Table of Contents

Introduction 1

Dataset 2

Data Processing Outcomes 2

Data Clean & Preparation 3

Further Data Cleaning and Initial Analysis 5

Further analysis initiated with df_without_c 6

Further Analysis and Data Visualisation 6

Analysis and Data Visualization for Dataframe df_without_c 6

Analysis and Data Visualization for Dataframe df_with_c 11

Data quality 13

Conclusion 14

Introduction

This dataset includes transactions from December 1, 2010, to December 9, 2011, for a UK online retailer specializing in unique gifts. Many customers are wholesalers. The purpose of this data analysis and visualization task is to gain insights into sales transactions, customer behaviour, and product performance. Key questions include: How do sales change over time and across different regions? What is the customer purchasing patterns, and how do they relate to seasons and regions? What are the popular products, and which products generate the most revenue?

Dataset

There are eight attributes in this dataset including invoice numbers, product codes, descriptions, quantities, invoice dates, unit prices, customer IDs, and countries.

Dataset Source: https://archive.ics.uci.edu/dataset/352/online+retail

Dataset Overview

```
[13]: retail_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

There are 541,909 rows and 8 columns in this dataset, and data types including strings, integers, dates, and floats.

Data Processing Outcomes

The libraries that were employed during the analysis include pandas, numpy, matplotlib and seaborn.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Three steps are used for data process purpose:

- Data Clean & Preparation
- Further Data Cleaning and Initial Analysis
- Further Analysis and Data Visualisation

Data Clean & Preparation

Operations including head and tail are used to get an overview of the dataset.

Displaying the top and bottom five rows

```
[8]: print(retail_df.head())

  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom

[10]: print(retail_df.tail())

       InvoiceNo StockCode                      Description  Quantity  \
541904    581587     22613      PACK OF 20 SPACEBOY NAPKINS        12
541905    581587     22899     CHILDREN'S APRON DOLLY GIRL          6
541906    581587     23254    CHILDRENS CUTLERY DOLLY GIRL          4
541907    581587     23255  CHILDRENS CUTLERY CIRCUS PARADE         4
541908    581587     22138    BAKING SET 9 PIECE RETROSPOT          3

               InvoiceDate  UnitPrice  CustomerID Country
541904 2011-12-09 12:50:00       0.85     12680.0  France
541905 2011-12-09 12:50:00       2.10     12680.0  France
541906 2011-12-09 12:50:00       4.15     12680.0  France
541907 2011-12-09 12:50:00       4.15     12680.0  France
541908 2011-12-09 12:50:00       4.95     12680.0  France
```

Summary statistics were generated at the beginning of the data cleaning process, and various approaches were applied accordingly.

Explore the dataset comprehensively by generating summary statistics.

```
retail_df.describe()
```

	Quantity	InvoiceDate	UnitPrice	CustomerID
count	534131.000000	534131	534131.000000	401564.000000
mean	9.916784	2011-07-04 12:02:14.286607360	4.654426	15281.266797
min	-80995.000000	2010-12-01 08:26:00	-11062.060000	12346.000000
25%	1.000000	2011-03-28 11:36:00	1.250000	13939.000000
50%	3.000000	2011-07-19 15:55:00	2.100000	15145.000000
75%	10.000000	2011-10-18 17:10:00	4.130000	16788.000000
max	80995.000000	2011-12-09 12:50:00	38970.000000	18287.000000
std	216.451709	NaN	97.460790	1713.978947

According to the summary statistics, several points need attention: there are negative values in quantity and unit price, and some extremely large values in both. The format of the invoice date needs to be changed to avoid hours and minutes. Additionally, the customer ID should be in string format, not integers.

The following modifications were made:

changing the invoice date format to year-month-day, changing the data type of customer ID to string, and conducting data exploration of the irregular values and outliers in quantity and unit price.

At the end of this step, all the column names were renamed and transformed to lower case to facilitate the analysis process.

Operations used for changing the data types include astype, fillna, dt.date, and dtypes (to check the result). Operations used for column reformatting include rename and str.lower.

2.2.1. Change data types

