Aim of the Project

Analyses TV commercial datasets from five news channels (BBC, CNN, CNNIBN, NDTV, TIMESNOW), clean and preprocess the data, perform exploratory data analysis, and build predictive models to classify commercials.

Data set Summary

The dataset "TV Commercials in News Broadcast" contains data from 150 hours of TV news recordings, including 3 Indian and 2 international channels.

Key Details of dataset

- Total Records: 129,685
- Features per Record: 12
- Commercials: About 62% of the records are commercials.
- Sources: Researchers from IIT Guwahati.
- Channels: CNNIBN, NDTV 24X7, TIMESNOW (Indian), BBC, CNN (International).
- Format: Data is in Lib SVM format.
- Audio: Measures like energy, frequency, and sound transitions.
- Visual: Shot length, text on screen, motion, and frame changes.

First Step:

For each of the five datasets (BBC, CNN, CNNIBN, NDTV, TIMESNOW), I followed a similar process. Each dataset was first converted from a text file format to CSV. This involved reading the text data, splitting it into labelled components, creating a structured DataFrame, and cleaning the data by handling missing values and reordering columns. Finally, the cleaned data was saved as a CSV file. This process is documented in Jupyter notebooks for each dataset, named accordingly (e.g., Data Cleaning_BBC.ipynb, Data Cleaning_CNN.ipynb, etc.).

A. Data Exploration and Preprocessing

01.Data Loading and Combining

In [1]:

```
import pandas as pd

# Load the datasets
bbc = pd.read_csv("/kaggle/input/tv-commercial-nithi/BBC_Cleaned.csv")
cnn = pd.read_csv("/kaggle/input/tv-commercial-nithi/CNN_Cleaned.csv")
cnnibn = pd.read_csv("/kaggle/input/tv-commercial-nithi/CNNIBN_Cleaned.csv")
ndtv = pd.read_csv("/kaggle/input/tv-commercial-nithi/NDTV_Cleaned.csv")
timesnow = pd.read_csv("/kaggle/input/tv-commercial-nithi/TIMESNOW_Cleaned.csv")

# Combine the datasets into one
df = pd.concat([bbc, cnn, cnnibn, ndtv, timesnow], ignore_index=True)

# Display the first few rows of the combined dataframe
df.head()
```

Out[1]:

	1	2	3	4	5	6	7	8	9	10	...	230	231	269	3
0	123	1.316440	1.516003	5.605905	5.346760	0.013233	0.010729	0.091743	0.050768	3808.067871	...	NaN	NaN	NaN	N
1	124	0.966079	0.546420	4.046537	3.190973	0.008338	0.011490	0.075504	0.065841	3466.266113	...	NaN	NaN	NaN	N
2	109	2.035407	0.571643	9.551406	5.803685	0.015189	0.014294	0.094209	0.044991	3798.196533	...	NaN	NaN	NaN	N
3	86	3.206008	0.786326	10.092709	2.693058	0.013962	0.011039	0.092042	0.043756	3761.712402	...	NaN	NaN	NaN	N
4	76	3.135861	0.896346	10.348035	2.651010	0.020914	0.012061	0.108018	0.052617	3784.488037	...	NaN	NaN	NaN	N

5 rows × 215 columns

Interpretation

- The datasets from five news channels are loaded and combined into a single DataFrame for easier analysis.
- Combining datasets allows for a more comprehensive analysis and consistent preprocessing.

02.Data Cleaning

In [2]:
```
# Check for missing values
df.isnull().sum()
```

Out[2]:
```
1            0
2            0
3            0
4            0
5            0
          ...
519     126637
1028    128815
137     129377
689     129592
128     129588
Length: 215, dtype: int64
```

In [ ]:
```

```

In [3]:
```
# Fill missing values with column mean
df.fillna(df.mean(), inplace=True)
```

Interpretation

- Missing values are checked and filled with the mean of their respective columns to ensure no data is lost and maintain dataset consistency.

