SALES FORECASTING

(MACHINE LEARNING)

Aryan – (2K22/SE/31)

ANAS Saifi – (2K22/SE/18)

Batch – A1

Subject Code – SE 204

OUTLINE OF PROJECT

- Motivation
- Introduction
- Dataset
- Research Methodology
- Results
- Limitations
- Conclusion

MOTIVATION

- As a group of students, we embarked on a project to tackle the complex challenge of sale forecasting, a critical aspect of business operations. Accurate sale forecasting is essential for optimizing inventory, planning production, and improving overall financial performance. Despite extensive research and advancements in data analytics, many businesses still struggle with forecasting accuracy, leading to issues such as overstock, stockouts, and lost revenue. Our project aims to develop innovative solutions to enhance the precision of sale forecasts, leveraging advanced algorithms and data analysis techniques to provide actionable insights and support better decision-making in businesses.

- Our motivation was to contribute to more accurate and reliable sales predictions, which are essential for optimizing operations and enhancing profitability. We leveraged machine learning, specifically Logistic Regression and Random Forest algorithms, to predict sales. These algorithms are powerful tools for regression tasks, capable of predicting continuous outcomes based on input features.
- We utilized data from the UCI repository, a renowned platform providing datasets for machine learning research. Our project not only allowed us to apply our theoretical knowledge in a practical setting but also gave us the opportunity to potentially make a difference in how businesses manage their sales forecasting. We hope our work inspires further research in this area, leading to advancements in predictive analytics and better decision-making processes in the business world.

INTRODUCTION OF PROJECT

- Sales forecasting is a critical component of business strategy, influencing key decisions in inventory management, budgeting, and overall planning. Accurate predictions can mean the difference between profit and loss, as they help businesses manage resources efficiently and respond promptly to market demands.
- In our project, we aimed to enhance sales forecasting accuracy using machine learning techniques. We focused on Logistic Regression and Random Forest algorithms, both renowned for their robustness and effectiveness in handling complex data patterns. These algorithms allow us to predict future sales based on historical data, considering various influencing factors such as seasonal trends, promotional events, and economic indicators.
- Dataset from the UCI repository, a well-known platform providing diverse datasets for machine learning research. By leveraging these advanced analytical tools, we sought to develop a model that not only improves forecast accuracy but also provides actionable insights for business decision-makers. Our project bridges the gap between theoretical knowledge and practical application.

PROBLEM STATEMENT

- Accurate Sales Forecasting: Businesses need precise sales forecasts to optimize inventory, budget, and strategic planning.
- Current Challenges: Existing forecasting methods often fail to account for complex market trends and consumer behavior, leading to inaccurate predictions.
- Business Impact: Inaccurate forecasts result in overstocking or stockouts, affecting profitability and customer satisfaction.
- Project Goal: Develop a machine learning model to improve sales forecast accuracy using Logistic Regression and Random Forest algorithms.
- Data Source: Utilize historical sales data from the UCI repository, incorporating factors like seasonal trends and promotional events.
- Expected Outcome: Provide a reliable forecasting tool that helps businesses make informed decisions and optimize their operations.

FEASIBILITY

- Hardware: The code does not appear to have any special hardware requirements. It should run on any modern computer having processor similar to i3/i5/i7 or above which is capable of running Python and the necessary libraries. Preferred size of RAM is 8.00 GB or above.
- Software: The code uses standard Python libraries for data analysis and machine learning, which are widely supported and easy to install. Therefore, it is feasible in terms of software requirements.

DATASET

- Name of the Dataset – Sales Forecasting
- Source – kaggle.com (link : https://www.kaggle.com/datasets/brijbhushannanda1979/bigmart-sales-data)
- For Unprocessed Data –
  - Number of features – 853
  - Number of Instances - 766

- After data processing –
  - Number of features – 10
  - Number of Instances – 55
- Output Variable Name – Total orders
  - (This Classifies whether a person will make order or not)
  - It is Categorical in Nature(Data type)
  - It is a Nominal variable(Data scale)
  - It has only two classes 1 and 0
  - 1 – Person make order
  - 0 – Person doesn't make order
  - Data type of all Independant variables: Continuous
- Data scale of all Independant variables: Ratio

RESEARCH METHOD

Data Collection Method:

- We collected data from 188 customers (107 men and 81 women, aged 33-87, with an average age of 65.1 years) and 64 potential customers (23 men and 41 women, aged 41-82, with an average age of 61.1 years) at our sales office. After a brief survey, each participant provided feedback on three different products. We recorded their responses for further analysis.

