**PHASE 3**

Project Title : **FAKE NEWS DETECTION USING NLP**

**Name : L.Keerthana(210521205028)**

**Introduction :**

In this project, the relevant technical theory to understand the techniques used is described. First, a short overview of machine learning is presented. thereafter, specific concepts for NLP and methods used in this project are described.

**Data Collection:**

Gather a large dataset of news articles. You'll need both real and fake news articles for training and testing.

**Data Labeling:**

Annotate each news article in the dataset as either "real" or "fake."

**Data Preprocessing:**

Text data often needs preprocessing, including lowercasing, tokenization, and removal of stopwords and special characters.

**Feature Extraction:**

Convert the text data into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings like Word2Vec or GloVe.

**Splitting the Dataset:**

Split the dataset into a training set and a testing set to evaluate the model's performance. Common splits are 70-30 or 80-20.

**Model Selection:**

Choose an appropriate machine learning or deep learning model for text classification. Common choices include Logistic Regression, Naive Bayes, or Recurrent Neural Networks (RNNs).

**Model Training:**

Train the selected model on the training data. Adjust hyperparameters to optimize the model's performance.

**Model Evaluation:**

Evaluate the model's performance on the testing dataset using metrics like accuracy, precision, recall, and F1 score.

**Fine-Tuning:**

If the model's performance is not satisfactory, fine-tune the model by adjusting hyperparameters or using more advanced techniques.

**Deployment:**

Once you have a satisfactory model, deploy it to make predictions on new, unseen news articles.

**Monitoring and Maintenance:**

Continuously monitor the model's performance in a real-world setting and update it as needed to maintain accuracy.

**Ethical Considerations:**

Consider the ethical implications of your model, such as bias and fairness, and implement mitigation strategies.

**Documentation and Reporting:**

Document the entire process, including dataset details, model architecture, hyperparameters, and results. Report your findings to stakeholders.

**User Interface (Optional):**

Develop a user interface or API for users to interact with your AI model.

S**cale and Optimize:**

If necessary, scale your solution to handle larger datasets and optimize its performance for production use.

Please note that building a reliable fake news detection model can be a complex task, and it's essential to stay updated on the latest developments in natural language processing and machine learning. Additionally, consider ethical and privacy concerns throughout the project

**Python program :**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import plotly.express as px

import plotly.graph\_objs as go

from plotly.subplots import make\_subplots

import nltk

from nltk.corpus import stopwords

import tensorflow as tf

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint

from sklearn.model\_selection import train\_test\_split

from transformers import AutoTokenizer, TFAutoModelForSequenceClassification

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

nltk.download('stopwords')

fake\_news\_path = "/kaggle/input/fake-and-real-news-dataset/Fake.csv"

real\_news\_path = "/kaggle/input/fake-and-real-news-dataset/True.csv"

fake\_news = pd.read\_csv(fake\_news\_path)

real\_news = pd.read\_csv(real\_news\_path)

fake\_news.head(3)

real\_news.head(3)

real = real\_news.copy()

fake = fake\_news.copy()

real['Label'] = 'Real'

fake['Label'] = 'Fake'

news = pd.concat([real, fake], axis=0, ignore\_index=True)

news.reset\_index()

news.head()

print(f"Samples available: {news.shape[0]}\n#features of dataset: {news.shape[1]}")

news\_ds = news.sample(1000).drop(['title', 'date', 'subject'], axis=1)

news\_ds.head(3)

CLASS\_NAMES = ['Fake', 'Real']

class\_mapper = {

'Fake':0,

'Real':1

}

news\_ds['Label'] = news\_ds['Label'].map(class\_mapper)

news\_ds.head(3)

