

## **PROJECT REPORT (CSN304)**

A report submitted in partial fulfilment of the requirement for the course

# **ARTIFICIAL INTELLIGENCE**

Part of the degree of

**BACHELOR OF TECHNOLOGY  
COMPUTER SCIENCE AND ENGINEERING**



**Submitted to:**

Ms. Shabnam, Assistant Professor

**Submitted by:**

Soham Sharma 20436

Anurag Kumar 20424

Akshit Rayal 21502

**SCHOOL OF COMPUTING  
DIT UNIVERSITY, DEHRADUN**

(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

Mussoorie Diversion Road, Dehradun, Uttarakhand – 248009,  
India.

# Declaration

I hereby declare that the work which is being presented in the report entitled "**Sentiment Analysis on Tweets using Scikit-learn and NLTK**" in partial fulfilment of the requirement as part of the course **Bachelor of Technology (Computer Science and Engineering)** submitted to DIT, Dehradun is an authentic record of my work carried out during the period **25/10/31** to **4/10/31** under the guidance of **Ms. Shabnam, Assistant Professor, School of Computing.**

Name of the Student Soham Sharma      Signature and Date \_\_\_\_\_

Name of the Student Anurag Kumar      Signature and Date \_\_\_\_\_

Name of the Student Akshit Rayal      Signature and Date \_\_\_\_\_

# Acknowledgement

We are sincerely grateful to all who have supported us in this project. First, we extend our heartfelt thanks to our project guide, **Ms Shabnam**, for her invaluable guidance, dedication, and encouragement throughout this endeavor. Her expertise and thoughtful insights were instrumental in shaping our progress.

We also thank each member of our team, whose hard work, collaboration, and commitment made this project possible. Your support has been invaluable, and we deeply appreciate your efforts.

Thank you all for helping make this project a success

**Soham Sharma** - 23A03A0440

**Anurag Kumar** - 23A03D0436

**Akshit Rayal** - 23A03A0773

# Table of Contents

<b>CHAPTER</b>	<b>PAGE No.</b>
1. Introduction	5
2. Requirement Analysis and System Specification	6
3. System Design	7
4. Implementation Modules	8
5. Results and Discussion	10
6. Summary and Conclusion	21
7. References	22

# Chapter 1 – Introduction

## Introduction

Sentiment analysis is a key technique in natural language processing (NLP) used to determine the emotional tone behind a body of text. With the rise of social media, platforms like Twitter have become a vast source of public opinion. This project focuses on harnessing this data by developing a machine learning pipeline to automatically analyze and classify the sentiment of these tweets, turning unstructured text into structured insights.

## Objective

The primary objective of this project is to build and evaluate a robust model capable of classifying tweets into four distinct categories: Positive, Negative, Neutral, or Irrelevant. This involves implementing a comprehensive text preprocessing pipeline using NLTK and comparing the performance of five different Scikit-learn models (including Logistic Regression, Naive Bayes, and Linear SVM) to identify the most accurate and reliable classifier for this specific dataset.

## Project Description

This project builds a sentiment classifier for tweets. Raw text data is loaded using **Pandas** and thoroughly cleaned using **NLTK** to remove noise (like URLs and mentions) and standardize words. This cleaned text is then converted into numerical features using **TF-IDF**. Finally, five different **Scikit-learn** models are trained and compared, with **Linear SVM** being selected as the final model due to its high accuracy (90.6%) and strong generalization.

