type-and-sales-prediction-1

June 7, 2024

0.1 # BIG MART PRICE ANALYSIS AND PREDICTION

USMAN JILLANI
Department of Computer Science, Superior University, Lahore, Pakistan

0.2 ##INTRODUCTION

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store. Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales We used Supervised machine learning and target value will be SALE The Aim to create model that can predict the sales per product for each store.Using machine learing models to help us to increase Big mart sales.

0.3 ##Dataset information

This dataset has 8523 values and 12 features in which 7 qualitative features and 5 quantitative features these columns names are below table. ####Qualitative Features Item_Identifier

1. Item_Fat_Content

2. Item_Type

3. Outlet_Identifier
4. Outlet_Size
5. Outlet_Location ####Quantitative Features
6. Item_Weight
7. Item_Visibility
8. Item_MRP
9. Outlet_Establishment_Year .

###LIBRARIES

[292]:
```
import pandas as pd #It is used for working with data sets
import numpy as np #It is used to perform a wide variety of mathematics
import matplotlib.pyplot as plt #It is used collection of functions
import seaborn as sns #It is used data visualization and graphical
import plotly.express as px # IT used for plots
%matplotlib inline
```

###READ DATASET

This dataset download from kaggle and datset link is https://www.kaggle.com/datasets/brijbhushannanda1979/big
sales-data

[293]:
```
data=pd.read_csv('/content/Train.csv')
```

Checked Dataset rows and columns

[294]:
```
data.shape
```

[294]:
```
(8523, 12)
```

Dataset Desciption

[295]:
```
data.describe()
```

[295]:
```
       Item_Weight  Item_Visibility     Item_MRP  Outlet_Establishment_Year  \
count  7060.000000      8523.000000  8523.000000                8523.000000
mean     12.857645         0.066132   140.992782                1997.831867
std       4.643456         0.051598    62.275067                   8.371760
min       4.555000         0.000000    31.290000                1985.000000
25%       8.773750         0.026989    93.826500                1987.000000
50%      12.600000         0.053931   143.012800                1999.000000
75%      16.850000         0.094585   185.643700                2004.000000
max      21.350000         0.328391   266.888400                2009.000000

       Item_Outlet_Sales
count        8523.000000
mean         2181.288914
std          1706.499616
min            33.290000
25%           834.247400
50%          1794.331000
75%          3101.296400
max         13086.964800
```

Checked Duplicate Values

[296]:
```
data.duplicated().sum()
```

[296]:
```
0
```

Checked Null values

[297]:
```
data.isnull().sum()
```

[297]:
```
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
```

Most Data scientist say if 30% Feauter have mising values if it is not a unique feature then we droped it . So in this dataset only Item_weight have 17.165317% null values and Outlet_Size have 28.276428% null values so we fill these empty entery. Item_weight is Quantative data so suitable Average is Mean

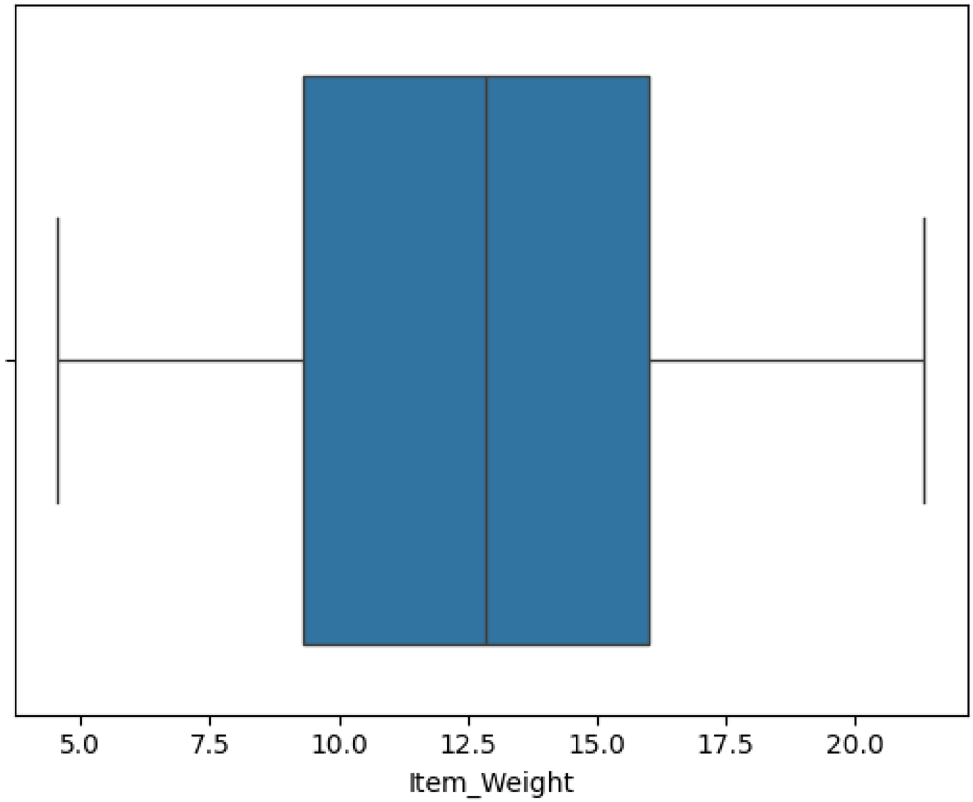
[298]:
```
data['Item_Weight'].fillna(data['Item_Weight'].mean(), inplace=True)
```

