Li Tao

850001413

GDDA612

Assessment2\_Project

Table of Contents

[Introduction: 2](#_Toc174644598)

[Task A – Data Preparation and Database Integration 2](#_Toc174644599)

[a) Load and inspect the dataset structure using Python 2](#_Toc174644600)

[b) Apply techniques to clean and organize the dataset into a tidy format. 5](#_Toc174644601)

[c) Display the first few rows of the organized dataset 7](#_Toc174644602)

[d) Filter the tidy dataset based on specified criteria and retrieve the filtered dataset 8](#_Toc174644603)

[e) Connect to an SQL or NoSQL database 10](#_Toc174644604)

[f) Import the dataset into a database table or collection 11](#_Toc174644605)

[g) Retrieve and display records from the table or collection 12](#_Toc174644606)

[h) Sort the records according to a specific condition 13](#_Toc174644607)

[i) Count the total number of records in the table or collection 13](#_Toc174644608)

[j) Group records in the table or collection based on a specific field 14](#_Toc174644609)

[k) Update records within the table or collection 15](#_Toc174644610)

[Task B – Data Export, Migration and Backup 16](#_Toc174644611)

[a) Export data from a database table to a specified file format 16](#_Toc174644612)

[b) Upload the exported file to a cloud storage service 17](#_Toc174644613)

[c) Set up automated backups for a specified directory to cloud storage 20](#_Toc174644614)

[d) Implement error handling to address backup failures, such as connectivity or upload issues 23](#_Toc174644615)

# Introduction:

In this project, I used the dataset titled ‘Adidas US Sales Datasets’, downloaded from Kaggle to perform various tasks by utilizing comprehensive data management techniques. The dataset comprises 9,648 rows and 14 columns, offering a comprehensive view of Adidas’ sales performance across different criteria. It spans dates from 2020 to 2021 and includes details on shopping locations, product types, price ranges, and operating profits.

For Task A, which involved data preparation and database integration, I utilized Python along with libraries such as Pandas and SQL connector libraries to perform efficient analysis and querying on ‘Adidas sales’ such as sorting, grouping, and updating operations.

For Task B, which focused on data export, migration, and backup, I implemented Google Drive functionality to establish a connection and performed automated backups in Google Colab to upload the local file to cloud storage while handling errors and failures efficiently, protecting the data from potential loss.

All in all, I focused on data handling and manipulation by processing data from diverse sources, such as online datasets, transforming it into appropriate formats within a Python environment, managing it through data stores for effective querying and exporting, and ensuring it was automatically stored for future analysis.

# Task A – Data Preparation and Database Integration

## a) Load and inspect the dataset structure using Python

I loaded the dataset and displayed several rows, basic information, data shape, descriptive statistics, and columns of the dataset.

Please find the screenshot below:

|  |
| --- |
| I imported relevant libraries.  A close-up of a computer code  Description automatically generated |
| I used ‘read\_excel()’ method in pandas to read the dataset into a DataFrame. |
| ‘df.head()’: display the first 5 rows of the dataset to understand the basic structure and content of the data.  A screenshot of a website  Description automatically generated |
| ‘df.shape’: quickly understand the size of the dataset.  A close-up of a number  Description automatically generated |
| ‘df.info()’: display the basic information, including the data type and number of non-null values for each column.  A screenshot of a computer  Description automatically generated |
| A screenshot of a screenshot of a data  Description automatically generated‘df.describe()’: display descriptive statistics about numerical columns, including the mean, standard deviation, minimum, and quartile. |
| ‘df.columns’: display all columns’ names in a data set.  A screenshot of a computer  Description automatically generated |
| ‘df.isnull().sum()’: count the number of missing values in each column.  A screenshot of a computer screen  Description automatically generated |
| ‘df.duplicated().sum()’: check how many rows are duplicated in the dataset.  A close-up of a computer screen  Description automatically generated |

## b) Apply techniques to clean and organize the dataset into a tidy format.

I checked the missing values in the previous task and found that all the values in the ‘Unnamed’ column were null, so I decided to drop the column. Additionally, I converted data types and applied various machine-learning methodologies in this task.

