LAB 03

EC9560 – DATA MINING

THEVARAJAN.R.J

2019/E/146

GROUP D

SEMESTER 7

03 NOV 2023

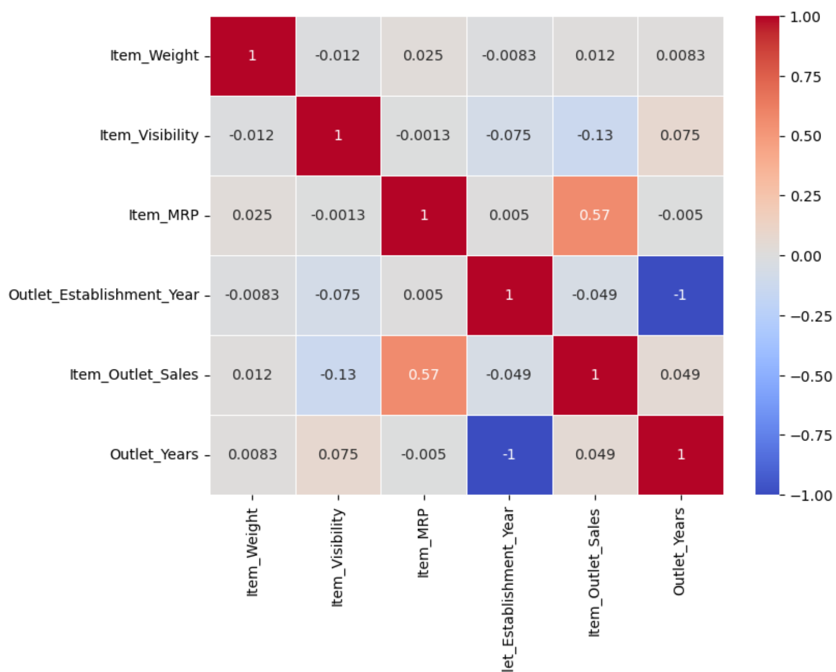
- From the previous report, we have discussed Exploratory Data Analysis (EDA) using dtale library.
- This report discusses klib (library). There is a way for EDA to use panda-profiling too.

First correlation matrix of training dataset and corresponding heatmap was found.

```
# Compute the correlation matrix
correlation_matrix = df.corr()

# Display the correlation matrix
print(correlation_matrix)
```

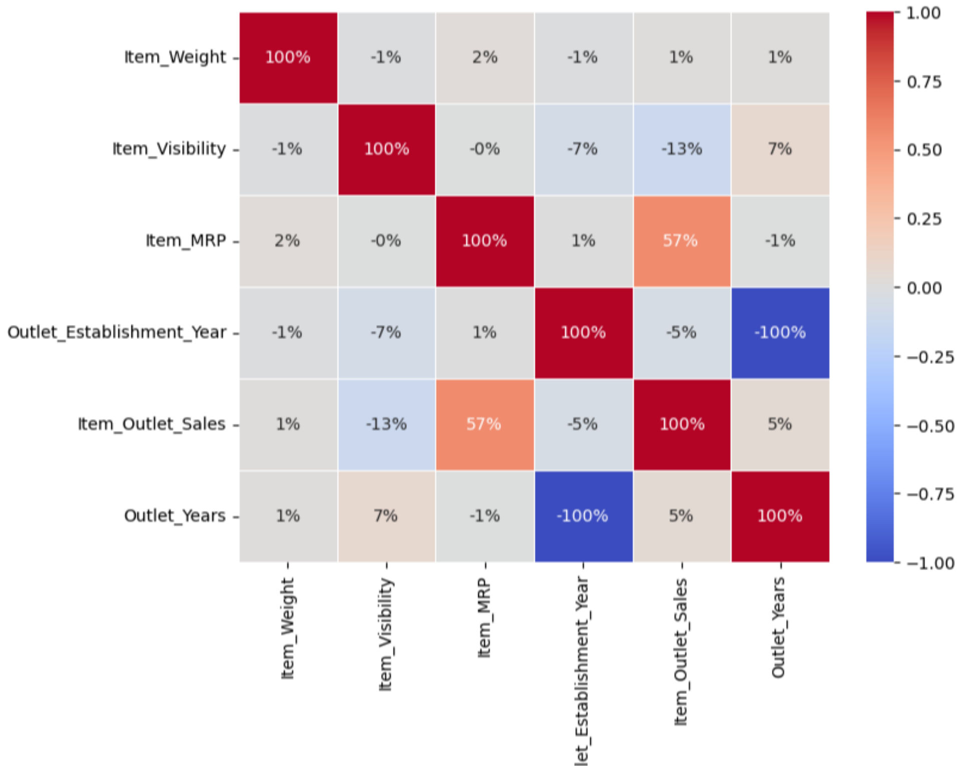
```
                           Item_Weight  Item_Visibility  Item_MRP  \
Item_Weight                   1.000000        -0.012049  0.024756
Item_Visibility              -0.012049         1.000000 -0.001315
Item_MRP                      0.024756        -0.001315  1.000000
Outlet_Establishment_Year    -0.008301        -0.074834  0.005020
Item_Outlet_Sales             0.011550        -0.128625  0.567574
Outlet_Years                  0.008301         0.074834 -0.005020

                           Outlet_Establishment_Year  Item_Outlet_Sales  \
Item_Weight                                -0.008301           0.011550
Item_Visibility                            -0.074834          -0.128625
Item_MRP                                    0.005020           0.567574
Outlet_Establishment_Year                   1.000000          -0.049135
Item_Outlet_Sales                          -0.049135           1.000000
Outlet_Years                               -1.000000           0.049135

                           Outlet_Years
Item_Weight                    0.008301
Item_Visibility                0.074834
Item_MRP                      -0.005020
Outlet_Establishment_Year     -1.000000
Item_Outlet_Sales              0.049135
Outlet_Years                   1.000000
```

```
# Create a heatmap
plt.figure(figsize=(8, 6))  # Set the figure size
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

# Show the heatmap
plt.show()
```



