
{'_id': ObjectId('66a839e7c68814fa3834cd4d'), 'index': 5, 'category': 'Clothing', 'quantity': 5, 'price': 170.0, 'invoice_date': '24-05-2022', 'shopping_mall': 'Forum Istanbul', 'gender': 'Female', 'age': 28.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd4e'), 'index': 6, 'category': 'Cosmetics', 'quantity': 1, 'price': 170.0, 'invoice_date': '13-03-2022', 'shopping_mall': 'Istinye Park', 'gender': 'Female', 'age': 49.0, 'payment_method': 'Cash'}
{'_id': ObjectId('66a839e7c68814fa3834cd4f'), 'index': 7, 'category': 'Clothing', 'quantity': 2, 'price': 600.16, 'invoice_date': '13-01-2021', 'shopping_mall': 'Mall of Istanbul', 'gender': 'Female', 'age': 32.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd50'), 'index': 8, 'category': 'Clothing', 'quantity': 3, 'price': 900.24, 'invoice_date': '04-11-2021', 'shopping_mall': 'Metrocity', 'gender': 'Male', 'age': 69.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd51'), 'index': 9, 'category': 'Clothing', 'quantity': 2, 'price': 600.16, 'invoice_da

_id: ObjectId('66a839e7c68814fa3834cd48')
index : 0
category : "Clothing"
quantity : 5
price : 170
invoice_date : "05-08-2022"
shopping_mall : "Kanyon"
gender : "Female"
age : 28
payment_method : "Credit Card"

_id: ObjectId('66a839e7c68814fa3834cd49')
index : 1
category : "Shoes"
quantity : 3
price : 1800.51
invoice_date : "12-12-2021"
shopping_mall : "Forum Istanbul"
gender : "Male"
age : 21
payment_method : "Debit Card"

I showed in my list and Mongo DB.

H: Sort the records or documents based on a given condition.

```
# Display the sorted ascending documents
print("Sorted Documents by Credit Card (Ascending):")
for doc in sorted_documents:
    print(doc)
```

```
Sorted Documents by Credit Card (Ascending):
{'_id': ObjectId('66a839e7c68814fa3834cdb5'), 'index': 109, 'category': 'Cosmetics', 'quantity': 4, 'price': 162.64, 'invoice
_date': '12-12-2021', 'shopping_mall': 'Istinye Park', 'gender': 'Male', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cedd'), 'index': 405, 'category': 'Clothing', 'quantity': 1, 'price': 300.08, 'invoice_
date': '08-03-2022', 'shopping_mall': 'Viaport Outlet', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d149'), 'index': 1025, 'category': 'Books', 'quantity': 4, 'price': 60.6, 'invoice_dat
e': '19-01-2021', 'shopping_mall': 'Mall of Istanbul', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d24b'), 'index': 1283, 'category': 'Souvenir', 'quantity': 2, 'price': 23.46, 'invoice_
date': '28-01-2023', 'shopping_mall': 'Istinye Park', 'gender': 'Male', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d26f'), 'index': 1319, 'category': 'Clothing', 'quantity': 5, 'price': 1500.4, 'invoice
_date': '16-06-2021', 'shopping_mall': 'Metrocity', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d2a1'), 'index': 1369, 'category': 'Shoes', 'quantity': 3, 'price': 1800.51, 'invoice_d
ate': '03-04-2022', 'shopping_mall': 'Metrocity', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d2cc'), 'index': 1412, 'category': 'Books', 'quantity': 4, 'price': 60.6, 'invoice_dat
e': '02-01-2023', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d39e'), 'index': 1622, 'category': 'Clothing', 'quantity': 4, 'price': 1200.32, 'invoic
e_date': '03-04-2021', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 18.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834d41f'), 'index': 1751, 'category': 'Cosmetics', 'quantity': 1, 'price': 40.66, 'invoice
_date': '20-12-2021', 'shopping_mall': 'Zorlu Center', 'gender': 'Male', 'age': 18.0, 'payment_method': 'Credit Card'}
```

Sorted my collection with a filter credit card.

I: Count the number of records or documents present in the table or collection.

```python
#Count the documents
document_count = collection.count_documents({})
print(f"Number of documents in the collection: {document_count}")
```

```
Number of documents in the collection: 99457
```

All registered in my collection.

