UNIVERSITY OF ECONOMICS AND LAW

FACULTY OF INFORMATION SYSTEMS

FINAL PROJECT

SUBJECT: BIG DATA AND APPLICATIONS

COURSE CODE: 242MI1401

TOPIC: APPLYING BIG DATA AND MACHINE LEARNING TO ANALYZE AND PREDICT SUPERSTORE SALES

LECTURER:

NGUYEN THON DA, PHD

STUDENT NAME	PHAN THANH GIANG
STUDENT CODE	K214162145

Ho Chi Minh City, December 27th, 2024

ACKNOWLEDGMENTS

I would like Dr. Nguyen Tong Da to express my deepest gratitude, my instructor for the course "Big Data and Applications," for your invaluable guidance and support throughout the completion of this report. Your expertise and insights have been crucial in helping me navigate the complexities of the topic, "Applying Big Data and Machine Learning to Analyze and Predict Superstore Sales."

I'm also truly grateful for the opportunity to learn under your mentorship. The knowledge and skills gained during this course will undoubtedly serve as a solid foundation for my future endeavors. Your dedication to teaching and unwavering support have been instrumental in the success of this project. Thank you for being a shining example in my academic journey.

Sincerely,

Ho Chi Minh City, January 1, 2025

Phan Thanh Giang

COMMITMENT

I declare that all results and research content in this report were conducted by myself. I confirm that I will not copy or illegally use the research results of any other individual or research group.

Sincerely,

Ho Chi Minh City, January 1, 2025

Phan Thanh Giang

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
RDD	Resilient Distributed Dataset
RMSE	Root Mean Square Error
R^2	Coefficient of Determination
ML	Machine Learning
Big Data	Large-scale datasets requiring advanced analysis
Streamlit	Open-source app framework for ML and Data Science
PySpark	Python API for Apache Spark
MAE	Mean Absolute Error

1. Introduction

1.1 Reasons for choosing topic

In the age of Big Data, businesses are constantly seeking ways to leverage data to gain competitive advantages. The ability to analyze and predict sales trends not only aids in decision-making but also enhances operational efficiency. The retail sector, particularly large-scale businesses like Superstore, generates vast amounts of data daily. Harnessing this data through advanced technologies such as machine learning and Big Data analytics has become a critical requirement. This project was chosen to demonstrate how these technologies can be applied to real-world problems, specifically in analyzing and predicting sales, thus offering valuable insights to businesses.

1.2. Urgency of the topic

The rapid advancement of technology and the competitive nature of the retail market have made it essential for businesses to utilize predictive analytics for sustainable growth.

By predicting future sales trends, companies can optimize inventory, plan marketing campaigns, and improve customer satisfaction. Failure to adopt such tools risks falling behind competitors who are quick to embrace data-driven strategies. Given the exponential growth of data, applying Big Data techniques combined with machine learning is not just an opportunity but a necessity for modern enterprises.

1.3 Objective of the topic

The primary objective of this project is to build a system that applies Big Data and machine learning techniques to analyze and predict Superstore sales. Specifically includes:

- Preprocess data using RDDs method and clean raw data to ensure prediction accuracy.
- Extract meaningful features from the data, including sales trends and revenue patterns.
- Train a machine learning model to predict sales based on key variables such as sales revenue, profit, and total revenue.
- Deploy a user-friendly application using Streamlit, allowing users to input variables and receive predictions in real-time.

1.4 Scope of project

This project focuses on dataset superstore, with an emphasis on utilizing Big Data tools like PySpark for data preprocessing and machine learning libraries for model training. The scope includes:

- Preprocessing data using techniques such as outlier detection, feature engineering, and encoding.
- Training and evaluating a regression model to predict future sales trends.
- Deploying the model as an interactive web application.
  While the dataset is limited to one Superstore, the methods and tools applied can be generalized to similar retail businesses, offering broad applicability.

1.5. Except Result

I hope that at the end of the project, the data will be pre-processed using a comprehensive RDDs method, so that key factors influencing sales trends can be identified. Additionally, regression models will be developed to accurately predict sales based on historical data and user inputs, providing valuable insights for decision-making .

Besides, a simple interactive online application demo using streamlit will be deployed to create an interface to simplify the Profit prediction process. Helps users without technical expertise understand the results of predictive analysis.

2. Related work

2.1. Overview of Big Data in Retail

Big Data has revolutionized the retail sector, offering transformative solutions to enhance operational efficiency, customer engagement, and decision-making processes. Retail generates vast amounts of data daily, encompassing structured data such as transaction records and unstructured data from social media and IoT devices. The volume, velocity, and variety of this data make traditional analysis insufficient, necessitating advanced Big Data analytics to extract actionable insights. The integration of Big Data analytics enables retailers to navigate through the "data tsunami" by transforming raw data into actionable strategies. (Lekhwar, Yadav, and Singh, 2018)

According to Aktas and Meng (2017), Big Data applications in retail focus on four key areas: availability, assortment, pricing, and layout planning. Historical sales data and loyalty schemes provide valuable insights into customer behavior and demand patterns, while external factors like weather and competitor pricing enrich forecasting models. However, challenges such as skill shortages, IT integration issues, and managerial barriers persist, requiring strategic initiatives to overcome

Similarly, Seetharaman et al. (2016) emphasize the impact of Big Data on improving customer engagement and market value. They highlight the critical role of data-driven decision-making (DDD) in boosting productivity and financial outcomes. Big Data tools such as Hadoop and machine learning algorithms enable retail organizations to identify trends, optimize inventory, and design personalized marketing campaigns, thus creating a competitive edge in the market

2.2. Machine Learning for Sales Prediction

The application of machine learning (ML) techniques in sales prediction has become a cornerstone of modern business analytics, offering businesses the ability to anticipate future sales with high accuracy. Haselbeck et al. (2022) highlight that advanced machine learning models like Extreme Gradient Boosting (XGBoost) and Long Short-Term Memory networks (LSTMs) outperform traditional methods, such as ARIMA, particularly in datasets with seasonality or multi-dimensional variables. These methods integrate historical sales data with external factors, such as promotional activities, holidays, or customer behavior, providing a more nuanced prediction capability.

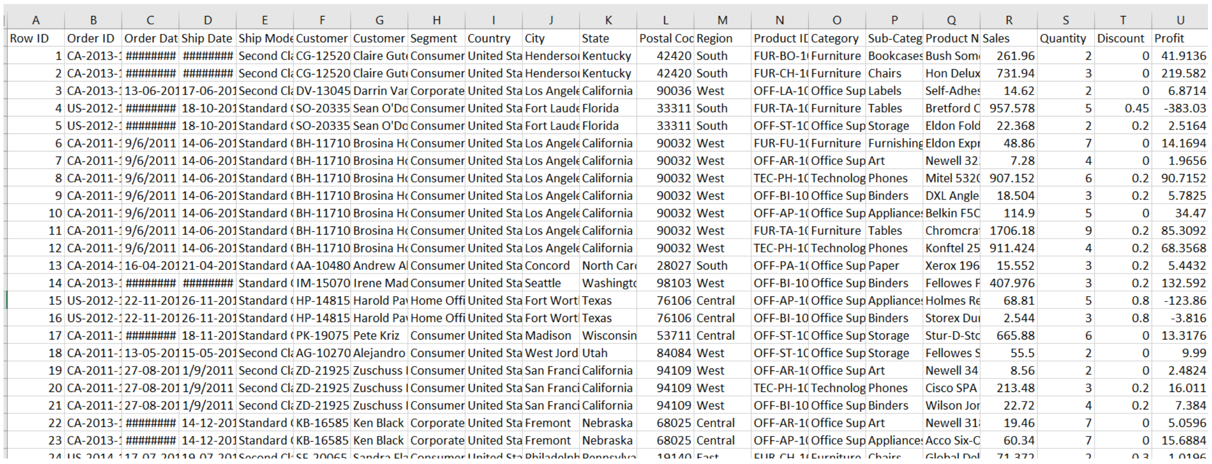
Furthermore, Sana et al. (2021) suggest the impact of machine learning in dynamic industries, such as the food and beverage sector, where algorithms can predict sales based on features like weather, customer preferences, and marketing efforts. The use of frameworks like TensorFlow has enabled researchers to achieve real-time predictive accuracy by processing and analyzing complex datasets effectively.

2.3. Applications of RDDs in Big Data

Resilient Distributed Datasets (RDDs) is a fundamental abstraction in Apache Spark, designed to efficiently process large data sets across distributed environments, while ensuring fault tolerance and scalability. According to Singh et al. (2019), RDDs provide a secure in-memory storage mechanism that supports iterative algorithms and in-memory computations, addressing the limitations of traditional MapReduce, such as high I/O overhead. and excessive disk write/read times.

An important application of RDD is in association rule mining combined with the ECLAT algorithm. Sheshikala M et al. (2020) proposed RDD-ECLAT, a Spark-based mining method that helps process large data efficiently by dividing the data set into partitions and applying transformations to reduce computational costs. RDDs are also used in algorithms such as clustering and social network analysis. RDD ensures the ability to process data in both batch and real-time modes, making it easier for organizations to apply big data analytics to industries such as retail, healthcare and finance.

