SALES PRICE OPTIMIZATION USING MACHINE LEARNING

A MINI PROJECT REPORT

Submitted by

THIRUMURUGAN R - 410620243029

MUTHU KUMAR R - 410620243019

in partial fulfillment for the award of the degree

of

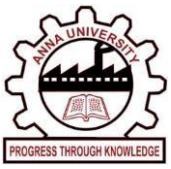
BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

DHAANISH AHMED COLLEGE OF ENGINEERING,

PADAPPAI, CHENNAI - 601 301.



ANNA UNIVERSITY: CHENNAI 600 025.

NOVEMBER 2023

BONAFIDE CERTIFICATE

Certified that this mini project report **"SALES PRICE OPTIMIZATION USING MACHINE LEARNING"** is the bonafide work of **THIRUMURUGAN R (410620243029)** and **MUTHU KUMAR R (410620243019)** who carried out the mini project work under my supervision.

SIGNATURE
Mr. G. RAJASEKARAN, M.E.,
HEAD OF THE DEPARTMENT

Artificial Intelligence and Data Science,
Dhaanish Ahmed College of Engineering,
Padappai, Chennai – 601 301.

SIGNATURE
Mrs. R. SIVAKANI, M.E.,(Ph.D),
Assistant Professor
Supervisor

Artificial Intelligence and Data Science,
Dhaanish Ahmed College of Engineering,
Padappai, Chennai – 601 301.

Submitted for the Anna University project practical held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We thank our almighty god and our beloved parents for their consistent support and encouragement provided from the beginning of our mini project work and till its completion.

We express our heart-felt gratitude to our Chairman **ALHAJ K. MOOSA,** and to our Secretary **Mr. M. KADARSHAH, B.A., M.B.A.,** for providing necessary facilities for the successful completion of our mini project.

We take the privilege to express our indebted sense of gratitude to our beloved Principal, **Dr. UMA GOWRI, M.E., Ph.D.,** and our respected director sir **Dr. P. PARAMASIVAN, M.Sc., M.Phil., Ph.D.,** Dhaanish Ahmed College of Engineering for granting permission to undertake the mini project in our college.

We would express our sincere thanks to our Head of the Department and Professor **Mr. G. RAJASEKARAN, M.E.,** for his kind permission to carry out the Mini project and our Mini Project Coordinator **Mr. G. RAJASEKARAN, M.E.,** for the encouragement given to complete the mini project.

We would like to extend our thanks to our Project Supervisor and assistant professor **Mrs. R. SIVAKANI, M.E., (Ph.D.,)** Assistant Professor, for her excellent guidance to complete the project.

We are extremely thankful to our Faculty Members of the Department of artificial intelligence and data science, for their valuable support throughout this mini project.

We also thank our Family and Friends for their moral.

Finally we thank one and all those who have rendered help directly or indirectly at various stages of the mini project.

ABSTRACT

Sales price optimization is a critical aspect of business strategy, influencing profitability, market competitiveness, and customer satisfaction. Traditional pricing methods often fall short in the dynamic and data-rich landscape of today's business environment. In response to this challenge, the integration of machine learning techniques has emerged as a transformative approach to price optimization, enabling companies to make data-driven pricing decisions with unprecedented accuracy and efficiency. the key concepts and implications of sales price optimization using machine learning, shedding light on its significance and the benefits it offers to businesses across various industries. Machine learning models can perform price sensitivity analysis, helping businesses understand how price changes impact demand and profitability. This insight is invaluable in determining optimal pricing strategies to balance market share and revenue goals. The abstract includes examples of successful applications of machine learning in sales price optimization across various industries, such as e-commerce, hospitality, and retail. These case studies highlight the tangible benefits and returns on investment achieved through the adoption of machine learning-based pricing strategies. They can anticipate shifts in consumer sentiment, market trends, and competitor moves, allowing them to adjust their pricing strategies proactively and maintain a strong market position.

KEYWORDS

Sales price optimization, Machine learning, Pricing strategies, Data-driven insights, Personalization, Dynamic pricing, Competitive advantage, Price sensitivity analysis, Ethical considerations.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**iv**
	LIST OF FIGURES	**vii**
1	**INTRODUCTION**	1
	1.1 DATA-DRIVEN INSIGHTS	2
	1.2 TRADITIONAL PRICING STRATERGIES	2
	1.3 COST-PLUS PRICING	3
	1.4 COMPETITOR-BASED PRICING	3
	1.5 VALUE-BASED PRICING	3
	1.6 PSYCHOLOGICAL PRICING	4
	1.7 STATIC PRICING	4
	1.8 EXPERT JUDGEMENT	4
	1.9 MARKET RESEARCH	4
	1.10 DYNAMIC PRICING	5
	1.11 PRICE SENSITIVITY ANALYSIS	5
2	**REQUIREMENT SPECIFICATION**	6
	2.1 GENERAL	6
	2.1.1 HARDWARE REQUIREMENTS	6
	2.1.2 SOFTWARE REQUIREMENTS	7
3	**METHODOLOGY**	8
	3.1 PROBLEM IDENTIFICATION	8
	3.2 COURSE SELECTION AND IDENTIFICATION	8
	3.3 MODELING RECOMMENDATION SYSTEM	9
	3.3.1 CONTENT BASED RECOMMENDATION	9
	3.4 DESIGNING	9
	3.5 SYSTEM ANALYSIS	9
	3.6 IMPLEMENTATION	10

4	**UML DIAGRAMS**	11
	4.1 USE-CASE DIAGRAM	11
	4.2 SEQUENCE DIAGRAM	13
	4.3 ACTIVITY DIAGRAM	14
5	**SYSTEM IMPLEMENTATION**	15
	5.1 DECISION TREE ALGORITHM	15
	5.2 OTHER ALGORITHM	17
6	**CONCLUSION AND FUTURE ENHANCEMENT**	20
	6.1 CONCLUTION	22
	6.2 FUTURE ENHANCEMENT	22
7	**APPENDIX**	23
8	**RESULTS AND DESCRIPTION**	18

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
4.1	USE CASE DIAGRAM	12
4.2	SEQUENCE DIAGRAM	13
4.3	ACTIVITY DIAGRAM	14
6.1	UESR INTERFRACE	18
6.2	API	19

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
1	TABLE OF CONTENTS I	17
2	TABLE OF CONTENTS II	17

CHAPTER 1
INTRODUCTION

In today's dynamic and data-rich business environment, the process of sales price optimization has evolved from being a traditional, often subjective endeavour to a cutting-edge, data-driven discipline. The integration of machine learning, a subset of artificial intelligence (AI), into pricing strategies has opened up new avenues for businesses to maximize profitability, maintain competitiveness, and enhance customer satisfaction. This introduction provides an overview of the transformation brought about by sales price optimization using machine learning and highlights its significance in contemporary business operations.

