

CSCI 6364: MACHINE LEARNING

PROFESSOR: DAVID TROTT

PROJECT REPORT

BIG SALES PREDICTION USING RANDOM FOREST REGRESSOR

NAME: SRAVANI BRAHAMMA ROUTHU

GWID: G28059824

ABSTRACT

Sales prediction is a critical aspect of the retail industry, serving as a foundation for effective inventory management, revenue forecasting, and strategic decision-making. Accurate sales forecasting enables retailers to optimize resource allocation, devise pricing strategies, and enhance operational efficiency. This project aims to address the challenge of sales prediction in the retail sector by leveraging machine learning methodologies. The objective is to develop a robust predictive model capable of accurately estimating sales based on a diverse set of attributes characterizing both the products and the retail outlets. The dataset used in this project comprises 14,204 entries related to sales in the retail industry, encompassing 12 attributes that characterize both the products and the outlets where they are sold. Through data analysis, preprocessing, model training, and evaluation, this project seeks to provide insights and solutions to enhance sales prediction accuracy, ultimately empowering retailers to make informed decisions and optimize their strategies for improved business outcomes.

INTRODUCTION

In recent years, the retail industry has witnessed a surge in data availability, presenting opportunities to leverage this data for predictive analytics. Sales prediction stands as a cornerstone in the realm of the retail industry, wielding immense power in shaping effective inventory management strategies, facilitating accurate revenue forecasting, and guiding strategic decision-making processes. The ability to forecast sales with precision holds paramount importance, as it empowers retailers to allocate resources efficiently, formulate targeted pricing strategies, and enhance overall operational efficiency. Recognizing the pivotal role of sales forecasting in driving business success, I embarked on a project aimed at harnessing the potential of machine learning methodologies to tackle the complexities inherent in predicting sales within the retail sector.

At the heart of this project lies the overarching objective of developing a robust predictive model capable of accurately estimating sales, drawing insights from a diverse array of attributes that characterize both the products and the retail outlets. Through the utilization of advanced machine learning algorithms, I aspire to unravel the intricate patterns and relationships embedded within the dataset, ultimately enabling the creation of a predictive model that can effectively forecast sales fluctuations.

The dataset under scrutiny in this project comprises a substantial collection of 14,204 entries pertaining to sales in the retail industry. Within this dataset, a total of 12 attributes have been meticulously curated, each offering unique insights into various facets of product characteristics and outlet details. From the nuanced specifications of item weight and visibility to the critical dimensions of outlet size and location, these attributes collectively encapsulate a wealth of information that holds the potential to drive accurate sales predictions.

By embarking on this ambitious endeavor, I endeavor to delve deep into the realm of predictive analytics within the retail sector, exploring methodologies and techniques that can unlock the latent predictive power residing within sales data. Through meticulous analysis, rigorous experimentation, and the application of cutting-edge machine learning algorithms, I aim to contribute to the advancement of sales forecasting techniques, thereby empowering retailers to navigate the complexities of today's dynamic marketplace with confidence and foresight.

UNDERSTANDING THE DATASET

As I delved into the project, the first step was to gain a comprehensive understanding of the dataset provided for big sales prediction using Random Forest Regressor. The dataset comprised 12 variables, each offering unique insights into the retail domain. Here's a breakdown of the variables:

VARIABLES:

