```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('Online_Retail.csv',encoding='latin1')
```

The encoding parameter in pd.read_csv allows you to specify the encoding of the file. By setting it to 'latin1' or 'ISO-8859-1', you can handle a wider range of characters and potentially resolve the decoding issue.

```
data.shape
```

```
(541909, 8)
```

```
data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data.dtypes
```

```
InvoiceNo       object
StockCode       object
Description     object
Quantity         int64
InvoiceDate     object
UnitPrice      float64
CustomerID     float64
Country         object
dtype: object
```

```
data.tail()
```

{"repr_error":"0","type":"dataframe"}

```
data.describe()
```

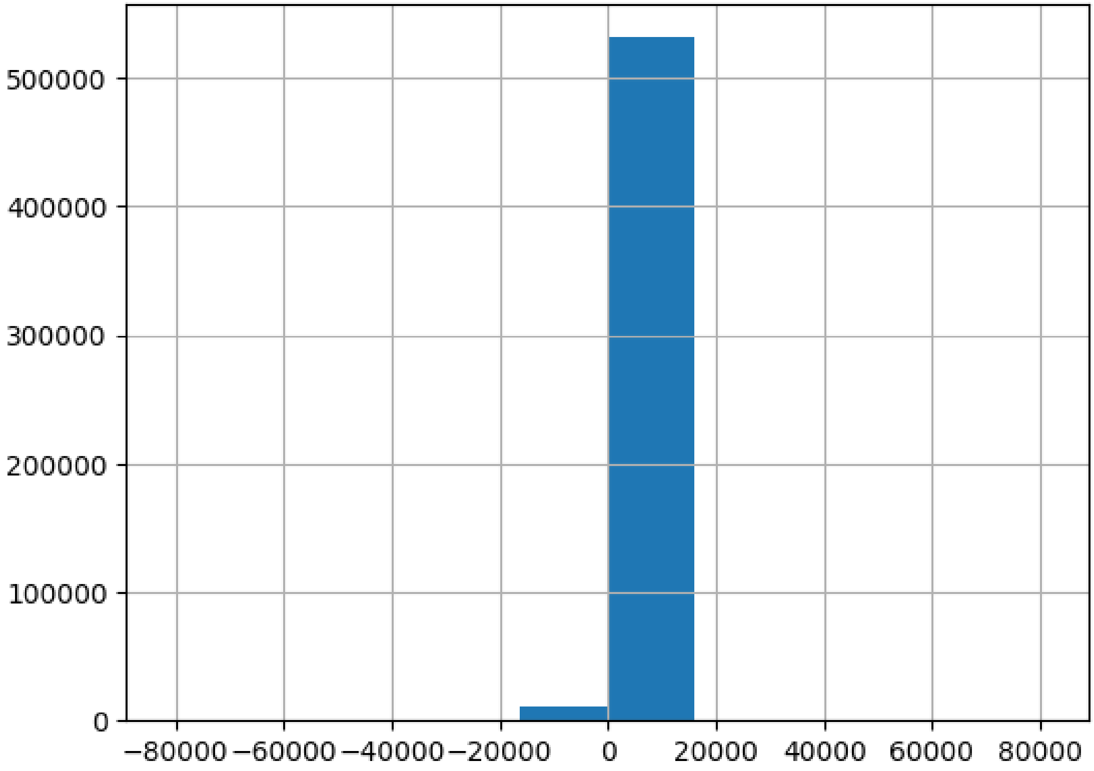
{"summary":"{\n \"name\": \"data\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"Quantity\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 196412.4226608867,\n   \"min\": -80995.0,\n   \"max\": 541909.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   9.55224954743324,\n   3.0,\n   541909.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"UnitPrice\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 190752.0757077193,\n   \"min\": -11062.06,\n   \"max\": 541909.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   4.611113626088513,\n   2.08,\n   541909.0\n   ],\n

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CustomerID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
139204.16800694188,\n        \"min\": 1713.600303321598,\n
\"max\": 406829.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          15287.690570239585,\n          15152.0,\n
406829.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
data.isnull().sum()
```

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