|  |
| --- |
| 1. Handle missing value   Drop the column ‘Unnamed’, in this column all the values are null.  A close-up of a white background  Description automatically generated |
| 1. Extract and create new columns ‘Month’ and ‘Year’ from the ‘Invoice Date’ column.   I used the ‘dt’ accessor, available on the pandas' datetime series.  A screenshot of a screen  Description automatically generated |
| 1. Convert categorical variables into binary vectors   To handle categorical variables, such as ‘Retailer’, ‘Region’, ‘State’, ‘City’, ‘Product’, and ‘Sales Method’, I used the one-hot encoding method in pandas. This approach is useful for machine learning algorithms that require numerical input.  A screenshot of a computer  Description automatically generated |
| Then I converted the encoded columns’ data type to integer.  A screenshot of a computer  Description automatically generated |
| 1. Use ‘Z-score’ method to find outliers   I used z-score technology to find out the outliers in ‘Unit Sold’ column. I set up the threshold be 3. The more units sold; the higher sales would be. In a retail store, it is possible that during a special period, the sales amount increased sharp, so I decided to keep these outliers.  A screenshot of a computer  Description automatically generated |
| 1. Perform standardization on encoded dataset   I performed the standardization on the encoded dataset excluding ‘Invoice Date’, ‘Month’, and ‘Year’ these datetime columns. Standardization helps to scale different numerical columns to the same scale, which can be beneficial for subsequent modeling.  A screenshot of a computer  Description automatically generated |

## c) Display the first few rows of the organized dataset

After performing encoding and standardization, the dataset is ready for the machine learning model. In the following tasks, I decided to use the dataset before applying machine learning package to clean and organize, because by loading the dataset into a table within the database in the following tasks, it becomes easier to understand the values and data types of each column.

|  |
| --- |
| I used the ‘df\_encoded.head()’ function to display a few rows of the organized dataset.  Please see the screenshot below:  A screenshot of a computer screen  Description automatically generated |
| I displayed a few rows of the dataset before performing machine learning package. It is clean and easy understanding.  Please see the screenshot below:  A screenshot of a website  Description automatically generated |

## d) Filter the tidy dataset based on specified criteria and retrieve the filtered dataset

I used the dataset before performing the machine learning package in the following tasks. I sorted the DataFrame by the ‘Operating Margin’ column in ascending order, so that I can locate the lowest and highest operating margin easily. The higher the operating margin is considered better, indicating a company is more efficient in converting sales into actual profit from its operations. Then I filtered the DataFrame based on the condition that the ‘Operating Margin’ column is greater than 0.40. After applying this method, I can identify the products and retailers with higher operating margins, allowing to focus on entries with higher-performance products.

|  |
| --- |
| The lowest and highest values for the operating margin are 0.1 and 0.8, respectively.  Please see the screenshot below:  A screenshot of a computer  Description automatically generated |

|  |
| --- |
| I retrieved a new dataset, ‘df\_filtered’, in which the operating margin is over 0.4.  Please find the screenshot below: |
| A screenshot of a computer screen  Description automatically generated |

## e) Connect to an SQL or NoSQL database

In this task, I would establish a connection with the MySQL database.

|  |
| --- |
| I Imported the relevant libraries.  A white rectangular object with black text  Description automatically generated with medium confidence |
| I established a connection with MySQL using the ‘SQLAlchemy’ and ‘mysql-connector-python’. The ‘create\_engine()’ function created a SQLAlchemy engine for connecting to the database by passing my database credentials and arguments. The ‘with engine.connect() as connection’ function established a connection to the database, and ‘connection. execute()’ executed a query to verify the connection is successful.  I printed the outcome that the connection was successfully established.  A screenshot of a computer program  Description automatically generated |

## f) Import the dataset into a database table or collection

I executed a SQL command to create the database if it did not exist and updated the engine connecting specifically to the newly created database.

|  |
| --- |
| I imported the dataset into a table named ‘Adidas’ in the newly created database. If the table named ‘Adidas’ already exists, it will be replaced.  And I printed a confirmation message indicating the dataset has been successfully imported into the table.  A screenshot of a computer code  Description automatically generated |
| To check and confirm the result, I refreshed the schemas in MySQL and found that the dataset was successfully inserted into the table ‘Adidas’.  Please find the screenshot below:  A screenshot of a computer  Description automatically generated |

## g) Retrieve and display records from the table or collection

I executed the SQL query ‘SELECT \* FROM Adidas LIMIT 10’ to retrieve the first 10 rows from the ‘Adidas’ table iterated over the result and printed each row.

This approach is useful for quickly inspecting the structure and contents of the Adidas table without loading the entire dataset. Additionally, it allows me to immediately check for any anomalies or connection errors, ensuring that the data retrieval process is functioning correctly.

|  |
| --- |
| Each row is a tuple representing a record from the table.  Please find the screenshot below: |
| A close-up of a screen  Description automatically generated |

## h) Sort the records according to a specific condition

I executed the SQL query to retrieve and sort by the ‘Price per Unit’ column in descending order, showing the top 10 records.