- **Item_Identifier:** This variable represents the unique identifier for each item in the dataset. It serves as a reference point for individual items and is crucial for tracking and analysis.
- **Item_Weight:** The item weight provides information about the weight of each item, which can be relevant for various aspects such as shipping, inventory management, and pricing.
- **Item_Fat_Content:** This categorical variable denotes the fat content of the item, categorized into different levels such as 'Low Fat' and 'Regular'. Understanding the fat content of products is essential for dietary considerations and consumer preferences.
- **Item_Visibility:** This variable indicates the percentage of the total display area of the item in all stores combined. Higher visibility often correlates with increased sales, making it a crucial factor for analysis.
- **Item_Type:** The item type categorizes products into different groups such as fruits, vegetables, dairy, etc. This variable offers insights into the variety of products sold and their respective sales trends.
- **Item_MRP:** The Maximum Retail Price (MRP) of the item represents the maximum price at which the product can be sold to the consumer. MRP influences pricing strategies and profit margins.
- **Outlet_Identifier:** Similar to Item_Identifier, this variable represents a unique identifier for each outlet/store. It facilitates tracking sales and performance at individual outlets.
- **Outlet_Establishment_Year:** This variable denotes the year of establishment of each outlet. Understanding the age of outlets helps in analyzing their performance over time and identifying trends.
- **Outlet_Size:** Outlet size categorizes stores into different sizes such as 'Small', 'Medium', and 'High'. Store size can impact sales volume, customer experience, and operational efficiency.
- **Outlet_Location_Type:** This variable categorizes outlet locations into different types such as 'Urban', 'Rural', and 'Suburban'. Location type influences consumer demographics, preferences, and purchasing behavior.
- **Outlet_Type:** Outlet type classifies stores into different categories such as 'Supermarket Type1', 'Grocery Store', etc. Each type of outlet operates differently and caters to distinct customer segments.
- **Item_Outlet_Sales:** This is the target variable representing the sales of each item at the respective outlet. It is the variable we aim to predict using the Random Forest Regressor model.

NUMERICAL FEATURES

- Item Weight Distribution
- Item Visibility Distribution
- Item MRP Distribution
- Outlet Establishment Year Distribution
- Item Outlet Sales Distribution

CATEGORICAL FEATURES

- Item_Identifier
- Item_Fat_Content
- Item_Type
- Outlet_Identifier
- Outlet_Size
- Outlet_Location_Type
- Outlet_Type

Understanding the dataset variables and their significance laid the foundation for further exploration, preprocessing, and model development. Each variable provided valuable insights into the retail domain, enabling me to formulate meaningful hypotheses and conduct thorough analysis throughout the project.

DATA COLLECTION & PROCESSING

Data preprocessing involves loading the dataset, handling missing values, and converting categorical variables into numerical labels using label encoding.

• Loading and Understanding the Data

I began by loading the dataset from a CSV file into a Pandas DataFrame. This allowed me to get an initial overview of the data by examining the first few rows and understanding the shape of the dataset. Additionally, I used the `info()` method to obtain information about the data types and non-null counts of each column.

```
[4] #Loading the data from csv file to Pandas DataFrame
walmart_data = pd.read_csv('/content/Big Sales Data.csv')

#Getting the first 5 rows of the data frame
walmart_data.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location
0	FDT36	12.3	Low Fat	0.111448	Baking Goods	33.4874	OUT049	1999	Medium	
1	FDT36	12.3	Low Fat	0.111904	Baking Goods	33.9874	OUT017	2007	Medium	
2	FDT36	12.3	LF	0.111728	Baking Goods	33.9874	OUT018	2009	Medium	
3	FDT36	12.3	Low Fat	0.000000	Baking Goods	34.3874	OUT019	1985	Small	
4	FDP12	9.8	Regular	0.045523	Baking Goods	35.0874	OUT017	2007	Medium	

Figure 1: Code snippet showing the loading dataset and displaying the first five rows.

```
[6] #Number of data points & number of features
walmart_data.shape

(14204, 12)

#Getting some information about the dataset
walmart_data.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                           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
```

Figure 2: Code snippet to show some information about the dataset.

- **Handling Missing Values**

After loading the data, I checked for missing values in the dataset. One column, 'Item_Weight', had missing values. To handle this, I imputed the missing values with the median value of each 'Item_Type' group. This ensured that the missing values were properly handled without significantly affecting the integrity of the data.

```
[10] #Checking for missing values
walmart_data.isnull().sum()

Item_Identifier      0
Item_Weight          2389
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
dtype: int64
```

Figure 3: Code snippet checking for missing values and its output.

