WEB-SCRAPPING PROJECT

E-COMMERCE ANALYSIS

WASHING MACHINE PRICE PREDICTION

NAME	PAVITHRA M
TITLE	E-COMMERCE ANALYSIS
BATCH	DA/DS (OFFLINE)
DATE	12/12/2025

TABLE OF CONTENT

S.NO	CONTENTS
1	INTRODUCTION
2	AIM
3	PROBLEM STATEMENT
4	PROJECT WORKFLOW
5	DATA COLLECTION
6	FEATURE ENGINEERING
7	DATA UNDERSTANDING
8	DATA CLEANING
9	EXPLORATORY DATA ANALYSIS
10	DATA STORAGE
11	DATA PREPROCESSING
12	UNSUPERVISED MACHINE LEARNING
13	SUPERVISED MACHINE LEARNING
14	HYPERPARAMETER TUNING
15	MODEL DEPLOYMENT
16	INSIGHTS
17	RECOMMENDATIONS

1. Introduction

The washing machine price prediction project is designed to analyze various washing machine models available in the market, categorize them into different price segments using clustering, and predict the price segment for a given machine based on its features. The project combines **data collection, preprocessing, feature engineering, machine learning modeling, and deployment**, providing a comprehensive approach to data-driven insights for business and consumers.

Key points:

- Helps understand market pricing trends.
- Provides insights into factors affecting washing machine prices.
- Uses advanced ML models for accurate predictions.
- Deployed as an interactive application for real-time use.

2. Aim

The aim of this project is to build a data-driven solution for analyzing and predicting washing machine segments based on their features. The project focuses on:

- Collecting and preprocessing real-world data from e-commerce platforms.
- Understanding patterns in product features, pricing, and types.
- Applying unsupervised learning to identify natural clusters in the data.
- Using supervised machine learning models to predict clusters for new products.
- Deploying the best-performing model to a user-friendly Streamlit application for real-time predictions.
- Providing actionable insights for businesses to make informed pricing, inventory, and marketing decisions.

3. Problem Statement

The washing machine market is highly competitive, and businesses often struggle to make informed decisions regarding product segmentation, pricing, and inventory management. Challenges include:

- Difficulty in accurately categorizing products into economy, standard, or premium segments based on features and price.
- Limited data-driven insights to guide strategic decisions in pricing, marketing, and inventory planning.
- Inability to predict the appropriate segment for newly launched or unlisted washing machine models.
- Manual analysis of large datasets is time-consuming and prone to errors.
- Lack of real-time predictive tools for actionable business decision-making.

This project addresses these challenges by leveraging machine learning and data analysis to automate product segmentation, generate actionable insights, and enable businesses to make informed, data-driven decisions.

4. Project Workflow

Data Collection

- Scrape washing machine data from e-commerce platforms using Python.
- Extract key features such as Brand, Price, Load Type, Capacity, Energy Rating, and more.

Feature Engineering

- Derive new features to enhance model performance.
- Select the most relevant features for clustering and prediction.

Data Cleaning & Preprocessing

- Handle missing values and standardize feature formats.
- Encode categorical variables and scale numerical features for modeling.

Exploratory Data Analysis (EDA)

- Analyze feature distributions, correlations, and outliers.
- Identify patterns in pricing, brand preferences, and product types.

Data Storage

- Store cleaned data in a MySQL database for easy access and scalability.
- Implement SQL connectivity to push and pull data directly from Python.

Unsupervised Learning (Clustering)

- Apply clustering algorithms to segment washing machines into economy, standard, and premium clusters.

Supervised Learning (Prediction Models)

- Train models like Logistic Regression, Random Forest, KNN, SVM, XGBoost to predict cluster labels.
- Optimize models using hyperparameter tuning for better accuracy.

Model Evaluation

- Compare model performance using metrics such as accuracy, precision, and recall.
- Select the best-performing model for deployment.

Deployment

- Deploy the model using Streamlit to enable real-time cluster predictions.
- Build a user-friendly interface for stakeholders to interact with the model.

Insights & Recommendations

- Generate actionable insights from the data and model predictions.
- Provide recommendations for pricing, inventory, and marketing strategies.

5. Data Collection

The data for this project was collected using web scraping in Python, utilizing BeautifulSoup, Selenium, and Regex. BeautifulSoup was used to extract structured data from static web pages, while Selenium handled dynamic content that required interaction, such as scrolling or clicking. Regex was applied to clean and extract specific patterns from text, ensuring consistent and usable data. The collected information, including product names, prices, types, and specifications, was stored in a structured format, ready for analysis and machine learning tasks.

6. Feature Engineering

Feature engineering was performed using Python and regular expressions (Regex) to extract useful information from raw text data. Numerical values, units, and keywords were identified and converted into structured features. Unnecessary characters were removed, and data formats were standardized for consistency. Categorical attributes like brand and machine type were also extracted. These processed features were then used for analysis, clustering, and predictive modeling.

7. Data Understanding

Data understanding involved analyzing the dataset to explore its structure and content. The shape of the dataset was examined to understand the number of records and features. All columns were reviewed to identify their types and relevance.This process helped create a clean and reliable dataset for further analysis and modeling.

Total Rows : 984

Total Columns : 15

- Dropped Unwanted Features

```
df = df.drop(['Product Name', 'Rating', 'Description', 'Special_Features', 'Color'], axis=1)

df = df[['Brand',
         'Machine_Type',
         'Load_Type',
         'RPM',
         'Capacity_kg',
         'Price',
         'Discount',
         'Rating_Score',
         'Total_Ratings',
         'Total_Reviews']]

df.head()
```

	Brand	Machine_Type	Load_Type	RPM	Capacity_kg	Price	Discount	Rating_Score	Total_Ratings	Total_Reviews
0	IFB	NaN	NaN	NaN	8	33990	26.0	4.329	329780	250
1	IFB	NaN	NaN	NaN	6	24990	27.0	4.329	329780	250
2	realme	Semi Automatic	Top Load	NaN	7	7390	40.0	4.296	296786	315
3	realme	Semi Automatic	Top Load	NaN	10	9990	50.0	4.296	296786	315
4	realme	Semi Automatic	Top Load	NaN	7.5	7790	44.0	4.296	296786	315

- Checked Info and Columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 984 entries, 0 to 983
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Brand          984 non-null    object
 1   Machine_Type   581 non-null    object
 2   Load_Type      521 non-null    object
 3   RPM            25 non-null     object
 4   Capacity_kg    979 non-null    object
 5   Price          984 non-null    int64
 6   Discount       966 non-null    float64
 7   Rating_Score   870 non-null    float64
 8   Total_Ratings  984 non-null    int64
 9   Total_Reviews  984 non-null    int64
dtypes: float64(2), int64(3), object(5)
memory usage: 77.0+ KB
```

```
df.columns
```

```
Index(['Brand', 'Machine_Type', 'Load_Type', 'RPM', 'Capacity_kg', 'Price',
       'Discount', 'Rating_Score', 'Total_Ratings', 'Total_Reviews'],
      dtype='object')
```

8. Data Cleaning

Data cleaning involved preparing the dataset for accurate and reliable analysis. Null values were identified and handled, and duplicates were removed to ensure data quality. The

datatypes of all columns were checked and converted to appropriate types to ensure consistency and compatibility for analysis. Outliers were detected using boxplots and removed using the IQR method for features such as capacity, rating score, and total ratings. These steps resulted in a clean, well-structured, and reliable dataset ready for further analysis and modeling.

```
df.drop_duplicates(inplace=True)
```

```
df.dtypes
```

```
Brand            object
Machine_Type     object
Load_Type        object
RPM              object
Capacity_kg      object
Price             int64
Discount        float64
Rating_Score    float64
Total_Ratings     int64
Total_Reviews     int64
dtype: object
```

```
cat_columns = ['Brand', 'Machine_Type', 'Load_Type']
df[cat_columns] = df[cat_columns].astype('category')

df['RPM'] = pd.to_numeric(df['RPM'], errors='coerce')
df['Capacity_kg'] = pd.to_numeric(df['Capacity_kg'], errors='coerce')

df.dtypes
```

```
Brand           category
Machine_Type    category
Load_Type       category
RPM              float64
Capacity_kg      float64
Price              int64
Discount         float64
Rating_Score     float64
Total_Ratings      int64
Total_Reviews      int64
dtype: object
```

```
df.isnull().sum()
```

```
Brand             0
Machine_Type     85
Load_Type       112
RPM             559
Capacity_kg       5
Price             0
Discount         18
Rating_Score    113
Total_Ratings     0
Total_Reviews     0
dtype: int64
```