Historically, companies have relied on traditional pricing strategies that are grounded in historical data, market surveys, and expert judgment. While these methods have served businesses well for decades, they exhibit limitations in adapting to the rapidly changing marketplace of the digital age. As markets become more complex, influenced by factors such as globalization, technological advancements, and shifts in consumer behaviour, businesses are compelled to adopt more sophisticated pricing techniques.

Machine learning, a subset of AI, has emerged as a disruptive force in the realm of pricing optimization. It is founded on the principle of utilizing advanced algorithms to analyse vast datasets, identifying intricate patterns, and leveraging these insights to make data-driven decisions. In the context of sales price optimization, machine learning equips businesses with the ability to navigate the complexities of the modern market, transcending the limitations of human cognition and intuition.

1.1 DATA-DRIVEN INSIGHTS:

Machine learning algorithms are capable of processing vast datasets to identify patterns and correlations that are often beyond human perception. These insights enable businesses to understand customer behaviour, market dynamics, and the impact of various factors on pricing. By recognizing seasonality, demand elasticity, and competitive positioning, companies can optimize prices to maximize revenue.

These insights offer a profound understanding of variables such as seasonality, demand elasticity, and competitive positioning. Armed with this information, companies can make informed pricing decisions that maximize revenue and maintain market relevance.

1.2 TRADITIONAL PRICING STRATEGIES:

Historically, companies have relied on traditional pricing strategies that are grounded in historical data, market surveys, and expert judgment. While these methods have served businesses well for decades, they exhibit limitations in adapting to the rapidly changing marketplace of the digital age. As markets become more complex, influenced by factors such as globalization, technological advancements, and shifts in consumer behaviour, businesses are compelled to adopt more sophisticated pricing techniques.

Traditional pricing strategies have long been the cornerstone of business decision-making, with companies relying on established principles and methods to set prices for their products and services. These strategies, although fundamental, have several characteristics and limitations that have prompted the exploration of alternative approaches, such as machine learning-based pricing, in today's data-driven business landscape.

1.3 COST-PLUS PRICING

One of the most straightforward traditional pricing strategies is cost-plus pricing, where a company adds a mark up to the production cost of an item to determine the selling price. While this approach ensures that costs are covered and profit is made, it often overlooks market dynamics and consumer behaviour, potentially leading to suboptimal pricing decisions.

1.4 COMPETITOR-BASED PRICING

This strategy involves setting prices in alignment with competitors' pricing. Companies monitor the prices of similar products or services and adjust their own accordingly. While this method can help maintain competitiveness, it may not fully capture the unique value proposition of the business's offerings.

1.5 VALUE-BASED PRICING

Value-based pricing is rooted in the idea that a product or service's price should be determined by the value it delivers to the customer. It considers factors such as perceived benefits, customer willingness to pay, and the problem-solving capabilities of the offering. This strategy is more customer-centric but can be challenging to implement without a deep understanding of customer preferences and segmentation.

1.6 PSYCHOLOGICAL PRICING

Psychological pricing strategies use pricing techniques that appeal to consumers' emotions and perceptions. For example, setting prices at $9.99 instead of $10 can create the illusion of a better deal. While these strategies can influence purchasing decisions, they may not optimize profitability.

1.7 STATIC PRICING

Traditional pricing often involves static, fixed prices that remain constant over extended periods. This approach does not adapt to changes in market conditions, demand fluctuations, or competitive moves. It lacks the agility required in dynamic and rapidly evolving markets.

1.8 EXPERT JUDGMENT

Many traditional pricing decisions rely on the expertise and experience of pricing professionals within the organization. This can be effective, but it may not fully leverage the wealth of data available in the digital age.

1.9 MARKET RESEARCH

Market research, including surveys and focus groups, is often used to gather insights into consumer preferences and price sensitivity. While valuable, this data may not provide real-time, granular insights required for agile pricing decisions.

Traditional pricing strategies are still used by many businesses and serve as a foundation for pricing discussions. However, they are increasingly being complemented or replaced by data-driven, machine learning-based pricing approaches. These newer methods leverage advanced analytics, predictive modelling, and real-time data to make pricing decisions that are more adaptive, personalized, and responsive to market conditions. As companies continue to grapple with the complexities of modern markets, the synergy between traditional and innovative pricing strategies will play a pivotal role in shaping the future of pricing decisions.

1.10 DYNAMIC PRICING

Machine learning enables dynamic pricing, where prices are adjusted in real-time to respond to changing market conditions. For instance, e-commerce platforms can adapt prices based on supply and demand fluctuations, competitor pricing, and customer demand, maximizing revenue in each transaction.

1.11 PRICE SENSITIVITY ANALYSIS

Machine learning models can perform price sensitivity analysis, helping businesses understand how price changes impact demand and profitability. This insight is invaluable in determining optimal pricing strategies to balance market share and revenue goals.

Price Sensitivity Analysis, also known as price elasticity analysis, is a crucial component of pricing strategy that aims to understand how changes in price affect consumer demand. It provides valuable insights into the relationship between pricing and sales, helping businesses make informed decisions about setting and adjusting their prices.

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 GENERAL

To use jupyter notebook, there are several requirements that need to be met:

- **Operating System**: jupyter supports development on Windows, macOS, and Linux operating systems.
- **Hardware:** The minimum hardware requirements for Flutter development are a 64-bit processor and 4 GB of RAM.
- **Development Environment**: jupyter requires the installation of a code editor, such as Visual Studio Code or Android Studio, along with the anaconda SDK and its dependencies.
- **Knowledge of python**: python uses the Dart programming language, so developers need to have a good understanding of Dart in order to effectively develop Flutter apps.

1. Equipment Requirements (Hardware Requirement)

2. Programming Requirements (Software Requirement)

2.1.1 HARDWARE REQUIREMENTS

The minimum hardware requirements for using Flutter are relatively modest. To develop Flutter applications, you will need a computer with a 64-bit processor and at least 4 GB of RAM. The Flutter development environment requires installation of the Flutter SDK and its dependencies, along with a code editor such as Visual Studio Code or Android Studio. While Flutter apps can be developed on Windows, macOS, and Linux operating systems, the hardware requirements for each platform may vary slightly. In general, a computer with a relatively recent processor, 8 GB of RAM, and a high-resolution display will provide a more optimal development experience. However, Flutter's efficient tooling means that even less powerful hardware can be used for development.