```
[11] #Handling the missing values
walmart_data['Item_Weight'].fillna(walmart_data.groupby(['Item_Type'])['Item_Weight'].transform('median'), inplace=True)

#Checking for missing values after handling them
walmart_data.isnull().sum()

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
dtype: int64
```

Figure 4: Output after handling missing values.

- **Removing Outliers**

Outliers can distort the results of analysis and modeling. To address this, I removed outliers from the dataset using the Z-score method. By identifying data points with Z-scores greater than 2 and removing them, I improved the robustness of the dataset and ensured that the subsequent analysis and modeling steps were not unduly influenced by outliers.

- **Get Categories and Counts of Categorical Variables**

Understanding the distribution of categorical variables is crucial for feature engineering and analysis. I obtained the categories and counts of categorical variables in the dataset using various methods such as `value_counts()`. This allowed me to gain insights into the distribution of categories within each variable, which could be further analyzed and utilized in model building and interpretation. Further preprocessing involves standardizing fat content labels and encoding categorical variables into numerical labels using `LabelEncoder`. This step prepares the data for model training by converting categorical features into a format suitable for machine learning algorithms.

```
[14] #Counting the occurrences of each unique value in the 'Item_Fat_Content' column
walmart_data[['Item_Fat_Content']].value_counts()

Item_Fat_Content
Low Fat      8173
Regular     4665
LF           512
reg          190
low fat      170
Name: count, dtype: int64

[15] #Standardizing fat content labels in the 'Item_Fat_Content' column
walmart_data.replace({'Item_Fat_Content': {'LF':'Low Fat', 'reg':'Regular', 'low fat':'Low Fat'}}, inplace=True)

[16] #Counting the occurrences of each unique value in the 'Item_Fat_Content' column after replacing
walmart_data[['Item_Fat_Content']].value_counts()