# Chapter 2 – Requirement Analysis and System Specification

## Hardware Requirements

- Processor: Dual-core or higher
- RAM: 4 GB minimum
- Display: 1024x768 or higher

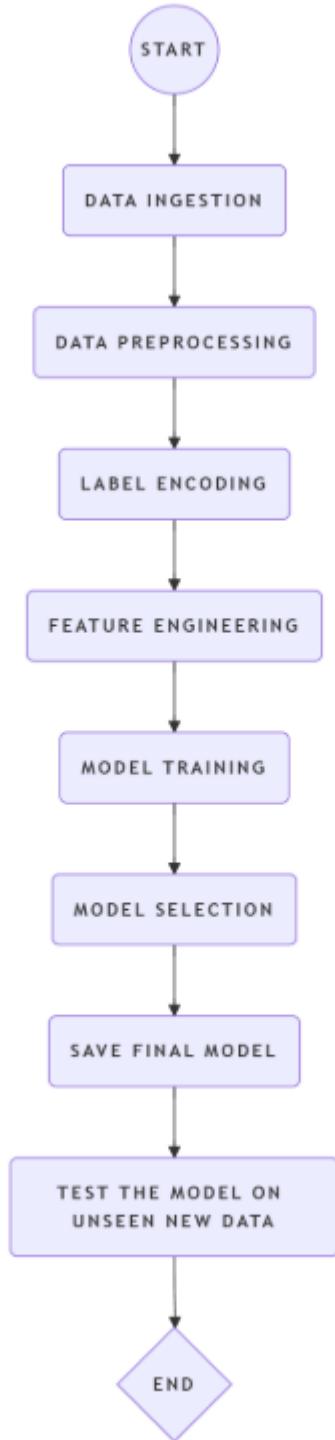
## Software Requirements:

- Python 3.8+
- Libraries: Pandas, numpy, sklearn, nltk, matplotlib
- OS: Windows, macOS, or Linux

## Functional Requirements

1. Data Ingestion to load CSV files
2. Text Pre-processing
3. Convert text data to numerical data
4. Train Models based on the vectorized data

# Chapter 3 – System Design



# Chapter 4 – Implementation Modules

The project is implemented in Python and follows a standard data science workflow.

1. **Data Handling:** **Pandas** is used to load, store, and manipulate the tweet data from CSV files.
2. **Text Preprocessing:** The **NLTK** (Natural Language Toolkit) library is used to clean the raw text. This involves lowercasing, removing noise (URLs, mentions), tokenizing (splitting text into words), removing stopwords, and lemmatizing (reducing words to their root form).
3. **Feature Engineering:** **Scikit-learn's TfidfVectorizer** converts the cleaned text into a numerical matrix, which machine learning models can understand.
4. **Modeling & Evaluation:** **Scikit-learn** is used to train all five machine learning models. **Matplotlib** and **Seaborn** are used to plot the results (accuracy chart and confusion matrices).

## ML Models Used

- **Logistic Regression:** A statistical model that predicts the probability of a tweet belonging to a specific sentiment category.
- **Multinomial Naive Bayes:** A fast probabilistic classifier (popular for text) that uses the frequency of words to calculate the most likely sentiment.
- **Decision Tree:** A flowchart-like model that learns a series of "if-then-else" questions about the words to determine the sentiment.

- **Random Forest:** An ensemble model that builds hundreds of individual Decision Trees and combines their votes to make a final, more accurate prediction.
- **Linear SVM (Support Vector Machine):** A powerful classifier that finds the optimal "line" or boundary that best separates the data points (tweets) into their different sentiment classes.

# Chapter 5 – Results and Discussion

## Results

As a result of this project, we successfully built a complete, end-to-end pipeline for analyzing tweet sentiment. We implemented a robust process that involved:

- Cleaning and normalizing raw tweet text using **NLTK**.
- Converting the clean text into a numerical **TF-IDF** matrix.
- Training and comparing five different machine learning models from **Scikit-learn**.

Our evaluation showed that the Random Forest model achieved the highest raw accuracy at **94.8%**. However, after analyzing the trade-offs, we selected the **Linear SVM** as our final, optimal model.

We made this decision for several key reasons:

- **Performance:** The Linear SVM still achieved a very high accuracy of **90.6%**.
- **Reliability:** It is known to be more robust and less prone to overfitting on high-dimensional text data, making it a safer and more generalizable choice.
- **Efficiency:** It is significantly faster to train and use for predictions than the 200-tree Random Forest.

Our final tests confirmed this choice, as our Linear SVM proved highly effective in accurately classifying the sentiment of new, unseen tweets.