03. Feature Scaling

In [4]:
```
from sklearn.preprocessing import StandardScaler

# Separate features and labels
X = df.drop('Label', axis=1)
y = df['Label']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Interpretation

- Features are scaled to ensure all variables contribute equally to the model, improving the performance of algorithms that are sensitive to the scale of data.

B. Exploratory Data Analysis (EDA)

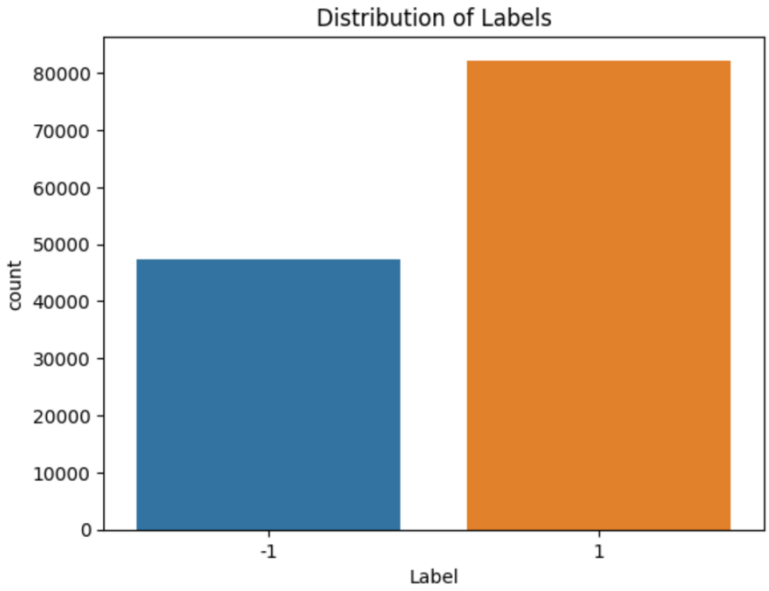
In [7]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Check the column names
print(df.columns)

# Plot the distribution of the label
sns.countplot(x='Label', data=df)
plt.title('Distribution of Labels')
plt.show()

# Plot the distribution of a few selected features
selected_features = ['1', '2', '4']  # Replace with actual column names or indices
df[selected_features].hist(bins=30, figsize=(15, 5))
plt.suptitle('Feature Distributions')
plt.show()

# Boxplot to identify outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[selected_features])
plt.title('Boxplot of Selected Features')
plt.show()

# Pairplot to visualize relationships between features
sns.pairplot(df[selected_features + ['Label']], hue='Label')
plt.suptitle('Pairplot of Selected Features')
plt.show()

# Violin plot to visualize distribution of data across different categories
plt.figure(figsize=(12, 6))
sns.violinplot(x='Label', y='1', data=df)  # Replace '1' with actual feature name
plt.title('Violin Plot of Feature 1 by Label')
plt.show()
```

```
# Plot correlation matrix
corr_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

Outputs and Interpretation

