Data Preprocessing

```
# Steps of preprocessing of data

1.Import necessasary library
2.Read Datset
3.sanity check of data
4.Exploratory Data Analysis(EDA)
5.Missing Value treatments
6.Outliers treatments
7.Duplicates & garbage value treatments
8.NormaliZation
9.Encoding of Data
```

1.Import neccessary library

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

2.Read Dataset

In [184…
```
df=pd.read_csv('bigmart_data.csv')
```

head()

In [158…
```
df.head()
```

Out[158…

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
1	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
2	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
3	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
4	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088

tail()

In [51]:
```
df.tail()
```

Out[51]:

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
8517	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	High	Tier 3	Supermarket Type1	2778.3834
8518	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	2002	NaN	Tier 2	Supermarket Type1	549.2850
8519	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	Small	Tier 2	Supermarket Type1	1193.1136
8520	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	Medium	Tier 3	Supermarket Type2	1845.5976
8521	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	Small	Tier 1	Supermarket Type1	765.6700

3.sanity check of data

shape()

In [53]:
```
#shape
df.shape
```

Out[53]:
```
(8522, 12)
```

Info()

In [55]:
```
#info()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8522 entries, 0 to 8521
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   FDA15              8522 non-null   object 
 1   9.3                7059 non-null   float64
 2   Low Fat            8522 non-null   object 
 3   0.016047301        8522 non-null   float64
 4   Dairy              8522 non-null   object 
 5   249.8092           8522 non-null   float64
 6   OUT049             8522 non-null   object 
 7   1999               8522 non-null   int64  
 8   Medium             6112 non-null   object 
 9   Tier 1             8522 non-null   object 
 10  Supermarket Type1  8522 non-null   object 
 11  3735.138           8522 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.1+ KB
```

finding the missing values

In [57]:
```
df.isnull().sum()
```

Out[57]:
```
FDA15                 0
9.3                1463
Low Fat               0
0.016047301           0
Dairy                 0
249.8092              0
OUT049                0
1999                  0
Medium             2410
Tier 1                0
Supermarket Type1     0
3735.138              0
dtype: int64
```

In [59]:
```
(df.isnull().sum()/df.shape[0])*100
```

Out[59]:
```
FDA15              0.000000
9.3               17.167332
Low Fat            0.000000
0.016047301        0.000000
Dairy              0.000000
249.8092           0.000000
OUT049             0.000000
1999               0.000000
Medium            28.279747
Tier 1             0.000000
Supermarket Type1  0.000000
3735.138           0.000000
dtype: float64
```

In [61]:
```
(df.isnull().sum()/len(df))*100
```

Out[61]:
```
FDA15              0.000000
9.3               17.167332
Low Fat            0.000000
0.016047301        0.000000
Dairy              0.000000
249.8092           0.000000
OUT049             0.000000
1999               0.000000
Medium            28.279747
Tier 1             0.000000
Supermarket Type1  0.000000
3735.138           0.000000
dtype: float64
```

finding the duplicates

In [63]:
```
df.duplicated().sum()
```

Out[63]:
```
0
```

Identifying garbage values

garbage values always in the form of object data types

In [71]:
```
for i in df.select_dtypes(include='object').columns:
    print(df[i].value_counts())
    print("*****"*10)
```

```
FDA15
FDW13    10
FDG33    10
FDX31     9
FDF56     9
NCI54     9
         ..
FDK57     1
FDY43     1
FDO33     1
DRF48     1
FDE52     1
Name: count, Length: 1559, dtype: int64
**************************************************
Low Fat
Low Fat    5088
Regular    2889
LF          316
reg         117
low fat     112
Name: count, dtype: int64
**************************************************
Dairy
Fruits and Vegetables    1232
Snack Foods              1200
Household                 910
Frozen Foods              856
Dairy                     681
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
**************************************************
OUT049
OUT027    935
OUT013    932
OUT046    930
OUT035    930
OUT049    929
OUT045    929
OUT018    928
OUT017    926
OUT010    555
OUT019    528
Name: count, dtype: int64
**************************************************
Medium
Medium    2792
Small     2388
High       932
Name: count, dtype: int64
**************************************************
Tier 1
Tier 3    3350
Tier 2    2785
Tier 1    2387
Name: count, dtype: int64
**************************************************
Supermarket Type1
Supermarket Type1    5576
Grocery Store        1083
Supermarket Type3     935
Supermarket Type2     928
Name: count, dtype: int64
**************************************************
```

4.Exploratory Data Analysis(EDA)

descriptive statistics

In [77]:
```
df.describe().T
```

Out[77]:

	count	mean	std	min	25%	50%	75%	max
9.3	7059.0	12.858149	4.643592	4.555	8.77250	12.600000	16.850000	21.350000
0.016047301	8522.0	0.066138	0.051598	0.000	0.02700	0.053935	0.094594	0.328391
249.8092	8522.0	140.980013	62.267562	31.290	93.81795	142.979900	185.625950	266.888400
1999	8522.0	1997.831730	8.372242	1985.000	1987.00000	1999.000000	2004.000000	2009.000000
3735.138	8522.0	2181.106580	1706.516719	33.290	833.91450	1794.331000	3100.963500	13086.964800

In [ ]:
```
# for objective data types
```

In [79]:
```
df.describe(include='object')
```

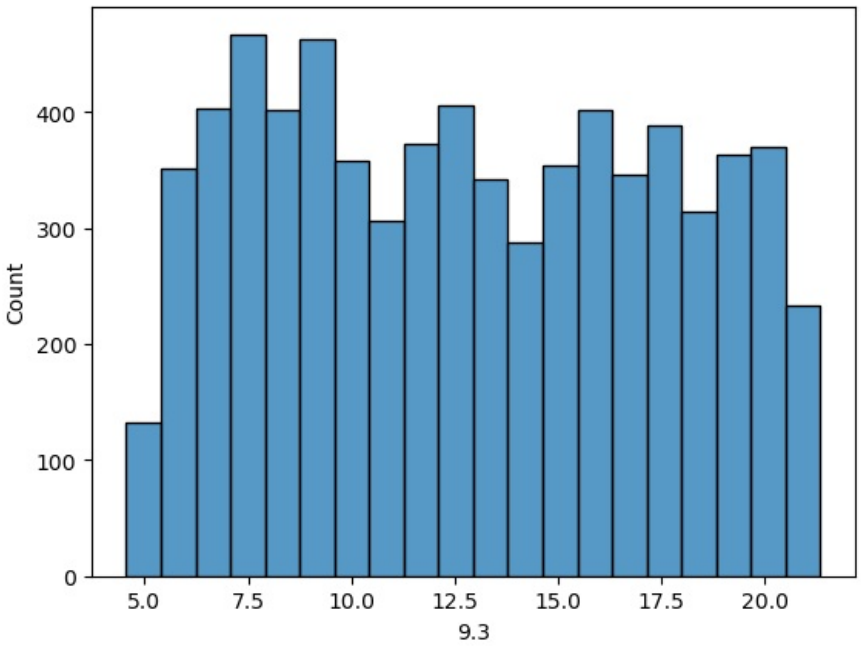
Out[79]:

