Input Layer

```
import pandas as pd
import numpy as np

reviews_df = pd.read_csv('singapore_airlines_reviews.csv')
reviews_df
```

	published_date	published_platform	rating	type	text	title	h
0	2024-03-12T14:41:14-04:00	Desktop	3	review	We used this airline to go from Singapore to L...	Ok	
1	2024-03-11T19:39:13-04:00	Desktop	5	review	The service on Singapore Airlines Suites Class...	The service in Suites Class makes one feel lik...	
2	2024-03-11T12:20:23-04:00	Desktop	1	review	Booked, paid and received email confirmation f...	Don’t give them your money	
	2024-03-				Best airline in the world	Best Airline	

Next steps: Generate code with reviews_df View recommended plots

```
# Map ratings to sentiment categories
reviews_df['sentiment'] = reviews_df['rating'].apply(lambda x: 'positive' if x > 

# For simplicity in this example, we'll focus on the 'text' and 'sentiment' colum
data = reviews_df[['text', 'sentiment']]

# Preprocessing: Lowercase conversion and removing punctuation (a simple example 
data['text'] = data['text'].str.lower().str.replace('[^\w\s]', '', regex=True)

data['sentiment'].value_counts()
```

```
<ipython-input-2-0b0e5fb8260d>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['text'] = data['text'].str.lower().str.replace('[^\w\s]', '', regex=Tru
sentiment
positive    7391
negative    1600
neutral     1009
Name: count, dtype: int64
```

```
print(data)
```

```
                                                   text sentiment
0     we used this airline to go from singapore to l...   neutral
1     the service on singapore airlines suites class...  positive
2     booked paid and received email confirmation fo...  negative
3     best airline in the world seats food service a...  positive
4     premium economy seating on singapore airlines ...  negative
...                                                 ...       ...
9995  first part done with singapore airlines  accep...  positive
9996  and again a great flight with singapore air gr...  positive
9997  we flew business class from frankfurt via sing...  positive
9998  as always the a380 aircraft was spotlessly pre...  positive
9999  as always singapore airlines has done it again...  positive

[10000 rows x 2 columns]
```

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

Process module with NLTK

```
# sentence tokenization
from nltk.tokenize import sent_tokenize

data['sentence'] = data['text'].apply(sent_tokenize)

print(data[['sentence']].head())
```

```
                                            sentence
0  [we used this airline to go from singapore to ...
1  [the service on singapore airlines suites clas...
2  [booked paid and received email confirmation f...
3  [best airline in the world seats food service ...
4  [premium economy seating on singapore airlines...
<ipython-input-6-dce104faeb69>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['sentence'] = data['text'].apply(sent_tokenize)
```

```
# word tokenization
from nltk.tokenize import word_tokenize

data['word'] = data['text'].apply(word_tokenize)

print(data)
```

```
                                                   text sentiment  \
0     we used this airline to go from singapore to l...   neutral
1     the service on singapore airlines suites class...  positive
2     booked paid and received email confirmation fo...  negative
3     best airline in the world seats food service a...  positive
4     premium economy seating on singapore airlines ...  negative
...                                                 ...       ...
9995  first part done with singapore airlines  accep...  positive
9996  and again a great flight with singapore air gr...  positive
9997  we flew business class from frankfurt via sing...  positive
9998  as always the a380 aircraft was spotlessly pre...  positive
9999  as always singapore airlines has done it again...  positive

                                               sentence  \
0     [we used this airline to go from singapore to ...
1     [the service on singapore airlines suites clas...
2     [booked paid and received email confirmation f...
3     [best airline in the world seats food service ...
4     [premium economy seating on singapore airlines...
...                                                 ...
9995  [first part done with singapore airlines  acce...
9996  [and again a great flight with singapore air g...
9997  [we flew business class from frankfurt via sin...
9998  [as always the a380 aircraft was spotlessly pr...
9999  [as always singapore airlines has done it agai...

                                                   word
0     [we, used, this, airline, to, go, from, singap...
1     [the, service, on, singapore, airlines, suites...
2     [booked, paid, and, received, email, confirmat...
3     [best, airline, in, the, world, seats, food, s...
4     [premium, economy, seating, on, singapore, air...
...                                                 ...
9995  [first, part, done, with, singapore, airlines,...
9996  [and, again, a, great, flight, with, singapore...
9997  [we, flew, business, class, from, frankfurt, v...
9998  [as, always, the, a380, aircraft, was, spotles...
9999  [as, always, singapore, airlines, has, done, i...

[10000 rows x 4 columns]
<ipython-input-7-4abeeb624a17>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['word'] = data['text'].apply(word_tokenize)
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# stop words removal
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

def remove_stop_words_from_list(words_list):
    return [word for word in words_list if word.lower() not in stop_words]

# Apply the function to each cell in the 'words' column
data['filtered_words'] = data['word'].apply(remove_stop_words_from_list)
```

```
<ipython-input-9-064539d8f519>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['filtered_words'] = data['word'].apply(remove_stop_words_from_list)
```

```
print(data['filtered_words'])
```

```
0       [used, airline, go, singapore, london, heathro...
1       [service, singapore, airlines, suites, class, ...
2       [booked, paid, received, email, confirmation, ...
3       [best, airline, world, seats, food, service, b...
4       [premium, economy, seating, singapore, airline...
                              ...
9995    [first, part, done, singapore, airlines, accep...
9996    [great, flight, singapore, air, great, unique,...
9997    [flew, business, class, frankfurt, via, singap...
9998    [always, a380, aircraft, spotlessly, presented...
9999    [always, singapore, airlines, done, redeye, fl...
Name: filtered_words, Length: 10000, dtype: object
```

```
# Stemmer
from nltk.stem import PorterStemmer

# Initialize the PorterStemmer
stemmer = PorterStemmer()

def stem_words(words_list):
    return [stemmer.stem(word) for word in words_list]

# Assuming 'filtered_words' is your column with stop words removed
data['filtered_words'] = data['filtered_words'].apply(stem_words)
```

```
<ipython-input-11-44cc522bfe27>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['filtered_words'] = data['filtered_words'].apply(stem_words)
```

