CASE STUDY – II

ZOMATO DATA ANALYSIS

TANMAY NAGDEVE

202301733

SCDL - PGDDS

2023-2025

INTRODUCTION:

Food delivery services have revolutionized the dining industry, with platforms like Zomato playing a crucial role in shaping consumer behaviour. This case study focuses on uncovering key insights from Zomato's restaurant and customer data using Python. The study delves into various aspects, including customer preferences, restaurant ratings, order modes, and spending patterns, to provide a comprehensive overview of trends that impact Zomato's business strategies. By leveraging data visualization techniques and classification models, this analysis aims to answer critical business questions such as the most popular restaurant type, customer voting trends, and the influence of online vs. offline orders on ratings.

The **Zomato** dataset, which was obtained from a public GitHub Repository, was the subject of exploratory data analysis (EDA) in this case study. Furthermore, a key aspect of this study involves identifying consumer spending habits, especially among couples who frequently order food online. By analysing factors such as restaurant ratings, the number of votes, and order modes, the project seeks to provide actionable insights that can help Zomato optimize its service offerings. Additionally, a **classification model** was fine-tuned to enhance predictive accuracy, aiding in strategic decision-making. This analysis is essential for businesses looking to improve customer satisfaction, boost restaurant engagement, and implement targeted marketing strategies to maximize revenue.

For EDA in the case study, the **Python** programming language has been used. The analysis part has been performed in the **Jupyter notebook**. Screenshots have been added accordingly, and the link to the git repository is included for verification.

PROCEDURE:

Following are the protocols followed to perform EDA on the dataset:

1. **Data Cleaning/ Data Transformation**:

The dataset underwent a comprehensive data cleaning and transformation process to ensure its quality and readiness for analysis. This included handling missing values, correcting inconsistencies, and standardizing data formats.

1. Checked for any null values in the dataset

```
#Checked for any null values

df.isna().sum()
```

```
name                            0
online_order                    0
book_table                      0
rate                            0
votes                           0
approx_cost(for two people)     0
listed_in(type)                 0
dtype: int64
```

2. Standardized the rate values

```
#Removing the denominator from the rate column to make it simpler, by creating a function

def handleRate(value):
    value=str(value).split('/')
    value = value[0];
    return float(value)

df['rate'] = df['rate'].apply(handleRate)

df.head()
```

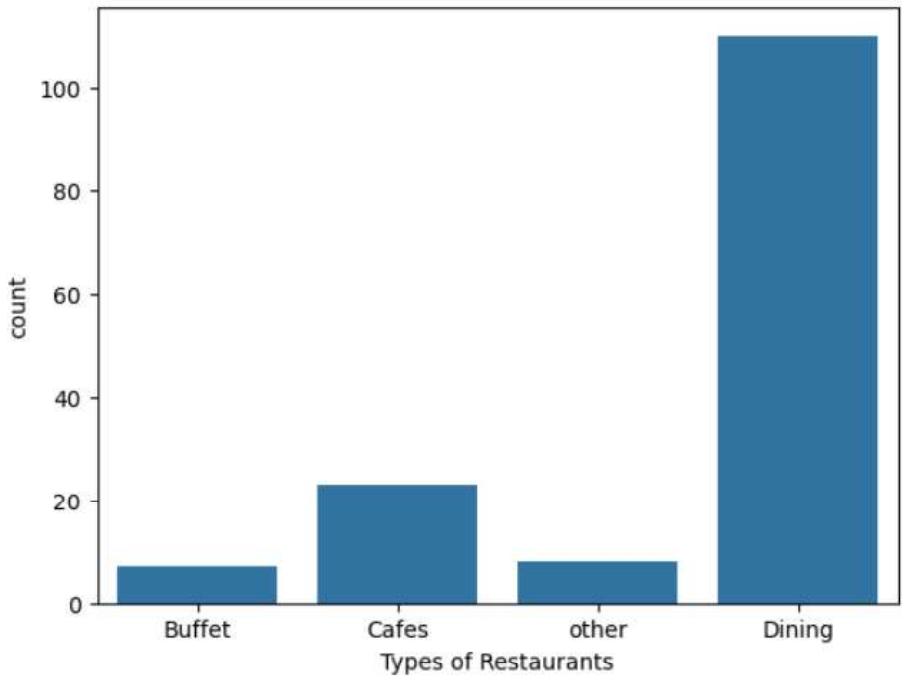
	name	online_order	book_table	rate	votes	approx_cost(for two people)	listed_in(type)
0	Jalsa	Yes	Yes	4.1	775	800	Buffet
1	Spice Elephant	Yes	No	4.1	787	800	Buffet
2	San Churro Cafe	Yes	No	3.8	918	800	Buffet
3	Addhuri Udupi Bhojana	No	No	3.7	88	300	Buffet
4	Grand Village	No	No	3.8	166	600	Buffet

2. **Exploratory Data Analysis :**

To determine which restaurant type listed on Zomato has the highest count, I used the Seaborn library to obtain the exact counts and visualized them with a bar graph, as shown in the screenshot.

```
sns.countplot(x=df['listed_in(type)'])
mp.xlabel('Types of Restaurants')
```

```
Text(0.5, 0, 'Types of Restaurants')
```



It seems like Majority of Customers Order from Dining type of restaurant.