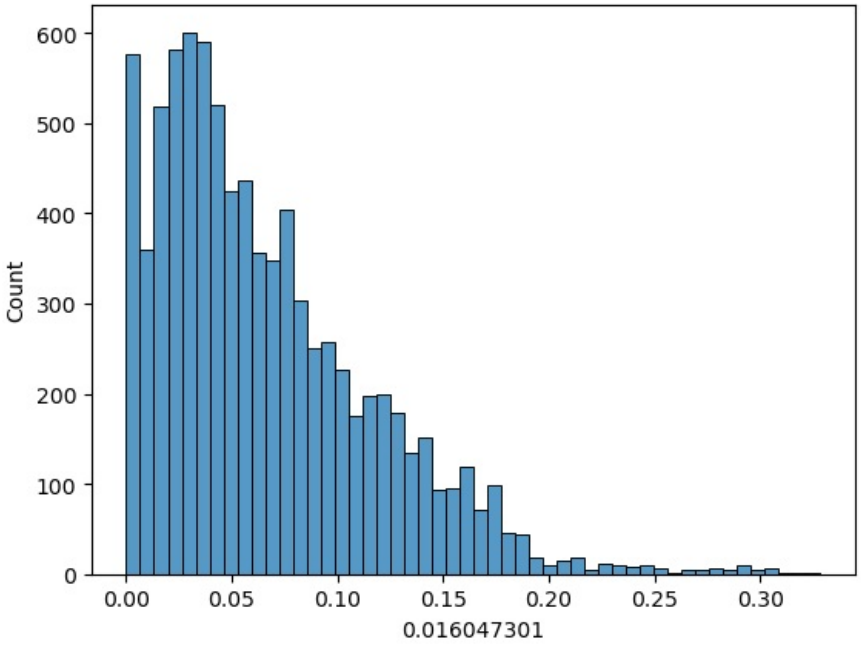
	FDA15	Low Fat	Dairy	OUT049	Medium	Tier 1	Supermarket Type1
count	8522	8522	8522	8522	6112	8522	8522
unique	1559	5	16	10	3	3	4
top	FDW13	Low Fat	Fruits and Vegetables	OUT027	Medium	Tier 3	Supermarket Type1
freq	10	5088	1232	935	2792	3350	5576

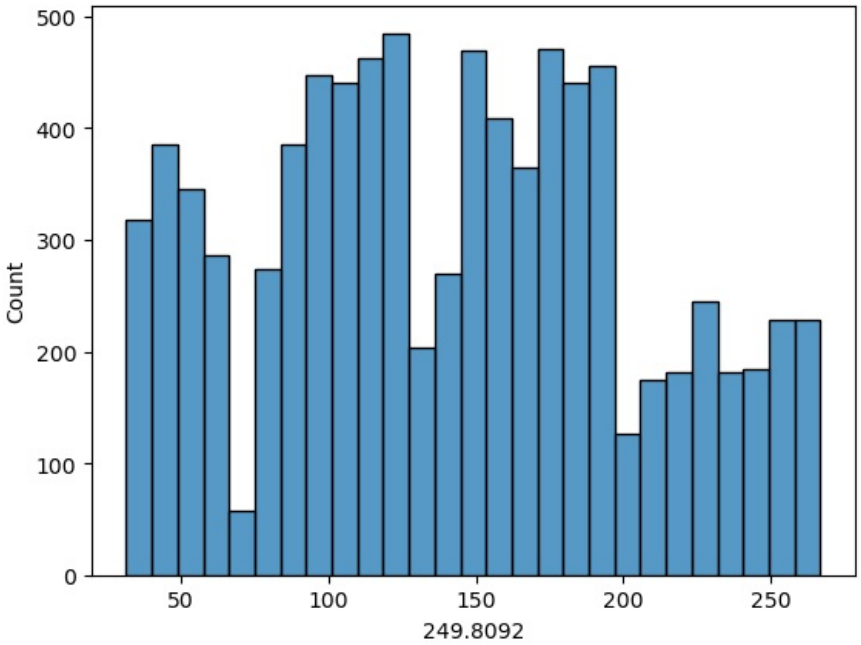
histogram to understand the distribution

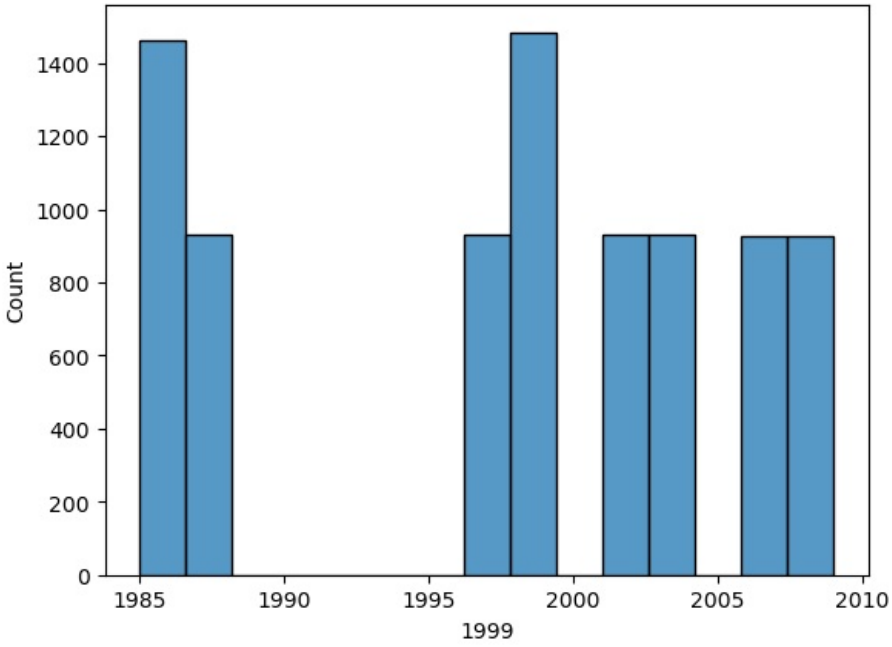
for understanding the distrbution of the data

