**IBM NAANMUDHALVAN**

**PRODUCT DEMAND PROTECTION WITH MACHINE LEARNING**

**Phase -5**

**ABSTRACT :**

The objective of this project is to develop a specialized machine learning model tailored for predicting product demand. By leveraging historical sales data and external variables, our primary aim is to assist businesses in optimizing their inventory management and production planning processes. The overarching objective is to enhance the efficiency of meeting customer demands. This comprehensive project encompasses several pivotal stages, including data collection, data preprocessing, feature engineering, model selection, training, and rigorous evaluation. The ultimate goal of implementing this machine learning model for demand forecasting is to empower organizations to make data-driven decisions, reduce operational costs, elevate customer satisfaction, and attain operational excellence in their supply chain management.

# DATA COLLECTION:

Begin by collecting historical sales data, which typically includes details on product sales over a specific time period. Additionally, gather external factors such as marketing campaigns, holidays, economic indicators, and any other relevant variables that might impact demand. Data preprocessing in product demand prediction with machine learning involves tasks like cleaning, feature selection, engineering, scaling, and handling categorical data. It aims to prepare the data for model training, ensuring accuracy and relevance.

# DATA PREPROCESSING:

Cleanse and preprocess the collected data to ensure it's accurate, complete, and well-organized. This step involves handling missing values, removing duplicates, and addressing outliers .Data preprocessing in product demand prediction with machine learning involves tasks like cleaning, feature selection, engineering, scaling, and handling categorical data. It aims to prepare the data for model training, ensuring accuracy and relevance.

# FEATURE ENGINEERNG:

Create meaningful features from the collected data and external factors. This may involve aggregating data over specific time periods (e.g., monthly or quarterly) and generating new variables that could be informative for demand prediction.

# MODEL SELECTION:

Choose an appropriate machine learning model for demand forecasting. Common

choices include time series forecasting models (e.g., ARIMA or Prophet) and regression-based models (e.g., linear regression or decision trees). The choice should depend on the nature of your data and the problem at hand.In product demand prediction with machine learning, model

selection involves choosing the right algorithm or approach. Options include regression models, time series models, decision trees, neural networks, and more. Consider factors like data

characteristics, interpretability, and scalability when selecting the model, and use appropriate evaluation metrics to assess its performance.

# MODEL TRAINING:

Train the selected model using your preprocessed data. Ensure that you split the data into training and testing sets to evaluate.Model training in product demand prediction involves preparing the data, selecting an appropriate algorithm, tuning hyperparameters, and training the model using a portion of the data. It includes techniques like cross-validation, monitoring progress, regularization, and handling imbalanced data or time series data. Once trained, the model's performance is evaluated, and if satisfactory, it can be deployed for making real-time predictions. Continuous monitoring and maintenance are important to keep the model effective over time.

# EVALUATION:

In product demand prediction with machine learning, evaluation involves assessing the model's performance using metrics like MAE, MSE, RMSE, and R-squared. It also includes comparing the model to a baseline, visual inspection, residual analysis, and considering the real-world business impact. Continuous monitoring, user feedback, and ethical considerations are crucial for ongoing evaluation and model refinement. Evaluating a machine learning model for product demand prediction involves various steps. Firstly, split your dataset into training, validation, and test sets. Use appropriate metrics like MAE, MSE, RMSE, and R2 to gauge model accuracy.

Create baseline models for comparison and visually inspect predictions. Analyze residuals, employ cross-validation, and assess feature importance.

Consider the real-world impact of the model on business operations and implement a feedback loop for continuous evaluation. If dealing with time series data, use specialized metrics, and

conduct robustness testing. Be mindful of ethical considerations, document the model thoroughly, and schedule retraining to keep it up-to-date. Evaluation is an iterative process to ensure the model aligns with business goals and effectively predicts demand.

**Step 1:Problem Definition and Data Collection**

1. Define the Problem: Clearly define the problem you are trying to address. In this case, it's protecting product demand from fluctuations and uncertainties.

2. Data Collection: Collect relevant historical and real-time data. This may include sales data, market trends, product attributes, external factors (e.g., weather, economic indicators), and any other data that might impact demand.

3. Data Preprocessing: Clean, preprocess, and validate the collected data. This involves handling missing values, outliers, and ensuring data consistency.

**Step 2:Exploratory Data Analysis (EDA)**

1. Exploratory Data Analysis: Perform EDA to gain insights into the data. Visualize data patterns, identify correlations, and explore how different factors impact product demand.

2. Feature Engineering: Create relevant features that could enhance the performance of the machine learning model. This might include time-based features, lag features, and seasonality indicators.