## What a Confusion Matrix Is

A **confusion matrix** is a table that summarizes how well a classification model performs.

Instead of just giving a single accuracy score, it provides a detailed breakdown of what the model got right and what it got wrong. This is crucial because it helps us see *exactly where* the model is getting "confused."

The matrix compares the **Actual (True) Labels** to the **Predicted Labels**.

- **The Diagonal (Correct Predictions):** The cells that run from the top-left to the bottom-right show all the correct guesses. We want the numbers here to be as high as possible.
  - **The Off-Diagonal (The Errors):** All other cells show the model's mistakes (the "confusions"). For example, it shows how many times the model predicted "Positive" when the tweet was actually "Negative."
- 

## How We Used It in Our Project

Since our project classifies tweets into four categories (Positive, Negative, Neutral, and Irrelevant), our confusion matrices were **4x4 grids**.

- The **rows** showed the **True Label** (what the sentiment *actually* was).
- The **columns** showed the **Predicted Label** (what our model *guessed* it was).

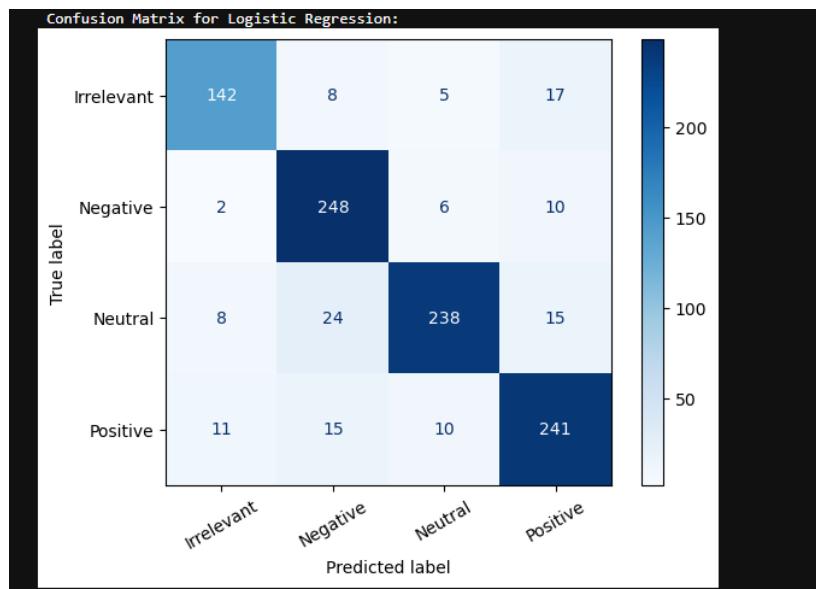
This was incredibly useful for us. By looking at the **off-diagonal cells**, we could see *patterns* in our models' mistakes. For instance, we noticed that while most models were good at telling

Positive and Negative apart, they often "confused" Neutral tweets with Irrelevant ones.

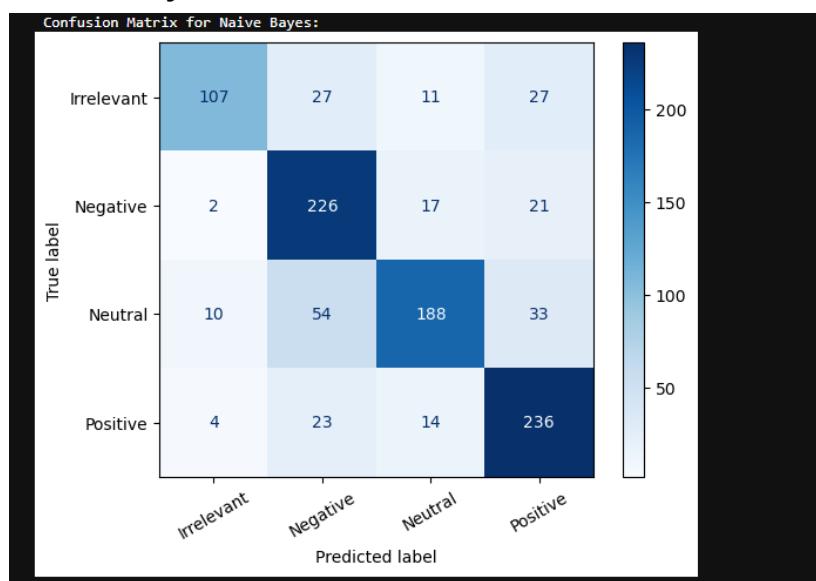
This level of detail allowed us to choose our final model based not just on its overall accuracy, but on its specific strengths and weaknesses for this particular task.