```
retail_df['InvoiceNo'] = retail_df['InvoiceNo'].astype('string')
retail_df['StockCode'] = retail_df['StockCode'].astype('string')
retail_df['Description'] = retail_df['Description'].astype('string')
retail_df['CustomerID'] = retail_df['CustomerID'].fillna(0).astype(int) # change to integer first to drop the decimal points.
retail_df['CustomerID'] = retail_df['CustomerID'].astype('string')
retail_df['Country'] = retail_df['Country'].astype('string')
retail_df['InvoiceDate'] = retail_df['InvoiceDate'].dt.date
retail_df.dtypes
```

```
InvoiceNo      string[python]
StockCode      string[python]
Description    string[python]
Quantity                int64
InvoiceDate            object
UnitPrice             float64
CustomerID     string[python]
Country        string[python]
dtype: object
```

2.2.2. Column cleanup

```
retail_df.rename(columns={
        'InvoiceNo':'Invoice_No',
        'StockCode':'Stock_Code',
        'InvoiceDate':'Invoice_Date',
        'UnitPrice':'Unit_Price',
        'CustomerID':'Customer_ID',
    }, inplace=True)

retail_df.columns=[col.lower() for col in retail_df]

retail_df.columns
```

```
Index(['invoice_no', 'stock_code', 'description', 'quantity', 'invoice_date',
       'unit_price', 'customer_id', 'country'],
      dtype='object')
```

Further Data Cleaning and Initial Analysis

This step begins with two for loops: one calculates the sum and average of 'unit_price', and the other converts all 'description' values to lowercase.

Employ a for loop to iterate through 'unit_price', calculating the sum and average of its values.

```
unit_price=retail_df.unit_price

total=0
count=0
for x in unit_price:
    total=total+x
    count+=1
print(f'The sum of unit price is: {round(total,2)}, and the average is: {round(total/count,2)}')
```

```
The sum of unit price is: 2486072.97, and the average is: 4.65
```

Utilise another loop to iterate through 'description', converting all values to lowercase for consistency.

```
lower_case=[]
description=retail_df.description
for x in description:
    x=x.lower()
    lower_case.append(x)

retail_df.description=lower_case
retail_df.description.sample(3)
```

```
408196    white hanging heart t-light holder
448394          pack of 6 small fruit straws
246824                  feltcraft doll rosie
Name: description, dtype: object
```

Exploration and data cleaning are applied to the columns 'invoice_no' and 'stock_no'. To efficiently observe these columns, a function named detect_non_number is created to detect any non-numerical characters. It is noted that invoice numbers with 'a' are entries for adjusting bad debt, and all three rows with 'a' are dropped from the dataset.

Next, exploration and data cleaning are employed for the 'unit_price' column, focusing on outliers. Methods used include selecting rows from the dataframe by specific conditions and dropping rows from the dataframe by index. The groupby operation is used to locate 'stock_code' and descriptions for rows that are not cancellation products but have corresponding 'invoice_no' containing 'c'.

Due to the complexity and size of the dataset, an analytic strategy is implemented to split the dataset into two parts: one with invoices containing 'c' and the other without. Exploration and analysis will be conducted separately for each. A new column, 'total_sales', is created for analysis and visualization purposes.

```
# Create new column 'total_sales'
# Make two copies of dataframe, one with invoice_no contains 'c', the other one not with 'c'


retail_df=retail_df.assign(total_sales=retail_df.quantity*retail_df.unit_price)

mask = retail_df.invoice_no.str.startswith('c')

df_with_c = retail_df[mask].copy()
df_without_c= retail_df[~mask].copy()
```

Operations and methods used in this process include assign, startswith, and copy.

Further analysis initiated with df_without_c.

For data cleaning and analysis purposes, the distribution of 'unit_price' needs to be examined repeatedly. Hence, a function called plot_box is defined to visualize the summary statistics for 'unit_price'. This provides a clear view of the distribution of outliers. Upon examination, all the outliers are rows that have been misplaced from df_with_c. All the misplaced rows are deleted from df_without_c and concatenated back to df_with_c.

3.7. Find all the misplaced rows, delete them from df_without_c, and concatenate them back to df_with_c.

