School of Information Technologies and Engineering, ADA University

INFT4836 – Intro to Big Data Analytics

Fall 2025 – 11/29/2025

Assignment 2 by Laman Panakhova 16882

Apache Spark

Instructions:

Overview

*In this assignment, you will apply the concepts learned in the course by working with **Apache Spark**, including **RDDs**, **Spark SQL**, **DataFrames**, and an introduction to **basic data preprocessing / feature engineering** using Spark. Your goal is to work with a real-world large dataset and demonstrate your understanding of distributed data processing, querying, and transformation.*

*You will complete **three major tasks** using the **same dataset** throughout the assignment. These tasks will allow you to practice data manipulation with RDDs, build SQL queries over distributed datasets, and perform more advanced operations using DataFrames.*

*Please submit a **single PDF file** containing all answers, code snippets, outputs (screenshots), figures, and explanations.*

Assignment Tasks

Task 1: Spark RDDs – Core Transformations & Actions (40 points)

Dataset Selection (no points, required)

Choose a dataset from a domain of your choice (healthcare, finance, IoT, cybersecurity, retail, sports analytics, transportation, or others). Your dataset should:

*Be at least **100,000 rows** or reasonably large*

*Contain **multiple features** (numeric + categorical preferred)*

Be obtained from a reliable source (Kaggle, UCI ML Repository, Open Data Portal, etc.)

Setup (5 points)

Install and configure Apache Spark on your local machine.

Load your dataset using standard Spark methods (textFile, wholeTextFiles, or csv reader).

Provide screenshots verifying successful installation and dataset loading.

RDD Transformations & Actions (35 points)

*Using Spark RDDs, perform the following operations **with explanations of why each step is meaningful for your dataset**:*

***1. map() transformation** – derive a new feature or convert raw entries into structured tuples (7 pts)*

***2. filter() transformation** – remove invalid / missing / noisy records (7 pts)*

***3. flatMap() transformation** – break complex fields (e.g., text, lists, logs) into smaller elements (7 pts)*

***4. reduceByKey()** or **aggregate()** – compute summary statistics or grouped aggregates (7 pts)*

***5. take()** or **collect()** (action) – extract sample outputs and comment on distributed execution (7 pts)*

Include relevant code, outputs, and short justifications for each transformation.

Task 2: Spark SQL – Querying Distributed Data (30 points)

*Load the same dataset using Spark SQL and create a **temporary view**.*
Using Spark SQL, perform and document the following:
1. Aggregation Query (10 pts) Example: average value by category, count by class label, sum by month, etc.
2. Filtering Query (10 pts) Use a WHERE clause with logical operators or range conditions.
*3. Join Query (10 pts) - Create (or simulate) a second small helper table (e.g., category descriptions, lookup table) and perform an **INNER** or **LEFT JOIN**. Explain why the join is meaningful for analysis. Include SQL query text, results, and screenshots/output tables.*

Task 3: Spark DataFrames – Data Cleaning & Transformations (30 points)
Use Spark DataFrames to explore and preprocess your dataset.
Perform the following DataFrame operations:
1. Data Cleaning: Handling Missing or Incorrect Values (10 pts) - Demonstrate .dropna(), .fillna(), or custom cleaning logic. Explain why the chosen strategy is appropriate.

2. groupBy() + agg() operations (10 pts) - Compute multiple aggregate functions (e.g., mean, stddev, count) on at least one categorical field.
3. orderBy() / sort() with multiple columns (10 pts) - Show sorting results and comment on distributed execution.
Include code, results, and interpretations for each step.

Solutions:

In this assignment we focus on using Apache Spark to process the *Online Retail II* dataset using three different APIs: RDDs, Spark SQL, and DataFrames. The goal is to demonstrate understanding of data loading, cleaning, transformation, querying, and performing basic analytical tasks at scale. All tasks were completed in a local PySpark environment.

Description of the dataset:

The dataset used in this project is the Online Retail II dataset, containing real transaction records from a UK-based online retail company. Each row represents a single product purchased within an invoice, with details such *as Invoice ID, StockCode, Description, Quantity, InvoiceDate, Price, Customer ID,* and *Country*. This dataset is commonly used in data analytics and e-commerce research to study customer behavior, analyze sales trends, forecast demand, and practice data cleaning and processing with real-world transactional data. Its structure allows exploration of revenue patterns, customer segmentation, and other insights relevant to online retail operations.

You can access the dataset here: https://www.kaggle.com/datasets/mashlyn/online-retail-ii-uci

The main criteria for choosing the dataset were below per assignment instruction:

- ✓ Be at least 100,000 rows or reasonably large,
- ✓ Contain multiple features (numeric + categorical preferred),
- ✓ Be obtained from a reliable source (Kaggle, UCI ML Repository, Open Data Portal, etc.)

Task 1: Spark RDD - Core Transformations & Actions

To begin the analysis, Apache Spark was installed and configured on the local machine to provide a scalable environment for handling large datasets. The dataset was loaded using Spark's standard CSV reader, which efficiently reads structured data into a DataFrame for processing. Spark's methods allow filtering, transformation, and aggregation on the dataset with high performance, even on modest hardware. Screenshots of the Spark session and the loaded dataset verify successful installation and loading, demonstrating that the environment is ready for further data analysis.

```
!pip install pyspark


Requirement already satisfied: pyspark in /usr/local/lib/python3.12/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark) (0.10.9.7)

!pip install pyspark
from pyspark import SparkContext
sc = SparkContext()

print(sc.parallelize([1,2,3]).take(3))


Requirement already satisfied: pyspark in /usr/local/lib/python3.12/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.12/dist-packages (from pyspark) (0.10.9.7)
[1, 2, 3]

import pyspark
print(pyspark.__version__)


3.5.1
```


```
# Imports and version check
import pyspark
print(pyspark.__version__)

# Cell: common setup
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_timestamp, month, year, when, sum as spark_sum, avg, stddev, count
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, DoubleType, TimestampType

# create spark session
spark = SparkSession.builder \
    .appName("NFT4836_Hw2_OnlineRetailIliI") \
    .master("local[1*]") \
    .config("spark.driver.memory", "6G") \
    .getOrCreate()

sc = spark.sparkContext

print("Spark version:", spark.version)

3.5.1
Spark version: 3.5.1
```

Furthermore, the dataset was loaded, and several required transformations were implemented using both RDD operations and their equivalent DataFrame APIs.