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# Lemmatization
# Initialize the WordNet Lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to lemmatize each word in a list of words
def lemmatize_words(words_list):
    return [lemmatizer.lemmatize(word) for word in words_list]

# Apply the function to lemmatize the words in each list
data['filtered_words'] = data['filtered_words'].apply(lemmatize_words)
```

```
<ipython-input-13-4637ecbd9e46>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['filtered_words'] = data['filtered_words'].apply(lemmatize_words)
```

```
print(data)
```

```
                                                   text sentiment  \
0     we used this airline to go from singapore to l...   neutral
1     the service on singapore airlines suites class...  positive
2     booked paid and received email confirmation fo...  negative
3     best airline in the world seats food service a...  positive
4     premium economy seating on singapore airlines ...  negative
...                                                 ...       ...
9995  first part done with singapore airlines  accep...  positive
9996  and again a great flight with singapore air gr...  positive
9997  we flew business class from frankfurt via sing...  positive
9998  as always the a380 aircraft was spotlessly pre...  positive
9999  as always singapore airlines has done it again...  positive

                                               sentence  \
0     [we used this airline to go from singapore to ...
1     [the service on singapore airlines suites clas...
2     [booked paid and received email confirmation f...
3     [best airline in the world seats food service ...
4     [premium economy seating on singapore airlines...
...                                                 ...
9995  [first part done with singapore airlines  acce...
9996  [and again a great flight with singapore air g...
9997  [we flew business class from frankfurt via sin...
9998  [as always the a380 aircraft was spotlessly pr...
9999  [as always singapore airlines has done it agai...
```

```
                                                   word  \
0     [we, used, this, airline, to, go, from, singap...
1     [the, service, on, singapore, airlines, suites...
2     [booked, paid, and, received, email, confirmat...
3     [best, airline, in, the, world, seats, food, s...
4     [premium, economy, seating, on, singapore, air...
...                                                 ...
9995  [first, part, done, with, singapore, airlines,...
9996  [and, again, a, great, flight, with, singapore...
9997  [we, flew, business, class, from, frankfurt, v...
9998  [as, always, the, a380, aircraft, was, spotles...
9999  [as, always, singapore, airlines, has, done, i...