3. Data Description



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
Row ID	Order ID	Order Dat	Ship Date	Ship Mode	Customer	Customer	Segment	Country	City	State	Postal Coc	Region	Product ID	Category	Sub-Categ	Product N	Sales	Quantity	Discount	Profit
1	CA-2013-1	########	########	Second Cla	CG-12520	Claire Gute	Consumer	United Sta	Henderson	Kentucky	42420	South	FUR-BO-1	Furniture	Bookcases	Bush Som	261.96	2	0	41.9136
2	CA-2013-1	########	########	Second Cla	CG-12520	Claire Gute	Consumer	United Sta	Henderson	Kentucky	42420	South	FUR-CH-1	Furniture	Chairs	Hon Delux	731.94	3	0	219.582
3	CA-2013-1	13-06-201	17-06-201	Second Cla	DV-13045	Darrin Var	Corporate	United Sta	Los Angele	California	90036	West	OFF-LA-1	Office Sup	Labels	Self-Adhes	14.62	2	0	6.8714
4	US-2012-1	########	18-10-201	Standard (	SO-20335	Sean O'Do	Consumer	United Sta	Fort Laude	Florida	33311	South	FUR-TA-1	Furniture	Tables	Bretford C	957.578	5	0.45	-383.03
5	US-2012-1	########	18-10-201	Standard (	SO-20335	Sean O'Do	Consumer	United Sta	Fort Laude	Florida	33311	South	OFF-ST-1	Office Sup	Storage	Eldon Fold	22.368	2	0.2	2.5164
6	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	FUR-FU-1	Furniture	Furnishing	Eldon Expr	48.86	7	0	14.1694
7	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	OFF-AR-1	Office Sup	Art	Newell 32	7.28	4	0	1.9656
8	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	TEC-PH-1	Technolog	Phones	Mitel 532	907.152	6	0.2	90.7152
9	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	OFF-BI-10	Office Sup	Binders	DXL Angle	18.504	3	0.2	5.7825
10	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	OFF-AP-1	Office Sup	Appliances	Belkin F5C	114.9	5	0	34.47
11	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	FUR-TA-1	Furniture	Tables	Chromcra	1706.18	9	0.2	85.3092
12	CA-2011-1	9/6/2011	14-06-201	Standard (	BH-11710	Brosina Ho	Consumer	United Sta	Los Angele	California	90032	West	TEC-PH-1	Technolog	Phones	Konftel 25	911.424	4	0.2	68.3568
13	CA-2014-1	16-04-201	21-04-201	Standard (	AA-10480	Andrew Al	Consumer	United Sta	Concord	North Car	28027	South	OFF-PA-1	Office Sup	Paper	Xerox 196	15.552	3	0.2	5.4432
14	CA-2013-1	########	########	Standard (	IM-15070	Irene Mad	Consumer	United Sta	Seattle	Washingt	98103	West	OFF-BI-10	Office Sup	Binders	Fellowes P	407.976	3	0.2	132.592
15	US-2012-1	22-11-201	26-11-201	Standard (	HP-14815	Harold Pa	Home Offi	United Sta	Fort Worth	Texas	76106	Central	OFF-AP-1	Office Sup	Appliances	Holmes Re	68.81	5	0.8	-123.86
16	US-2012-1	22-11-201	26-11-201	Standard (	HP-14815	Harold Pa	Home Offi	United Sta	Fort Worth	Texas	76106	Central	OFF-BI-10	Office Sup	Binders	Storex Du	2.544	3	0.8	-3.816
17	CA-2011-1	########	18-11-201	Standard (	PK-19075	Pete Kriz	Consumer	United Sta	Madison	Wisconsin	53711	Central	OFF-ST-1	Office Sup	Storage	Stur-D-Stc	665.88	6	0	13.3176
18	CA-2011-1	13-05-201	15-05-201	Second Cla	AG-10270	Alejandro	Consumer	United Sta	West Jord	Utah	84084	West	OFF-ST-1	Office Sup	Storage	Fellowes S	55.5	2	0	9.99
19	CA-2011-1	27-08-201	1/9/2011	Second Cla	ZD-21925	Zuschuss I	Consumer	United Sta	San Franci	California	94109	West	OFF-AR-1	Office Sup	Art	Newell 34	8.56	2	0	2.4824
20	CA-2011-1	27-08-201	1/9/2011	Second Cla	ZD-21925	Zuschuss I	Consumer	United Sta	San Franci	California	94109	West	TEC-PH-1	Technolog	Phones	Cisco SPA	213.48	3	0.2	16.011
21	CA-2011-1	27-08-201	1/9/2011	Second Cla	ZD-21925	Zuschuss I	Consumer	United Sta	San Franci	California	94109	West	OFF-BI-10	Office Sup	Binders	Wilson Jor	22.72	4	0.2	7.384
22	CA-2013-1	########	14-12-201	Standard (	KB-16585	Ken Black	Corporate	United Sta	Fremont	Nebraska	68025	Central	OFF-AR-1	Office Sup	Art	Newell 31	19.46	7	0	5.0596
23	CA-2013-1	########	14-12-201	Standard (	KB-16585	Ken Black	Corporate	United Sta	Fremont	Nebraska	68025	Central	OFF-AP-1	Office Sup	Appliances	Acco Six-C	60.34	7	0	15.6884
24	US-2014-1	17-07-201	19-07-201	Second Cl	SE-20065	Sandra Fla	Consumer	United Sta	Philadelph	Pennsylva	19140	East	FUR-CH-1	Furniture	Chairs	Global Del	71.372	2	0.3	1.0196

Figure: Raw Data. Source: Author.

Dataset consists of 9994 rows and 21 columns.

Table 1: Dataset Description

Column Name	Description
Row ID	Unique identifier for each row in the dataset.
Order ID	Unique identifier for each order.
Order Date	The date when the order was placed.
Ship Date	The date when the order was shipped.
Ship Mode	The mode of shipment.
Customer ID	Unique identifier for each customer.
Customer Name	The name of the customer.
Segment	Customer segment.
Country	The country where the transaction occurred.
City	The city of the customer.
State	The state of the customer.
Postal Code	The postal code of the customer.
Region	The region of the customer.
Product ID	Unique identifier for each product.
Category	Category of the product.

Sub-Category	Subcategory of the product.
Product Name	The name of the product.
Sales	Total sales revenue generated by the product.
Quantity	Quantity of products sold in the transaction.
Discount	Discount applied to the transaction.
Profit	Profit earned from the transaction.

Link Dataset:
https://www.kaggle.com/code/arunjangir245/supermarket-sales-prediction-and-eda/input

4. Methodology

Initially, the project begins with data preprocessing, where the raw Superstore dataset is cleaned and prepared for analysis. Using the Resilient Distributed Dataset (RDD) method in PySpark, data is processed to handle missing values, remove duplicates, detect and filter outliers, and normalize text columns. Additionally, feature engineering is applied to create new variables, such as total revenue, by combining existing columns such as sales and quantity.

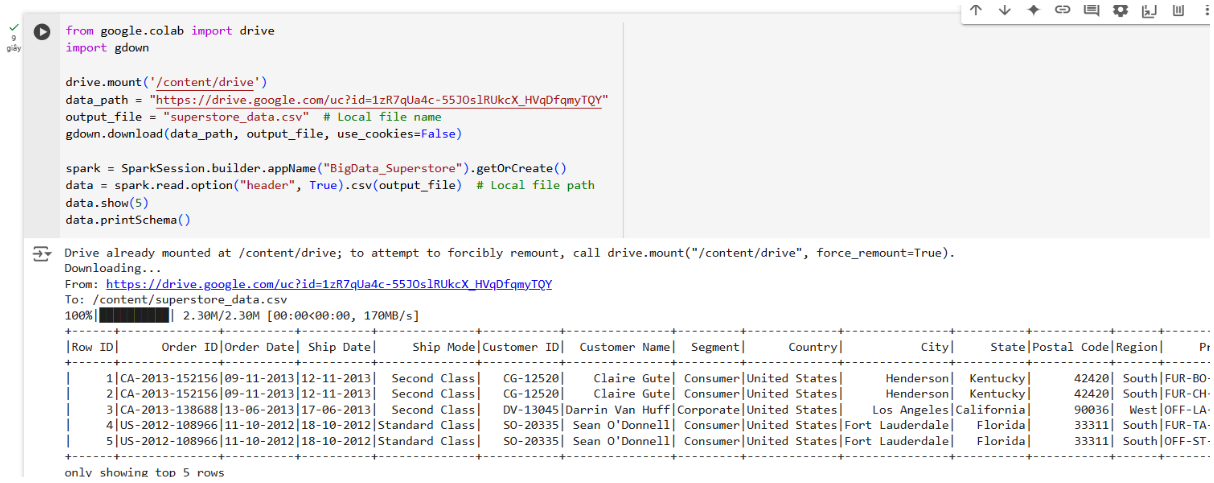
Next, the project focuses on **feature extraction and transformation**. Essential variables, including sales, profit, and total revenue, are identified and encoded using techniques such as one-hot encoding and binarization to ensure they are suitable for machine learning models. Date and time features, such as the year and month of order, are extracted to analyze seasonal trends and patterns. Finally, the selected features are assembled into a single vector for input into the regression model.

For **model training**, a linear regression model is chosen due to its simplicity and effectiveness in predicting continuous variables. The model is trained on the processed data to predict profit based on features like sales, profit, and total revenue. Key metrics, including Root Mean Square Error (RMSE) and R-squared (R^2), are used to evaluate the model's accuracy and reliability. The trained model is then saved on **Google Drive** and reloaded for deployment purposes, ensuring reproducibility and ease of integration.

Finally, I deployed the model as an interactive web application using Streamlit. The application provides a user-friendly interface where users can enter key variables and get real-time profit predictions. To make the application accessible, I will obtain tokens through the ngrok tool to host the application.

5. Data Preprocessing

5.1. Connect to Google Drive and Load Data



```
from google.colab import drive
import gdown

drive.mount('/content/drive')
data_path = "https://drive.google.com/uc?id=1zR7qUa4c-55JOslRUkcX_HVqDfqmyTQY"
output_file = "superstore_data.csv"  # Local file name
gdown.download(data_path, output_file, use_cookies=False)

spark = SparkSession.builder.appName("BigData_Superstore").getOrCreate()
data = spark.read.option("header", True).csv(output_file)  # Local file path
data.show(5)
data.printSchema()
```

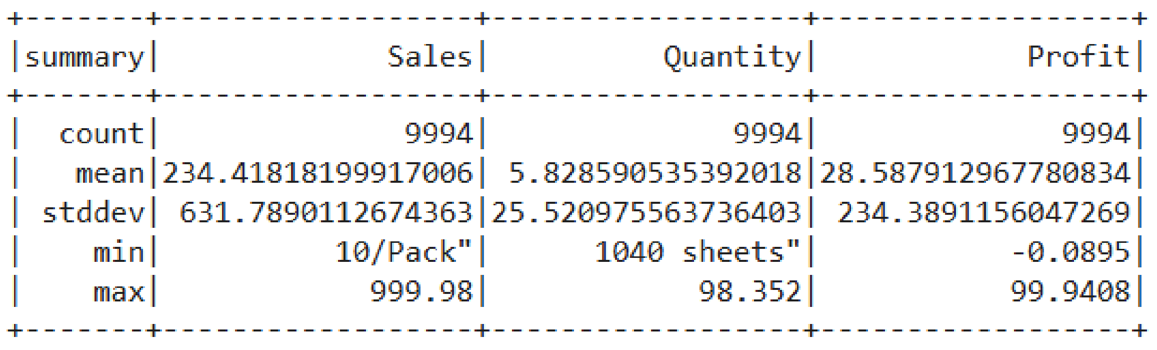
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Downloading...
From: https://drive.google.com/uc?id=1zR7qUa4c-55JOslRUkcX_HVqDfqmyTQY
To: /content/superstore_data.csv
100%|██████████| 2.30M/2.30M [00:00<00:00, 170MB/s]
+------+--------------+----------+----------+--------------+-----------+----------------+---------+-------------+---------------+----------+-----------+------+------
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|   Customer Name|  Segment|      Country|           City|     State|Postal Code|Region|    Pr
+------+--------------+----------+----------+--------------+-----------+----------------+---------+-------------+---------------+----------+-----------+------+------
|     1|CA-2013-152156|09-11-2013|12-11-2013|  Second Class|   CG-12520|     Claire Gute| Consumer|United States|      Henderson|  Kentucky|      42420| South|FUR-BO-
|     2|CA-2013-152156|09-11-2013|12-11-2013|  Second Class|   CG-12520|     Claire Gute| Consumer|United States|      Henderson|  Kentucky|      42420| South|FUR-CH-
|     3|CA-2013-138688|13-06-2013|17-06-2013|  Second Class|   DV-13045|Darrin Van Huff|Corporate|United States|    Los Angeles|California|      90036|  West|OFF-LA-
|     4|US-2012-108966|11-10-2012|18-10-2012|Standard Class|   SO-20335| Sean O'Donnell| Consumer|United States|Fort Lauderdale|   Florida|      33311| South|FUR-TA-
|     5|US-2012-108966|11-10-2012|18-10-2012|Standard Class|   SO-20335| Sean O'Donnell| Consumer|United States|Fort Lauderdale|   Florida|      33311| South|OFF-ST-
+------+--------------+----------+----------+--------------+-----------+----------------+---------+-------------+---------------+----------+-----------+------+------
only showing top 5 rows
```

First, I connect to Google Drive to load the necessary data for analysis. Specifically, the "Superstore Sales" dataset is stored in Google Drive as a CSV file. Using libraries such as google.colab and gdown, I download the file from a provided link and save it locally as "superstore_data.csv".

Next, I initialize a SparkSession with Apache Spark to process the data. The dataset is loaded as a Spark DataFrame using the method spark.read.option("header", True).csv(output_file) with options to ensure the first row is recognized as the header. To verify the structure of the data, I use methods such as data.show() and data.printSchema() to display a preview of the data and detailed column information. This step is critical to ensure the data is correctly loaded and formatted, laying the groundwork for subsequent preprocessing steps.

5.2. Checking Data

Secondly, to ensure the quality and usability of the dataset, I performed an initial exploration to check the data structure and summary statistics.