```
# list the possible misplaced Stock Codes

possible_misplaced_stockcode = df_with_c.groupby(['stock_code','description'])['quantity'].sum().tail(9).index.get_level_values('stock_code').tolist()
print(f'The possible misplaced Stock Code include:\n{possible_misplaced_stockcode}')


# make a copy of misplaced rows, and add the missing 'c' to the invoice_no
misplaced=df_without_c[df_without_c.stock_code.isin(possible_misplaced_stockcode)].copy()
misplaced['invoice_no'] = 'c' + misplaced['invoice_no']

# make appropriate modifications as needed
misplaced['quantity']=misplaced['quantity'].mul(-1)
misplaced['total_sales']=misplaced['total_sales'].mul(-1)
print('\n', misplaced.sample(2))

# concatenate misplaced rows back to df_with_c
df_with_c=pd.concat([df_with_c, misplaced])

# delete misplaced rows from df_without_c
df_without_c=df_without_c[~df_without_c.stock_code.isin(possible_misplaced_stockcode)]
```

```
The possible misplaced Stock Code include:
['AMAZONFEE', 'BANK CHARGES', 'C2', 'CRUK', 'D', 'DOT', 'M', 'POST', 'S']

        invoice_no stock_code     description  quantity invoice_date  \
313193     c564471       POST         postage        -1   2011-08-25
163441     c550542        DOT  dotcom postage        -1   2011-04-19

        unit_price customer_id         country  total_sales
313193       18.00       12583          France       -18.00
163441      176.52           0  United Kingdom      -176.52
```

Operations and methods used in this process include:

groupby, sum, index.get_level_value, tolist, mul, sample, concat, isin.

The data clean has completed by now.

Further Analysis and Data Visualisation

Analysis and Data Visualization for Dataframe df_without_c

The total sales trends and the correlation relationships between numerical columns are examined first. Two new columns, ‘year’ and ‘month’ are created by extracting values from 'invoice_date'.

Operations and methods used include:

to_datetime, dt.year, dt.month, groupby, sum, plot, plt.figure, plt.xlabel, plt.ylabel, plt.title, plt.xticks

```
# Convert 'invoice_date' to datetime type
df_without_c['invoice_date'] = pd.to_datetime(df_without_c['invoice_date'])

# Extract year and month from 'invoice_date' and create new columns
df_without_c['year'] = df_without_c['invoice_date'].dt.year
df_without_c['month'] = df_without_c['invoice_date'].dt.month

groupby_month = df_without_c.groupby(['year', 'month'])['total_sales'].sum()
fig = plt.figure(figsize=(10, 5))
groupby_month.plot(kind='line')
plt.xlabel('Year, Month')
plt.ylabel('Sales')
plt.title('Trends in Total Sales Over Time', fontsize=20, fontweight='bold')
plt.xticks(range(len(groupby_month)), [f'{year}-{month}' for (year, month) in groupby_month.index], rotation=45)
plt.show()
```


```
numerical_colums=df_without_c.select_dtypes(include=np.number).columns
corrtest=df_without_c[numerical_colums]
print(corrtest.corr())
print('\nComment:\nOnly sales and quantity show a strong correlation, and these two are naturally
```

```
             quantity  unit_price  total_sales      year     month
quantity     1.000000   -0.022708     0.934333  0.003505 -0.002319
unit_price  -0.022708    1.000000     0.021568 -0.037256 -0.013453
total_sales  0.934333    0.021568     1.000000  0.000590  0.000300
year         0.003505   -0.037256     0.000590  1.000000 -0.368921
month       -0.002319   -0.013453     0.000300 -0.368921  1.000000