                                          filtered_words
0     [use, airlin, go, singapor, london, heathrow, ...
1     [servic, singapor, airlin, suit, class, noth, ...
2     [book, paid, receiv, email, confirm, extra, le...
3     [best, airlin, world, seat, food, servic, bril...
4     [premium, economi, seat, singapor, airlin, nar...
...                                                 ...
9995  [first, part, done, singapor, airlin, accept, ...
9996  [great, flight, singapor, air, great, uniqu, s...
9997  [flew, busi, class, frankfurt, via, singapor, ...
9998  [alway, a380, aircraft, spotlessli, present, b...
9999  [alway, singapor, airlin, done, redey, flight,...

[10000 rows x 5 columns]
```

Part of Speech Tagging

```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
True
```

```
from nltk import pos_tag
from collections import Counter

data['sentence_pos_counts'] = data['filtered_words'].apply(lambda tokens: Counter

print(data['sentence_pos_counts'])
```

```
0       {'NN': 69, 'VBP': 8, 'JJ': 24, 'CD': 5, 'VB': ...
1       {'JJ': 56, 'NN': 269, 'DT': 2, 'VBP': 11, 'VBD...
2       {'NN': 29, 'VBD': 3, 'JJ': 9, 'CD': 6, 'VBN': ...
3                  {'JJS': 1, 'NN': 11, 'VBD': 1, 'JJ': 1}
4       {'JJ': 17, 'NN': 34, 'VBD': 1, 'IN': 1, 'VBP':...
                              ...
9995    {'JJ': 2, 'NN': 15, 'VBN': 1, 'IN': 1, 'NNS': ...
9996      {'JJ': 4, 'NN': 10, 'VB': 1, 'NNS': 1, 'IN': 1}
9997      {'JJ': 8, 'NN': 21, 'IN': 1, 'VBD': 4, 'RB': 1}
9998    {'RB': 1, 'NN': 21, 'JJ': 5, 'VBD': 1, 'CD': 1...
9999    {'RB': 6, 'JJ': 7, 'NN': 12, 'VBN': 1, 'VBP': ...
Name: sentence_pos_counts, Length: 10000, dtype: object
<ipython-input-122-3968acac01d7>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['sentence_pos_counts'] = data['filtered_words'].apply(lambda tokens: Co
```

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

data['filtered_words'] = data['filtered_words'].apply(' '.join)

vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=10000, use_idf=T

tfidf_matrix = vectorizer.fit_transform(data['filtered_words'])
```

```
<ipython-input-123-67596a9b6dbb>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  data['filtered_words'] = data['filtered_words'].apply(' '.join)
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# First, split the data into training plus validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(tfidf_matrix, data['s

# Now split the training plus validation set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_

# Initialize a classifier (example: Random Forest)
classifier = RandomForestClassifier()

# Train the classifier on the training set
classifier.fit(X_train, y_train)

# Optionally evaluate on the validation set
val_predictions = classifier.predict(X_val)

# Make predictions on the test set
test_predictions = classifier.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, classification_report

print("Accuracy:", accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
Accuracy: 0.8105
              precision    recall  f1-score   support

    negative       0.78      0.55      0.64       316
     neutral       1.00      0.00      0.01       211
    positive       0.81      0.98      0.89      1473

    accuracy                           0.81      2000
   macro avg       0.87      0.51      0.51      2000
weighted avg       0.83      0.81      0.76      2000
```

SVM model

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score



# Sample text data and labels
texts = data['filtered_words']  # Assuming 'text' contains the cleaned, raw text
labels = data['sentiment']  # Assuming 'sentiment' is the target variable

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=10000, use_idf=T

# Apply TF-IDF to the text data
X = vectorizer.fit_transform(texts)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, ran
```

```
from sklearn.svm import SVC
# Initialize the Support Vector Classifier
svm_classifier = SVC(kernel='linear')

# Train the classifier
svm_classifier.fit(X_train, y_train)
```

```
▾ SVC
SVC(kernel='linear')
```

```
# Predict on the test set
y_pred = svm_classifier.predict(X_test)