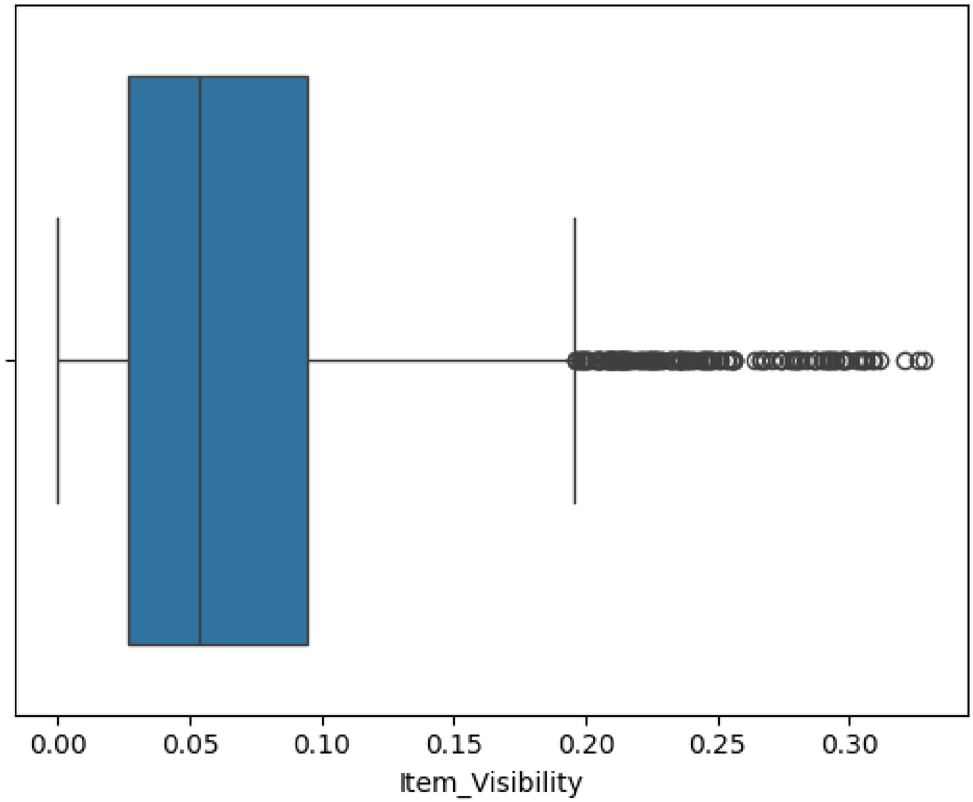
[299]:
```
data.isnull().sum()
```

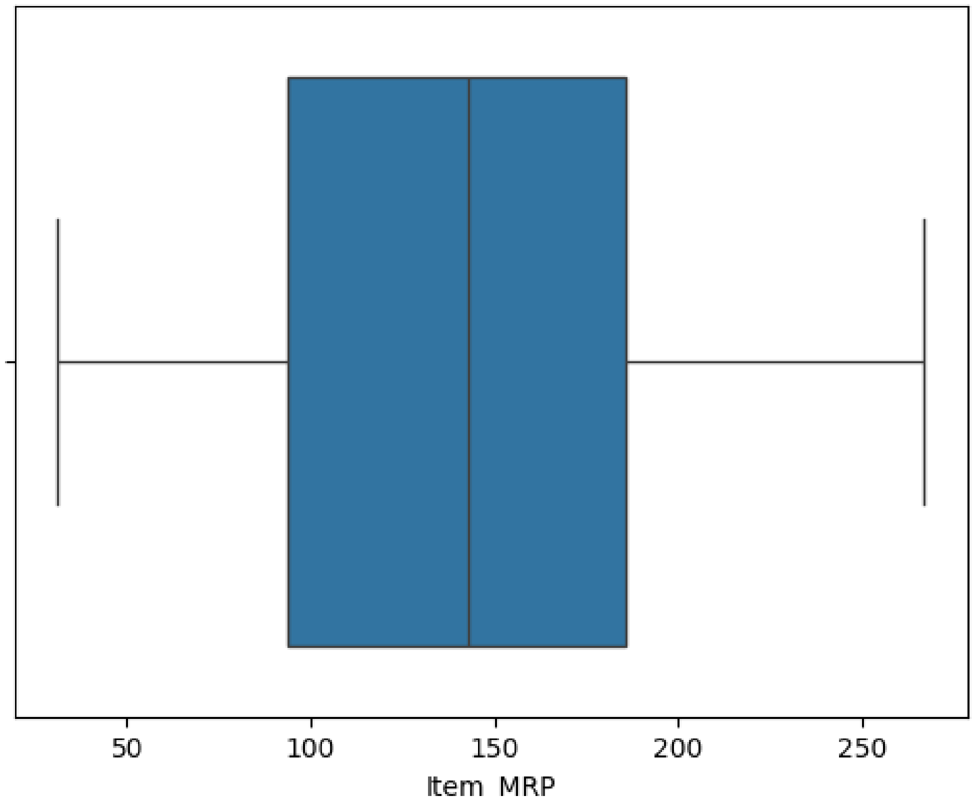
[299]:
```
Item_Identifier                 0
Item_Weight                     0
