# ***Project 10: Product Demand Prediction with Machine Learnings***

## ***SHORT EXPLANATION(Production demand prediction with machine learning)***

Production demand prediction with machine learning refers to the use of machine learning algorithms and techniques to forecast the future demand for a product or service. This is a crucial aspect of supply chain and inventory management for businesses. Here's a short explanation of the concept:

**Objective**: The primary goal is to accurately estimate how much of a product or service will be needed in the future. This prediction helps businesses optimize their production processes, manage inventory efficiently, and ensure they meet customer demand without overstocking or understocking.

**Data Input:** Machine learning models for demand prediction require historical data, such as sales records, customer orders, and relevant external factors like seasonality, economic indicators, or marketing campaigns. The quality and quantity of data are critical for model accuracy.

**Model Selection:** Various machine learning algorithms can be used, including regression models, time series forecasting, neural networks, and more. The choice of the model depends on the nature of the data and the complexity of the demand patterns.

**Feature Engineering:** Data preprocessing involves feature engineering, which may include data normalization, encoding categorical variables, handling missing data, and creating relevant features like lag variables or rolling averages.

**Training and Testing:** The model is trained on historical data, where it learns the patterns and relationships between input variables and demand. It is then tested on a separate dataset to evaluate its performance and ensure it can make accurate predictions.

**Forecasting:** Once the model is trained and validated, it can be used to make demand predictions for future time periods. These predictions can be at different levels, such as daily, weekly, or monthly, depending on the business needs.

**Optimization:** Businesses can use the demand predictions to optimize their production schedules, inventory levels, and procurement strategies. This can lead to cost savings, improved customer service, and better overall operational efficiency.

**Monitoring and Iteration:** Demand patterns can change over time due to various factors, so it's essential to monitor the model's performance regularly. If the accuracy decreases, the model may need to be retrained with updated data or modified to account for changing conditions.

In summary, production demand prediction with machine learning leverages historical data and advanced algorithms to forecast future demand accurately. This enables businesses to make informed decisions, reduce costs, and ensure they meet customer needs effectively.

## ***DATASET***

A dataset is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question.

Dataset based on Product Demand Prediction with Machine Learning are collected in www.kaggle.com/data as detail in following columns:

- ID
- Store ID
- Total Price
- Base Price
- Units Sold

**Dataset link**

https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning

## ***Details about column in the dataset***

In future demand prediction using machine learning, the dataset we mentioned appears to be a tabular dataset with several columns that likely represent various attributes of products or items for which we want to predict future demand. Let me provide a detailed explanation of each column:

1. **ID (Identifier):**

This column typically contains a unique identifier for each item or product in your dataset. It's useful for tracking and referencing specific items but usually not used as a feature for prediction.

**2. Store ID:**

This column represents the unique identifier of the store or location where the product is sold or stored. This can be crucial if you want to predict demand variations based on different stores.

**3. Total Price:**

This column likely contains the total price of the product, which is the actual amount a customer pays for it. Total price can be influenced by various factors like discounts, promotions, and taxes.

**4. Base Price:**

This column represents the base price of the product, which is the price before any discounts or promotions are applied. It provides information about the inherent value of the product.

**5. Units Sold (or Sold Quantity):**

This column contains the number of units or quantity of the product sold during a specific period. This is typically the target variable you want to predict in future demand prediction models. It's the value you aim to forecast based on the other features.

**In a machine learning context, we might use these columns in the following way:**

**ID and Store Id:**

These columns are usually not used for prediction directly but can be useful for grouping and aggregation if we want to analyze sales patterns for specific products or stores.

**Total Price and Base Price:**

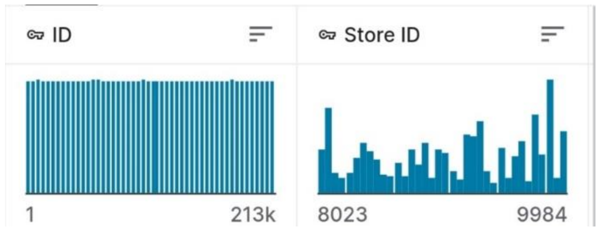
These columns can be used as features in our model if we want to examine the impact of pricing on future demand. For example, we can calculate price elasticity.

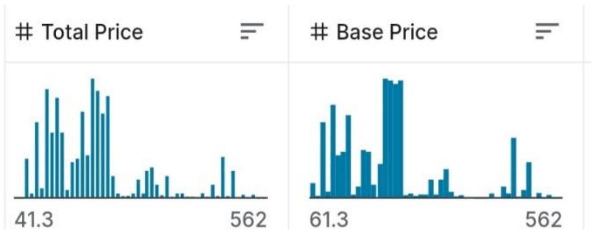
**Units Sold:**

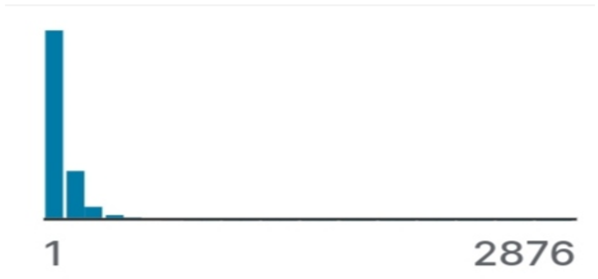
This is your target variable. You'll build a predictive model using machine learning techniques where you'll use the other columns (Total Price, Base Price, Store ID, etc.) as input features to predict the future demand, which is the number of units sold.

In our machine learning model will learn patterns and relationships between these input features and the target variable (Units Sold) from historical data. Once the model is trained, we can use it to make predictions for future demand based on the values of the Input features.

## Data set flowchart:







## ***<u>DETAILS OF LIBRARIES TO BE USER AND WAY TO DOWNLOAD</u>***

**DETAILS OF LIBRARIES:**

**1. PANDAS:**

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Pandas is one of the tools in Machine Learning which is used for data cleaning and analysis. It has features which are used for exploring, cleaning, transforming and visualizing from data.

**2. NUMPY:**

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices

**3. PLOTPY.EXPRESS :**

- The plotly.express module (usually imported as px ) contains functions that can create entire figures at once, and is referred to as Plotly Express or PX.
- Plotly Express is a built-in part of the plotly library, and is the recommended starting point for creating most common figures.

**4. SEABORN:**

- Seaborn is a library for making statistical graphics in Python.
- It builds on top of matplotlib and integrates closely with pandas data structures.
- Seaborn helps you explore and understand your data.

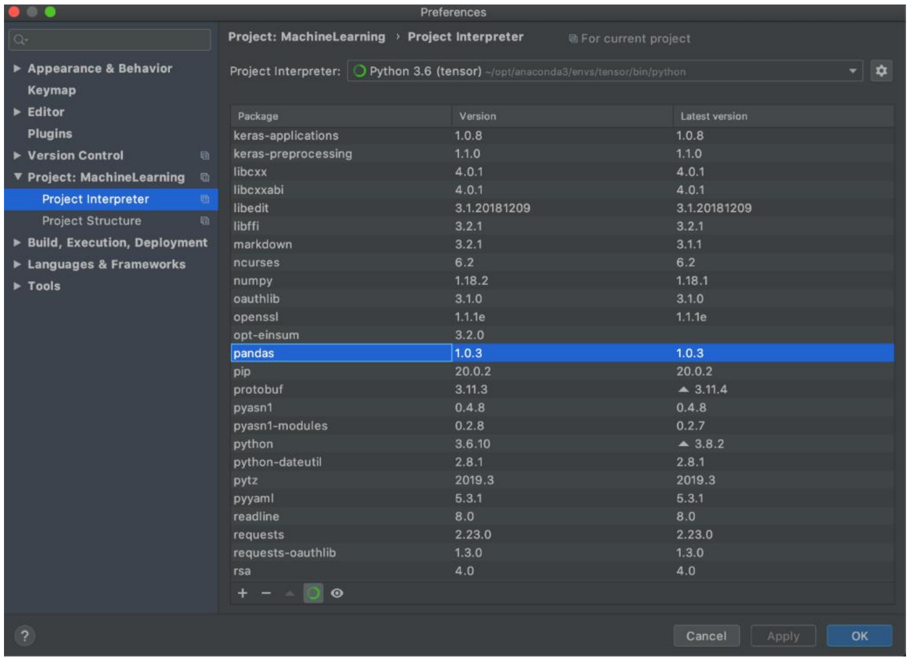
**5. MATPLOTLIB:**

- Matplotlib is a low-level library of Python which is used for data visualization. It is easy to use and emulates
- MATLAB like graphs and visualization

**<u>How to import packages in pycharm:</u>**

To install Pandas on PyCharm,

1. click on File
2. Go to the Settings.
3. Under Settings, choose your Python project and select Python Interpreter.
4. Then, search for the Pandas package and
5. click Install Package



