Big Sales Prediction using Random Forest Regressor

Objective

The primary objective of this project is to develop an accurate and reliable sales prediction model using Python and the Random Forest Regressor algorithm. This model aims to forecast future sales for a business, enabling better decision-making and strategic planning. The specific goals of the project include:

1. **Data Preparation and Preprocessing:** Collect and preprocess historical sales data, including handling missing values, outliers, and feature engineering to create a clean and informative dataset for model training.
2. **Model Development:** Implement the Random Forest Regressor algorithm to build a robust sales prediction model. This involves tuning hyperparameters to optimize model performance.
3. **Feature Importance Analysis:** Analyze the importance of different features in predicting sales to provide insights into the factors that most significantly impact sales performance.
4. **Model Evaluation:** Evaluate the performance of the sales prediction model using appropriate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2) to ensure accuracy and reliability.
5. **Scalability:** Ensure the model can handle large datasets and provide quick predictions, making it suitable for real-time sales forecasting applications.
6. **Visualization and Reporting:** Develop visualizations to present the prediction results and feature importance clearly. Provide comprehensive reports that summarize the findings and recommendations based on the model's outputs.
7. **Deployment:** Deploy the model in a production environment where it can be used by stakeholders to make informed decisions regarding inventory management, marketing strategies, and sales planning.

By achieving these goals, the project aims to deliver a powerful tool for predicting sales, which can help businesses optimize their operations, reduce costs, and increase revenue through data-driven decision-making.

About the Dataset-

The dataset consists of 12 Varaibles:

1. Item_Identifier - This is an object type column and it has no null values. It contains unique identifiers for each item.
2. Item_Weight - This is a float type column and it has some null values. It represents the weight of each item.
3. Item_Fat_Content - This is an object type column and it has no null values. It represents the fat content of each item.
4. Item_Visibility - This is a float type column and it has no null values. It might represent the visibility of each item in the store.

5. Item_Type - This is an object type column and it has no null values. It represents the type of each item.
6. Item_MRP - This is a float type column and it has no null values. It represents the maximum retail price of each item.
7. Outlet_Identifier - This is an object type column and it has no null values. It contains unique identifiers for each outlet.
8. Outlet_Establishment_Year - This is an integer type column and it has no null values. It represents the year each outlet was established.
9. Outlet_Size - This is an object type column and it has no null values. It represents the size of each outlet.
10. Outlet_Location_Type - This is an object type column and it has no null values. It represents the location type of each outlet.
11. Outlet_Type - This is an object type column and it has no null values. It represents the type of each outlet.
12. Item_Outlet_Sales - This is a float type column and it has no null values. It represents the sales of each item at each outlet.

Data Source

The source of the data is **Ybi Foundation's** Github Dataset Repository:

https://github.com/YBIFoundation/Dataset

Project Workings

1. Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Import Dataset

```
# Dataset imported directly from Github raw link:
df = pd.read_csv('https://github.com/YBIFoundation/Dataset/raw/main/Big%20Sales%20Data.csv')
```

3. Describe Data

```
df.head()
# Gives top 5 Rows of the DataFrame
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visit
0	FDT36	12.3	Low Fat	0.′
1	FDT36	12.3	Low Fat	0.′
2	FDT36	12.3	LF	0.′
3	FDT36	12.3	Low Fat	0.C
4	FDP12	9.8	Regular	0.C

```
df.info()
# Gives dataframe information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            14204 non-null  object 
 1   Item_Weight                11815 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                14204 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
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

```
The DataFrame consists of 14203 entries with No Null values.
It contains 12 columns which inlcudes 7 object, 4 float64
and 1 int64 datatypes.
```

```
df.describe()
# Gives the Statistical Summary of the Numerical values of DataFrame
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Est
count	11815.000000	14204.000000	14204.000000	
mean	12.788355	0.065953	141.004977	
std	4.654126	0.051459	62.086938	
min	4.555000	0.000000	31.290000	
25%	8.710000	0.027036	94.012000	
50%	12.500000	0.054021	142.247000	
75%	16.750000	0.094037	185.855600	
max	30.000000	0.328391	266.888400	