- PROCESSOR : INTEL CORE i3
- RAM : 4 GB HDD RAM
- HARD DISK : 250 GB

2.1.2 SOFTWARE REQUIREMENTS

To use Flutter, you will need to install a number of software components. First, you will need a code editor such as Visual Studio Code or Android Studio. Flutter also requires the installation of the Flutter SDK, which includes the Flutter framework, the Dart programming language, and a set of command-line tools. In addition, you will need to install the necessary platform-specific development tools for the operating system you are using, such as code for iOS development or Android Studio for Android development. Finally, you may also need to install additional tools or libraries depending on the specific requirements of your application. Overall, while there are several software components to install, the Flutter installation process is relatively straightforward.

- OPERATING SYSTEM : WINDOWS 10
- SOFTWARE : JUPYTER NOTEBOOK
- PROGRAMING LANGUAGE : PYTHON

CHAPTER 3

METHODOLOGY

Our research consists of six stages that is Beginning stage, comprises

1. Problem identification
2. Course selection and identification
3. Modelling recommender system
4. Designing
5. Analysing the system
6. Implementation.

3.1 PROBLEM IDENTIFICATION

Price analysis are becoming increasingly popular as people look for convenient and accessible ways to learn new skills and improve their knowledge. However, like any technology, self-learning apps are not without their challenges. Here are some of the most common problem areas associated with self-learning apps:

- **User Engagement**: One of the biggest challenges with self-learning apps is keeping users engaged and motivated to continue using the app. This requires careful consideration of the app's user interface, content, and learning pathways.
- **Content Quality**: The quality of the app's content is critical to its success. Low quality content can turn users off and undermine the app's credibility. Additionally, content needs to be regularly updated to ensure that it remains relevant and up-to-date.
- **Learning Outcomes**: Another challenge is ensuring that the app's learning outcomes are aligned with user needs and expectations. This requires careful consideration of the learning goals and objectives, as well as user feedback and performance data.

3.2 PRICE SELECTION AND IDENTIFICATION

Course selection is an essential feature of self-learning apps, as it enables users to find and choose the courses that are most relevant to their needs and interests. A well designed course selection process can help users to identify the courses that will provide the most value and ensure that they have a positive learning experience. The app should provide a range of courses that cover a variety of topics and skill levels. This ensures that users can find courses that meet their specific needs and interests.

The app should provide detailed information about each course, including the course description, learning objectives, course structure, and any prerequisites or recommended experience. This information can help users to determine whether a course is a good fit for their learning goals and skill level.

3.3 MODELING RECOMMENDATION SYSTEM

Within recommendation systems, there is a group of models called content-based recommendation, which is tries to find the similarity between the previous data and the whole existing data.

3.3.1 CONTENT BASED RECOMMENDATION

Content-based methods gives recommendations based on the similarity of two song contents or attributes while collaborative methods make a prediction on possible preferences using a matrix with ratings on different songs. Content-based recommendation algorithm has to perform the following two steps. First, extract features out of the content of the song descriptions to create an object representation. Second, define a similarity function among these object representations which mimics what human understands as an item-item similarity. Because we are working with text and words, Term Frequency Inverse Document Frequency (TF-IDF) can be used for this matching process.

3.4 DESIGNING

Designing a self-learning app requires careful consideration of several key factors to ensure a user-friendly, engaging, and effective learning experience. Here are some important

considerations for designing a self-learning app. User Experience (UX): The app's UX should be intuitive and user-friendly, with clear navigation and easy access to content. Designers should consider user personas and user testing to optimize the app's UX. Visual Design: The visual design of the app should be appealing and consistent, with a clear visual hierarchy and appropriate use of colour, typography, and imagery..

3.5 SYSTEM ANALYSIS

System analysis is a critical step in the development of a self-learning app, as it enables the development team to identify user requirements, define system functionality, and create a blueprint for the app's architecture. Here are some important considerations for system analysis in the development of a self-learning app:

- **User Requirements**: System analysis should begin by gathering user requirements, which can be obtained through surveys, focus groups, or user testing. This helps the development team to understand the needs and preferences of the target audience, which can inform the app's design and functionality.
- **Functional Requirements**: Based on the user requirements, the development team should define the functional requirements of the app, such as the features, content, and interactions required to meet user needs
- **Non-Functional Requirements**: In addition to functional requirements, system analysis should also consider non-functional requirements such as performance, security, and scalability.

3.6 IMPLEMENTATION

The implementation of a self-learning app created with Flutter and Dart involves several steps that must be executed to ensure the app is properly developed and functions as intended. Here are some important considerations for implementing a self-learning app development. Environment Setup: The development team should first set up a development environment, including installing the Flutter and Dart SDKs, an IDE (such as Android Studio or VS Code), and any required plugins or dependencies. App
Development: With the architecture design in place, the development team can begin building the app, coding the required features and functionality using Dart programming language.

CHAPTER 4

UML DIAGRAMS

Unified Modelling Language (UML) diagrams can be helpful in designing and documenting software systems, including self-learning apps created with Flutter and Dart. Here are some examples of UML diagrams that could be used for a self-learning app:

- **Use Case Diagram**: This diagram illustrates the various use cases of the app, including the different interactions that users can have with the app, such as browsing courses, taking quizzes, and tracking progress.
- **Class Diagram**: This diagram shows the classes and relationships between them in the app. It includes the data structures used in the app, such as user profiles, course information, and quiz questions.
- **Sequence Diagram**: This diagram shows the interactions between objects or components of the app over time. It can be used to illustrate the sequence of actions that occur when a user interacts with the app, such as signing up for a course, taking a quiz, or viewing progress diagram. A sequence diagram shows sequence of message exchanges between lifelines.
- **Component Diagram**: This diagram shows the components and dependencies of the app. It includes the different software components that make up the app, such as the user interface, database, and network components.

These UML diagrams can be useful tools for visualizing and communicating the design and architecture of a self-learning app created with Flutter and Dart, helping to ensure that the app meets the requirements and functions as intended.

4.1 USE-CASE DIAGRAM

A use case diagram is a graphical representation of the various ways in which users can interact with a software system. It shows the different use cases of the system, the actors

involved in each use case, and the relationships between the actors and the system. In the context of a self-learning app created with Flutter and Dart, the use case diagram could show use cases such as browsing courses, taking quizzes, and tracking progress. This diagram helps to ensure that the app meets the requirements of its users and functions as intended.

The scope of our system.

