TECHNICAL REPORT

TEAM NAME: SYNTAX GIRLS	
TEAM NAME	SITI AIN ATHIQAH BINTI SAHRUN HANA SYAKIRAH BINTI HASSAN KHAIRULLAH MUNA 'ILYANI BINTI MUA'AD
COURSE	BACHELOR OF COMPUTER SCIENCE WITH HONOURS
ORGANIZATION	UNIVERSITI UTARA MALAYSIA (UUM)
SUBMISSION DATE	20th OCTOBER 2025

TABLE OF CONTENTS

1.0 INTRODUCTION

This report presents the data preprocessing, transformation, and visualization steps taken to build a Power BI dashboard for analyzing the company's sales, purchases, inventory, and employee performance. The goal is to provide clear business insights using data from multiple relational tables.

2.0 DATA UNDERSTANDING

Based on the given dataset, it contains 8 sheets, which represent the employee, customer, supplier, product, order, order_product, sale and sale_product data.

Table/Sheet	Description	Key Fields
employee	Employee information including role, gender, and employment duration.	● employee_id (categorical) ● name (categorical) ● role (categorical) ● date_joined (date) ● date_left (categorical) ● rage (categorical) ● gender (categorical) ● ic_number (numerical)
customer	Customer registration details.	● customer_id (categorical) ● name (categorical) ● gender (categorical) ● race (categorical) ● ic_number (numerical) ● date_registered (date)
supplier	Supplier contact and location info.	● supplier_id (categorical) ● name (categorical) ● contact_person (categorical) ● contact_number (categorical) ● address_1 (categorical) ● address_2 (categorical) ● postcode (numerical) ● state (categorical) ● district (categorical)
product	Master list of products.	● product_id (categorical) ● product_name (categorical) ● purchase_price (numerical) ● selling_price (numerical) ● balance (numerical)
order	Purchase transactions from suppliers.	● order_id (categorical) ● date (date)

		● supplier_id (categorical) ● amount (numerical)
order_product	Quantity of products per order.	● order_id (categorical) ● product_id (categorical) ● quantity (numerical)
sale	Sales transactions.	● date (date) ● customer_id (categorical) ● payment_method (categorical) ● employee_id (categorical) ● amount (numerical)
sale_product	Quantity of products per sale.	● sale_id (categorical) ● product_id (categorical) ● quantity (numerical)

3.0 METHOD AND STEPS

3.1 DATA LOADING & CHECKING

Before importing the dataset into Power BI, we performed an initial data quality checking using Python (Jupyter Notebook) to identify missing values, duplicates and data type issues across all tables. This step to ensure that the data was well-understood before cleaning and modelling in Power BI.

Step 1: Data Import

All sheets from the original file international_business_datathon_case_study.xlsx were imported using the pandas library in Python. The original file name has been changed into originalDataset.xlsx to make it short and simple to be checked inside the Jupyter Notebook.

Step 2: Data Quality Check & Summary

```
import pandas as pd

# Load Excel file
file_path = "originalDataset.xlsx"

# Read all sheets
all_sheets = pd.read_excel(file_path, sheet_name=None)

print("Checking missing values in each sheet:\n")

# Loop through each sheet
for sheet_name, df in all_sheets.items():
    # Count missing values in each column
    missing = df.isnull().sum()
    total_missing = missing.sum()

    print(f"{sheet_name} sheet:")
    if total_missing == 0:
        print("No missing values found.\n")
    else:
        # Show columns that have missing values
        for col, count in missing.items():
            if count > 0:
                print(f"{col}: {count} missing value(s)")
        print("")  # Blank line for spacing
```