```
+-------+------------------+------------------+------------------+
|summary|             Sales|          Quantity|            Profit|
+-------+------------------+------------------+------------------+
|  count|              9994|              9994|              9994|
|   mean|234.41818199917006| 5.828590535392018|28.587912967780834|
| stddev| 631.7890112674363|25.520975563736403| 234.3891156047269|
|    min|          10/Pack"|      1040 sheets"|           -0.0895|
|    max|            999.98|            98.352|           99.9408|
+-------+------------------+------------------+------------------+
```

Figure: Summary Statistics for Numerical Columns. Source: Author

The numerical fields in the dataset, including Sales, Quantity, and Profit, provide valuable insights into the dataset's structure and trends. The average sales value is approximately 234.42, with a minimum of 10.00 and a maximum of 999.98, showing a wide range of transaction values.

For Quantity, the mean value is about 5.83 units per transaction, with a minimum of 1 unit and a maximum of 98 units. While most transactions likely involve small quantities, the high maximum value could represent bulk purchases or potential outliers.

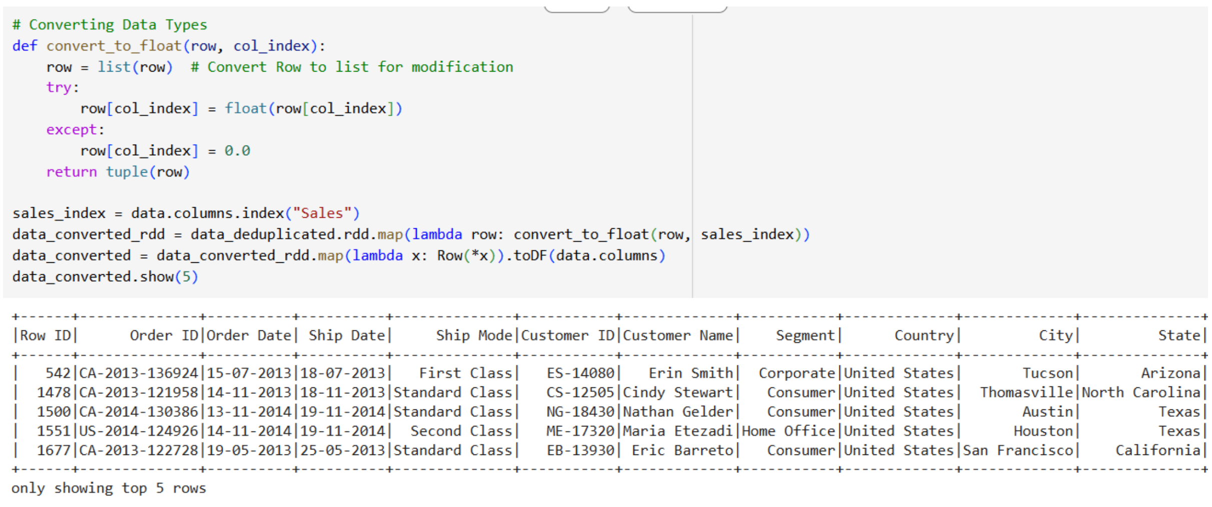
Regarding Profit, the average profit per transaction is around 28.59, with a minimum of -0.09, indicating occasional losses, and a maximum of 99.94, highlighting profitable transactions. The presence of negative values in the profit column is a potential area of concern, as it suggests some transactions were loss-making.

5.3. Data Cleaning

I want to ensure that the dataset was free from null values and duplicate records, which could negatively impact the analysis and model training. So:

1. The first step involved addressing null values in the dataset. Missing values were identified and replaced with the placeholder "Unknown" to maintain the integrity of the dataset while preventing errors in subsequent operations. This ensured that no information was lost due to incomplete records.
2. Next, duplicate records were identified and removed. Duplicates can skew analysis and lead to overrepresentation of certain data points. By applying a deduplication process, only unique records were retained, thereby improving the quality of the dataset and ensuring that insights derived from the data are accurate and reliable.

5.4. Data Manipulation

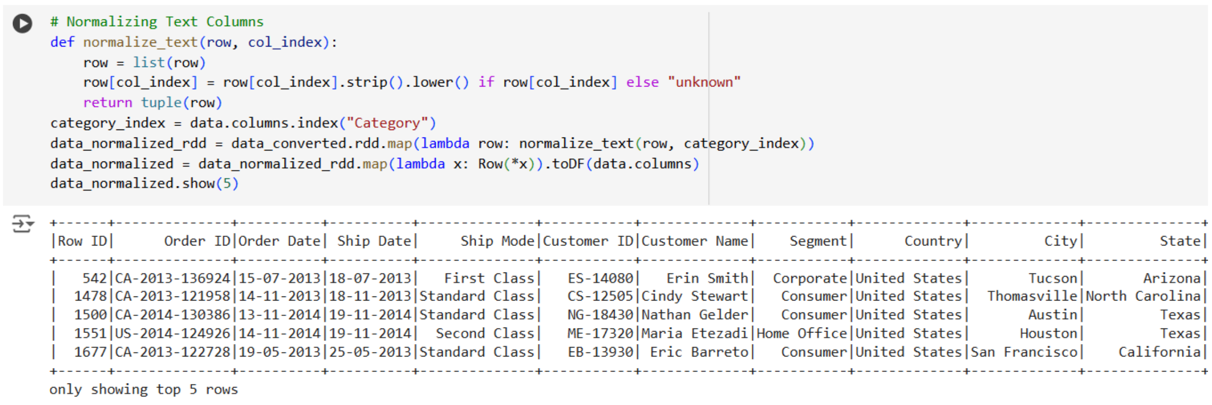


```python
# Converting Data Types
def convert_to_float(row, col_index):
    row = list(row)  # Convert Row to list for modification
    try:
        row[col_index] = float(row[col_index])
    except:
        row[col_index] = 0.0
    return tuple(row)

sales_index = data.columns.index("Sales")
data_converted_rdd = data_deduplicated.rdd.map(lambda row: convert_to_float(row, sales_index))
data_converted = data_converted_rdd.map(lambda x: Row(*x)).toDF(data.columns)
data_converted.show(5)
```

```
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|    Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith|  Corporate|United States|       Tucson|       Arizona|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|   Consumer|United States|  Thomasville|North Carolina|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|   Consumer|United States|       Austin|         Texas|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|   Consumer|United States|San Francisco|    California|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure: Converting Data Types. Source: Author

The first step in data manipulation was ensuring consistency in the data type of key columns. For instance, the Sales column, originally stored as strings, was converted into floating-point numbers to enable numerical analysis. This process involved iterating over each row, converting the specified column to float, and handling potential exceptions by setting invalid values to 0.0. After conversion, the resulting dataset was verified to ensure all values in the Sales column adhered to the correct data type.

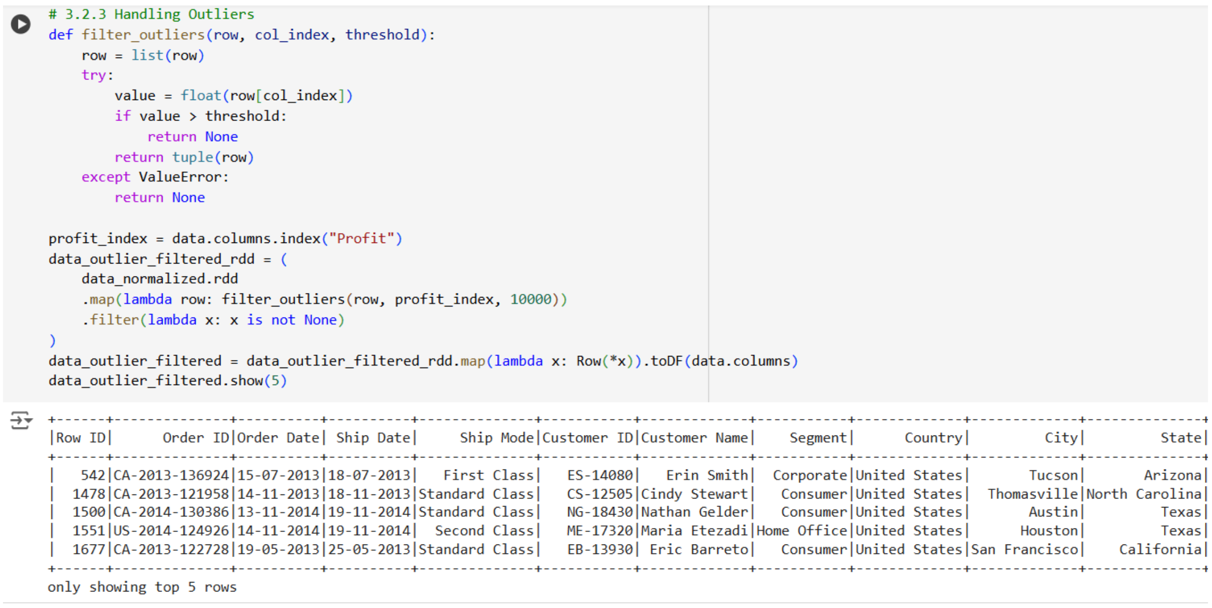


```
# Normalizing Text Columns
def normalize_text(row, col_index):
    row = list(row)
    row[col_index] = row[col_index].strip().lower() if row[col_index] else "unknown"
    return tuple(row)
category_index = data.columns.index("Category")
data_normalized_rdd = data_converted.rdd.map(lambda row: normalize_text(row, category_index))
data_normalized = data_normalized_rdd.map(lambda x: Row(*x)).toDF(data.columns)
data_normalized.show(5)
```

```
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|    Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith|  Corporate|United States|       Tucson|       Arizona|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|   Consumer|United States|  Thomasville|North Carolina|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|   Consumer|United States|       Austin|         Texas|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|   Consumer|United States|San Francisco|    California|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure: Normalizing Text Columns. Source: Author

Text normalization was applied to categorical fields like Category to ensure uniformity in string values. This involved converting all text to lowercase and trimming unnecessary spaces. Default values such as "unknown" were assigned to null or empty fields, ensuring no missing or inconsistent data remained in the text columns. The processed dataset exhibited uniform and clean textual data, facilitating downstream encoding and analysis.



