

Product Demand Analysis (Phase 3)

Abstract:

Supply and demand are two fundamental concepts of sellers and customers. Predicting demand accurately is critical for organisations in order to be able to make plans. The main aim of our project is to develop a machine learning model (Decision Tree Regression, Random Forest Regression) that forecasts product demand based on historical sales data.

Problem Description:

The problem is to create a machine learning model that forecasts product demand based on historical sales data and external factors. The goal is to help businesses optimise inventory management and production planning to efficiently meet customer needs. This project involves data collection, data pre-processing, feature engineering, model selection, training, and evaluation.

3.1 Dataset Information:

The primary goal of this project is to create a predictive model that can estimate the demand for the product in the market under different pricing strategies. By analysing historical sales data, the project aims to identify the price point at which the product is perceived as a better deal compared to competitors, leading to increased sales.

1. **Data Collection:** Historical sales data

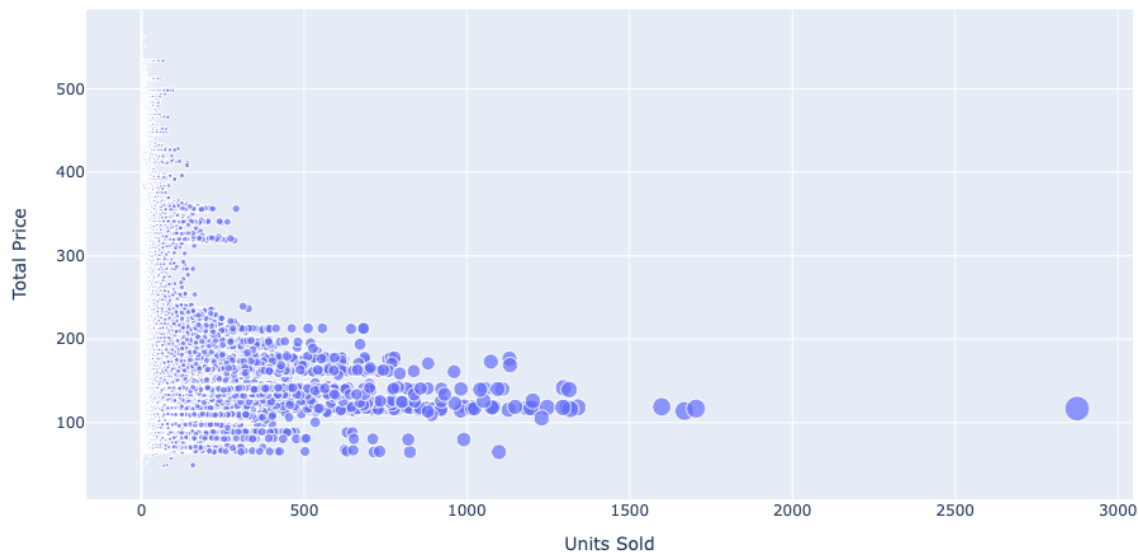
Dataset Link: <https://www.kaggle.com/datasets/chakradharmattapalli/product-demand-prediction-with-machine-learning>

2. **Data Pre-processing:** Clean and pre-process the data, handle missing values, and convert categorical features into numerical representations
3. **Feature Engineering:** Create additional features that capture seasonal patterns, trends, and external influences on product demand.
4. **Model Selection:** Choose suitable regression algorithms (e.g., Decision Tree, Random Forest, Gradient Boost) for demand forecasting.
5. **Model Training:** Train the selected model using the pre-processed data.
6. **Evaluation:** Evaluate the model's performance using appropriate regression metrics (e.g., Mean Absolute Error, Root Mean Squared Error ,R Squared).

Dataset Columns:

The dataset provided for this task includes the following key information:

- **Product ID:** Each product in the company's inventory is assigned a unique identifier, allowing us to distinguish between different items.
- **Store ID:** This represents the specific store or location where the product was sold. It helps in understanding regional variations in demand.
- **Total Price:** This is the actual price at which the product was sold to customers during the sales transaction. It includes any discounts or promotions applied.
- **Base Price:** The base price is the original or standard price at which the product is typically sold before any discounts or promotions are applied.
- **Units Sold (Quantity Demanded):** This is the quantity of the product that was purchased by customers at a given price point. It reflects the demand for the product at that particular price.



Plot of Total Price vs Units Sold

Training Data:

1. Independent Variables (Features):

- Total Price: The total price of a product.
- Base Price: The base price of the product.

2. Dependent Variable (Target):

- Units Sold: The number of units of the product sold.

During the training phase, machine learning model uses "Total Price" and "Base Price" as independent variables to predict the "Units Sold."

