Project Description (AI_Capstone_Project**):**

“””

DESCRIPTION

Problem Statement

- Amazon is an online shopping website that now caters to millions of people everywhere. Over 34,000 consumer reviews for Amazon brand products like Kindle, Fire TV Stick and more are provided.
- The dataset has attributes like brand, categories, primary categories, reviews.title, reviews.text, and the sentiment. Sentiment is a categorical variable with three levels "Positive", "Negative“, and "Neutral". For a given unseen data, the sentiment needs to be predicted.
- You are required to predict Sentiment or Satisfaction of a purchase based on multiple features and review text.

Project Task: Week 1

Class Imbalance Problem:

1. Perform an EDA on the dataset.

   a) See what a positive, negative, and neutral review looks like

   b) Check the class count for each class. It’s a class imbalance problem.

2. Convert the reviews in Tf-Idf score.

3. Run multinomial Naive Bayes classifier. Everything will be classified as positive because of the class imbalance.

Project Task: Week 2

Tackling Class Imbalance Problem:

1. Oversampling or undersampling can be used to tackle the class imbalance problem.
2. In case of class imbalance criteria, use the following metrics for evaluating model performance: precision, recall, F1-score, AUC-ROC curve. Use F1-Score as the evaluation criteria for this project.
3. Use Tree-based classifiers like Random Forest and XGBoost.

Note: Tree-based classifiers work on two ideologies namely, Bagging or Boosting and have fine-tuning parameter which takes care of the imbalanced class.

Project Task: Week 3

Model Selection:

1. Apply multi-class SVM’s and neural nets.
2. Use possible ensemble techniques like: XGboost + oversampled_multinomial_NB.
3. Assign a score to the sentence sentiment (engineer a feature called sentiment score). Use this engineered feature in the model and check for improvements. Draw insights on the same.

Project Task: Week 4

Applying LSTM:

1. Use LSTM for the previous problem (use parameters of LSTM like top-word, embedding-length, Dropout, epochs, number of layers, etc.)

Hint: Another variation of LSTM, GRU (Gated Recurrent Units) can be tried as well.

2. Compare the accuracy of neural nets with traditional ML based algorithms.

3. Find the best setting of LSTM (Neural Net) and GRU that can best classify the reviews as positive, negative, and neutral.

Hint: Use techniques like Grid Search, Cross-Validation and Random Search

Optional Tasks: Week 4

Topic Modeling:

1. Cluster similar reviews.

Note: Some reviews may talk about the device as a gift-option. Other reviews may be about product looks and some may highlight about its battery and performance. Try naming the clusters.

2. Perform Topic Modeling

Hint: Use scikit-learn provided Latent Dirchlette Allocation (LDA) and Non-Negative Matrix Factorization (NMF).

“””

Source Code:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import string
import nltk
import seaborn as sns
from sklearn.dummy import DummyClassifier
from sklearn.metrics import precision_score, recall_score, confusion_matrix
from sklearn.metrics import f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score, accuracy_score
```

```
from sklearn.pipeline import Pipeline


from bs4 import BeautifulSoup

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from nltk.stem import SnowballStemmer, WordNetLemmatizer

from nltk import sent_tokenize, word_tokenize, pos_tag


import logging

from gensim.models import word2vec

from gensim.models.keyedvectors import KeyedVectors

from gensim.models import Word2Vec


from keras.preprocessing import sequence

from keras.utils import np_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Lambda

from keras.layers.embeddings import Embedding

from keras.layers.recurrent import LSTM, SimpleRNN, GRU
```

```python
from keras.preprocessing.text import Tokenizer

from collections import defaultdict

from keras.layers.convolutional import Convolution1D

from keras import backend as K

from keras.layers.embeddings import Embedding

from keras.callbacks import EarlyStopping




def preprocess(document):

    document = document.lower() # Convert to lowercase

    words = tokenizer.tokenize(document) # Tokenize

    words = [w for w in words if not w in stop_words] # Removing stopwords

    # Lemmatizing

    for pos in [wordnet.NOUN, wordnet.VERB, wordnet.ADJ, wordnet.ADV]:

        words = [wordnet_lemmatizer.lemmatize(x, pos) for x in words]

    return " ".join(words)


def textPreprocessing(data2):

    #Remove Punctuation Logic
```

```
    import string

    removePunctuation = [char for char in data2 if char not in string.punctuation]

    #Join Chars to form sentences

    sentenceWithoutPunctuations = ''.join(removePunctuation)

    words = sentenceWithoutPunctuations.split()

    #StopwordRemoval

    from nltk.corpus import stopwords

    removeStopwords = [word for word in words if word.lower() not in
stopwords.words('english')]



    return removeStopwords



def cleanText(raw_text, remove_stopwords=False, stemming=False,
split_text=False, \

             ):

    '''

    Convert a raw review to a cleaned review

    '''

    text = BeautifulSoup(raw_text, 'lxml').get_text()  #remove html

    letters_only = re.sub("[^a-zA-Z]", " ", text)  # remove non-character

    words = letters_only.lower().split() # convert to lower case
```

```
    if remove_stopwords: # remove stopword

        stops = set(stopwords.words("english"))

        words = [w for w in words if not w in stops]


    if stemming==True: # stemming

        stemmer = SnowballStemmer('english')

        words = [stemmer.stem(w) for w in words]


    if split_text==True:  # split text

        return (words)


    return( " ".join(words))




def modelEvaluation(predictions):

