In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()

# ML Library
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# ML Algorithms (Traditional)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.neighbors import KNeighborsClassifier

# Ml Algorithms (Modern)
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# Hide warnings for cleaner production
import warnings
warnings.filterwarnings('ignore')

# Libraries of text preprocessing
import re
import string
import unicodedata
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

```
# DL Models
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, GRU
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping

# Evaluation Metrics Libraries
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\student7\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\student7\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\student7\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Preprocessing for Datasets:

In [2]:

```
# Load training and validation datasets
df_training = pd.read_csv('twitter_training.csv')
df_val = pd.read_csv('twitter_validation.csv')

# Drop index column that was incorrectly read as a feature
df_training.drop(['2401'], axis=1, inplace=True)
df_training.rename(columns={'Borderlands':'Entity', 'Positive':'Sentiment',
                            'im getting on borderlands and i will murder you all ,': 'Text'}, inplace=True)

# Drop index column from validation data
df_val.drop(['3364'], axis=1, inplace=True)

# Rename columns in validation data to match training format
df_val.rename(columns={'Facebook':'Entity', 'Irrelevant':'Sentiment',
                       'I mentioned on Facebook that I was struggling for motivation to go for a run the other day, which has
```

```
# Combine training and validation datasets into a single DataFrame
df = pd.concat([df_training, df_val], ignore_index=True)
df
```

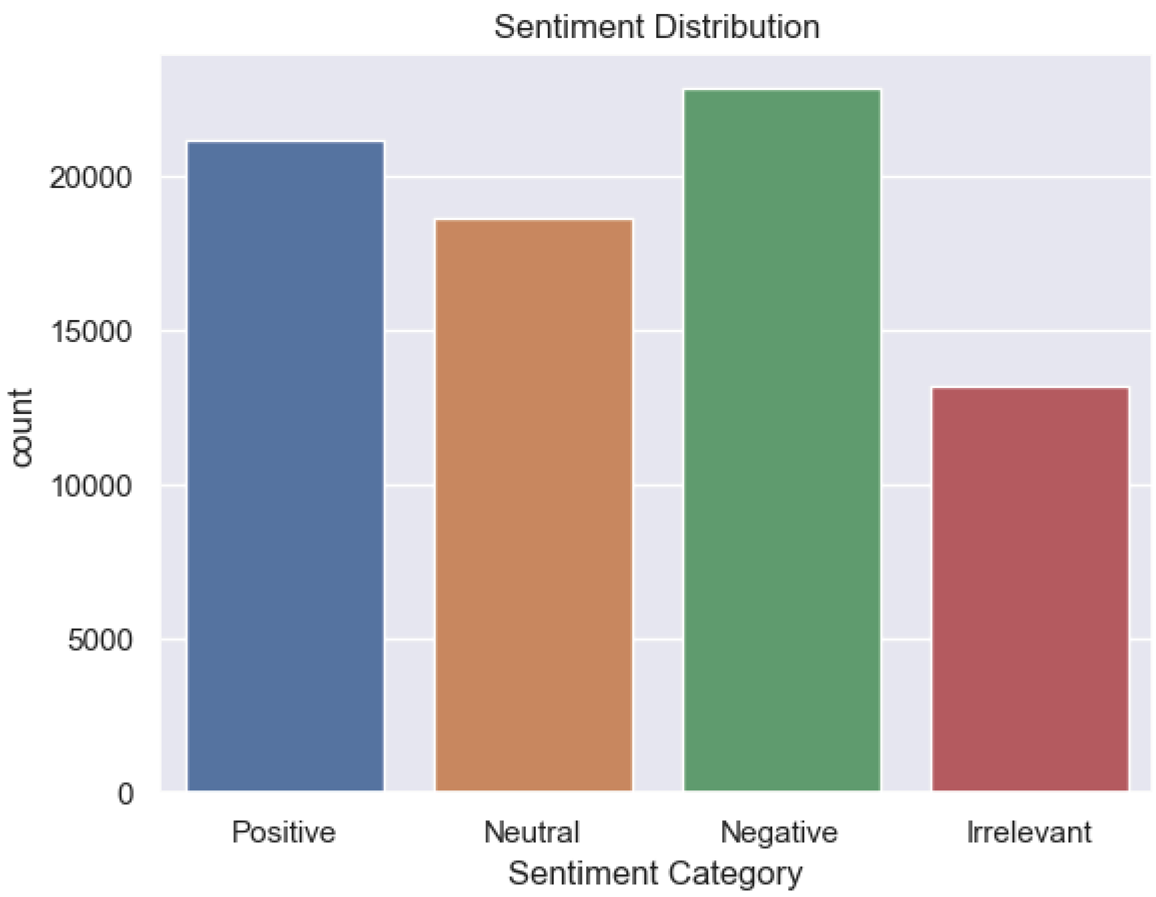
Out[2]:

	Entity	Sentiment	Text
0	Borderlands	Positive	I am coming to the borders and I will kill you...
1	Borderlands	Positive	im getting on borderlands and i will kill you ...
2	Borderlands	Positive	im coming on borderlands and i will murder you...
3	Borderlands	Positive	im getting on borderlands 2 and i will murder ...
4	Borderlands	Positive	im getting into borderlands and i can murder y...
...	...	...	...
75675	GrandTheftAuto(GTA)	Irrelevant	⭐️ Toronto is the arts and culture capital of ...
75676	CS-GO	Irrelevant	tHIS IS ACTUALLY A GOOD MOVE TOT BRING MORE VI...
75677	Borderlands	Positive	Today sucked so it’s time to drink wine n play...
75678	Microsoft	Positive	Bought a fraction of Microsoft today. Small wins.
75679	johnson&johnson	Neutral	Johnson & Johnson to stop selling talc baby po...

75680 rows × 3 columns

In [3]:

```
# Create a count plot showing the distribution of sentiment labels
sns.countplot(data=df, x= 'Sentiment',hue= 'Sentiment', palette='deep')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment Category')
plt.show()
```



Data Cleaning:

In [4]:
```
# Check for missing values
print(df.isna().sum())
print("-"*15)

# Count number of duplicated in the dataset
print(f"Duplicated:{df.duplicated().sum()}")
```

```
Entity          0
Sentiment       0
Text          686
dtype: int64
---------------
Duplicated:4137
```

In [5]:
```python
# Drop any missing values & duplicate
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)

# Drop the 'Entity' column
df.drop(['Entity'], axis=1, inplace=True)

# Remove rows where the sentiment is labeled as 'Irrelevant'
df = df[df['Sentiment'] != 'Irrelevant']

# Reset the index after dropping rows
df.reset_index(drop=True, inplace=True)
```

In [6]:
```python
# Handling the Text:

# 1- Load the set of English stopwords
stop_words = set(stopwords.words('english'))

# 2- Initialize the WordNet lemmatizer for word normalization
lemmatizer = WordNetLemmatizer()

# 3- Delete: Links, Hashtags, Punctuation, numbers, Extra spaces and Emojis
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"@\w+|#\w+", "", text)
    text = re.sub(r"[%s]" % re.escape(string.punctuation), "", text)
    text = re.sub(r"\d+", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    text = ''.join(c for c in text if c.isprintable())
    text = unicodedata.normalize("NFKD", text).encode("ascii", "ignore").decode("utf-8", "ignore")
    return text
```

```
df['Text'] = df['Text'].astype(str).apply(clean_text)

# 4- Define a function to remove stopwords
def remove_stopwords(text):
    return " ".join([word for word in text.split() if word not in stop_words])