This method effectively identifies the most expensive products, and in this case, it reveals that women’s apparel is selling at a high price. This could indicate a strong market demand or premium positioning for women’s products. Furthermore, identifying the most expensive products can help in understanding their contribution to overall revenue.

|  |
| --- |
| The ‘pd.DataFrame()’ function created a DataFrame from the fetched data using column names. The ‘result.fetchall()’ function got the rows from the result set as a list of tuples.  Please find the screenshot below:  A screenshot of a computer screen  Description automatically generated |

## i) Count the total number of records in the table or collection

I executed the SQL query ‘SELECT COUNT (\*) FROM Adidas’ to count the number of records in the table.

Knowing the total number of records is crucial for understanding the dataset’s scope, as it ensures the data is complete and that no records are missing. In addition, this total count serves as a baseline for calculating other metrics, such as percentages and averages.

|  |
| --- |
| The ‘scalar()’ function retrieved the single value from the query result, which is the count of records, showing in number 9648.  A close-up of a computer code  Description automatically generated |

## j) Group records in the table or collection based on a specific field

I established the connection to the database and executed the query to group records by ‘City’ and calculated the total units sold and total operating profit for each city. I used the ‘fetchall()’ function to retrieve all rows from the result set. I iterated over the result set and oriented each row.

Grouping records by ‘City’ and calculating total units sold and total operating profit provides valuable insights into how sales and profitability vary across different cities. This information can help in making data-driven decisions such as resource allocation and inventory management. Additionally, analysing these metrics can reveal trends and patterns, such as identifying cities that consistently show high profitability.

|  |
| --- |
| Each row is tuple containing the city name and the aggregated values.  Please find the screenshot below:  A screen shot of a computer code  Description automatically generated |

|  |
| --- |
| Please find the output below: |
| A screenshot of a computer code  Description automatically generated |

## k) Update records within the table or collection

I executed the update query using ‘connection.execute()’ and captured the result object.

I committed the transaction to apply the changes. The ‘result.rowcount’ attribute returned the number of rows affected by the update query, and this was used to verify the update operation was successful and how many records were updated.

This approach not only confirmed the successful execution of the query but also helped in verifying data integrity and efficiency.

|  |
| --- |
| The number of rows affected is 52.  Please find the screenshot below:  A screenshot of a computer program  Description automatically generated |

# Task B – Data Export, Migration and Backup

## a) Export data from a database table to a specified file format

I exported the data after it was updated in the MySQL database and identified the table ‘Adidas’ from which to export data.

Firstly, I defined the query to retrieve the updated data, ensuring that all columns and rows would be selected from the ‘Adidas’ table. Secondly, I established a connection to the MySQL database using the ‘with the engine.connect() as connection’ context manager. The ‘pd.read\_sql’ function from the Pandas library was used to execute the SQL query and convert the results into a DataFrame. Thirdly, I exported the DataFrame ‘df\_after’ to a JSON file named ‘output.json’. Finally, I printed a message to confirm that the updated dataset has been successfully exported to the specified JSON file.

|  |
| --- |
| I exported the updated data from the table ‘Adidas’ in MySQL to a JSON format file.  A screenshot of a computer code  Description automatically generated |
| After the implementation, I can confirm that the file is saved in my local folder.  A screenshot of a computer  Description automatically generated |

## b) Upload the exported file to a cloud storage service

I used Google Drive functionality to establish a connection and upload a local file by performing operations in Google Colab in conjunction with Google Drive.

First, I used ‘drive.mount(‘/content/drive’) to mount Google Drive in Colab. After installing and importing the relevant libraries, I used the ‘files.upload()’ method to upload a file to the temporary folder in Colab. Then I used ‘os.getcwd()’ function to get the current working directory. After that, I used ‘os.path.join()’ to construct the full paths for the source and target files once I created the backup path. Finally, I used ‘shutil.copy()’ method to copy the file from the source path to the target path.

With these steps, I can create a connection with Google Drive, upload a file to the Drive, and ensure it is properly backed up to the specified directory.

|  |
| --- |
| I mounted my Google Drive in Google Colab, so that I can access my files in Colab.  I installed the relevant ‘schedule’ package and imported required libraries.  A close-up of a computer code  Description automatically generated  A screenshot of a computer program  Description automatically generated |
| I uploaded the locally stored file to Google Drive.  The ‘files.upload()’ prompted a file selection dialog to upload local files.  A screenshot of a computer  Description automatically generated |
| The ‘os.getcwd()’ retrieved the current working directory path and printed the file which was under ‘content’ folder, which was a temporary path to store the file in Google Drive. I specified the ‘terget\_dir\_root’ directory, generated the unique ‘bucket\_name’ by using the current time as a unique suffix, and constructed the ‘backup\_path’ by joining the target root and the bucket name. I created the ‘backup\_path’ where the backup would be stored and printed the path.  A screenshot of a computer program  Description automatically generated  A screenshot of a computer  Description automatically generated |
| I used ‘input()’ function to prompt checking for existence of the file and passed the file name ‘output.json’ input to ‘file\_name’.  A screen shot of a computer code  Description automatically generated |
| I constructed the full paths of connection between the ‘source\_file’ which is the ‘current\_path’ join the ‘file\_name’ and the ‘target\_file’ which is the ‘backup\_path’ join the ‘file\_name’.  A screenshot of a computer  Description automatically generated |
| I used ‘shutil.copy()’ function to copy the file from the source folder to the target directory if it exists.  A white background with blue text  Description automatically generated |
| I can confirm that the file has been updated under the folder ‘612A2\_BACKUP\_20240727051423’. A screenshot of a computer  Description automatically generated |

## c) Set up automated backups for a specified directory to cloud storage

I structured two functions, one is ‘backup\_files’ and another is ‘schedule\_backup”. Here I would break down the steps and implementation of each function.

The ‘backup\_files’ function: this function handles the actual backup process.

I used the ‘os.makedirs()’ function to create the backup path directory if it does not exist. I iterated over and constructed the full path for each item in the source directory. I checked if the item was a file, and if so, copied it to the backup path, otherwise, skip copying. Then I copied the file from the source path to the target path by printing several messages, indicating the backup path directory, the number of items found in the source directory, and items copied to the backup path.

The ‘schedule\_backup’ function: this function manages the timing of the backup.

I created an infinite loop that will run until the backup condition is met by using the ‘while True’ measure. With the ‘datetime.now()’ function, I got the current date and time. The function will check if the current matches the target time and then create a unique suffix based on the current time for naming the backup directory. Then I called the ‘backup\_files’ function to perform the back, printing a message indicating the backup completed and the task finished. The ‘break’ indicates exist the infinite loop after the backup is completed. And ‘time.sleep(1)’ pauses the execution for 1 second before checking the time again.

In Parameter setup:

I defined the source directory containing files to be backed up and the target directory where files will be backed up and stored. I set up the time when the backup operation will start.

Finally, I called the ‘schedule\_backup’ function, which has the specified parameters to execute the backup.

All in all, the ‘backup\_files’ function handles the copying of files, while the ‘schedule\_backup’ function manages the timing of the backup operation. This setup ensures that files are backed up at the exact time and after the operation task is completed, it should exist the scheduler.

|  |
| --- |
| Please find the screenshot below:  A screenshot of a computer program  Description automatically generated |
| Here I passed the parameters defining the path and root directory in Google Drive and set up the time when the backup should start.  A computer code with red text  Description automatically generated |
| The output indicated that two files was copied from a source directory to a new backup directory created and named based on the current date and time, ‘BACKUP\_20240727065359’. The output provided a count and listed 7 items were founded in the source directory. With format checking some skipping methods operated as these items were not files. After the task was finished, a complete message was printed.  A white background with black text  Description automatically generated |
| I can confirm that the file has been updated under the folder ‘BACKUP\_202407270065359’. A screenshot of a computer  Description automatically generated |

## d) Implement error handling to address backup failures, such as connectivity or upload issues

To handle potential errors in the backup process gracefully, including issues with directory creation, file access, and other unforeseen exceptions, the focus would be on robust error handling in both the ‘backup\_files’ and the ‘schedule\_backup’ functions.

The ‘backup\_files’ function:

* ‘PermissionError’: this exception catches errors related to insufficient permissions. For example, if the script does not have the necessary rights to read a file, this error will be triggered.
* ‘FileNotFoundError’: this exception handles cases where the source item cannot be found. If the script tries to access a file that does not exist, this error will be triggered.
* General exception: this catches any other unexpected errors that may arise during the backup process.
* Directory creation issues: If the backup directory creation fails, the ‘except’ block will catch the exception and print an error message, preventing the script from crashing and informing the issue with a clear message.

The ‘schedule\_backup’ function:

* Backup failure issue: If an error occurs during the backup process, the ‘except’ block will catch any exceptions and print an error message, allowing for troubleshooting and ensuring the backup task does not terminate silently.

These approaches to handling errors ensure that the backup process is resilient and robust, providing clear feedback in case of problems and preventing the script from stopping unexpectedly.

|  |
| --- |
| Please find the screenshot below:  A screenshot of a computer program  Description automatically generated |
| A screenshot of a computer program  Description automatically generated |

|  |
| --- |
| Please find the output below:  A screen shot of a computer code  Description automatically generated |
| I can confirm that the file has been successfully stored in Google Drive folder. |

For the reference of the code, please find the GitHub link below:

<https://github.com/YiliaTao0122/612Assessment2>