
FUTURE SALES PREDICTION WITH
MACHINE LEARNING

PROBLEM STATEMENT

In today's highly competitive business landscape, accurate sales prediction is critical for organizations to optimize resource allocation, inventory management, and overall business strategy. However, traditional sales forecasting methods often fall short in addressing the dynamic and complex nature of markets, making it challenging to meet sales targets efficiently.

Design Thinking in Sales Prediction

User-Centric Data Collection

The first step in applying design thinking to sales prediction is to collect user-centric data. This includes not only historical sales data but also feedback from sales teams, customers, and other stakeholders. Understanding the challenges faced in the field and the specific questions that need to be answered is essential.

DEFINE

Research in the field of future sales prediction using machine learning is crucial for several reasons. Accurate sales prediction has a profound impact on businesses, influencing decisions related to inventory management, resource allocation, marketing strategies, and overall financial planning. Here are some key research needs in this domain:

- **Algorithm Development and Enhancement**
- **Feature Engineering**
- **Handling Seasonality and Trends**
- **Real-time Prediction**
- **Datasets**
- **Caesstudies and practice**

Understanding user needs and problems in future sales prediction is essential for developing effective solutions that meet the expectations of various stakeholders. Here are some common user needs and problems in this domain:

Accurate Sales Forecasts: Users require accurate predictions of future sales to make informed decisions about inventory management, production planning, and resource allocation. The primary need is for predictions that closely align with actual sales.

Timeliness: Users need timely sales forecasts to respond to market changes, seasonal fluctuations, and emerging trends. Delayed predictions can result in missed opportunities or overstocking/understocking issues.

Customization: Different users may have unique requirements for sales predictions. Sales teams, for example, may need forecasts tailored to specific product categories or regions. Users need customizable solutions that address their specific needs.

Interpretability: Users, especially those without a deep understanding of machine learning, require interpretable predictions. They need to understand why a particular prediction was made and what factors contributed to it.

Actionable Insights: Sales predictions should not just be numbers; they should provide actionable insights. Users need guidance on how to adjust marketing strategies, pricing, and inventory based on the predictions.

Adaptability: Users need models that can adapt to changing market conditions and evolving customer behavior. Predictions should remain accurate even as external factors change.

USER FACING PROBLEMS

1.Data Quality and Integration: Users often face challenges related to data quality, missing data, and integrating diverse data sources. Inaccurate or incomplete data can lead to unreliable predictions.

2.Model Complexity: Users may struggle to understand and trust highly complex machine learning models. They need models that strike a balance between accuracy and interpretability.

3.Overfitting: Overfitting is a common problem where models perform well on training data but poorly on new data. Users need models that generalize effectively to unseen data.

4.Seasonality and Trends: Handling seasonality, trends, and cyclic patterns in sales data can be challenging. Users need models that can capture and predict these patterns accurately.

5.Resource Constraints: Users may have limited computational resources or budget constraints for implementing advanced machine learning solutions. Cost-effective approaches are needed.

6.Privacy and Ethics: Users must navigate data privacy and ethical considerations, especially when collecting and using customer data for predictions. Compliance with regulations is critical.

Challenges and assumptions play a crucial role in the development and deployment of future sales prediction models using machine learning. Identifying these challenges and making explicit assumptions is essential for building robust and effective predictive systems. Here are some challenges and assumptions to conside

- IDEATE
  - DATA VOLUME
  - DATA QUALITY
  - DATA INTEGRATION
  - DATA INTERGITY
  - REAL TIME PREDICTIONS
  - SEASONALITY CHANGES
  - ALGORITHM SELECTION

- CREATING PROTOTYPE
  - DATA COLLECTION
  - DATA PRE PROCESSING
    - DATA CLEANING
    - DATA AGGREGATION
    - DATA VISUVALIZATION
  - UNDERSTANDING REALTIONALITIES IN VARIABLES
  - IMPLEMENTING IN PROGRAM
  - MODEL SELECTION
  - TESTING THE MODEL ACCURACY

Testing machine learning models is a critical step in the model development process to ensure that the model performs well, is reliable, and generalizes effectively to unseen data. Here are the key steps and techniques for testing machine learning models:

- DATA SPLLITING
- EVALUVATION MATRIX
- BSAELINE MODEL
- CROSS VALIDATION
- VISUVALAZTION AND PLOTS
- MODEL COMPARISON
- OVERFITTING AND UNDERFITTING ANALYSIS
-

```
#!pip install pandas numpy seaborn matplotlib klib dtale scikit-learn joblib pandas-profiling
```

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df_train= pd.read_csv(r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\Dataset\train.csv')
df_test= pd.read_csv(r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\Dataset\test.csv')
```

```
df_train.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_S
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Med
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Med
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Med
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	H

```
#df_test
```

```
df_train.shape
```

```
(8523, 12)
```

```
df_train.isnull().sum()
```

```
Item_Identifier               0
Item_Weight                1463
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                2410
Outlet_Location_Type          0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

```
df_test.isnull().sum()
```

```
Item_Identifier               0
Item_Weight                 976
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                1606
Outlet_Location_Type          0
Outlet_Type                   0
dtype: int64
```

```
df_train.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
```

```
df_train.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

Item_Weight is numerical column so we fill it with Mean Imputation

```
df_train['Item_Weight'].describe()
```

```
count    7060.000000
mean       12.857645
std         4.643456
min         4.555000
25%         8.773750
50%        12.600000
75%        16.850000
max        21.350000
Name: Item_Weight, dtype: float64
```

```
df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

```
df_train.isnull().sum()
```

```
Item_Identifier                0
Item_Weight                    0
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Size                 2410
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64
```

```
df_train['Item_Weight'].describe()
```

```
count    8523.000000
mean       12.857645
std         4.226124
min         4.555000
25%         9.310000
50%        12.857645
75%        16.000000
max        21.350000
Name: Item_Weight, dtype: float64
```

Outlet_Size is catagorical column so we fill it with Mode Imputation

```
df_train['Outlet_Size'].value_counts()
```

```
Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64
```

```
df_train['Outlet_Size'].mode()
```

```
0    Medium
dtype: object
```

```
df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
```

```
df_train.isnull().sum()
```

```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

```
df_test.isnull().sum()
```

```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
dtype: int64
```

Selecting features based on general requirements

```
df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

```
df_train
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	O
0	9.300	Low Fat	0.016047	Dairy	249.8092	1999	Medium	Tier 1	Superm
1	5.920	Regular	0.019278	Soft Drinks	48.2692	2009	Medium	Tier 3	Superm
2	17.500	Low Fat	0.016760	Meat	141.6180	1999	Medium	Tier 1	Superm
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	1998	Medium	Tier 3	G
4	8.930	Low Fat	0.000000	Household	53.8614	1987	High	Tier 3	Superm
...	...	...	...	...	...	...	...	...	
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	1987	High	Tier 3	Superm
8519	8.380	Regular	0.046982	Baking Goods	108.1570	2002	Medium	Tier 2	Superm
8520	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	2004	Small	Tier 2	Superm
8521	7.210	Regular	0.145221	Snack Foods	103.1332	2009	Medium	Tier 3	Superm
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	1997	Small	Tier 1	Superm

EDA with Dtale Library

```
import dtale
```

```
C:\Users\thero\Anaconda3\lib\site-packages\dtale\dash_application\charts.py:13: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
C:\Users\thero\Anaconda3\lib\site-packages\dtale\dash_application\charts.py:14: UserWarning:
The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`
  import dash_html_components as html
```

```
dtale.show(df_train)
```

```
2021-10-18 12:57:23,097 - INFO     - NumExpr defaulting to 8 threads.
```

EDA using Pandas Profiling

```
from pandas_profiling import ProfileReport
```

```
profile = ProfileReport(df_train, title="Pandas Profiling Report")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-b38bd2f54af1> in <cell line: 1>()
```

```
profile
```

Overview

Overview Reproduction Warnings 1

Dataset statistics

Number of variables	10
Number of observations	8523
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	3.0 MiB
Average record size in memory	371.1 B

Variable types

NUM	5
CAT	5

Variables

Item_Weight

```
plt.figure(figsize=(10,5))
sns.heatmap(df_train.corr(),annot=True)
plt.show()
```

```
C:\Users\thero\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: UserWarning:

Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure.
```

