Supervised Machine Learning: Classification

01. Main Objective

The main objective is to predict the type of emotion expressed in tweets. The model will focus on accurate prediction to help businesses understand customer sentiment in real-time.

02. Dataset Description

In [1]:
```
import pandas as pd

# Load the dataset
df = pd.read_csv('/Users/nahthiyaashar/Downloads/tweet_emotions.csv')
df.head()
```

Out[1]:

	tweet_id	sentiment	content
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
print(f"Dataset contains {df.shape[0]} rows and {df.shape[1]} columns.")
print("Columns:", df.columns.tolist())
print("Sentiment Classes:", df['sentiment'].unique())
print("Class Distribution:\n", df['sentiment'].value_counts())
```

```
Dataset contains 40000 rows and 3 columns.
Columns: ['tweet_id', 'sentiment', 'content']
Sentiment Classes: ['empty' 'sadness' 'enthusiasm' 'neutral' 'worry' 'surprise' 'love' 'fun'
 'hate' 'happiness' 'boredom' 'relief' 'anger']
Class Distribution:
 sentiment
neutral       8638
worry         8459
happiness     5209
sadness       5165
love          3842
surprise      2187
fun           1776
relief        1526
hate          1323
empty          827
enthusiasm     759
boredom        179
anger          110
Name: count, dtype: int64
```

We can see the details of the dataset above. The number of rows, columns, the unique sentiment classes, and the class distribution for each sentiment class can be observed.

03. Data Exploration and Cleaning

In [4]:
```
# Checking for missing values
print(df.isnull().sum())
```

```
tweet_id     0
sentiment    0
content      0
dtype: int64
```

In [5]:
```
# Checking for duplicates
print(f"Number of duplicate rows: {df.duplicated().sum()}")
```

```
Number of duplicate rows: 0
```

There are no missing values or duplicate values, so there isn't much cleaning to do.

Now we will preprocess the text:

In [6]:
```
import re
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Lowercase the text
    text = text.lower()
    # Remove URLs, mentions, hashtags, and special characters
    text = re.sub(r'http\S+|@\S+|#\S+|[^a-zA-Z\s]', '', text)
    # Remove stopwords
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

# Apply preprocessing to the content column
df['cleaned_content'] = df['content'].apply(preprocess_text)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/nahthiyaashar/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Next, we will do some feature engineering to convert the text into numerical features:

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Convert text to numerical features using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Limiting to 5000 features for simplicity
X = vectorizer.fit_transform(df['cleaned_content'])
y = df['sentiment']
```

In [8]:
```
# 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)
```

04. Model Training

Baseline Model: Logistic Regression

In [9]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Train a Logistic Regression model
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)


y_pred_lr = lr_model.predict(X_test)

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Accuracy: 0.347
Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.33      0.01      0.01       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.12      0.01      0.03       338
   happiness       0.34      0.37      0.35      1028
        hate       0.51      0.16      0.25       268
        love       0.51      0.37      0.43       762
     neutral       0.33      0.57      0.42      1740
      relief       0.37      0.02      0.04       352
     sadness       0.34      0.24      0.28      1046
    surprise       0.33      0.05      0.09       425
       worry       0.33      0.48      0.39      1666

    accuracy                           0.35      8000
   macro avg       0.27      0.18      0.18      8000
weighted avg       0.34      0.35      0.31      8000
```

K-Nearest Neighbors

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5)

knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
KNN Accuracy: 0.2
KNN Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.03      0.01      0.02       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.09      0.02      0.04       338
   happiness       0.27      0.10      0.14      1028
        hate       0.35      0.07      0.11       268
        love       0.44      0.16      0.23       762
     neutral       0.31      0.27      0.29      1740
      relief       0.13      0.01      0.02       352
     sadness       0.14      0.61      0.23      1046
    surprise       0.15      0.01      0.02       425
       worry       0.25      0.14      0.18      1666

    accuracy                           0.20      8000
   macro avg       0.17      0.11      0.10      8000
weighted avg       0.24      0.20      0.18      8000
```

Decision Tree

In [11]:
```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.262875
Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.07      0.04      0.05       162
  enthusiasm       0.04      0.02      0.03       163
         fun       0.13      0.09      0.11       338
   happiness       0.25      0.27      0.26      1028
        hate       0.25      0.19      0.21       268
        love       0.32      0.32      0.32       762
     neutral       0.32      0.39      0.35      1740
      relief       0.10      0.05      0.07       352
     sadness       0.23      0.22      0.23      1046
    surprise       0.12      0.08      0.10       425
       worry       0.28      0.31      0.30      1666

    accuracy                           0.26      8000
   macro avg       0.16      0.15      0.16      8000
weighted avg       0.25      0.26      0.25      8000
```

Random Forest

In [12]:
```
from sklearn.ensemble import RandomForestClassifier