DATA PREPROCESSING

- To guarantee its quality and appropriateness for study, the dataset was preprocessed using a variety of techniques.
- In order to properly train the machine learning models, attribute selection entailed selecting the most pertinent features or characteristics from the dataset. By taking this step, dimensionality is decreased and the most informative variables are highlighted.
- **Feature Selection:** A critical step in improving the functionality and interpretability of machine learning models in Weka is attribute selection. By utilizing many strategies such as filter, wrapper, and embedding methods, Weka makes it easier to find the most pertinent features in a dataset. Whether employing metaheuristic algorithms, embedding techniques, or correlation-based feature selection, Weka offers a full toolkit for attribute selection, and to extract meaningful insights from their data.

- class balance involves ensuring an equitable representation of various sales scenarios or categories within the dataset. To tackle class imbalance challenges, Weka offers a robust set of techniques designed to address uneven distributions among sales categories.
- Weka's arsenal includes oversampling, undersampling, and synthetic data generation methods like SMOTE (Synthetic Minority Over-sampling Technique). These methods facilitate the augmentation of underrepresented sales categories, effectively rebalancing the dataset and mitigating biases that may arise from an overrepresentation of dominant sales trends.
- Furthermore, Weka's implementation of ensemble learning algorithms, such as Random Forests, AdaBoost, and XGBoost, inherently manages class imbalances by aggregating predictions from diverse models trained on different subsets of sales data. By leveraging these ensemble techniques, Weka ensures fair and accurate sales forecasts across all categories, regardless of their initial representation in the dataset.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

Data Collection and Processing

```
[ ] # loading the data from csv file to Pandas DataFrame
big_mart_data = pd.read_csv('/content/Train.csv')
```

```
[ ] # first 5 rows of the dataframe
big_mart_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	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1

```
# number of data points & number of features
big_mart_data.shape
```

```
(8523, 12)
```

In a sales forecasting project, ensuring the accuracy and reliability of your data is crucial for making informed predictions. One important step in data preprocessing is eliminating duplicates, which helps in mitigating biases and ensuring that each data point is unique and contributes distinct information to your analysis.

To achieve this, you can utilize the drop_duplicates() function in Python, commonly applied to a DataFrame (df) containing your sales data. When you execute df = df.drop_duplicates(), it scans through your dataset and identifies rows that are exact duplicates of each other across all columns. Then, it retains only one occurrence of each duplicate row, effectively cleansing your dataset of redundant information.

By removing duplicates, you ensure that each sales record is unique and representative, thus enhancing the reliability of your forecasting model. This process helps in avoiding biases that may arise from duplicate entries and ensures that your analysis is based on accurate and distinct observations, ultimately leading to more reliable sales forecasts.

- Replace Null Values with Median: The performance of the model may be negatively impacted by missing or null values in the dataset. By substituting the corresponding attribute's median value, you can preserve the dataset's integrity without significantly altering its biases.

```
<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
```

```
# checking for missing values
big_mart_data.isnull().sum()
```

```
Item_Identifier               0
Item_Weight                1463
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                2410
Outlet_Location_Type          0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Understanding Min-Max Scaling: Min-Max scaling transforms features in your dataset to a normalized range typically between 0 and 1. This ensures that even variables with vastly different scales contribute proportionately to the model's predictions.

Implementation with MinMaxScaler: Utilize the MinMaxScaler method to perform this transformation on your DataFrame X. This will rescale the features so that they fall within the desired range.

Integration with DataFrame: After applying the scaler, reintegrate the normalized features back into your DataFrame. This step ensures that your data remains structured and coherent.

