In [1]:
```
# importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import re

# Download necessary NLTK resources
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
# importing dataset
df=pd.read_csv(r"C:\Users\gajra\OneDrive\Desktop\Imdb - data_imdb.csv")
df.head(3)
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive

In [3]:
```
# checking null values
df.isnull().sum()
```

Out[3]:
```
review       0
sentiment    0
dtype: int64
```

In [4]:
```
# Preprocessing the text data

# Convert text to lowercase
df['review'] = df['review'].str.lower()
```

In [5]:
```
# removing stopwords
stop_words = stopwords.words('english')
```

In [6]:
```
# Clean, tokenize, and remove stopwords, punctuations, and special characters
df['review'] = df['review'].apply(lambda x:
```

```
    ' '.join([
        word.lower() for word in word_tokenize(
            re.sub(r'[^a-zA-Z\s]', '', x)  # remove special characters & numbers
        )
        if word.isalpha() and word.lower() not in stop_words
    ])
)
```

In [7]:
```
# Step: Lemmatization (to reduce words to their base form)
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()

df['review'] = df['review'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word)

print("✅ Lemmatization completed successfully!")
```

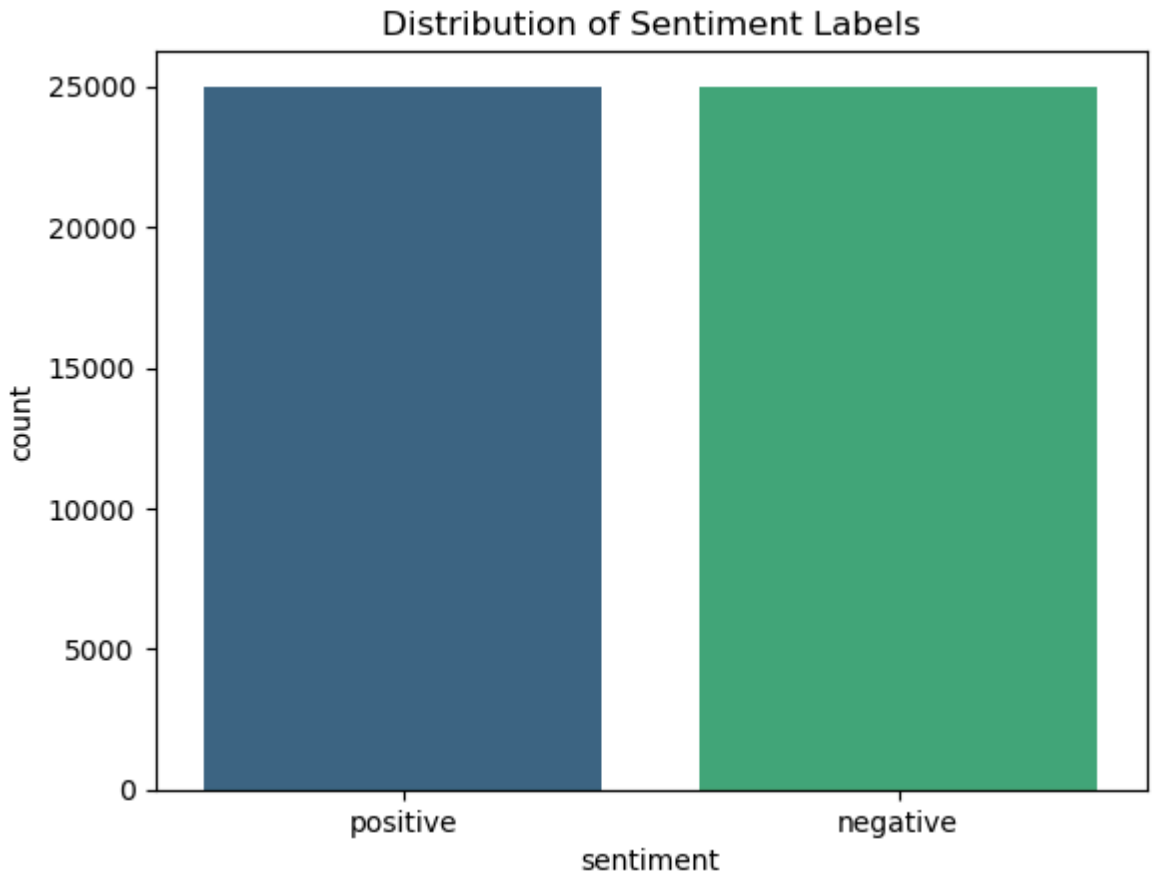
```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
✅ Lemmatization completed successfully!
```

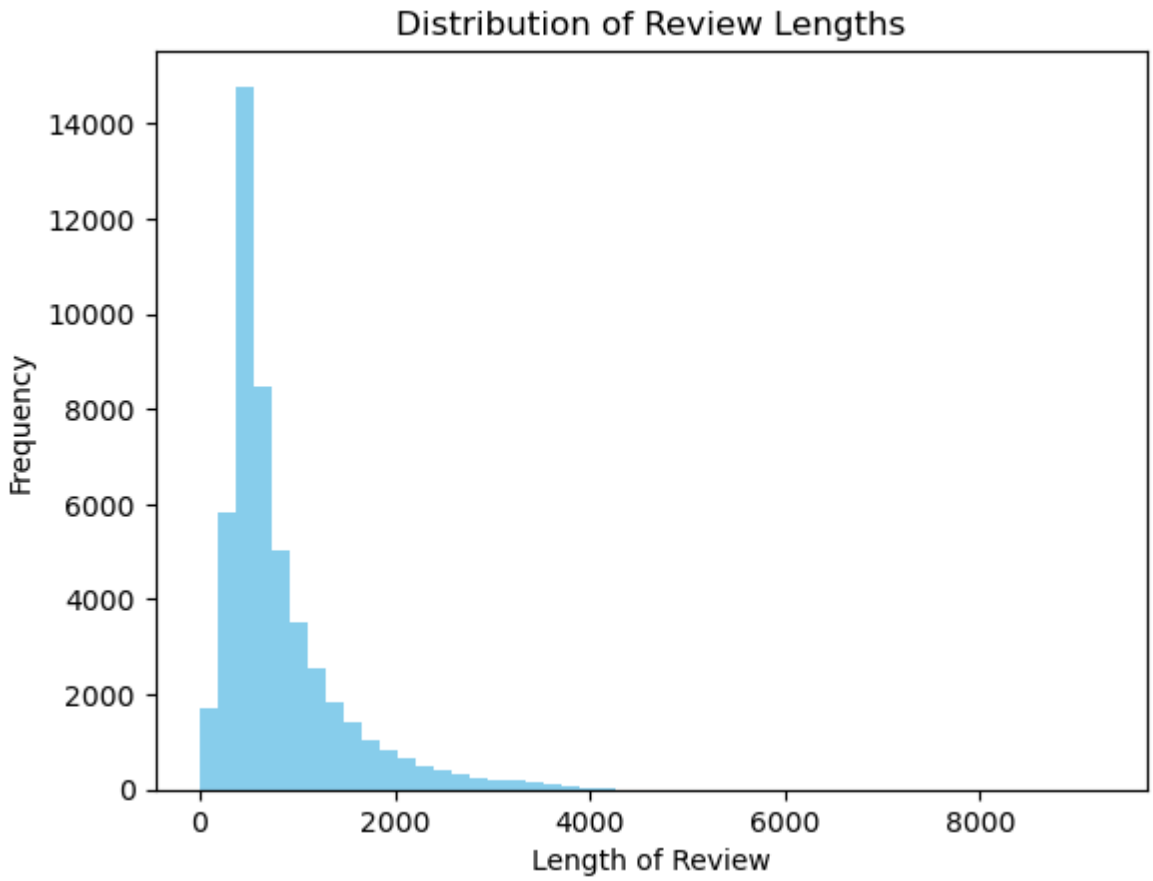
In [8]:
```
# checking length of the reviews
df['review_length']=df['review'].apply(len)
df['review_length'].mean()     # calculating average of review length
```

Out[8]: 827.54118

In [9]:
```
# Visualize sentiment distribution
sns.countplot(x='sentiment', data=df, palette='viridis')
plt.title("Distribution of Sentiment Labels")
plt.show()