- ✓ ***Dataset Loading***

The CSV file was read with header and schema inference enabled. A sample of the dataset was inspected using .show() to confirm successful loading.

```
# Cell: load CSV as DataFrame (with inferSchema disabled initially then explicit schema)
file_path = r"/content/online_retail_II.csv"

# If CSV has header and default separators, read with Spark CSV reader:
df_raw = spark.read.option("header", True).option("inferSchema", True).option("multiLine", True).option("escape", "\
print("Columns:", df_raw.columns)
print("Count (rows):", df_raw.count())
df_raw.show(5, truncate=False)
```

```
Columns: ['Invoice', 'StockCode', 'Description', 'Quantity', 'InvoiceDate', 'Price', 'Customer ID', 'Country']
Count (rows): 259454
+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+
|Invoice|StockCode|Description                        |Quantity|InvoiceDate        |Price|Customer ID|Country       |
+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+
|489434 |85048    |15CM CHRISTMAS GLASS BALL 20 LIGHTS|12      |2009-12-01 07:45:00|6.95 |13085.0    |United Kingdom|
|489434 |79323P   |PINK CHERRY LIGHTS                 |12      |2009-12-01 07:45:00|6.75 |13085.0    |United Kingdom|
|489434 |79323W   | WHITE CHERRY LIGHTS               |12      |2009-12-01 07:45:00|6.75 |13085.0    |United Kingdom|
|489434 |22041    |RECORD FRAME 7" SINGLE SIZE        |48      |2009-12-01 07:45:00|2.1  |13085.0    |United Kingdom|
|489434 |21232    |STRAWBERRY CERAMIC TRINKET BOX     |24      |2009-12-01 07:45:00|1.25 |13085.0    |United Kingdom|
+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+
only showing top 5 rows
```

After doing some preprocessing steps on data let's continue with below transformations:

- ✓ ***map() transformation – derive a new feature or convert raw entries into structured tuples***

  A new column TotalValue = Quantity × UnitPrice was created.
  - RDD version: by mapping each row tuple
  - DF version: using withColumn() and col()

This identifies revenue per invoice line.

```
# Now create an RDD of tuples
rdd = df.rdd.map(lambda row: (
    row['Invoice'],
    row['StockCode'],
    row['Description'],
    row['Quantity'] if row['Quantity'] is not None else 0,
    row['InvoiceDateTS'],
    row['Price'] if row['Price'] is not None else 0.0,
    row['Customer ID'],
    row['Country']
))
# sample
print("Sample RDD tuples:")
for t in rdd.take(5):
    print(t)
```

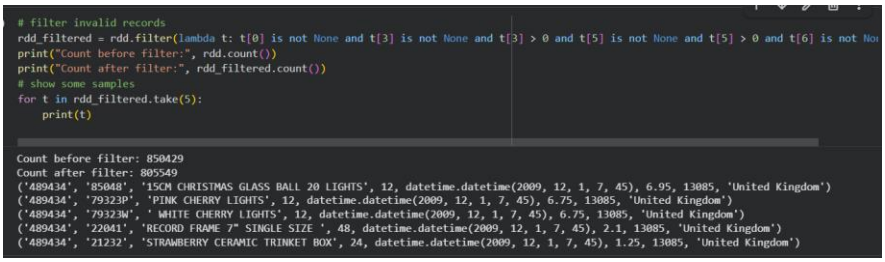
```
root
 |-- Invoice: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: timestamp (nullable = true)
 |-- Price: double (nullable = true)
 |-- Customer ID: integer (nullable = true)
 |-- Country: string (nullable = true)
 |-- InvoiceDateTS: timestamp (nullable = true)

+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+-------------------+
|Invoice|StockCode|Description                        |Quantity|InvoiceDate        |Price|Customer ID|Country       |InvoiceDateTS      |
+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+-------------------+
|489434 |85048    |15CM CHRISTMAS GLASS BALL 20 LIGHTS|12      |2009-12-01 07:45:00|6.95 |13085      |United Kingdom|2009-12-01 07:45:00|
|489434 |79323P   |PINK CHERRY LIGHTS                 |12      |2009-12-01 07:45:00|6.75 |13085      |United Kingdom|2009-12-01 07:45:00|
|489434 |79323W   | WHITE CHERRY LIGHTS               |12      |2009-12-01 07:45:00|6.75 |13085      |United Kingdom|2009-12-01 07:45:00|
|489434 |22041    |RECORD FRAME 7" SINGLE SIZE        |48      |2009-12-01 07:45:00|2.1  |13085      |United Kingdom|2009-12-01 07:45:00|
|489434 |21232    |STRAWBERRY CERAMIC TRINKET BOX     |24      |2009-12-01 07:45:00|1.25 |13085      |United Kingdom|2009-12-01 07:45:00|
+-------+---------+-----------------------------------+--------+-------------------+-----+-----------+--------------+-------------------+
only showing top 5 rows

Sample RDD tuples:
('489434', '85048', '15CM CHRISTMAS GLASS BALL 20 LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.95, 13085, 'United Kingdom')
('489434', '79323P', 'PINK CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '79323W', ' WHITE CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '22041', 'RECORD FRAME 7" SINGLE SIZE ', 48, datetime.datetime(2009, 12, 1, 7, 45), 2.1, 13085, 'United Kingdom')
('489434', '21232', 'STRAWBERRY CERAMIC TRINKET BOX', 24, datetime.datetime(2009, 12, 1, 7, 45), 1.25, 13085, 'United Kingdom')
```

✓ ***filter() transformation – remove invalid / missing / noisy records***

Rows with invalid values were removed:

- Negative or zero quantities
- Negative prices
- Null CustomerID entries

This ensures clean data for analysis.



```
# filter invalid records
rdd_filtered = rdd.filter(lambda t: t[0] is not None and t[3] is not None and t[3] > 0 and t[5] is not None and t[5] > 0 and t[6] is not No
print("Count before filter:", rdd.count())
print("Count after filter:", rdd_filtered.count())
# show some samples
for t in rdd_filtered.take(5):
    print(t)
```

```
Count before filter: 850429
Count after filter: 805549
('489434', '85048', '15CM CHRISTMAS GLASS BALL 20 LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.95, 13085, 'United Kingdom')
('489434', '79323P', 'PINK CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '79323W', ' WHITE CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '22041', 'RECORD FRAME 7" SINGLE SIZE ', 48, datetime.datetime(2009, 12, 1, 7, 45), 2.1, 13085, 'United Kingdom')
('489434', '21232', 'STRAWBERRY CERAMIC TRINKET BOX', 24, datetime.datetime(2009, 12, 1, 7, 45), 1.25, 13085, 'United Kingdom')
```