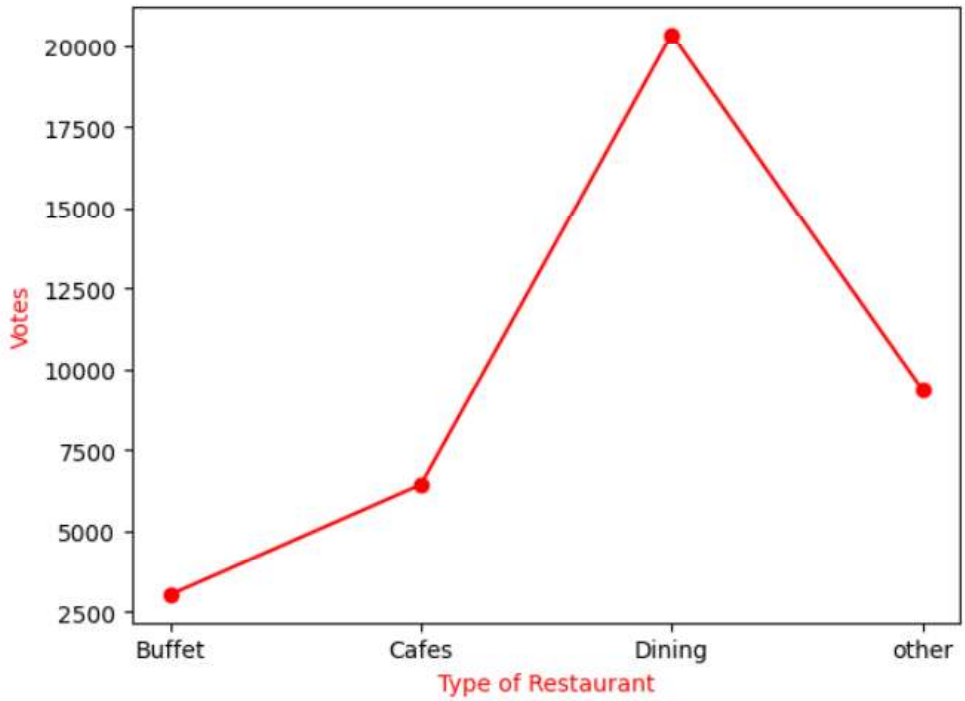
Now to determine which type of restaurant has got the highest count of votes. I've created a line graph as shown.

```
# The following graphs denote the number of votes for each type of restaurant.

am = df.groupby('listed_in(type)').agg({'votes' : 'sum'})

mp.plot(am, c='red', marker='o')
mp.xlabel('Type of Restaurant', c='red', size=10)
mp.ylabel('Votes', c='red', size=10)
```