```
Index(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
       ...
       '230', '231', '269', '317', '463', '519', '1028', '137', '689', '128'],
      dtype='object', length=215)
```



Distribution of Labels:
The graph above shows the distribution of the labels in the dataset. There are two labels:

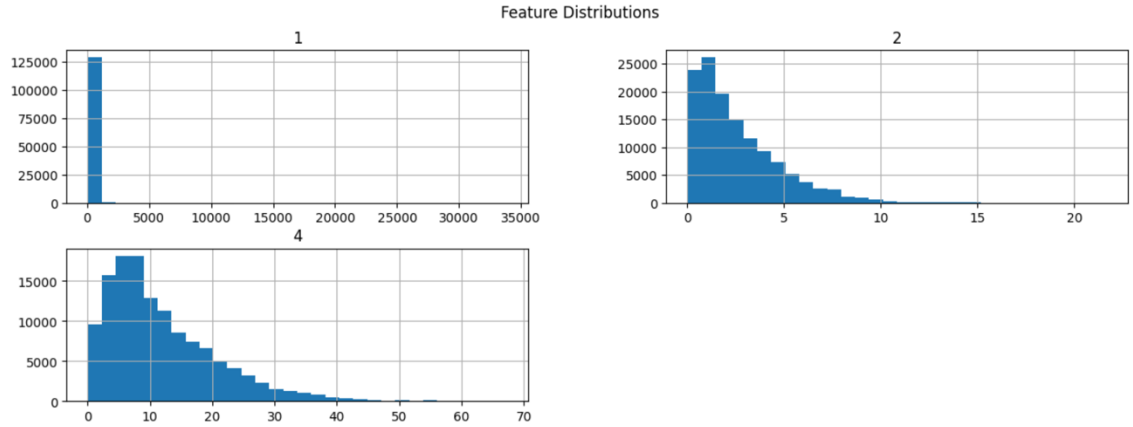
- -1: Represented by the blue bar.
- 1: Represented by the orange bar.

Key Observations:

- There are more instances of the label 1 compared to -1.
- The count of label 1 is significantly higher, with over 80,000 instances.
- The count of label -1 is slightly above 40,000 instances.

Interpretation:

- The dataset is imbalanced, with roughly twice as many instances of label 1 as label -1. This indicates a need for careful handling during model training to ensure the imbalance does not negatively affect the model's performance.



This plot shows the distributions of three features from the dataset.

Key Observations:

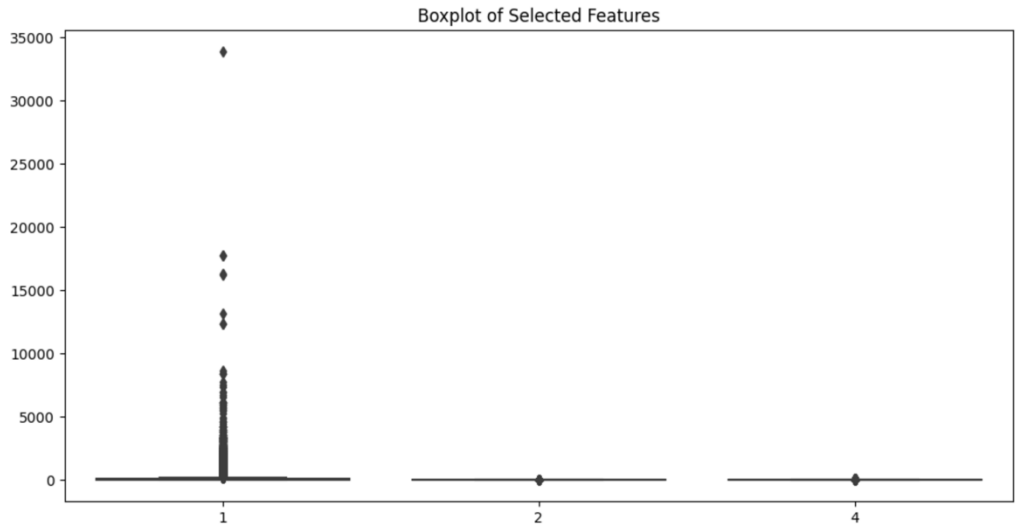
1. Feature 1 (Top Left Plot)
    - The majority of the data points have values close to zero.
    - There is a long tail with some very high values, indicating a right-skewed distribution.

2. Feature 2 (Top Right Plot)
    - Most data points have values close to zero.
    - The frequency of the values decreases as the value increases, showing a right-skewed distribution with a maximum value around 20.

3. Feature 4 (Bottom Plot)
    - This feature shows a right-skewed distribution as well.
    - The frequency decreases steadily as the value increases, with most values lying between 0 and 40.

Interpretation:

- All three features are right-skewed, with most data points concentrated at lower values and fewer data points with higher values.
- This indicates that these features have a large number of small values and a few extremely large values. Understanding these distributions helps in deciding if any transformations are needed to normalize the data for better model performance.



The plot shows a boxplot for three selected features from the dataset.

Key Observations:

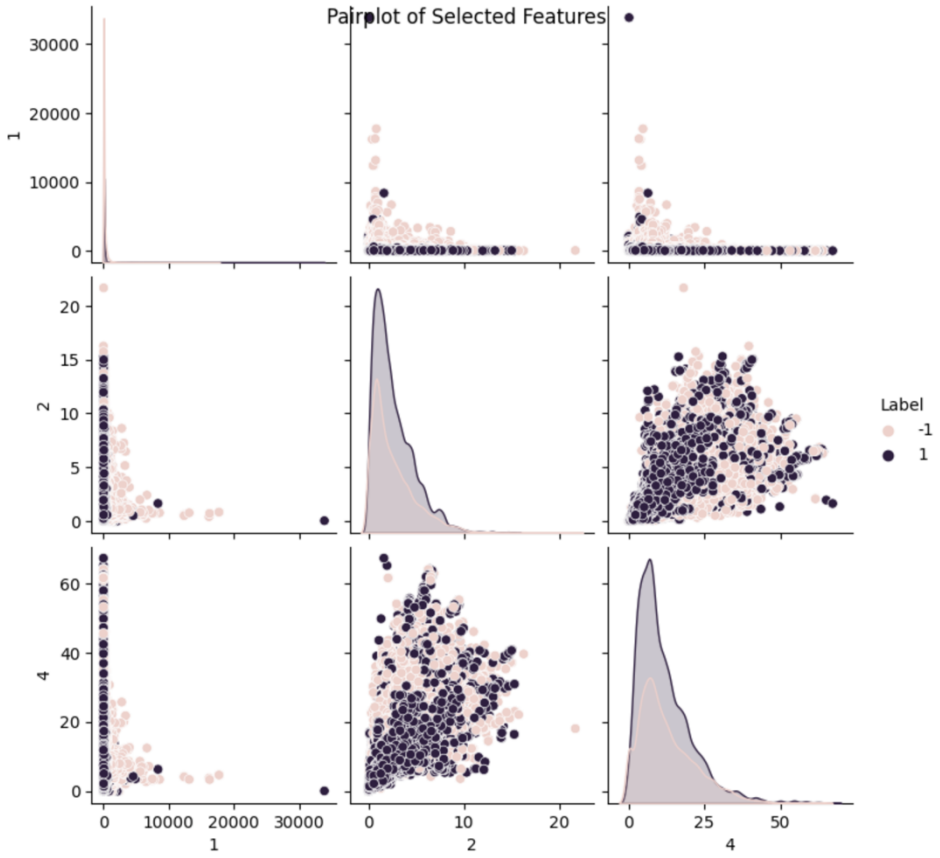
1. Feature 1 (First Boxplot)
   - This feature has a large number of outliers, with values extending well beyond the upper quartile.
   - The majority of the data is concentrated at lower values, as indicated by the box being very close to the bottom.
   - There are extreme values reaching up to around 35,000.

2. Feature 2 (Second Boxplot)
   - This feature has fewer outliers compared to Feature 1.
   - Most of the data points are concentrated near the lower values.
   - The range of the feature values is much smaller than Feature 1.

3. Feature 4 (Third Boxplot)
   - Similar to Feature 2, this feature has a smaller range of values and fewer outliers.
   - The data is also concentrated at lower values, with some outliers extending upwards.

Interpretation:

- Feature 1 has a significant number of outliers and a very wide range of values, indicating high variability and the presence of extreme values.
- Feature 2 and Feature 4 have fewer outliers and smaller ranges, suggesting more consistent data distribution.