✓ ***flatMap() transformation – break complex fields (e.g., text, lists, logs) into smaller elements***

Product descriptions were tokenized into individual lowercase words.
This demonstrates how flatMap() can produce more output elements than input elements.

```
# flatMap: tokenizing descriptions into words (lowercase, simple split, remove empty)
desc_words = rdd_filtered.flatMap(lambda t: [(w.strip().lower()) for w in (t[2] or "").split() if w.strip() != ""])
# Count distinct words and top 20 frequent words
word_counts = desc_words.map(lambda w: (w, 1)).reduceByKey(lambda a, b: a + b)
top_words = word_counts.takeOrdered(20, key=lambda x: -x[1])
print("Top description words (word, count):")
for w,c in top_words:
    print(w, c)
```

```
Top description words (word, count):
of 70769
set 70506
bag 69889
red 69818
heart 59294
pink 45740
retrospot 45325
design 43086
vintage 42804
box 38404
white 38263
cake 35570
metal 35472
christmas 34466
blue 30160
hanging 28817
holder 27851
jumbo 27225
sign 26759
3 26420
```

✓ ***reduceByKey() or aggregate() – compute summary statistics or grouped aggregates***

Two key-value aggregations were implemented:

- Total revenue per Country
- Total quantity per StockCode

This step shows how to aggregate distributed key-value data efficiently.

```
# revenue per row = Quantity * Price
rdd_revenue = rdd_filtered.map(lambda t: (t[7], float(t[3]) * float(t[5]) if t[3] is not None and t[5] is not None else 0.0))
# sum revenue by country
revenue_by_country = rdd_revenue.reduceByKey(lambda a, b: a + b)
print("Top revenue by country (top 10):")
for c, tot in revenue_by_country.takeOrdered(10, key=lambda x: -x[1]):
    print(c, round(tot,2))

# total quantity by stock code
qty_by_stock = rdd_filtered.map(lambda t: (t[1], t[3])).reduceByKey(lambda a, b: a + b)
print("Top 10 stock codes by total quantity:")
for s, q in qty_by_stock.takeOrdered(10, key=lambda x: -x[1]):
    print(s, q)
```

```
Top revenue by country (top 10):
United Kingdom 14723147.52
EIRE 621631.11
Netherlands 554232.34
Germany 431262.46
France 355257.47
Australia 169968.11
Spain 109178.53
Switzerland 100365.34
Sweden 91549.72
Denmark 69862.19
Top 10 stock codes by total quantity:
84077 109169
85099B 94983
85123A 93697
21212 91263
23843 80995
84879 79913
22197 77971
23166 77916
17003 71129
21977 55270
```

- ✓ ***take() (action) – extract sample outputs and comment on distributed execution***

  A small sample of the cleaned dataset (first 5 rows) was shown to verify transformations.

```
# take a small sample of filtered RDD
sample_rows = rdd_filtered.take(10)
print("Sample rows (first 10):")
for r in sample_rows:
    print(r)
```

```
Sample rows (first 10):
('489434', '85048', '15CM CHRISTMAS GLASS BALL 20 LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.95, 13085, 'United Kingdom')
('489434', '79323P', 'PINK CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '79323W', ' WHITE CHERRY LIGHTS', 12, datetime.datetime(2009, 12, 1, 7, 45), 6.75, 13085, 'United Kingdom')
('489434', '22041', 'RECORD FRAME 7" SINGLE SIZE ', 48, datetime.datetime(2009, 12, 1, 7, 45), 2.1, 13085, 'United Kingdom')
('489434', '21232', 'STRAWBERRY CERAMIC TRINKET BOX', 24, datetime.datetime(2009, 12, 1, 7, 45), 1.25, 13085, 'United Kingdom')
('489434', '22064', 'PINK DOUGHNUT TRINKET POT ', 24, datetime.datetime(2009, 12, 1, 7, 45), 1.65, 13085, 'United Kingdom')
('489434', '21871', 'SAVE THE PLANET MUG', 24, datetime.datetime(2009, 12, 1, 7, 45), 1.25, 13085, 'United Kingdom')
('489434', '21523', 'FANCY FONT HOME SWEET HOME DOORMAT', 10, datetime.datetime(2009, 12, 1, 7, 45), 5.95, 13085, 'United Kingdom')
('489435', '22350', 'CAT BOWL ', 12, datetime.datetime(2009, 12, 1, 7, 46), 2.55, 13085, 'United Kingdom')
('489435', '22349', 'DOG BOWL , CHASING BALL DESIGN', 12, datetime.datetime(2009, 12, 1, 7, 46), 3.75, 13085, 'United Kingdom')
```

Task 2: Spark SQL Queries

A temporary SQL view called Retail was created, enabling SQL queries directly over the DataFrame.

```
# create a temp view for Spark SQL
df.createOrReplaceTempView("retail")
# quick check
spark.sql("SELECT COUNT(*) as cnt FROM retail").show()
```

```
+-------+
|    cnt|
+-------+
|1067371|
+-------+
```

✓ ***Aggregation Query***

A monthly sales report was generated, containing:

- Total monthly revenue
- Average unit price
- Number of distinct invoices

This query shows how SQL can be used for high-level business insights.

```
# ensure InvoiceDateTS is available
spark.sql("""
SELECT year(InvoiceDateTS) as yr, month(InvoiceDateTS) as mth,
       SUM(Quantity * Price) as total_revenue,
       AVG(Price) as avg_price,
       COUNT(DISTINCT Invoice) as invoice_count
FROM retail
WHERE InvoiceDateTS IS NOT NULL
GROUP BY yr, mth
ORDER BY yr DESC, mth DESC
LIMIT 20
""").show(truncate=False)
```

```
+----+---+------------------+------------------+-------------+
|yr  |mth|total_revenue     |avg_price         |invoice_count|
+----+---+------------------+------------------+-------------+
|2011|12 |433686.01000000007|5.246950560213116 |1015         |
|2011|11 |1461756.2500000012|3.8619524028756738|3462         |
|2011|10 |1070704.6700000009|4.336934740377341 |2637         |
|2011|9  |1019687.6220000024|3.9667744196232895|2327         |
|2011|8  |682680.5099999998 |4.262149416165961 |1737         |
|2011|7  |681300.1110000005 |4.350088339490869 |1927         |
|2011|6  |691123.1200000003 |5.443329717416069 |2012         |
|2011|5  |723333.5099999984 |5.149485822306237 |2162         |
|2011|4  |493207.1210000001 |4.317587946249489 |1744         |
|2011|3  |683267.0800000025 |4.666553553934908 |1983         |
|2011|2  |498062.64999999973|4.59987620456924  |1393         |
|2011|1  |560000.2599999993 |4.915150652971798 |1476         |
|2010|12 |1126445.4699999988|6.564512953049056 |2025         |
|2010|11 |1422654.6419999998|3.869946318015791 |3669         |
|2010|10 |1045168.3499999987|4.500547226640501 |2965         |
|2010|9  |853650.4300999999 |5.316177353828634 |2375         |
|2010|8  |656776.3400000015 |4.737227226325586 |1877         |
|2010|7  |575236.3600000006 |3.015034598448303 |2017         |
|2010|6  |679786.6100000001 |5.8742388014906455|2216         |
|2010|5  |615322.8300000001 |4.045113099113884 |2418         |
+----+---+------------------+------------------+-------------+
```

✓ ***Filtering Query***

All high-value transactions from the United Kingdom after January 1, 2010, were selected using:

- WHERE Country = 'United Kingdom'
- InvoiceDateTS >= '2010-01-01'
- TotalValue > 100

This demonstrates SQL filtering using both numeric and temporal conditions.

```
spark.sql("""
SELECT Invoice, StockCode, Description, Quantity, Price, (Quantity*Price) as revenue, InvoiceDateTS, `Customer ID`, Country
FROM retail
WHERE Country = 'United Kingdom'
  AND InvoiceDateTS >= '2010-01-01' AND InvoiceDateTS < '2011-01-01'
  AND (Quantity * Price) > 100
ORDER BY revenue DESC
LIMIT 50
""").show(truncate=False)
```

```
+-------+---------+----------------------------------+--------+--------+--------+-----------------+-----------+--------------+
|Invoice|StockCode|Description                       |Quantity|Price   |revenue |InvoiceDateTS    |Customer ID|Country       |
+-------+---------+----------------------------------+--------+--------+--------+-----------------+-----------+--------------+
|512771 |M        |Manual                            |1       |25111.09|25111.09|2010-06-17 16:53:00|NULL       |United Kingdom|
|530715 |84347    |ROTATING SILVER ANGELS T-LIGHT HLDR|9360    |1.69    |15818.4 |2010-11-04 11:36:00|15838      |United Kingdom|
|537632 |AMAZONFEE|AMAZON FEE                        |1       |13541.33|13541.33|2010-12-07 15:08:00|NULL       |United Kingdom|
|537632 |AMAZONFEE|AMAZON FEE                        |1       |13541.33|13541.33|2010-12-07 15:08:00|NULL       |United Kingdom|
|502263 |M        |Manual                            |1       |10953.5 |10953.5 |2010-03-23 15:22:00|12918      |United Kingdom|
|502265 |M        |Manual                            |1       |10953.5 |10953.5 |2010-03-23 15:28:00|NULL       |United Kingdom|
|522796 |M        |Manual                            |1       |10468.8 |10468.8 |2010-09-16 15:12:00|NULL       |United Kingdom|
|524159 |M        |Manual                            |1       |10468.8 |10468.8 |2010-09-27 16:12:00|14063      |United Kingdom|
|525399 |M        |Manual                            |1       |10468.8 |10468.8 |2010-10-05 11:49:00|NULL       |United Kingdom|
|496115 |M        |Manual                            |1       |8985.6  |8985.6  |2010-01-29 11:04:00|17949      |United Kingdom|
|511465 |15044A   |PINK PAPER PARASOL                |3500    |2.55    |8925.0  |2010-06-08 12:59:00|18008      |United Kingdom|
|525473 |M        |Manual                            |1       |7044.79 |7044.79 |2010-10-05 15:16:00|NULL       |United Kingdom|
|531411 |AMAZONFEE|AMAZON FEE                        |1       |6706.71 |6706.71 |2010-11-08 10:11:00|NULL       |United Kingdom|
|505505 |M        |Manual                            |1       |5876.34 |5876.34 |2010-04-22 14:09:00|NULL       |United Kingdom|
|504639 |M        |Manual                            |1       |5843.7  |5843.7  |2010-04-15 14:12:00|NULL       |United Kingdom|
|533027 |22086    |PAPER CHAIN KIT 50'S CHRISTMAS    |835     |6.95    |5803.25 |2010-11-15 16:02:00|NULL       |United Kingdom|
|501772 |M        |Manual                            |1       |5795.87 |5795.87 |2010-03-19 11:56:00|NULL       |United Kingdom|
|525968 |84347    |ROTATING SILVER ANGELS T-LIGHT HLDR|3120    |1.66    |5179.2  |2010-10-08 10:10:00|15838      |United Kingdom|
|495798 |ADJUST   |Adjustment by john on 26/01/2010 17|1       |5117.03 |5117.03 |2010-01-26 17:25:00|NULL       |United Kingdom|
|512759 |M        |Manual                            |1       |4620.86 |4620.86 |2010-06-17 15:51:00|NULL       |United Kingdom|
+-------+---------+----------------------------------+--------+--------+--------+-----------------+-----------+--------------+
only showing top 20 rows
```