Comment:
Only sales and quantity show a strong correlation, and these two are naturally correlated.
Scatter plots and pair plots are not applicable for this dataset.
```

Customer Segmentation Analysis by Region

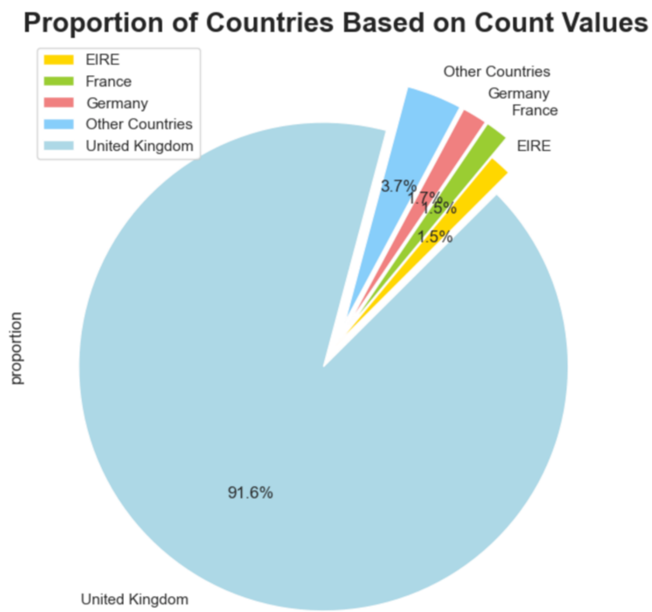
To facilitate the analysis and visualisation, countries with small contributions are grouped as 'Other Country'.

```python
# Modify the dataset for analytical purposes.

number_of_countries=len(df_without_c.country.unique())
print(f'The total number of countris is:\n\t\t\t>>> {number_of_countries}'
      '\nCountries with small contributions are grouped as \'Other Country\' for visualisation purpose.'
     )

df_without_c_copy=df_without_c.copy()
top_countries = df_without_c_copy['country'].value_counts(normalize=True).nlargest(4).index
df_without_c_copy.loc[~df_without_c_copy['country'].isin(top_countries), 'country'] = 'Other Countries'

df_without_c_copy=df_without_c.copy()
top_countries = df_without_c_copy['country'].value_counts(normalize=True).nlargest(4).index
df_without_c_copy.loc[~df_without_c_copy['country'].isin(top_countries), 'country'] = 'Other Countries'
```

```
The total number of countris is:
			>>> 38
Countries with small contributions are grouped as 'Other Country' for visualisation purpose.
```

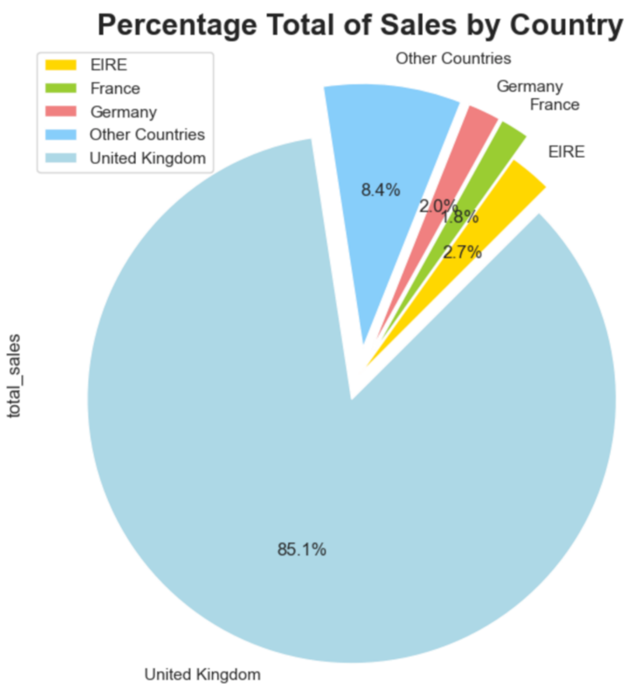
Analysis and visualisation of the proportion of count values of different regions:

```python
explode=(0,0.1,0.1,0.1,0.1)
fig = plt.figure(figsize=(8, 8))
colors = ['gold','yellowgreen', 'lightcoral', 'lightskyblue', 'Lightblue']
df_without_c_copy['country'].value_counts(normalize=True).sort_values().plot(kind='pie', autopct='%0.1f%%', startangle=45, explode=explode, colors=colors)
plt.legend()
plt.title('Proportion of Countries Based on Count Values',fontsize=20,fontweight='bold')
plt.show()
```



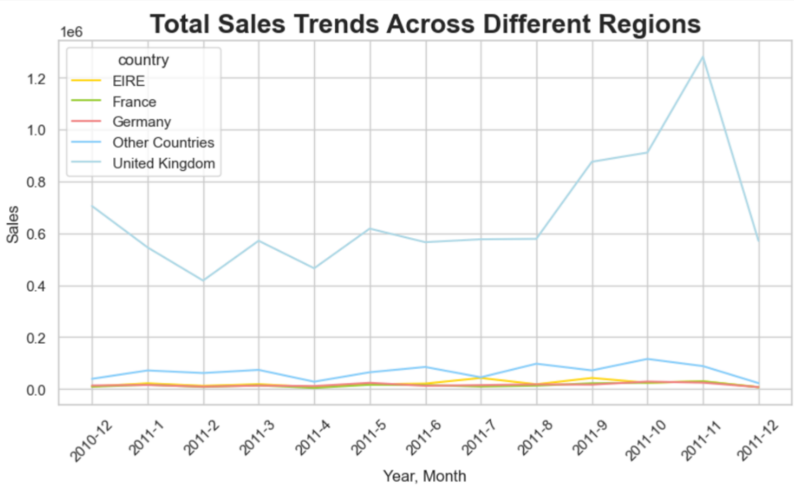
Analysis and visualisation of the percentage of total sales by region:

```
explode=(0,0.1,0.1,0.1,0.1)
fig = plt.figure(figsize=(8, 8))
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'Lightblue']
df_without_c_copy.groupby(['country'])['total_sales'].sum().plot(kind='pie', autopct='%0.1f%%', startangle=45, explode=explode, colors=colors)
plt.legend()
plt.title('Percentage Total of Sales by Country',fontsize=20,fontweight='bold')
plt.show()
```



Analysis and visualisation of total sales trends across different regions:

```
fig, ax = plt.subplots(figsize=(10, 5))
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'Lightblue']
df_without_c_copy.groupby(['country','year', 'month' ])['total_sales'].sum().unstack('country').plot.line(ax=ax, color=colors)
plt.xlabel('Year, Month')
plt.ylabel('Sales')
plt.xticks(range(len(groupby_month)), [f'{year}-{month}' for (year, month) in groupby_month.index], rotation=45)
plt.title('Total Sales Trends Across Different Regions',fontsize=20,fontweight='bold')
plt.show()
```



Product Segmentation Analysis by Price Range

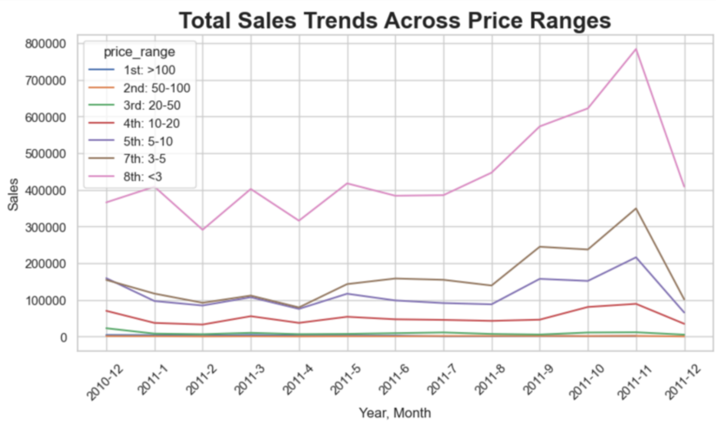
A new column, 'price_range', is created to segment all products into different price ranges based on the summary statistics for 'unit_price'.

A function called assign_price_range is defined and applied using the apply operation.

