SHRI MADHWA VADIRAJA INSTITUTE OF TECHNOLOGY

AND MANAGEMENT

Vishwothamanagar, Bantakal – 574 115, Udupi, Karnataka, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Big Data Analytics Assignment(21CS71)

(2024-2025)

Assignment - 2

SI.NO.	STUDENT NAME	USN	SIGNATURE
1	VAISHNAVI RAO.B	4MW21CS113	

Name and Signature of the

Faculty:Ms. Sowmya NH

1. Explore the dataset of the New Car Sales in Norway. The dataset is open source and downloadable from at www.kaggle.com. The dataset contains monthly car sales for 2007– 2017 by make and most popular models. Norway New Car Sales dataset can be used for analyzing and predicting future car sales. Explore the dataset to solve the following problems for analysis, prediction and visualization using Sklearn and PySpark:

a. Print year-wise total car sales and visualize the output (Hint: use bar chart for Year vs. total car sales).

b. Print monthly total car sales and visualize for a specified year.

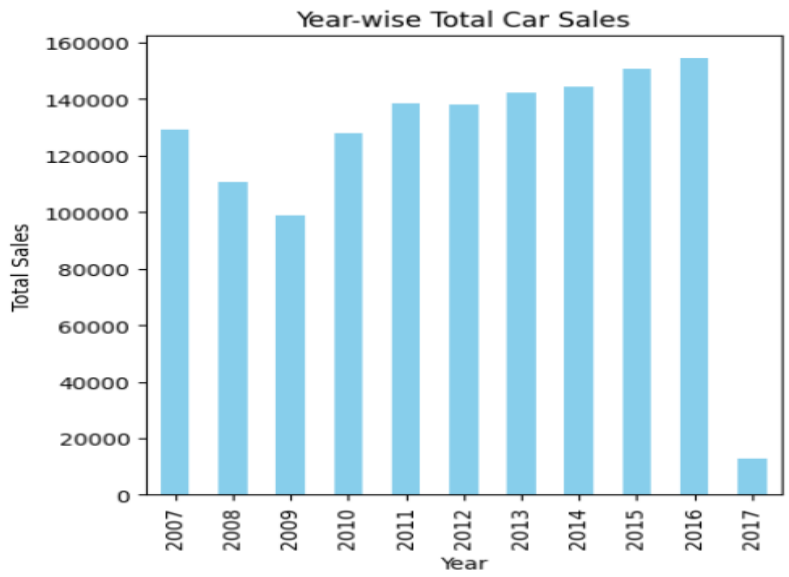
c. Draw pie chart for the sales of all the models of "Toyota" in year 2012.

d. Compare Car models with Percentage shares

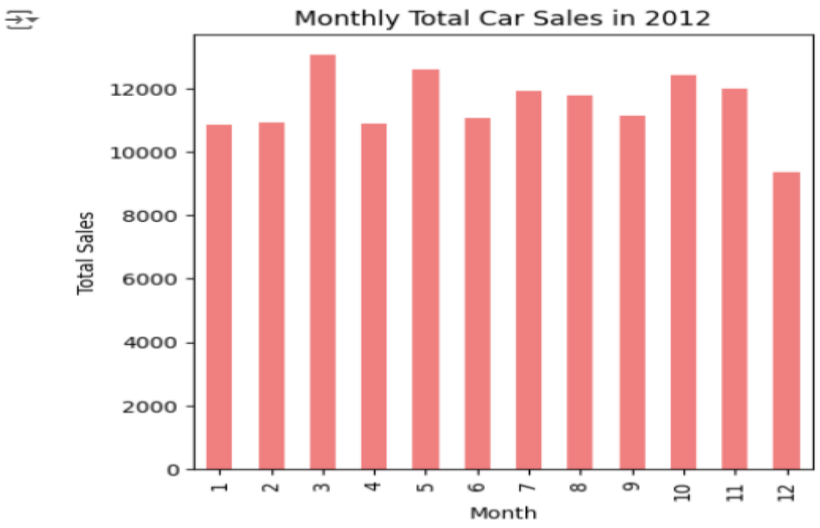
Solution:

```
import pandas as pd
import matplotlib.pyplot as plt

# a: Print year-wise total car sales and visualize the output
data_make = pd.read_csv('/content/norway_new_car_sales_by_make.csv')
year_sales = data_make.groupby('Year')['Quantity'].sum()

year_sales.plot(kind='bar', color='skyblue', title='Year-wise Total Car Sales',figsize=(5, 5))
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.show()
print(year_sales)

# b: Print monthly total car sales for a specified year (e.g., 2012)
data_month = pd.read_csv('/content/norway_new_car_sales_by_month.csv')
monthly_sales_2012 = data_month[data_month['Year'] == 2012].groupby('Month')['Quantity'].sum()

monthly_sales_2012.plot(kind='bar', color='lightcoral', title='Monthly Total Car Sales in 2012',figsize=(5, 5))
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.show()
print(monthly_sales_2012)
```

