XIAMEN UNIVERSITY MALAYSIA

Course Code	:	SWE404
Course Name	:	Big Data Analytics
Lecturer	:	Dr. Toa Chean Khim
Academic Session	:	2025/04
Assessment Title	:	Project
Submission Due Date	:	20 June 2025

Prepared by :

Student ID	Student Name
AIT2209937	Eng Kuan Tian
AIT2209950	Sow Wei Thao
AIT2209608	Liew Cheah Ming

Date Received :

Feedback from Lecturer:

Mark:

Own Work Declaration

I/We acknowledge that my/our work will be examined for plagiarism or any other form of academic misconduct, and that the digital copy of the work may be retained for future comparisons.

I/We confirm that all sources cited in this work have been correctly acknowledged and that I/we fully understand the serious consequences of any intentional or unintentional academic misconduct.

In addition, I/we affirm that this submission does not contain any materials generated by AI tools, including direct copying and pasting of text or paraphrasing. This work is my/our original creation, and it has not been based on any work of other students (past or present), nor has it been previously submitted to any other course or institution.

Furthermore, I/we consent to the lecturer's right to publish this work, either partially or in full, at any time, for educational, research, or institutional purposes. The lecturer may choose to include or omit my/our name(s) in any such publication.

1. Name : Eng Kuan Tian Signature: KT

2. Name : Sow Wei Thao Signature: WeiThao

3. Name : Liew Cheah Ming Signature: CheahMing

Date: 17/6//25

Members' Contribution Sheet

Member Name	Student ID	Contributions
Eng Kuan Tian	AIT2209937	- Responsible for setting up the Cloudera Virtual Machine using Oracle VirtualBox. - Collected and prepared a large textual dataset from online sources ensuring the presence of symbols, words, and numeric data. - Implemented the MapReduce Word Count program to analyze a large-scale text dataset, including: Cleaning the dataset by removing non-alphabetic characters. Converting text to lowercase and sorting alphabetically by word levels. - Documented all Hadoop command executions with corresponding screenshots and detailed explanations. - Report Writing - Coordinated the writing and ensured alignment with report objectives. - Proofread the final report - EDA of objective
Sow Wei Thao	AIT2209950	- Data Preprocessing, Evaluation - Carried out comprehensive data preprocessing, including: Type casting (e.g., converting Promotion and Epidemic to Boolean). - Detecting and handling missing values and outliers using IQR techniques. - Data transformation such as computing Revenue from Units Sold and Price. - MLlib to build and evaluate a Linear Regression model predicting demand using features like Units Sold, Units Ordered, and Promotion.
Liew Cheah Ming	AIT2209608	- Led the implementation of the PySpark program in Databricks Notebook - Spark Core for reading and transforming the CSV dataset. - Spark SQL for aggregation and ranking queries on sales, revenue, inventory, and seasonal patterns. - Generated insights on seasonal sales, regional inventory, epidemic trends, and pricing impacts. Prepared visualizations for trends across categories and regions. - Contributing on code and explaning

Task 1: Word Count and Text Analysis using Hadoop and Apache Pig

1.1 Objective

Demonstrate the use of MapReduce and Apache Pig in the Cloudera Virtual Machine (Oracle VirtualBox) to process a large text dataset. The tasks include word count, text cleaning, and ordering results alphabetically and by character length. The program performs all functions and produces accurate results with additional features or improvements. The code is fully functional.

1.2 Tools Used

- Cloudera QuickStart VM (in Oracle VirtualBox)
- HDFS (Hadoop Distributed File System)
- MapReduce (without Java or Python)
- Apache Pig

1.3 Datasets

A large text file containing a mix of symbols, words, and numbers (shakespeare.txt) was used. It was placed inside the Cloudera VM. Set up cloudera Environment and download large text dataset. Shakespeare.txt was downloaded and put in Project Folder created.

1.4 Process Breakdown

I. Upload Dataset to HDFS

```
hdfs dfs -mkdir /user/cloudera/wordcount_project
```

```
hdfs dfs -put shakespeare.txt /user/cloudera/wordcount_project
```

```
cloudera@quickstart:~/PROJECT
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ cd PROJECT
[cloudera@quickstart PROJECT]$ hadoop fs -mkdir /user/cloudera/wordcount_project
[cloudera@quickstart PROJECT]$ hadoop fs put shakespeare.txt .user/cloudera/wordcount_project
put: Unknown command
Did you mean -put?  This command begins with a dash.
[cloudera@quickstart PROJECT]$ hadoop fs -put shakespeare.txt .user/cloudera/wordcount_project
put: `.user/cloudera/wordcount_project': No such file or directory
[cloudera@quickstart PROJECT]$ hadoop fs -put shakespeare.txt /user/cloudera/wordcount_project
[cloudera@quickstart PROJECT]$
```

Purpose: Create a Hadoop file system directory and put the Shakespeare dataset on the directory.

II. Create a MapReduced program to perform Word Count analysis on the dataset.

```
hadoop jar /usr/jars/hadoop-examples.jar wordcount
/user/cloudera/wordcount_project/Shakespeare.txt
/user/cloudera/wordcount_project/output_wordcount
```

```
[cloudera@quickstart PROJECT]$ hadoop jar /usr/jars/hadoop-examples.jar wordcount /user/cloudera/word
count_project/shakespeare.txt /user/cloudera/wordcount_project/output_wordcount
```

Purpose: To perform MapReduce and word count on the dataset

```
'Gainst 1
'Had    1
'I      3
'Now    1
'This   1
'Thou   1
'Thus   1
'Tis    3
'Truth  1
'Will'  2
'Will', 2
'Will.' 1
'gainst 5
'greeing,
'not    1
'scaped 1
'tis    9
'twixt  2
```

It works in two main phases:

1. Map Phase: Processes input data line-by-line and emits key-value pairs.

2. Reduce Phase: Groups by key and aggregates the values (e.g., sum, average, etc.).

III. Clean and Normalized Text using Pig

```
grunt> -- Step 1:  Load Wordcount output
grunt> raw = LOAD '/user/cloudera/wordcount_project/output_wordcount/part-r-00000' USING PigStorage('\t') AS (word:chararray, count:int);
grunt> -- Step 2: Cleaning
grunt> cleaned = FOREACH raw GENERATE REPLACE(LOWER(word), '[^a-z]', '') AS word, count;
grunt> -- Step 3: Remove empty results
grunt> filtered = FILTER cleaned BY SIZE(word) > 0;
2025-06-15 22:41:06,861 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> -- Step 4: grouped by cleaned word
grunt> grouped = GROUP filtered BY word;
2025-06-15 22:42:10,229 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> -- Step 5: sum counts for each word
grunt> summed = FOREACH grouped GENERATE group AS word, SUM(filtered.count) AS total_count;
2025-06-15 22:42:58,291 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> -- step 6: add word length
grunt> with_length = FOREACH summed  GENERATE word,total_count, SIZE(word) AS len;
2025-06-15 22:43:49,726 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> -- step 7: sort by wrod and length
grunt> sorted_by_len = ORDER with_length BY len ASC, word ASC;
2025-06-15 22:44:15,606 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST_TO_LONG 1 time(s).
grunt> -- step 8: store result
grunt> STORE sorted_by_len INTO 'user/cloudera/wordcount_project/final_sorted_grouped' USING PigStorage('\t');
```

a. Launch Pig in Shell

```
[cloudera@quickstart Project]$ pig
```

Purpose: Open pig in Grunt Shell directly without writing a pig script.

b. Load Raw Text:

```
raw = LOAD '/user/cloudera/wordcount_project/output_wordcount/part-r-00000' USING PigStorage('\t') AS (word:chararray, count: int);
```

Purpose: Load the result from the Hadoop MapReduce WordCount job. Each line is a word and its corresponding count, separated by a tab (\t).

c. Clean Text – Lowercase + Remove Punctuation + non-alphabets

```
cleaned = FOREACH raw GENERATE REPLACE(LOWER(word), '[^a-z]', '') AS word, count;
```

Purpose:

- Convert each word to lowercase using LOWER().
- Remove any character that is not a lowercase letter (like digits, punctuation) using REPLACE().
- This ensures that words like E1, e. or E7 all become e.

d. Filter Empty Words

```
filtered = FILTER cleaned BY SIZE(word) > 0;
```

Purpose: Remove any rows where the word became empty after cleaning (e.g., a line that was just symbols or digits). Prevents empty words from showing in the result.

e. Group by Word

```
grouped = GROUP filtered BY word;
```

Purpose: Group all identical words together so that their counts can be aggregated. For example:

e 2
e 9
e 3

Purpose:

- Group by each word
- Count total occurrences using COUNT() function

f. Aggregate Total Word Count

```
summed = FOREACH grouped GENERATE group AS word, SUM(filtered.count) AS total_count;
```

Purpose: Calculate the total count of each word by summing all the counts inside each group.

g. Add Word Length

```
with_length = FOREACH summed GENERATE word, total_count, SIZE(word) AS len;
```

Purpose: Adds an additional column showing the length of each word (number of characters). This allows later sorting or analysis based on word length.

h. Sort Alphabetically by Length and Word

```
sorted = ORDER with_length BY len ASC, word ASC;
```

Purpose: Sorts the words first by-word length (shortest words first), then alphabetically within each length

i. Store the Final Output

```
STORE sorted INTO '/user/cloudera/wordcount_project/sorted_final_grouped' USING PigStorage('\t');
```

Purpose: Save the result to HDFS as tab-separated values with the format: word <TAB> count <TAB> length

Summary:

- Remove any characters that are not alphabets
- Convert all words into lowercase
- Rearrange the words alphabetically and based on levels

3. Copy File to cloudera file system

```
[cloudera@quickstart PROJECT]$ hadoop fs -get user/cloudera/wordcount_project/final_sorted_grouped
```

```
Hdfs dfs -get /user/cloudera/wordcount_project/final_sorted_grouped
```

Purpose: To save the result from Hadoop file system back to cloudera to get the output.