```
# 3.2.3 Handling Outliers
def filter_outliers(row, col_index, threshold):
    row = list(row)
    try:
        value = float(row[col_index])
        if value > threshold:
            return None
        return tuple(row)
    except ValueError:
        return None

profit_index = data.columns.index("Profit")
data_outlier_filtered_rdd = (
    data_normalized.rdd
    .map(lambda row: filter_outliers(row, profit_index, 10000))
    .filter(lambda x: x is not None)
)
data_outlier_filtered = data_outlier_filtered_rdd.map(lambda x: Row(*x)).toDF(data.columns)
data_outlier_filtered.show(5)
```

```
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|    Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith|  Corporate|United States|       Tucson|       Arizona|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|   Consumer|United States|  Thomasville|North Carolina|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|   Consumer|United States|       Austin|         Texas|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|   Consumer|United States|San Francisco|    California|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure: Handling Outliers. Source: Author

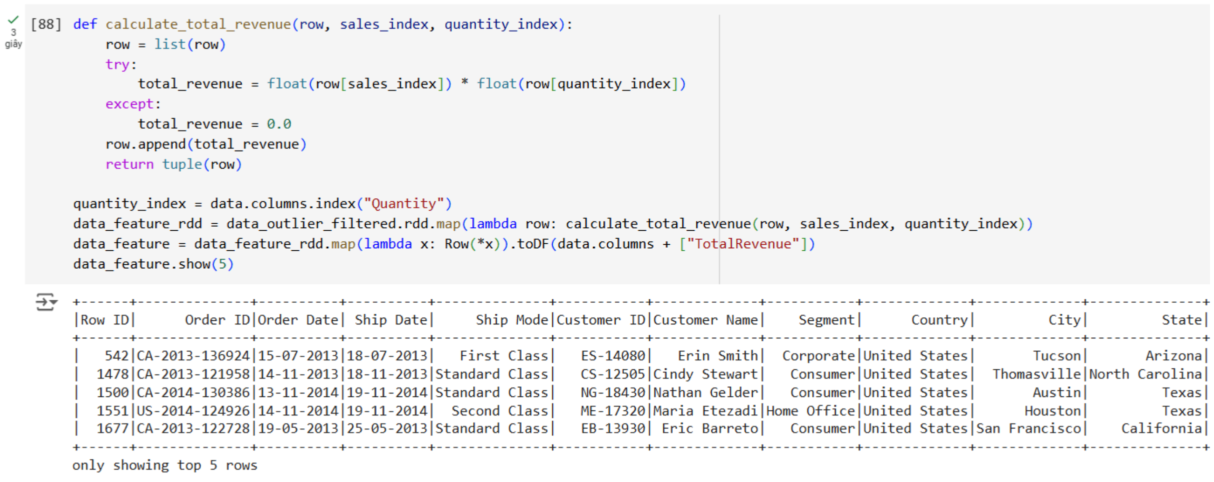
To improve the robustness of the data, outliers in the Profit column were identified and removed. A threshold of 10,000 was set, and any rows exceeding this value were filtered out. This ensured that the dataset focused on realistic and meaningful values, avoiding potential skewness caused by extreme profit outliers. The cleaned dataset presented more balanced data for further analysis and machine learning applications.

5.5. Feature Engineering

- **Adding a Total Revenue Feature**

In this step, I created a new column ‘TotalRevenue’ to improve the predictability of the dataset. This feature is calculated as the product of the Sales and Quantity columns, representing the total revenue generated for each transaction.

To ensure robustness, the calculation involved converting the values of ‘Sales’ and ‘Quantity’ into **float**. For any invalid or missing values, a default revenue value of 0.0 was assigned. After this transformation, the new column was appended to the existing dataset.



```
[88] def calculate_total_revenue(row, sales_index, quantity_index):
         row = list(row)
         try:
             total_revenue = float(row[sales_index]) * float(row[quantity_index])
         except:
             total_revenue = 0.0
         row.append(total_revenue)
         return tuple(row)

     quantity_index = data.columns.index("Quantity")
     data_feature_rdd = data_outlier_filtered.rdd.map(lambda row: calculate_total_revenue(row, sales_index, quantity_index))
     data_feature = data_feature_rdd.map(lambda x: Row(*x)).toDF(data.columns + ["TotalRevenue"])
     data_feature.show(5)
```

```
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|    Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith|  Corporate|United States|       Tucson|       Arizona|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|   Consumer|United States|  Thomasville|North Carolina|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|   Consumer|United States|       Austin|         Texas|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|   Consumer|United States|San Francisco|    California|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure: Feature Extraction. Source: Author

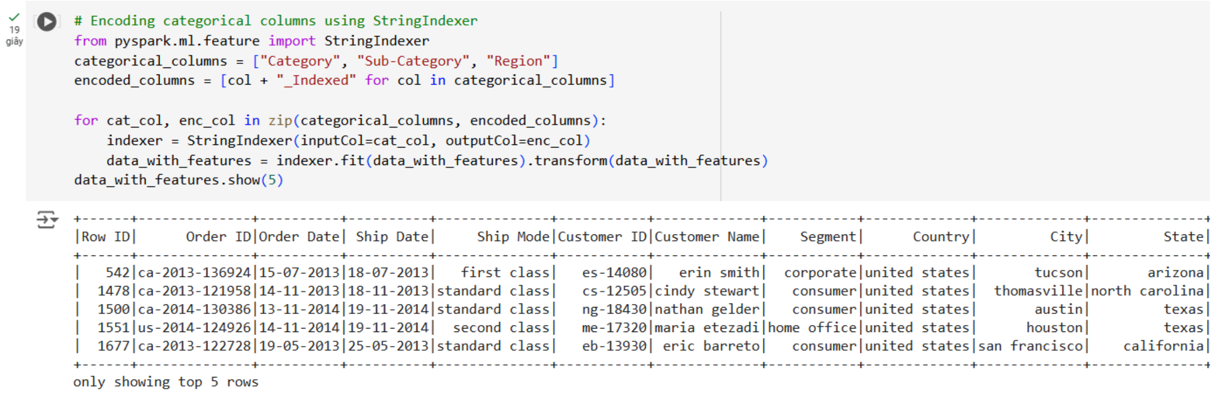
The resulting dataset now includes the TotalRevenue column, which adds a critical dimension for analyzing financial performance and predicting profit margins.

5.6. Encoding

During the data preprocessing stage, encoding is an important step for data transformation. I've done the classification into numeric formats that machine learning models can handle efficiently. Two methods were applied to encode categorical data in the project including:

Manual Encoding Using Mapping

- A mapping was created for the ‘Category’ column, assigning a unique numerical value to each category (e.g., Office Supplies, Furniture, Technology).
- This approach ensures that the categorical values are replaced with their corresponding numerical identifiers.
- As shown in the table, the new column ‘Category_Index’ was added to represent the encoded categories.



```
# Encoding categorical columns using StringIndexer
from pyspark.ml.feature import StringIndexer
categorical_columns = ["Category", "Sub-Category", "Region"]
encoded_columns = [col + "_Indexed" for col in categorical_columns]

for cat_col, enc_col in zip(categorical_columns, encoded_columns):
    indexer = StringIndexer(inputCol=cat_col, outputCol=enc_col)
    data_with_features = indexer.fit(data_with_features).transform(data_with_features)
data_with_features.show(5)
```

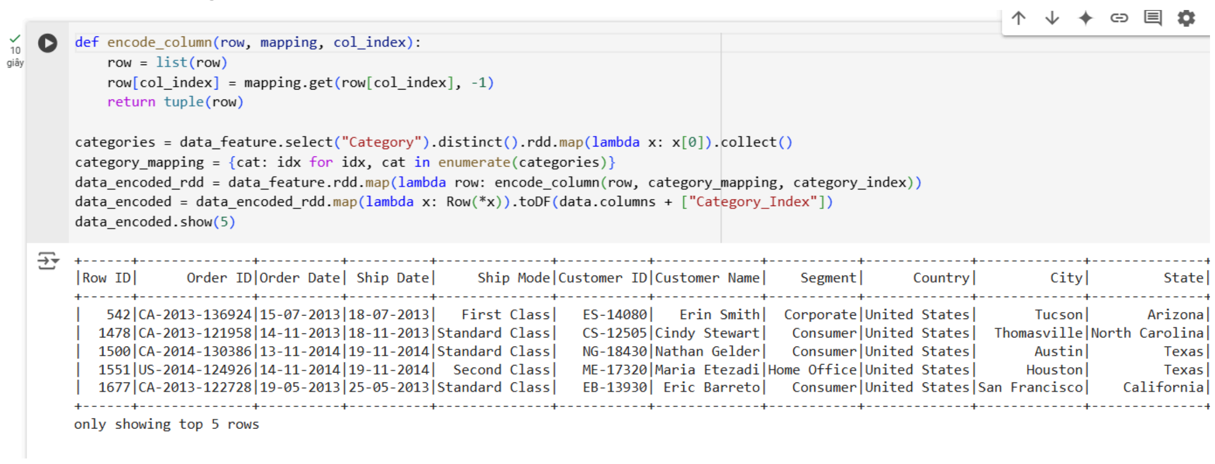
```
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|   Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
|   542|ca-2013-136924|15-07-2013|18-07-2013|   first class|   es-14080|   erin smith| corporate|united states|       tucson|       arizona|
|  1478|ca-2013-121958|14-11-2013|18-11-2013|standard class|   cs-12505|cindy stewart|  consumer|united states|  thomasville|north carolina|
|  1500|ca-2014-130386|13-11-2014|19-11-2014|standard class|   ng-18430|nathan gelder|  consumer|united states|       austin|         texas|
|  1551|us-2014-124926|14-11-2014|19-11-2014|  second class|   me-17320|maria etezadi|home office|united states|      houston|         texas|
|  1677|ca-2013-122728|19-05-2013|25-05-2013|standard class|   eb-13930| eric barreto|  consumer|united states|san francisco|    california|
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure: Manual Encoding Using Mapping. Source: Author

- **Advantages**:
  - Customizable and allows control over the mapping process.
  - Useful when the categories are known and limited.