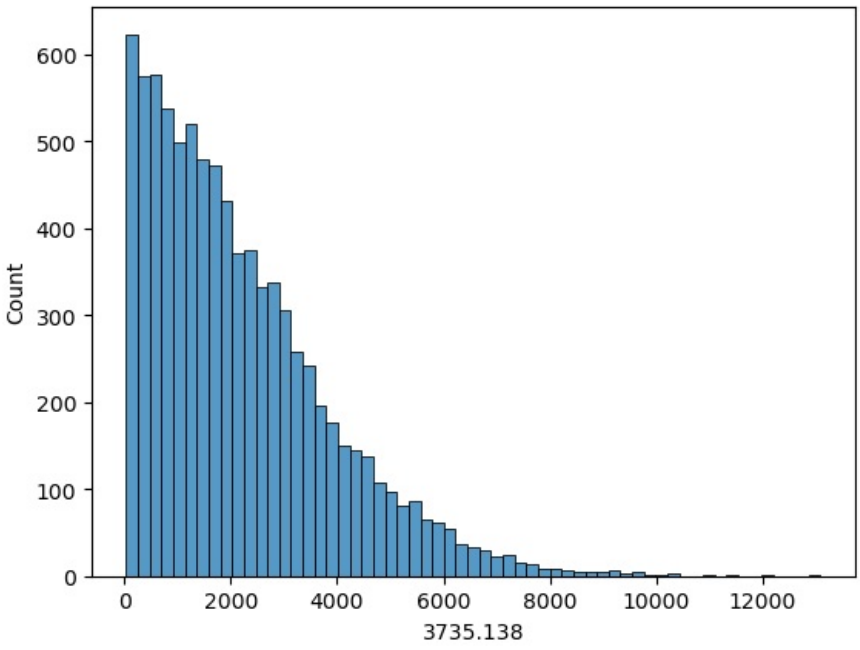
In [86]:
```
import warnings
warnings.filterwarnings("ignore")
for i in df.select_dtypes(include='number').columns:
    sns.histplot(data=df,x=i)
    plt.show()
```




Count
600
500
400
300
200
100
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.016047301

