Project Coversheet

Full Name	Polislava Kalcheva
Email	polislavakk@gmail.com
Contact Number	+359884761900
Date of Submission	05-08-2025
Project Week	Week 2

Project Guidelines and Rules

1. Submission Format

- **Document Style**:
    - Use a clean, readable font such as *Arial* or *Times New Roman*, size 12.
    - Set line spacing to **1.5** for readability.
- **File Naming**:
    - Use the following naming format:
      Week X – [Project Title] – [Your Full Name Used During Registration]
      Example: Week 1 – Customer Sign-Up Behaviour – Mark Robb
- **File Types**:
    - Submit your report as a **PDF**.
    - If your project includes code or analysis, attach the **.ipynb notebook** as well.

2. Writing Requirements

- Use formal, professional language.
- Structure your content using headings, bullet points, or numbered lists.

3. Content Expectations

- Answer **all** parts of each question or task.

- Reference tools, frameworks, or ideas covered in the programme and case studies.
- Support your points with practical or real-world examples where relevant.
- Go beyond surface-level responses. Analyse problems, evaluate solutions, and demonstrate depth of understanding.

4. Academic Integrity & Referencing

- All submissions must be your own. Plagiarism is strictly prohibited.
- If you refer to any external materials (e.g., articles, studies, books), cite them using a consistent referencing style such as APA or MLA.
- Include a references section at the end where necessary.

5. Evaluation Criteria

Your work will be evaluated on the following:

- Clarity: Are your answers well-organised and easy to understand?
- Completeness: Have you answered all parts of the task?
- Creativity: Have you demonstrated original thinking and thoughtful examples?
- Application: Have you effectively used programme concepts and tools?
- Professionalism: Is your presentation, language, and formatting appropriate?

6. Deadlines and Extensions

- Submit your work by the stated deadline.
- If you are unable to meet a deadline due to genuine circumstances (e.g., illness or emergency), request an extension **before the deadline** by emailing: **support@uptrail.co.uk**
  Include your full name, week number, and reason for extension.

7. Technical Support

- If you face technical issues with submission or file access, contact our support team promptly at **support@uptrail.co.uk.**

8. Completion and Certification

- Certificate of Completion will be awarded to participants who submit at least two projects.
- Certificate of Excellence will be awarded to those who:
  - Submit all four weekly projects, and
  - Meet the required standard and quality in each.
- If any project does not meet expectations, you may be asked to revise and resubmit it before receiving your certificate.

YOU CAN START YOUR PROJECT FROM HERE

1. Introduction

I have joined Green Cart Ltd., a growing UK-based e-commerce company focused on eco friendly household products. The company is preparing for its Q2 performance review, and your manager on the Data & Insights team has asked me to investigate sales and customer behaviour across regions and product lines. I have been given access to sales, product, and customer datasets, and I job was to clean and merge the data, to create new features, to analyse patterns and performance, and to present insights using charts and summary tables. The findings will inform upcoming marketing and operational strategies.

2. Data Cleaning Summary

2.1. What was cleaned

Converting date columns order_date, signup_date, launch_date from the three tables to datetime using pd.to_datetime() python function.

Moreover, numeric columns were validates by ensuring quantity, unit_price, and discount_applied columns are all non-negative. If there is a value smaller than 0, it was made equal to 0.

```
#converting to numeric all the values in these 3 columns
df_sales_data.loc[:, 'quantity'] = pd.to_numeric(df_sales_data['quantity'], errors='coerce')
df_sales_data.loc[:, 'unit_price'] = pd.to_numeric(df_sales_data['unit_price'], errors='coerce')
df_sales_data.loc[:, 'discount_applied'] = pd.to_numeric(df_sales_data['discount_applied'], errors='coerce')
#checking for negative values
df_sales_data[df_sales_data['quantity'] < 0]
df_sales_data[df_sales_data['unit_price'] < 0]
df_sales_data[df_sales_data['discount_applied'] < 0]
#replacing negative values with 0
df_sales_data.loc[df_sales_data['quantity'] < 0, 'quantity'] = 0
df_sales_data.loc[df_sales_data['unit_price'] < 0, 'unit_price'] = 0
df_sales_data.loc[df_sales_data['discount_applied'] < 0, 'discount_applied'] = 0
```