```
df.columns
# Get column names of the DataFrame
```

```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

Now, We need to get the Missing Values Complete

```
df['Item_Weight'].fillna(df.groupby(['Item_Type'])['Item_Weight'].transform('mean'), inplace=True)
```

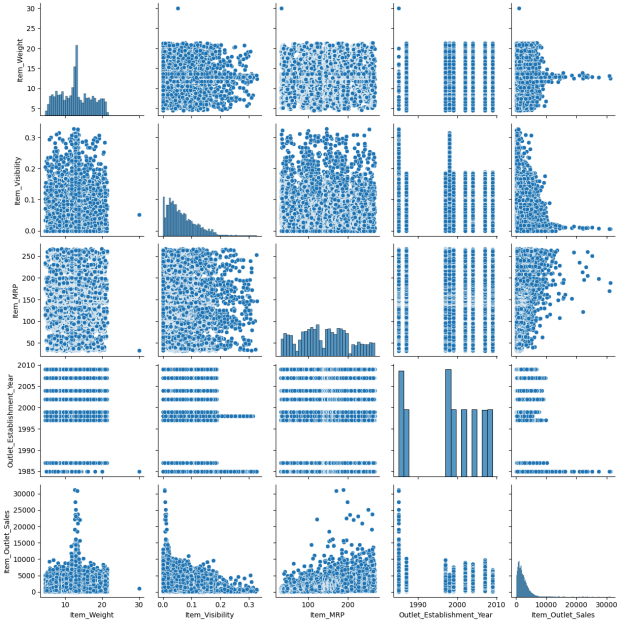
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            14204 non-null  object 
 1   Item_Weight                14204 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                14204 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
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

Data Visualization

```
# Pairplot
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7a4fbb44b850>
```



Data Preprocessing

Get Categories and Counts of Categorical Variables

```
df[['Item_Identifier']].value_counts()
```

```
Item_Identifier
FDQ08              10
FDO24              10
FDQ19              10
FDQ28              10
FDQ31              10
                   ..
FDM52               7
FDM50               7
FDL50               7
FDM10               7
FDR51               7
Name: count, Length: 1559, dtype: int64
```

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

```
Item_Fat_Content
Low Fat             8485
Regular             4824
LF                   522
reg                  195
low fat              178
Name: count, dtype: int64
```

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

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

```
Item_Fat_Content
Low Fat             9185
Regular             5019
Name: count, dtype: int64
```

```
df.replace({'Item_Fat_Content': {'Low Fat': 0,'Regular': 1}}, inplace=True)
```

```
# This is done to Replace "Low Fat" as "0" and "Regular" as "1".
```

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

```
Item_Type
Fruits and Vegetables    2013
Snack Foods              1989
Household                1548
```

```
Frozen Foods             1426
Dairy                    1136
Baking Goods             1086
Canned                   1084
Health and Hygiene        858
Meat                      736
Soft Drinks               726
Breads                    416
Hard Drinks               362
Others                    280
Starchy Foods             269
Breakfast                 186
Seafood                    89
Name: count, dtype: int64
```

```
# Let's segregate the Item Types on the basis of their nature.
df.replace({'Item_Type': {'Fruits and Vegetables':0, 'Snack Foods':0, 'Household':1, 'Frozen Foods':
                          'Dairy':0, 'Baking Goods':0, 'Canned':0, 'Health and Hygiene':1,
                          'Meat':0, 'Soft Drinks':0, 'Breads':0, 'Hard Drinks':0, 'Others':2,
                          'Starchy Foods':0, 'Breakfast':0, 'Seafood':0}}, inplace=True)
```