    '''

    Print model evaluation to predicted result

    '''

    print ("\nAccuracy on validation set: {:.4f}".format(accuracy_score(y_test, 
predictions)))

    #print("\nAUC score : {:.4f}".format(roc_auc_score(y_test, predictions)))
```

```
    print("\nClassification report : \n", metrics.classification_report(y_test,
predictions))

    print("\nConfusion Matrix : \n", metrics.confusion_matrix(y_test, predictions))




def parseSent(review, tokenizer, remove_stopwords=False):

    '''

    Parse text into sentences

    '''

    raw_sentences = tokenizer.tokenize(review.strip())

    sentences = []

    for raw_sentence in raw_sentences:

        if len(raw_sentence) > 0:

            sentences.append(cleanText(raw_sentence, remove_stopwords,
split_text=True))

    return sentences




def makeFeatureVec(review, model, num_features):

    '''
```

```
    Transform a review to a feature vector by averaging feature vectors of words
    appeared in that review and in the vocabulary list created
    '''
    featureVec = np.zeros((num_features,),dtype="float32")
    nwords = 0.
    index2word_set = set(model.wv.index_to_key) #index2word is the vocabulary list of the Word2Vec model
    isZeroVec = True
    for word in review:
        if word in index2word_set:
            nwords = nwords + 1.
            featureVec = np.add(featureVec, model.wv[word])
            isZeroVec = False
    if isZeroVec == False:
        featureVec = np.divide(featureVec, nwords)
    return featureVec



def getAvgFeatureVecs(reviews, model, num_features):
    '''
    Transform all reviews to feature vectors using makeFeatureVec()
```

```
    '''
    counter = 0
    reviewFeatureVecs = np.zeros((len(reviews),num_features),dtype="float32")
    for review in reviews:
        reviewFeatureVecs[counter] = makeFeatureVec(review,
model,num_features)
        counter = counter + 1
    return reviewFeatureVecs
```

```
#Reading Data sheets

file_path=input("enter path for the loan data file to load:")
df_path=file_path.replace("\\",'/')
data = pd.read_csv(df_path)

file_path=input("enter path for the loan data file to load:")
df_path=file_path.replace("\\",'/')
```

```
test = pd.read_csv(df_path)

file_path=input("enter path for the loan data file to load:")
df_path=file_path.replace("\\",'/')
test_prediction = pd.read_csv(df_path)

print(data.head())


#WEEK1


#See what a positive, negative, and neutral review looks like

Positive = data[data['sentiment']== "Positive"].iloc[:,[5,6,7]]
Neutral = data[data['sentiment']== "Neutral"].iloc[:,[5,6,7]]
Negative = data[data['sentiment']== "Negative"].iloc[:,[5,6,7]]

print("See what a positive, negative, and neutral review looks like:")

positive=data[['sentiment']]== "Positive"
```

```
sns.distplot(positive['sentiment'])

plt.title("Positive Reviews")

plt.show()


negative=data[['sentiment']]== "Negative"

sns.displot(negative['sentiment'])

plt.title("Negative Reviews")

plt.show()


neutral=data[['sentiment']]== "Neutral"

sns.distplot(neutral['sentiment'])

plt.title("Neutral Reviews")

plt.show()

print("_________________________________________________________________________")


#Check the class count for each class. It’s a class imbalance problem

print("Check the class count for each class")

print("-------------------------------------\n")

print(Positive['sentiment'].value_counts())

print(Neutral['sentiment'].value_counts())
```

```
print(Negative['sentiment'].value_counts())

print("_______________________________________________________________________________")

# Keeping only those Features that we need for further exploring.

data1 = data [["sentiment","reviews.text"]]


# Resetting the Index.

data1.index = pd.Series(list(range(data1.shape[0])))


from nltk.tokenize import RegexpTokenizer

from nltk.corpus import stopwords

import nltk

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

#Download Stopwords

nltk.download('stopwords')


wordnet_lemmatizer = WordNetLemmatizer()

tokenizer = RegexpTokenizer(r'[a-z]+')

stop_words = set(stopwords.words('english'))
```

```
data1['Processed_Review'] = data1['reviews.text'].apply(preprocess)

data2 = data1 [["sentiment","Processed_Review"]]

print(data2.groupby('sentiment').describe())

#Text preprocessing

data2['Processed_Review'].head(2).apply(textPreprocessing)

from sklearn.feature_extraction.text import CountVectorizer

bow =
CountVectorizer(analyzer=textPreprocessing).fit(data2['Processed_Review'])

reviews_bow = bow.transform(data2['Processed_Review'])

print("_______________________________________________________")

#Convert the reviews in Tf-Idf score.

from sklearn.feature_extraction.text import TfidfTransformer
```

```
tfidfData = TfidfTransformer().fit(reviews_bow)

tfidfDataFinal = tfidfData.transform(reviews_bow)

print("Convert the reviews in Tf-Idf score:\n")

print(tfidfDataFinal)


print("____________________________________________________________________________________")