# Evaluate the performance
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8445
              precision    recall  f1-score   support

    negative       0.70      0.71      0.71       316
     neutral       0.48      0.24      0.32       211
    positive       0.90      0.96      0.93      1473

    accuracy                           0.84      2000
   macro avg       0.69      0.64      0.65      2000
weighted avg       0.82      0.84      0.83      2000
```

```
from sklearn.model_selection import GridSearchCV

# Define parameter range
param_grid = {
    'C': [0.1, 1, 10],  # Note: Adjust these values based on your specific needs
    'gamma': [1, 0.1, 0.01],
    'kernel': ['linear', 'rbf']
}

# Create a GridSearchCV object
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
```

```
# Fitting the model for grid search
grid.fit(X_train, y_train)

# Print best parameter after tuning
print(grid.best_params_)

# Print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)

best_svm = grid.best_estimator_
final_predictions = best_svm.predict(X_test)
print("Final model accuracy: {:.2f}".format(accuracy_score(y_test, final_predicti
```

```
[CV 4/5] END .......C=1, gamma=1, kernel=linear;, score=0.854 total time=  1
[CV 5/5] END .......C=1, gamma=1, kernel=linear;, score=0.853 total time=  1
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.851 total time=  1
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.848 total time=  1
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.848 total time=  1
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.848 total time=  1
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.848 total time=  1
[CV 1/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.859 total time=  1
[CV 2/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.854 total time=  1
[CV 3/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.854 total time=  1
[CV 4/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.854 total time=  1
[CV 5/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.853 total time=  1
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.824 total time=  1
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.816 total time=  1
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.820 total time=  1
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.822 total time=  1
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.822 total time=  1
[CV 1/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.859 total time=  1
[CV 2/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.854 total time=  1
[CV 3/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.854 total time=  1
[CV 4/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.854 total time=  1
[CV 5/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.853 total time=  1
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.740 total time=  1
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.740 total time=  1
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.740 total time=  1
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.739 total time=  1
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.739 total time=  1
[CV 1/5] END ......C=10, gamma=1, kernel=linear;, score=0.837 total time=  1
[CV 2/5] END ......C=10, gamma=1, kernel=linear;, score=0.826 total time=  1
[CV 3/5] END ......C=10, gamma=1, kernel=linear;, score=0.833 total time=  1
[CV 4/5] END ......C=10, gamma=1, kernel=linear;, score=0.834 total time=  1
[CV 5/5] END ......C=10, gamma=1, kernel=linear;, score=0.825 total time=  1
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.869 total time=  1
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.854 total time=  1
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.858 total time=  2
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.855 total time=  1
```

```
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.854 total time=  2
[CV 1/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.837 total time=  1
[CV 2/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.826 total time=  1
[CV 3/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.833 total time=  1
[CV 4/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.834 total time=  1
[CV 5/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.825 total time=  1
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.858 total time=  1
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.851 total time=  1
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.854 total time=  1
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.851 total time=  1
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.854 total time=  1
[CV 1/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.837 total time=  1
[CV 2/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.826 total time=  1
[CV 3/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.833 total time=  1
[CV 4/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.834 total time=  1
[CV 5/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.825 total time=  1
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.831 total time=  1
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.825 total time=  1
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.828 total time=  1
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.829 total time=  1
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.826 total time=  1
{'C': 10, 'gamma': 1, 'kernel': 'rbf'}
SVC(C=10, gamma=1)
```

```
# This mapping assumes the model outputs string labels directly
sentiment_mapping = {'negative': 'negative', 'neutral': 'neutral', 'positive': 'p
```

```
def classify_review_svm(review, vectorizer, model):
    # Vectorize the review using the trained TF-IDF vectorizer
    review_vector = vectorizer.transform([review])

    # Predict the sentiment using the trained SVM model
    sentiment = model.predict(review_vector)

    # Check the type of output from the model and prepare mapping accordingly
    if isinstance(sentiment[0], str):
        sentiment_mapping = {'negative': 'negative', 'neutral': 'neutral', 'posit
    else:
        sentiment_mapping = {0: 'negative', 1: 'neutral', 2: 'positive'}