Project: MachineLearning › Project Interpreter

Project Interpreter: Python 3.6 (tensor) ~/opt/anaconda3/envs/tensor/bin/python

| Package | Version | Latest version |
|---|---|---|
| keras-applications | 1.0.8 | 1.0.8 |
| keras-preprocessing | 1.1.0 | 1.1.0 |
| libcxx | 4.0.1 | 4.0.1 |
| libcxxabi | 4.0.1 | 4.0.1 |
| libedit | 3.1.20181209 | 3.1.20181209 |
| libffi | 3.2.1 | 3.2.1 |
| markdown | 3.2.1 | 3.1.1 |
| ncurses | 6.2 | 6.2 |
| numpy | 1.18.2 | 1.18.1 |
| oauthlib | 3.1.0 | 3.1.0 |
| openssl | 1.1.1e | 1.1.1e |
| opt-einsum | 3.2.0 | |
| pandas | 1.0.3 | 1.0.3 |
| pip | 20.0.2 | 20.0.2 |
| protobuf | 3.11.3 | ▲ 3.11.4 |
| pyasn1 | 0.4.8 | 0.4.8 |
| pyasn1-modules | 0.2.8 | 0.2.7 |
| python | 3.6.10 | ▲ 3.8.2 |
| python-dateutil | 2.8.1 | 2.8.1 |
| pytz | 2019.3 | 2019.3 |
| pyyaml | 5.3.1 | 5.3.1 |
| readline | 8.0 | 8.0 |
| requests | 2.23.0 | 2.23.0 |
| requests-oauthlib | 1.3.0 | 1.3.0 |
| rsa | 4.0 | 4.0 |

**NOTE:**

Simlarly the other packages are Downloaded in Pycharm

**How to import packages?**

**Demo code:**

```
 import pandas as pd

# Replace 'your_dataset.csv' with the actual path or URL to your dataset
dataset_path = 'your_dataset.csv'

# Read the dataset into a pandas DataFrame
df = pd.read_csv(dataset_path)

# Display the first few rows of the dataset to verify that it was read correctly
print(df.head())
```

## ***<u>Data Collection:</u>***

Collecting data for demand prediction in machine learning involves gathering relevant information that can be used to forecast the future demand for a product. Here's a breakdown of how and what data you can collect for this purpose:

**1. Historical Sales Data:**

- This is the most critical dataset. It includes records of past sales transactions, typically with columns such as:
    - Date of sale
    - Product ID or SKU
    - Quantity sold
    - Sales price
    - Customer ID (if applicable)

**2. Product Data:**

- Information about the products being sold, including attributes like:
    - Product category
    - Product description
    - Product size

    - Product color
    - Brand name
    - Supplier or manufacturer details

**3. Market Data:**

- Data on external factors that can affect demand, such as:
    - Economic indicators (e.g., GDP, inflation rate)
    - Competitor prices and promotions
    - Market trends and industry reports
    - Seasonal patterns and holidays
    - Weather conditions (especially for weather-sensitive products)

**4. Customer Data:**

- Information about your customers can help you understand their buying behavior:
    - Customer demographics (age, gender, location)
    - Customer segmentation data
    - Customer purchase history
    - Customer loyalty program data

**5. Marketing and Promotion Data:**

- Details about marketing campaigns and promotions that may influence demand:
    - Start and end dates of promotions
    - Advertising channels used
    - Promotion type (discounts, buy-one-get-one, etc.)

**6. Inventory Data:**

- Data on current stock levels, including:
    - Inventory levels for each product
    - Reorder points and lead times
    - Stockouts and backorders

**7. Social Media and Online Presence:**

- Social media mentions, engagement, and sentiment related to your products or brand can provide valuable insights:
    - Social media posts and comments
    - Likes, shares, and comments on social media platforms

    - Website traffic and clickstream data

**8. Customer Feedback and Reviews:**

- Collecting and analyzing customer reviews and feedback:
    - Product reviews and ratings
    - Customer comments and complaints

**9. Point of Sale (POS) Data:**

- If applicable, data from physical and online point-of-sale systems:
    - Transaction data
    - Receipt data
    - In-store foot traffic data

**10. Supplier and Supply Chain Data:** - Information related to your supply chain and suppliers: - Supplier lead times - Transportation costs - Production schedules

**11. Geographic Data:** - Location-based data that can affect demand variations: - Geographic sales data - Store locations and traffic patterns

**12. IOT and Sensor Data (Advanced):** - Data from IOT sensors on products or in-store environments for real-time insights: - Product usage data - Environmental conditions in stores

**13. External APIs:** - Access external APIs for relevant data, such as weather APIs, economic indicators APIs, or social media APIs.

**14. Market Research and Reports:** - Industry-specific reports, studies, and market research can provide valuable insights into market trends.

**15. User Behavior Data (Advanced):** - For online businesses, analyze user behavior on your website or app: - Product views - Cart additions - Checkout abandonment rates

**16. Subscription and Membership Data (Advanced):** - Data from subscription-based services or loyalty programs can offer insights into recurring revenue and customer retention.

**17. Customer Support Data (Advanced):** - Records of customer inquiries, complaints, and support interactions.

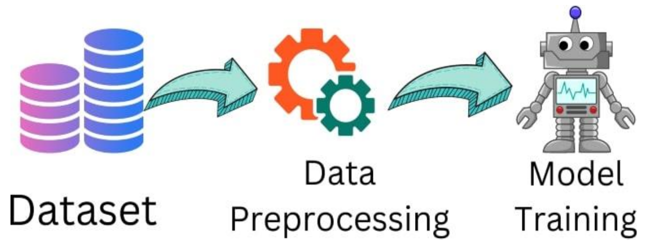
**18. Employee Data (Advanced):** - Staffing levels, shifts, and workforce data if they affect product availability.

**19. Transportation and Logistics Data (Advanced):** - Data on transportation routes, shipping times, and logistics can be relevant.

When collecting data, it's essential to ensure that you have a sufficient historical dataset for accurate prediction, and that the data is clean, properly formatted, and relevant to your specific prediction task. Data preprocessing, including data cleaning, feature engineering, and data integration, will be necessary to prepare the collected data for machine learning model training. Additionally, consider ethical and privacy considerations when handling customer data.

## ***Data Preprocessing:***

Preprocessing in machine learning involves getting the data ready for analysis by making it more useful and easier for machine learning algorithms to understand. The main reason for data preprocessing is that we never get clean datasets. Always while we train the model or perform further operations, we need a clean dataset.



**Need of Preprocessing :**

- Data Quality Improvement
- Feature Extraction and Selection
- Handling Missing Data
- Data Normalization and Scaling
- Outlier Detection and Treatment
- Reducing Computational Requirements

Steps of Preprocessing in Machine Learning:

**Obtaining the Dataset:**

Gather the dataset you will be working with. Here is a dataset of cars. The dataset can be in any format. This dataset is in **"CSV".** It also can be in **"JSON"** or **"XLXS"** format. **CSV** stands for **"Comma Separated Values"** where all data is represented in a tabular format like a spreadsheet which is easy to understand.

**Importing Libraries:**

Bring in the necessary libraries for data manipulation and analysis, like Pandas, NumPy, and Scikit-learn.

- **pandas:** pandas are the Python library that is basically used for loading and managing the dataset.
- **numpy:** numpy is also a Python library used for mathematical operations, or we can say that in scientific calculations like adding two large multidimensional arrays.
- **sklearn and sci-kit-learn:** both are the same libraries used for data analytics.

Here pd and np are short naming conventions of pandas and numpy so that we can use these libraries by these short names.

**Loading the Dataset:**

Load the dataset into your programming environment, which can be in different formats such as CSV, Excel, or datasets.

**read_csv()** is the function of the panda's library, which is basically used to read the CSV file, the file can be saved locally, or we can also give a URL to this.

**dataset.drop()** is the function that is used to remove the specified column name, which is not needed in the dataset.

**Find Empty /Missing Data:**
Identify any missing values in the dataset and decide on a strategy to address them. Options include removing the rows or columns with missing data or filling in the missing values with methods like mean, median, or mode. In this code snippet, missing values are handled by simply dropping the rows that contain them.

- **sklearn and sci-kit-learn:** both are the same libraries used for data analytics.

Here pd and np are short naming conventions of pandas and numpy so that we can use these libraries by these short names.

**Loading the Dataset:**

Load the dataset into your programming environment, which can be in different formats such as CSV, Excel, or datasets.

**read_csv()** is the function of the panda's library, which is basically used to read the CSV file, the file can be saved locally, or we can also give a URL to this.