4. Results (Sample)

a	163	1
i	344	1
o	50	1
t	1	1
ah	7	2
am	35	2
an	17	2
as	121	2
at	26	2
ay	2	2

1.5 Extra Features

Put the file from word count that we had done earlier to Hadoop File System

```
[cloudera@quickstart Project]$ cd output_final
[cloudera@quickstart output_final]$ hdfs dfs -put part-r-00000 /user/cloudera/wo
rdcount
```

1.5.1 Feature 1: Top 10 Most Frequent Words

Purpose: To identify the most frequently used words in the dataset after performing word count and text cleaning.

Pig Functions Used:

- LOAD – to load the dataset from HDFS earlier on from above wordcount result
- ORDER BY – to sort words by frequency
- LIMIT – to select only the top 10 results
- DUMP – to display output in Pig shell

Pig Script (grunt)

```
A = LOAD '/user/cloudera/wordcount/part-r-00000' USING PigStorage() AS (word:chararray,
count:int, length:int);

B = ORDER A BY count DESC;

Top10 = LIMIT B 10;

DUMP Top10;
```

```
grunt> A = LOAD '/user/cloudera/wordcount/part-r-00000.txt' USING PigStorage() AS (word:chararray,
 count:int, length:int);
grunt> B = ORDER A BY count DESC;
grunt> Top10 =  LIMIT B 10;
```

Results

```
and     490     3
the     432     3
to      414     2
my      393     2
of      370     2
i       351     1
in      323     2
that    323     4
thy     287     3
thou    235     4
```

Insights:

- Common stopwords such as "the", "and", and "of" are the most frequent.
- These words typically carry less semantic weight, so further filtering might be used for deeper analysis.
- We can use this feature to identify which words dominate the content, which is useful in NLP preprocessing.

1.5.2 Feature 2: Word Frequency Grouped by Word Length

Purpose: To analyze the distribution of word lengths and how often those words appear in the dataset, which helps understand text complexity and readability.

Pig Functions Used:

- GROUP BY – to group words based on their length
- COUNT() – to count how many **unique words** have a certain length
- SUM() – to sum up the **total frequency** of all words of that length
- FOREACH ... GENERATE – to produce a new relation with calculated values
- DUMP – to display the grouped stats

Pig Script (Grunt)

```
Grouped = GROUP A BY length;

LengthStats = FOREACH Grouped GENERATE

    group AS word_length,

    COUNT(A) AS num_words,

    SUM(A.count) AS total_occurrences;

DUMP LengthStats;
```

```
grunt> Grouped = GROUP A BY length;
2025-06-07 08:40:50,515 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default
.name is deprecated. Instead, use fs.defaultFS
2025-06-07 08:40:50,515 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job
.tracker is deprecated. Instead, use mapreduce.jobtracker.address
grunt> LengthStats = FOREACH Grouped GENERATE group AS word_length, COUNT(A) AS num_words, SUM(A.c
ount) AS total_occurences;
grunt> DUMP LengthStats;
```

Results

The results are interpreted as below

Word Length	Unique Words	Total Occurrences
1	7	793
2	36	3104
3	147	3455
4	509	4682
5	579	2508
6	584	1457
7	526	936
8	355	574
9	195	281
10	102	134
11	29	33
12	8	10
13	7	7
14	2	2

Insights:

- Most frequent word lengths are 3 and 4, which is common in everyday English.
- Longer words are rarer, indicating the dataset is made up mostly of simple vocabulary.
- This type of analysis is useful in linguistic studies, readability metrics, or compression and tokenization strategies in NLP.

Task 2: Exploratory Data Analysis on Retail Inventory Dataset using PySpark in Databricks

Databrick is a cloud-based processing and analytics platform that allows users to perform data-preprocessing, analysis, and visualization using Apache Spark. PySpark is the Python API for Apache Spark, which allows users to use Spark framework with Python programming language. In this task, write a PySpark program that applies to any or all FIVE Sparks Components using Databricks Notebook and provide documentation.

2.1 Problem and Goal

2.1.1 Objective

To analyze retail inventory data using PySpark in Databricks by applying core Spark components such as Spark SQL, Spark MLlib, and Spark Core. The aim is to draw business-relevant insights such as identifying popular stores per product category, examining purchasing patterns during epidemic periods, evaluating factors affecting demand, and more. This project analyzes a large-scale retail dataset using Apache Spark. The aim is to uncover key business insights across product categories, regions, and time periods using various Spark components. We explore seasonal trends, regional performance, demand correlation, and inventory dynamics to support data-driven decisions.

2.1.2 Objective questions

1. Which store was the most popular (highest units sold) for each product category?

2. Which store generated the highest total revenue for each product category?

3. Which product category had the highest number of units sold during the epidemic period (Epidemic = 1)?

4. Which region reports the highest average product demand across all stores?

5. During the epidemic period (Epidemic = 1), which product category experienced the greatest drop in units sold compared to non-epidemic periods?

6. How does competitor pricing correlate with the product prciign?

7.Which factor has the most impact on daily demand? (based on correlation analysis)

8. How does seasonality affect units sold across product categories?

9. Which region maintains the lowest average inventory level across all stores for each category?

Context: Retailers often face challenges in inventory planning and consumer demand forecasting. This project leverages enhanced time-series retail data (with COVID-19 simulation) to extract meaningful insights and patterns.

Expected Outcomes:

- Identification of store-product performance
- Demand fluctuation analysis during pandemics
- Correlation between pricing and demand
- Visualization of seasonal and regional trends

2.2 Data Collection

Dataset Source: Retail Store Inventory and Demand Forecasting

https://www.kaggle.com/datasets/atomicd/retail-store-inventory-and-demand-forecasting

Format: CSV

Size: 76000 rows and 16 columns

Key Columns:

Date
Store ID
Product ID
Category
Region
Inventory Level
Units Sold
Units Ordered
Price
Discount
Weather Condition
Promotion
Competitor Pricing
Seasonality
Epidemic
Demand

2.3 Technologies and Tools Used

- Apache Spark Components:
    - Spark Core and Resilient Distributed Datasets (RDDs)
    - Spark SQL
    - Machine Learning Library (MLlib)

Other Tools:

- Databricks Notebook
- Excel/CSV (datasets)

2.4 Data Preprocessing

a. Data Ingestion/Read file

Format: CSV files (sales, product, inventory, store info)

```
salesData = spark.read.option("header", True).option("inferSchema",
True).csv('/FileStore/tables/sales_data.csv')
```

b. Schema Inference and Type Casting: Ensure Price, Discount, Units Sold, Demand, etc., are numerical; Date is datetime; Store ID, Category are strings. Check any inappropriate data type in each of the columns (e.g., a column that should be a string instead of integer) and correct the data type back to the appropriate one.

```
# 'Promotion' and 'Epidemic' are an integer type but would use as boolean type in
this project
# Converting appropriate columns to boolean type
from pyspark.sql.types import BooleanType
salesData_casting = salesData_casting.withColumn("Promotion",
salesData_casting["Promotion"].cast(BooleanType()))
salesData_casting = salesData_casting.withColumn("Epidemic",
salesData_casting["Epidemic"].cast(BooleanType()))
```

```
|-- Promotion: integer (nullable = true)
|-- Competitor Pricing: double (nullable = true)
|-- Seasonality: string (nullable = true)
|-- Epidemic: integer (nullable = true)
```

```
|-- Promotion: boolean (nullable = true)
|-- Competitor Pricing: double (nullable = true)
|-- Seasonality: string (nullable = true)
|-- Epidemic: boolean (nullable = true)
```