Matrix for Diff Models:

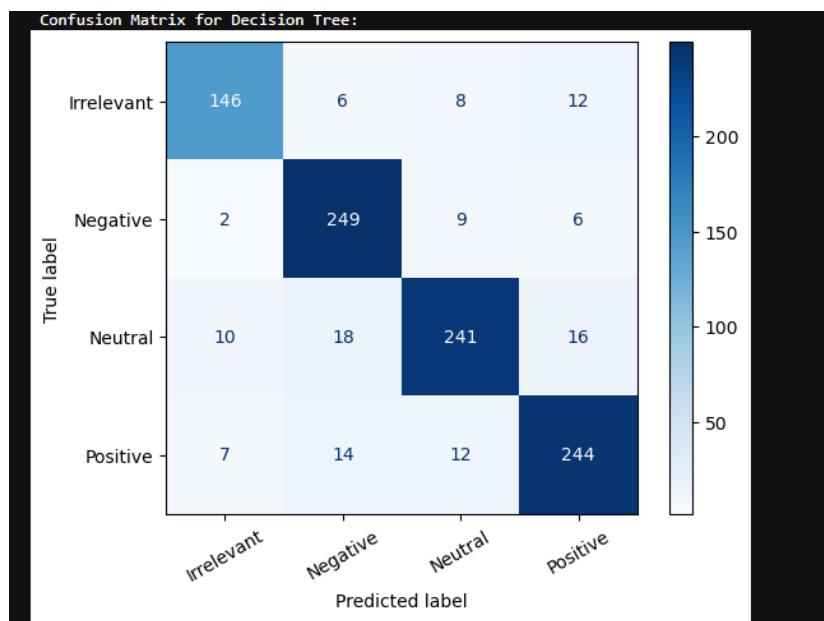
Logistic Regression:



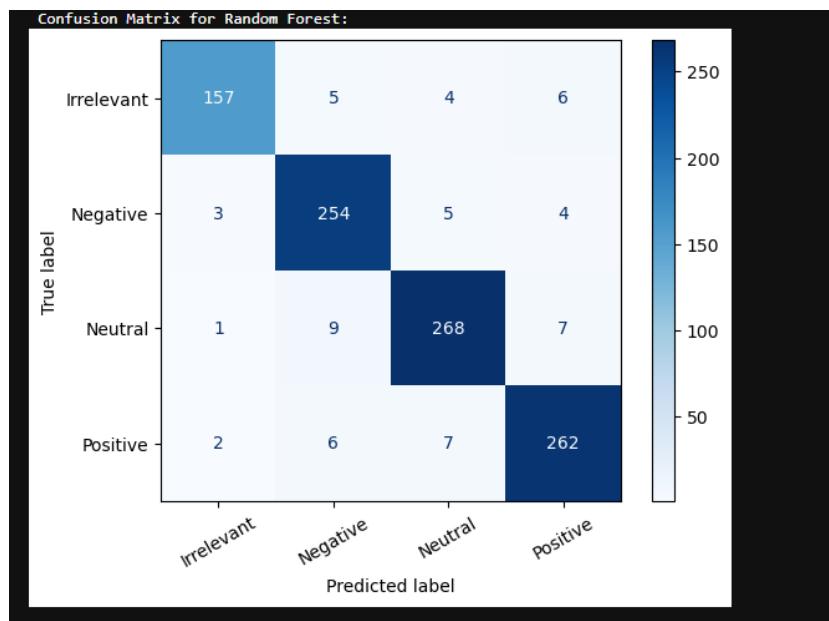
Naïve Bayes:



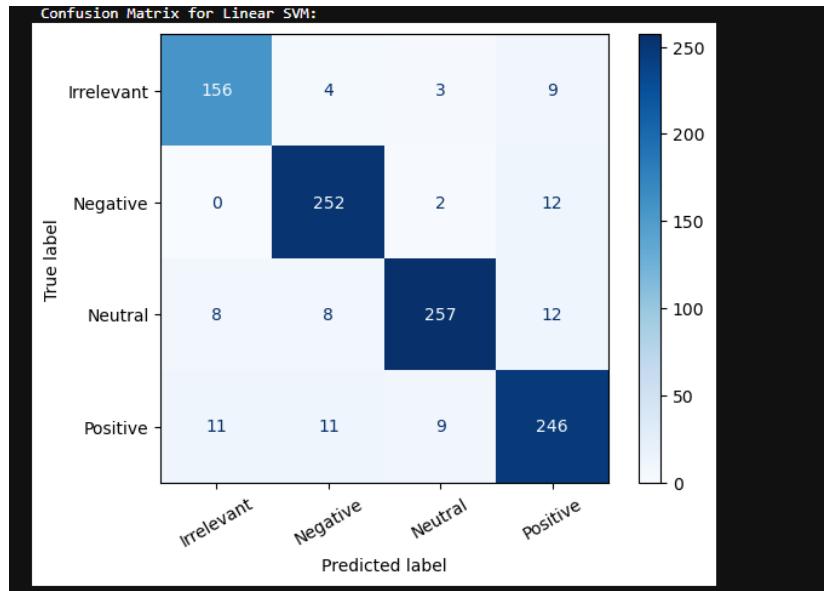
## Decision Tree:



## Random Forest:



## Linear SVM:



# Discussion

## 1. The Importance of Text Preprocessing

Our models' success was highly dependent on our **NLTK preprocessing pipeline**. The raw tweet data was very "noisy" (URLs, @mentions, punctuation). By cleaning, tokenizing, removing stopwords, and lemmatizing the text, we created a standardized feature set that was essential for achieving high accuracy.

---

## 2. Model Choice: Accuracy vs. Reliability

One of our main discussions was choosing the final model.

- **Random Forest** gave the highest accuracy at **94.8%**.
- We chose **Linear SVM** (at **90.6%**) instead.

We did this because Random Forest is prone to **overfitting** on text data. We felt the Linear SVM was more **robust** and would **generalize better** to new, unseen tweets, while also being much faster to train.

---

### 3. Future Work: Addressing Model Confusion

The confusion matrices showed that our best models, while great at separating Positive and Negative, often **confused Neutral and Irrelevant tweets**. This suggests the simple TF-IDF features couldn't capture the subtle difference. In the future, we would use more advanced methods like **Word Embeddings (Word2Vec)** or **Deep Learning (LSTMs)** to better understand the context and improve this specific weakness.

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import html

[2]: import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import warnings
from sklearn.preprocessing import LabelEncoder, label_binarize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from sklearn.metrics import ConfusionMatrixDisplay
import seaborn as sns
import joblib

[3]: # NLTK Downloads
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[4]: # ignore warnings
warnings.filterwarnings('ignore')

[5]: nltk.download('punkt_tab')

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

[5]: True

[6]: train = pd.read_csv(
    "twitter_training.csv",
    header=None,
    names=["ID", "Topic", "Sentiment", "Text"]
)

# Validation dataset
valid = pd.read_csv(
    "twitter_validation.csv",
    header=None,
    names=["ID", "Topic", "Sentiment", "Text"]
)

[7]: train.head()

