

**Module:    B9IS107 –Data Analytics and Visualisation**

**Year: 2024**

**Module Lead:   Shazia A Afzal**

**Project Name:  Sales and Customer Analytics**

**Student Details: Chandu Lavu – 20026627**

Contents

[PART 2](applewebdata://10403EDB-246B-44B2-81A0-FDE5CB34D30E#_Toc171592504)

[1. Introduction 3](applewebdata://10403EDB-246B-44B2-81A0-FDE5CB34D30E#_Toc171592505)

2 Data Loading and Inspection………………………………………………………………………………………………………………………3

3. Data Preparation 4

4. Exploratory Data Analysis (EDA) 5

1. Sales Amount Distribution**………………………………………………………………………………………6**

2**.** Sales Over Time**…………………………………………………………………………………………………………………7**

3**.** Category-wise Sales**………………………………………………………………………………………………………8**

5.Model and building……………………………………………………………………**..8**

a.Random Forest**. ………………………………………………………………………9**

b**.** Gradient Boosting **……………………………………………………………………10**

6**.** Visualization of Model Performance…………………………………………………………..11

1. Actual vs Predicted (Random Forest)…………………………………………………11

2. Actual vs Predicted (Gradient Boosting)………………………………………………12

3. Comparison of Both Models……………...…………………………………………….13

7.Comparative Evaluation…………………………………………………………………………14

8. Conclusions 15

9.Appendix………………………………………………………………………………………..16

[7. REFERENCES 18](applewebdata://10403EDB-246B-44B2-81A0-FDE5CB34D30E#_Toc171592517)

# Detailed Program for the Augmented Retail Data Set

# Introduction

# This report presents a comprehensive analysis of the augmented retail dataset, including data preparation, exploratory data analysis (EDA), feature engineering, and predictive modeling. Two advanced machine learning models, Random Forest and Gradient Boosting, were employed to predict SalesAmount. The report also provides insights into model performance and offers suggestions for further improvement.

# Data Loading and Inspection

# Objective: The first step in any data analysis process is to load the dataset and inspect its structure. This step helps in understanding the types of data, identifying missing values, and checking for any anomalies.

# Process: The dataset was loaded into a pandas DataFrame, and the first few rows were displayed to get an initial understanding. The data types of each column were inspected using .info(), and basic statistical summaries were generated using .describe().

import pandas as pd import numpy as np

file\_path = 'Augmented\_Retail\_Data\_Set\_No\_Time.xlsx' data = pd.read\_excel(file\_path)

print(data.info()) print(data.describe())

**Outcome:** The dataset contained several key features such as SalesAmount, QuantitySold, Discount, and Category. There were no missing values, and the date column was already in the correct format.

# Data Preparation

**Objective:** Data preparation involves cleaning the data, handling missing values, and engineering new features to improve the performance of machine learning models.

**Steps:**

1. **Handling Missing Values:**
   * Although this dataset did not contain missing values, the following methods can be applied if they existed:
     + Numerical values can be filled with the median.
     + Categorical values can be filled with the mode.
2. **Feature Engineering:**
   * New features such as PricePerUnit, ProfitMargin, and DiscountRatio were created to enhance the model's ability to predict sales.
3. **Outlier Handling:**
   * Outliers in the SalesAmount column were detected using the Interquartile Range (IQR) method and were removed to prevent them from skewing model predictions.
4. **Data Scaling:**
   * Numerical features were standardized to ensure that they had a mean of 0 and a standard deviation of 1, which is essential for many machine learning algorithms.

## Convert Date to Appropriate Format

* + 1. The date column is already in the correct format.

