Importing Libraries

```
In [13]:
import pandas as pd
import sqlite3
import numpy as np
import warnings
pd.set_option('display.max_columns',None)
warnings.filterwarnings("ignore")
```

```
In [2]:
data = pd.read_csv("/Users/ankush/Documents/code/Machine_Learning/Unified /Project_1/Amazon Sales data.csv")
data.head()
```

Out[2]:

	Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
0	Australia and Oceania	Tuvalu	Baby Food	Offline	H	5/28/2010	669165933	6/27/2010	9925	255.28	159.42	2533654.00	1582243.50	951410.50
1	Central America and the Caribbean	Grenada	Cereal	Online	C	8/22/2012	963881480	9/15/2012	2804	205.70	117.11	576782.80	328376.44	248406.36
2	Europe	Russia	Office Supplies	Offline	L	5/2/2014	341417157	5/8/2014	1779	651.21	524.96	1158502.59	933903.84	224598.75
3	Sub-Saharan Africa	Sao Tome and Principe	Fruits	Online	C	6/20/2014	514321792	7/5/2014	8102	9.33	6.92	75591.66	56065.84	19525.82
4	Sub-Saharan Africa	Rwanda	Office Supplies	Offline	L	2/1/2013	115456712	2/6/2013	5062	651.21	524.96	3296425.02	2657347.52	639077.50

Sales Performance Data

New SQLite database file named sales_performance.db, store the aggregated sales performance data in a table named sales_performance, execute an SQL query to retrieve the aggregated data, and finally export the query result to a new CSV file named sales_performance_by_region_and_country.csv.

```
In [5]:
# Step 2: Perform Data Aggregation
sales_performance = df.groupby(['Region', 'Country']).agg({
    'Total Revenue': 'sum',
    'Total Cost': 'sum',
    'Total Profit': 'sum'
}).reset_index()

# Step 3: Export Aggregated Data to SQLite Database
conn = sqlite3.connect('sales_performance.db')
sales_performance.to_sql('sales_performance', conn, index=False, if_exists='replace')

# Step 4: Execute SQL Query to Retrieve Aggregated Data
sql_query = """
SELECT Region, Country, SUM("Total Revenue") AS "Total Revenue",
       SUM("Total Cost") AS "Total Cost", SUM("Total Profit") AS "Total Profit"
FROM sales_performance
GROUP BY Region, Country
"""

# Execute the SQL query and load the results into a DataFrame
result_df = pd.read_sql_query(sql_query, conn)

# Step 5: Export Query Result to a New CSV File
result_df.to_csv('sales_performance_by_region_and_country.csv', index=False)

# Close the connection to the SQLite database
conn.close()
```

Product Performance Data

A SQLite database file named product_performance.db, store the aggregated product performance data in a table named product_performance, execute an SQL query to retrieve the aggregated data, and finally export the query result to a new CSV file named product_performance_by_region_and_country.csv.

```
In [7]:
# Step 2: Perform Data Aggregation
product_performance = df.groupby(['Region', 'Country', 'Item Type']).agg({
    'Units Sold': 'sum',
    'Total Revenue': 'sum',
    'Total Cost': 'sum',
    'Total Profit': 'sum'
}).reset_index()

# Step 3: Export Aggregated Data to SQLite Database
conn = sqlite3.connect('product_performance.db')
product_performance.to_sql('product_performance', conn, index=False, if_exists='replace')

# Step 4: Execute SQL Query to Retrieve Aggregated Data
sql_query = """
SELECT Region, Country, "Item Type", SUM("Units Sold") AS "Sales Volume",
       SUM("Total Revenue") AS "Total Revenue", SUM("Total Cost") AS "Total Cost",
       SUM("Total Profit") AS "Total Profit"
FROM product_performance
GROUP BY Region, Country, "Item Type"
"""

# Execute the SQL query and load the results into a DataFrame
result_df = pd.read_sql_query(sql_query, conn)

# Step 5: Export Query Result to a New CSV File
result_df.to_csv('product_performance_by_region_and_country.csv', index=False)

# Close the connection to the SQLite database
conn.close()
```