3.2 Loading the dataset:

1. Importing the datasets:

Import the dataset using a function in pandas called `pd.read_csv`.

```
data = pd.read_csv("ProductDemand.csv")
```

2. View of the dataset:

By using a special function called `head()`, we will be displaying the first 5 columns and if mentioned any number(`n`) within the parentheses, then 'n' rows of data will be printed.

```
data.head()
```



	ID	Store ID	Total Price	Base Price	Units Sold
0	1	8091	99.0375	111.8625	20
1	2	8091	99.0375	99.0375	28
2	3	8091	133.9500	133.9500	19
3	4	8091	133.9500	133.9500	44
4	5	8091	141.0750	141.0750	52

3. Shape of a dataset:

The shape of the dataset is basically a representation of total rows and columns present in the dataset.

```
data.shape
```

```
(150150, 5)
```

4. Descriptive statistics of the data-sets:

In pandas, `describe()` function is used to view central tendency, mean, median, standard deviation, percentile & many other things to give you the idea about the data.

```
data.describe().transpose()
```



	count	mean	std	min	25%	50%	75%	max
ID	150150.0	106271.555504	61386.037861	1.000	53111.2500	106226.5000	159452.7500	212644.0000
Store ID	150150.0	9199.422511	615.591445	8023.000	8562.0000	9371.0000	9731.0000	9984.0000
Total Price	150149.0	206.626751	103.308516	41.325	130.3875	198.0750	233.7000	562.1625
Base Price	150150.0	219.425927	110.961712	61.275	133.2375	205.9125	234.4125	562.1625
Units Sold	150150.0	51.674206	60.207904	1.000	20.0000	35.0000	62.0000	2876.0000

5. Checking about the correlation between features in a dataset:

`pd.DataFrame.corr()` calculates the correlation between features pairwise excluding null values. A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses.



Heat Map Representation of Correlation Matrix

	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

Correlation Matrix

Insights:

1. **Total Price and Base Price (Correlation: 0.958885)** : Total Price and Base Price have a very strong positive correlation. This means that there is a strong linear relationship between the total price and the base price of the products. As the base price increases, the total price tends to increase accordingly.
2. **Total Price and Units Sold (Correlation: -0.235625)** : Total Price and Units Sold have a moderate negative correlation. This suggests that as the total price of the products increases, the number of units sold tends to decrease. Customers may buy fewer units when the price is higher.
3. **Base Price and Units Sold (Correlation: -0.140022)** : Base Price and Units Sold also have a moderate negative correlation. This indicates that as the base price of the products increases, the number of units sold tends to decrease. This relationship is similar to the one observed with total price.

3.3 Analysis on Dataset:

6. Checking about data types and more information about the data:

`pd.dataFrame.info()` which returns the data type of each column present in the dataset. It tells about null and not null values present.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150150 entries, 0 to 150149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype  
---  -
0   ID               150150 non-null  int64  
1   Store ID        150150 non-null  int64  
2   Total Price     150149 non-null  float64 
3   Base Price      150150 non-null  float64 
4   Units Sold      150150 non-null  int64  
dtypes: float64(2), int64(3)
memory usage: 5.7 MB
```

7. Checking about missing values in the data:

Missing values in the data can be checked by using `isnull()` function present in pandas.

```
data.isnull().sum()
```

```
ID          0
Store ID     0
Total Price  1
Base Price   0
Units Sold   0
dtype: int64
```

Printing the missing row:

```
S = pd.isnull(data['Total Price'])
data[S]
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	NaN	469.5375	1

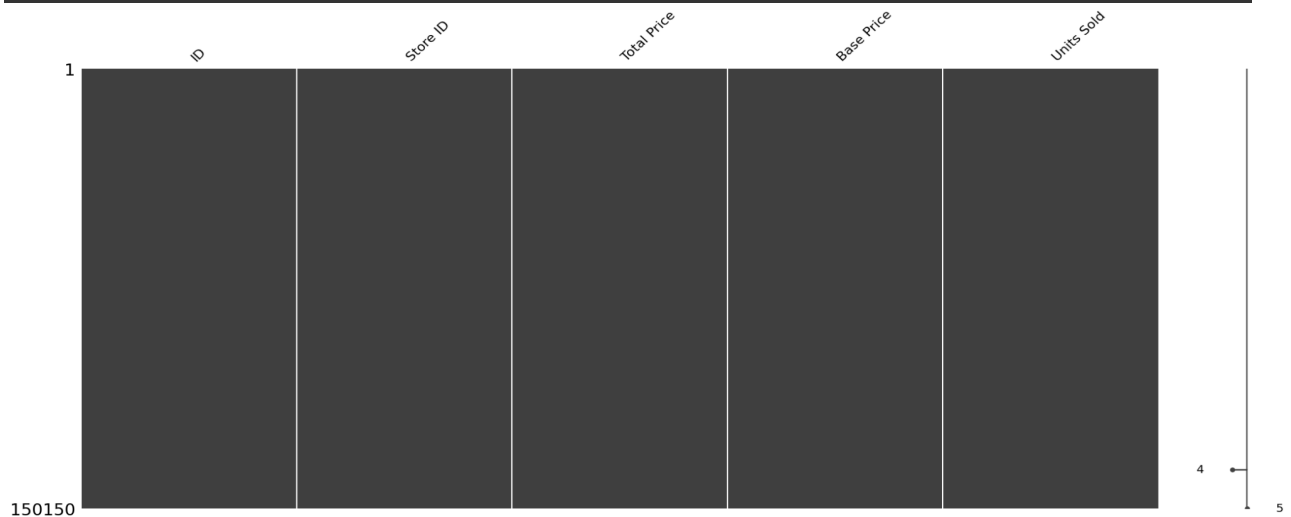
Visualize missing values (NaN) values using Missingno Library:

Missingno library offers a very nice way to visualize the distribution of NaN values. Missingno is a Python library and compatible with Pandas.

Matrix :

Matrix can quickly find the pattern of missingness in the dataset.

```
import missingno as msno
msno.matrix(data)
```

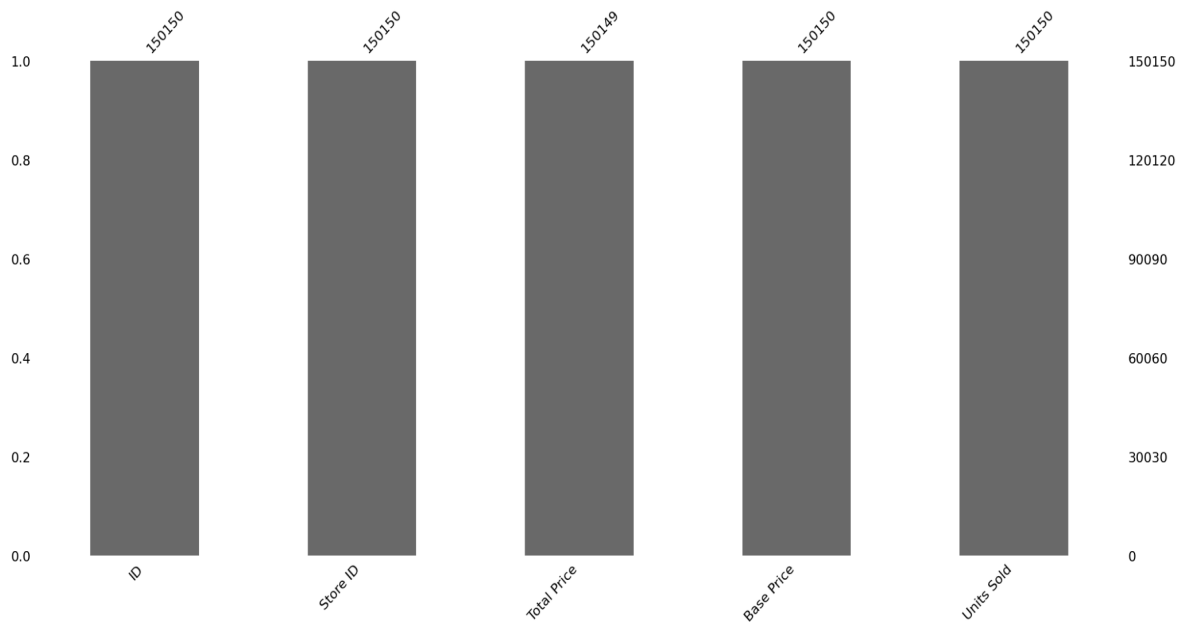


missingno.matrix representation

Bar Chart :

This bar chart gives you an idea about how many missing values are there in each column.

```
msno.bar(data)
```



missingno.bar representation

3.4 Pre-Processing Data

Handling the missing data:

Missing Value

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	NaN	469.5375	1


1.Dropping data:

dropna() function is used to remove missing values from the dataset.

```
data = data.dropna()
```

After removing the null values by checking the shape of the dataset, we come to know that one row has been removed.

```
data.shape
```

 (150149, 5)

2.Replacing with an arbitrary value:

If we can make an educated guess about the missing value, then it can be replaced with some arbitrary value.

```
data['Total Price'] = data['Total Price'].fillna(data['Base Price'])
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	469.5375	469.5375	1

3.Replacing with the Mean:

This is the most common method of imputing missing values of numeric columns. If there are outliers, then the mean will not be appropriate.

```
data['Total Price'] = data['Total Price'].fillna(data['Base Price'].mean())
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	219.425927	469.5375	1

4.Replacing with the Mode:

Mode is the most frequently occurring value. It is used in the case of categorical features.

```
data['Total Price'] = data['Total Price'].fillna(data['Base Price'].mode()[0])
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	205.9125	469.5375	1

5.Replacing with the Median:

The median is the middlemost value. It's better to use the median value for imputation in the case of outliers.