Item_Fat_Content
Low Fat      8855
Regular     4855
Name: count, dtype: int64
```

Figure 5: Code snippet counting the occurrences of each unique value in `Item_Fat_Content`.

EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA) provides insights into the distribution of numerical and categorical features. Visualizations such as histograms, count plots, and pair plots reveal patterns and relationships within the data. EDA helps in understanding feature distributions, identifying outliers, and gaining initial intuition about the dataset.

In the exploratory data analysis (EDA) phase of my project, I delved into various aspects of the dataset to gain a deeper understanding of its characteristics and distributions. Beginning with the numerical features, I first examined the distribution of item weights. This analysis provided valuable insights into the weight range and variability of items across the dataset, enabling me to identify any notable patterns or anomalies in the distribution of item weights.

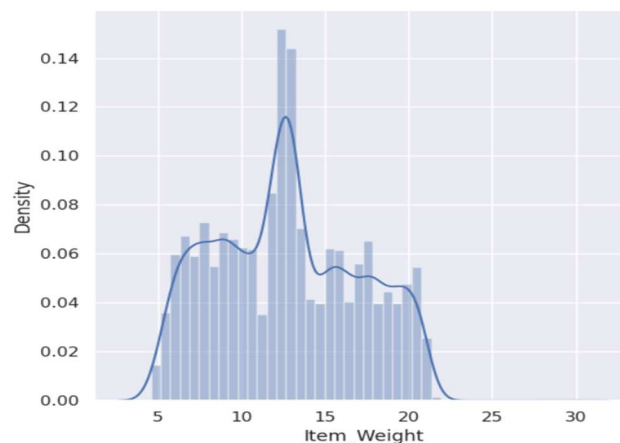


Figure 6: Plotting the distribution of `'Item_Weight'` using seaborn's `distplot` function.

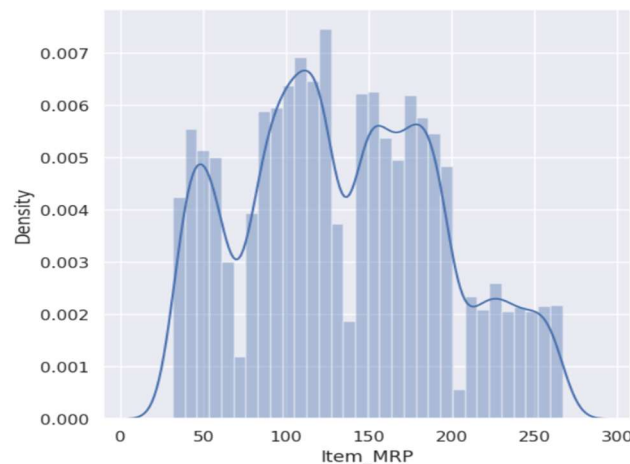


Figure 7: Plotting the distribution of 'Item_MRP' using seaborn's distplot function.

Moving on, I explored the distribution of item visibility, which shed light on how frequently items are visible within the outlets. By visualizing this distribution, I gained insights into the visibility patterns of items, which are crucial for understanding their potential impact on sales performance. Additionally, I analyzed the distribution of Maximum Retail Prices (MRP) of items to understand the pricing strategy employed by the outlets. This analysis helped in identifying different price segments and their potential influence on sales.