class\_dist = px.histogram(data\_frame=news,

y='Label',

color='Label',

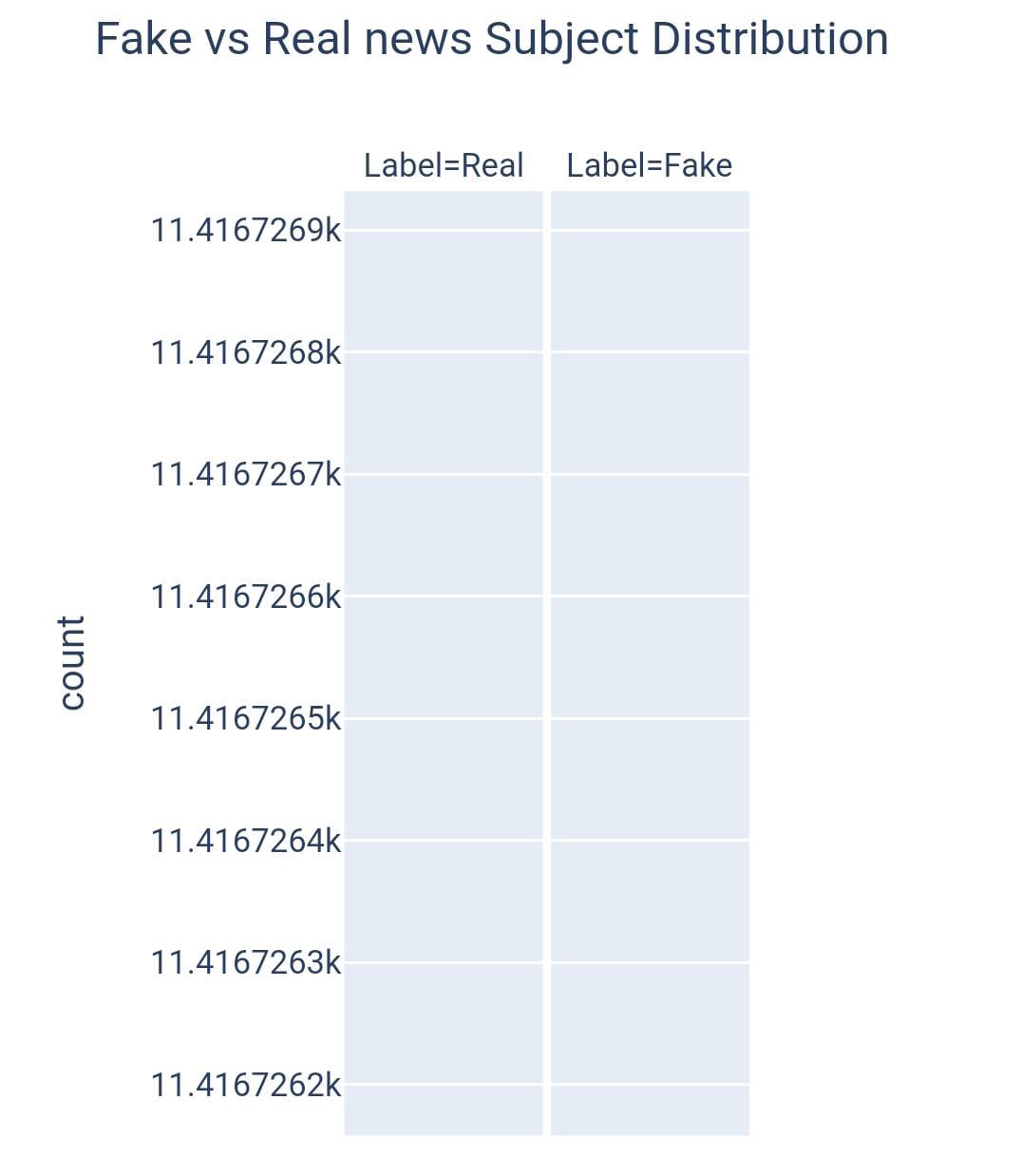
title='Fake vs Real news Original dataset',

text\_auto=True)

class\_dist.update\_layout(showlegend=False)

class\_dist.show()

OUTPUT :



news.date.unique().max()

[2:39 am, 21/10/2023] Jayasri R: news = news[news['date'].map(lambda x:len(x)) <= 20]

news.date = pd.to\_datetime(news['date'], format='mixed')

news.head()

