APPLIED DATA SCIENCE

IBM NAAN MUDHALVAN Phase-5

TEAM MEMBERS

- DHANULESH.S
- SURIYA.V
- NARENTHIRA KUMAR.G
- MAHESWAREN.R
- ARYAN REDDY.R

PROJECT TITLE : PRODUCT DEMAND PREDICTION WITH MACHINE LEARNING

PHASE 5 : PROJECT DOCUMENTATION & SUBMISSION

TOPIC : In this section we will document the complete project and prepare it for submission.

PRODUCT DEMAND PREDICTION

INTRODUCTION:

- Product Demand Prediction with Machine Learning: A Paradigm Shift in Business Strategy

- In the contemporary business landscape, the accurate prediction of product demand is the linchpin upon which success pivots. The traditional methods of demand forecasting, reliant on historical data and static models, are being rapidly overshadowed by the transformative power of machine learning. This evolution represents more than just a technological advancement; it signifies a fundamental change in how organizations perceive and respond to consumer behavior.

- At the heart of every business operation, demand prediction influences inventory management, production planning, pricing strategies, and the overall customer experience. Machine learning, powered by advanced algorithms, vast datasets, and the agility to adapt to dynamic market conditions, is reshaping the landscape. It enables companies to extract profound insights from intricate patterns and connections in data, often imperceptible to the human eye.

- Machine learning-driven demand prediction is a game-changer. It empowers businesses to foresee consumer preferences, seasonal fluctuations, market trends, and a multitude of variables with an unprecedented level of precision. By doing so, it delivers tangible benefits, such as optimized resource allocation, reduced waste, improved customer satisfaction, and enhanced competitiveness.

- This paradigm shift is not limited to industry giants but has become a necessity for organizations of all sizes. As businesses embrace machine learning, they can proactively tailor their strategies, avoid stockouts, minimize overstock situations, and optimize marketing efforts. In essence, the adoption of machine learning in demand prediction represents an evolution that transcends industries and scales, with the potential to redefine business practices.

- Throughout this comprehensive exploration of product demand prediction with machine learning, we will delve into the methodologies, cutting-edge tools, and best practices that are revolutionizing the way companies operate. We will uncover the significance of integrating machine learning into the contemporary

business landscape and reveal the real-world applications and success stories that underscore its transformative potential. Join us as we embark on a journey through the future of demand prediction, where data-driven insights and machine learning are reshaping the way businesses navigate the complexities of consumer preferences and market dynamics.

DATASET LINK :
(https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning)

LIST OF TOOLS AND SOFTWARE COMMONLY USED USED IN THE PROCESS:

here are sentences for each of the topics you mentioned:

1.Programming Language:

Python serves as the foundation for our product demand prediction, providing a robust and flexible language for machine learning.

2.Integrated Development Environment (IDE):

Jupyter Notebooks, with its interactive interface, enables our data scientists to craft and fine-tune predictive models with ease and efficiency.

3.Machine Learning Libraries:

Our toolkit includes renowned libraries like Scikit-Learn, TensorFlow, and XGBoost, harnessing their powerful algorithms to uncover intricate demand patterns.

4.Data Visualization Tools:

To convey our insights effectively, we rely on Tableau, Matplotlib, and Plotly, creating compelling visual representations of demand trends.

5.Data Processing Tools:

Pandas and NumPy handle data manipulation with precision, ensuring that our datasets are prepared and structured optimally for analysis.

6.Data Collection and Storage:

We securely store and access our data using Amazon S3 for object storage, allowing for seamless data retrieval when needed.

7.Version Control:

Git, alongside platforms like GitHub and GitLab, safeguards our codebase, ensuring that collaborative development remains synchronized and error-free.

8.Notebooks and Documents:

Our documentation and insights come to life through Jupyter Notebooks, while RMarkdown caters to the specific needs of our R-based analysts.

9.Hyperparameter Tuning:

Fine-tuning the performance of our models is achieved through tools such as GridSearchCV and RandomizedSearchCV from Scikit-Learn.

10.Web Development Tools:

When deploying our solutions online, Flask and Django emerge as the go-to web development frameworks, allowing for scalable and user-friendly interfaces.

11.Cloud Services:

We take advantage of Amazon Web Services (AWS) for scalable cloud resources that power our machine learning infrastructure.

12.External Data Sources:

Supplementing our internal data, we gather valuable information from diverse sources, including public datasets and APIs.

13.Data Annotations and Labeling Tools:

In the complex task of data labeling, Labelbox and Prodigy ensure the accuracy and quality of our annotated datasets.

14.Geospatial Tools:

For projects involving geographic elements, we turn to geospatial tools like ArcGIS and QGIS to perform intricate spatial analyses and visualizations.

1.DESIGN THINKING AND PRESENT IN FORM OF DOCUMENT

Introduction:

- In the pursuit of effective product demand prediction with machine learning, we embark on a journey guided by the principles of design thinking. This document is not just a blueprint but a dynamic canvas, where empathy, ideation, and experimentation converge to create innovative solutions. We'll employ this methodology to transform our product demand prediction process, enhancing its accuracy and impact.

1. Empathize: Understanding Stakeholder Needs

- Stakeholder Mapping: Identify key stakeholders, including data scientists, business analysts, and decision-makers.
- User Personas: Create personas that represent the needs, goals, and pain points of each stakeholder group.
- User Interviews: Conduct interviews to gain deep insights into their expectations and challenges.

2. Define: Framing the Challenge

- Problem Statement: Clearly define the challenge, such as "How might we improve demand prediction accuracy to reduce inventory costs?"
- Success Criteria: Establish measurable goals, e.g., "Achieve a 10% reduction in stockouts within six months."

3. Ideate: Generating Innovative Solutions

- Brainstorming Sessions: Organize brainstorming sessions involving cross-functional teams to generate a wide range of ideas.
- Idea Prioritization: Use techniques like affinity mapping and dot voting to prioritize ideas.
- Prototyping: Develop rapid prototypes of potential solutions, including machine learning models and data visualization techniques.

4. Prototype: Building & Testing

- Model Development: Create machine learning models using Python and relevant libraries (e.g., Scikit-Learn, TensorFlow).
- Data Visualization: Design interactive dashboards using Tableau for exploring data trends.
- User Testing: Collect feedback from stakeholders through usability testing and refine prototypes accordingly.

5. Test: Gathering Feedback

- A/B Testing: Implement A/B tests to assess the performance of the new demand prediction model.
- Feedback Loops: Continuously collect feedback from users to iterate and improve the solution.
- Data Validation: Ensure data quality and accuracy throughout the process.

6. Implement: Deployment and Integration

- Deployment Plan: Develop a plan for integrating the improved demand prediction model into existing systems.
- User Training: Provide training sessions to ensure that stakeholders can effectively use the new tools.
- Monitoring: Implement monitoring systems to track model performance in real-time.

7. Evaluate: Measuring Impact

- Key Performance Indicators (KPIs): Measure the defined success criteria and assess the impact of the solution.
- Feedback Analysis: Analyze feedback and user experiences to identify areas for further improvement.

Conclusion: By embracing the principles of design thinking, we have approached the challenge of product demand prediction with empathy and creativity. This iterative process ensures that our solution is not just technically sound but also deeply aligned with the needs and expectations of our stakeholders. The journey continues as we iterate and refine our solution to meet the evolving demands of our dynamic marketplace.

2.DESIGN INTO INNOVATION

here is a complete process for product demand prediction with machine learning:

1. Data Collection:

- Gather historical sales data, including product attributes, pricing, and time-based information.
- Include external data sources like market trends, economic indicators, and weather data that may impact demand.

2. Data Preprocessing:

- Clean the data by handling missing values and outliers.
- Transform data into a suitable format for analysis.
- Feature engineering: Create relevant features, such as seasonality, holidays, and promotional events.

3. Data Exploration:

- Visualize the data to understand trends, patterns, and seasonality.
- Identify correlations between variables and their impact on demand.

