PRODUCT DEMAND ANALYSIS

PROBLEM STATEMENT:

Create a machine learning model that forecasts product demand based on historical sales and external factors, helping businesses optimize inventory management and production planning to meet customer needs efficiently.

**PHASES OF DEVELOPMENT:**

**PHASE 1:** PROBLEM DEFINITION AND DESIGN THINKING

PROBLEM DEFINITION:

The problem is to create a machine learning model that forecasts product demand based on historical sales data and external factors. The goal is to help businesses optimize inventory management and production planning to efficiently meet customer needs. This project involves data collection, data preprocessing, feature engineering, model selection, training and evaluation.

DESIGN THINKING:

1. **Data Collection**: Collect historical sales data and external factors that influence demand, such as marketing campaigns, holidays, economic indicators, etc.
2. **Data Preprocessing**: Clean and preprocess the data, handle missing values, and convert categorical features into numerical representations.
3. **Feature Engineering**: Create additional features that capture seasonal patterns, trends, and external influences on product demand.
4. **Model Selection**: Choose suitable regression algorithms (e.g., Linear Regression, Random Forest, XG Boost) for demand forecasting.
5. **Model Training**: Train the selected model using the preprocessed data.
6. **Evaluation**: Evaluate the model's performance using appropriate regression metrics (e.g., Mean Absolute Error, Root Mean Squared Error).

**PHASE 2:** INNOVATION

By using incorporating time series forecasting techniques like ARIMA or Prophet for capture temporal patterns in demand data.

DATA SET LINK: <https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning>

**PHASE 3:** DEVELPOMENT PART 1

Begin building the product demand prediction model by loading and preprocessing the dataset.

**PHASE 4**: DEVELOPMENT PART 2

Continue building the product demand prediction model by feature engineering, model training, and evaluation.

**PHASE 5:** PROJECT DOCUMENTATION & SUBMISSION

Document the product demand prediction project and prepare it for submission.

**DESCRIBTION OF THE DATASET**:

The dataset contain the index of store id, product id, total price, base price and unsold units for 4 columns and contain the row 150151 products.

DATA PREPROCESSING STEPS:

1. Import all the necessary libraries
2. Import the dataset and read it as CSV file.
3. Check the null values
4. Fill the NA values

ANALYSIS TECHNIQUES APPLIED:

For analysis time series forecasting techniques like ARIMA model is used to capture temporal patterns in demand data. In the ARIMA model PACF and TSEA plots are used.

After analysis with ARIMA model perform the encoding categorical data, feature engineering (selection), splitting the data for model training, model training and evaluation.

In the model training there are several model like ‘Linear Regression, Random forest, Support vector machine and Gradient Boosting’ are used in predicting the model.

In model the ‘Mean Squared Error, R-squared’ are used to evaluating the model.

PROGRAM:

# Import necessary libraries

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

# Data Collection

# Assuming your dataset is named 'product\_demand\_data.csv' and located in the same directory as your Python script

data = pd.read\_csv('/content/PoductDemand.csv')

# Data Preprocessing

# Handling Missing Values (if any)

data.fillna(0, inplace=True)

data.isnull().sum()

# Data Transformation

# No categorical variables to encode in this case

# Split Data

X = data[features] # Features

y = data[target] # Target variable

# Split the data into training and testing sets (70-30 split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Data Standardization (optional, but often necessary for many machine learning algorithms)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Encode categorical data

import numpy as np

import pandas as pd

# One-hot encode the categorical data

encoded\_df = pd.get\_dummies(data)

# Print the encoded DataFrame

print(encoded\_df)

# Feature Selection

features = ['ID', 'Store ID', 'Total Price', 'Base Price'] # Features

target = 'Units Sold' # Target variable

# Histograms and Box Plots

import matplotlib.pyplot as plt

# Histograms

data[features].hist(bins=20, figsize=(12, 10))

plt.suptitle("Histograms of Features")

plt.show()

# Box Plots

data[features].plot(kind='box', vert=False, figsize=(12, 6))

plt.title("Box Plots of Features")

plt.show()

# Correlation Matrix

import seaborn as sns

correlation\_matrix = data[features].corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title("Correlation Matrix")

plt.show()

# Pair Plot

sns.pairplot(data[features])

plt.suptitle("Pair Plot of Features")

plt.show()