EDA using Klib Library

```
import klib
```

```
# klib.describe - functions for visualizing datasets
klib.cat_plot(df_train) # returns a visualization of the number and frequency of categorical features
```

```
GridSpec(6, 5)
```

```
klib.corr_mat(df_train) # returns a color-encoded correlation matrix
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.00	-0.01	0.02	-0.01	0.01
Item_Visibility	-0.01	1.00	-0.00	-0.07	-0.13
Item_MRP	0.02	-0.00	1.00	0.01	0.57
Outlet_Establishment_Year	-0.01	-0.07	0.01	1.00	-0.05

```
klib.corr_plot(df_train) # returns a color-encoded heatmap, ideal for correlations
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1daa3b33ec8>
```

```
klib.dist_plot(df_train) # returns a distribution plot for every numeric feature
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1daa9e7d688>
```

```
klib.missingval_plot(df_train) # returns a figure containing information about missing values
```

```
No missing values found in the dataset.
```

Data Cleaning using Klib Library

```
# klib.clean - functions for cleaning datasets
klib.data_cleaning(df_train) # performs datacleaning (drop duplicates & empty rows/cols, adjust dtypes,...)
```

```
Shape of cleaned data: (8523, 10)Remaining NAs: 0

Changes:
Dropped rows: 0
     of which 0 duplicates. (Rows: [])
Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.08 MB (-12.31%)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_
0	9.300000	Low Fat	0.016047	Dairy	249.809204	
1	5.920000	Regular	0.019278	Soft Drinks	48.269199	
2	17.500000	Low Fat	0.016760	Meat	141.617996	
3	19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	
4	8.930000	Low Fat	0.000000	Household	53.861401	
...	...	...	...	...	...	
8518	6.865000	Low Fat	0.056783	Snack Foods	214.521805	
8519	8.380000	Regular	0.046982	Baking Goods	108.156998	
8520	10.600000	Low Fat	0.035186	Health and Hygiene	85.122398	
8521	7.210000	Regular	0.145221	Snack Foods	103.133202	
8522	14.800000	Low Fat	0.044878	Soft Drinks	75.467003	

8523 rows × 10 columns

```
klib.clean_column_names(df_train) # cleans and standardizes column names, also called inside data_cleaning()
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_y
0	9.300	Low Fat	0.016047	Dairy	249.8092	19
1	5.920	Regular	0.019278	Soft Drinks	48.2692	20
2	17.500	Low Fat	0.016760	Meat	141.6180	19
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	19
4	8.930	Low Fat	0.000000	Household	53.8614	19
...	...	...	...	...	...	
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	19
8519	8.380	Regular	0.046982	Baking Goods	108.1570	20
8520	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	20
8521	7.210	Regular	0.145221	Snack Foods	103.1332	20
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	19

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float64
 1   item_fat_content           8523 non-null   object
 2   item_visibility            8523 non-null   float64
 3   item_type                  8523 non-null   object
 4   item_mrp                   8523 non-null   float64
 5   outlet_establishment_year  8523 non-null   int64
 6   outlet_size                8523 non-null   object
 7   outlet_location_type       8523 non-null   object
 8   outlet_type                8523 non-null   object
 9   item_outlet_sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(5)
memory usage: 666.0+ KB
```

```
df_train=klib.convert_datatypes(df_train) # converts existing to more efficient dtypes, also called inside data_cleaning()
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float32
 1   item_fat_content           8523 non-null   category
 2   item_visibility            8523 non-null   float32
 3   item_type                  8523 non-null   category
 4   item_mrp                   8523 non-null   float32
 5   outlet_establishment_year  8523 non-null   int16
 6   outlet_size                8523 non-null   category
 7   outlet_location_type       8523 non-null   category
 8   outlet_type                8523 non-null   category
 9   item_outlet_sales          8523 non-null   float32
dtypes: category(5), float32(4), int16(1)
memory usage: 192.9 KB
```

```
klib.mv_col_handling(df_train)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_
0	9.300000	Low Fat	0.016047	Dairy	249.809204	
1	5.920000	Regular	0.019278	Soft Drinks	48.269199	
2	17.500000	Low Fat	0.016760	Meat	141.617996	
3	19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	
4	8.930000	Low Fat	0.000000	Household	53.861401	
...	...	...	...	...	...	
8518	6.865000	Low Fat	0.056783	Snack Foods	214.521805	
8519	8.380000	Regular	0.046982	Baking Goods	108.156998	

Preprocessing Task before Model Building

Foods

1) Label Encoding

8522 rows × 10 columns

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
df_train['item_fat_content']= le.fit_transform(df_train['item_fat_content'])
df_train['item_type']= le.fit_transform(df_train['item_type'])
df_train['outlet_size']= le.fit_transform(df_train['outlet_size'])
df_train['outlet_location_type']= le.fit_transform(df_train['outlet_location_type'])
df_train['outlet_type']= le.fit_transform(df_train['outlet_type'])
```

```
df_train
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_
0	9.300000	1	0.016047	4	249.809204	
1	5.920000	2	0.019278	14	48.269199	
2	17.500000	1	0.016760	10	141.617996	
3	19.200001	2	0.000000	6	182.095001	
4	8.930000	1	0.000000	9	53.861401	
...	...	...	...	...	...	
8518	6.865000	1	0.056783	13	214.521805	
8519	8.380000	2	0.046982	0	108.156998	
8520	10.600000	1	0.035186	8	85.122398	
8521	7.210000	2	0.145221	13	103.133202	
8522	14.800000	1	0.044878	14	75.467003	

8522 rows × 10 columns

2) Splitting our data into train and test

```
X=df_train.drop('item_outlet_sales',axis=1)
```

```
Y=df_train['item_outlet_sales']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=101, test_size=0.2)
```

3) Standarization

```
X.describe()
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_typ
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.00000
mean	12.858088	1.369354	0.066132	7.226681	140.992767	1997.831867	1.170832	1.112871	1.20122
std	4.226130	0.644810	0.051598	4.209990	62.275051	8.371760	0.600327	0.812757	0.79645
min	4.555000	0.000000	0.000000	0.000000	31.290001	1985.000000	0.000000	0.000000	0.00000
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	0.000000	1.00000
50%	12.857645	1.000000	0.053931	6.000000	143.012802	1999.000000	1.000000	1.000000	1.00000
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	2.000000	1.00000
max	21.350000	4.000000	0.328391	15.000000	266.888397	2009.000000	2.000000	2.000000	3.00000

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
```

```
X_train_std= sc.fit_transform(X_train)
```

```
X_test_std= sc.transform(X_test)
```

```
X_train_std
```

```
array([[ 1.52290023, -0.57382672,  0.68469731, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.239856  , -0.57382672, -0.09514746, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 1.54667619,  0.97378032, -0.0083859 , ..., -0.28872895,
        -0.13870429, -0.25964107],
       ...,
       [-0.08197109, -0.57382672, -0.91916229, ...,  1.37953713,
        -1.36527477, -0.25964107],
       [-0.74888436,  0.97378032,  1.21363045, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 0.67885675, -0.57382672,  1.83915361, ..., -0.28872895,
         1.08786619,  0.98524841]])
```

```
X_test_std
```

```
array([[-0.43860916, -0.57382672, -0.21609253, ..., -0.28872895,
         1.08786619,  0.98524841],
       [ 1.22570184, -0.57382672, -0.52943464, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.2184578 ,  0.97378032,  0.16277341, ...,  1.37953713,
        -1.36527477, -0.25964107],
       ...,
       [ 0.65508101, -0.57382672,  0.8782423 , ..., -0.28872895,
         1.08786619, -1.50453056],
       [ 1.01171909, -0.57382672, -1.28409256, ..., -0.28872895,
         1.08786619,  0.98524841],
       [-1.56558541,  0.97378032, -1.09265374, ..., -0.28872895,
        -0.13870429, -0.25964107]])
```

```
Y_train
```

```
3684     163.786804
1935    1607.241211
5142    1510.034424
4978    1784.343994
2299    3558.035156
           ...
599     5502.836914
5695    1436.796387
8006    2167.844727
1361    2700.484863
1547     829.586792
Name: item_outlet_sales, Length: 6818, dtype: float32
```

```
Y_test
```

```
8179     904.822205
8355    2795.694092
3411    1947.464966
7089     872.863770
6954    2450.144043
           ...
1317    1721.093018
4996     914.809204
531      370.184814
3891    1358.232056
6629    2418.185547
Name: item_outlet_sales, Length: 1705, dtype: float32
```