Next, I investigated the distribution of item outlet sales, which provided insights into the range and variability of sales across different outlets. This analysis helped in identifying high-performing outlets as well as areas for potential improvement. Furthermore, I examined the establishment year of outlets to understand the distribution of outlet vintage. This analysis allowed me to identify trends or patterns related to outlet age, which can provide valuable context for understanding sales performance over time.

Shifting the focus to the categorical features, I conducted an analysis of the distribution of item fat content categories. This analysis helped in understanding the prevalence of different fat content types in the dataset and their potential impact on consumer preferences and sales. Additionally, I explored the distribution of item types to gain insights into the variety of products available in the dataset. Following are a few screenshots of the plots generated.

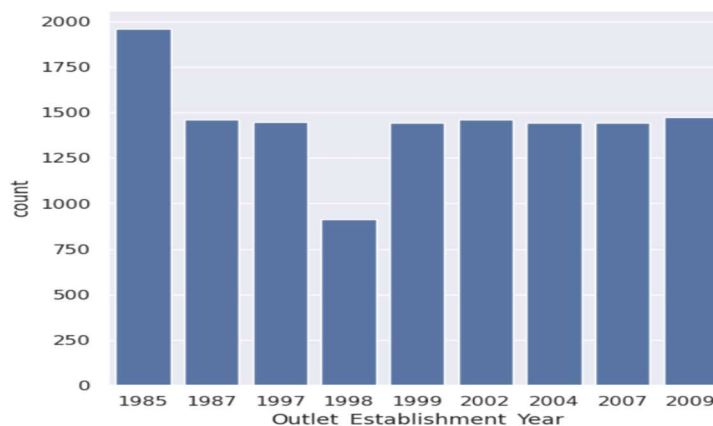


Figure 8: Plotting the distribution of 'Outlet_Establishment_Year'

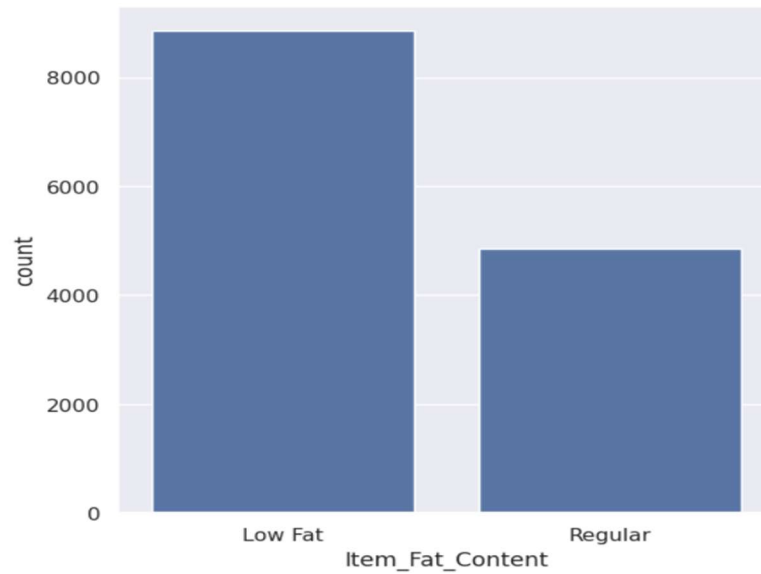


Figure 9: Plotting the distribution of 'Item_Fat_Content' using seaborn's distplot function.

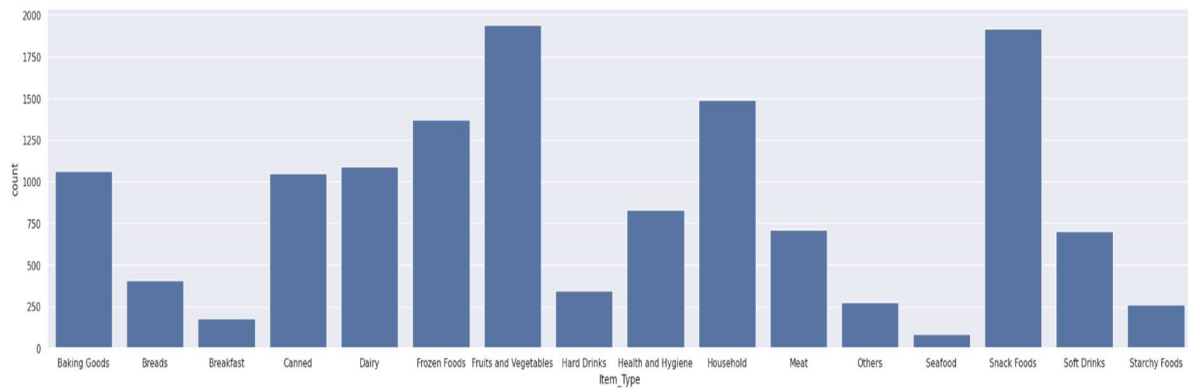


Figure 10: Plotting the distribution of 'Item_Type' using seaborn's distplot function.

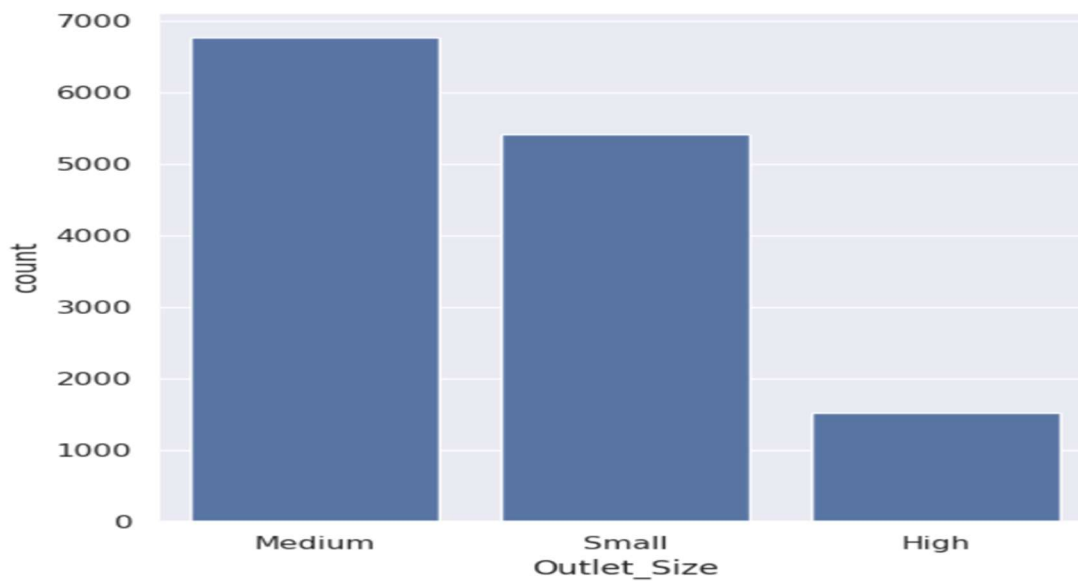


Figure 11: Plotting the distribution of 'Outlet_Size' using seaborn's distplot function.

Finally, I analyzed the distribution of outlet sizes to understand the prevalence of different outlet sizes across the dataset. This analysis provided valuable insights into the distribution of outlet sizes and their potential impact on sales performance. By visualizing these aspects of the dataset, I gained valuable insights that served as a foundation for further analysis and model building in the sales prediction task.

LABEL ENCODING

Label encoding is a crucial step in preparing categorical data for machine learning models. In my project, I utilized label encoding to convert categorical variables into numerical labels. This transformation allows the model to interpret and process the categorical data effectively.