# Target Variable Distribution

plt.figure(figsize=(8, 6))

sns.histplot(data[target], bins=20, kde=True)

plt.title("Distribution of Target Variable")

plt.xlabel(target)

plt.ylabel("Frequency")

plt.show()

# Feature vs. Target Plots

for feature in features:

plt.figure(figsize=(8, 6))

sns.scatterplot(x=data[feature], y=data[target])

plt.title(f"{feature} vs. {target}")

plt.xlabel(feature)

plt.ylabel(target)

plt.show()

# Box Plot of Target Variable Grouped by Categorical Feature

categorical\_feature = 'Store ID' # Example categorical feature

plt.figure(figsize=(10, 6))

sns.boxplot(x=categorical\_feature, y=target, data=data)

plt.title(f"Box Plot of {target} Grouped by {categorical\_feature}")

plt.xlabel(categorical\_feature)

plt.ylabel(target)

plt.xticks(rotation=45)

plt.show()

**# MODEL SELECTION**

# Import necessary libraries for different algorithms

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.metrics import mean\_squared\_error, r2\_score

# Initialize models

linear\_reg = LinearRegression()

random\_forest = RandomForestRegressor(random\_state=42)

svm = SVR()

gradient\_boosting = GradientBoostingRegressor(random\_state=42)

# Train and predict using each algorithm

models = [linear\_reg, random\_forest, svm, gradient\_boosting]

model\_names = ['Linear Regression', 'Random Forest', 'Support Vector Machine', 'Gradient Boosting']

for model, name in zip(models, model\_names):

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

predictions = model.predict(X\_test)

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

r2 = r2\_score(y\_test, predictions)

print(f"Model: {name}")

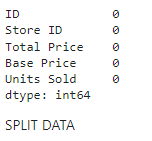
print(f"Mean Squared Error: {mse:.2f}")

print(f"R-squared: {r2:.2f}")

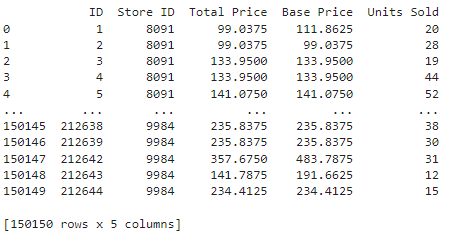
print("-" \* 30)

