**TIME SERIES FORECASTING FOR PRODUCT DEMAND PREDICTION**

**Step 1: Data Acquisition and Preprocessing**

***1.1 Dataset Information***

The dataset, labeled as dataset.csv, was sourced from [Kaggle](https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning).

It contains 4750 rows and 5 columns: Id, store id, total price, base price, and unit sold.

***1.2 Handling Missing Values***

A thorough examination of each column was performed to identify any missing or incomplete data.

Records with Null values in the "Total Price" column were removed to ensure data integrity and accuracy.

***1.3 Libraries Used***

Pandas for data manipulation

Installation command: pip install pandas

Numpy for numerical operations

Installation command: pip install numpy

Plotly for interactive plots

Installation command: pip install plotly

Seaborn for statistical data visualization

Installation command: pip install seaborn

Matplotlib for static plots

Installation command: pip install matplotlib

Scikit-learn (sklearn) for machine learning tasks

Installation command: pip install scikit-learn

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

***2.1 Analyzing Price-Demand Relationship***

A scatter plot was used to visualize how demand changes with price variations.

This helps in understanding the relationship between product price and demand.

***2.2 Correlations in the Dataset***

Correlation analysis was performed to identify relationships between different features and demand.

Understanding these correlations is crucial for predicting product demand accurately.

**Step 3: Time Series Forecasting**

***3.1 Importance of Time Series Forecasting***

**Time Series Forecasting** is a critical technique for making predictions based on historical data that is indexed in time order. In the context of product demand prediction, this means that we use past demand data to predict future demand patterns.

This is particularly crucial because demand for products tends to follow specific temporal patterns. For example, some products may have seasonal demand spikes (e.g., winter coats in winter), while others may have long-term trends (e.g., electronics becoming more popular over time). Understanding and capturing these patterns allows for more accurate predictions.

***3.2 Libraries Required for Time Series Forecasting***

To perform time series forecasting for product demand prediction, we'll be using two key libraries:

**Statsmodels**: This library provides a wide range of models and tools for time series analysis. It's particularly known for its implementation of the ARIMA (AutoRegressive Integrated Moving Average) model, which is a widely used method for time series forecasting.

**Installation command**: **pip install statsmodels**

**Elaboration**: The ARIMA model is capable of capturing different aspects of time series data, including trends, seasonality, and autocorrelation. By using ARIMA, we can model and predict demand patterns effectively.

**Prophet**: Prophet is an open-source forecasting tool developed by Facebook. It's designed to handle time series data with strong seasonal patterns, as well as data that may have missing or irregularly spaced observations.

**Installation command**: **pip install prophet**

**Elaboration**: Prophet is particularly useful for product demand prediction because it can handle data with daily or seasonal fluctuations. It also allows for the inclusion of additional external factors (holidays, promotions, etc.) that might affect demand.

Application to Product Demand Prediction:

In the context of predicting product demand, let's consider an example:

Imagine we're dealing with a retail business that sells winter jackets. Using time series forecasting, we can analyze past data of jacket sales over several years. By applying a model like ARIMA or Prophet, we can capture the seasonal patterns (higher sales in winter, lower in summer), trends (increasing or decreasing popularity of jackets over time), and any irregularities in the data.

This analysis helps us make accurate predictions about future demand for jackets. For instance, it can guide decisions on how much inventory to stock for the upcoming winter season. If the forecast predicts a particularly cold winter, the business might choose to stock more jackets than in a milder year.

**Step 4: Model Training for Demand Prediction**

***4.1 Selecting Features and Labels***

For training the machine learning model, 'Total Price' and 'Base Price' columns were chosen as features.

'Units Sold' column was selected as the labels for the model.

***4.2 Regression Metrics for Model Evaluation***

Mean Absolute Error (MAE), Mean Squared Error (MSE), or R-squared (R2) can be used to assess the model's performance.

In summary, time series forecasting is a powerful tool for making informed decisions about inventory management and production planning based on historical demand patterns. The choice of library (Statsmodels, Prophet, or others) will depend on the specific characteristics of the data and the desired level of detail in the forecasting model.

**LOADING AND PREPROCESSING**

**THE DATASET**

Introduction:

In this section of the code, we are loading and preparing the dataset for analysis. This involves importing necessary libraries, reading the dataset, exploring its contents, handling missing values, and visualizing relationships between variables.

Importing Libraries:

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

Explanation:

Pandas (as pd): This library provides data structures and functions for efficiently manipulating large datasets.

numpy (as np): NumPy is used for numerical operations and provides support for arrays and matrices.

plotly.express (as px): Plotly Express is a high-level interface for creating various types of visualizations.

seaborn (as sns): Seaborn is a statistical data visualization library based on Matplotlib, providing a high-level interface for creating informative and attractive statistical graphics.

matplotlib.pyplot (as plt): Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations.

train\_test\_split: This function from scikit-learn helps split the dataset into training and testing sets for model evaluation.