Sales Channel Analysis

A SQLite database file named sales_channel_analysis.db, store the aggregated sales channel analysis data in a table named sales_channel_analysis, execute an SQL query to retrieve the aggregated data, and finally export the query result to a new CSV file named sales_channel_analysis.csv.

```
In [9]:
# Step 2: Perform Data Aggregation
sales_channel_analysis = df.groupby('Sales Channel').agg({
    'Units Sold': 'sum',
    'Total Revenue': 'sum',
    'Total Cost': 'sum',
    'Total Profit': 'sum'
}).reset_index()

# Step 3: Export Aggregated Data to SQLite Database
conn = sqlite3.connect('sales_channel_analysis.db')
sales_channel_analysis.to_sql('sales_channel_analysis', conn, index=False, if_exists='replace')

# Step 4: Execute SQL Query to Retrieve Aggregated Data
sql_query = """
SELECT "Sales Channel", SUM("Units Sold") AS "Total Units Sold",
       SUM("Total Revenue") AS "Total Revenue", SUM("Total Cost") AS "Total Cost",
       SUM("Total Profit") AS "Total Profit"
FROM sales_channel_analysis
GROUP BY "Sales Channel"
"""

# Execute the SQL query and load the results into a DataFrame
result_df = pd.read_sql_query(sql_query, conn)

# Step 5: Export Query Result to a New CSV File
result_df.to_csv('sales_channel_analysis.csv', index=False)

# Close the connection to the SQLite database
conn.close()
```

Order Timeliness Analysis

Creating a CSV file named order_timeliness_analysis.csv containing information on order IDs, order dates, ship dates, order priorities, and fulfillment times.

```
In [11]:
# Step 2: Extract relevant columns for analysis
order_timeliness_data = df[['Order ID', 'Order Date', 'Ship Date', 'Order Priority']]

# Step 3: Calculate order fulfillment time
order_timeliness_data['Order Date'] = pd.to_datetime(order_timeliness_data['Order Date'])
order_timeliness_data['Ship Date'] = pd.to_datetime(order_timeliness_data['Ship Date'])
order_timeliness_data['Fulfillment Time'] = (order_timeliness_data['Ship Date'] - order_timeliness_data['Order Date']).dt.days

# Step 4: Export data to CSV file
order_timeliness_data.to_csv('order_timeliness_analysis.csv', index=False)

# Step 5: (Optional) Store data in SQLite database
conn = sqlite3.connect('order_timeliness_analysis.db')
order_timeliness_data.to_sql('order_timeliness_analysis', conn, index=False, if_exists='replace')
conn.close()
```

```
/var/folders/jy/jp1snnb92llffd_1pxnt510w0000gn/T/ipykernel_59638/185823077.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  order_timeliness_data['Order Date'] = pd.to_datetime(order_timeliness_data['Order Date'])
/var/folders/jy/jp1snnb92llffd_1pxnt510w0000gn/T/ipykernel_59638/185823077.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  order_timeliness_data['Ship Date'] = pd.to_datetime(order_timeliness_data['Ship Date'])
/var/folders/jy/jp1snnb92llffd_1pxnt510w0000gn/T/ipykernel_59638/185823077.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  order_timeliness_data['Fulfillment Time'] = (order_timeliness_data['Ship Date'] - order_timeliness_data['Order Date']).dt.days
```

Seasonality Trends (monthly)

A CSV file named seasonality_trends_monthly.csv (or seasonality_trends_quarterly.csv

```
In [14]:
# Step 2: Convert 'Order Date' column to datetime format
df['Order Date'] = pd.to_datetime(df['Order Date'])

# Step 3: Extract year and month (or quarter) from the 'Order Date' column
df['Year'] = df['Order Date'].dt.year
df['Month'] = df['Order Date'].dt.month
# Alternatively, for quarterly analysis:
# df['Quarter'] = df['Order Date'].dt.quarter

# Step 4: Perform Data Aggregation
seasonality_trends = df.groupby(['Year', 'Month']).agg({
    'Total Revenue': 'sum',
    'Total Cost': 'sum',
    'Total Profit': 'sum',
    'Units Sold': 'sum'
}).reset_index()

# Step 5: Export Aggregated Data to CSV Files
# For monthly analysis
seasonality_trends.to_csv('seasonality_trends_monthly.csv', index=False)
```