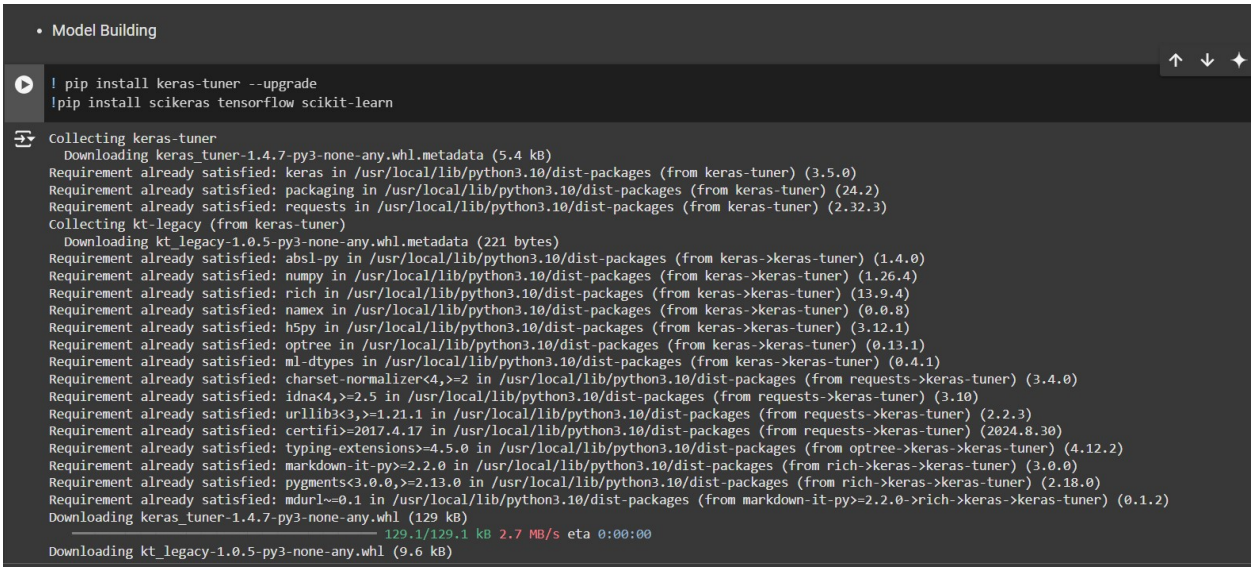
Model Development Phase Template

Date	24 November 2024
Team ID	SWTID1727420425
Project Title	Analysis amazon cell phone review with nlp technique
Maximum Marks	10 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code for analyzing Amazon cell phone reviews using NLP techniques will be showcased in the future through a screenshot. The model validation and evaluation report will provide a summary of the performance, including training and validation metrics for multiple models. These performance metrics will be presented through respective screenshots, highlighting key indicators such as accuracy, precision, recall, and F1 score to assess the effectiveness of the NLP models in classifying and analyzing customer sentiments from the reviews.

Initial Model Training Code (5 marks):



```
[ ]  import keras_tuner as kt
     import tensorflow as tf
     from keras.models import Sequential
     from keras.layers import Dense, Conv1D, LSTM, Bidirectional, Embedding, MaxPooling1D, Dropout, Flatten
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelEncoder
     from sklearn.metrics import classification_report
     from scikeras.wrappers import KerasClassifier
     from sklearn.model_selection import GridSearchCV
     from tensorflow.keras.optimizers import Adam
```

- Data Pre processing

```
[ ]  # Selecting features and target
     features = ['rating_x', 'verified', 'title_x', 'body', 'brand', 'price', 'originalPrice']
     target = 'helpfulVotes'
```

```
     # Clean text function
     def clean_text(text):
         text = text.lower()
         text = re.sub(r'\W', ' ', text)  # Remove special characters
         text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)  # Remove single characters
         text = re.sub(r'\s+', ' ', text)  # Remove multiple spaces
         return text
```

+ Code + Text

```
[ ]  # Apply cleaning to text features
     merged_data['body'] = merged_data['body'].fillna('').apply(clean_text)
     merged_data['title_x'] = merged_data['title_x'].fillna('').apply(clean_text)
```

```
[ ]  # Filter data and drop missing rows
     filtered_data = merged_data[features + [target]].dropna()
```