[7]:   ID      Topic  Sentiment          Text
0  2401  Borderlands    Positive  im getting on borderlands and i will murder yo...
1  2401  Borderlands    Positive  I am coming to the borders and I will kill you...
2  2401  Borderlands    Positive  im getting on borderlands and i will kill you ...
3  2401  Borderlands    Positive  im coming on borderlands and i will murder you...
4  2401  Borderlands    Positive  im getting on borderlands 2 and i will murder ...
```

```
[8]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74682 entries, 0 to 74681
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ID          74682 non-null   int64  
 1   Topic        74682 non-null   object  
 2   Sentiment    74682 non-null   object  
 3   Text         73996 non-null   object  
dtypes: int64(1), object(3)
memory usage: 2.3+ MB

[9]: valid.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   ID          1000 non-null   int64  
 1   Topic        1000 non-null   object  
 2   Sentiment    1000 non-null   object  
 3   Text         1000 non-null   object  
dtypes: int64(1), object(3)
memory usage: 31.4+ KB

[10]: train.describe()

[10]:      ID
count  74682.000000
mean   6432.586165
std    3740.427870
min    1.000000
25%   3195.000000
50%   6422.000000
75%   9601.000000
max   13200.000000
```

```
[11]: valid.describe()

[11]:          ID
   count    1000.000000
   mean     6432.088000
   std      3728.310569
   min      6.000000
  25%    3247.750000
  50%    6550.000000
  75%    9661.750000
   max   13197.000000

[12]: # ## Step 3: Preprocessing

# In this step, we clean the raw tweet texts to make them suitable for analysis.
# The preprocessing pipeline includes:
# 1- Converting text to lowercase
# 2- Removing URLs, mentions, hashtags, punctuation, and special characters
# 3- Tokenization (splitting sentences into words)
# 4- Stopword removal (removing common words like "the", "is", "at")
# 5- Lemmatization (converting words to their base form, e.g., "running" → "run")
#
# This ensures that the machine learning model focuses only on meaningful information.

[13]: stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# cleaning function
def clean_text(text):
    if not isinstance(text, str):
        return ""
    text = html.unescape(text)           # decode HTML
    text = text.lower()                 # lowercase
    text = re.sub(r'http\S+|www\.\S+', ' ', text) # remove urls
    text = re.sub(r@\w+, ' ', text)      # remove mentions
    text = re.sub(r'\#', ' ', text)       # remove hash only
    text = re.sub(r'[^\w\s]', ' ', text) # keep alnum + spaces

    tokens = nltk.word_tokenize(text)
    tokens = [t for t in tokens if t not in stop_words and len(t) > 1]
    tokens = [lemmatizer.lemmatize(t) for t in tokens] # lemmatize
    return " ".join(tokens)

# apply on both datasets
train['clean_text'] = train['Text'].apply(clean_text)
valid['clean_text'] = valid['Text'].apply(clean_text)

# check sample
print(train[['Text','clean_text']].head(10))

          Text \
0  im getting on borderlands and i will murder yo...
1  I am coming to the borders and I will kill you...
2  im getting on borderlands and i will kill you ...
3  im coming on borderlands and i will murder you...
4  im getting on borderlands 2 and i will murder ...
5  im getting into borderlands and i can murder y...
6  So I spent a few hours making something for fu...
7  So I spent a couple of hours doing something f...
8  So I spent a few hours doing something for fun...
9  So I spent a few hours making something for fu...

          clean_text
0           im getting borderland murder
1           coming border kill
2           im getting borderland kill
3           im coming borderland murder
```

```
[14]: # ## Step 4: Encode labels
# Label encoding, fit on combined labels to ensure consistent mapping

[15]: le = LabelEncoder()
le.fit(pd.concat([train['Sentiment'], valid['Sentiment']], axis=0))
train['label_enc'] = le.transform(train['Sentiment'])
valid['label_enc'] = le.transform(valid['Sentiment'])
print("Label mapping (index -> label):", dict(enumerate(le.classes_)))

# Show class distribution
print("\nTrain class counts:\n", train['Sentiment'].value_counts())
print("\nValid class counts:\n", valid['Sentiment'].value_counts())