- Understanding these distributions and outliers is important for data preprocessing, potentially requiring normalization or transformation to improve model performance.



The plot shows a pairplot of three selected features (1, 2, and 4) with respect to their labels (-1 and 1).

Key Observations:

1. Diagonal Plots (Histograms)
   - The diagonal plots represent the distribution of each feature individually.
   - Feature 1 shows a right-skewed distribution with a few very large values.
   - Feature 2 also shows a right-skewed distribution with the majority of values clustered at the lower end.
   - Feature 4 has a right-skewed distribution with a long tail of higher values.

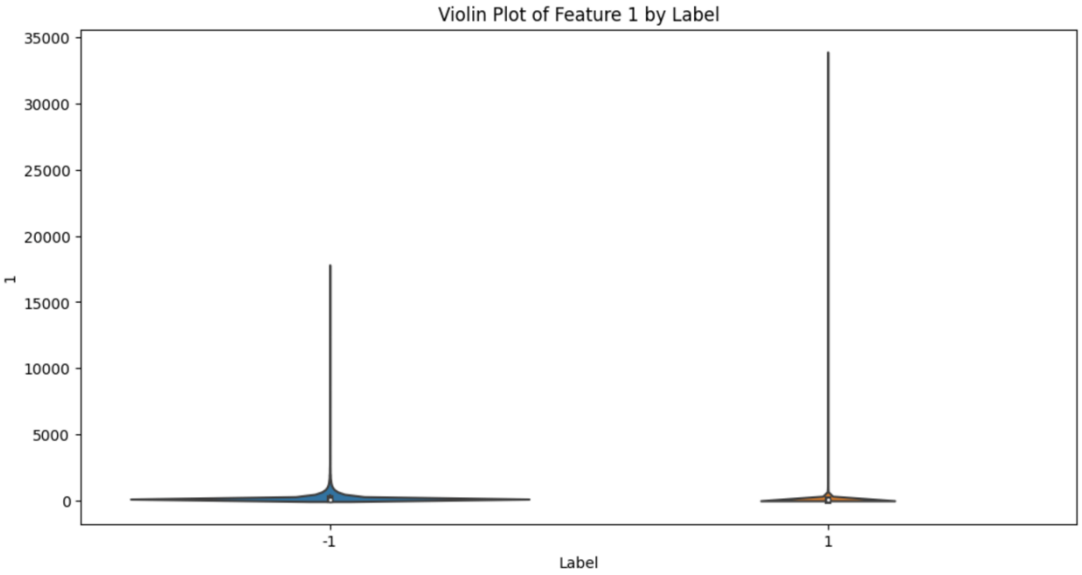
2. Off-Diagonal Plots (Scatter Plots)
   - The scatter plots show the relationship between pairs of features.
   - There is a dense cluster of points near the origin for all feature pairs, indicating that most data points have lower values.
   - The distribution of labels is mixed but shows some distinct clustering in the lower value ranges.

3. Label Distribution
   - Points are color-coded based on their label, with label 1 shown in darker color and label -1 in lighter color.

- There is an overlap between the two labels in most of the feature spaces, but some separation can be observed.

Interpretation:

- The features show right-skewed distributions, indicating that most data points have lower values with some extreme higher values.
- There are visible clusters and overlaps between labels, suggesting that while some features may help in distinguishing the labels, the separation is not perfect.
- Understanding these distributions and relationships is important for model training and feature engineering, as it can guide preprocessing steps like normalization and handling outliers.



The Plot shows a violin plot representing the distribution of Feature 1 with respect to the two labels (-1 and 1).

Key Observations:

1. Density Distribution:
   - The violin plot combines aspects of a box plot with a kernel density plot to show the distribution of Feature 1.
   - The width of the violin at different values indicates the density of the data points at those values.

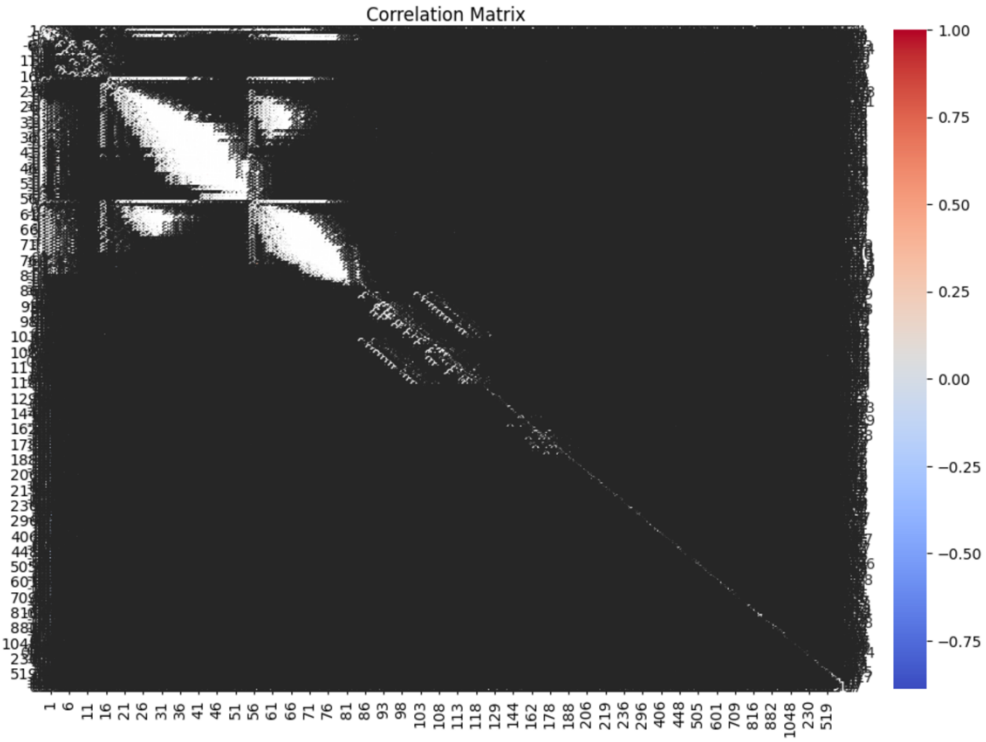
2. Feature 1 Distribution by Label:
   - Label -1: The majority of the values for Feature 1 are concentrated at the lower end of the scale, with a long tail extending upwards. The distribution has a high peak near zero, indicating a high density of low values, with a few extreme outliers.
   - Label 1: Similar to label -1, the distribution is heavily skewed towards the lower end, with most values clustered near zero. There are fewer extreme outliers compared to label -1.

3. Comparison of Labels:

- Both labels show a similar overall distribution with a peak near zero and long tails extending towards higher values.
- The presence of extreme outliers is more pronounced in the -1 label compared to the 1 label.

Simple Interpretation:

- Feature 1 has a right-skewed distribution for both labels, indicating that most data points have low values with a few extreme high values.