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

```
Item_Type
0            11518
1             2406
2              280
Name: count, dtype: int64
```

```
df[['Outlet_Identifier']].value_counts()
```

```
Outlet_Identifier
OUT027               1559
OUT013               1553
OUT035               1550
OUT046               1550
OUT049               1550
OUT045               1548
OUT018               1546
OUT017               1543
OUT010                925
OUT019                880
Name: count, dtype: int64
```

```
df.replace({'Outlet_Identifier':{'OUT027':0, 'OUT013':1, 'OUT035':2, 'OUT046':3, 'OUT049':4,
                                 'OUT045':5, 'OUT018':6, 'OUT017':7, 'OUT010':8, 'OUT019':9,
                                 }}, inplace=True)
```

```
df[['Outlet_Identifier']].value_counts()
```

```
Outlet_Identifier
0                    1559
1                    1553
2                    1550
3                    1550
4                    1550
5                    1548
6                    1546
7                    1543
8                     925
```

```
9                      880
Name: count, dtype: int64
```

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

```
Outlet_Size
Medium         7122
Small          5529
High           1553
Name: count, dtype: int64
```

```
df.replace({'Outlet_Size': {'Small':0, 'Medium':1, 'High':2}}, inplace=True)
```

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

```
Outlet_Size
1              7122
0              5529
2              1553
Name: count, dtype: int64
```

```
df[['Outlet_Location_Type']].value_counts()
```

```
Outlet_Location_Type
Tier 3                  5583
Tier 2                  4641
Tier 1                  3980
Name: count, dtype: int64
```

```
df.replace({'Outlet_Location_Type': {'Tier 1':0, 'Tier 2':1, 'Tier 3':2}}, inplace=True)
```

```
df[['Outlet_Location_Type']].value_counts()
```

```
Outlet_Location_Type
2                       5583
1                       4641
0                       3980
Name: count, dtype: int64
```

```
df[['Outlet_Type']].value_counts()
```

```
Outlet_Type
Supermarket Type1    9294
Grocery Store        1805
Supermarket Type3    1559
Supermarket Type2    1546
Name: count, dtype: int64
```

```
df.replace({'Outlet_Type': {'Grocery Store':0, 'Supermarket Type1':1, 'Supermarket Type2':2, 'Superma
```

```
df[['Outlet_Type']].value_counts()
```

```
Outlet_Type
1              9294
0              1805
3              1559
```

```
2            1546
Name: count, dtype: int64
```

```
df.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visit
0	FDT36	12.3	0	0.‹
1	FDT36	12.3	0	0.‹
2	FDT36	12.3	0	0.‹
3	FDT36	12.3	0	0.C
4	FDP12	9.8	1	0.C

Next steps: Generate code with df View recommended plots

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                14204 non-null  float64
 2   Item_Fat_Content           14204 non-null  int64
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  int64
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  int64
 7   Outlet_Establishment_Year  14204 non-null  int64
 8   Outlet_Size                14204 non-null  int64
 9   Outlet_Location_Type       14204 non-null  int64
 10  Outlet_Type                14204 non-null  int64
 11  Item_Outlet_Sales          14204 non-null  float64
dtypes: float64(4), int64(7), object(1)
memory usage: 1.3+ MB
```

```
# Shape of DataFrame
df.shape
```

```
(14204, 12)
```

Define Dependant or Target Variable (y) and Features or Independant Variable (x)

```
y = df['Item_Outlet_Sales']
y
```

```
0         436.608721
1         443.127721
2         564.598400
3        1719.370000
4         352.874000
            ...
14199    4984.178800
```

```
14200    2885.577200
14201    2885.577200
14202    3803.676434
14203    3644.354765
Name: Item_Outlet_Sales, Length: 14204, dtype: float64
```

```
y.shape
```

```
(14204,)
```

```
x = df.drop(['Item_Identifier','Item_Outlet_Sales'], axis=1)
```

```
x
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_I
0	12.300000	0	0.111448	
1	12.300000	0	0.111904	
2	12.300000	0	0.111728	
3	12.300000	0	0.000000	
4	9.800000	1	0.045523	
...	...	...	...	
14199	12.800000	0	0.069606	
14200	12.800000	0	0.070013	
14201	12.800000	0	0.069561	
14202	13.659758	0	0.069282	
14203	12.800000	0	0.069727	

14204 rows × 10 columns

Next steps: Generate code with x View recommended plots

