# Sentiment Analysis:

## > Nltk , Hugging Face Pipeline and  Strealit App Integration

### Part 1: Importing Libraries and Loading Data

Python

```
# -*- coding: utf-8 -*-

# Initial comments (optional)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Install NLTK (if not already installed)
!pip install nltk

import nltk
nltk.download('vader_lexicon')
nltk.download('averaged_perceptron_tagger')

# Load data
df = pd.read_csv('Reviews.csv')
print(df.shape)
df = df.head(500)
print(df.shape)

# Display first few rows
df.head()
```

- **Encoding:** The `# -*- coding: utf-8 -*-` line ensures proper UTF-8 character encoding.
- **Library Imports:** Essential libraries for data manipulation, analysis, and visualization are imported.

- **NLTK Installation and Resource Downloads:** If NLTK is not installed, it's installed using `!pip install nltk`. The `nltk.download()` calls download the necessary resources for sentiment analysis and part-of-speech tagging.
- **Data Loading and Shaping:** The `Reviews.csv` file is read into a pandas DataFrame `df`. The DataFrame's shape is printed before and after limiting it to the first 500 rows for faster processing.
- **Exploratory Data Analysis (EDA):** The first few rows are displayed to gain insights into the data.

**Part 2: Text Preprocessing with NLTK**

Python

```
nltk.download('punkt')

# Tokenize example review
example = df['Text'][50]
tokens = nltk.word_tokenize(example)

# Part-of-speech tagging
tagged = nltk.pos_tag(tokens)
print(tagged[:10])

# Download additional NLTK resources
!python -m nltk.downloader words
!python -m nltk.downloader maxent_ne_chunker

# Named entity chunking
entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()
```

- **Punkt Resource Download:** The `nltk.download('punkt')` call downloads the resource for sentence and word tokenization.
- **Tokenization:** The example review text is split into individual words using `nltk.word_tokenize()`.

- **Part-of-Speech Tagging:** The tokenized text is tagged with part-of-speech information (e.g., noun, verb) using `nltk.pos_tag()`.
- **Words and Maxent NE Chunker Downloads:** These resources are downloaded using shell commands for further text processing.
- **Named Entity Chunking:** The part-of-speech tagged text is chunked into named entities (e.g., person names, organizations) using `nltk.chunk.ne_chunk()`.

**Part 3: Sentiment Analysis with VADER**

Python

```
from nltk.sentiment import SentimentIntensityAnalyzer

sia = SentimentIntensityAnalyzer()

# Example sentiment scores
print(sia.polarity_scores("I love to provide to human!"))
print(sia.polarity_scores("This is the worst thing"))

# Calculate sentiment scores for example review
scores = sia.polarity_scores(example)
print(scores)

# Initialize sentiment scores dictionary
res = {}

# Calculate and store sentiment scores for all reviews
for index, row in df.iterrows():
    text = row['Text']
    myid = row['ID']
    scores = sia.polarity_scores(text)
    res[myid] = scores

# Create DataFrame from sentiment scores
vaders = pd.DataFrame(res).T.reset_index()

# Merge sentiment scores with original DataFrame
vaders = vaders.merge(df, how='left')

# Visualize sentiment scores vs. review scores
sns.barplot(data=vaders, x='Score', y='compound')
```

```
plt.subplots(figsize=(10, 5))
sns.barplot(data=vaders, x='Score', y='neg')
sns.barplot(data=vaders, x='Score', y='neu')
sns.barplot(data=vaders, x='Score', y='pos')
plt.show()
```

- **SentimentIntensityAnalyzer:** The `SentimentIntensityAnalyzer` class is imported for VADER-based sentiment analysis.
- **Example Sentiment Scores:** Sentiment scores are calculated for two example sentences to demonstrate VADER's functionality.
- **Review Sentiment Scores:** The `sia.polarity_scores()` method is used to calculate sentiment scores for the example review, storing them in the `scores` dictionary.
- **Sentiment Score Dictionary:** An empty dictionary `res` is initialised to store sentiment scores for all reviews.
- **Sentiment Score Calculation and Storage:** The code iterates through each review in the DataFrame, calculates VADER sentiment scores, and stores them in the `res` dictionary with the review ID as the key.
- **Sentiment Scores DataFrame:** The `res` dictionary is converted into a pandas DataFrame `vaders` and merged with the original DataFrame `df` to retain all review data.
- **Sentiment Score Visualisation:** Bar plots are created using `sns.barplot()` to explore the relationship between review scores and VADER sentiment scores (compound, negative, neutral, positive).

