# Product Demand Prediction with Machine Learnings

TEAM MEMBER

911721104061:MOHAMMED JAHIR USAIN .V

Phase-1 Documentation submission

# Project: Product Demand Prediction with Machine Learnings

OBJECTIVES:

Uninterrupted supply of products/services. Sales target setting and evaluating sales performance. Optimization of prices according to market fluctuations and inflation.

Phase 1: Problem Definition and Design Thinking

# Data Collection:

Collect historical sales data and external factors that influence demand, such as marketing campaigns, holidays, economic indicators, etc.

**Datalink**:([https://www.kaggle.com/datasets/chakradharmattapalli/produ](https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning) [ct-demand-prediction-with-machine-learning](https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning))

# Data preprocessing

Data preprocessing is a crucial step in building machine learning models for product demand prediction. It involves cleaning, transforming, and organizing your data to make it suitable for training and testing your model. Here are the key steps in data preprocessing for product demand prediction:

Data Collection:

Gather historical data on product demand. This data should include information such as product attributes, time/date of sales, quantity sold, and any other relevant features.

Data Cleaning:

Handle missing data: Identify and decide how to handle missing values in your dataset. You can choose to remove rows with missing values, impute them with a suitable strategy (e.g., mean, median, or more advanced imputation methods), or treat them as a separate category.

Remove duplicates: Check for and remove any duplicate records in your dataset.

Feature Engineering:

Create new features: Generate additional features that might be relevant for demand prediction. For example, you can extract time-based features like day of the week, month, or season.

Encode categorical variables: Convert categorical variables (e.g., product categories) into numerical format using techniques like one-hot encoding or label encoding.

Data Transformation:

Scale features: Standardize or normalize numerical features to have a consistent scale, which can help some machine learning algorithms perform better.

Log transformations: If your target variable (demand) is highly skewed, applying a log transformation can make it more normally distributed, which is often beneficial for regression models.

Time Series Data:

If your data involves time series, consider handling it appropriately. This may include resampling to a consistent time interval, creating lag features, and dealing with seasonality and trends.

Train-Test Split:

Split your data into training and testing sets. The training set is used to train your machine learning model, while the testing set is used to evaluate its performance.

Outlier Detection and Handling:

Identify and handle outliers in your data. Outliers can have a significant impact on the performance of predictive models. You can choose to remove them or transform them using robust methods.

Feature Selection (Optional):

If you have many features, you may want to perform feature selection to choose the most relevant ones. Techniques like feature importance from tree-based models or dimensionality reduction methods can help.

Data Scaling and Normalization (Optional):

Depending on the machine learning algorithm you plan to use, scaling and normalization of features may be necessary. Some algorithms, like SVM and k-Nearest Neighbors, are sensitive to feature scales.

Data Pipeline:

Set up a data preprocessing pipeline to automate these steps. This ensures that the same preprocessing is applied to both the training and testing data consistently.

Validation and Cross-Validation:

Use cross-validation techniques to assess the performance of your model and prevent overfitting. This involves splitting your data into multiple folds for training and testing.

Save Processed Data:

After preprocessing, save your cleaned and transformed data to avoid having to repeat these steps every time you want to train or evaluate your model.

# FUTURE ENGINEER’S:

**Handle missing data:** Identify and decide how to handle missing values in your dataset. You can choose to remove rows with missing values, impute them with a suitable strategy (e.g., mean, median, or more advanced imputation methods), or treat them as a separate category.

Remove duplicates: Check for and remove any duplicate records in your dataset.

Create new features: Generate additional features that might be relevant for demand prediction. For example, you can extract time-based features like day of the week, month, or season.

Encode categorical variables: Convert categorical variables (e.g., product categories) into numerical format using techniques like one-hot encoding or label encoding.