```
import matplotlib.pyplot as plt
data['Quantity'].hist()
plt.show()
```



```
data.dropna()
```

```
{"type":"dataframe"}
```

```
data.fillna(value={'Description': 'WHITE HANGING HEART T-LIGHT
HOLDER'}, inplace=True)
```

```
data.fillna(value={'UnitPrice': '2.95'}, inplace=True)
```

```
data.fillna(value={'Country': 'United Kingdom'}, inplace=True)
```

```
data.isnull().sum()
```

```
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

```
data = data.drop_duplicates()
```

```
data.isnull().sum()
```

```
InvoiceNo           0
StockCode           0
Description         0
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135037
Country             0
dtype: int64
```

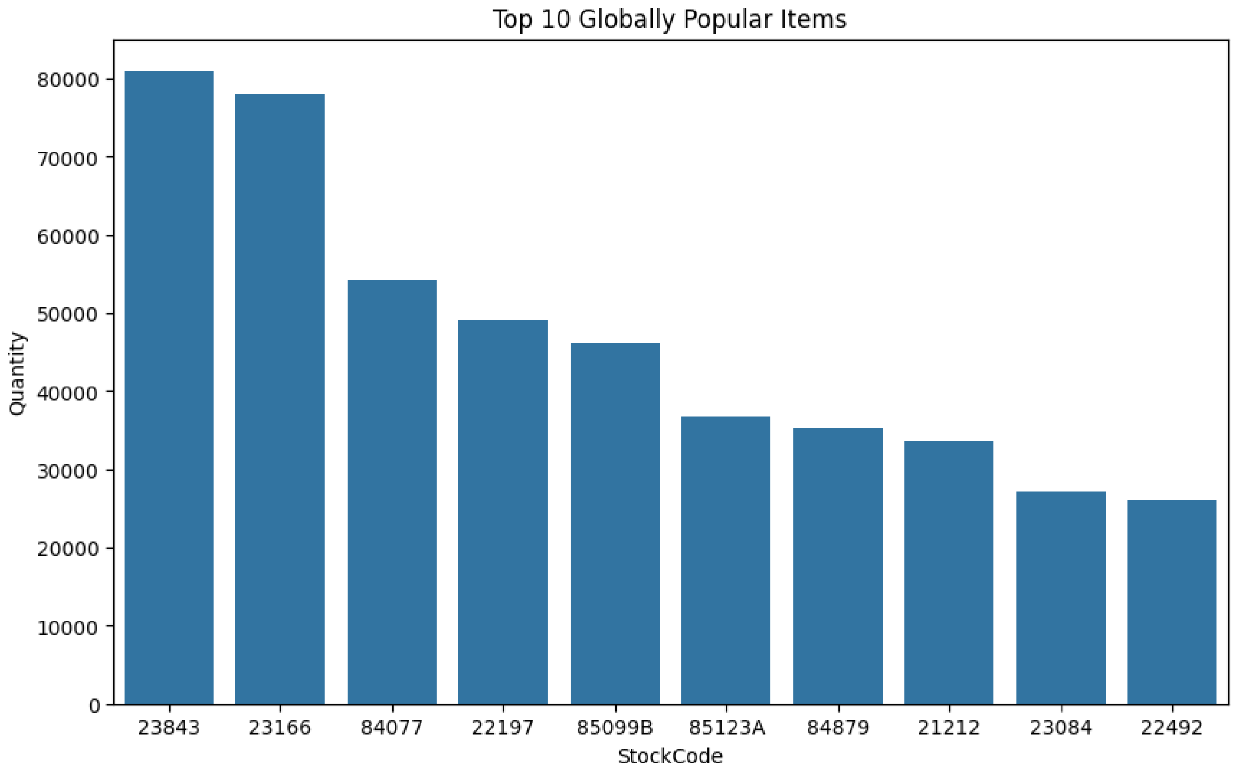
```
# Example: Remove outliers in a numerical column
from scipy import stats
outlier = data[(np.abs(stats.zscore(data['Quantity'])) < 3)]
outlier.count()
```

```
InvoiceNo      536298
StockCode      536298
Description    536298
Quantity       536298
InvoiceDate    536298
UnitPrice      536298
CustomerID     401350
Country        536298
dtype: int64
```

```python
# Data cleaning
data.dropna(subset=['CustomerID'], inplace=True)  # Remove rows with missing CustomerID
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])  # Convert InvoiceDate to datetime
data['TotalPrice'] = data['Quantity'] * data['UnitPrice']  # Calculate total price per row

# Remove negative quantities (returns)
data = data[data['Quantity'] > 0]
```

```
<ipython-input-34-6c5c316158ec>:3: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])  # Convert InvoiceDate to datetime
```