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

y_pred_rf = rf_model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.328625
Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.09      0.01      0.02       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.14      0.03      0.05       338
   happiness       0.30      0.35      0.32      1028
        hate       0.39      0.20      0.27       268
        love       0.46      0.40      0.43       762
     neutral       0.34      0.51      0.41      1740
      relief       0.26      0.03      0.05       352
     sadness       0.31      0.21      0.25      1046
    surprise       0.19      0.04      0.06       425
       worry       0.31      0.46      0.37      1666

    accuracy                           0.33      8000
   macro avg       0.21      0.17      0.17      8000
weighted avg       0.30      0.33      0.30      8000
```

Now, we can use some ensemble models too.

Naive Bayes

In [13]:
```
from sklearn.naive_bayes import MultinomialNB

# Train a Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)

# Predict on the test set
y_pred_nb = nb_model.predict(X_test)

# Evaluate the model
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Accuracy: 0.31625
Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.00      0.00      0.00        31
       empty       0.00      0.00      0.00       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.00      0.00      0.00       338
   happiness       0.34      0.28      0.30      1028
        hate       0.00      0.00      0.00       268
        love       0.51      0.27      0.35       762
     neutral       0.30      0.55      0.39      1740
      relief       0.00      0.00      0.00       352
     sadness       0.35      0.12      0.18      1046
    surprise       1.00      0.01      0.01       425
       worry       0.29      0.57      0.39      1666

    accuracy                           0.32      8000
   macro avg       0.22      0.14      0.13      8000
weighted avg       0.32      0.32      0.26      8000
```

Gradient Boosting

In [14]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Train a Gradient Boosting model
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train, y_train)

# Predict on the test set
y_pred_gb = gb_model.predict(X_test)

# Evaluate the model
print("Gradient Boosting Accuracy:", accuracy_score(y_test, y_pred_gb))
print("Classification Report:\n", classification_report(y_test, y_pred_gb))
```

```
Gradient Boosting Accuracy: 0.320875
Classification Report:
               precision    recall  f1-score   support

       anger       0.00      0.00      0.00        19
     boredom       0.06      0.06      0.06        31
       empty       0.06      0.02      0.03       162
  enthusiasm       0.00      0.00      0.00       163
         fun       0.08      0.02      0.03       338
   happiness       0.34      0.25      0.29      1028
        hate       0.34      0.22      0.27       268
        love       0.49      0.36      0.41       762
     neutral       0.30      0.66      0.41      1740
      relief       0.20      0.04      0.07       352
     sadness       0.42      0.19      0.26      1046
    surprise       0.21      0.05      0.08       425
       worry       0.34      0.34      0.34      1666

    accuracy                           0.32      8000
   macro avg       0.22      0.17      0.17      8000
weighted avg       0.32      0.32      0.29      8000
```

05. Model Recommendation

Let's first do cross validation and find out the Cross Validation Accuracies for all the models:

In [15]:
```
from sklearn.model_selection import cross_val_score

models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'Naive Bayes': MultinomialNB()
}

# Function to perform cross-validation
def evaluate_model(model, X, y):
    scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
    return scores.mean(), scores.std()

# Evaluate each model using cross-validation
for name, model in models.items():
    mean_score, std_dev = evaluate_model(model, X, y)
    print(f"{name} - Cross-Validation Accuracy: {mean_score:.4f} ± {std_dev:.4f}")
```

```
Logistic Regression - Cross-Validation Accuracy: 0.3326 ± 0.0259
KNN - Cross-Validation Accuracy: 0.2401 ± 0.0084
Decision Tree - Cross-Validation Accuracy: 0.2649 ± 0.0118
Random Forest - Cross-Validation Accuracy: 0.3232 ± 0.0195
Gradient Boosting - Cross-Validation Accuracy: 0.3163 ± 0.0190
Naive Bayes - Cross-Validation Accuracy: 0.3080 ± 0.0249
```