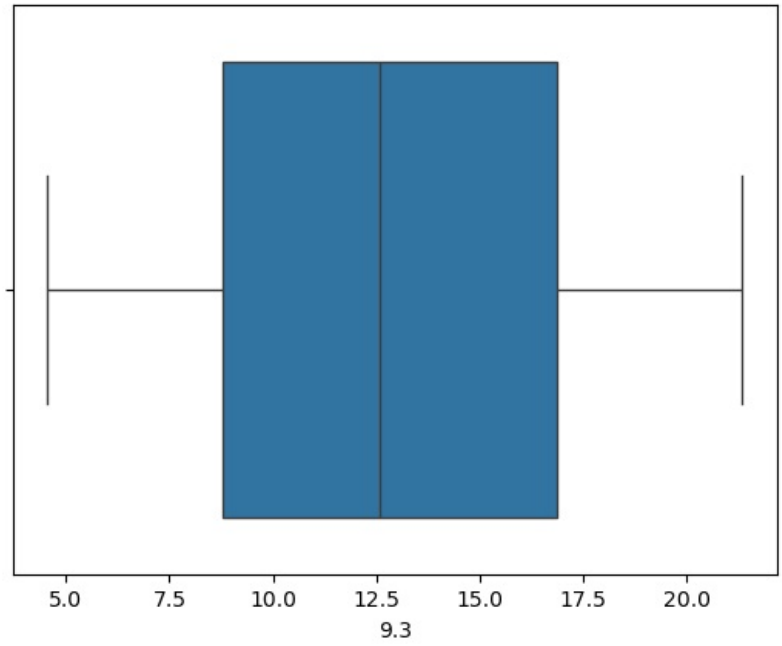

Count
500
400
300
200
100
0
50
100
150
200
250
249.8092

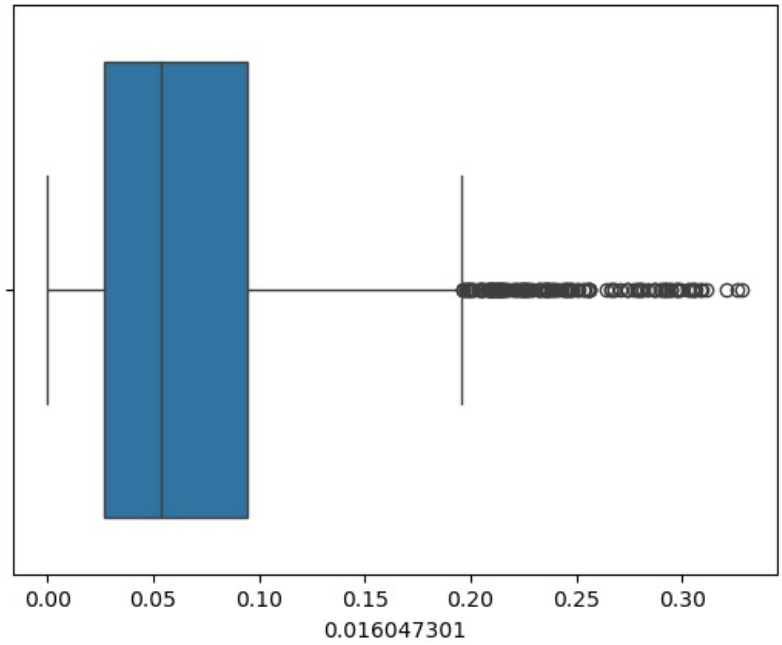


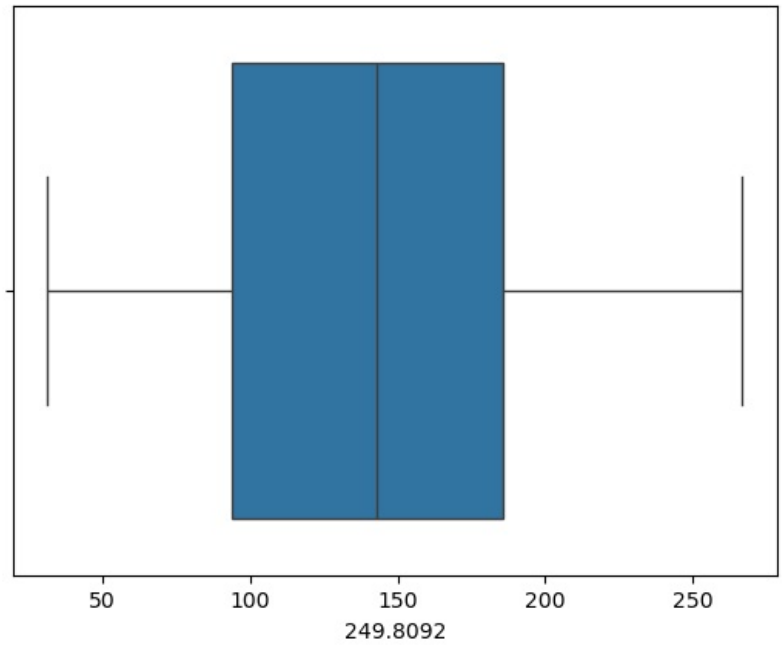


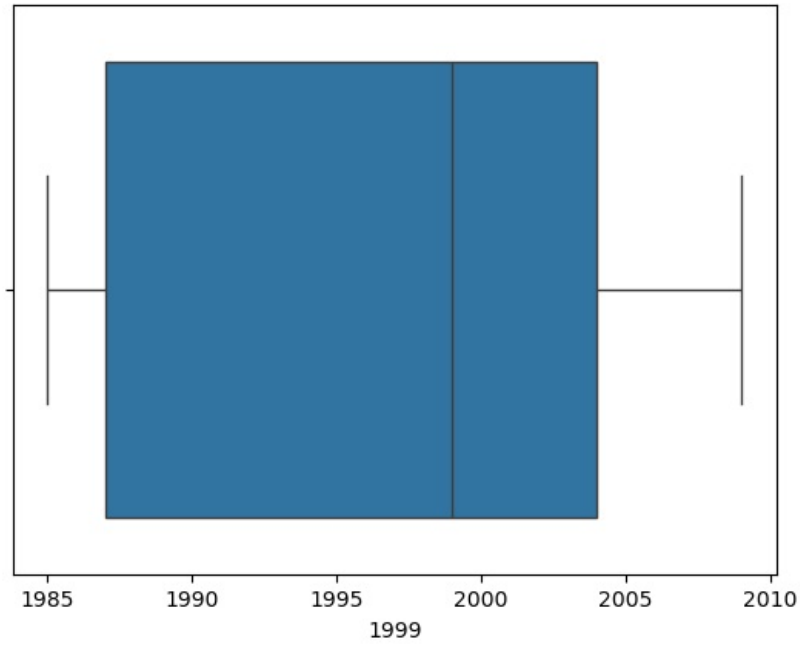
Box-plot -to-identify the outliers

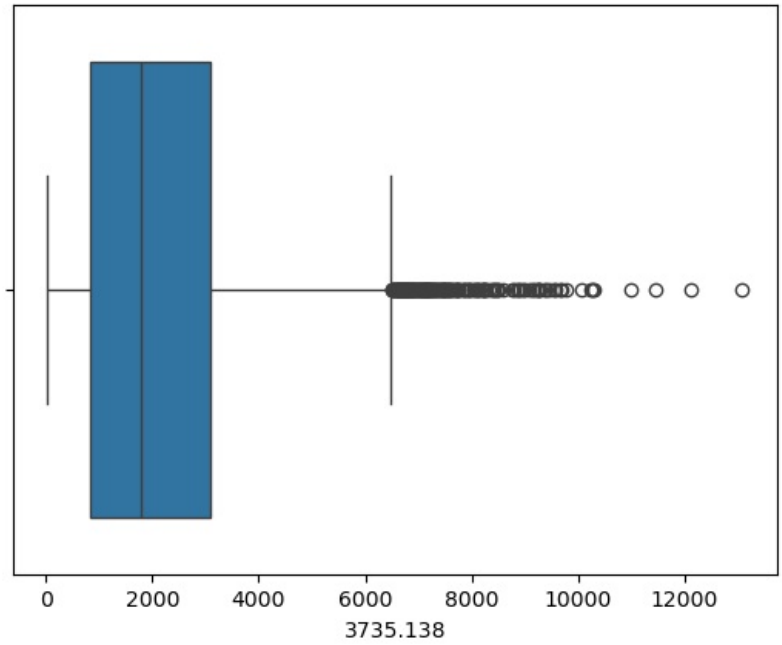
In [90]:
```
import warnings
warnings.filterwarnings("ignore")
for i in df.select_dtypes(include='number').columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```


5.0
7.5
10.0
12.5
15.0
17.5
20.0
9.3


0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.016047301


50
100
150
200
250
249.8092





scatter plot to undesrand the relationships

here it is used to show the relationship between the target variable and independent variable.

In [100…
```
df.head()
```

Out[100…

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
1	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
2	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
3	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052
4	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	2009	Medium	Tier 3	Supermarket Type2	556.6088

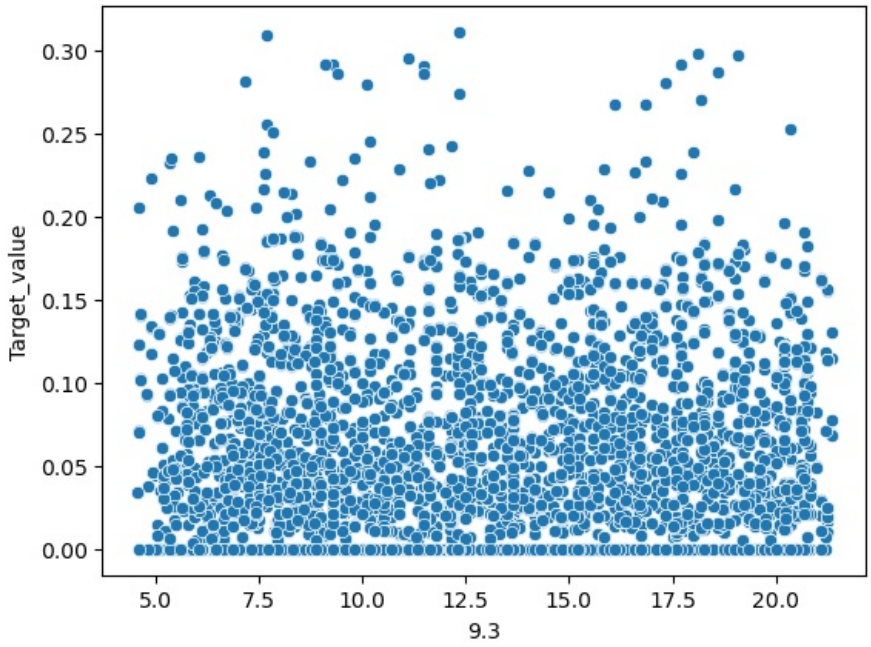
In [114…
```
df.select_dtypes(include='number').columns
```

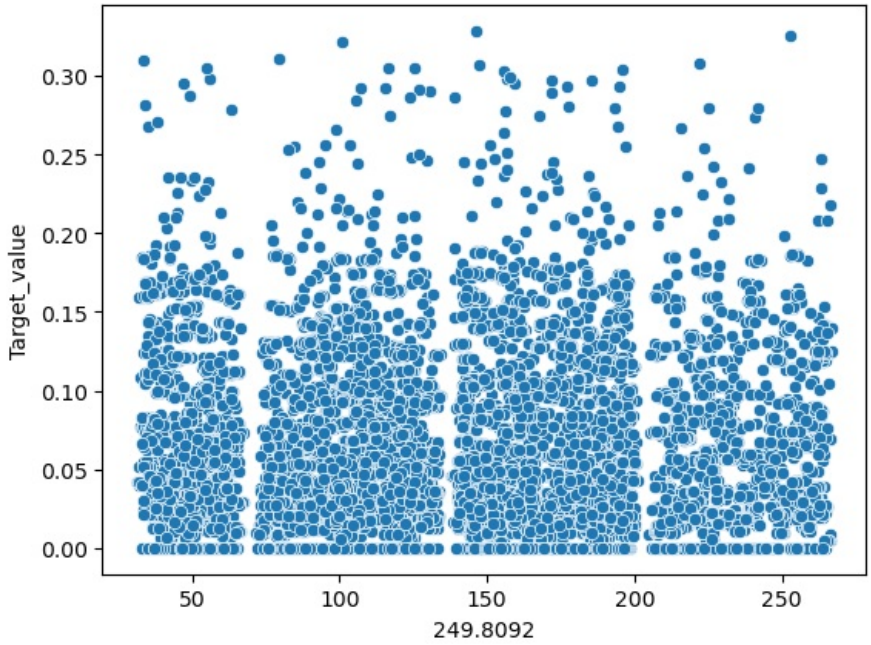
Out[114… `Index(['9.3', '249.8092', '1999', '3735.138', 'Target_value'], dtype='object')`

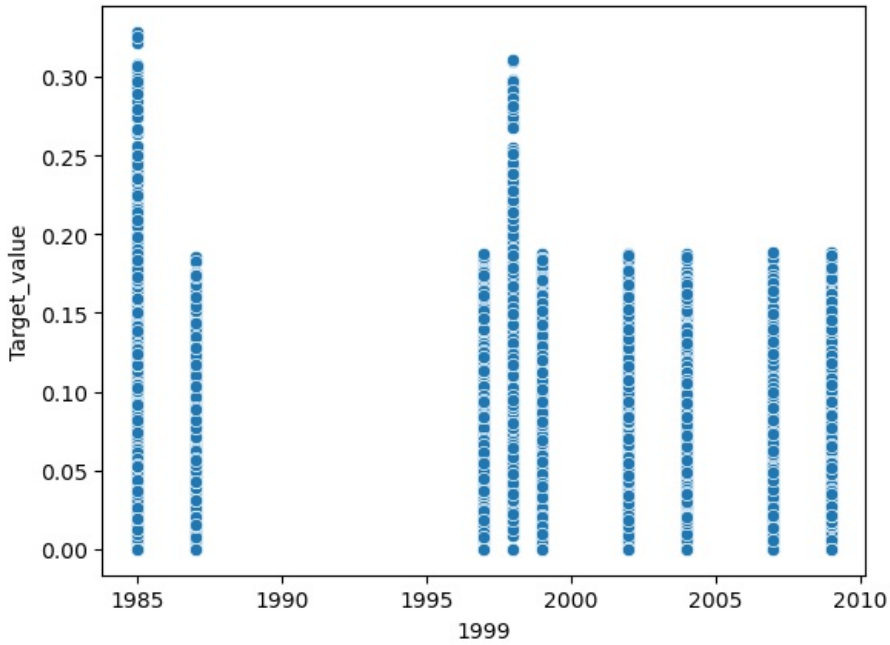
In [112…
```
df.drop(columns='0.016047301',inplace=True)
```

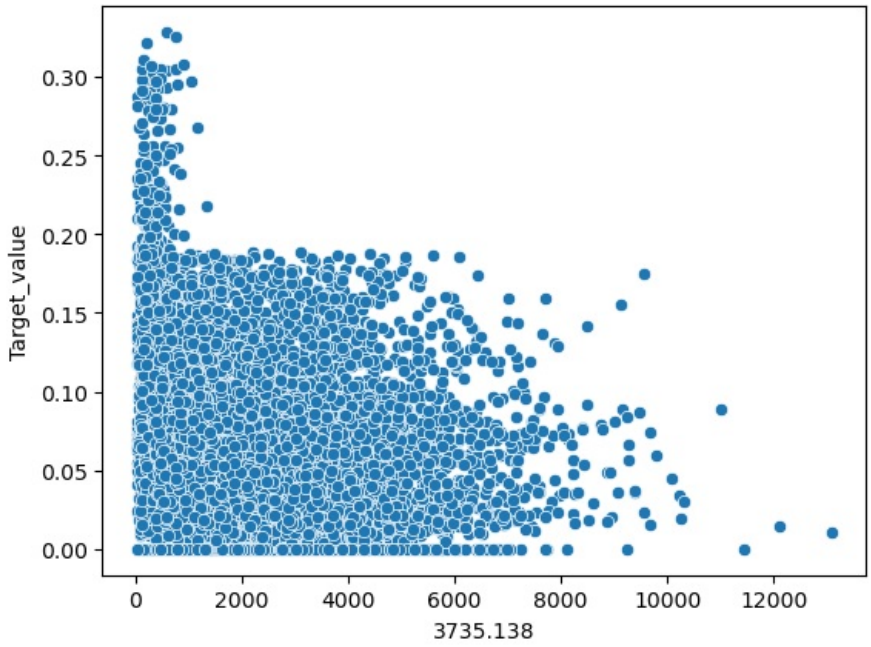
In [118…