2.2. Duplicates removed

Duplicates were identified and removed using .duplicated() and .drop_duplicates() python functions. For example, I check the duplicates in table df_sales_data ordered based on the column ‘order_id’.

```
#table1
#check the duplicates before drop
df_sales_data[df_sales_data.duplicated(subset='order_id', keep=False)]
```

	order_id	customer_id	product_id	quantity	unit_price	order_date	delivery_status	payment_method	region	discount_applied
156	O515400	C00103	P0024	4	44.15	2006-07-25	Delivered	Paypal	East	0.15
793	O916245	C00390	P0010	1	24.57	2006-07-25	Delayed	Paypal	South	0.10
1461	O515400	C00389	P0027	2	22.04	2006-07-25	Delayed	Paypal	North	0.05
2712	O916245	C00070	P0011	3	20.83	2006-07-25	Delayed	Bank Transfer	West	0.05

Then, I removed them all with the .duplicated().

```
#table1
df_sales_data = df_sales_data[~df_sales_data.duplicated(subset='order_id', keep=False)]
#removing all duplicates because I am not sure which one is the most logical to keep
df_sales_data
```

The same process was repeated with the other two tables.

2.3. Missing data handled

The functions .isnull().sum() were used to identify missing values in all tables.

```
df_sales_data.isnull().sum()
```

```
[88]:

order_id            0
customer_id         0
product_id          0
quantity            0
unit_price          1
order_date          3
delivery_status     0
payment_method      0
region              0
discount_applied  517
```

```
df_product_info.isnull().sum()
```

```
[89]:

product_id      0
product_name    0
category        0
launch_date     0
base_price      0
supplier_code   0
dtype: int64
```

```
df_customer_info.isnull().sum()
```

```
[90]:

customer_id     3
email           6
signup_date     4
gender          4
region          3
loyalty_tier    2
dtype: int64
```

Therefore, I filled these empty values with 0.0 or ‘unknown’ as entry in df_sales_data table.

```
df_sales_data['order_id'] = df_sales_data['order_id'].fillna(0.0)
df_sales_data['customer_id'] = df_sales_data['customer_id'].fillna('unknown')
df_sales_data['product_id'] = df_sales_data['product_id'].fillna('unknown')
df_sales_data['quantity'] = df_sales_data['quantity'].fillna(0)
df_sales_data['unit_price'] = df_sales_data['unit_price'].fillna(00.00)
df_sales_data['order_date'] = df_sales_data['order_date'].fillna(df_sales_data['order_date'].mode()[0])
df_sales_data['delivery_status'] = df_sales_data['delivery_status'].fillna('unknown')
df_sales_data['payment_method'] = df_sales_data['payment_method'].fillna('unknown')
```

The same logic was applied to df_customer_info table as well.

```
df_customer_info['customer_id'] = df_customer_info['customer_id'].fillna('unknown')
df_customer_info['email'] = df_customer_info['email'].fillna('unknown')
df_customer_info['signup_date'] = df_customer_info['signup_date'].fillna(df_customer_info['signup_date'].mode()[0])
df_customer_info['gender'] = df_customer_info['gender'].fillna('unknown')
df_customer_info['region'] = df_customer_info['region'].fillna('unknown')
df_customer_info['loyalty_tier'] = df_customer_info['loyalty_tier'].fillna('unknown')
```

The df_product_info did not have any empty values from the beginning.

2.4. Inconsistent labels standardized

Standardising text formatting using python functions str.strip(), .str.lower(), and .str.title() to remove any white spaces in each cell of the table, to make each word lowercase, and to make only the first letter of each word capital letter. This was applied on each of the three given tables df_sales_data, df_product_info, df_customer_info.