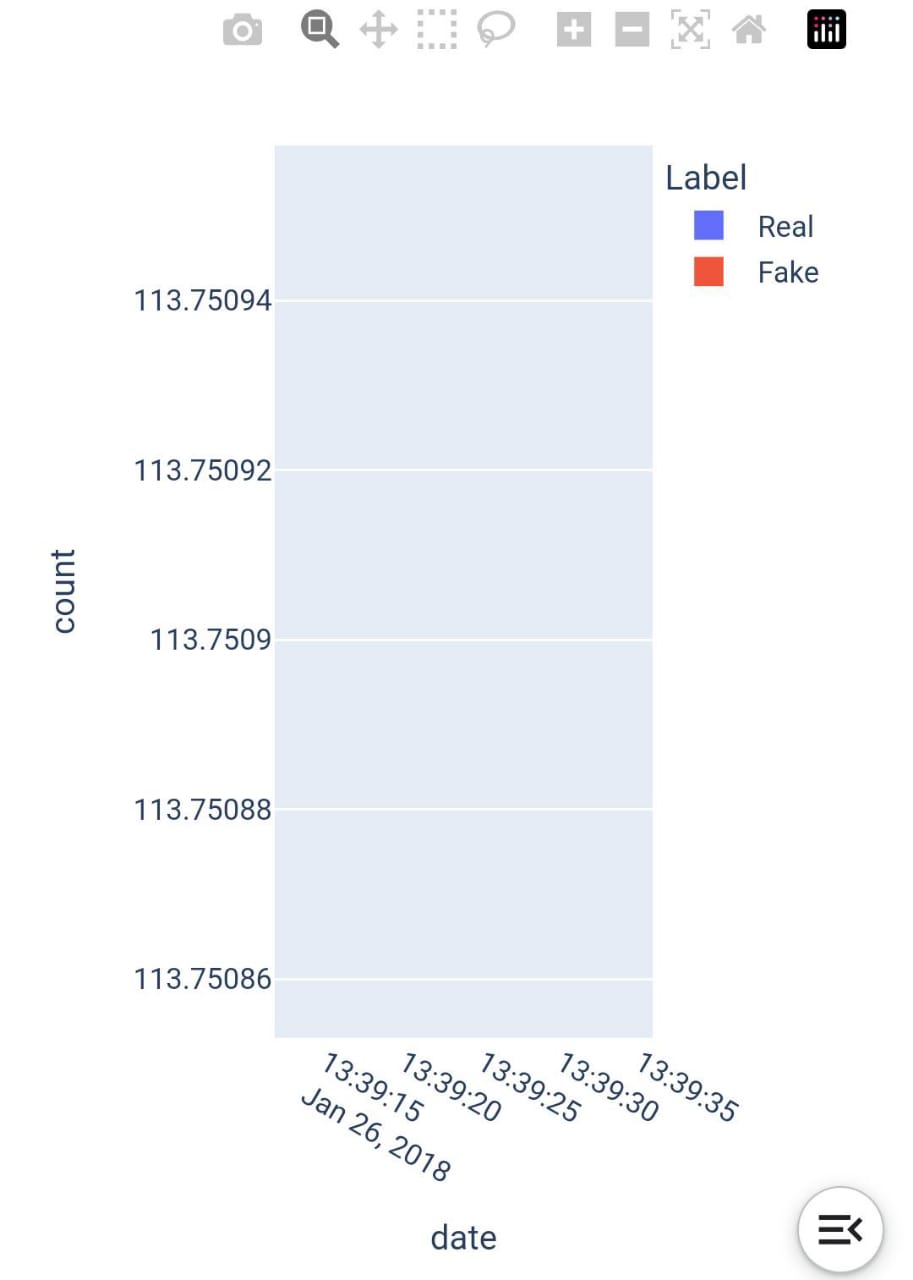
date\_dist = px.histogram(data\_frame=news,

x='date',

color='Label')

date\_dist.show()

OUTPUT :