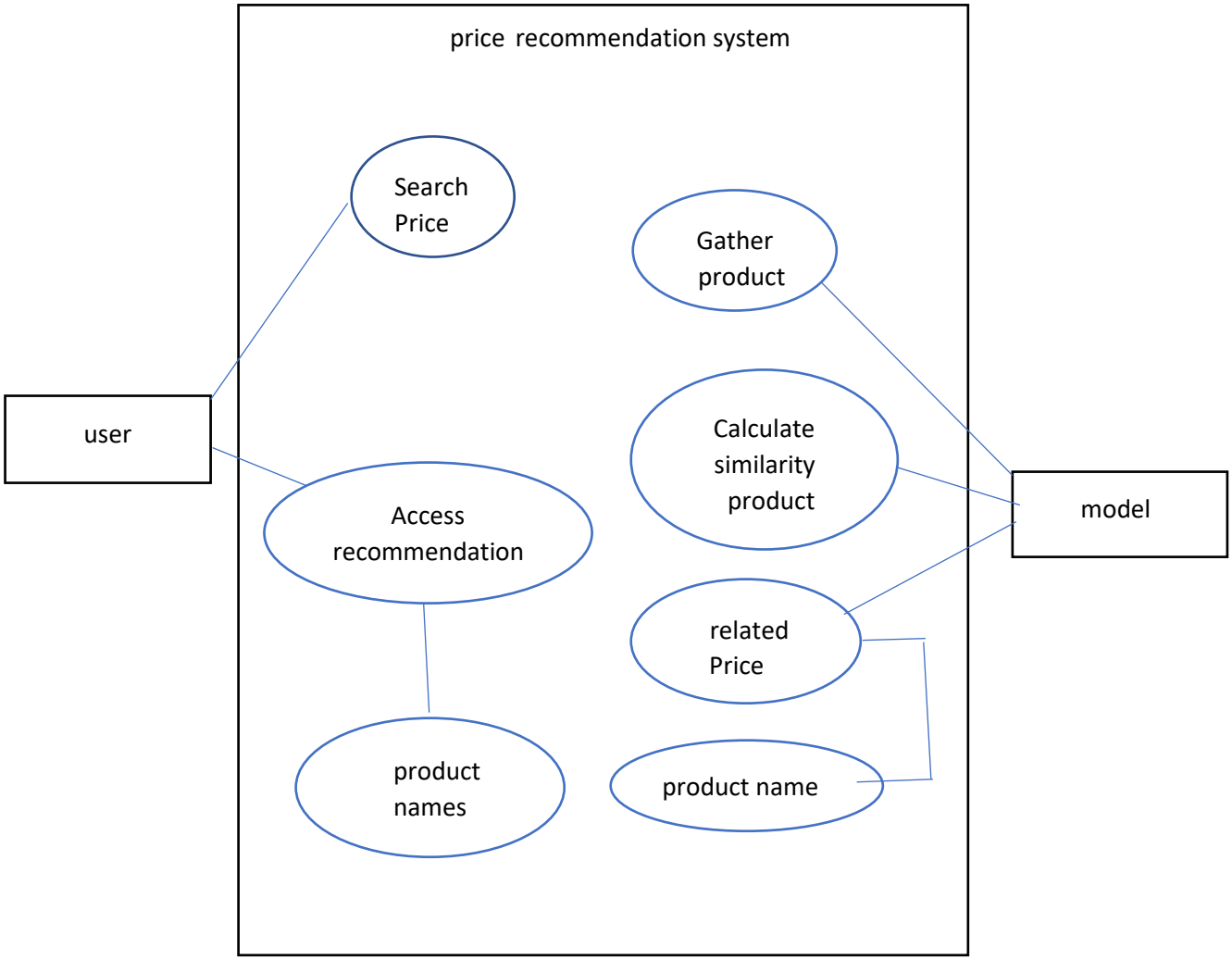


Fig **4.1** Use Case Diagram

4.2 SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram that shows the interactions between objects or components in a software system over time. It can be used to illustrate the sequence of actions that occur when a user interacts with the self-learning app, such as

signing up for a course, taking a quiz, or viewing progress. This diagram helps to ensure that the app's functions are well-defined, and that the various components of the system work together correctly.