Moreover, records with invalid value were corrected. For example, in df_sales_data table with the following code the unique values were identified in the table per column.

```
#table1
for col in df_sales_data.columns:
    print(f"\n Checking the unique values if there is anything uncommon'{col}':")
    print(df_sales_data[col].unique())
```

Then, the inconsistencies were corrected with the right name such as ‘Nrth’ became ‘North’, ‘Delrd' and 'Delyd' became 'Delayed', ‘Bank Transfr' became 'Bank Transfer’.

```
#delivery_status
df_sales_data.loc[:, 'delivery_status'] = df_sales_data['delivery_status'].replace({
    'Delrd': 'Delayed',
    'Delyd': 'Delayed'})
#payment_method
df_sales_data.loc[:, 'payment_method'] = df_sales_data['payment_method'].replace({
    'Bank Transfr': 'Bank Transfer'})
#region
df_sales_data.loc[:, 'region'] = df_sales_data['region'].replace({
    'Nrth': 'North'})
#there are mistakes in the entries in delivery_status: 'Delrd', 'Delyd';
                                      #payment_method: 'Bank transf';
                                      #region:         'Nrth'
```

The same process was repeated for the df_customer_info with inconsistencies such as 'Femle' - 'Female', 'Gld' - 'Gold', 'Brnze' - 'Bronze', and 'Sllver' - 'Silver'. The df_product_info did not have any noticeable inconsistencies.

In the end when the tables were merged (check 3. Feature engineering), the newly added ‘days_to_order’ feature has negative entries because some of the original entries in ‘launch_date’ are later than the entries in the ‘order_date’ column. Therefore, the rows were removed that contain negative entries in ‘days_to_order’.

```
merged_df = merged_df[merged_df['days_to_order'] >= 0]
merged_df
```

3. Feature engineering

Firstly, sales_data with product_info using product_id were merged.

```
merged_sales_data_product_info = pd.merge(df_sales_data, df_product_info, on='product_id', how='left')
merged_sales_data_product_info
```

The result was merged with customer_info table using customer_id column.

```
merged_df = pd.merge(
    merged_sales_data_product_info,
    df_customer_info,
    on='customer_id',
    how='left'
)
```

Then, the following new features were created: revenue = quantity × unit_price × (1 - the feature discount_applied); order_week = from order_date column; price_band = a category for the unit price column as ‘Low’ for lower than £15, ‘Medium’ for the range £15-30, ‘High’ for higher than £30; days_to_order = the days between launch_date and order_date; email_domain = extracting the domain from each email (e.g., gmail.com); is_late = ‘True’ if delivery_status column is ‘Delayed’. The used code is below.

```
merged_df['revenue'] = merged_df['quantity'] * merged_df['unit_price'] * (1 - merged_df['discount_applied'])
merged_df['order_week'] = merged_df['order_date'].dt.isocalendar().week
merged_df['price_band'] = pd.cut(
    merged_df['unit_price'],
    bins=[-float('inf'), 15, 30, float('inf')],
    labels=['Low', 'Medium', 'High'])
merged_df['days_to_order'] = (merged_df['order_date'] - merged_df['launch_date']).dt.days
merged_df['email_domain'] = merged_df['email'].map(lambda x: x.split('@')[1] if pd.notnull(x) and '@' in x else None)
merged_df['is_late'] = merged_df['delivery_status'] == 'Delayed'
merged_df
```

4. Key findings & trends

4.1. The week was 30 for all entries and the highest revenue was 4262.5505 for region West. (Because it was told in the description to merge sales_data with product_info tables, then merge with customer_info table using a left join to preserve all sales transactions.Therefore, the analysis is about sales. A left join preserves the original data from sales_data. So, the region tied to the sale itself (region_x, from df_sales_data) is the one that aligns directly with each order. A region from sales_data shows where the order happened. A region from customer_info shows where the customer is registered, not where the transaction occurred.)

```
weekly_revenue = merged_df.groupby(['order_week', 'region_x'])['revenue'].sum().reset_index()
weekly_revenue
```

[120]:

	order_week	region_x	revenue
0	30	Central	4073.51
1	30	East	4223.303
2	30	North	2891.934
3	30	South	4197.6225
4	30	West	4262.5505

4.2. A customer behaviour was measured by loyalty_tier and signup_month features. Therefore, for 2001, the leading loyalty tier is silver with 51.204 and for 2031 is again silver tier with 137.28. The months entries differ per year.