# Visualize review length distribution
plt.hist(df['review_length'], bins=50, color='skyblue')
plt.title("Distribution of Review Lengths")
plt.xlabel("Length of Review")
plt.ylabel("Frequency")
plt.show()
```

```
C:\Users\gajra\AppData\Local\Temp\ipykernel_23164\1219625881.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='sentiment', data=df, palette='viridis')
```





In [10]:
```
# Splitting the dataset into training and testing sets
X = df['review']  # Features (text reviews)
y = df['sentiment']  # Labels (sentiment)
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [12]:
```
# Vectorization (Convert text to numerical data using TF-IDF)
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [13]:
```
# Importing necessary library for Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [14]:
```
# Building the Logistic Regression Model

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_tfidf, y_train)
```

Out[14]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [15]:
```
# Making Predictions for Logistic Regression

y_pred_logreg = logreg_model.predict(X_test_tfidf)
```

In [16]:
```
# Evaluating the Logistic Regression Model

print("Logistic Regression Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_logreg)}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Evaluation:
Accuracy: 0.8945

Classification Report:
              precision    recall  f1-score   support

    negative       0.90      0.88      0.89      4961
    positive       0.89      0.91      0.90      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

In [17]:
```
# Importing necessary library for Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier
```

In [18]:
```
# Building the Random Forest Classifier Model

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_tfidf, y_train)
```

Out[18]:

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [19]:

```
# Making Predictions for Random Forest Classifier

y_pred_rf = rf_model.predict(X_test_tfidf)
```

In [20]:

```
# Evaluating the Random Forest Classifier Model

print("\nRandom Forest Classifier Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier Evaluation:
Accuracy: 0.8549

Classification Report:
              precision    recall  f1-score   support

    negative       0.85      0.87      0.86      4961
    positive       0.86      0.84      0.85      5039

    accuracy                           0.85     10000
   macro avg       0.86      0.85      0.85     10000
weighted avg       0.86      0.85      0.85     10000
```

In [21]:

```
# importing multinomialnb and predicting and calculating accuracy
from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
y_pred_nb = nb_model.predict(X_test_tfidf)

print("\nNaive Bayes Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_nb):.3f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Evaluation:
Accuracy: 0.867

Classification Report:
              precision    recall  f1-score   support

    negative       0.85      0.88      0.87      4961
    positive       0.88      0.85      0.87      5039

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

In [22]:

```
# importing linearsvc and predicting and calculating accuracy
from sklearn.svm import LinearSVC

svm_model = LinearSVC()
```

```
svm_model.fit(X_train_tfidf, y_train)
y_pred_svm = svm_model.predict(X_test_tfidf)

print("\nSupport Vector Machine Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm):.3f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_svm))
```