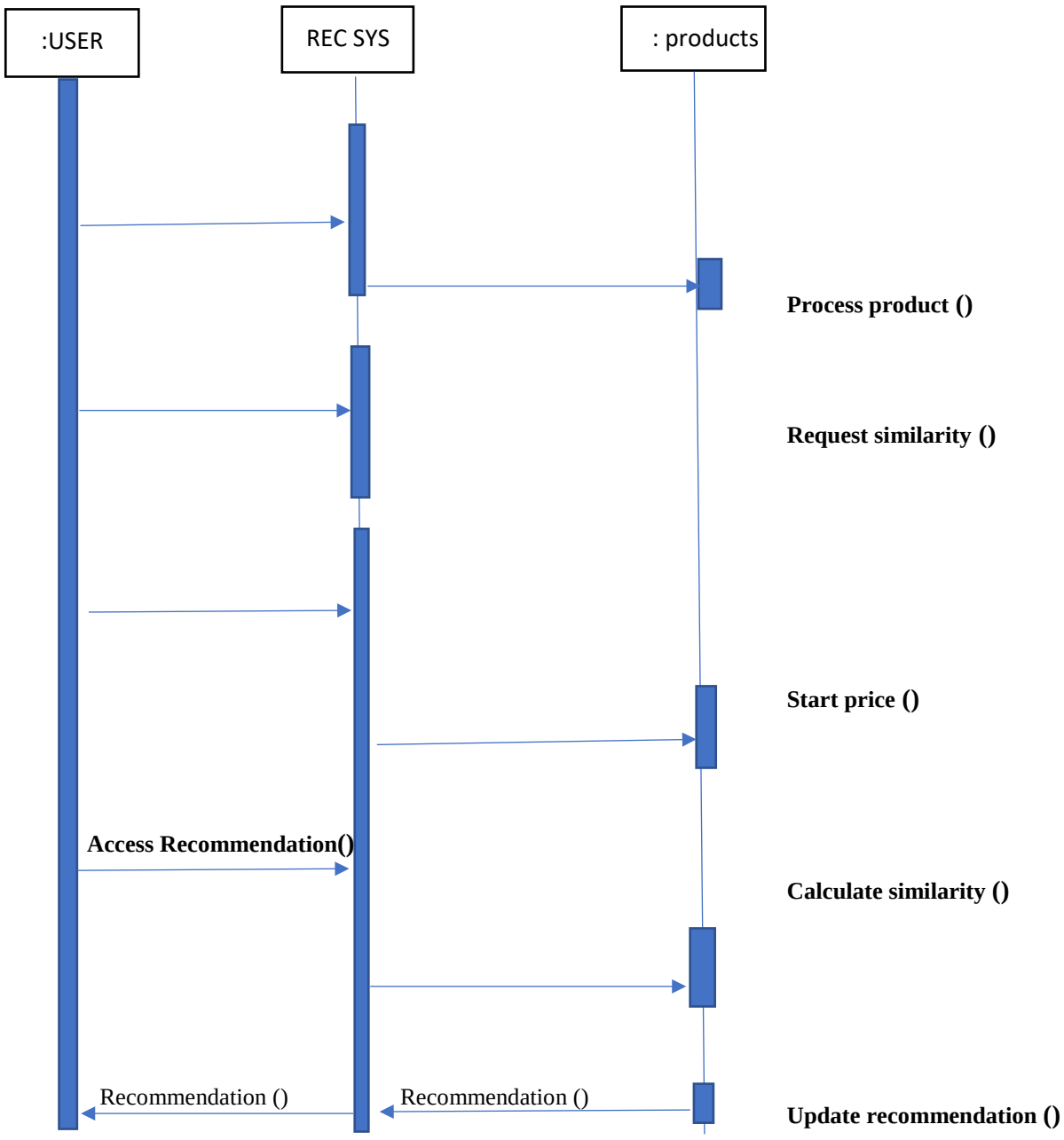


Fig **4.2** Sequence diagram for content based recommendation

4.3 ACTIVITY DIAGRAM

An activity diagram is a type of behaviour diagram that shows the flow of activities or processes within a software system. It can be used to visualize the steps involved in completing a task or process in the app, such as enrolling in a course or completing a quiz. This diagram helps to ensure that the self-learning app is easy to use and that the user's actions are well-defined and straightforward. This diagram illustrates the flow of activities or processes within the app. It can be used to visualize the steps involved in completing a task or process in the app, such as enrolling in a course or completing a quiz.

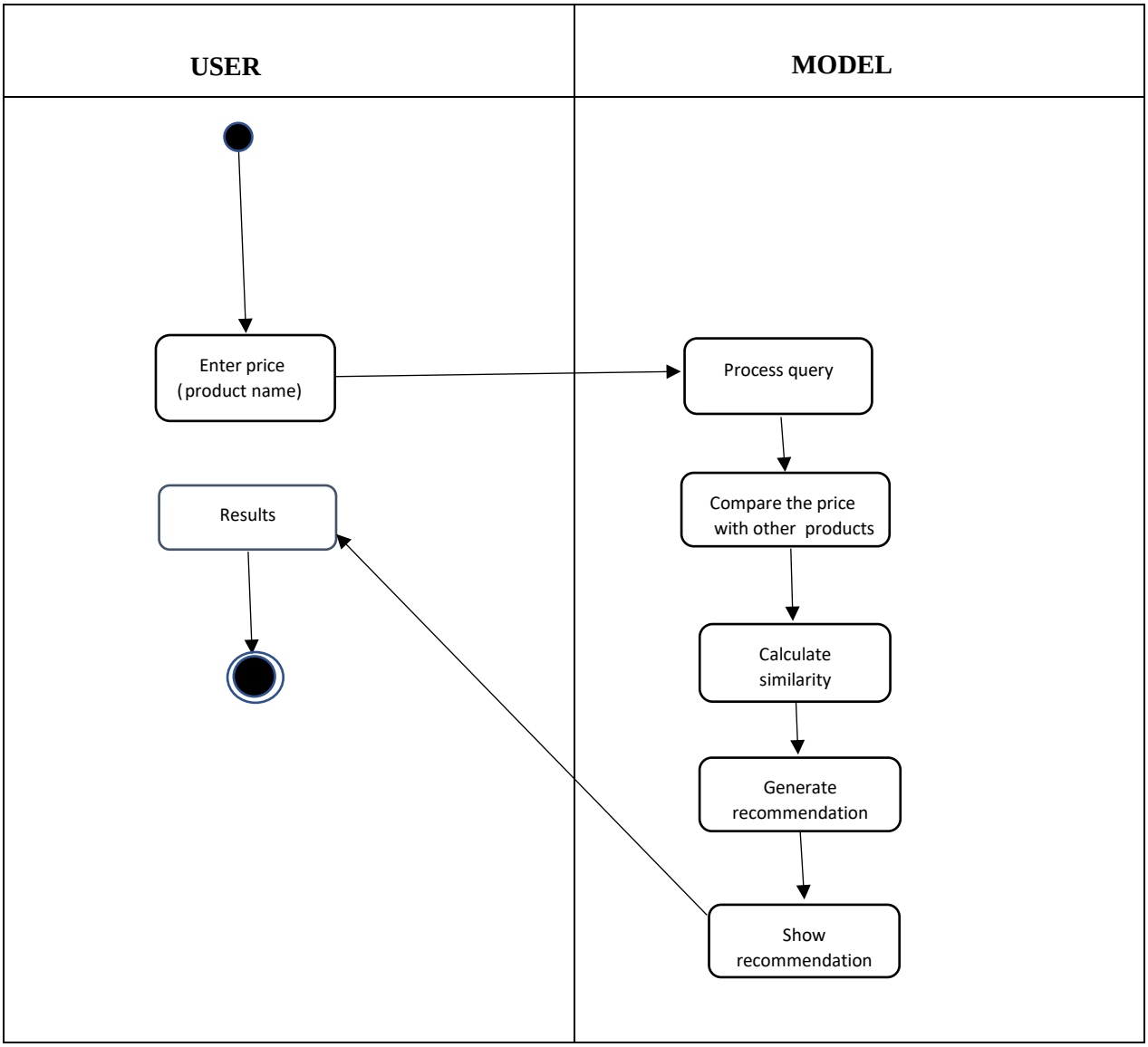


Fig **4.3** Activity diagram

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 DECISION TREE ALGORITHM

Decision tree algorithms are algorithms that are designed to process data in less than linear time, often achieved by processing only a subset of the input data. In the context of Flutter, sublinear algorithms can be used to optimize app performance, especially when dealing with large amounts of data.

The working of the sub-linear algorithm is explained in the below steps:

Step-1: Input data: The algorithm takes the input data, which could be in various forms such as arrays, lists, graphs, or any other data structures.

Step-2: Partitioning: The algorithm partitions the input data into smaller subsets or chunks, which can be processed independently. The partitions can be chosen based on various criteria such as size, structure, or relevance.

Step-3: Processing subsets: The algorithm processes the subsets in parallel, often using specialized data structures and techniques such as hashing, sampling, or randomization. By processing only a subset of the input data, the algorithm can achieve sublinear running time.

Step-4: Merging: After processing the subsets, the algorithm merges the results together to produce the final output. The merging can be done by various techniques such as hashing, sorting, or indexing.

Step-5: Analysis: Finally, the performance of the algorithm is analysed to ensure that it meets the required performance criteria and that any trade-offs made in the algorithm design are acceptable.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

5.2 OTHER ALGORITHM

Flutter provides a rich set of algorithms and data structures to help developers build efficient and high-performance mobile apps. Here are some of the commonly used algorithms in Flutter:

1) **Sorting algorithms**: Flutter includes several sorting algorithms such as quicksort, merge sort, and heapsort, which are used to sort data in various contexts such as search results, user lists, and data tables.