```
df['Load_Type'] = df['Load_Type'].str.title()

for col in ["Machine_Type", "Load_Type"]:
    df[col].fillna(df[col].mode()[0], inplace=True)

for col in ["Rating_Score", "Price"]:
    df[col].fillna(df[col].median(), inplace=True)

df["RPM"].replace(0, np.nan, inplace=True)
df["RPM"] = df.groupby(["Machine_Type", "Load_Type"])["RPM"].transform(lambda x: x.fillna(x.median()))
df["RPM"].fillna(df["RPM"].median(), inplace=True)

df["Capacity_kg"] = df.groupby("Machine_Type")["Capacity_kg"].transform(lambda x: x.fillna(x.median()))

df["Discount"].fillna(0, inplace=True)
```

```
df.isnull().sum().sum()
```

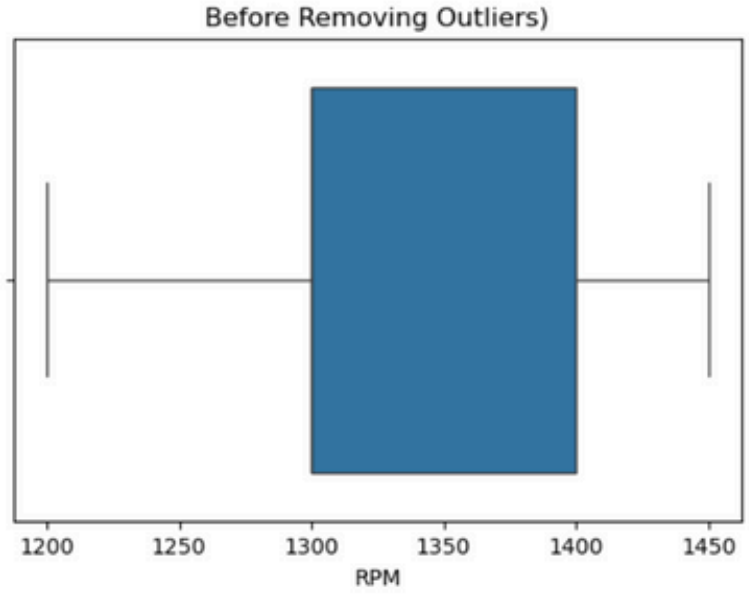
```
np.int64(0)
```

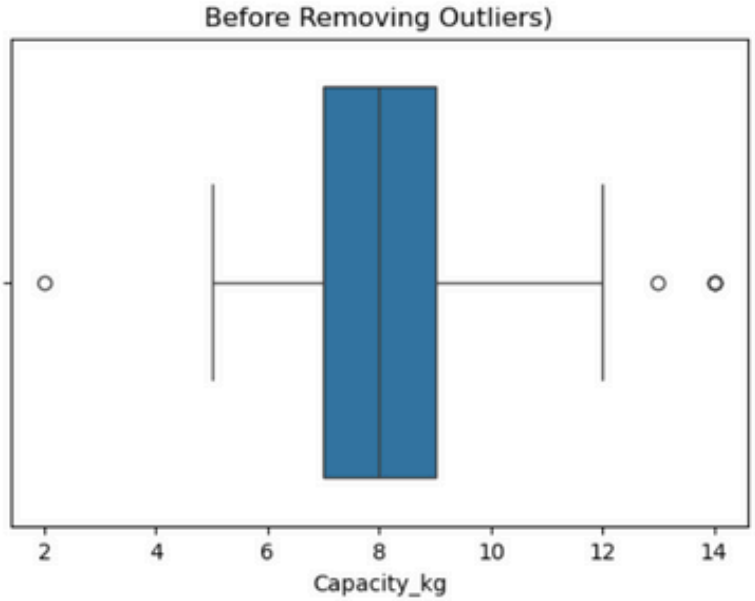
```
df.describe()
```

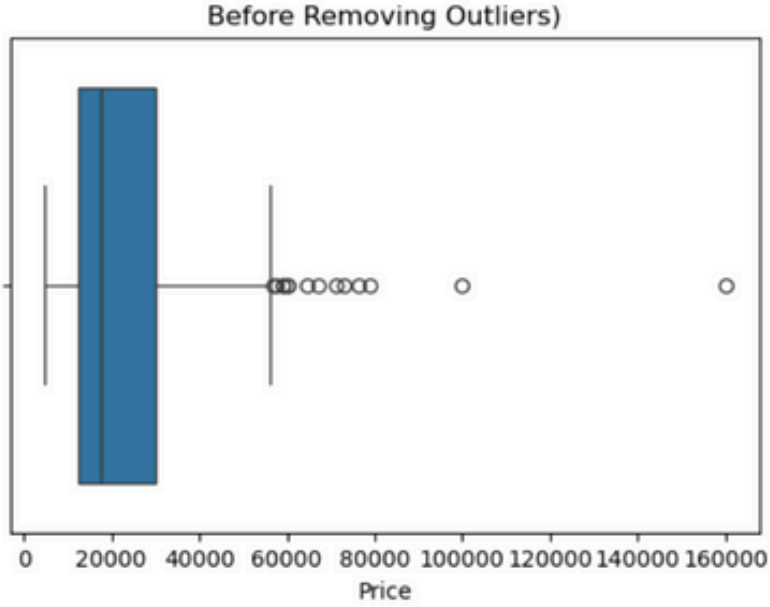
	RPM	Capacity_kg	Price	Discount	Rating_Score	Total_Ratings	Total_Reviews
count	582.000000	582.000000	582.000000	582.000000	582.000000	5.820000e+02	582.000000
mean	1336.855670	8.189175	22445.627148	28.969072	124.696403	9.239264e+04	176.201031
std	49.818014	1.548717	14465.505450	13.396473	700.969378	2.838387e+05	270.490253
min	1200.000000	2.000000	4690.000000	0.000000	2.911000	0.000000e+00	0.000000
25%	1300.000000	7.000000	12510.000000	21.000000	4.256000	7.300000e+01	0.000000
50%	1300.000000	8.000000	17490.000000	28.000000	4.325000	3.244500e+03	18.500000
75%	1400.000000	9.000000	29990.000000	38.000000	4.418800	4.251500e+04	255.250000
max	1450.000000	14.000000	159990.000000	64.000000	4970.000000	3.106829e+06	992.000000

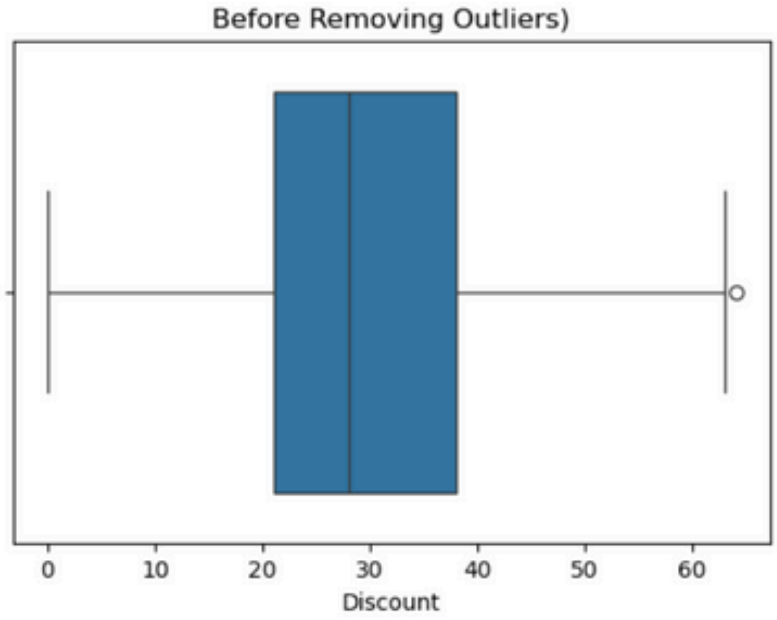
- Identifying outliers using Boxplot

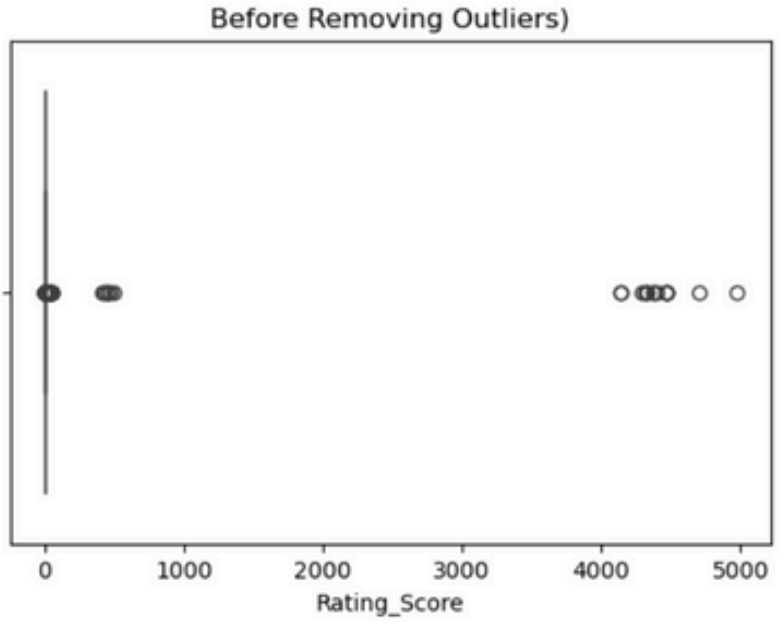
```
import matplotlib.pyplot as plt
import seaborn as sns

num_cols = ['RPM','Capacity_kg','Price','Discount','Rating_Score','Total_Ratings','Total_Reviews']
for col in num_cols:
    plt.figure(figsize=(6,4))
    sns.boxplot(x=df[col])
    plt.title("Before Removing Outliers)")
    plt.show()
```

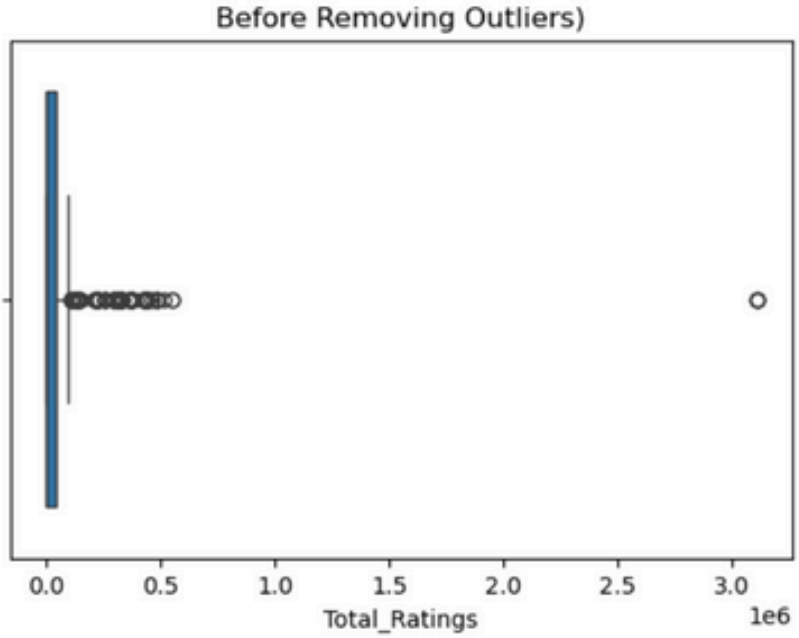

Before Removing Outliers)
1200
1250
1300
1350
1400
1450
RPM

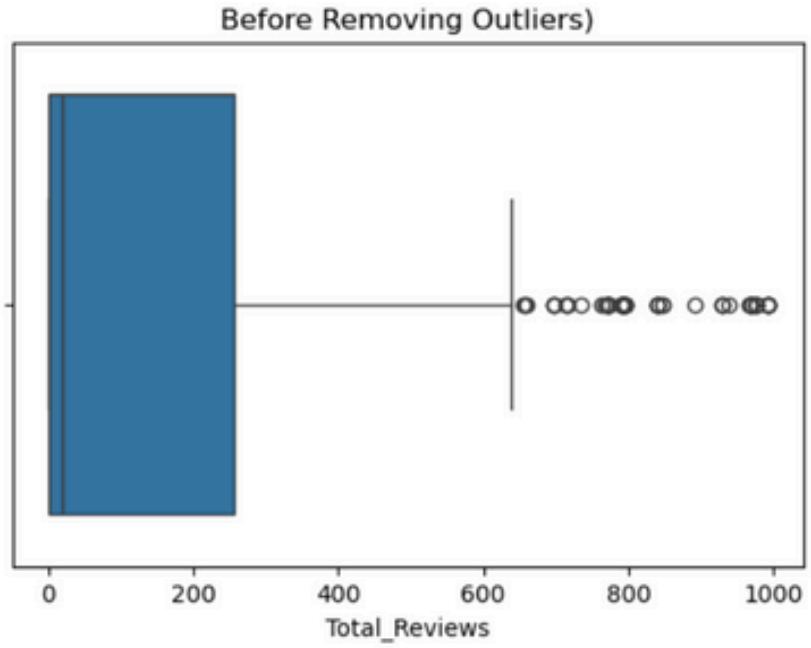

Before Removing Outliers)
2
4
6
8
10
12
14
Capacity_kg


Before Removing Outliers)
0
20000
40000
60000
80000
100000
120000
140000
160000
Price


Before Removing Outliers)
0
10
20
30
40
50
60
Discount


Before Removing Outliers)
0
1000
2000
3000
4000
5000
Rating_Score


Before Removing Outliers)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Total_Ratings
1e6

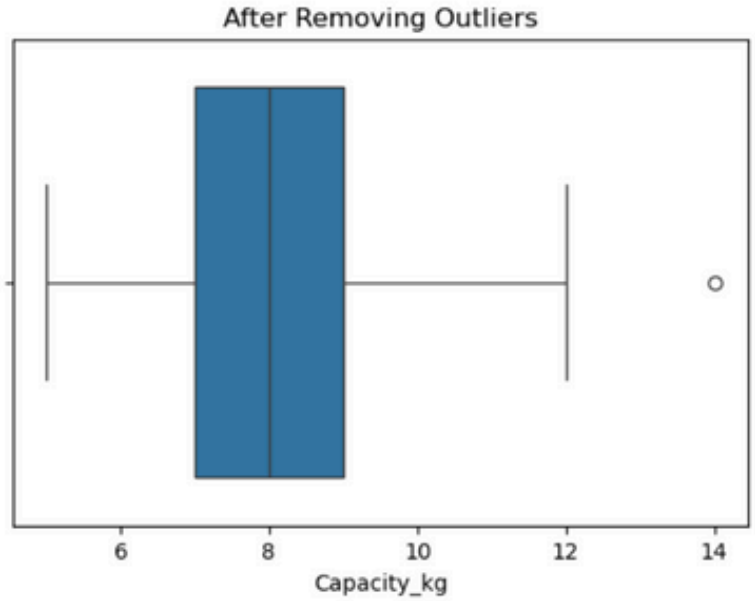


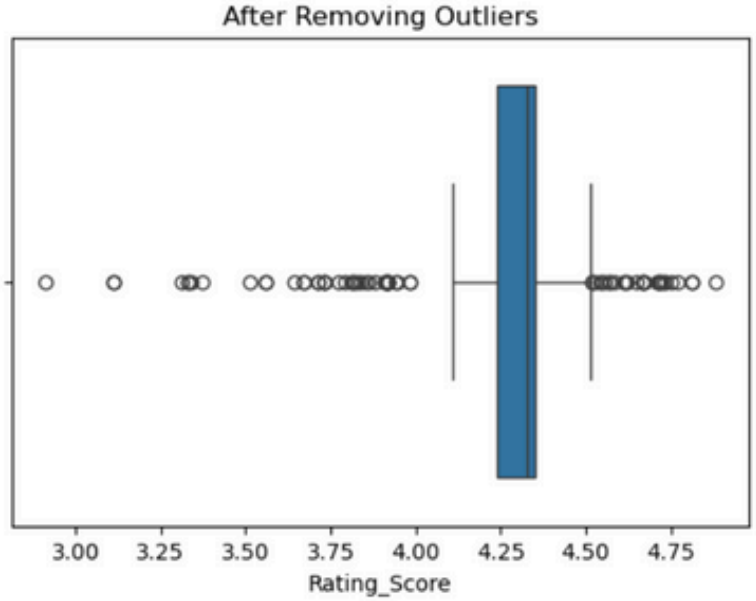
- Removed Outliers in Capacity_kg, Rating_Score and Total_Ratings using Inter-Quartile Method.

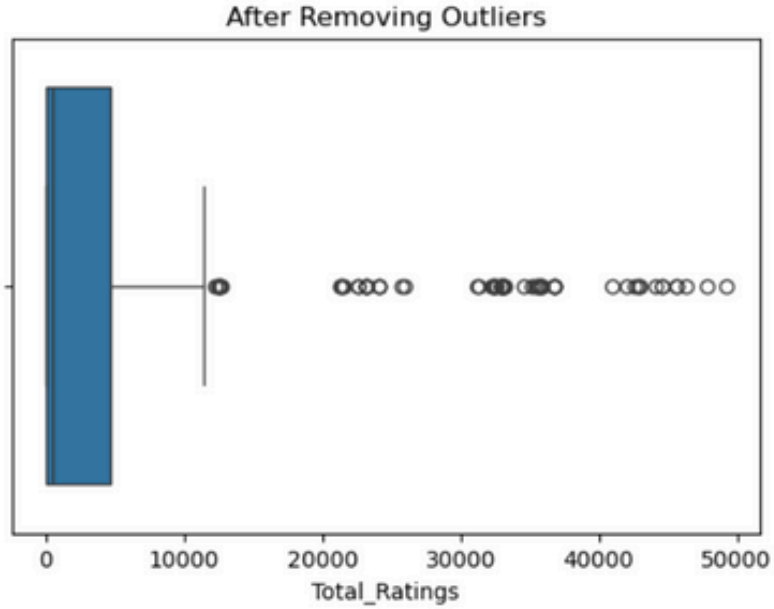
```python
# Removing outliers using Inter Quartile Method (IQR)
change_cols = ["Capacity_kg", "Rating_Score", "Total_Ratings"]

for col in change_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

     # Realistic limits
    if col == 'Capacity_kg':
        lower, upper = 5, 18

    if col == 'Rating_Score':
        lower, upper = 1, 5

    if col == 'Total_Ratings':
        lower, upper = 0, 50000

    df = df[(df[col] >= lower) & (df[col] <= upper)]

print("Final rows after outlier removal:", df.shape[0])
```

```
Final rows after outlier removal: 406
```

- After removing outliers :


After Removing Outliers
6
8
10
12
14
Capacity_kg


After Removing Outliers
3.00
3.25
3.50
3.75
4.00
4.25
4.50
4.75
Rating_Score


After Removing Outliers
0
10000
20000
30000
40000
50000
Total_Ratings

Insights

- Outliers in Rating_Score, Capacity_kg, and Total_Ratings were removed to get more realistic values.
- RPM, Price, Discount, and Total_Reviews kept their outliers because high or low values are common in premium or specialized models.
- This way, the data is cleaner for analysis while keeping meaningful extremes.

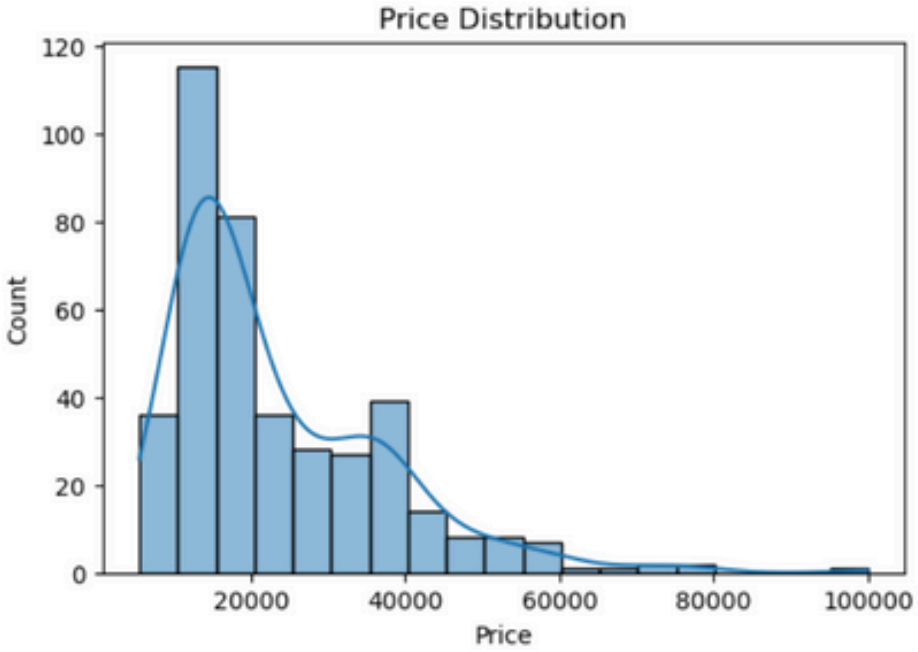
After Removing Outliers :

- Total Rows : 406
- Total Columns : 10

9. Exploratory Data Analysis

Univariate Analysis

```
# Price Range Distribution
plt.figure(figsize=(6,4))
sns.histplot(df['Price'], kde=True)
plt.title("Price Distribution")
plt.xlabel("Price")
plt.ylabel("Count")
plt.show()
```



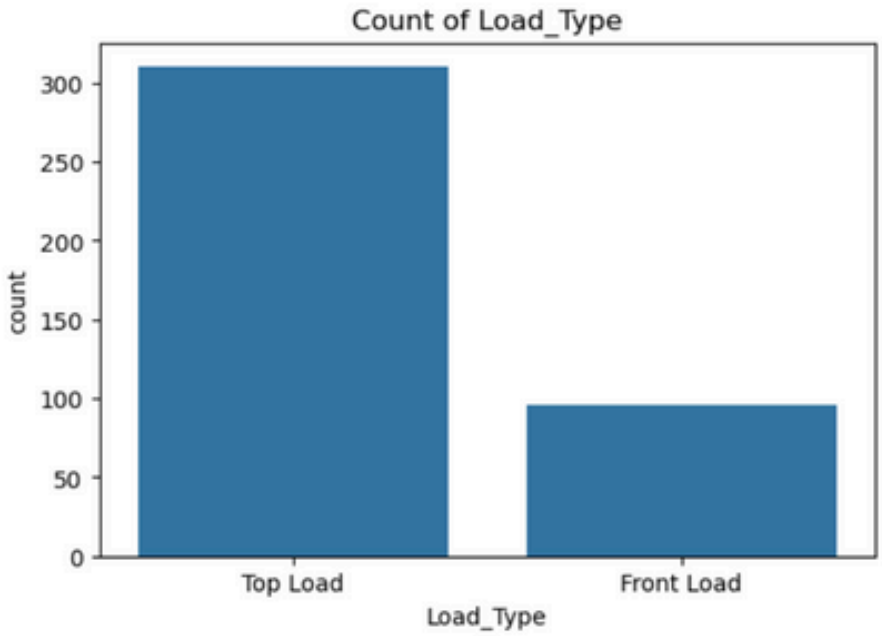
Insights

- Most washing machines are priced between ₹10,000 and ₹20,000, making this the most common range for budget-conscious buyers.

- Machines in the ₹20,000 to ₹40,000 range cater to mid-range buyers and are less common than economy models.
- High-end machines above ₹40,000 are relatively rare, targeting niche buyers seeking advanced features or luxury options.

```
# Distribution of Load Types
categorical_cols = df.select_dtypes(include='object').columns

for col in categorical_cols:
    print(f"\n--- {col} ---")
    print(df[col].value_counts())

    # Barplot
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df)
    plt.title(f'Count of {col}')
    plt.show()
```

```
--- Load_Type ---
Load_Type
Top Load      310
Front Load     96
Name: count, dtype: int64
```



Insights

- The dataset shows a clear dominance of top load washing machines, with 310 units compared to 96 front load machines.
- This indicates that top load models are more widely available or more commonly purchased in the market.
- Front load machines are less frequent, suggesting they are either a premium option or cater to a niche segment of buyers.

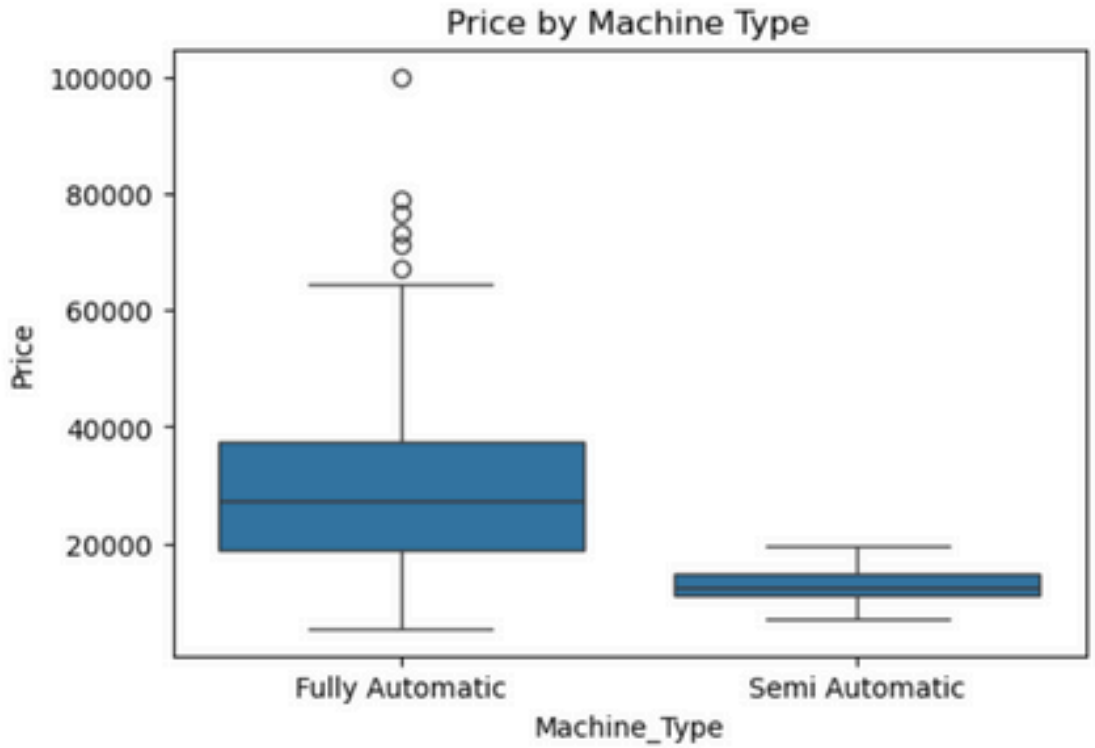
Bivariate Analysis

```
# Rating vs Price Analysis
plt.figure(figsize=(6,4))
plt.scatter(df['Rating_Score'], df['Price'])
plt.title('Rating vs Price')
plt.xlabel('Rating')
plt.ylabel('Price')
plt.show()
```

Insights

- Most washing machines are clustered in the rating range of approximately 4.0 to 4.5, indicating generally high customer satisfaction.
- The majority of machines are priced between ₹10,000 and ₹40,000, suggesting that mid-range models dominate the market.
- A few high-priced outliers above ₹60,000 exist, but they do not necessarily correspond to higher ratings, indicating that premium pricing does not always guarantee better customer ratings.
- Lower-rated machines are relatively rare, showing that most products meet acceptable quality standards.

