**PRODUCT DEMAND PREDICTION WITH**

**MACHINE LEARNING**

Team member

712221121015: SUGASRI K

Phase 4 submission document

Title: Product Demand Prediction with Machine Learning

Phase 4: Development Part 2

Topic: Start to build the product demand prediction model by detail about feature engineering, model training and evaluation



PRODUCT DEMAND PREDICTION

INTRODUCTION:

Product demand prediction is the process of forecasting future customer demand for a product or service. It is a critical task for businesses of all sizes, as it helps them to make informed decisions about production, inventory, marketing, and pricing.

Machine learning(ML) is a type of artificial intelligence (AI) that allows computers to learn without being explicitly programmed. ML algorithms can be used to analyze historical data to identify patterns and trends, and then use this information to make predictions about the future.

For Example: Let's say the company's ML model predicts that demand for its flagship product, a smart speaker, will be 100,000 units during the holiday season. The company can use this information to make sure that it has enough inventory on hand to meet demand. The company can also use this information to tailor its marketing campaigns to target customers who are most likely to be interested in buying a smart speaker. Overall, ML-powered product demand prediction can help businesses to make better decisions, improve their efficiency, and increase their profitability.

***FEATURE ENGINEERING***

Feature engineering is the process of selecting and transforming relevant features from the raw data to improve the performance of ML models. In demand prediction for drugs on pharmacies. Feature engineering is a critical step in building predictive models for demand forecasting. Here are some key reasons why it matters:

**Enhanced Model Performance:**

Well-engineered features can capture underlying patterns and relationships in the data, leading to more accurate predictions. By selecting the right features and transforming them appropriately, you can extract valuable information that might be hidden in the raw data.

**Improved Interpretability:**

Feature engineering can make your models more interpretable. By creating meaningful features, you can gain insights into which factors are driving demand and how they impact your predictions. This knowledge is invaluable for making informed business decisions.

**Handling Non-linearity:**

Real-world demand data is often non-linear, and feature engineering allows you to transform variables to better fit the assumptions of your chosen machine learning algorithm. This can result in better model performance.

***some of the most important features are:***

***Time-based features:*** These features capture trends and patterns over time. Examples include day of the week, month, year, and holidays.

***Store-based features:*** These features capture pharmacy-specific characteristics. Examples include the location of the pharmacy, the size of the pharmacy, and the customer demographics.

In addition to these features, external factors such as economic conditions, weather conditions, and cultural event can also be taken into account.

**1. Time-based features**

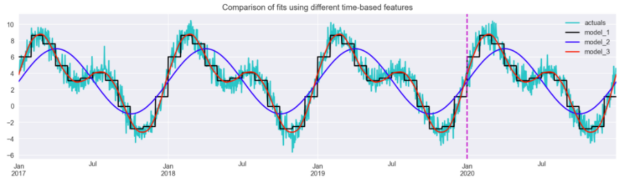
Time-based features are an important category of features that can be used in demand prediction for drugs on pharmacies. These features capture trends and patterns over time and can help the ML model to better understand the dynamics of the demand for drugs.

***EXAMPLES:***

Day of the week: Many drugs have different demand patterns depending on the day of the week. For example, pain relievers may be more in demand on weekdays, while sleep aids may be more in demand on weekends.

Month: Seasonal changes can have a significant impact on the demand for drugs. For example, allergy medications may be more in demand during the spring and summer months.

Year: Long-term trends can also have an impact on the demand for drugs. For example, the demand for certain drugs may increase or decrease over time due to changes in demographics or health trends.



In Python we can convert the data into time-based feature, create a lag feature, and rolling window using pandas , example :

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# Load the sales data

sales\_data = pd.read\_csv('sales\_data.csv')

# Convert the date column to a datetime object

sales\_data['date'] = pd.to\_datetime(sales\_data['date'])

# Create time-based features

sales\_data['day\_of\_week'] = sales\_data['date'].dt.dayofweek

sales\_data['day\_of\_month'] = sales\_data['date'].dt.day

sales\_data['month'] = sales\_data['date'].dt.month

sales\_data['quarter'] = sales\_data['date'].dt.quarter

sales\_data['year'] = sales\_data['date'].dt.year

sales\_data['day\_of\_year'] = sales\_data['date'].dt.dayofyear

sales\_data['week\_of\_year'] = sales\_data['date'].dt.weekofyear

# Create lag features

sales\_data['lag\_7'] = sales\_data['demand'].shift(7)

sales\_data['lag\_14'] = sales\_data['demand'].shift(14)

sales\_data['lag\_21'] = sales\_data['demand'].shift(21)

sales\_data['lag\_28'] = sales\_data['demand'].shift(28)