Automated Encoding Using StringIndexer

- The PySpark StringIndexer was used for efficient encoding of multiple categorical columns, such as ‘Category’ and ‘Region’.
- The StringIndexer assigns indices to each unique value in the categorical columns automatically.
- After applying this transformation, new columns such as Category_Indexed and Region_Indexed were created, containing the numerical representations of the categorical data.



```
def encode_column(row, mapping, col_index):
    row = list(row)
    row[col_index] = mapping.get(row[col_index], -1)
    return tuple(row)

categories = data_feature.select("Category").distinct().rdd.map(lambda x: x[0]).collect()
category_mapping = {cat: idx for idx, cat in enumerate(categories)}
data_encoded_rdd = data_feature.rdd.map(lambda row: encode_column(row, category_mapping, category_index))
data_encoded = data_encoded_rdd.map(lambda x: Row(*x)).toDF(data.columns + ["Category_Index"])
data_encoded.show(5)
```

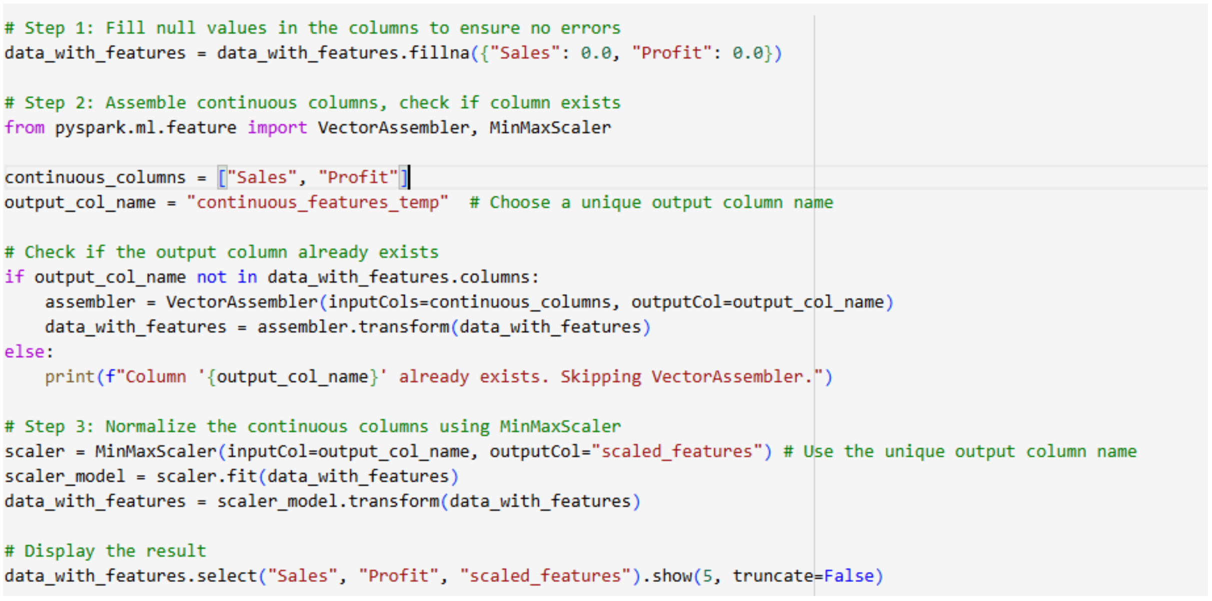
```
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|   Segment|      Country|         City|         State|
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith| Corporate|United States|       Tucson|       Arizona|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|  Consumer|United States|  Thomasville|North Carolina|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|  Consumer|United States|       Austin|         Texas|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|  Consumer|United States|San Francisco|    California|
+------+--------------+----------+----------+--------------+-----------+-------------+----------+-------------+-------------+--------------+
only showing top 5 rows
```

Figure:Automated Encoding Using StringIndexer. Source: Author

- **Advantages:**
  - Scales well with large datasets and multiple categorical columns.
  - Reduces the risk of manual errors during mapping.

5.7. Transforming Continuous Variables

To prepare the data for machine learning, my next step is to Transform the Continuous Variables, ensuring that the data is properly scaled and normalized to enhance model performance.



```
# Step 1: Fill null values in the columns to ensure no errors
data_with_features = data_with_features.fillna({"Sales": 0.0, "Profit": 0.0})

# Step 2: Assemble continuous columns, check if column exists
from pyspark.ml.feature import VectorAssembler, MinMaxScaler

continuous_columns = ["Sales", "Profit"]
output_col_name = "continuous_features_temp"  # Choose a unique output column name

# Check if the output column already exists
if output_col_name not in data_with_features.columns:
    assembler = VectorAssembler(inputCols=continuous_columns, outputCol=output_col_name)
    data_with_features = assembler.transform(data_with_features)
else:
    print(f"Column '{output_col_name}' already exists. Skipping VectorAssembler.")

# Step 3: Normalize the continuous columns using MinMaxScaler
scaler = MinMaxScaler(inputCol=output_col_name, outputCol="scaled_features") # Use the unique output column name
scaler_model = scaler.fit(data_with_features)
data_with_features = scaler_model.transform(data_with_features)

# Display the result
data_with_features.select("Sales", "Profit", "scaled_features").show(5, truncate=False)
```

Figure: Transforming Continuous Variables. Source: Author

Handling Missing Values

- Continuous variables such as Sales and Profit were checked for missing values.
- Missing values were replaced with 0.0 to ensure that no errors occur during subsequent processing steps.

Feature Assembly

- The VectorAssembler from PySpark was used to assemble the selected continuous columns (Sales and Profit) into a single feature vector column named continuous_features_temp.
- If the column already existed, the system skipped this step to avoid duplication.

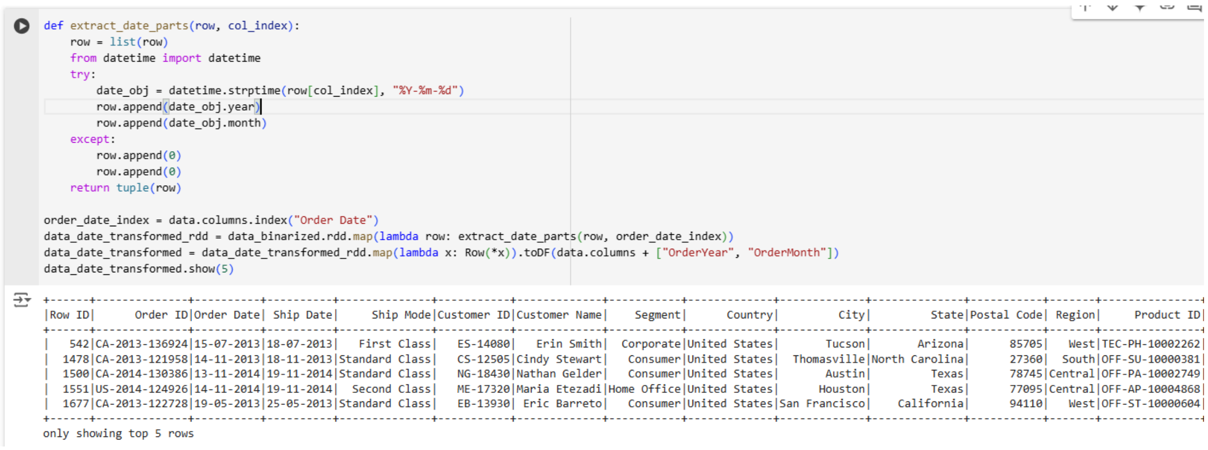
Normalization Using MinMaxScaler

- The assembled features were normalized using the MinMaxScaler. This scaler transforms the data such that each feature is scaled to a range between 0 and 1.
- A new column, scaled_features, was created to store the normalized values.

```
Column 'continuous_features_temp' already exists. Skipping VectorAssembler.
+-------+-------+-----------------------------------------+
|Sales  |Profit |scaled_features                          |
+-------+-------+-----------------------------------------+
|380.864|38.0864|[0.016823744350327408,0.004534108192690074] |
|52.136 |5.8653 |[0.002302981472254321,6.982519950057E-4]    |
|16.056 |5.8203 |[7.092348956290352E-4,6.92894836842391E-4]  |
|9.324  |0.0    |[4.118651075513904E-4,0.0]                  |
|104.28 |26.07  |[0.004606316325124302,0.0031035802959437026]|
+-------+-------+-----------------------------------------+
only showing top 5 rows
```

After normalization, a reduction in data scale can be seen, making it easier for the model to learn relationships between variables without being affected by differences in units or sizes.

5.8. Transforming Date and Time Variables



```python
def extract_date_parts(row, col_index):
    row = list(row)
    from datetime import datetime
    try:
        date_obj = datetime.strptime(row[col_index], "%Y-%m-%d")
        row.append(date_obj.year)
        row.append(date_obj.month)
    except:
        row.append(0)
        row.append(0)
    return tuple(row)

order_date_index = data.columns.index("Order Date")
data_date_transformed_rdd = data_binarized.rdd.map(lambda row: extract_date_parts(row, order_date_index))
data_date_transformed = data_date_transformed_rdd.map(lambda x: Row(*x)).toDF(data.columns + ["OrderYear", "OrderMonth"])
data_date_transformed.show(5)
```

