SALES PREDICTOR FOR SUPERMARKET

By-
Perumalla Dharan -AP21110010201
Pavan Sastry NVSS -AP21110010209
Phalgun Vatala -AP21110010223
Grandhi Dinesh -AP21110010240
Gurram V S Rehan -AP21110010259

TABLE OF CONTENTS

1. Abstract
2. Motivation
3. Goal
4. Methodology in detail
5. Result/outcome

Abstract

- This project revolves around building a predictive model for Supermarket using sales data collected in 2013.
- Retail businesses need to predict and improve their sales to do well. In this project, we're looking at sales data from Supermarket. They have data about 1559 products in 10 stores in different cities.
- The goal is to predict the sales of each product in a particular store, which will enable Supermarket to understand the factors influencing sales and optimize their strategies accordingly.
- Supermarket wants to use this model to figure out what things about products and stores help sales the most. This will help them make better plans to make more money.

Motivation

- The retail sector faces increasing pressure to adapt to changing consumer behaviors and market dynamics. Previous studies have demonstrated the efficacy of predictive modeling in driving business outcomes such as inventory optimization, demand forecasting, and personalized marketing.
- By harnessing the power of machine learning and analytics, companies like Super Markets can unlock hidden insights within their data, enabling more informed decision-making and strategic planning.
- This project aims to help Super Market by using data analysis to provide useful information. This can help the company stay competitive.

Goal

- Our primary goal of this project is to develop a predictive model that accurately forecasts the sales of each product in the Supermarket By addressing the following key objectives, our aim to provide actionable insights to Supermarket management team:
- Utilize machine learning techniques to analyze the relationship between various product and store attributes and sales.
- Identify the most significant factors influencing sales and their impact on product performance.
- Provide actionable insights to Supermarket to optimize product placement, pricing strategies, and inventory management.

Methodology in Detail

- About Data Set
- Data Preprocessing
- Exploratory Data Analysis(EDA)
- Model Selection and Training

About Data Set

- **Our dataset contains 12 features and data for 1559 products across 10 stores in different cities**
- **Total data contains in dataset is 14204**

	Item_Iden	Item_Weig	Item_Fat_	Item_Visib	Item_Type	Item_MRP	Outlet_Ide	Outlet_Est	Outlet_Siz	Outlet_Lo	Outlet_Ty	Item_Outl	t_Sales
2	FDA15	9.3	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermark	3735.138	
3	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermark	443.4228	
4	FDN15	17.5	Low Fat	0.01676	Meat	141.618	OUT049	1999	Medium	Tier 1	Supermark	2097.27	
5	FDX07	19.2	Regular	0	Fruits and	182.095	OUT010	1998		Tier 3	Grocery St	732.38	
6	NCD19	8.93	Low Fat	0	Household	53.8614	OUT013	1987	High	Tier 3	Supermark	994.7052	
7	FDP36	10.395	Regular	0	Baking Goo	51.4008	OUT018	2009	Medium	Tier 3	Supermark	556.6088	
8	FDO10	13.65	Regular	0.012741	Snack Foo	57.6588	OUT013	1987	High	Tier 3	Supermark	343.5528	
9	FDP10		Low Fat	0.12747	Snack Foo	107.7622	OUT027	1985	Medium	Tier 3	Supermark	4022.764	
10	FDH17	16.2	Regular	0.016687	Frozen Foo	96.9726	OUT045	2002		Tier 2	Supermark	1076.599	
11	FDU28	19.2	Regular	0.09445	Frozen Foo	187.8214	OUT017	2007		Tier 2	Supermark	4710.535	
12	FDY07	11.8	Low Fat	0	Fruits and	45.5402	OUT049	1999	Medium	Tier 1	Supermark	1516.027	
13	FDA03	18.5	Regular	0.045464	Dairy	144.1102	OUT046	1997	Small	Tier 1	Supermark	2187.153	
14	FDX32	15.1	Regular	0.100014	Fruits and	145.4786	OUT049	1999	Medium	Tier 1	Supermark	1589.265	
15	FDS46	17.6	Regular	0.047257	Snack Foo	119.6782	OUT046	1997	Small	Tier 1	Supermark	2145.208	
16	FDF32	16.35	Low Fat	0.068024	Fruits and	196.4426	OUT013	1987	High	Tier 3	Supermark	1977.426	
17	FDP49	9	Regular	0.069089	Breakfast	56.3614	OUT046	1997	Small	Tier 1	Supermark	1547.319	
18	NCB42	11.8	Low Fat	0.008596	Health and	115.3492	OUT018	2009	Medium	Tier 3	Supermark	1621.889	
19	FDP49	9	Regular	0.069196	Breakfast	54.3614	OUT049	1999	Medium	Tier 1	Supermark	718.3982	
20	DRI11		Low Fat	0.034238	Hard Drink	113.2834	OUT027	1985	Medium	Tier 3	Supermark	2303.668	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 14204 entries, 0 to 5680
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                11765 non-null  float64
 2   Item_Fat_Content           14204 non-null  object
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object
 7   Outlet_Establishment_Year  14204 non-null  int64
 8   Outlet_Size                10188 non-null  object
 9   Outlet_Location_Type       14204 non-null  object
 10  Outlet_Type                14204 non-null  object
 11  Item_Outlet_Sales          14204 non-null  float64
 12  source                     14204 non-null  object
dtypes: float64(4), int64(1), object(8)
memory usage: 1.5+ MB
```

Data Preprocessing

- **Handling missing values in the dataset :** Identifying and addressing any missing values in the dataset, which may arise due to data entry errors, technical issues, or incomplete records. Common techniques for handling missing values include imputation (replacing missing values with estimates such as mean, median, or mode), deletion (removing rows or columns with missing values), or advanced imputation methods based on the relationships between variables.
- **Dealing with Outliers :** Detecting and addressing outliers, which are data points that significantly deviate from the rest of the dataset. Outliers can distort statistical analyses and machine learning models, leading to inaccurate results.
- **Correcting Data Types :** Ensuring that variables are encoded with the appropriate data types (e.g., numeric, categorical, datetime) to facilitate analysis and modeling. This may involve converting categorical variables to dummy variables (one-hot encoding), parsing datetime variables, or converting numerical variables to the desired format.
- cleaning the dataset like identifying and rectifying any inconsistencies and errors , removing duplicates, addressing inconsistent data such as conflicting information across variables or logical errors
- standardizing the dataset to prepare it for analysis.

Handling Null Values

```
df.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                  2439
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  4016
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
source                          0
dtype: int64
```

```
[21]: df.isnull().sum()
```

```
[21]: 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
      source                          0
      dtype: int64
```

Dealing with Missing Values

```
from sklearn.impute import SimpleImputer
from scipy.stats import pearsonr
```

```
# Dealing with Item_Weight Column
```

```
df['Item_Weight'].mean()
```

```
12.792854228644284
```

```
py()
mpleImputer(strategy='mean')
 = mean_imputer.fit_transform(df['Item_Weight'].values.
```

```
.isnull().sum()
```

Dealing with Missing Values

```
#Dealing with Outlet_Size Column
```

```
df['Outlet_Size'].unique()
```

```
array(['Medium', nan, 'High', 'Small'], dtype=object)
```

```
df['Outlet_Size'].mode()
```

```
0    Medium
Name: Outlet_Size, dtype: object
```

```
mode_of_outlet_size = df.pivot_table(values='Outlet_Size', columns='Outlet_Type', aggfunc=(lambda x: x.mode()[0]))
```

```
missing_bool = df['Outlet_Size'].isnull()
df.loc[missing_bool, 'Outlet_Size'] = df.loc[missing_bool, 'Outlet_Type'].apply(lambda x: mode_of_outlet_size[x])
```

```
df['Outlet_Size'].isnull().sum()
```

```
0
```

Dealing With Irregular Data

```
'Item_Fat_Content'].unique()
```

```
ray(['Low Fat', 'Regular', 'low fat', 'LF', 'reg'], dtype=object)
```

```
'Item_Fat_Content'] = df['Item_Fat_Content'].replace({'LF':'Low Fat', 'reg':'Regular', 'low fat':'Low Fat'})
'Item_Fat_Content'].unique()
```

```
ray(['Low Fat', 'Regular'], dtype=object)
```

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

```
em_Fat_Content
w Fat    9185
gular    5019
me: count, dtype: int64
```

Data Preprocessing

```
df['Item_Type'].unique()
```

```
array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
       'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
       'Breakfast', 'Health and Hygiene', 'Hard Drinks', 'Canned',
       'Breads', 'Starchy Foods', 'Others', 'Seafood'], dtype=object)
```

```
df['New_Item_Type'].unique()
```

```
array(['FD', 'DR', 'NC'], dtype=object)
```

```
df['New_Item_Type'] = df['New_Item_Type'].map({'FD':'Food', 'NC':'Non-Consumable', 'DR':'Drinks'})
df['New_Item_Type'].value_counts()
```

```
New_Item_Type
Food              10201
Non-Consumable     2686
Drinks             1317
```

```
df[['Item_Identifier','Item_Type']]
```

	Item_Identifier	Item_Type
0	FDA15	Dairy
1	DRC01	Soft Drinks
2	FDN15	Meat
3	FDX07	Fruits and Vegetables

One-Hot encoding

- One-hot encoding is a technique used in machine learning and data preprocessing to handle categorical variables, transforming them into a format that can be provided to machine learning algorithms.

```
encoder = LabelEncoder()
```

```
df['Item_Identifier'] = encoder.fit_transform(df['Item_Identifier'])
df['Item_Fat_Content'] = encoder.fit_transform(df['Item_Fat_Content'])
df['Item_Type'] = encoder.fit_transform(df['Item_Type'])
df['Outlet_Identifier'] = encoder.fit_transform(df['Outlet_Identifier'])
df['Outlet_Size'] = encoder.fit_transform(df['Outlet_Size'])
df['Outlet_Location_Type'] = encoder.fit_transform(df['Outlet_Location_Type'])
df['Outlet_Type'] = encoder.fit_transform(df['Outlet_Type'])
df['New_Item_Type'] = encoder.fit_transform(df['New_Item_Type'])
```

```
import pandas as pd
df = pd.get_dummies(df, columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type',
                                 'New_Item_Type','Outlet_Identifier'])
```

Pre processing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 14204 entries, 0 to 5680
Data columns (total 33 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Item_Identifier         14204 non-null  int32
 1   Item_Weight             14204 non-null  float64
 2   Item_Visibility         14204 non-null  float64
 3   Item_MRP                14204 non-null  float64
 4   Item_Outlet_Sales       14204 non-null  float64
 5   source                  14204 non-null  object
 6   Outlet_Years            14204 non-null  int64
 7   Item_Fat_Content_0      14204 non-null  bool
 8   Item_Fat_Content_1      14204 non-null  bool
 9   Item_Fat_Content_2      14204 non-null  bool
 10  Outlet_Location_Type_0  14204 non-null  bool
 11  Outlet_Location_Type_1  14204 non-null  bool
 12  Outlet_Location_Type_2  14204 non-null  bool
 13  Outlet_Size_0           14204 non-null  bool
 14  Outlet_Size_1           14204 non-null  bool
 15  Outlet_Size_2           14204 non-null  bool
 16  Outlet_Type_0           14204 non-null  bool
 17  Outlet_Type_1           14204 non-null  bool
 18  Outlet_Type_2           14204 non-null  bool
 19  Outlet_Type_3           14204 non-null  bool
 20  New_Item_Type_0         14204 non-null  bool
 21  New_Item_Type_1         14204 non-null  bool
 22  New_Item_Type_2         14204 non-null  bool
 23  Outlet_Identifier_0     14204 non-null  bool
 24  Outlet_Identifier_1     14204 non-null  bool
 25  Outlet_Identifier_2     14204 non-null  bool
 26  Outlet_Identifier_3     14204 non-null  bool
 27  Outlet_Identifier_4     14204 non-null  bool
 28  Outlet_Identifier_5     14204 non-null  bool
 29  Outlet_Identifier_6     14204 non-null  bool
 30  Outlet_Identifier_7     14204 non-null  bool
 31  Outlet_Identifier_8     14204 non-null  bool
 32  Outlet_Identifier_9     14204 non-null  bool
dtypes: bool(26), float64(4), int32(1), int64(1), object(1)
memory usage: 1.2+ MB
```

```
vide into test and train:
in = df.loc[df['source']=="train"]
t = df.loc[df['source']=="test"]

op unnecessary columns:
t.drop(['Item_Outlet_Sales','source'],axis=1,inplace=True
in.drop(['source'],axis=1,inplace=True)

port files as modified versions:
in.to_csv("train_modified.csv",index=False)
t.to_csv("test_modified.csv",index=False)
```

Exploratory Data Analysis(EDA)

Data Visualization:

- We used Seaborn and Matplotlib to visualize various aspects of the dataset, such as the distribution of 'Item_Weight', 'Item_Visibility', 'Item_MRP', and 'Item_Outlet_Sales'.
- Count plots were utilized to visualize the distribution of categorical variables like 'Item_Fat_Content', 'Item_Type', 'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Type', and 'Outlet_Location_Type'.
- The sns.displot function was used to create KDE plots for continuous variables.

Exploring Missing Values:

- We checked for missing values in the dataset using df.isnull().sum() and visualized them using the msno.matrix() function from the missingno library.

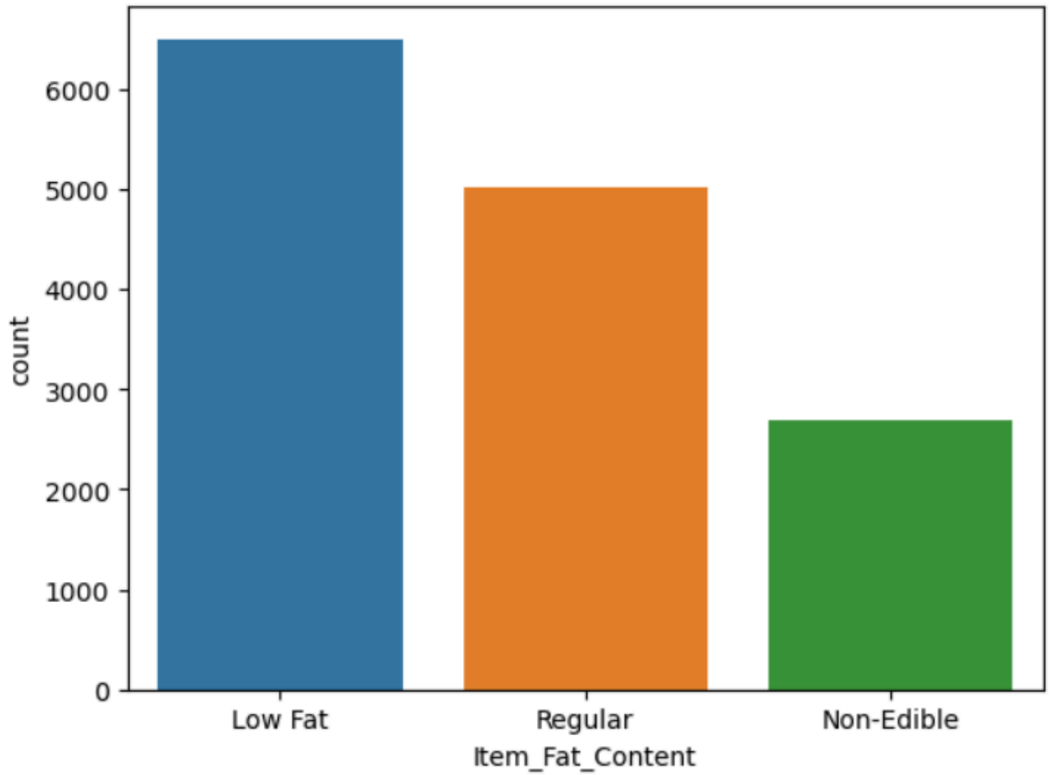
Feature Engineering:

- New features such as 'New_Item_Type' and 'Outlet_Years' were created based on existing columns, which is a part of EDA to derive meaningful insights from the data.

Visualizing Insights

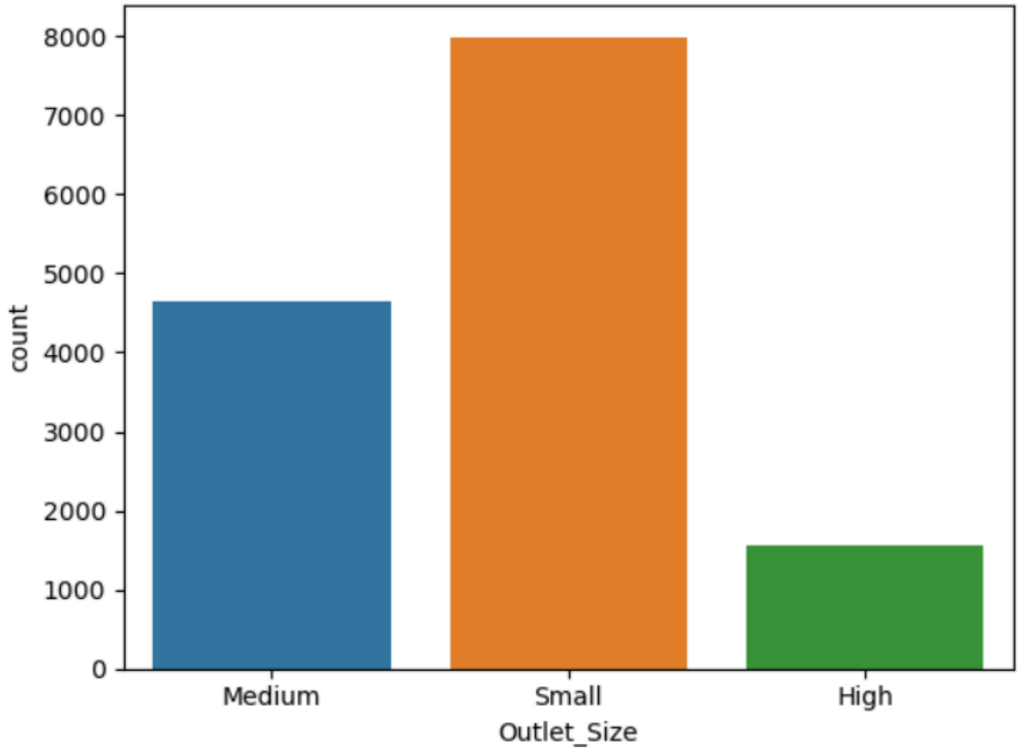
```
sns.countplot(x=df["Item_Fat_Content"])
```

```
<Axes: xlabel='Item_Fat_Content', ylabel='count'>
```




```
sns.countplot(x = df['Outlet_Size'])
```

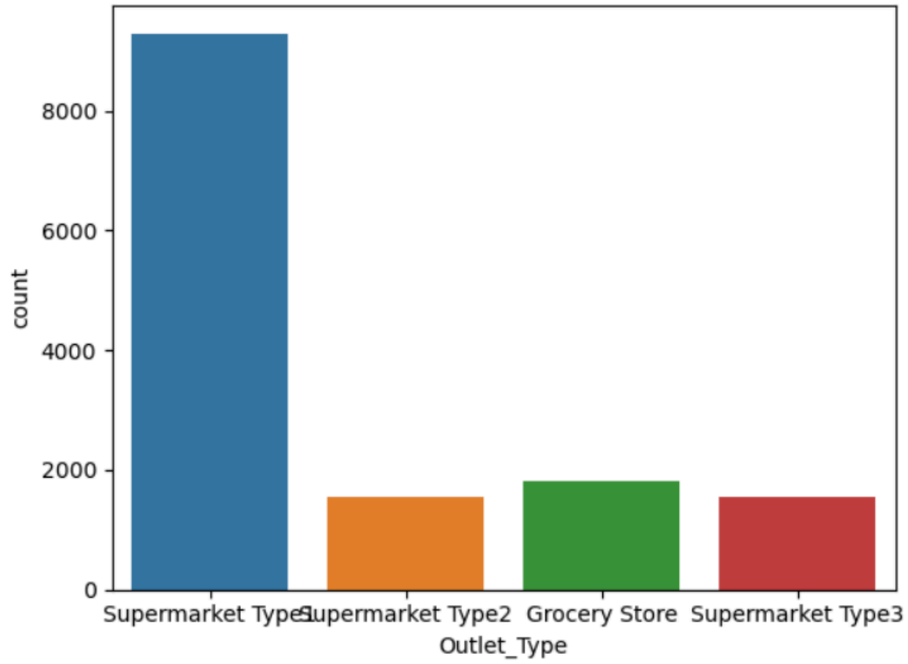
```
<Axes: xlabel='Outlet_Size', ylabel='count'>
```



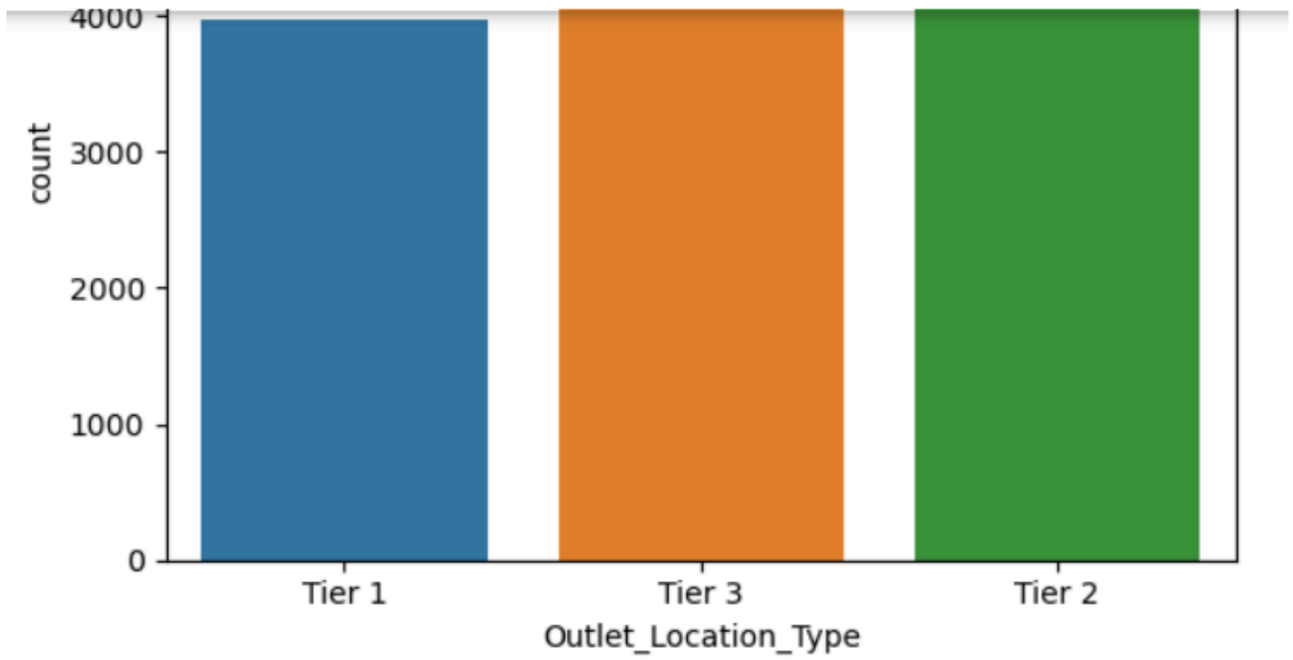
Visualizing Insights

```
sns.countplot(x = df['Outlet_Type'])
```

```
<Axes: xlabel='Outlet_Type', ylabel='count'>
```



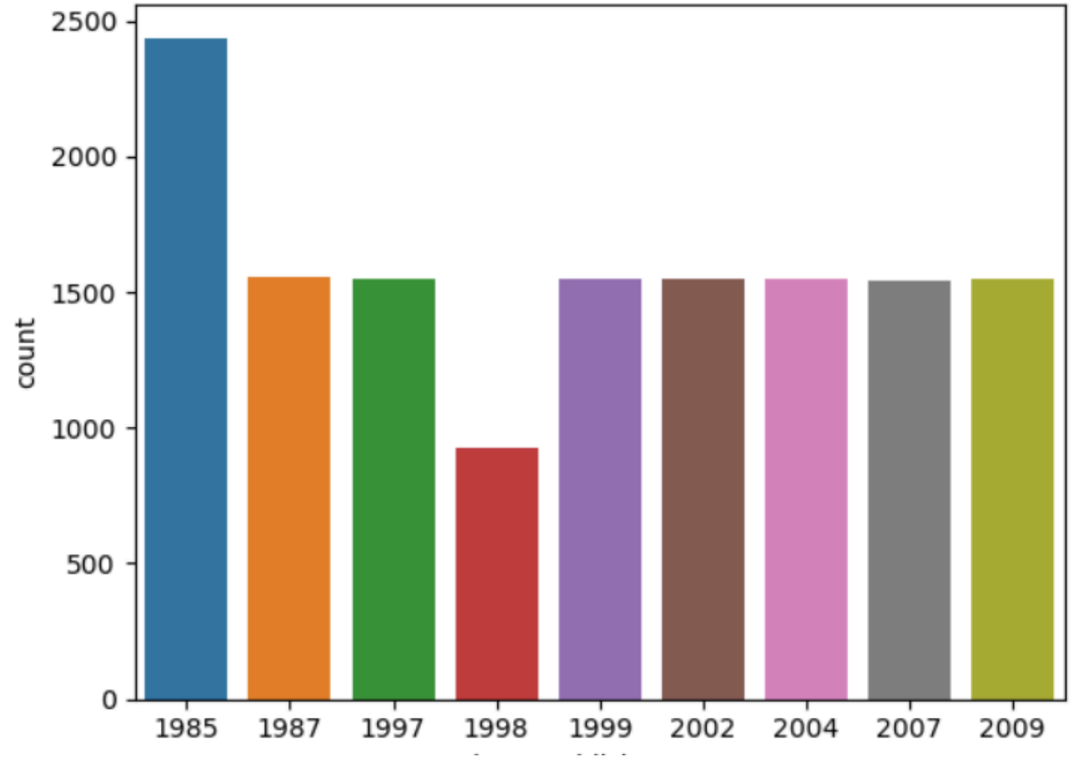

```
sns.countplot(x = df['Outlet_Location_Type'])
```



Visualizing Insights

```
df['Outlet_Years'] = 2013 - df['Outlet_Establishment_Year']
```

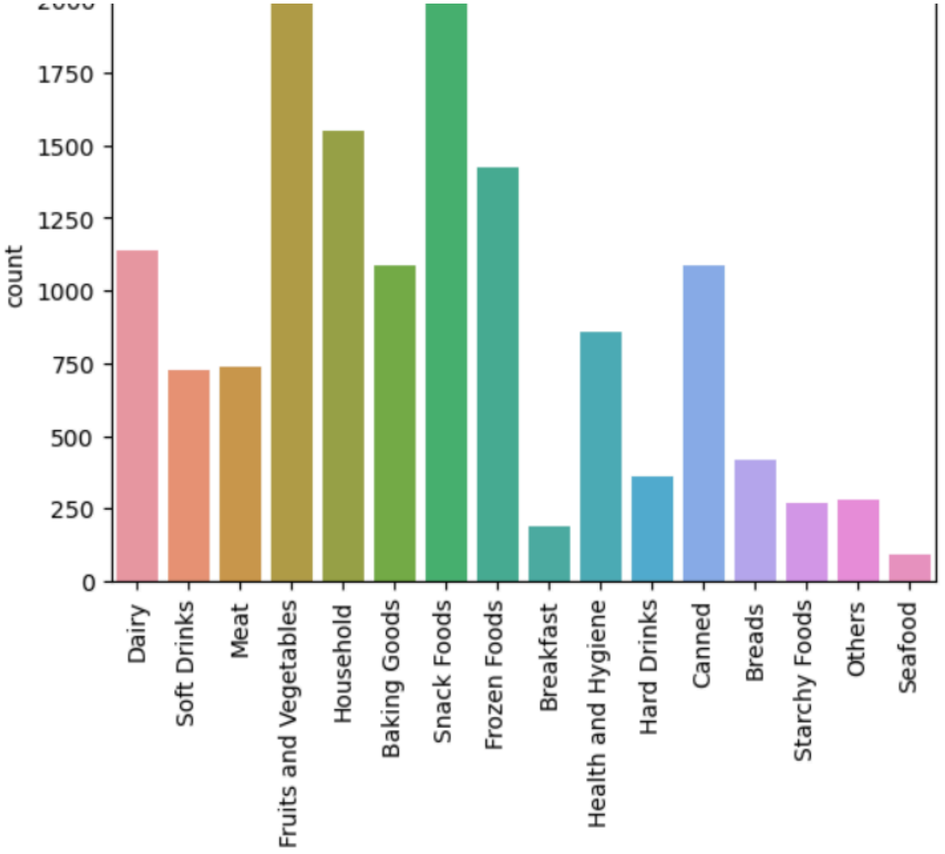
```
<Axes: xlabel='Outlet_Establishment_Year', ylabel='count'>
```



Visualizing Insights

```
# plt.figure(figsize=(15,5))
l = list(df['Item_Type'].unique())
chart = sns.countplot(x=df["Item_Type"])
chart.set_xticklabels(labels=l, rotation=90)
```

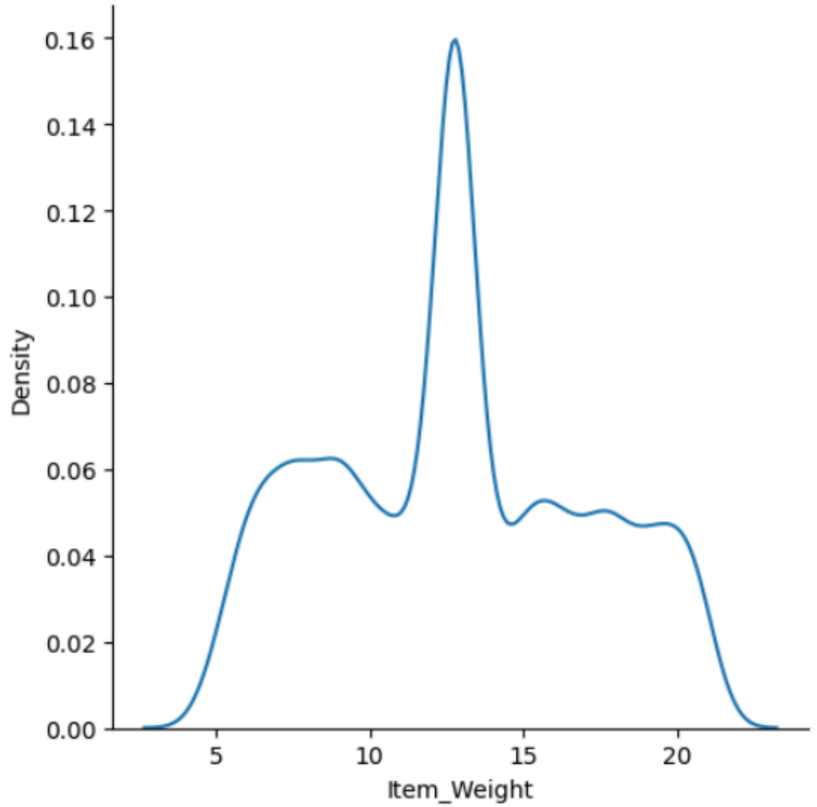
```
[Text(0, 0, 'Dairy'),
 Text(1, 0, 'Soft Drinks'),
 Text(2, 0, 'Meat'),
 Text(3, 0, 'Fruits and Vegetables'),
 Text(4, 0, 'Household'),
 Text(5, 0, 'Baking Goods'),
 Text(6, 0, 'Snack Foods'),
 Text(7, 0, 'Frozen Foods'),
 Text(8, 0, 'Breakfast'),
 Text(9, 0, 'Health and Hygiene'),
 Text(10, 0, 'Hard Drinks'),
 Text(11, 0, 'Canned'),
 Text(12, 0, 'Breads'),
 Text(13, 0, 'Starchy Foods'),
 Text(14, 0, 'Others'),
 Text(15, 0, 'Seafood')]
```



Seaborn

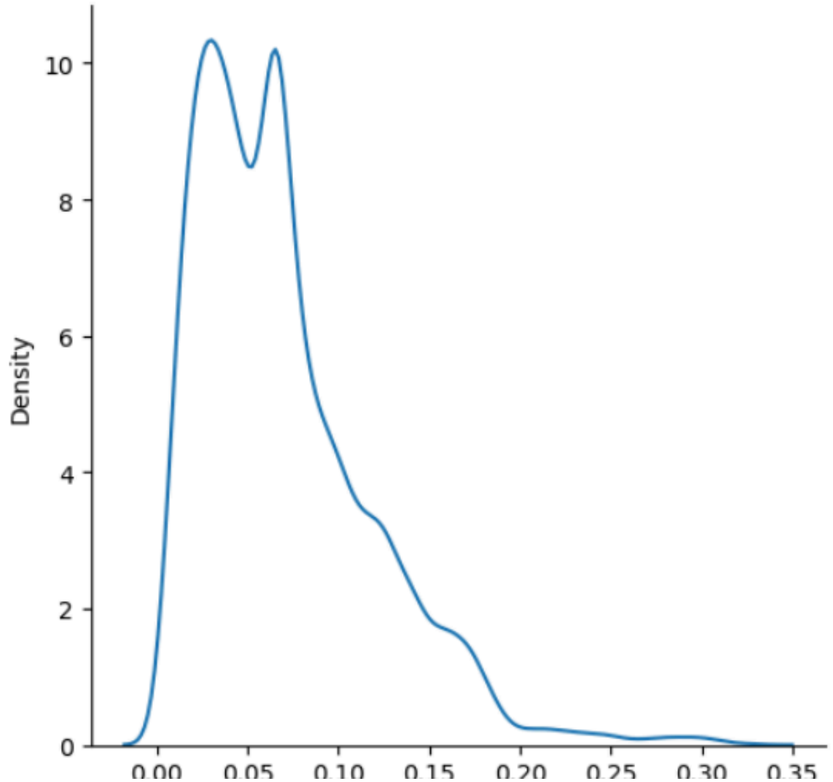
```
sns.displot(df['Item_Weight'],kind = 'kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x292ad96aa70>
```




```
sns.displot(df['Item_Visibility'],kind='kde')
```