To find it in percentage values, fmt = ‘.0% ‘

```python
# Create a heatmap with percentage values
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, fmt='.0%', cmap='coolwarm', linewidths=0.5)

# Show the heatmap
plt.show()
```



- From visualizing heatmap, we can conclude that Outlet_Years and Outlet_Establishment_Year has the -1 negative correlation.
- So, we can drop either one feature Outlet_Years or Outlet_Establishment_Year.
- But considering the datasets Outlet_Years were calculated from Outlet_Establishment_Year as Outlet_Establishment_Year contains large number values.
- It was normalized and a new feature Outlet_Years was created.
- So, we can drop Outlet_Establishment_Year.

From heatmap, we can see that Item_Outlet_Sales and Item_MRP are aprroximately positively correlated with each other (0.57) and Outlet_Years and Outlet_Establishment_Year are negatively highly correlated (-1).

```
9]: df = df.drop('Outlet_Establishment_Year', axis=1)
```

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

```
C:\Users\94774\AppData\Local\Temp\ipykernel_13316\3749504314.py:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only v
alid columns or specify the value of numeric_only to silence this warning.
```

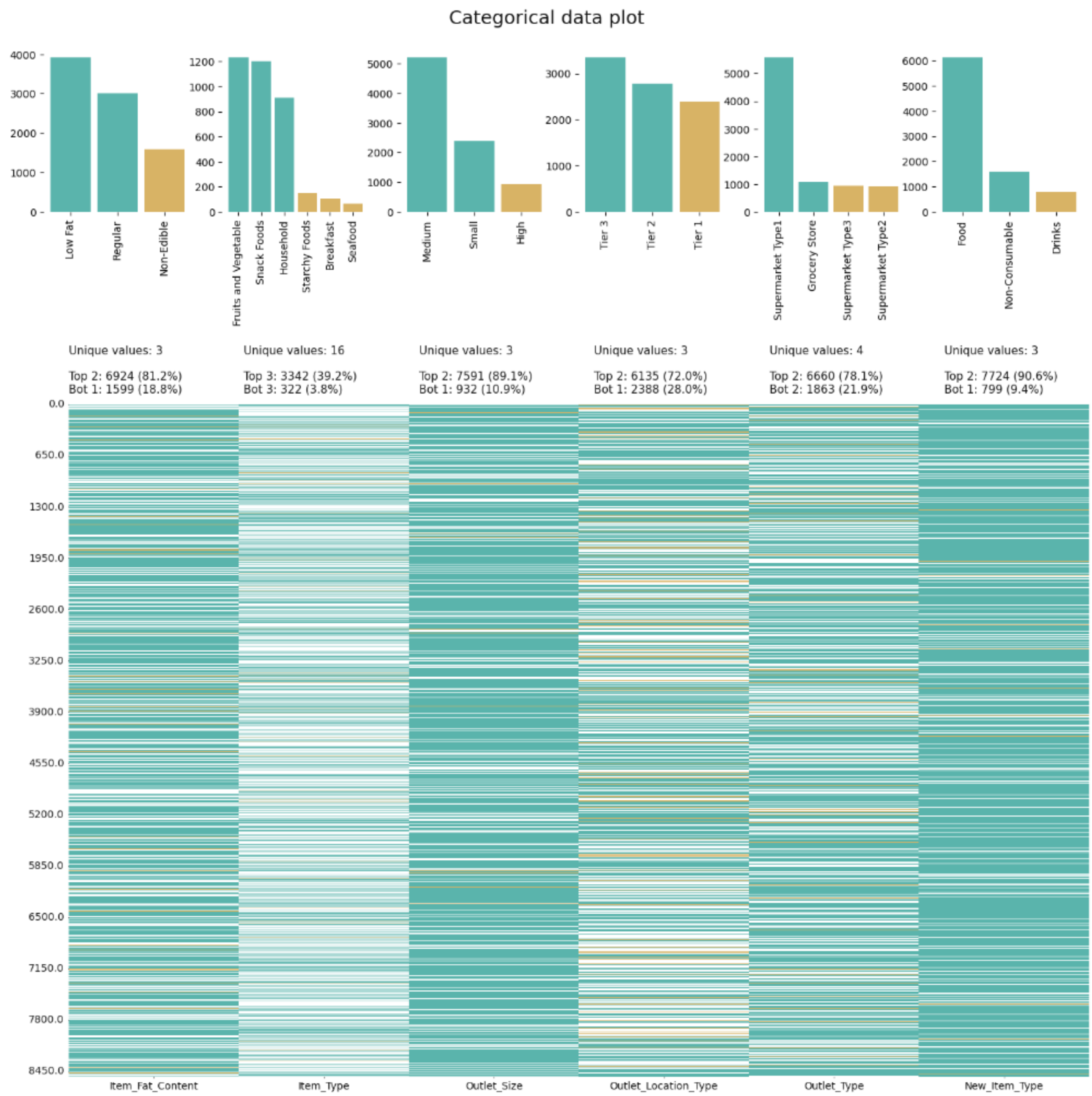
Let's consider about klib for EDA

```
]: import klib
```

Plotting graphs for categorical features.

```
: klib.cat_plot(df)
```

```
: GridSpec(6, 6)
```



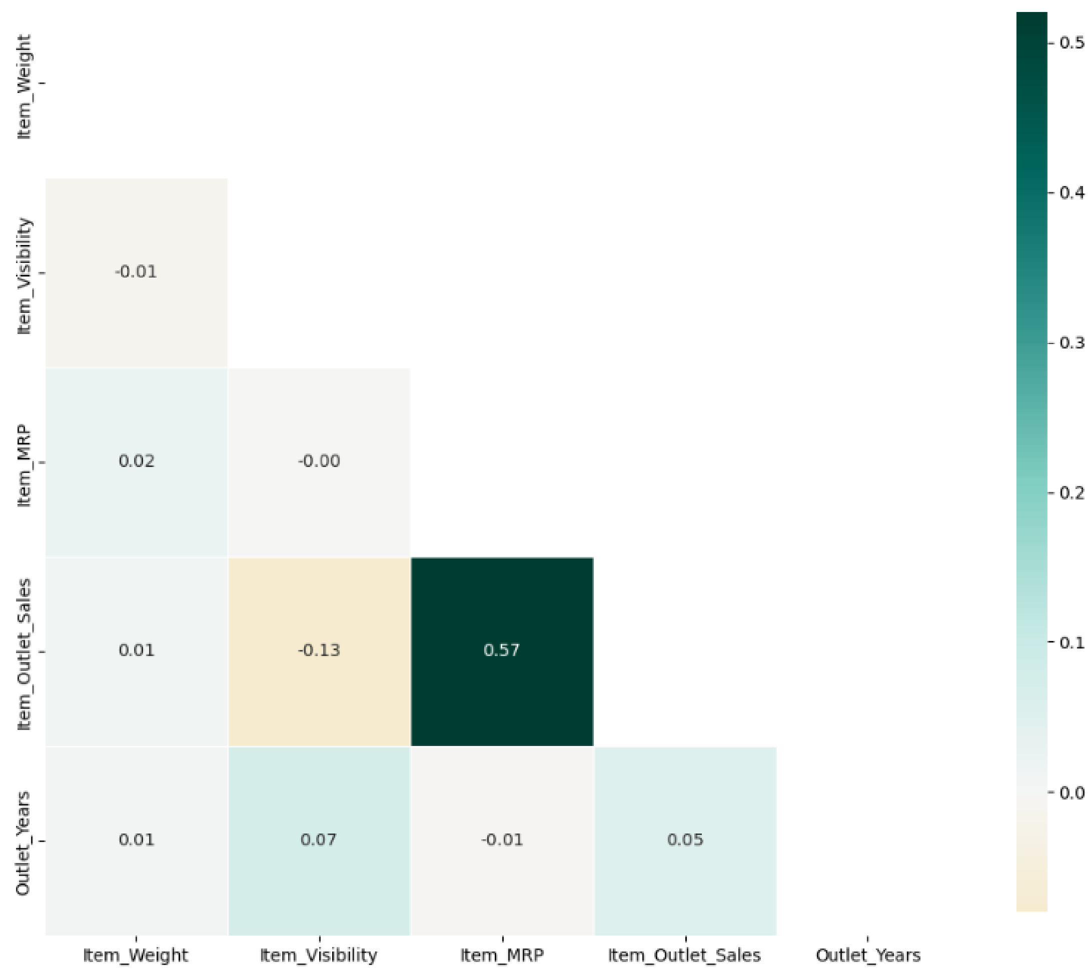
Finding correlation matrix and heatmap using klib:

```
#correlation matrix
klib.corr_mat(df)
```