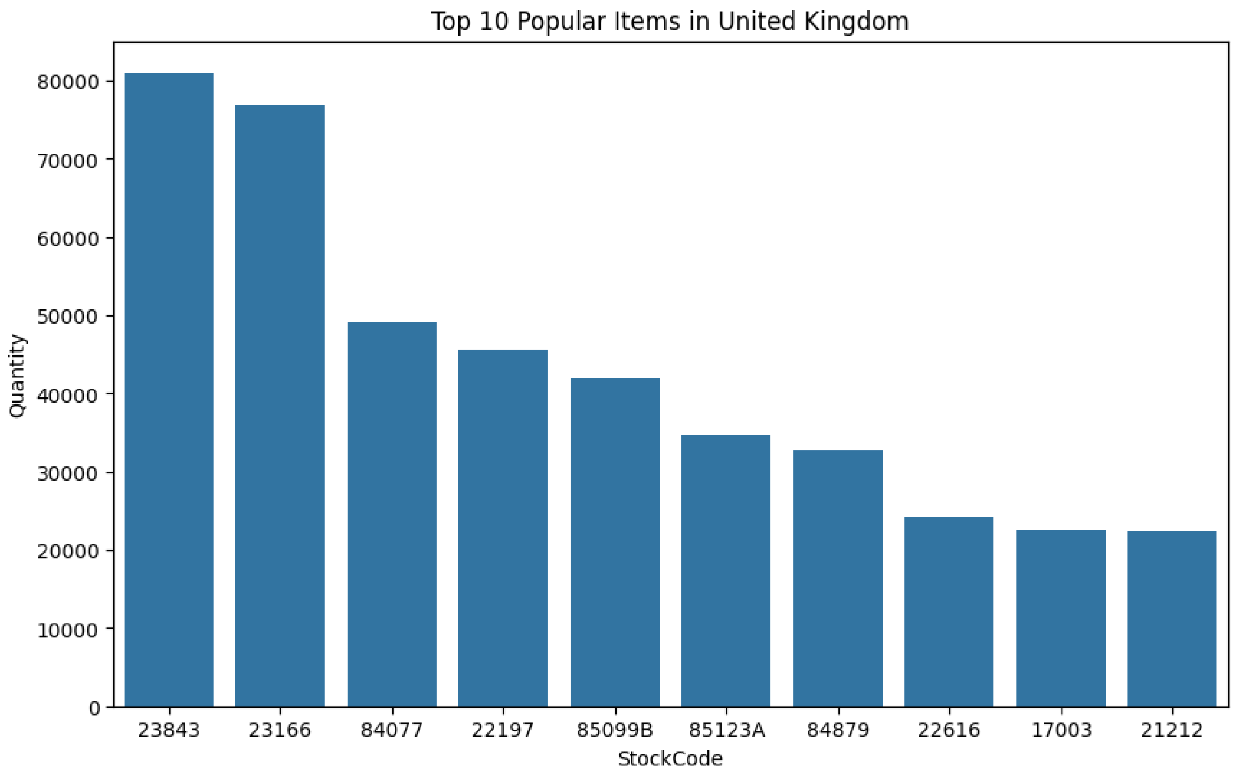
```python
# Calculate total quantity sold per item
popular_items_global = data.groupby('StockCode')['Quantity'].sum().reset_index()
popular_items_global = popular_items_global.sort_values(by='Quantity', ascending=False)

# Plot globally popular items
plt.figure(figsize=(10, 6))
sns.barplot(x='StockCode', y='Quantity', data=popular_items_global.head(10))
plt.title('Top 10 Globally Popular Items')
plt.show()
```



```
popular_items_global.head(10)
```

```
{"summary":"{\n  \"name\": \"popular_items_global\",\n  \"rows\":
3665,\n  \"fields\": [\n    {\n      \"column\": \"StockCode\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3665,\n        \"samples\": [\n
\"22589\",\n          \"20778\",\n          \"35921\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quantity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 3512,\n
\"min\": 1,\n        \"max\": 80995,\n        \"num_unique_values\":
1777,\n        \"samples\": [\n          10522,\n          1163,\n
1422\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"popular_items_global"}
```

```
# Calculate total quantity sold per item per country
popular_items_country = data.groupby(['Country', 'StockCode'])
['Quantity'].sum().reset_index()

# Plot country-wise popular items (for a specific country, e.g.,
'United Kingdom')
country = 'United Kingdom'
plt.figure(figsize=(10, 6))
sns.barplot(x='StockCode', y='Quantity',
data=popular_items_country[popular_items_country['Country'] ==
country].sort_values(by='Quantity', ascending=False).head(10))
plt.title(f'Top 10 Popular Items in {country}')
plt.show()
```