#Run multinomial Naive Bayes classifier. Everything will be classified as positive because of the class imbalance.

print("Run multinomial Naive Bayes classifier. Everything will be classified as positive because of the class imbalance.")

from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB().fit(tfidfDataFinal,data2['sentiment'])


inputData = "very bad dont like it at all it sucks"

l1 = textPreprocessing(inputData)

l2 = bow.transform(l1)

l3 = tfidfData.transform(l2)

prediction = model.predict(l3[0])

print(prediction)


print("______________________________________________________________________________________________________________________")
```

#Creating independent and Dependent Features

columns = data2.columns.tolist()

Filtering the columns to remove data we do not want

columns = [c for c in columns if c not in ["sentiment"]]

Store the variable we are predicting

target = "sentiment"

Defining a random state

state = np.random.RandomState(42)

X = data2[columns]

Y = data2[target]

#WEEK2

#Oversampling or undersampling can be used to tackle the class imbalance problem

print("Oversampling or undersampling can be used to tackle the class imbalance problem")

RandomOverSampler to handle imbalanced data

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)

X_res,Y_res=ros.fit_resample(X,Y)

```
from collections import Counter

print(sorted(Counter(Y_res).items()))


#Checking out both old & new data

print('Original dataset shape {}'.format(Counter(Y)))

print('Resampled dataset shape {}'.format(Counter(Y_res)))


#Creating X output to dataframe

X1=pd.DataFrame(X_res,columns=['Processed_Review'])


#Creating Y output to dataframe for merging

Y1=pd.DataFrame(Y_res,columns=['sentiment'])

#Merging the X & Y output to Final data

Final_data=pd.concat([X1,Y1],axis=1)

print(Final_data.head())



df = Final_data.sample(frac=0.1, random_state=0)


# Dropping missing values
```

```
df.dropna(inplace=True)


df.head()

print("______________________________________________________________________
_______________")


# Splitting data into training set and validation

X_train, X_test, y_train, y_test = train_test_split(df['Processed_Review'],
df['sentiment'], \
                                                   test_size=0.1, random_state=0)




y_tra=y_train

# Preprocess text data in training set and validation set

X_train_cleaned = []

X_test_cleaned = []


for d in X_train:

    X_train_cleaned.append(cleanText(d))


for d in X_test:

    X_test_cleaned.append(cleanText(d))
```

```
# Fit and transform the training data to a document-term matrix using CountVectorizer

countVect = CountVectorizer()

X_train_countVect = countVect.fit_transform(X_train_cleaned)



# Train MultinomialNB classifier

mnb = MultinomialNB()

mnb.fit(X_train_countVect, y_train)

print("MultinomialNB classifier")

predictions = mnb.predict(countVect.transform(X_test_cleaned))

modelEvaluation(predictions)

print("____________________________________________________________")


print("XGBoost Classifier")

print("--------------------\n")

from xgboost import XGBClassifier

# Fitting and transforming the training data to a document-term matrix using TfidfVectorizer

tfidf = TfidfVectorizer(min_df=5) #minimum document frequency of 5
```

```
X_train_tfidf = tfidf.fit_transform(X_train)

print("Number of features : %d \n" %len(tfidf.get_feature_names())) #1722

print("Show some feature names : \n", tfidf.get_feature_names()[::1000])


# XGBoost Classifier

xgb = XGBClassifier()

xgb.fit(X_train_tfidf, y_train)


# Evaluating on the validaton set

predictions = xgb.predict(tfidf.transform(X_test_cleaned))

modelEvaluation(predictions)


print("_____________________________________________________________________
___")

sentences = []

for review in X_train_cleaned:

    sentences += parseSent(review, tokenizer)


from gensim.models import Word2Vec


w2v = Word2Vec()
```

```
# Fitting parsed sentences to Word2Vec model


num_features = 300  #embedding dimension

min_word_count = 10

num_workers = 4

context = 10

downsampling = 1e-3


w2v = Word2Vec(sentences, workers=num_workers, vector_size=num_features, min_count = min_word_count,\

               window = context, sample = downsampling)

w2v.init_sims(replace=True)

w2v.save("w2v_300features_10minwordcounts_10context") #save trained word2vec model




X_train_cleaned1 = []

for review in X_train:

   X_train_cleaned1.append(cleanText(review, remove_stopwords=True, split_text=True))
```

```
trainVector = getAvgFeatureVecs(X_train_cleaned1, w2v, num_features)



# Getting feature vectors for validation set

X_test_cleaned1 = []

for review in X_test:

   X_test_cleaned1.append(cleanText(review, remove_stopwords=True,
split_text=True))

testVector = getAvgFeatureVecs(X_test_cleaned1, w2v, num_features)




# Getting feature vectors for training set

trainVector = getAvgFeatureVecs(X_train, w2v, num_features)


# Getting feature vectors for validation set

testVector = getAvgFeatureVecs(X_test, w2v, num_features)


# Random Forest Classifier

print("Random Forest Classifier")

print("-------------------------\n")

rf = RandomForestClassifier(n_estimators=100)
```

```
rf.fit(trainVector, y_train)

predictions = rf.predict(testVector)

modelEvaluation(predictions)


df = Final_data.sample(frac=0.1, random_state=0)