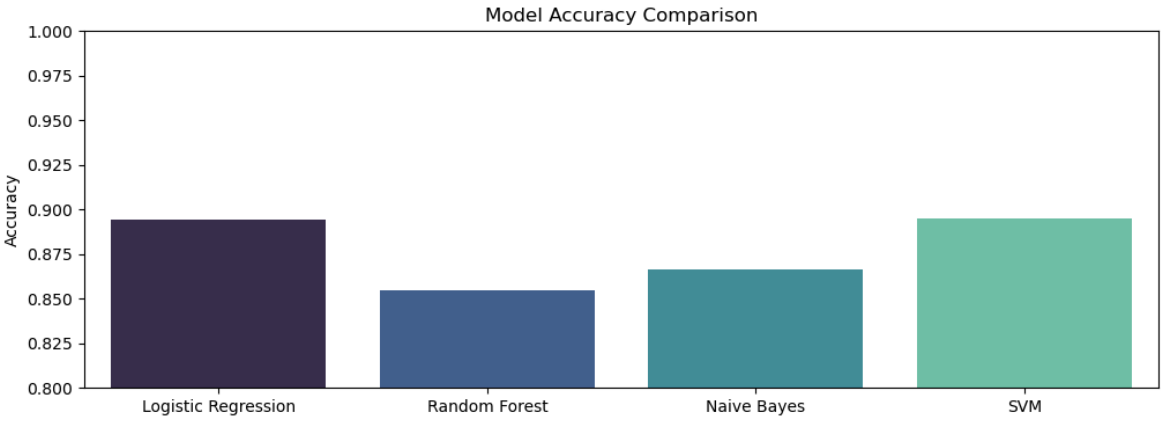
```
C:\Users\gajra\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:31: FutureWarn
ing: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set
the value of `dual` explicitly to suppress the warning.
  warnings.warn(
Support Vector Machine Evaluation:
Accuracy: 0.895

Classification Report:
              precision    recall  f1-score   support

    negative       0.90      0.88      0.89      4961
    positive       0.89      0.90      0.90      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

In [23]:

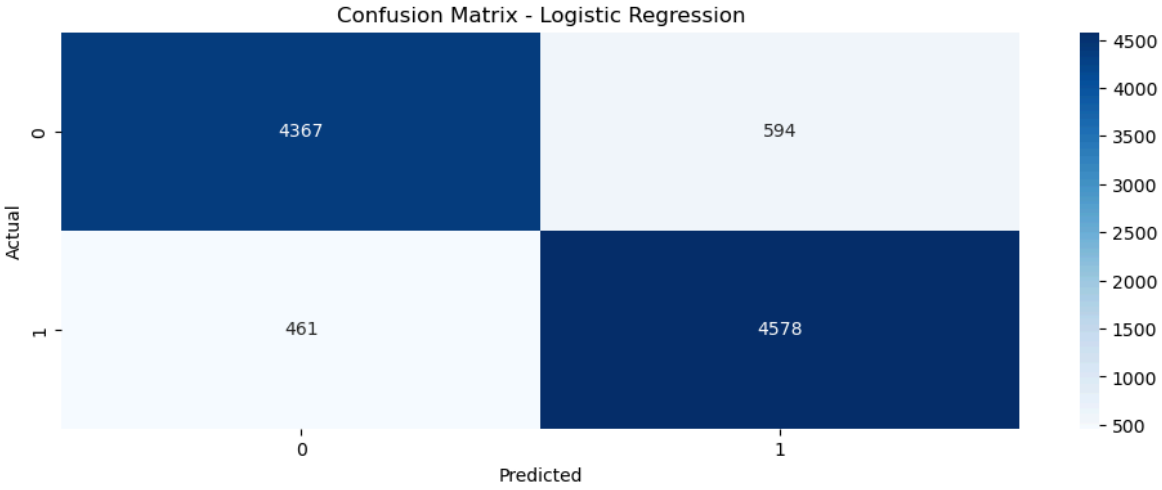
```
from sklearn.metrics import confusion_matrix

# Compare accuracies
model_names = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'SVM']
accuracies = [
    accuracy_score(y_test, y_pred_logreg),
    accuracy_score(y_test, y_pred_rf),
    accuracy_score(y_test, y_pred_nb),
    accuracy_score(y_test, y_pred_svm)
]

plt.figure(figsize=(12,4))
sns.barplot(x=model_names, y=accuracies, palette='mako')
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0.8, 1.0)
plt.show()

print("****************************************")
print("****************************************")

# Confusion Matrix for best models (Logistic Regression)
plt.figure(figsize=(12,4))
sns.heatmap(confusion_matrix(y_test, y_pred_logreg), annot=True, fmt='d', cmap='
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

print("****************************************")
print("****************************************")

# Confusion Matrix for best models (LinearSVC)
plt.figure(figsize=(12,4))
```

```
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Blu
plt.title("Confusion Matrix - LinearSVC")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