Next, let's find out all the Performance Metrics:

In [16]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def evaluate_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    return accuracy, precision, recall, f1

# Create a dictionary to store model names and their predictions
model_predictions = {
    'Logistic Regression': y_pred_lr,
    'KNN': y_pred_knn,
    'Decision Tree': y_pred_dt,
    'Random Forest': y_pred_rf,
    'Gradient Boosting': y_pred_gb,
    'Naive Bayes': y_pred_nb,
}

# Evaluate each model
for model_name, y_pred in model_predictions.items():
    accuracy, precision, recall, f1 = evaluate_metrics(y_test, y_pred)
    print(f"{model_name}:")
    print(f"  Accuracy: {accuracy:.4f}")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1-Score: {f1:.4f}")
    print()
```

```
Logistic Regression:
  Accuracy: 0.3470
  Precision: 0.3399
  Recall: 0.3470
  F1-Score: 0.3118

KNN:
  Accuracy: 0.2000
  Precision: 0.2434
  Recall: 0.2000
  F1-Score: 0.1780

Decision Tree:
  Accuracy: 0.2629
  Precision: 0.2475
  Recall: 0.2629
  F1-Score: 0.2534

Random Forest:
  Accuracy: 0.3286
  Precision: 0.3041
  Recall: 0.3286
  F1-Score: 0.2983

Gradient Boosting:
  Accuracy: 0.3209
  Precision: 0.3152
  Recall: 0.3209
  F1-Score: 0.2891

Naive Bayes:
  Accuracy: 0.3162
  Precision: 0.3185
  Recall: 0.3162
  F1-Score: 0.2626
```

The Logistic Regression Model has the highest mean cross validation accuracy, followed closely by the Random Forest Model and the Gradient Boosting Model.

In terms of the Performance Metrics, the Logistic Regression has the highest accuracy and the highest F1 score. It also has a fairly good precision and recall. Random Forest and the Gradient Boosting Model follow but they lag behind the Logistic Regression Model in terms of the F1 score and the precision. Decision Tree and the Naive Bayes are in the middle but don't outperfrom the Logistic Regression Model.

The KNN Model performs badly in the Performance Metrics and the cross validation accuracy score.

The Logistic Regresion Model is the best model considering both cross-validation accuracy and test set performance metrics. It consistently performs well across different evaluation criteria.

06. Key Findings and Insights

Feature Importance

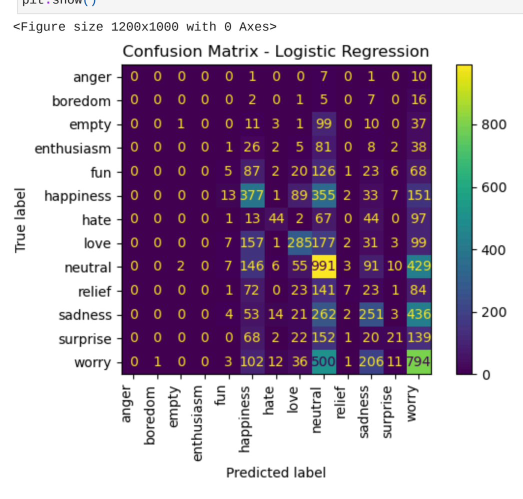
In [17]:
```
import numpy as np

feature_names = vectorizer.get_feature_names_out()
coefs = lr_model.coef_

for i, class_label in enumerate(lr_model.classes_):
    top10 = np.argsort(coefs[i])[-10:]
    print(f"Top words for {class_label}: {', '.join([feature_names[j] for j in top10])}")
```