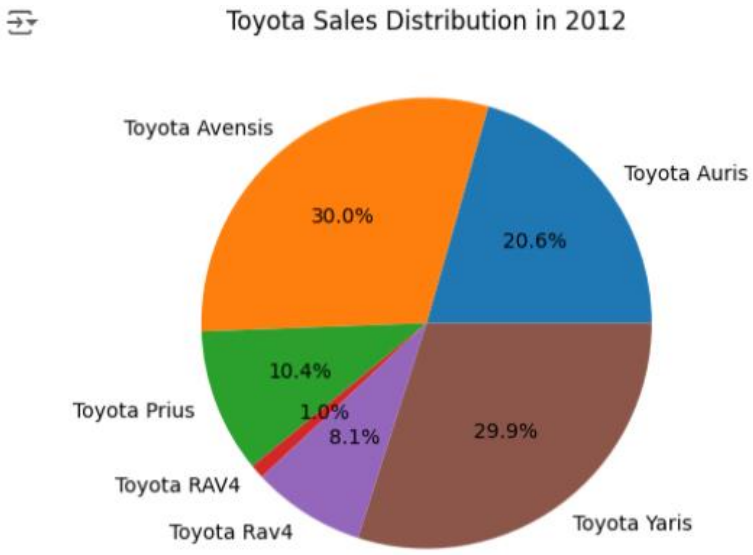
```python
# c: Draw pie chart for the sales of all models of Toyota in the year 2012
data_models = pd.read_csv('/content/norway_new_car_sales_by_model.csv')
data_models.columns = data_models.columns.str.strip()

toyota_sales_2012 = data_models[(data_models['Make'].str.contains('Toyota', case=False, na=False)) & (data_models['Year'] == 2012)]
model_sales = toyota_sales_2012.groupby('Model')['Quantity'].sum()

if model_sales.empty:
    print("No sales data found for Toyota in 2012.")
else:
    model_sales.plot(kind='pie', autopct='%1.1f%%', title='Toyota Sales Distribution in 2012', figsize=(5, 5))
    plt.ylabel('')
    plt.show()
    print("Model and Quantity Sales in 2012:")
    print(model_sales)

# d: For all models, sum the sales and calculate percentage share
model_sales_all = data_models.groupby('Model')['Quantity'].sum()
top_models = model_sales_all.sort_values(ascending=False).head(10)

total_sales = model_sales_all.sum()
top_models_percentage = (top_models / total_sales) * 100

top_models_percentage.plot(kind='bar', color='skyblue', title='Top Car Models Sales Share (Percentage)', figsize=(5, 5))
plt.xlabel('Car Model')
plt.ylabel('Percentage Share (%)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
print(top_models_percentage)
```



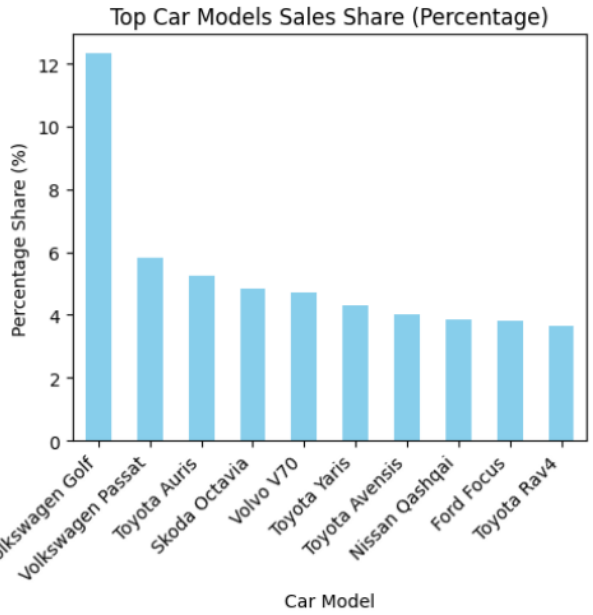

```
Year
2007    129195
2008    110617
2009     98675
2010    127754
2011    138345
2012    137967
2013    142151
2014    144202
2015    150686
2016    154603
2017     13055
Name: Quantity, dtype: int64
```




```
Month
1      10838
2      10925
3      13051
4      10876
5      12612
6      11053
7      11920
8      11790
9      11134
10     12413
11     11986
12      9369
Name: Quantity, dtype: int64
```




```
Model and Quantity Sales in 2012:
Model
Toyota Auris      2555
Toyota Avensis    3726
Toyota Prius      1286
Toyota RAV4        125
Toyota Rav4       1010
Toyota Yaris      3715
Name: Quantity, dtype: int64
```




```
Model
Volkswagen Golf      12.317878
Volkswagen Passat     5.826033
Toyota Auris          5.265040
Skoda Octavia         4.822649
Volvo V70             4.708210
Toyota Yaris          4.289942
Toyota Avensis        4.011814
Nissan Qashqai        3.838074
Ford Focus            3.827449
Toyota Rav4           3.626571
Name: Quantity, dtype: float64
```

2. Describe a real-world use case where HDFS plays a critical role.

1) HDFS in E-Commerce Big Data Analytics

1. Handling Massive Data Volumes

E-commerce platforms generate enormous volumes of data from user interactions, purchases, and product feedback.