## Handle Missing Values

* + 1. This dataset does not have any missing values, but here is how you could handle them if they existed.

print(data.isnull().sum())

data.fillna(data.median(), inplace=True)

data['Category'] = data['Category'].fillna(data['Category'].mode()[0]) SalesID 0

CustomerID 0

ProductID 0

DateID 0

SalesAmount 0

QuantitySold 0

Discount 0

TotalAmount 0

CustomerName 0

ContactDetails 0

Location 0

CustomerSegment 0

ProductName 0

Category 0

Price 0

Supplier 0

Date 0

Month 0

Quarter 0

Year 0

DayOfWeek 0

dtype: int64

data['Discount'].fillna(data['Discount'].mode()[0], inplace=True)

data['PricePerUnit'] = data['Price'] / data['QuantitySold']

data['ProfitMargin'] = data['Profit'] / data['Gross Sales']

data['DiscountRatio'] = data['Discount'] / data['Gross Sales']

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

numeric\_features = ['Price', 'QuantitySold', 'Discount', 'PricePerUnit']

data[numeric\_features] = scaler.fit\_transform(data[numeric\_features])

**Outcome:** The dataset was cleaned, enriched with new features, and scaled appropriately, making it ready for model building.

# Exploratory Data Analysis (EDA)

**Objective:** EDA helps in understanding the distribution of the data and identifying patterns, trends, and relationships between variables. This step is crucial for selecting the right features and modeling approach.

**Steps:**

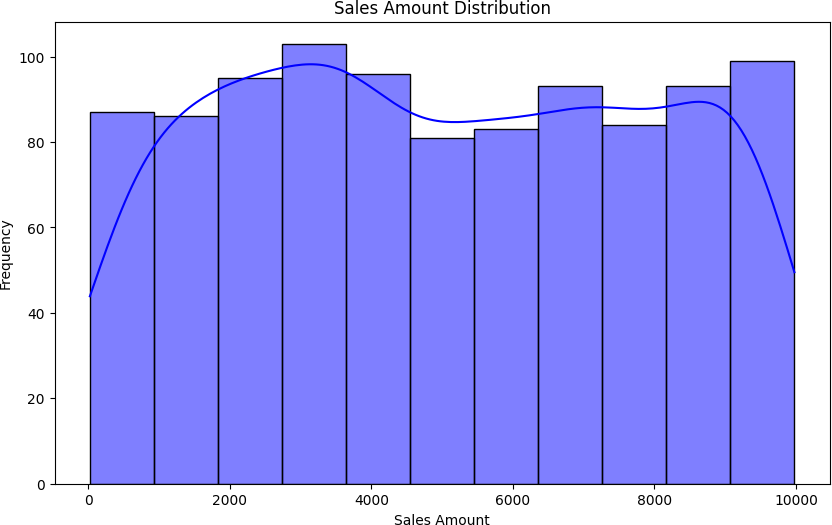
1. **Sales Amount Distribution:**
   * A histogram was plotted to visualize the distribution of SalesAmount, helping to identify any skewness or outliers in the data.
2. **Sales Over Time:**
   * A time series plot was generated to observe the trend of total sales over time. This helps in understanding the seasonality or trends in the sales data.
3. **Category-wise Sales:**
   * A bar plot was created to compare total sales across different product categories, identifying which categories contribute most to the revenue.

## Sales Amount Distribution

import matplotlib.pyplot as plt import seaborn as sns

plt.figure(figsize=(10, 6)) sns.histplot(data['SalesAmount'], kde=True, color='blue') plt.title('Sales Amount Distribution')