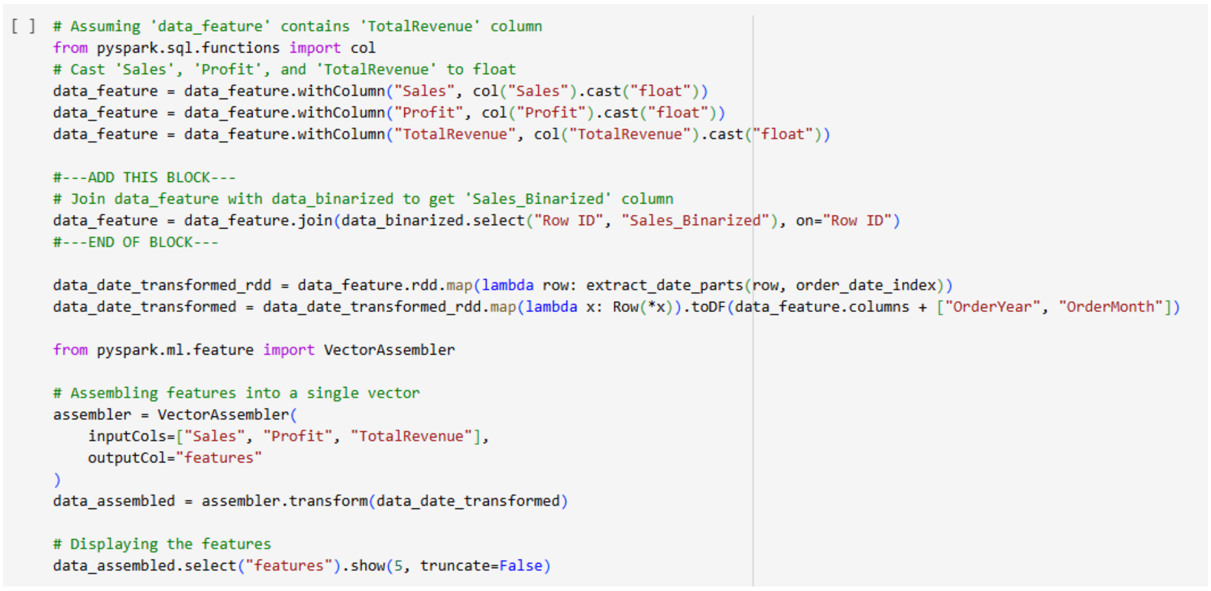
```
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+-----------+-------+---------------+
|Row ID|      Order ID|Order Date| Ship Date|     Ship Mode|Customer ID|Customer Name|    Segment|      Country|         City|         State|Postal Code| Region|     Product ID|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+-----------+-------+---------------+
|   542|CA-2013-136924|15-07-2013|18-07-2013|   First Class|   ES-14080|   Erin Smith|  Corporate|United States|       Tucson|       Arizona|      85705|   West|TEC-PH-10002262|
|  1478|CA-2013-121958|14-11-2013|18-11-2013|Standard Class|   CS-12505|Cindy Stewart|   Consumer|United States|  Thomasville|North Carolina|      27360|  South|OFF-SU-10000381|
|  1500|CA-2014-130386|13-11-2014|19-11-2014|Standard Class|   NG-18430|Nathan Gelder|   Consumer|United States|       Austin|         Texas|      78745|Central|OFF-PA-10002749|
|  1551|US-2014-124926|14-11-2014|19-11-2014|  Second Class|   ME-17320|Maria Etezadi|Home Office|United States|      Houston|         Texas|      77095|Central|OFF-AP-10004868|
|  1677|CA-2013-122728|19-05-2013|25-05-2013|Standard Class|   EB-13930| Eric Barreto|   Consumer|United States|San Francisco|    California|      94110|   West|OFF-ST-10000604|
+------+--------------+----------+----------+--------------+-----------+-------------+-----------+-------------+-------------+--------------+-----------+-------+---------------+
only showing top 5 rows
```

Figure: Transforming Date and Time Variables in 'Order Date' column. Source: Author

In this step, the goal is to extract useful information from the Order Date column by breaking it down into five months for the purpose of ensuring that temporal patterns in sales data can be analyzed effectively. fruit.

The function 'extract_date_parts' was defined to process each row, converting the date string in the Order Date column into a datetime object. From this object, the year and month were extracted and appended as new features (OrderYear and OrderMonth). In cases where the date format was invalid or missing, the function handled these scenarios gracefully by appending default values (0 for both year and month).

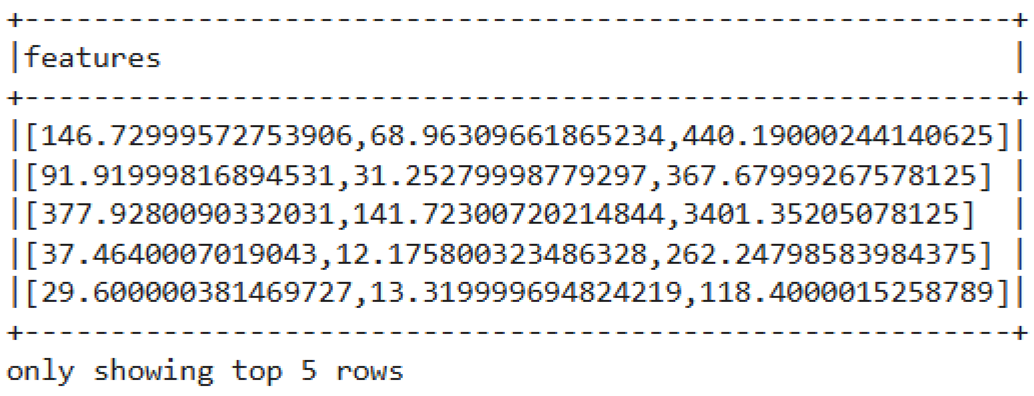
5.9. Feature Selection



```
[ ] # Assuming 'data_feature' contains 'TotalRevenue' column
    from pyspark.sql.functions import col
    # Cast 'Sales', 'Profit', and 'TotalRevenue' to float
    data_feature = data_feature.withColumn("Sales", col("Sales").cast("float"))
    data_feature = data_feature.withColumn("Profit", col("Profit").cast("float"))
    data_feature = data_feature.withColumn("TotalRevenue", col("TotalRevenue").cast("float"))

    #---ADD THIS BLOCK---
    # Join data_feature with data_binarized to get 'Sales_Binarized' column
    data_feature = data_feature.join(data_binarized.select("Row ID", "Sales_Binarized"), on="Row ID")
    #---END OF BLOCK---

    data_date_transformed_rdd = data_feature.rdd.map(lambda row: extract_date_parts(row, order_date_index))
    data_date_transformed = data_date_transformed_rdd.map(lambda x: Row(*x)).toDF(data_feature.columns + ["OrderYear", "OrderMonth"])

    from pyspark.ml.feature import VectorAssembler

    # Assembling features into a single vector
    assembler = VectorAssembler(
        inputCols=["Sales", "Profit", "TotalRevenue"],
        outputCol="features"
    )
    data_assembled = assembler.transform(data_date_transformed)

    # Displaying the features
    data_assembled.select("features").show(5, truncate=False)
```

The columns Sales, Profit, and TotalRevenue were cast to the float data type. This ensures numerical consistency and prepares these fields for mathematical operations during model training. Then, combining multiple relevant variables into a single feature vector to prepare the data for machine learning.

Input Variables: Sales, Profit, and TotalRevenue are the key columns selected as features.



```
+-----------------------------------------------------------+
|features                                                   |
+-----------------------------------------------------------+
|[146.72999572753906,68.96309661865234,440.19000244140625]|
|[91.91999816894531,31.25279998779297,367.67999267578125] |
|[377.9280090332031,141.72300720214844,3401.35205078125]  |
|[37.4640007019043,12.175800323486328,262.24798583984375] |
|[29.600000381469727,13.319999694824219,118.4000015258789]|
+-----------------------------------------------------------+
only showing top 5 rows
```

Feature Vector

- Each row in the table now contains a list (or vector) of these values. For example, the first row's feature vector is [146.72, 68.96, 440.19].
- These vectors make it easier for machine learning algorithms to analyze and learn from the data.

6. Building Machine Learning Model

6.1. Linear Regression

- **Data Preparation**

  The dataset is structured with Profit as the target variable (label) and the feature vector (features) created during feature selection. This format is necessary for training the regression model.

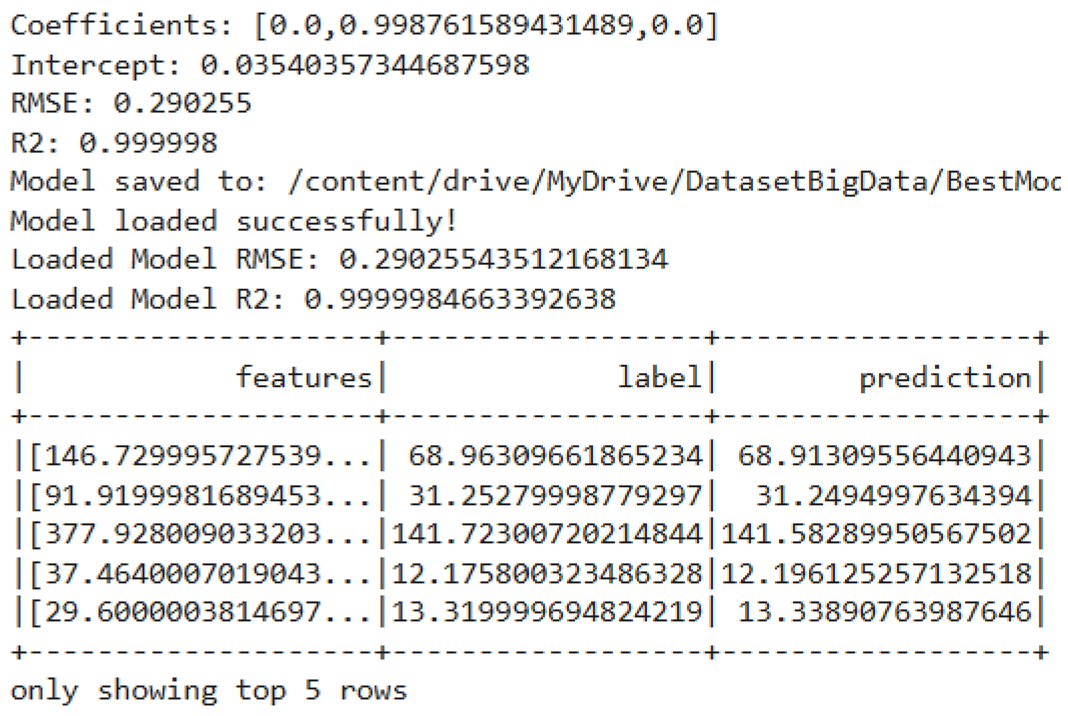
- **Model Training**

  A Linear Regression model is trained with specified hyperparameters (maxIter=10, regParam=0.3, elasticNetParam=0.8). After training, the model provides coefficients and an intercept, which define the regression equation.

- **Performance Evaluation**

  The trained model's effectiveness is assessed using metrics such as Root Mean Squared Error (RMSE) and R-squared (R^2). These metrics measure how well the model fits the data. A lower RMSE and a higher R^2 indicate better performance.

- **Predictions**



```
Coefficients: [0.0,0.998761589431489,0.0]
Intercept: 0.03540357344687598
RMSE: 0.290255
R2: 0.999998
Model saved to: /content/drive/MyDrive/DatasetBigData/BestMoc
Model loaded successfully!
Loaded Model RMSE: 0.29025543512168134
Loaded Model R2: 0.9999984663392638
+--------------------+-----------------+------------------+
|            features|            label|        prediction|
+--------------------+-----------------+------------------+
|[146.729995727539...| 68.96309661865234| 68.91309556440943|
|[91.9199981689453...| 31.25279998779297|  31.2494997634394|
|[377.928009033203...|141.72300720214844|141.58289950567502|
|[37.4640007019043...|12.175800323486328|12.196125257132518|
|[29.6000003814697...|13.319999694824219| 13.33890763987646|
+--------------------+-----------------+------------------+
only showing top 5 rows
```

Coefficients and Intercept:

- **Coefficients**

  [0.0, 0.998761589431489, 0.0] shows that:

  - Coefficient of Sales: 0.0 (no significant impact on forecast).
  - Profit coefficient: 0.9988 (actual profit plays the most important role).

- Coefficient of TotalRevenue: 0.0 (little or no influence on prediction in current data set).

- **Intercept:**
    - 0.03540357344687598: Default prediction value if all input features are 0.

