**Phase - 3**

**Technology : AI**

**Project Name: Sentiment Analysis for Marketing**

**Email ID: IndustryEvaluator@skillup.online**

**Faculty Mentor:** [**ezhilmathia@jkkn.ac.in**](mailto:ezhilmathia@jkkn.ac.in)

**Group Members:**

**V.Suvathi,**

**M.Subitha,**

**P.NavinRaj,**

**R.Kowshick,**

**S.NavaneethaKrishnan.**

**NLP-Sentiment analysis for marketing**

**Preprocessing data**

**Data cleaning**

from nltk.corpus import stopwords

import string

import re

import contractions

def text\_cleaning(text):

if text:

text = contractions.fix(text)

text = &#39; &#39;.join(text.split(&#39;.&#39;))

text = re.sub(r&#39;\s+&#39;, &#39; &#39;, re.sub(&#39;[^A-Za-z0-9]&#39;, &#39; &#39;, text.strip().lower())).strip()

text = re.sub(r&#39;\W+&#39;, &#39; &#39;, text.strip().lower()).strip() text = [word for word in text.split()]

return text

return []

**EDA**

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings(&#39;ignore&#39;)

linkcode

import nltk

nltk.download(&#39;stopwords&#39;)

from nltk.corpus import stopwords

stopWords\_nltk = set(stopwords.words(&#39;english&#39;))

import re

from typing import Union, List

class CleanText():

&quot;&quot;&quot; clearing text except digits () . , word character &quot;&quot;&quot;

def \_\_init\_\_(self, clean\_pattern = r&quot;[^A-ZĞÜŞİÖÇIa-zğüı&#39;şöç0-9.\&quot;‘, ()]&quot;):

self.clean\_pattern =clean\_pattern

def \_\_call\_\_(self, text: Union[str, list]) -&gt; List[List[str]]:

if isinstance(text, str):

docs = [[text]]

if isinstance (text, list):

docs = text

text = [[re.sub (self. clean\_pattern, &quot; &quot;, sent) for sent in sents] for sents in docs]

return text

def remove\_emoji(data):

emoj = re.compile(&quot;[&quot;

u&quot;\U0001F600-\U0001F64F&quot;

u&quot;\U0001F300-\U0001F5FF&quot;

u&quot;\U0001F680-\U0001F6FF&quot;

u&quot;\U0001F1E0-\U0001F1FF&quot;

u&quot;\U00002500-\U00002BEF&quot;

u&quot;\U00002702-\U000027B0&quot;

u&quot;\U00002702-\U000027B0&quot;

u&quot;\U000024C2-\U0001F251&quot;

u&quot;\U0001f926-\U0001f937&quot;

u&quot;\U00010000-\U0010ffff&quot;

u&quot;\u2640-\u2642&quot;

u&quot;\u2600-\u2B55&quot;

u&quot;\u200d&quot;

u&quot;\u23cf&quot;

u&quot;\u23e9&quot;

u&quot;\u231a&quot;

u&quot;\ufe0f&quot;

u&quot;\u3030&quot;

&quot;]+&quot;, re.UNICODE)

return re.sub(emoj, &#39;&#39;, data)

def tokenize(text):

&quot;&quot;&quot; basic tokenize method with word character, non word character and di gits &quot;&quot;&quot;

text = re.sub(r&quot; +&quot;, &quot; &quot;, str(text))

text = re. split(r&quot;(\d+|[a-zA-ZğüşıöçĞÜŞİÖÇ] +|\W)&quot;, text)

text = list (filter (lambda x: x!= &#39;&#39; and x!= &#39; &#39;, text))

sent\_tokenized = &#39; ‘. join(text)

return sent\_tokenized

regex = re.compile(&#39;[%s]&#39; % re.escape(string.punctuation))

def remove\_punct(text):

text = regex.sub(&quot; &quot;, text)

return text

clean = CleanText()

def label\_encode(x):

if x == 1 or x == 2:

return 0

if x == 3:

return 1

if x == 5 or x == 4:

return 2

def label2name(x):

if x == 0:

return &quot;Negative&quot;

if x == 1:

return &quot;Neutral&quot;

if x == 2:

return &quot;Positive&quot;

df = pd.read\_csv(&quot;../input/trip-advisor-hotel-reviews/tripadvisor\_hotel\_reviews.csv&quot;)

print (&quot;df. columns: &quot;, df. columns)

df.columns: Index([&#39;Review&#39;, &#39;Rating&#39;], dtype=&#39;object&#39;)

df. head ()

fig = px.histogram(df,

x = &#39;Rating&#39;,

title = &#39;Histogram of Review Rating&#39;,

template = &#39;ggplot2&#39;,

color = &#39;Rating&#39;,

color\_discrete\_sequence= px.colors.sequential.Blues\_r,

opacity = 0.8,

height = 525,

width = 835,

(fig.update\_yaxes(title=&#39;Count&#39;)

fig.show()

**Data preprocessing**

import pandas as pd

import numpy as np

import os

import random

from pathlib import Path

import json

import torch

from tqdm.notebook import tqdm

from transformers import BertTokenizer

from torch. utils.data import TensorDataset

from transformers import BertForSequenceClassification

class Config ():

seed\_val = 17

device = torch. Device (&quot;cuda:0&quot; if torch.cuda.is\_available () else &quot;cpu&quot;)

epochs = 5

batch\_size = 6

seq\_length = 512

lr = 2e-5

eps = 1e-8

pretrained\_model = &#39;bert-base-uncased&#39;

test\_size=0.15

random\_state=42

add\_special\_tokens=True

return\_attention\_mask=True

pad\_to\_max\_length=True

do\_lower\_case=False

return\_tensors=&#39;pt&#39;

config = Config()

params = {&quot;seed\_val&quot;: config. seed\_val,

&quot;device&quot;:str(config.device),

&quot;epochs”: config. epochs,

&quot;batch\_size”: config.batch\_size,

&quot;seq\_length&quot;:config.seq\_length,

&quot;lr&quot;:config.lr,

&quot;eps&quot;:config.eps,

&quot;pretrained\_model&quot;: config.pretrained\_model,

&quot;test\_size&quot;:config.test\_size,

&quot;Random\_state&quot;:config. random\_state,

&quot;add\_special\_tokens”: config.add\_special\_tokens,

&quot;Return\_attention\_mask”: config. return\_attention\_mask,

&quot;Pad\_to\_max\_length”: config.pad\_to\_max\_length,

&quot;Do\_lower\_case&quot;:config.do\_lower\_case,

&quot;Return\_tensors&quot;:config. return\_tensors,

}

import random

device = config.device

random. seed(config.seed\_val)

np.random.seed(config.seed\_val)

torch.manual\_seed(config.seed\_val)

torch. cuda. manual\_seed\_all(config.seed\_val)

df.head()