In [30]:
```python
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [9]:
```python
df_train= pd.read_csv('train.csv')
df_test= pd.read_csv('test.csv')
```

In [11]:
```python
df_train.head()
```

Out[11]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999
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

In [13]:
```python
df_test.head()
```

Out[13]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	1998
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985

In [15]:
```python
df_train.shape
```

Out[15]:
```
(8523, 12)
```

In [17]:
```python
df_train.isnull().sum()
```

Out[17]:
```
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
```

In [19]:
```python
df_test.isnull().sum()
```

Out[19]:
```
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
```

In [21]:
```
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
```

In [23]:
```
df_train.describe()
```

Out[23]:

	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

In [26]:
```
df_train['Item_Weight'].describe()
```

Out[26]:
```
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
```

In [32]:
```
df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

In [34]:
```
df_train.isnull().sum()
```

Out[34]:
```
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
```

In [36]:
```
df_train['Item_Weight'].describe()
```

Out[36]:
```
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

In [39]:
```
df_train['Outlet_Size'].value_counts()
```

Out[39]:
```
Outlet_Size
Medium    2793
Small     2388
High       932
Name: count, dtype: int64
```

In [41]:
```
df_train['Outlet_Size'].mode()
```

Out[41]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [43]:
```
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)
```

In [45]:
```
df_train.isnull().sum()
```

Out[45]:
```
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
```

In [47]:
```
df_test.isnull().sum()
```

Out[47]:
```
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