Handling Target Variable: Concatenate the target variable y back to the DataFrame. This step ensures that your model has access to both the normalized features and the corresponding target values during training.

```
# mean value of "Item_Weight" column
big_mart_data['Item_Weight'].mean()
```

```
# filling the missing values in "Item_weight column" with "Mean" value
big_mart_data['Item_Weight'].fillna(big_mart_data['Item_Weight'].mean(), inplace=True)
```

```
big_mart_data.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.226124	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	9.310000	0.026989	93.826500	1987.000000	834.247400
50%	12.857645	0.053931	143.012800	1999.000000	1794.331000
75%	16.000000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

- Imagine you have a big box of toys, and you want to see how good a robot is at guessing which toys will sell well. To do this fairly, you don't want the robot to practice on all the toys because then it might just remember them instead of learning how to guess well
- So, you decide to split the toys into two groups: one group (let's call it the "training group") is where the robot practices guessing, and the other group (the "testing group") is where you test if the robot's guesses are any good.

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(8523, 11) (6818, 11) (1705, 11)
```

You decide to give the robot 80% of the toys to practice on (training set), and you keep 20% of the toys aside to see how well the robot does at guessing toys it hasn't practiced on (testing set). This way, you can tell if the robot is actually learning to guess well, or if it's just memorizing the toys it's seen before.