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

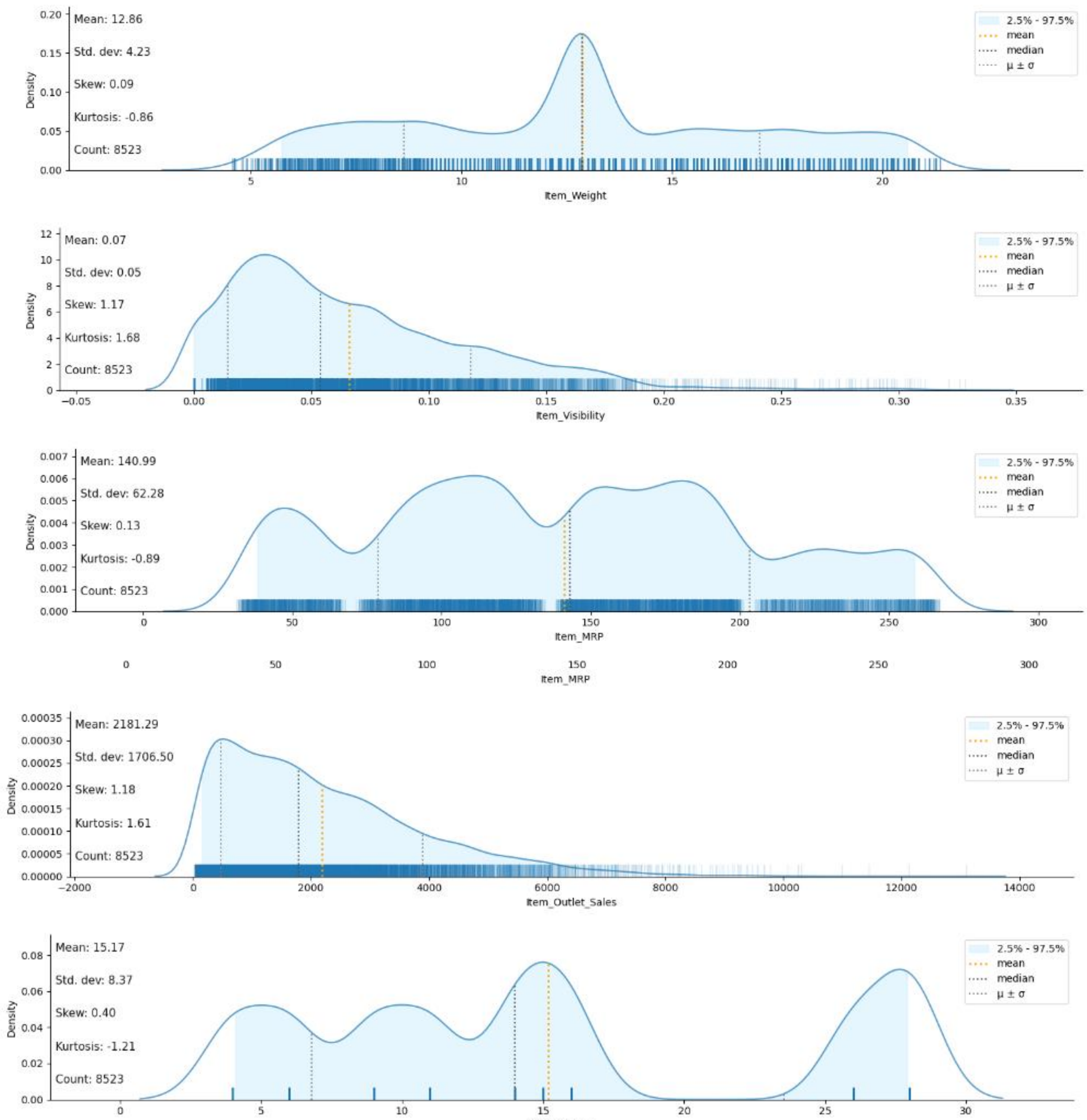
```
klib.corr_plot(df)
```

Feature-correlation (pearson)



We can plot distance graphs and analyze them further.

```
6]: # Get the numeric columns
    numeric_columns = df.select_dtypes(include='number').columns

    # Create individual distribution plots for each numeric feature
    for column in numeric_columns:
        klib.dist_plot(df[column])
```



```
]: klib.missingval_plot(df)
```

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

Data cleaning using klib library:

DATA CLEANING USING KLIB LIBRARY

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

```
Shape of cleaned data: (8523, 11) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.53 MB (-73.61%)
```

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

]:

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_size	outlet_location_type	outlet_type	item_outlet_sales	new_item_type	outlet
0	9.300	Low Fat	0.016047	Dairy	249.8092	Medium	Tier 1	Supermarket Type1	3735.1380	Food	
1	5.920	Regular	0.019278	Soft Drinks	48.2692	Medium	Tier 3	Supermarket Type2	443.4228	Drinks	
2	17.500	Low Fat	0.016760	Meat	141.6180	Medium	Tier 1	Supermarket Type1	2097.2700	Food	
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	Medium	Tier 3	Grocery Store	732.3800	Food	
4	8.930	Non-Edible	0.000000	Household	53.8614	High	Tier 3	Supermarket Type1	994.7052	Non-Consumable	
...	...	...	...	...	...	...	...	...	...	...	
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	High	Tier 3	Supermarket Type1	2778.3834	Food	