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

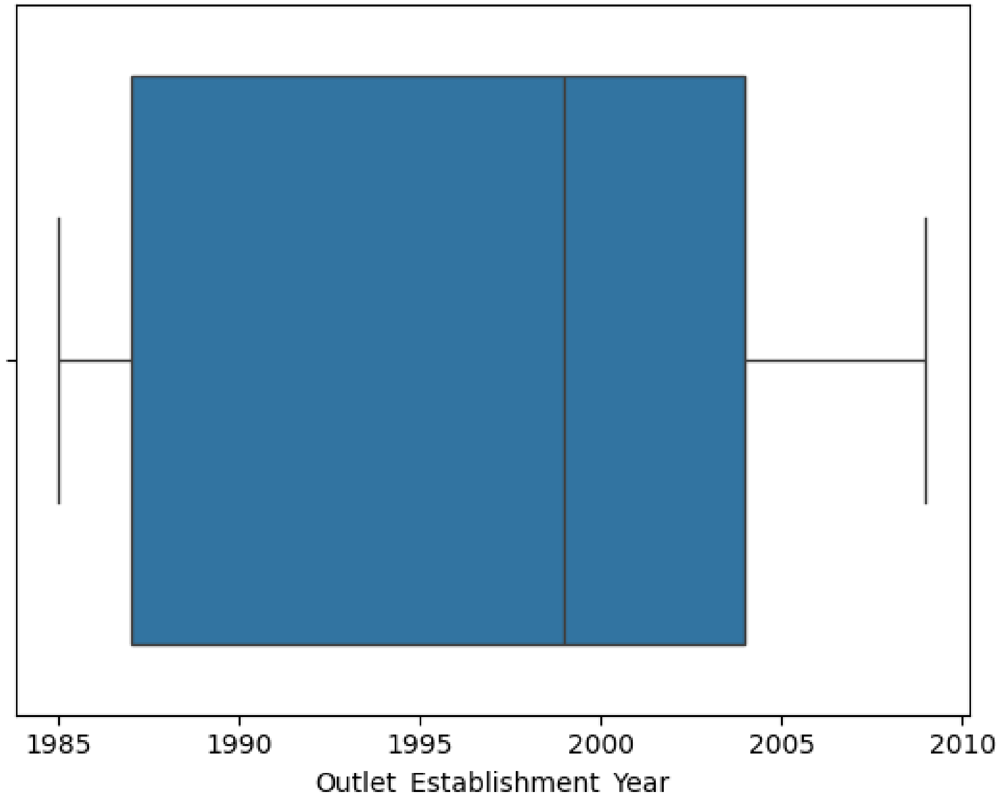
Checked Dataset Outliers

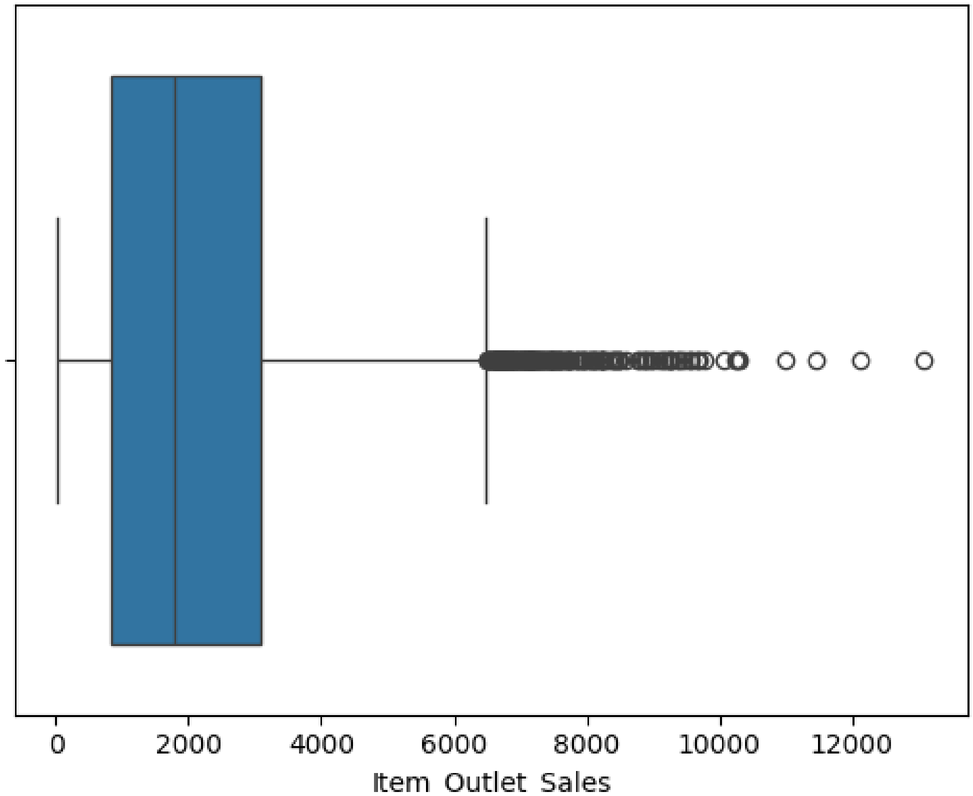
[300]:
```
# prompt: checked outliers all  features