```
for i in ['9.3', '249.8092', '1999', '3735.138']:
    sns.scatterplot(data=df,x=i,y= 'Target_value')
    plt.show()
```









correlation with heatmaps to interpret the relation and multicolliniarity

In [128…
```
s=df.select_dtypes(include='number').corr()
s
```

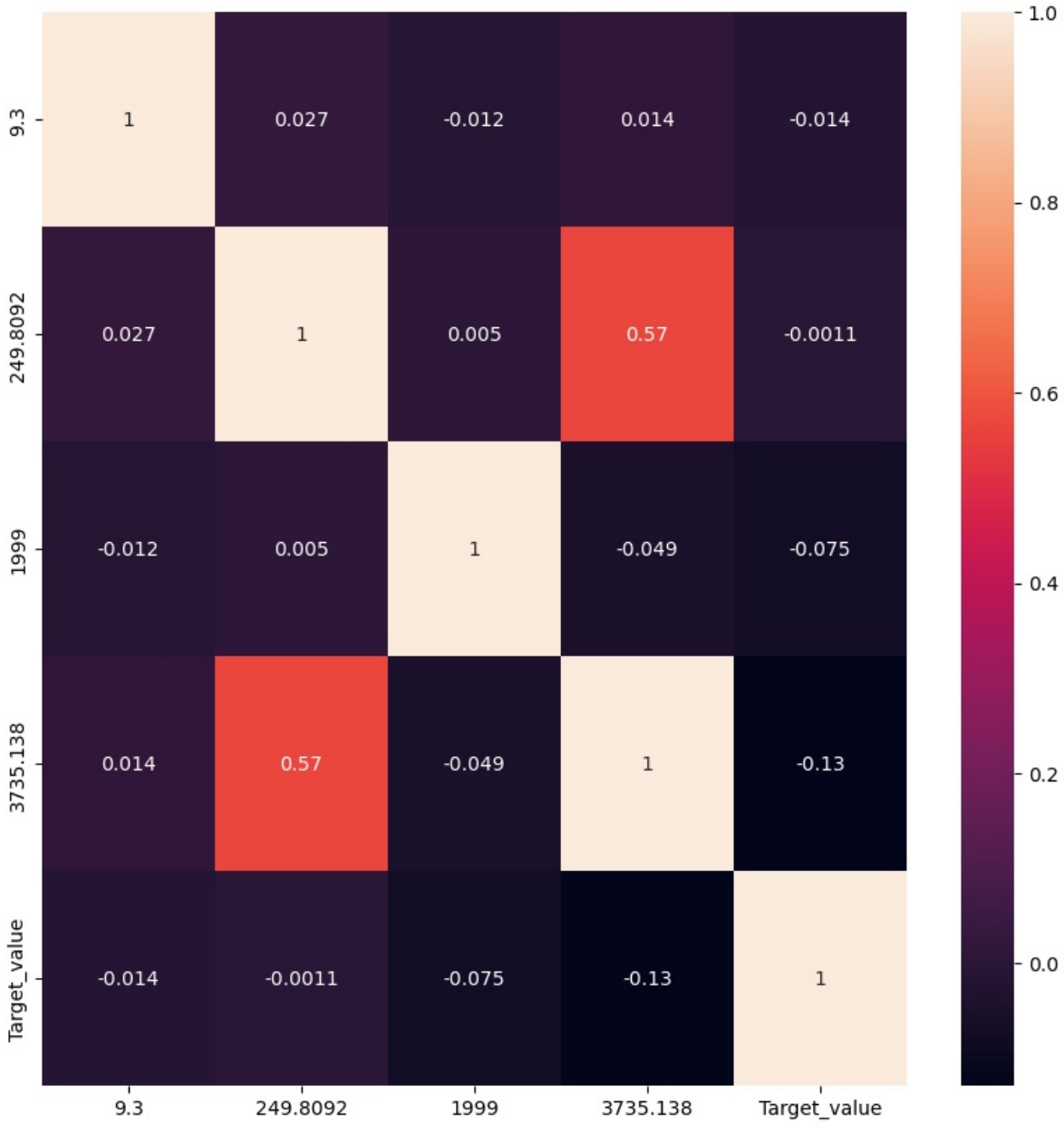
Out[128…

	9.3	249.8092	1999	3735.138	Target_value
9.3	1.000000	0.027337	-0.011613	0.014239	-0.014156
249.8092	0.027337	1.000000	0.004992	0.567517	-0.001116
1999	-0.011613	0.004992	1.000000	-0.049152	-0.074822
3735.138	0.014239	0.567517	-0.049152	1.000000	-0.128534
Target_value	-0.014156	-0.001116	-0.074822	-0.128534	1.000000