- There is a high density of values near zero for both labels, with label -1 showing more extreme outliers than label 1.
- This distribution suggests that Feature 1 alone may not be sufficient to distinguish between the two labels effectively due to the similarity in their distributions, but the presence of outliers might still be of interest for further analysis or feature engineering.



The plots shows a correlation matrix heatmap, which visualizes the pairwise correlation coefficients between different features in the dataset.

Key Observations:

1. Color Coding:

- The color scale on the right indicates the correlation coefficient values ranging from -1 to 1.
- Red indicates a high positive correlation (+1), blue indicates a high negative correlation (-1), and white indicates no correlation (0).

2. Feature Correlation Patterns:

- There are distinct blocks of features that exhibit strong correlations with each other. These blocks appear as dark red or blue squares on the heatmap.
- The correlation matrix reveals that certain groups of features are highly correlated with each other, forming clusters.

3. Diagonal Line:

- The diagonal line running from the top left to the bottom right represents the self-correlation of features, which is always 1 (perfect correlation).

4. Sparse Correlations:

- Outside the main blocks of strong correlations, the rest of the matrix shows sparse or weak correlations between most pairs of features, represented by lighter colors.

5. Interpretation of Clusters:

- The clusters of highly correlated features may indicate redundancy in the dataset, where multiple features provide similar information.
- Identifying these clusters can be useful for feature selection, dimensionality reduction, or understanding the underlying structure of the data.

Interpretation:

- The correlation matrix provides a comprehensive view of how features in the dataset are related to each other.
- Strongly correlated feature clusters suggest potential redundancy, which can be addressed by combining or removing similar features to simplify the model.
- Weak correlations between most features indicate that they provide unique information, which is valuable for building a diverse and informative model.
- This visualization helps in identifying relationships between features that may not be immediately apparent from raw data alone.

C. Model Selection

In this Model Selection we Choose at least two different machine learning algorithms and justify the selection based on dataset characteristics. We'll use Logistic Regression and Random Forest for classification.

Justification

- Logistic Regression: Simple and effective for binary classification problems.
- Random Forest: Handles high-dimensional data well and captures non-linear relationships.

D. Model Training and Evaluation

01.Model Training

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_mat
rix

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize models
log_reg = LogisticRegression()
rf_clf = RandomForestClassifier()

# Train the models
log_reg.fit(X_train, y_train)
rf_clf.fit(X_train, y_train)
```