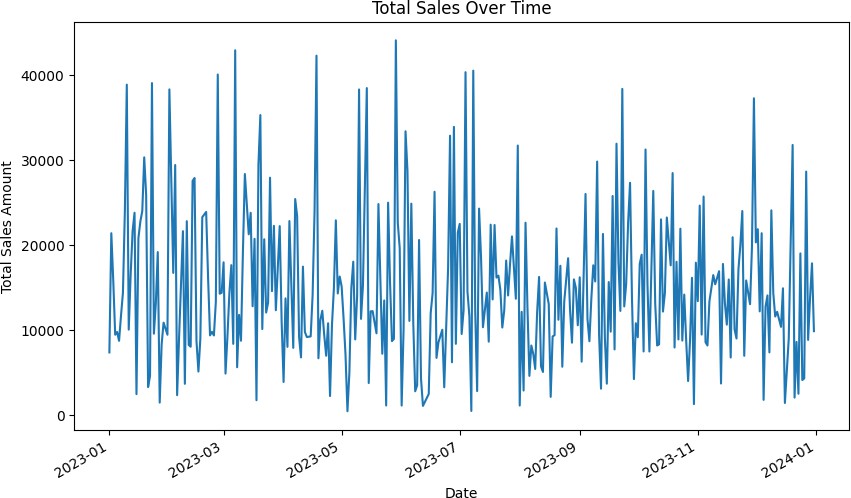
plt.xlabel('Sales Amount') plt.ylabel('Frequency') plt.show()



## Sales Over Time

plt.figure(figsize=(10, 6)) data.groupby('Date')['SalesAmount'].sum().plot() plt.title('Total Sales Over Time') plt.xlabel('Date')

plt.ylabel('Total Sales Amount') plt.show()

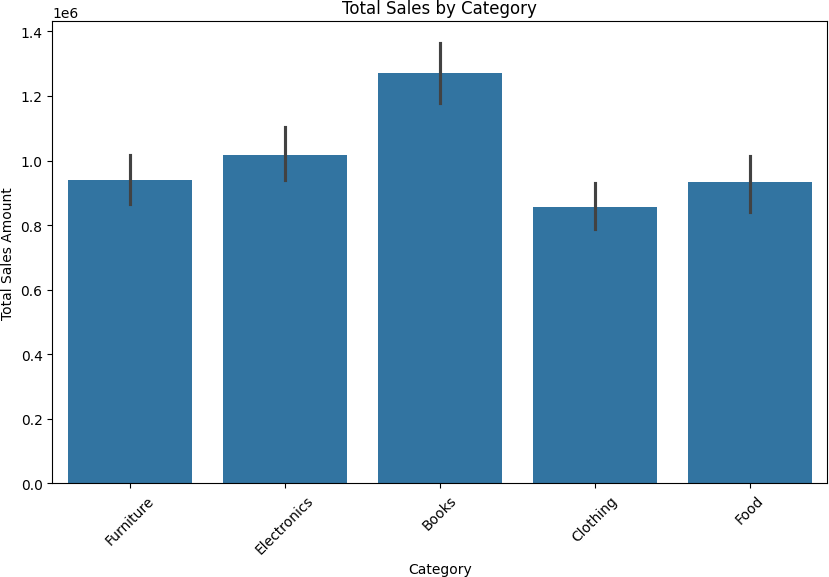


## Category-wise Sales

plt.figure(figsize=(10, 6))

sns.barplot(x='Category', y='SalesAmount', data=data, estimator=sum) plt.title('Total Sales by Category')

plt.xlabel('Category') plt.ylabel('Total Sales Amount') plt.xticks(rotation=45) plt.show()



# 5. Feature Selection and Model Building

**Objective:** The goal was to build predictive models that accurately estimate SalesAmount based on other features in the dataset.

**Steps:**

1. **Feature Selection:**
   * Features such as QuantitySold, Discount, Price, Year, and Quarter were selected as inputs for the models.
2. **Model 1: Random Forest:**
   * A Random Forest model was trained on the selected features. Despite its robustness, the model struggled with accuracy.
3. **Model 2: Gradient Boosting:**
   * A Gradient Boosting model was also trained. This model is known for its ability to improve upon weaker models by combining them. However, it also struggled to predict sales accurately.