J: Perform grouping operations on records or documents within the table or collection.

```python
# Define the aggregation
group_by = [
    {
        '$group': {
            '_id': '$category',
            'totalQuantity': {'$sum': '$quantity'},
            'totalRevenue': {'$sum': {'$multiply': ['$price', '$quantity']}}
        }
    }
]
```

```
# Execute the aggregation
result = collection.aggregate(group_by)

# Print the result
for doc in result:
    print(doc)
```

```
{'_id': 'Souvenir', 'totalQuantity': 14871, 'totalRevenue': 635824.65}
{'_id': 'Clothing', 'totalQuantity': 103558, 'totalRevenue': 113996791.04}
{'_id': 'Books', 'totalQuantity': 14982, 'totalRevenue': 834552.9}
{'_id': 'Cosmetics', 'totalQuantity': 45465, 'totalRevenue': 6792862.899999999}
{'_id': 'Toys', 'totalQuantity': 30321, 'totalRevenue': 3980426.24}
{'_id': 'Food & Beverage', 'totalQuantity': 44277, 'totalRevenue': 849535.05}
{'_id': 'Shoes', 'totalQuantity': 30217, 'totalRevenue': 66553451.47}
{'_id': 'Technology', 'totalQuantity': 15021, 'totalRevenue': 57862350.0}
```

Group by categories, total quantity, and total revenue.

K: Execute update operations on records or documents within the table or collection.

```
def show():
    print(collection.count_documents({}))
    for entry in collection.find({"price": 170.00}):
        print(entry)
    print()
```

```
myquery = { "price": 1500.4 }
newvalues = { "$set": { "price": 170.00 } }

collection.update_one(myquery, newvalues)
show()
```

```
99457
{'_id': ObjectId('66a839e7c68814fa3834cd48'), 'index': 0, 'category': 'Clothing', 'quantity': 5, 'price': 170.0, 'invoice_dat
e': '05-08-2022', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 28.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd4d'), 'index': 5, 'category': 'Clothing', 'quantity': 5, 'price': 170.0, 'invoice_dat
e': '24-05-2022', 'shopping_mall': 'Forum Istanbul', 'gender': 'Female', 'age': 28.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd4e'), 'index': 6, 'category': 'Cosmetics', 'quantity': 1, 'price': 170.0, 'invoice_dat
e': '13-03-2022', 'shopping_mall': 'Istinye Park', 'gender': 'Female', 'age': 49.0, 'payment_method': 'Cash'}
{'_id': ObjectId('66a839e7c68814fa3834cd6c'), 'index': 36, 'category': 'Clothing', 'quantity': 5, 'price': 170.0, 'invoice_dat
e': '21-06-2022', 'shopping_mall': 'Kanyon', 'gender': 'Male', 'age': 43.0, 'payment_method': 'Credit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd88'), 'index': 64, 'category': 'Cosmetics', 'quantity': 4, 'price': 170.0, 'invoice_dat
e': '09-05-2022', 'shopping_mall': 'Metropol AVM', 'gender': 'Female', 'age': 29.0, 'payment_method': 'Debit Card'}
{'_id': ObjectId('66a839e7c68814fa3834cd8f'), 'index': 71, 'category': 'Cosmetics', 'quantity': 4, 'price': 170.0, 'invoice_dat
e': '04-09-2021', 'shopping_mall': 'Kanyon', 'gender': 'Female', 'age': 47.0, 'payment_method': 'Cash'}
```

Find all registered $1500.4 and update to $170.00

Task B- Data Expert, Migration and Backup

A: Export data from a specified table or collection in a database to a specified format file.

```
df =  pd.DataFrame(list(collection.find()))
df.head(n =5)
```

	_id	level_0	index	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
0	66b85db3a835a6f3c32666e4	0	0	Clothing	5	170.00	05-08-2022	Kanyon	0	28.0	Credit Card
1	66b85db3a835a6f3c32666e5	1	1	Shoes	3	1800.51	12-12-2021	Forum Istanbul	1	21.0	Debit Card
2	66b85db3a835a6f3c32666e6	2	2	Clothing	1	300.08	09-11-2021	Metrocity	1	20.0	Cash
3	66b85db3a835a6f3c32666e7	3	3	Shoes	5	3000.85	16-05-2021	Metropol AVM	0	66.0	Credit Card
4	66b85db3a835a6f3c32666e8	4	4	Books	4	60.60	24-10-2021	Kanyon	0	53.0	Cash

I showed my collection in the data frame.

```
# Save the DataFrame to a new CSV file
sales_customer_csv_file = 'Sales_and_Customer.csv'
df.to_csv(sales_customer_csv_file, index=False)

print(f"Data saved to {sales_customer_csv_file}")
```

```
Data saved to Sales_and_Customer.csv
```

Export to csv.