```
import joblib
```

```

```

Loading and preprocessing the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
```

```
df=pd.read_csv("C:\\Users\\HARIHARAPRASAD R\\Downloads\\archive (8)\\Train.csv")
```

```
df.isnull().sum()
```

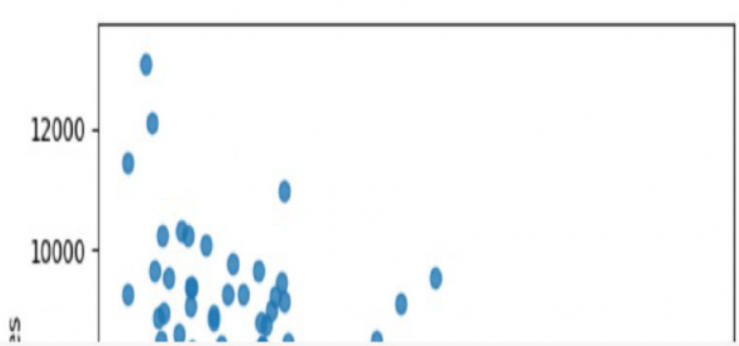
```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

```
df.fillna(method="ffill")#cleaning & handling
missing data
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outle
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	
...	...	...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	2002	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	

8523 rows × 12 columns

```
sns.regplot(x=df['Item_Visibility'],y=df['Item_Outlet_Sales'],fit_reg=False)
```



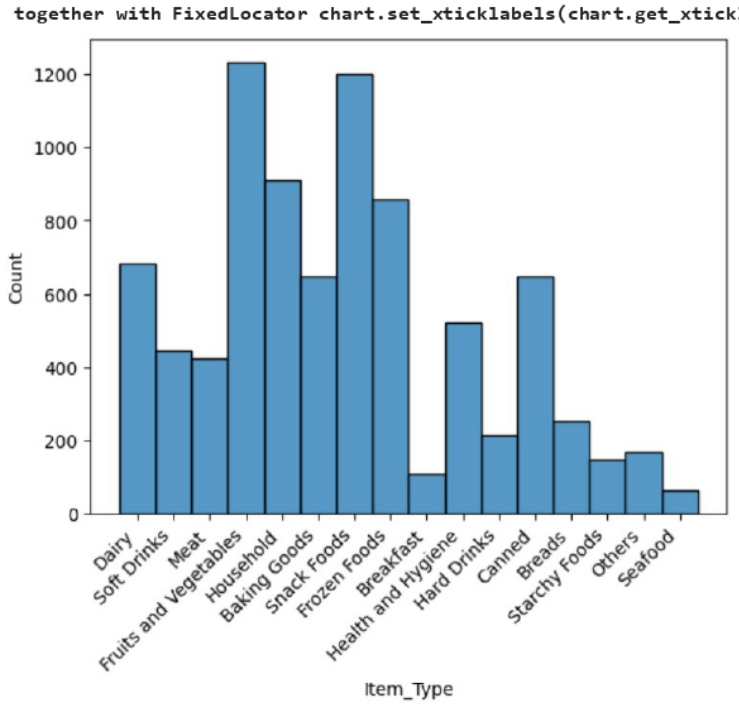
```
<Axes: xlabel='Item_Visibility', ylabel='Item_Outlet_Sales'>
```

```
print('product type and number of sales ')
print(df['Item_Type'].unique())
df['Item_Type'].value_counts()
```

```
product type and number of sales
['Dairy' 'Soft Drinks' 'Meat' 'Fruits and Vegetables' 'Household'
 'Baking Goods' 'Snack Foods' 'Frozen Foods' 'Breakfast'
 'Health and Hygiene' 'Hard Drinks' 'Canned' 'Breads' 'Starchy Foods'
 'Others' 'Seafood']
Item_Type
Fruits and Vegetables    1232
Snack Foods              1200
Household                 910
Frozen Foods              856
Dairy                     682
Canned                    649
Baking Goods              648
Health and Hygiene        520
Soft Drinks               445
Meat                      425
Breads                    251
Hard Drinks               214
Others                    169
Starchy Foods             148
Breakfast                 110
Seafood                    64
Name: count, dtype: int64
```

```
chart=sns.histplot(x='Item_Type',data=df,)
chart.set_xticklabels(chart.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.show()
```

```
C:\Users\HARIHARAPRASAD R\AppData\Local\Temp\ipykernel_7368\2574513022.py:2: UserWarning: FixedFormatter should only be used
  together with FixedLocator chart.set_xticklabels(chart.get_xticklabels(), rotation=45, horizontalalignment='right')
```




```
print('item_type vs outlet_sales')
print(pd.crosstab(index=df['Item_Type'],columns='Item_Outlet_Sales',dropna=True,))
print("\njoint probalities of the item type and item_outletsales")
print(pd.crosstab(index=df['Item_Type'],columns='Item_Outlet_Sales',dropna=True,normalize=True))
```

```
item_type vs outlet_sales
col_0                Item_Outlet_Sales
```

```
Item_Type
Baking Goods                    648
Breads                          251
Breakfast                       110
Canned                          649
Dairy                           682
Frozen Foods                    856
Fruits and Vegetables          1232
Hard Drinks                     214
Health and Hygiene              520
Household                       910
Meat                            425
Others                          169
Seafood                          64
Snack Foods                     1200
Soft Drinks                     445
Starchy Foods                   148

joint probalities of the item type and item_outletsales
col_0               Item_Outlet_Sales
Item_Type
Baking Goods                0.076030
Breads                      0.029450
Breakfast                   0.012906
Canned                      0.076147
Dairy                       0.080019
Frozen Foods                0.100434
Fruits and Vegetables       0.144550
Hard Drinks                 0.025109
Health and Hygiene          0.061011
Household                   0.106770
Meat                        0.049865
Others                      0.019829
Seafood                     0.007509
Snack Foods                 0.140795
Soft Drinks                 0.052212
Starchy Foods               0.017365
```

```
chart1=sns.histplot(x=df['Item_Type'],y=df['Item_Outlet_Sales'],kde_kws=True)
chart1.set_xticklabels(chart1.get_xticklabels(), rotation=45, horizontalalignment='right')

plt.show()
print("\nsales describtion ")
print(df['Item_Outlet_Sales'].describe())
```

```
print('location wise sales')
pd.crosstab(index=df['Outlet_Location_Type'],columns='item_outlet_sales',dropna=True)
```

```
    location wise sales
```

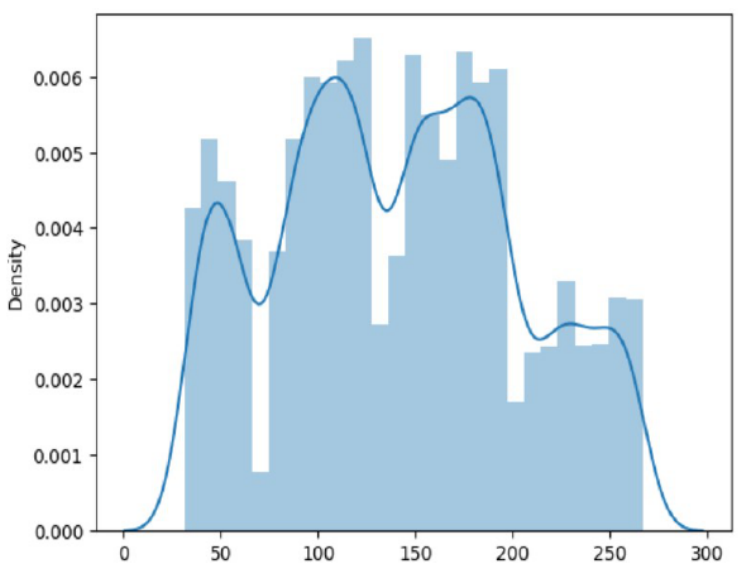
Outlet_Location_Type	col_0item_outlet_sales
Tier 1	2388
Tier 2	2785
Tier 3	3350