# 5- Apply the stopword removal function
df['Text'] = df['Text'].apply(remove_stopwords)

# 6- Encoding for Sentiment column
def convert_to_num (sentiment):
    if sentiment == 'Negative':  # Negative --> 0
        return 0
    elif sentiment == 'Neutral': # Neutral  --> 1
        return 1

    else:                        # Positive --> 2
        return 2

df['Sentiment'] = df['Sentiment'].apply(convert_to_num)
```

In [7]:

```
df
```

Out[7]:

	Sentiment	Text
0	2	coming borders kill
1	2	im getting borderlands kill
2	2	im coming borderlands murder
3	2	im getting borderlands murder
4	2	im getting borderlands murder
...	...	...
58902	0	noticed streamers watch playing games battlefi...
58903	1	playing red dead redemption oh shit bear start...
58904	1	suikoden alex kidd miracle world persona soul ...
58905	2	thank matching funds home depot rw payment gen...
58906	1	late night stream boys come watch warzone runs...

58907 rows × 2 columns

ML:

In [8]:

```
# Feature Selection
X= df['Text']
y= df['Sentiment']
```

In [34]:

```
#Spliting into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [35]:

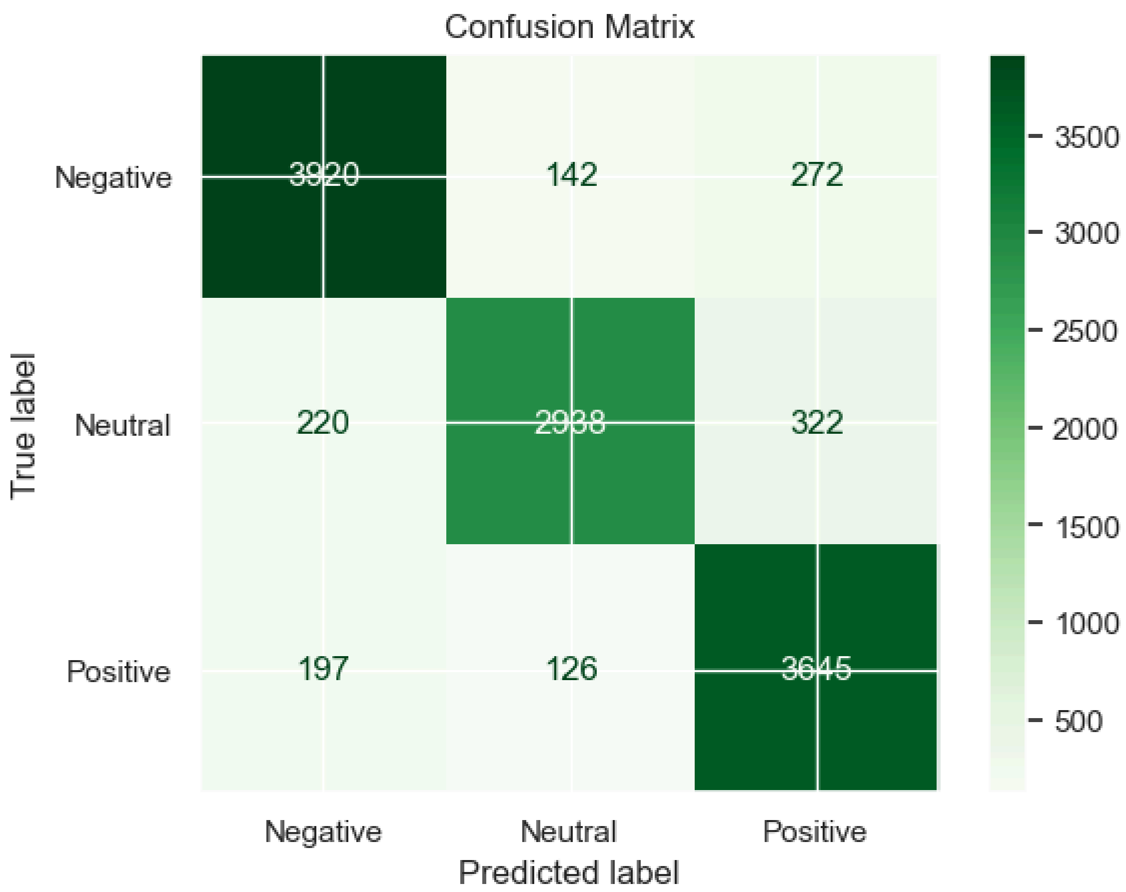
```
# Transforming the text data using TF-IDF:
tfidf = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

In [20]:
```
models = {
        "Logistic Regression": LogisticRegression(),
        "CatBoost": CatBoostClassifier(verbose=0),
        "Gradient Boosting": GradientBoostingClassifier(),
        "Naive Bayes Multinomia": MultinomialNB(),
        "KNN": KNeighborsClassifier(),
        "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss'),
        "LightGBM": LGBMClassifier(verbose=0),
        "Random Forest": RandomForestClassifier()
    }