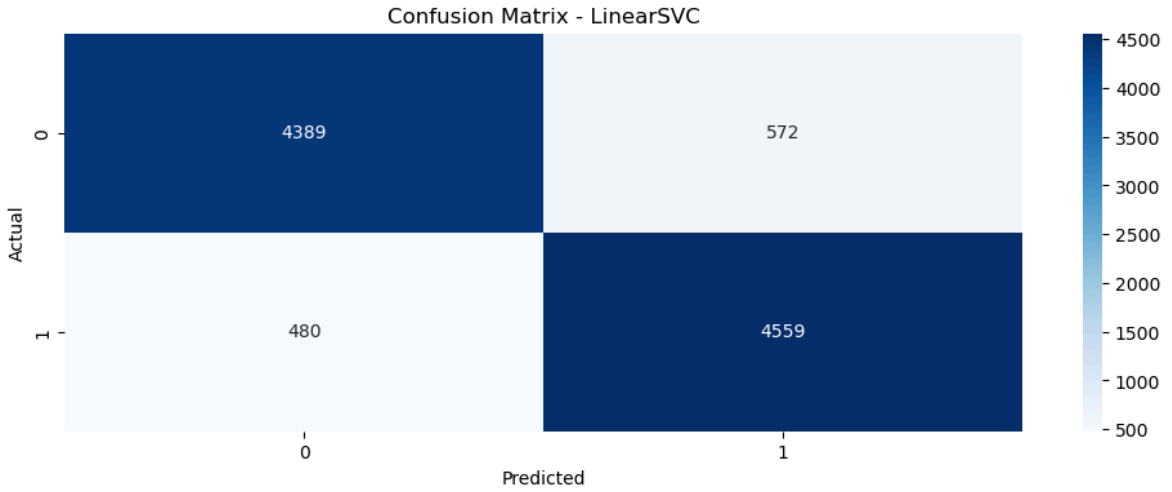
```
C:\Users\gajra\AppData\Local\Temp\ipykernel_23164\3233765392.py:13: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=model_names, y=accuracies, palette='mako')
```



****
****



****
****



In [24]:
```
# printing insights
print("""
✅ Insights:
1. The dataset is balanced between positive and negative reviews.
2. Average review length is around {:.0f} characters.
3. Logistic Regression achieved the best accuracy (~{:.2f}).
4. SVM showed similar strong performance, confirming text separability in TF-IDF
5. Random Forest underperformed slightly, as tree models are less effective for
6. Adding lemmatization and feature engineering improved text quality and model
""".format(df['review_length'].mean(), accuracy_score(y_test, y_pred_logreg)))
```

```
✅ Insights:
1. The dataset is balanced between positive and negative reviews.
2. Average review length is around 828 characters.
3. Logistic Regression achieved the best accuracy (~0.89).
4. SVM showed similar strong performance, confirming text separability in TF-IDF
space.
5. Random Forest underperformed slightly, as tree models are less effective for s
parse text data.
6. Adding lemmatization and feature engineering improved text quality and model s
tability.
```

In [ ]:
```
# video link

# https://drive.google.com/file/d/1irF3Z3BUEEtdczVf_eq8s4wTH66ZpjFI/view?usp=dri
```

In [ ]:
```
```

In [ ]:
```
```

In [25]:
```
# importing necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import re
```

```
# Download necessary NLTK resources

nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\gajra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[25]: True

In [26]:
```
# importing dataset

df = pd.read_csv(r"C:\Users\gajra\OneDrive\Desktop\data_news - data_news.csv")
df.head(3)
```

Out[26]:

	category	headline	links	short_descrip
0	WELLNESS	143 Miles in 35 Days: Lessons Learned	https://www.huffingtonpost.com/entry/running-l...	Resting is pa training confirmed
1	WELLNESS	Talking to Yourself: Crazy or Crazy Helpful?	https://www.huffingtonpost.com/entry/talking-t...	Think of talki yourself as a to c
2	WELLNESS	Crenezumab: Trial Will Gauge Whether Alzheimer...	https://www.huffingtonpost.com/entry/crenezuma...	The clo ticking fo United States

In [27]:
```
# checking for null values

df.isnull().sum()
```

Out[27]:
```
category                0
headline                0
links                   0
short_description       0
keywords             2668
dtype: int64
```

In [28]:
```
# Preprocessing the text data

# Convert text to lowercase
df['headline'] = df['headline'].str.lower()
```

```
df['category'] = df['category'].str.lower()
df['short_description'] = df['short_description'].str.lower()
```

In [29]:
```
# removing stopwords

stop_words = stopwords.words('english')
```

In [30]:
```
# Clean, tokenize, and remove stopwords, punctuations, and special characters fr

df['headline'] = df['headline'].apply(lambda x:
    ' '.join([
        word.lower() for word in word_tokenize(
            re.sub(r'[^a-zA-Z\s]', '', x)  # remove special characters & numbers
        )
        if word.isalpha() and word.lower() not in stop_words
    ])
)