✓ *Join Query*

A helper table was created to classify Product IDs by prefix (e.g., A, B, C, …).
A LEFT JOIN was performed to attach category labels to the main dataset.

```
# Simulated helper table: we map a few StockCode prefixes to categories.
# In a real scenario, you'd have a proper catalog. Here we create a small lookup to fulfill assignment.
prod_cat = [
    ("85", "Christmas"),
    ("793", "Lights"),
    ("220", "Frames"),
    ("M", "Misc"),
    ("BANK", "Services")
]

prod_schema = StructType([StructField("prefix", StringType(), False), StructField("category", StringType(), False)])
prod_df = spark.createDataFrame(prod_cat, schema=prod_schema)
prod_df.createOrReplaceTempView("prod_cat")

# Join: we'll create a derived column that finds prefix (first 2-3 characters) and left join to category.
spark.sql("""
SELECT r.StockCode, pc.category,
       SUM(r.Quantity * r.Price) as total_revenue,
       SUM(r.Quantity) as total_quantity
FROM retail r
LEFT JOIN prod_cat pc
  ON r.StockCode LIKE concat(pc.prefix, '%')
WHERE r.Quantity > 0 AND r.Price > 0
GROUP BY r.StockCode, pc.category
ORDER BY total_revenue DESC
LIMIT 20
""").show(truncate=False)
```

```
+---------+---------+------------------+--------------+
|StockCode|category |total_revenue     |total_quantity|
+---------+---------+------------------+--------------+
|22423    |NULL     |344563.2500000036 |27577         |
|M        |Misc     |341089.8500000007 |10051         |
|DOT      |NULL     |322657.48000000016|1436          |
|85123A   |Christmas|263109.6700000069 |96147         |
|85099B   |Christmas|183454.82999999603|98349         |
|23843    |NULL     |168469.6          |80995         |
|47566    |NULL     |149187.04999999984|28378         |
|84879    |NULL     |132187.9200000002 |81809         |
|POST     |NULL     |127597.42         |5461          |
|22086    |Frames   |123141.53999999803|36581         |
|79321    |Lights   |85489.9100000002  |16840         |
|23166    |NULL     |81700.92000000007 |78033         |
|22197    |NULL     |80920.63999999929 |89898         |
|22386    |NULL     |77111.91000000112 |39840         |
|84347    |NULL     |74448.92000000032 |32525         |
|20725    |NULL     |72292.8499999999  |40942         |
|21137    |NULL     |69329.47000000007 |19954         |
|85099F   |Christmas|68488.81000000068 |37124         |
|20685    |NULL     |67859.56000000007 |9974          |
|48138    |NULL     |67757.62999999995 |10175         |
+---------+---------+------------------+--------------+
```

Task 3: DataFrames API

✓ *Data Cleaning*

Several cleaning steps were applied using DataFrame transformations:

- dropna(subset=['Description'])
- Fixing missing Customer IDs
- Removing negative quantities and prices
- Creating parsed timestamp column InvoiceDateTS

This produced a reliable dataset for further analysis.

```
# show counts before cleaning
print("Before cleaning:", df.count())

# Mark missing descriptions
df_clean = df.withColumn("Description", when(col("Description").isNull() | (col("Description") == ""), "UNKNOWN").otherwise(col("Description")))

# Drop rows with missing InvoiceDateTS or Customer ID, and non-positive amounts
df_clean = df_clean.filter((col("InvoiceDateTS").isNotNull()) & (col("Customer ID").isNotNull()) & (col("Quantity") > 0) & (col("Price") > 0))
print("After cleaning:", df_clean.count())

# Show a few cleaned rows
df_clean.select("Invoice", "StockCode", "Description", "Quantity", "Price", "InvoiceDateTS", "Customer ID", "Country").show(5, truncate=False)
```

```
Before cleaning: 1067371
After cleaning: 805549
+-------+---------+----------------------------------+--------+-----+-------------------+-----------+--------------+
|Invoice|StockCode|Description                       |Quantity|Price|InvoiceDateTS      |Customer ID|Country       |
+-------+---------+----------------------------------+--------+-----+-------------------+-----------+--------------+
|489434 |85048    |15CM CHRISTMAS GLASS BALL 20 LIGHTS|12      |6.95 |2009-12-01 07:45:00|13085      |United Kingdom|
|489434 |79323P   |PINK CHERRY LIGHTS                |12      |6.75 |2009-12-01 07:45:00|13085      |United Kingdom|
|489434 |79323W   | WHITE CHERRY LIGHTS              |12      |6.75 |2009-12-01 07:45:00|13085      |United Kingdom|
|489434 |22041    |RECORD FRAME 7" SINGLE SIZE       |48      |2.1  |2009-12-01 07:45:00|13085      |United Kingdom|
|489434 |21232    |STRAWBERRY CERAMIC TRINKET BOX    |24      |1.25 |2009-12-01 07:45:00|13085      |United Kingdom|
+-------+---------+----------------------------------+--------+-----+-------------------+-----------+--------------+
only showing top 5 rows
```

✓ *Transformation and Grouping*

Advanced DataFrame operations were used:

- Grouping by Country to calculate total revenue
- Monthly sales summary using date_format()
- Counting products per country
- Sorting using orderBy()