**Part 4: Sentiment Analysis with RoBERTa**

Python

```
# Install transformers library
```

```
!pip install transformers

from transformers import AutoTokenizer, AutoModelForSequenceClassification

MODEL = "cardiffnlp/twitter-roberta-base-sentiment"

# Load RoBERTa model
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

# Print example review and VADER scores
print(example)
print(scores)

# Calculate RoBERTa sentiment scores
encoded_text = tokenizer(example, return_tensors='pt')
output = model(**encoded_text)
logits = output.logits.squeeze().detach().numpy()
scores_dict = {
    'neg': float(softmax(logits[0])[0]),
    'neu': float(softmax(logits[0])[1]),
    'pos': float(softmax(logits[0])[2])
}
print(scores_dict)

# Function for RoBERTa sentiment scores
def polarity_scores_roberta(text):
    encoded_text = tokenizer(text, return_tensors='pt')
    output = model(**encoded_text)
    logits = output.logits.squeeze().detach().numpy()
    scores_dict = {
        'neg': float(softmax(logits[0])[0]),
        'neu': float(softmax(logits[0])[1]),
        'pos': float(softmax(logits[0])[2])
    }
    return scores_dict

# Initialize sentiment scores dictionary
res = {}

# Calculate and store sentiment scores for all reviews
for index, row in df.iterrows():
    text = row['Text']
    myid = row['ID']
    try:
        vader_scores = sia.polarity_scores(text)
        roberta_scores = polarity_scores_roberta(text)
```

```
        both = {'vader': vader_scores, 'roberta': roberta_scores}
        res[myid] = both
    except RuntimeError as e:
        print(f"RuntimeError for review ID {myid}: {e}")

# Create DataFrame from sentiment scores
results_df = pd.DataFrame(res).T.reset_index()

# Merge sentiment scores with original DataFrame
results_df = results_df.merge(df, how='left')
```

- **Transformers Library Installation:** The `!pip install transformers` command instals the transformers library for RoBERTa.
- **RoBERTa Model Loading:** The `AutoTokenizer` and `AutoModelForSequenceClassification` classes are used to load the pre-trained RoBERTa model for sentiment analysis.
- **Example Review and VADER Scores:** The example review text and its VADER sentiment scores are printed for comparison.
- **RoBERTa Sentiment Score Calculation:**
    - The review text is tokenized using the `tokenizer`.
    - The tokens are passed to the RoBERTa model using `model(**encoded_text)`.
    - The model's output (logits) is converted to probabilities using the `softmax` function from `scipy.special`.
    - The probabilities for negative, neutral, and positive sentiment are stored in `scores_dict`.
- `polarity_scores_roberta` **Function:** This function takes a text as input and returns sentiment scores (negative, neutral, positive) using the RoBERTa model.
- **Sentiment Score Calculation and Storage:** The code iterates through each review, calculates VADER and RoBERTa sentiment scores, combines them into a

`both` dictionary, stores them in `res` with the review ID as the key, and handles potential `RuntimeError` exceptions (e.g., due to long text).

- **Sentiment Scores DataFrame:** The `res` dictionary is converted into a pandas DataFrame `results_df` and merged with the original DataFrame `df` to retain all review data.

> The script creates visualizations to explore the relationship between review scores and VADER sentiment scores using `sns.barplot()`

Here's a breakdown of the provided information about sentiment analysis with Hugging Face Pipeline and Streamlit App:

**Part 6: Sentiment Analysis with Hugging Face Pipeline**

- **Importing Libraries:**
  - `from transformers import pipeline`: This line imports the `pipeline` function from the transformers library, which allows you to easily use pre-trained models for various tasks.
- 
- **Creating Sentiment Analysis Pipeline:**
  - `sent_pipeline = pipeline("sentiment-analysis")`: This creates a sentiment analysis pipeline using the `pipeline` function with the name "sentiment-analysis". This pipeline can be used to analyze the sentiment of text data.
- 
- **Using the Pipeline:**
  - `result = nlp('I love nature! ')`: This line demonstrates how to use the pipeline. It sends the text "I love nature!" to the pipeline and stores the sentiment analysis result in the `result` variable.

- `print(result)`: This prints the `result` variable, which contains the sentiment scores (positive, negative, and neutral) for the input text.

**Part 7: Streamlit App**

- **Installing Streamlit:**
  - `!pip install streamlit`: This command installs the Streamlit library, which is used to create web applications.
- **Creating a Basic App:**
  - `st.title("User Input Demo")`: This sets the title of the Streamlit app to "User Input Demo".
  - `user_input = st.text_input("Enter some text:")`: This creates a text input field where users can enter text for sentiment analysis.
  - `if user_input:`: This checks if the user has entered any text.
  - `process_user_input(user_input)`: If the user has entered text, this function is called to process the input (perform sentiment analysis). The result is then displayed using `st.write(result)`.
  - `process_user_input() function`: This function is not implemented in the provided code. It's expected to take the user input text, perform sentiment analysis (or potentially other tasks like summarization), and return the result.

- **Running the App:**
  - `!streamlit run sentiment_analysis.py`: This command attempts to run the Streamlit app using the provided Python script. If successful, the app

will be launched in a local web server and accessible through your web browser.