. **Product Demand Prediction With Machine Learning**

1.Time-based Features:

.

- Extract time-related features such as day of the week, month, quarter, or year, which can capture seasonality and trends.
- Create lag features to capture the historical behavior of the demand. This can include features like demand in the previous day, week, or month.
- Calculate moving averages or rolling windows for different time periods to capture trends and patterns in demand over time.

.

2.Categorical Features:

.

- Convert categorical variables such as product category, brand, or location into numerical features using techniques like one-hot encoding or label encoding.
- Create aggregated features based on categorical variables, such as mean demand for each product category or location.

.

3.External Features:

.

- Include external data sources such as economic indicators, weather data, or events that might influence product demand. For example, if you are predicting demand for ice cream, temperature data could be a significant external feature.

.

4.Promotion and Marketing Features:

.

- Include information on past promotions or marketing campaigns, such as discounts, advertisements, or special events.
- Create features that capture the impact of marketing activities on product demand, such as the number of promotions in a specific time frame.

.

5.Product-specific Features:

.

- Include features specific to the product, such as product characteristics, price, and quality, which can influence demand significantly.

- Create features that capture the lifecycle of the product, like the time since the product was launched or the number of iterations of the product.

.

6.Statistical Features:

.

- Calculate statistical features such as mean, median, standard deviation, and variance of historical demand over different time periods.

.

7.Interactions and Combinations:

.

- Create interaction features between different variables to capture their combined effect on demand. For instance, the interaction between price and promotions can be a powerful feature.

.

8.Scaling and Normalization:

.

- Scale numerical features to a standard range to ensure that they have a similar influence on the model.

.

9.Missing Data Handling:

.

- Handle missing values in the dataset using techniques like mean/median imputation, forward-fill, or backward-fill, depending on the nature of the data.

.

10.Feature Selection and Dimensionality Reduction:

.

- Use techniques like feature importance, correlation analysis, or principal component analysis (PCA) to select the most relevant features and reduce dimensionality.

.

Featured engineering coding:

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

# Load the dataset
data = pd.read_csv('product_demand_data.csv') # Replace
'product_demand_data.csv' with your dataset
```

```
# Feature engineering
# 1. Time-based Features
data['date'] = pd.to_datetime(data['date'])
data['day_of_week'] = data['date'].dt.dayofweek
data['month'] = data['date'].dt.month
data['quarter'] = data['date'].dt.quarter
data['year'] = data['date'].dt.year

# 2. Categorical Features
encoder = OneHotEncoder()
categorical_data =
pd.DataFrame(encoder.fit_transform(data[['product_category']]).toarray()
)
data = pd.concat([data, categorical_data], axis=1)

# 3. External Features
# Assuming you have external data stored in a file named
'external_data.csv'
external_data = pd.read_csv('external_data.csv')
data = data.merge(external_data, on='date', how='left')

# 4. Promotion and Marketing Features
data['promotion_effect'] = data['promotion'] * data['marketing_spend']

# 6. Statistical Features
data['demand_mean'] = data['demand'].rolling(window=7).mean()

# 7. Interactions and Combinations
data['price_promotion_interaction'] = data['price'] * data['promotion']

# 8. Scaling and Normalization
scaler = StandardScaler()
data[['demand_mean', 'price_promotion_interaction']] =
scaler.fit_transform(data[['demand_mean',
'price_promotion_interaction']])