- **Example**: Amazon, with millions of products and users, collects vast amounts of data every time a user searches for a product or makes a purchase. HDFS efficiently stores and scales with this massive data growth.

2. Distributed Storage and Fault Tolerance

HDFS splits large files into smaller blocks, storing them across various machines. These blocks are replicated to ensure fault tolerance.

- **Example**: If one server fails, the data is still accessible from other servers, preventing any downtime in the e-commerce platform's operations.

3. Cost-Effective Data Storage

Traditional data storage methods can be expensive when handling large amounts of data. HDFS, by leveraging commodity hardware, provides a more affordable solution.

- **Example**: Instead of using costly relational databases, e-commerce platforms can store petabytes of data in a Hadoop cluster, significantly cutting down storage costs.

4. Scalable Data Processing

HDFS integrates with Hadoop ecosystem tools such as Apache Spark and MapReduce, enabling distributed and parallel data processing.

- **Example**: A recommendation system can process billions of data points quickly, suggesting products based on users' past behaviors, all thanks to the scalability of HDFS.

5. Data Integration from Multiple Sources

E-commerce platforms gather data from various sources like websites, apps, social media, and emails. HDFS provides a unified platform for integrating and analyzing this data.

- **Example**: Analyzing customer reviews, transaction histories, and social media sentiment together helps businesses understand product performance and customer satisfaction.

6. Support for Batch and Real-Time Processing

HDFS supports both batch processing for historical data and real-time analytics for immediate decision-making.

- **Example**: E-commerce sites can process large amounts of data for monthly sales reports (batch processing) or adjust product prices dynamically based on live customer demand and competitor pricing (real-time processing).

2) HDFS in Healthcare Data Management

1. Storing Medical Data

Healthcare facilities generate large amounts of data, such as medical records, images, and sensor data from devices like heart rate monitors. HDFS can store these files efficiently.

- **Example**: Patient records, X-rays, and diagnostic images are stored in HDFS, ensuring that the healthcare data is easily accessible and fault-tolerant.

2. Fault-Tolerant Storage

HDFS ensures that patient records and medical data are safe, even in the event of server failure. Data is replicated across nodes, providing continuous access.

- **Example**: If a server fails, the data remains accessible from other servers, ensuring that patient information is always available for doctors and medical staff.

3. Scalable Data Handling

Healthcare data, including medical histories, patient records, and real-time monitoring data, grows quickly. HDFS offers a scalable solution for managing this increasing data load.

- **Example**: With growing numbers of patients and expanding data from medical devices, HDFS scales to accommodate larger datasets, allowing for better healthcare management.

4. Real-Time Analytics for Patient Monitoring

HDFS integrates with tools like Apache Spark and MapReduce to process sensor data for real-time health monitoring and predictive analytics.

- **Example**: Analyzing heart rate data in real time can help detect early signs of cardiac distress, allowing for immediate medical intervention.

5. Data Integration Across Healthcare Systems

HDFS provides a unified platform to integrate data from various healthcare sources like electronic health records (EHR), wearable devices, and lab results.

- **Example**: By combining real-time sensor data with historical health records, doctors can make more informed decisions about patient care and predict future health conditions.

6. Support for Batch and Real-Time Processing

HDFS allows for both batch and real-time data processing, crucial for handling the variety of healthcare data.

- **Example**: Batch processing could be used to analyze monthly hospital performance, while real-time processing allows for immediate analysis of patient vitals during a surgical procedure.

3. Analyze the impact of changing the replication factor on performance and fault tolerance.

1) Fault Tolerance

The replication factor directly influences the system's ability to tolerate failures. A higher replication factor means more copies of data are stored, providing increased redundancy.

- **Increased Replication Factor**:
    - **Enhanced Fault Tolerance**: With more replicas, the system can tolerate multiple node failures without losing data. Even if some nodes are unavailable due to failure or network issues, the system can still serve requests from the remaining replicas, ensuring high availability.
    - **Better Data Durability**: As more replicas exist, the likelihood of data being lost during simultaneous node failures decreases. The system can recover even from significant disruptions.
- **Decreased Replication Factor**:
    - **Weakened Fault Tolerance**: Fewer replicas reduce the number of copies available to withstand node failures. A system with a single replica, for example, risks total data loss if that replica becomes unavailable.
    - **Increased Risk of Downtime**: A low replication factor makes the system more vulnerable to downtime and data loss, especially during hardware failures or network partitions.

2) Performance

The replication factor also affects the system's performance, particularly in terms of read and write operations.