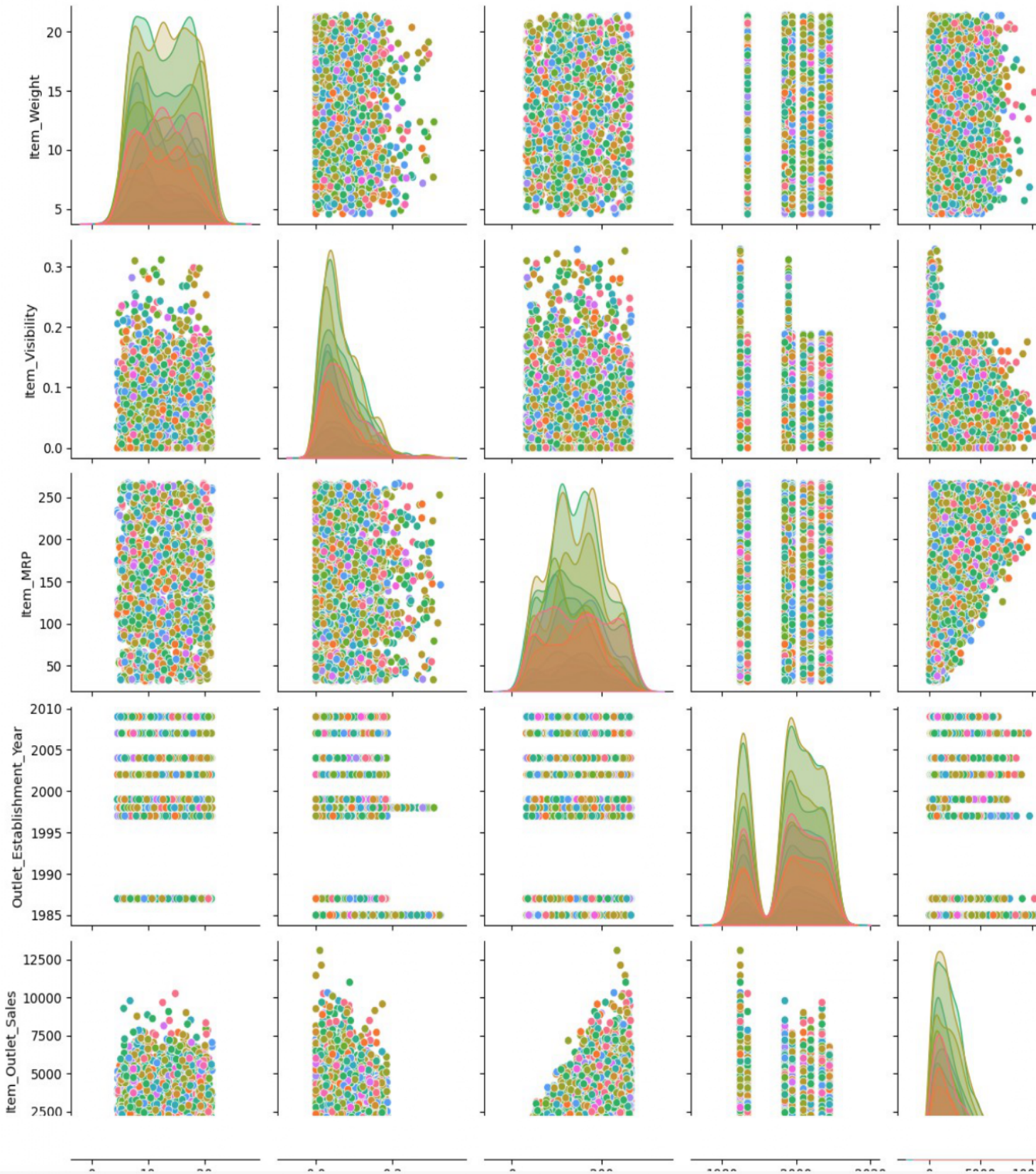
```
print("box distribution plot for item_weight")
sns.boxplot(y=df['Item_Weight'])
plt.show()
print('box distribution plot for item mrp')
sns.boxplot(y=df['Item_MRP'])
plt.show()
```

```
    box distribution plot for item_weight
pd.crosstab(index=df['Outlet_Type'],columns=df['Outlet_Establishment_Year'],dropna=True)
```

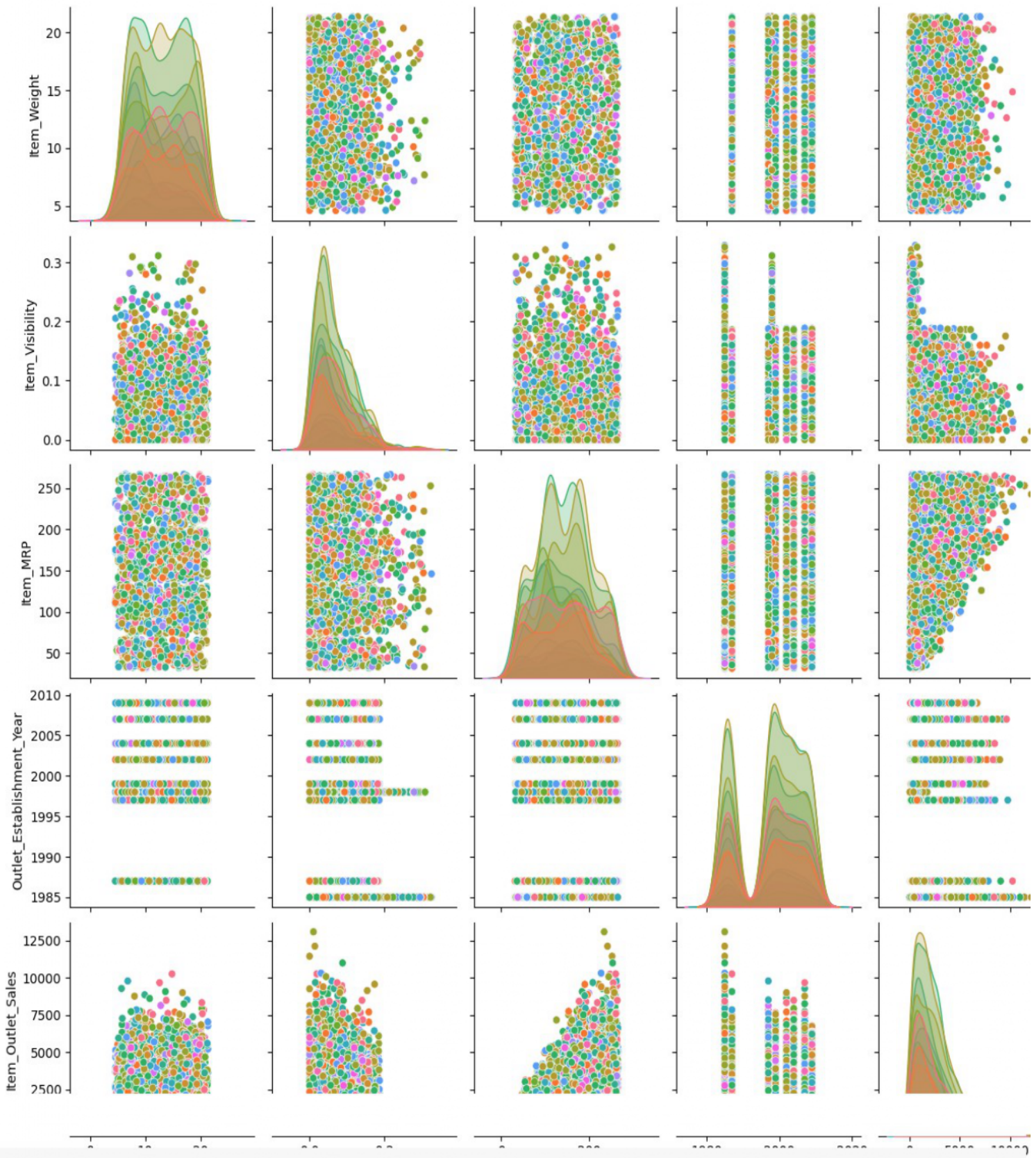
```
print(df['Item_MRP'].describe())
sns.distplot(x=df['Item_MRP'],kde=True)
```

```
    count  8523.000000
    mean    140.992782
    std      62.275067
    min      31.290000
    25%      93.826500
    50%     143.012800
    75%     185.643700
    max     266.888400
    Name: Item_MRP, dtype: float64
      sns.distplot(x=df['Item_MRP'],kde=True)
    <Axes: ylabel='Density'>
```




