```
import pandas as pd

# It's data loading time! Let's get our hands on that sweet online retail data.
file_path = "/content/sample_data/online_retail_II.xlsx"

# Directly read the file (Excel format) - pandas makes this a breeze!
df = pd.read_excel(file_path)

# Peek at first few rows - always good to see what we're working with!
print(" Behold! The first 5 rows of our dataset:")
print(df.head())

# Let's see the dimensions of our data treasure chest!
print("\n shape of dataset:", df.shape)
# And what shiny columns does it contain?
print("\n Column names:", df .columns. tolist())
```

```
 Behold! The first 5 rows of our dataset:
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

          InvoiceDate  Price  Customer ID         Country
0 2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25      13085.0  United Kingdom

 shape of dataset: (525461, 8)

 Column names: ['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country']
```

```
# Time to clean up our data! Let's make it sparkling clean for analysis.

# Check for missing values - where are the gaps in our data story?
print("Checking for missing values:")
print(df.isnull().sum())

# Remove duplicate rows - no room for repeats in our dataset!
df = df.drop_duplicates()
print("\nRemoving duplicate rows. New shape:", df.shape)

# Remove rows with negative or zero Quantity or Price - only valid transactions here!
df = df[(df['Quantity']>0) & (df['Price']>0)]
print("\nRemoving rows with non-positive Quantity or Price. New shape:", df.shape)

# Convert InvoiceDate to datetime - making sure our dates are in the right format
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])
print("\nConverting InvoiceDate to datetime.")

# Resetting the index - giving our cleaned data a fresh start with the index
df = df.reset_index(drop=True)
print("\nResetting index.")

# Removing cancelled transactions - saying goodbye to cancelled orders
df = df[~df['Invoice'].str.contains('C', na=False)]
print("\nRemoving cancelled transactions. New shape:", df.shape)

# removing negative sales - already done above, but double checking (belt and suspenders!)
df = df[(df['Quantity'] >0) & (df['Price'] >0)]
print("\nRemoving negative sales (double check!). New shape:", df.shape)

# Removing outliers - keeping our data grounded by removing extreme values
df = df[(df['Quantity'] < 1000) & (df['Price'] < 1000)]
print("\nRemoving outliers (Quantity < 1000 and Price < 1000). New shape:", df.shape)


# Printing cleaned data - behold the beauty of clean data!
print("\nBehold! The first 10 rows of our cleaned dataset:")
```

```
print(df.head(10))
print("\nShape of the cleaned dataset:", df.shape)
print("\nDescriptive statistics of the cleaned dataset:")
print(df.describe())
```

```
Checking for missing values:
Invoice             0
StockCode           0
Description      2928
Quantity            0
InvoiceDate         0
Price               0
Customer ID    107927
Country             0
dtype: int64

Removing duplicate rows. New shape: (518596, 8)

Removing rows with non-positive Quantity or Price. New shape: (504731, 8)

Converting InvoiceDate to datetime.

Resetting index.

Removing cancelled transactions. New shape: (504730, 8)

Removing negative sales (double check!). New shape: (504730, 8)

Removing outliers (Quantity < 1000 and Price < 1000). New shape: (504456, 8)

Behold! The first 10 rows of our cleaned dataset:
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24
5  489434     22064           PINK DOUGHNUT TRINKET POT         24
6  489434     21871                   SAVE THE PLANET MUG        24
7  489434     21523   FANCY FONT HOME SWEET HOME DOORMAT        10
8  489435     22350                      CAT BOWL               12
9  489435     22349       DOG BOWL , CHASING BALL DESIGN        12

           InvoiceDate  Price  Customer ID         Country
0 2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
5 2009-12-01 07:45:00   1.65      13085.0  United Kingdom
6 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
7 2009-12-01 07:45:00   5.95      13085.0  United Kingdom
8 2009-12-01 07:46:00   2.55      13085.0  United Kingdom
9 2009-12-01 07:46:00   3.75      13085.0  United Kingdom

Shape of the cleaned dataset: (504456, 8)

Descriptive statistics of the cleaned dataset:
            Quantity                    InvoiceDate          Price  \
count  504456.000000                         504456  504456.000000
mean       10.266919  2010-06-28 17:28:03.438198784       3.804684
min         1.000000            2009-12-01 07:45:00       0.001000
25%         1.000000            2010-03-21 14:07:00       1.250000
```

```
# Time for a quick peek into our data's soul!

print("Number of Rows:", len(df)) # How many transactions are we looking at?
print("Number of Columns:", len(df.columns)) # How many details do we have for each transaction?
print("Column Names:", df.columns.tolist()) # What are the names of those details?