    # Return the mapped sentiment text
    return sentiment_mapping[sentiment[0]]

# Example usage
new_review = "The flight was delayed, but the staff were incredibly helpful."
svm_sentiment = classify_review_svm(new_review, vectorizer, svm_classifier)
print("SVM Model Sentiment:", svm_sentiment)
```

```
SVM Model Sentiment: positive
```

BERT

```
import pandas as pd
from transformers import BertTokenizer
import tensorflow as tf

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenization and encoding the dataset
def encode_reviews(data, max_length):
    input_ids = []
    attention_masks = []

    for review in data:
        encoded = tokenizer.encode_plus(
            review,
            add_special_tokens=True,
            max_length=max_length,
            truncation=True,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='tf'
        )
        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])

    return tf.concat(input_ids, axis=0), tf.concat(attention_masks, axis=0)

max_length = 128  # You can adjust this depending on the length of your reviews
X_train_ids, X_train_masks = encode_reviews(data['text'], max_length)



from sklearn.preprocessing import LabelEncoder

# Encode labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(data['sentiment'])
```

```
from transformers import TFBertForSequenceClassification
import tensorflow as tf

def build_model(learning_rate, optimizer_choice, hidden_layers, neurons):
    # Load BERT model with classification head
    model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased',

    # Input layers
    input_ids = tf.keras.layers.Input(shape=(max_length,), dtype=tf.int32, name='
    attention_masks = tf.keras.layers.Input(shape=(max_length,), dtype=tf.int32,

    # BERT layer
    outputs = model.bert(input_ids, attention_mask=attention_masks)

    # Additional hidden layers
    x = outputs[1]
    for _ in range(hidden_layers):
        x = tf.keras.layers.Dense(neurons, activation='relu')(x)

    # Output layer
    classifier = tf.keras.layers.Dense(3, activation='softmax')(x)

    # Assemble final model
    final_model = tf.keras.Model(inputs=[input_ids, attention_masks], outputs=cla

    # Optimizer selection
    if optimizer_choice == 'adam':
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    elif optimizer_choice == 'rmsprop':
        optimizer = tf.keras.optimizers.RMSprop(learning_rate=learning_rate)
    elif optimizer_choice == 'sgd':
        optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)

    # Compile the model
    final_model.compile(optimizer=optimizer, loss='sparse_categorical_crossentrop

    return final_model



import numpy as np

# Hyperparameters space definition
learning_rates = [0.001, 0.01]
optimizers = ['adam', 'rmsprop', 'sgd']
```

```
hidden_layers_options = [1, 3, 5]
neurons_options = [50, 500]

# Placeholder for storing results
results = []

# Grid search
for lr in learning_rates:
    for optimizer in optimizers:
        for hidden_layers in hidden_layers_options:
            for neurons in neurons_options:
                print(f"Training with lr={lr}, optimizer={optimizer}, hidden_layers
                model = build_model(lr, optimizer, hidden_layers, neurons)
                history = model.fit(
                    {'input_ids': X_train_ids, 'attention_mask': X_train_masks},
                    y_train,
                    validation_split=0.1,  # Using part of the training data as val
                    batch_size=16,
                    epochs=3,
                    verbose=1
                )
                val_accuracy = np.max(history.history['val_accuracy'])
                results.append((lr, optimizer, hidden_layers, neurons, val_accuracy

# Find the best configuration
best_config = max(results, key=lambda x: x[4])
print("Best configuration:", best_config)
```