```
# heatmap , for knowing the correlation ship between the columns we use the heatmap
```

In [140…
```
plt.figure(figsize=(10,10))
sns.heatmap(s,annot=True)  #annot is for showing the values on the boxes
```

Out[140… <Axes: >



5.Missing Value treatments

In [ ]:
```
#choose the method pf imputing missing values
```

```
#like mean,median,mode or KNNIputer--> for filling with  numerical value

#mode-> for str
```

In [160…
```
df.isnull().sum()
```

Out[160…
```
FDA15                 0
9.3                1463
Low Fat               0
0.016047301           0
Dairy                 0
249.8092              0
OUT049                0
1999                  0
Medium             2410
Tier 1                0
Supermarket Type1     0
3735.138              0
dtype: int64
```

In [162…
```
for i in ['9.3']:
    df[i].fillna(df[i].mean(),inplace=True)
```

In [164…
```
df.isnull().sum()
```

Out[164…
```
FDA15                 0
9.3                   0
Low Fat               0
0.016047301           0
Dairy                 0
249.8092              0
OUT049                0
1999                  0
Medium             2410
Tier 1                0
Supermarket Type1     0
3735.138              0
dtype: int64
```

In [196…
```
df['Medium'].fillna(df['Medium'].mode()[0],inplace=True)
```

In [178…
```
df.isnull().sum()
```

Out[178…
```
FDA15              0
9.3                0
Low Fat            0
0.016047301        0
Dairy              0
249.8092           0
OUT049             0
1999               0
Medium             0
Tier 1             0
Supermarket Type1  0
3735.138           0
dtype: int64
```

In [182…
```
from sklearn.impute import KNNImputer
impute=KNNImputer()
```

In [194…
```
for i in df.select_dtypes(include='number').columns:
    df[i]=impute.fit_transform(df[[i]])