4. Data Splitting:

- Split the data into training and testing datasets. The training data is used to build the prediction model, and the testing data is used to evaluate the model's performance.

5. Model Selection:

- Choose a machine learning model appropriate for demand prediction. Common choices include linear regression, decision trees, random forests, XGBoost, and neural networks for deep learning.

6. Model Training:

- Train the selected model on the training dataset using historical data to learn the relationship between input variables (e.g., product attributes, time, price) and demand.

7. Model Evaluation:

- Assess the model's performance using appropriate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).
- Consider using time series forecasting evaluation metrics like Mean Absolute Percentage Error (MAPE) and forecasting accuracy.

8. Hyperparameter Tuning:

- Optimize the model by fine-tuning hyperparameters to improve its accuracy and generalization.

9. Model Validation:

- Validate the model's performance on the testing dataset to ensure it generalizes well to unseen data.

10. Predict Demand:

- Utilize the trained model to make demand predictions based on new data inputs, such as product features, price changes, and upcoming events.

11. Interpret Results:

- Analyze model predictions and understand the key drivers affecting demand.

12. Deployment:

- Deploy the trained model into a production environment where it can be used for real-time or batch demand predictions.

13. Continuous Monitoring:

- Continuously monitor the model's performance in the production environment and retrain it periodically as new data becomes available.

14. Reporting and Visualization:

- Create dashboards and reports that provide insights into demand trends and predictions for decision-makers.

15. Feedback Loop:

- Establish feedback mechanisms to capture the impact of predictions on inventory management and sales strategies. Use this feedback to further improve the model.

16. Adaptation:

- Adapt the demand prediction model as market conditions, customer preferences, and product attributes change over time.

This comprehensive process for product demand prediction with machine learning combines data collection, preprocessing, model development, evaluation, and ongoing monitoring to optimize inventory management, pricing strategies, and customer satisfaction.

3.BUILD LOADING AND PREPROCESSING THE DATASET

Certainly, here are the steps for loading and preprocessing a dataset for product demand prediction with machine learning:

Step 1: Import Libraries

- Import necessary libraries, including Pandas for data manipulation and other libraries as needed for your specific preprocessing tasks.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Step 2: Load the Dataset

- Use Pandas to load your dataset from a file (e.g., CSV) into a DataFrame. Replace `"dataset.csv"` with the actual path to your dataset file.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('dataset.csv')
```

Step 3: Explore the Data

- Check the first few rows of the dataset to get an initial understanding of its structure.

```
print(df.head())
```

Step 4: Check for Missing Values

- Examine the dataset for missing values in each column.

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

Step 5: Data Preprocessing

- Depending on your dataset, you may need to perform various preprocessing tasks, such as:
- Handling missing values (e.g., dropping rows, imputing values).
- Encoding categorical variables (e.g., one-hot encoding).
- Feature engineering (creating new features).
- Scaling or normalizing features.

Step 6: Split Data into Features and Target

- Separate the dataset into features (X) and the target variable (y) that you want to predict.

```
X = df.drop('demand', axis=1)  # Replace 'demand' with your target variable
y = df['demand']
```

Step 7: Split Data into Training and Testing Sets

- Divide the data into training and testing sets to evaluate the model's performance.

```
from sklearn.model_selection import train_test_split

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

Step 8: Feature Scaling (Optional)

- If necessary, perform feature scaling to ensure that different features have similar scales.

```
from sklearn.preprocessing import StandardScaler

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

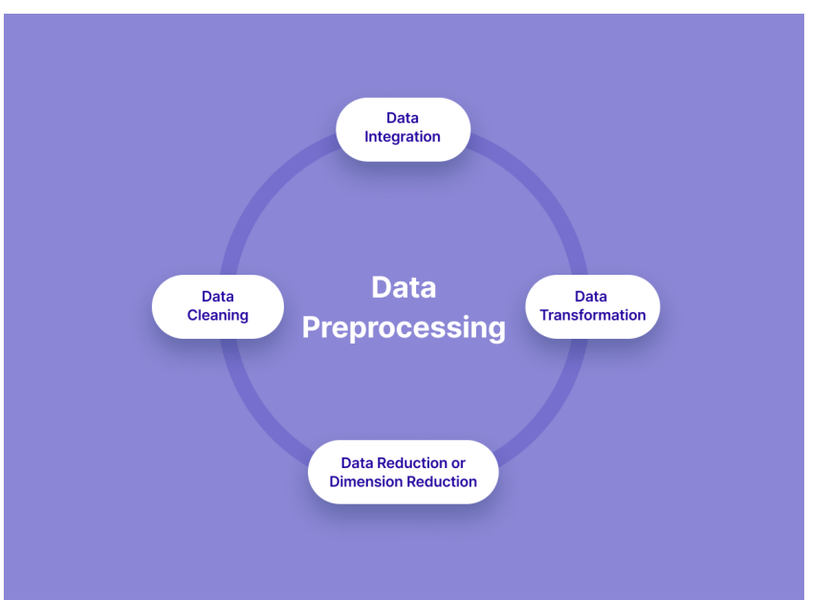
Step 9: Model Building

- Develop and train your machine learning model for demand prediction using the preprocessed training data.

Step 10: Model Evaluation

- Evaluate the model's performance on the testing dataset using appropriate evaluation metrics (e.g., Mean Absolute Error, Root Mean Squared Error).

These steps provide a framework for loading and preprocessing a dataset for product demand prediction with machine learning. The specific preprocessing tasks may vary depending on the characteristics of your dataset and the machine learning model you intend to use.



Import necessary libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
```

In [2]:
```
data = pd.read_csv("PoductDemand.csv")
data
```

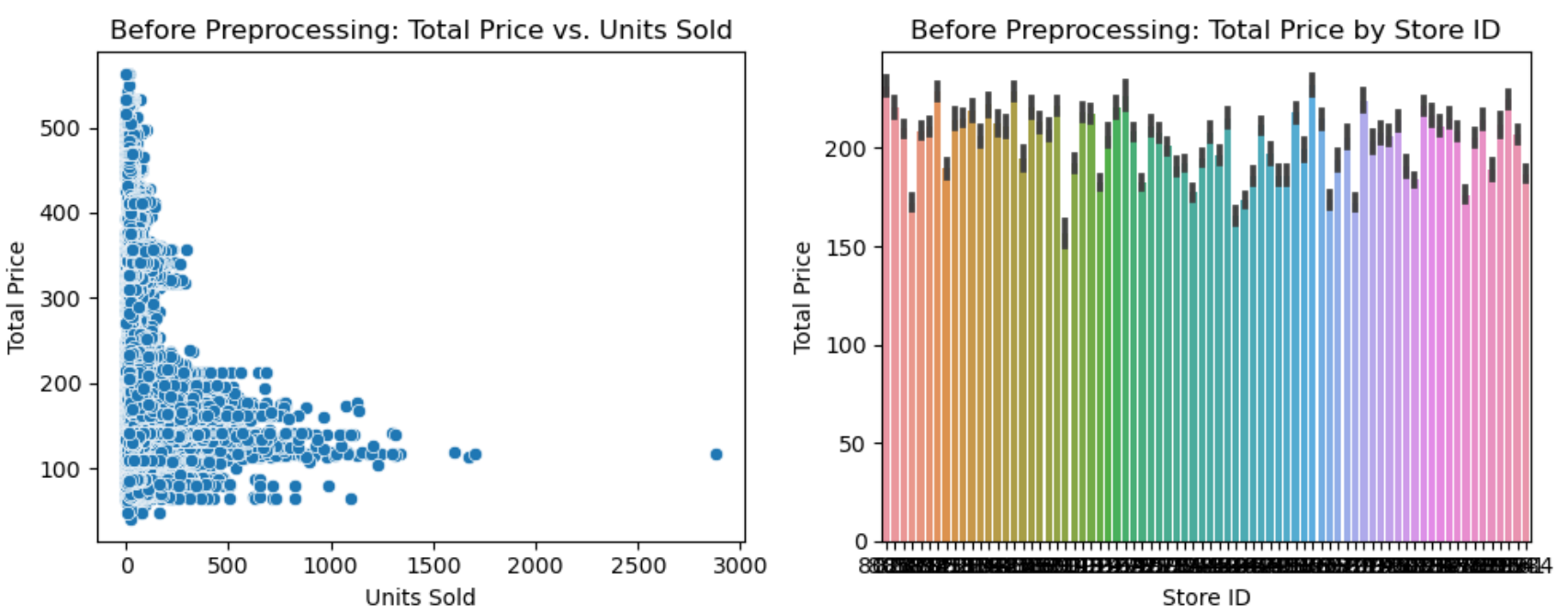
Out[2]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38
150146	212639	9984	235.8375	235.8375	30
150147	212642	9984	357.6750	483.7875	31
150148	212643	9984	141.7875	191.6625	12
150149	212644	9984	234.4125	234.4125	15