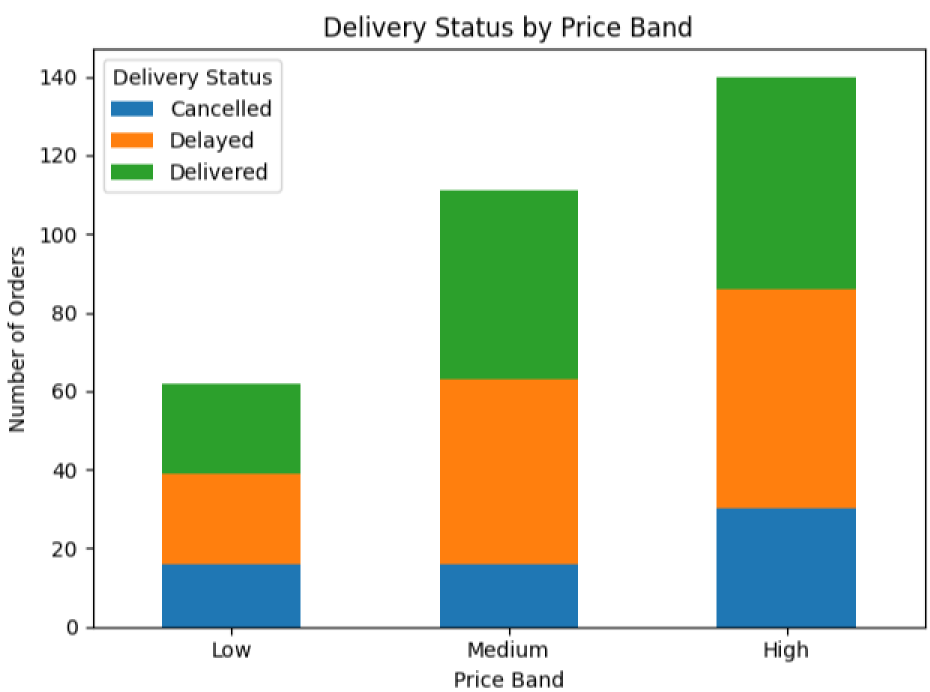
```
customer_behavior = (
    merged_df
    .groupby(['loyalty_tier', 'signup_month'])['revenue']
    .sum()
    .unstack(fill_value=0)
)
customer_behavior
```

signup_month	2001-01	2001-02	2001-04	2001-07	2001-10	2001-11	2002-01	2002-03	2002-05	2002-06	...	2030-01	2030-05	2030-08	2030-09	2030-11	2030-12	2031-01	2031-05	20
loyalty_tier																				
Bronze	0	0	0	0	47.796	0	0	0	44.4315	0	...	0	0	0	0	0	0	0	0	
Gold	0	0	37.288	42.194	0	0	11.2	34.28	0	0	...	0	59.7355	100.56	679.3875	65.568	265.62	10.6685	0	
Silver	51.204	0	0	0	0	46.821	0	0	0	157.985	...	0	0	133.86	0	45.461	0	0	137.28	84
unknown	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

2031-01	2031-05	2031-10	2031-12
0	0	0	0
10.6685	0	0	69.635
0	137.28	84.36	0
0	0	0	0

4.3. Another insight is that the delivery performance by region and price_band was measured and was shown the mean value of how late each delivery has been. The latest record for the central region is with medium price, for the east region is with high price, for the north region is with medium price, for the south region is with high price and for the west region is with low price. Consequently, attention is needed in the east and south regions in order to improve the delivery time and not loose clients.

	region_x	price_band	is_late
0	Central	Low	0.461538
1	Central	Medium	0.576923
2	Central	High	0.428571
3	East	Low	0.400000
4	East	Medium	0.347826
5	East	High	0.480000
6	North	Low	0.428571
7	North	Medium	0.450000
8	North	High	0.360000
9	South	Low	0.176471
10	South	Medium	0.318182
11	South	High	0.343750
12	West	Low	0.500000
13	West	Medium	0.400000
14	West	High	0.400000



4.4. The last insight is about a preferred payment method based on the loyalty tier. Gold tier has the highest number of payments in total of 170 indicating to be the most preferred payment method. On second place is Bronze with 71 and Silver with 70.