# Create rolling window features

sales\_data['rolling\_mean\_7'] = sales\_data['demand'].rolling(window=7).mean()

sales\_data['rolling\_mean\_14'] = sales\_data['demand'].rolling(window=14).mean()

sales\_data['rolling\_mean\_21'] = sales\_data['demand'].rolling(window=21).mean()

sales\_data['rolling\_mean\_28'] = sales\_data['demand'].rolling(window=28).mean()

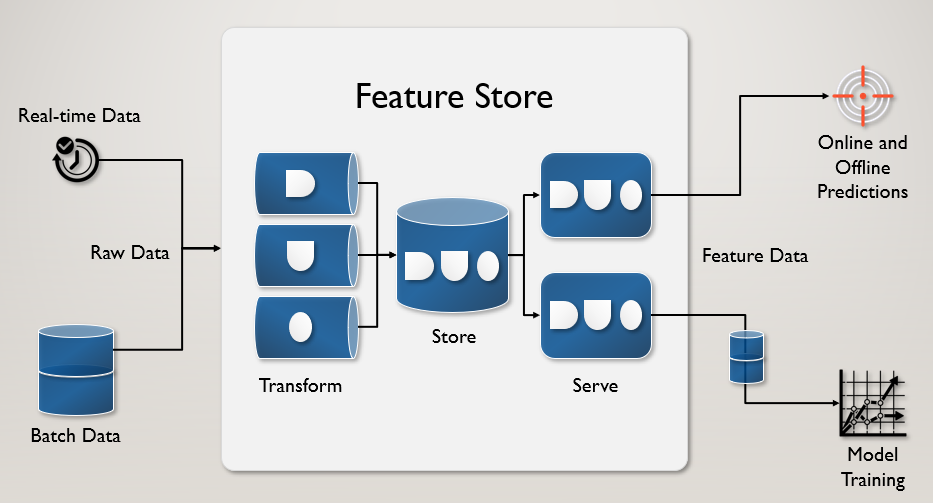
Here are some tips for incorporating time-based features in your ML model for demand prediction:

Use a variety of time granularities: Different time granularities, such as day, week, or month, may be relevant for different drugs. Experiment with different granularities to find the most useful features.

Use rolling windows: Instead of using static time-based features, consider using rolling windows to capture the recent trends in demand. For example, you could use the average demand over the past week as a feature.

Consider the time horizon: The time horizon for demand prediction can vary depending on the needs of the pharmacy. For example, some pharmacies may need to predict demand on a daily basis, while others may need to predict demand on a weekly or monthly basis. Make sure that the time-based features you use are appropriate for the time horizon of the prediction.

2. Store-based features



Store or pharmacy-based features are a type of feature engineering technique used in demand prediction for drugs on pharmacies using machine learning. This technique involves adding features related to the store or pharmacy where the drugs are sold, such as location, size, and the number of employees.

Store-based features can provide valuable insights into demand patterns that are specific to a particular store or location. By understanding the unique characteristics of each store, pharmacies can better forecast demand and optimize their inventory levels.

We can use geographic coordinates to augment feature sets involves gathering latitude and longitude data for each store's location. These coordinates can be leveraged to ascertain whether a store is situated in a suburban, downtown, or urban area. To convert latitude and longitude values into categories like "suburb," "downtown," or "rural," you can use geospatial data or external APIs that provide information about the location. In Python we can use geopy library to demonstrate how to do this based on proximity to predefined locations:

from geopy.distance import great\_circle

# Define coordinates for reference locations (e.g., downtown, suburb, rural)

# Example downtown,suburb,rural coordinates (latitude, longitude)

downtown\_coords = (40.7128, -74.0060)

suburb\_coords = (40.7459, -73.9804)

rural\_coords = (40.8256, -74.2207)

# Function to categorize a given latitude and longitude

def categorize\_location(latitude, longitude):

location\_coords = (latitude, longitude)