```python
# Divide all products into different price ranges according to the above statistics.

def assign_price_range(price):
    if price >= 100:
        return '1st: >100'
    elif 50 <= price < 100:
        return '2nd: 50-100'
    elif 20 <= price < 50:
        return '3rd: 20-50'
    elif 10 <= price < 20:
        return '4th: 10-20'
    elif 5 <= price < 10:
        return '5th: 5-10'
    elif 3 <= price < 5:
        return '7th: 3-5'
    else:
        return '8th: <3'

df_without_c_p['price_range'] = df_without_c_p['unit_price'].apply(assign_price_range)
```

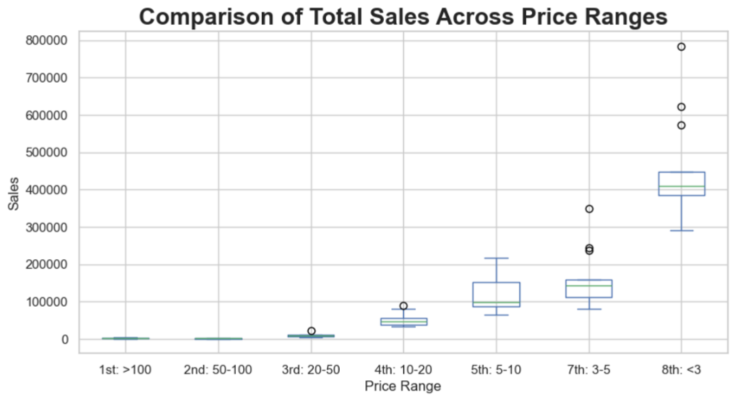
Analysis and visualisation of total sales trends across different price ranges:

```python
df_without_c_p.groupby(['price_range', 'year', 'month'])['total_sales'].sum().unstack('price_range').plot.line(figsize=(10, 5))
plt.xlabel('Year, Month')
plt.ylabel('Sales')
plt.xticks(range(len(groupby_month)), [f'{year}-{month}' for (year, month) in groupby_month.index], rotation=45)
plt.title('Total Sales Trends Across Price Ranges',fontsize=20,fontweight='bold')
plt.show()
```



Analysis and visualisation of the total sales between different price ranges:

```
df_without_c_p.groupby(['price_range', 'year', 'month'])['total_sales'].sum().unstack('price_range').plot.box(figsize=(10, 5))
plt.xlabel('Price Range')
plt.ylabel('Sales')
plt.title('Comparison of Total Sales Across Price Ranges',fontsize=20,fontweight='bold')
plt.show()
```

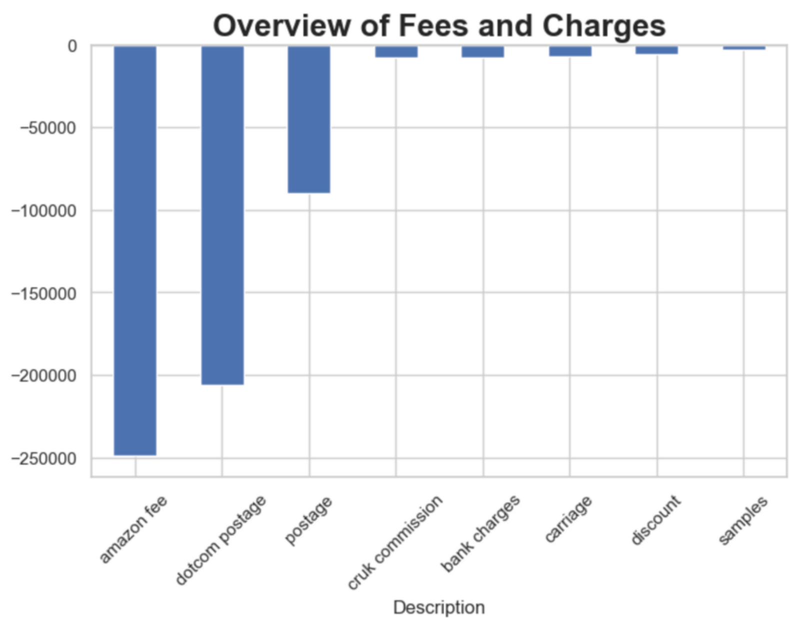


Analysis and Data Visualization for Dataframe df_with_c

In the previous exploration, the dataset includes two types of information: one concerning fees and charges, and the other related to cancelled orders. Analyses and visualizations have been conducted separately for each category.

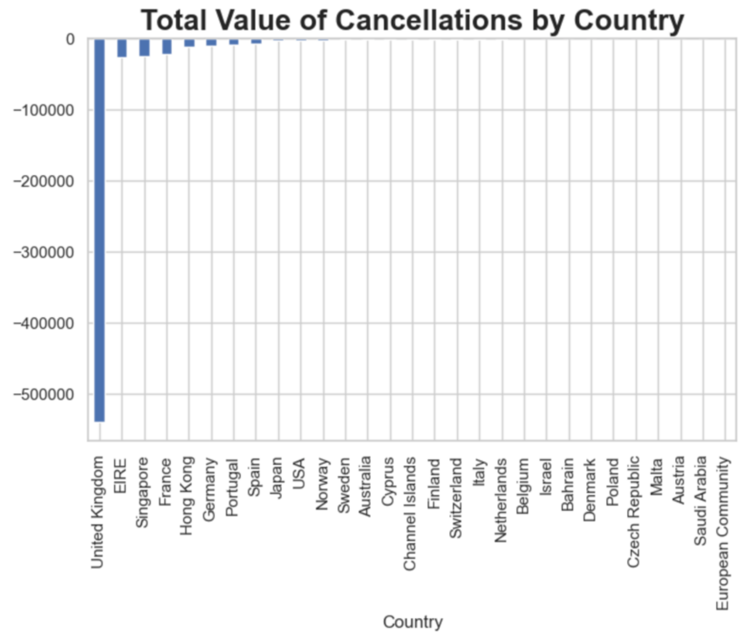
Summary of fees and charges:

```
fees_and_charges=df_with_c.groupby(['stock_code','description'])['quantity'].sum().tail(9).index.get_level_values('stock_code').tolist()
fees_and_charges.remove('M')

df_fees_and_charges=df_with_c[df_with_c.stock_code.isin(fees_and_charges)]

bar_chart=df_fees_and_charges.groupby(['stock_code'])['total_sales'].sum().sort_values().plot(kind='bar', figsize=(8, 5))
custom_labels = ['amazon fee', 'dotcom postage', 'postage', 'cruk commission', 'bank charges', 'carriage', 'discount', 'samples']
bar_chart.set_xticklabels(custom_labels, rotation=45)
plt.xlabel('Description')
plt.title('Overview of Fees and Charges',fontsize=20,fontweight='bold')

plt.show()
```



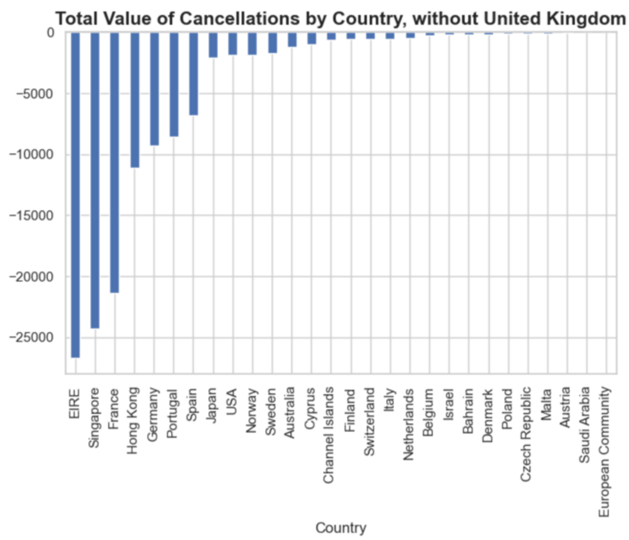
Summary of total cancellation value by country (with UK):