import matplotlib.pyplot as plt
import seaborn as sns

# Create boxplots for each quantitative feature
features = ['Item_Weight', 'Item_Visibility', 'Item_MRP',␣
 ↪'Outlet_Establishment_Year','Item_Outlet_Sales']
for feature in features:
  sns.boxplot(x=data[feature])
  plt.show()
```


5.0
7.5
10.0
12.5
15.0
17.5
20.0
Item_Weight


0.00
0.05
0.10
0.15
0.20
0.25
0.30
Item_Visibility


50
100
150
200
250
Item_MRP


1985
1990
1995
2000
2005
2010
Outlet_Establishment_Year



Remove Outliers in Item_Visibility or Item_Outlet_Sales

[301]:
```
def calculate_outliers(col):
    sorted(col)
    Q1, Q3 = col.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range
```

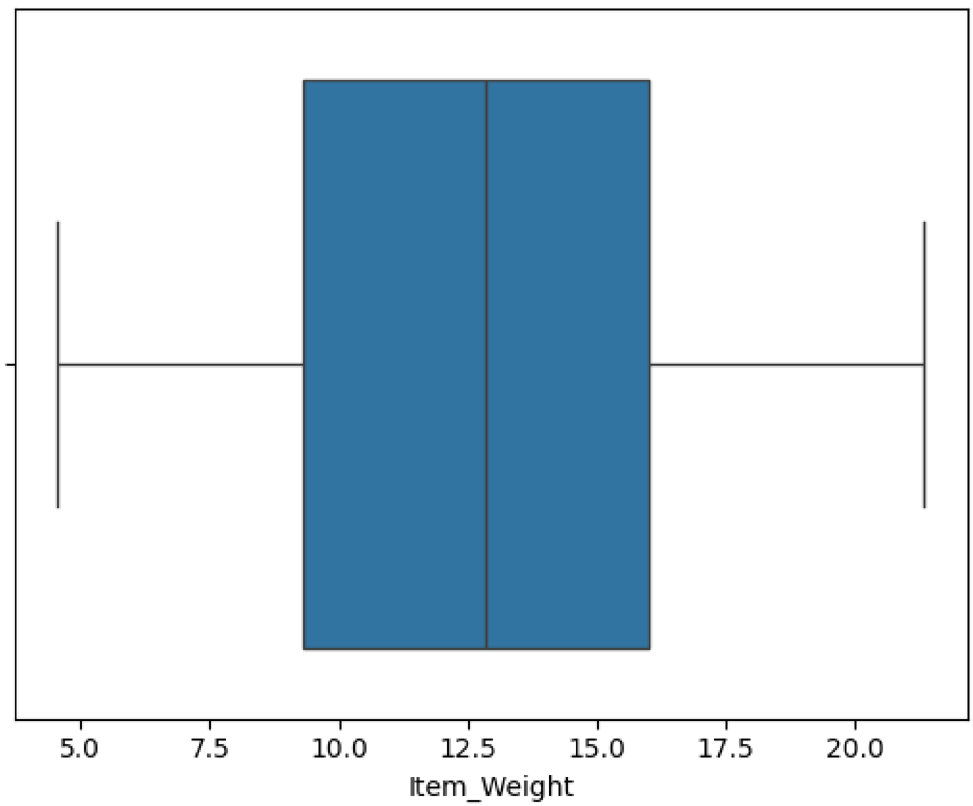
[302]:
```
data['Item_Visibility']=np.log1p(data['Item_Visibility'])
lower_limit, upper_limit = calculate_outliers(data['Item_Visibility']) #lower␣
 ↪and upper range
data['Item_Visibility'] = np.where(data['Item_Visibility'] < lower_limit,␣
 ↪lower_limit, data['Item_Visibility'])
data['Item_Visibility'] = np.where(data['Item_Visibility'] > upper_limit,␣
 ↪upper_limit, data['Item_Visibility'])
```

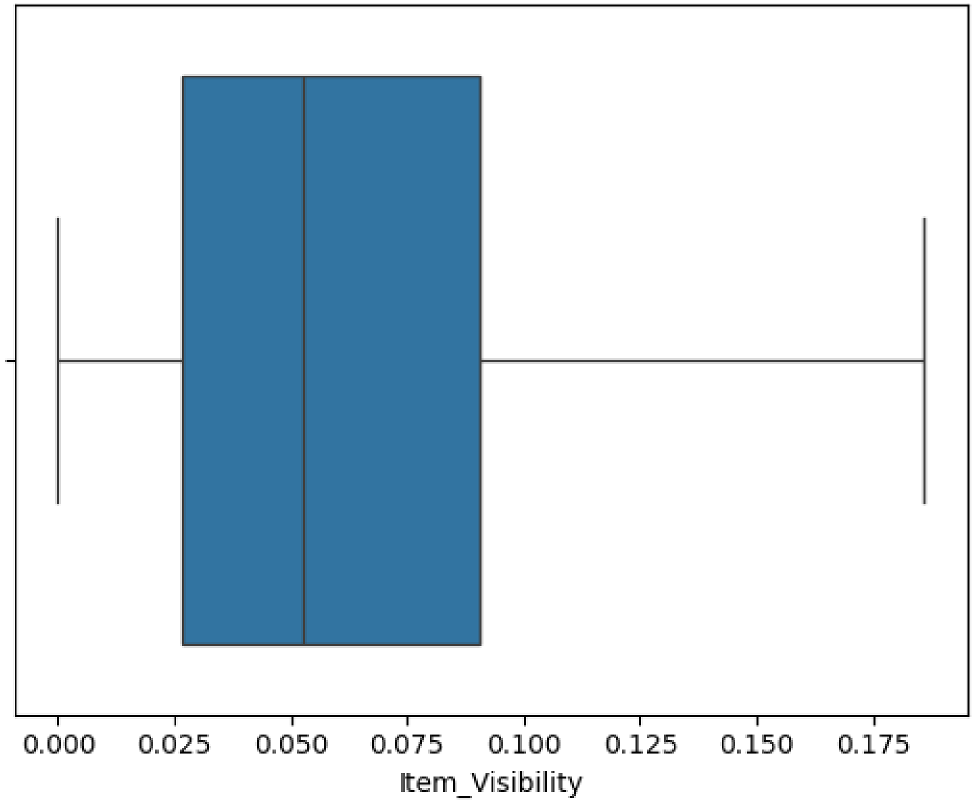
[303]:
```
data['Item_Outlet_Sales']=np.log1p(data['Item_Outlet_Sales'])
lower_limit, upper_limit = calculate_outliers(data['Item_Outlet_Sales']) #lower␣
 ↪and upper range
```

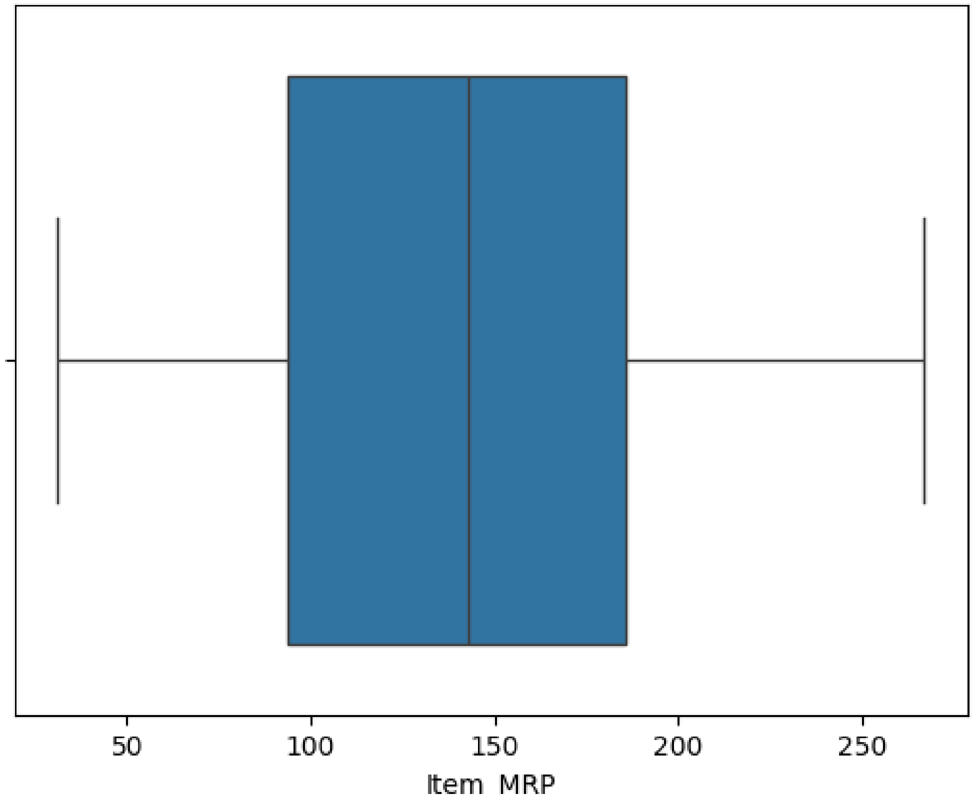
```