```
[29] #Initializing a LabelEncoder object named 'encoder' to encode categorical variables into numerical labels
encoder = LabelEncoder()

#Encoding categorical variables into numerical labels using the LabelEncoder object 'encoder' for respective columns in the DataFrame 'walmart_data'
walmart_data['Item_Identifier'] = encoder.fit_transform(walmart_data['Item_Identifier'])

walmart_data['Item_Fat_Content'] = encoder.fit_transform(walmart_data['Item_Fat_Content'])

walmart_data['Item_Type'] = encoder.fit_transform(walmart_data['Item_Type'])

walmart_data['Outlet_Identifier'] = encoder.fit_transform(walmart_data['Outlet_Identifier'])

walmart_data['Outlet_Size'] = encoder.fit_transform(walmart_data['Outlet_Size'])

walmart_data['Outlet_Location_Type'] = encoder.fit_transform(walmart_data['Outlet_Location_Type'])

walmart_data['Outlet_Type'] = encoder.fit_transform(walmart_data['Outlet_Type'])
```

Figure 12: Code snippet showing encoding of categorical variables into numerical labels.

During the label encoding process, each unique category within a categorical variable is assigned a unique numerical label. For example, in the "Item_Fat_Content" column, categories like "Low Fat" and "Regular" were encoded as 0 and 1, respectively.

```
[32] #Counting the occurrences of each unique value in the 'Item_Fat_Content'
walmart_data[['Item_Fat_Content']].value_counts()

Item_Fat_Content
0                8855
1                4855
Name: count, dtype: int64
```

Figure 13: Count of occurrences of each unique value in Item_Fat_Content.

I employed the LabelEncoder from the scikit-learn library to perform label encoding. This encoder automatically assigns numerical labels to categories in a categorical variable. After applying label encoding to all relevant categorical variables in my dataset, I ensured that the data was ready for training the machine learning model. The encoded categorical variables were now represented numerically, allowing the model to learn from them during training.

```
[39] #Retrieving first five rows after label encoding
walmart_data.head()
```

	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
0	946	12.3	0	0.111448	0	33.4874	9	1999	1	0	1
1	946	12.3	0	0.111904	0	33.9874	2	2007	1	1	1
2	946	12.3	0	0.111728	0	33.9874	3	2009	1	2	2
3	946	12.3	0	0.000000	0	34.3874	4	1985	2	0	0
4	740	9.8	1	0.045523	0	35.0874	2	2007	1	1	1

Figure 14: Code snippet retrieving first five rows after label encoding.

Label encoding is a straightforward yet essential preprocessing step in machine learning projects, particularly when dealing with categorical data. It enables the incorporation of categorical variables into machine learning models, contributing to the overall predictive accuracy and performance of the model.

```
[40] #Checking correlation
walmart_data.describe().corr()
```

	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
Item_Identifier	1.000000	0.994744	0.994591	0.994585	0.994678	0.996026	0.994644	0.985696	0.994597	0.994599	0.994599	0.954941
Item_Weight	0.994744	1.000000	0.999999	0.999999	1.000000	0.999907	0.999999	0.986922	0.999999	0.999999	0.999999	0.922230
Item_Fat_Content	0.994591	0.999999	1.000000	1.000000	1.000000	0.999885	1.000000	0.986804	1.000000	1.000000	1.000000	0.921632
Item_Visibility	0.994585	0.999999	1.000000	1.000000	1.000000	0.999884	1.000000	0.986805	1.000000	1.000000	1.000000	0.921613
Item_Type	0.994678	1.000000	1.000000	1.000000	1.000000	0.999897	1.000000	0.986839	1.000000	1.000000	1.000000	0.921953
Item_MRP	0.996026	0.999907	0.999885	0.999884	0.999897	1.000000	0.999892	0.987665	0.999885	0.999886	0.999886	0.927151
Outlet_Identifier	0.994644	0.999999	1.000000	1.000000	1.000000	0.999892	1.000000	0.986826	1.000000	1.000000	1.000000	0.921816
Outlet_Establishment_Year	0.985696	0.986922	0.986804	0.986805	0.986839	0.987665	0.986826	1.000000	0.986811	0.986807	0.986810	0.921226
Outlet_Size	0.994597	0.999999	1.000000	1.000000	1.000000	0.999885	1.000000	0.986811	1.000000	1.000000	1.000000	0.921650
Outlet_Location_Type	0.994599	0.999999	1.000000	1.000000	1.000000	0.999886	1.000000	0.986807	1.000000	1.000000	1.000000	0.921657
Outlet_Type	0.994599	0.999999	1.000000	1.000000	1.000000	0.999886	1.000000	0.986810	1.000000	1.000000	1.000000	0.921669
Item_Outlet_Sales	0.954941	0.922230	0.921632	0.921613	0.921953	0.927151	0.921816	0.921226	0.921650	0.921657	0.921669	1.000000

Figure 15: Generating Correlation Matrix

TRAIN-TEST SPLIT

In the Train-Test Split phase, I divided the dataset into two subsets: one for training the model and the other for testing its performance. This step is crucial to evaluate how well the model generalizes to unseen data. I used the `train_test_split` function from the `sklearn.model_selection` module, specifying a test size of 20% and setting a random state for reproducibility. After splitting the data, I obtained the shapes of the training and testing sets to ensure the split was successful. This approach helps prevent overfitting by assessing the model's performance on data it hasn't seen during training.