In [50]:
```
df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

In [52]:
```
df_train
```

Out[52]:

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

8523 rows × 10 columns

EDA with Dtale Library

In [68]:
```
import dtale
```

In [69]:
```
dtale.show(df_train)
```

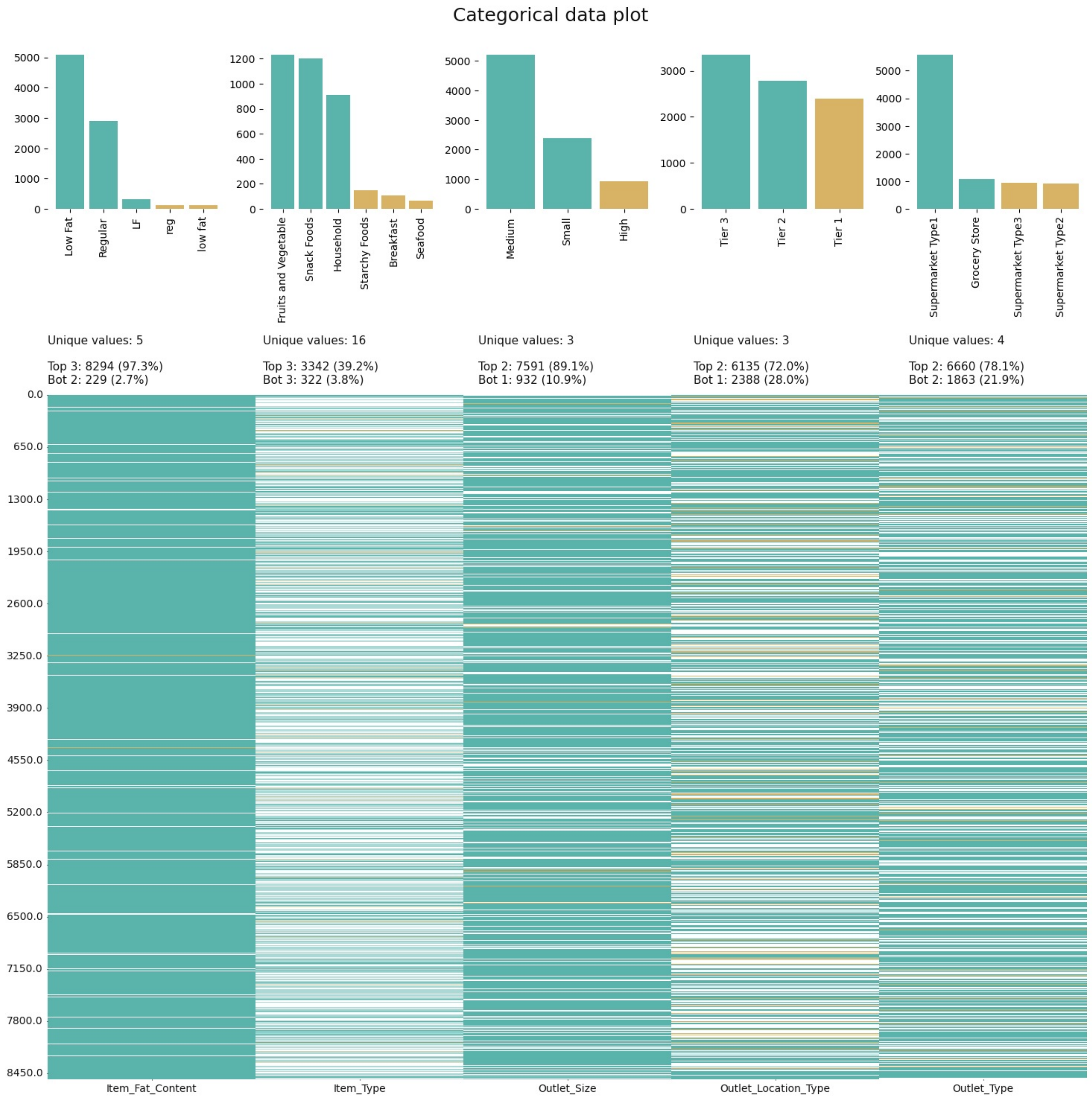
Out[69]:

EDA using Klib Library

In [230…
```
import klib
```

In [232…
```
# klib.describe - functions for visualizing datasets
klib.cat_plot(df_train) # returns a visualization of the number and frequency of categorical features
