

Aim: Introduction to Data science and Data preparation using Pandas steps.

Theory:

For machine learning algorithms to perform effectively, it is essential to preprocess raw data and convert it into a clean, usable dataset. This often involves encoding categorical data into numerical representations since most algorithms require numeric input. One common approach is to transform categorical labels into column vectors with binary values, a process known as one-hot encoding. This ensures that the categorical data is represented in a way that the algorithm can interpret without assigning implicit ordinal relationships between categories.

Another critical preprocessing step is handling missing values, which are often represented as NaNs (Not a Number). Missing values can arise from various issues, such as incomplete data collection, sensor malfunctions, or user input errors. These missing entries can lead to errors or biases in the training process if not addressed properly.

Problem Statement

The given dataset provides comprehensive details about retail product sales, focusing on the relationship between product attributes, outlet characteristics, and sales performance. This analysis aims to address the following key objectives:

- **Product Performance:** Identifying products or product types that drive the highest sales and those underperforming.
- **Outlet Insights:** Understanding the impact of outlet size, location, and type on overall sales performance.
- **Pricing Analysis:** Investigating how pricing strategies, such as maximum retail price (MRP), influence customer purchasing behavior.
- **Possible sales Prediction:** Developing models to predict item-level sales and provide actionable insights for inventory and pricing strategies.

By preprocessing the dataset and applying statistical analysis, the goal is to extract meaningful patterns that can guide data-driven decisions in retail operations.

Dataset Overview

The dataset comprises **12 columns**, each detailing specific aspects of retail products, their pricing, visibility, and sales performance across outlets. Below is a breakdown of the dataset's columns and their relevance:

1. **Item_Identifier:** A unique code for each product, essential for distinguishing between items in the inventory.

2. **Item_Weight**: Represents the weight of each product, which may impact logistics and consumer preference.
3. **Item_Fat_Content**: Categorizes items as Low Fat or Regular Fat, reflecting their nutritional value and target audience.
4. **Item_Visibility**: A measure of the shelf visibility of an item, influencing its likelihood of being purchased.
5. **Item_Type**: Broad categories like Dairy, Beverages, or Snacks, helping analyze trends across product types.
6. **Item_MRP**: The maximum retail price, a key factor in determining product affordability and customer demand.
7. **Outlet_Identifier**: A unique code assigned to each retail outlet, linking products to specific store locations.
8. **Outlet_Establishment_Year**: Indicates the year the outlet began operations, useful for analyzing store maturity's effect on sales.
9. **Outlet_Size**: Categorizes stores as Small, Medium, or Large, impacting foot traffic and product demand.
10. **Outlet_Location_Type**: Describes whether the store is Urban, Suburban, or in a Tier 3 area, capturing demographic influences.
11. **Outlet_Type**: Differentiates between store types, such as Grocery Stores or Supermarkets, reflecting varying business models.
12. **Item_Outlet_Sales**: The target variable, representing the total sales of a product at a specific outlet.

Workflow: Combining NumPy and Pandas

The typical data preparation workflow involves using NumPy for low-level numerical operations and array manipulations and Pandas for higher-level data manipulation and analysis tasks. For instance, you might use NumPy to handle missing values at a lower level, and then switch to Pandas to leverage its powerful tools for cleaning, transforming, and exploring tabular data.

OUTPUT :

1. Load data in pandas

Loading data into Pandas is a crucial initial step in the data analysis process. The pandas library provides functions like `read_csv()`, `read_excel()`, or `read_sql()` to read data from different file formats or databases into a Pandas DataFrame. Once loaded, a DataFrame provides a structured and flexible representation of the data, making it easy to perform various operations.

```
[2] import pandas as pd
     df = pd.read_csv("/content/market_data.csv")
```

2. Description of the dataset.

Understanding the structure and characteristics of the dataset is important for effective analysis. The `info()`, `describe()`, and `head()` methods in Pandas provide information about the data types, summary statistics, and a preview of the dataset, respectively.

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	1716.97
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	903.66
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	165.00
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	141.67
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	137.36

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
#   Column                               Non-Null Count  Dtype  
---  -
0   Item_Identifier                      8523 non-null   object  
1   Item_Weight                         7060 non-null   float64  
2   Item_Fat_Content                    8523 non-null   object  
3   Item_Visibility                     8523 non-null   float64  
4   Item_Type                           8523 non-null   object  
5   Item_MRP                           8523 non-null   float64  
6   Outlet_Identifier                   8523 non-null   object  
7   Outlet_Establishment_Year           8523 non-null   int64  
8   Outlet_Size                         6113 non-null   object  
9   Outlet_Location_Type                8523 non-null   object  
10  Outlet_Type                         8523 non-null   object  
11  Item_Outlet_Sales                   8523 non-null   float64  
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

3. Drop columns that aren't useful

Not all columns in a dataset are relevant for analysis or modeling. You can use the `drop()` method to eliminate columns that are not useful.

```
[ ] cols = ['Outlet_Establishment_Year']
df = df.drop(cols, axis=1)
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 11 columns):
#   Column                               Non-Null Count  Dtype  
---  -
0   Item_Identifier                      8523 non-null   object  
1   Item_Weight                         7060 non-null   float64  
2   Item_Fat_Content                    8523 non-null   object  
3   Item_Visibility                     8523 non-null   float64  
4   Item_Type                           8523 non-null   object  
5   Item_MRP                           8523 non-null   float64  
6   Outlet_Identifier                   8523 non-null   object  
7   Outlet_Size                         6113 non-null   object  
8   Outlet_Location_Type                8523 non-null   object  
9   Outlet_Type                         8523 non-null   object  
10  Item_Outlet_Sales                   8523 non-null   float64  
dtypes: float64(4), object(7)
memory usage: 732.6+ KB
```

4. Drop rows with maximum missing values

Handling missing data is crucial. You can identify and drop rows with a significant number of missing values using the `dropna()` method.

```
[ ] df.size
```

```
↔ 102276
```

```
[ ] df = df.dropna()
```

```
▶ df.size
```

```
↔ 73356
```

5. Take care of missing data

For columns with missing values, you can choose to fill them with a specific value or use methods like mean, median, or interpolation to impute missing values.

```
✓ [3] df['Item_Weight'] = df['Item_Weight'].fillna(df['Item_Weight'].mean())
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

6. Create dummy variables

For categorical variables, creating dummy variables is essential for including them in machine learning models.