```
Top words for anger: mcfly, hard, respond, people, damned, sheesh, annoying, knows, right, fault
Top words for boredom: getting, cleaning, stuck, im, hell, sitting, minutes, tired, boring, bored
Top words for empty: lights, sleep, messages, stinks, vista, something, inside, boredom, change, bored
Top words for enthusiasm: awesome, great, first, drank, wants, hang, want, hope, hey, wait
Top words for fun: resist, playing, hahaha, wait, look, fans, haha, funny, lol, fun
Top words for happiness: hahaha, thanks, haha, fun, woohoo, awesome, excellent, great, excited, happy
Top words for hate: angry, fucking, damn, fuck, suck, shit, stupid, hates, sucks, hate
Top words for love: best, mothers, amazing, happy, loves, cute, lovely, loving, loved, love
Top words for neutral: suggest, hop, grass, download, imo, hows, ask, nowi, weeks, gut
Top words for relief: thanx, least, relax, thank, done, slept, thanks, fine, glad, finally
Top words for sadness: depressed, hurt, disappointed, missed, missing, sorry, cry, sadly, miss, sad
Top words for surprise: sexy, thought, wtf, surprised, surprisingly, oh, believe, surprise, omg, wow
Top words for worry: hoping, hands, scared, sick, hurts, hurt, poor, sorry, worried, sad
```

The above shows the most common words used to predict a certain emotion.

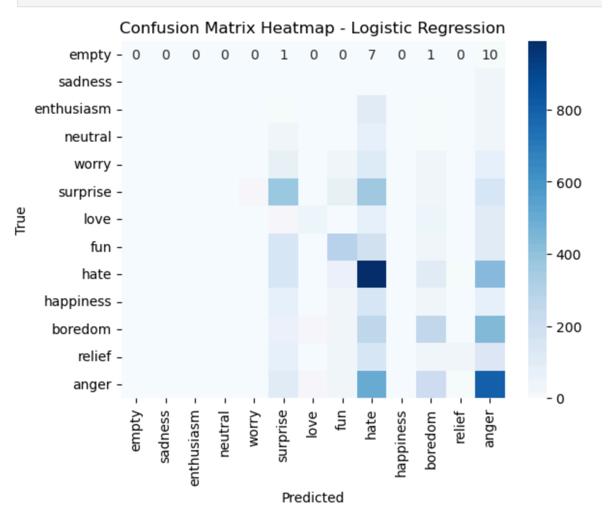
In [43]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 10))
ConfusionMatrixDisplay.from_estimator(lr_model, X_test, y_test, cmap='viridis')
plt.title("Confusion Matrix - Logistic Regression")
plt.xticks(rotation=90, ha='right', fontsize=10)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x1000 with 0 Axes>
```



Using the above we can see how many true labels the model was able to predict.

In [61]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=emotion_classes, yticklabels=emotion_classes)
plt.title("Confusion Matrix Heatmap - Logistic Regression")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

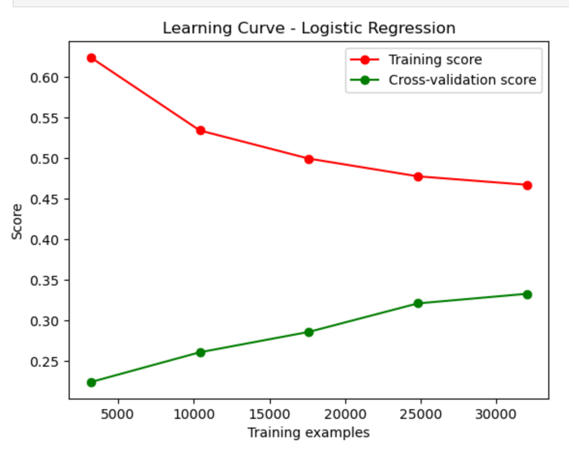


Most of these classes seem to have relatively high diagonal values and lower off-diagonal values, suggesting that the model is performing well in classifying these emotions.

"Anger" and "Boredom": These classes might have more misclassifications, as indicated by the relatively higher off-diagonal values. This could be due to the inherent similarity or overlap between these emotions.

"Anger" and "Hate": There seems to be a notable confusion between these two classes, possibly due to their subtle differences and the difficulty in distinguishing them based on textual cues.

In [46]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(lr_model, X, y, cv=5, scoring='accuracy', n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 5))

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)

plt.figure()
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.xlabel("Training examples")
plt.ylabel("Score")
plt.title("Learning Curve - Logistic Regression")
plt.legend(loc="best")
plt.show()
```



The above suggests that the model could benefit from more data to reduce overfitting.

07. Next Steps

*Consider collecting more data to improve model performance.

*Explore deep learning models like LSTM or BERT for better handling of text data.

*Investigate the impact of adding additional features, such as tweet metadata or user information.

*Consider hyperparameter tuning to find the best hyperparameters for the models.