Label mapping (index -> label): {0: 'Irrelevant', 1: 'Negative', 2: 'Neutral', 3: 'Positive'}

Train class counts:
Sentiment
Negative      22542
Positive      20832
Neutral       18318
Irrelevant    12990
Name: count, dtype: int64

Valid class counts:
Sentiment
Neutral        285
Positive       277
Negative       266
Irrelevant     172
Name: count, dtype: int64
```

	ID	Topic	Sentiment	Text	clean_text	label_enc
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting borderland murder	3
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	coming border kill	3
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting borderland kill	3
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming borderland murder	3
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting borderland murder	3

```
[16]: train.head()

[17]: # TF-IDF vectorization
tfv = TfidfVectorizer(max_features=10000, ngram_range=(1,2), stop_words='english')
X_train = tfv.fit_transform(train['clean_text'])
X_valid = tfv.transform(valid['clean_text'])
y_train = train['label_enc']
y_valid = valid['label_enc']

print("\nTF-IDF matrix shapes, X_train, X_valid:", X_train.shape, X_valid.shape)
```

TF-IDF matrix shapes, X\_train, X\_valid: (74682, 10000) (1000, 10000)

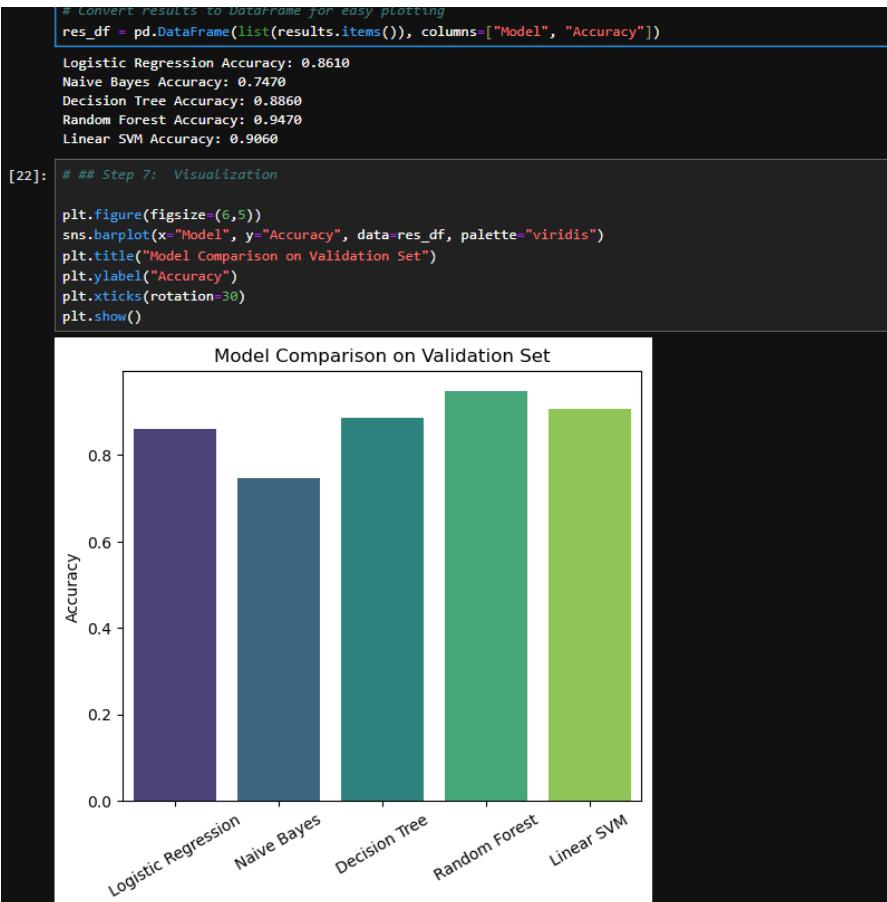
```
[21]: # ## Step 6: Compare Multiple Models