```
# Price by Machine Type
plt.figure(figsize=(6,4))
sns.boxplot(x='Machine_Type', y='Price', data=df)
plt.title('Price by Machine Type')
plt.show()
```



Insights

Fully Automatic washing machines:

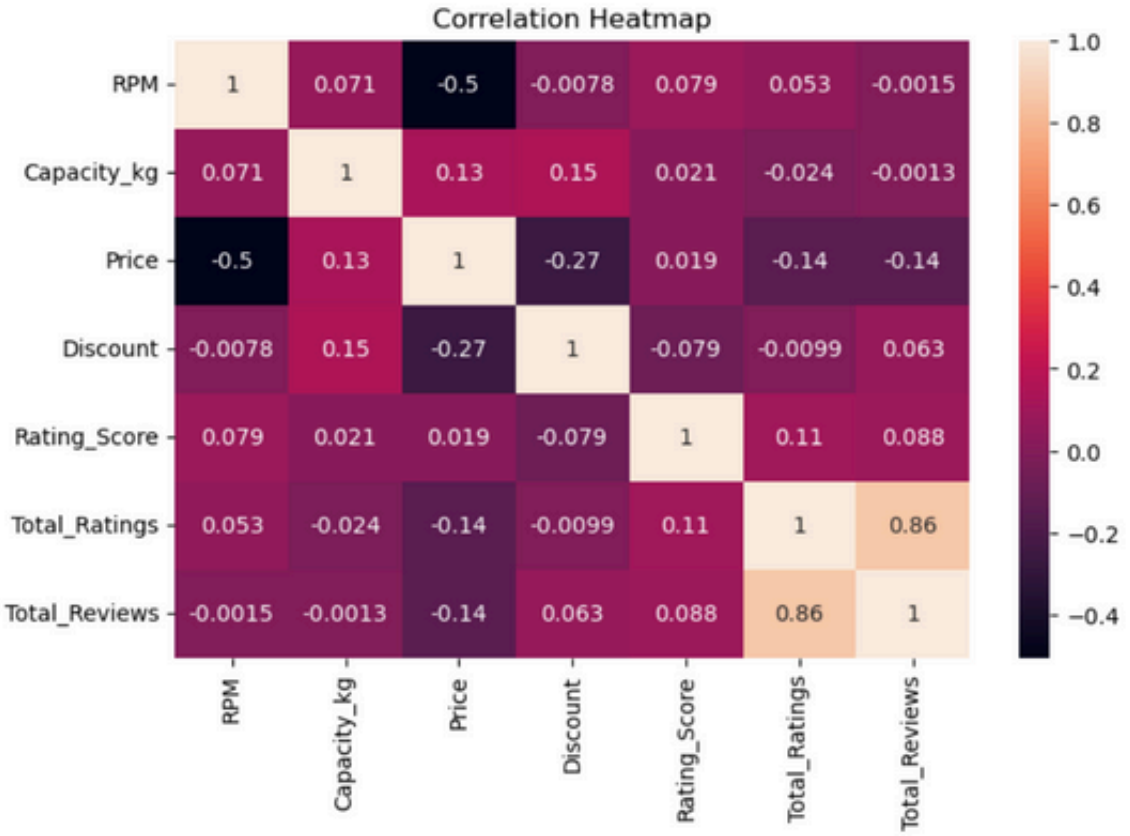
- Prices range widely from ₹5,000 to ₹65,000.
- Some outliers exceed ₹70,000, even nearing ₹100,000. These high-end outliers are common in premium brands, so they are expected.
- The median price is around ₹25,000–₹30,000.

Semi Automatic washing machines:

- Prices are more concentrated, mostly between ₹7,000 and ₹15,000, with a few outliers slightly above ₹19,000.
- Again, these higher prices usually correspond to premium models, so such outliers are typical.
- The median price is roughly ₹12,000–₹13,000.

Multivariate Analysis

```python
# Correlation Heatmap for only numeric features
numeric_df = df.select_dtypes(include='number')

plt.figure(figsize=(8,5))
sns.heatmap(numeric_df.corr(), annot=True)
plt.title('Correlation Heatmap')
plt.show()
```



Insights

- RPM is moderately negatively related to price, meaning higher RPM usually comes with lower prices.
- Price decreases as discount increases, showing a clear inverse relationship.
- Capacity has a small positive effect on price; higher capacity slightly increases price.
- Rating score has very little relationship with other variables.
- Total ratings and total reviews are strongly related and represent similar information.
- Price is slightly lower for items with more ratings and reviews.
- Overall, most relationships are weak, with limited multicollinearity except for ratings and reviews.

10. Data Storage

The cleaned and processed dataset was stored in MySQL Workbench to enable efficient management, querying, and retrieval of data. A database and relevant tables were created with appropriate column names and datatypes to match the dataset structure. The data was then inserted into these tables, ensuring integrity and consistency. Storing the data in MySQL allows seamless access for analysis, feature extraction, and machine learning tasks, as well as easy integration with Python for further processing and model deployment.

```
import sqlalchemy
from sqlalchemy import create_engine
from urllib.parse import quote

real_password = "pavisql@3015"
encoded_password = quote(real_password)

print("Encoded password:", encoded_password)
# try for pushing to mysql DB Create an SQLAlchemy engine
engine = sqlalchemy.create_engine('mysql+pymysql://root:pavisql%403015@localhost/FLIPKART')

# Push the DataFrame to MySQL
df.to_sql('flipkart_data', con=engine, if_exists='replace', index=False)
```

Extracting Data from MySQL Workbench :

```
import mysql.connector

db = mysql.connector.connect (
    host = "localhost",
    user = "root",
    port = 3306,
    password = "pavisql@3015",
    database = 'FLIPKART')

print(db)

mycursor = db.cursor(buffered=True)

# Show all tables in the database
mycursor.execute("SHOW TABLES;")
for table in mycursor:
    print(table)
    print("\n")

# using select query from mysql
mycursor.execute("SELECT * FROM FLIPKART_DATA;")
result = mycursor.fetchall()

# Print first 5 rows
for row in result[:5]:
    print(row)
```

```
mycursor.description
```

```
[('Brand', 252, None, None, None, None, 1, 16, 255),
 ('Machine_Type', 252, None, None, None, None, 1, 16, 255),
 ('Load_Type', 252, None, None, None, None, 1, 16, 255),
 ('RPM', 5, None, None, None, None, 1, 32768, 63),
 ('Capacity_kg', 5, None, None, None, None, 1, 32768, 63),
 ('Price', 8, None, None, None, None, 1, 32768, 63),
 ('Discount', 5, None, None, None, None, 1, 32768, 63),
 ('Rating_Score', 5, None, None, None, None, 1, 32768, 63),
 ('Total_Ratings', 8, None, None, None, None, 1, 32768, 63),
 ('Total_Reviews', 8, None, None, None, None, 1, 32768, 63)]
```

```
column_names = [i[0] for i in mycursor.description]

df = pd.DataFrame(result, columns=column_names)

mycursor.close()
db.close()

df.head()
```

	Brand	Machine_Type	Load_Type	RPM	Capacity_kg	Price	Discount	Rating_Score	Total_Ratings	Total_Reviews
0	LG	Fully Automatic	Top Load	1300.0	8.0	35990	28.0	4.320	32368	140
1	Haier	Fully Automatic	Top Load	1300.0	11.0	14920	39.0	4.360	36807	408
2	Voltas	Semi Automatic	Top Load	1400.0	9.0	11990	48.0	4.350	35840	381
3	Voltas	Semi Automatic	Top Load	1400.0	6.5	7590	39.0	4.350	35840	381
4	Voltas	Semi Automatic	Top Load	1400.0	12.0	14849	29.0	4.241	241	3

11. Data Preprocessing

In data preprocessing, categorical features were encoded to make them suitable for machine learning models. Label Encoding was applied to convert categorical variables, such as brand and machine type, into numerical values. This transformation preserves the uniqueness of each category while enabling algorithms to process the data effectively. Encoding ensured that all features were in a numerical format, allowing seamless integration into modeling and analysis workflows.