1. MODEL SELECTION:
   1. decisiontree 2.linear regression
2. MODEL TRAINING:

Code :

import pandas as pd import numpy as np import plotly.express as px import seaborn as sns

import matplotlib.pyplot as plt

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

df=pd.read\_csv('PoductDemand.csv')

# EVALUATION:

**Code:**

import pandas as pd import numpy as np import plotly.express as px import seaborn as sns

import matplotlib.pyplot as plt

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

df=pd.read\_csv('PoductDemand.csv') print(df.head())

print(df.info()) print(df.describe()) print(df.isnull().sum()) df1=df.dropna() print(df1) print(df.corr())

x=df[["Total Price", "Base Price"]] y=df["Units Sold"]

x\_train, x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=42) model=DecisionTreeRegressor(random\_state = 0) print(model.fit(x\_train,y\_train))

feature=np.array([[133.00,140.00]]) print(model.predict(feature)) plt.plot(x,y)

plt.xlabel('x')

plt.ylabel('y') plt.show()

output:

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

1. ID 150150 non-null int64
2. Store ID 150150 non-null int64
3. Total Price 150149 non-null float64
4. Base Price 150150 non-null float64
5. Units Sold 150150 non-null int64 dtypes: float64(2), int64(3)

memory usage: 5.7 MB None

ID Store ID Total Price Base Price Units Sold

count 150150.000000 150150.000000 150149.000000 150150.000000 150150.000000

mean 106271.555504 9199.422511 206.626751 219.425927 51.674206

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| std | 61386.037861 615.591445 103.308516 110.961712 60.207904 | | | | |
| min | 1.000000 8023.000000 41.325000 61.275000 1.000000 | | | | |
| 25% | 53111.250000 | 8562.000000 | 130.387500 | 133.237500 | 20.000000 |
| 50% | 106226.500000 | 9371.000000 | 198.075000 | 205.912500 | 35.000000 |
| 75% | 159452.750000 | 9731.000000 | 233.700000 | 234.412500 | 62.000000 |
| max | 212644.000000 | 9984.000000 | 562.162500 | 562.162500 | 2876.000000 |
| ID | 0 |  |  |  |  |

Store ID 0

Total Price 1

Base Price 0

Units Sold 0 dtype: int64

ID Store ID Total Price Base Price Units Sold

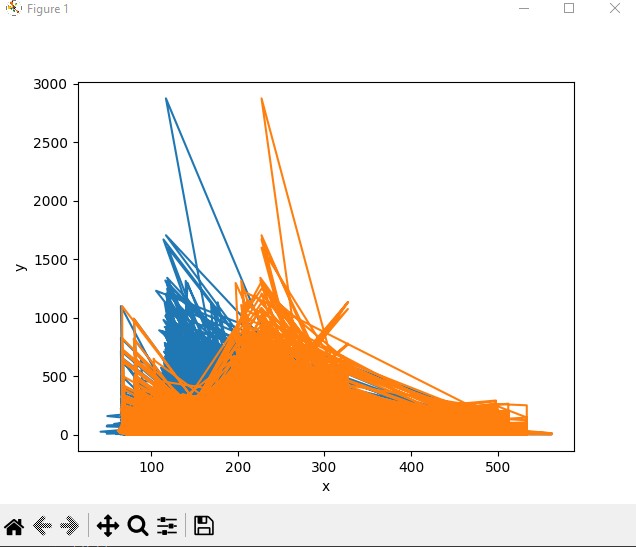
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 8091 | 99.0375 | | 111.8625 | | 20 | |
| 1 | 2 | 8091 | 99.0375 | | 99.0375 | | 28 | |
| 2 | 3 | 8091 | 133.9500 | | 133.9500 | | 19 | |
| 3 | 4 | 8091 | 133.9500 | | 133.9500 | | 44 | |
| 4 | 5 | 8091 | 141.0750 | | 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 |

[150149 rows x 5 columns]

ID Store ID Total Price Base Price Units Sold

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | 1.000000 0.007464 0.008473 0.018932 -0.010616 | | | |
| Store ID | 0.007464 1.000000 -0.038315 -0.038848 -0.004372 | | | |
| Total Price 0.008473 -0.038315 | | 1.000000 | 0.958885 | -0.235625 |
| Base Price 0.018932 -0.038848 | | 0.958885 | 1.000000 | -0.140032 |
| Units Sold -0.010616 -0.004372 | | -0.235625 | -0.140032 | 1.000000 |

DecisionTreeRegressor(random\_state=0)

Linear regression