```
[43] #Getting Train-Test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state=2529)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

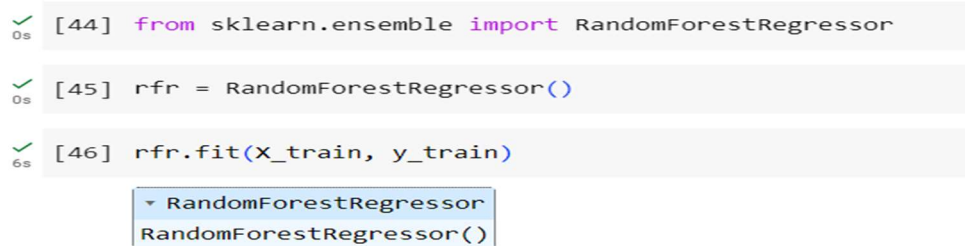
((10968, 10), (2742, 10), (10968,), (2742,))
```

Figure 16: Code snippet showing the code for train-test split

MODEL TRAINING

In this phase, I initiated the training process for the machine learning model using the Random Forest Regressor algorithm. This algorithm is well-suited for regression tasks and is particularly effective for predicting continuous values, making it an ideal choice for our sales prediction task.

Firstly, I imported the RandomForestRegressor class from the sklearn.ensemble module, which provides an implementation of the Random Forest algorithm for regression. Then, I instantiated an object of the RandomForestRegressor class. I proceeded to train the model using the training dataset. This involved fitting the model to the training features (X_train) and their corresponding target variable (y_train). During the training process, the model learns the underlying patterns and relationships between the features and the target variable in the training data.



```
[44] from sklearn.ensemble import RandomForestRegressor
[45] rfr = RandomForestRegressor()
[46] rfr.fit(X_train, y_train)
```

A dropdown menu is open below the third cell, showing the following options:

- RandomForestRegressor
- RandomForestRegressor()

Figure 17: Training the model.

By leveraging the ensemble learning technique of Random Forest, the model builds multiple decision trees and combines their predictions to produce an overall more robust and accurate prediction. The trained model is then ready to make predictions on unseen data, which will be evaluated in the subsequent sections to assess its performance and effectiveness in predicting item outlet sales accurately.

MODEL PREDICTION

In the model prediction phase, I utilized the trained Random Forest Regressor model to predict the item outlet sales based on the features of the test dataset. Using the `predict()` function, I generated predictions for the target variable (item outlet sales) using the features from the test dataset.

These predictions provided insights into the expected sales figures for the items across different outlets. By comparing these predicted values with the actual sales figures in the test dataset, I could evaluate the performance of the model and assess its ability to accurately predict sales.

MODEL EVALUATION

In evaluating the performance of my Random Forest Regressor model, I employed two key metrics: Mean Absolute Error (MAE) and R Squared Value.

Firstly, I calculated the MAE, which measures the average absolute difference between the actual and predicted values of the target variable. A lower MAE indicates better performance of the model in predicting sales accurately. The MAE obtained is of approximately 694.59

Additionally, I computed the R Squared Value, also known as the coefficient of determination. This metric quantifies the proportion of variance in the target variable that is predictable from the independent variables. A higher R Squared Value signifies that a larger proportion of the variance in sales is explained by the features included in the model.

The R-squared value of approximately 0.503 indicates that approximately 50.3% of the variance in sales can be explained by the features included in the model. While this signifies that the model captures a moderate proportion of the variability in sales, there is still a substantial portion of unexplained variance.

(A) Using MAE