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Naive Bayes": MultinomialNB(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
    "Linear SVM": LinearSVC(random_state=42)
}

results = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    acc = accuracy_score(y_valid, preds)
    results[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")

# Convert results to DataFrame for easy plotting
```



```
[19]: for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_valid)

    # Accuracy
    acc = accuracy_score(y_valid, y_pred)
    print(f"\n{name} Accuracy: {acc:.4f}")

    # Classification report
    print(classification_report(y_valid, y_pred, target_names=le.classes_))

    # Confusion matrix
    print(f"Confusion Matrix for {name}:")
    disp = ConfusionMatrixDisplay.from_estimator(model, X_valid, y_valid,
                                                display_labels=le.classes_,
                                                cmap='Blues',
                                                xticks_rotation=30)

    plt.show()
```

```
Logistic Regression Accuracy: 0.8690
      precision    recall  f1-score   support

Irrelevant       0.87      0.83      0.85      172
  Negative       0.84      0.93      0.88      266
  Neutral        0.92      0.84      0.88      285
 Positive        0.85      0.87      0.86      277

   accuracy         0.87      0.87      0.87     1000
  macro avg       0.87      0.87      0.87     1000
weighted avg     0.87      0.87      0.87     1000
```

```
[23]: final_model = models["Linear SVM"]
#choosing SVM to be the final model since Random Forest is prone to overfitting
print("--- Final Model Performance ---")
print(f"Model: Random Forest")
print(f"Accuracy on validation set: {results['Random Forest']:.4f}\n")

#Test the model on new, unseen data
print("--- Testing on New Data ---")
new_tweets = [
    "I love this game, it's so much fun!",
    "This update is terrible, it ruined everything.",
    "Just bought a new controller, it's okay I guess.",
    "That company is the worst.",
    "I don't really have an opinion on this."
]

cleaned_tweets = [clean_text(t) for t in new_tweets]
new_tweets_tfidf = tfv.transform(cleaned_tweets)
predictions = final_model.predict(new_tweets_tfidf)

#Convert numerical Labels back to text Labels (e.g., 3 -> "Positive")
predicted_labels = le.inverse_transform(predictions)

# Print the results
for i in range(len(new_tweets)):
    print(f"Tweet: '{new_tweets[i]}'")
    print(f"Predicted Sentiment: {predicted_labels[i]}\n")
```

```
--- Final Model Performance ---
Model: Random Forest
Accuracy on validation set: 0.9470

--- Testing on New Data ---
Tweet: 'I love this game, it's so much fun!'
Predicted Sentiment: Positive

Tweet: 'This update is terrible, it ruined everything.'
Predicted Sentiment: Negative

Tweet: 'Just bought a new controller, it's okay I guess.'
Predicted Sentiment: Neutral

Tweet: 'That company is the worst.'
Predicted Sentiment: Negative

Tweet: 'I don't really have an opinion on this.'
Predicted Sentiment: Positive
```

Kaggle notebook:

<https://www.kaggle.com/code/sxham04/sent-analysis-main>

# Chapter 6 – Summary and Conclusion

## Summary

In this project, we built a pipeline to classify tweet sentiment. We used **Pandas** to load the data, **NLTK** to clean and preprocess the raw text (lemmatizing, removing stopwords, etc.), and **Scikit-learn's** TfidfVectorizer to convert the text into numerical features. We then trained and evaluated five different machine learning models to compare their performance.

## Conclusion

Our results showed that while Random Forest had the highest accuracy (94.8%), we selected the **Linear SVM** as our final model. The Linear SVM provided a strong balance of high accuracy (90.6%) and better generalization, as it is less prone to overfitting on this type of text data. It was also significantly faster to train. The final model proved effective, successfully classifying the sentiment of new, unseen tweets.

# References

1. Dataset:

<https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/code?datasetId=1520310&sortBy=dateCreated>

2. Kaggle notebook code:

<https://www.kaggle.com/code/sxham04/sent-analysis-main>

3. NLTK lib: <https://www.nltk.org/>