```
data['Total Price'] = data['Total Price'].fillna(data['Base Price'].median())
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	205.9125	469.5375	1

6.Replacing with the Previous value – forward fill:

In some cases, imputing the values with the previous value instead of the mean, mode, or median is more appropriate. This is called forward fill. It is mostly used in time series data.

```
data = data.fillna(method = "ffill")
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	241.5375	469.5375	1

7.Replacing with the Next value – backward fill:

In backward fill, the missing value is imputed using the next value.

```
data = data.fillna(method = "bfill")
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	88.35	469.5375	1

8.Interpolation:

Missing values can also be imputed using interpolation. Pandas' interpolate method can be used to replace the missing values with different interpolation methods like 'polynomial,' 'linear,' and 'quadratic.' The default method is 'linear.'

```
data = data.interpolate()
```

	ID	Store ID	Total Price	Base Price	Units Sold
136949	193915	9436	164.94375	469.5375	1

9. Multivariate Approach:

9.1 Iterative Imputer:

Iterative imputer is a machine learning-based method for imputing missing values in a dataset by iteratively estimating missing values based on other observed features, refining predictions in multiple cycles.

```
X = data[['Total Price', 'Base Price', 'Units Sold']]
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
impute_it = IterativeImputer()
impute_it.fit_transform(X)[136949]
```

```
array([435.53632734, 469.5375, 1.])
```

9.2 KNN Imputer:

Missing values are imputed using the k-Nearest Neighbours approach, where a Euclidean distance is used to find the nearest neighbours.

```
X = data[['Total Price', 'Base Price', 'Units Sold']]
from sklearn.impute import KNNImputer
impute_knn = KNNImputer(n_neighbors=2)
impute_knn.fit_transform(X)[136949]
```

```
array([469.5375, 469.5375, 1.])
```

Feature Scaling:

Feature scaling is a data pre-processing technique used to transform the values of features or variables in a dataset to a similar scale. The purpose is to ensure that all features contribute equally to the model and to avoid the domination of features with larger values.

Feature scaling becomes necessary when dealing with datasets containing features that have different ranges, units of measurement, or orders of magnitude. In such cases, the variation in feature values can lead to biased model performance or difficulties during the learning process

There are several common techniques for feature scaling, including standardization, normalization, and min-max scaling. These methods adjust the feature values while preserving their relative relationships and distributions.

Tree-based algorithms are fairly insensitive to the scale of the features. A decision tree only splits a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. Other features do not influence this split on a feature.

So, the remaining features have virtually no effect on the split. This is what makes them invariant to the scale of the features!

1. **Normalization:** Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization Equation

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

X_{\max} and X_{\min} are the maximum and the minimum values of the feature

2. **Standardization:** Standardization is another scaling method where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero, and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

Normalization also helps to reduce the impact of outliers and improve the accuracy and stability of statistical models.

Standardization:

```
Std = preprocessing.StandardScaler()  
X_train = Std.fit_transform(X_train)  
X_test = Std.fit_transform(X_test)
```

```
array([[ 0.41709875, -1.15840409, -1.19369132],  
       [ 0.36676581, -1.18600747, -1.21938245],  
       [-0.14143448,  0.17345926,  0.04590534],  
       ...,  
       [-1.04580369, -1.3240244 , -1.18084576],  
       [-1.42898346, -0.2819966 ,  0.18078374],  
       [ 0.45444254, -0.14397967, -0.24954257]])
```

Normalization:

```
norm = preprocessing.Normalizer()  
X_train=norm.fit_transform(X_train)  
X_test=norm.fit_transform(X_test)
```

```
array([[0.99945984, 0.0232381 , 0.0232381 ],  
       [0.99937338, 0.0250284 , 0.0250284 ],  
       [0.99945594, 0.01732997, 0.02806224],  
       ...,  
       [0.99862972, 0.03322495, 0.04043248],  
       [0.99738054, 0.04366533, 0.05766624],  
       [0.99984938, 0.01227237, 0.01227237]])
```

Does Random Forest Need Feature Scaling or Normalization?

Normalization and Standardization are not typically required for **Random Forest** models because Random Forest is an ensemble of decision trees that doesn't rely on the scale of features and is robust to outliers. Scaling features can potentially hinder the interpretability and computational efficiency of the model and may not provide any significant improvement in performance.

Conclusion:

In conclusion, the rigorous pre-processing of the dataset, sourced from Kaggle, has rendered it well-suited for the subsequent stages of model development and analysis. Addressing correlations, providing detailed feature descriptions, inspecting data properties, handling missing values, and applying feature scaling techniques were pivotal steps in ensuring data quality and reliability. The attention to detail and rigor applied in the pre-processing phase will ultimately contribute to the success and accuracy of our machine learning endeavours.