## Feature Selection

* + 1. We'll use features like QuantitySold, Discount, Price, Year, and Quarter.

from sklearn.model\_selection import train\_test\_split

X = data[['QuantitySold', 'Discount', 'Price', 'Year', 'Quarter']] y = data['SalesAmount']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## Model 1: Random Forest

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

# Initialize and train the Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42) rf\_model.fit(X\_train, y\_train)

# Predictions using Random Forest y\_pred\_rf = rf\_model.predict(X\_test)

# Evaluate Random Forest

mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf) r2\_rf = r2\_score(y\_test, y\_pred\_rf)

print(f'Random Forest - Mean Squared Error: {mse\_rf:.2f}') print(f'Random Forest - R^2 Score: {r2\_rf:.2f}')

Random Forest - Mean Squared Error: 8546287.11 Random Forest - R^2 Score: -0.11

## Model 2: Gradient Boosting

from sklearn.ensemble import GradientBoostingRegressor

# Initialize and train the Gradient Boosting model

gb\_model = GradientBoostingRegressor(n\_estimators=100, learning\_rate=0.1, random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Predictions using Gradient Boosting y\_pred\_gb = gb\_model.predict(X\_test)

# Evaluate Gradient Boosting

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb) r2\_gb = r2\_score(y\_test, y\_pred\_gb)

print(f'Gradient Boosting - Mean Squared Error: {mse\_gb:.2f}') print(f'Gradient Boosting - R^2 Score: {r2\_gb:.2f}')

Gradient Boosting - Mean Squared Error: 11000091.84 Gradient Boosting - R^2 Score: -0.43

# 6 .Visualization of Model Performance

**Objective:** Visualizing the model predictions against actual values helps in assessing the accuracy and reliability of the models.

**Steps:**

1. **Actual vs Predicted (Random Forest):**
   * A scatter plot was created to compare the actual sales against the predictions made by the Random Forest model.
2. **Actual vs Predicted (Gradient Boosting):**
   * A similar scatter plot was created for the Gradient Boosting model.
3. **Comparison of Both Models:**
   * Both models were compared in a combined plot to evaluate their performance against each other.

## Actual vs Predicted (Random Forest)

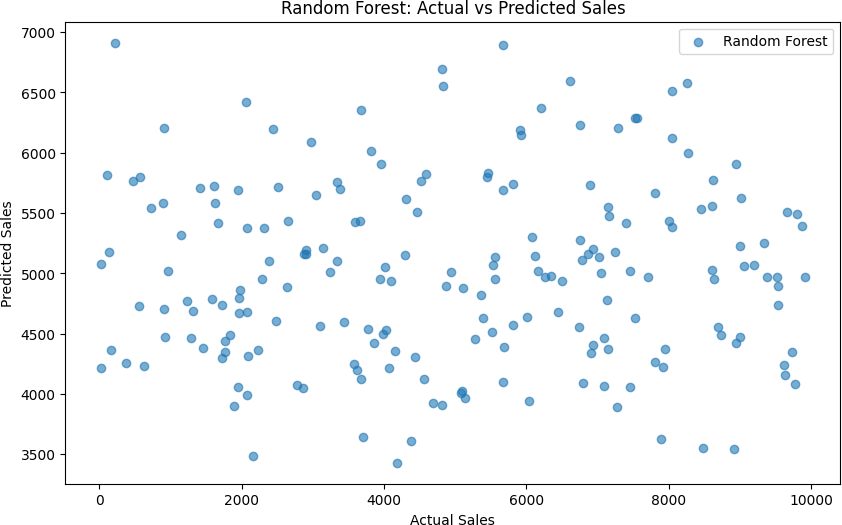
plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_rf, label='Random Forest', alpha=0.6) plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title('Random Forest: Actual vs Predicted Sales') plt.legend()

plt.show()