# Descriptive statistics - a summary of our numerical data
print("\nDescriptive statistics of the cleaned dataset:")
print(df.describe())

# Unique customers and products - who is buying and what are they buying?

unique_customers = df['Customer ID'].nunique()
unique_products = df['StockCode'].nunique()
```

```
print("\nNumber of Unique Customers:", unique_customers) # How many different customers do we have?
print("Number of Unique Products:", unique_products) # How many different items are being sold?

# Total sales per month - let's see which months are the champions!
df['Revenue'] = df['Quantity'] * df['Price'] # Calculate revenue for each transaction
df['Month'] = df['InvoiceDate'].dt.month # Extract the month from the invoice date
monthly_sales = df.groupby('Month')['Revenue'].sum() # Group by month and sum the revenue
print("\nMonthly Sales:")
print(monthly_sales)

# Top 10 products by quantity - what are the crowd favorites?
top_products = df.groupby('Description')['Quantity'].sum().nlargest(10) # Group by product description and find the top 10 by total q
print("\nTop 10 Products by Quantity Sold:")
print(top_products)

# Top 10 customers by revenue - who are our biggest spenders?
top_customers = df.groupby('Customer ID')['Revenue'].sum().nlargest(10) # Group by customer and find the top 10 by total revenue
print("\nTop 10 Customers by Revenue:")
print(top_customers)
```

```
Number of Rows: 504456
Number of Columns: 8
Column Names: ['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country']

Descriptive statistics of the cleaned dataset:
            Quantity                    InvoiceDate          Price  \
count  504456.000000                         504456  504456.000000
mean       10.266919  2010-06-28 17:28:03.438198784       3.804684
min         1.000000            2009-12-01 07:45:00       0.001000
25%         1.000000            2010-03-21 14:07:00       1.250000
50%         3.000000            2010-07-06 14:25:00       2.100000
75%        12.000000            2010-10-15 13:39:00       4.210000
max       992.000000            2010-12-09 20:01:00     975.110000
std        30.196644                            NaN      12.812008

         Customer ID
count  400684.000000
mean    15361.737259
min     12346.000000
25%     13985.000000
50%     15311.000000
75%     16805.000000
max     18287.000000
std      1680.581785

Number of Unique Customers: 4304
Number of Unique Products: 4250

Monthly Sales:
Month
1      627216.652
2      530677.716
3      762447.941
4      659789.402
5      636625.600
6      701880.250
7      645199.770
8      674118.560
9      862895.381
10    1098214.850
11    1418487.502
12    1225316.360
Name: Revenue, dtype: float64

Top 10 Products by Quantity Sold:
Description
WHITE HANGING HEART T-LIGHT HOLDER     58691
PACK OF 72 RETRO SPOT CAKE CASES       46728
ASSORTED COLOUR BIRD ORNAMENT          41108
60 TEATIME FAIRY CAKE CASES            35148
WORLD WAR 2 GLIDERS ASSTD DESIGNS      34451
PACK OF 60 PINK PAISLEY CAKE CASES     31805
JUMBO BAG RED RETROSPOT                30746
STRAWBERRY CERAMIC TRINKET BOX         27059
PACK OF 72 SKULL CAKE CASES            24194
COLOUR GLASS T-LIGHT HOLDER HANGING    22862
Name: Quantity, dtype: int64
```

```
# Create a TotalSales column - because who doesn't love seeing the money roll in?
df['TotalSales'] = df['Quantity'] * df['Price']

# Convert InvoiceDate to datetime - making sure our dates are in the right format
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

# Group by Customer ID - let's see what each customer is up to!
customer_behavior = df.groupby('Customer ID').agg({
    'InvoiceDate': 'max',   # When did they last grace us with their presence?
    'Invoice': 'count',   # How often do they shop? The more the merrier!