2) **Graph algorithms**: Graph algorithms such as Dijkstra's algorithm and A* algorithm are used to solve problems related to shortest path routing and navigation.

3) **Search algorithms**: Flutter includes several search algorithms such as binary search and linear search, which are used to search for specific data elements in large datasets.

4) **Data compression algorithms**: Flutter includes several data compression algorithms such as g-zip and deflate, which are used to compress and decompress data to optimize storage and network usage.

5) **Encryption algorithms**: Flutter includes several encryption algorithms such as AES and RSA, which are used to protect data and provide secure communication channels.

6) These algorithms are used in various aspects of app development such as data processing, user interface, and networking. Flutter also provides several libraries and packages that include algorithms to help developers build their apps more efficiently and effectively.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

In conclusion, the self-learning app created using Flutter and Dart programming language is an innovative solution to modern-day learning challenges. The app is designed to provide personalized learning experiences and improve the learning outcomes of students. It offers features such as a recommendation system, course selection, progress tracking, and interactive learning tools to enhance the learning experience. Flutter provides a rich set of tools and features for creating high-performance and visually appealing mobile applications. It is an open-source framework that allows developers to build cross-platform apps with ease. The use of Dart programming language enables fast and efficient app development, making the app responsive and interactive.

6.2 FUTURE ENHANCEMENT

In the future, we would like to try the following things:

1. AI price analysis bot.
2. Machine learning for product attachment.

CHAPTER 7

APPENDIX

SAMPLE CODING

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
import os
import math
import warnings

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error, r2_score

for dirname, _, filenames in os.walk("retail_price.csv"):
    for filename in filenames:
        print(os.path.join(dirname, filename))

warnings.filterwarnings('ignore')
pio.templates.default = 'plotly_white'
pd.set_option("display.max_columns", None)

data = pd.read_csv('retail_price.csv')
display(data.head(20), data.describe(), data.shape)

data.dtypes


# No duplicated rows. Ok
print("Duplicated: ", data.duplicated().sum())

# Missing values. Ok
print("Missing values: ", data.isna().sum().sum())
fig = px.histogram(data, x='total_price', nbins=10, \
                   title='Distribution of Total Price')
fig.show()
```

```
sns.displot(data['unit_price'])
plt.title("Unit Price Distribution")
plt.show()

correlation_matrix = data.corr()
correlation_matrix.style.background_gradient(cmap='viridis')

data2 = data.select_dtypes(['int', 'float'])

# Transform to log scale the target (y)
y = np.log(data2['total_price'])
X = data2.drop('total_price', axis=1)

sns.displot(data=y)
plt.show()

def results_regression(y_test_ ,y_pred_):
    mse = mean_squared_error(y_test_ ,y_pred_)
    print(f"mse: {mse}")

    rmse = math.sqrt(mse)
    print(f"rmse: {rmse}")

    mae = mean_absolute_error(y_test_ ,y_pred_)
    print(f"mae: {mae}")

    mape = mean_absolute_percentage_error(y_test_ ,y_pred_)
    print(f"mape: {mape}")

    r2 = r2_score(y_test_ ,y_pred_)
    print(f"r2_score {r2}")

def compute_decision_tree_regression(X_, y_):
    X_train, X_test, y_train, y_test = train_test_split(X_, y_, test_size=0.3, random_state=42)

    # Train a linear regression model
    model = DecisionTreeRegressor()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    results_regression(y_test, y_pred)
    return y_pred, y_test
```

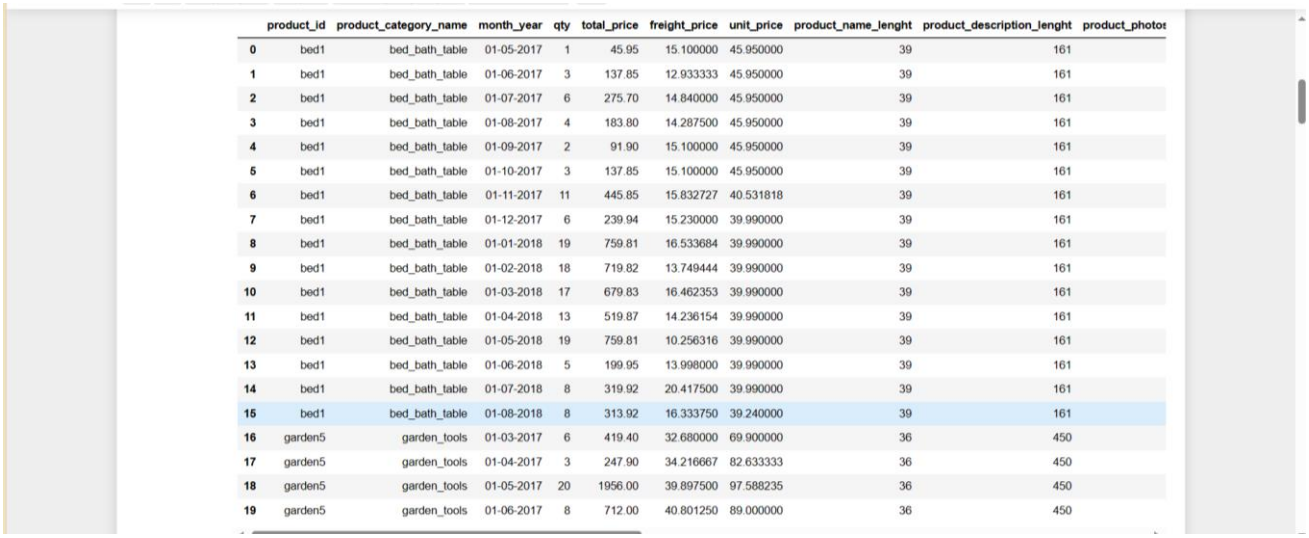
```
y_pred, y_test = compute_decision_tree_regression(X, y)
fig = go.Figure()
_title='Predicted vs. Actual Retail Price'

fig.add_trace(go.Scatter(x=y, y=y_pred, mode='markers',
                         marker=dict(color='blue'),
                         name=_title))

fig.add_trace(go.Scatter(x=[min(y_test), max(y_test)],
                         y=[min(y_test), max(y_test)],
                         mode='lines',
                         marker=dict(color='red'),
                         name='Ideal Prediction'))