```
Training with lr=0.001, optimizer=adam, hidden_layers=3, neurons=500
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/3
563/563 [==============================] - 167s 197ms/step - loss: 0.8000 -
Epoch 2/3
563/563 [==============================] - 93s 165ms/step - loss: 0.7743 - a
Epoch 3/3
563/563 [==============================] - 92s 163ms/step - loss: 0.7709 - a
Training with lr=0.001, optimizer=adam, hidden_layers=5, neurons=50
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/3
563/563 [==============================] - 169s 198ms/step - loss: 0.7816 -
Epoch 2/3
```

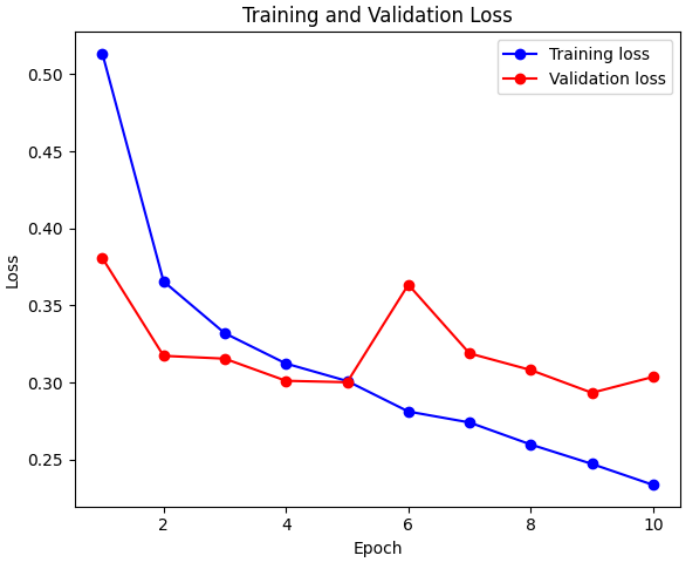
```
563/563 [==============================] - 93s 166ms/step - loss: 0.7720 - a
Epoch 3/3
563/563 [==============================] - 93s 165ms/step - loss: 0.7711 - a
Training with lr=0.001, optimizer=adam, hidden_layers=5, neurons=500
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/3
563/563 [==============================] - 168s 197ms/step - loss: 0.7915 -
Epoch 2/3
563/563 [==============================] - 94s 166ms/step - loss: 0.7742 - a
Epoch 3/3
563/563 [==============================] - 92s 164ms/step - loss: 0.7666 - a
Training with lr=0.001, optimizer=rmsprop, hidden_layers=1, neurons=50
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/3
563/563 [==============================] - 146s 176ms/step - loss: 0.7994 -
Epoch 2/3
563/563 [==============================] - 88s 157ms/step - loss: 0.7726 - a
Epoch 3/3
563/563 [==============================] - 88s 157ms/step - loss: 0.7712 - a
Training with lr=0.001, optimizer=rmsprop, hidden_layers=1, neurons=500
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/3
563/563 [==============================] - 145s 176ms/step - loss: 0.8970 -
Epoch 2/3
563/563 [==============================] - 89s 157ms/step - loss: 0.8191 - a
Epoch 3/3
563/563 [==============================] - 88s 156ms/step - loss: 0.7759 - a
Training with lr=0.001, optimizer=rmsprop, hidden_layers=3, neurons=50
All PyTorch model weights were used when initializing TFBertForSequenceClass

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification
```

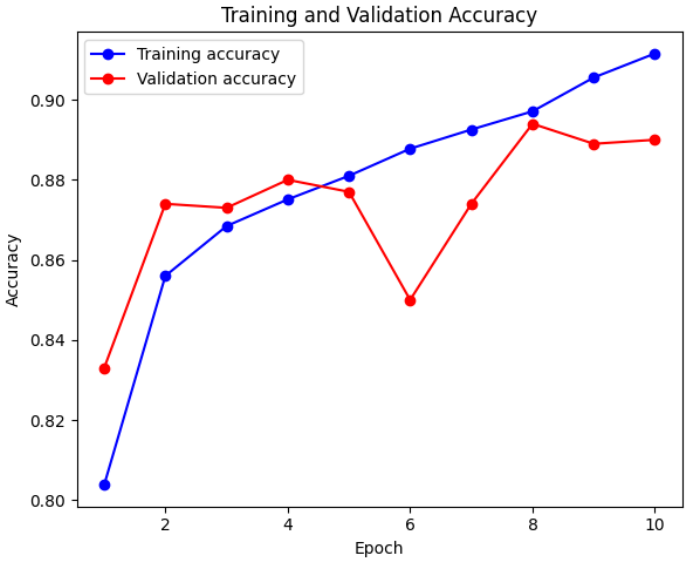
```
# Build the model with the best configuration
best_lr = 0.001
best_optimizer = 'sgd'
best_hidden_layers = 1
best_neurons = 50

model = build_model(best_lr, best_optimizer, best_hidden_layers, best_neurons)