    'TotalSales': 'sum'     # How much treasure have they spent?
}).reset_index()

# Rename for clarity - giving our columns some snazzy names
customer_behavior.rename(columns={
    'InvoiceDate': 'LastPurchaseDate',
    'Invoice': 'Frequency',
    'TotalSales': 'Monetary'
}, inplace=True)

# Find the latest date in the dataset - our reference point for recency
latest_date = df['InvoiceDate'].max()

# Calculate Recency (days since last purchase) - how long has it been? We miss them!
customer_behavior['Recency'] = (latest_date - customer_behavior['LastPurchaseDate']).dt.days

# Keep only useful columns - decluttering time!
customer_behavior = customer_behavior[['Customer ID', 'Recency', 'Frequency', 'Monetary']]

# Behold the customer behavior summary!
print("Customer Behavior Summary (a peek at our loyal subjects):")
print(customer_behavior.head())
```

```
Customer Behavior Summary (a peek at our loyal subjects):
   Customer ID  Recency  Frequency  Monetary
0      12346.0      164         33    372.86
1      12347.0        2         71   1323.32
2      12348.0       73         20    222.16
3      12349.0       42        102   2671.14
4      12351.0       10         21    300.93
```

```
# SEGMENTING CUSTOMERS – BEGINNER FRIENDLY VERSION

# Step 1: Calculate averages for Recency, Frequency, and Monetary
# These will be our benchmarks – customers above/below these get different nicknames.

rfm = customer_behavior

avg_recency = rfm['Recency'].mean()
avg_frequency = rfm['Frequency'].mean()
avg_monetary = rfm['Monetary'].mean()

# Step 2: Create a function to assign each customer to a segment
def segment_customer(row):
    # VIP Shoppers: Buy often AND spend big
    if row['Frequency'] >= avg_frequency and row['Monetary'] >= avg_monetary:
        return "VIP Shoppers"

    # Loyal Fans: Shop very often, but may not spend a lot each time
    elif row['Frequency'] >= avg_frequency:
        return "Loyal Fans"

    # Big Spenders: Spend a lot when they do shop, but not very frequent
    elif row['Monetary'] >= avg_monetary:
        return "Big Spenders"

    # Sleeping Beauties: Haven’t shopped in a long time (we miss them!)
    elif row['Recency'] > avg_recency:
        return "Sleeping Beauties"
```

```
        # Regulars: Average customers who keep the business alive
        else:
            return "Regulars"

    # Step 3: Apply the function to our RFM table
    rfm['Segment'] = rfm.apply(segment_customer, axis=1)

    # Step 4: Count how many customers fall into each segment
    print(rfm['Segment'].value_counts())
```

```
Segment
Regulars             1680
Sleeping Beauties    1264
VIP Shoppers          739
Loyal Fans            429
Big Spenders          192
Name: count, dtype: int64
```

```
# Question 1
# Calculate revenue
df['Revenue'] = df['Quantity'] * df['Price']

# Revenue by country
country_revenue = df.groupby('Country')['Revenue'].sum().sort_values(ascending=False)