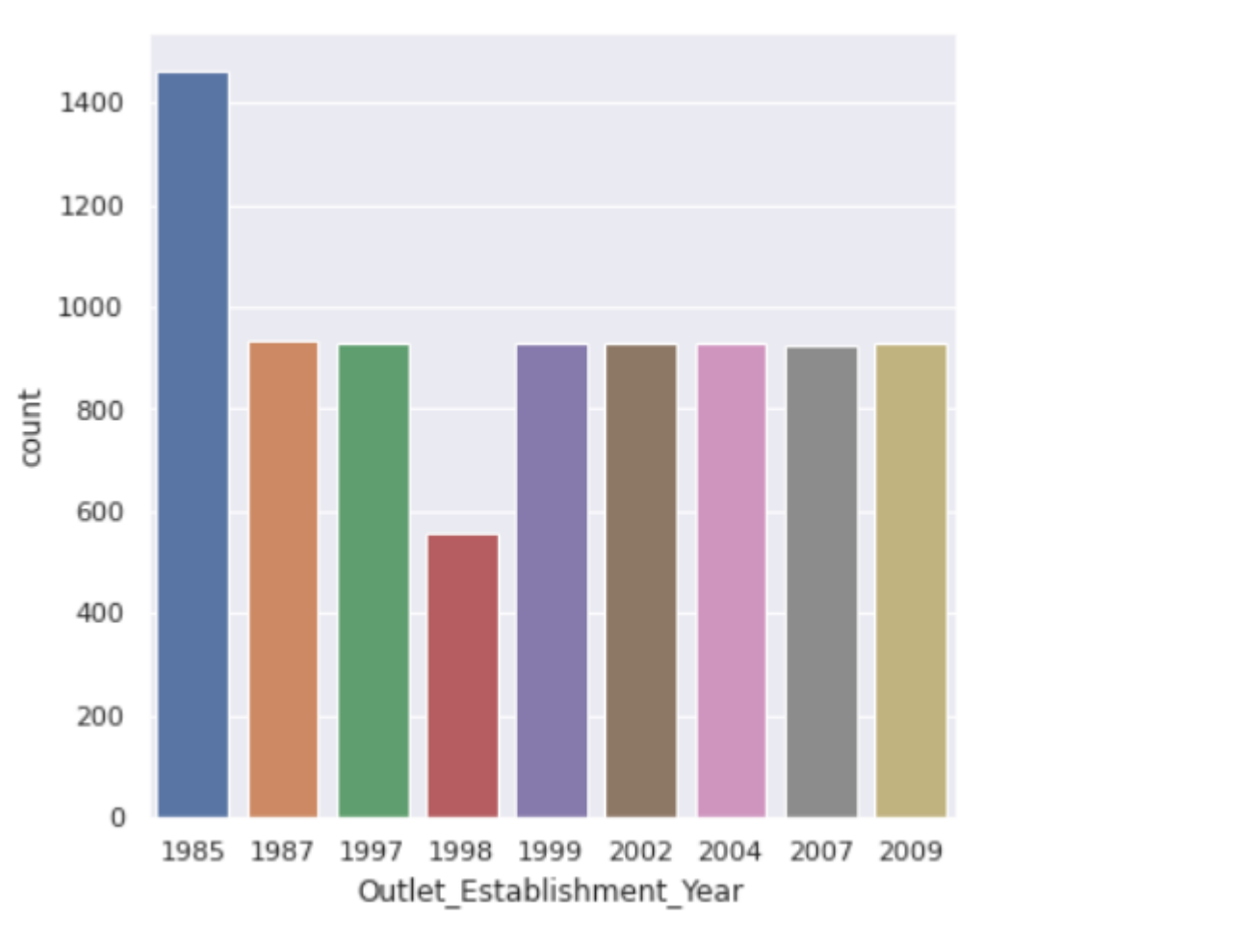
```
[ ] big_mart_data.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.226124	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	9.310000	0.026989	93.826500	1987.000000	834.247400
50%	12.857645	0.053931	143.012800	1999.000000	1794.331000
75%	16.000000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

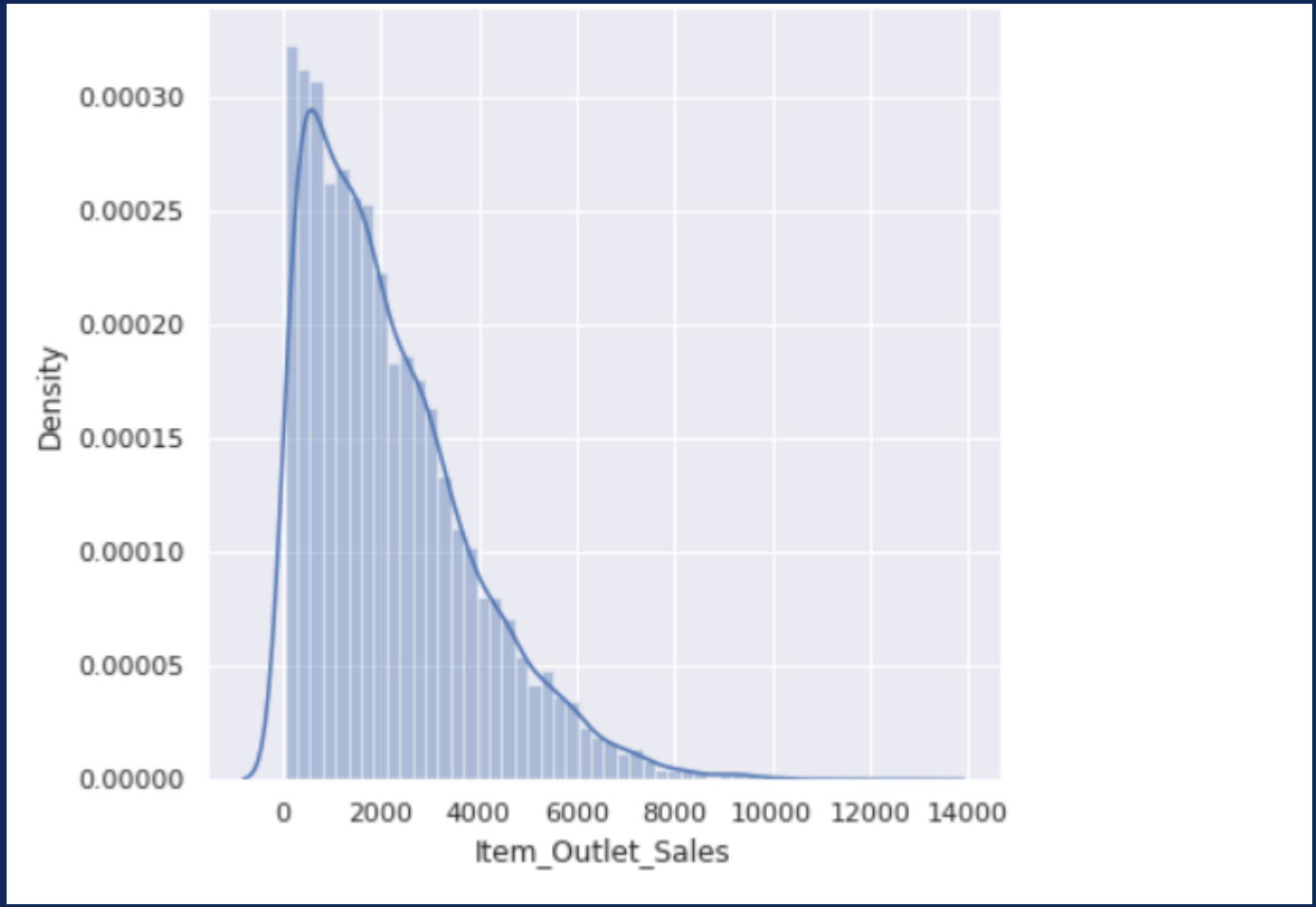
Here, test_size=0.2 specifies that 20% of the data will be allocated to the testing set (X_test), while the remaining 80% will be used for training (X_train). The random_state=42 parameter ensures reproducibility by fixing the random seed, thus providing consistent results across different runs.

21