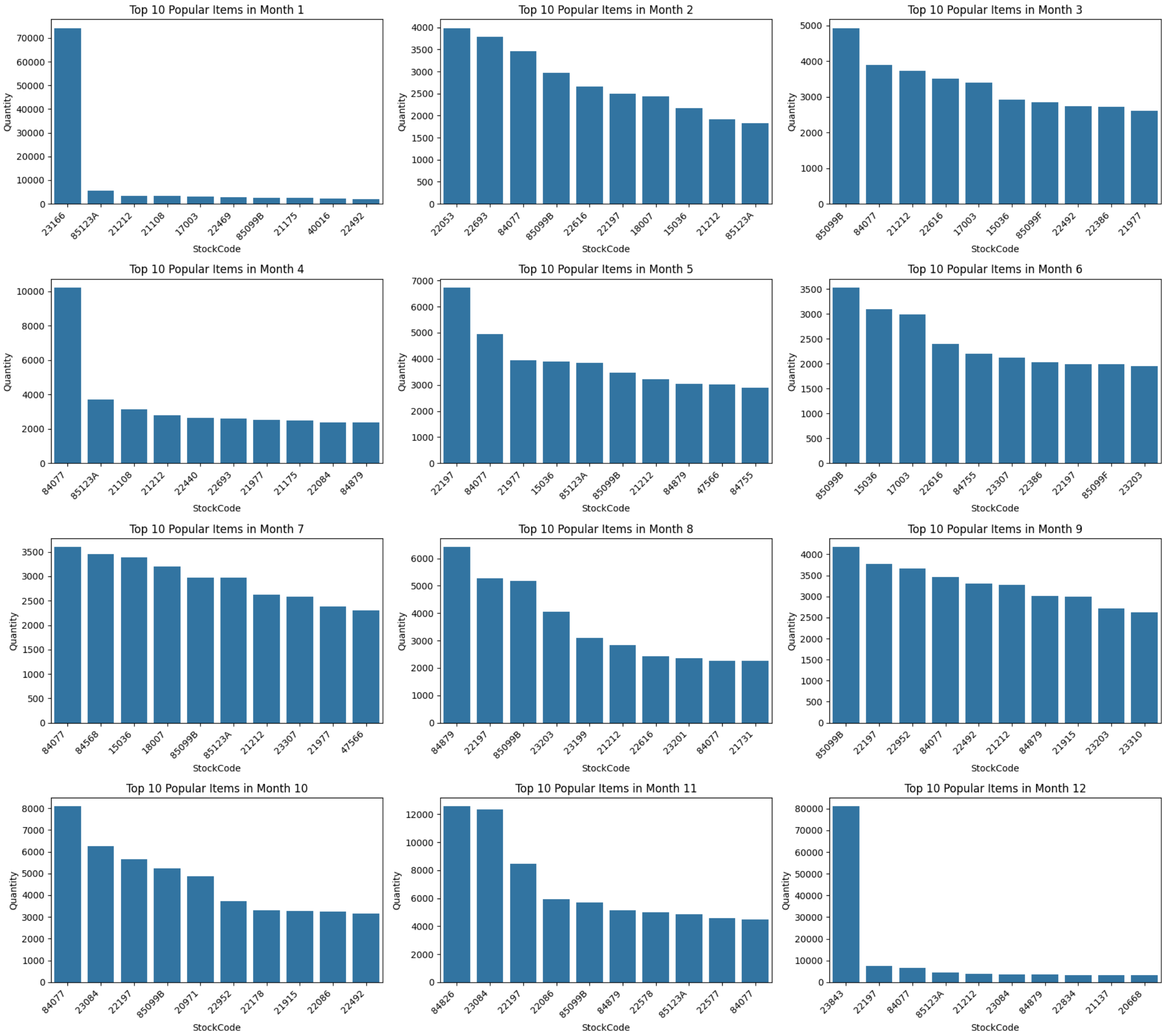


```
popular_items_country.head(10)
```

```
{"summary":"{\n  \"name\": \"popular_items_country\",\n  \"rows\":
18937,\n  \"fields\": [\n    {\n      \"column\": \"Country\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 37,\n        \"samples\": [\n
\"Israel\",\n          \"France\",\n          \"Brazil\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StockCode\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3665,\n        \"samples\": [\n
\"23088\",\n          \"79062D\",\n          \"17091A\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quantity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1462,\n
\"min\": 1,\n        \"max\": 80995,\n        \"num_unique_values\":
1709,\n        \"samples\": [\n          427,\n          6129,\n
8079\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"popular_items_country"}
```

```
# Set up the figure and subplots
fig, axes = plt.subplots(4, 3, figsize=(18, 16))  # 4 rows, 3 columns
for 12 months

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Loop through each month and create a plot
for month in range(1, 13):
    month_data = popular_items_month[popular_items_month['Month'] ==
month].sort_values(by='Quantity', ascending=False).head(10)

    # Plot on the corresponding subplot
    sns.barplot(x='StockCode', y='Quantity', data=month_data,
ax=axes[month-1])
    axes[month-1].set_title(f'Top 10 Popular Items in Month {month}')
    axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