Item_Weight
Item_Visibility
Item_MRP
Outlet_Establishment_Year
Item_Outlet_Sales

```
c:\Users\HARIHARAPRASAD R\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed
  self._figure.tight_layout(*args, **kwargs)
<seaborn.axisgrid.PairGrid at 0x25693caebc0>
```



```
#!pip install pandas numpy seaborn matplotlib klib dtale scikit-learn joblib pandas-profiling
```

```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df_train= pd.read_csv(r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\Dataset\train.csv')
df_test= pd.read_csv(r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\Dataset\test.csv')
```

```
df_test.isnull().sum()
```

```
Item_Identifier           0
Item_Weight             976
Item_Fat_Content          0
Item_Visibility           0
Item_Type                 0
Item_MRP                  0
Outlet_Identifier         0
Outlet_Establishment
_Year                     0
Outlet_Size            1606
Outlet_Location_Type      0
Outlet_Type               0
dtype: int64
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to
8522
Data columns (total 12
columns):
                                Non-Null
 #   Column                        Count Dtype
                            -------------
--- ------                  -             -----
                            852
 0   Item_Identifier        3   non-null  object
                            706
 1   Item_Weight            0   non-null  float64
                            852
 2   Item_Fat_Content       3   non-null  object
                            852
 3   Item_Visibility        3   non-null  float64
                            852
 4   Item_Type              3   non-null  object
                            852
 5   Item_MRP               3   non-null  float64
     Outlet_Identifie       852
 6   r                      3   non-null  object
     Outlet_Establishment_Ye    8523 non-
 7   ar                             null  int64
                            611
 8   Outlet_Size            3   non-null  object
                               8523 non-
 9   Outlet_Location_Type           null  object
                               8523 non-
 10  Outlet_Type                    null  object
     Item_Outlet_Sale          8523 non-
 11  s                              null  float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+
KB
```

```
df_train.describe(
                )
```

	Item_Weight	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales

	Item_Visibility				
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

Data Cleaning using Klib Library

```
# klib.clean - functions for cleaning datasets
klib.data_cleaning(df_train) # performs datacleaning (drop duplicates & empty rows/cols, adjust dtypes,...)
```

```
Shape of cleaned data: (8523, 10)Remaining NAs: 0

Changes:
Dropped rows: 0
   of which 0 duplicates. (Rows: [])
Dropped columns: 0
   of which 0 single valued.  Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.08 MB (-12.31%)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	
0	9.300000	Low Fat	0.016047	Dairy	249.809204	1999	Medium	Tier 1	Supe
1	5.920000	Regular	0.019278	Soft Drinks	48.269199	2009	Medium	Tier 3	Supe
2	17.500000	Low Fat	0.016760	Meat	141.617996	1999	Medium	Tier 1	Supe
3	19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	1998	Medium	Tier 3	
4	8.930000	Low Fat	0.000000	Household	53.861401	1987	High	Tier 3	Supe
...	...	...	...	...	...	...	...	...	
8518	6.865000	Low Fat	0.056783	Snack Foods	214.521805	1987	High	Tier 3	Supe
8519	8.380000	Regular	0.046982	Baking Goods	108.156998	2002	Medium	Tier 2	Supe
8520	10.600000	Low Fat	0.035186	Health and Hygiene	85.122398	2004	Small	Tier 2	Supe
8521	7.210000	Regular	0.145221	Snack Foods	103.133202	2009	Medium	Tier 3	Supe
8522	14.800000	Low Fat	0.044878	Soft Drinks	75.467003	1997	Small	Tier 1	Supe

8523 rows × 10 columns

```
klib.clean_column_names(df_train) # cleans and standardizes column names, also called inside data_cleaning()
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	c
0	9.300	Low Fat	0.016047	Dairy	249.8092	1999	Medium	Tier 1	Superm
1	5.920	Regular	0.019278	Soft Drinks	48.2692	2009	Medium	Tier 3	Superm
2	17.500	Low Fat	0.016760	Meat	141.6180	1999	Medium	Tier 1	Superm
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	1998	Medium	Tier 3	G
4	8.930	Low Fat	0.000000	Household	53.8614	1987	High	Tier 3	Superm
...	...	...	...	...	...	...	...	...	
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	1987	High	Tier 3	Superm
8519	8.380	Regular	0.046982	Baking Goods	108.1570	2002	Medium	Tier 2	Superm
8520	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	2004	Small	Tier 2	Superm
8521	7.210	Regular	0.145221	Snack Foods	103.1332	2009	Medium	Tier 3	Superm
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	1997	Small	Tier 1	Superm

8523 rows × 10 columns

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
```

```
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float64
 1   item_fat_content           8523 non-null   object
 2   item_visibility            8523 non-null   float64
 3   item_type                  8523 non-null   object
 4   item_mrp                   8523 non-null   float64
 5   outlet_establishment_year  8523 non-null   int64
 6   outlet_size                8523 non-null   object
 7   outlet_location_type       8523 non-null   object
 8   outlet_type                8523 non-null   object
 9   item_outlet_sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(5)
memory usage: 666.0+ KB
```

```
df_train=klib.convert_datatypes(df_train) # converts existing to more efficient dtypes, also called inside data_cleaning()
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float32
 1   item_fat_content           8523 non-null   category
 2   item_visibility            8523 non-null   float32
 3   item_type                  8523 non-null   category
 4   item_mrp                   8523 non-null   float32
 5   outlet_establishment_year  8523 non-null   int16
 6   outlet_size                8523 non-null   category
 7   outlet_location_type       8523 non-null   category
 8   outlet_type                8523 non-null   category
 9   item_outlet_sales          8523 non-null   float32
dtypes: category(5), float32(4), int16(1)
memory usage: 192.9 KB
```

```
klib.mv_col_handling(df_train)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	
0	9.300000	Low Fat	0.016047	Dairy	249.809204	1999	Medium	Tier 1	Supe
1	5.920000	Regular	0.019278	Soft Drinks	48.269199	2009	Medium	Tier 3	Supe
2	17.500000	Low Fat	0.016760	Meat	141.617996	1999	Medium	Tier 1	Supe
3	19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	1998	Medium	Tier 3	
4	8.930000	Low Fat	0.000000	Household	53.861401	1987	High	Tier 3	Supe
...	...	...	...	...	...	...	...	...	
8518	6.865000	Low Fat	0.056783	Snack Foods	214.521805	1987	High	Tier 3	Supe
8519	8.380000	Regular	0.046982	Baking Goods	108.156998	2002	Medium	Tier 2	Supe
8520	10.600000	Low Fat	0.035186	Health and Hygiene	85.122398	2004	Small	Tier 2	Supe
8521	7.210000	Regular	0.145221	Snack Foods	103.133202	2009	Medium	Tier 3	Supe
8522	14.800000	Low Fat	0.044878	Soft Drinks	75.467003	1997	Small	Tier 1	Supe

8523 rows × 10 columns

Preprocessing Task before Model Building

1) Label Encoding

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
df_train['item_fat_content']= le.fit_transform(df_train['item_fat_content'])
df_train['item_type']= le.fit_transform(df_train['item_type'])
df_train['outlet_size']= le.fit_transform(df_train['outlet_size'])
df_train['outlet_location_type']= le.fit_transform(df_train['outlet_location_type'])
df_train['outlet_type']= le.fit_transform(df_train['outlet_type'])
```

```
df_train
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_type
0	9.300000	1	0.016047	4	249.809204	1999	1	0	1
1	5.920000	2	0.019278	14	48.269199	2009	1	2	2
2	17.500000	1	0.016760	10	141.617996	1999	1	0	1
3	19.200001	2	0.000000	6	182.095001	1998	1	2	0
4	8.930000	1	0.000000	9	53.861401	1987	0	2	1
...	...	...	...	...	...	...	...	...	...
8518	6.865000	1	0.056783	13	214.521805	1987	0	2	1
8519	8.380000	2	0.046982	0	108.156998	2002	1	1	1
8520	10.600000	1	0.035186	8	85.122398	2004	2	1	1
8521	7.210000	2	0.145221	13	103.133202	2009	1	2	2
8522	14.800000	1	0.044878	14	75.467003	1997	2	0	1

8523 rows × 10 columns

2) Splitting our data into train and test

```
X=df_train.drop('item_outlet_sales',axis=1)
```

```
Y=df_train['item_outlet_sales']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=101, test_size=0.2)
```