B: Establish a connection to the cloud storage service and upload the locally stored file to a specified cloud storage bucket using cloud service (GCP/Azure/AWS)

```
# how to import dataset from github
# https://github.com/patriciarangelsantos/GDDA612_A2_Sales_and_Customer_V2.git
# step 1 : clone the resource from github
!git clone https://github.com/patriciarangelsantos/GDDA612_A2_Sales_and_Customer_V2.git
```

```
Cloning into 'GDDA612_A2_Sales_and_Customer_V2'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 18 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (18/18), 4.03 MiB | 17.77 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

Connect my GitHub and I had my files.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Copy data from GitHub to Google Drive
!cp -r /content/GDDA612_A2/Sales_and_Customer.zip /content/drive/MyDrive/GDDA612_A2_Backup_Sales_Customer/
```

C: Schedule automated backups of a specified directory to a cloud storage service.

```
import pandas as pd
df = pd.read_csv('/content/GDDA612_A2_Sales_and_Customer_V2/Sales_and_Customer.zip')
df.head()
```

	_id	index	category	quantity	price	invoice_date	shopping_mall	gender	age	payment_method
0	66a839e7c68814fa3834cd48	0	Clothing	5	170.00	05-08-2022	Kanyon	Female	28.0	Credit Card
1	66a839e7c68814fa3834cd49	1	Shoes	3	1800.51	12-12-2021	Forum Istanbul	Male	21.0	Debit Card
2	66a839e7c68814fa3834cd4a	2	Clothing	1	300.08	09-11-2021	Metrocity	Male	20.0	Cash
3	66a839e7c68814fa3834cd4b	3	Shoes	5	3000.85	16-05-2021	Metropol AVM	Female	66.0	Credit Card
4	66a839e7c68814fa3834cd4c	4	Books	4	60.60	24-10-2021	Kanyon	Female	53.0	Cash

```
# backup_path  = '/content/drive/MyDrive/GDDA612_A2_Backup_Sales_Customer/'

backup_path  = '/content/drive/MyDrive/GDDA612_A2_Backup_Sales_Customer/'
```

```
import os
import pandas as pd
from google.colab import drive

def setup_and_load_data(github_repo, file_names, drive_folder):
    # Step 1: Clone the resource from GitHub
    repo_name = github_repo.split('/')[-1].replace('.git', '')
    clone_command = f"git clone {github_repo}"
    os.system(clone_command)

    # Step 2: Mount Google Drive
    drive.mount('/content/drive', force_remount=True)

    # Define base paths
    source_base_path = f'/content/{repo_name}/'
    backup_base_path = f'/content/drive/MyDrive/{drive_folder}/'

    # Check and create the target directory on Google Drive
    if not os.path.exists(backup_base_path):
        os.makedirs(backup_base_path)
        print(f"Directory '{backup_base_path}' created successfully.")
    else:
        print(f"Directory '{backup_base_path}' already exists.")

    # Copy data from GitHub to Google Drive and load the first file specified to DataFrame
    for file_name in file_names:
        resource_path = os.path.join(source_base_path, file_name)
        backup_path = os.path.join(backup_base_path, file_name)
        os.system(f"cp {resource_path} {backup_path}")
```

```python
    # Copy data from GitHub to Google Drive and load the first file specified to DataFrame
    for file_name in file_names:
        resource_path = os.path.join(source_base_path, file_name)
        backup_path = os.path.join(backup_base_path, file_name)
        os.system(f"cp {resource_path} {backup_path}")

    # Load the first file into a DataFrame as an example
    first_file_path = os.path.join(backup_base_path, file_names[0])
    df = pd.read_csv(first_file_path)
    return df

# Usage of the function
github_repo = 'https://github.com/patriciarangelsantos/GDDA612_A2.git'
file_names = ['Sales_and_Customer.zip']  # List of filenames to be copied and backed up
drive_folder = 'GDDA612_A2'

df = setup_and_load_data(github_repo, file_names, drive_folder)
print(df.head())  # Display the first few rows of the DataFrame from the first file
```

D: Include error handling to handle backup failures gracefully, such as connectivity issues or file upload errors.

```
Mounted at /content/drive
Directory '/content/drive/MyDrive/GDDA612_A2/' already exists.
                        _id  index  category  quantity    price invoice_date  \
0  66a839e7c68814fa3834cd48      0  Clothing         5   170.00   05-08-2022
1  66a839e7c68814fa3834cd49      1     Shoes         3  1800.51   12-12-2021
2  66a839e7c68814fa3834cd4a      2  Clothing         1   300.08   09-11-2021
3  66a839e7c68814fa3834cd4b      3     Shoes         5  3000.85   16-05-2021
4  66a839e7c68814fa3834cd4c      4     Books         4    60.60   24-10-2021

    shopping_mall  gender   age payment_method
0          Kanyon  Female  28.0    Credit Card
1  Forum Istanbul    Male  21.0     Debit Card
2       Metrocity    Male  20.0           Cash
3    Metropol AVM  Female  66.0    Credit Card
4          Kanyon  Female  53.0           Cash
```

I show my backup no google drive.

Reference

GITHUB - https://github.com/patriciarangelsantos/GDDA612_A2_Sales_and_Customer_V2