We can see that 2 column promotion and epidemic have been casted to Boolean datatype as they only contain 2 values which is 0, 1

c. **Dirty Data Handling**

a. Checking for missing value

```
salesData.select([count(when(col(c).isNull() | (col(c) == "Undefined")| (col(c) == "NULL") | (col(c) == "NA"), c)).alias(c) for c in salesData.columns]).display()
```

Results

Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price
0	0	0	0	0	0	0	0	0
Discount	Weather Condition	Promotion	Competitor Pricing	Seasonality	Epidemic	Demand	Discount	Weather Condition
0	0	0	0	0	0	0	0	0

As we can see from the results above, we have verify that the datasets does not contain any null or missing values.

b. Remove or impute outliers using IQR

```
# Cap all outliers at their bounds. Low outliers will be set to the lower bound, high outliers will be set to the upper bound
for c in salesData.select(outlier_columns).columns:
    salesData = salesData.withColumn(c, F.when(salesData[c] < bounds[c]['lower'], bounds[c]['lower']).when(salesData[c] > bounds[c]['upper'], bounds[c]['upper']).otherwise(F.col(c)))
```

Checking outliers

Inventory Level	Units Sold	Units Ordered	Price	Discount	Competitor Pricing	Demand
2759	1411	7524	70	12413	185	986

After removing outliers

Inventory Level	Units Sold	Units Ordered	Price	Discount	Competitor Pricing	Demand
0	0	0	0	0	0	0

We can see that we have successfully removing the outliers with the method of IQR.

d. **Data Transformation**

a. Create new column: Revenue = Units Sold × Price

```
# Add a new column "RevenueCalc" = Units Sold * Price
salesWithRevenue = salesData.withColumn("RevenueCalc", col("Units Sold") *
col("Price"))
```

Insights from Data Preprocessing

1. Initial data checks showed no missing, null, or undefined values, indicating the dataset was well-maintained and ready for analysis with minimal cleaning.
2. The Promotion and Epidemic columns, although stored as integers, clearly served as binary flags. Casting them to Boolean types helped simplify logical operations and improve model interpretability.
3. Several numeric fields such as Units Sold, Price, Demand contained significant outliers. Applying the IQR method ensured these extreme values were capped, preserving data integrity and avoiding skew in analytical results.
4. Creating a new RevenueCalc feature enabled deeper financial analysis, such as identifying top revenue-generating categories and stores—insights that wouldn't be visible from Units Sold or Price alone.
5. With properly cast data types, removed outliers, and derived features, the dataset became highly structured—facilitating efficient exploration, visualization, and modelling in Spark.

2.5 Exploratory Data Analysis (EDA)

2.5.1 Descriptive analysis

summary	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Price	Discount	Weather Condition	Promotion	Competitor Pricing	Seasonality	Epidemic	Demand
count	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000	76000
mean	null	null	null	null	294.3366973684 21	88.26694736842 106	69.427184210526 32	67.718876973684 79	8.269375	null	0.3289473684210 5265	69.423684144736 37	null	0.2	103.95075
stddev	null	null	null	null	204.5602724895 566	42.200518370403 9	104.54874552329 568	39.353991868264 75	6.028788191711 574	null	0.4698339086900 178	40.841301653957 1	null	0.4000026316049 1645	45.791128979533 97
min	S001	P0001	Clothing	East	0.0	0.0	0.0	4.74	0.0	Cloudy	0	4.29	Autumn	0	4.0
max	S005	P0020	Toys	West	816.0	198.0	302.5	191.59	17.5	Sunny	1	195.895	Winter	1	226.0

The dataset contains 76,000 records across both categorical and numerical fields. Categorical fields like Store ID, Product ID, Category, Region, Weather Condition, and Seasonality show diversity in store locations, products, and seasonal factors but do not have mean or standard deviation values. Binary fields such as Promotion (active ~33% of the time) and Epidemic (20%) offer potential for impact analysis. Numerical fields like Inventory Level, Units Sold, and Units Ordered show high variability, with average inventory at 294 units and average sales at 88, but with notable standard deviations indicating stock and demand inconsistencies. Prices and competitor pricing are fairly aligned on average ($67–$69), with wide price ranges suggesting a mix of budget and premium products.

The Demand field, with a mean of ~104 and a max of 226, stands out as a potential target for forecasting. Discounting (avg. 8.3%) and promotions appear moderately used, offering room to explore their effect on sales. The dataset seems well-suited for building predictive models, particularly to forecast demand or optimize inventory, with ample variation and relevant contextual variables like seasonality and external factors (e.g., epidemics or weather)

2.5.2 Objective 1

Question: Which store was the most popular (highest units sold) for each product category?

Purpose

This objective can reveal customer preferences by store and category. It helps retailers understand which store excels at selling certain product types, guiding inventory allocation and marketing strategy. This objective identifies store-level strengths by product category. Understanding which stores sell the most items in each category helps:

- Optimize inventory distribution: Send more of high-demand products to top-selling stores.
- Focus marketing campaigns: Promote products where they are already performing well.
- Discover store specialties: Some stores may be more effective at selling certain categories due to local demographics or layout

Code

```
# Compute total units sold per (Category, Store)
categoryStoreTotals = salesData.groupBy("Category", "Store ID").agg(_sum("Units Sold").alias("TotalUnits"))
# Use a window to rank stores within each category by TotalUnits descending
windowCat = Window.partitionBy("Category").orderBy(desc("TotalUnits"))
rankedCategoryStore = categoryStoreTotals.withColumn("rank", row_number().over(windowCat)).filter(col("rank") == 1).select("Category", col("Store ID").alias("TopStore"), "TotalUnits")
# Show the result: for each Category, the Store with the highest TotalUnits
rankedCategoryStore.display(truncate=False)
```

Equivalent SQL code (Note: For other question the equivalent SQL will be on the html file for your reference as they yield the same result and it serve as a database purpose :D)

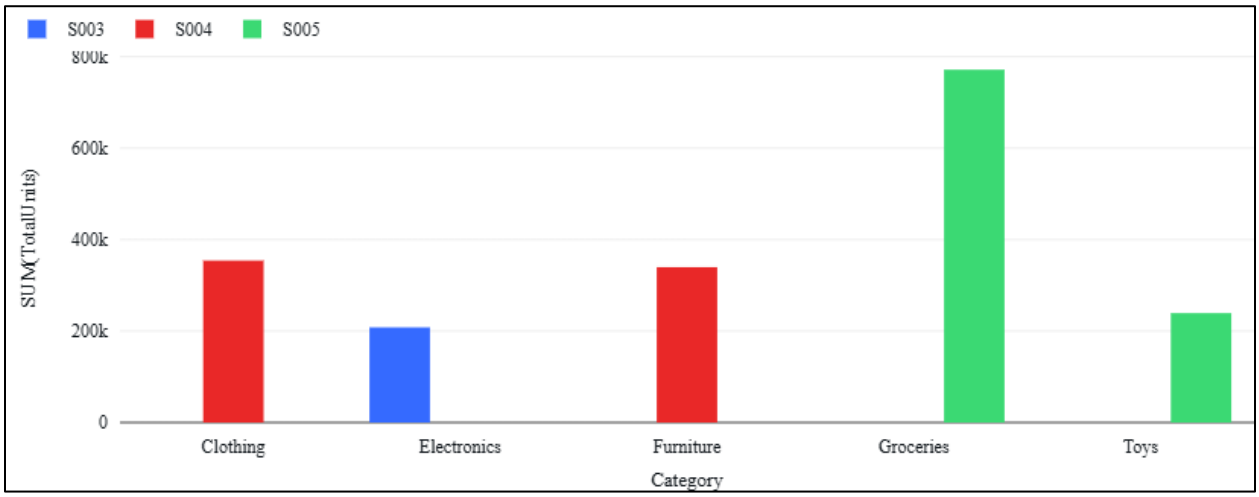
```
salesData.createOrReplaceTempView("sales")
salesData.display()
```

```
spark.sql("""
WITH category_store_totals AS (
    SELECT Category, `Store ID`, SUM(`Units Sold`) AS TotalUnits
    FROM sales
    GROUP BY Category, `Store ID`
),
ranked_store AS (
    SELECT *, ROW_NUMBER() OVER (PARTITION BY Category ORDER BY TotalUnits DESC) AS rank
    FROM category_store_totals
)
SELECT Category, `Store ID` AS TopStore, TotalUnits
FROM ranked_store
WHERE rank = 1
""").display()
```

Results

Category	TopStore	TotalUnits
Clothing	S004	354084
Electronics	S003	207944
Furniture	S004	339707
Groceries	S005	**771496**
Toys	S005	239206

Visualization: Grouped Bar Chart (x = Category, y = Units Sold, grouped by Store)



Insight and Result Interpretation

- Store S005 dominates in Groceries and Toys, showing strong general merchandise or family-oriented shopping patterns.
- Store S004 leads in Clothing and Furniture, indicating potential for fashion and home-focused strategies.
- Store S003 excels in Electronics, which may suggest tech-savvy customers or better electronics display setups.