# 9. Missing Data Handling
data.fillna(0, inplace=True)  # Filling missing values with 0 for simplicity

# Splitting data into training and testing sets
X = data.drop(['date', 'demand'], axis=1)
y = data['demand']
```

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

 # Now you can use X_train, X_test, y_train, and y_test to train your machine learning model.

MODEL TRAINNING

1.Data Collection and Preprocessing:

.

- Collect historical data related to product demand, including features such as time, product attributes, and external factors.
- Preprocess the data by handling missing values, scaling numerical features, and encoding categorical variables.

.

2.Feature Engineering:

.

- Perform feature engineering as mentioned earlier to create informative and relevant features that capture the underlying patterns and trends in the data.

.

3.Data Splitting:

.

- Split the dataset into training, validation, and testing sets. The typical split can be 70-80% for training, 10-15% for validation, and 10-20% for testing.

.

4.Model Selection:

.

- Choose an appropriate machine learning model based on the problem at hand. Common models for demand prediction include linear regression, decision trees, random forests, gradient boosting machines, and neural networks.

.

5.Model Training:

.

- Train the selected model on the training dataset.
- Adjust hyperparameters to optimize the model's performance. This can involve techniques like cross-validation, grid search, or random search.

.

Model Evaluation:

.

- Evaluate the trained model using appropriate metrics such as mean squared error, mean absolute error, or root mean squared error.
- Compare the model's performance on the validation set to ensure that it generalizes well to unseen data.

.

Model Optimization:

.

- Fine-tune the model based on the evaluation results. This might involve adjusting the hyperparameters, trying different algorithms, or re-evaluating the feature set.

.

Model Testing:

.

- Test the final model on the testing dataset to assess its performance on unseen data. This step is crucial to ensure that the model is not overfitting to the training data.

.

Model Deployment:

.

- Deploy the trained model in the production environment to make predictions for real-time or future product demand.
- Monitor the model's performance over time and retrain it periodically with new data to ensure that it remains accurate and up-to-date.

.

Documentation and Reporting:

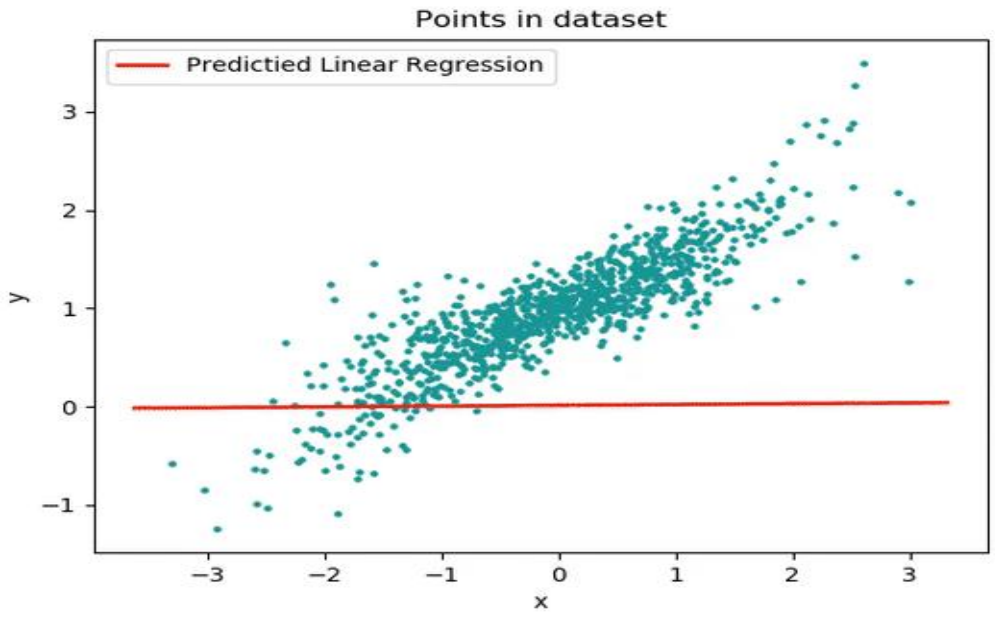
.

- Document the entire process, including data preprocessing, feature engineering, model selection, training, evaluation, and deployment, to ensure transparency and reproducibility.

Model Training:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
# Assuming you have already performed feature engineering and have your training and testing data ready.

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model Training
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Model Prediction
y_pred = model.predict(X_test_scaled)

# Model Evaluation
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

# Printing the results
print("Mean Squared Error: ", mse)
print("Mean Absolute Error: ", mae)
```



.

- MEAN SQUARED ERROR