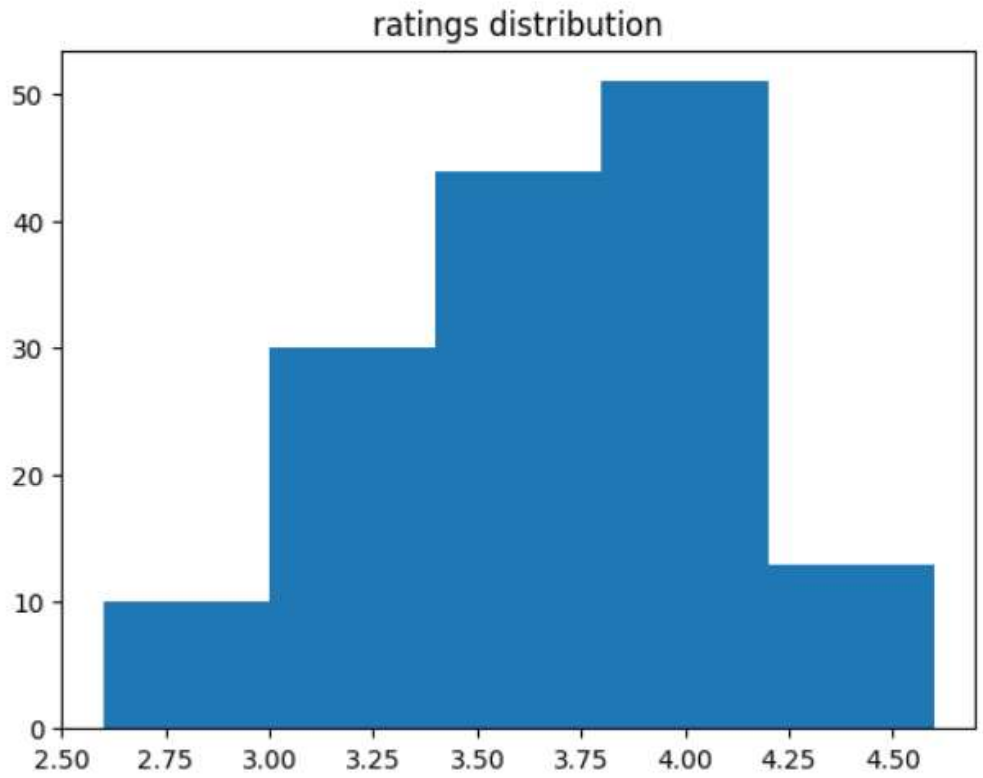
```
Text(0, 0.5, 'Votes')
```



The restaurants coming under the dining category has the highest count of votes!

Next, I've determined the ratings which the majority of restaurants have received.

```
mp.hist(df['rate'], bins = 5)
mp.title('ratings distribution')
mp.show()
```

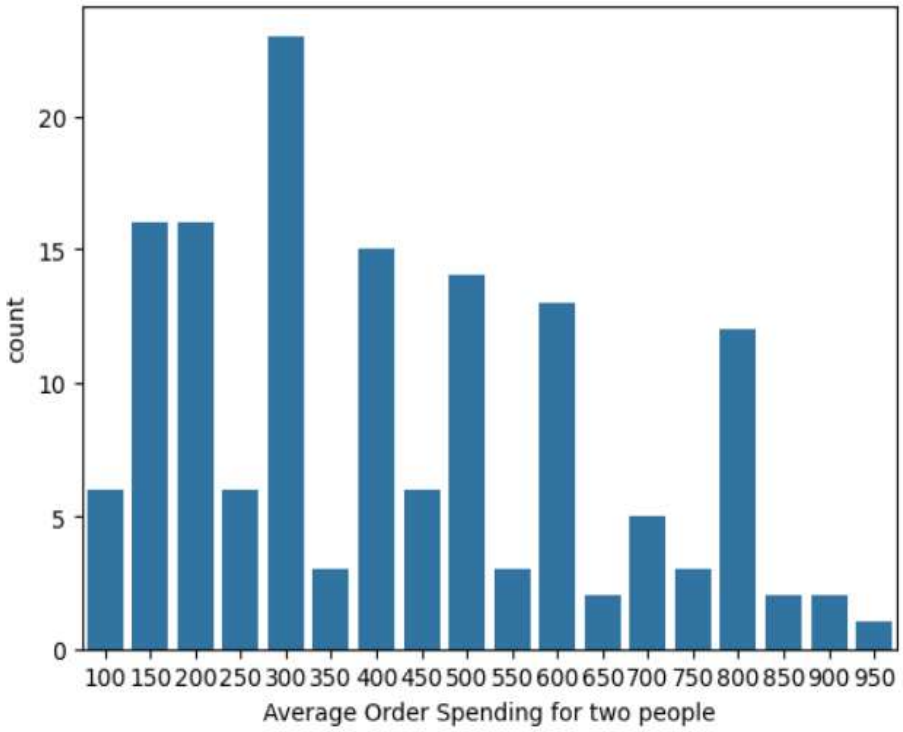


The majority of restaurants have received ratings from **3.5 to 4** as per the histogram.

Next, I've determined the average spending most couples do while they order their food online.

```
sns.countplot(x=df['approx_cost(for two people)'])
mp.xlabel('Average Order Spending for two people')
```

```
Text(0.5, 0, 'Average Order Spending for two people')
```