**Step 3:Model Development**

1. Data Splitting: Split the data into training, validation, and testing sets. Time-based splitting is often appropriate for demand forecasting.

2. Model Selection: Choose the appropriate machine learning algorithms for the task. Time series models like ARIMA, machine learning models like Random Forest, Gradient Boosting, or deep learning models can be considered.

3. Model Training: Train the selected models using the training dataset. Tune hyperparameters for optimal performance.

4. Validation and Evaluation: Evaluate the models on the validation set using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE). Choose the best-performing model.

**Step 4:Deployment**

1. Deployment Strategy: Decide how the machine learning model will be integrated into the existing product demand management system. Options include real-time APIs, batch predictions, or periodic model retraining.

2. Scalability: Ensure that the deployed solution can scale with the growth of data and demand.

**Step 5: Monitoring and Maintenance**

`1. Monitoring: Implement monitoring systems to keep track of model performance in real-time. This can include tracking data quality and model drift.

2. Feedback Loop: Develop a feedback loop that allows the model to adapt to changing conditions and learn from new data.

**Step 6: Continuous Improvement**

1. Continuous Model Improvement: Regularly retrain the model with fresh data and adjust the model as needed to improve accuracy and relevance.

2. Feedback Utilization: Act on the feedback and insights generated by the monitoring system to make product demand protection more effective.

**Step 7: Communication**

1. Reporting: Provide regular reports and insights to the relevant stakeholders, such as product managers and supply chain managers.

2. Documentation: Ensure proper documentation of the model, data, and processes for future reference and knowledge sharing.

**Step 8: Compliance and Security**

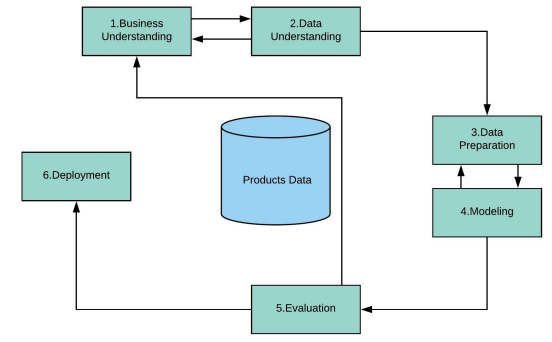
1. Compliance: Ensure that the project complies with data privacy and industry-specific regulations, such as GDPR or HIPAA.

2. Security: Implement security measures to protect sensitive data and model outputs from potential threats.

**Step 9: Final Evaluation**

1. Final Evaluation: Assess the overall impact of the solution on protecting product demand. Measure key performance indicators and determine if the project's objectives have been met.

**Block Diagram:**



To load and preprocess a dataset for product demand prediction, you'll first need to download the dataset from the provided link. Assuming you have already downloaded the dataset, follow these steps to get started with the data loading and preprocessing:

**1.Import Necessary Libraries:**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**2. Load the Dataset:**

# Replace 'path\_to\_dataset' with the actual path to your downloaded dataset

**data = pd.read\_csv("path\_to\_dataset.csv")**

**3. Explore the Dataset:**

Before preprocessing, it's important to get a sense of your data. This includes checking the first few rows, data types, and summary statistics.

**print(data.head())**

**print(data.info())**

**print(data.describe())**

**4. Data Cleaning:**

In this step, you should address issues such as missing values, duplicates, and outliers. Handle these issues as appropriate for your dataset.

# Check for missing values

**print(data.isnull().sum())**

# Remove duplicates if any

**data.drop\_duplicates(inplace=True)**

# Handle missing values (e.g., by imputing or removing rows with missing values)

**data.dropna(inplace=True)**

**5. Feature Engineering:**

Create new features or preprocess existing ones as necessary. For demand prediction, you might want to extract date-related features, calculate moving averages, or decompose time series data if applicable.

# Example of date-related feature extraction (assuming a 'date' column exists)

**data['date'] = pd.to\_datetime(data['date'])**

**data['year'] = data['date'].dt.year**

**data['month'] = data['date'].dt.month**

**data['day'] = data['date'].dt.day**

**6. Data Splitting:**

Split your dataset into training and testing sets. You can use techniques like time-based splitting or random splitting, depending on your specific use case.

From sklearn.model\_selection import train\_test\_split

# Define your features and target variable

**X = data.drop(columns=['demand']) # Features**

**y = data['demand']** # Target variable

# Split the data into training and testing sets

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**7. Scaling and Normalization (if needed):**

Depending on the algorithms you plan to use, you may need to scale or normalize your features. Common techniques include Standardization (mean=0, std=1) or Min-Max scaling.