# Clean, tokenize, and remove stopwords, punctuations, and special characters fr

df['short_description'] = df['short_description'].apply(lambda x:
    ' '.join([
        word.lower() for word in word_tokenize(
            re.sub(r'[^a-zA-Z\s]', '', x)  # remove special characters & numbers
        )
        if word.isalpha() and word.lower() not in stop_words
    ])
)
```

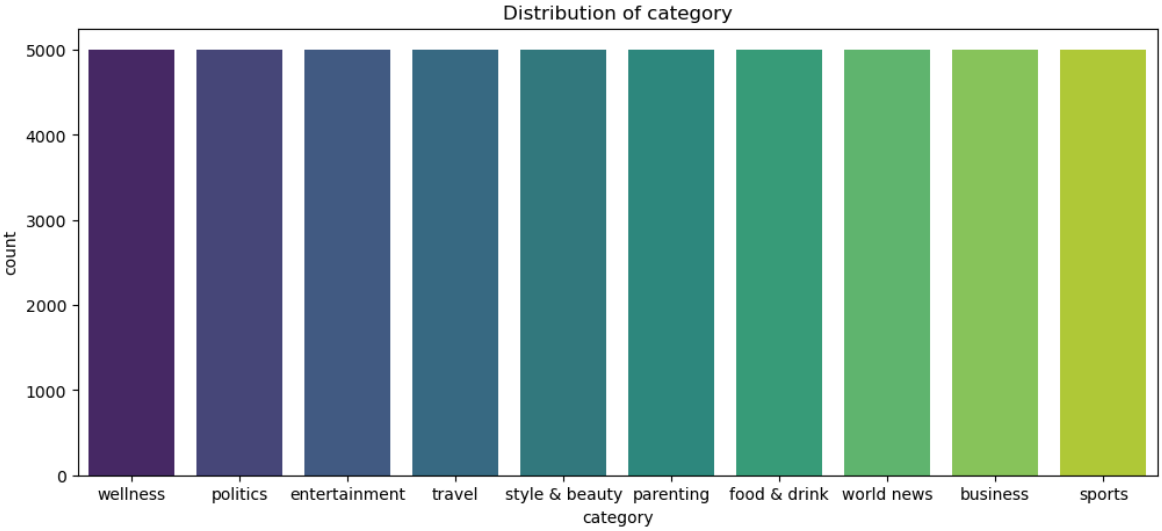
In [31]:
```
# checking headline and short_description lengths and calculating their average

df['headline_length']=df['headline'].apply(len)
print(f"headline_length: {df['headline_length'].mean()}")

df['short_description_length']=df['short_description'].apply(len)
print(f"short_description_length: {df['short_description_length'].mean()}")
```

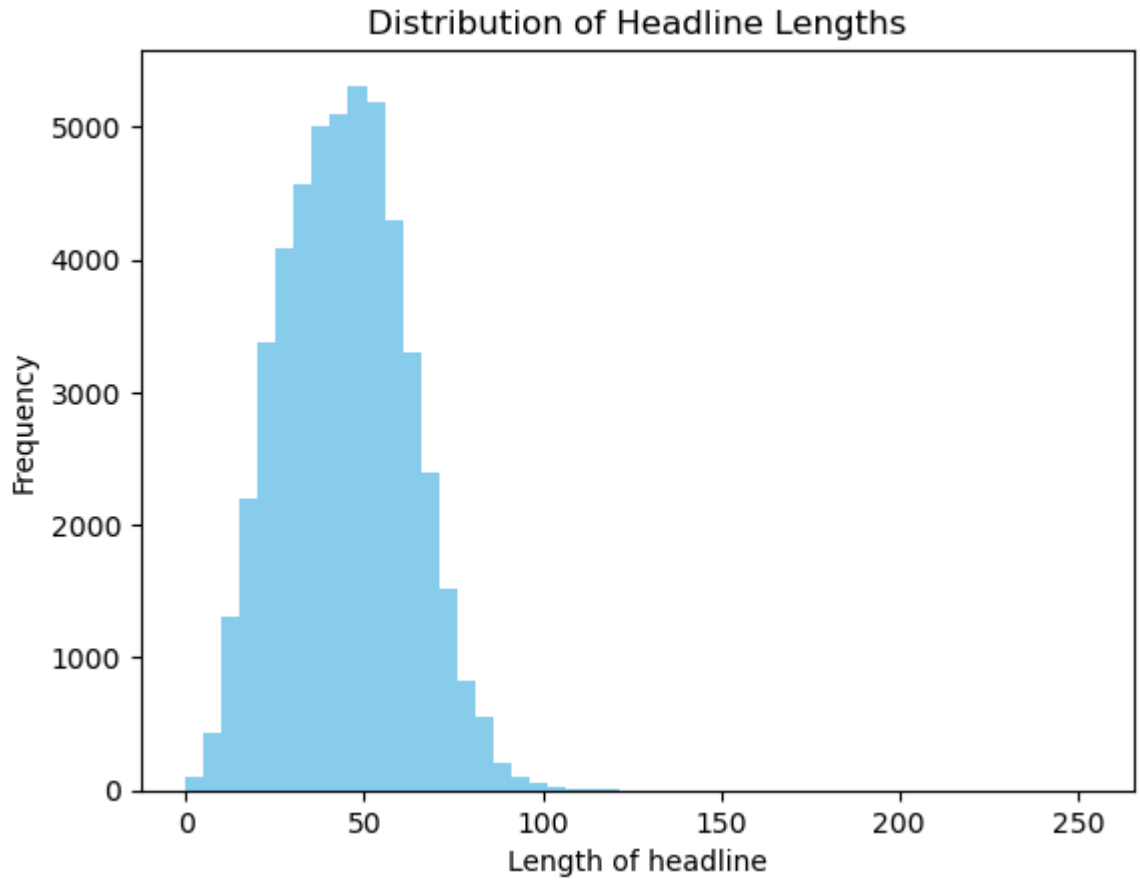
```
headline_length: 44.48926
short_description_length: 90.08196
```

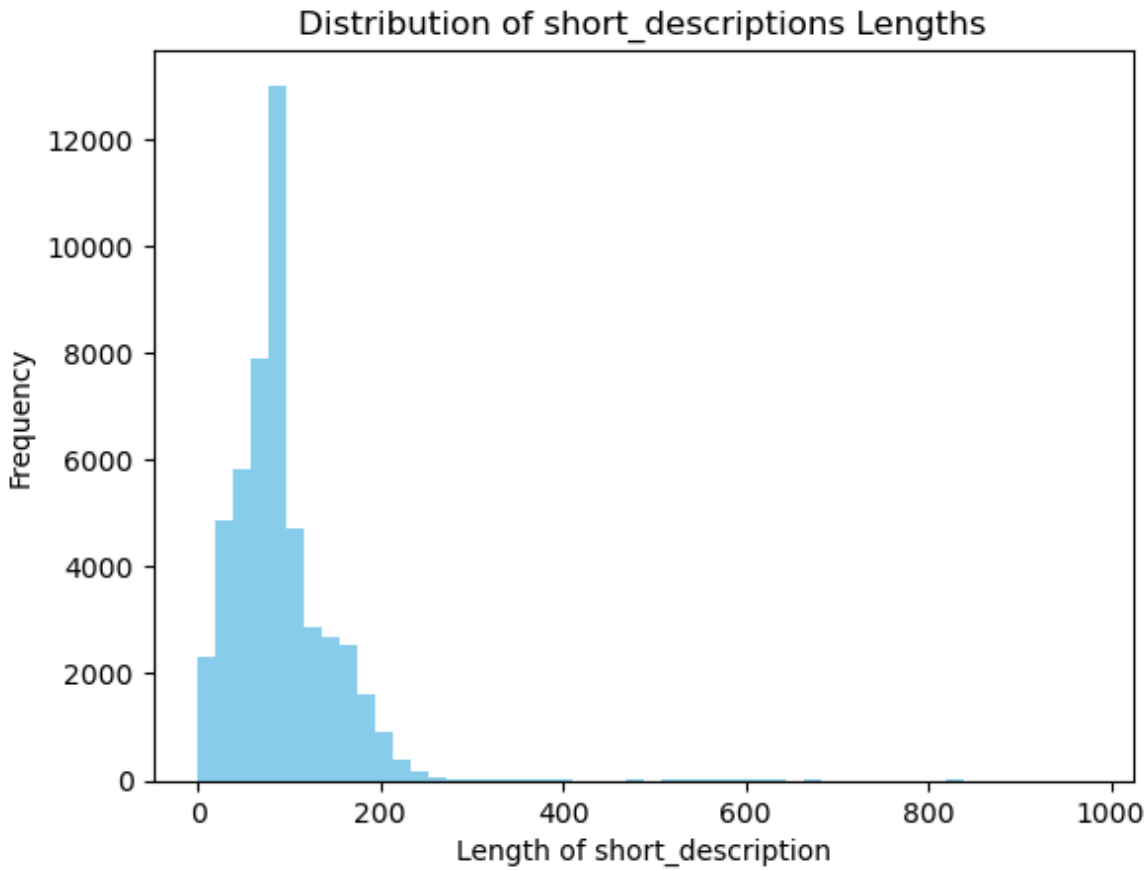
In [32]:
```
# Visualize category distribution