data['Item_Outlet_Sales'] = np.where(data['Item_Outlet_Sales'] < lower_limit,␣
↪lower_limit, data['Item_Outlet_Sales'])
data['Item_Outlet_Sales'] = np.where(data['Item_Outlet_Sales'] > upper_limit,␣
↪upper_limit, data['Item_Outlet_Sales'])
```

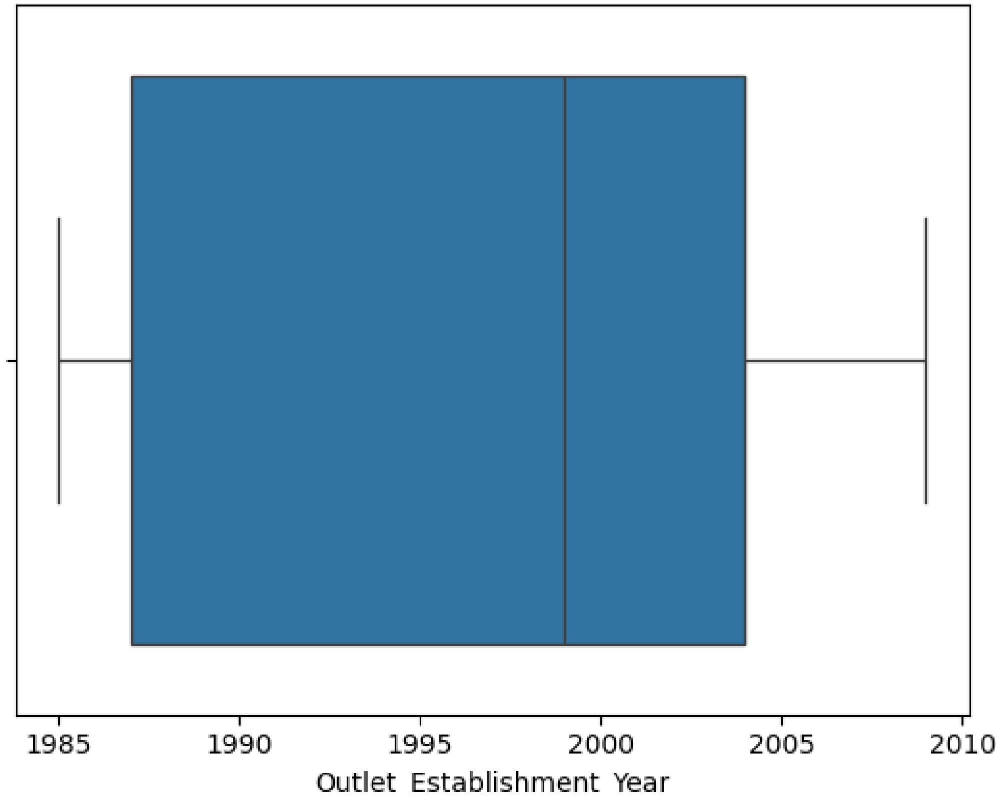
Now Checked Outliers

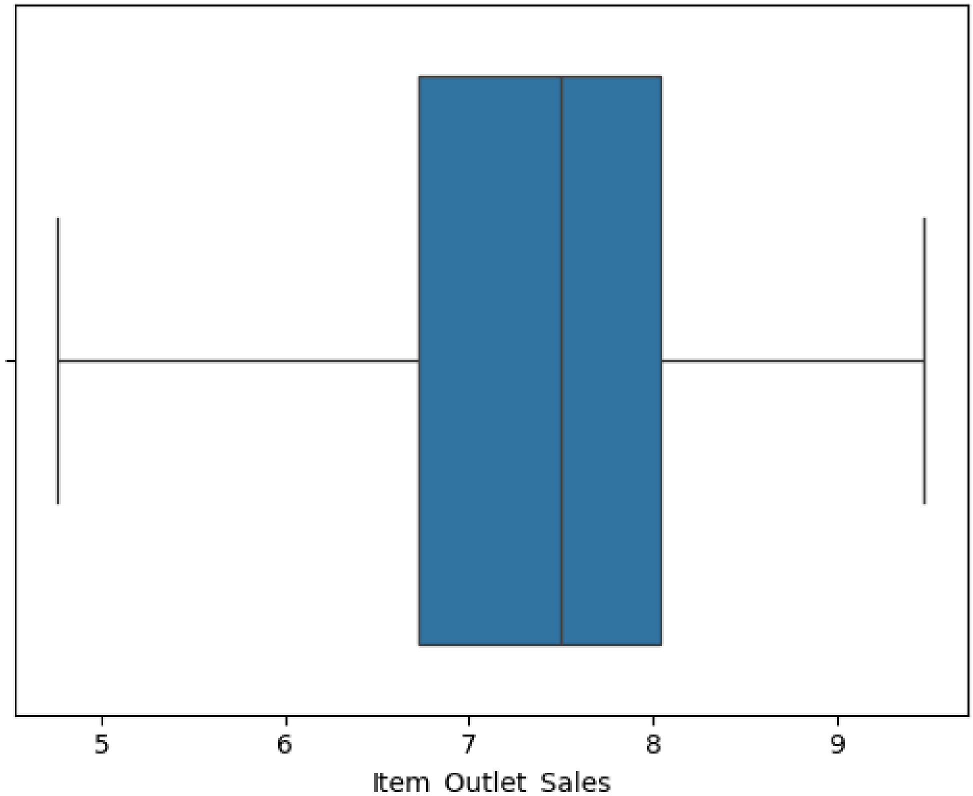
[304]:
```
# Create boxplots for each quantitative feature
features = ['Item_Weight', 'Item_Visibility', 'Item_MRP',␣
↪'Outlet_Establishment_Year','Item_Outlet_Sales']
for feature in features:
  sns.boxplot(x=data[feature])
  plt.show()
```




0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Item_Visibility


50
100
150
200
250
Item_MRP


1985
1990
1995
2000
2005
2010
Outlet_Establishment_Year



Drop 2 features that are not importnat

[305]:
```
data=data.drop(['Item_Identifier','Outlet_Identifier'],axis=1)
```

Now Checked dataset rows and coloumns

[306]:
```
data.shape
```

[306]:
```
(8523, 10)
```

[307]:
```
data.head()
```

[307]:
```
   Item_Weight Item_Fat_Content  Item_Visibility              Item_Type  \
0         9.30          Low Fat         0.015920                  Dairy
1         5.92          Regular         0.019095            Soft Drinks
2        17.50          Low Fat         0.016621                   Meat
3        19.20          Regular         0.000000  Fruits and Vegetables
4         8.93          Low Fat         0.000000              Household