**dataset.drop()** is the function that is used to remove the specified column name, which is not needed in the dataset.

**Find Empty /Missing Data:**
Identify any missing values in the dataset and decide on a strategy to address them. Options include removing the rows or columns with missing data or filling in the missing values with methods like mean, median, or mode. In this code snippet, missing values are handled by simply dropping the rows that contain them.

```
# Handling missing values (if any)
df = df.dropna()  # Remove rows with missing values
```

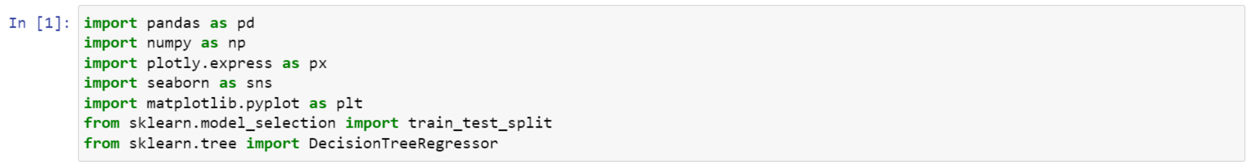
**Encoding Categorical Data :**
Convert categorical variables into numerical representations that can be understood by machine learning algorithms. This can involve techniques like one-hot encoding or label encoding. Categorical variables need to be encoded numerically before using them in machine learning models

**Scaling Numerical Features :**
Normalize or standardize the numerical features in the dataset to ensure they are on a similar scale. This helps prevent certain features from dominating the model's calculations. Numerical features are often scaled to a similar range to avoid the dominance of certain features during modeling.

**Split Dataset :**
Divide the dataset into separate training and test sets. The training set is used to train the model, while the test set is used to evaluate its performance. This helps assess how well the model generalizes to new, unseen data. The code splits the preprocessed DataFrame into input features (X) and the target variable (y).

## 1. Required libraries

- Import the CSV library. import csv.
- Open the CSV file the .open() method in python is used to open files and return a file object.
- Use the csv.reader object to read the CSV file. csvreader = csv.reader(file)
- Extract the field names
- Extract the rows/records
- Close the file.



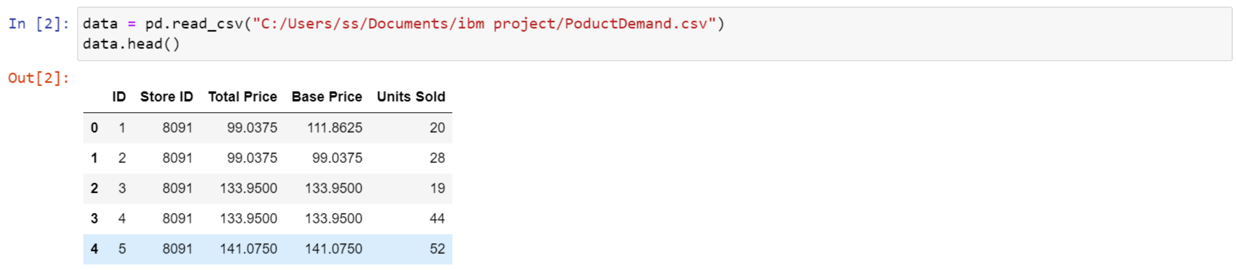
```
In [1]: import pandas as pd
        import numpy as np
        import plotly.express as px
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeRegressor
```

## 2.To read the dataset

To work on the data, you can either load the CSV in Excel or in Pandas. For the purposes of this tutorial, we'll load the CSV data in Pandas

df = pd.read_csv('data.csv')

df.head()



```
In [2]: data = pd.read_csv("C:/Users/ss/Documents/ibm project/PoductDemand.csv")
        data.head()

Out[2]:
           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
```

## 3.To create the matrix

how to convert CSV data into a matrix We will use read.csv() function to load the csv file.

**Syntax:** object=read.csv(path)

Where, path is the location of a file present in our local system.

**Matrix:** Matrix is a two-dimensional data structure that contains rows and columns. It can hold multiple data types. We can convert csv file data into matrix by using the method called as.matrix()

**Syntax:**as.matrix(csv_file_object)

## 4.Handing the Missing data

The data we get is rarely homogenous. Sometimes data can be missing and it needs to be handled so that it does not reduce the performance of our machine learning model.

To do this we need to replace the missing data by the Mean or Median of the entire column. For this we will be using the sklearn.preprocessing Library which contains a class called Imputer which will help us in taking care of our missing data.

From sklearn.preprocessing import Imputer

Imputer = Imputer(missing_values = "NaN", strategy = "mean", axis = 0)

Our object name is imputer. The Imputer class can take parameters like :

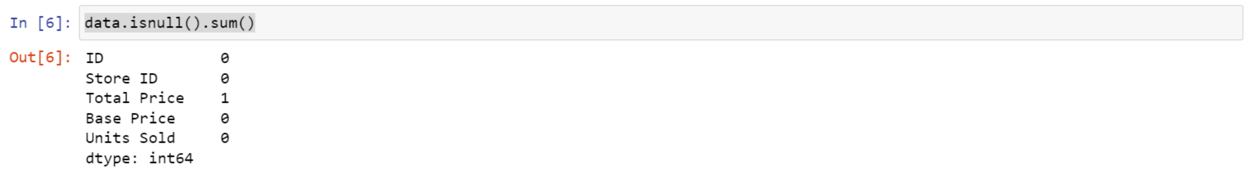
**Missing_values :** It is the placeholder for the missing values. All occurrences of missing_values will be imputed. We can give it an integer or "NaN" for it to find missing values.

**Strategy :** It is the imputation strategy — If "mean", then replace missing values using the mean along the axis (Column). Other strategies include "median" and "most_frequent".

**Axis :** It can be assigned 0 or 1, 0 to impute along columns and 1 to impute along rows.

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

data.isnull().sum()



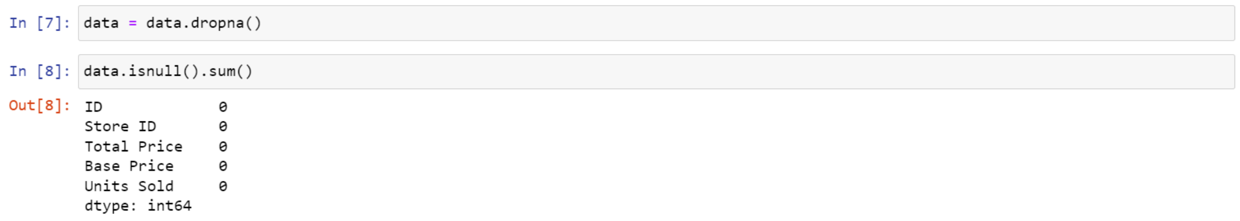
```
In [6]: data.isnull().sum()

Out[6]: 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:

data = data.dropna()

Now we have eliminated the missing data or null values in the datset:

data.isnull().sum()



```
In [7]: data = data.dropna()

In [8]: data.isnull().sum()