3) Standarization

```
X.describe()
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_typ
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.00000
mean	12.858088	1.369354	0.066132	7.226681	140.992767	1997.831867	1.170832	1.112871	1.20122
std	4.226130	0.644810	0.051598	4.209990	62.275051	8.371760	0.600327	0.812757	0.79645
min	4.555000	0.000000	0.000000	0.000000	31.290001	1985.000000	0.000000	0.000000	0.00000
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	0.000000	1.00000
50%	12.857645	1.000000	0.053931	6.000000	143.012802	1999.000000	1.000000	1.000000	1.00000
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	2.000000	1.00000
max	21.350000	4.000000	0.328391	15.000000	266.888397	2009.000000	2.000000	2.000000	3.00000

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
```

```
X_train_std= sc.fit_transform(X_train)
```

```
X_test_std= sc.transform(X_test)
```

```
X_train_std
```

```
array([[ 1.52290023, -0.57382672,  0.68469731, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.239856 , -0.57382672, -0.09514746, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 1.54667619, 0.97378032, -0.0083859 , ..., -
        0.28872895, -0.13870429, -0.25964107],
       ...,
       [-0.08197109, -0.57382672, -0.91916229, ...,  1.37953713,
        -1.36527477, -0.25964107],
       [-0.74888436,           0.97378032,  1.21363045, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 0.67885675, -0.57382672, 1.83915361, ..., -
         0.28872895, 1.08786619, 0.98524841]])
```

```
X_test_std
```

```
array([[-0.43860916, -0.57382672, -0.21609253, ..., -0.28872895,
         1.08786619,0.98524841],
       [ 1.22570184, -0.57382672, -0.52943464, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.2184578 ,           0.97378032,    0.16277341, ...,       1.37953713,
        -1.36527477, -0.25964107],
       ...,
       [ 0.65508101, -0.57382672,    0.8782423 , ..., -0.28872895,
         1.08786619, -1.50453056],
       [ 1.01171909, -0.57382672, -1.28409256, ..., -
         0.28872895, 1.08786619, 0.98524841],
       [-1.56558541, 0.97378032, -1.09265374, ..., -0.28872895,
        -0.13870429, -0.25964107]])
```

```
Y_train
```

```
3684   163.786804
1935  1607.241211
5142  1510.034424
4978  1784.343994
2299  3558.035156
         ...
599    5502.836914
56951436.796387
80062167.844727
13612700.484863
1547829.586792
Name: item_outlet_sales, Length: 6818, dtype: float32
```

```
Y_test
```

```
8179   904.822205
8355  2795.694092
3411  1947.464966
7089   872.863770
6954  2450.144043
         ...
1317  1721.093018
4996   914.809204
531    370.184814
38911358.232056
66292418.185547
Name: item_outlet_sales, Length: 1705, dtype: float32
```

```
import joblib
```

```
joblib.dump(sc,r'D:\BigMart-Sales-Prediction-using-Machine-Learning-main (1)\BigMart-Sales-Prediction-using-Machine-Learning-main\sc.sav')
```

```
['D:\\BigMart-Sales-Prediction-using-Machine-Learning-main (1)\\BigMart-Sales-Prediction-using-Machine-Learning-main\\sc.sav']
```

Model Building

Model building

```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

```
lr.fit(X_train_std,Y_train)
```

```
    LinearRegression()
```

```
X_test.head()
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_type
8179	11.000000	1	0.055163	8	100.335800	2009	1	2	2
8355	18.000000	1	0.038979	13	148.641800	1987	0	2	1
3411	7.720000	2	0.074731	1	77.598602	1997	2	0	1
7089	20.700001	1	0.049035	6	39.950600	2007	1	1	1
6954	7.550000	1	0.027225	3	152.934006	2002	1	1	1

```
Y_pred_lr=lr.predict(X_test_std)
```

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

```
print(r2_score(Y_test,Y_pred_lr))
print(mean_absolute_error(Y_test,Y_pred_lr))
print(np.sqrt(mean_squared_error(Y_test,Y_pred_lr)))
```

```
    0.5041875773270632
    880.9999044084501
    1162.4412631603454
```

```
joblib.dump(lr,r'D:\Python37\Projects\iNeuron Intership Projects\BigMart-Sales\models\lr.sav')
```

```
    ['D:\\Python37\\Projects\\iNeuron Intership Projects\\BigMart-Sales\\models\\lr.sav']
```

```
from sklearn.ensemble import RandomForestRegressor
rf= RandomForestRegressor(n_estimators=1000)
```

```
rf.fit(X_train_std,Y_train)
```

```
    RandomForestRegressor(n_estimators=1000)
```

```
Y_pred_rf= rf.predict(X_test_std)
```

```
print(r2_score(Y_test,Y_pred_rf))
print(mean_absolute_error(Y_test,Y_pred_rf))
print(np.sqrt(mean_squared_error(Y_test,Y_pred_rf)))
```

```
    0.5473443624488158
    783.2320127534548
    1110.6987486230885
```

Hyper Parameter Tuning

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = RandomForestRegressor()
n_estimators = [10, 100, 1000]
max_depth=range(1,31)
min_samples_leaf=np.linspace(0.1,
1.0) max_features=["auto",
"sqrt", "log2"]

min_samples_split=np.linspace(0.1, 1.0, 10)

# define grid search
grid = dict(n_estimators=n_estimators)

#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=101)

grid_search_forest = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,


grid_search_forest.fit(X_train_std, Y_train)

# summarize results
print(f"Best: {grid_search_forest.best_score_:.3f} using {grid_search_forest.best_params_}")
means = grid_search_forest.cv_results_['mean_test_score']
stds = grid_search_forest.cv_results_['std_test_score']
params = grid_search_forest.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")


    Fitting 2 folds for each of 3 candidates, totalling 6 fits
    Best: 0.623 using {'n_estimators': 1000}
    0.586 (0.007) with: {'n_estimators': 10}
    0.618 (0.004) with: {'n_estimators': 100}
    0.623 (0.003) with: {'n_estimators': 1000}
```

```
grid_search_forest.best_params_
```

```
    {'n_estimators': 1000}
```

```
grid_search_forest.best_score_
```

```
    0.6225226127048935
```

```
Y_pred_rf_grid=grid_search_forest.predict(X_test_std)
```

```
r2_score(Y_test,Y_pred_rf_grid)
```

```
    0.6156338068690421
```

Save your model

```
joblib.dump(grid_search_forest,r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\models\random_forest_grid.sav')
```

```
['D:\\Python37\\Projects\\iNeuron Intership Projects\\ML_BigMart Sales Prediction\\models\\random_forest_grid.sav']
```

```
model=joblib.load(r'D:\Python37\Projects\iNeuron Intership Projects\ML_BigMart Sales Prediction\models\random_forest_grid.sav')
```

Importing modified data.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings(action="ignore")
plt.style.use(["seaborn-bright","dark_background"])
```

```
import os
os.getcwd()
```

```
'/content'
```

```
train = pd.read_csv("/modified_train.csv")
test = pd.read_csv("/modified_test.csv")
```

```
X = train.drop(columns=["Item_Outlet_Sales"])
y = train["Item_Outlet_Sales"]
```

```
from sklearn.model_selection import train_test_split
x_train, x_valid, y_train, y_valid = train_test_split(X,y,test_size=0.2,random_state=101)
```

Model training and evaluation.