   Item_MRP  Outlet_Establishment_Year Outlet_Size Outlet_Location_Type  \
0  249.8092                       1999      Medium               Tier 1
1   48.2692                       2009      Medium               Tier 3
```

```
2  141.6180                        1999      Medium               Tier 1
3  182.0950                        1998         NaN               Tier 3
4   53.8614                        1987        High               Tier 3

         Outlet_Type  Item_Outlet_Sales
0  Supermarket Type1           8.225808
1  Supermarket Type2           6.096776
2  Supermarket Type1           7.648868
3      Grocery Store           6.597664
4  Supermarket Type1           6.903451
```

Encode the Qualtative features

[308]:
```python
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
data['Item_Type']= label_encoder.fit_transform(data['Item_Type'])
data['Outlet_Type']= label_encoder.fit_transform(data['Outlet_Type'])
data['Outlet_Location_Type']= label_encoder.
 ↪fit_transform(data['Outlet_Location_Type'])
data['Item_Fat_Content']= label_encoder.fit_transform(data['Item_Fat_Content'])
```

[309]:
```python
data.head()
```

[309]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0         9.30                 1         0.015920          4  249.8092
1         5.92                 2         0.019095         14   48.2692
2        17.50                 1         0.016621         10  141.6180
3        19.20                 2         0.000000          6  182.0950
4         8.93                 1         0.000000          9   53.8614

   Outlet_Establishment_Year Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                       1999      Medium                     0            1
1                       2009      Medium                     2            2
2                       1999      Medium                     0            1
3                       1998         NaN                     2            0
4                       1987        High                     2            1

   Item_Outlet_Sales
0           8.225808
1           6.096776
2           7.648868
3           6.597664
4           6.903451
```

[310]:
```python
data.shape
```

[310]: (8523, 10)

[311]:
```
data.head()
```

[311]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0         9.30                 1         0.015920          4  249.8092
1         5.92                 2         0.019095         14   48.2692
2        17.50                 1         0.016621         10  141.6180
3        19.20                 2         0.000000          6  182.0950
4         8.93                 1         0.000000          9   53.8614

   Outlet_Establishment_Year Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                       1999      Medium                     0            1
1                       2009      Medium                     2            2
2                       1999      Medium                     0            1
3                       1998         NaN                     2            0
4                       1987        High                     2            1

   Item_Outlet_Sales
0           8.225808
1           6.096776
2           7.648868
3           6.597664
4           6.903451
```

[312]:
```
data.shape
```

[312]: (8523, 10)

Standardize the quantatitive features

[313]:
```
from sklearn.preprocessing import StandardScaler
```

[314]:
```
standard_scaler = StandardScaler()
data['Item_Weight'] = standard_scaler.fit_transform(np.
 ↪array(data['Item_Weight']).reshape(-1,1))
data['Item_Visibility'] = standard_scaler.fit_transform(np.
 ↪array(data['Item_Visibility']).reshape(-1,1))
data['Item_Type']=standard_scaler.fit_transform(np.array(data['Item_Type']).
 ↪reshape(-1,1))
data['Item_MRP']=standard_scaler.fit_transform(np.array(data['Item_MRP']).
 ↪reshape(-1,1))
data['Outlet_Establishment_Year']=standard_scaler.fit_transform(np.
 ↪array(data['Outlet_Establishment_Year']).reshape(-1,1))
data['Item_Outlet_Sales']=standard_scaler.fit_transform(np.
 ↪array(data['Item_Outlet_Sales']).reshape(-1,1))
data.head()
```

[314]:

```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0    -0.841872                 1        -1.022884  -0.766479  1.747454
1    -1.641706                 2        -0.952936   1.608963 -1.489023
2     1.098554                 1        -1.007433   0.658786  0.010040
3     1.500838                 2        -1.373631  -0.291391  0.660050
4    -0.929428                 1        -1.373631   0.421242 -1.399220

   Outlet_Establishment_Year Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                   0.139541      Medium                     0            1
1                   1.334103      Medium                     2            2
2                   0.139541      Medium                     0            1
3                   0.020085         NaN                     2            0
4                  -1.293934        High                     2            1

   Item_Outlet_Sales
0           0.929362
1          -1.223928
2           0.345849
3          -0.717333
4          -0.408062
```

##Now we used KNN Classifer for non null values of dataset

[315]:

```
not_null_data=data[data['Outlet_Size'].notnull()].iloc[:,:]
```

[316]:

```
not_null_data.head()
```

[316]:

```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0    -0.841872                 1        -1.022884  -0.766479  1.747454
1    -1.641706                 2        -0.952936   1.608963 -1.489023
2     1.098554                 1        -1.007433   0.658786  0.010040
4    -0.929428                 1        -1.373631   0.421242 -1.399220
5    -0.582754                 2        -1.373631  -1.716656 -1.438734

   Outlet_Establishment_Year Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                   0.139541      Medium                     0            1
1                   1.334103      Medium                     2            2
2                   0.139541      Medium                     0            1
4                  -1.293934        High                     2            1
5                   1.334103      Medium                     2            2

   Item_Outlet_Sales
0           0.929362
1          -1.223928
2           0.345849
4          -0.408062
5          -0.994462
```

[317]:
```
not_null_data.shape
```

[317]: (6113, 10)

[318]:
```
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
not_null_data['Outlet_Size']= label_encoder.
 ↪fit_transform(not_null_data['Outlet_Size'])
```

[319]:
```
not_null_data.head()
```

[319]:
```
   Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
0    -0.841872                 1        -1.022884  -0.766479  1.747454
1    -1.641706                 2        -0.952936   1.608963 -1.489023
2     1.098554                 1        -1.007433   0.658786  0.010040
4    -0.929428                 1        -1.373631   0.421242 -1.399220
5    -0.582754                 2        -1.373631  -1.716656 -1.438734

   Outlet_Establishment_Year  Outlet_Size  Outlet_Location_Type  Outlet_Type  \
0                   0.139541            1                     0            1
1                   1.334103            1                     2            2
2                   0.139541            1                     0            1
4                  -1.293934            0                     2            1
5                   1.334103            1                     2            2

   Item_Outlet_Sales
0           0.929362
1          -1.223928
2           0.345849
4          -0.408062
5          -0.994462
```

Split the data into features and target variable (outlet_size), which has missing values that we are trying to estimate

[320]:
```
x=not_null_data.drop(['Outlet_Size'],axis=1)
y=not_null_data['Outlet_Size']
```

[321]:
```
x.shape
```

[321]: (6113, 9)

[322]:
```
y.shape
```

[322]: (6113,)

Split the data into train and test sets to evaluate the accuracy of our model.

[323]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪30,random_state=42)
```

[324]:
```
from sklearn.neighbors import KNeighborsClassifier
```

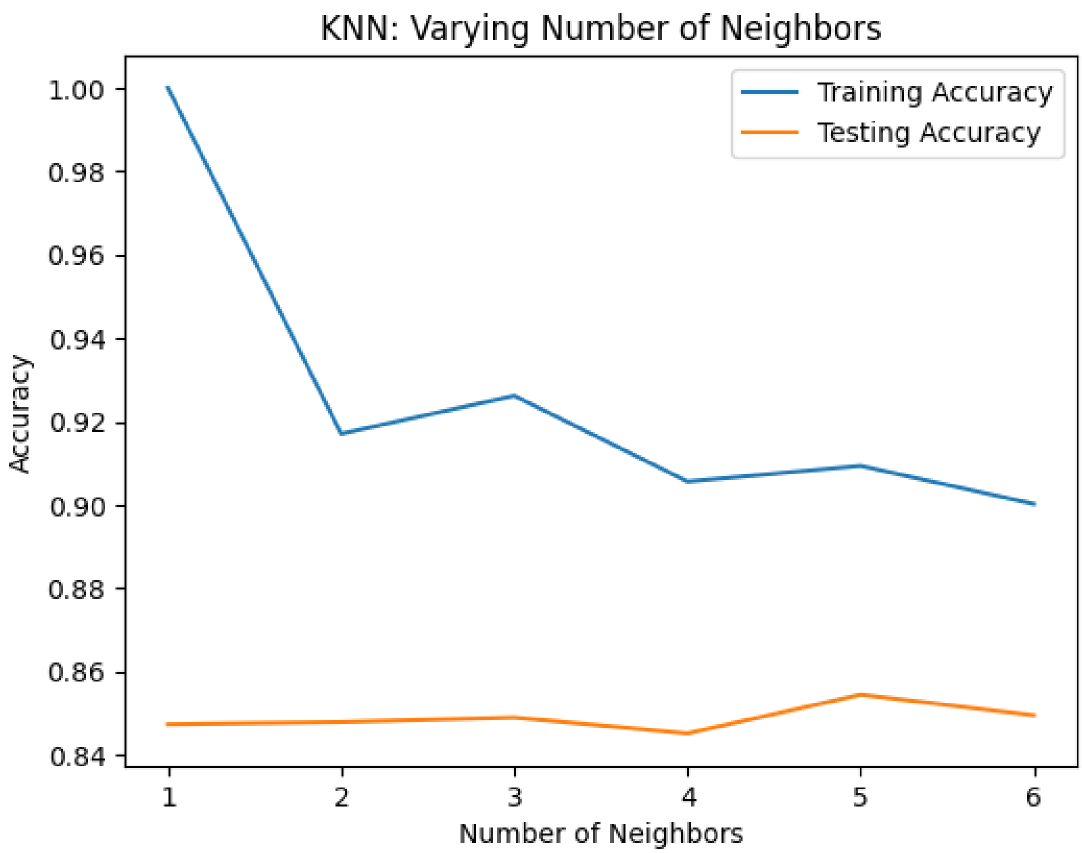
Tune the KNN model by trying different values for the number of neighbors (k) from 1 to 14 to identify the value that produces the best accuracy on the test set.

[275]:
```
# Create neighbors
neighbors = np.arange(1, 14)
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:

        # Set up a KNN Classifier
        knn = KNeighborsClassifier(n_neighbors=neighbor)

        # Fit the model
        knn.fit(x_train,y_train)

        # Compute accuracy
        train_accuracies[neighbor] = knn.score(x_train,y_train)
        test_accuracies[neighbor] = knn.score(x_test,y_test)
print(neighbors, '\n', train_accuracies, '\n', test_accuracies)
```

```
[1 2 3 4 5 6]
 {1: 1.0, 2: 0.9170366908156111, 3: 0.9261509698527693, 4: 0.9055854171535406,
5: 0.9093246085534004, 6: 0.9002103295162421}
 {1: 0.8473282442748091, 2: 0.8478735005452562, 3: 0.8489640130861504, 4:
0.8451472191930207, 5: 0.8544165757906216, 6: 0.8495092693565977}
```

[276]:
```
# Add a title
plt.title("KNN: Varying Number of Neighbors")

# Plot training accuracies
plt.plot(neighbors,train_accuracies.values(), label="Training Accuracy")

# Plot test accuracies
plt.plot(neighbors,test_accuracies.values(), label="Testing Accuracy")

plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")

# Display the plot
```

```
plt.show()
```



The best test accuracy was achieved with a KNN model using 3 neighbors.

[277]:
```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
knn.score(x_test,y_test)
```

[277]:
```
0.8489640130861504
```

An accuracy of 85%, I think the KNN model is a good model for

###Now filled missing values of Out_let_sales with mode #KNN Classifer with non null values

[278]:
```
data.isnull().sum()
```

[278]:
```
Item_Weight                0
Item_Fat_Content           0
Item_Visibility            0
Item_Type                  0
```

```
Item_MRP                        0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

[279]:
```
y=data[data['Outlet_Size'].isnull()].iloc[:,:]
y.drop(['Outlet_Size'],axis=1,inplace=True)
```

[280]:
```
y.shape
```

[280]:
```
(2410, 9)
```

[281]:
```
y.head()
```

[281]:
```
    Item_Weight  Item_Fat_Content  Item_Visibility  Item_Type  Item_MRP  \
3      1.500838                 2        -1.373631  -0.291391  0.660050
8      0.790926                 2        -1.009015  -0.528935 -0.706908
9      1.500838                 2         0.614789  -0.528935  0.752008
25     0.033686                 1         0.723979   0.421242 -1.526973
28    -1.640523                 2         1.924191  -0.766479 -1.533355

    Outlet_Establishment_Year  Outlet_Location_Type  Outlet_Type  \
3                    0.020085                     2            0
8                    0.497909                     1            1
9                    1.095190                     1            1
25                   1.095190                     1            1
28                   0.020085                     2            0

    Item_Outlet_Sales
3           -0.717333
8           -0.328122
9            1.163966
25          -0.580160
28          -2.141229
```

[282]:
```
pred=knn.predict(y)
pred.shape
```

[282]:
```
(2410,)
```

[283]:
```
data['Outlet_Size'].fillna(data['Outlet_Size'].mode(), inplace=True)
print(data.isnull().sum())
```

```
Item_Weight                     0
```

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

[284]:
```
data['Outlet_Size']= label_encoder.fit_transform(data['Outlet_Size'])
# data_mode['Outlet_Size']= label_encoder.
 ↪fit_transform(data_mode['Outlet_Size'])
data['Outlet_Size'].value_counts()
```

[284]:
```
Outlet_Size
1    2793
3    2410
2    2388
0     932
Name: count, dtype: int64
```

[285]:
```
data.isnull().sum()
```

[285]:
```
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

Splitting data into features(x) and target(y).

[286]:
```
x=data.drop('Outlet_Type',axis=1)
y=data['Outlet_Type']
```

Split the data into train and test sets to evaluate the accuracy of our model.

[287]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪30,random_state=0,stratify=y)
```

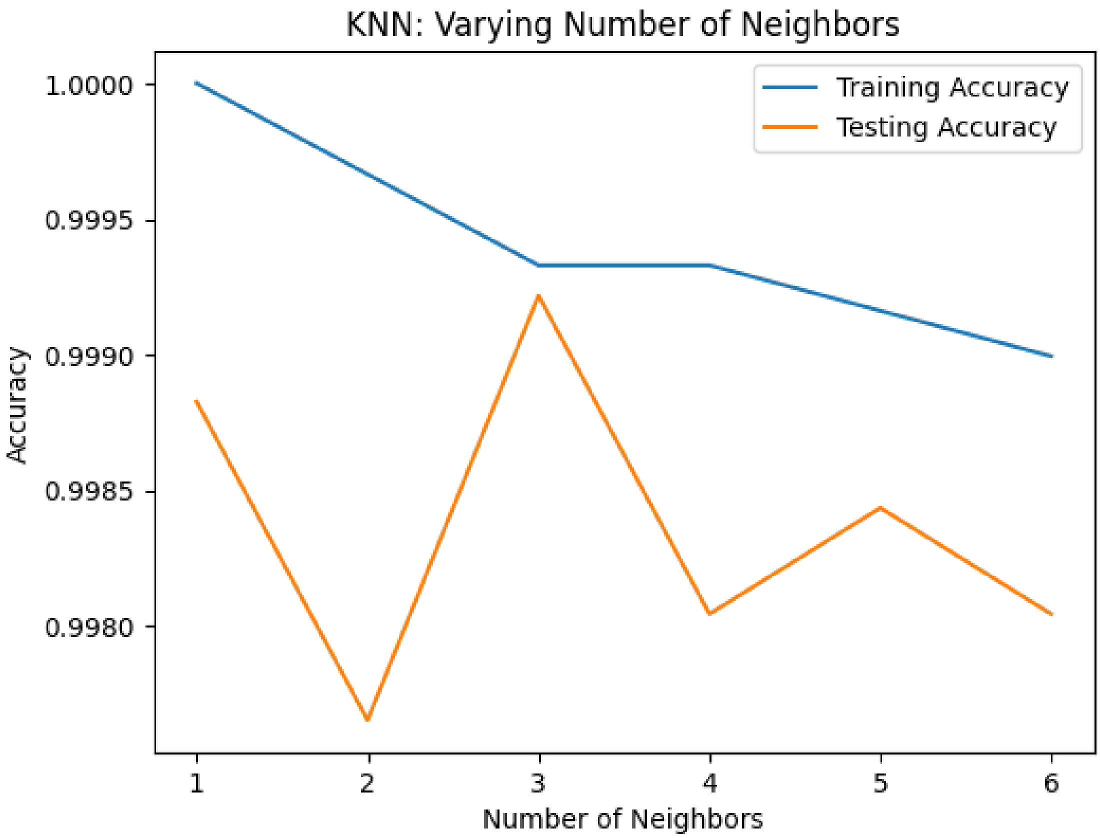
[288]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Tune the KNN model by trying different values for the number of neighbors (k) from 1 to 14 to identify the value that produces the best accuracy on the test set.

[289]:
```
# Create neighbors
neighbors = np.arange(1, 14)
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:

        # Set up a KNN Classifier
        knn = KNeighborsClassifier(n_neighbors=neighbor)

        # Fit the model
        knn.fit(x_train,y_train)

        # Compute accuracy
        train_accuracies[neighbor] = knn.score(x_train,y_train)
        test_accuracies[neighbor] = knn.score(x_test,y_test)
print(neighbors, '\n', train_accuracies, '\n', test_accuracies)
```

```
[1 2 3 4 5 6]
 {1: 1.0, 2: 0.999664767013074, 3: 0.9993295340261482, 4: 0.9993295340261482, 5:
0.9991619175326852, 6: 0.9989943010392223}
 {1: 0.9988267500977708, 2: 0.9976535001955417, 3: 0.9992178333985139, 4:
0.9980445834962847, 5: 0.9984356667970278, 6: 0.9980445834962847}
```

[290]:
```
# Add a title
plt.title("KNN: Varying Number of Neighbors")

# Plot training accuracies
plt.plot(neighbors,train_accuracies.values(), label="Training Accuracy")

# Plot test accuracies
plt.plot(neighbors,test_accuracies.values(), label="Testing Accuracy")

plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")

# Display the plot
plt.show()
```



[291]:
```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
knn.score(x_test,y_test)
```

[291]: 0.9976535001955417

0.4 [4] Conclusion :

- **Our estimator KNN model has an accuracy of 85%, which is reliable.**
- **Our Estimated data Has a better Accuracy more than The data we the mode.**
- **Our KNN model has an accuracy of 99% , which is a very good result.**