#It will take the k nearest value to fill the values,
```

In [198…
```
df.isnull().sum()
```

Out[198…
```
FDA15              0
9.3                0
Low Fat            0
0.016047301        0
Dairy              0
249.8092           0
OUT049             0
1999               0
Medium             0
Tier 1             0
Supermarket Type1  0
3735.138           0
dtype: int64
```

6.Outliers treatements

This line of code is a common technique used to filter out outliers in a DataFrame based on the interquartile range (IQR). Here's a step-by-step breakdown:

In [ ]:
```
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape  explain this
```

1.Calculate IQR:-> Q1 and Q3 are the 1st and 3rd quartiles of each column,and IQR(interquartile range) is calculated as:

IQR=Q3-Q1

2.Define Outliers Boundaries:
* Lower bound:Q1-1.5*IQR
* Upper bound:Q3+1.5*IQR
* Any data point outside these bounds is considered an outlier.

3.Create Filter Boundaries:
* The expression df<(Q1-1.5*IQR) checks for values below the lower bound.
* Similarly ,df>(Q3 + 1.5*IQR) checks for values above the upper bound.
* | is used as a logical OR to capture values either below the lower bound or above the upper bound.
* The ~ symbol negates the filter,so df[..] selects only rows without outliers.

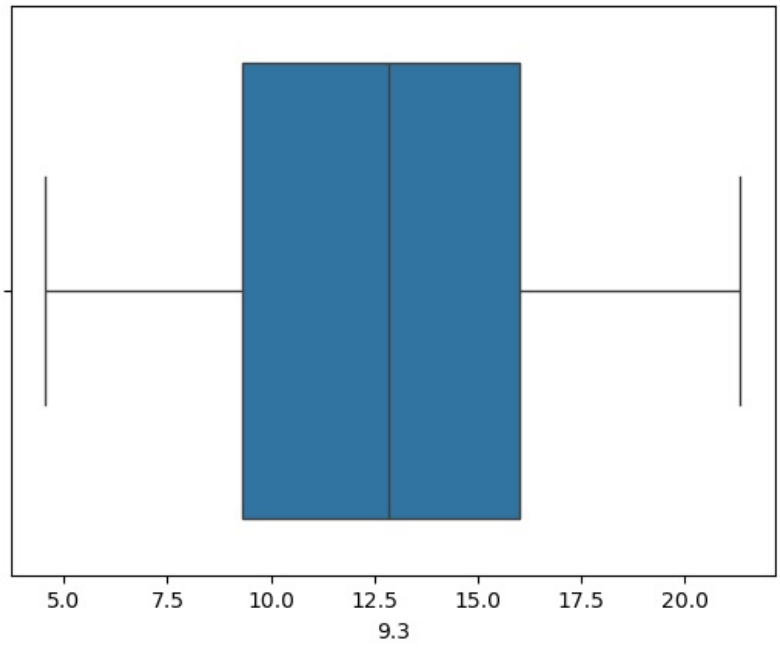
4.Filter Rows.
* .any(axis=1) checks if any value in each row is True (indicating an outliers).
* This line removes rows containing any outliers accross columns.

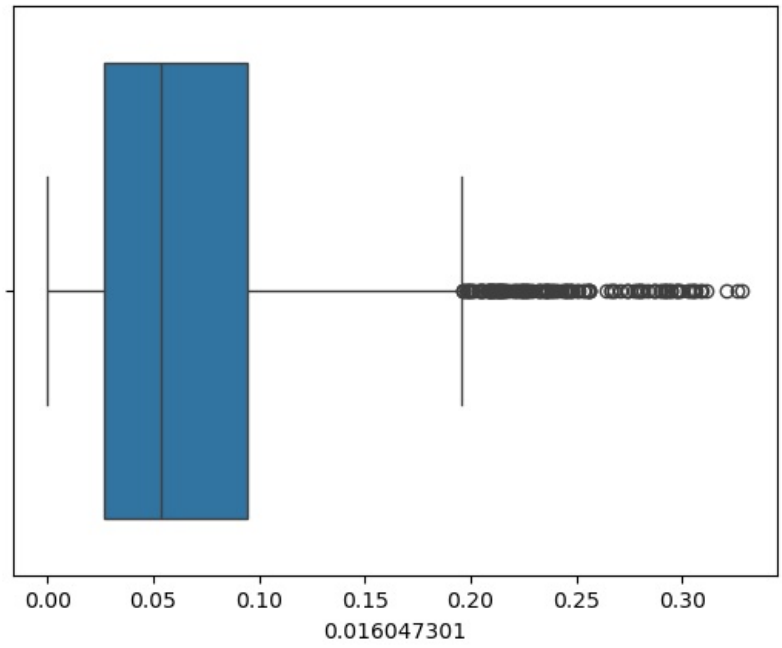
5.Result
    * df.shape returns the shape of the filtered DataFrame, showing the number of rows and columns after outlier removal.

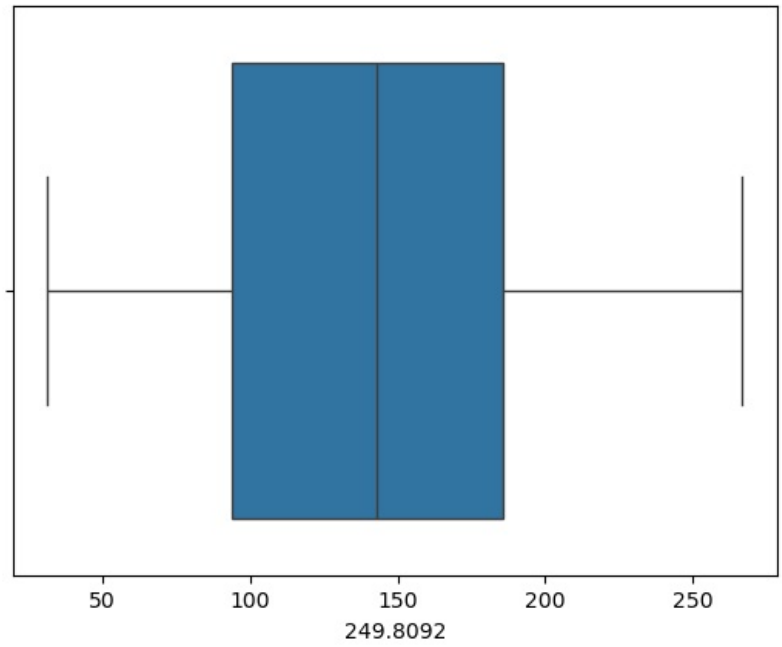
decide weather to do outliers treatment or not ,if do how?

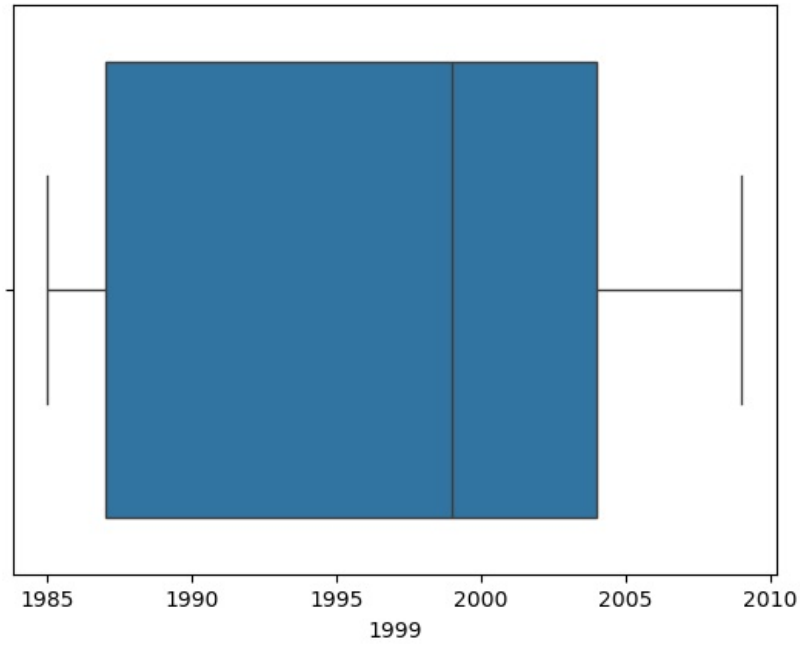
It is only for continual numerical data we can't to do any of objects and any non-continuous data columns

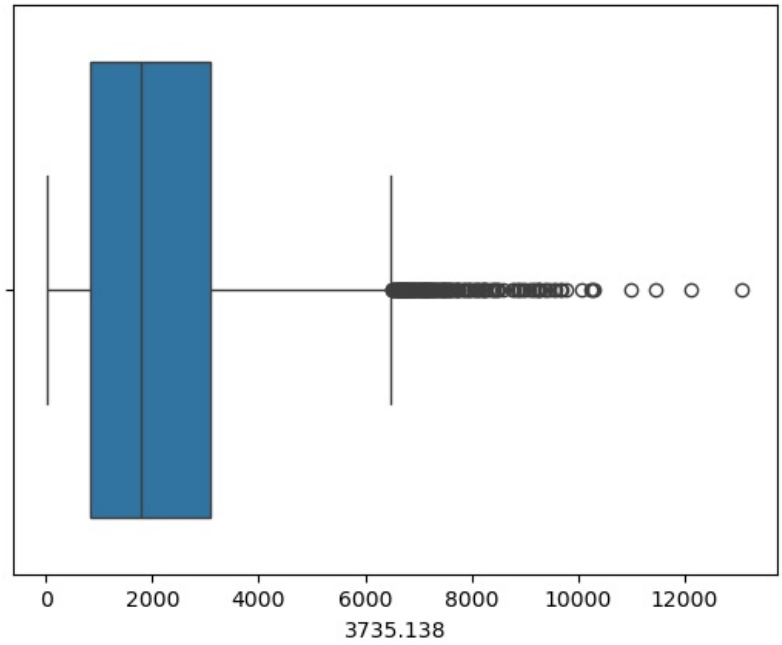
In [223…
```
for i in df.select_dtypes(include='number').columns:
    sns.boxplot(data=df,x=i)
    plt.show()
```




0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.016047301


50
100
150
200
250
249.8092


1985
1990
1995
2000
2005
2010
1999



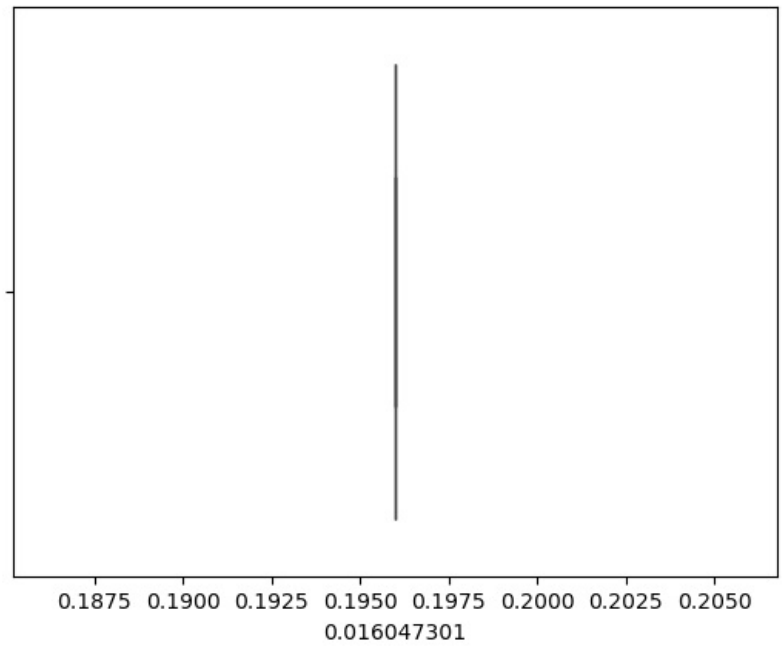
In [3]:
```
def whisker(col):
    q1,q3=np.percentile(col,[25,75])
    IQR=q3-q1
    lw=q1-1.5*IQR
    uw=q3+1.5*IQR
    return lw,uw
```

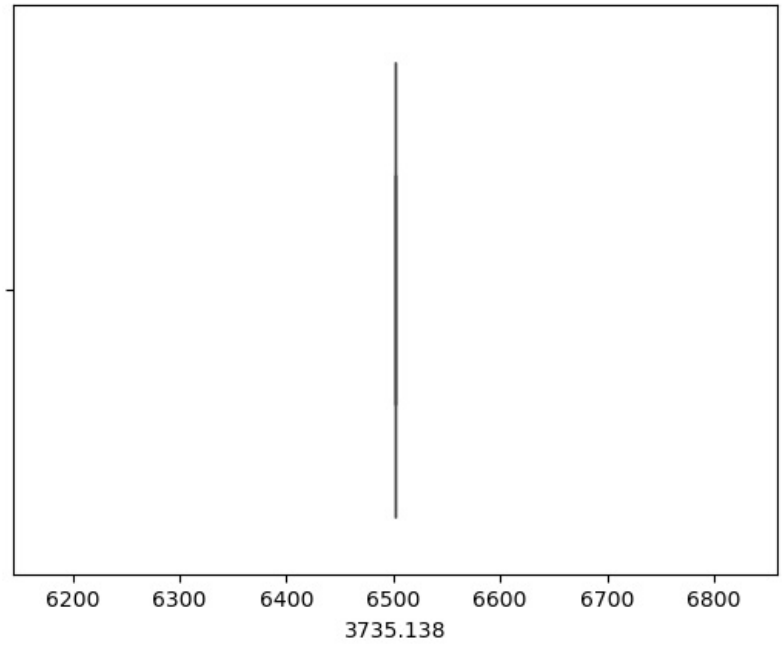
In [221…
```
whisker(df['9.3'])
```

Out[221… (-0.7249999999999996, 26.035)

In [235…
```
for i in ['0.016047301','3735.138']:
    lw,uw=whisker(df[i])
    df[i]=np.where(df[i]<lw,lw,df[i])