Output:

```
Checking missing values in each sheet:

employee sheet:
date_left: 10 missing value(s)

customer sheet:
No missing values found.

supplier sheet:
address_2: 1 missing value(s)

product sheet:
No missing values found.

order sheet:
No missing values found.

order_product sheet:
No missing values found.

sale sheet:
customer_id: 14210 missing value(s)

sale_product sheet:
No missing values found.
```

```
import pandas as pd

# Load Excel file
file_path = "originalDataset.xlsx"

# Read all sheets
all_sheets = pd.read_excel(file_path, sheet_name=None)

print("Checking duplicate IDs in each sheet:\n")

# Function to count duplicates for single ID column
def count_duplicates(df, id_column, sheet_name):
    if id_column not in df.columns:
        print(f"{sheet_name}: Column '{id_column}' not found.")
        return
    duplicate_count = df.duplicated(subset=[id_column]).sum()
    print(f"{sheet_name}: {duplicate_count} duplicate {id_column}(s)")

# Function for composite key duplicates
def count_composite_duplicates(df, cols, sheet_name):
    duplicate_count = df.duplicated(subset=cols).sum()
    print(f"{sheet_name}: {duplicate_count} duplicate combination(s) for {cols}")

# Check duplicates for each ID
count_duplicates(all_sheets['employee'], 'employee_id', 'employee')
count_duplicates(all_sheets['customer'], 'customer_id', 'customer')
count_duplicates(all_sheets['supplier'], 'supplier_id', 'supplier')
count_duplicates(all_sheets['product'], 'product_id', 'product')
count_duplicates(all_sheets['order'], 'order_id', 'order')
count_duplicates(all_sheets['sale'], 'sale_id', 'sale')

# Check for composite key duplicates
count_composite_duplicates(all_sheets['order_product'], ['order_id',
'product_id'], 'order_product')
count_composite_duplicates(all_sheets['sale_product'], ['sale_id', 'product_id'],
'sale_product')
```

Output:

```
Checking duplicate IDs in each sheet:

employee: 0 duplicate employee_id(s)
customer: 0 duplicate customer_id(s)
supplier: 0 duplicate supplier_id(s)
product: 0 duplicate product_id(s)
order: 0 duplicate order_id(s)
sale: 0 duplicate sale_id(s)
order_product: 0 duplicate combination(s) for ['order_id', 'product_id']
sale_product: 202 duplicate combination(s) for ['sale_id', 'product_id']
```

```
import pandas as pd
# Load all sheets
file_path = "originalDataset.xlsx"

tables = {
    "employee": pd.read_excel(file_path, sheet_name="employee"),
    "customer": pd.read_excel(file_path, sheet_name="customer"),
    "supplier": pd.read_excel(file_path, sheet_name="supplier"),
    "product": pd.read_excel(file_path, sheet_name="product"),
    "order": pd.read_excel(file_path, sheet_name="order"),
    "order_product": pd.read_excel(file_path, sheet_name="order_product"),
    "sale": pd.read_excel(file_path, sheet_name="sale"),
    "sale_product": pd.read_excel(file_path, sheet_name="sale_product")
}
# Create Summary Table
summary = []

for name, df in tables.items():
    summary.append({
        "Table": name,
        "Rows": df.shape[0],
        "Columns": df.shape[1],
        "Total Missing Values": df.isnull().sum().sum(),
        "Duplicate Rows": df.duplicated().sum(),
        "Numeric Columns": df.select_dtypes(include=["number"]).shape[1],
        "Categorical Columns": df.select_dtypes(exclude=["number"]).shape[1]
    })

summary_df = pd.DataFrame(summary)

# Display Summary
print("\n DATA QUALITY SUMMARY")
display(summary_df)
```

Output:

DATA QUALITY SUMMARY

	Table	Rows	Columns	Total Missing Values	Duplicate Rows	Numeric Columns	Categorical Columns
0	employee	17	8	10	0	1	7
1	customer	2604	6	0	0	1	5
2	supplier	3	9	1	0	1	8
3	product	30	5	0	0	3	2
4	order	35	4	0	0	1	3
5	order_product	95	3	0	0	1	2
6	sale	14700	6	14210	0	1	5
7	sale_product	23117	3	0	107	1	2

Based on the data quality summary above, we identified missing values and duplicate rows across multiple tables using Python (Jupyter Notebook).

Missing values: The dataset contains missing values, mainly within the employee and sale tables. However, instead of removing or imputing these missing entries, we decided to remain the original dataset to preserve data integrity. To handle this issue effectively, we created new derived columns to represent missing or null entries. For example, in the employee sheet, we add a new column into “Available” or “Resigned” based on a missing value in date_left. These new columns will be added in Power BI to ensure the reports remain complete without discarding important transactional records.

Duplicate rows: The duplicate row was detected in the sale_product tables. These duplicates are not data quality errors, but are expected both tables act as a bridge relationship between their parent tables. Each sale or order can contain multiple product items, so the same sale_id or product_id may appear multiple times. In Power BI, these duplicates were handled during analysis using DAX measure.

Step 3: Data Type Check (numerical)

```
import pandas as pd

# Load all sheets
file_path = "originalDataset.xlsx"

tables = {
    "employee": pd.read_excel(file_path, sheet_name="employee"),
    "customer": pd.read_excel(file_path, sheet_name="customer"),
    "supplier": pd.read_excel(file_path, sheet_name="supplier"),
    "product": pd.read_excel(file_path, sheet_name="product"),
    "order": pd.read_excel(file_path, sheet_name="order"),
    "order_product": pd.read_excel(file_path, sheet_name="order_product"),
    "sale": pd.read_excel(file_path, sheet_name="sale"),
    "sale_product": pd.read_excel(file_path, sheet_name="sale_product")
}

# Validate all numeric columns
numeric_checks = {
    "product.purchase_price": tables['product']['purchase_price'],
    "product.selling_price": tables['product']['selling_price'],
    "product.balance": tables['product']['balance'],
    "order.amount": tables['order']['amount'],
    "order_product.quantity": tables['order_product']['quantity'],
    "sale.amount": tables['sale']['amount'],
    "sale_product.quantity": tables['sale_product']['quantity']
}

for name, col in numeric_checks.items():
    invalid_count = col.apply(pd.to_numeric, errors='coerce').isna().sum()
    total = len(col)
    print(f"{name} -> Invalid: {invalid_count} / {total}
({invalid_count/total:.2%})")
```

Output:

```
product.purchase_price -> Invalid: 0 / 30 (0.00%)
product.selling_price -> Invalid: 0 / 30 (0.00%)
product.balance -> Invalid: 0 / 30 (0.00%)
order.amount -> Invalid: 0 / 35 (0.00%)
order_product.quantity -> Invalid: 0 / 95 (0.00%)
sale.amount -> Invalid: 0 / 14700 (0.00%)
sale_product.quantity -> Invalid: 0 / 23117 (0.00%)
```

It shows that all the numerical data types above are all clean and realistic.

3.2 POWER QUERY TRANSFORMATION

In this part, the data transformation process was carried out in Power BI to ensure data consistency and completeness before visualisation.

In the employee table, the date_left column contained several null values. Rather than deleting these records, a new column was created in Power Query using the Custom Column feature. The formula applied was as follows:

New column: employee_status

```
each if [date_left] = "NULL" or [date_left] = null then "Available" else
"Resigned"
```

As a result, each employee record is now categorized as either Available or Resigned, depending on whether the date_left field is NULL. If the date_left is NULL, the employee is categorized as Available, meanwhile if the date is present, the employee is categorised as Resigned. This approach ensures that all employee data is retained and clearly represented in the report.

New column: customer_status

```
each if [customer_id] = "NULL" or [customer_id] = null then "Unregistered" else
"Registered "
```

Similarly, other tables with missing values, such as the sale table, were handled in the same way. Records with the NULL customer_id were categorised in the customer_status column as either Registered or Unregistered.

3.3 DATA MODELING (RELATIONSHIP)

Initially, Power BI automatically detected and created relationships between the tables. However, we performed a cross-check to ensure all relationships were accurate and logically consistent. After verification, we adjusted and created additional relationships where necessary to improve data integrity and visualization performance.

The final relationship established are as follows:

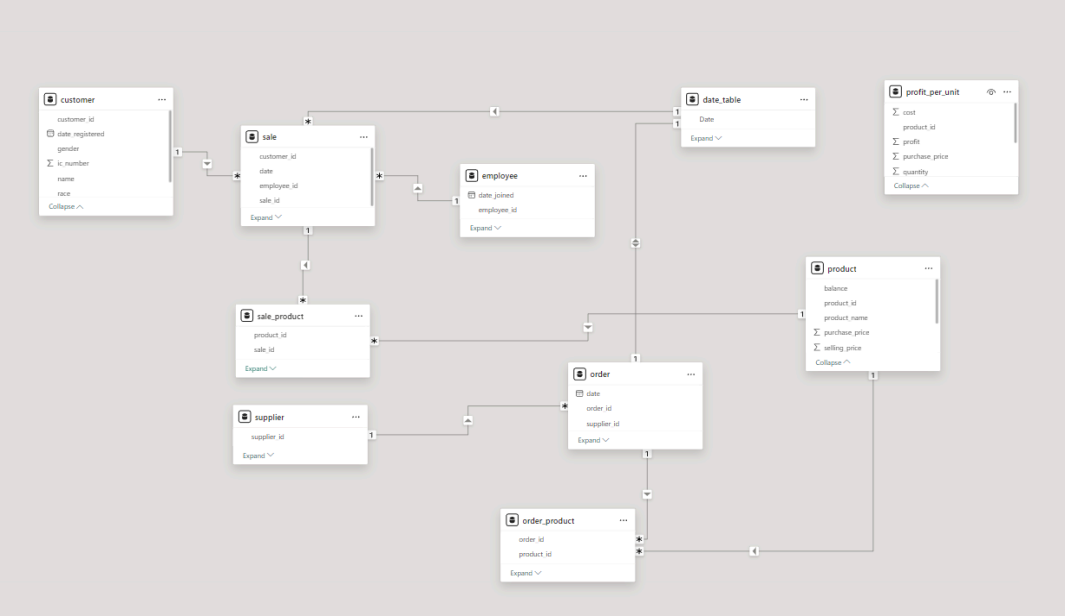

Figure 1: Data Modelling (Relationship)

From: table (column)	Relationship	To: table (column)	Status
date_table (Date)	1 — 1	order (date)	Active
order (supplier_id)	* — 1	supplier (supplier_id)	Active
order_product (order_id)	* — 1	order (order_id)	Active
order_product (product_id)	* — 1	product (product_id)	Active
sale (customer_id)	* — 1	customer (customer_id)	Active
sale (date)	* — 1	date_table (Date)	Active
sale (employee_id)	* — 1	employee (employee_id)	Active
sale_product (product_id)	* — 1	product (product_id)	Active
sale_product (sale_id)	* — 1	sale (sale_id)	Active

Figure 2: Cardinality of Relationships Between Tables

3.4 CALCULATIONS (DAX, MEASURES)

Measure Name	Formula	Purpose
Total Customers	`Total Customers = COUNT(Customer[customer_id])`	To calculate the total customers registered based on the customer ID.
Total Employees	`Total Employees = COUNT(employee[employee_id])`	To calculate the total employees in the workforce based on employee ID.
Total Quantity Supplied	`Total Quantity Supplied = SUM('Order_Product'[quantity])`	To calculate the supply flow of inventory.
% of Total Sales	`% of Total Sales = DIVIDE( [Revenue (RM)], CALCULATE([Revenue (RM)], ALL('product')), 0 ) * 100`	To calculate each product's (or category's) share of total revenue.
Remaining Stock	`Remaining Stock = SUM('product'[balance])`	To calculate the overall total of remaining stock left in inventory.
Total Cost	`Total Cost = SUMX( 'sale_product', 'sale_product'[quantity] * RELATED('product'[purchase_price]) )`	To know how much money is spent to purchase all the sold items. (Purchase Cost)
Total Revenue	`Total Revenue = SUM(profit_per_unit[revenue])`	To calculate the total revenue overall.
Average Sales Made	`Average Sales Made = DIVIDE( SUM (sale[amount]), COUNTROWS(sale), 0 )`	To know the average sales made from all the transactions.
Total Customer Overall	`Customer Count (Include Null) = COUNTROWS( FILTER( sale, sale[customer_status] =`	To calculate the total customers overall (including registered and unregistered) based on customer status.

	"Registered" || sale[customer_status] = "Unregistered"))	
Total Transactions (Sales)	Total Transactions (Sales) = COUNT(sale[customer_id])	To find out the total transactions overall during sales.
Profit (RM)	Profit (RM) = SUMX('sale_product', ('sale_product'[quantity] * RELATED('product'[selling_price])) - ('sale_product'[quantity] * RELATED('product'[purchase_price])))	To calculate how much profit made from the sale, then sum up all the profits.
Profit Margin %	Profit Margin % = DIVIDE([Profit (RM)], [Revenue (RM)], 0) * 100	To know how profitable the sales made per ringgit of revenue.
Revenue (RM)	Revenue (RM) = SUMX('sale_product', 'sale_product'[quantity] * RELATED('product'[selling_price]))	To calculate total sales income generated from all products before deducting any costs (calculated as selling price × quantity sold).
Total Sales (Overall)	Total Sales (Overall) = SUM(Sale[amount])	To calculate overall sales throughout the years and the cumulative amount made.
Total Units Sold	Total Units Sold = SUM('sale_product'[quantity])	To calculate the total stock sold for each product.

4.0 VISUALIZATION (DASHBOARD)

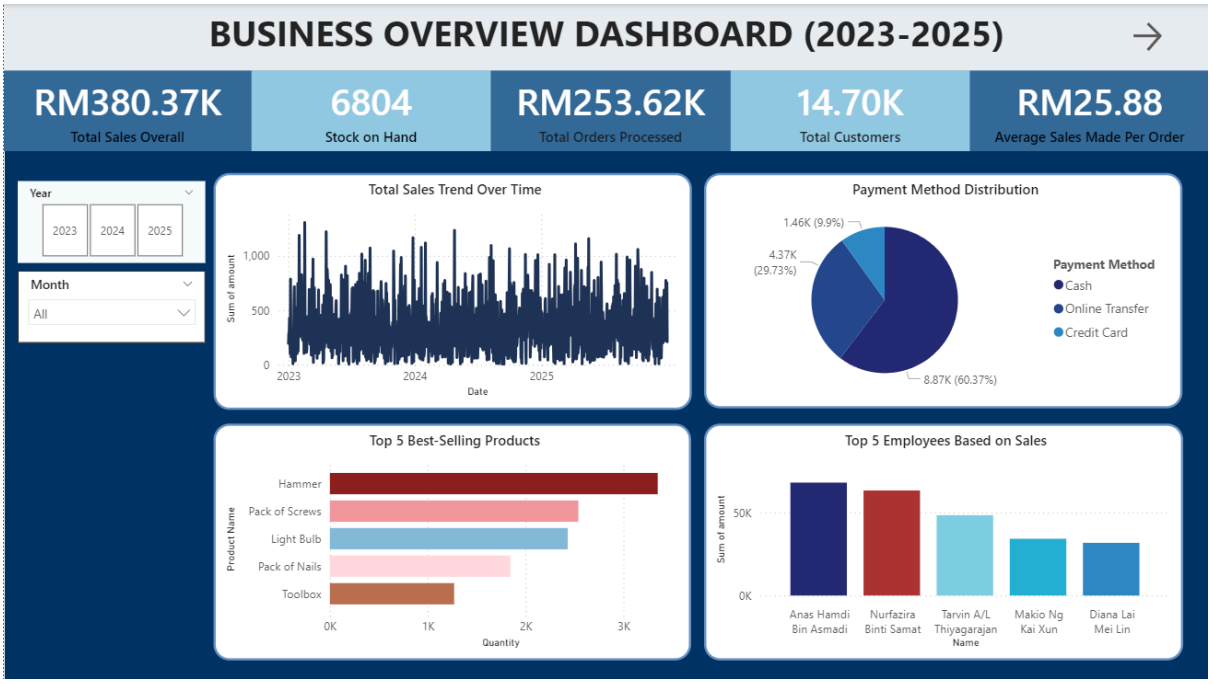

Figure 3: Business Performance Overview (2023-2025)

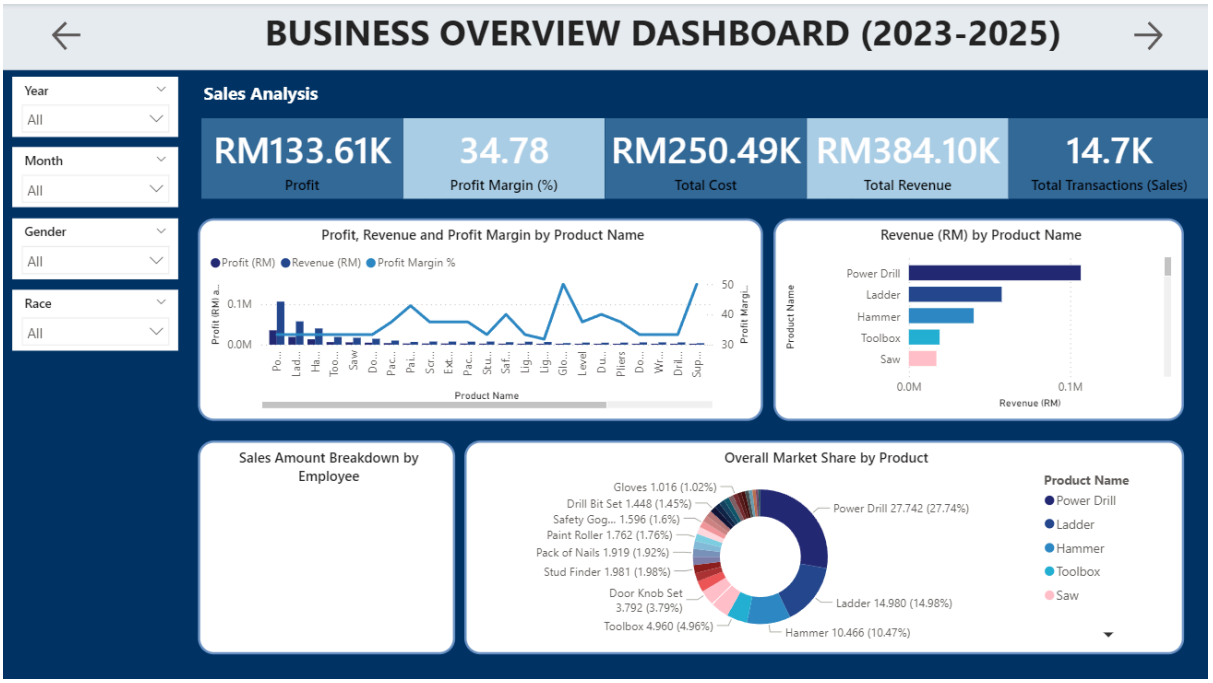

Figure 4: Sales Analysis (2023-2025)

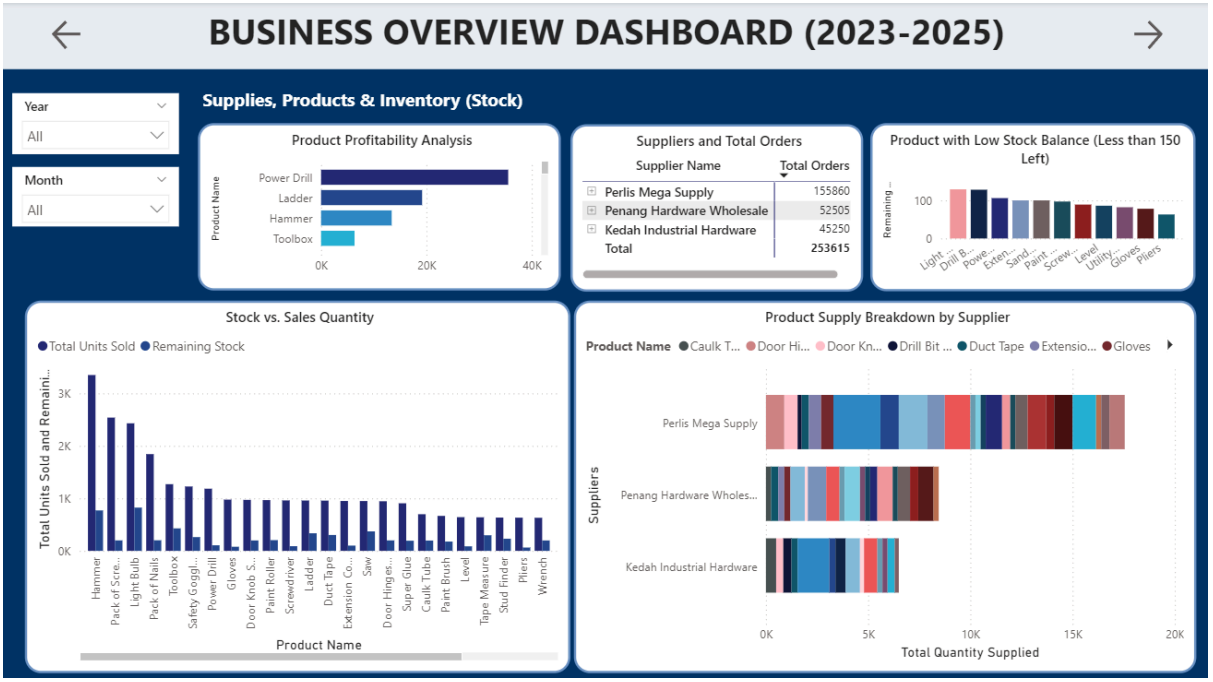

Figure 5: Supplies, Products, and Inventory (2023-2025)

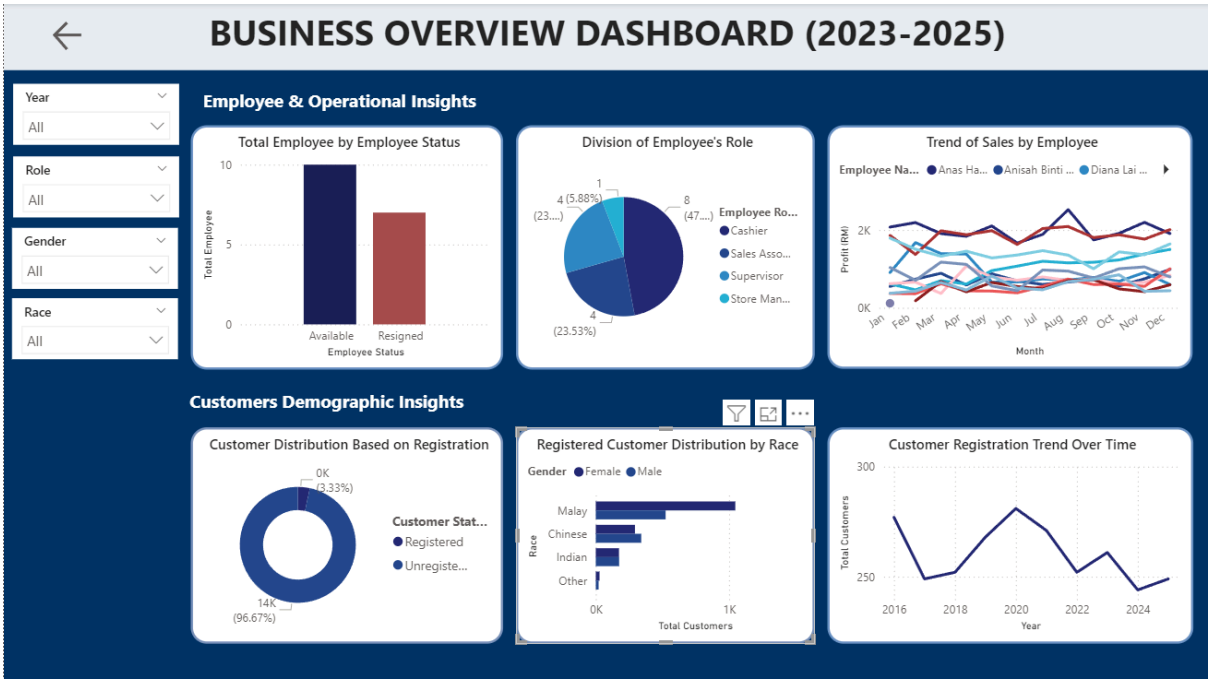

Figure 6: Overview for Employee & Operational and Customer Demographic Insights (2023-2025)

5.0 REFERENCES

OpenAI. (2025). *ChatGPT (GPT-5)* [Large language model]. https://chatgpt.com/share/68f60dfd-6168-8006-a063-7d67cc53826f