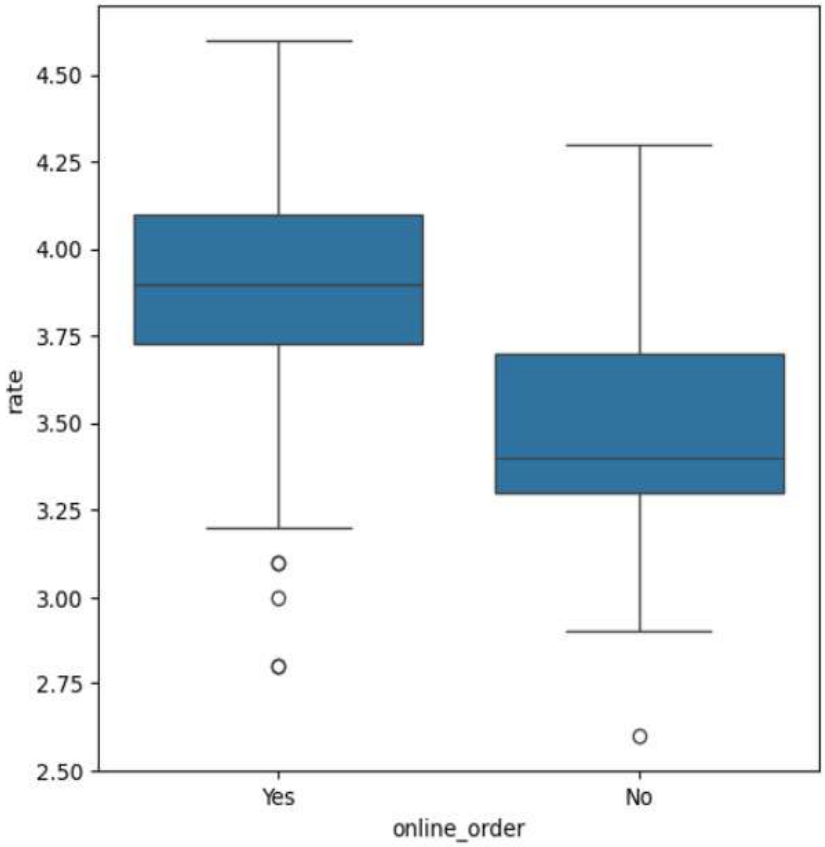
Majority of couples prefer restaurants with an approximate cost of **300 Rs.**

Next, I've determined which mode of order (online or offline) has the highest rating.

```
mp.figure(figsize = (6,6))

sns.boxplot(x = 'online_order', y = 'rate', data = df)
```

```
<Axes: xlabel='online_order', ylabel='rate'>
```



Offline order received lower rating as compared to Online orders.

Next, I determined which type of restaurant received more offline orders, so that Zomato can provide customers with some good offers. Using Heatmap.

```
pivot_table = df.pivot_table(index='listed_in(type)', columns='online_order', aggfunc='size', fill_value=0)
sns.heatmap(pivot_table, annot=True, cmap='YlGnBu', fmt= 'd')
mp.title('Heatmap')
mp.xlabel('Online Order')
mp.ylabel('Listed_in(type)')
mp.show()
```

Dining Restaurants primarily accept offline orders, whereas cafes primarily receive online orders. This suggests that clients prefer orders in person at restaurants, but prefer online ordering at cafes.

CLASSIFICATION ANALYSIS:

1. Two predictive models were implemented to classify restaurants based on various factors.
2. Insights from classification models help in understanding **what influences customer ratings and order preferences**.
3. While the provided dataset presented limitations in predicting restaurant type, this case study demonstrates the process of **applying classification algorithms** and **evaluating their performance**.
4. The models I've chosen are the **Decision Tree** & the **Random Forest** as the dataset is considerably small, and decision trees works well with structured data and random forest works better than a single decision tree, handles small datasets well by averaging multiple trees.
5. The performances of both the models were evaluated to choose the best model for the selected dataset for classification analysis.

The following screenshots shows the implementation of the models along with their results.

I've made use of the **scikit-learn** python library to train, evaluate, and tune the chosen models.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
```

First, I've selected the target variable -> Made the independent variables numeric -> Split the data into training and test sets -> As per the requirement dropped the columns with String and Object datatypes.

```
#For classification analysis I've chosen the 'listed_in(type)' column as the target variable and the rest of the columns as feature variables.

X = df.drop('listed_in(type)', axis=1)  # Features
y = df['listed_in(type)']               # Target variable
```

```
# Convert 'online_order' and 'book_table' to numerical
X['online_order'] = X['online_order'].map({'Yes': 1, 'No': 0})
X['book_table'] = X['book_table'].map({'Yes': 1, 'No': 0})
```

```
# Split 'rate' into a numerical value
X['rate'] = X['rate'].str.split('/').str[0].astype(float)
```

```
# Splitting 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)
```

```
X_train = X_train.drop('name', axis=1)
X_test = X_test.drop('name', axis=1)
```

Next, wrote the code for the decision tree classifier and got the result as follows:

The accuracy is **56.66%**.