- **Increased Replication Factor**:
    - **Improved Read Performance**: With more replicas, read operations can be distributed among different copies of the data. This helps reduce latency by allowing clients to fetch data from the nearest or least-loaded replica, improving overall read throughput.

    - **Slower Write Performance**: Write operations become slower with a higher replication factor because every write must be propagated to all replicas. The system needs to ensure consistency across replicas, which can introduce delays and additional coordination overhead.
    - **Increased Network Traffic**: More replicas mean more data needs to be replicated across the network, leading to higher network bandwidth usage and potential congestion. This can slow down the system, particularly during periods of heavy data replication.
- **Decreased Replication Factor**:
    - **Faster Write Performance**: With fewer replicas, the system can update data more quickly because it only needs to modify a smaller number of copies. This reduces latency and can improve the performance of write-heavy applications.
    - **Weakened Read Performance**: Fewer replicas may reduce the ability to distribute read requests, leading to higher load on the remaining replicas. This can increase response times for read-heavy workloads.

3) Trade-Offs and Considerations

Adjusting the replication factor involves balancing several factors, as there are both benefits and drawbacks to increasing or decreasing the number of replicas.

- **Scalability**: A higher replication factor typically improves scalability for read-heavy applications. However, as the number of replicas increases, the write performance may degrade due to the need to update more copies, which can create a bottleneck in the system.
- **Cost**: Increasing the replication factor raises storage and operational costs. More replicas mean more storage space is required, leading to higher infrastructure costs. Additionally, the need to manage and synchronize multiple replicas across machines increases the operational complexity.
- **Consistency vs. Availability (CAP Theorem)**: The replication factor also impacts the consistency and availability trade-offs, as outlined by the CAP theorem. A higher replication factor often increases data availability and fault tolerance, but it may compromise consistency in certain cases. For instance, achieving strong consistency might require a quorum of replicas to confirm writes, which could delay the process and affect system responsiveness.

4. Discuss the challenges you might face in implementing join operations inMongoDB for a social media platform and how you might resolve them.

1) Lack of Native Join Support

MongoDB doesn't support multi-table joins like relational databases. To merge data from different collections, you can use the **$lookup** operator in the aggregation framework to simulate joins.

Example:

```
db.posts.aggregate([
  {
    $lookup: {
      from: "users",
      localField: "userId",
      foreignField: "_id",
      as: "userDetails"
    }
  }
])
}
```

2) Performance Issues with Large Datasets

Join-like operations in MongoDB can be costly when handling large amounts of data. One way to mitigate this is by using **denormalization**, where you store redundant data (e.g., user information inside posts). This reduces the need for joins but increases storage. Additionally, using proper **indexes** on frequently queried fields can speed up the performance of operations like `$lookup`.

```
{
  "_id": "post_id",
  "author": {
    "userId": "user_id",
    "name": "John Doe",
    "profile_picture": "url_to_picture"
  },
```

```
  "content": "This is a post",
  "likes": ["user_id1", "user_id2"],
  "comments": [
    {
      "userId": "user_id3",
      "text": "Nice post!"
    }
  ]
}
```

3) Data Consistency Challenges

Maintaining data consistency across collections is difficult, especially if user data is embedded in posts or comments. This can lead to complications when updating user profiles. To address this, you can use **eventual consistency** with background processes that update data asynchronously, or **atomic operations** to ensure consistency within individual documents.

```
db.users.updateOne(
  { _id: "user_id" },
  { $set: { "name": "Updated Name" } }
)
```

4) Handling Complex Relationships

Social media platforms often have complex relationships like users following other users or liking posts. For such relationships, integrating a **graph database** like Neo4j can be beneficial. MongoDB can store the data, and the graph database can handle complex queries related to relationships. For simpler relationships, references to user IDs can be stored directly in posts or comments.

```
{
  "_id": "user_id",
  "name": "John Doe",
  "followers": ["user_id1", "user_id2"],
  "following": ["user_id3", "user_id4"]
}
```

5) Maintaining Schema Flexibility

MongoDB's flexible schema can cause issues when trying to join documents with different structures. The best approach is to carefully design your schema so that related information is stored together. If schema changes are required, versioning and schema migrations should be implemented to handle evolving data structures.

```
{
  "_id": "post_id",
  "author_id": "user_id",
  "content": "This is a post",
  "likes": ["user_id1", "user_id2"],
  "comments": [
    {
      "user_id": "user_id3",
      "text": "Nice post!"
    }
  ]
}
```

5. How would you load CSV or JSON formatted data into a Hive table from HDFS? Provide step-by-step instructions.

Prerequisites:

1. **Hadoop and Hive Setup**: Ensure that both Hadoop and Hive are properly installed and running.
2. **Upload Files to HDFS**: Use the `hdfs dfs -put` command to upload your CSV or JSON files to HDFS.