DecisionTreeRegressor: This is a regression algorithm from scikit-learn used for modelling.

Loading the dataset:

Code:

data = pd.read\_csv("demand.csv")

Explanation:

pd.read\_csv(): This function from Pandas reads a CSV file into a DataFrame. In this case, it's reading a file named "demand.csv" and assigning it to the variable data.

At this point, we have successfully imported the necessary libraries and loaded the dataset into a Pandas DataFrame named data. The dataset is now ready for further exploration and analysis.

Displaying the First Few Rows:

Code:

print("HEAD:\n")

print(data.head())

Explanation:

**data.head():** This function displays the first few rows (default is 5) of the DataFrame, providing an initial look at the dataset's structure and content.

Output:

Checking for Missing Values:

Code:

print("\ncontains any null values or not:\n")

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

Explanation:

**data.isnull():** This returns a DataFrame of boolean values indicating whether each element in data is missing or not.

**sum():** This function sums up the True values (i.e., the missing values) for each column.This section prints out the number of missing values for each column in the dataset.

Output:

Handling Missing Values:

Code:

data = data.dropna()

Explanation:

**data.dropna():** This function removes rows with any missing values from the DataFrame and assigns the modified DataFrame back to data.

This step is taken to ensure that the dataset is clean and ready for analysis, as some machine learning algorithms do not handle missing values.

Total code for preprocessing:

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

data = pd.read\_csv("demand.csv")

print("HEAD:\n")

print(data.head())

print("\ncontains any null values or not:\n")

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

data = data.dropna()

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

size='Units Sold')

fig = px.scatter(data, x="Units Sold", y="Total Price", size='Units Sold', color\_discrete\_sequence=['red'])

fig = px.scatter(data, x="Units Sold", y="Total Price", size='Units Sold', template='plotly\_dark')

# Change the color of the scatter points

fig.update\_traces(marker=dict(color='green'))

fig.show()

print("correlation between the features of the dataset:\n")

print(data.corr())

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

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

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

plt.show()

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

y = data["Units Sold"]

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

model = DecisionTreeRegressor()

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

train\_score = model.score(X\_train, y\_train)

test\_score = model.score(X\_test, y\_test)

print("\nR-squared scores for Train and Test:\n")

print(f"Training R-squared score: {train\_score}")

print(f"Testing R-squared score: {test\_score}")

Output:

**Predicting Units Sold with Decision Tree Regression**

**Introduction:**

The provided code implements a Decision Tree Regressor model to predict the number of units sold based on features like "Total Price" and "Base Price". This type of model is commonly used in regression tasks, where the goal is to predict a continuous target variable.

**Code Explanation:**

**Data Splitting**

The first step in the code involves splitting the data into training and testing sets. This is crucial to evaluate how well the model generalizes to unseen data. The data is divided into two sets:

X: Contains the features used for prediction.

y: Contains the target variable (number of units sold).

test\_size=0.2: Specifies that 20% of the data will be used for testing, and the remaining 80% for training.

random\_state=42: Ensures that the data is split in a reproducible manner.

**Code:**

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

**Model Training:**

Next, a Decision Tree Regressor model is created and trained using the training data:

DecisionTreeRegressor(): Initializes a Decision Tree Regressor model.

model.fit(X\_train, y\_train): Fits the model to the training data, allowing it to learn the relationships between the features and the target variable.

**Code:**

model = DecisionTreeRegressor()

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

**Model Evaluation:**

The code then calculates the R-squared scores to assess the model's performance:

model.score(X\_train, y\_train): Computes the R-squared score on the training data, indicating how well the model fits the training set.

model.score(X\_test, y\_test): Calculates the R-squared score on the testing data, providing an evaluation of the model's generalization ability.

**Code:**

train\_score = model.score(X\_train, y\_train)

test\_score = model.score(X\_test, y\_test)

**Making a Prediction:**

After training, the model is used to make a prediction based on specific feature values:

features: Contains the feature values for which a prediction is desired.

model.predict(features): Predicts the target variable (number of units sold) based on the provided features.

**Code:**

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

prediction = model.predict(features)

**Printing Results:**

Finally, the code prints out the R-squared scores and the predicted number of units sold:

The R-squared scores provide an indication of how well the model is performing on both the training and testing sets.

The predicted number of units sold is displayed for the specified "Total Price" and "Base Price".

**Code:**

print("\nR-squared scores for Train and Test:")

print(f"Training R-squared score: {train\_score}")

print(f"Testing R-squared score: {test\_score}")

print(f"\nPredicted Units Sold for Total Price={Total Price} and Base Price={Base Price}: {prediction[0]}")

**Output:**

**Conclusion:**

The provided code efficiently implements a Decision Tree Regressor model for predicting the number of units sold. By splitting the data into training and testing sets, training the model, and evaluating its performance, this code serves as a foundation for regression tasks involving similar datasets. The resulting model can be used to make accurate predictions based on input features.

Top of Form