```

Out[232… GridSpec(6, 5)



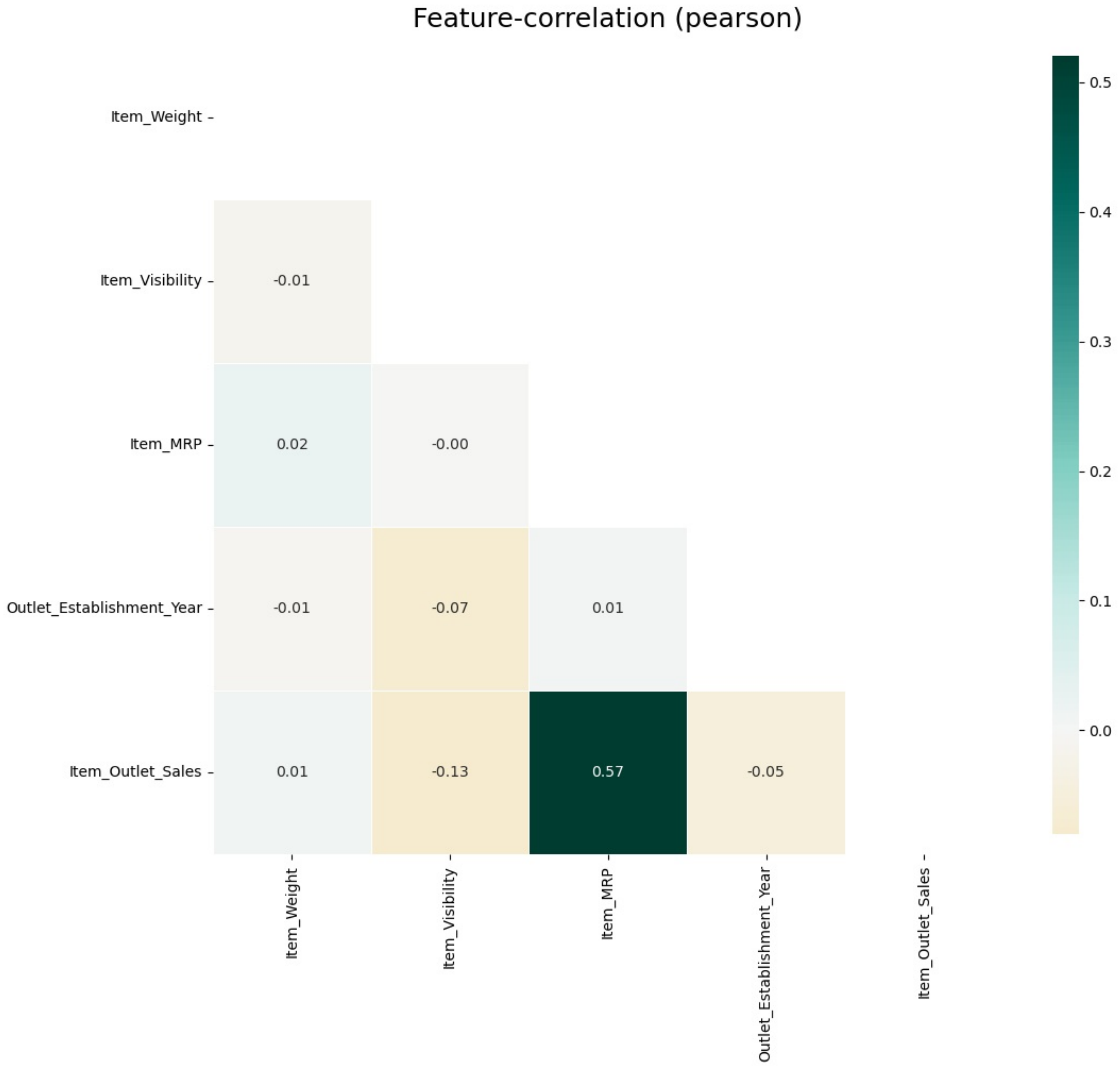
In [235…
```
klib.corr_mat(df_train) # returns a color-encoded correlation matrix
```

Out[235…

	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
Item_Outlet_Sales	0.01	-0.13	0.57	-0.05	1.00

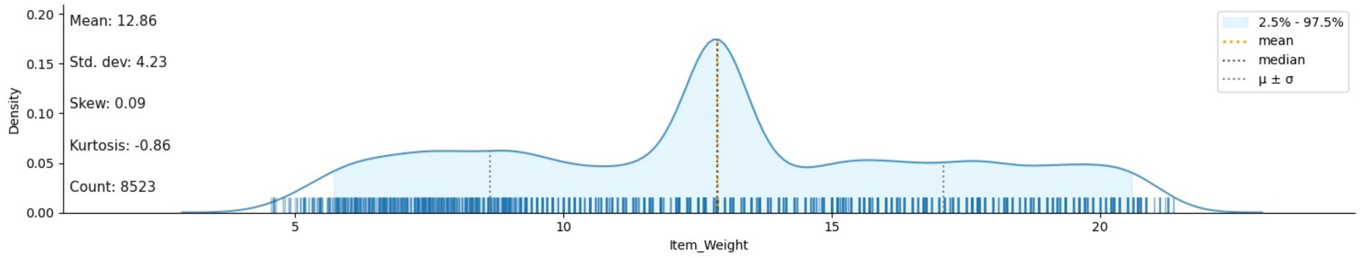
In [237…
```
klib.corr_plot(df_train) # returns a color-encoded heatmap, ideal for correlations
```

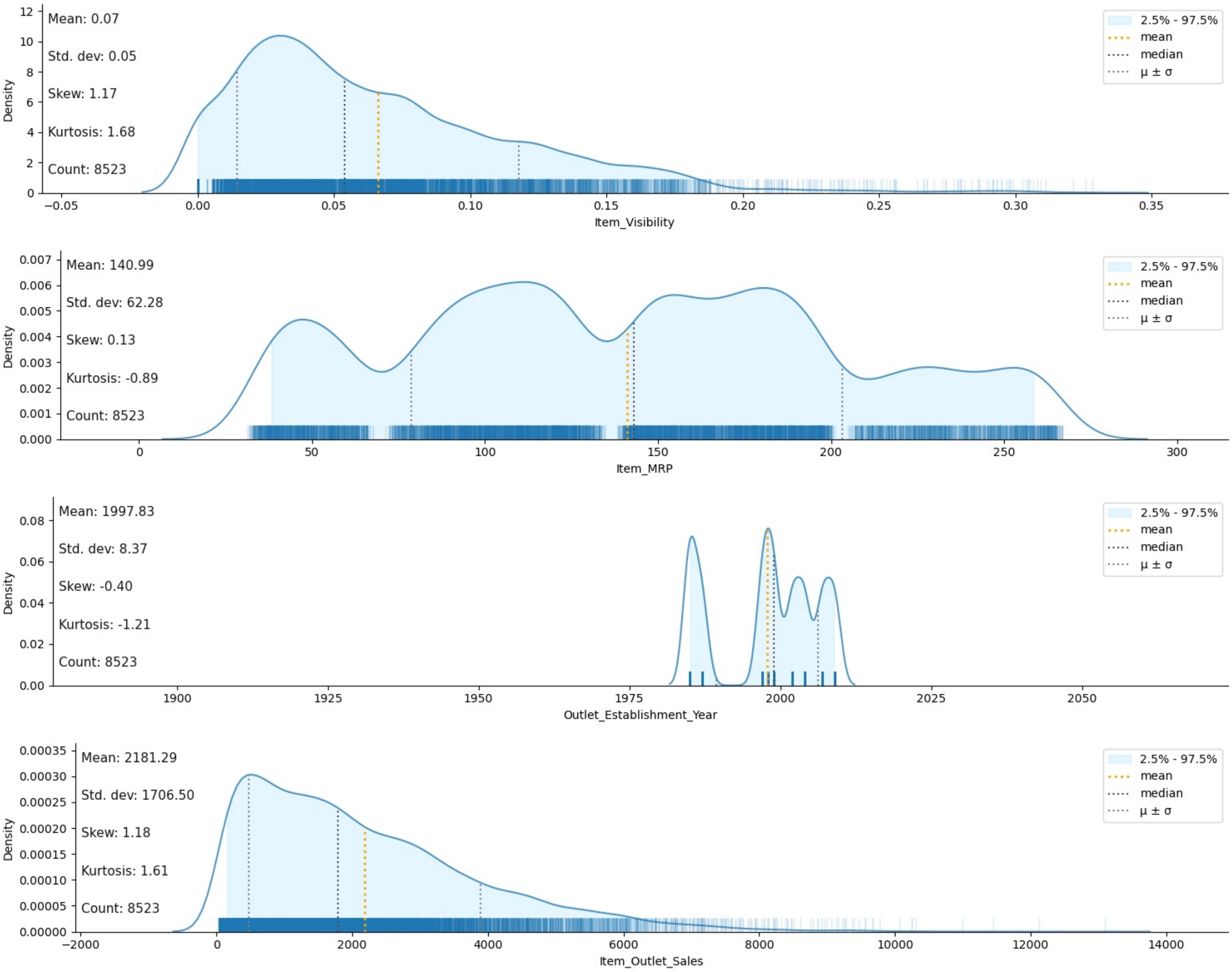
Out[237… <Axes: title={'center': 'Feature-correlation (pearson)'}>