Out[8]: ID             0
        Store ID       0
        Total Price    0
        Base Price     0
        Units Sold     0
        dtype: int64
```

## 5. Encoding Categorical Data

Data Encoding is an important pre-processing step in Machine Learning. It refers to the process of converting categorical or textual data into numerical format, so that it can be used as input for algorithms to process. The reason for encoding is that most machine learning algorithms work with numbers and not with text or categorical variables.

We often encounter different types of variables. One such type is categorical variables. Categorical variables are usually represented as 'strings' or 'categories' and are finite in number.

There are two types of categorical data –

- Ordinal Data
- Nominal Data

**Ordinal Data:**

The categories of ordinal data have an Inherent Order. This means that the categories can be Ranked or ordered from highest to lowest or vice versa.

**For example,** the variable "highest degree a person has" is an ordinal variable. The categories (High school, Diploma, Bachelors, Masters, PhD) can be ranked in order of the level of education attained.

**Nominal Data:** The categories of nominal data do not have an Inherent Order. This means that the categories cannot be ranked or ordered.

**For example,** the variable "city where a person lives" is a nominal variable. The categories (Delhi, Mumbai, Ahmedabad, Bangalore, etc.) cannot be ranked or ordered.

**Why it is Important?**

- Most machine learning algorithms work only with numerical data, so categorical variables (such as text labels) must be transformed into numerical values.
- This allows the model to identify patterns in the data and make predictions based on those patterns.
- Encoding also helps to prevent bias in the model by ensuring that all features are equally weighted.
- The choice of encoding method can have a significant impact on model performance, so it is important to choose an appropriate encoding technique based on the nature of the data and the specific requirements of the model.

There are several methods for encoding categorical variables, including

1. One-Hot Encoding
2. Dummy Encoding
3. Ordinal Encoding
4. Binary Encoding
5. Count Encoding
6. Target Encoding

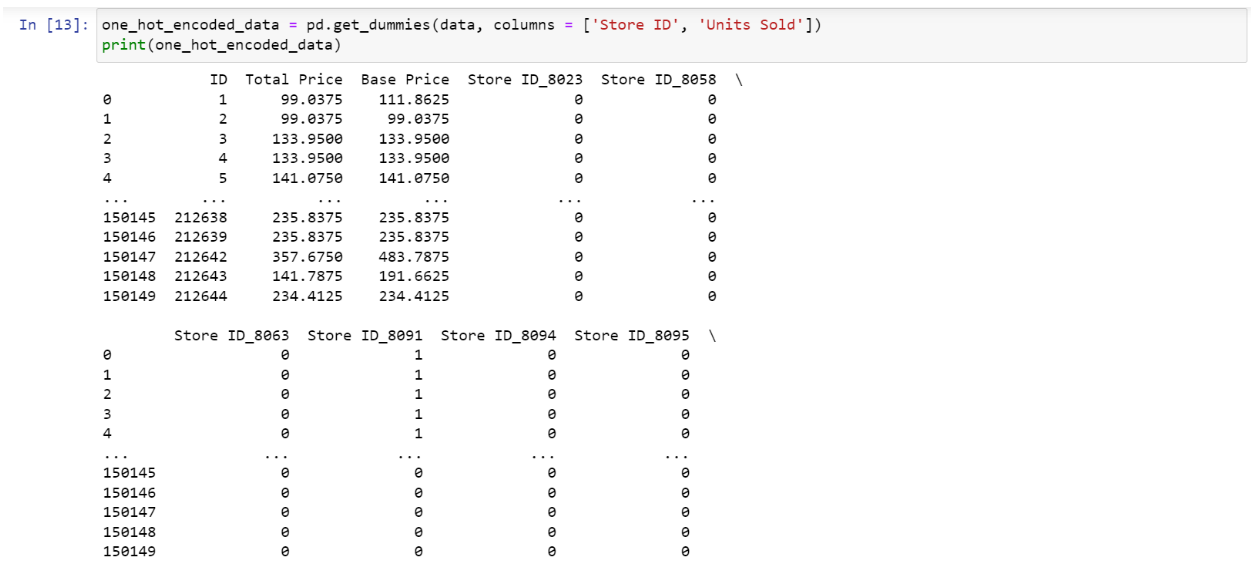
Let us discuss briefly about One-Hot Encoding ….

**One-Hot Encoding**

One-Hot Encoding is the process of creating dummy variables. This technique is used for categorical variables where order does not matter. One-Hot encoding technique is used when the features are nominal(do not have any order). In one hot encoding, for every categorical feature, a new variable is created.

- One-Hot Encoding is the Most Common method for encoding Categorical variables.
- a Binary Column is created for each Unique Category in the variable.
- If a category is present in a sample, the corresponding column is set to 1, and all other columns are set to 0.

```
one_hot_encoded_data = pd.get_dummies(data, columns = ['Store ID', 'Units Sold'])

print(one_hot_encoded_data)
```



```
In [13]: one_hot_encoded_data = pd.get_dummies(data, columns = ['Store ID', 'Units Sold'])
         print(one_hot_encoded_data)

                    ID  Total Price  Base Price  Store ID_8023  Store ID_8058  \
         0           1      99.0375    111.8625              0              0
         1           2      99.0375     99.0375              0              0
         2           3     133.9500    133.9500              0              0
         3           4     133.9500    133.9500              0              0
         4           5     141.0750    141.0750              0              0
         ...       ...          ...         ...            ...            ...
         150145  212638     235.8375    235.8375              0              0
         150146  212639     235.8375    235.8375              0              0
         150147  212642     357.6750    483.7875              0              0
         150148  212643     141.7875    191.6625              0              0
         150149  212644     234.4125    234.4125              0              0

                 Store ID_8063  Store ID_8091  Store ID_8094  Store ID_8095  \
         0                   0              1              0              0
         1                   0              1              0              0
         2                   0              1              0              0
         3                   0              1              0              0
         4                   0              1              0              0
         ...               ...            ...            ...            ...
         150145              0              0              0              0
         150146              0              0              0              0
         150147              0              0              0              0
         150148              0              0              0              0
         150149              0              0              0              0
```

## ***Feature Engineering:***

1. Feature engineering is a skill every data scientist should know how to perform, especially in the case of time series.
2. There are 6 powerful feature engineering techniques for time series

Introduction to Time Series:
In a time series, the data is captured at equal intervals and each successive data point in the series depends on its past values.

Setting up the Problem Statement for Time Series Data:
We'll be working on a fascinating problem to learn feature engineering techniques for time series.

1. Feature Engineering for Time Series #1: Date-Related Features

Task of forecasting the sales for a particular product. We can find out the sales pattern for weekdays and weekends based on historical data. Thus, having information about the day, month, year, etc. can be useful for forecasting the values.

2. Feature Engineering for Time Series #2: Time-Based Features

Extracting time-based features is very similar to what we did above when extracting date-related features. We start by converting the column to DateTime format and use the *.dt* accessor

3. Feature Engineering for Time Series #3: Lag Features

The lag value we choose will depend on the correlation of individual values with its past values.

4. Feature Engineering for Time Series #4: Rolling Window Feature

Recency in an important factor in a time series. Values closer to the current date would hold more information.

5. Feature Engineering for Time Series #5: Expanding Window Feature

The idea behind the expanding window feature is that it takes all the past values into account

6. Feature Engineering for Time Series #6: Domain-Specific Features

Having a good understanding of the problem statement, clarity of the end objective and knowledge of the available data is essential to engineer domain-specific features for the model.

**Feature that captures Seasonal Pattern:**

Seasonal demand refers to the changes in consumer buying habits depending on the time of year. This may be related to changes in season (i.e., higher demand for snow boots during winter vs. higher demand for sunglasses during summer)

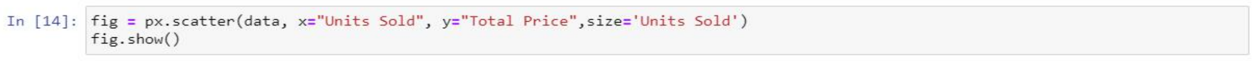
**External Influences on the Product Demand**

- Political Factors.
- Economic Factors.
- Social Factors.
- Technological Factors.
- Legal Factors.
- Demographic Factors.
- Ethical Factors.
- Natural Factors.

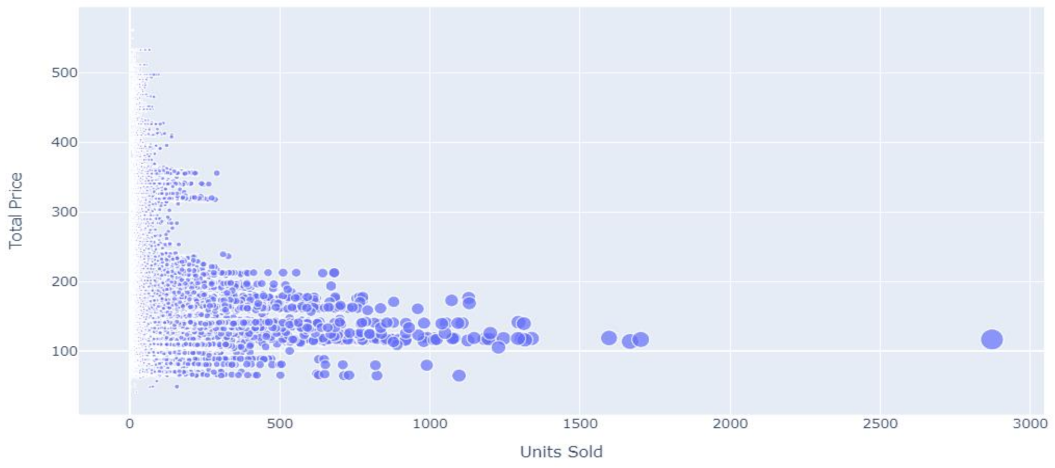
**Scatter plot**

Let us now analyze the relationship between the price and the demand for the product. Here I will use a scatter plot to see how the demand for the product varies with the price change:

```
fig = px.scatter(data, x="Units Sold", y="Total Price",size='Units Sold')

fig.show()
```



```
In [14]: fig = px.scatter(data, x="Units Sold", y="Total Price",size='Units Sold')
         fig.show()
```



## Correlation

Correlation analysis is a statistical method used to measure the strength of the linear relationship between two variables and compute their association. Correlation analysis calculates the level of change in one variable due to the change in the other. A high correlation points to a strong relationship between the two variables, while a low correlation means that the variables are weakly related.

Researchers use correlation analysis to analyze quantitative data collected through research methods like surveys and live polls for market research. They try to identify relationships, patterns, significant connections, and trends between two variables or datasets. There is a positive correlation between two variables when an increase in one variable leads to an increase in the other. On the other hand, a negative correlation means that when one variable increases, the other decreases and vice-versa.

## Types of Correlation Analysis in Data Mining

1. Pearson r correlation

2. Kendall rank correlation

3. Spearman rank correlation

## Benefits of Correlation Analysis

1. Reduce Time to Detection

2. Reduce Alert Fatigue

3. Reduce Costs

**Pearson r correlation**

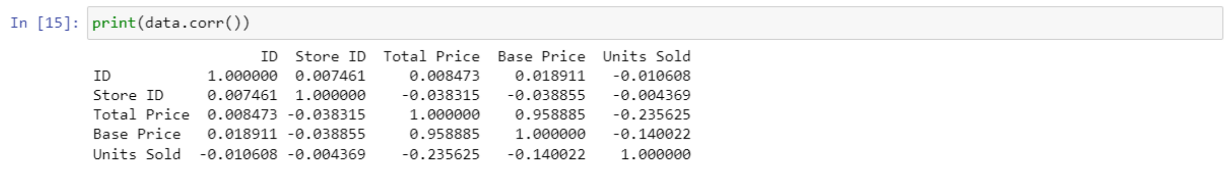
Pearson r correlation is the most widely used correlation statistic to measure the degree of the relationship between linearly related variables. For example, in the stock market, if we want to measure how two stocks are related to each other, Pearson r correlation is used to measure the degree of relationship between the two. The point-biserial correlation is conducted with the Pearson correlation formula, except that one of the variables is dichotomous. The following formula is used to calculate the Pearson r correlation:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\ \sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

- $r_{xy}$= Pearson r correlation coefficient between x and y
- n= number of observations
- $x_i$ = value of x (for ith observation)
- $y_i$= value of y (for ith observation)

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:

```
print(data.corr())
```



```
In [15]: print(data.corr())

                   ID  Store ID  Total Price  Base Price  Units Sold
ID           1.000000  0.007461     0.008473    0.018911   -0.010608
Store ID     0.007461  1.000000    -0.038315   -0.038855   -0.004369
Total Price  0.008473 -0.038315     1.000000    0.958885   -0.235625
Base Price   0.018911 -0.038855     0.958885    1.000000   -0.140022
Units Sold  -0.010608 -0.004369    -0.235625   -0.140022    1.000000
```

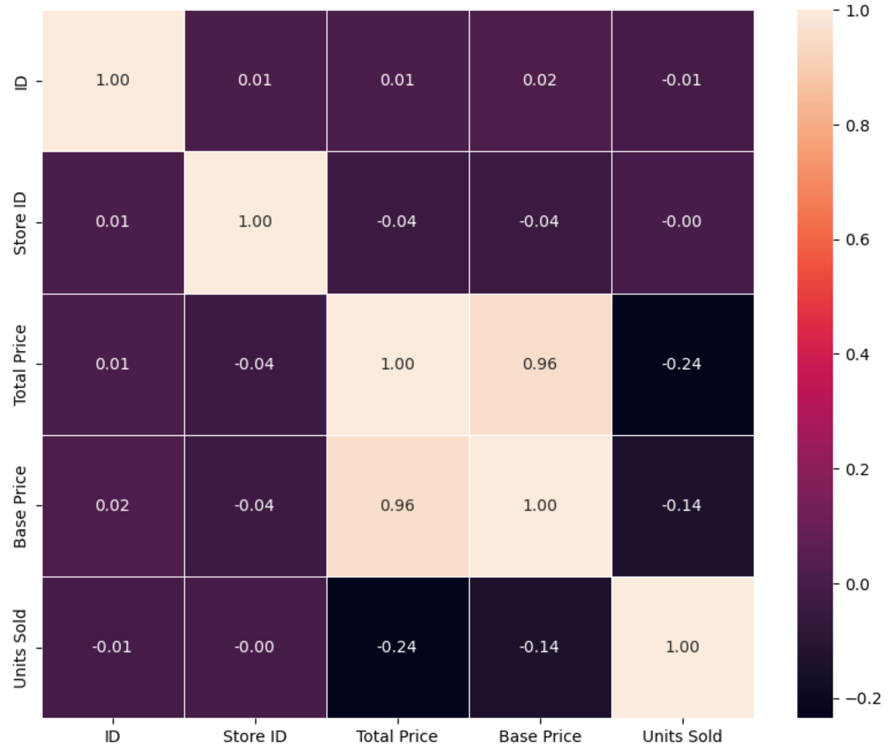
**SEABORN HEATMAP:**

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

seaborn.heatmap()

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

plt.figure(figsize=(10, 8))

sns.heatmap(correlations, annot=True,fmt=".2f", linewidth=.5)

plt.show()
```

```
In [16]: correlations = data.corr(method='pearson')
         plt.figure(figsize=(10, 8))
         sns.heatmap(correlations, annot=True,fmt=".2f", linewidth=.5)
         plt.show()
```



## ***Model Training:***

Training a machine learning model for demand product prediction using preprocessed data involves several steps. Here's a high-level guide on how to do it:

1. **Split Data into Training and Testing Sets:**
   - Split your preprocessed data into two parts: a training set and a testing (or validation) set. The training set will be used to train the model, while the testing set will be used to evaluate its performance.
2. **Select a Machine Learning Algorithm:**
   - Choose a machine learning algorithm suitable for your demand prediction task. Common choices include:
     - Linear Regression
     - Decision Trees
     - Random Forest
     - Gradient Boosting (e.g., XGBoost, LightGBM)
     - Time Series Forecasting Models (e.g., ARIMA, Prophet)
     - Deep Learning Models (e.g., LSTM, GRU)

3. **Feature Selection (if not done during preprocessing):**
    - If you haven't already, select the most relevant features from your preprocessed data to use in training the model.
4. **Train the Model:**
    - Fit the selected machine learning model to your training data. This involves providing the model with your historical data (features) and the corresponding target variable (demand).

```
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestRegressor


# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)


# Initialize and train the model (Random Forest as an example)

model = RandomForestRegressor(n_estimators=100, random_state=42)

model.fit(X_train, y_train)
```

5. **Hyperparameter Tuning (Optional)**:
    - Fine-tune the hyperparameters of your model to optimize its performance. This can be done using techniques like grid search or random search.

6. **Model Evaluation:**

Evaluate the model's performance on the testing set using appropriate evaluation metrics. Common metrics for regression tasks include Mean

- Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2) score.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


# Make predictions on the test set

y_pred = model.predict(X_test)

# Calculate evaluation metrics

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)
```

```
rmse = np.sqrt(mse)

r2 = r2_score(y_test, y_pred)
```

7. **Model Interpretation (Optional):**
    - Depending on the model, you may want to interpret its predictions to understand the factors influencing demand. Techniques like feature importance analysis and SHAP (SHapley Additive exPlanations) can help with model interpretability.
8. **Deployment (Optional):**
    - If you plan to use the model in a production environment to make real-time predictions, you'll need to deploy it as part of an application, API, or workflow.
9. **Monitoring and Maintenance:**
    - Continuously monitor the model's performance in production and retrain it as needed to adapt to changing demand patterns. Regularly update the training data to incorporate new historical data.
10. **Documentation and Reporting:**
    - Document the model, its performance, and any insights gained from it. Create reports or dashboards to present the predictions and insights to stakeholders.
11. **Iterate and Improve:**
    - Based on the model's performance and feedback from stakeholders, consider making improvements to the model or the data preprocessing pipeline.

Remember that demand prediction is an iterative process, and the accuracy of your model may improve with time as you collect more data and refine your approach. Additionally, the choice of algorithm and preprocessing steps should be tailored to the specific characteristics of your data and the nature of your demand prediction problem.

## Splitting the Data Set into test set and training set:

In product demand prediction, splitting the dataset into a training set and a test set is crucial for assessing the performance of your predictive model. we can use the train_test_split function from the sklearn.model_selection module in Python to accomplish this. Here's how we can do it with code and an explanation:

```
 from sklearn.model_selection import train_test_split

# Assuming we have our feature data (X) and target variable (Y) ready

# X is our input features, and Y is our target variable (demand prediction)

# Split the data into a training set and a test set

 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

 # Explanation:
```

# - X: our feature data, which contains the information used to make predictions.

# - Y: our target variable, which is what we want to predict (e.g., product demand).

# The `test_size` parameter determines the proportion of the data that should be allocated to the test set. In this case, 0.2 means that 20% of the data will be used for testing, and 80% will be used for training. we can adjust this value according to our needs.

# The `random_state` parameter is optional, and it is used to ensure reproducibility. If we set it to a specific number (e.g., 1), we'll get the same random split every time we run the code. This is useful for debugging and sharing results with others.

# After running this code, you will have four sets of data:

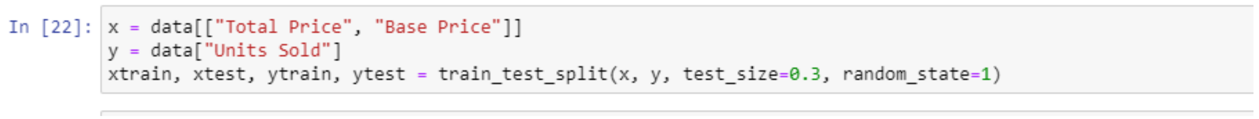
# - xtrain: The feature data for training the model.

# - xtest: The feature data for evaluating the model.

# - ytrain: The corresponding target variable for the training data.

# - ytest: The corresponding target variable for the test data.

# we can then use  xtrain and ytrain to train our product demand prediction model, and xtest and ytest to evaluate its performance.



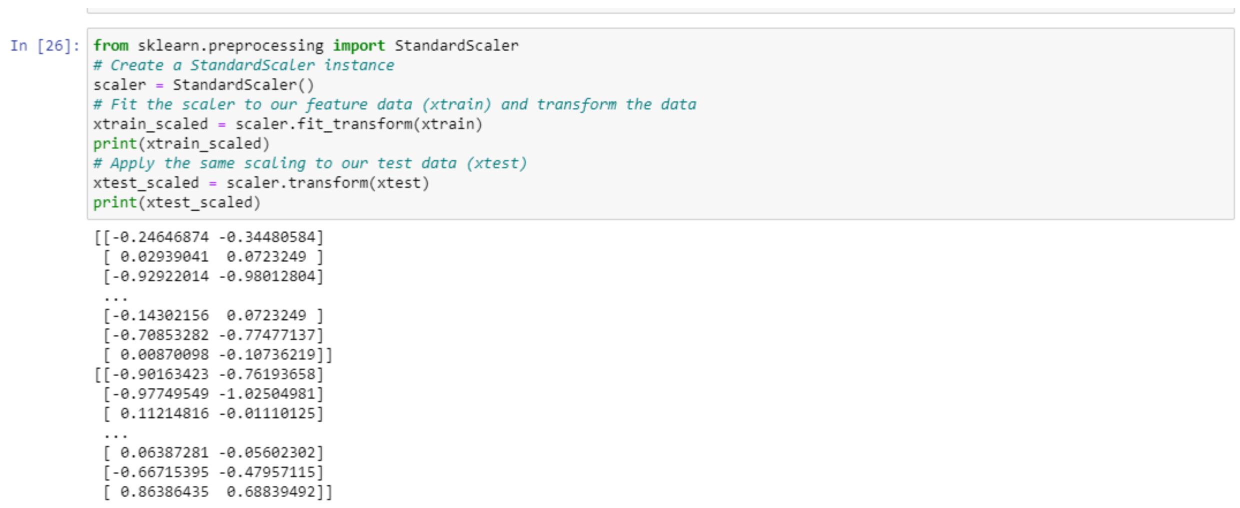
```
In [22]: x = data[["Total Price", "Base Price"]]
         y = data["Units Sold"]
         xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_state=1)
```

## Feature scaling:

Feature scaling is an important preprocessing step in product demand prediction, especially when you are working with machine learning algorithms that are sensitive to the scale of input features. StandardScaler is a common method for scaling features in Python. Here's how we can use it in product demand prediction:

First, import the StandardScaler from sklearn.preprocessing:

```
from sklearn.preprocessing import StandardScaler

# Create a StandardScaler instance

scaler = StandardScaler()

# Fit the scaler to our feature data (xtrain) and transform the data

xtrain_scaled = scaler.fit_transform(xtrain)

print(xtrain_scaled)

# Apply the same scaling to our test data (xtest)

xtest_scaled = scaler.transform(xtest)

print(xtest_scaled)
```



```
In [26]: from sklearn.preprocessing import StandardScaler
         # Create a StandardScaler instance
         scaler = StandardScaler()
         # Fit the scaler to our feature data (xtrain) and transform the data
         xtrain_scaled = scaler.fit_transform(xtrain)
         print(xtrain_scaled)
         # Apply the same scaling to our test data (xtest)
         xtest_scaled = scaler.transform(xtest)
         print(xtest_scaled)
```

```
[[-0.24646874 -0.34480584]
 [ 0.02939041  0.0723249 ]
 [-0.92922014 -0.98012804]
 ...
 [-0.14302156  0.0723249 ]
 [-0.70853282 -0.77477137]
 [ 0.00870098 -0.10736219]]
[[-0.90163423 -0.76193658]
 [-0.97749549 -1.02504981]
 [ 0.11214816 -0.01110125]
 ...
 [ 0.06387281 -0.05602302]
 [-0.66715395 -0.47957115]
 [ 0.86386435  0.68839492]]
```

**Explanation:**

1. Import StandardScaler from the sklearn.preprocessing module.

2. Create a StandardScaler object named scaler.

3. Use the fit_transform method to compute the mean and standard deviation of our training data (X_train) and then scale the training data accordingly. This ensures that our training data has a mean of 0 and a standard deviation of 1.

4. Use the transform method to apply the same scaling to our test data (X_test). It's important to use the same scaling parameters (mean and standard deviation) calculated from the training data to ensure that the test data is scaled consistently.

## ***Model Selection:***

**Model selection** is the task of selecting a model from among various candidates on the basis of performance criterion to choose the best one. In the context of learning, this may be the selection of a statistical model from a set of candidate models, given data. In the simplest cases, a pre-existing set of data is considered. However, the task can also involve the design_of experiments such that the data collected is well-suited to the problem of model selection. Given candidate models of similar predictive or explanatory power, the simplest model is most likely to be the best choice

**Regression Algorithm:**

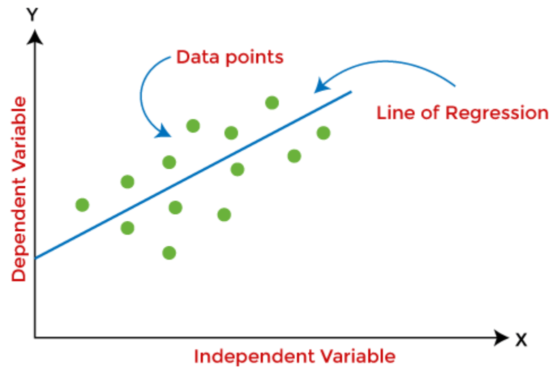
Regression analysis is often used in finance, investing, and others, and finds out the relationship between a single dependent variable(target variable) dependent on several independent ones. For example, predicting house price, stock market or salary of an employee, etc are the most common regression problems.Here is a list of top 5 regression algorithms

- Linear Regression
- Decision Tree
- Support Vector Regression
- Lasso Regression
- Random Forest

**Linear Regression:**

Linear regression is the simplest machine learning model in which we try to predict one output variable using one or more input variables. The representation of linear regression is a linear equation, which combines a set of input values(x) and predicted output(y) for the set of those input values. It is represented in the form of a line:

$$Y = bx + c.$$



The main aim of the linear regression model is to find the best fit line that best fits the data points.

Linear regression is extended to multiple linear regression (find a plane of best fit) and polynomial regression (find the best fit curve).

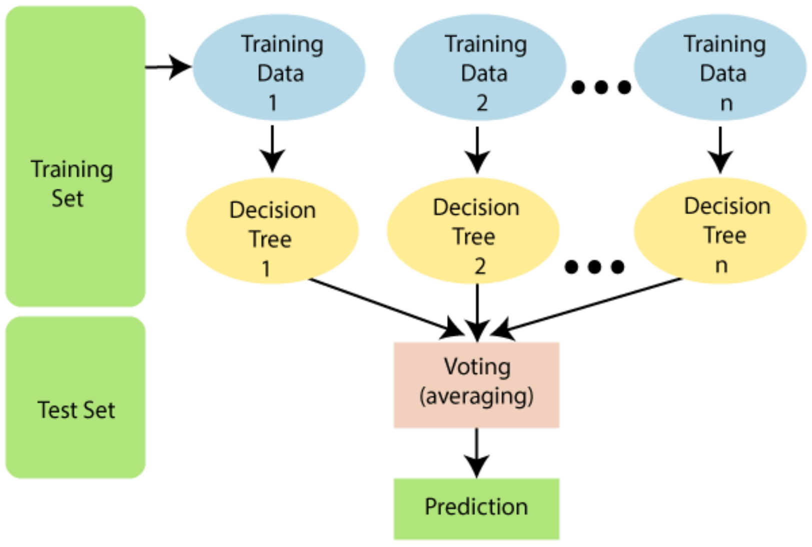
**Random Forest Algorithm**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

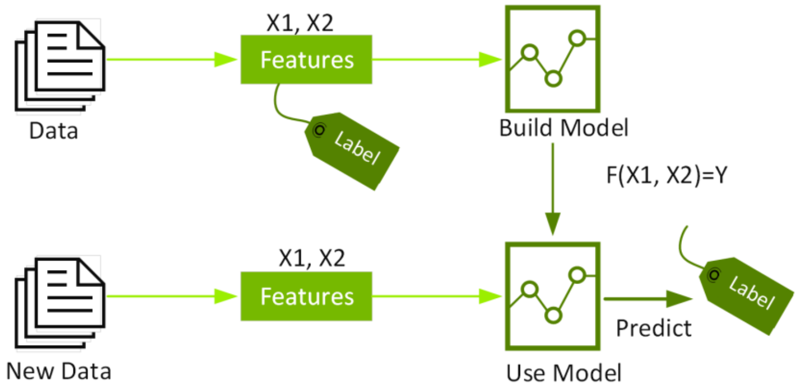
The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



**XGBoost:**

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.
It's vital to an understanding of XGBoost to first grasp the machine learning concepts and algorithms that XGBoost builds upon: supervised machine learning, decision trees, ensemble learning, and gradient boosting.
Supervised machine learning uses algorithms to train a model to find patterns in a dataset with labels and features and then uses the trained model to predict the labels on a new dataset's features.



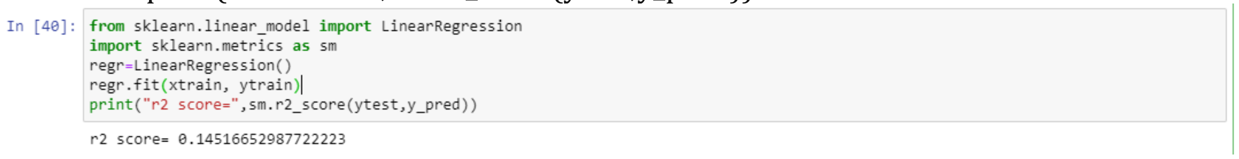
Decision trees create a model that predicts the label by evaluating a tree of if-then-else true/false feature questions, and estimating the minimum number of questions needed to assess the probability of making a correct decision. Decision trees can be used for classification to predict a category, or regression to predict a continuous numeric value.

## LINEAR REGRESSION

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a "least squares" method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable).