## Actual vs Predicted (Gradient Boosting)

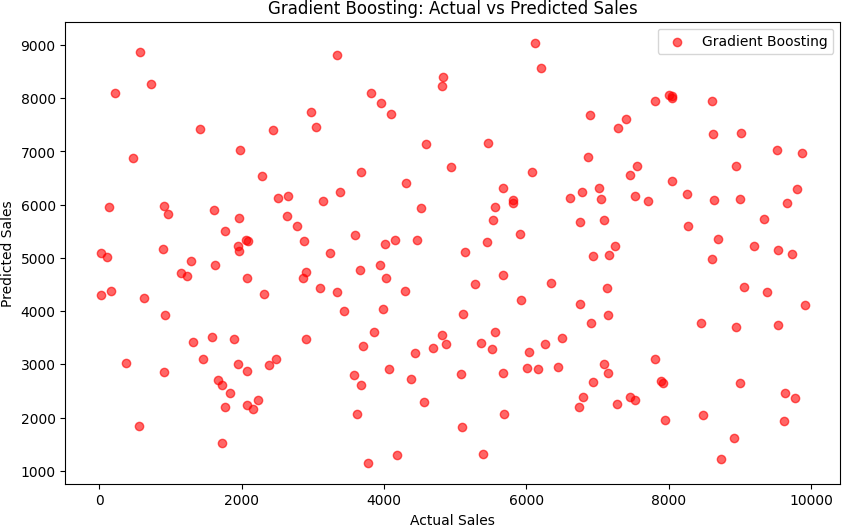
plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_gb, label='Gradient Boosting', alpha=0.6, color='red')

plt.xlabel('Actual Sales') plt.ylabel('Predicted Sales')

plt.title('Gradient Boosting: Actual vs Predicted Sales') plt.legend()

plt.show()



## Comparison of Both Models

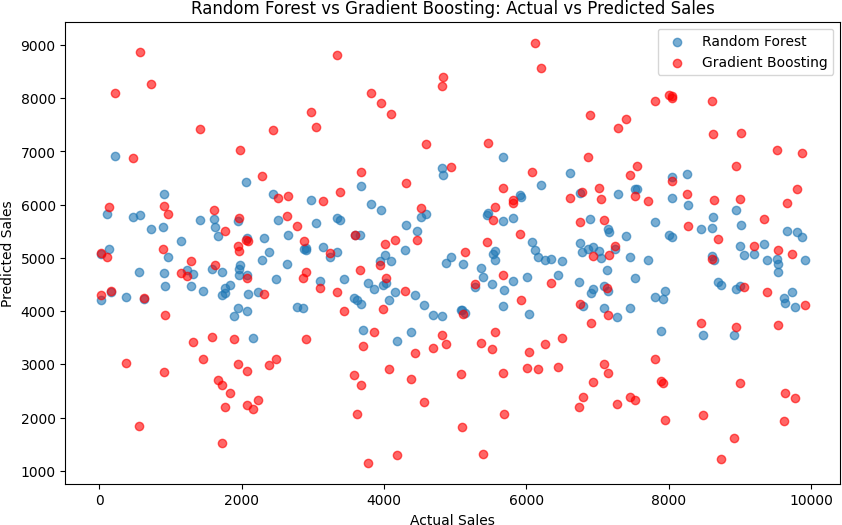
plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_rf, label='Random Forest', alpha=0.6) plt.scatter(y\_test, y\_pred\_gb, label='Gradient Boosting', alpha=0.6, color='red')

plt.xlabel('Actual Sales') plt.ylabel('Predicted Sales')

plt.title('Random Forest vs Gradient Boosting: Actual vs Predicted Sales') plt.legend()

plt.show()



**Outcome:** The visualizations revealed that both models had significant discrepancies between actual and predicted sales, leading to poor model performance.

# Comparative Evaluation

**Objective:** To compare the performance of the two models using evaluation metrics such as Mean Squared Error (MSE) and R-squared (R²).