# Drop missing values

df.dropna(inplace=True)




print("_______________________________________________________________________________")

#Apply multi-class SVM's and neural nets.

print("Apply multi-class SVM's and neural nets.")

print("----------------------------------------")

# Fitting and transforming the training data to a document-term matrix using TfidfVectorizer

tfidf = TfidfVectorizer(min_df=5) #minimum document frequency of 5

X_train_tfidf = tfidf.fit_transform(X_train)


# Logistic Regression
```

```
print("Logistic Regression")

lr = LogisticRegression()

lr.fit(X_train_tfidf, y_train)


# Have a look at the top 10 features with the smallest and largest coefficients

feature_names = np.array(tfidf.get_feature_names())

sorted_coef_index = lr.coef_[0].argsort()


# Evaluating on the validaton set

predictions = lr.predict(tfidf.transform(X_test_cleaned))

modelEvaluation(predictions)

print("_______________________________________________________")

# Fitting and transforming the training data to a document-term matrix using TfidfVectorizer

tfidf = TfidfVectorizer(min_df=5) #minimum document frequency of 5

X_train_tfidf = tfidf.fit_transform(X_train)




#Apply multi-class SVM's and neural nets.
```

```
# SVM

from sklearn.linear_model import SGDClassifier

clf = SGDClassifier(loss="hinge", penalty="l2")

clf.fit(X_train_tfidf, y_train)

print("SGDClassifier")

print("--------------\n")

# Have a look at the top 10 features with the smallest and largest coefficients

feature_names = np.array(tfidf.get_feature_names())

sorted_coef_index = clf.coef_[0].argsort()


# Evaluating on the validaton set

predictions = clf.predict(tfidf.transform(X_test_cleaned))

modelEvaluation(predictions)


print("_____________________________________________________________________
_______________")

from xgboost import XGBClassifier

# Fitting and transforming the training data to a document-term matrix using
TfidfVectorizer

tfidf = TfidfVectorizer(min_df=5) #minimum document frequency of 5

X_train_tfidf = tfidf.fit_transform(X_train)
```

```
# XGBoost Classifier

print("XGBoost Classifier")

print("----------------------")

xgb = XGBClassifier()

xgb.fit(X_train_tfidf, y_train)


# Look at the top 10 features with smallest and the largest coefficients

feature_names = np.array(tfidf.get_feature_names())

# sorted_coef_index = xgb.coef_[0].argsort()


# Evaluating on the validaton set

predictions = xgb.predict(tfidf.transform(X_test_cleaned))

modelEvaluation(predictions)


print("______________________________________________________________________________
_")



#Assign a score to the sentence sentiment


print("Assign a score to the sentence sentiment :\n")
```

```
print("-------------------------------------------")

# Convert the sentiments

df.sentiment.replace(('Positive','Negative','Neutral'),(1,0,2),inplace=True)


print(df.head())

print("____________________________________________________________________
__________")


# Splitting data into training set and validation

X_train, X_test, y_train, y_test = train_test_split(df['Processed_Review'],
df['sentiment'], \
                                                    test_size=0.1, random_state=1)


top_words = 20000

maxlen = 100

batch_size = 32

nb_classes = 3

nb_epoch = 3


# Vectorize X_train and X_test to 2D tensor
```

```
tokenizer = Tokenizer(nb_words=top_words) #Considering only top 20000 words in the corpus

tokenizer.fit_on_texts(X_train)

# tokenizer.word_index #access word-to-index dictionary of trained tokenizer


sequences_train = tokenizer.texts_to_sequences(X_train)

sequences_test = tokenizer.texts_to_sequences(X_test)


X_train_seq = sequence.pad_sequences(sequences_train, maxlen=maxlen)

X_test_seq = sequence.pad_sequences(sequences_test, maxlen=maxlen)



# One-Hot Encoding of y_train and y_test

y_train_seq = np_utils.to_categorical(y_train, nb_classes)

y_test_seq = np_utils.to_categorical(y_test, nb_classes)




# Constructing a Simple LSTM

print("Constructing a Simple LSTM")
```

```
print("---------------------------\n")
model1 = Sequential()
model1.add(Embedding(top_words, 128))
model1.add(Dropout(0.2))
model1.add(LSTM(128))
model1.add(Dropout(0.2))
model1.add(Dropout(0.2))
model1.add(Dense(nb_classes))
model1.add(Activation('softmax'))
model1.summary()

# Compiling LSTM
model1.compile(loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

model1.fit(X_train_seq, y_train_seq, batch_size=batch_size, epochs=nb_epoch, verbose=1)

# Model Evaluation
score = model1.evaluate(X_test_seq, y_test_seq, batch_size=batch_size)
```

```
print('Test loss : {:.4f}'.format(score[0]))

print('Test accuracy : {:.4f}'.format(score[1]))


# Getting weight matrix of the embedding layer

model1.layers[0].get_weights()[0] # weight matrix of the embedding layer, word-by-dim matrix

print("Size of weight matrix in the embedding layer : ", \

    model1.layers[0].get_weights()[0].shape) #(20000, 128)