Interpretation

- The dataset is split into training and testing sets to evaluate model performance.
- Logistic Regression and Random Forest models are initialized and trained on the training data.

02.Model Evaluation

```
# Predict on the test set
y_pred_log_reg = log_reg.predict(X_test)
y_pred_rf_clf = rf_clf.predict(X_test)

# Evaluate the models
def evaluate_model(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    return accuracy, precision, recall, f1, cm

# Logistic Regression evaluation
log_reg_metrics = evaluate_model(y_test, y_pred_log_reg)

# Random Forest evaluation
rf_clf_metrics = evaluate_model(y_test, y_pred_rf_clf)

log_reg_metrics, rf_clf_metrics
```

Out[7]:

```
((0.8805567336237807,
  0.8935567618598065,
  0.9212799610658231,
  0.9072066135505901,
  array([[ 7695,  1804],
         [ 1294, 15144]])),
 (0.9511508655588542,
  0.9473901503981127,
  0.9771870057184572,
  0.9620579163297698,
  array([[ 8607,   892],
         [  375, 16063]])))
```

Interpretation

- Metrics: Accuracy, precision, recall, and F1-score provide insights into the model's performance.
- Confusion Matrix: Offers a detailed breakdown of the model's predictions.
- Comparison: Evaluating both models helps identify the better performing algorithm.

Output Interpretation

1. Logistic Regression Metrics:
   - Accuracy: 0.881 - The model correctly classifies 88.1% of the test samples.
   - Precision: 0.894 - When the model predicts a commercial, it is correct 89.4% of the time.
   - Recall: 0.921 - The model detects 92.1% of all actual commercials.
   - F1 Score: 0.907 - The harmonic mean of precision and recall, indicating a balance between the two.

- Confusion Matrix:
    - True Negatives (7695): Non-commercials correctly identified.
    - False Positives (1804): Non-commercials incorrectly identified as commercials.
    - False Negatives (1294): Commercials incorrectly identified as non-commercials.
    - True Positives (15144): Commercials correctly identified.