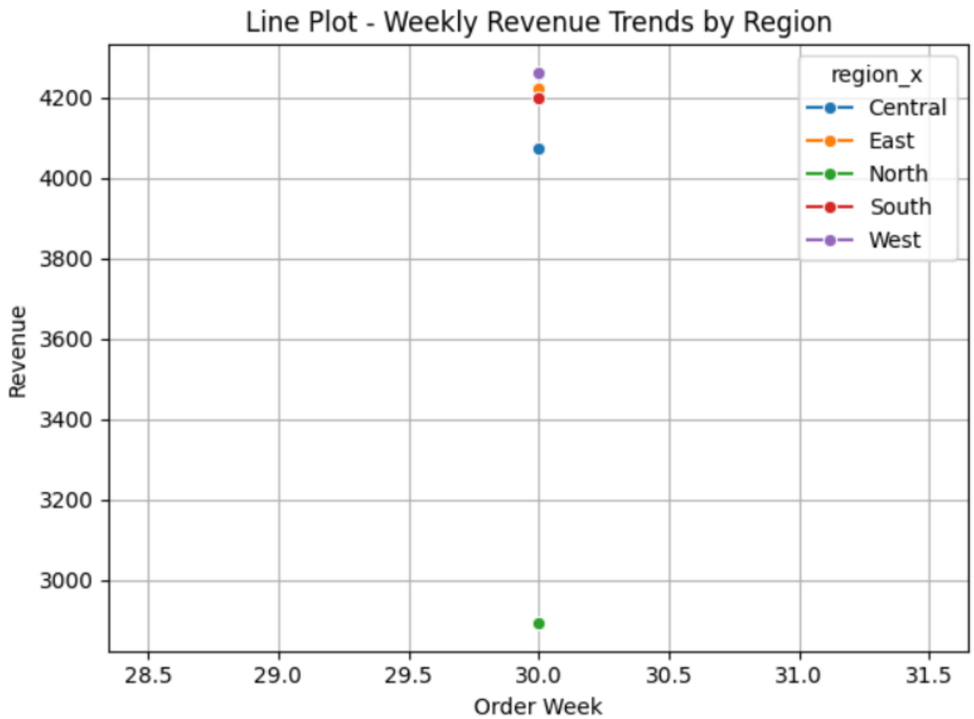
	loyalty_tier	payment_method	count
0	Bronze	Bank Transfer	18
1	Bronze	Credit Card	36
2	Bronze	Paypal	17
3	Gold	Bank Transfer	25
4	Gold	Credit Card	101
5	Gold	Paypal	44
6	Silver	Bank Transfer	20
7	Silver	Credit Card	33
8	Silver	Paypal	17

5. Business Questions

5.1. Which product categories drive the most revenue, and in which regions?

The regions west, east, and south have similar revenue of approximately 4300, 4250, and 4200 respectively. However, the central region has revenue of approximately 4080. Surprisingly, the north region has revenue of less than 3000 which is completely opposite from the rest of the regions. Therefore, the highest revenue has the west region with approximately 4300. According to the categories, the cleaning categories has highest total revenue of 12500. (there are 54 non numeric values and because of them not all 5 categories can be shown even though I tried to remove the rows containing such entries.)

```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.lineplot(data=weekly_revenue, x='order_week', y='revenue', hue='region_x', marker='o')
plt.title('Line Plot - Weekly Revenue Trends by Region')
plt.xlabel('Order Week')
plt.ylabel('Revenue')
plt.grid(True)
plt.tight_layout()
plt.show()
```

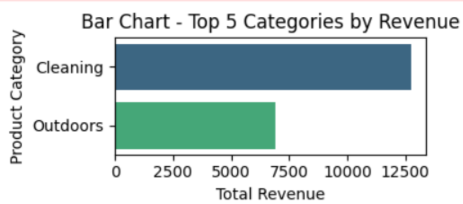



```python
print(merged_df['revenue'].dtype)
print(merged_df['revenue'].isna().sum())
#there are 54 NaN values and because of them not all 5 categories are shown.
```

```
float64
54
```