# Getting weight matrix of the hidden layer

print("Size of weight matrix in the hidden layer : ", \

    model1.layers[2].get_weights()[0].shape) #(128, 512)  weight dim of LSTM - w


# Getting weight matrix of the output layer

print("Size of weight matrix in the output layer : ", \

    model1.layers[5].get_weights()[0].shape) #(128, 2) weight dim of dense layer



# Loading pretrained Word2Vec model

w2v = Word2Vec.load("w2v_300features_10minwordcounts_10context")
```

Getting Word2Vec embedding matrix

embedding_matrix = w2v.wv.vectors # embedding matrix, type = numpy.ndarray

print("Shape of embedding matrix : ", embedding_matrix.shape) #(4016, 300) = (vocabulary size, embedding dimension)

w2v.wv.syn0[0] #feature vector of the first word in the vocabulary list

top_words = embedding_matrix.shape[0] #4016

maxlen = 100

batch_size = 32

nb_classes = 3

nb_epoch = 3

Vectorizing X_train and X_test to 2D tensor

tokenizer = Tokenizer(nb_words=top_words) #Considering only top 20000 words in the corpus

tokenizer.fit_on_texts(X_train)

tokenizer.word_index #access word-to-index dictionary of trained tokenizer

sequences_train = tokenizer.texts_to_sequences(X_train)

```
sequences_test = tokenizer.texts_to_sequences(X_test)


X_train_seq = sequence.pad_sequences(sequences_train, maxlen=maxlen)

X_test_seq = sequence.pad_sequences(sequences_test, maxlen=maxlen)



# One-Hot Encoding of y_train and y_test

y_train_seq = np_utils.to_categorical(y_train, nb_classes)

y_test_seq = np_utils.to_categorical(y_test, nb_classes)


print('X_train shape:', X_train_seq.shape) #(27799, 100)

print('X_test shape:', X_test_seq.shape) #(3089, 100)

print('y_train shape:', y_train_seq.shape) #(27799, 2)

print('y_test shape:', y_test_seq.shape) #(3089, 2)



# Constructing Word2Vec embedding layer

embedding_layer = Embedding(embedding_matrix.shape[0], #4016

                    embedding_matrix.shape[1], #300

                    weights=[embedding_matrix])
```

```
print("___________________________________________________________________
______")

# Constructing LSTM with Word2Vec embedding

print("Constructing LSTM with Word2Vec embedding")

print("-----------------------------------------\n")

model2 = Sequential()

model2.add(embedding_layer)

model2.add(LSTM(128))

model2.add(Dropout(0.2))

model2.add(Dropout(0.2))

model2.add(Dense(nb_classes))

model2.add(Activation('softmax'))

model2.summary()


# Compiling model

model2.compile(loss='binary_crossentropy',

            optimizer='adam',

            metrics=['accuracy'])


model2.fit(X_train_seq, y_train_seq, batch_size=batch_size, epochs=nb_epoch,
verbose=1)
```

```
# Model evaluation
score = model2.evaluate(X_test_seq, y_test_seq, batch_size=batch_size)
print('Test loss : {:.4f}'.format(score[0]))
print('Test accuracy : {:.4f}'.format(score[1]))


# Getting weight matrix of the embedding layer
print("Size of weight matrix in the embedding layer : ", \
    model2.layers[0].get_weights()[0].shape) #(20000, 128)

# Getting weight matrix of the hidden layer
print("Size of weight matrix in the hidden layer : ", \
    model2.layers[1].get_weights()[0].shape) #(128, 512)  weight dim of LSTM - w

# Getting weight matrix of the output layer
print("Size of weight matrix in the output layer : ", \
    model2.layers[4].get_weights()[0].shape) #(128, 2) weight dim of dense layer
```

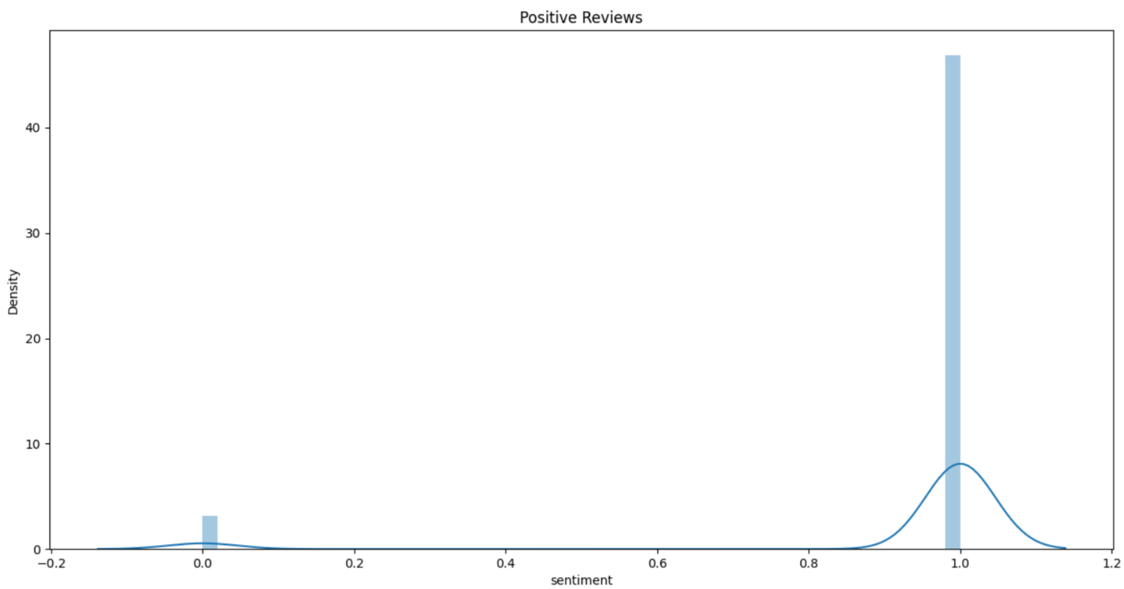
```
print("_____________________________________________________________
_____________________________________________________________")