    df[i]=np.where(df[i]>uw,uw,df[i])
```

In [243…
```
for i in ['0.016047301','3735.138']:
    sns.boxplot(data=df,x=df[i])
    plt.show()
```





7.duplicates & garbage value treatments

In [ ]:
```
# check for duplicates if we have any unique column in the data set
#clean the garbage value
```

In [252…
```
df.drop_duplicates().head()
```

Out[252…

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.195986	Soft Drinks	48.2692	OUT018	2009.0	Medium	Tier 3	Supermarket Type2	6501.537
1	FDN15	17.500	Low Fat	0.195986	Meat	141.6180	OUT049	1999.0	Medium	Tier 1	Supermarket Type1	6501.537
2	FDX07	19.200	Regular	0.195986	Fruits and Vegetables	182.0950	OUT010	1998.0	Medium	Tier 3	Grocery Store	6501.537
3	NCD19	8.930	Low Fat	0.195986	Household	53.8614	OUT013	1987.0	High	Tier 3	Supermarket Type1	6501.537
4	FDP36	10.395	Regular	0.195986	Baking Goods	51.4008	OUT018	2009.0	Medium	Tier 3	Supermarket Type2	6501.537

8.Encoding of Data

To convert object data types into numerical to understand the module when we will be going for machine learning

In [ ]:
```
# Do label encoding and one hot encoding with pd.getdummies
```

In [258…
```
df['Medium'].unique()
df.head()
```

Out[258…

	FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.138
0	DRC01	5.920	Regular	0.195986	Soft Drinks	48.2692	OUT018	2009.0	Medium	Tier 3	Supermarket Type2	6501.537
1	FDN15	17.500	Low Fat	0.195986	Meat	141.6180	OUT049	1999.0	Medium	Tier 1	Supermarket Type1	6501.537
2	FDX07	19.200	Regular	0.195986	Fruits and Vegetables	182.0950	OUT010	1998.0	Medium	Tier 3	Grocery Store	6501.537
3	NCD19	8.930	Low Fat	0.195986	Household	53.8614	OUT013	1987.0	High	Tier 3	Supermarket Type1	6501.537
4	FDP36	10.395	Regular	0.195986	Baking Goods	51.4008	OUT018	2009.0	Medium	Tier 3	Supermarket Type2	6501.537

In [264…
```
dummy=pd.get_dummies(data=df,columns=['Medium','Low Fat'],drop_first=True).head()
```

In [266…
```
dummy
```

Out[266…

	FDA15	9.3	0.016047301	Dairy	249.8092	OUT049	1999	Tier 1	Supermarket Type1	3735.138	Medium_Medium	Medium_Sm
0	DRC01	5.920	0.195986	Soft Drinks	48.2692	OUT018	2009.0	Tier 3	Supermarket Type2	6501.537	True	Fal
1	FDN15	17.500	0.195986	Meat	141.6180	OUT049	1999.0	Tier 1	Supermarket Type1	6501.537	True	Fal
2	FDX07	19.200	0.195986	Fruits and Vegetables	182.0950	OUT010	1998.0	Tier 3	Grocery Store	6501.537	True	Fal
3	NCD19	8.930	0.195986	Household	53.8614	OUT013	1987.0	Tier 3	Supermarket Type1	6501.537	False	Fal
4	FDP36	10.395	0.195986	Baking Goods	51.4008	OUT018	2009.0	Tier 3	Supermarket Type2	6501.537	True	Fal

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: