In [91]:

```
import pandas as pd
import numpy as np

from scipy.stats import zscore
import statsmodels.api as sm

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.impute import KNNImputer

from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import matplotlib.pyplot as plt
import seaborn as sns

import pickle
import json

import warnings
warnings.filterwarnings('ignore')
```

1. Problem Statement

```
>>> In this project, We are going to predict the sales done by outlets using various feature information.
>>> Initially there are 11 independent features outof which there are 5 categorigal feature and remaining are
continuous features.
>>> Target feature is a continuous data.
>>> Dataset is a collection of item information and outlet information

>>> So we are going to develop a regression model that can predict the sales done by outlet with the best accuracy.
```

2. Data Gathering

In [2]:

```
sales_df = pd.read_csv(r'G:\DataScience\Assignments Solutions\Regression\Sales_Data\Dat
sales_df
```

Out[2]:

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

8523 rows × 12 columns

3. EDA

In [3]:

```
1 sales_df.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
```

```
1 >>> After having a first look our dataset, there are null value in 'Item_Weight' and
  'Outlet_Size' features.
2
3 >>> We also have to change dtype of features from object to either int or float using
  appropriate operations.
```

In [4]:

```
1 sales_df['Item_Fat_Content'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Item_Fat_Content
Non-Null Count  Dtype
--------------  -----
8523 non-null   object
dtypes: object(1)
memory usage: 66.7+ KB
```

In [5]:

```
1 sales_df['Item_Fat_Content'].value_counts()
```

Out[5]:

```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

```
1 >>> 'Item_Fat_Content' feature contain a categorical data. Different terminology is
  use for same category.
```

In [6]:

```
sales_df['Item_Fat_Content'].loc[sales_df['Item_Fat_Content'] == 'LF'] = 'Low Fat'
sales_df['Item_Fat_Content'].loc[sales_df['Item_Fat_Content'] == 'low fat'] = 'Low Fat'
sales_df['Item_Fat_Content'].loc[sales_df['Item_Fat_Content'] == 'reg'] = 'Regular'

sales_df['Item_Fat_Content'].value_counts()
```

Out[6]:

```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

In [7]:

```
sales_df = pd.get_dummies(sales_df, columns = ['Item_Fat_Content'])
sales_df.head()
```

Out[7]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Es
0	FDA15	9.30	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.92	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.20	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	0.000000	Household	53.8614	OUT013	

In [8]:

```
sales_df['Item_Type'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Item_Type
Non-Null Count  Dtype
--------------  -----
8523 non-null   object
dtypes: object(1)
memory usage: 66.7+ KB
```

In [9]:

```
1 sales_df['Item_Type'].value_counts()
```

Out[9]:

```
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: Item_Type, dtype: int64
```

In [10]:

```
1 sales_df['Item_Type'].nunique()
```

Out[10]:

```
16
```

In [11]:

```
1 sales_df = pd.get_dummies(sales_df, columns = ['Item_Type'])
2 sales_df.head()
```

Out[11]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0.016047	249.8092	OUT049	
1	DRC01	5.92	0.019278	48.2692	OUT018	
2	FDN15	17.50	0.016760	141.6180	OUT049	
3	FDX07	19.20	0.000000	182.0950	OUT010	
4	NCD19	8.93	0.000000	53.8614	OUT013	

5 rows × 28 columns

In [12]:

```
sales_df['Outlet_Establishment_Year'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Outlet_Establishment_Year
Non-Null Count  Dtype
--------------  -----
8523 non-null   int64
dtypes: int64(1)
memory usage: 66.7 KB
```

In [13]:

```
sales_df['Outlet_Establishment_Year'].value_counts().sort_index()
```

Out[13]:

```
1985    1463
1987     932
1997     930
1998     555
1999     930
2002     929
2004     930
2007     926
2009     928
Name: Outlet_Establishment_Year, dtype: int64
```

In [14]:

```
sales_df['Outlet_Establishment_Year'].replace({1985 : 0, 1987 : 1, 1997 : 2, 1998 : 3,
                                               2002 : 5, 2004 : 6, 2007 : 7, 2009 : 8},
json_data = {'Outlet_Establishment_Year' : {1985 : 0, 1987 : 1, 1997 : 2, 1998 : 3, 199
                                            2002 : 5, 2004 : 6, 2007 : 7, 2009 : 8}}

sales_df.head()
```

Out[14]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0.016047	249.8092	OUT049	
1	DRC01	5.92	0.019278	48.2692	OUT018	
2	FDN15	17.50	0.016760	141.6180	OUT049	
3	FDX07	19.20	0.000000	182.0950	OUT010	
4	NCD19	8.93	0.000000	53.8614	OUT013	

5 rows × 28 columns

In [15]:

```
1 sales_df['Outlet_Size'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Outlet_Size
Non-Null Count  Dtype
--------------  -----
6113 non-null   object
dtypes: object(1)
memory usage: 66.7+ KB
```

In [16]:

```
1 sales_df['Outlet_Size'].value_counts()
```

Out[16]:

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

In [17]:

```
1 sales_df['Outlet_Size'].replace({'Small' : 0, 'Medium' : 1, 'High' : 2}, inplace = True
2 json_data['Outlet_Size'] = {'Small' : 0, 'Medium' : 1, 'High' : 2}
3
4 sales_df.head()
```

Out[17]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0.016047	249.8092	OUT049	
1	DRC01	5.92	0.019278	48.2692	OUT018	
2	FDN15	17.50	0.016760	141.6180	OUT049	
3	FDX07	19.20	0.000000	182.0950	OUT010	
4	NCD19	8.93	0.000000	53.8614	OUT013	

5 rows × 28 columns

In [18]:

```
1 sales_df['Outlet_Location_Type'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Outlet_Location_Type
Non-Null Count  Dtype
--------------  -----
8523 non-null   object
dtypes: object(1)
memory usage: 66.7+ KB
```

In [19]:

```
1 sales_df['Outlet_Location_Type'].value_counts()
```

Out[19]:

```
Tier 3    3350
Tier 2    2785
Tier 1    2388
Name: Outlet_Location_Type, dtype: int64
```

In [20]:

```
1 sales_df['Outlet_Location_Type'].replace({'Tier 3' : 0, 'Tier 2' : 1, 'Tier 1' : 2}, ir
2 json_data['Outlet_Location_Type'] = {'Tier 3' : 0, 'Tier 2' : 1, 'Tier 1' : 3}
3 sales_df.head()
```

Out[20]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0.016047	249.8092	OUT049	
1	DRC01	5.92	0.019278	48.2692	OUT018	
2	FDN15	17.50	0.016760	141.6180	OUT049	
3	FDX07	19.20	0.000000	182.0950	OUT010	
4	NCD19	8.93	0.000000	53.8614	OUT013	

5 rows × 28 columns

In [21]:

```
1 sales_df['Outlet_Type'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 8523 entries, 0 to 8522
Series name: Outlet_Type
Non-Null Count  Dtype
--------------  -----
8523 non-null   object
dtypes: object(1)
memory usage: 66.7+ KB
```

In [22]:

```
1 sales_df['Outlet_Type'].value_counts()
```

Out[22]:

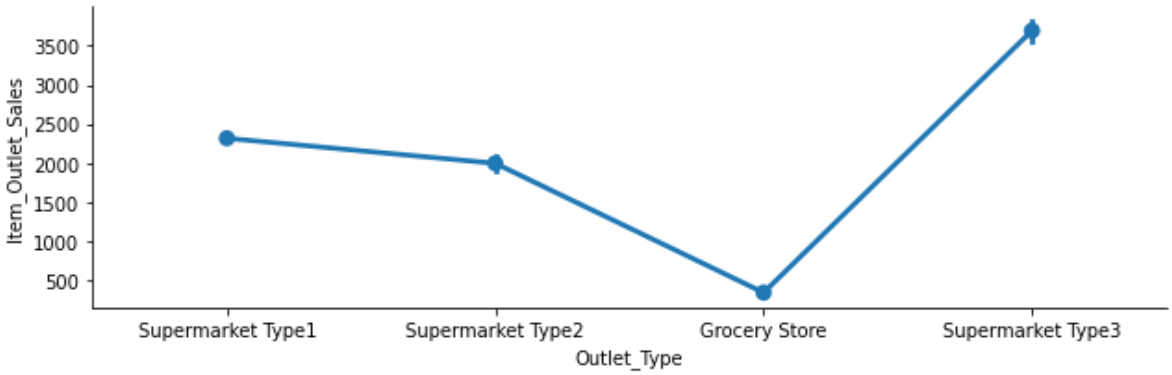
```
Supermarket Type1    5577
Grocery Store        1083
Supermarket Type3     935
Supermarket Type2     928
Name: Outlet_Type, dtype: int64
```

In [23]:

```
1 sns.catplot(x = 'Outlet_Type', y = 'Item_Outlet_Sales', data = sales_df, kind = 'point'
```

Out[23]:

```
<seaborn.axisgrid.FacetGrid at 0x215cb101f60>
```



In [24]:

```
sales_df = pd.get_dummies(sales_df, columns = ['Outlet_Type'])
sales_df.head()
```

Out[24]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0.016047	249.8092	OUT049	
1	DRC01	5.92	0.019278	48.2692	OUT018	
2	FDN15	17.50	0.016760	141.6180	OUT049	
3	FDX07	19.20	0.000000	182.0950	OUT010	
4	NCD19	8.93	0.000000	53.8614	OUT013	

5 rows × 31 columns

In [25]:

```
json_data['Encoded_cols'] = ['Item_Fat_Content_Low Fat', 'Item_Fat_Content_Regular', 'I
                             'Item_Type_Snack Foods', 'Item_Type_Household', 'Item_Type_
                             'Item_Type_Canned', 'Item_Type_Baking Goods', 'Item_Type_He
                             'Item_Type_Soft Drinks', 'Item_Type_Meat', 'Item_Type_Breac
                             'Item_Type_Others', 'Item_Type_Starchy Foods', 'Item_Type_E
                             'Outlet_Type_Grocery Store', 'Outlet_Type_Supermarket Type1
                             'Outlet_Type_Supermarket Type3']
```

4. Feature Selection & Feature Engineering

```
>>> 'Item_Identifier' and 'Outlet_Identifier' Features will not have any impact on
our model so we can drop them out.
```

In [26]:

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

Out[26]:

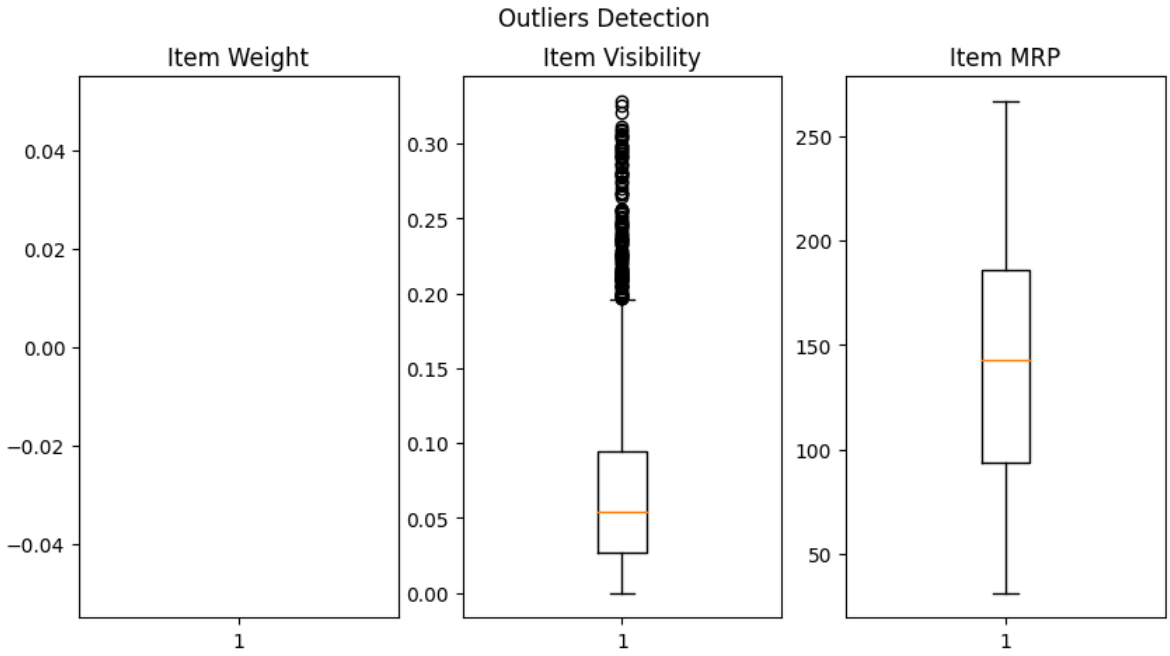
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Locati
0	9.30	0.016047	249.8092	4	1.0	
1	5.92	0.019278	48.2692	8	1.0	
2	17.50	0.016760	141.6180	4	1.0	
3	19.20	0.000000	182.0950	3	NaN	
4	8.93	0.000000	53.8614	1	2.0	

5 rows × 29 columns

Dealing with Outliers

In [27]:

```
cont_col = ['Item_Weight', 'Item_Visibility', 'Item_MRP']

plt.figure(figsize = (10, 5))
plt.subplot(1,3,1)
plt.boxplot(sales_df['Item_Weight'])
plt.title('Item Weight')

plt.subplot(1,3,2)
plt.boxplot(sales_df['Item_Visibility'])
plt.title('Item Visibility')

plt.subplot(1,3,3)
plt.boxplot(sales_df['Item_MRP'])
plt.title('Item MRP')

plt.suptitle('Outliers Detection')
plt.show()
```




```
>>> 'Item_Visibility' feature contains outliers which has to remove using an
appropriate method of imputation.
```

In [28]:

```
z_val = zscore(sales_df['Item_Visibility'])
outlier_index = np.where(abs(z_val) >= 3)[0]
outlier_num = len(outlier_index)
print('Number of Outliers :', outlier_num)
outlier_perc = (outlier_num / len(sales_df['Item_Visibility'])) * 100
print('Outlier Percentage :', outlier_perc)

```

```
Number of Outliers : 95
Outlier Percentage : 1.1146309984747156
```

```
>>> As percentage of outliers present in 'Item_Visibility' feature is very less we
can impute outliers using mean
imputation technique.
```

In [29]:

```
col_mean = sales_df['Item_Visibility'].mean()
print('Mean (Before Dealing with Outliers) :', col_mean)
```

```
Mean (Before Dealing with Outliers) : 0.06613202877895108
```

In [30]:

```
def remove_outlier(col) :

    z_val = zscore(sales_df[col])
    outlier_index = list(np.where(abs(z_val) >= 3)[0])
    mean = sales_df[col].drop(outlier_index).mean()
    sales_df[col].iloc[outlier_index] = mean

remove_outlier('Item_Visibility')
```

In [31]:

```
col_mean = sales_df['Item_Visibility'].mean()
print('Mean (After Dealing with Outliers) :', col_mean)
```

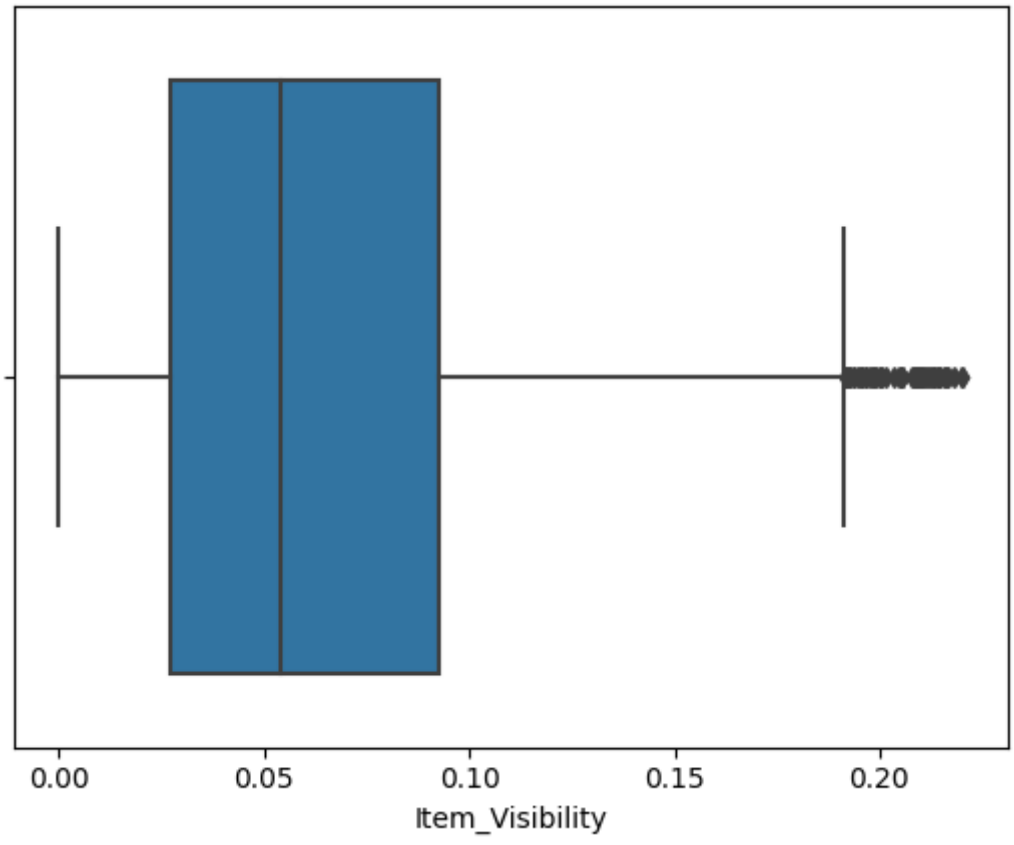
```
Mean (After Dealing with Outliers) : 0.06390474693296157
```

In [32]:

```
sns.boxplot(sales_df['Item_Visibility'])
```

Out[32]:

```
<AxesSubplot:xlabel='Item_Visibility'>
```



Dealing with Missing Values

In [33]:

```
np.around((sales_df['Item_Weight'].isna().mean() * 100), 2)
```

Out[33]:

```
17.17
```

In [34]:

```
np.around((sales_df['Outlet_Size'].isna().mean() * 100), 2)
```

Out[34]:

```
28.28
```

```
>>> As 'Item_Weight' and 'Outlet_Size' contains lot of missing values, it is better to opt for KNNImputer technique to deal with missing values to optain better accuracy.
```

In [35]:

```
impute = KNNImputer()
array = impute.fit_transform(sales_df)
sales_df = pd.DataFrame(array, columns = sales_df.columns)

```

In [36]:

```
np.around((sales_df['Item_Weight'].isna().mean() * 100), 2)
```

Out[36]:

```
0.0
```

In [37]:

```
np.around((sales_df['Outlet_Size'].isna().mean() * 100), 2)
```

Out[37]:

```
0.0
```

In [38]:

```
1 sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Item_Weight                     8523 non-null   float64
 1   Item_Visibility                 8523 non-null   float64
 2   Item_MRP                        8523 non-null   float64
 3   Outlet_Establishment_Year       8523 non-null   float64
 4   Outlet_Size                     8523 non-null   float64
 5   Outlet_Location_Type            8523 non-null   float64
 6   Item_Outlet_Sales               8523 non-null   float64
 7   Item_Fat_Content_Low Fat        8523 non-null   float64
 8   Item_Fat_Content_Regular        8523 non-null   float64
 9   Item_Type_Baking Goods          8523 non-null   float64
 10  Item_Type_Breads                8523 non-null   float64
 11  Item_Type_Breakfast             8523 non-null   float64
 12  Item_Type_Canned                8523 non-null   float64
 13  Item_Type_Dairy                 8523 non-null   float64
 14  Item_Type_Frozen Foods          8523 non-null   float64
 15  Item_Type_Fruits and Vegetables 8523 non-null   float64
 16  Item_Type_Hard Drinks           8523 non-null   float64
 17  Item_Type_Health and Hygiene    8523 non-null   float64
 18  Item_Type_Household             8523 non-null   float64
 19  Item_Type_Meat                  8523 non-null   float64
 20  Item_Type_Others                8523 non-null   float64
 21  Item_Type_Seafood               8523 non-null   float64
 22  Item_Type_Snack Foods           8523 non-null   float64
 23  Item_Type_Soft Drinks           8523 non-null   float64
 24  Item_Type_Starchy Foods         8523 non-null   float64
 25  Outlet_Type_Grocery Store       8523 non-null   float64
 26  Outlet_Type_Supermarket Type1   8523 non-null   float64
 27  Outlet_Type_Supermarket Type2   8523 non-null   float64
 28  Outlet_Type_Supermarket Type3   8523 non-null   float64
dtypes: float64(29)
memory usage: 1.9 MB
```

In [39]:

```
1 sales_df.head()
```

Out[39]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Locati
0	9.30	0.016047	249.8092	4.0	1.0	
1	5.92	0.019278	48.2692	8.0	1.0	
2	17.50	0.016760	141.6180	4.0	1.0	
3	19.20	0.000000	182.0950	3.0	0.4	
4	8.93	0.000000	53.8614	1.0	2.0	

5 rows × 29 columns

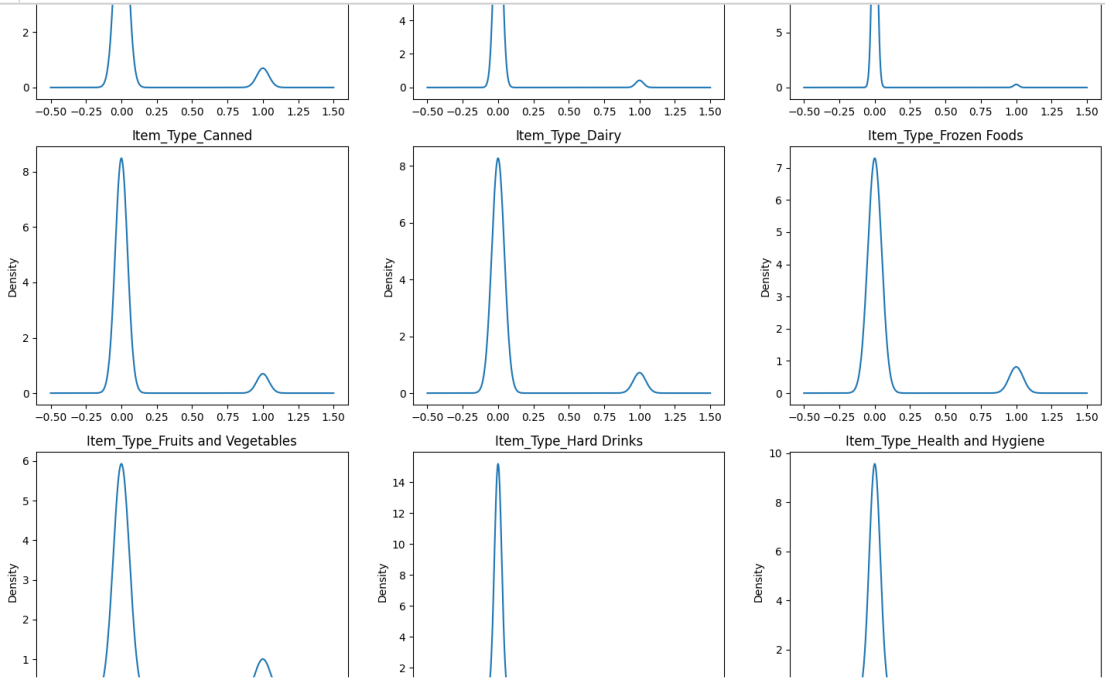
In [40]:

```
sales_df = sales_df.astype({'Outlet_Establishment_Year' : int, 'Outlet_Size' : int, 'Ou
for col in json_data['Encoded_cols'] :
    sales_df = sales_df.astype({col : int})
sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Item_Weight                     8523 non-null   float64
 1   Item_Visibility                 8523 non-null   float64
 2   Item_MRP                        8523 non-null   float64
 3   Outlet_Establishment_Year       8523 non-null   int32
 4   Outlet_Size                     8523 non-null   int32
 5   Outlet_Location_Type            8523 non-null   int32
 6   Item_Outlet_Sales               8523 non-null   float64
 7   Item_Fat_Content_Low Fat        8523 non-null   int32
 8   Item_Fat_Content_Regular        8523 non-null   int32
 9   Item_Type_Baking Goods          8523 non-null   int32
 10  Item_Type_Breads                8523 non-null   int32
 11  Item_Type_Breakfast             8523 non-null   int32
 12  Item_Type_Canned                8523 non-null   int32
 13  Item_Type_Dairy                 8523 non-null   int32
 14  Item_Type_Frozen Foods          8523 non-null   int32
 15  Item_Type_Fruits and Vegetables 8523 non-null   int32
 16  Item_Type_Hard Drinks           8523 non-null   int32
 17  Item_Type_Health and Hygiene    8523 non-null   int32
 18  Item_Type_Household             8523 non-null   int32
 19  Item_Type_Meat                  8523 non-null   int32
 20  Item_Type_Others                8523 non-null   int32
 21  Item_Type_Seafood               8523 non-null   int32
 22  Item_Type_Snack Foods           8523 non-null   int32
 23  Item_Type_Soft Drinks           8523 non-null   int32
 24  Item_Type_Starchy Foods         8523 non-null   int32
 25  Outlet_Type_Grocery Store       8523 non-null   int32
 26  Outlet_Type_Supermarket Type1   8523 non-null   int32
 27  Outlet_Type_Supermarket Type2   8523 non-null   int32
 28  Outlet_Type_Supermarket Type3   8523 non-null   int32
dtypes: float64(4), int32(25)
memory usage: 1.1 MB
```

In [41]:

```
plt.figure(figsize = (15,40))
for i in range(0, 29) :
    col = sales_df.columns[i]
    plt.subplot(10, 3, i+1)
    sales_df[col].plot(kind = 'kde')
    plt.title(col)

plt.suptitle('Normality of Data')
plt.tight_layout()
plt.show()
```



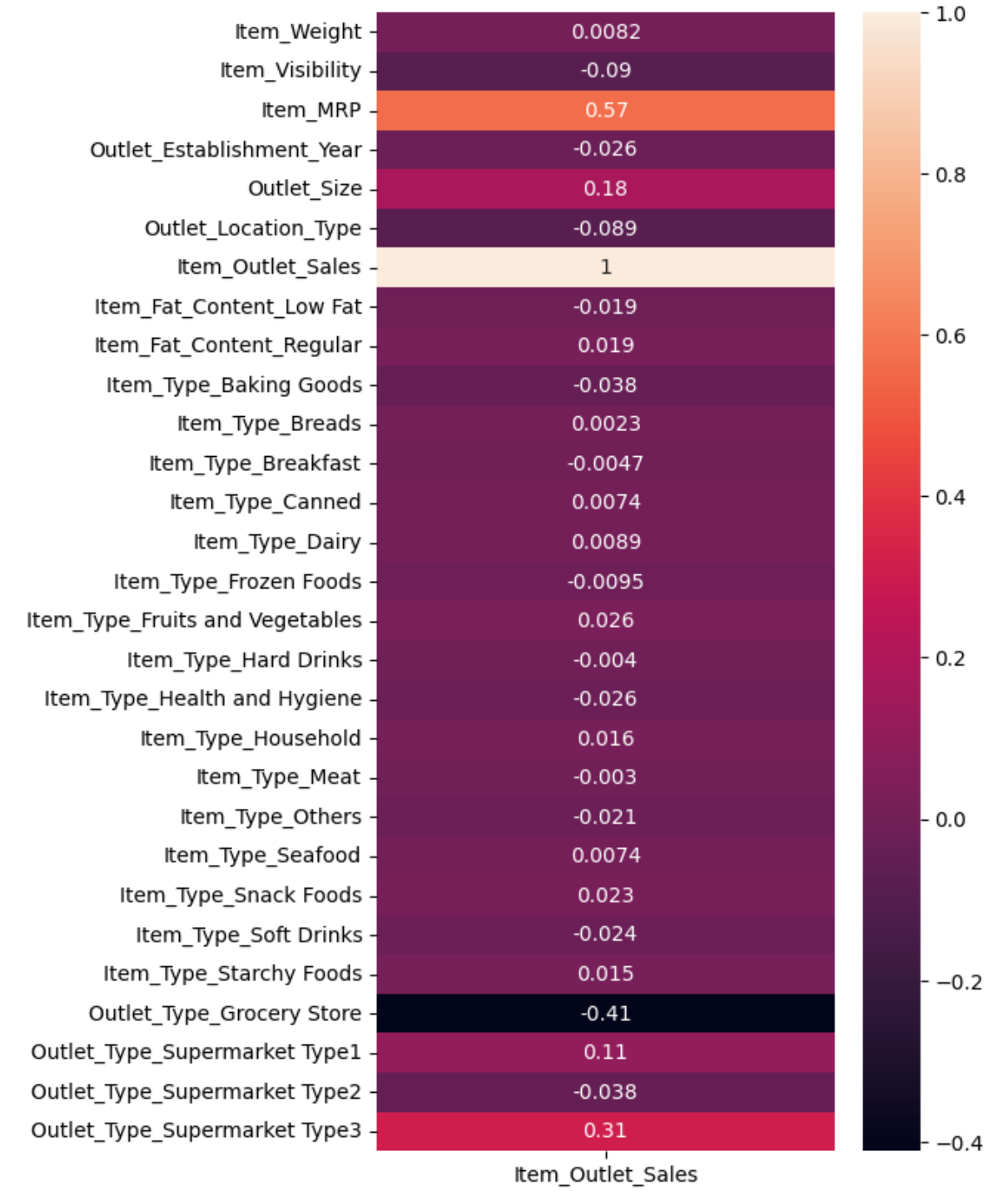
In [42]:

```
plt.figure(figsize = (5, 10))
relation = sales_df.corr().loc[['Item_Outlet_Sales'],:]
sns.heatmap(sales_df.corr().loc[:,['Item_Outlet_Sales']], annot = True)
relation
```

Out[42]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Siz
Item_Outlet_Sales	0.008164	-0.0903	0.567574	-0.025525	0.17693

1 rows × 29 columns



5. Model Training & Evaluation

Data Splitting

In [43]:

```
x = sales_df.drop('Item_Outlet_Sales', axis = 1)
y = sales_df['Item_Outlet_Sales']
y.head()
```

Out[43]:

```
0    3735.1380
1     443.4228
2    2097.2700
3     732.3800
4     994.7052
Name: Item_Outlet_Sales, dtype: float64
```

Scaling (Normalization)

In [44]:

```
normal_data = MinMaxScaler()
data = normal_data.fit_transform(x)
x_normal = pd.DataFrame(data, columns = x.columns)
x_normal.head()
```

Out[44]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Loc
0	0.282525	0.072868	0.927507	0.500	0.5	
1	0.081274	0.087538	0.072068	1.000	0.5	
2	0.770765	0.076104	0.468288	0.500	0.5	
3	0.871986	0.000000	0.640093	0.375	0.0	
4	0.260494	0.000000	0.095805	0.125	1.0	

5 rows × 28 columns

Scaling(Standarization)

In [45]:

```
std_data = StandardScaler()
data = std_data.fit_transform(x)
x_std = pd.DataFrame(data, columns = x.columns)
x_std.head()
```

Out[45]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Locati
0	-0.821178	-1.017488	1.747454	0.076668	0.555221	
1	-1.604314	-0.948796	-1.489023	1.541783	0.555221	
2	1.078738	-1.002334	0.010040	0.076668	0.555221	
3	1.472623	-1.358667	0.660050	-0.289611	-0.931007	
4	-0.906906	-1.358667	-1.399220	-1.022168	2.041449	

5 rows × 28 columns

Train test Split

In [46]:

```
x_train, x_test, y_train, y_test  = train_test_split(x, y, test_size = 0.25, random_sta
y_train.head()
```

Out[46]:

```
2699    3621.9520
7890    2554.0088
3168    2885.5772
3408      67.9116
7632      41.9454
Name: Item_Outlet_Sales, dtype: float64
```

Train test Split(normalisation)

In [47]:

```
x_normal_train, x_normal_test, y_train, y_test  = train_test_split(x_normal, y, test_si
x_normal_train.head()
```

Out[47]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Lo
2699	0.288181	0.722904	0.826754	0.125	1.0	
7890	0.240845	0.315147	0.254351	0.750	0.0	
3168	0.413516	0.080110	0.426882	0.250	0.0	
3408	0.874963	0.821071	0.019804	0.375	0.0	
7632	0.562370	0.184926	0.051594	0.375	0.0	

5 rows × 28 columns

Train test Split (Standadiation)

In [48]:

```
x_std_train, x_std_test, y_train, y_test  = train_test_split(x_std, y, test_size = 0.25
x_std_train.head()
```

Out[48]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Lo
2699	-0.799167	2.026095	1.366264	-1.022168	2.041449	
7890	-0.983366	0.116905	-0.799375	0.809225	-0.931007	
3168	-0.311444	-0.983579	-0.146619	-0.655889	-0.931007	
3408	1.484208	2.485731	-1.686761	-0.289611	-0.931007	
7632	0.267798	-0.492812	-1.566487	-0.289611	-0.931007	

5 rows × 28 columns

List of Result

In [49]:

```
model_name = []
train_accuracy = []
test_accuracy = []
bias = []
variance = []
remark = []
```

5.1 Linear Regression

In [50]:

```
linear_model = LinearRegression()
linear_model.fit(x_train, y_train)
```

Out[50]:

```
▾ LinearRegression
LinearRegression()
```

Testing Accuracy

In [51]:

```
y_pred = linear_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('LinearRegression')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1352410.95790206
Root Mean Squared Error : 1162.9320521432282
Mean Absolute Error : 854.5292663472214
Accuracy : 0.5427823401646741
```

Training Evaluation

In [52]:

```
y_pred = linear_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1246057.92663186
Root Mean Squared Error : 1116.2696478144785
Mean Absolute Error : 831.8195121792752
Accuracy : 0.5697049116150893
```

5.1.1 With Normatisation

In [53]:

```
linear_normal_model = LinearRegression()
linear_normal_model.fit(x_normal_train, y_train)
```

Out[53]:

```
▾ LinearRegression
LinearRegression()
```

Testing Evaluation

In [54]:

```
y_pred = linear_normal_model.predict(x_normal_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('LinearRegression_Normalization')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1352420.7520236364
Root Mean Squared Error : 1162.9362630959774
Mean Absolute Error : 854.5687185358986
Accuracy : 0.5427790290074248
```

Training Evaluation

In [55]:

```
y_pred = linear_normal_model.predict(x_normal_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1246059.2697157748
Root Mean Squared Error : 1116.2702494090645
Mean Absolute Error : 831.8482652065084
Accuracy : 0.569704447814491
```

5.1.2 With Standardisation

In [56]:

```
linear_std_model = LinearRegression()
linear_std_model.fit(x_std_train, y_train)
```

Out[56]:

LinearRegression

LinearRegression()

Testing Evaluation

In [57]:

```
y_pred = linear_std_model.predict(x_std_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('LinearRegression_Standardisation')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1351557.2239350867
Root Mean Squared Error : 1162.564933212372
Mean Absolute Error : 854.4452610839915
Accuracy : 0.543070967112142
```

Training Evalution

In [58]:

```
y_pred = linear_std_model.predict(x_std_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1246244.2946949129
Root Mean Squared Error : 1116.3531227595115
Mean Absolute Error : 831.9800436579659
Accuracy : 0.5696405540435435
```

5.2 KNeighborsRegressor

5.2.1 Hyperparameter Tuning with Normalization

In [59]:

```
knn_model = KNeighborsRegressor()

parameter = {'n_neighbors' : np.arange(3,31,2), 'p' : [1, 2]}

gscv_knn = GridSearchCV(knn_model, parameter, cv = 5)
gscv_knn.fit(x_normal_train, y_train)
knn_normal_model = gscv_knn.best_estimator_
knn_normal_model
```

Out[59]:

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=9)
```

testing Evaluation

In [60]:

```
y_pred = knn_normal_model.predict(x_normal_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('KNeighborsRegressor_Normalization')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1576914.4488190177
Root Mean Squared Error : 1255.7525428280119
Mean Absolute Error : 897.9387257312686
Accuracy : 0.466883102479448
```

Training Evaluation

In [61]:

```
y_pred = knn_normal_model.predict(x_normal_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1135646.196749953
Root Mean Squared Error : 1065.6670196407285
Mean Absolute Error : 769.4671825789183
Accuracy : 0.6078328541872988
```

5.2.2 Hyperparameter tuning with standardisation

In [62]:

```
knn_model = KNeighborsRegressor()

parameter = {'n_neighbors' : np.arange(3,31,2), 'p' : [1, 2]}

gscv_knn = GridSearchCV(knn_model, parameter, cv = 5)
gscv_knn.fit(x_std_train, y_train)
knn_std_model = gscv_knn.best_estimator_
knn_std_model
```

Out[62]:

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=7)
```

Testing Evaluation

In [63]:

```
y_pred = knn_std_model.predict(x_std_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('KNeighborsRegressor_Standardisation')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1579850.792789063
Root Mean Squared Error : 1256.9211561546185
Mean Absolute Error : 898.764343916337
Accuracy : 0.46589039511441865
```

Training Evaluation

In [64]:

```
y_pred = knn_std_model.predict(x_std_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1081594.9402968064
Root Mean Squared Error : 1039.997567447543
Mean Absolute Error : 748.381520311103
Accuracy : 0.6264981101723792
```

5.3 DecisionTreeRegressor

5.3.1 Hyperparameter

In [65]:

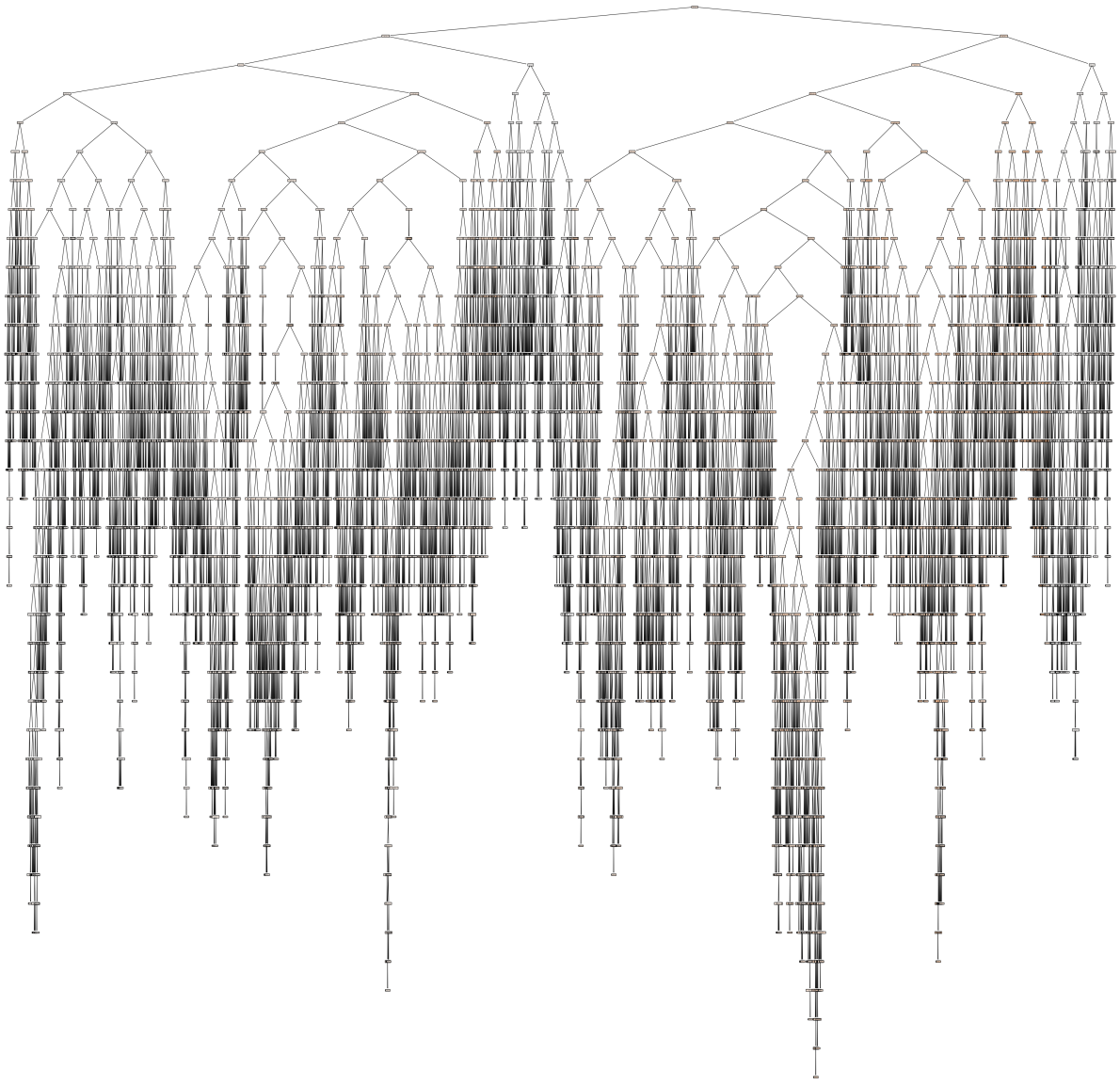
```
dt_model = DecisionTreeRegressor(random_state = 7)
dt_model.fit(x_train, y_train)
```

Out[65]:

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=7)
```

In [66]:

```
plt.figure(figsize = (50, 50 ))
plot_tree(dt_model, feature_names = x.columns, filled = True)
plt.savefig(r'G:\DataScience\Assignments Solutions\Regression\Sales_Data\Statics\Tree_V
```



In [67]:

```
dt_hyper_model = DecisionTreeRegressor(random_state = 7)

parameter = {'criterion' : ['mse', 'mae'],
             'max_depth' : np.arange(15, 28),
             'min_samples_split' : np.arange(10, 25),
             'min_samples_leaf' : np.arange(5,10)}

rscv_dt_model = RandomizedSearchCV(dt_hyper_model, parameter, cv = 7)
rscv_dt_model.fit(x_train, y_train)
dt_hyper_model = rscv_dt_model.best_estimator_
dt_hyper_model

```

Out[67]:

```
DecisionTreeRegressor(criterion='mae', max_depth=22, min_samples_leaf=9,
                      min_samples_split=11, random_state=7)
```

Testing Evaluation

In [68]:

```
y_pred = dt_hyper_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('DecisionTreeRegressor_Hyperparameter')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1560988.324190253
Root Mean Squared Error : 1249.3951833548315
Mean Absolute Error : 856.2158318629752
Accuracy : 0.47226734266950476
```

Training Evaluation

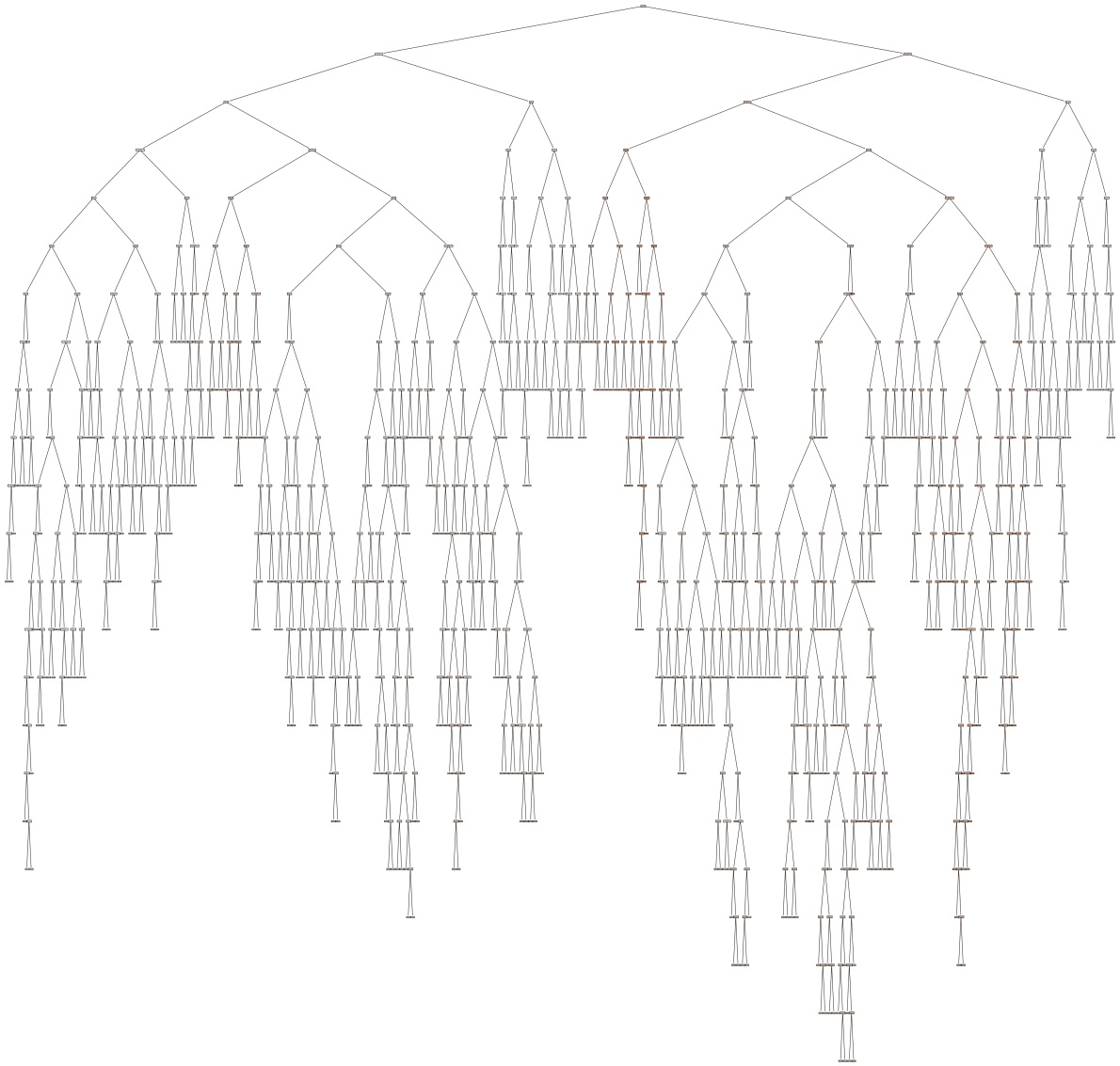
In [69]:

```
y_pred = dt_hyper_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 888815.3591970996
Root Mean Squared Error : 942.7700457678424
Mean Absolute Error : 588.2322167709636
Accuracy : 0.6930697398817034
```

In [70]:

```
plt.figure(figsize = (50, 50 ))
plot_tree(dt_hyper_model, feature_names = x.columns, filled = True)
plt.savefig(r'G:\DataScience\Assignments Solutions\Regression\Sales_Data\Statics\Tree_
```



5.3.2 Pruning

In [71]:

```
train_list = []
test_list = []
ccp_alpha_val = dt_model.cost_complexity_pruning_path(x_train, y_train)['ccp_alphas']

for ccp_alpha in ccp_alpha_val:
    dt_reg_model = DecisionTreeRegressor(random_state=10,ccp_alpha=ccp_alpha)
    dt_reg_model.fit(x_train, y_train)
    train_list.append(dt_reg_model.score(x_train, y_train))
    test_list.append(dt_reg_model.score(x_test, y_test))

index = np.where(test_list == max(test_list))
best_ccp = ccp_alpha_val[index][0]
best_ccp
```

Out[71]:

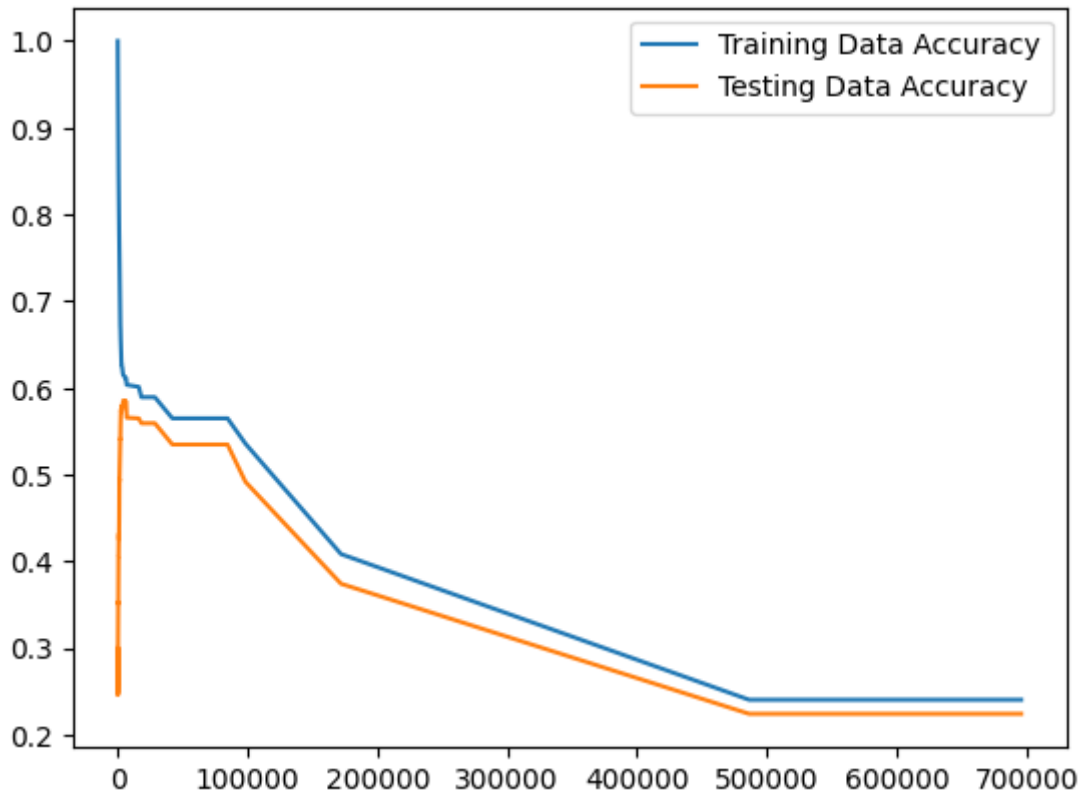
```
4290.160941621916
```

In [72]:

```
fig,ax = plt.subplots()
ax.plot(ccp_alpha_val, train_list, label = "Training Data Accuracy")
ax.plot(ccp_alpha_val, test_list, label = "Testing Data Accuracy")
ax.legend()
```

Out[72]:

```
<matplotlib.legend.Legend at 0x215cfd06920>
```



In [73]:

```
dt_prun_model = DecisionTreeRegressor(ccp_alpha = best_ccp, random_state = 7)
dt_prun_model.fit(x_train, y_train)
```

Out[73]:

```
DecisionTreeRegressor
DecisionTreeRegressor(ccp_alpha=4290.160941621916, random_state=7)
```

Testing Evaluation

In [74]:

```
y_pred = dt_prun_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('DecisionTreeRegressor_Pruning')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1226266.0723095413
Root Mean Squared Error : 1107.3689865214492
Mean Absolute Error : 778.4519685134645
Accuracy : 0.5854288959721456
```

Training Evaluation

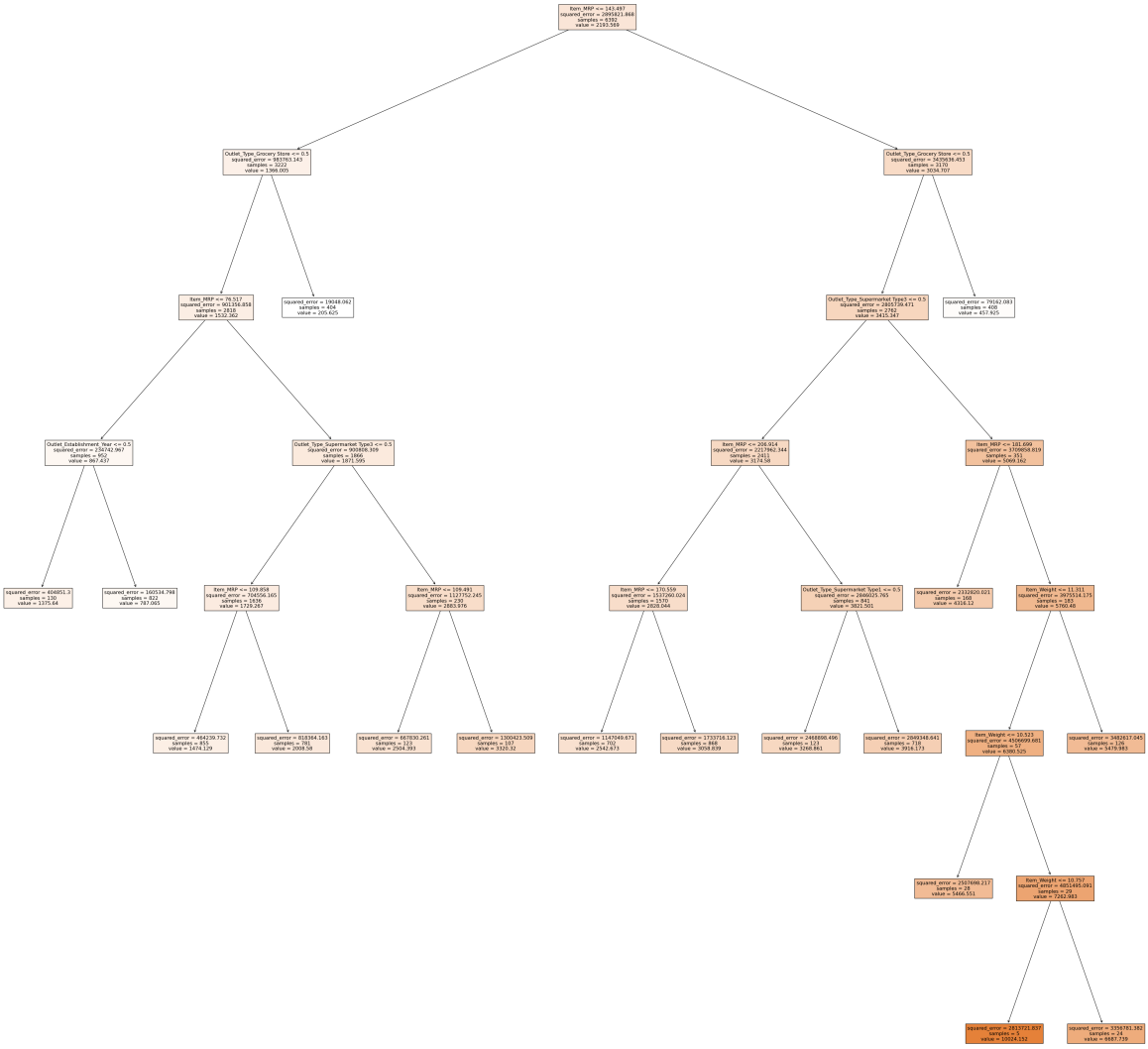
In [75]:

```
y_pred = dt_prun_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1116569.3628017846
Root Mean Squared Error : 1056.6784576217046
Mean Absolute Error : 752.0777300095195
Accuracy : 0.6144205639352877
```

In [76]:

```
plt.figure(figsize = (50, 50 ))
plot_tree(dt_prun_model, feature_names = x.columns, filled = True)
plt.savefig(r'G:\DataScience\Assignments Solutions\Regression\Sales_Data\Statics\Tree_V
```



5.4 Random Forest

In [77]:

```
rf_model= RandomForestRegressor()
rf_model.fit(x_train, y_train)
```

Out[77]:

RandomForestRegressor

RandomForestRegressor()

Testing Evaluation

In [78]:

```
y_pred = rf_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('RandomForestRegressor')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1320890.1520200134
Root Mean Squared Error : 1149.2998529626693
Mean Absolute Error : 799.6950138310652
Accuracy : 0.5534387675008363
```

Training Evaluation

In [79]:

```
y_pred = rf_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 174584.05958012847
Root Mean Squared Error : 417.83257362265147
Mean Absolute Error : 291.321537879224
Accuracy : 0.9397117407288703
```

5.4.1 Hyperparameter

In [80]:

```
rf_model= RandomForestRegressor()

parameters = {"n_estimators" :np.arange(10,100),
              "criterion" : ['mse','mae'],
              "max_depth" : np.arange(10, 28),
              "min_samples_split" : np.arange(10,25),
              "min_samples_leaf" : np.arange(5,10)}

rscv_rf_model = RandomizedSearchCV(rf_model,parameters,cv = 5)
rscv_rf_model.fit(x_train, y_train)
rf_hyper_model = rscv_rf_model.best_estimator_
rf_hyper_model
```

Out[80]:

```
RandomForestRegressor(criterion='mae', max_depth=12, min_samples_leaf=7,
                      min_samples_split=24, n_estimators=93)
```

Testing Evaluation

In [81]:

```
y_pred = rf_hyper_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('RandomForestRegressor_Hyperparameter')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1257682.403050851
Root Mean Squared Error : 1121.4644011518383
Mean Absolute Error : 770.6452410020031
Accuracy : 0.5748077891715643
```

Training Evaluation

In [82]:

```
y_pred = rf_hyper_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 908462.2315044
Root Mean Squared Error : 953.1328509207938
Mean Absolute Error : 647.1754222484086
Accuracy : 0.6862851815756477
```

5.5 AdaBoost

In [83]:

```
adb_model = AdaBoostRegressor( random_state = 7)
adb_model.fit(x_train, y_train)
```

Out[83]:

```
AdaBoostRegressor(random_state=7)
```

Testing Evaluation

In [84]:

```
y_pred = adb_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('AdaBoostRegressor')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1507516.9951446354
Root Mean Squared Error : 1227.8098367192842
Mean Absolute Error : 946.102198272143
Accuracy : 0.49034471463375384
```

Training Evaluation

In [85]:

```
y_pred = adb_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1348199.807961966
Root Mean Squared Error : 1161.1200661266541
Mean Absolute Error : 907.3501308667953
Accuracy : 0.5344327553891419
```

5.5.1 Hyperparameter

In [86]:

```
parameters = {"n_estimators" : np.arange(10,100),
              "learning_rate" : np.arange(0,2,0.001)}

rscv_adb_model = RandomizedSearchCV(adb_model,parameters,cv= 3)
rscv_adb_model.fit(x_train, y_train)
adb_hyper_model = rscv_adb_model.best_estimator_
adb_hyper_model
```

Out[86]:

```
AdaBoostRegressor(learning_rate=0.125, n_estimators=10, random_state=7)
```

Testing Evaluation

In [87]:

```
y_pred = adb_hyper_model.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_test, y_pred)
print('Mean Absolute Error :', mae)
r2_val_1 = r2_score(y_test, y_pred)
print('Accuracy :', r2_val_1)

model_name.append('AdaBoostRegressor_Hyperparameter')
test_accuracy.append(np.around(r2_val_1, 6))
```

```
Mean Squared Error : 1485073.3968972499
Root Mean Squared Error : 1218.6358754350085
Mean Absolute Error : 861.8529243395288
Accuracy : 0.49793235610397113
```

Training Evaluation

In [88]:

```
y_pred = adb_hyper_model.predict(x_train)

mse = mean_squared_error(y_train, y_pred)
print('Mean Squared Error :', mse)
rmse = np.sqrt(mse)
print('Root Mean Squared Error :', rmse)
mae = mean_absolute_error(y_train, y_pred)
print('Mean Absolute Error :', mae)
r2_val_2 = r2_score(y_train, y_pred)
print('Accuracy :', r2_val_2)

train_accuracy.append(np.around(r2_val_2, 6))
bias.append(1 - r2_val_2)
variance.append(r2_val_2 - r2_val_1)
```

```
Mean Squared Error : 1290044.448125114
Root Mean Squared Error : 1135.8012361875267
Mean Absolute Error : 817.8302291990555
Accuracy : 0.5545152613194202
```

6. Conclusion

Result of Models

In [89]:

```
result = pd.DataFrame(list(zip(model_name, test_accuracy, train_accuracy, bias, varianc
                       columns = ['Model_name', 'test_accuracy', 'train_accuracy', 'bias
result = result.sort_values(by = 'test_accuracy', ascending = False)
result
```

Out[89]:

	Model_name	test_accuracy	train_accuracy	bias	variance
6	DecisionTreeRegressor_Pruning	0.585429	0.614421	0.385579	0.028992
8	RandomForestRegressor_Hyperparameter	0.574808	0.686285	0.313715	0.111477
7	RandomForestRegressor	0.553439	0.939712	0.060288	0.386273
2	LinearRegression_Standardisation	0.543071	0.569641	0.430359	0.026570
0	LinearRegression	0.542782	0.542782	0.430295	0.026923
1	LinearRegression_Normalization	0.542779	0.569704	0.430296	0.026925
10	AdaBoostRegressor_Hyperparameter	0.497932	0.554515	0.445485	0.056583
9	AdaBoostRegressor	0.490345	0.534433	0.465567	0.044088
5	DecisionTreeRegressor_Hyperparameter	0.472267	0.693070	0.306930	0.220802
3	KNeighborsRegressor_Normalization	0.466883	0.607833	0.392167	0.140950
4	KNeighborsRegressor_Standardisation	0.465890	0.626498	0.373502	0.160608

```
>>>> After comparing result we can see that "DecisionTreeRegressor with Pruing"
giving maximum accuracy on testing data.
>>>> Thus, we will select "DecisionTreeRegressor with Pruing" as our best model tp
predict the result.
```

Best Model

In [90]:

```
dt_prun_model
```

Out[90]:

```
DecisionTreeRegressor
DecisionTreeRegressor(ccp_alpha=4290.160941621916, random_state=7)
```

In [96]:

```
with open('clf_model.pkl', 'wb') as f:
    pickle.dump(dt_prun_model, f)

json_data['Columns'] = list(x.columns)

with open('json_data.json', 'w') as f:
    json.dump(json_data, f)
```