150150 rows × 5 columns

Data Visualization(before preprocessing)

In [4]:
```
plt.figure(figsize=(10, 4))

# Plot 1: Total Price vs. Units Sold using Seaborn
plt.subplot(1, 2, 1)
sns.scatterplot(x='Units Sold', y='Total Price', data=data)
plt.title('Before Preprocessing: Total Price vs. Units Sold')

# Plot 2: Total Price by Store ID using Seaborn
plt.subplot(1, 2, 2)
sns.barplot(x='Store ID', y='Total Price', data=data)
plt.title('Before Preprocessing: Total Price by Store ID')

plt.tight_layout()
plt.show()
```



1.Data Cleaning

In [5]:
```
data.fillna(0, inplace=True)
```

2.Data Encoding

In [6]:
```
label_encoder = LabelEncoder()
data['Store ID'] = label_encoder.fit_transform(data['Store ID'])
```

3.Data Scaling and Normalization

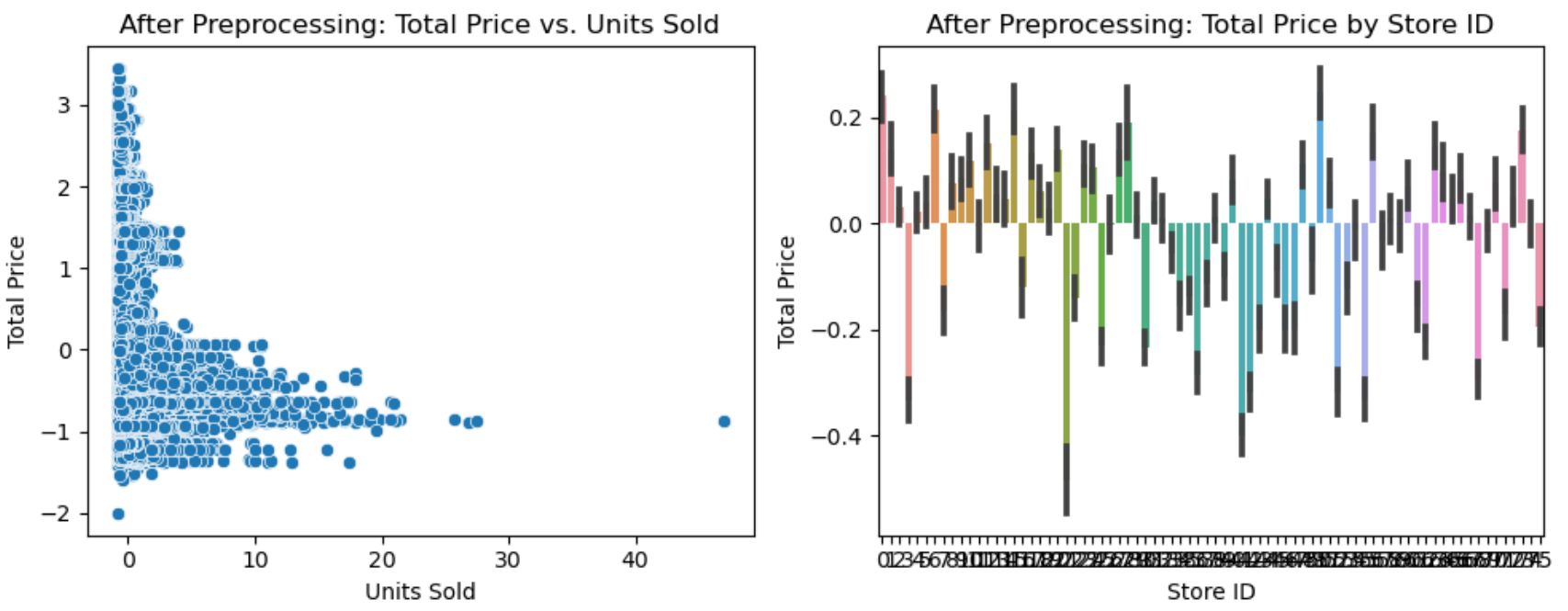
In [7]:
```
scaler = StandardScaler()
data[['Total Price', 'Base Price', 'Units Sold']] = scaler.fit_transform(data[['Total Price', 'Base Price', '
```

4.Feature Selection

In [8]:
```
selected_features = data[['Store ID', 'Total Price', 'Base Price', 'Units Sold']]
```

Data Visualization after Preprocessing

In [9]:
```
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
sns.scatterplot(x='Units Sold', y='Total Price', data=data)
plt.title('After Preprocessing: Total Price vs. Units Sold')

plt.subplot(1, 2, 2)
sns.barplot(x='Store ID', y='Total Price', data=data)
plt.title('After Preprocessing: Total Price by Store ID')

plt.tight_layout()
plt.show()
```



5.Data plitting

In [10]:
```
X = selected_features
y = data['Total Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Importing Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
```

Load csv File

In [2]:
```
data = pd.read_csv("ProductDemand.csv")
data
```

Out[2]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52
...	...	...	...	...	...
150145	212638	9984	235.8375	235.8375	38
150146	212639	9984	235.8375	235.8375	30
150147	212642	9984	357.6750	483.7875	31
150148	212643	9984	141.7875	191.6625	12
150149	212644	9984	234.4125	234.4125	15

150150 rows × 5 columns

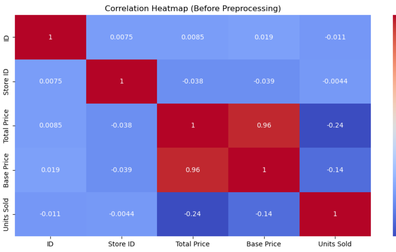
Data Inspection

In [3]:
```
print(data.head())
print(data.info())
```

```
   ID  Store ID  Total Price  Base Price  Units Sold
0   1      8091      99.0375    111.8625          20
1   2      8091      99.0375     99.0375          28
2   3      8091     133.9500    133.9500          19
3   4      8091     133.9500    133.9500          44
4   5      8091     141.0750    141.0750          52
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150150 entries, 0 to 150149
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   ID            150150 non-null  int64  
 1   Store ID      150150 non-null  int64  
 2   Total Price   150149 non-null  float64
 3   Base Price    150150 non-null  float64
 4   Units Sold    150150 non-null  int64  
dtypes: float64(2), int64(3)
memory usage: 5.7 MB
None
```

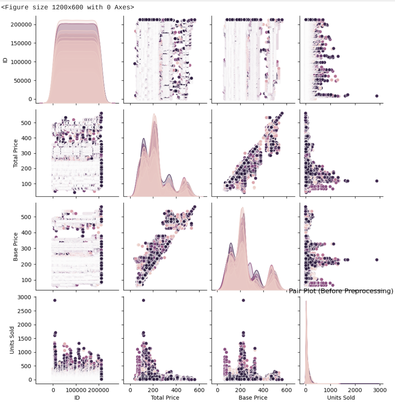
Data Visualization

In [4]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap (Before Preprocessing)")
plt.show()
```



In [6]:
```
plt.figure(figsize=(12, 6))
sns.pairplot(data, hue='Store ID')
plt.title("Pair Plot (Before Preprocessing)")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Data Cleaning