fig.update_layout(
   title=_title,
   xaxis_title='Actual Retail Price',
   yaxis_title='Predicted Retail Price'
)
fig.show()
```

CHAPTER 8

RESULTS AND DESCRIPTION

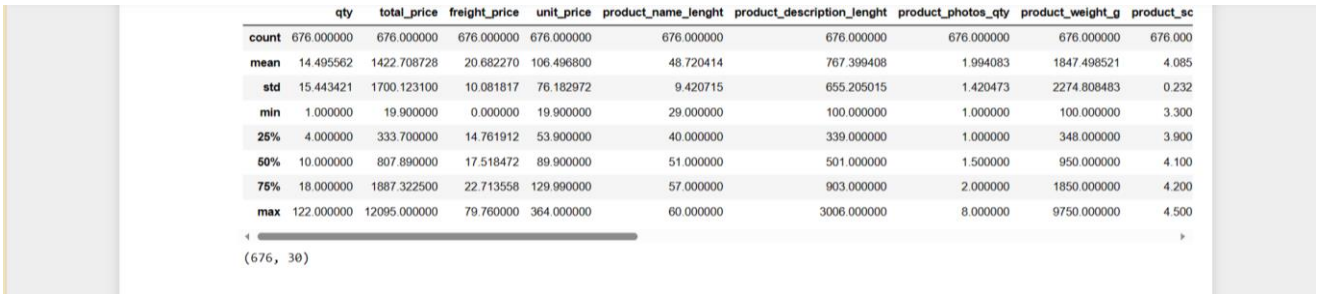
6.1 RESULTS

The algorithm partitions the input data into smaller subsets or chunks, which can be processed independently. The partitions can be chosen based on various criteria such as size, structure, or relevance.



	product_id	product_category_name	month_year	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos
0	bed1	bed_bath_table	01-05-2017	1	45.95	15.100000	45.950000	39	161	
1	bed1	bed_bath_table	01-06-2017	3	137.85	12.933333	45.950000	39	161	
2	bed1	bed_bath_table	01-07-2017	6	275.70	14.840000	45.950000	39	161	
3	bed1	bed_bath_table	01-08-2017	4	183.80	14.287500	45.950000	39	161	
4	bed1	bed_bath_table	01-09-2017	2	91.90	15.100000	45.950000	39	161	
5	bed1	bed_bath_table	01-10-2017	3	137.85	15.100000	45.950000	39	161	
6	bed1	bed_bath_table	01-11-2017	11	445.85	15.832727	40.531818	39	161	
7	bed1	bed_bath_table	01-12-2017	6	239.94	15.230000	39.990000	39	161	
8	bed1	bed_bath_table	01-01-2018	19	759.81	16.533684	39.990000	39	161	
9	bed1	bed_bath_table	01-02-2018	18	719.82	13.749444	39.990000	39	161	
10	bed1	bed_bath_table	01-03-2018	17	679.83	16.462353	39.990000	39	161	
11	bed1	bed_bath_table	01-04-2018	13	519.87	14.236154	39.990000	39	161	
12	bed1	bed_bath_table	01-05-2018	19	759.81	10.256316	39.990000	39	161	
13	bed1	bed_bath_table	01-06-2018	5	199.95	13.998000	39.990000	39	161	
14	bed1	bed_bath_table	01-07-2018	8	319.92	20.417500	39.990000	39	161	
15	bed1	bed_bath_table	01-08-2018	8	313.92	16.333750	39.240000	39	161	
16	garden5	garden_tools	01-03-2017	6	419.40	32.680000	69.900000	36	450	
17	garden5	garden_tools	01-04-2017	3	247.90	34.216667	82.633333	36	450	
18	garden5	garden_tools	01-05-2017	20	1956.00	39.897500	97.588235	36	450	
19	garden5	garden_tools	01-06-2017	8	712.00	40.801250	89.000000	36	450	

FIG.NO : 6.1 DATASET



	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos_qty	product_weight_g	product_sc
count	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000000	676.000
mean	14.495562	1422.708728	20.682270	106.496800	48.720414	767.399408	1.994083	1847.498521	4.085
std	15.443421	1700.123100	10.081817	76.182972	9.420715	655.205015	1.420473	2274.808483	0.232
min	1.000000	19.900000	0.000000	19.900000	29.000000	100.000000	1.000000	100.000000	3.300
25%	4.000000	333.700000	14.761912	53.900000	40.000000	339.000000	1.000000	348.000000	3.900
50%	10.000000	807.890000	17.518472	89.900000	51.000000	501.000000	1.500000	950.000000	4.100
75%	18.000000	1887.322500	22.713558	129.990000	57.000000	903.000000	2.000000	1850.000000	4.200
max	122.000000	12095.000000	79.760000	364.000000	60.000000	3006.000000	8.000000	9750.000000	4.500

(676, 30)

FIG.NO : 6.2 DATASET

```
In [24]: sns.displot(data['unit_price'])
         plt.title("Unit Price Distribution")
         plt.show()
```

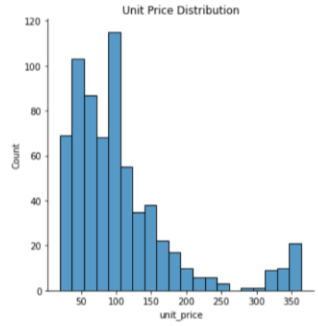


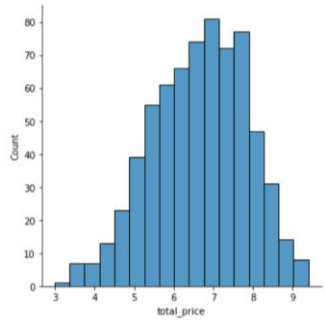
FIG.NO : 6.3 GRAPH

Out[26]:

	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos_qty	product_weigh
qty	1.000000	0.749605	-0.135521	-0.103432	0.079973	-0.022749	0.128515	-0.034
total_price	0.749605	1.000000	0.025848	0.409001	-0.002594	0.175376	0.157945	0.060
freight_price	-0.135521	0.025848	1.000000	0.203659	0.013398	0.423219	-0.200990	0.670
unit_price	-0.103432	0.409001	0.203659	1.000000	-0.170613	0.280176	0.076990	0.112
product_name_lenght	0.079973	-0.002594	0.013398	-0.170613	1.000000	0.124510	0.131951	-0.044
product_description_lenght	-0.022749	0.175376	0.423219	0.280176	0.124510	1.000000	0.060124	0.386
product_photos_qty	0.128515	0.157945	-0.200990	0.076990	0.131951	0.060124	1.000000	-0.129
product_weight_g	-0.034301	0.060092	0.670689	0.112958	-0.044050	0.386973	-0.129291	1.000
product_score	-0.004028	0.036119	0.199468	0.042162	0.163520	0.187544	0.048286	0.178
customers	0.441547	0.386389	0.088261	0.043391	0.082239	0.067497	-0.022536	0.053
weekday	0.030918	0.018798	-0.016132	-0.011949	0.023797	-0.019320	0.022311	-0.029
weekend	-0.075118	-0.053788	0.030275	-0.000042	-0.018183	-0.012465	0.008326	0.011
holiday	0.211610	0.136558	-0.081518	0.012573	-0.014317	0.018342	-0.008898	-0.003
month	-0.005129	-0.029918	-0.028336	-0.004249	-0.004250	-0.029561	0.041728	-0.011
year	0.058562	0.082140	0.076595	-0.068072	-0.035479	0.030949	0.033602	-0.108
s	0.411001	0.334500	-0.109359	-0.016552	-0.080830	0.001335	0.083799	-0.050
volume	0.049827	-0.088726	0.122097	-0.197233	0.329476	-0.141580	-0.153538	0.251
comp_1	-0.033570	0.144426	-0.013969	0.317113	-0.344125	-0.004371	-0.045398	0.064
ps1	-0.047883	0.058941	-0.053927	0.197425	0.019053	0.188601	-0.005677	-0.241
fp1	-0.053477	-0.006729	0.306479	-0.004518	-0.079388	0.031771	-0.194512	0.138
comp_2	-0.027044	0.203050	-0.084208	0.466459	-0.240613	0.015030	-0.133835	-0.017

FIG.NO : 6.4 DATA

In [29]:
```
sns.displot(data=y)
plt.show()
```



In [30]:
```
def results_regression(y_test_ ,y_pred_):
    mse = mean_squared_error(y_test_ ,y_pred_)
    print(f"mse: {mse}")

