Project 10: Product Demand Prediction with Machine Learnings

DATA PREPROCESSING

For machine learning algorithms to work, it's necessary to convert raw data into a clean data set, which means we must convert the data set to numeric data. We do this by encoding all the categorical labels to column vectors with binary values. Missing values, or NaNs (not a number) in the data set is an annoying problem. You have to either drop the missing rows or fill them up with a mean or interpolated values.

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
```

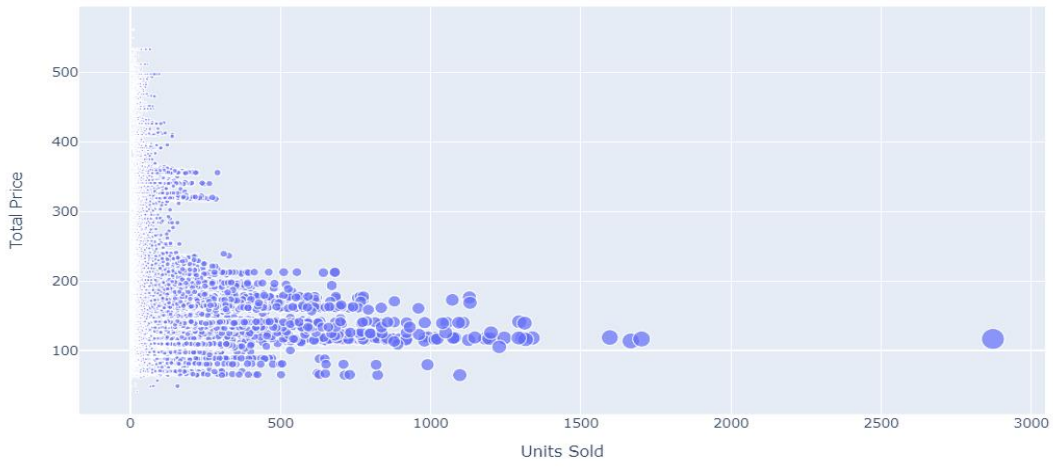
Scatter plot

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()
```

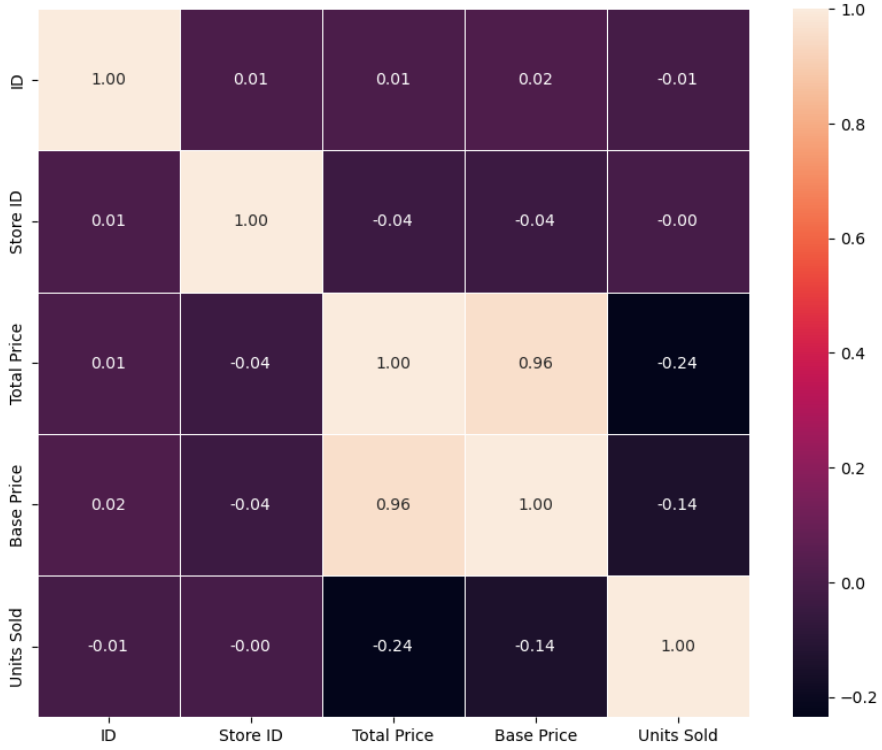
```
correlations = data.corr(method='pearson')

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()
```



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.

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)

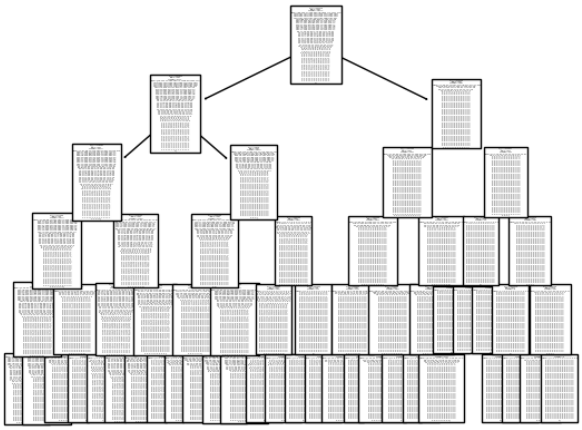
fig=plt.figure()

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 %
```

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]]
```