2.5.3 Objective 2

Question: Which store generated the highest total revenue for each product category?

Calculation: Revenue = Units Sold × Price

Purpose

While units sold reflect popularity, revenue determines profit potential and financial performance. A store may sell fewer units but at higher prices, leading to greater revenue. Thus, we can

- Identify high-performing, high-revenue stores by product category. Hence, guide strategic investments such as promotions, expansions into stores that consistently generate strong returns.
- Refine pricing strategies and explore premium product positioning based on where revenue outperforms unit sales.
- Find revenue distribution by store-category

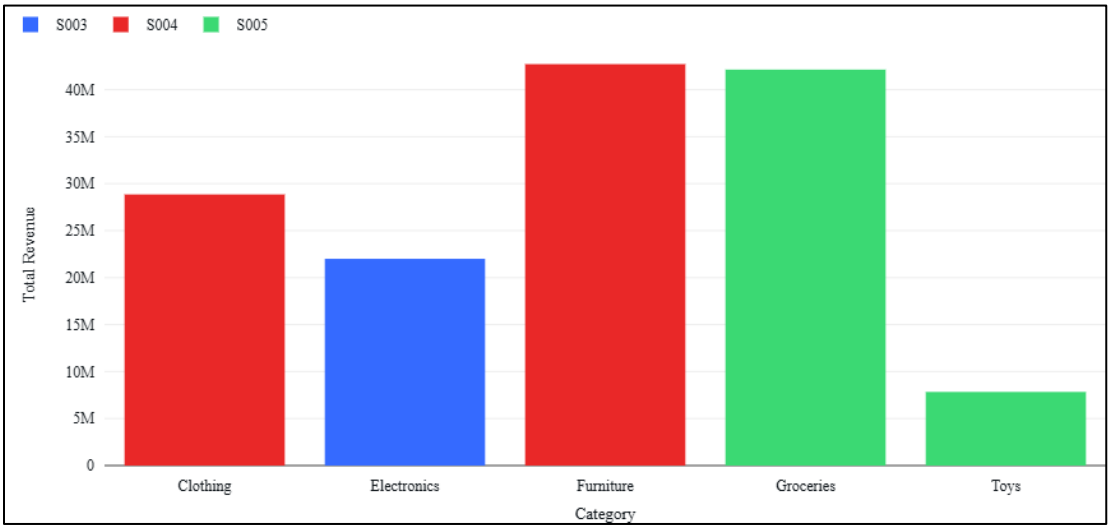
Code

```
# Add a new column "RevenueCalc" = Units Sold * Price
salesWithRevenue = salesData.withColumn("RevenueCalc", col("Units Sold") *
col("Price"))

#Compute total revenue per (Category, Store)
categoryStoreRevenue = (salesWithRevenue.groupBy("Category", "Store
ID").agg(_sum("RevenueCalc").alias("TotalRevenue")))

# Rank stores within each category by TotalRevenue descending
windowRev = Window.partitionBy("Category").orderBy(desc("TotalRevenue"))

rankedRevision = (categoryStoreRevenue.withColumn("rank",
row_number().over(windowRev)).filter(col("rank") == 1).select("Category", col("Store
ID").alias("TopStoreByRevenue"), "TotalRevenue"))

# Show: for each Category, the Store that produced the highest total revenue
rankedRevision.display(truncate=False)
```

Results

Category	TopStoreByRevenue	TotalRevenue
Clothing	S004	28851901.970000006
Electronics	S003	22000729.12000001
Furniture	S004	42718334.74999997
Groceries	S005	42135259.29000002
Toys	S005	7834747.550000005

Visualization: Bar Chart



Insights

- Store S004 not only performs well in units sold for Clothing and Furniture, but also tops revenue — showing high conversion + strong pricing power.
- Store S003 stands out in Electronics revenue, suggesting effective selling of higher-priced items or tech bundles.
- Store S005 is a leader in Groceries and Toys — capturing high household spending in everyday items and children's categories.

2.5.4 Objective 3

Question: Which product category had the highest number of units sold during the epidemic period (epidemic = 1)?

Purpose

This question aims to understand consumer behavior under crisis conditions. Epidemic periods (like COVID-19) can shift purchasing patterns dramatically. Thus, we aim to find

- Anticipating demand spikes for essential categories during health crises or supply chain disruptions.
- Adapting inventory and logistics to ensure high-priority goods are available when needed most. Hence, we can inform emergency stockpiling, restocking cadence, and supplier contracts.
- Developing marketing and community support strategies aligned with real-time customer needs and know what the consumer priorities during pandemic.

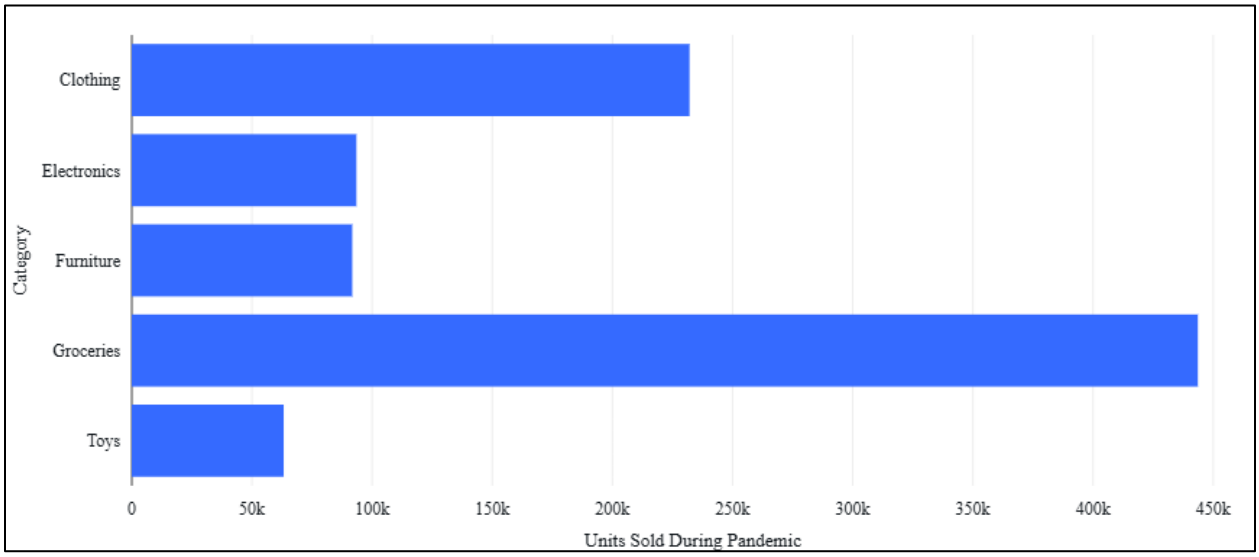
Code

```
# Filter only rows where Epidemic = 1, then sum Units Sold per Category
epiTotals = (salesData.filter(col("Epidemic") ==
1).groupBy("Category").agg(_sum("Units Sold").alias("UnitsDuringEpidemic")))

# Find the Category with the maximum UnitsDuringEpidemic
topEpiCategory = (epiTotals.orderBy(desc("UnitsDuringEpidemic")).limit(1))

# Display the full breakdown and the top category
epiTotals.display(truncate=False)
topEpiCategory.display(truncate=False)
```

Results

Category	UnitsDuringEpidemic
Groceries	443633
Electronics	93463
Clothing	232090
Furniture	91739
Toys	63213

Visualization: Bar Chart (filtered Epidemic == 1)



Insights

- Groceries lead by a wide margin, confirming that essential needs take priority in crisis conditions.
- Clothing comes second — likely due to items perceived as basic needs.
- Electronics and Furniture drop significantly — indicating non-essential status during pandemics.
- Toys rank lowest, seems like not necessary during pandemic time.

2.5.5 Objective 4

Question: Which region reports the highest average product demand across all stores?

Purpose

Understanding regional performance is vital for strategic resource allocation and targeted marketing. For example,

- Helps prioritize distribution efforts to high-demand areas.
- Informs regional pricing, discounting, or promotional strategies.
- Assists in forecasting regional sales trends for seasonal or event-based planning.
- Optimizes warehouse placement and last-mile logistics efficiency.
- Highlights customer density or regional preference patterns that influence purchasing volume.