1) Loading CSV Data into Hive

Step 1: Create a Hive Table for CSV Data

```
CREATE TABLE IF NOT EXISTS employee_sales (
    emp_id INT,
    first_name STRING,
    last_name STRING,
    department STRING,
    year INT,
    month STRING,
```

```
    total_sales DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

- This creates a Hive table `employee_sales` to store CSV data.

Step 2: Upload CSV File to HDFS

```
hdfs dfs -put /local/path/to/employee_sales.csv
/hdfs/path/to/employee_sales.csv
```

Step 3: Load Data into Hive Table

Now, load the CSV data into the Hive table:

```
LOAD DATA INPATH '/hdfs/path/to/employee_sales.csv' INTO TABLE
employee_sales;
```

Step 4: Verify Data in the Table

Check if the data is loaded correctly by running the following query:

```
SELECT * FROM employee_sales;
```

2) Loading JSON Data into Hive

Step 1: Create a Hive Table for JSON Data

For JSON data, you need to use the **JsonSerDe** (Serializer/Deserializer) to read the JSON format. Here's how to create the table:

```
CREATE TABLE IF NOT EXISTS employee_sales_json (
    emp_id INT,
    first_name STRING,
    last_name STRING,
    department STRING,
    year INT,
    month STRING,
    total_sales DOUBLE
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE;
```

- This creates a Hive table `employee_sales_json` to store JSON data.

Step 2: Upload JSON File to HDFS

Upload your JSON file to HDFS using the following command:

```
hdfs dfs -put /local/path/to/employee_sales.json
/hdfs/path/to/employee_sales.json
```

Step 3: Load Data into Hive Table

Load the JSON data into the Hive table:

```
LOAD DATA INPATH '/hdfs/path/to/employee_sales.json' INTO
TABLE employee_sales_json;
```

Step 4: Verify Data in the Table

Check if the JSON data has been loaded correctly by running this query:

```
SELECT * FROM employee_sales_json;
```

6. Write a Pig Latin script that loads data from a CSV file, transforms it into a tuple of values, and then filters the dataset based on certain conditions.

Step 1: Prepare Your Data (CSV File)

Create a CSV file named `product_sales.csv` with the following content:

```
year,month,product,category,quantity,price

2021,1,Laptop,Electronics,150,800.5
2021,1,Smartphone,Electronics,300,600.2
2021,2,Laptop,Electronics,220,799.9
2021,2,Smartphone,Electronics,180,599.9
2021,2,Refrigerator,Appliances,100,1200.0
2021,2,Microwave,Appliances,250,150.0
2022,1,Laptop,Electronics,200,780.0
2022,1,Smartphone,Electronics,320,590.0
```

Save this file in a directory accessible to your Pig script.

Step 2: Write the Pig Latin Script

Save the following Pig Latin script as `product_sales_filter.pig`:

```
-- Load the CSV file into Pig with a schema
sales_data = LOAD 'product_sales.csv' USING PigStorage(',')
    AS (year:int, month:int, product:chararray, category:chararray, quantity:int,
price:float);

-- Filter records for February 2021 with quantity greater than 200
filtered_sales = FILTER sales_data BY (year == 2021) AND (month == 2) AND (quantity
> 200);

-- Group data by product category and calculate total quantity and average price
category_totals = GROUP filtered_sales BY category;
category_summary = FOREACH category_totals GENERATE
    group AS category, SUM(filtered_sales.quantity) AS total_quantity,
AVG(filtered_sales.price) AS avg_price;

-- Order the results by total quantity in descending order
ordered_summary = ORDER category_summary BY total_quantity DESC;

-- Output the results to the console
DUMP ordered_summary;
```

Step 3: Explanation of the Script

1. **Load Data:**
    - The `LOAD` command imports the CSV file into Pig, with a schema defining the columns: `year`, `month`, `product`, `category`, `quantity`, and `price`.

```
sales_data = LOAD 'product_sales.csv' USING PigStorage(',')
    AS (year:int, month:int, product:chararray, category:chararray,
quantity:int, price:float);
```

2. **Filter Data:**
    - The `FILTER` command selects records where the year is 2021, the month is February (2), and the quantity is greater than 200.

```
filtered_sales = FILTER sales_data BY (year == 2021) AND (month == 2) AND
(quantity > 200);
```

3. **Group and Aggregate Data:**
    - The `GROUP` command groups the filtered data by `category`, and the `FOREACH` command calculates the total quantity and average price for each group.

```
category_totals = GROUP filtered_sales BY category;
```

```
category_summary = FOREACH category_totals GENERATE
    group AS category, SUM(filtered_sales.quantity) AS total_quantity,
AVG(filtered_sales.price) AS avg_price;
```

4. **Order the Results:**
    - The `ORDER` command sorts the results by total quantity in descending order.

```
ordered_summary = ORDER category_summary BY total_quantity DESC;
```

5. **Display the Output:**
    - The `DUMP` command outputs the final processed data to the console.