    rmse = math.sqrt(mse)
    print(f"rmse: {rmse}")
```

FIG.NO : 6.5 GRAPH

In [30]:
```
def results_regression(y_test_ ,y_pred_):
    mse = mean_squared_error(y_test_ ,y_pred_)
    print(f"mse: {mse}")

    rmse = math.sqrt(mse)
    print(f"rmse: {rmse}")

    mae = mean_absolute_error(y_test_ ,y_pred_)
    print(f"mae: {mae}")

    mape = mean_absolute_percentage_error(y_test_ ,y_pred_)
    print(f"mape: {mape}")

    r2 = r2_score(y_test_ ,y_pred_)
    print(f"r2_score {r2}")

def compute_decision_tree_regression(X_, y_):
    X_train, X_test, y_train, y_test = train_test_split(X_, y_, test_size=0.3, random_state=42)

    # Train a linear regression model
    model = DecisionTreeRegressor()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    results_regression(y_test, y_pred)
    return y_pred, y_test

y_pred, y_test = compute_decision_tree_regression(X, y)
```

FIG .NO 6.6 IMPLEMENTATION

```
In [9]: fig = px.histogram(data, x='total_price', nbins=10, \
                           title='Distribution of Total Price')
        fig.show()
```

FIG.NO: 6.7 GRPAH

```
In [11]: correlation_matrix = data.corr()
         correlation_matrix.style.background_gradient(cmap='viridis')
```

Out[11]:

	qty	total_price	freight_price	unit_price	product_name_lenght	product_description_lenght	product_photos_qty	product_weigh
qty	1.000000	0.749605	-0.135521	-0.103432	0.079973	-0.022749	0.128515	-0.034
total_price	0.749605	1.000000	0.025848	0.409001	-0.002594	0.175376	0.157945	0.060
freight_price	-0.135521	0.025848	1.000000	0.203659	0.013398	0.423219	-0.200990	0.670
unit_price	-0.103432	0.409001	0.203659	1.000000	-0.170613	0.280176	0.076990	0.112
product_name_lenght	0.079973	-0.002594	0.013398	-0.170613	1.000000	0.124510	0.131951	-0.044
product_description_lenght	-0.022749	0.175376	0.423219	0.280176	0.124510	1.000000	0.060124	0.386
product_photos_qty	0.128515	0.157945	-0.200990	0.076990	0.131951	0.060124	1.000000	-0.129
product_weight_g	-0.034301	0.060092	0.670689	0.112958	-0.044050	0.386973	-0.129291	1.000
product_score	-0.004028	0.036119	0.199468	0.042162	0.163520	0.187544	0.048286	0.178
customers	0.441547	0.386389	0.088261	0.043391	0.082239	0.067497	-0.022536	0.053
weekday	0.030918	0.018798	-0.016132	-0.011949	0.023797	-0.019320	0.022311	-0.029
weekend	-0.075118	-0.053788	0.030275	-0.000042	-0.018183	-0.012465	0.008326	0.011
holiday	0.211610	0.136558	-0.081518	0.012573	-0.014317	0.018342	-0.008898	-0.003
month	-0.005129	-0.029918	-0.028336	-0.004249	-0.004250	-0.029561	0.041728	-0.011
year	0.058562	0.082140	0.076595	-0.068072	-0.035479	0.030949	0.033602	-0.108
s	0.411001	0.334500	-0.109359	-0.016552	-0.080830	0.001335	0.083799	-0.050
volume	0.049827	-0.088726	0.122097	-0.197233	0.329476	-0.141580	-0.153538	0.251
comp_1	-0.033570	0.144426	-0.013969	0.317113	-0.344125	-0.004371	-0.045398	0.064

FIG.NO : 6.8 CORRELATION GRAPH

FIG.NO : 6.9 PRICE GRAPH

REFERENCE

[1] N. H. M. Shamsuddin, N. A. Ali, and R. Alwee, “An overview on crime predictionmethods,” 6th International Student Project Conference ICT, (ICT-ISPC),2020, pp. 1–5, 2020, doi: 10.1109/ICT-ISPC.2017.8075335.

[2] S. Kim, P. Joshi, P. S. Kalsi and P. Taheri, "Crime Analysis Through Machine Learning," 2019 IEEE 9th Annual Information Technology, Electronics and MobileCommunication Conference (IEMCON), Vancouver, BC, 2018, pp. 415-420, doi:10.1109/IEMCON.2018.8614828.

[3] N. Shama, “A Machine Learning Approach to Predict Crime Using Time and Location Data,” pp. 1–52, 2017.

[4] V. Jain, Y. Sharma, A. Bhatia, and V. Arora, “Crime Prediction using K-means Algorithm,” Global Research and Development Journals for Engineering, volume. 2, issue. 5, pp. 206–209, 2017,

[5] E. Ahishakiye, D. Taremwa, E. O. Omulo, and I. Niyonzima. (2016). Crime Prediction Using Decision Tree (J48) Classification Algorithm. International Journal of Computer and Information Technology (ISSN: 2279 – 0764).

[6] C. Catlett, E. Cesario, D. Talia, and A. Vinci, “Spatio-temporal crime predictions in smart cities: A data-driven approach and experiments,” Pervasive and Mobile Computing., vol. 53, pp. 62–74.