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

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

```
models = []
models.append(("LinearRegressor",LinearRegression()))
models.append(("Lasso",Lasso()))
models.append(("Ridge",Ridge()))
models.append(("DecisionTree",DecisionTreeRegressor(criterion="mse")))
models.append(("RandomForestRegressor",RandomForestRegressor(criterion="mse")))
models.append(("ExtraTreeRegressor",ExtraTreesRegressor()))
models.append(("AdaBoostRegressor",AdaBoostRegressor()))
models.append(("GradientBoosingRegressor",GradientBoostingRegressor()))
models.append(("SVR",SVR()))
models.append(("KNeighborsRegressor",KNeighborsRegressor()))
```

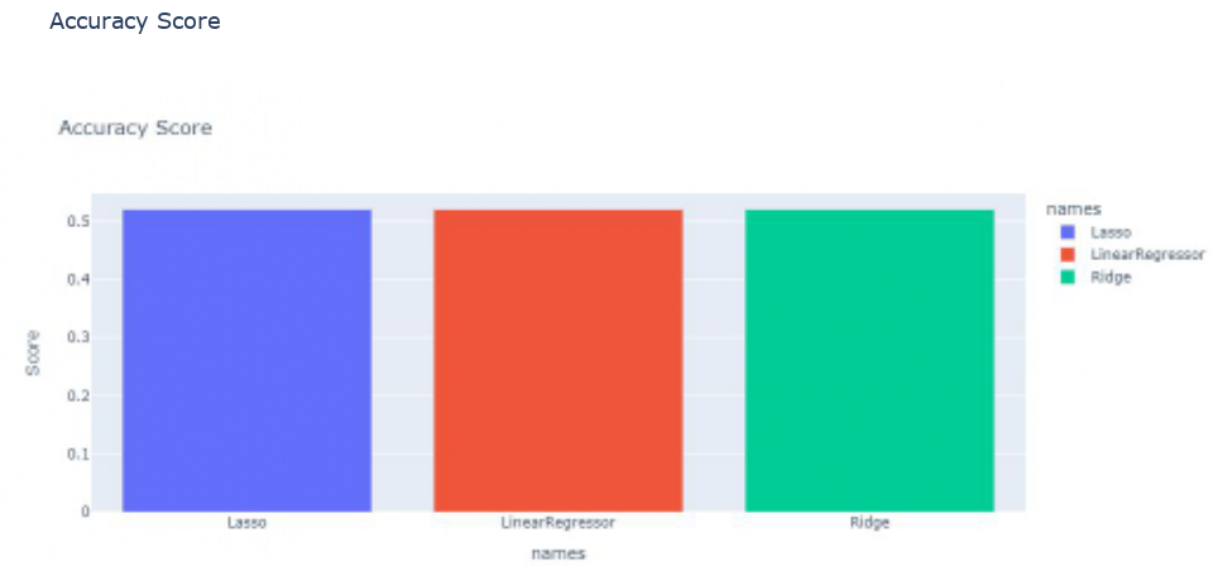
```
names = []
score = []
rmse = []
mse = []
mae = []
r2 = []
timing = []
```

```
for name,model in models:
    model = model
    beg = time.time()
    model.fit(x_train,y_train)
    pred = model.predict(x_valid)
    end = time.time()
    print("Model = {}".format(name))
    print("Score = {}, RMSE = {}, MSE = {}, MAE = {}, R2 = {},Time = {}\n".format(round(model.score(x_valid,y_valid),4),
                                            round(np.sqrt(mean_squared_error(pred,y_valid)),4),
                                            round(mean_squared_error(pred,y_valid),4),
                                            round(mean_absolute_error(pred,y_valid),4),
```

```
                                        round(r2_score(y_valid,pred),4),(end-beg)))
    names.append(name)
    score.append(round(model.score(x_valid,y_valid),4))
    rmse.append(round(np.sqrt(mean_squared_error(pred,y_valid)),4))
    mse.append(round(mean_squared_error(pred,y_valid),4))
    mae.append(round(mean_absolute_error(pred,y_valid),4))
    r2.append(round(r2_score(pred,y_valid),4))
    timing.append(end-beg)
```

```
df = pd.DataFrame()
df["names"] = names
df["Score"] = score
df["time"] = timing
df["rmse"] = rmse
df["mse"] = mse
df["mae"] = mae
df["r2"] = r2
```

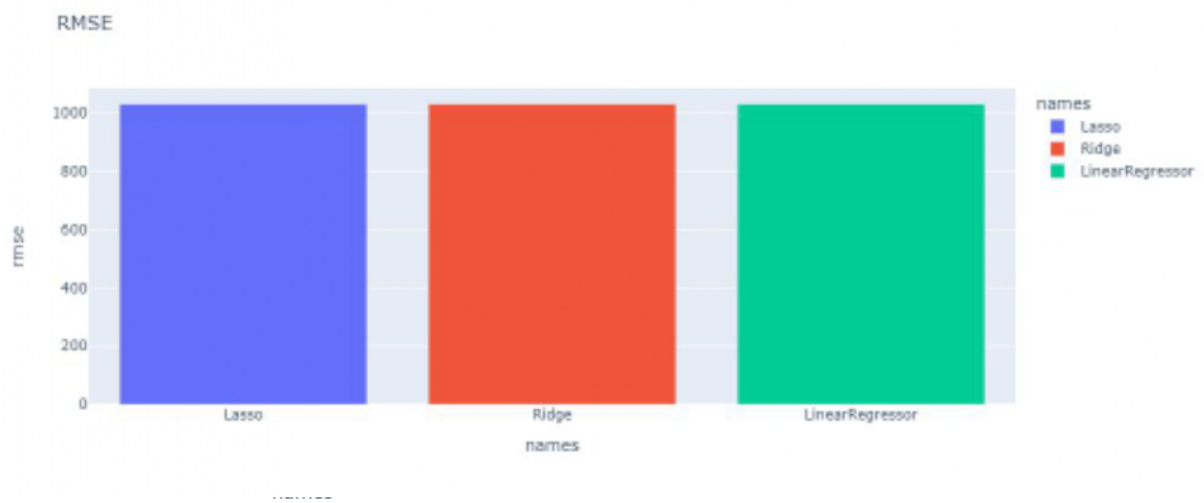
```
df = df.sort_values(by="Score",ascending=False)
fig = px.bar(df,"names","Score",color="names",title="Accuracy Score")
fig.show()
```



+ Code + Text

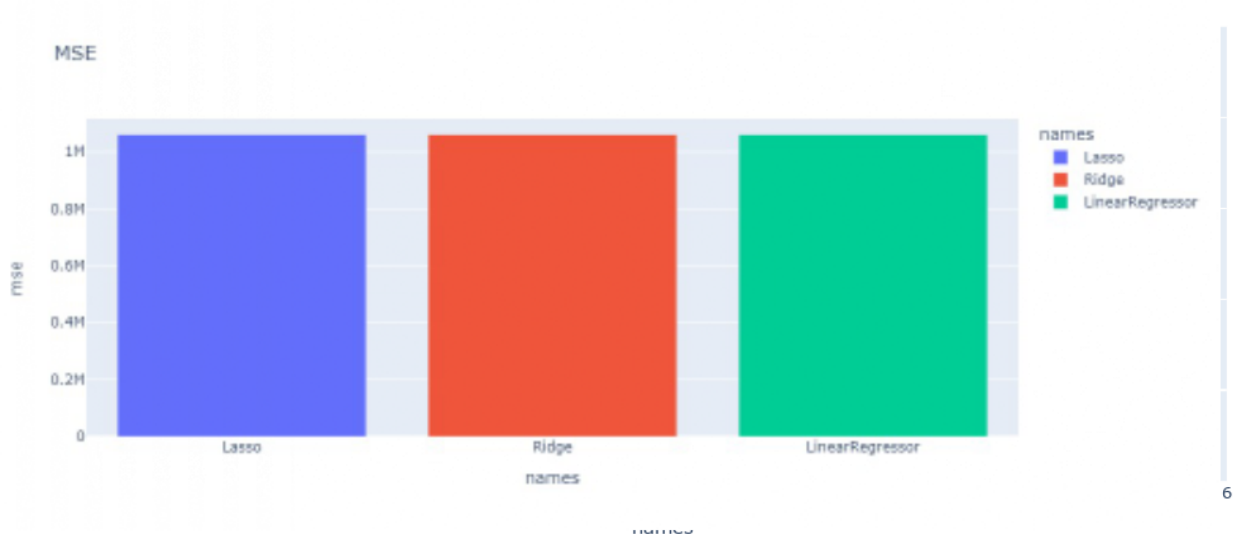
```
df = df.sort_values(by="rmse",ascending=True)
fig = px.bar(df,"names","rmse",color="names",title="RMSE")
fig.show()
```

RMSE




```
df = df.sort_values(by="mse",ascending=True)
fig = px.bar(df,"names","mse",color="names",title="MSE")
fig.show()
```