```
DUMP ordered_summary;
```

Step 4: Run the Pig Script

1. **Running Locally:**
    - Place both the CSV file and the script in the same directory.
    - Execute the Pig script using the command:

```
pig -x local product_sales_filter.pig
```

2. **Running on Hadoop (HDFS Mode):**
    - Upload the CSV file to HDFS:

```
hdfs dfs -put product_sales.csv /user/yourusername/
```

    - Modify the `LOAD` command in the script to reference the HDFS path:

```
sales_data = LOAD '/user/yourusername/product_sales.csv' USING
PigStorage(',')
```

    - Then execute the script on Hadoop:

```
pig product_sales_filter.pig
```

Step 5: Output

After running the script, the output will be displayed as:

```
(Appliances,250,150.0)
```

This result indicates that:

- The category is **Appliances**.
- The **total quantity** of items sold in this category is **250**.
- The **average price** of items in this category is **150.0**.

Step 6: Save Results to Output Directory

To save the processed results to an output directory for further analysis, you can modify the script to include the `STORE` command:

```
STORE ordered_summary INTO 'output_directory' USING PigStorage(',');
```

This will store the results in the specified directory.

7. Write a Python script using BeautifulSoup or Selenium to scrape data from a website. What techniques will you use to ensure the script is efficient and handles dynamic content?

In this solution, we'll create a Python script that efficiently scrapes data from a website using the BeautifulSoup library. To ensure the script handles dynamic content and is efficient, we will implement several strategies such as error handling, session management, dynamic content handling, and respecting the website's usage policies with polite delays between requests.

```
import requests
from bs4 import BeautifulSoup
import csv
import logging
import time
import random
class WebScraper:
def __init__(self, base_url='https://quotes.toscrape.com'):
self.base_url = base_url
self.session = requests.Session()
self.session.headers.update({
'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
})
logging.basicConfig(
format='%(asctime)s - %(levelname)s: %(message)s',
level=logging.INFO,
datefmt='%Y-%m-%d %H:%M:%S'
```

```
)
def get_page(self, url):
try:
response = self.session.get(url, timeout=10)
response.raise_for_status()  # Check for HTTP errors
return response.text
except requests.RequestException as e:
logging.error(f"Error fetching {url}: {e}")
return None
def parse_quotes(self, html):
"""Parse quotes from the HTML content."""
soup = BeautifulSoup(html, 'lxml')
quotes = []
for quote_block in soup.select('.quote'):
quote = {
'text': quote_block.select_one('.text').text.strip(),
'author': quote_block.select_one('.author').text.strip(),
'tags': [tag.text for tag in quote_block.select('.tags .tag')]}
quotes.append(quote)
return quotes
def scrape_multiple_pages(self, max_pages=5):
"""Scrape quotes from multiple pages."""
all_quotes = []
for page in range(1, max_pages + 1):
url = f'{self.base_url}/page/{page}/'
logging.info(f"Scraping page {page}: {url}")
html = self.get_page(url)
if html:
quotes = self.parse_quotes(html)
all_quotes.extend(quotes)
logging.info(f"Scraped {len(quotes)} quotes from {url}")
time.sleep(random.uniform(1, 3))
return all_quotes
def save_to_csv(self, quotes, filename='quotes.csv'):
"""Save the scraped quotes to a CSV file."""
with open(filename, 'w', newline='', encoding='utf-8') as csvfile:
fieldnames = ['text', 'author', 'tags']
writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
writer.writeheader()
for quote in quotes:
quote['tags'] = ', '.join(quote['tags'])
writer.writerow(quote)
logging.info(f"Data saved to {filename}")
def main():
scraper = WebScraper()
quotes = scraper.scrape_multiple_pages()
```