# Calculate distances to reference locations

downtown\_distance = great\_circle(location\_coords, downtown\_coords).miles

suburb\_distance = great\_circle(location\_coords, suburb\_coords).miles

rural\_distance = great\_circle(location\_coords, rural\_coords).miles

# Determine the category based on distances

if downtown\_distance < suburb\_distance and downtown\_distance < rural\_distance:

return "Downtown"

elif suburb\_distance < downtown\_distance and suburb\_distance < rural\_distance:

return "Suburb"

else:

return "Rural"

# Example latitude and longitude

latitude = 40.7128 # Example latitude

longitude = -74.0060 # Example longitude

# Categorize the location

location\_category = categorize\_location(latitude, longitude)

print(f"The location is categorized as: {location\_category}")

Example of how to retrieve population data for a city using the censusdata Python library:

import censusdata

# Define the geographic area (in this example, we'll use New York City)

state = 'NY'

county = 'New York'

place = 'New York city'

# Define the year for the population estimate (e.g., 2020)

year = 2020

# Retrieve population data for the specified location

pop\_data = censusdata.download(

'acs5', # Dataset identifier for American Community Survey 5-Year Estimates

year,

censusdata.censusgeo([('state', state), ('county', county), ('place', place)]),

['DP05\_0001E'], # Population estimate variable

)

# Extract the population estimate from the DataFrame

population\_estimate = pop\_data.iloc[0]['DP05\_0001E']

# Print the population estimate

print(f"Population estimate for {place}, {county}, {state} in {year}: {population\_estimate}")

Here are some tips to consider when creating store-based features:

Consider the granularity of the features: Store-based features can be very specific, such as the number of employees working at a particular store, or more general, such as the store's location. It's important to consider what level of granularity is appropriate for the specific problem you're trying to solve.

Use external data sources: Additional information about the store or location, such as demographic data, foot traffic, and nearby competitors, can provide valuable insights into demand patterns.

## Be mindful of privacy concerns: When working with store-based data, it's important to be mindful of privacy concerns and to ensure that sensitive information is properly protected.

## *MODEL TRAINING:*

## Product Demand Prediction:

A product company plans to offer discounts on its product during the upcoming holiday season. The company wants to find the price at which its product can be a better deal compared to its competitors. For this task, the company provided a dataset of past changes in sales based on price changes. You need to train a model that can predict the demand for the product in the market with different price segments.

The [**dataset**](https://raw.githubusercontent.com/amankharwal/Website-data/master/demand.csv) that we have for this task contains data about:

1. the product id;
2. store id;
3. total price at which product was sold;
4. base price at which product was sold;
5. Units sold (quantity demanded);

I hope you now understand what kind of problem statements you will get for the product demand prediction task. In the section below, I will walk you through predicting product demand with machine learning using Python.

## Product Demand Prediction using Python

Let’s start by importing the necessary Python libraries and the dataset we need for the task of product demand prediction:



1

import pandas as pd

2

import numpy as np

3

import plotly.express as px

4

import seaborn as sns

5

import matplotlib.pyplot as plt

6

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

7

from sklearn.tree import DecisionTreeRegressor

8

​

9

data = pd.read\_csv("https://raw.githubusercontent.com/amankharwal/Website-data/master/demand.csv")

10

data.head()

**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**

Now let’s have a look at whether this dataset contains any null values or not:



1

data.isnull().sum()

**ID 0**

**Store ID 0**

**Total Price 1**

**Base Price 0**

**Units Sold 0**

**dtype: int64**

So the dataset has only one missing value in the **Total Price** column, I will remove that entire row for now:



1

data = data.dropna()

Let us now analyze the relationship between the price and the demand for the product. Here I will use a [**scatter plot**](https://thecleverprogrammer.com/2020/12/20/scatter-plot-with-python/) to see how the demand for the product varies with the price change:



1

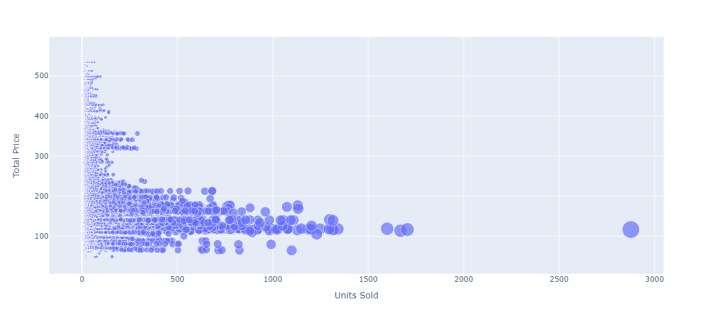
fig = px.scatter(data, x="Units Sold", y="Total Price",

2

size='Units Sold')

3

fig.show()



We can see that most of the data points show the sales of the product is increasing as the price is decreasing with some exceptions. Now let’s have a look at the correlation between the features of the dataset:



1

print(data.corr())

**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**



1

correlations = data.corr(method='pearson')

2

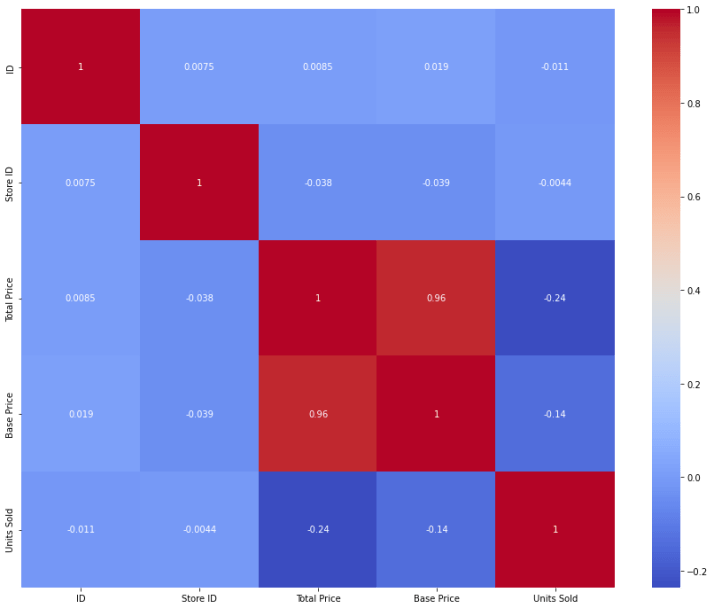
plt.figure(figsize=(15, 12))

3

sns.heatmap(correlations, cmap="coolwarm", annot=True)

4

plt.show()



## Product Demand Prediction Model

Now let’s move to the task of training a machine learning model to predict the demand for the product at different prices. I will choose the **Total Price** and the **Base Price** column as the features to train the model, and the **Units Sold** column as labels for the model:



1

x = data[["Total Price", "Base Price"]]

2

y = data["Units Sold"]

Now let’s split the data into training and test sets and use the decision tree regression algorithm to train our model:



1

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y,

2

test\_size=0.2,

3

random\_state=42)

4

from sklearn.tree import DecisionTreeRegressor

5

model = DecisionTreeRegressor()

6

model.fit(xtrain, ytrain)

Now let’s input the features **(Total Price, Base Price)** into the model and predict how much quantity can be demanded based on those values:



1

#features = [["Total Price", "Base Price"]]

2

features = np.array([[133.00, 140.00]])

3

model.predict(features)

**array([27.])**

Evaluation

Once the model has been selected, it needs to be trained on the historical sales data. This involves splitting the data into training and testing sets, and using the training set to fit the model to the data. Once the model has been trained, it can be used to predict the demand for drugs in the pharmacy for a given period of time.

To evaluate the performance of the model, we can use metrics such as mean absolute percentage error (MAPE), root mean squared error (RMSE), mean absolute error (MAE), or R-squared(R²). These metrics provide a measure of how well the model is able to predict the demand for drugs in the pharmacy.

To evaluate the impact of feature engineering, you should compare the performance of models with and without engineered features using the evaluation metrics mentioned earlier. In most cases, you will likely observe that models with feature engineering outperform those without. Lower MAPE, RMSE, and MAE values and higher R² values are indicative of improved predictive accuracy.

***CONCLUSION:***

**We**can train a machine learning model for the task of product demand prediction using Python. Price is one of the major factors that affect the demand for the product. If a product is not a necessity, only a few people buy the product even if the price increases. I hope you liked this article on product demand prediction with machine learning using Python. Feel free to ask your valuable questions in the comments section below.