```python
In [239… klib.dist_plot(df_train) # returns a distribution plot for every numeric feature
```

Out[239… <Axes: xlabel='Item_Outlet_Sales', ylabel='Density'>





In [241…
```
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

In [244…
```
# 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


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: 0.46 MB (-70.77%)
```

Out[244…

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

8523 rows × 10 columns

In [246…

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

Out[246…

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

8523 rows × 10 columns

In [248…

```
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
```

In [250…

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

In [252…

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

Out[252…

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

8523 rows × 10 columns

Preprocessing Task before Model Building

1. Label Encoding

In [258…

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

In [260…
```
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'])
```

In [262…
```
df_train
```

Out[262…

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

8523 rows × 10 columns

2. Splitting our data into train and test

In [265…
```
X=df_train.drop('item_outlet_sales',axis=1)
```

In [267…
```
Y=df_train['item_outlet_sales']
```

In [269…
```
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

In [272…
```
X.describe()
```

Out[272…

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_locat
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	852
mean	12.857646	1.369354	0.066132	7.226681	140.992767	1997.831867	1.170832	
std	4.226130	0.644810	0.051598	4.209990	62.275051	8.371760	0.600327	
min	4.555000	0.000000	0.000000	0.000000	31.290001	1985.000000	0.000000	
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	
50%	12.857645	1.000000	0.053931	6.000000	143.012802	1999.000000	1.000000	
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	
max	21.350000	4.000000	0.328391	15.000000	266.888397	2009.000000	2.000000	

In [274…
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
```

In [276…
```
X_train_std= sc.fit_transform(X_train)
```

In [278…
```
X_test_std= sc.transform(X_test)
```

In [280…
```
X_train_std
```

Out[280… 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]])

In [282…
```
X_test_std
```

Out[282… 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]])

In [284…
```
Y_train
```

Out[284…
```
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
```

In [286…
```
Y_test
```

Out[286…
```
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
```

Model Building

In [299…
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

In [301…
```
lr.fit(X_train_std,Y_train)
```

Out[301…
▾ LinearRegression

LinearRegression()

In [303…
```
X_test.head()
```

Out[303…

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

In [305…
```
Y_pred_lr=lr.predict(X_test_std)
```

In [307…
```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

In [309…
```
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
```

In [311…
```
from sklearn.ensemble import RandomForestRegressor
rf= RandomForestRegressor(n_estimators=1000)
```

In [313…
```
rf.fit(X_train_std,Y_train)
```

Out[313…

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

In [314…
```
Y_pred_rf= rf.predict(X_test_std)
```

In [315…
```
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)))
```

```
0.5502867249136534
781.7839205871022
1107.0829685725055
```

Hyper Parameter Tuning

In [317…
```
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,
                           scoring='r2',error_score=0,verbose=2,cv=2)

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: 0.549 using {'n_estimators': 1000}
0.507 (0.008) with: {'n_estimators': 10}
0.548 (0.007) with: {'n_estimators': 100}
0.549 (0.006) with: {'n_estimators': 1000}
```

In [318…
```
grid_search_forest.best_params_
```
Out[318…
```
{'n_estimators': 1000}
```
In [320…
```
grid_search_forest.best_score_
```
Out[320…
```
0.5491399629884066
```
In [323…
```
Y_pred_rf_grid=grid_search_forest.predict(X_test_std)
```
In [324…
```
r2_score(Y_test,Y_pred_rf_grid)
```
Out[324…
```
0.5490552021660631
```
The End

In [ ]:
```

```