plt.figure(figsize=(12,5))
sns.countplot(x='category', data=df, palette='viridis')
plt.title("Distribution of category")
plt.show()

# Visualize headline length distribution
plt.hist(df['headline_length'], bins=50, color='skyblue')
plt.title("Distribution of Headline Lengths")
plt.xlabel("Length of headline")
plt.ylabel("Frequency")
plt.show()

# Visualize short_decription length distribution
plt.hist(df['short_description_length'], bins=50, color='skyblue')
plt.title("Distribution of short_descriptions Lengths")
plt.xlabel("Length of short_description")
```

```
plt.ylabel("Frequency")
plt.show()
```

```
C:\Users\gajra\AppData\Local\Temp\ipykernel_23164\2766346793.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='category', data=df, palette='viridis')
```







In [33]:
```
# Combine headline and short description into one text field
df['combined_text'] = df['headline'] + " " + df['short_description']

# Define features and labels
X = df['combined_text']
y = df['category']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_


# Vectorization (Convert text to numerical data using TF-IDF)

vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

In [34]:
```
# Importing necessary library for Logistic Regression

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [35]:
```
# Building the Logistic Regression Model

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_tfidf, y_train)
```

Out[35]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [36]:
```
# Making Predictions for Logistic Regression

y_pred_logreg = logreg_model.predict(X_test_tfidf)
```

In [37]:
```
# Evaluating the Logistic Regression Model

print("Logistic Regression Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_logreg)}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_logreg))
```

```
Logistic Regression Evaluation:
Accuracy: 0.7868

Classification Report:
                precision    recall  f1-score   support

      business       0.73      0.75      0.74      1000
 entertainment       0.77      0.78      0.77      1000
  food & drink       0.83      0.85      0.84      1000
     parenting       0.77      0.76      0.77      1000
      politics       0.75      0.73      0.74      1000
        sports       0.87      0.87      0.87      1000
style & beauty       0.87      0.83      0.85      1000
        travel       0.80      0.78      0.79      1000
      wellness       0.71      0.76      0.73      1000
    world news       0.78      0.77      0.77      1000

      accuracy                           0.79     10000
     macro avg       0.79      0.79      0.79     10000
  weighted avg       0.79      0.79      0.79     10000
```

In [38]:
```
# Importing necessary library for Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier
```

In [39]:
```
# Building the Random Forest Classifier Model

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_tfidf, y_train)
```

Out[39]:
```
RandomForestClassifier(random_state=42)
```

In [40]:
```
# Making Predictions for Random Forest Classifier

y_pred_rf = rf_model.predict(X_test_tfidf)
```

In [41]:
```
# Evaluating the Random Forest Classifier Model

print("\nRandom Forest Classifier Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier Evaluation:
Accuracy: 0.73