```
x.shape
```

```
(14204, 10)
```

X Variables Standardization

Standardization of the independent variable X is a preprocessing step in machine learning and statistical modeling where the features (or predictors) are transformed to have a mean of zero and a standard deviation of one.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
x_std = df[['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year']]
```

```
x_std = sc.fit_transform(x_std)
```

```
x_std
```

```
array([[-0.11541705,  0.88413635, -1.73178716,  0.13968068],
       [-0.11541705,  0.89300616, -1.72373366,  1.09531886],
       [-0.11541705,  0.88958331, -1.72373366,  1.3342284 ],
       ...,
       [ 0.00220132,  0.07011952,  1.96538148, -1.29377659],
       [ 0.20444792,  0.06469366,  1.97343499, -1.53268614],
       [ 0.00220132,  0.07334891,  1.97504569,  0.13968068]])
```

```
x[['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year']] = pd.DataFrame(x_std,
```

```
x
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_1
0	-0.115417	0	0.884136	
1	-0.115417	0	0.893006	
2	-0.115417	0	0.889583	
3	-0.115417	0	-1.281712	
4	-0.703509	1	-0.397031	
...	...	...	...	
14199	0.002201	0	0.070990	
14200	0.002201	0	0.078898	
14201	0.002201	0	0.070120	
14202	0.204448	0	0.064694	
14203	0.002201	0	0.073349	

14204 rows × 10 columns

Next steps: Generate code with x | View recommended plots

Train Test Split

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, train_size=0.1, random_state=2529)
```

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((1420, 10), (12784, 10), (1420,), (12784,))
```

Fit Model

```
from sklearn.ensemble import RandomForestRegressor
```

```
rfr = RandomForestRegressor(random_state=2529)
```

```
rfr.fit(x_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=2529)
```

Model Prediction

```
y_pred = rfr.predict(x_test)
y_pred
```

```
array([1244.96860546, 1126.98222234, 1690.99249499, ...,  664.77915272,
       1807.79289687, 3024.5640115 ])
```

Model Evaluation

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
mean_squared_error(y_test, y_pred)
```

```
1841246.1246989528
```

```
r2_score(y_test, y_pred)
```

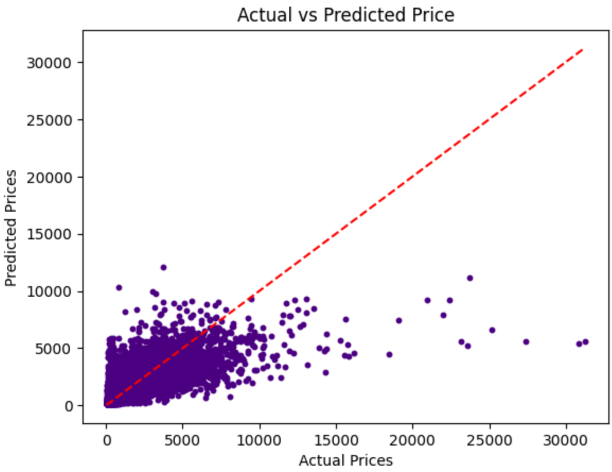
```
0.45492462499764463
```

```
mean_absolute_error(y_test, y_pred)
```

```
843.0664547522249
```

Visualization of Actual vs Predicted Results

```
import matplotlib.pyplot as plt

#Scatter Plot

plt.scatter(y_test, y_pred, color='indigo', s=10)

min_val = min(min(y_test), min(y_pred))
max_val = max(max(y_test), max(y_pred))
plt.plot([min_val, max_val], [min_val, max_val], color='red', linestyle='--')

plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Price")

plt.show()
```



Explanation

Actual Prices: These are the true values for Item_Outlet_Sales in our dataset. They are what actually happened in the past and what we are trying to predict accurately with our model.

Predicted Prices: These are the values that our model predicts for Item_Outlet_Sales. The model generates these predictions based on the patterns it learned during the training process.

By comparing the actual and predicted prices, we can see where our model is making accurate predictions and where it is making errors. For example, if the actual price for an item is 10 and our model predicts a price of 15, then our model is overestimating the price for that item.

Points above the line represent items where our model overestimated the price, and points below the line represent items where our model underestimated the price.

Thank You!

Ankit Joshi

+ Code + Text