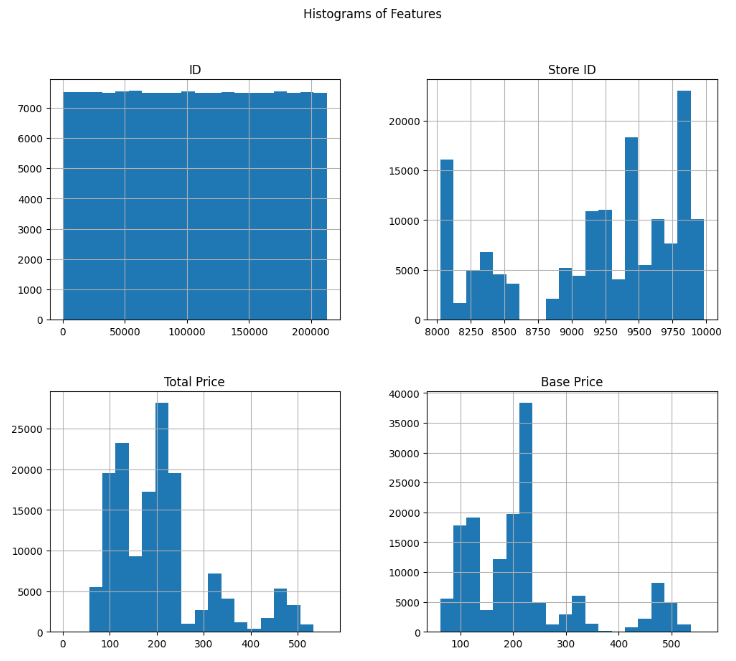
**OUTPUT:**

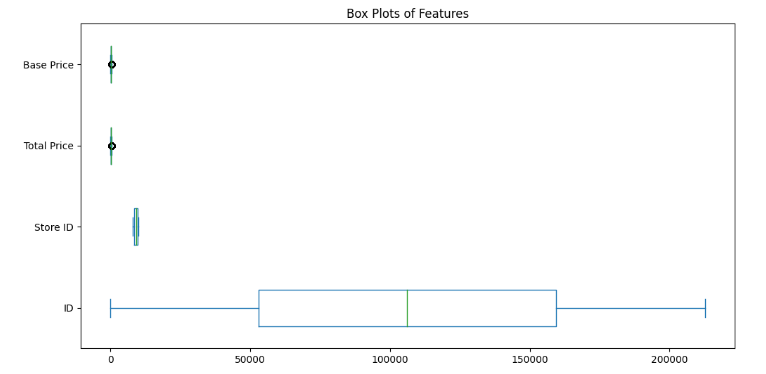
Data Preprocessing

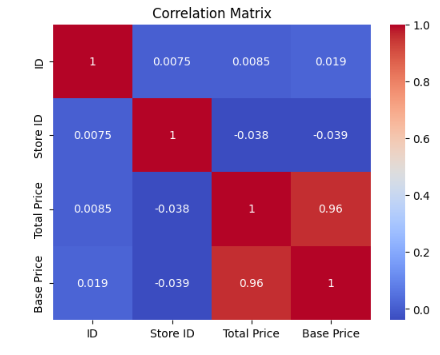


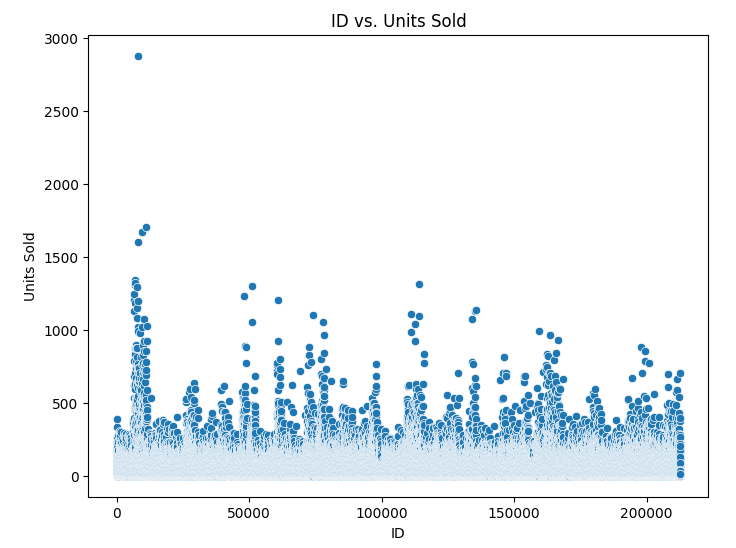
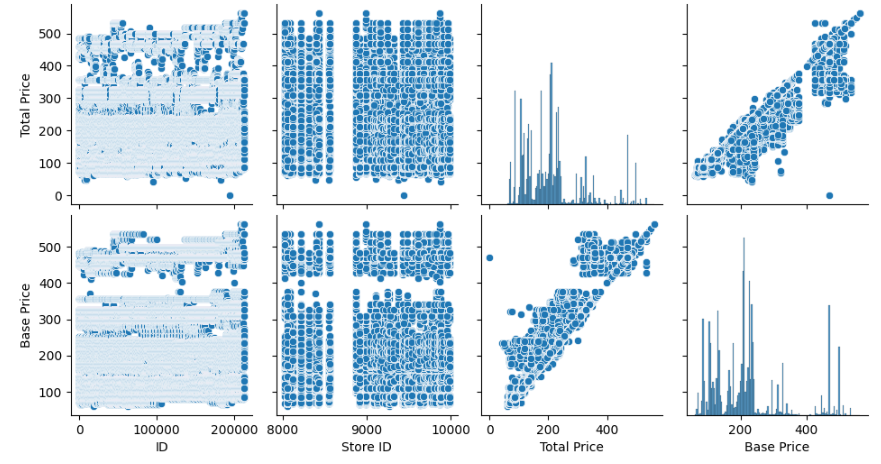
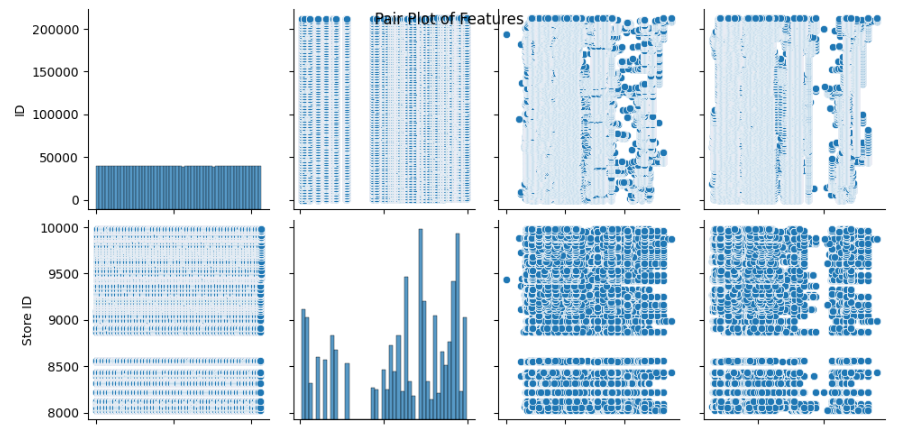
Encoding categorical data