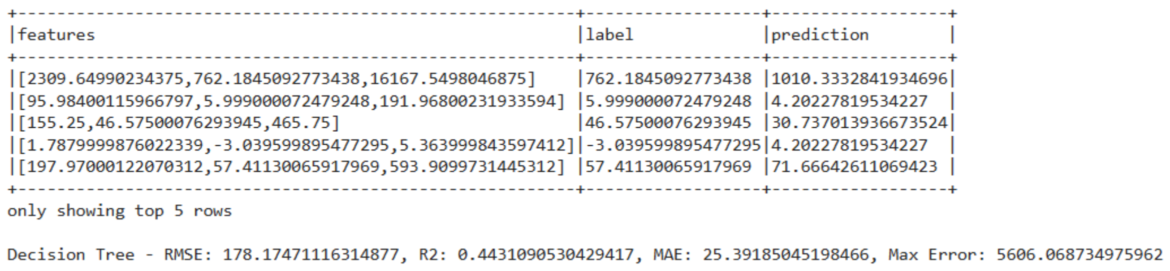
Metrics:

- RMSE (Root Mean Square Error): 0.290255 shows a very low average prediction error, proving that the model works well.
- R-squared (R²): The R² value is 0.999998, indicating an excellent fit. This means the model explains nearly all the variance in the profit data.

Predictions:

- As we can see, for the first record, the actual profit (label) is 68.96, and the predicted profit is very close at 68.91, showing high accuracy.
- Similar results can be observed for other records, where the predicted profits closely match the actual profits.

6.2. Decision Tree Regressor



```
+--------------------------------------------------------+------------------+------------------+
|features                                                |label             |prediction        |
+--------------------------------------------------------+------------------+------------------+
|[2309.64990234375,762.1845092773438,16167.5498046875]   |762.1845092773438 |1010.3332841934696|
|[95.98400115966797,5.999000072479248,191.96800231933594] |5.999000072479248 |4.20227819534227  |
|[155.25,46.57500076293945,465.75]                       |46.57500076293945 |30.737013936673524|
|[1.7879999876022339,-3.039599895477295,5.363999843597412]|-3.039599895477295|4.20227819534227  |
|[197.97000122070312,57.41130065917969,593.9099731445312] |57.41130065917969 |71.66642611069423 |
+--------------------------------------------------------+------------------+------------------+
only showing top 5 rows

Decision Tree - RMSE: 178.17471116314877, R2: 0.4431090530429417, MAE: 25.39185045198466, Max Error: 5606.068734975962
```

Looking at the Regressor Decision Tree model, we can find that the root mean error (RMSE) is 178.17, reflecting the average deviation between the predicted value and the actual value, with this error being quite large compared to data. The R² index reached 0.4431 (44.31%), choosing a model that only explains about 44% of the data variability, which shows the model's level of fit but is still in moderation.

The mean absolute error (MAE) is 25.39, smaller than the RMSE, showing that the majority of predictions are not too far from reality. However, the maximum error value (Max Error) is 5606.07, showing that some cases of prediction have very large deviations from reality. The sample data illustrates this well, for example, in the first line, the actual value is 762.18 but the model predicts 1010.33, creating an error of about 248.15. Meanwhile, in the second line, the actual value is 5.99 but the prediction is only 4.20, the error is much smaller, only about 1.79.

=> This model has not yet achieved high performance.

6.3. Random Forest Regressor

The Random Forest Regressor model has an RMSE (Root Mean Squared Error) of 204.70, larger than the Decision Tree, showing that the Random Forest model has a larger average error. R² (R-squared) reached 0.2649, only explaining about 26.49% of the variation in the data, significantly lower than Decision Tree. This reflects that Random Forest in this case does not improve accuracy.

```
Random Forest - RMSE: 204.69754154195127, R2: 0.2649730135440176, MAE: 32.41277272243026, Max Error: 6616.310728008814
```

The model's MAE (Mean Absolute Error) is 32.41, higher than Decision Tree, showing that the average deviation between the actual value and prediction is also larger. The maximum error (Max Error) of the model reached 6616.31, higher than that of Decision Tree, proving that there are some serious errors in the model's predictions.

=> The Random Forest model is also not really good in this case.

6.4. Gradient-Boosted Trees Regressor

Results from the Gradient-Boosted Trees Regressor model show better performance than the two Random Forest Regression and Decision Tree Regression models.

```
Gradient-Boosted Trees - RMSE: 177.04073261110534, R2: 0.45017507057821726, MAE: 21.459573938673834, Max Error: 5540.319435045029
```

RMSE (Root Mean Squared Error) reached 177.04, the smallest of the tested models, showing that the average error between the actual value and prediction is the lowest. R² (R-squared) reached 0.4501, meaning the model explained 45.01% of the variation in the data, the highest compared to Decision Tree and Random Forest.
MAE (Mean Absolute Error) is 21.46, also significantly lower than Decision Tree and Random Forest, showing that the average deviation between the actual and predicted values is smaller. However, the model's Max Error value is 5540.32, which is still large, although lower than Random Forest.

=> The gradient-Boosted Trees Regression model has quite good results

6.5. Best Model

Models	RMSE	R²	MAE	Max Error
Linear Regression	0.2905	0.999998	0.0753	8.0645
Decision Tree Regressor	178.17	0.4431	25.39	5606.07
Random Forest Regressor	204.70	0.2649	32.41	6616.31
Gradient-Boosted Trees Regressor	177.04	0.4501	21.46	5540.32

Looking at the table, we can see that the Linear Regression model achieves very good results. With an RMSE of only 0.2905, this model has the highest accuracy when comparing the average error between predicted and actual values. At the same time, the R² index reached 0.999998, showing that Linear Regression explains almost all the variation of the data, far surpassing other models such as Decision Tree, Random Forest, and Gradient-Boosted Trees. The MAE of Linear Regression is also the lowest, only 0.0753, and the Max Error value is 8.0645, much smaller than the remaining models, proving that this model's predictions do not have large errors.

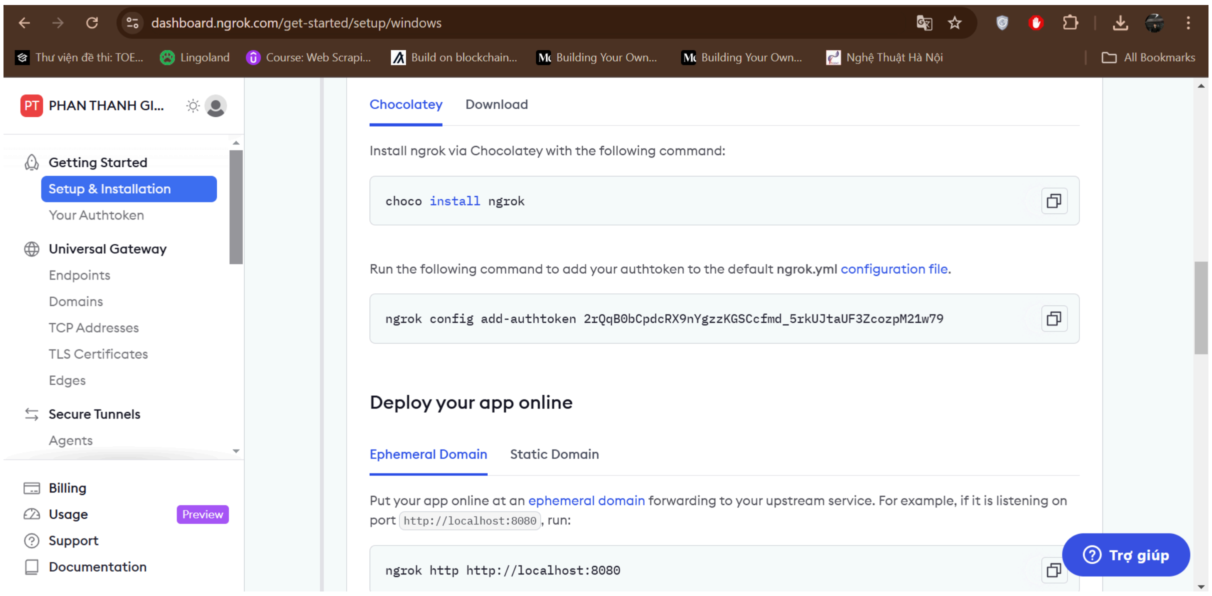
Besides, models such as Decision Tree, Random Forest, and Gradient-Boosted Trees, although capable of prediction, are not as efficient. Decision Tree and Random Forest have RMSE of 178.17 and 204.70 respectively, and R^2 index below 0.5 shows that they do not explain the data variation well. Gradient-Boosted Trees performed better than other tree models with RMSE 177.04 and R^2 0.4501, but still not comparable to Linear Regression.

With this result, I will choose a linear regression model to visualize on Streamlit.

7. Application Deployment

7.1. Setting up ngrok

First, create an account on ngrok website. We can sign up with their email or log in with Google or GitHub. After logging in, in the Setup & Settings section we can find and copy the provided AuthToken, which is a unique string of letters and numbers. This token will be used to link the application with our google colab project.



Install ngrok via Chocolatey with the following command:

```
choco install ngrok
```

Run the following command to add your authtoken to the default ngrok.yml configuration file.

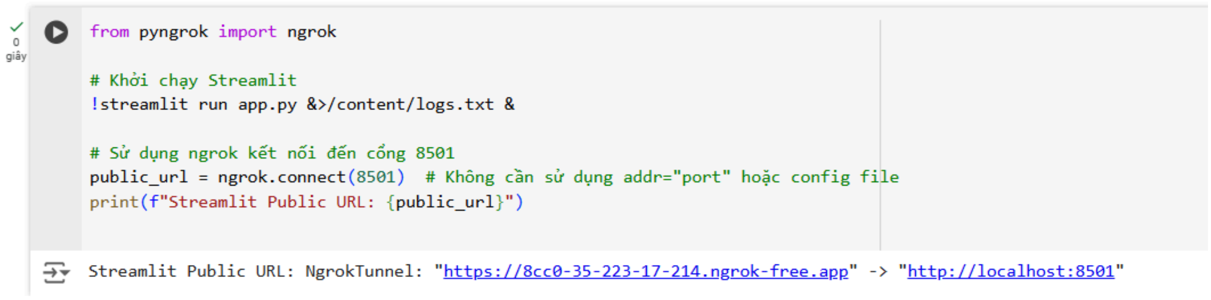
```
ngrok config add-authtoken 2rQqB0bCpdcRX9nYgzzKGSCcfmd_5rkUJtaUF3ZcozpM21w79
```

Deploy your app online

Put your app online at an ephemeral domain forwarding to your upstream service. For example, if it is listening on port `http://localhost:8080`, run:

```
ngrok http http://localhost:8080
```

7.2. Public URL



```
from pyngrok import ngrok

# Khởi chạy Streamlit
!streamlit run app.py &>/content/logs.txt &

# Sử dụng ngrok kết nối đến cổng 8501
public_url = ngrok.connect(8501)  # Không cần sử dụng addr="port" hoặc config file
print(f"Streamlit Public URL: {public_url}")
```