print("Find the best setting of LSTM (Neural Net) and GRU that can best classify the reviews as positive, negative, and neutral.\nHint: Use techniques like Grid Search, Cross-Validation and Random Search")

print("-----------------------------------------------------------------------------------------------------------")

# Building a pipeline

estimators = [("tfidf", TfidfVectorizer()), ("lr", LogisticRegression())]

model = Pipeline(estimators)

# Grid search

params = {"lr__C":[0.1, 1, 10], #regularization param of logistic regression

        "tfidf__min_df": [1, 3], #min count of words

        "tfidf__max_features": [1000, None], #max features

        "tfidf__ngram_range": [(1,1), (1,2)], #1-grams or 2-grams

        "tfidf__stop_words": [None, "english"]} #use stopwords or don't

grid = GridSearchCV(estimator=model, param_grid=params, scoring="accuracy", n_jobs=-1)

grid.fit(X_train, y_train)

print("\nGrid search:\n")

print("The best paramenter set is : \n", grid.best_params_)

predictions = grid.predict(X_test)

modelEvaluation(predictions)
```

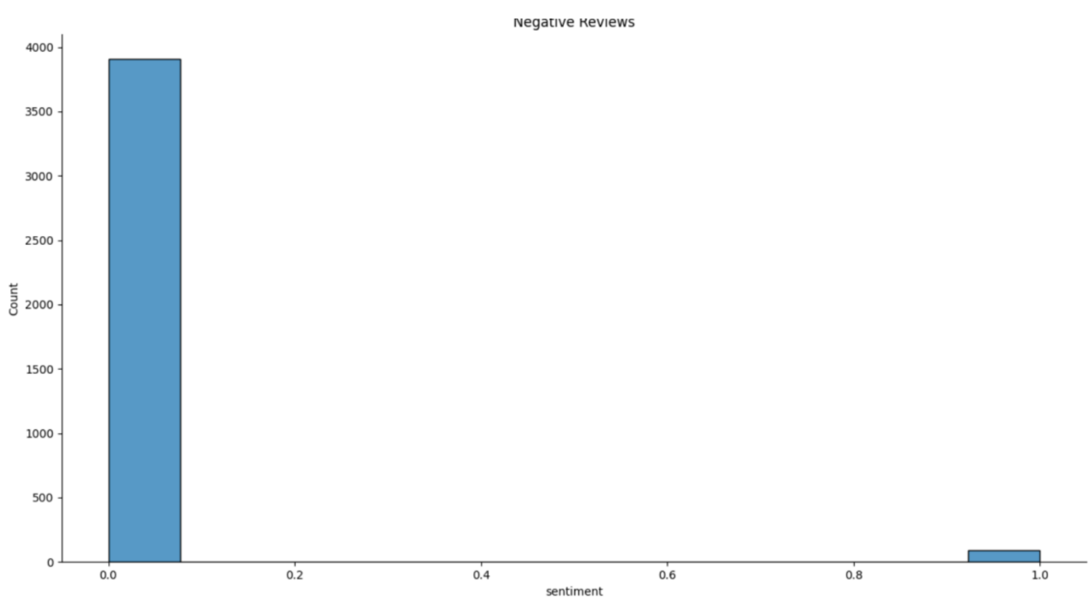
Screenshot of the output:

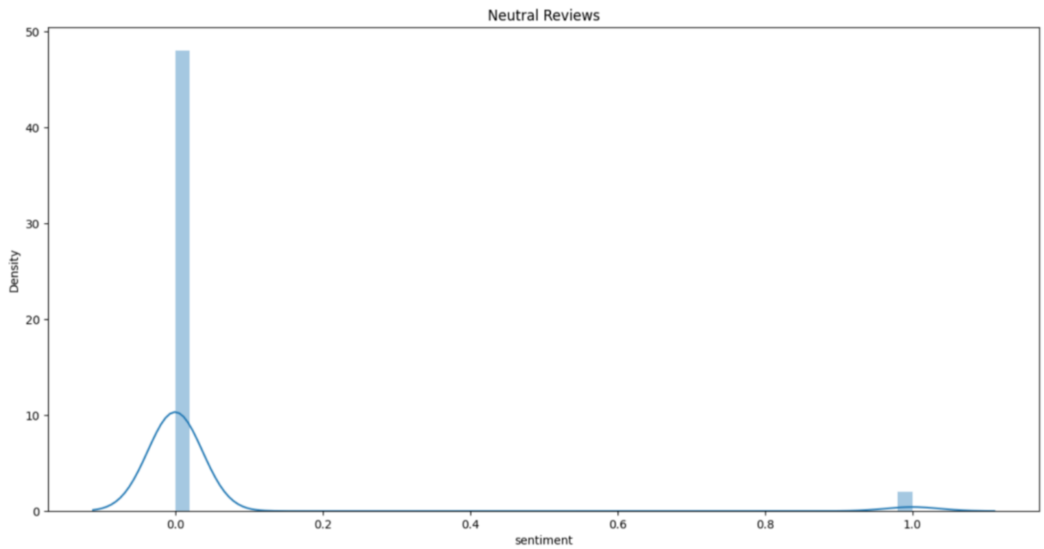
Week 1:

Task1: Perform an EDA on the dataset.

a) **See what a positive, negative, and neutral review looks like**







b) Check the class count for each class. It's a class imbalance problem.:

```
Check the class count for each class
-------------------------------------

Positive    3749
Name: sentiment, dtype: int64
Neutral    158
Name: sentiment, dtype: int64
Negative    93
Name: sentiment, dtype: int64
```

Task 2: Convert the reviews in Tf-Idf score:

```
Convert the reviews in Tf-Idf score:

  (0, 3292)	0.12348731897288433
  (0, 2955)	0.13262765127937107
  (0, 2894)	0.22889129312178152
  (0, 2861)	0.22889129312178152
  (0, 2782)	0.22889129312178152
  (0, 2565)	0.13750705381771683
  (0, 2480)	0.18120952660343198
  (0, 2361)	0.21044546789025603
  (0, 2350)	0.08672269357481559
  (0, 2315)	0.17527129722802176
  (0, 2292)	0.08954066746770499
  (0, 2259)	0.18120952660343198
  (0, 2210)	0.09077190449626162
  (0, 2139)	0.14545045985637628
  (0, 2055)	0.08206248864024848
  (0, 1971)	0.09636033823619856
  (0, 1922)	0.13279862678760768
  (0, 1523)	0.22889129312178152
  (0, 1348)	0.12108339495186818
  (0, 1319)	0.13174106906037267
  (0, 1236)	0.22889129312178152
  (0, 1128)	0.11971297627771318
  (0, 1062)	0.17527129722802176
  (0, 1041)	0.11719661158900058
  (0, 678)	0.21044546789025603
  :	:
```

Task 3: Run multinomial Naive Bayes classifier. Everything will be classified as positive because of the class imbalance.:

```
Run multinomial Naive Bayes classifier. Everything will be classified as positive because of the class imbalance.
['Positive']
```

WEEK 2:

Task 1: Oversampling or undersampling can be used to tackle the class imbalance problem.:

```
Oversampling or undersampling can be used to tackle the class imbalance problem
[('Negative', 3749), ('Neutral', 3749), ('Positive', 3749)]
Original dataset shape Counter({'Positive': 3749, 'Neutral': 158, 'Negative': 93})
Resampled dataset shape Counter({'Positive': 3749, 'Neutral': 3749, 'Negative': 3749})
                                    Processed_Review sentiment
0  purchase black fridaypros great price even sal...  Positive
1  purchase two amazon echo plus two dot plus fou...  Positive
2  average alexa option show thing screen still l...   Neutral
3               good product exactly want good price  Positive
4  rd one purchase buy one niece case compare one...  Positive
```

Task 2: In case of class imbalance criteria, use the following metrices for evaluating model performance: precision, recall, F1-score, AUC-ROC curve. Use F1-Score as the evaluation criteria for this project.:

```
MultinomialNB classifier

Accuracy on validation set: 0.8938

Classification report :
              precision    recall  f1-score   support

    Negative       0.93      0.95      0.94        39
     Neutral       0.85      0.90      0.88        39
    Positive       0.91      0.83      0.87        35

    accuracy                           0.89       113
   macro avg       0.89      0.89      0.89       113
weighted avg       0.89      0.89      0.89       113


Confusion Matrix :
 [[37  0  2]
 [ 3 35  1]
 [ 0  6 29]]
```

Task 3: Use Tree-based classifiers like Random Forest and XGBoost.:

```
XGBoost Classifier
-------------------

Number of features : 691

Show some feature names :
 ['able']
[08:43:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the
 from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

Accuracy on validation set: 0.9292

Classification report :
              precision    recall  f1-score   support

    Negative       0.95      0.97      0.96        39
     Neutral       0.90      0.95      0.92        39
    Positive       0.94      0.86      0.90        35

    accuracy                           0.93       113
   macro avg       0.93      0.93      0.93       113
weighted avg       0.93      0.93      0.93       113


Confusion Matrix :
 [[38  0  1]
 [ 1 37  1]
 [ 1  4 30]]
```

```
Random Forest Classifier
-------------------------


Accuracy on validation set: 0.4602

Classification report :
              precision    recall  f1-score   support

    Negative       0.49      0.69      0.57        39
     Neutral       0.44      0.36      0.39        39
    Positive       0.42      0.31      0.36        35

    accuracy                           0.46       113
   macro avg       0.45      0.46      0.44       113
weighted avg       0.45      0.46      0.45       113