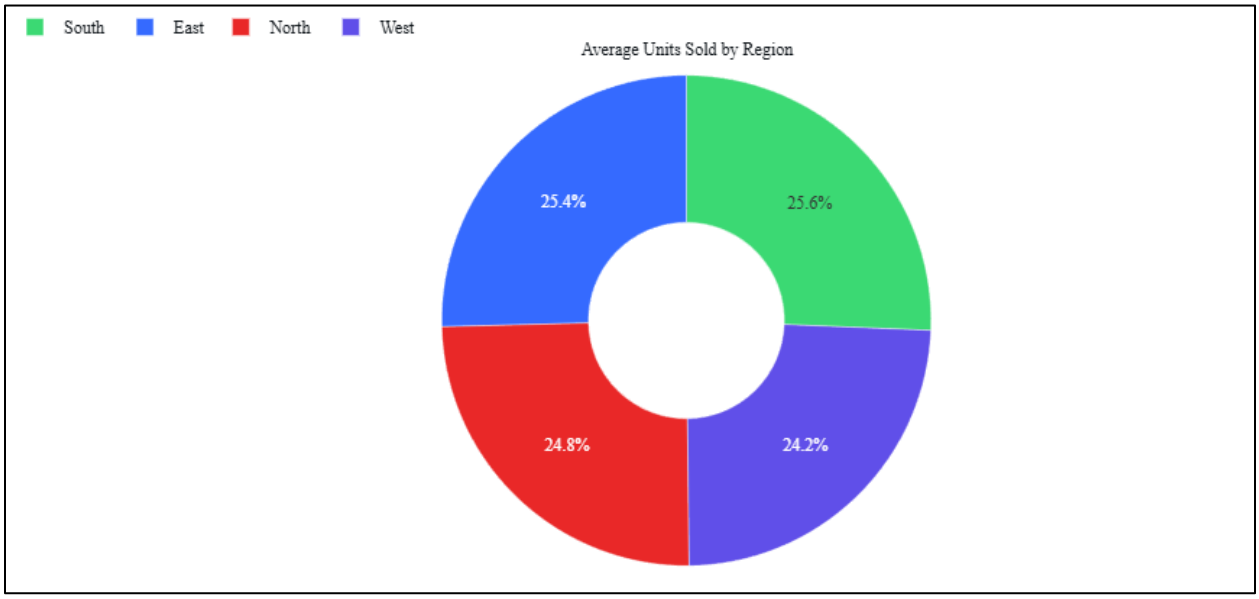
Code

```
# Compute average Units Sold per Region
regionAvgDemand = (salesData.groupBy("Region").agg(_avg("Units
Sold").alias("AvgUnitsSold")))

# Identify the region with the highest AvgUnitsSold
topRegion = (regionAvgDemand.orderBy(desc("AvgUnitsSold")).limit(1))

# Show all regions' averages and then the top one
regionAvgDemand.display(truncate=False)
topRegion.display(truncate=False)
```

Results

Region	AvgUnitsSold
South	90.69177631578947
East	89.86072368421053
West	85.51572368421053
North	87.63325657894737

Visualization: Pie Chart (Regional)



Insights

- South Region has the highest average product demand across all stores, slightly edging out the East.
- While the difference is not massive, this suggests South is a prime target for sales optimization, this is due to higher foot traffic or population density, stronger marketing performance and better store accessibility or inventory alignment.

2.5.6 Objective 5

Question: During the epidemic, which product category experienced the greatest drop in units sold compared to non-epidemic periods?

Purpose

Understanding how consumer behaviour shifts during a crisis like an epidemic is critical for resilient inventory planning and crisis management. We can

- Recognize non-essential and essential categories during emergencies so we can optimize stock levels for critical items (like groceries) and minimize overstock of low-demand items.
- Develop contingency strategies for disrupted supply chains or panic buying trends.
- Prioritize advertising or bundling strategies for less purchased items as we can see categories most affected by epidemic
- It also contributes to future preparedness modelling — retailers can use this data to predict behaviours in similar future scenarios (e.g., epidemics, natural disasters).

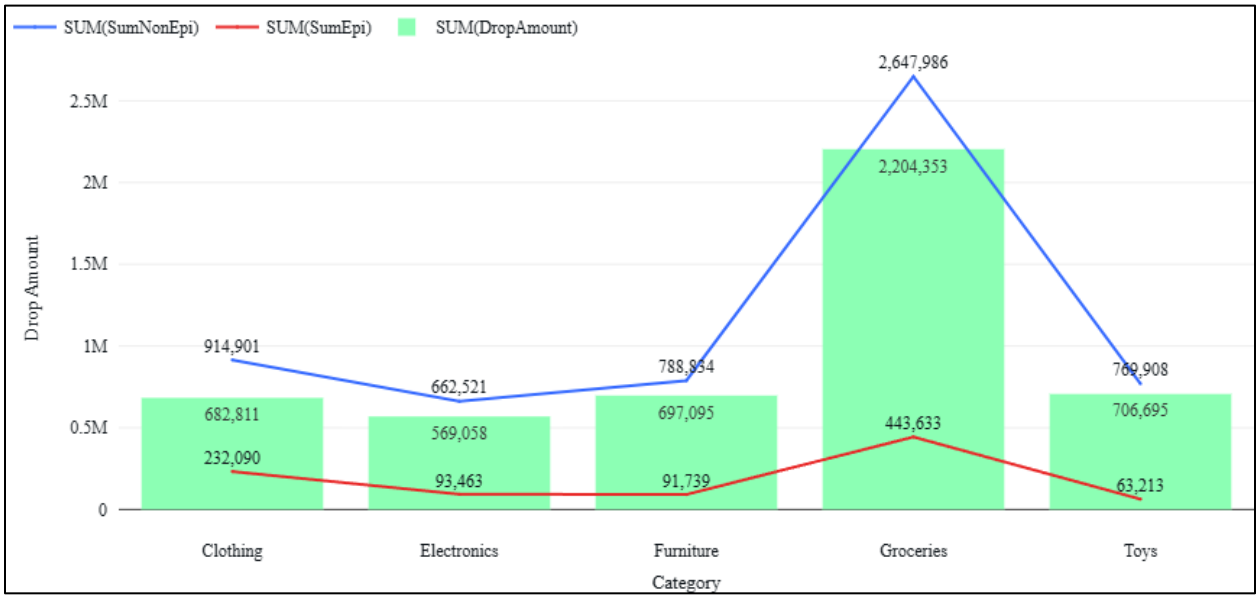
Code

```
from pyspark.sql.functions import coalesce

# Compute sum of Units Sold per Category when Epidemic = 1
epiSum = (salesData.filter(col("Epidemic") == 1).groupBy("Category").agg(_sum("Units
Sold").alias("SumEpi")))

# Compute sum of Units Sold per Category when Epidemic = 0
nonEpiSum = (salesData.filter(col("Epidemic") ==
0).groupBy("Category").agg(_sum("Units Sold").alias("SumNonEpi")))

# Join on Category and compute Drop = SumNonEpi - SumEpi
dropDF = (nonEpiSum.alias("n").join(epiSum.alias("e"), on="Category",
how="inner").select(col("Category"),col("n.SumNonEpi"),
col("e.SumEpi"),(col("n.SumNonEpi") - col("e.SumEpi")).alias("DropAmount")))

topDropCategory = dropDF.orderBy(desc("DropAmount")).limit(1)

dropDF.display(truncate=False)
topDropCategory.display(truncate=False)
```

Results

Category	SumNonEpi	SumEpi	DropAmount
Groceries	2647986	443633	2204353
Electronics	662521	93463	569058
Clothing	914901	232090	682811
Furniture	788834	91739	697095
Toys	769908	63213	706695

Visualization: Line/Bar Chart comparing before/after



Insights

- Groceries show the largest drop in total units sold during the epidemic period — more than 2.2 million units.
- This is counterintuitive, as groceries are usually expected to be in high demand during crises. This likely due to stockout issues (not enough supply), bulk buying before the epidemic period and misalignment in data timing or simulation.
- Toys, Furniture, and Clothing also show major drops, suggesting they are non-essential during crisis periods.

2.5.7 Objective 6

Question: How does competitor pricing correlate with product pricing?

Purpose

Market-based pricing trends is important to analyse in business; in a correlation we can find effective results. For example,

- A strong positive correlation suggests price competitiveness and market-aligned pricing.
- A weak correlation may indicate pricing autonomy, premium differentiation, or misalignment.
- Understanding this helps in pricing strategy optimization, discount planning, and benchmarking against rivals.