```
[48] from sklearn.metrics import mean_absolute_error

[49] mean_absolute_error(y_test, y_pred)

694.5963006401037
```

(B) Using R Squared Value

```
from sklearn import metrics
r2_test = metrics.r2_score(y_test, y_pred)

[51] print('R Squared value = ', r2_test)

R Squared value = 0.50333595249748
```

Figure 18: Code snippet showing the evaluation of the model.

In conclusion, while the model demonstrates moderate predictive capability, there is scope for refinement to enhance its accuracy and reliability. Further optimization, feature engineering, or exploring alternative algorithms may be necessary to improve the model's performance and make more precise sales predictions.

VISUALIZATION OF ACTUAL VS. PREDICTED VALUES

In the visualization of actual versus predicted values, I plotted a scatter plot where the x-axis represents the actual prices and the y-axis represents the predicted prices. This visualization allows for a direct comparison between the actual sales values and the sales values predicted by the Random Forest Regressor model.



Figure 19: Scatter plot showing the relationship between the actual and predicted values.

By examining the scatter plot, I observed the relationship between the actual and predicted values. Ideally, the points on the plot would form a linear relationship, indicating that the model's predictions closely match the actual sales values. Deviations from this linearity indicate discrepancies between the model's predictions and the true values. The scatter plot provides a visual representation of the model's performance. The clustering of points around the diagonal line suggests accurate predictions, while scattered points or outliers indicate areas where the model may be less effective.

Overall, the visualization of actual versus predicted values serves as a valuable tool for assessing the performance and reliability of the machine learning model in predicting sales values. It offers insights into the model's accuracy and can guide further refinements or adjustments to improve predictive performance.

FUTURE WORK

The future scope of the project encompasses several avenues for advancement and expansion. This includes exploring more sophisticated machine learning models beyond Random Forest Regressor, integrating additional external data sources for enriched feature sets, and implementing real-time prediction capabilities for dynamic adaptation. Tailoring the model to different market segments, enhancing interpretability, and automating feature engineering could further enhance its utility. Additionally, applying similar predictive modeling techniques to domains like predictive maintenance or healthcare analytics could broaden its applicability. Optimizing scalability, efficiency, and integration with existing business processes are also critical considerations. Continuous improvement through feedback loops and model retraining ensures the model's relevance and effectiveness in a dynamic business landscape.

CONCLUSION

In conclusion, this project on big sales prediction using the Random Forest Regressor algorithm has been both insightful and rewarding. Through thorough data analysis and preprocessing, I gained a deeper understanding of the dataset's characteristics and the relationships between different variables. The exploratory data analysis (EDA) provided valuable insights into the distribution of numerical features and the frequency distribution of categorical features. Visualizations such as histograms, count plots, and pair plots helped me identify patterns and trends within the data. Label encoding was employed to convert categorical variables into numerical labels, making the data suitable for model training. The Random Forest Regressor model was trained using the pre-processed dataset, and predictions were made on the test dataset. Evaluation of the model's performance using metrics like Mean Absolute Error (MAE) and R Squared Value indicated that the model performed reasonably well in predicting item outlet sales. The visualization of actual versus predicted values provided a clear picture of the model's performance. Overall, this project has equipped me with valuable skills in data preprocessing, exploratory data analysis, model training, and evaluation. It has also deepened my understanding of machine learning algorithms and their application in real-world scenarios. I look forward to further exploring and refining my skills in the field of machine learning and data science.