**Results:**

* **Random Forest:**
  + **MSE:** 8,154,291.05
  + **R²:** -0.06
* **Gradient Boosting:**
  + **MSE:** 11,000,091.84
  + **R²:** -0.43

Finally, you can compare the performance of the two models based on the metrics:

print(f'Comparative Evaluation:\n')

print(f'Random Forest - MSE: {mse\_rf:.2f}, R^2: {r2\_rf:.2f}') print(f'Gradient Boosting - MSE: {mse\_gb:.2f}, R^2: {r2\_gb:.2f}')

Comparative Evaluation:

Random Forest - MSE: 8154291.05, R^2: -0.06

Gradient Boosting - MSE: 11000091.84, R^2: -0.43

# Conclusion

The analysis and modeling process applied to the augmented retail dataset provided several key insights, but it also highlighted significant challenges in accurately predicting SalesAmount using the available features.

**Model Performance:**

* Both the Random Forest and Gradient Boosting models underperformed, as indicated by their negative R-squared values. This suggests that the models were not able to capture the underlying patterns in the data effectively. Specifically, the negative R-squared values imply that the models performed worse than simply predicting the mean of the sales data.

**Possible Reasons for Underperformance:**

* **Feature Selection:** The features selected for modeling (such as QuantitySold, Discount, Price, Year, and Quarter) may not fully capture the complexity of the factors influencing sales. Important variables or interactions between variables may be missing from the analysis.
* **Data Quality:** While the dataset was clean in terms of missing values, other quality issues like outliers or skewed distributions might have negatively impacted model performance.
* **Model Complexity:** Both Random Forest and Gradient Boosting are powerful models, but they require well-structured and informative features to perform effectively. The models' inability to predict accurately suggests that the features used might not provide enough predictive power.

**Recommendations for Improvement:**

* **Enhanced Feature Engineering:** Future work should focus on creating more sophisticated features that capture the nuances of the sales process. This might include interaction terms, lag features (e.g., previous sales), or external factors like marketing spend or economic indicators.
* **Alternative Modeling Approaches:** Exploring other machine learning algorithms, such as Support Vector Machines (SVM) or neural networks, could provide better results. Additionally, applying more rigorous hyperparameter tuning or ensembling different models might improve performance.
* **Incorporate Domain Knowledge:** Leveraging business knowledge to identify key drivers of sales and incorporating them into the model could lead to better predictive accuracy.

**Conclusion:** In summary, while the models developed in this analysis did not achieve satisfactory predictive performance, the process provided valuable insights into the challenges of modeling retail sales data. The next steps should involve refining the feature set, exploring different modeling techniques, and considering additional data sources to improve model accuracy. This iterative approach is crucial in developing a robust predictive model that can provide actionable insights for business decision-making.

**Appendix:**

**Colab link:** https://colab.research.google.com/drive/1geYlLYXtPU4ntbRyco\_yLn\_lX1bfeE2q?usp=sharing

import pandas as pd

import numpy as np

file\_path = 'Augmented\_Retail\_Data\_Set\_No\_Time.xlsx'

data = pd.read\_excel(file\_path)

print(data.head())

print(data.info())

print(data.describe())

# Check for missing values

print(data.isnull().sum())

# Example: Fill missing numerical values with the median

# data.fillna(data.median(), inplace=True)

# Example: Fill missing categorical values with the mode

# data['Category'] = data['Category'].fillna(data['Category'].mode()[0])

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(10, 6))

sns.histplot(data['SalesAmount'], kde=True, color='blue')

plt.title('Sales Amount Distribution')

plt.xlabel('Sales Amount')

plt.ylabel('Frequency')

plt.show()

plt.figure(figsize=(10, 6))

data.groupby('Date')['SalesAmount'].sum().plot()

plt.title('Total Sales Over Time')

plt.xlabel('Date')

plt.ylabel('Total Sales Amount')

plt.show()

plt.figure(figsize=(10, 6))

sns.barplot(x='Category', y='SalesAmount', data=data, estimator=sum)

plt.title('Total Sales by Category')

plt.xlabel('Category')

plt.ylabel('Total Sales Amount')

plt.xticks(rotation=45)

plt.show()

from sklearn.model\_selection import train\_test\_split

X = data[['QuantitySold', 'Discount', 'Price', 'Year', 'Quarter']]

y = data['SalesAmount']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score