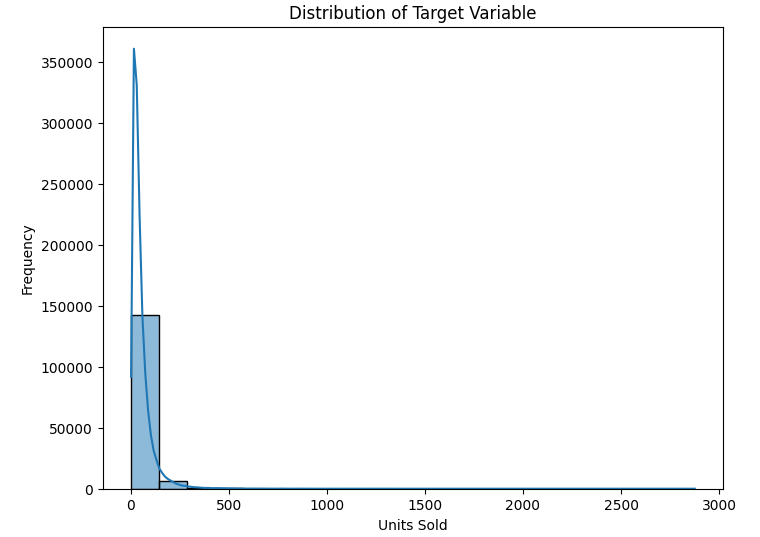


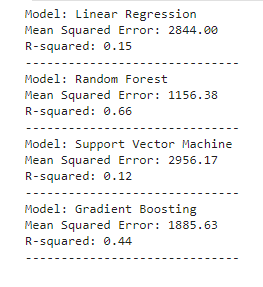
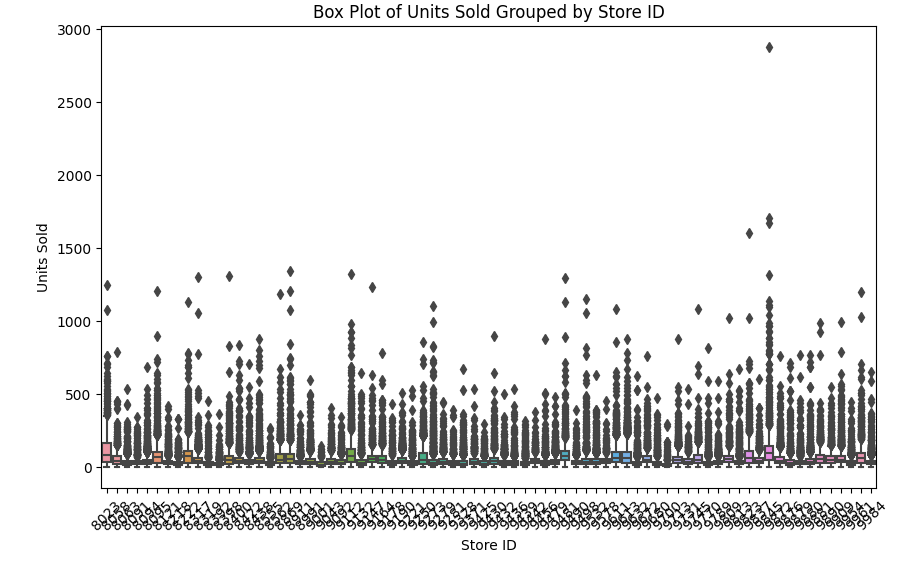
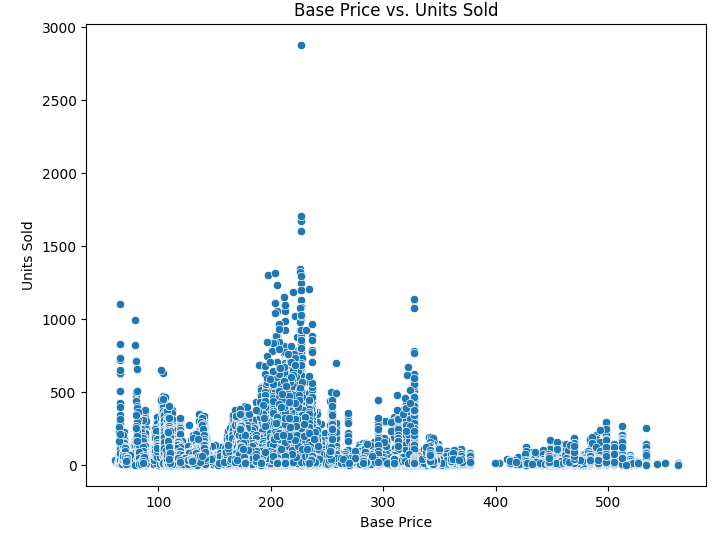
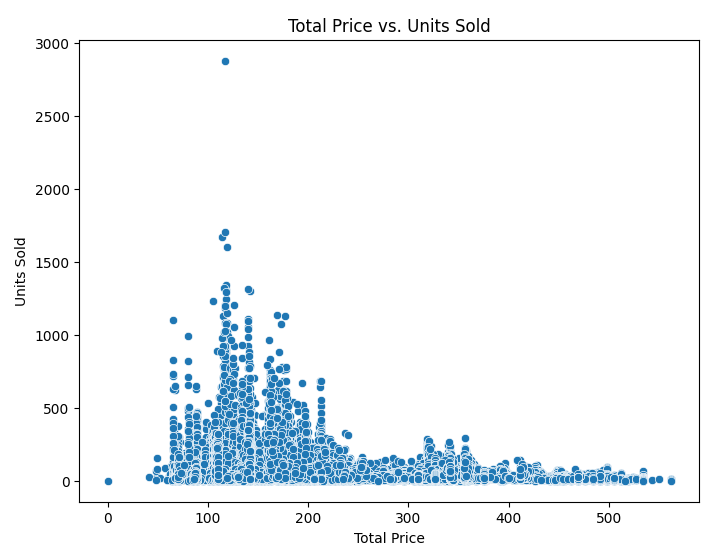
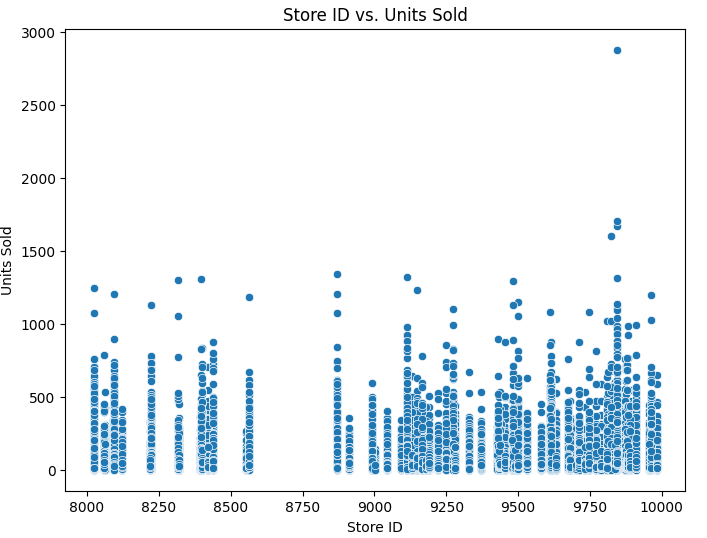


****

****



****



**ARIMA MODEL:**

**LINK:**

**https://colab.research.google.com/drive/1gNQWTceGFUHsXDgdAMRgesCE5hFBYAAt?usp=sharing**

**CONCLUSION:**

Random Forest and Gradient Boosting model typically perform well in a variety of datasets due to their ability to capture complex patterns in the data.

Support Vector Machine (SVM) might perform well if the dataset has high dimensionality and complex relationships, although it might require fine-tuning of hyper parameter for optimal results.

Linear Regression provides a basic understanding of the relationships between variables but might not capture intricate patterns present in the data.