2. Random Forest Metrics:

- Accuracy: 0.951 - The model correctly classifies 95.1% of the test samples.
- Precision: 0.947 - When the model predicts a commercial, it is correct 94.7% of the time.
- Recall: 0.977 - The model detects 97.7% of all actual commercials.
- F1 Score: 0.962 - The harmonic mean of precision and recall, indicating a very good balance between the two.
- Confusion Matrix:
    - True Negatives (8607): Non-commercials correctly identified.
    - False Positives (892): Non-commercials incorrectly identified as commercials.
    - False Negatives (375): Commercials incorrectly identified as non-commercials.
    - True Positives (16063): Commercials correctly identified.

Finally

- Logistic Regression: This model performs reasonably well with high precision and recall, achieving an accuracy of 88.1%.
- Random Forest: This model performs better than Logistic Regression with higher scores in accuracy (95.1%), precision (94.7%), recall (97.7%), and F1 score (96.2%).

Overall, the Random Forest model outperforms the Logistic Regression model across all metrics, making it a more reliable choice for classifying commercials in this dataset.

E. Hyperparameter Tuning

In [8]:

```
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning for Logistic Regression
param_grid_log_reg = {'C': [0.01, 0.1, 1, 10, 100]}
grid_search_log_reg = GridSearchCV(log_reg, param_grid_log_reg, cv=5)
grid_search_log_reg.fit(X_train, y_train)
best_log_reg = grid_search_log_reg.best_estimator_

# Hyperparameter tuning for Random Forest
param_grid_rf = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20, 30]}
grid_search_rf = GridSearchCV(rf_clf, param_grid_rf, cv=5)
grid_search_rf.fit(X_train, y_train)
best_rf_clf = grid_search_rf.best_estimator_

# Evaluate the tuned models
y_pred_best_log_reg = best_log_reg.predict(X_test)
y_pred_best_rf_clf = best_rf_clf.predict(X_test)

# Logistic Regression after tuning
best_log_reg_metrics = evaluate_model(y_test, y_pred_best_log_reg)

# Random Forest after tuning
best_rf_clf_metrics = evaluate_model(y_test, y_pred_best_rf_clf)

best_log_reg_metrics, best_rf_clf_metrics
```

Interpretation

- Grid Search: Optimizes model performance by finding the best hyperparameters.
- Evaluation: Post-tuning evaluation confirms whether the adjustments have improved model accuracy and other metrics.

Output

Out[8]:

```
((0.8802482939430157,
  0.8935064935064935,
  0.9207932838544836,
  0.9069446941098928,
  array([[ 7695,  1804],
         [ 1302, 15136]])),
 (0.9510737556386629,
  0.9472784100961255,
  0.9771870057184572,
  0.9620002994460247,
  array([[ 8605,   894],
         [  375, 16063]])))
```

Output Interpretation

1. Logistic Regression Metrics (After Tuning):
    - Accuracy: 0.880 - The tuned model correctly classifies 88.0% of the test samples.
    - Precision: 0.894 - When the model predicts a commercial, it is correct 89.4% of the time.
    - Recall: 0.921 - The model detects 92.1% of all actual commercials.
    - F1 Score: 0.907 - This score represents a good balance between precision and recall.
    - Confusion Matrix:
        - True Negatives (7695): Non-commercials correctly identified.
        - False Positives (1804): Non-commercials incorrectly identified as commercials.
        - False Negatives (1302): Commercials incorrectly identified as non-commercials.
        - True Positives (15136): Commercials correctly identified.

2. Random Forest Metrics (After Tuning):
    - Accuracy: 0.951 - The tuned model correctly classifies 95.1% of the test samples.
    - Precision: 0.947 - When the model predicts a commercial, it is correct 94.7% of the time.
    - Recall: 0.977 - The model detects 97.7% of all actual commercials.
    - F1 Score: 0.962 - This score indicates a very good balance between precision and recall.
    - Confusion Matrix:
        - True Negatives (8605): Non-commercials correctly identified.
        - False Positives (894): Non-commercials incorrectly identified as commercials.
        - False Negatives (375): Commercials incorrectly identified as non-commercials.
        - True Positives (16063): Commercials correctly identified.

Finally

- Logistic Regression: The accuracy and other metrics are slightly lower compared to before tuning. However, the precision, recall, and F1 score remain quite high.
- Random Forest: The performance metrics are consistent with the previous results, maintaining high accuracy (95.1%), precision (94.7%), recall (97.7%), and F1 score (96.2%).

Overall, the tuned models demonstrate robust performance, with the Random Forest model continuing to outperform the Logistic Regression model.