```
from sklearn.linear_model import LinearRegression
import sklearn.metrics as sm
regr=LinearRegression()
regr.fit(xtrain, ytrain)
print("r2 score=",sm.r2_score(ytest,y_pred))
```



```
In [40]: from sklearn.linear_model import LinearRegression
         import sklearn.metrics as sm
         regr=LinearRegression()
         regr.fit(xtrain, ytrain)
         print("r2 score=",sm.r2_score(ytest,y_pred))

         r2 score= 0.14516652987722223
```

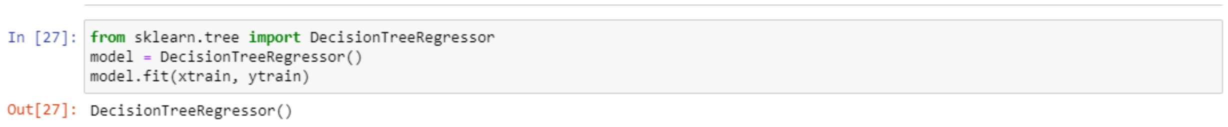
## DECISION TREE

A decision tree is a flowchart-like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine-learning algorithm, which is used for both classification and regression problems.

## DECISION TREE REGRESSOR

use the decision tree regression algorithm to train our model

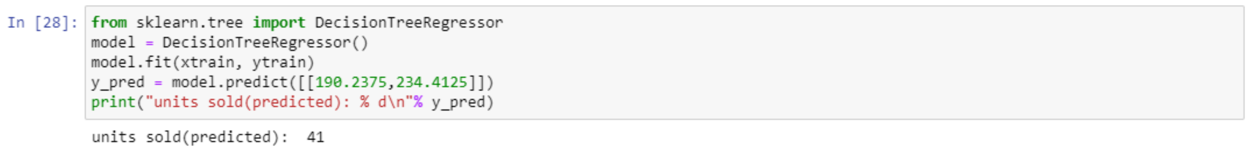
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
```



```
In [27]: from sklearn.tree import DecisionTreeRegressor
         model = DecisionTreeRegressor()
         model.fit(xtrain, ytrain)

Out[27]: DecisionTreeRegressor()
```

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:

```
model.fit(xtrain, ytrain)

y_pred = model.predict([[190.2375,234.4125]])

print("units sold(predicted): % d\n"% y_pred)
```

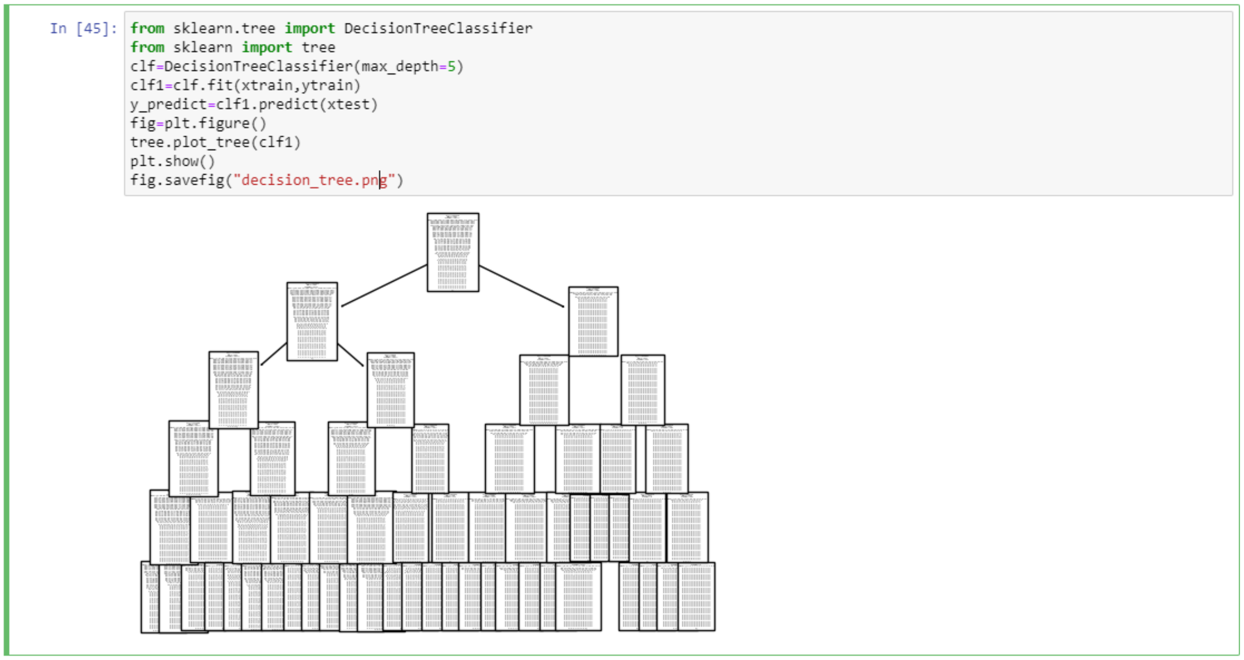


```
In [28]: from sklearn.tree import DecisionTreeRegressor
         model = DecisionTreeRegressor()
         model.fit(xtrain, ytrain)
         y_pred = model.predict([[190.2375,234.4125]])
         print("units sold(predicted): % d\n"% y_pred)

         units sold(predicted):  41
```

**DECISION TREE CLASSIFIER**

use the decision tree classifier algorithm to train our model

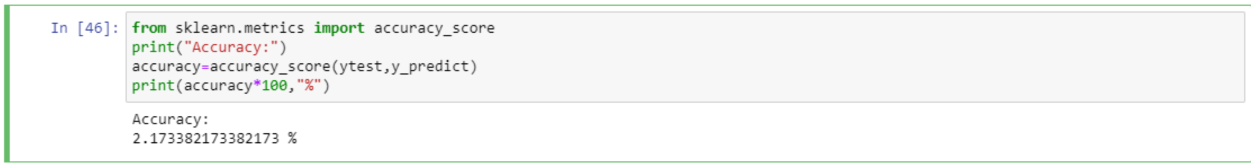
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
clf=DecisionTreeClassifier(max_depth=5)
clf1=clf.fit(xtrain,ytrain)
y_predict=clf1.predict(xtest)
fig=plt.figure()
tree.plot_tree(clf1)
plt.show()
fig.savefig("decision_tree.png")
```



```
In [45]: from sklearn.tree import DecisionTreeClassifier
         from sklearn import tree
         clf=DecisionTreeClassifier(max_depth=5)
         clf1=clf.fit(xtrain,ytrain)
         y_predict=clf1.predict(xtest)
         fig=plt.figure()
         tree.plot_tree(clf1)
         plt.show()
         fig.savefig("decision_tree.png")
```

finding the accuracy of the model

```
from sklearn.metrics import accuracy_score
print("Accuracy:")
accuracy=accuracy_score(ytest,y_predict)
print(accuracy*100,"%")
```