These tasks demonstrate ability to work with structured data programmatically.

```
from pyspark.sql.functions import mean, stddev, sum as _sum

# Aggregates by Country
agg_by_country = df_clean.groupBy("Country").agg(
    count("*").alias("tx_count"),
    mean("Price").alias("avg_price"),
    stddev("Price").alias("stddev_price"),
    mean("Quantity").alias("avg_qty")
).orderBy(col("tx_count").desc())

agg_by_country.show(20, truncate=False)

# Sales by month for 'United Kingdom' (example)
sales_month_uk = df_clean.filter(col("Country") == "United Kingdom") \
    .withColumn("year", year(col("InvoiceDateTS"))) \
    .withColumn("month", month(col("InvoiceDateTS"))) \
    .groupBy("year", "month") \
    .agg(
        _sum(col("Quantity") * col("Price")).alias("monthly_revenue"),
        count("*").alias("monthly_transactions")
    ).orderBy("year", "month")

sales_month_uk.show(24, truncate=False)
```

```
+--------------+--------+------------------+------------------+------------------+
|Country       |tx_count|avg_price         |stddev_price      |avg_qty           |
+--------------+--------+------------------+------------------+------------------+
|United Kingdom|725250  |3.056777065832837 |25.165511573954838|12.003189245087901|
|Germany       |16694   |3.5952588355096005|12.079504653541388|13.656583203546184|
|EIRE          |15743   |5.315841326303199 |42.66892193877868 |20.43466937686591 |
|France        |13812   |4.271104836374085 |60.458187552642244|19.814219519258614|
|Netherlands   |5088    |2.687150157232728 |7.142500208944403 |75.46717767295597 |
|Spain         |3719    |4.24339069642386  |19.02973750067578 |13.658510352245226|
|Belgium       |3068    |4.238813559322064 |27.93134542671336 |11.50651890482399 |
|Switzerland   |3011    |3.8293324476918165|16.442703754660528|17.381932912653603|
|Portugal      |2446    |5.058556827473441 |40.16102680396904 |11.492641046606705|
|Australia     |1812    |3.6036810154525036|19.9943812254724  |57.4448123620309  |
|Channel Islands|1569   |4.637233906947068 |9.762179650442981 |13.691523263224983|
|Italy         |1468    |4.765660762942743 |13.619571247558474|10.559264305177111|
|Norway        |1436    |14.20522980501404 |260.45576258977036|18.94359331476323 |
|Sweden        |1319    |4.89183472327519  |35.465212928198575|67.10538286580743 |
|Cyprus        |1155    |5.095645021644989 |14.209840379119353|9.511688311688312 |
|Finland       |1032    |4.7656976744185915|11.667788661022604|13.929263565891473|
|Austria       |922     |4.281681127982629 |9.359875604369083 |12.557483731019524|
|Denmark       |798     |2.85459899749374  |3.555707699144516 |298.1516290726817 |
|Greece        |657     |3.8520243531202425|5.08544467795853  |11.756468797564688|
|Unspecified   |521     |3.2311132437620014|3.215763264559387 |9.821497120921306 |
+--------------+--------+------------------+------------------+------------------+
only showing top 20 rows
```

```
+----+-----+-------------------+---------------------+
|year|month|monthly_revenue    |monthly_transactions|
+----+-----+-------------------+---------------------+
|2009|12   |613214.9000000025  |28606                |
|2010|1    |416635.24200001615|19569                |
|2010|2    |411077.9360000116  |21344                |
|2010|3    |589725.0110000053  |29766                |
|2010|4    |503425.74100001546|24848                |
|2010|5    |503483.1200000187  |25808                |
|2010|6    |541357.8600000184  |28436                |
|2010|7    |504239.9200000161  |24481                |
|2010|8    |508488.09000001557|24085                |
|2010|9    |681585.7710000142  |30594                |
|2010|10   |879189.4499999321  |44782                |
|2010|11   |983677.1319999143  |55242                |
|2010|12   |777317.6399999786  |37310                |
|2011|1    |442190.0600000155  |18158                |
|2011|2    |355655.6300000114  |17758                |
|2011|3    |467198.5900000209  |24012                |
|2011|4    |409559.14100000996|20865                |
|2011|5    |551568.8200000015  |25202                |
|2011|6    |524915.4800000143  |23714                |
|2011|7    |485612.25100000994|23598                |
|2011|8    |498453.32000000845|23104                |
|2011|9    |796780.2719999977  |35634                |
|2011|10   |824766.2200000049  |43733                |
|2011|11   |980645.7499999859  |58800                |
+----+-----+-------------------+---------------------+
only showing top 24 rows
```

```
# create revenue column
df_with_revenue = df_clean.withColumn("revenue", col("Quantity") * col("Price"))

# sort descending by revenue, ascending by InvoiceDateTS
sorted_tx = df_with_revenue.orderBy(col("revenue").desc(), col("InvoiceDateTS").asc())
sorted_tx.select("Invoice", "StockCode", "Description", "Quantity", "Price", "revenue", "InvoiceDateTS").show(50, truncate=False)
```

```
+-------+---------+-----------------------------------+--------+-------+-----------------+-------------------+
|Invoice|StockCode|Description                        |Quantity|Price  |revenue          |InvoiceDateTS      |
+-------+---------+-----------------------------------+--------+-------+-----------------+-------------------+
|581483 |23843    |PAPER CRAFT , LITTLE BIRDIE        |80995   |2.08   |168469.6         |2011-12-09 09:15:00|
|541431 |23166    |MEDIUM CERAMIC TOP STORAGE JAR     |74215   |1.04   |77183.6          |2011-01-18 10:01:00|
|556444 |22502    |PICNIC BASKET WICKER 60 PIECES     |60      |649.5  |38970.0          |2011-06-10 15:28:00|
|530715 |84347    |ROTATING SILVER ANGELS T-LIGHT HLDR|9360    |1.69   |15818.4          |2010-11-04 11:36:00|
|502263 |M        |Manual                             |1       |10953.5|10953.5          |2010-03-23 15:22:00|
|524159 |M        |Manual                             |1       |10468.8|10468.8          |2010-09-27 16:12:00|
|496115 |M        |Manual                             |1       |8985.6 |8985.6           |2010-01-29 11:04:00|
|511465 |15044A   |PINK PAPER PARASOL                 |3500    |2.55   |8925.0           |2010-06-08 12:59:00|
|551697 |POST     |POSTAGE                            |1       |8142.75|8142.75          |2011-05-03 13:46:00|
|567423 |23243    |SET OF TEA COFFEE SUGAR TINS PANTRY|1412    |5.06   |7144.719999999999|2011-09-20 11:05:00|
|501766 |M        |Manual                             |1       |6958.17|6958.17          |2010-03-19 11:35:00|
|501768 |M        |Manual                             |1       |6958.17|6958.17          |2010-03-19 11:45:00|
|540815 |21108    |FAIRY CAKE FLANNEL ASSORTED COLOUR |3114    |2.1    |6539.400000000001|2011-01-11 12:55:00|
|550461 |21108    |FAIRY CAKE FLANNEL ASSORTED COLOUR |3114    |2.1    |6539.400000000001|2011-04-18 13:20:00|
|525968 |84347    |ROTATING SILVER ANGELS T-LIGHT HLDR|3120    |1.66   |5179.2           |2010-10-08 10:10:00|
|573003 |23084    |RABBIT NIGHT LIGHT                 |2400    |2.08   |4992.0           |2011-10-27 12:11:00|
|540815 |85123A   |WHITE HANGING HEART T-LIGHT HOLDER |1930    |2.55   |4921.5           |2011-01-11 12:55:00|
|550461 |85123A   |WHITE HANGING HEART T-LIGHT HOLDER |1930    |2.4    |4632.0           |2011-04-18 13:20:00|
|540818 |48185    |DOORMAT FAIRY CAKE                 |670     |6.75   |4522.5           |2011-01-11 12:57:00|
|524181 |21622    |VINTAGE UNION JACK CUSHION COVER   |648     |6.89   |4464.719999999999|2010-09-27 16:59:00|
|558526 |23173    |REGENCY TEAPOT ROSES               |540     |8.15   |4401.0           |2011-06-30 11:01:00|
|550461 |48185    |DOORMAT FAIRY CAKE                 |670     |6.35   |4254.5           |2011-04-18 13:20:00|
|532358 |84879    |ASSORTED COLOUR BIRD ORNAMENT      |2880    |1.45   |4176.0           |2010-11-11 17:05:00|
|562439 |84879    |ASSORTED COLOUR BIRD ORNAMENT      |2880    |1.45   |4176.0           |2011-08-04 18:06:00|
|573077 |M        |Manual                             |1       |4161.06|4161.06          |2011-10-27 14:13:00|
|573080 |M        |Manual                             |1       |4161.06|4161.06          |2011-10-27 14:20:00|
|540689 |22470    |HEART OF WICKER LARGE              |1284    |3.21   |4121.64          |2011-01-11 08:43:00|
|571751 |M        |Manual                             |1       |3949.32|3949.32          |2011-10-19 11:18:00|
|529613 |22423    |REGENCY CAKESTAND 3 TIER           |360     |10.95  |3941.9999999999995|2010-10-29 10:46:00|
|526934 |22790    |MIRROR, ARCHED GEORGIAN            |200     |19.42  |3884.0000000000005|2010-10-14 09:46:00|
|524181 |21624    |VINTAGE UNION JACK DOORSTOP        |648     |5.96   |3862.08          |2010-09-27 16:59:00|
|581115 |22413    |METAL SIGN TAKE IT OR LEAVE IT     |1404    |2.75   |3861.0           |2011-12-07 12:20:00|
|537659 |21623    |VINTAGE UNION JACK MEMOBOARD       |600     |6.38   |3828.0           |2010-12-07 16:43:00|
|537659 |21623    |VINTAGE UNION JACK MEMOBOARD       |600     |6.38   |3828.0           |2010-12-07 16:43:00|
|567423 |23113    |PANTRY CHOPPING BOARD              |756     |5.06   |3825.3599999999997|2011-09-20 11:05:00|
|537899 |22328    |ROUND SNACK BOXES SET OF 4 FRUITS  |1488    |2.55   |3794.3999999999996|2010-12-09 10:44:00|
|537899 |22328    |ROUND SNACK BOXES SET OF 4 FRUITS  |1488    |2.55   |3794.3999999999996|2010-12-09 10:44:00|
|523166 |22632    |HAND WARMER RED POLKA DOT          |2004    |1.85   |3707.4           |2010-09-20 15:33:00|
|540815 |21175    |GIN + TONIC DIET METAL SIGN        |2000    |1.85   |3700.0           |2011-01-11 12:55:00|
|579498 |23084    |RABBIT NIGHT LIGHT                 |2040    |1.79   |3651.6           |2011-11-29 15:52:00|
|497946 |85220    |SMALL FAIRY CAKE FRIDGE MAGNETS    |2504    |1.45   |3630.7999999999997|2010-02-15 11:57:00|
|567423 |22722    |SET OF 6 SPICE TINS PANTRY DESIGN  |852     |4.25   |3621.0           |2011-09-20 11:05:00|
|524181 |21621    |VINTAGE UNION JACK BUNTING         |408     |8.87   |3618.9599999999996|2010-09-27 16:59:00|
|517483 |M        |Manual                             |1       |3610.5 |3610.5           |2010-07-29 12:29:00|
|524181 |21770    |OPEN CLOSED METAL SIGN             |684     |5.15   |3522.6000000000004|2010-09-27 16:59:00|
|550461 |21175    |GIN + TONIC DIET METAL SIGN        |2000    |1.69   |3380.0           |2011-04-18 13:20:00|
|540818 |47556B   |TEA TIME TEA TOWELS                |1300    |2.55   |3314.9999999999995|2011-01-11 12:57:00|
|524181 |21906    |PHARMACIE FIRST AID TIN            |464     |7.13   |3308.32          |2010-09-27 16:59:00|
|572209 |23556    |LANDMARK FRAME COVENT GARDEN       |300     |10.95  |3285.0           |2011-10-21 12:08:00|
|572209 |23554    |LANDMARK FRAME OXFORD STREET       |300     |10.95  |3285.0           |2011-10-21 12:08:00|
+-------+---------+-----------------------------------+--------+-------+-----------------+-------------------+
only showing top 50 rows
```