Removal of Data Duplicates

In [7]:
```
data = data.drop_duplicates()
```

Handle Missing Values

In [8]:
```
data = data.dropna()
```

Data Tranformation

In [9]:
```
numeric_features = [ "Base Price", "Units Sold", "Store ID"]
```

Data Splitting

In [10]:
```
X = data[numeric_features]
y = data["Total Price"]
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Feature Transformation Pipeline

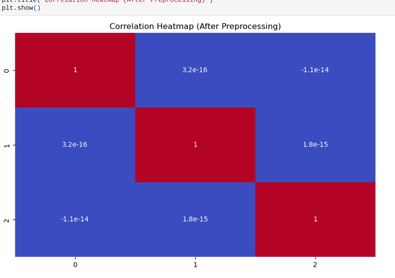
In [12]:
```
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features)
    ])
```

Dimensionality Reduction with PCA

In [13]:
```
pca = PCA(n_components=3)
data_preprocessed_pca = pca.fit_transform(preprocessor.fit_transform(X))
```

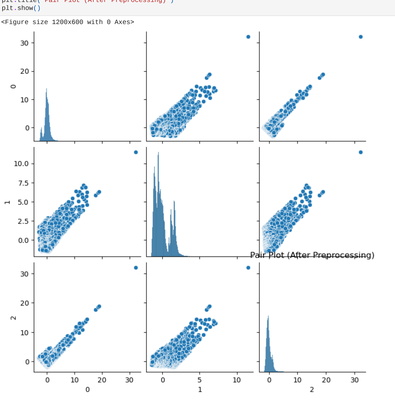
Data Visualization

In [14]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(pd.DataFrame(data_preprocessed_pca).corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap (After Preprocessing)")
plt.show()
```



In [15]:
```
plt.figure(figsize=(12, 6))
sns.pairplot(pd.DataFrame(data_preprocessed_pca))
plt.title("Pair Plot (After Preprocessing)")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



4.PERFORMING DIFFERENT ACTIVITIES LIKE FEATURE ENGINEERING,MODEL TRAINING AND EVALUATION

Certainly, here's an expanded set of steps that includes various activities such as feature engineering, model training, evaluation, and more:

Step 1: Import Libraries

- Import necessary libraries, including Pandas for data manipulation, scikit-learn for machine learning, and other libraries for specific tasks.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
```

Step 2: Load the Dataset

- Load your dataset from a file into a Pandas DataFrame.

```
df = pd.read_csv('dataset.csv')
```

Step 3: Explore the Data

- Check the first few rows and summary statistics of the dataset to understand its structure.

```
print(df.head())
```

```
print(df.describe())
```

Step 4: Data Preprocessing

- Handle missing values, encode categorical variables, and perform feature engineering as needed. For example, create new features or transform existing ones.

```
# Handle missing values
df.dropna(inplace=True)

# Encode categorical variables if necessary
df = pd.get_dummies(df, columns=['category'], drop_first=True)

# Feature engineering (e.g., creating time-based features)
df['month'] = pd.to_datetime(df['date']).dt.month
```

Step 5: Split Data into Features and Target

- Separate the dataset into features (X) and the target variable (y) that you want to predict.

```
X = df.drop('demand', axis=1)
y = df['demand']
```

Step 6: Split Data into Training and Testing Sets

- Divide the data into training and testing sets to evaluate the model's performance.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

```
random_state=42)
```

Step 7: Feature Scaling (Optional)

- Perform feature scaling if necessary to ensure that different features have similar scales.

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

Step 8: Model Training

- Develop and train your machine learning model for demand prediction using the preprocessed training data.

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Step 9: Model Evaluation

- Evaluate the model's performance on the testing dataset using appropriate evaluation metrics, such as Mean Absolute Error (MAE).

```
y_pred = model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')
```

Step 10: Hyperparameter Tuning (Optional)

- Fine-tune the model's hyperparameters to optimize performance.

Step 11: Deployment (Optional)

- Deploy the trained model into a production environment for real-time predictions.

These steps encompass a full workflow, from data loading and preprocessing to model training and evaluation. You can further enhance this process by experimenting with different models and techniques based on your specific demand prediction requirements.

LIST OF MACHINE LEARNING MODEL

1. Linear Regression:

- Definition: Linear regression is a statistical and machine learning technique used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.
- Equation: The simple linear regression equation is y = mx + b, where y is the dependent variable, x is the independent variable, m is the slope, and b is the intercept.
- Use Cases: Linear regression is commonly used for tasks like predicting stock prices, sales forecasting, and analyzing the relationship between variables.

2. XGBoost Regression:

- Definition: XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that falls under the category of ensemble methods. It uses a gradient boosting framework to build an ensemble of decision trees and makes predictions by combining their outputs.
- Key Features: XGBoost is known for its efficiency, speed, and ability to handle complex relationships in data. It can perform both regression and classification tasks.
- Use Cases: XGBoost is widely used in Kaggle competitions and real-world applications for tasks like housing price prediction, credit risk assessment, and click-through rate prediction.

3. Random Forest Regression:

- Definition: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. In the case of regression, it averages the predictions from individual trees to make a final prediction.
- Key Features: Random Forest is known for its ability to handle high-dimensional data, work with both numerical and categorical features, and handle missing values effectively.
- Use Cases: Random Forest regression is used in various applications, including stock price prediction, medical diagnosis, and environmental modeling.

4. Ridge Regression:

- Definition: Ridge regression is a regularization technique used in linear regression to prevent overfitting. It adds a penalty term to the linear regression equation to restrict the coefficients from becoming too large.
- Purpose: Ridge regression helps improve the generalization of the model by reducing the impact of multicollinearity (correlation between independent variables) and preventing overfitting.
- Use Cases: Ridge regression is applied in scenarios where there are multiple correlated independent variables, such as in economics, finance, and healthcare.

5. Lasso Regression:

- Definition: Lasso (Least Absolute Shrinkage and Selection Operator) regression is another regularization technique used in linear regression. It adds a penalty term that forces some of the coefficient values to become exactly zero.
- Purpose: Lasso regression is particularly useful for feature selection, as it can automatically select a subset of the most important features while setting others to zero.
- Use Cases: Lasso regression is used in fields like genetics, image processing, and finance, where feature selection and interpretability are crucial.

Each of these regression techniques has its unique characteristics and use cases, making them valuable tools in the field of data analysis and machine learning. The choice of which regression method to use depends on the specific problem and the nature of the dataset.