8519	8.380	Regular	0.046982	Baking Goods	108.1570	Medium	Tier 2	Supermarket Type1	549.2850	Food	
8520	10.600	Non-Edible	0.035186	Health and Hygiene	85.1224	Small	Tier 2	Supermarket Type1	1193.1136	Non-Consumable	
8521	7.210	Regular	0.145221	Snack Foods	103.1332	Medium	Tier 3	Supermarket Type2	1845.5976	Food	
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	Small	Tier 1	Supermarket Type1	765.6700	Drinks	

8523 rows × 11 columns

You can see what has happened in the above code below:

item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_size	outlet_location_type	outlet_type	item_outlet_sales	nev
9.300000	Low Fat	0.016047	Dairy	249.809204	Medium	Tier 1	Supermarket Type1	3735.137939	
5.920000	Regular	0.019278	Soft Drinks	48.269199	Medium	Tier 3	Supermarket Type2	443.422791	
17.500000	Low Fat	0.016760	Meat	141.617996	Medium	Tier 1	Supermarket Type1	2097.270020	
19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	Medium	Tier 3	Grocery Store	732.380005	
8.930000	Non-Edible	0.000000	Household	53.861401	High	Tier 3	Supermarket Type1	994.705200	
...	...	...	...	...	...	...	...	...	
6.865000	Low Fat	0.056783	Snack Foods	214.521805	High	Tier 3	Supermarket Type1	2778.383301	
8.380000	Regular	0.046982	Baking Goods	108.156998	Medium	Tier 2	Supermarket Type1	549.284973	
10.600000	Non-Edible	0.035186	Health and Hygiene	85.122398	Small	Tier 2	Supermarket Type1	1193.113647	
7.210000	Regular	0.145221	Snack Foods	103.133202	Medium	Tier 3	Supermarket Type2	1845.597656	
14.800000	Low Fat	0.044878	Soft Drinks	75.467003	Small	Tier 1	Supermarket Type1	765.669983	