```
In [46]: from sklearn.metrics import accuracy_score
         print("Accuracy:")
         accuracy=accuracy_score(ytest,y_predict)
         print(accuracy*100,"%")

         Accuracy:
         2.173382173382173 %
```

## ***Evaluation:***

Machine Learning is a branch of Artificial Intelligence. It contains many algorithms to solve various real-world problems. Building a Machine learning model is not only the Goal of any data scientist but deploying a more generalized model is a target of every Machine learning engineer.

Regression is also one type of supervised Machine learning and in this tutorial, we will discuss various metrics for evaluating regression Models and How to implement them using the sci-kit-learn library.

**Regression:**

Regression is a type of Machine learning which helps in finding the relationship between independent and dependent variable.

In simple words, Regression can be defined as a Machine learning problem where we have to predict discrete values like price, Rating, Fees, etc.

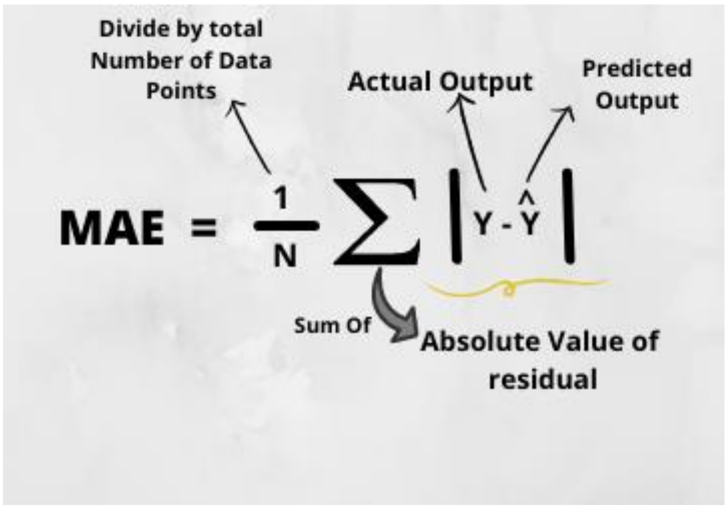
Mean Absolute Error(MAE):

MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

To better understand, let's take an example you have input data and output data and use Linear Regression, which draws a best-fit line.

Now you have to find the MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset.

so, sum all the errors and divide them by a total number of observations And this is MAE. And we aim to get a minimum MAE because this is a loss.


Divide by total Number of Data Points
Actual Output
Predicted Output
MAE = 1/N ∑ |Y - Ŷ|
Sum Of
Absolute Value of residual

**Advantages of MAE:**

- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

**Disadvantages of MAE:**

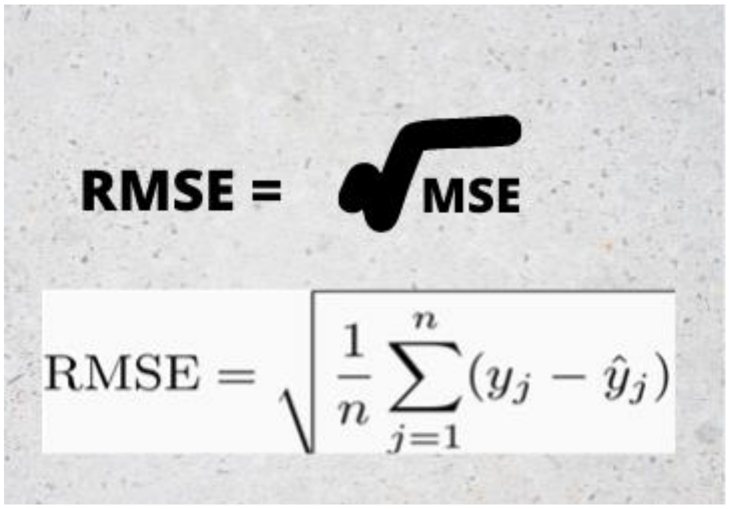
- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

```
from sklearn.metrics import mean_absolute_error
print("MAE",mean_absolute_error(y_test,y_pred))
```

Now to overcome the disadvantage of MAE next metric came as MSE.

Root Mean Squared Error(RMSE):

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.



$$\text{RMSE} = \sqrt{\text{MSE}}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$$

**Advantages of RMSE:**

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

**Disadvantages of RMSE:**

- It is not that robust to outliers as compared to MAE.

for performing RMSE we have to NumPy NumPy square root function over MSE.

```
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

Most of the time people use RMSE as an evaluation metric and mostly when you are working with deep learning techniques the most preferred metric is RMSE.

**CONFUSION MATRIX**

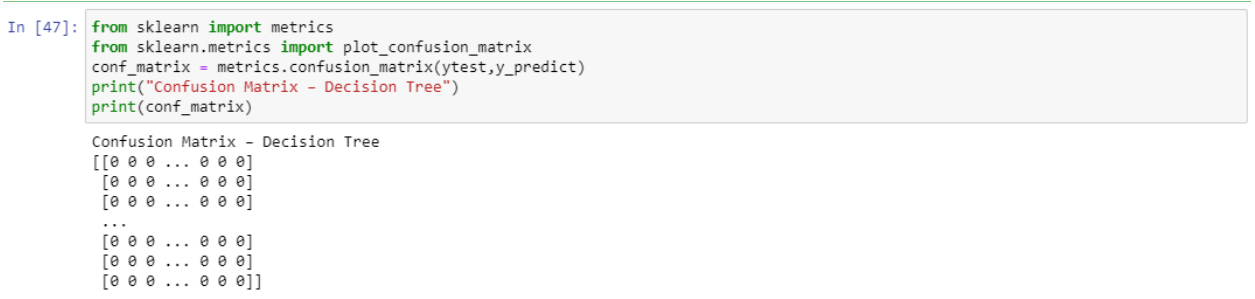
A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

The following 4 are the basic terminology which will help us in determining the metrics we are looking for.

- True Positives (TP):when the actual value is Positive and predicted is also Positive.
- True Negatives (TN): when the actual value is Negative and prediction is also Negative.
- False Positives (FP): When the actual is negative but prediction is Positive. Also known as the Type 1 error
- False Negatives (FN): When the actual is Positive but the prediction is Negative. Also known as the Type 2 error.

Confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance.

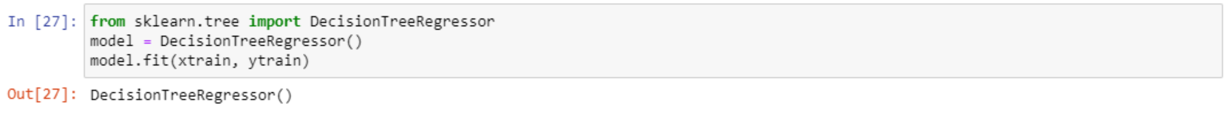
```
from sklearn import metrics

from sklearn.metrics import plot_confusion_matrix

conf_matrix = metrics.confusion_matrix(ytest,y_predict)

print("Confusion Matrix – Decision Tree")

print(conf_matrix)
```



```
In [47]: from sklearn import metrics
         from sklearn.metrics import plot_confusion_matrix
         conf_matrix = metrics.confusion_matrix(ytest,y_predict)
         print("Confusion Matrix - Decision Tree")
         print(conf_matrix)

         Confusion Matrix - Decision Tree
         [[0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          ...
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]
          [0 0 0 ... 0 0 0]]
```

use the decision tree regression algorithm to Evaluate our model

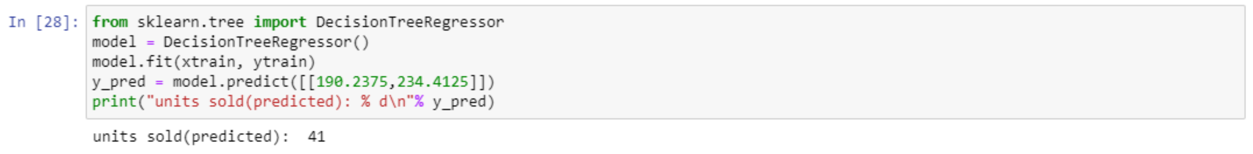
```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
```



```
In [27]: from sklearn.tree import DecisionTreeRegressor
         model = DecisionTreeRegressor()
         model.fit(xtrain, ytrain)

Out[27]: DecisionTreeRegressor()
```

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:

```
model.fit(xtrain, ytrain)

y_pred = model.predict([[190.2375,234.4125]])

print("units sold(predicted): % d\ n"% y_pred)
```



```
In [28]: from sklearn.tree import DecisionTreeRegressor
         model = DecisionTreeRegressor()
         model.fit(xtrain, ytrain)
         y_pred = model.predict([[190.2375,234.4125]])
         print("units sold(predicted): % d\n"% y_pred)

         units sold(predicted):  41
```

Here,by giving the total price and base price we predict the number of units sold. And can predict the demand or products needed by this model.