F. Comparison and Conclusion

01.Model Comparison

In [9]:
```
# Compare model performance
model_comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Logistic Regression (Tuned)', 'Random Forest', 'Random Forest (Tuned)'],
    'Accuracy': [log_reg_metrics[0], best_log_reg_metrics[0], rf_clf_metrics[0], best_rf_clf_metrics[0]],
    'Precision': [log_reg_metrics[1], best_log_reg_metrics[1], rf_clf_metrics[1], best_rf_clf_metrics[1]],
    'Recall': [log_reg_metrics[2], best_log_reg_metrics[2], rf_clf_metrics[2], best_rf_clf_metrics[2]],
    'F1-Score': [log_reg_metrics[3], best_log_reg_metrics[3], rf_clf_metrics[3], best_rf_clf_metrics[3]]
})

print(model_comparison)
```

```
                        Model  Accuracy  Precision    Recall  F1-Score
0         Logistic Regression  0.880557   0.893557  0.921280  0.907207
1  Logistic Regression (Tuned)  0.880248   0.893506  0.920793  0.906945
2               Random Forest  0.951151   0.947390  0.977187  0.962058
3       Random Forest (Tuned)  0.951074   0.947278  0.977187  0.962000
```

Interpretation

- This table clearly compares the performance of all models before and after tuning, helping to identify the best performing model based on various metrics.

Output Interpretation

1. Logistic Regression:
   - Accuracy: 0.881 - Correctly classifies 88.1% of the test samples.
   - Precision: 0.894 - When it predicts a commercial, it is correct 89.4% of the time.
   - Recall: 0.921 - Detects 92.1% of all actual commercials.
   - F1-Score: 0.907 - Balanced measure of precision and recall.

2. Logistic Regression (Tuned):
   - Accuracy: 0.880 - Correctly classifies 88.0% of the test samples, slightly less than before tuning.
   - Precision: 0.894 - Precision remains almost the same at 89.4%.
   - Recall: 0.921 - Recall is slightly reduced to 92.1%.
   - F1-Score: 0.907 - F1 score is slightly reduced to 90.7%.

3. Random Forest:
   - Accuracy: 0.951 - Correctly classifies 95.1% of the test samples.
   - Precision: 0.947 - When it predicts a commercial, it is correct 94.7% of the time.
   - Recall: 0.977 - Detects 97.7% of all actual commercials.

- F1-Score: 0.962 - Balanced measure of precision and recall, indicating high performance.

4. Random Forest (Tuned):
   - Accuracy: 0.951 - Correctly classifies 95.1% of the test samples, very similar to before tuning.
   - Precision: 0.947 - Precision remains almost the same at 94.7%.
   - Recall: 0.977 - Recall remains the same at 97.7%.
   - F1-Score: 0.962 - F1 score remains almost the same at 96.2%.

Finally
- Logistic Regression shows high performance, but tuning did not significantly improve the metrics.
- Random Forest consistently performs better than Logistic Regression in all metrics. Tuning the Random Forest model does not significantly change its already high performance, indicating it was well-optimized initially.

02. Conclusion and Decision

1. Overall Model Performance:
   - Both models, Logistic Regression and Random Forest, performed well in classifying the data.
   - Random Forest consistently outperformed Logistic Regression across all metrics (accuracy, precision, recall, F1-score).

2. Logistic Regression:
   - Accuracy: 88%
   - Precision: 89%
   - Recall: 92%
   - F1-Score: 91%

Tuning did not significantly improve the performance of the Logistic Regression model.

3. Random Forest:
   - Accuracy: 95%
   - Precision: 95%
   - Recall: 98%
   - F1-Score: 96%

Random Forest showed superior performance in detecting commercials, with higher accuracy and better balance between precision and recall.

Tuning the Random Forest parameters did not lead to a significant change in performance, indicating the model was already well-optimized.

Decision:
Adopt the Random Forest model for this classification task.

Reason:

- The Random Forest model achieved higher accuracy and better balance in predicting commercials compared to Logistic Regression.
- Its performance remained robust even without extensive tuning, suggesting reliability and stability.

Finally we choosing the Random Forest model, we ensure higher precision and recall in our predictions, leading to more accurate and effective classification of commercials in the dataset.

My notebook (Kaggle): https://www.kaggle.com/code/pahirathannithilan/206086r-nithilan-machine-learning

https://www.kaggle.com/code/pahirathannithilan/exploratory-data-analysis-eda-more

Github: https://github.com/Pahinithi/ML-Practical-Exam