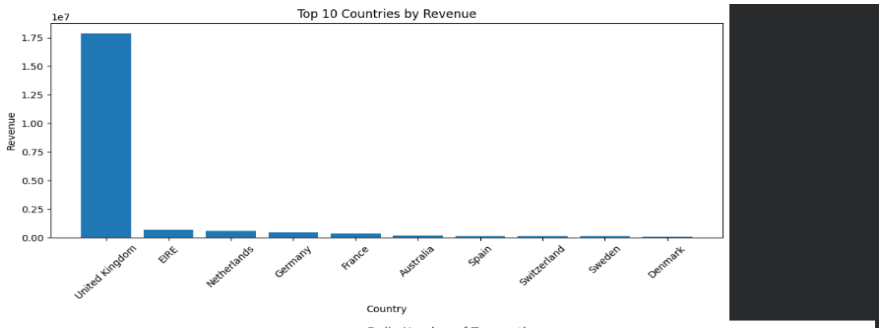
Results Summary / Visual Representations

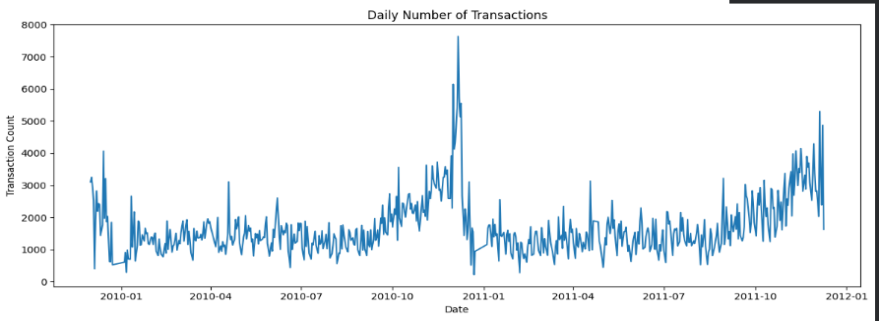
Each stage of the assignment produced meaningful results:

- RDDs: Low-level data manipulation and aggregation
- SQL: Business-style queries for insights
- DataFrames: High-level analytics and transformations