Classification Report:
                precision    recall  f1-score   support

      business       0.78      0.82      0.80      1000
 entertainment       0.70      0.68      0.69      1000
  food & drink       0.71      0.81      0.76      1000
     parenting       0.73      0.72      0.73      1000
      politics       0.74      0.63      0.68      1000
        sports       0.83      0.92      0.87      1000
style & beauty       0.76      0.76      0.76      1000
        travel       0.76      0.62      0.68      1000
      wellness       0.60      0.68      0.63      1000
    world news       0.70      0.67      0.69      1000

      accuracy                           0.73     10000
     macro avg       0.73      0.73      0.73     10000
  weighted avg       0.73      0.73      0.73     10000
```

In [42]:
```python
# importing multinomialnb and predicting and calculating accuracy

from sklearn.naive_bayes import MultinomialNB

nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
y_pred_nb = nb_model.predict(X_test_tfidf)

print("\nNaive Bayes Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_nb):.3f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Evaluation:
Accuracy: 0.769

Classification Report:
                precision    recall  f1-score   support

      business       0.73      0.69      0.71      1000
 entertainment       0.77      0.74      0.76      1000
  food & drink       0.79      0.86      0.82      1000
     parenting       0.68      0.77      0.72      1000
      politics       0.76      0.73      0.75      1000
        sports       0.87      0.82      0.85      1000
style & beauty       0.88      0.80      0.84      1000
        travel       0.75      0.78      0.77      1000
      wellness       0.70      0.73      0.72      1000
    world news       0.78      0.77      0.78      1000

      accuracy                           0.77     10000
     macro avg       0.77      0.77      0.77     10000
  weighted avg       0.77      0.77      0.77     10000
```

In [43]:
```python
# importing linearsvc and predicting and calculating accuracy

from sklearn.svm import LinearSVC
```

```
svm_model = LinearSVC()
svm_model.fit(X_train_tfidf, y_train)
y_pred_svm = svm_model.predict(X_test_tfidf)

print("\nSupport Vector Machine Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm):.3f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_svm))
```

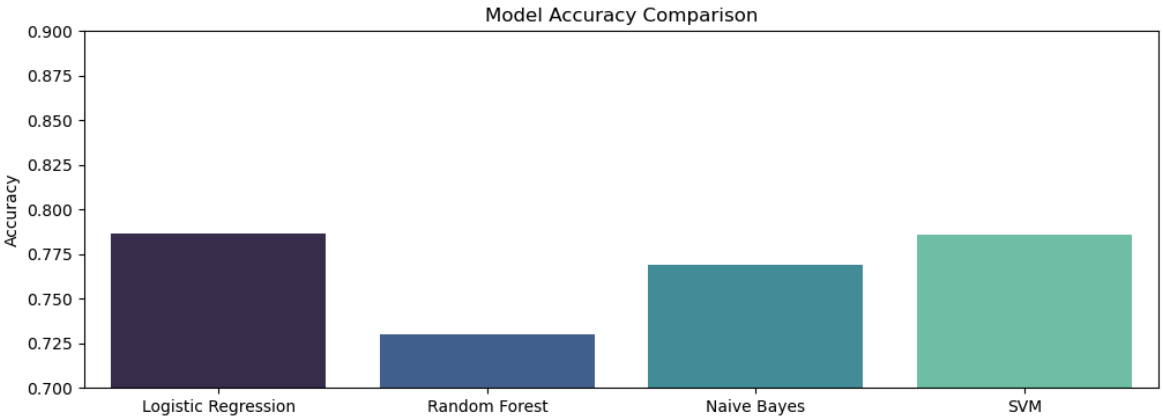
```
C:\Users\gajra\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:31: FutureWarn
ing: The default value of `dual` will change from `True` to `'auto'` in 1.5. Set
the value of `dual` explicitly to suppress the warning.
  warnings.warn(
Support Vector Machine Evaluation:
Accuracy: 0.786

Classification Report:
                precision    recall  f1-score   support

      business       0.74      0.78      0.76      1000
 entertainment       0.76      0.75      0.76      1000
  food & drink       0.83      0.84      0.83      1000
     parenting       0.76      0.76      0.76      1000
      politics       0.76      0.72      0.74      1000
        sports       0.86      0.91      0.88      1000
style & beauty       0.85      0.83      0.84      1000
        travel       0.78      0.78      0.78      1000
      wellness       0.73      0.72      0.73      1000
    world news       0.78      0.76      0.77      1000