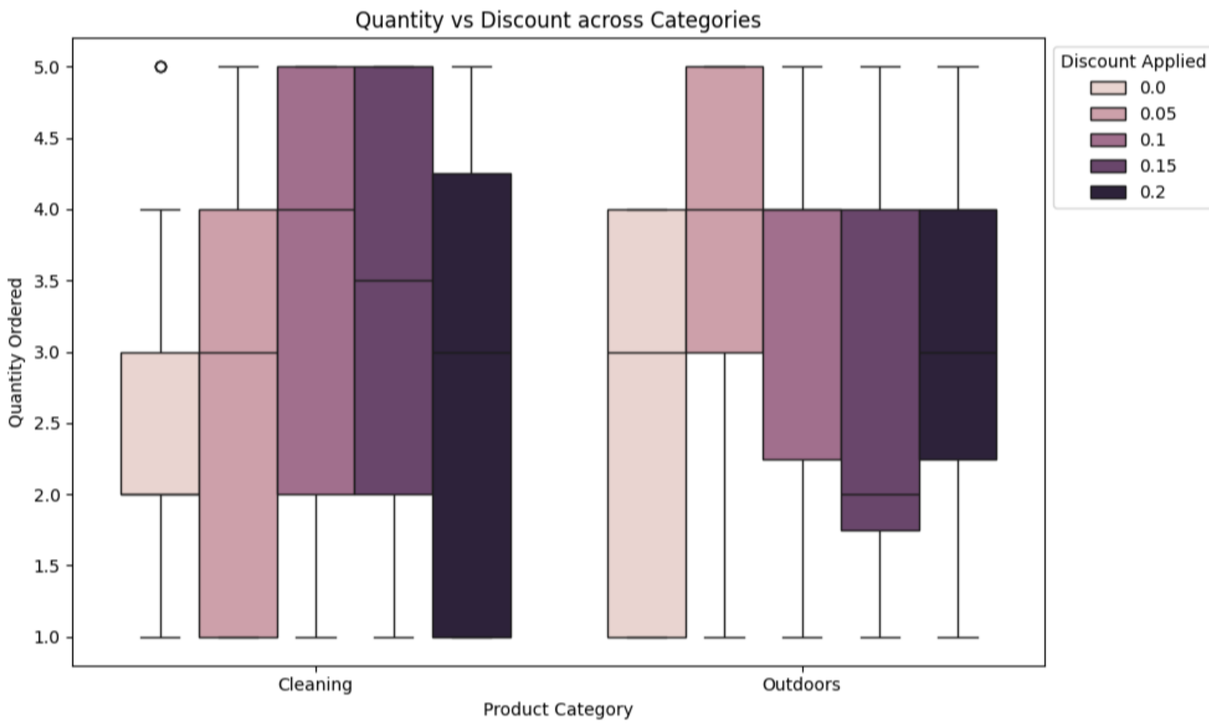
```python
category_revenue = (
    merged_df.groupby('category')['revenue'].sum().sort_values(ascending=False).head(5).reset_index()
)
plt.figure(figsize=(4, 2))
sns.barplot(data=category_revenue, x='revenue', y='category', palette='viridis')
plt.title('Bar Chart - Top 5 Categories by Revenue')
plt.xlabel('Total Revenue')
plt.ylabel('Product Category')
plt.tight_layout()
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_41760\2080711058.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=category_revenue, x='revenue', y='category', palette='viridis')
```



5.2. Do discounts lead to more items sold?

Due to the empty entries, only two categories are visible. Between the two Cleaning - discounts may slightly boost quantity sold and Outdoors - discounts show no real effect. Therefore, discount effectiveness is dependent on the category, and not strong across the board.



Which loyalty tier generates the most value? Refer to 4.2. in the report.

Are certain regions struggling with delivery delays? Yes, refer to 4.3. in the report.

Do customer signup patterns influence purchasing activity? No clear output for this question.

6. Empty entries need to be decreased. Delivery status needs to be enhanced – the late and cancelled shipments.
7. Adding automated checks will enhance significantly the data from the beginning. Many empty values disturb the dataset and its output overview.