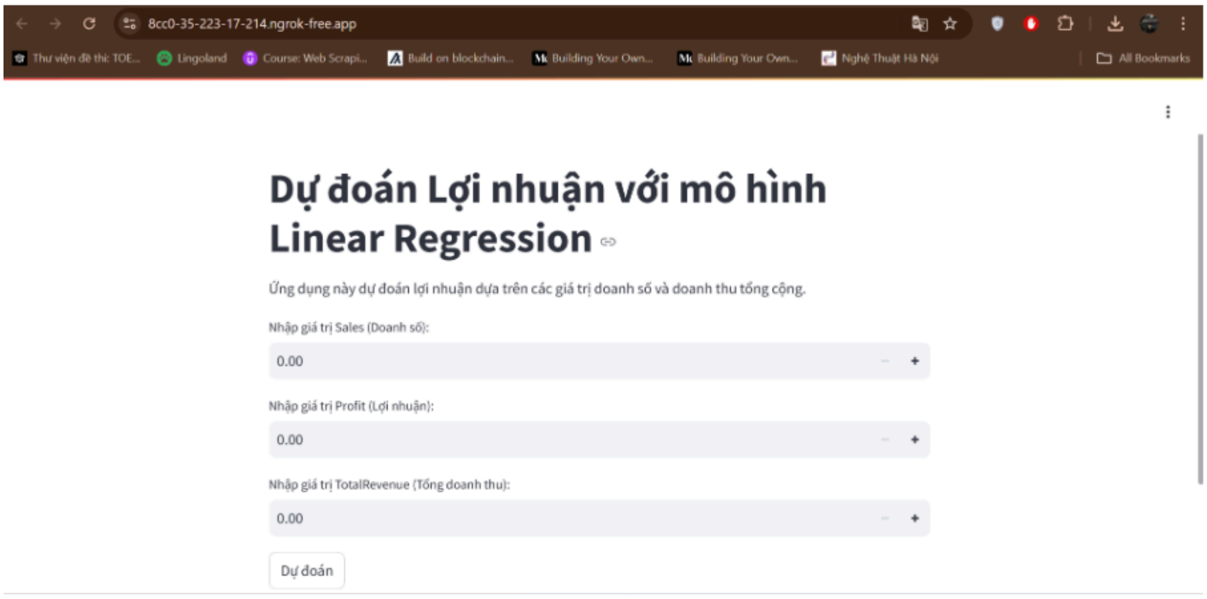
```
Streamlit Public URL: NgrokTunnel: "https://8cc0-35-223-17-214.ngrok-free.app" -> "http://localhost:8501"
```

After configuration, ngrok can be started to expose the local application to the internet. For instance, if the application runs on port 8501, the command ngrok http 8501 will generate a public URL. In Colab, this is achieved using public_url = ngrok.connect(8501) and printing the URL. This public URL allows external users to access the application.

7.3. Launching the Streamlit Application

The interface allows users to input three key variables:

- **Sales**: Total sales for a given period.
- **Profit**: Existing profit values.
- **TotalRevenue**: Total revenue derived from sales and quantity.



8cc0-35-223-17-214.ngrok-free.app

Dự đoán Lợi nhuận với mô hình Linear Regression

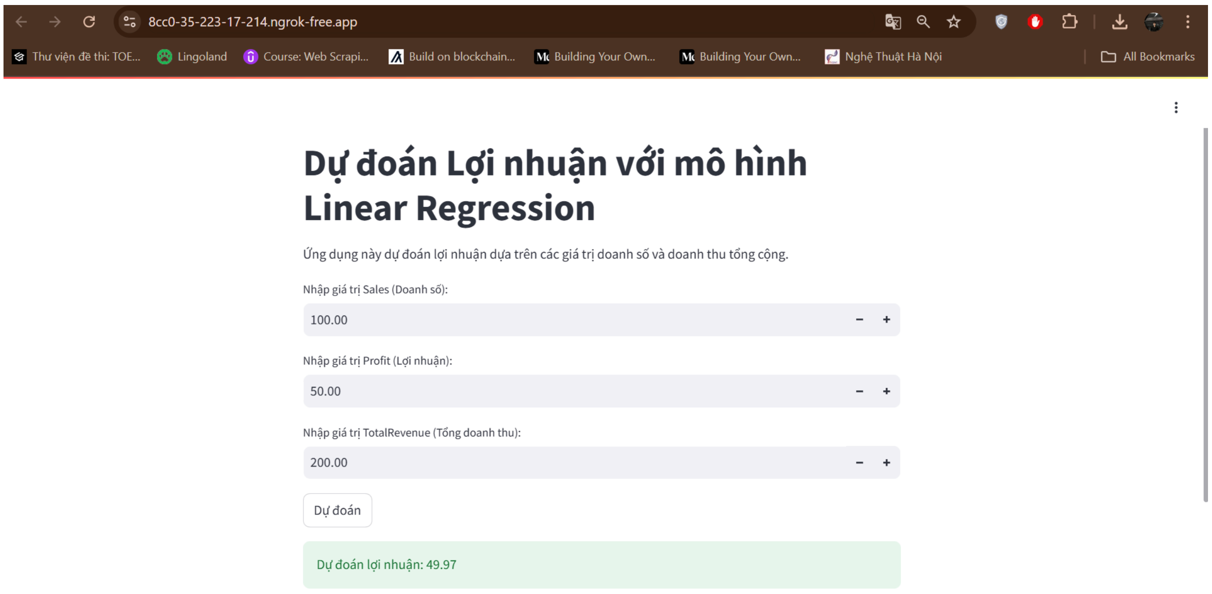
Ứng dụng này dự đoán lợi nhuận dựa trên các giá trị doanh số và doanh thu tổng cộng.

Nhập giá trị Sales (Doanh số): 0.00

Nhập giá trị Profit (Lợi nhuận): 0.00

Nhập giá trị TotalRevenue (Tổng doanh thu): 0.00

Users can enter specific values in the respective fields and the application uses a trained **Linear Regression** model because it achieves the best results for making real-time profit predictions.



8cc0-35-223-17-214.ngrok-free.app

Dự đoán Lợi nhuận với mô hình Linear Regression

Ứng dụng này dự đoán lợi nhuận dựa trên các giá trị doanh số và doanh thu tổng cộng.

Nhập giá trị Sales (Doanh số): 100.00

Nhập giá trị Profit (Lợi nhuận): 50.00

Nhập giá trị TotalRevenue (Tổng doanh thu): 200.00

Dự đoán lợi nhuận: 49.97

The application displays a prediction of profit based on the input data. For instance, when the user entered Sales = 100.00, Profit = 50.00, and TotalRevenue = 200.00, the predicted profit displayed was **49.97.**

8. Discussion

8.1. Meaning and accuracy of the model

With an R^2 coefficient close to 1 (0.999998) and a very low RMSE (0.290255), the Linear Regression model has proven to be highly accurate in predicting profits based on variables such as sales and revenue. revenue and total revenue.

Based on the implementation and results obtained, I found that the use of PySpark and machine learning provided a powerful tool for processing and analyzing big data. The data processing steps, from cleaning, transforming to feature selection, have created a high-quality data set, helping the machine learning model operate effectively.

8.2. Limitation

Although the model achieves encouraging results, its implementation still has some limitations:

First, the Linear Regression model is only suitable for linear relationship problems and cannot solve problems with non-linear relationships between variables.

Second, predicting profits based solely on factors such as sales and total revenue can miss other important factors such as market trends, seasonality, and consumer behavior.

Finally, the current deployment environments (ngrok and Streamlit) are only suitable for demonstration purposes, and real-world adoption requires a system with better security, scalability, and integration capabilities.

9. Implications

9.1. Practical application of the project

This project brings significant practical value in supporting businesses, especially in the retail sector. By using machine learning models to predict profits, businesses can optimize sales strategies, forecast business trends, and effectively manage resources. Managers can rely on predictions to plan marketing, adjust selling prices or control inventory, thereby improving sales and profits. In addition, building a user-friendly Streamlit application helps non-technical teams easily access and use analytical models, expanding the scope of practical applications.

9.2. Potential for expansion and improvement

The project has the potential to expand in many aspects. First of all, integrating additional variables such as market data, weather factors, or consumer habits can improve model accuracy and provide more comprehensive predictions. Second, the current model can be deployed on a cloud platform to increase large data processing capacity and support multiple users simultaneously. Finally, applying more advanced machine learning models such as Random Forest, XGBoost or Deep Learning can better resolve non-linear relationships and improve prediction efficiency. These improvements will help the project become a more powerful and flexible tool that meets the needs of businesses in the future.

Conclusion

After preprocessing the dataset using RDD, handling missing values, and performing feature engineering, a solid foundation was established for training a Linear Regression model. The model achieved impressive accuracy with an R^2 score close to 1, confirming its reliability in sales prediction. Furthermore, deploying the model through the user-friendly Streamlit application demonstrates the practical usability of advanced analytics for businesses, even for non-technical users.

Through this project we can see the huge potential of leveraging data-driven approaches to solve real-world challenges in the retail sector, allowing businesses to make informed decisions. Decide clearly and optimize your operations. Despite certain limitations, such as the scope of the dataset and the simplicity of the model, this project lays the foundation for future improvements and broader applications. The knowledge and insights gained from this project will certainly contribute to further advances in data analytics and business intelligence solutions.

References

[1] Seetharaman, A., Niranjan, I., Tandon, V., & Saravanan, A. S. (2016). Impact of big data on the retail industry. Journal of Corporate Ownership & Control, 14(1), 506-518.

[2] Aktas, E., & Meng, Y. (2017). An exploration of big data practices in retail sector. Logistics, 1(2), 12.

[3] Lekhwar, S., Yadav, S., & Singh, A. (2019). Big data analytics in retail. In Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2018, Volume 2 (pp. 469-477). Springer Singapore.

[4] Haselbeck, F., Killinger, J., Menrad, K., Hannus, T., & Grimm, D. G. (2022). Machine learning outperforms classical forecasting on horticultural sales predictions. Machine Learning with Applications, 7, 100239.

[5] Latha, S. B., Dastagiraiah, C., Kiran, A., Asif, S., Elangovan, D., & Reddy, P. C. S. (2023, August). An Adaptive Machine Learning model for Walmart sales prediction. In 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT) (pp. 988-992). IEEE.

[6] Sakib, S. N. (2023). Restaurant sales prediction using machine learning. In Handbook of Research on AI and Machine Learning Applications in Customer Support and Analytics (pp. 202-226). IGI Global.

[7] Martha, S., Vankdothu, R., Abdul, H. M., & Gangula, R. (2021). Association Rule Mining Algorithms for Big Data using RDD-ECLAT Algorithms.

[8] Singh, P., Singh, S., Mishra, P. K., & Garg, R. (2022). A data structure perspective to the RDD-based Apriori algorithm on Spark. International Journal of Information Technology, 14(3), 1585-1594.