rows × 11 columns

```
eans and standardizes column names, also called inside data_cleaning()
.clean_column_names(df)
```

item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_size	outlet_location_type	outlet_type	item_outlet_sales	new_
9.300	Low Fat	0.016047	Dairy	249.8092	Medium	Tier 1	Supermarket Type1	3735.1380	
5.920	Regular	0.019278	Soft Drinks	48.2692	Medium	Tier 3	Supermarket Type2	443.4228	
17.500	Low Fat	0.016760	Meat	141.6180	Medium	Tier 1	Supermarket Type1	2097.2700	
19.200	Regular	0.000000	Fruits and Vegetables	182.0950	Medium	Tier 3	Grocery Store	732.3800	

Now, converts datatypes more efficient,

```
: 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_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_size           8523 non-null   object 
 6   outlet_location_type  8523 non-null   object 
 7   outlet_type           8523 non-null   object 
 8   item_outlet_sales     8523 non-null   float64
 9   new_item_type         8523 non-null   object 
 10  outlet_years          8523 non-null   int64  
dtypes: float64(4), int64(1), object(6)
memory usage: 732.6+ KB
```

```
: # converts existing to more efficient dtypes, also called inside data_cleaning()
df=klib.convert_datatypes(df)
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_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_size           8523 non-null   category
 6   outlet_location_type  8523 non-null   category
 7   outlet_type           8523 non-null   category
 8   item_outlet_sales     8523 non-null   float32 
 9   new_item_type         8523 non-null   category
 10  outlet_years          8523 non-null   int8    
dtypes: category(6), float32(4), int8(1)
memory usage: 192.9 KB
```

]:

```python
# pools subset of cols based on duplicates with min. loss of information
klib.pool_duplicate_subsets(df)
```

]:

	item_visibility	item_mrp	item_outlet_sales	pooled_vars
0	0.016047	249.809204	3735.137939	0
1	0.019278	48.269199	443.422791	1
2	0.016760	141.617996	2097.270020	2
3	0.000000	182.095001	732.380005	3
4	0.000000	53.861401	994.705200	4
...	...	...	...	...
8518	0.056783	214.521805	2778.383301	8518
8519	0.046982	108.156998	549.284973	8519
8520	0.035186	85.122398	1193.113647	8520
8521	0.145221	103.133202	1845.597656	8521
8522	0.044878	75.467003	765.669983	8522

8523 rows × 4 columns

Label Encoding:

Preprocessing Task before Model Building

Label encoding

```
]: df.head()
```

]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	New_Item_Type	Out
0	9.30	Low Fat	0.016047	Dairy	249.8092	Medium	Tier 1	Supermarket Type1	3735.1380	Food	
1	5.92	Regular	0.019278	Soft Drinks	48.2692	Medium	Tier 3	Supermarket Type2	443.4228	Drinks	
2	17.50	Low Fat	0.016760	Meat	141.6180	Medium	Tier 1	Supermarket Type1	2097.2700	Food	
3	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	Medium	Tier 3	Grocery Store	732.3800	Food	
4	8.93	Non-Edible	0.000000	Household	53.8614	High	Tier 3	Supermarket Type1	994.7052	Non-Consumable	

Here label encoding is used to convert categorical data into numerical format to process the data effectively.

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

```
df=df.apply(Lab_En.fit_transform)
```

```
df
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_size	outlet_location_type	outlet_type	item_outlet_sales	new_item_type	outlet_
0	284	0	664	4	5592	1	0	1	2540	1	
1	57	2	880	14	473	1	2	2	422	0	
2	376	0	715	10	2901	1	0	1	1639	1	
3	393	2	0	6	4227	1	2	0	670	1	
4	265	1	0	9	627	0	2	1	865	2	
...	...	...	...	...	...	...	...	...	...	...	
8518	125	0	3912	13	4955	0	2	1	2047	1	
8519	233	2	3278	0	2023	1	1	1	516	1	
8520	299	1	2302	8	1263	2	1	1	1018	2	
8521	149	2	7174	13	1857	1	2	2	1466	1	
8522	347	0	3108	14	1011	2	0	1	697	0	

8523 rows × 11 columns

For example:

Consider Item_Fat_Content feature.

Low Fat – 0

Non-Edible – 1

Regular - 2

DISCUSSION:

Until now data preprocessing is going on for the dataset and preparing for the model building.