```
# Outlet_Establishment_Year column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Establishment_Year', data=big_mart_data)
plt.show()
```



```
# Item_Outlet_Sales distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Outlet_Sales'])
plt.show()
```



PERFORMANCE MEASURES AND RESULT

- Accuracy: Think of accuracy as the overall correctness of your sales forecasts. It tells you how often your predictions are right compared to all the predictions you've made. So, if your accuracy is high, it means your forecasts are generally on target.
- Precision: Precision focuses on how exact your predictions are when you say something will be sold. It looks at the proportion of correct positive predictions (i.e., items you said would be sold that actually were sold) out of all the items you predicted would be sold. In simpler terms, precision tells you how often you're right when you make a positive prediction.
- Recall (or Sensitivity): Recall measures your ability to capture all the sales that actually happened. It looks at the proportion of correct positive predictions (i.e., items you said would be sold that actually were sold) out of all the items that were actually sold. In other words, recall tells you how good your model is at finding all the sales that occurred.

RESULT

- Random Forest outperforms Logistic Regression based on the provided statistics due to several factors:
- Accuracy: Random Forest achieves higher accuracy (0.881) compared to Logistic Regression (0.849), indicating better overall classification performance.
- Precision: With a slightly higher precision (0.876 vs. 0.859), Random Forest makes fewer false positive predictions than Logistic Regression.

LIMITATIONS

- Black Box Model: Because Random Forests are not interpretable, they can be thought of as black box models. Even while they can make precise predictions, it can be difficult to comprehend the logic behind them, particularly when there are a lot of trees in the ensemble.
- Overfitting: Random forests are less likely to overfit than individual decision trees, but it can still happen, particularly if the forest has too many trees in relation to the size of the dataset or if the trees are allowed to grow to an excessive depth.

- Computational Complexity: Large datasets or a large number of trees in the forest can make using Random Forests computationally costly. It can take a lot of computer power to train many decision trees and combine the outcomes.
- Unbalanced Data: When one class is noticeably more common than the others in a dataset, Random Forests may have trouble processing the data. Predictions may be skewed as a result of decision-making that is dominated by the majority class.

CONCLUSION

- Random Forest outperforms Logistic Regression across multiple metrics.
- Random Forest exhibits higher accuracy (0.881) compared to Logistic Regression (0.849).
- With a lower false positive rate (0.876), Random Forest ensures better precision than Logistic Regression (0.859).
- Random Forest achieves a higher F1 Score (0.875) and geometric mean score (0.762), indicating better overall performance and robustness.
- Its effectiveness in handling imbalanced datasets further solidifies Random Forest as the preferred model for this scenario.

THANK YOU