Linear Regression

Import libraries

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Load the dataset

In [4]:
```
data = pd.read_csv('PoductDemand.csv')
data.head()
```

Out[4]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

Data Cleaning

In [9]:
```
data.fillna(0, inplace=True)
```

Define features and target variable

In [10]:
```
X = data[['Store ID', 'Total Price', 'Base Price']]
y = data['Units Sold']
```

Data Split

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection

In [12]:
```
model = LinearRegression()
```

Model Training

In [13]:
```
model.fit(X_train, y_train)
```

Out[13]:
```
LinearRegression()
```

Model Evaluation

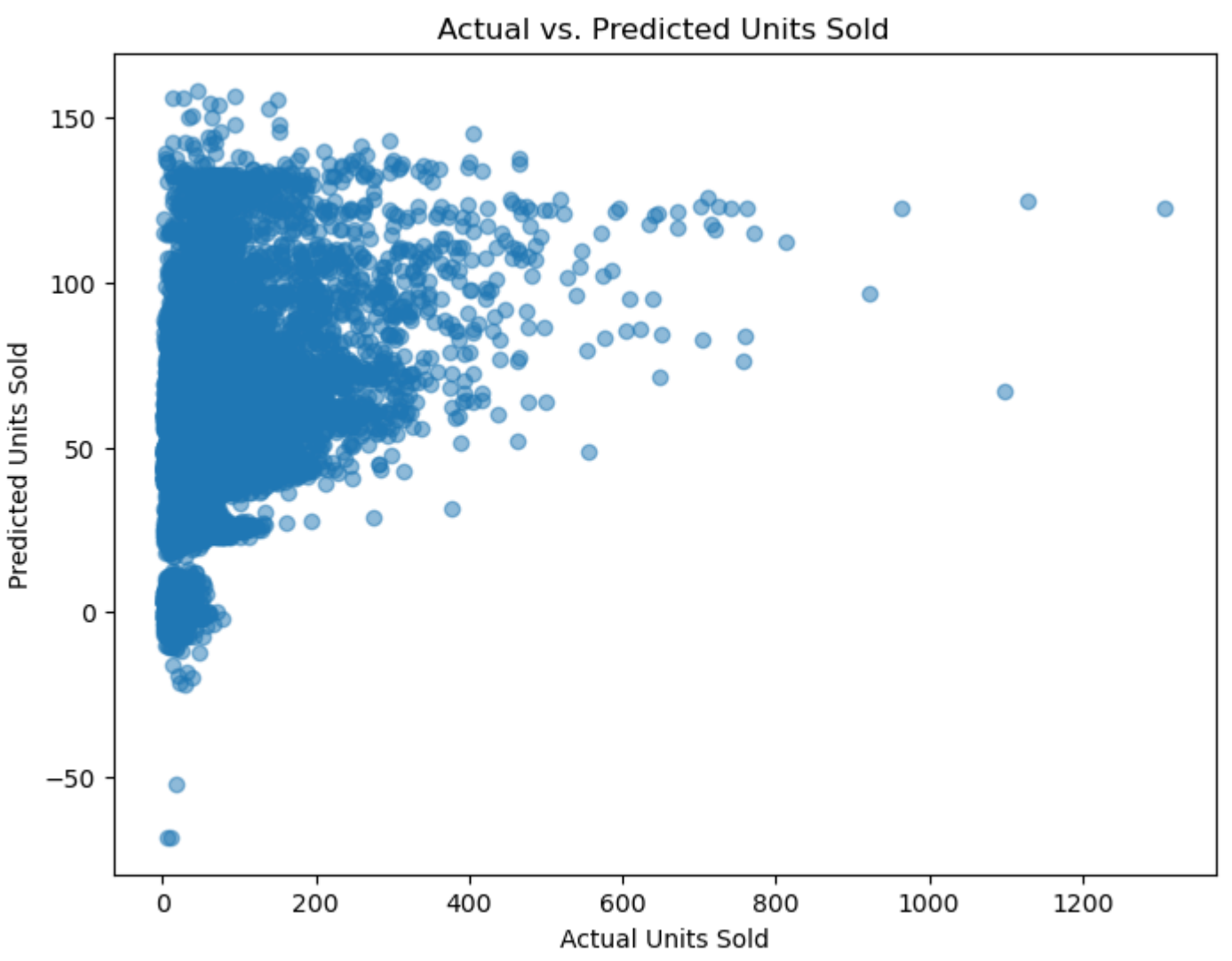
In [14]:
```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

In [15]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 32.626495087826754
Mean Squared Error: 2780.795549996203
Root Mean Squared Error: 52.73324899905375
R-squared: 0.15248746523857437
```

Visualization - Scatter plot of Actual vs. Predicted

In [16]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Units Sold')
plt.ylabel('Predicted Units Sold')
plt.title('Actual vs. Predicted Units Sold')
plt.show()
```



Random Forest Regressor

Import libraries

In [17]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Load the dataset

In [18]:
```
data = pd.read_csv('PoductDemand.csv')
data.head()
```

Out[18]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

Data Cleaning

In [19]:
```
data.fillna(0, inplace=True)
```

Define features and target variable

In [20]:
```
X = data[['Store ID', 'Total Price', 'Base Price']]
y = data['Units Sold']
```

Data Split

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection and Training

In [22]:
```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[22]:
```
RandomForestRegressor(random_state=42)
```

Model Evaluation

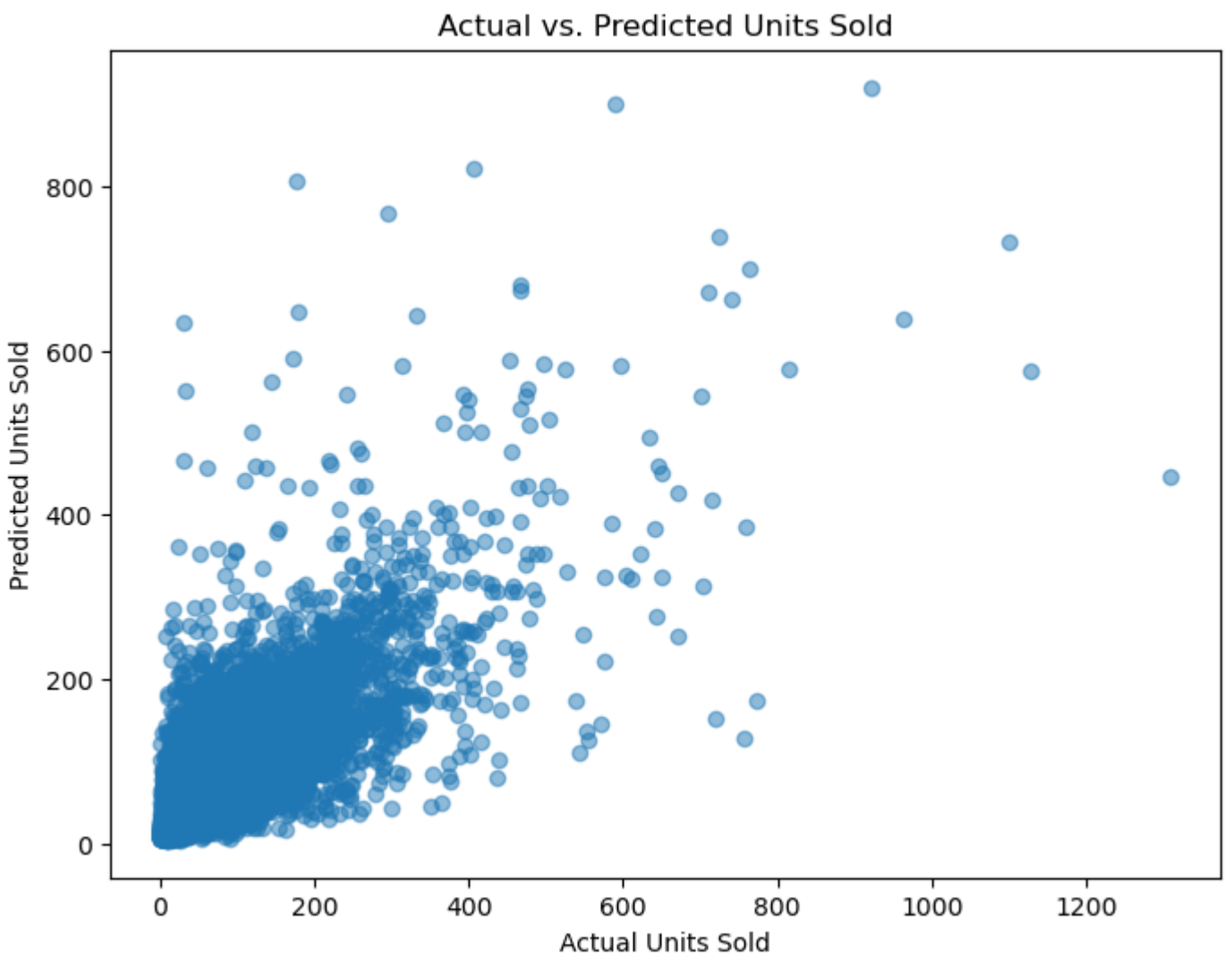
In [23]:
```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

In [24]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 19.003132154269775
Mean Squared Error: 1251.6950005899928
Root Mean Squared Error: 35.37930186691072
R-squared: 0.618516649776832
```