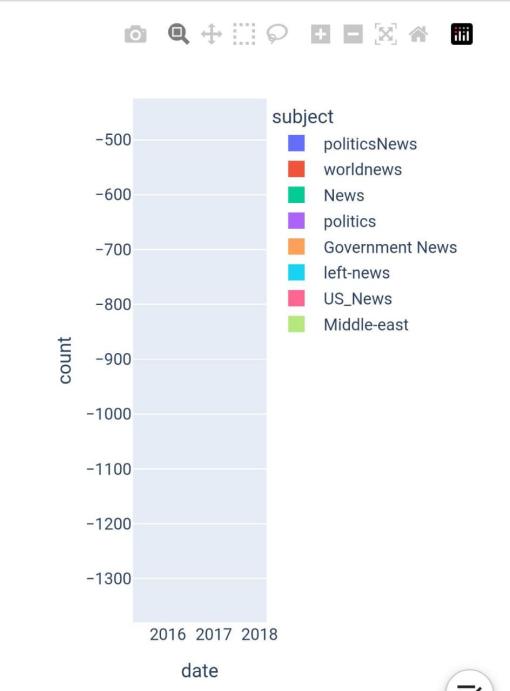
subject\_dist = px.histogram(data\_frame=news,

x='date',

color='subject')

subject\_dist.show()

OUTPUT :



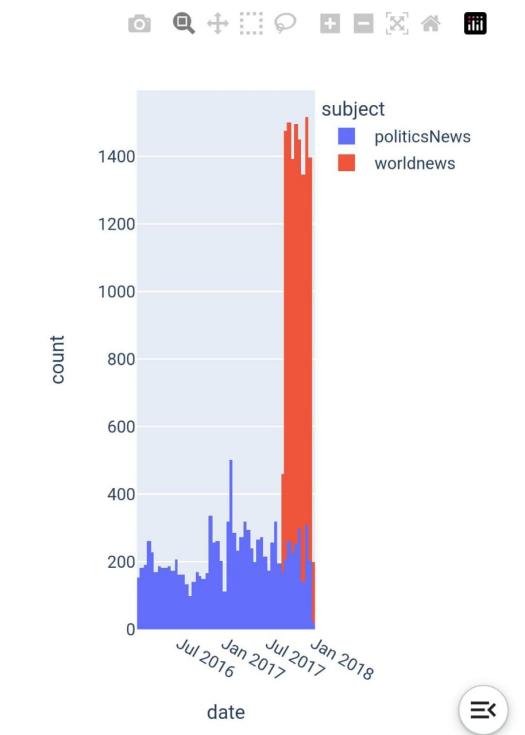
real\_sub\_dist = px.histogram(data\_frame=news[news['Label']=='Real'],

x='date',

color='subject')

real\_sub\_dist.show()

OUTPUT :



import string

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

def text\_preprocessing(text):

words = text.lower().split()

filtered\_words = [word for word in words if word not in stop\_words]

pure\_text = ' '.join(filtered\_words)

pure\_text = pure\_text.translate(str.maketrans('', '', string.punctuation)).strip()

return pure\_text

X = news\_ds.text.apply(text\_preprocessing).to\_numpy()

y = news\_ds.Label.to\_numpy().astype('float32').reshape(-1, 1)

train\_X, test\_X, train\_y, test\_y = train\_test\_split(X, y,

train\_size=0.9,

stratify=y,

random\_state=7)

train\_X, val\_X, train\_y, val\_y = train\_test\_split(train\_X, train\_y,

train\_size=0.9,

stratify=train\_y,

random\_state=7)

model\_name = "BERTFakeNewsDetector"

model\_callbacks = ModelCheckpoint(model\_name, save\_best\_only=True)

bert\_name = 'bert-base-uncased'

tokenizer = AutoTokenizer.from\_pretrained(bert\_name,

padding='max\_length',

do\_lower\_case=True,

add\_special\_tokens=True)

def tokenize(df):

inputs = tokenizer(df.tolist(),

padding=True,

truncation=True,

return\_tensors='tf').input\_ids

return inputs

train\_X\_encoded = tokenize(train\_X)

val\_X\_encoded = tokenize(val\_X)

test\_X\_encoded = tokenize(test\_X)

def prepare\_datasets(encoded, true\_df, true\_target\_df):

return tf.data.Dataset.from\_tensor\_slices((encoded, true\_target\_df)).shuffle(true\_df.shape[0]).batch(8).prefetch(tf.data.AUTOTUNE)

train\_ds = prepare\_datasets(train\_X\_encoded, train\_X, train\_y)

test\_ds = prepare\_datasets(test\_X\_encoded, test\_X, test\_y)

val\_ds = prepare\_datasets(val\_X\_encoded, val\_X, val\_y)

model = TFAutoModelForSequenceClassification.from\_pretrained(bert\_name,

num\_labels=1)

model.save(model\_name)

fig = make\_subplots(rows=2, cols=2, subplot\_titles=('Loss', 'Accuracy', 'Precision', 'Recall'))

fig.add\_trace(go.Scatter(y=model\_history['loss'], mode='lines', name='Training Loss'), row=1, col=1)

fig.add\_trace(go.Scatter(y=model\_history['val\_loss'], mode='lines', name='Validation Loss'), row=1, col=1)

fig.add\_trace(go.Scatter(y=model\_history['Accuracy'], mode='lines', name='Training Accuracy'), row=1, col=2)

fig.add\_trace(go.Scatter(y=model\_history['val\_Accuracy'], mode='lines', name='Validation Accuracy'), row=1, col=2)

fig.add\_trace(go.Scatter(y=model\_history['Precision'], mode='lines', name='Training Precision'), row=2, col=1)

fig.add\_trace(go.Scatter(y=model\_history['val\_Precision'], mode='lines', name='Validation Precision'), row=2, col=1)

fig.add\_trace(go.Scatter(y=model\_history['Recall'], mode='lines', name='Training Recall'), row=2, col=2)

fig.add\_trace(go.Scatter(y=model\_history['val\_Recall'], mode='lines', name='Validation Recall'), row=2, col=2)

fig.update\_layout(title='Model Training History')

fig.update\_xaxes(title\_text='Epoch', row=1, col=1)

fig.update\_xaxes(title\_text='Epoch', row=1, col=2)

fig.update\_xaxes(title\_text='Epoch', row=2, col=1)

fig.update\_xaxes(title\_text='Epoch', row=2, col=2)

fig.update\_yaxes(title\_text='Loss', row=1, col=1)

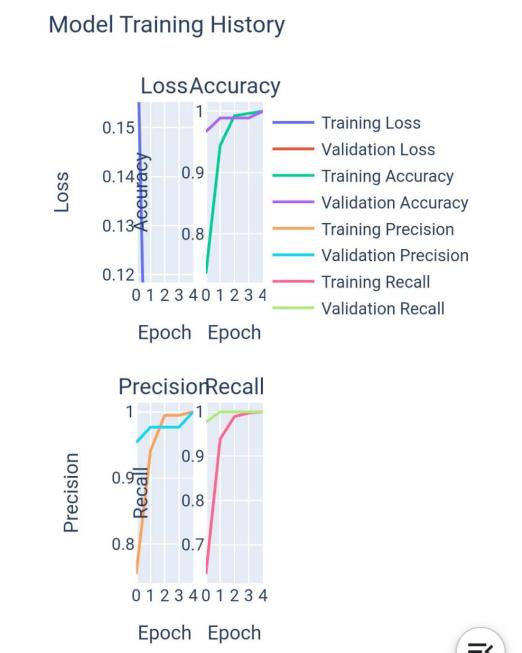
fig.update\_yaxes(title\_text='Accuracy', row=1, col=2)

fig.update\_yaxes(title\_text='Precision', row=2, col=1)

fig.update\_yaxes(title\_text='Recall', row=2, col=2)

fig.show()

OUTPUT :



test\_loss, test\_acc, test\_precision, test\_recall = model.evaluate(test\_ds, verbose = 0)

print(f"Test Loss : {test\_loss}")

print(f"Test Accuracy : {test\_acc}")

print(f"Test Precision : {test\_precision}")

print(f"Test Recall : {test\_recall}")

OUTPUT :



def make\_prediction(text, model=model):

text = np.array([text])

inputs = tokenize(text)

return np.abs(np.round(model.predict(inputs, verbose=1).logits))

for \_ in range(5):

index = np.random.randint(test\_X.shape[0])

text = test\_X[index]

real = test\_y[index]

model\_pred = make\_prediction(text)

print(f"Original Text:\n\n{text}\n\nTrue: {CLASS\_NAMES[int(real)]}\t\tPredicted: {CLASS\_NAMES[int(model\_pred)]}\n{'-'\*100}\n")