      accuracy                           0.79     10000
     macro avg       0.79      0.79      0.79     10000
  weighted avg       0.79      0.79      0.79     10000
```

In [44]:

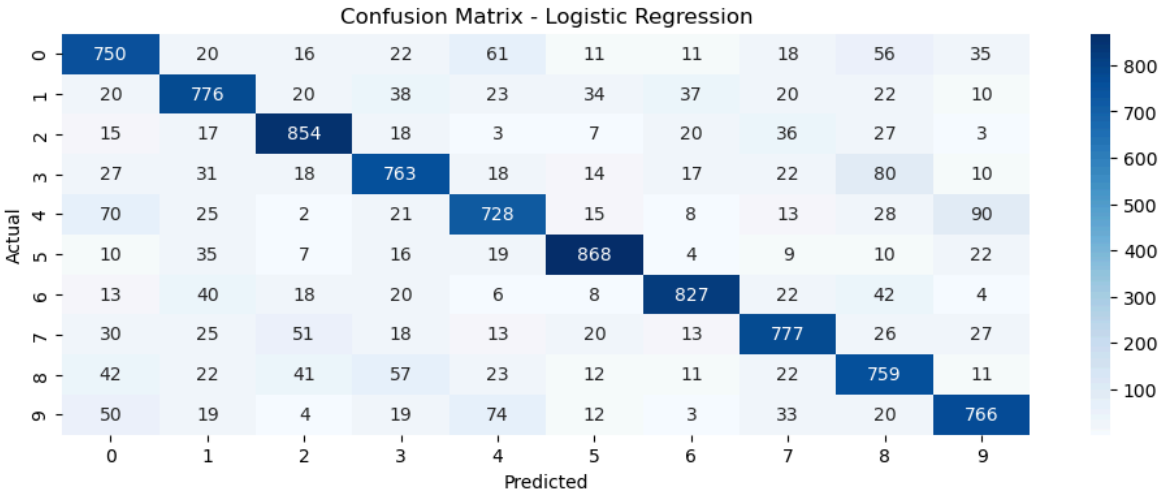
```
from sklearn.metrics import confusion_matrix

# Compare accuracies
model_names = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'SVM']
accuracies = [
    accuracy_score(y_test, y_pred_logreg),
    accuracy_score(y_test, y_pred_rf),
    accuracy_score(y_test, y_pred_nb),
    accuracy_score(y_test, y_pred_svm)
]

plt.figure(figsize=(12,4))
sns.barplot(x=model_names, y=accuracies, palette='mako')
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0.7, .9)
plt.show()

print("****************************************")
print("****************************************")

# Confusion Matrix for best models (Logistic Regression)
plt.figure(figsize=(12,4))
sns.heatmap(confusion_matrix(y_test, y_pred_logreg), annot=True, fmt='d', cmap='
```

```
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

print("****************************************")
print("****************************************")

# Confusion Matrix for best models (LinearSVC)
plt.figure(figsize=(12,4))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Blu
plt.title("Confusion Matrix - LinearSVC")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

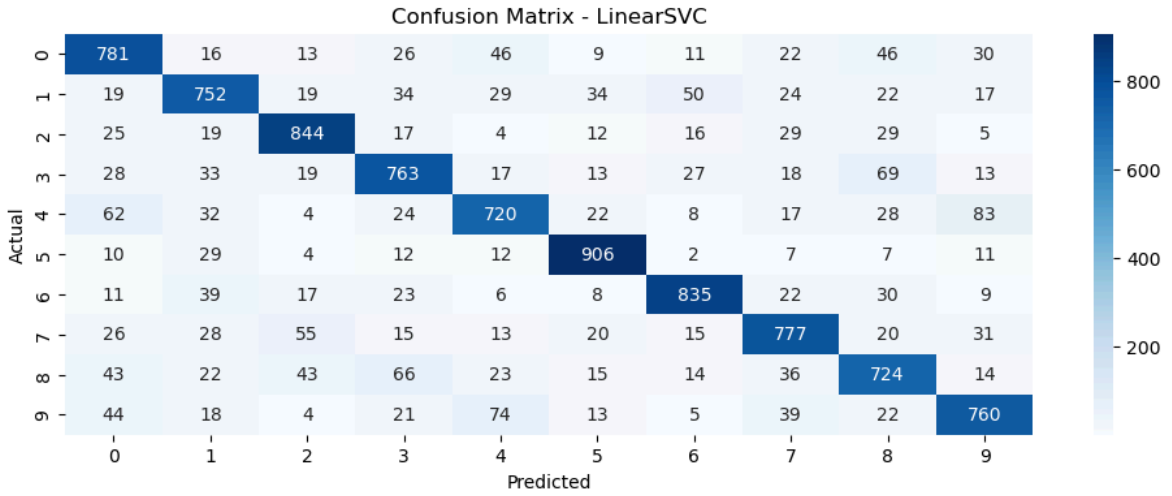
```
C:\Users\gajra\AppData\Local\Temp\ipykernel_23164\2563959306.py:13: FutureWarnin
g:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=model_names, y=accuracies, palette='mako')
```




```
****************************************
****************************************
```




```
****************************************
****************************************
```



In [45]:
```
print("""
✅ Insights Summary (Auto Output)
--------------------------------
1. Most categories are balanced in the dataset.
2. TF-IDF (1,2)-grams worked effectively for capturing news context.
3. Logistic Regression and SVM gave top performance (~0.79 accuracy).
4. Random Forest and Naive Bayes slightly lower due to high-dimensional sparsity
5. Sports and Style categories are easiest to classify.
6. Politics and Wellness have overlapping vocabularies, causing confusion.
7. Model can generalize well for unseen data with slight fine-tuning.
""")
```

```
✅ Insights Summary (Auto Output)
--------------------------------
1. Most categories are balanced in the dataset.
2. TF-IDF (1,2)-grams worked effectively for capturing news context.
3. Logistic Regression and SVM gave top performance (~0.79 accuracy).
4. Random Forest and Naive Bayes slightly lower due to high-dimensional sparsity.
5. Sports and Style categories are easiest to classify.
6. Politics and Wellness have overlapping vocabularies, causing confusion.
7. Model can generalize well for unseen data with slight fine-tuning.
```

In [ ]:
```
# video link

# https://drive.google.com/file/d/1dSbCDIBxI-iBo-jNuVms328XnwuKymL0/view?usp=dri
```

In [ ]:
```
```