**Task-4**

**Sentiment Analysis on Twitter Data - Project Documentation**

**Project Overview**

As part of my internship, I worked on a project to analize and visualize sentiment patterns in social media data. The dataset used for this task comes from Twitter, containing user opinions and comments. The objective was to classify the sentiment of each tweet into categories such as Positive, Negative, or Neutral, providing insights into public opinion trends.

**Dataset**

The dataset includes the following key fields:

* **User ID**: Unique identifier for each user.
* **Platform**: Platform or service where the tweet originated.
* **Sentiment**: Target variable (Positive, Negative, or Neutral).
* **Text**: Actual content of the tweet.

**Key Steps**

**1. Data Loading**

I imported the dataset using pandas to ensure it was loaded correctly and checked for missing or duplicate entries. Here's a snapshot of the code:

df = pd.read\_csv('../data/twitter\_validation.csv')

print(df.isnull().sum()) # Checking for missing values

df.drop\_duplicates(inplace=True) # Removing duplicates

**2. Data Cleaning**

I cleaned the text data to make it ready for machine learning by:

* Converting all text to lowercase.
* Removing punctuation and special characters.
* Removing common stopwords like 'the' and 'and' using the NLTK library.

def clean\_text(text):

text = text.lower()

text = re.sub(r'[^\w\s]', '', text) # Remove punctuation

text = re.sub(r'\d+', '', text) # Remove numbers

stop\_words = set(stopwords.words('english'))

text = ' '.join([word for word in text.split() if word not in stop\_words])

return text

df['cleaned\_text'] = df['text '].apply(lambda x: clean\_text(x))

**3. Text Vectorization**

The cleaned text was then converted into numerical features using CountVectorizer. This step transformed the text into a bag-of-words model, allowing it to be used for machine learning.

vectorizer = CountVectorizer(max\_features=1000, stop\_words='english')

X = vectorizer.fit\_transform(df['cleaned\_text']).toarray()

**4. Model Training**

I used a **Multinomial Naive Bayes** algorithm for sentiment classification. This algorithm is well-suited for text classification tasks, especially for multi-class scenarios like sentiment analysis.

model = MultinomialNB()

model.fit(X\_train, y\_train)

**5. Model Evaluation**

The model's performance was evaluated using metrics like accuracy, classification report, and confusion matrix to analyze its predictions and overall effectiveness.

y\_pred = model.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred) \* 100:.2f}%")

print(classification\_report(y\_test, y\_pred))

**6. Data Visualization**

I visualized the performance of the model using a **confusion matrix**, which helped in understanding the distribution of correct and incorrect predictions.

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes\_, yticklabels=model.classes\_)

plt.show()

**Conclusion**

This project improved my ability to handle real-world data and perform sentiment analysis. I gained experience in:

* Cleaning and preprocessing textual data.
* Implementing machine learning algorithms for text classification.
* Evaluating model performance using key metrics.

The combination of data preprocessing, feature extraction, and model evaluation techniques helped me understand how to handle text data for sentiment analysis.

**Thank you!**