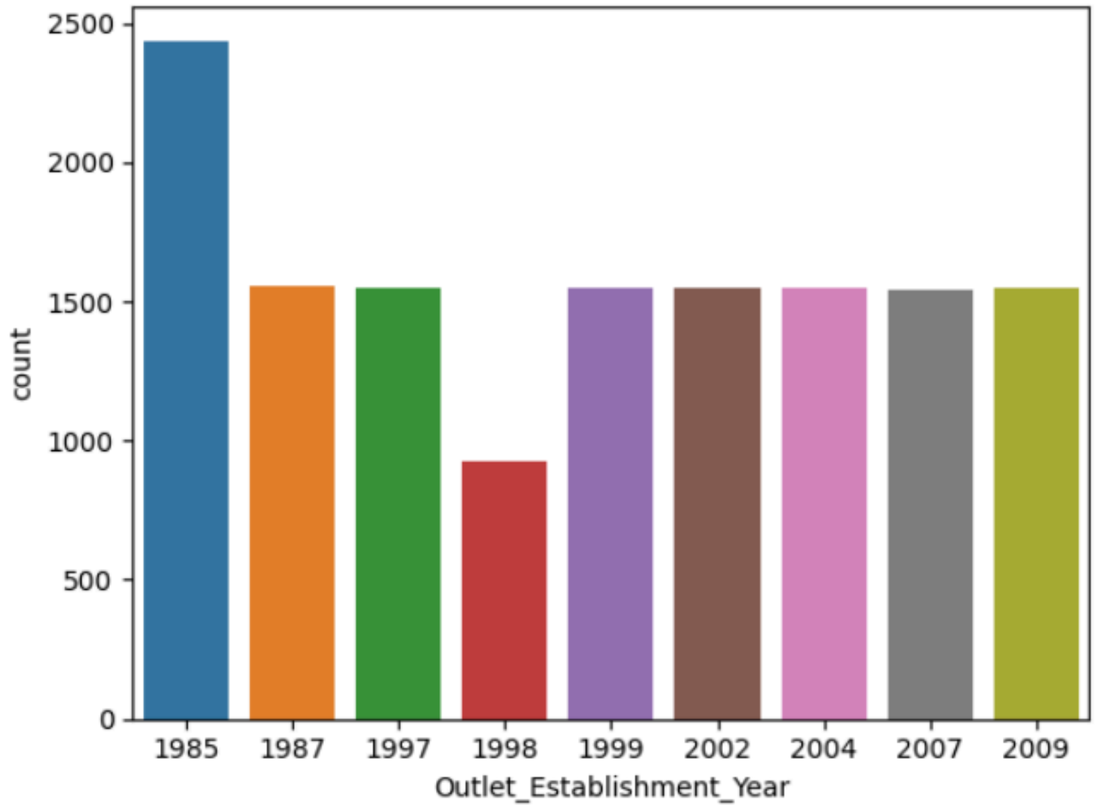
```
<seaborn.axisgrid.FacetGrid at 0x292ad96bac0>
```



Count Plot

```
sns.countplot(x=df["Outlet_Establishment_Year"])
```

```
<AxesSubplot: xlabel='Outlet_Establishment_Year', ylabel='count'>
```



Model selection and training

- We imported various regression algorithms from scikit-learn, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regression, Random Forest Regression, and XGBoost Regression.

- For each algorithm, we created an instance of the algorithm and called the modelfit function to train the model, evaluate its performance on the training set, and make predictions on the test set.

Model selection and training

```
target = 'Item_Outlet_Sales'
IDcol = ['Item_Identifier','Outlet_Identifier']
from sklearn import metrics
def modelfit(alg, dtrain, dtest, predictors, target, IDcol):
    alg.fit(dtrain[predictors], dtrain[target])
    #Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors])
    r2_train = metrics.r2_score(dtrain[target], dtrain_predictions)
    print("R Square Value  :",r2_train)
    #Predict on testing data:
    dtest[target] = alg.predict(dtest[predictors])
```

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
```

Implementation Results of regression models:

- After Selecting models we implemented the models and their outputs are displayed in the next slides.
- Linear Regression
- Ridge
- Lasso
- Decision tree
- XGB Regressor
- Random Forest

Linear regression OUTPUT

```
predictors = [x for x in train.columns if x not in [target]+IDc
alg1 = LinearRegression()
modelfit(alg1, train, test, predictors, target, IDcol)
```

```
R Square Value  : 0.7205206644669215
```

Ridge OUTPUT

```
alg2 = Ridge()
modelfit(alg2, train, test, predictors, target, IDcol)
```

```
R Square Value   : 0.7205205606833663
```

Lasso OUTPUT

```
alg3 = Lasso()
modelfit(alg3, train, test, predictors, target, IDcol)
```

```
R Square Value   : 0.2597339047834789
```

Decision Tree OUTPUT

```
alg4 = DecisionTreeRegressor()
modelfit(alg4, train, test, predictors, target, IDcol)
```

```
R Square Value  : 1.0
```

Random Forest OUTPUT

```
alg5 = RandomForestRegressor()
modelfit(alg5, train, test, predictors, target, IDcol)
```

```
R Square Value   : 0.9587193682116294
```

XGB Regressor OUTPUT

```
alg6 = XGBRegressor()
modelfit(alg6, train, test, predictors, target, IDcol)
```

```
R Square Value   : 0.8634991269855565
```

Result/Outcome

By implementing all regression models we came to know to know that decision tree performed well.