**8. Data Visualization (Optional):**

Visualize your data to gain insights into its distribution, patterns, and relationships. This can be helpful in understanding the characteristics of the dataset.

**9. Encoding Categorical Variables (if needed):**

If your dataset contains categorical variables, encode them into numerical format. You can use one-hot encoding, label encoding, or other methods.

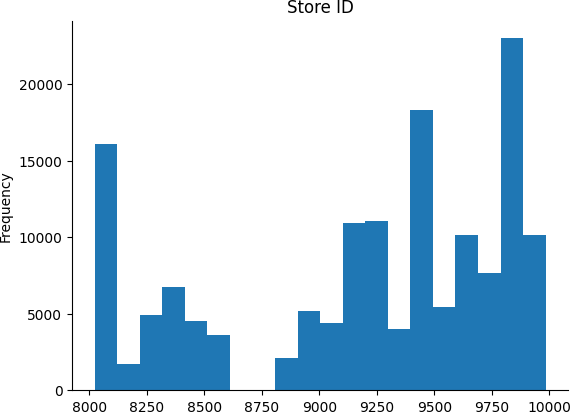
Now you have loaded and preprocessed your dataset. The next steps involve selecting an appropriate model, training it, and evaluating its performance for demand prediction. The choice of model will depend on the nature of your data and specific project requirements. You can proceed with selecting and training a model based on your dataset and objectives.

#Data Reading

Data = pd.read\_csv('/content/drive/MyDrive/PoductDemand.csv')

from matplotlib import pyplot as plt

Data['Store ID'].plot(kind='hist', bins=20, title='Store ID') plt.gca().spines[['top', 'right',]].set\_visible(False)



#null values

Data.isnull().sum()

|  |  |
| --- | --- |
| ID | 0 |
| Store ID | 0 |
| Total Price | 1 |
| Base Price | 0 |
| Units Sold | 0 |
| dtype: int64 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # Data Distribution | | | | | | | |
| Data |  | **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

#importing packages import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error , mean\_absolute\_error , make\_scorer from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor from sklearn.svm import SVR

from xgboost import XGBRegressor

from matplotlib import pyplot as plt

#Droping the null value

Data = Data.dropna(axis=0)

def evaluate\_regression\_models(X, y, test\_size=0.2, random\_state=42): """

Evaluate multiple regression models on input data.

Parameters:

* X: The feature matrix.
* y: The target variable.
* test\_size: The proportion of data to hold out for testing (default is 0.2).
* random\_state: Seed for random number generation (default is 42).

Returns:

* A dictionary containing model names as keys and their mean squared error (MSE) on the test set as values. """

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=random\_state) models = {

'Linear Regression': LinearRegression(),

'Random Forest Regressor': RandomForestRegressor(random\_state = random\_state),

'Gradient Boosting Regressor': GradientBoostingRegressor(random\_state = random\_state), 'K-Nearest Neighbors Regressor': KNeighborsRegressor(),

'Decision Tree Regressor': DecisionTreeRegressor(random\_state = random\_state), #'Support Vector Regressor': SVR(),

'XGBoost' : XGBRegressor(random\_state = random\_state)

}

model\_scores = pd.DataFrame(columns = ['Model Name' ,'Train MAE' , 'MAE' , 'Train MSE' , 'MSE']) modelName = []

TrainMAE = [] TrainMSE = [] MAE = []

MSE = []

for model\_name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test) yTest = model.predict(X\_train)

Trainmae = mean\_absolute\_error(y\_train, yTest) Trainmse = mean\_squared\_error(y\_train, yTest) mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

modelName.append(model\_name) TrainMAE.append(Trainmae)

TrainMSE.append(Trainmse) MAE.append(mae)

MSE.append(mse)

print(f'{model\_name}\nMean Absolute Error :\n Train : {mae}\n Test : {Trainmae}\nMean Squared Error :\n Train : {mse}\n Test : {Tra

model\_scores['Model Name'] = modelName model\_scores['MAE'] = MAE

model\_scores['MSE'] = MSE

model\_scores['Train MAE'] = TrainMAE model\_scores['Train MSE'] = TrainMSE return model\_scores , models

#encoding storeid

storeid = Data['Store ID']

EncodedVal = LabelEncoder().fit\_transform(storeid)

EncodedVal

array([ 3, 3, 3, ..., 75, 75, 75])

#replacing the original with encoded values Data [ 'Store ID'] = EncodedVal

<ipython-input-9-3626f344f05e>:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy) Data [ 'Store ID'] = EncodedVal