best_model_name = None
best_accuracy = 0
best_model = None
best_report = ""

for name, model in models.items():
    model.fit(X_train_tfidf, y_train)
    y_pred = model.predict(X_test_tfidf)
    acc = accuracy_score(y_test, y_pred)

    print(f"{name} Accuracy: {acc:.2f}")

    if acc > best_accuracy:
        best_accuracy = acc
        best_model_name = name
        best_model = model
        best_report = classification_report(y_test, y_pred, target_names=target_names if 'target_names' in locals() else None)

print("\n" + "=" * 60)
print(f"Best Model: {best_model_name} with Accuracy: {best_accuracy:.2f}")
print("Classification Report for the Best Model:")
print(best_report)
```

```
Logistic Regression Accuracy: 0.76
CatBoost Accuracy: 0.70
Gradient Boosting Accuracy: 0.60
Naive Bayes Multinomia Accuracy: 0.71
KNN Accuracy: 0.88
XGBoost Accuracy: 0.70
LightGBM Accuracy: 0.70
Random Forest Accuracy: 0.89

============================================================
Best Model: Random Forest with Accuracy: 0.89
Classification Report for the Best Model:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      4334
           1       0.92      0.84      0.88      3480
           2       0.86      0.92      0.89      3968

    accuracy                           0.89     11782
   macro avg       0.89      0.89      0.89     11782
weighted avg       0.89      0.89      0.89     11782
```

In [45]:

```python
# confusion matrix for each model to analyze
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negative', 'Neutral', 'Positive'])
disp.plot(cmap='Greens')
plt.title("Confusion Matrix")
plt.show()
```



DL:

1- LSTM:

In [13]:
```
# Tokenization
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(df['Text'])
```

```
sequences = tokenizer.texts_to_sequences(df['Text'])


# Padding
padded = pad_sequences(sequences, maxlen=100)

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(padded, y, test_size=0.2, random_state=42, stratify=y)

# Build LSTM Model
lstm = Sequential()
lstm.add(Embedding(input_dim=10000, output_dim=128))
lstm.add(LSTM(64, return_sequences=False, dropout=0.3))

lstm.add(Dense(200, activation='relu')) # Hidden layer 1
lstm.add(Dense(100, activation='relu')) # Hidden layer 2
lstm.add(Dense(50, activation='relu'))  # Hidden leyer 3
lstm.add(Dense(3, activation='softmax')) # Output layer


# Compile & Train
lstm.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
lstm.fit(X_train, y_train, batch_size=50, epochs=10, validation_data=(X_test, y_test))