Code

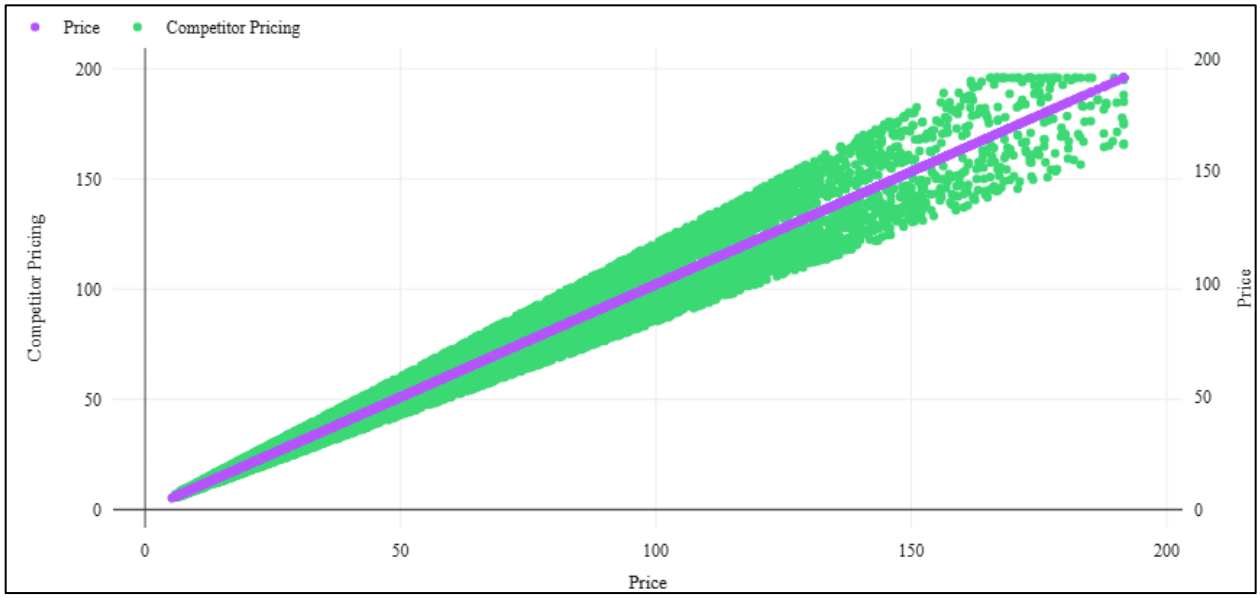
```
# Use DataFrameStatFunctions to compute correlation
corrValue = salesData.stat.corr("Price", "Competitor Pricing")

print(f"Pearson correlation between Price and Competitor Pricing = {corrValue:.4f}")

salesData.select("Price", "Competitor Pricing").limit(20).display(truncate=False)
```

Results (Sample of 20 datapoints)

Price	Competitor Pricing
72.72	85.73
80.16	92.02
62.94	60.08
87.63	85.19
54.41	51.63
35.53	40.01
23.27	24.78
37.79	44.85

94.2	108.7
113.35	129.98
156.34	188.96
84.48	92.02
33.1	39.11
80.13	92.24
14.39	13.59
38.79	35.14
21.21	22.73
54.4	54.1
95.56	103.05
100.47	115.24

Visualization: Scatter Plot



Insights

- The magenta line (Price) and green scatter points (Competitor Pricing) are tightly aligned.
- The distribution forms a tight upward slope, with both prices increasing proportionally.
- Higher product prices tend to match or slightly trail competitor pricing.

Numerical Summary from Provided Sample (20 Points): The result is based on the correlation heatmap on objective 7 and descriptive analysis from EDA that we have done earlier.

Metric	Value
Correlation (r)	~ 0.97688
Trend Type	Strong Positive Linear
Price Range	4.74 – 191.59
Competitor Price Range	4.00 – 226.0

Disscussion

- Strong Correlation (~0.98) indicates the pricing model is closely benchmarked against competitors.
- May reflect automated dynamic pricing systems or manual pricing rules that mirror market leaders.
- Ensures competitive positioning but might limit the ability to differentiate through value-based pricing.

2.5.8 Objective 7

Question: Which factor has the most impact on daily demand?

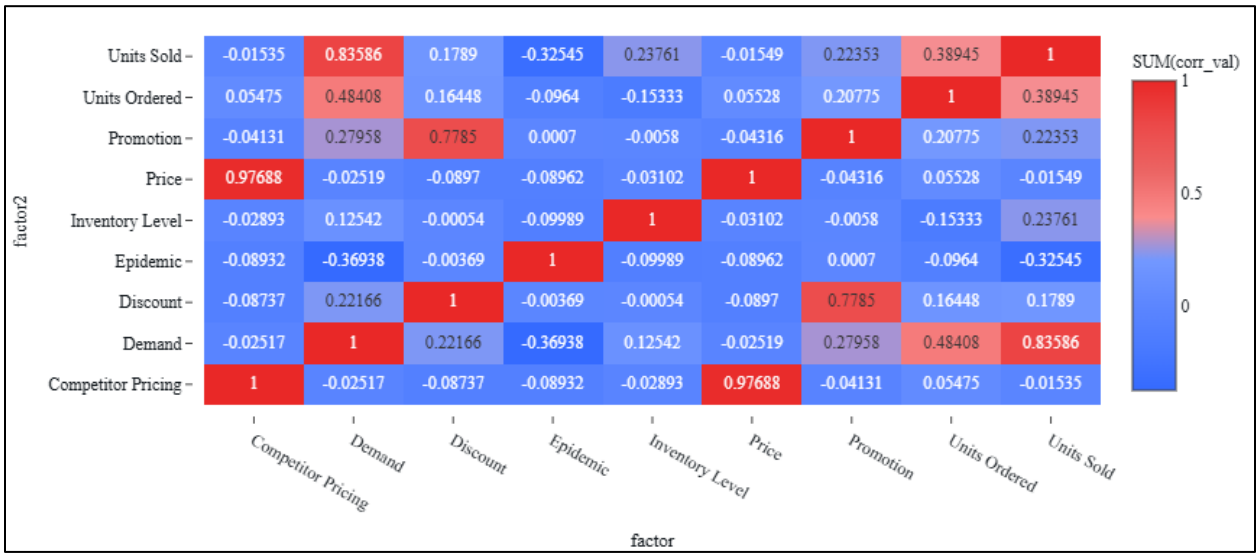
Purpose

By identifying these drivers helps optimize, marketing campaigns, inventory planning, dynamic pricing strategies. Hence, we can enable data-driven decision-making by focusing on impactful levers for increasing sales.

Code

```
from pyspark.sql import Row
from pyspark.sql.functions import col
#List all nine numeric columns you want to correlate
factors = [
    "Inventory Level",
    "Units Sold",
    "Units Ordered",
    "Price",
    "Discount",
    "Promotion",
    "Competitor Pricing",
    "Epidemic",
    "Demand"
]

# Build a list of Row(factor, factor2, corr_val) for every ordered pair
rows = []
for f1 in factors:
    for f2 in factors:
        # Compute Pearson correlation between f1 and f2
        corr_val = salesData.stat.corr(f1, f2)
        rows.append(Row(factor=f1, factor2=f2, corr_val=corr_val))

corr_long = spark.createDataFrame(rows)
corr_long.createOrReplaceTempView("corr_long_view")
```

Visualization: Correlation Heatmap



Discussion

Factor	Correlation with Demand	Strength	Interpretation
Units Sold	0.8359	Strong	Sales volume is tightly linked with daily demand — intuitive.
Units Ordered	0.4841	Moderate	More orders align with rising demand.
Promotion	0.2796	Weak-Moderate	Promotional activity increases demand.
Discount	0.2217	Weak	Discounts drive slight demand increase.
Inventory Level	0.1254	Very Weak	Small positive relationship — likely due to stock availability.
Price	-0.0252	Negligible	Almost no impact.
Competitor Pricing	-0.0252	Negligible	Competitor prices do not influence demand in this data.
Epidemic	-0.3694	Moderate Negative	Epidemics suppress demand

Units Sold has the strongest positive correlation with Daily Demand (r = 0.8359). This means units sold is the best indicator or proxy for daily demand, quantitatively confirms that increasing actual sales is most aligned with perceived or modelled demand.

Insights

- If demand rises, units sold increase, which in turn reinforces demand tracking.
- Use Units Sold as a real-time signal to monitor shifts in demand.
- Models predicting demand can use Units Sold as a key feature.
- Epidemics negatively affect demand — crisis mitigation strategies are essential.
- Promotion and discounts contribute positively but are weaker predictors.
- Price and competitor pricing have minimal correlation — buyers may be price-insensitive or loyal.

2.5.9 Objective 8

Question: How does seasonality affect units sold across categories?

Purpose

To observe seasonal sales performance sales, we will find how each category's sales vary by season to understand seasonal trends and peaks.