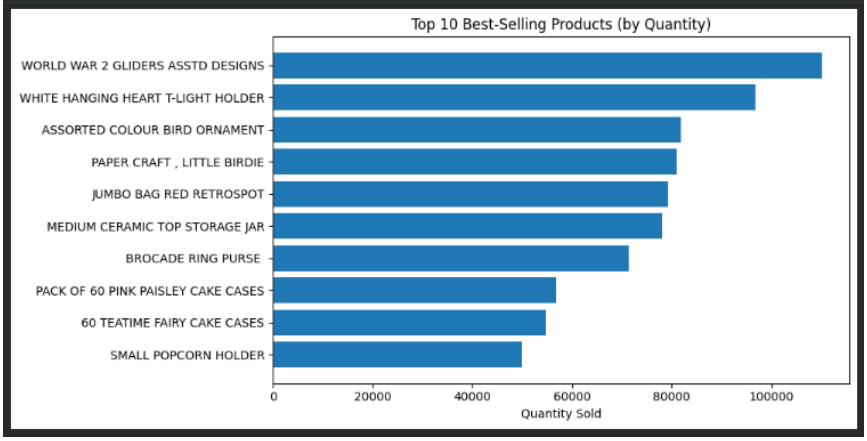
The outputs confirm that Spark successfully handled the dataset, and all tasks required by the assignment instructions were completed. Meanwhile, additional visual representations are displayed below:

```python
# ------------------------------
# 5. VISUAL 1 — Top 10 Countries by Revenue
# ------------------------------
plt.figure(figsize=(10,5))
plt.bar(country_sales["Country"], country_sales["Revenue"])
plt.xticks(rotation=45)
plt.title("Top 10 Countries by Revenue")
plt.xlabel("Country")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()

# ------------------------------
# 6. VISUAL 2 — Daily Transaction Count Over Time
# ------------------------------
plt.figure(figsize=(12,5))
plt.plot(daily_sales["Day"], daily_sales["Transactions"])
plt.title("Daily Number of Transactions")
plt.xlabel("Date")
plt.ylabel("Transaction Count")
plt.tight_layout()
plt.show()
```

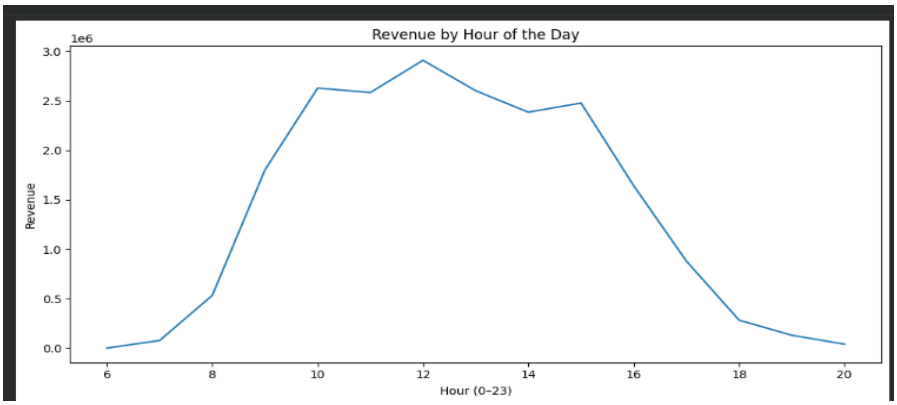




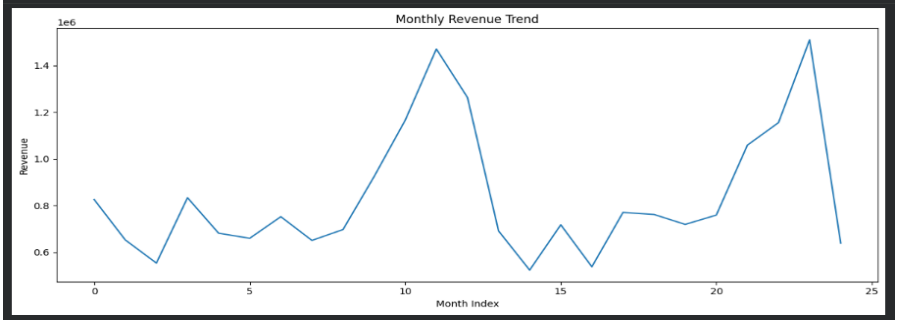

```python
# Top 10 products by quantity sold
top_products = (df.groupBy("Description")
                  .agg(spark_sum("Quantity").alias("TotalQty"))
                  .orderBy(col("TotalQty").desc())
                  .limit(10)
                  .toPandas())

plt.figure(figsize=(10,5))
plt.barh(top_products["Description"], top_products["TotalQty"])
plt.gca().invert_yaxis()
plt.title("Top 10 Best-Selling Products (by Quantity)")
plt.xlabel("Quantity Sold")
plt.tight_layout()
plt.show()
```




```python
from pyspark.sql.functions import hour

hourly_sales = (df.withColumn("Hour", hour("InvoiceTS"))
                  .groupBy("Hour")
                  .agg(spark_sum("TotalValue").alias("Revenue"))
                  .orderBy("Hour")
                  .toPandas())

plt.figure(figsize=(10,5))
plt.plot(hourly_sales["Hour"], hourly_sales["Revenue"])
plt.title("Revenue by Hour of the Day")
plt.xlabel("Hour (0-23)")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```




```
from pyspark.sql.functions import month, year

monthly_sales = (df.withColumn("Month", month("InvoiceTS"))
                   .withColumn("Year", year("InvoiceTS"))
                   .groupBy("Year", "Month")
                   .agg(spark_sum("TotalValue").alias("Revenue"))
                   .orderBy("Year", "Month")
                   .toPandas())

plt.figure(figsize=(12,5))
plt.plot(monthly_sales["Revenue"])
plt.title("Monthly Revenue Trend")
plt.xlabel("Month Index")
plt.ylabel("Revenue")
plt.tight_layout()
plt.show()
```



Discussion of Results / Conclusion

The analysis demonstrates that Apache Spark efficiently handled the large Online Retail II dataset across all three APIs. RDD operations provided low-level control for custom transformations and aggregations, revealing key insights such as total revenue per country and quantity sold per product. Spark SQL enabled intuitive, business-oriented queries, producing monthly sales summaries and filtered high-value transactions for targeted analysis. DataFrames offered a high-level, expressive interface for data cleaning, grouping, and sorting, which streamlined preprocessing and advanced analytics.

Overall, the results show that Spark's distributed computing capabilities make it possible to process, query, and transform large datasets quickly and reliably, while each API offers unique advantages depending on the task, demonstrating both the flexibility and power of Spark in real-world data analysis. This assignment provided hands-on experience with three core Spark APIs: RDDs, SQL, and DataFrames.

By completing each task, I learned how different abstractions can be used for similar goals and how to choose the appropriate API based on the type of analysis.