# Evaluate
loss, accuracy = lstm.evaluate(X_test, y_test)
print(f'LSTM Accuracy: {accuracy:.2f}')
```

```
Epoch 1/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 51s 51ms/step - accuracy: 0.5850 - loss: 0.8586 - val_accuracy: 0.7901 - val_loss: 0.5263
Epoch 2/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 52s 55ms/step - accuracy: 0.8460 - loss: 0.3979 - val_accuracy: 0.8336 - val_loss: 0.4194
Epoch 3/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 51s 54ms/step - accuracy: 0.8916 - loss: 0.2716 - val_accuracy: 0.8543 - val_loss: 0.3945
Epoch 4/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 51s 54ms/step - accuracy: 0.9143 - loss: 0.2085 - val_accuracy: 0.8617 - val_loss: 0.3821
Epoch 5/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 54s 57ms/step - accuracy: 0.9279 - loss: 0.1721 - val_accuracy: 0.8628 - val_loss: 0.3959
Epoch 6/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 53s 57ms/step - accuracy: 0.9397 - loss: 0.1433 - val_accuracy: 0.8682 - val_loss: 0.4313
Epoch 7/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 54s 57ms/step - accuracy: 0.9447 - loss: 0.1292 - val_accuracy: 0.8712 - val_loss: 0.4372
Epoch 8/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 53s 56ms/step - accuracy: 0.9466 - loss: 0.1186 - val_accuracy: 0.8795 - val_loss: 0.4573
Epoch 9/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 53s 56ms/step - accuracy: 0.9528 - loss: 0.1050 - val_accuracy: 0.8772 - val_loss: 0.4739
Epoch 10/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 54s 57ms/step - accuracy: 0.9558 - loss: 0.0971 - val_accuracy: 0.8832 - val_loss: 0.4506
369/369 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.8828 - loss: 0.4500
LSTM Accuracy: 0.88
```

2- GRU:

In [18]:

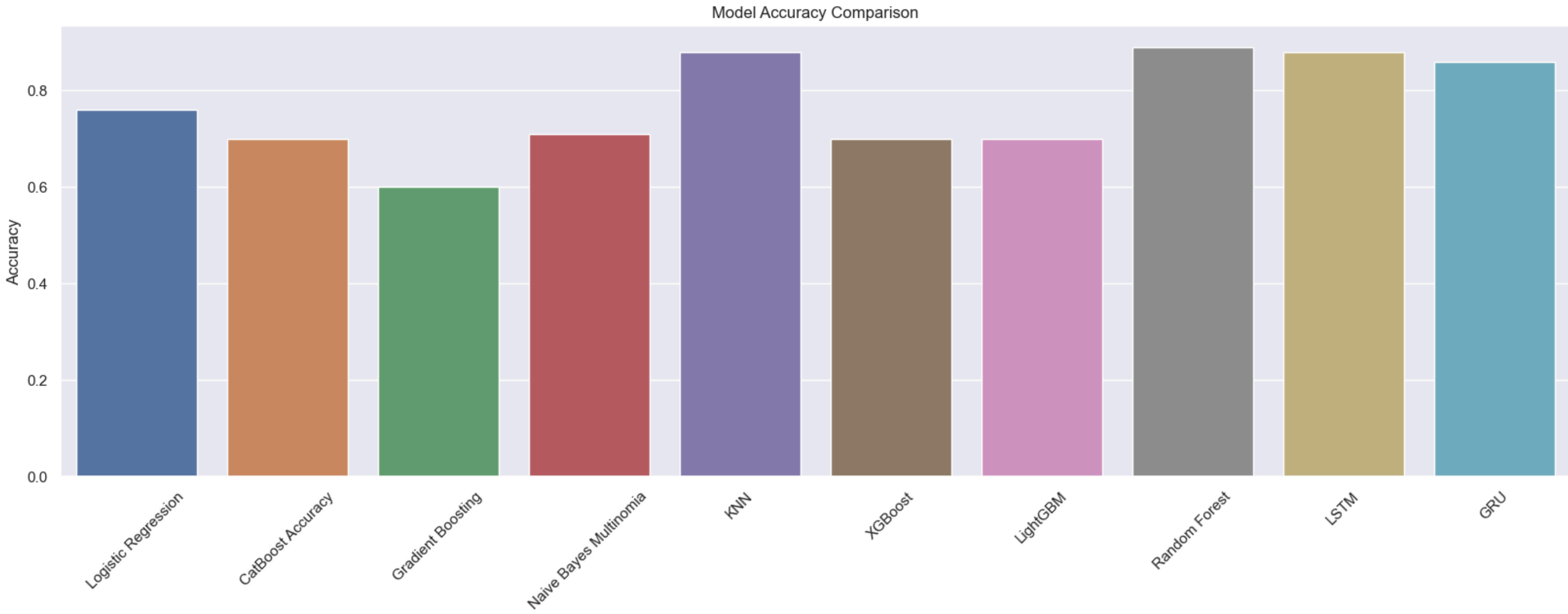
```python
# Build GRU Model
gru = Sequential()
gru.add(Embedding(input_dim=10000, output_dim=128))
gru.add(GRU(64))
gru.add(Dense(3, activation='softmax'))