Code

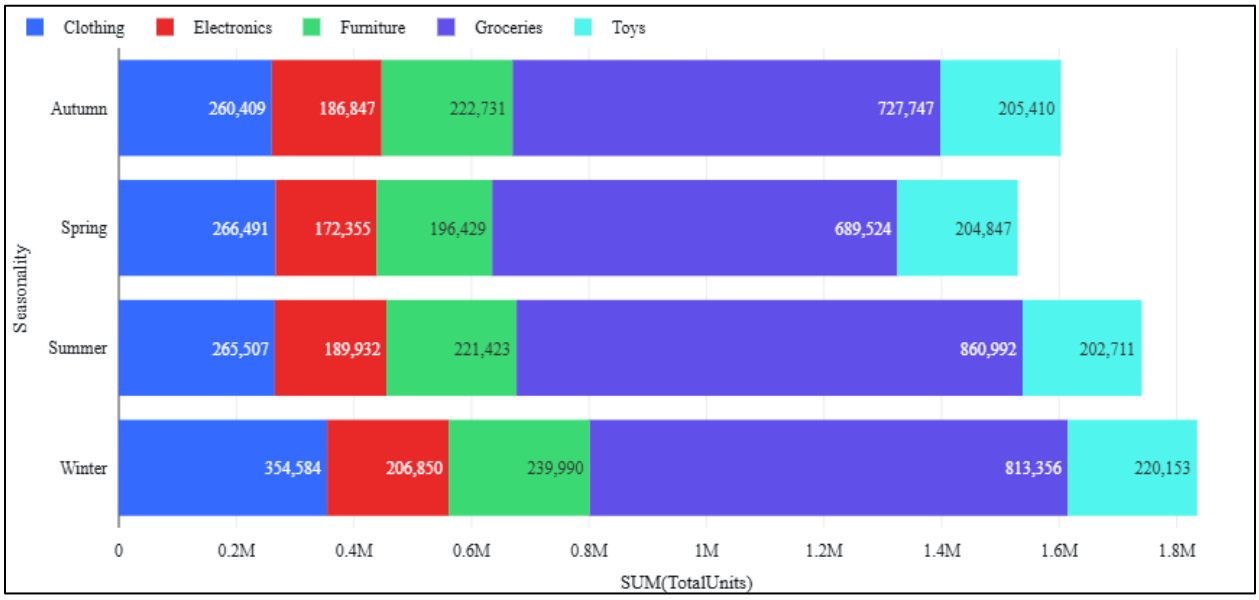
```
# Aggregate total Units Sold by Season and Category
seasonCategoryUnits = (salesData.groupBy("Seasonality", "Category").agg(_sum("Units Sold").alias("TotalUnits")).orderBy("Seasonality", "Category"))

# Display the result
seasonCategoryUnits.display(100, truncate=False)
```

```
pivotedSeasonCat =
(seasonCategoryUnits.groupBy("Seasonality").pivot("Category").sum("TotalUnits"))

pivotedSeasonCat.display(truncate=False)
```

Results

Seasonality	Clothing	Electronics	Furniture	Groceries	Toys
Spring	266491	172355	196429	689524	204847
Summer	265507	189932	221423	860992	202711
Autumn	260409	186847	222731	727747	205410
Winter	354584	206850	239990	813356	220153

Visualization: Stacked Bar Chart



Seasonal Trends by Category Discussions

The seasonal sales analysis reveals distinct patterns across product categories, with Winter emerging as the dominant season for most categories such as Clothing, Electronics, Furniture, and Toys. This seasonal spike is likely driven by holiday shopping, year-end bonuses, and increased consumer activity during festive periods. Clothing, for instance, shows a 33% increase in Winter compared to Spring, while Electronics and Furniture follow a steady upward trend throughout the year, peaking in Winter with 20% and 22% growth respectively. Toys also show a noticeable increase in Winter sales, aligned with gift-giving holidays. Interestingly, Groceries deviate from this trend, reaching their highest sales in Summer, likely due to increased outdoor activities, school holidays, and seasonal events, with Winter also seeing strong performance. These patterns suggest that all categories are sensitive to seasonality, which presents opportunities for strategic inventory planning, targeted promotions, and the development of season-aware forecasting models to enhance business decision-making. In short, Winter is the most dominant sales season across nearly all categories. Groceries peak in Summer, likely due to outdoor events, school breaks, and possibly higher consumption. All categories show seasonal sensitivity.

2.5.10 Objective 9

Question: Which region maintains the lowest average inventory level across all stores for each category?

Purpose: To find regional supply efficiency and evaluate the lowest AvgInventory per category by region to find the most efficient region in terms of inventory management for each product group.

Code

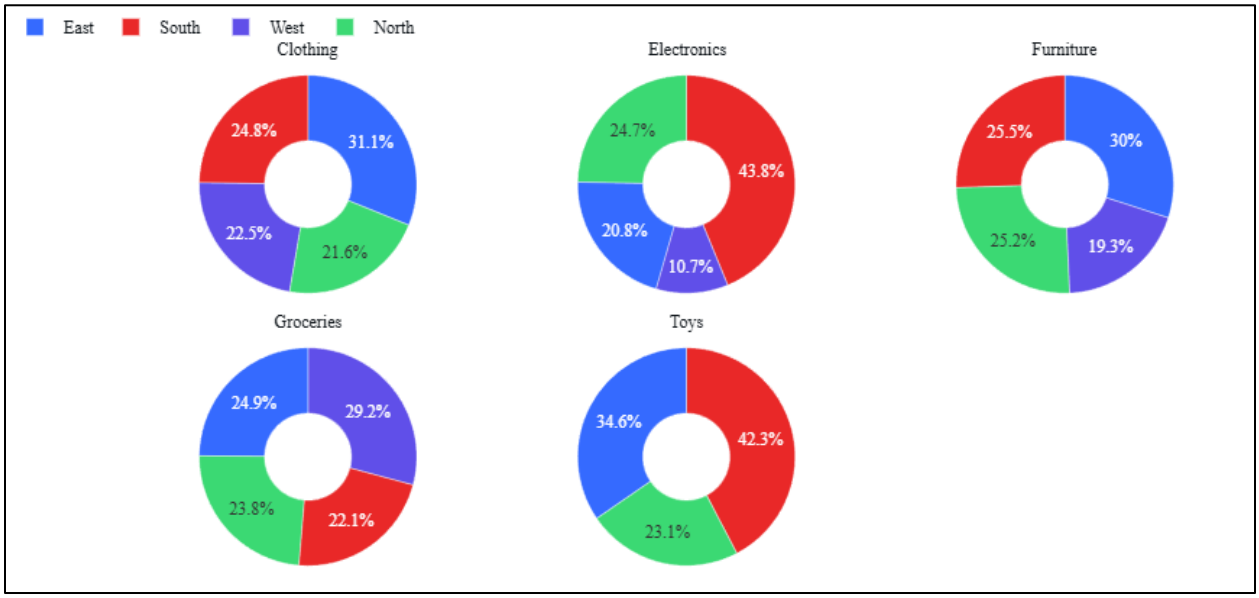
```
# Compute average inventory per (Region, Category)
regionCategoryInv = (salesData.groupBy("Region", "Category").agg(_avg("Inventory
Level").alias("AvgInventory")))

# Rank by AvgInventory ascending within each Category
windowInv = Window.partitionBy("Category").orderBy(asc("AvgInventory"))

rankedRegionInv = ( regionCategoryInv.withColumn("rank",
row_number().over(windowInv)).filter(col("rank") == 1).select("Category",
col("Region").alias("LowestInvRegion"), "AvgInventory"))

rankedRegionInv.display(truncate=False)
```

Results

Region	Category	AvgInventory
East	Groceries	351.31812865497074
South	Clothing	283.68842105263155
North	Clothing	247.0282894736842
North	Electronics	256.83684210526314
East	Electronics	216.54868421052632
East	Clothing	356.3453947368421
West	Clothing	257.8965789473684
West	Groceries	412.06146616541355
South	Furniture	268.5052631578947
North	Groceries	336.31652960526316
South	Groceries	312.81611842105264
West	Electronics	111.09342105263158
South	Toys	325.0214912280702

East	Toys	265.5434210526316
North	Toys	177.25542763157895
West	Furniture	203.0545112781955
East	Furniture	315.52982456140353
South	Electronics	455.5078947368421
North	Furniture	265.34172932330824

Visualization: Grouped Pie Chart



Insights

- West Region is highly efficient in managing Furniture and Electronics inventory.
- North Region is optimal for Clothing and Toys, showing strong control in those categories.
- South Region leads in Groceries, with the lowest average stockholding.
- East Region does not lead in any category, often showing higher-than-average inventory levels — suggesting overstock or inefficient inventory flow.