```
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
<ipython-input-45-ab25f4e58b4a>:14: UserWarning: FixedFormatter should
only be used together with FixedLocator
  axes[month-1].set_xticklabels(axes[month-1].get_xticklabels(),
rotation=45, ha="right")
```



```python
# Global pivot table
global_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', aggfunc=np.sum)

# Country-wise pivot table
country_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', columns='Country', aggfunc=np.sum)

# Month-wise pivot table
month_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', columns='Month', aggfunc=np.sum)
```

```
<ipython-input-47-53af19d54a05>:2: FutureWarning: The provided
callable <function sum at 0x7cbad0f12e60> is currently using
DataFrameGroupBy.sum. In a future version of pandas, the provided
callable will be used directly. To keep current behavior pass the
string "sum" instead.
```

```
  global_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', aggfunc=np.sum)
<ipython-input-47-53af19d54a05>:5: FutureWarning: The provided
callable <function sum at 0x7cbad0f12e60> is currently using
DataFrameGroupBy.sum. In a future version of pandas, the provided
callable will be used directly. To keep current behavior pass the
string "sum" instead.
  country_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', columns='Country', aggfunc=np.sum)
<ipython-input-47-53af19d54a05>:8: FutureWarning: The provided
callable <function sum at 0x7cbad0f12e60> is currently using
DataFrameGroupBy.sum. In a future version of pandas, the provided
callable will be used directly. To keep current behavior pass the
string "sum" instead.
  month_pivot = pd.pivot_table(data, values='Quantity',
index='StockCode', columns='Month', aggfunc=np.sum)
```