# Compile & Train
gru.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
gru.fit(X_train, y_train, batch_size=50, epochs=10, validation_data=(X_test, y_test))

# Evaluate
loss, accuracy = gru.evaluate(X_test, y_test)
print(f'GRU Accuracy: {accuracy:.2f}')
```

```
Epoch 1/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 48ms/step - accuracy: 0.6136 - loss: 0.8226 - val_accuracy: 0.8040 - val_loss: 0.4970
Epoch 2/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 45s 48ms/step - accuracy: 0.8534 - loss: 0.3701 - val_accuracy: 0.8398 - val_loss: 0.4104
Epoch 3/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 45s 48ms/step - accuracy: 0.9082 - loss: 0.2391 - val_accuracy: 0.8529 - val_loss: 0.4005
Epoch 4/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9270 - loss: 0.1844 - val_accuracy: 0.8597 - val_loss: 0.3918
Epoch 5/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9393 - loss: 0.1538 - val_accuracy: 0.8631 - val_loss: 0.4091
Epoch 6/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9487 - loss: 0.1269 - val_accuracy: 0.8655 - val_loss: 0.4332
Epoch 7/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9525 - loss: 0.1138 - val_accuracy: 0.8685 - val_loss: 0.4559
Epoch 8/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9564 - loss: 0.0985 - val_accuracy: 0.8715 - val_loss: 0.4634
Epoch 9/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 46s 49ms/step - accuracy: 0.9623 - loss: 0.0880 - val_accuracy: 0.8723 - val_loss: 0.4914
Epoch 10/10
943/943 ━━━━━━━━━━━━━━━━━━━━ 47s 50ms/step - accuracy: 0.9676 - loss: 0.0754 - val_accuracy: 0.8635 - val_loss: 0.5542
369/369 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8647 - loss: 0.5602
GRU Accuracy: 0.86
```

In [14]:

```python
# Finally: model accuracy comparison
plt.figure(figsize=(20,6))
model_scores = {
    "Logistic Regression": 0.76,
    "CatBoost Accuracy": 0.70,
    "Gradient Boosting": 0.60,
    "Naive Bayes Multinomia": 0.71,
    "KNN": 0.88,
    "XGBoost": 0.70,
    "LightGBM": 0.70,
    "Random Forest": 0.89,
    'LSTM': 0.88,
    'GRU': 0.86
                }
sns.barplot(x= list(model_scores.keys()), y= list(model_scores.values()), palette='deep')
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
```

```
plt.xticks(rotation=45)
plt.show()
```



Summary:

This project involved multi-class sentiment analysis on tweets mentioning various brands (Amazon, Nvidia, etc.). After extensive preprocessing, both Machine Learning (Logistic Regression, Random Forest, KNN, etc.) and Deep Learning (LSTM, GRU) models were applied.

Best Model: Random Forest with 89% accuracy
Best Deep Learning Model: LSTM with 88% accuracy

The models were compared visually using bar plots. Future improvements could include advanced hyperparameter tuning, ensemble methods, and domain-specific word embeddings to improve DL performance.