```
from sklearn.preprocessing import LabelEncoder

categorical_cols = ['Machine_Type', 'Load_Type']
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

df.head()
```

	Brand	Machine_Type	Load_Type	RPM	Capacity_kg	Price	Discount	Rating_Score	Total_Ratings	Total_Reviews
0	LG	0	1	1300.0	8.0	35990	28.0	4.320	32368	140
1	Haier	0	1	1300.0	11.0	14920	39.0	4.360	36807	408
2	Voltas	1	1	1400.0	9.0	11990	48.0	4.350	35840	381
3	Voltas	1	1	1400.0	6.5	7590	39.0	4.350	35840	381
4	Voltas	1	1	1400.0	12.0	14849	29.0	4.241	241	3

- Saved Encoded data as :

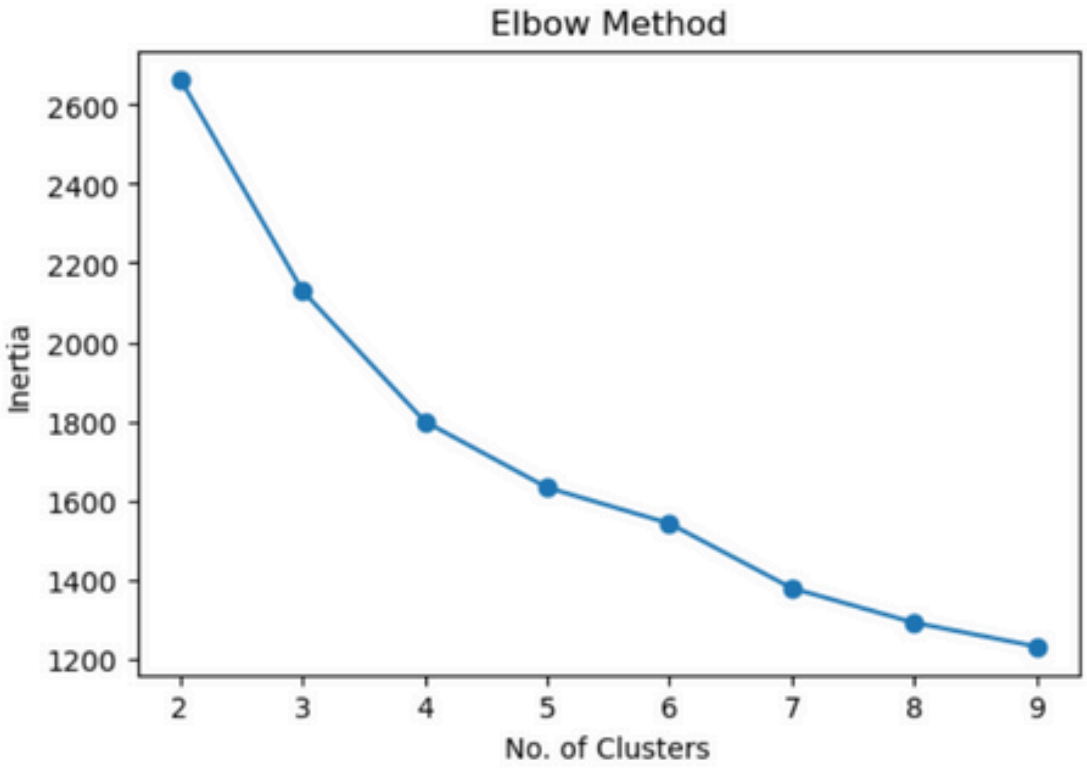
```
# Saving encoded file
df.to_csv("wash_machine.csv", index=False)
```

12. Unsupervised Machine Learning

Unsupervised machine learning was applied using K-Means clustering to identify patterns within the dataset. The algorithm grouped similar data points based on feature similarity. From this process, cluster labels were generated and assigned to each record. These labels represented distinct groups within the data and were later used for further analysis and supervised modeling.

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

# Loading data
df = pd.read_csv(r"C:\Users\WELCOME\wash_machine.csv")

# Feature Selection
features = [
    "Machine_Type",
    "Load_Type",
    "Capacity_kg",
    "RPM",
    "Price",
    "Discount",
    "Rating_Score",
    "Total_Ratings",
    "Total_Reviews"
]

X = df[features]

# Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

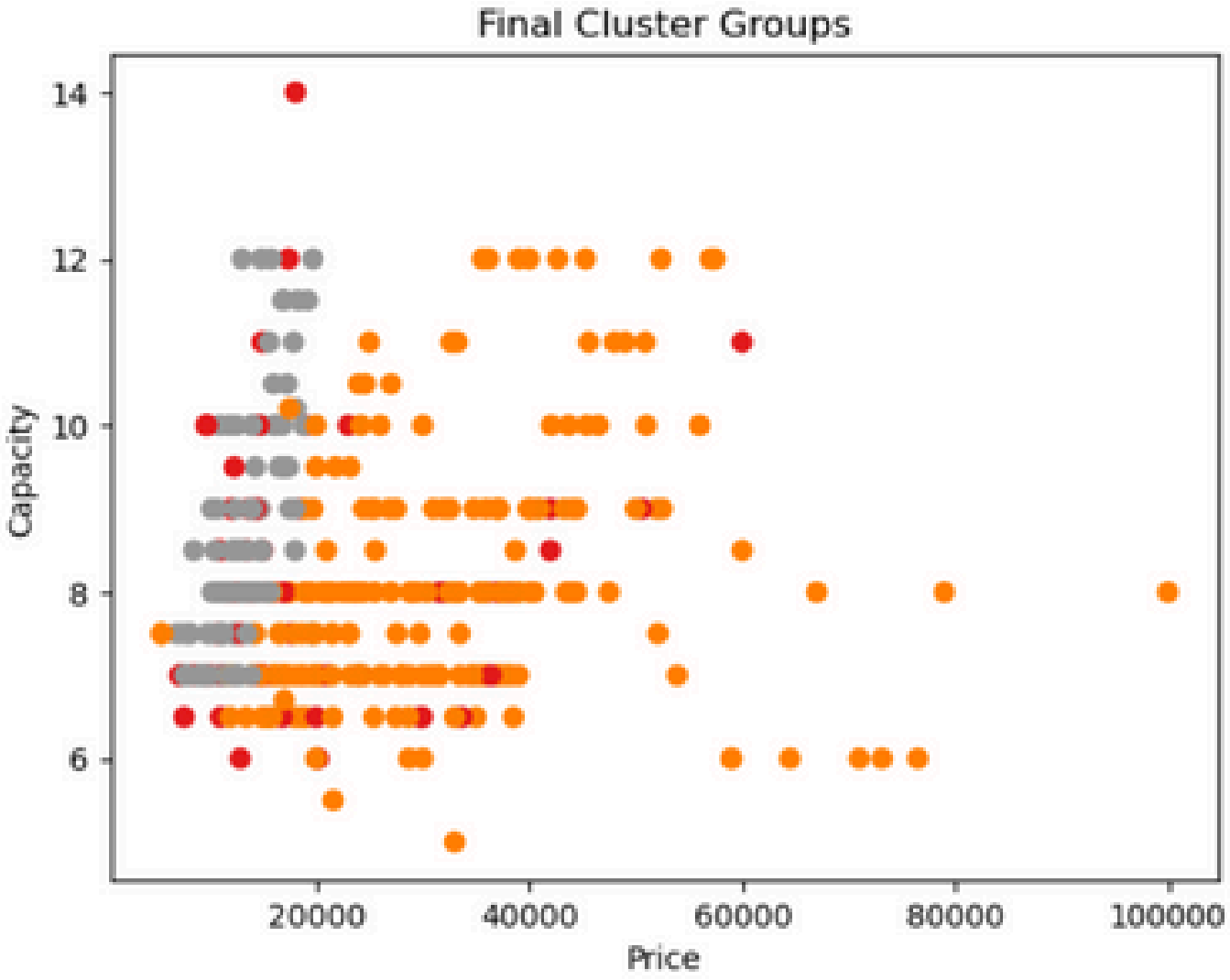
```python
# Elbow Curve & Silhoutte Score
inertia = []
sil_scores = []
K = range(2, 10)

for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(X_scaled)
    inertia.append(km.inertia_)
    sil_scores.append(silhouette_score(X_scaled, km.labels_))

plt.figure(figsize=(6,4))
plt.plot(K, inertia, marker='o')
plt.xlabel("No. of Clusters")
plt.ylabel("Inertia")
plt.title("Elbow Method")
plt.show()
```



```
# Best K
best_k = 3
kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
df["Cluster_Label"] = kmeans.fit_predict(X_scaled)

print("\nCluster Label Distribution:\n ")
print(df["Cluster_Label"].value_counts())

# Silhouette Score
sil_score = silhouette_score(X_scaled, df["Cluster_Label"])
print("\nSilhouette Score:", sil_score)
print("\n")

plt.scatter(df['Price'], df['Capacity_kg'],c=df['Cluster_Label'], cmap='Set1')
plt.xlabel("Price")
plt.ylabel("Capacity")
plt.title("Final Cluster Groups")
plt.show()


# Saving labelled data
df.to_csv("Clustered_Data.csv", index=False)
print("\n File Saved Succesfully : Clustered_Data.csv \n")
```

Output :

```
Cluster Label Distribution:

Cluster_Label
1    217
2    123
0     66
Name: count, dtype: int64

Silhouette Score: 0.3206257005616002
```




```
File Saved Succesfully : Clustered_Data.csv
```

Data Splitting & Data Preprocessing

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data.drop(["Cluster_Label", "Brand"], axis=1)
y = data["Cluster_Label"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

13. Supervised Machine Learning

Supervised machine learning techniques were applied using the generated cluster labels as the target variable. Multiple classification models were trained to learn patterns from the labeled data. Model performance was evaluated using appropriate metrics to compare results. The best-performing model was selected for further tuning and deployment.

The following machine learning algorithms were applied:

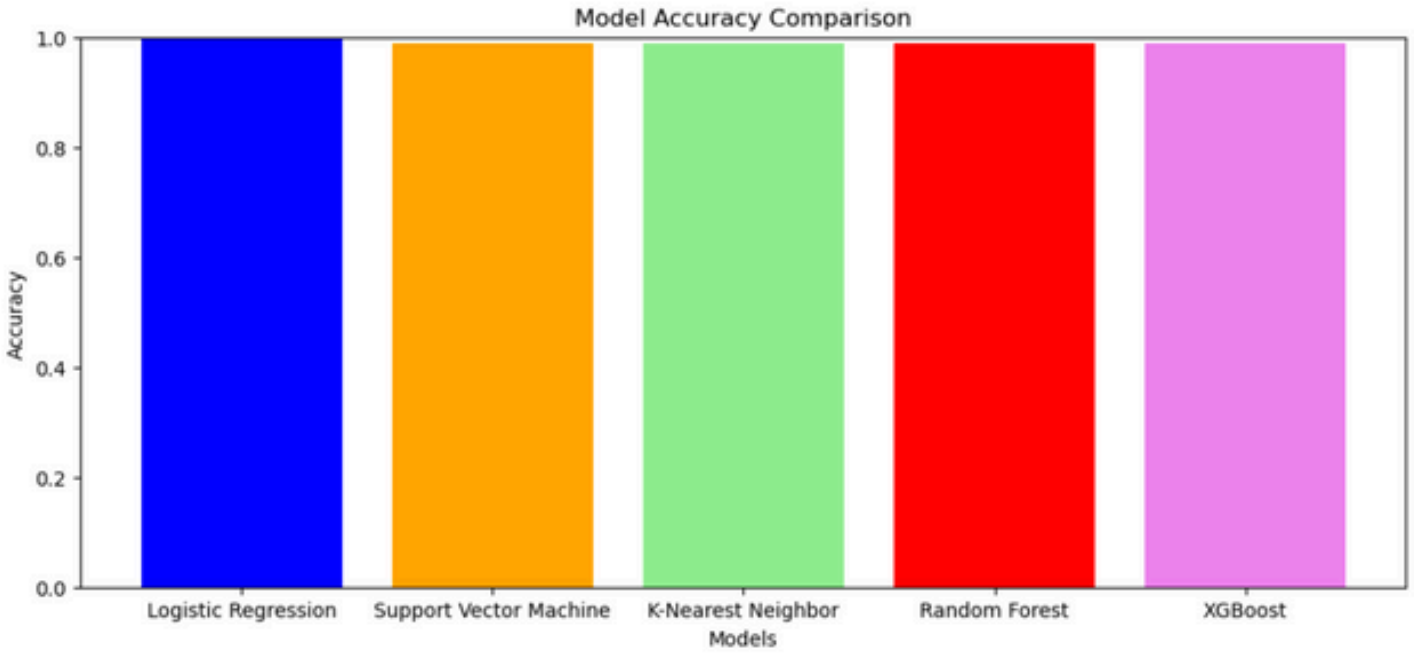
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Random Forest Classifier
- XG Boost

```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Functions to run ML Models
def classification_models(X_train, X_test, y_train, y_test):
    models = {
        "Logistic Regression": LogisticRegression(max_iter=200),
        "Support Vector Machine": SVC(kernel='rbf'),
        "K-Nearest Neighbor": KNeighborsClassifier(n_neighbors=5),
        "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
        "XGBoost": XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.05, eval_metric='mlogloss', random_state=42)}

    # List to store metrics
    results = []

    for model_name, model in models.items():
        model.fit(X_train, y_train)

        y_pred_train = model.predict(X_train)
        y_pred_test = model.predict(X_test)

        # Training Accuracy
        train_acc = accuracy_score(y_train, y_pred_train)

        # Testing metrics
        acc = accuracy_score(y_test, y_pred_test)
        precision = precision_score(y_test, y_pred_test, average='weighted', zero_division=0)
        recall = recall_score(y_test, y_pred_test, average='weighted', zero_division=0)
        f1 = f1_score(y_test, y_pred_test, average='weighted', zero_division=0)

        results.append({
            "Model": model_name,
            "Training Accuracy": train_acc,
            "Testing Accuracy": acc,
            "Precision": precision,
            "Recall": recall,
            "F1-Score": f1})

    # Converting to Dataframe
    results_df = pd.DataFrame(results)
    results_df = results_df.reset_index(drop=True)
    return results_df

comparison_df = classification_models(X_train, X_test, y_train, y_test)
comparison_df
```

	Model	Training Accuracy	Testing Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	1.000000	1.000000	1.000000	1.000000	1.000000
1	Support Vector Machine	0.998914	0.987805	0.988186	0.987805	0.987876
2	K-Nearest Neighbor	0.998914	0.987805	0.988186	0.987805	0.987876
3	Random Forest	1.000000	0.987805	0.988186	0.987805	0.987876
4	XGBoost	1.000000	0.987805	0.988186	0.987805	0.987876

- Compared Accuracy Scores for all Classification Models.

```python
# Plot Comparison
models = comparison_df['Model']
accuracy = comparison_df['Testing Accuracy']

plt.figure(figsize=(12,5))
plt.bar(models, accuracy, color=['blue','orange','lightgreen','red','violet'])
plt.title("Model Accuracy Comparison")
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.show()
```



14. Hyperparameter Tuning

Hyperparameter tuning was performed for all machine learning models to optimize performance and improve accuracy. Multiple parameter combinations were tested to identify the most effective settings for each model. This process helped reduce overfitting and improve generalization. The best-tuned versions of all models were evaluated and compared using performance metrics.

GRIDSEARCHCV

```python
from sklearn.model_selection import GridSearchCV

# Parameter Grids
params = {
    "Logistic Regression": {"C": [0.1, 1, 10], "solver": ["liblinear"]},
    "Support Vector Machine": {"C": [0.1, 1, 10], "kernel": ["rbf","linear"]},
    "K-Nearest Neighbor": {"n_neighbors": [3, 5, 7]},
    "Random Forest": {"n_estimators": [100, 200], "max_depth": [None, 5, 10]},
    "XGBoost": {"n_estimators": [100, 200], "max_depth": [3, 5],
                "learning_rate": [0.05, 0.1], "subsample": [0.8, 1.0],"colsample_bytree": [0.8, 1.0] }
}

# Models
models = {
    "Logistic Regression": LogisticRegression(max_iter=300),
    "Support Vector Machine": SVC(),
    "K-Nearest Neighbor": KNeighborsClassifier(),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(eval_metric='mlogloss', use_label_encoder=False, random_state=42)
}

# GRID SEARCH LOOP
results = []

for name, model in models.items():
    grid = GridSearchCV(model, params[name], cv=3, scoring='accuracy', n_jobs=-1)
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    best_params = grid.best_params_
    test_acc = best_model.score(X_test, y_test)

    results.append({
        "Model": name,
        "Best Params": best_params,
        "Test Accuracy": test_acc
    })