```
global_pivot.head(10)
```

{"summary":"{\n \"name\": \"global_pivot\",\n \"rows\": 3665,\n
\"fields\": [\n   {\n   \"column\": \"StockCode\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3665,\n   \"samples\": [\n
\"23129\",\n   \"85048\",\n   \"84206B\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Quantity\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 3512,\n
\"min\": 1,\n   \"max\": 80995,\n   \"num_unique_values\":
1777,\n   \"samples\": [\n   4595,\n   375,\n
1294\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"global_pivot"}

```
country_pivot.head(10)
```

{"type":"dataframe","variable_name":"country_pivot"}

```
month_pivot.head(10)
```

{"summary":"{\n \"name\": \"month_pivot\",\n \"rows\": 3665,\n
\"fields\": [\n   {\n   \"column\": \"StockCode\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3665,\n   \"samples\": [\n
\"23129\",\n   \"85048\",\n   \"84206B\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": 1,\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 1633.4376340628114,\n
\"min\": 1.0,\n   \"max\": 74215.0,\n
\"num_unique_values\": 461,\n   \"samples\": [\n   384.0,\
n   87.0,\n   115.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 272.7258061972111,\n
\"min\": 1.0,\n        \"max\": 3986.0,\n
\"num_unique_values\": 442,\n        \"samples\": [\n          304.0,\
n          104.0,\n          55.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 342.60637782288114,\n
\"min\": 1.0,\n        \"max\": 4924.0,\n
\"num_unique_values\": 513,\n        \"samples\": [\n          226.0,\
n          1789.0,\n          392.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 4,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 352.9037921697623,\n
\"min\": 1.0,\n        \"max\": 10224.0,\n
\"num_unique_values\": 458,\n        \"samples\": [\n          304.0,\
n          120.0,\n          252.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 5,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 389.45442929121197,\n
\"min\": 1.0,\n        \"max\": 6730.0,\n
\"num_unique_values\": 541,\n        \"samples\": [\n          314.0,\
n          937.0,\n          142.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 6,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 309.7548528189728,\n
\"min\": 1.0,\n        \"max\": 3529.0,\n
\"num_unique_values\": 529,\n        \"samples\": [\n          17.0,\n
770.0,\n          1620.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": 7,\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 326.3415066734399,\n        \"min\":
1.0,\n        \"max\": 3600.0,\n        \"num_unique_values\": 525,\n
\"samples\": [\n          528.0,\n          1081.0,\n          89.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": 8,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 370.4388868359878,\n
\"min\": 1.0,\n        \"max\": 6417.0,\n
\"num_unique_values\": 546,\n        \"samples\": [\n          203.0,\
n          1447.0,\n          175.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 9,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 389.757236624454,\n
\"min\": 1.0,\n        \"max\": 4175.0,\n
\"num_unique_values\": 676,\n        \"samples\": [\n          756.0,\
n          238.0,\n          888.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 10,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 449.8987700057313,\n
\"min\": 1.0,\n         \"max\": 8078.0,\n
\"num_unique_values\": 700,\n        \"samples\": [\n          929.0,\
n          826.0,\n          3168.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 11,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 599.5395475274992,\n
\"min\": 1.0,\n         \"max\": 12551.0,\n
\"num_unique_values\": 732,\n        \"samples\": [\n          442.0,\
n          1625.0,\n          118.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": 12,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1508.1785360126798,\n
\"min\": 1.0,\n         \"max\": 80995.0,\n
\"num_unique_values\": 611,\n        \"samples\": [\n          210.0,\