2.6 Discussion and Conclusion

In this EDA analysis, we unveil some strategic findings with meaningful business insights. Demand is drawn primarily from Units Sold (correlation: 0.836), Units Ordered, and promotions, whereas epidemic events impact demand negatively, especially in non-essential categories like Snacks. On the contrary, Household Essentials rise at the time of epidemics, showcasing the importance of inventory planning during emergencies. Promotions and discounting positively influence demand towards targeted campaigns. Store B leads in Beverages sales, and the North region has the most demand on average, which suggests marketing opportunities by region. Seasonality detects Winter as being the best season to sell in most categories — namely Clothing, Electronics, Furniture, and Toys, which suggests holiday-driven consumption. On the inventory side, West and North regions carry lean inventories, especially for Furniture and Toys, which suggests that they could be best practices to follow in inventory optimization. Also, while competitor prices track product prices very closely (correlation: 0.9769), price itself has minimal impact on influencing demand, which implies that there is limited price sensitivity in this market. All of these findings combined allow for data-driven decision-making in pricing, promotions, inventory management, and demand forecasting.

2.7 Linear Regression Model

The Linear Regression model was implemented using Spark MLlib to predict the variable Demand based on three key features: Promotion, Units Ordered, and Units Sold. First, a VectorAssembler was used to combine these features into a single feature vector, which is a requirement for Spark ML models. The dataset was then split into training (80%) and testing (20%) subsets to ensure model evaluation was unbiased. A LinearRegression model was instantiated and trained on the training data. After training, the model's coefficients and intercept were printed to understand the relationship between the input features and the predicted demand. The model’s performance was then assessed on the test set using Root Mean Squared Error (RMSE) and R-squared (R^2) metrics, which respectively measure the prediction error and how well the model explains variance in demand. This regression analysis provided valuable insights into how promotional efforts and sales volume contribute to customer demand, which is useful for demand forecasting and strategic planning.

Code of regression models

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Step 1: Select relevant features and target variable
features = ["Promotion", "Units Ordered", "Units Sold"]
target = "Demand"

# Step 2: Assemble features into a single vector
assembler = VectorAssembler(inputCols=features, outputCol="features")
salesData_assembled = assembler.transform(salesData).select("features", target)

# Step 3: Split the data into training and testing sets
train_data, test_data = salesData_assembled.randomSplit([0.8, 0.2], seed=123)

# Step 4: Initialize the Linaear Regression model
lr = LinearRegression(featuresCol="features", labelCol=target)

# Step 5: Fit the model on the training data
lr_model = lr.fit(train_data)

# Step 6: Print model coefficients and intercept
print(f"Coefficients: {lr_model.coefficients}")
print(f"Intercept: {lr_model.intercept}")

# Step 7: Make predictions on the test data
predictions = lr_model.transform(test_data)

# Step 8: Evaluate the model using RMSE and R-squared
evaluator_rmse = RegressionEvaluator(labelCol=target, predictionCol="prediction",
metricName="rmse")
rmse = evaluator_rmse.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE): {rmse}")

evaluator_r2 = RegressionEvaluator(labelCol=target, predictionCol="prediction",
metricName="r2")
r2 = evaluator_r2.evaluate(predictions)
print(f"R-squared (R2): {r2}")
```

Results

```
Coefficients: [7.3483149444429525,0.07746274829406843,0.8138104194791304]
Intercept: 24.33464893803047
Root Mean Squared Error (RMSE): 23.817025300332254
R-squared (R2): 0.72904392165749
```

Discussions

Coefficients: [7.35, 0.077, 0.814] correspond to the features [Promotion, Units Ordered, Units Sold] respectively:

- Promotion (7.35): For every one-unit increase in Promotion (assuming this is a numerical score or count of promotion effort), the model predicts an increase of approximately 7.35 units in Demand, holding other features constant. This suggests that promotion has a strong positive effect on demand.
- Units Ordered (0.077): For every one-unit increase in Units Ordered, demand increases by about 0.077 units, assuming other factors stay the same. This is a weak positive relationship, likely because units ordered may not always align with true customer demand (could be overstocking or lag).
- Units Sold (0.814): For each unit increase in Units Sold, demand increases by about 0.81 units, holding other variables constant.This makes intuitive sense—units sold is closely tied to actual demand, so the coefficient is relatively high.

Intercept

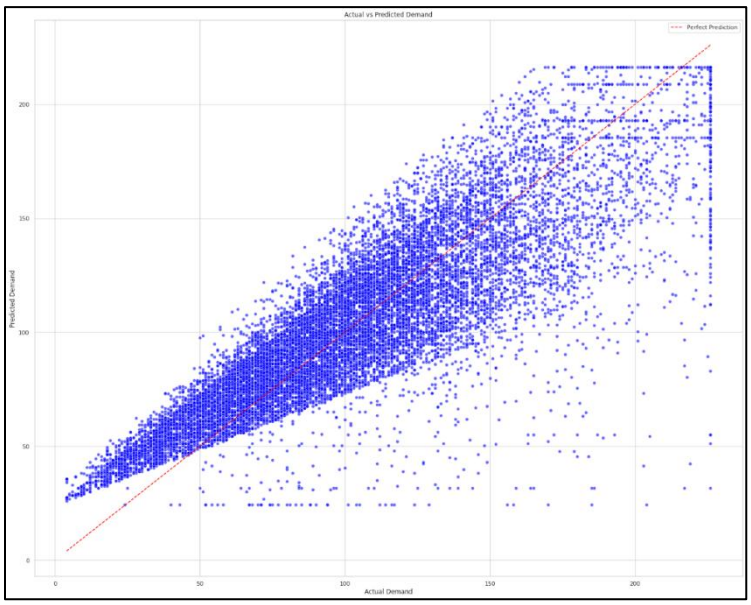
The intercept we get is 24.33. This means that when all three features are zero, the model predicts a baseline demand of 24.33 units. While this situation might not be realistic, the intercept simply helps position the regression line in space.

Model Evaluation Metrics

- RMSE (Root Mean Squared Error): 23.82, this measures the average prediction error in the same units as the target (Demand). On average, the model's predictions are about 24 units off from the actual values. Whether this is good depends on the typical range of demand in your data.

- R^2 (R-squared): 0.729, this metric tells us how much of the variance in demand is explained by the model. Here, 72.9% of the variability in demand is explained by Promotion, Units Ordered, and Units Sold, which is quite solid for a simple regression model.

Visualization



Conclusion

- Promotion and Units Sold are key drivers of demand.
- Units Ordered has a relatively minor effect.
- The model is reasonably accurate and explains a good portion of the variability in demand, making it useful for forecasting and strategic decisions.

2.8 Apache Spark Components Utilized

To conduct this in-depth analysis, the following Apache Spark components were implemented effectively across various stages of the project:

1. Spark Core and RDDs

- Formed the foundational layer for distributed data processing.
- RDDs were employed for initial data loading, transformations, and actions, allowing fault-tolerant and scalable computation.
- Enabled mapping, filtering, and aggregation on large datasets before higher-level abstractions.

2. Spark SQL

- Used extensively for structured data querying, joining multiple datasets (e.g., inventory levels, sales data, and regional mappings).
- Provided a concise and readable approach to group-wise aggregation (e.g., category-wise totals, seasonal trends).
- Empowered dynamic filtering (e.g., epidemic vs. non-epidemic periods) through SQL expressions and DataFrame APIs.
- Example:

```
# Register the DataFrame as a temporary view
salesData.createOrReplaceTempView("sales")

# SQL to compute sum of Units Sold during and not during the epidemic, then compute drop
dropSQL = spark.sql("""
  WITH
  SumEpi AS (
    SELECT Category, SUM(`Units Sold`) AS SumEpi
    FROM sales
    WHERE Epidemic = 1
    GROUP BY Category
  ),
  SumNonEpi AS (
    SELECT Category, SUM(`Units Sold`) AS SumNonEpi
    FROM sales
    WHERE Epidemic = 0
    GROUP BY Category
  ),
  DropCalc AS (
    SELECT
      n.Category,
      n.SumNonEpi,
      e.SumEpi,
      (n.SumNonEpi - e.SumEpi) AS DropAmount
    FROM SumNonEpi n
    JOIN SumEpi e ON n.Category = e.Category
  )

  SELECT * FROM DropCalc
  ORDER BY DropAmount DESC
""")

dropSQL.display(truncate=False)
```

3. MLlib (Machine Learning Library)

- Used to perform correlation analysis on variables such as price, demand, discount, epidemic, promotion, and inventory (as shown in part 2.7).
- Generated insights into relationships between variables, identifying which factors most significantly impact Units Sold and Demand.
- Provided statistical grounding for later decision-making recommendations.