# Results
tuning_results = pd.DataFrame(results)
tuning_results
```

	Model	Best Params	Test Accuracy
0	Logistic Regression	{'C': 0.1, 'solver': 'liblinear'}	0.987805
1	Support Vector Machine	{'C': 1, 'kernel': 'linear'}	0.987805
2	K-Nearest Neighbor	{'n_neighbors': 7}	0.987805
3	Random Forest	{'max_depth': None, 'n_estimators': 100}	0.987805
4	XGBoost	{'colsample_bytree': 0.8, 'learning_rate': 0.0...	0.987805

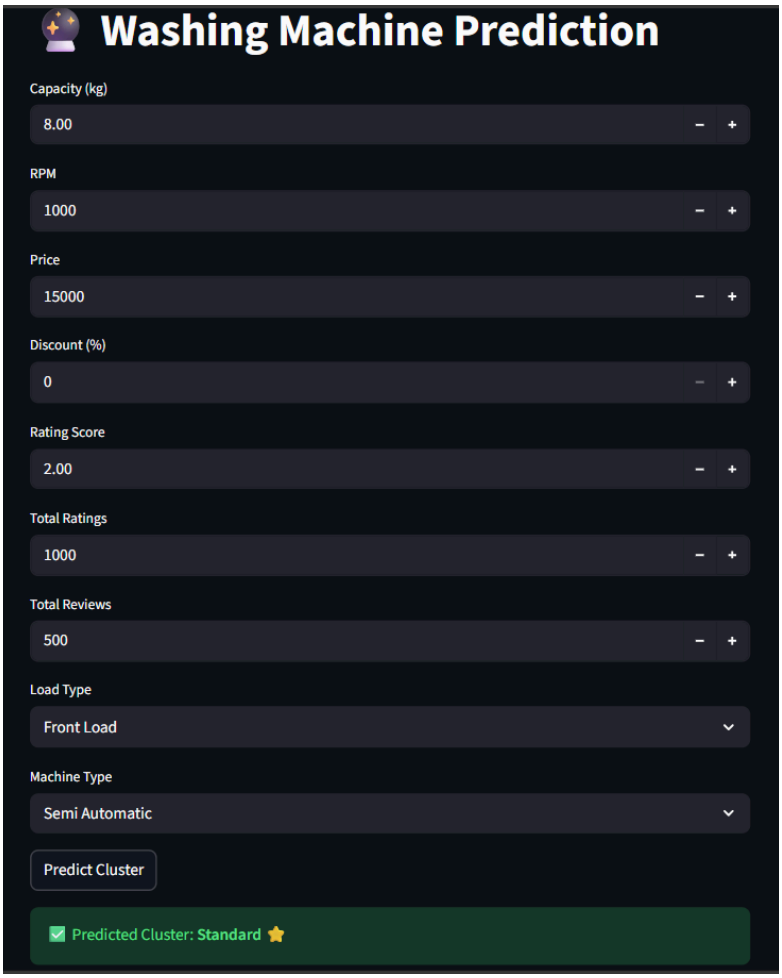
- Comparing Accuracy for all models Before and After Tuning :

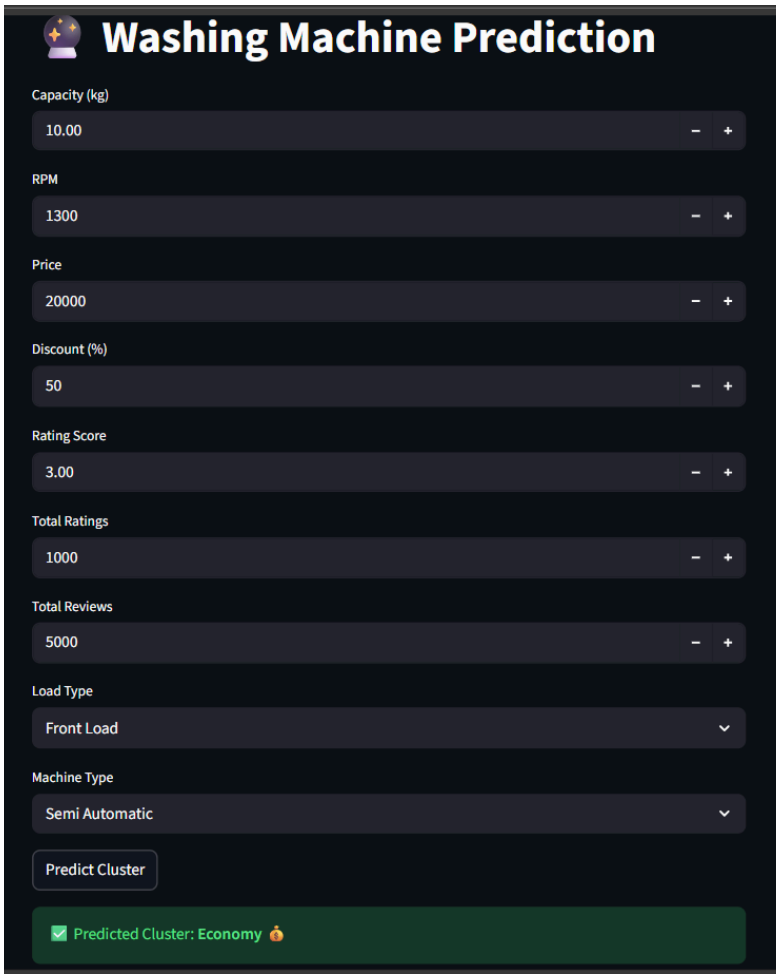
```
comparison_df["Model"] = comparison_df["Model"].str.strip()
tuning_results["Model"] = tuning_results["Model"].str.strip()

before_acc = dict(zip(comparison_df["Model"], comparison_df["Testing Accuracy"]))
after_acc = dict(zip(tuning_results["Model"], tuning_results["Test Accuracy"]))

comparison_list = []

for model in before_acc.keys():
    comparison_list.append([
        model,
        before_acc[model],
        after_acc.get(model, None)
    ])

accuracy_compare = pd.DataFrame(
    comparison_list,
    columns=["Model", "Before Tuning", "After Tuning"]
)

accuracy_compare
```

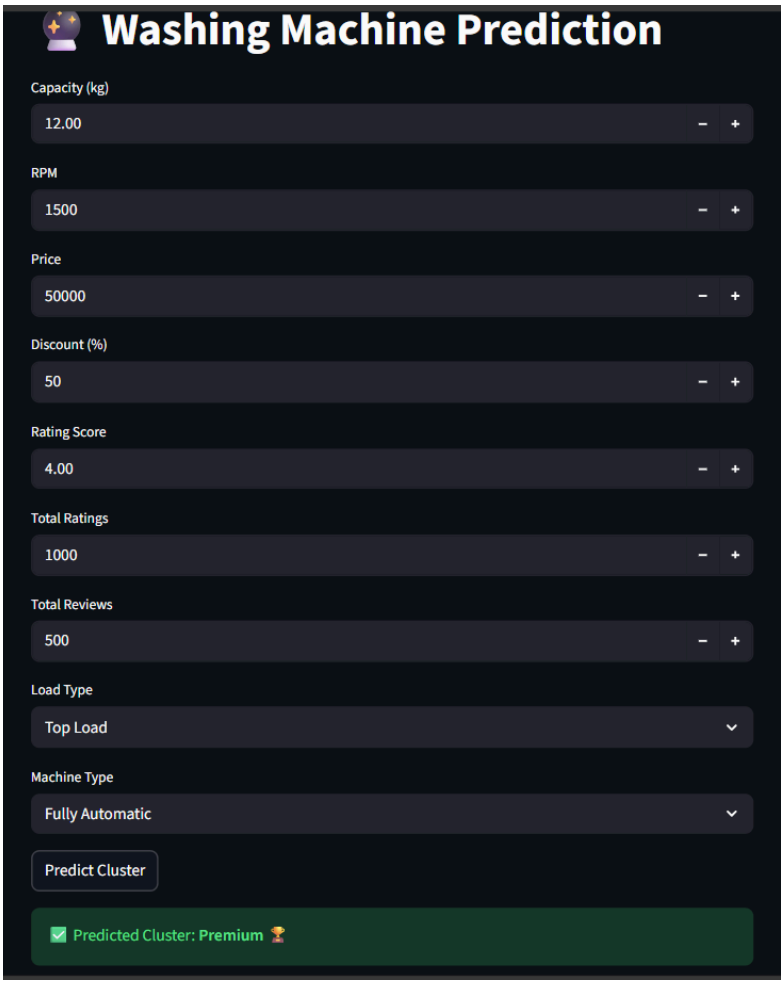
	Model	Before Tuning	After Tuning
0	Logistic Regression	1.000000	0.987805
1	Support Vector Machine	0.987805	0.987805
2	K-Nearest Neighbor	0.987805	0.987805
3	Random Forest	0.987805	0.987805
4	XGBoost	0.987805	0.987805

15. Model Deployment:

The best-performing machine learning model was deployed to make real-time predictions. The model was integrated into a user-friendly application using Python and Streamlit. User inputs are processed and passed to the trained model to generate predictions. This deployment enables practical usage of the model for decision-making and end-user interaction.


Washing Machine Prediction
Capacity (kg)
8.00
RPM
1000
Price
15000
Discount (%)
0
Rating Score
2.00
Total Ratings
1000
Total Reviews
500
Load Type
Front Load
Machine Type
Semi Automatic
Predict Cluster
Predicted Cluster: Standard


Washing Machine Prediction
Capacity (kg)
10.00
RPM
1300
Price
20000
Discount (%)
50
Rating Score
3.00
Total Ratings
1000
Total Reviews
5000
Load Type
Front Load
Machine Type
Semi Automatic
Predict Cluster
Predicted Cluster: Economy



16. Overall Insights

- Built an end-to-end machine learning pipeline from data collection to deployment.
- Web scraping ensured up-to-date and relevant data from online sources.
- Feature engineering improved data quality and model interpretability.
- K-Means clustering helped generate meaningful labels for supervised learning.
- Multiple machine learning models were trained and evaluated.
- Hyperparameter tuning improved accuracy and reduced overfitting across models.
- Logistic Regression emerged as the best-performing model.
- Model deployment enabled real-time prediction using a Streamlit-based application.
- The deployed model supports data-driven decision-making.
- The solution is scalable and can be extended with new data or features.

17. Recommendations

- Update the dataset periodically to maintain model accuracy and relevance.
- Add more features to capture user behavior and product characteristics.
- Monitor deployed model performance to identify data or concept drift.
- Retrain the model at regular intervals using new data.
- Improve application UI to enhance user experience and usability.
- Integrate the system with live databases or APIs for automated data flow.
- Extend the solution to support advanced analytics and business insights.

18. Conclusion

This project successfully demonstrated an end-to-end machine learning workflow, starting from web-based data collection to model deployment. Effective data cleaning, preprocessing, and feature engineering ensured high-quality inputs for modeling. Unsupervised and supervised learning techniques were applied to extract patterns and build accurate predictive models. The final deployed application enables real-time predictions, making the solution practical, scalable, and valuable for data-driven decision-making.