```
dummy_data = {
    "Item_Identifier": "FDX01",
    "Item_Weight": 12.5,
    "Item_Fat_Content": "Low Fat",
    "Item_Visibility": 0.05,
    "Item_Type": "Snack Foods",
    "Item_MRP": 250.75,
    "Outlet_Identifier": "OUT013",
    "Outlet_Size": "Medium",
    "Outlet_Location_Type": "Tier 1",
    "Outlet_Type": "Supermarket Type1",
    "Item_Outlet_Sales": 3400.25
}
new_row = pd.DataFrame([dummy_data])
result = pd.concat([df, new_row], ignore_index=True)
```

```
result.tail()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
6109	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987.0	High	Tier 3	Supermarket Type1	
6110	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004.0	Small	Tier 2	Supermarket Type1	
6111	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009.0	Medium	Tier 3	Supermarket Type2	
6112	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997.0	Small	Tier 1	Supermarket Type1	
6113	FDX01	12.500	Low Fat	0.050000	Snack Foods	250.7500	OUT013	NaN	Medium	Tier 1	Supermarket Type1	

7. Find out outliers (manually):

Outliers can be detected visually or using statistical methods. One of the methods I have applied is the IQR method.

The Interquartile Range (IQR) helps determine the spread of middle values in a dataset:

Q1 (25th percentile) → The value below which 25% of the data falls.

Q3 (75th percentile) → The value below which 75% of the data falls.

$IQR = Q3 - Q1$ → This represents the range of the middle 50% of the data.

Outliers are values that fall outside the following range:

- **Lower Bound** = $Q1 - 1.5 \times IQR$
- **Upper Bound** = $Q3 + 1.5 \times IQR$

```
def remove_outliers_iqr(df, column):  
    Q1 = df[column].quantile(0.25)  
    Q3 = df[column].quantile(0.75)  
    IQR = Q3 - Q1  
  
    lower_bound = Q1 - 1.5 * IQR # Lower threshold  
    upper_bound = Q3 + 1.5 * IQR # Upper threshold  
  
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]  
  
#applicable to numerical columns  
numerical_columns = ["Item_Weight", "Item_Visibility", "Item_MRP", "Item_Outlet_Sales"]  
df_cleaned = df.copy()  
  
for col in numerical_columns:  
    df_cleaned = remove_outliers_iqr(df_cleaned, col)
```

```
df.shape[0], df_cleaned.shape[0]
```

```
(6113, 5897)
```

After calculating the outliers manually we get the dataset values to be reduced to 5897 rows

Outliers can distort predictions and **reduce** model accuracy **hence not every time** but sometimes it should be removed for better accuracy.

8. Standardization of columns

In statistics and machine learning, data **standardization** is a **process of converting data to z-score values based on the mean and standard deviation of the data.**

The resulting standardized value shows the number of standard deviations the raw value is away from the mean. Basically, each value of a given feature of a dataset will be converted to a representative number of standard deviations that it's away from the mean of the feature

9. Normalization of columns

In statistics and machine learning, **the min-max normalization of data** is a **process of converting the original range of data to the range between 0 and 1.** The resulting normalized

values represent the original data on a 0-1 scale. This will allow us to compare multiple features together and get more relevant information since now all the data will be on the same scale.

```

▶ from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas as pd

numerical_columns = ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Item_Outlet_Sales']

# Initialize scalers
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()

# Standardization of the data
standardized_data = standard_scaler.fit_transform(df[numerical_columns])
df_standardized = pd.DataFrame(standardized_data, columns=numerical_columns)

# Normalization of the data
normalized_data = minmax_scaler.fit_transform(df[numerical_columns])
df_normalized = pd.DataFrame(normalized_data, columns=numerical_columns)

print("Standardized Data:")
print(df_standardized.head())

print("\nNormalized Data:")
print(df_normalized.head())

```

```

⇒ Standardized Data:
   Item_Weight  Item_Visibility  Item_MRP  Item_Outlet_Sales
0   -0.881033   -0.967450    1.744524     0.811077
1   -1.710793   -0.902945   -1.494387    -1.079139
2    1.131996   -0.953220    0.005804    -0.129443
3   -0.971864   -1.287832   -1.404516    -0.762573
4   -0.612220   -1.287832   -1.444060    -1.014143

Normalized Data:
   Item_Weight  Item_Visibility  Item_MRP  Item_Outlet_Sales
0    0.282525    0.048866    0.927507     0.283550
1    0.081274    0.058705    0.072068     0.031370
2    0.770765    0.051037    0.468288     0.158072
3    0.260494    0.000000    0.095805     0.073604
4    0.347723    0.000000    0.085361     0.040041

```

Conclusion: In conclusion, the series of experiments performed on data preparation using NumPy and Pandas, along with various data preprocessing techniques, is essential for ensuring the data is in a suitable and meaningful form for analysis or machine learning applications. Here we used the retail sales data to understand the max profit of each product.