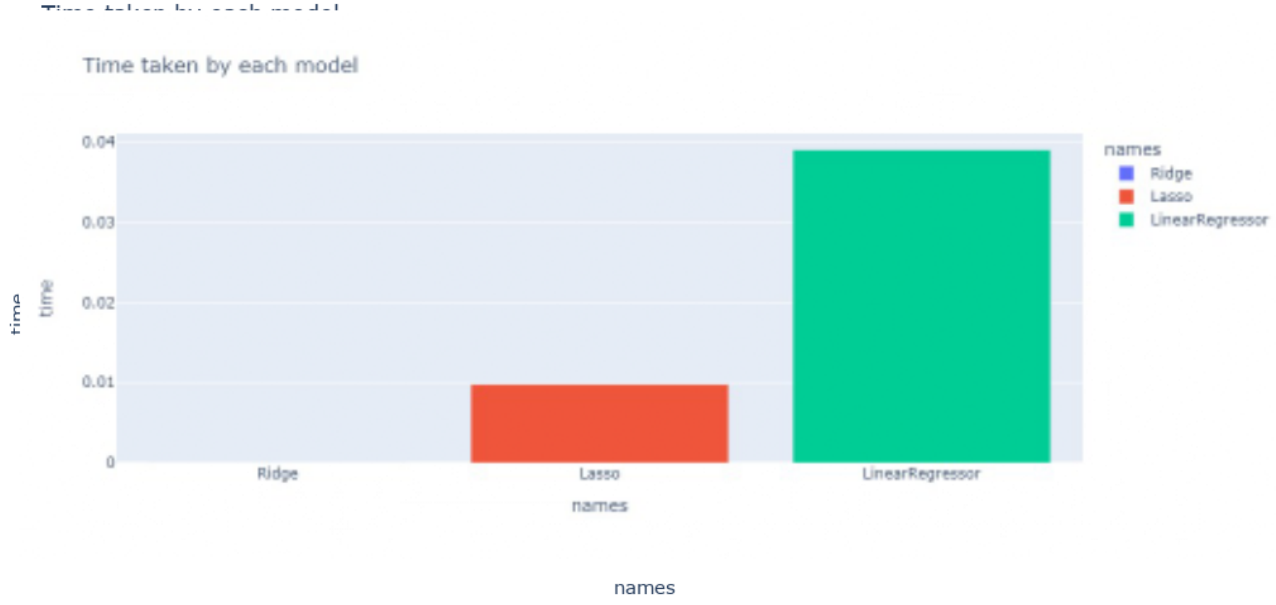
MSE




```
df = df.sort_values(by="mae",ascending=True)
fig = px.bar(df,"names","mae",color="names",title="MAE")
fig.show()
```

```
df = df.sort_values(by="r2",ascending=True)
fig = px.bar(df,"names","r2",color="names",title="R2 Score")
fig.show()
```


```
df = df.sort_values(by="time",ascending=True)
fig = px.bar(df,"names","time",color="names",title="Time taken by each model")
fig.show()
```




```
# On the basis of all above metrices and time taken by the model we choose the GradientBoostingRegressor.
```

```
#Below cell will take much time to complete. Have patience.
```

Hyperparameter tuning.

```
from sklearn import model_selection
classifer = GradientBoostingRegressor()
dt_grid = {'loss':['ls', 'lad', 'huber'],
           'learning_rate':[0.01,0.05,0.1,0.5],
          'n_estimators' : [100,200],
          'max_depth':[3,5,8,10]}
grid_classifier = model_selection.GridSearchCV(classifer, dt_grid, cv=10, refit=True,
                                               return_train_score=True)
grid_classifier.fit(X, y)
results = grid_classifier.cv_results_
print(results.get('params'))
print(results.get('mean_test_score'))
print(results.get('mean_train_score'))
print(grid_classifier.best_params_)
print(grid_classifier.best_score_)
final_model = grid_classifier.best_estimator_
```

```
[{'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 100}, {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 200
[0.45666354 0.5383241  0.48639658 0.55214666 0.48356294 0.54289183
 0.47475018 0.52884411 0.41172121 0.50903553 0.4563813  0.53846537
 0.4559354  0.53120754 0.45145131 0.52270371 0.44465863 0.53200664
 0.47660263 0.54837358 0.47345107 0.53773397 0.46421036 0.5252806
 0.56219903 0.55898499 0.5578043  0.54757802 0.54253485 0.5230378
 0.51707634 0.49204565 0.55527959 0.55695709 0.55687108 0.55511645
 0.54269681 0.54141506 0.53111017 0.53135754 0.56080537 0.55835633
 0.55615383 0.54802692 0.54092345 0.52414699 0.5202874  0.49699505
 0.55875926 0.55197103 0.54627866 0.5321496  0.51876295 0.49125825
 0.48673411 0.46290662 0.55670994 0.55701126 0.55499688 0.55357483
 0.54300196 0.53971384 0.53260159 0.52492466 0.55871537 0.5520153
 0.54773068 0.53261227 0.51619728 0.4957597  0.49439778 0.46594476
 0.5155621  0.48658162 0.45113661 0.39913738 0.36877039 0.34593855
 0.34136894 0.33905061 0.55212045 0.5522085  0.54296472 0.54205124
 0.5123446  0.50850891 0.47423237 0.47005396 0.51607627 0.4931469
 0.45444863 0.40817854 0.3685745  0.33761779 0.35153939 0.3449091 ]
[0.4614503  0.54480375 0.49479414 0.56771154 0.53587114 0.62805944
 0.58708333 0.70057    0.41432976 0.5129928  0.46278081 0.54835934
 0.4856318  0.57953852 0.50915615 0.61503382 0.44928354 0.53806567
 0.48530486 0.56373609 0.521811   0.6167868  0.56385221 0.67718955
 0.57700238 0.58994745 0.60493998 0.64472983 0.70162853 0.78821659
 0.79290549 0.88414095 0.56235384 0.57014873 0.57993772 0.58838613
 0.62471353 0.63238344 0.67046719 0.67979937 0.57548673 0.58839026
 0.60355567 0.63842554 0.6891488  0.77083813 0.7727666  0.86364983
 0.59098229 0.61509492 0.64633342 0.71000828 0.79003584 0.87945228
 0.88946099 0.95312369 0.57025086 0.57354789 0.58822297 0.59045152
 0.63439349 0.63710729 0.68094054 0.69299451 0.58882959 0.61032526
 0.64031666 0.69784656 0.77657243 0.86146278 0.86549791 0.93586116
 0.66995309 0.732847   0.82114121 0.9082088  0.97316366 0.99699947
 0.99690316 0.99993363 0.58002685 0.57964509 0.60411933 0.60087394
 0.65668795 0.66120262 0.7142575  0.72134682 0.66264662 0.72045787
 0.80111138 0.8828194  0.95322973 0.98867128 0.98944244 0.99869967]
{'learning_rate': 0.05, 'loss': 'ls', 'max_depth': 3, 'n_estimators': 100}
0.5621990292342698
```

```
pred  = final_model.predict(x_valid)
mean_absolute_error(pred,y_valid)
```

Saving best model for deployment.

```
import pickle
```

```
pickle_out = open("sales_flask.pkl","wb")
pickle.dump(final_model,pickle_out)
```

```
0.5749856087550711
```

Loading model.

```
loaded_model = pickle.load(open("sales_flask.pkl","rb"))
pred = loaded_model.predict(test)
```

```
pred
```

```
array([1662.3091974 , 1381.05716057,  643.17042028, ..., 1902.46738382,
       3384.9306051 , 1326.95857518])
```

References:

Datset :https://www.kaggle.com/search?q=sales+datset

python :

PANDAS:
:https://www.google.com/search?q=python+pandas+documantation&rlz=1C1RXQR_enIN1074IN1075&oq=python+pandas+documantation&gs_lcrp=EgZjaHJvbWUyBggAEEUYOTIJCAEQABgNGIAEMgkIAhAAGA0YgAQyCggDEAAYDxgWGB4yCAgEEAAYFhgeMgwIBRAAGAUYDRgPGB4yCggGEAAYBRgNGB4yCggHEAAYBRgNGB4yCggIEAAYCBgNGB4yCggJEAAYCBgNGB7SAQg4NzA4ajBqN6gCALACAA&sourceid=chrome&ie=UTF-8

SEABORN:
https://seaborn.pydata.org/

SKLEARN:
:https://scikit-learn.org/stable/

STATISTICS:
:https://www.mathsisfun.com/

DATA PREPROCESSING:
skillup recoeded sessions

Youtube:

https://www.youtube.com/live/W9p7B0HR820si=cemOaTkYRGRfWAsR

SUBMITTED BY,

DIVAGAR-au513121106024
HARIHARAPRASAD-au513121106702
AGARSHA-
ABINAYA C-
Thanthai periyar govt institue of technology, vellore