```
scraper.save_to_csv(quotes)
logging.info(f"Total quotes scraped: {len(quotes)}")
if __name__ == "__main__":
main()
```

Explanation and Techniques Used:

1. **Object-Oriented Design:**
   - The script utilizes a class (`WebScraper`) to encapsulate functionality. This promotes better structure, modularity, and reuse of methods.
   - Methods inside the class handle specific tasks like fetching pages, parsing content, scraping multiple pages, and saving to CSV.
2. **Session Management:**
   - A `requests.Session()` object is used to maintain persistent connections. This reduces the overhead of opening a new connection for every request, which can improve performance.
   - The session is also configured with a custom `User-Agent` header to avoid being flagged as a bot by websites.
3. **Error Handling:**
   - The `get_page()` method handles errors gracefully using a try-except block, ensuring that the script does not crash due to network or HTTP errors.
   - `raise_for_status()` is used to check for HTTP error statuses (like 404 or 500) and log them appropriately.
4. **Polite Scraping:**
   - A random delay (`time.sleep(random.uniform(1, 3))`) is added between requests to prevent overwhelming the server, making the script more respectful of the website's usage policies and reducing the likelihood of being blocked.
5. **Handling Dynamic Content:**
   - While this script uses BeautifulSoup to parse static content, if the website uses JavaScript to load data dynamically, you would use **Selenium** instead. Selenium can interact with dynamic elements on the page by simulating user actions like scrolling or clicking, making it a better choice for such websites.
6. **CSV Output:**

- The scraped data (quotes) is saved in a CSV file using Python's `csv` module. Tags are joined into a comma-separated string to store them in a single CSV column, making it easy to handle and analyze the data later.

Running the Script:

1. **Installing Dependencies:** Before running the script, ensure the required libraries are installed:

   ```
   pip install requests beautifulsoup4 lxml
   ```

2. **Execution:** Simply run the script from the command line:

   ```
   python scraper.py
   ```

   This will scrape quotes from the first 5 pages of the site and save the results in `quotes.csv`.

Enhancements for Dynamic Content Handling:

If you need to handle dynamic content (e.g., content loaded by JavaScript), you can replace `requests` and `BeautifulSoup` with **Selenium**, which will allow the script to interact with the web page as a real browser would.

Example with Selenium:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import time
driver = webdriver.Chrome()
driver.get("https://quotes.toscrape.com")
quotes = []
time.sleep(2)
quote_elements = driver.find_elements(By.CSS_SELECTOR,
".quote")
```

```
for quote_element in quote_elements:
text = quote_element.find_element(By.CSS_SELECTOR,
".text").text.strip()
author = quote_element.find_element(By.CSS_SELECTOR,
".author").text.strip()
tags = [tag.text for tag in
quote_element.find_elements(By.CSS_SELECTOR, ".tags .tag")]
quotes.append({'text': text, 'author': author, 'tags': tags})
driver.quit()
```

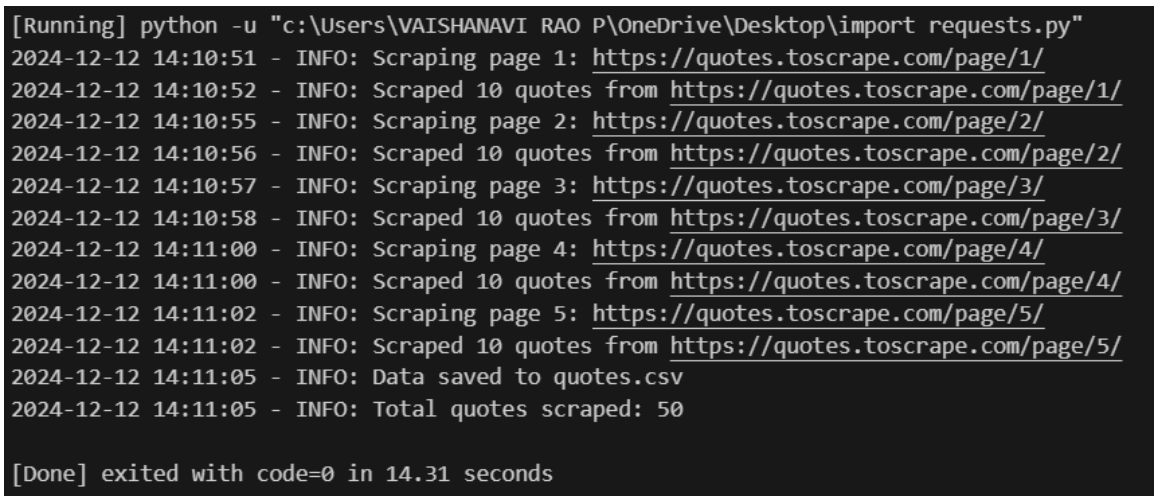
In this version, **Selenium** is used to open the page in a real browser environment, where JavaScript is fully rendered. Once the page is loaded, it extracts the quotes and their associated details.

Conclusion:

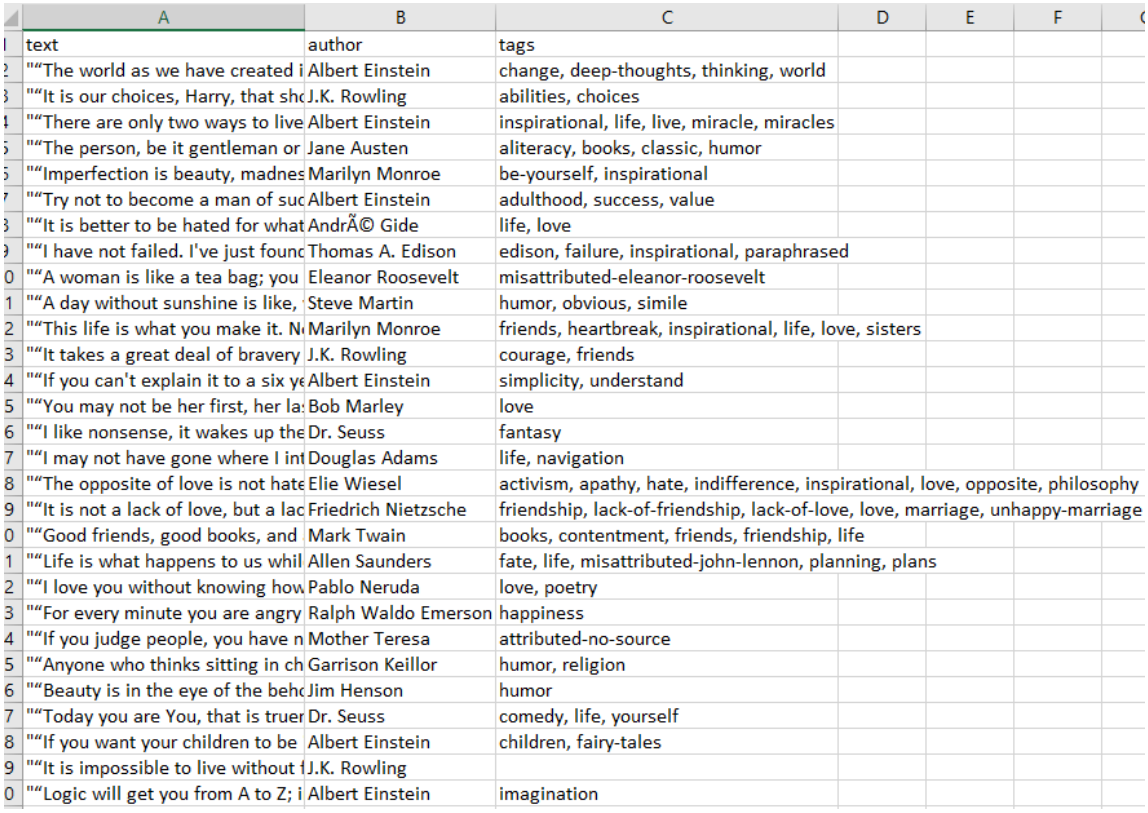
This solution ensures efficiency by:

- Using session management for persistent connections.
- Implementing error handling for robustness.
- Adding polite delays to respect the server.
- Using object-oriented design for better organization and extensibility.

1. Logging Output



2. CSV File Output



	A	B	C
1	text	author	tags
2	""The world as we have created i	Albert Einstein	change, deep-thoughts, thinking, world
3	""It is our choices, Harry, that sh	J.K. Rowling	abilities, choices
4	""There are only two ways to live	Albert Einstein	inspirational, life, live, miracle, miracles
5	""The person, be it gentleman or	Jane Austen	aliteracy, books, classic, humor
6	""Imperfection is beauty, madnes	Marilyn Monroe	be-yourself, inspirational
7	""Try not to become a man of suc	Albert Einstein	adulthood, success, value
8	""It is better to be hated for what	AndrÃ© Gide	life, love
9	""I have not failed. I've just found	Thomas A. Edison	edison, failure, inspirational, paraphrased
10	""A woman is like a tea bag; you	Eleanor Roosevelt	misattributed-eleanor-roosevelt
11	""A day without sunshine is like,	Steve Martin	humor, obvious, simile
12	""This life is what you make it. N	Marilyn Monroe	friends, heartbreak, inspirational, life, love, sisters
13	""It takes a great deal of bravery	J.K. Rowling	courage, friends
14	""If you can't explain it to a six ye	Albert Einstein	simplicity, understand
15	""You may not be her first, her la	Bob Marley	love
16	""I like nonsense, it wakes up the	Dr. Seuss	fantasy
17	""I may not have gone where I int	Douglas Adams	life, navigation
18	""The opposite of love is not hate	Elie Wiesel	activism, apathy, hate, indifference, inspirational, love, opposite, philosophy
19	""It is not a lack of love, but a lac	Friedrich Nietzsche	friendship, lack-of-friendship, lack-of-love, love, marriage, unhappy-marriage
20	""Good friends, good books, and	Mark Twain	books, contentment, friends, friendship, life
21	""Life is what happens to us whil	Allen Saunders	fate, life, misattributed-john-lennon, planning, plans
22	""I love you without knowing how	Pablo Neruda	love, poetry
23	""For every minute you are angry	Ralph Waldo Emerson	happiness
24	""If you judge people, you have n	Mother Teresa	attributed-no-source
25	""Anyone who thinks sitting in ch	Garrison Keillor	humor, religion
26	""Beauty is in the eye of the beh	Jim Henson	humor
27	""Today you are You, that is truer	Dr. Seuss	comedy, life, yourself
28	""If you want your children to be	Albert Einstein	children, fairy-tales
29	""It is impossible to live without t	J.K. Rowling	
30	""Logic will get you from A to Z; i	Albert Einstein	imagination

3. CSV File Structure

- **Columns:**
  - `text`: The actual quote.
  - `author`: The person who said the quote.
  - `tags`: A list of tags associated with the quote (e.g., themes or topics relevant to the quote).

The quotes are stored as a string in the `tags` column, where each tag list is represented as a string of the form `['tag1', 'tag2', ...]`.