# Time to reveal the top revenue-generating countries!
print("🥁 Drumroll please! Our top 10 revenue champions by Country are:")
print(country_revenue.head(10))  # Top 10 countries
print("\nLooks like some countries are really loving our products! 😊")
```

```
🥁 Drumroll please! Our top 10 revenue champions by Country are:
Country
United Kingdom     8466461.983
EIRE                372817.270
Netherlands         268784.350
Germany             202025.391
France              132014.890
Sweden               52234.790
Spain                47568.650
Switzerland          43921.390
Australia            31446.800
Channel Islands      24546.320
Name: Revenue, dtype: float64

Looks like some countries are really loving our products! 😊
```

```
#Question 2
# Revenue by customer
customer_revenue = df.groupby('Customer ID')['Revenue'].sum().sort_values(ascending=False)

print("🌟 Behold the big spenders! Our top 10 customers by revenue are: 💰")
print(customer_revenue.head(10))  # Top 10 customers
print("\nThese customers are truly golden! ✨")
```

```
🌟 Behold the big spenders! Our top 10 customers by revenue are: 💰
Customer ID
18102.0    344507.39
14646.0    248396.50
14156.0    188457.44
14911.0    152121.22
13694.0    130096.79
17511.0     84541.17
15061.0     83284.38
16684.0     79659.77
13089.0     57885.45
15311.0     55810.74
Name: Revenue, dtype: float64

These customers are truly golden! ✨
```

```
# Question 3
# Total quantity sold per product
product_sales = df.groupby('Description')['Quantity'].sum().sort_values(ascending=False)

print("🚀 Flying off the shelves! Our top 10 bestsellers by quantity are: ")
```

```
print(product_sales.head(10))  # Top 10 bestselling products
print("\nThese products are definitely customer favorites! 👍")
```

```
🚀 Flying off the shelves! Our top 10 bestsellers by quantity are:
Description
WHITE HANGING HEART T-LIGHT HOLDER     58691
PACK OF 72 RETRO SPOT CAKE CASES       46728
ASSORTED COLOUR BIRD ORNAMENT          41108
60 TEATIME FAIRY CAKE CASES            35148
WORLD WAR 2 GLIDERS ASSTD DESIGNS      34451
PACK OF 60 PINK PAISLEY CAKE CASES     31805
JUMBO BAG RED RETROSPOT                30746
STRAWBERRY CERAMIC TRINKET BOX         27059
PACK OF 72 SKULL CAKE CASES            24194
COLOUR GLASS T-LIGHT HOLDER HANGING    22862
Name: Quantity, dtype: int64

These products are definitely customer favorites! 👍
```

```
# Question 4

import calendar

# Extract month and weekday
df['Month'] = df['InvoiceDate'].dt.month
df['Weekday'] = df['InvoiceDate'].dt.day_name()

# Map month numbers to names
df['MonthName'] = df['Month'].apply(lambda x: calendar.month_name[x])

# Revenue by month (ordered from Jan–Dec)
monthly_sales = df.groupby('MonthName', sort=False)['Revenue'].sum()
monthly_sales = monthly_sales.reindex(calendar.month_name[1:])

print("📊 Monthly Revenue Breakdown: See which months brought in the most cash! 💸")
print(monthly_sales)
print("\nLooks like some months are real powerhouses! 💪")

# Set weekday order manually (Monday → Sunday)
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df['Weekday'] = pd.Categorical(df['Weekday'], categories=weekday_order, ordered=True)

# Revenue by weekday
weekday_sales = df.groupby('Weekday')['Revenue'].sum()

print("\n🗓️ Revenue by Day of the Week: Which days are our busiest? 🏃‍♂️🏃‍♂️")
print(weekday_sales)
print("\nInteresting to see the daily trends! 🤔")
```

```
📊 Monthly Revenue Breakdown: See which months brought in the most cash! 💸
MonthName
January       627216.652
February      530677.716
March         762447.941
April         659789.402
May           636625.600
June          701880.250
July          645199.770
August        674118.560
September     862895.381
October      1098214.850
November     1418487.502
December     1225316.360
Name: Revenue, dtype: float64

Looks like some months are real powerhouses! 💪

🗓️ Revenue by Day of the Week: Which days are our busiest? 🏃‍♂️🏃‍♂️
Weekday
Monday       1776137.805
Tuesday      1888921.151
Wednesday    1717933.903
Thursday     1964890.422
Friday       1462900.922
Saturday        9803.050
Sunday       1022282.731
```

```
Name: Revenue, dtype: float64

Interesting to see the daily trends! 🤔
/tmp/ipython-input-1326744640.py:25: FutureWarning: The default of observed=False is deprecated and will be changed to True in a
  weekday_sales = df.groupby('Weekday')['Revenue'].sum()
```

```
# Question 5
# Separate cancelled and valid orders
cancelled = df[df['Quantity'] < 0]
valid = df[df['Quantity'] > 0]

# Revenue lost from cancellations
cancelled_revenue = cancelled['Revenue'].sum()
valid_revenue = valid['Revenue'].sum()

print("Valid Revenue:", valid_revenue)
print("Cancelled Revenue:", cancelled_revenue)
print("Cancellation %:", (abs(cancelled_revenue) / valid_revenue) * 100)
```

```
Valid Revenue: 9842869.984000001
Cancelled Revenue: 0.0
Cancellation %: 0.0
```