```
# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, dt_predictions))
print("Decision Tree Classification Report:\n", classification_report(y_test, dt_predictions))
```

```
Decision Tree Accuracy: 0.5666666666666667
Decision Tree Classification Report:
              precision    recall  f1-score   support

      Buffet       0.00      0.00      0.00         0
       Cafes       0.25      0.11      0.15         9
      Dining       0.73      0.80      0.76        20
       other       0.00      0.00      0.00         1

    accuracy                           0.57        30
   macro avg       0.24      0.23      0.23        30
weighted avg       0.56      0.57      0.55        30
```

Next, wrote the code for the Random Forest Classifier and the accuracy came around to be the same i.e. **56.66%**

```
# 3. Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, rf_predictions))
print("Random Forest Classification Report:\n", classification_report(y_test, rf_predictions))
```

```
Random Forest Accuracy: 0.5666666666666667
Random Forest Classification Report:
              precision    recall  f1-score   support

      Buffet       0.00      0.00      0.00         0
       Cafes       0.00      0.00      0.00         9
      Dining       0.68      0.85      0.76        20
       other       0.00      0.00      0.00         1

    accuracy                           0.57        30
   macro avg       0.17      0.21      0.19        30
weighted avg       0.45      0.57      0.50        30
```

The performance of model could have been better, so I tried tuning the models to get better results and the accuracy for the **Decision tree** shot up to **63.33%**.

```
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', None],
    'criterion': ['gini', 'entropy']
}

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Perform grid search
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Train the model with the best parameters
best_dt_classifier = DecisionTreeClassifier(**best_params, random_state=42)
best_dt_classifier.fit(X_train, y_train)
dt_predictions = best_dt_classifier.predict(X_test)
print("Tuned Decision Tree Accuracy:", accuracy_score(y_test, dt_predictions))
print("Tuned Decision Tree Classification Report:\n", classification_report(y_test, dt_predictions))
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
Best Score: 0.7721014492753623
Tuned Decision Tree Accuracy: 0.6333333333333333
Tuned Decision Tree Classification Report:
               precision    recall  f1-score   support

      Buffet       0.00      0.00      0.00         0
       Cafes       0.50      0.22      0.31         9
      Dining       0.77      0.85      0.81        20
       other       0.00      0.00      0.00         1

    accuracy                           0.63        30
   macro avg       0.32      0.27      0.28        30
weighted avg       0.67      0.63      0.63        30
```

Further, tuned the **random forest model** as well. And the accuracy shot up to **60%**.

```
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy']
}

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Perform grid search
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)

# Train the model with the best parameters
best_rf_classifier = RandomForestClassifier(**best_params, random_state=42)
best_rf_classifier.fit(X_train, y_train)
rf_predictions = best_rf_classifier.predict(X_test)
print("Tuned Random Forest Accuracy:", accuracy_score(y_test, rf_predictions))
print("Tuned Random Forest Classification Report:\n", classification_report(y_test, rf_predictions))
```

```
Best Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_
stimators': 50}
Best Score: 0.788768115942029
Tuned Random Forest Accuracy: 0.6
Tuned Random Forest Classification Report:
              precision    recall  f1-score   support

      Buffet       0.00      0.00      0.00         0
       Cafes       0.33      0.11      0.17         9
      Dining       0.68      0.85      0.76        20
       other       0.00      0.00      0.00         1

    accuracy                           0.60        30
   macro avg       0.25      0.24      0.23        30
weighted avg       0.55      0.60      0.55        30
```

Results of Examination

Key Findings from the Data Analysis

1. **Customer Ordering Preferences**
   - A majority of customers prefer **Dining-type restaurants** over other options.
   - Dining restaurants receive the highest number of **votes**, indicating their popularity.
2. **Restaurant Ratings & Cost Analysis**
   - Most restaurants receive ratings in the range of **3.5 to 4.0 stars**.
   - **Couples tend to prefer restaurants** with an approximate cost of **₹300** per meal.
3. **Offline vs. Online Ordering Trends**
   - **Offline orders** tend to receive **lower ratings** compared to **online orders**.
   - **Dining restaurants primarily accept offline orders**, while **cafés receive more online orders**.
   - This suggests that customers prefer **in-person dining at restaurants** but prefer **online ordering for cafés**.

Classification Model Analysis

Performance Metrics & Model Selection

- Models were evaluated based on **accuracy and classification reports**.
- **Hyperparameter tuning** was applied using **GridSearchCV** to optimize model performance.
- The best-performing model was **Decision Tree**, achieving an accuracy of **63.33%** upon tuning.