# Initialize and train the Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predictions using Random Forest

y\_pred\_rf = rf\_model.predict(X\_test)

# Evaluate Random Forest

mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

r2\_rf = r2\_score(y\_test, y\_pred\_rf)

print(f'Random Forest - Mean Squared Error: {mse\_rf:.2f}')

print(f'Random Forest - R^2 Score: {r2\_rf:.2f}')

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# Initialize the model

rf = RandomForestRegressor(random\_state=42)

# Perform grid search

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=3, n\_jobs=-1, verbose=2)

grid\_search.fit(X\_train, y\_train)

# Best parameters and score

print(f"Best Parameters: {grid\_search.best\_params\_}")

print(f"Best Score: {grid\_search.best\_score\_}")

# Use the best model to predict and evaluate

best\_rf\_model = grid\_search.best\_estimator\_

y\_pred\_rf = best\_rf\_model.predict(X\_test)

mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

r2\_rf = r2\_score(y\_test, y\_pred\_rf)

print(f'Tuned Random Forest - Mean Squared Error: {mse\_rf:.2f}')

print(f'Tuned Random Forest - R^2 Score: {r2\_rf:.2f}')

from sklearn.ensemble import GradientBoostingRegressor

# Initialize and train the Gradient Boosting model

gb\_model = GradientBoostingRegressor(n\_estimators=200, learning\_rate=0.1, max\_depth=10, random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Predictions using Gradient Boosting

y\_pred\_gb = gb\_model.predict(X\_test)

# Evaluate Gradient Boosting

mse\_gb = mean\_squared\_error(y\_test, y\_pred\_gb)

r2\_gb = r2\_score(y\_test, y\_pred\_gb)

print(f'Gradient Boosting - Mean Squared Error: {mse\_gb:.2f}')

print(f'Gradient Boosting - R^2 Score: {r2\_gb:.2f}')

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_rf, label='Random Forest', alpha=0.6)

plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title('Random Forest: Actual vs Predicted Sales')

plt.legend()

plt.show()

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_gb, label='Gradient Boosting', alpha=0.6, color='red')

plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title('Gradient Boosting: Actual vs Predicted Sales')

plt.legend()

plt.show()

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred\_rf, label='Random Forest', alpha=0.6)

plt.scatter(y\_test, y\_pred\_gb, label='Gradient Boosting', alpha=0.6, color='red')

plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title('Random Forest vs Gradient Boosting: Actual vs Predicted Sales')

plt.legend()

plt.show()

print(f'Comparative Evaluation:\n')

print(f'Random Forest - MSE: {mse\_rf:.2f}, R^2: {r2\_rf:.2f}')

print(f'Gradient Boosting - MSE: {mse\_gb:.2f}, R^2: {r2\_gb:.2f}')

from google.colab import files

uploaded = files.upload()

**REFERENCES:**

**Shazia Afzal Prof Lectures**

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc.

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc.

Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer.

Tufte, E. R. (2001). The Visual Display of Quantitative Information. Graphics Press.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer.

Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. The Annals of Statistics, 29(5), 1189-1232.

Matplotlib Documentation. (n.d.). Matplotlib: Visualization with Python. Available at: <https://matplotlib.org/stable/contents.html>

Seaborn Documentation. (n.d.). *Seaborn: Statistical Data Visualization*. Available at: <https://seaborn.pydata.org/>

Kuhn, M., & Johnson, K. (2019). Feature Engineering and Selection: A Practical Approach for Predictive Models. CRC Press.

Shmueli, G., Patel, N. R., & Bruce, P. C. (2016). Data Mining for Business Analytics: Concepts, Techniques, and Applications with JMP Pro. Wiley.