n          575.0,\n          1728.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"month_pivot"}
```

```python
# Function to recommend items based on similar users
def recommend_items(customer_id, num_recommendations=5):
    customer_data = data[data['CustomerID'] == customer_id]
    customer_bought = customer_data['StockCode'].unique()

    # Calculate similarity
    item_matrix = pd.pivot_table(data, values='Quantity',
index='CustomerID', columns='StockCode', aggfunc=np.sum, fill_value=0)
    item_similarity = cosine_similarity(item_matrix)
    sim_df = pd.DataFrame(item_similarity, index=item_matrix.index,
columns=item_matrix.index)

    # Check if customer_id is in the index
    if customer_id not in sim_df.index:
        print(f"Customer ID {customer_id} not found.")
        return []  # Return an empty list if customer ID is not found

    # Find similar users
    similar_users =
sim_df[customer_id].sort_values(ascending=False).index[1:num_recommend
ations+1]

    # Recommend items that similar users bought, including
descriptions
    recommended_items = []
    for item in data[data['CustomerID'].isin(similar_users)]
['StockCode'].value_counts().index:
        if item not in customer_bought:
            item_description = data[data['StockCode'] == item]
['Description'].iloc[0]
            recommended_items.append((item, item_description))
            if len(recommended_items) >= num_recommendations:
                break

    return recommended_items
```

```
<ipython-input-59-4ec79ac21413>:9: FutureWarning: The provided
callable <function sum at 0x7cbad0f12e60> is currently using
DataFrameGroupBy.sum. In a future version of pandas, the provided
callable will be used directly. To keep current behavior pass the
string "sum" instead.
  item_matrix = pd.pivot_table(data, values='Quantity',
index='CustomerID', columns='StockCode', aggfunc=np.sum, fill_value=0)
```

```
Recommended items for customer 12583:
  - Item ID: 22382, Description: LUNCH BAG SPACEBOY DESIGN
  - Item ID: 22630, Description: DOLLY GIRL LUNCH BOX
  - Item ID: 22662, Description: LUNCH BAG DOLLY GIRL DESIGN
  - Item ID: 84596B, Description: SMALL DOLLY MIX DESIGN ORANGE BOWL
  - Item ID: 22139, Description: RETROSPOT TEA SET CERAMIC 11 PC
```

```
# Example of how to use the recommendation function
customer_id = 12583  # Replace with an actual CustomerID
recommendations = recommend_items(customer_id)
print(f"Recommended items for customer {customer_id}:")
for item, description in recommendations:
    print(f"  - Item ID: {item}, Description: {description}")
```

```
<ipython-input-59-4ec79ac21413>:9: FutureWarning: The provided
callable <function sum at 0x7cbad0f12e60> is currently using
DataFrameGroupBy.sum. In a future version of pandas, the provided
callable will be used directly. To keep current behavior pass the
string "sum" instead.
  item_matrix = pd.pivot_table(data, values='Quantity',
index='CustomerID', columns='StockCode', aggfunc=np.sum, fill_value=0)
```

```
Recommended items for customer 12583:
  - Item ID: 22382, Description: LUNCH BAG SPACEBOY DESIGN
  - Item ID: 22630, Description: DOLLY GIRL LUNCH BOX
  - Item ID: 22662, Description: LUNCH BAG DOLLY GIRL DESIGN
  - Item ID: 84596B, Description: SMALL DOLLY MIX DESIGN ORANGE BOWL
  - Item ID: 22139, Description: RETROSPOT TEA SET CERAMIC 11 PC
```

```
# Function to predict future purchases based on past data
def predict_future_purchases(customer_id, num_predictions=5):
    customer_data = data[data['CustomerID'] == customer_id]
    future_purchases = customer_data.groupby('StockCode')
['Quantity'].sum().sort_values(ascending=False).head(num_predictions).
index
    return future_purchases
```

```python
# Example of how to use the prediction function
print(f"Predicted future purchases for customer {customer_id}:
{predict_future_purchases(customer_id)}")
```

```
Predicted future purchases for customer 12583: Index(['22492',
'22610', '16218', '22609', '21883'], dtype='object', name='StockCode')
```