```
df_canceled = df_with_c[~df_with_c.stock_code.isin(fees_and_charges)]
df_canceled.groupby(['country'])['total_sales'].sum().sort_values().plot(kind='bar', figsize=(8, 5))
plt.xlabel('Country')
plt.title('Total Value of Cancellations by Country', fontsize=20, fontweight='bold')
plt.show()
```



Summary of total cancellation value by country (without UK):

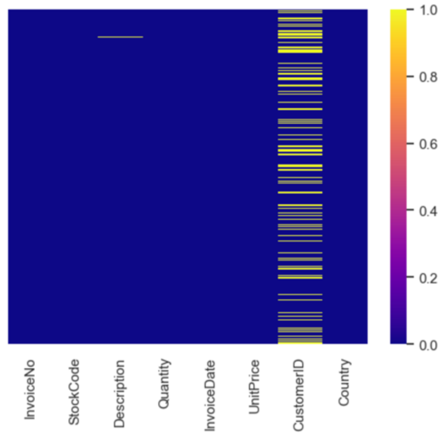
```
df_canceled=df_with_c[~df_with_c.stock_code.isin(fees_and_charges)]

df_canceled[df_canceled['country']!='United Kingdom'].groupby(['country'])['total_sales'].sum().sort_values().plot(kind='bar',figsize=(8, 5))
plt.xlabel('Country')
plt.title('Total Value of Cancellations by Country, without United Kingdom',fontsize=15,fontweight='bold')
plt.show()
```



Data quality

The code 'retail_df.isnull().sum()' generates a summary of the total missing values in each column. This finding is visualized using 'sns.heatmap'.

```
Summary of number of missing values in the dataset:
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```



Upon exploration, rows with missing descriptions are considered invalid inputs, as their unit price equals 0. Therefore, all rows with missing descriptions have been removed from the dataset. The percentage of these invalid inputs is 0.5%.

Although there were a large number of rows with missing Customer ID, these rows have complete entries for all other columns. Given the wholesale nature of the business, it is reasonable to assume that the Customer ID was missing due to small purchases. These null values were retained and replaced with empty strings during the data cleaning process.

There were 5263 duplicates in the dataset and is removed by the below process:

Remove duplicates

```
# Find duplicateds in the dataset.

print(f'The numberof duplicated rows found in the dataset is: \n {retail_df.duplicated().sum()}')
print(f'The total number of rows and colums for the dataset before dropping duplicates is: \n {retail_df.shape}')

# Remove duplicates from the dataset.

retail_df=retail_df.drop_duplicates()
print(f'The total number of rows and colums for the dataset after dropping duplicates becomes: \n {retail_df.shape}')
```

```
The numberof duplicated rows found in the dataset is:
 5263
The total number of rows and colums for the dataset before dropping duplicates is:
 (539394, 8)
The total number of rows and colums for the dataset after dropping duplicates becomes:
 (534131, 8)
```

Conclusion

The dataset maintains consistency and integrity, with only 5% invalid inputs.

In terms of sales trends, total sales showed minimal fluctuations between late 2010 and the first five months of 2011. Subsequently, they began to rise, peaking in November 2011, before sharply declining toward the year's end.

Customer segmentation analysis reveals that the majority of customers are from the UK, followed by Germany, France, EIRE, and 34 other countries. Orders from the UK represent 91.6% of the total, contributing 85.1% to overall sales. Conversely, orders from other countries constitute 3.7% of orders but contribute 8.4% to total sales. Sales from the UK follow the overall sales pattern, while sales from other countries exhibit a consistent flat trend throughout the year.

Product segmentation analysis shows that products priced under $3 generated the highest sales, with sales decreasing as prices increased.

Among fees and charges, the largest is the Amazon fee, followed by dotcom postage and postage costs.

Regarding total cancellations by value, the UK, as the dominant country, also leads in cancellation value. Apart from the UK, EIRE has the highest cancellation value, followed by Singapore and France. Interestingly, this ranking does not align with their respective sales rankings.