Confusion Matrix :
 [[27  6  6]
 [16 14  9]
 [12 12 11]]
```

WEEK 3:

Task 1: Apply multi-class SVM’s and neural nets.:

```
Apply multi-class SVM’s and neural nets.
----------------------------------------
Logistic Regression

Accuracy on validation set: 0.9292

Classification report :
               precision    recall  f1-score   support

    Negative       0.93      1.00      0.96        39
     Neutral       0.88      0.92      0.90        39
    Positive       1.00      0.86      0.92        35

    accuracy                           0.93       113
   macro avg       0.94      0.93      0.93       113
weighted avg       0.93      0.93      0.93       113


Confusion Matrix :
 [[39  0  0]
 [ 3 36  0]
 [ 0  5 30]]
```

```
SGDClassifier
--------------


Accuracy on validation set: 0.9204

Classification report :
               precision    recall  f1-score   support

    Negative       0.93      1.00      0.96        39
     Neutral       0.86      0.95      0.90        39
    Positive       1.00      0.80      0.89        35

    accuracy                           0.92       113
   macro avg       0.93      0.92      0.92       113
weighted avg       0.93      0.92      0.92       113


Confusion Matrix :
 [[39  0  0]
 [ 2 37  0]
 [ 1  6 28]]
```

Task 2: Use possible ensemble techniques like: XGboost + oversampled_multinomial_NB.:

```
XGBoost Classifier
----------------------
[08:44:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src
 from 'merror' to 'mlogloss'. Explicitly set eval_metric if you'd like to restore th

Accuracy on validation set: 0.9292

Classification report :
               precision    recall  f1-score   support

    Negative       0.95      0.97      0.96        39
     Neutral       0.90      0.95      0.92        39
    Positive       0.94      0.86      0.90        35

    accuracy                           0.93       113
   macro avg       0.93      0.93      0.93       113
weighted avg       0.93      0.93      0.93       113


Confusion Matrix :
 [[38  0  1]
 [ 1 37  1]
 [ 1  4 30]]
```

Task 3: Assign a score to the sentence sentiment:

```
Assign a score to the sentence sentiment :

---------------------------------------------
                                        Processed_Review  sentiment
8805   buy think would great read book play game howe...          2
9736             good tablet kid lot appts download game          2
125                       item work expect great product          1
10143  great beginner like child limit use many apps ...          2
10937  buy kindle past time one come defective port b...          2
```

WEEK 4:

Task 1: Use LSTM for the previous problem (use parameters of LSTM like top-word, embedding-length, Dropout, epochs, number of layers, etc.):

```
Constructing a Simple LSTM
--------------------------

2022-03-02 08:44:04.153837: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic l
2022-03-02 08:44:04.153942: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERR
2022-03-02 08:44:04.157280: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic in
2022-03-02 08:44:04.157576: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: LAPTOP-TU8FR7UU
2022-03-02 08:44:04.158152: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized
ormance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 128)         2560000
_________________________________________________________________
dropout (Dropout)            (None, None, 128)         0
_________________________________________________________________
lstm (LSTM)                  (None, 128)               131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 387
_________________________________________________________________
activation (Activation)      (None, 3)                 0
=================================================================
Total params: 2,691,971
Trainable params: 2,691,971
Non-trainable params: 0
_________________________________________________________________
2022-03-02 08:44:04.648983: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimiz
Epoch 1/3
32/32 [==============================] - 7s 174ms/step - loss: 0.6507 - accuracy: 0.4140
Epoch 2/3
32/32 [==============================] - 5s 171ms/step - loss: 0.5617 - accuracy: 0.6709
Epoch 3/3
32/32 [==============================] - 6s 174ms/step - loss: 0.2963 - accuracy: 0.8715
4/4 [==============================] - 0s 24ms/step - loss: 0.2711 - accuracy: 0.8407
Test loss : 0.2711
Test accuracy : 0.8407
Size of weight matrix in the embedding layer :  (20000, 128)
Size of weight matrix in the hidden layer :  (128, 512)
Size of weight matrix in the output layer :  (128, 3)
Shape of embedding matrix :  (416, 300)
X_train shape: (1012, 100)
X_test shape: (113, 100)
```

Task 3: Find the best setting of LSTM (Neural Net) and GRU that can best classify the reviews as positive, negative, and neutral.

Hint: Use techniques like Grid Search, Cross-Validation and Random Search:

```
Find the best setting of LSTM (Neural Net) and GRU that can best classify the reviews as positive, negative, and neutral.
Hint: Use techniques like Grid Search, Cross-Validation and Random Search
------------------------------------------------------------------------------------------------------------------

Grid search:

The best paramenter set is :
 {'lr__C': 10, 'tfidf__max_features': None, 'tfidf__min_df': 1, 'tfidf__ngram_range': (1, 2), 'tfidf__stop_words': None}

Accuracy on validation set: 0.9381

Classification report :
               precision    recall  f1-score   support

           0       0.97      1.00      0.99        33
           1       0.94      0.89      0.91        35
           2       0.91      0.93      0.92        45

    accuracy                           0.94       113
   macro avg       0.94      0.94      0.94       113
weighted avg       0.94      0.94      0.94       113


Confusion Matrix :
 [[33  0  0]
 [ 0 31  4]
 [ 1  2 42]]
```