# Train the model
history = model.fit(
    {'input_ids': X_train_ids, 'attention_mask': X_train_masks},
    y_train,
    validation_split=0.1,  # Using part of the training data as validation
    batch_size=16,
    epochs=10,  # More epochs for better training
    verbose=1
)
```

```
All PyTorch model weights were used when initializing TFBertForSequenceClassi

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification we
You should probably TRAIN this model on a down-stream task to be able to use
Epoch 1/10
563/563 [==============================] - 140s 173ms/step - loss: 0.5135 - ac
Epoch 2/10
563/563 [==============================] - 88s 157ms/step - loss: 0.3656 - acc
Epoch 3/10
563/563 [==============================] - 88s 156ms/step - loss: 0.3320 - acc
Epoch 4/10
563/563 [==============================] - 88s 157ms/step - loss: 0.3123 - acc
Epoch 5/10
563/563 [==============================] - 88s 157ms/step - loss: 0.3010 - acc
Epoch 6/10
563/563 [==============================] - 88s 157ms/step - loss: 0.2812 - acc
Epoch 7/10
563/563 [==============================] - 88s 157ms/step - loss: 0.2741 - acc
Epoch 8/10
563/563 [==============================] - 88s 157ms/step - loss: 0.2598 - acc
Epoch 9/10
563/563 [==============================] - 88s 157ms/step - loss: 0.2472 - acc
Epoch 10/10
563/563 [==============================] - 88s 156ms/step - loss: 0.2336 - acc
```

```
import matplotlib.pyplot as plt
```

```
# Function to plot loss and accuracy
def plot_loss_accuracy(history):
    epochs = range(1, len(history.history['accuracy']) + 1)

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['loss'], 'bo-', label='Training loss')
    plt.plot(epochs, history.history['val_loss'], 'ro-', label='Validation loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['accuracy'], 'bo-', label='Training accuracy
    plt.plot(epochs, history.history['val_accuracy'], 'ro-', label='Validation ac
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Plotting the graphs for the trained model
plot_loss_accuracy(history)
```


Training and Validation Loss
Training loss
Validation loss
Loss
Epoch
0.50
0.45
0.40
0.35
0.30
0.25
2
4
6
8
10


Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epoch
0.90
0.88
0.86
0.84
0.82
0.80
2
4
6
8
10

```
def predict_sentiment(sentence, model, tokenizer, label_encoder):
    # Preprocess the sentence
    encoded = tokenizer.encode_plus(
        sentence,
        add_special_tokens=True,
        max_length=128,  # Ensure this matches the training setup
        truncation=True,
        padding='max_length',
        return_attention_mask=True,
        return_tensors='tf'
    )

    # Extract input_ids and attention_mask from encoded dictionary
    input_ids = encoded['input_ids']
    attention_mask = encoded['attention_mask']

    # Make a prediction
    logits = model.predict({'input_ids': input_ids, 'attention_mask': attention_m

    # Convert logits to softmax to derive probabilities
    probabilities = tf.nn.softmax(logits, axis=-1)

    # Get the class with the highest probability
    predicted_class_id = tf.argmax(probabilities, axis=-1)

    # Decode class ID to label
    predicted_class = label_encoder.inverse_transform([predicted_class_id.numpy()

    return predicted_class, probabilities.numpy()

def display_class_probabilities(probabilities, label_encoder):
    # Display the class probabilities as percentages
    print("Class probabilities:")
    for idx, prob in enumerate(probabilities.flatten(), 1):
        label = label_encoder.inverse_transform([idx - 1])[0]
        print(f"{label}: {prob * 100:.2f}%")
```

```
# Example usage
test_sentence = "The airline is the best I have ever seen."

# Predict sentiment and get probabilities
predicted_sentiment, probabilities = predict_sentiment(test_sentence, model, toke
print(f"Predicted sentiment: {predicted_sentiment}")
display_class_probabilities(probabilities, label_encoder)
```

```
1/1 [==============================] - 0s 46ms/step
Predicted sentiment: positive
Class probabilities:
negative: 21.23%
neutral: 21.26%
positive: 57.51%
```