Visualization - Scatter plot of Actual vs. Predicted

In [25]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Units Sold')
plt.ylabel('Predicted Units Sold')
plt.title('Actual vs. Predicted Units Sold')
plt.show()
```



Extreme Gradient Boosting

Import libraries

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Load the dataset

In [3]:
```
data = pd.read_csv('PoductDemand.csv')
data.head()
```

Out[3]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

Data Cleaning

In [4]:
```
data.fillna(0, inplace=True)
```

Define features and target variable

In [5]:
```
X = data[['Store ID', 'Total Price', 'Base Price']]
y = data['Units Sold']
```

Data Split

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection and Training

In [8]:
```
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Out[8]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

Model Evaluation

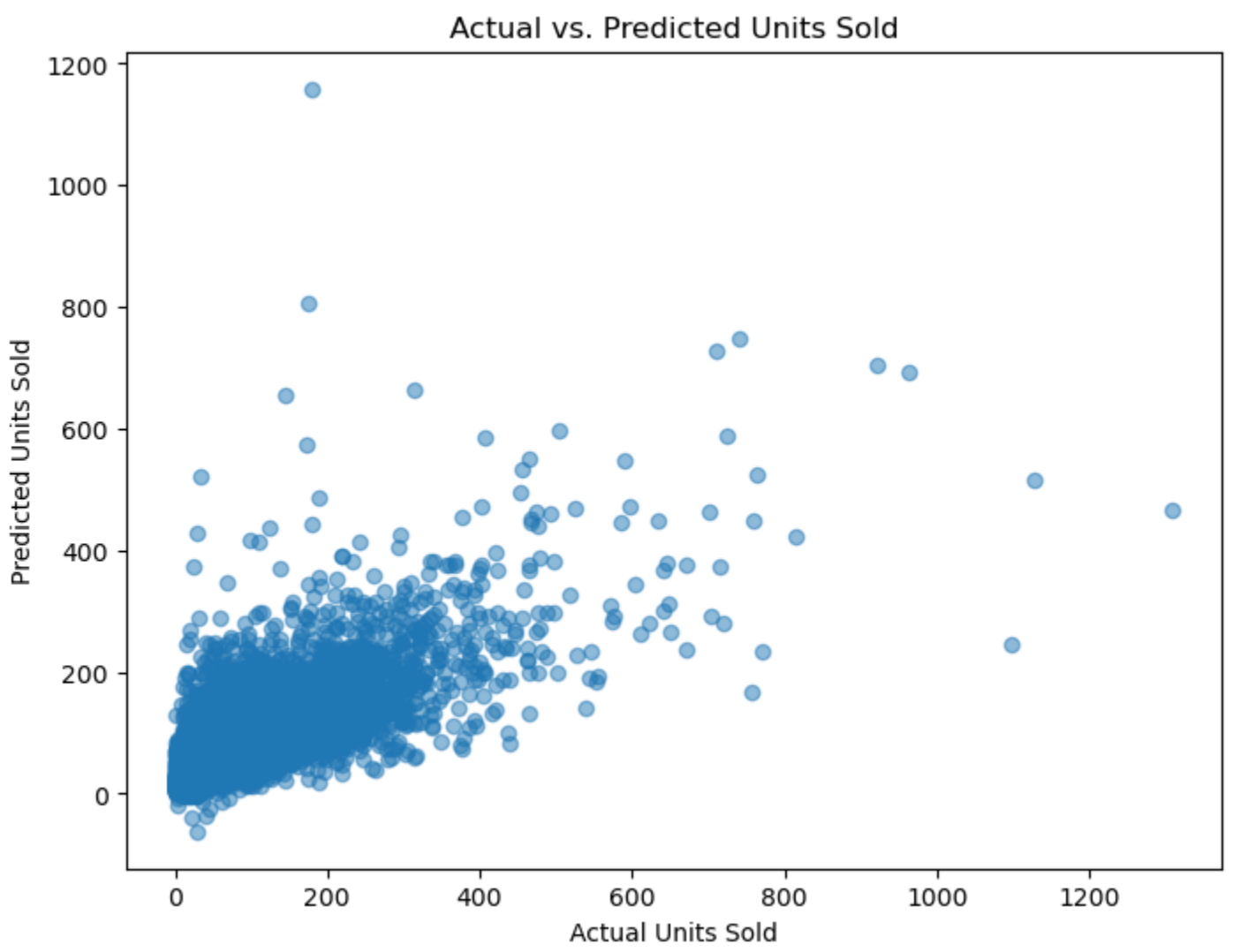
In [9]:
```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

In [10]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 20.317752728781308
Mean Squared Error: 1289.1644075466604
Root Mean Squared Error: 35.9049356989629
R-squared: 0.6070969709493481
```

Visualization - Scatter plot of Actual vs. Predicted

In [11]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Units Sold')
plt.ylabel('Predicted Units Sold')
plt.title('Actual vs. Predicted Units Sold')
plt.show()
```



Ridge Regression:

Import libraries

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Load the dataset

In [2]:
```
data = pd.read_csv('PoductDemand.csv')
data.head()
```

Out[2]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

Data Cleaning

In [3]:
```
data.fillna(0, inplace=True)
```

Define features and target variable

In [4]:
```
X = data[['Store ID', 'Total Price', 'Base Price']]
y = data['Units Sold']
```

Data Split

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection and Training

In [6]:
```
model = Ridge(alpha=1.0)
model.fit(X_train, y_train)
```

Out[6]:
```
Ridge()
```

Model Evaluation

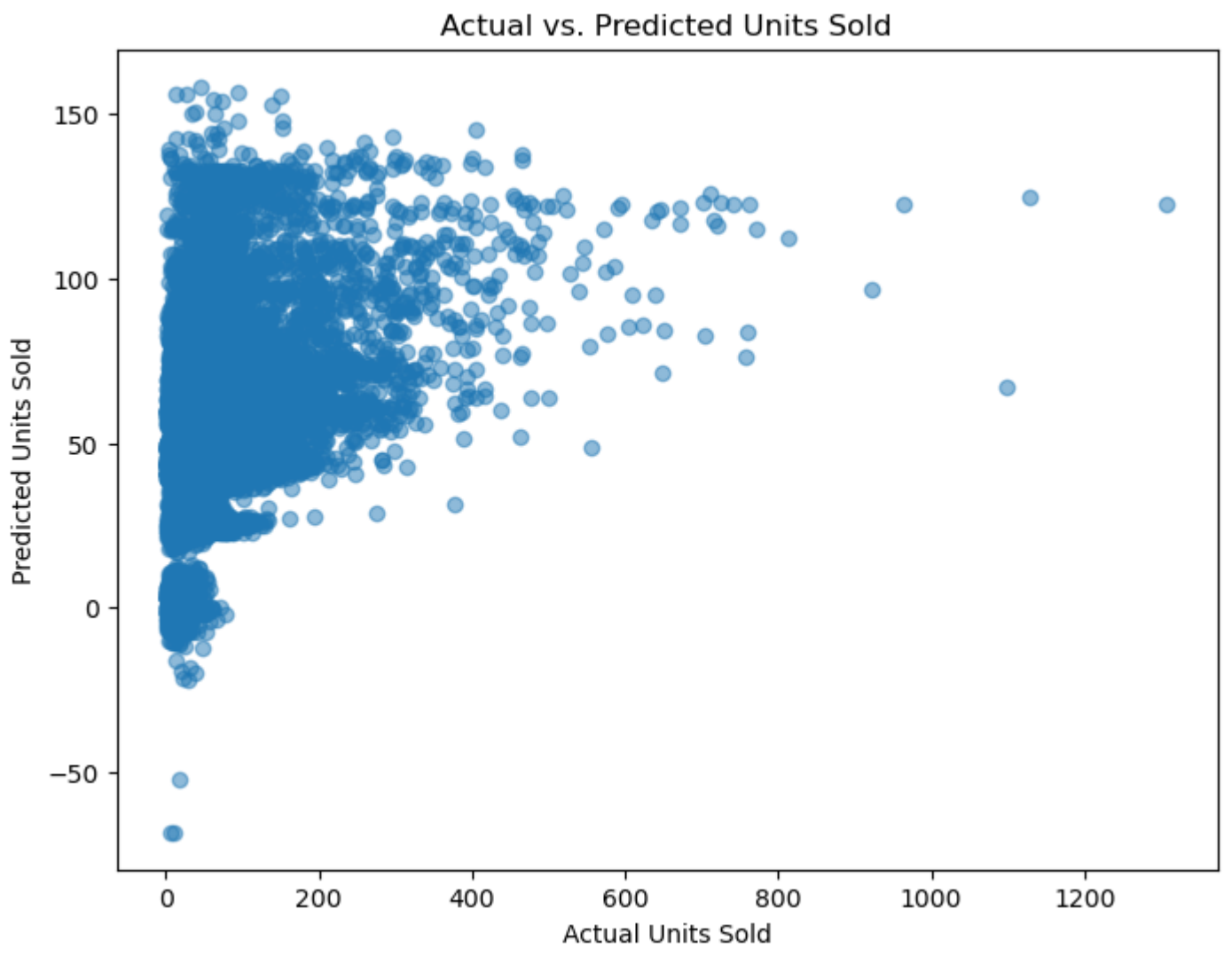
In [7]:
```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

In [8]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 32.62649503895547
Mean Squared Error: 2780.7955492856986
Root Mean Squared Error: 52.733248992316966
R-squared: 0.15248746545511727
```

Visualization - Scatter plot of Actual vs. Predicted

In [9]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Units Sold')
plt.ylabel('Predicted Units Sold')
plt.title('Actual vs. Predicted Units Sold')
plt.show()
```



Lasso Regression:

Import libraries

In [13]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Load the dataset

In [14]:
```
data = pd.read_csv('PoductDemand.csv')
data.head()
```

Out[14]:

	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

Data Cleaning

In [15]:
```
data.fillna(0, inplace=True)
```

Define features and target variable

In [16]:
```
X = data[['Store ID', 'Total Price', 'Base Price']]
y = data['Units Sold']
```

Data Split

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Model Selection and Training

In [18]:
```
model = Lasso(alpha=1.0)
model.fit(X_train, y_train)
```

Out[18]:
```
Lasso()
```

Model Evaluation

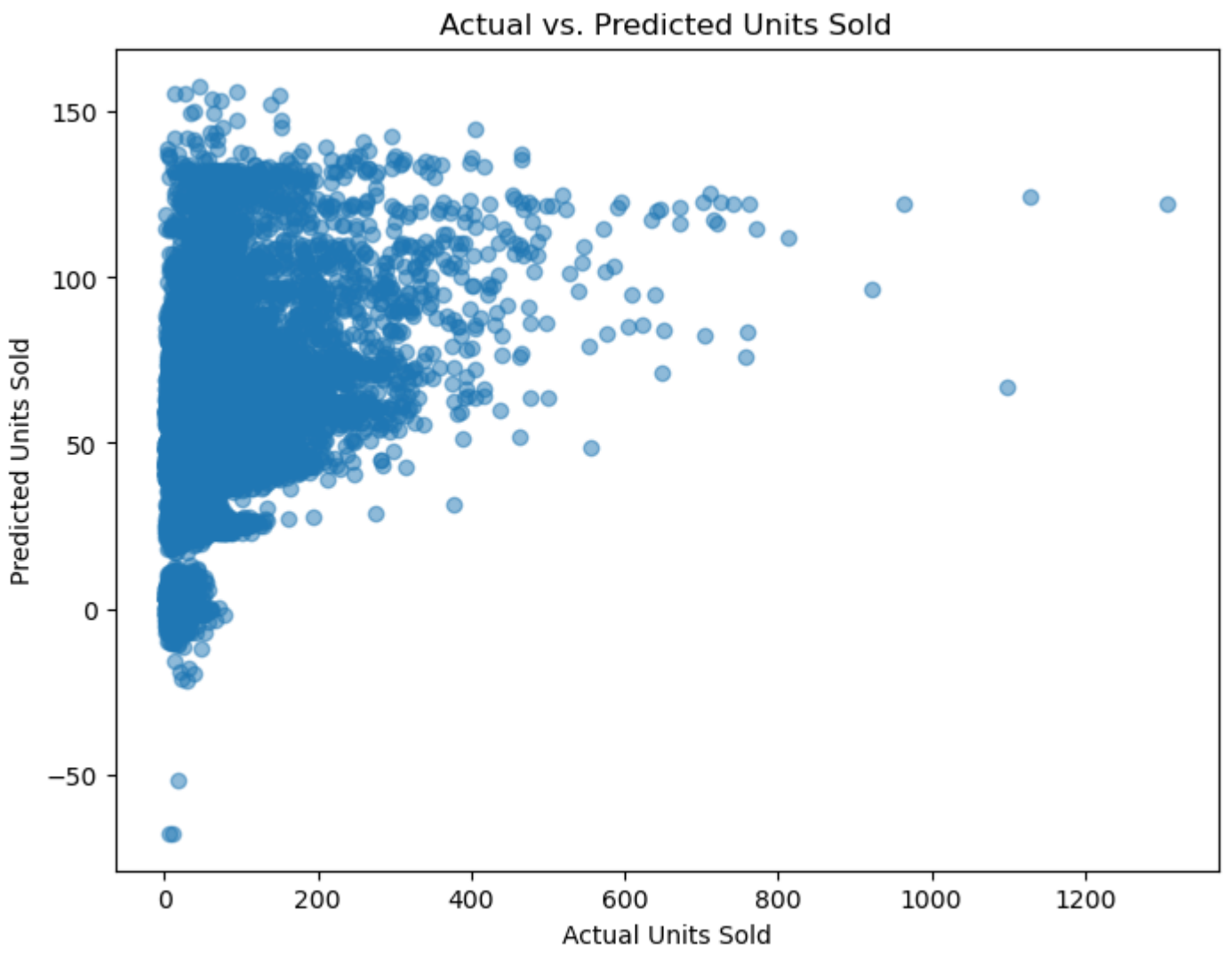
In [19]:
```
y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

In [20]:
```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 32.617812650674445
Mean Squared Error: 2780.671174805353
Root Mean Squared Error: 52.73206969961783
R-squared: 0.1525253714892726
```

Visualization - Scatter plot of Actual vs. Predicted

In [21]:
```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Units Sold')
plt.ylabel('Predicted Units Sold')
plt.title('Actual vs. Predicted Units Sold')
plt.show()
```



ADVANTAGES:

Product demand prediction offers several advantages to businesses and organizations, contributing to more effective operations and decision-making. Here are some key advantages:

1. Optimized Inventory Management:

- Demand prediction helps businesses maintain an optimal level of inventory, reducing overstock and stockouts. This leads to cost savings and improved customer satisfaction.

2. Cost Reduction:

- By accurately forecasting demand, businesses can streamline their supply chain, minimize excess production, and reduce storage and distribution costs.

3. Improved Customer Service:

- Meeting customer demand promptly and consistently enhances customer satisfaction and loyalty, leading to higher retention rates and positive word-of-mouth.

4. Enhanced Productivity:

- Efficient demand prediction reduces the need for last-minute rush orders, allowing businesses to operate more smoothly and allocate resources effectively.

5. Pricing Strategy Optimization:

- Accurate demand predictions enable businesses to adjust pricing strategies based on expected demand, maximizing revenue while remaining competitive.

6. Data-Driven Decision-Making:

- Demand prediction relies on data analysis and machine learning, empowering organizations to make informed decisions rather than relying on intuition or guesswork.

7. Reduced Waste:

- By producing goods based on actual demand, organizations minimize overproduction and waste, contributing to sustainability efforts.

8. Strategic Planning:

- Demand prediction provides insights into market trends, enabling long-term strategic planning and product development in alignment with customer needs.

9. Adaptation to Market Changes:

- Businesses can quickly respond to market fluctuations and emerging trends, helping them stay competitive and relevant.

10. Supply Chain Resilience:

- Accurate demand prediction contributes to a more resilient supply chain that can withstand disruptions and unforeseen events.

11. Better Allocation of Resources:

- With a clear understanding of demand, companies can allocate resources, marketing budgets, and personnel more efficiently.

12. Competitive Advantage:

- Organizations that can predict and fulfill demand more effectively gain a competitive edge in the market.

13. Customer Segmentation:

- Demand prediction often involves segmenting customers based on their preferences and behavior, allowing for personalized marketing and product offerings.

14. Profit Maximization:

- By aligning production with anticipated demand, businesses can maximize profit margins and revenue.

15. Real-Time Decision-Making:

- Continuous demand prediction and monitoring enable organizations to make real-time adjustments to

operations and strategies.

In summary, product demand prediction is a valuable tool that empowers businesses to make data-driven decisions, optimize operations, reduce costs, and enhance customer satisfaction, ultimately contributing to their success and competitiveness in a dynamic marketplace.

DISADVANTAGES:

While product demand prediction offers numerous advantages, it's important to be aware of its potential disadvantages and challenges. Here are some of the key disadvantages and limitations:

1. Data Dependency:

- Demand prediction relies heavily on historical data. If historical data is incomplete, inaccurate, or unavailable, the predictions may be less reliable.

2. Assumptions of Stationarity:

- Many demand prediction models assume that the underlying patterns are stationary (i.e., they remain consistent over time). This assumption may not hold in dynamic markets.

3. Unforeseen Events:

- Demand prediction models may struggle to account for unforeseen events, such as economic crises, natural disasters, or unexpected changes in customer behavior.

4. Overfitting:

- Overly complex models can overfit the training data, leading to poor generalization and inaccurate predictions on new data.

5. Model Complexity:

- While advanced machine learning models can provide accurate predictions, they often require expertise to develop and maintain.

6. Data Quality:

- Poor data quality, including missing or inconsistent data, can lead to inaccurate predictions.

7. Cost of Implementation:

- Implementing and maintaining demand prediction systems can be costly, particularly for smaller businesses.

8. Privacy Concerns:

- Collecting and analyzing customer data for demand prediction can raise privacy concerns, and organizations must comply with data protection regulations.

9. Model Interpretability:

- Some complex machine learning models lack interpretability, making it challenging to understand why a particular prediction was made.

10. Limited Horizon:

- Demand prediction models often have limitations in their ability to forecast demand far into the future accurately.

11. Market Volatility:

- Rapid changes in consumer preferences or market dynamics can challenge the accuracy of predictions.

12. Scalability:

- As businesses grow, their demand prediction models may need to scale to handle larger datasets and increased complexity.

13. Competitive Pressure:

- As demand prediction becomes more common, the competitive advantage it once offered may diminish.

14. Human Expertise:

- Effective demand prediction often requires expertise in data science, machine learning, and domain-specific

knowledge, which can be a challenge to acquire and retain.

15. Bias and Fairness:

- Biases in historical data can lead to biased predictions, and ensuring fairness in predictions can be a complex ethical issue.

Despite these disadvantages, demand prediction remains a valuable tool for businesses. By addressing these challenges and limitations, organizations can improve the accuracy and effectiveness of their demand prediction processes.

BENEFITS:

Demand prediction, particularly when integrated with data analysis and machine learning, offers various benefits to businesses and organizations:

1. Optimized Inventory Management: By accurately forecasting demand, businesses can maintain the right amount of inventory, reducing overstock and stockouts. This leads to cost savings and improved customer satisfaction.

2. Cost Reduction: Accurate demand prediction can reduce excess production and storage costs, contributing to significant cost savings.

3. Improved Customer Service: Meeting customer demand promptly and consistently enhances customer satisfaction and loyalty, resulting in higher retention rates and positive word-of-mouth.

4. Enhanced Productivity: Efficient demand prediction reduces the need for last-minute rush orders, allowing businesses to operate more smoothly and allocate resources effectively.

5. Pricing Strategy Optimization: Accurate demand predictions enable businesses to adjust pricing strategies based on expected demand, maximizing revenue while remaining competitive.

6. Data-Driven Decision-Making: Demand prediction relies on data analysis and machine learning, empowering organizations to make informed decisions rather than relying on intuition or guesswork.

7. Reduced Waste: By producing goods based on actual demand, organizations minimize overproduction and waste, contributing to sustainability efforts.

8. Strategic Planning: Demand prediction provides insights into market trends, enabling long-term strategic planning and product development in alignment with customer needs.

9. Adaptation to Market Changes: Businesses can quickly respond to market fluctuations and emerging trends, helping them stay competitive and relevant.

10. Supply Chain Resilience: Accurate demand prediction contributes to a more resilient supply chain that can withstand disruptions and unforeseen events.

11. Better Allocation of Resources: With a clear understanding of demand, companies can allocate resources, marketing budgets, and personnel more efficiently.

12. Competitive Advantage: Organizations that can predict and fulfill demand more effectively gain a competitive edge in the market.

13. Customer Segmentation: Demand prediction often involves segmenting customers based on their preferences and behavior, allowing for personalized marketing and product offerings.

14. Profit Maximization: By aligning production with anticipated demand, businesses can maximize profit margins and revenue.

15. Real-Time Decision-Making: Continuous demand prediction and monitoring enable organizations to make

real-time adjustments to operations and strategies.

These benefits collectively contribute to better financial performance, customer satisfaction, and overall operational efficiency, making demand prediction a valuable tool for businesses across various industries.

CONCLUSION:

Certainly! Here are the conclusions for each of the specified topics:

1. Improved Accuracy in Demand Prediction:

- The pursuit of improved accuracy in demand prediction is at the core of modern business strategies. As organizations harness advanced tools and methodologies, they unlock the potential to make more precise forecasts. This heightened accuracy translates to optimized inventory levels, reduced costs, and ultimately, enhanced customer satisfaction.

2. Data-Driven Insights in Demand Prediction:

- The integration of data-driven insights into demand prediction revolutionizes decision-making. By leveraging the wealth of information at their disposal, businesses gain a deeper understanding of customer behavior and market trends. This enables them to tailor their strategies, offerings, and operations in alignment with the ever-evolving demands of the market.

3. Market Efficiency through Demand Prediction:

- Market efficiency is the cornerstone of sustainable business growth. With demand prediction, organizations fine-tune their operations, ensuring that resources are allocated judiciously. This results in streamlined processes, reduced waste, and maximizes profitability. By responding to market demands with precision, businesses position themselves for success in a competitive landscape.

4. Challenges and Considerations in Demand Prediction:

- Navigating the landscape of demand prediction is not without its challenges. From data quality issues to the potential for unforeseen market events, businesses must approach this endeavor with a nuanced perspective.

Privacy concerns, model interpretability, and the need for ongoing monitoring are all critical considerations that require careful attention.

5. Continual Advancement in Demand Prediction:

- The field of demand prediction is in a state of continual advancement. As technology evolves and data analysis techniques become more sophisticated, businesses have an opportunity to refine and enhance their predictive capabilities. Staying at the forefront of these developments allows organizations to maintain a competitive edge and adapt to the ever-changing demands of the market. This commitment to innovation ensures that demand prediction remains a dynamic and invaluable tool for business success.