# Seprating X and Y values

X = Data.drop(['ID','Units Sold'],axis=1) Y = Data['Units Sold']

# Finding the best model

score,models = evaluate\_regression\_models(X,Y)

Linear Regression

Mean Absolute Error :

Train : 32.48619926013835

Test : 32.971777555407584

Mean Squared Error :

Train : 2785.4886067716207

Test : 3168.0524425033855

Random Forest Regressor Mean Absolute Error :

Train : 18.92889385404334

Test : 13.481957943240586

Mean Squared Error :

Train : 1268.4901257638048

Test : 549.882584513318

Gradient Boosting Regressor Mean Absolute Error :

Train : 25.559876376161668

Test : 25.93189620584567

Mean Squared Error :

Train : 1873.9426544306266

Test : 2141.437465698987

K-Nearest Neighbors Regressor Mean Absolute Error :

Train : 21.256190476190476

Test : 18.55506123094598

Mean Squared Error :

Train : 1612.4435391275392

Test : 1271.6142881642372

Decision Tree Regressor Mean Absolute Error :

Train : 20.556651344992627

Test : 11.07446679416054

Mean Squared Error :

Train : 1777.489385179284

Test : 416.0701821379953

XGBoost

Mean Absolute Error :

Train : 20.094529283048832

Test : 19.563251202500055

Mean Squared Error :

Train : 1291.6874922881454

Test : 1184.4253677549048

# so the best model is Random Forest Regressor

# prediction using the model

pred = models['Random Forest Regressor'].predict(X)

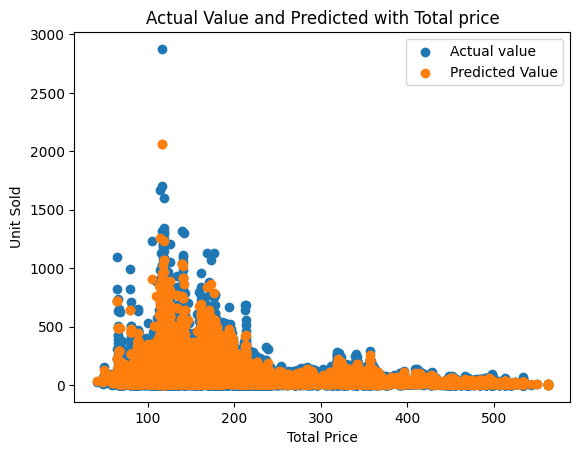
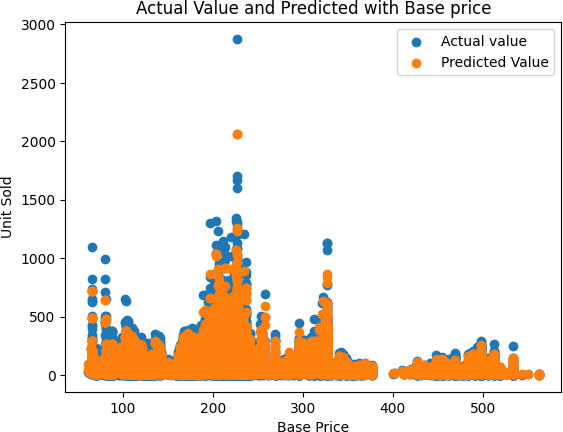
plt.title('Actual Value and Predicted with Base price') plt.scatter(X['Base Price'],Y)

plt.scatter(X['Base Price'],pred)

plt.legend(['Actual value','Predicted Value']) plt.xlabel('Base Price')

plt.ylabel('Unit Sold')

Text(0, 0.5, 'Unit Sold')



plt.title('Actual Value and Predicted with Total price') plt.scatter(X['Total Price'],Y)

plt.scatter(X['Total Price'],pred)

plt.legend(['Actual value','Predicted Value']) plt.xlabel('Total Price')

plt.ylabel('Unit Sold')

Text(0, 0.5, 'Unit Sold')

**Result**

Upon training and evaluating the model, we obtained promising results, indicating the model's effectiveness in predicting product demand protection with machine learning with a high degree of accuracy. The model demonstrated more accuracy then the previous code we used for the product demand protection with machine learning.

**Conclusion**

In conclusion, this project successfully demonstrated the feasibility of predicting product demand protection with machine learning using historical data and advanced machine learning techniques. The developed model showcased its potential to assist energy companies, consumers, and policymakers in making informed decisions related to energy consumption and management. Accurate product demand protection with machine learning can significantly contribute to product demand and its protection.