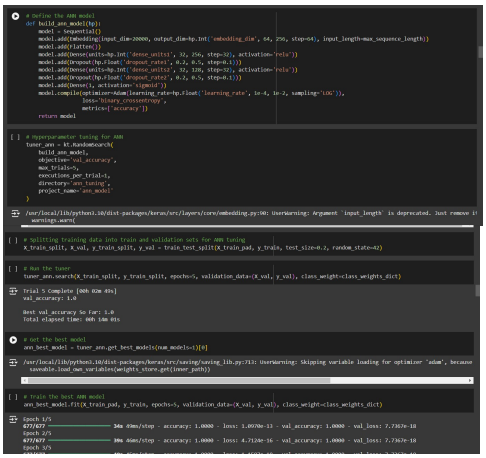
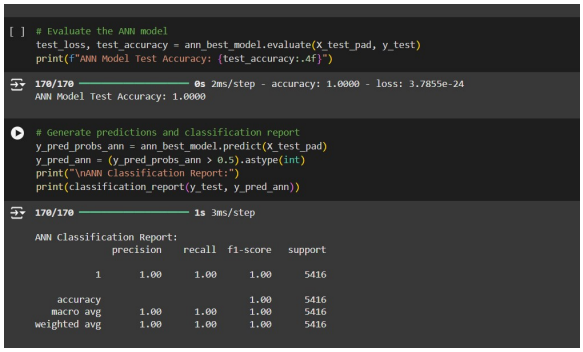
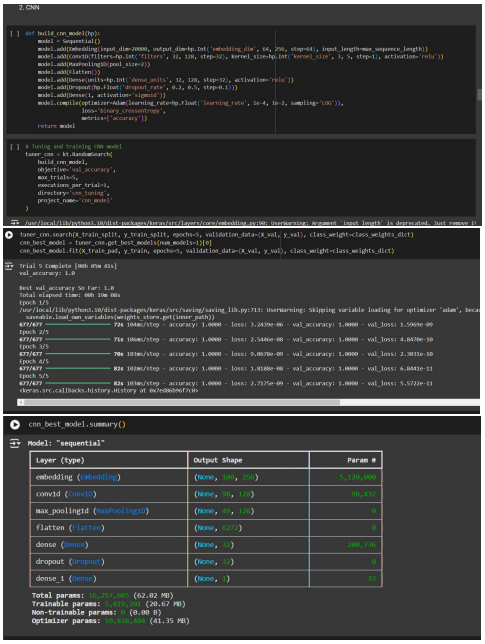
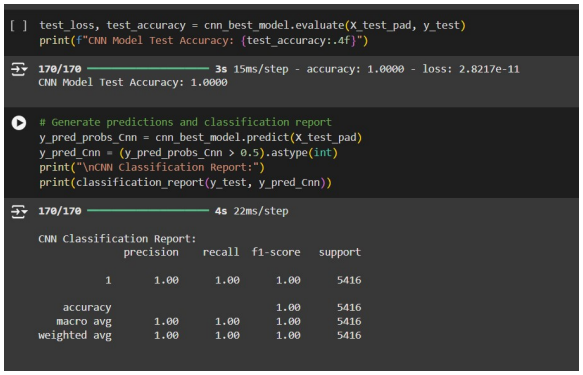
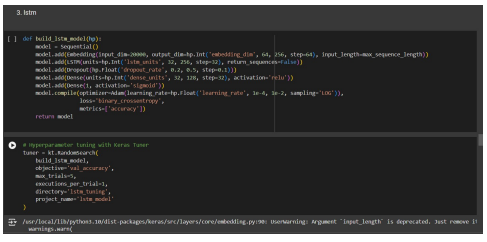
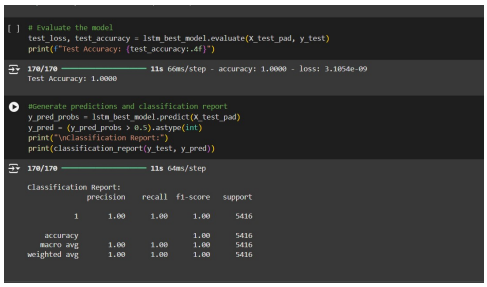
```
[ ]  max_sequence_length = 100
     X_train_pad = pad_sequences(X_train_seq, maxlen=max_sequence_length)
     X_test_pad = pad_sequences(X_test_seq, maxlen=max_sequence_length)
```

```
[ ]  # Binarizing the target
     y_train = y_train.values
     y_test = y_test.values
```

```
     # Define class weights to handle imbalance
     class_weights = class_weight.compute_class_weight(
         class_weight='balanced',
         classes=np.unique(y_train),
         y=y_train
     )
     class_weights_dict = {i: class_weights[i] for i in range(len(class_weights))}
```

Model Validation and Evaluation Report (5 marks):

Model	Model code	Model Classification

ANN		
CNN		
LSTM		

ANN

```
# Define the ANN model
def build_ann_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=hp.Int('embedding_dim', 64, 256, step=64), input_length=max_sequence_length))
    model.add(Flatten())
    model.add(Dense(units=hp.Int('dense_units1', 32, 256, step=32), activation='relu'))
    model.add(Dropout(hp.Float('dropout_rate1', 0.2, 0.5, step=0.1)))
    model.add(Dense(units=hp.Int('dense_units2', 32, 128, step=32), activation='relu'))
    model.add(Dropout(hp.Float('dropout_rate2', 0.2, 0.5, step=0.1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Hyperparameter tuning for ANN
tuner_ann = kt.RandomSearch(
    build_ann_model,
    objective='val_accuracy',
    max_trials=5,
    executions_per_trial=1,
    directory='ann_tuning',
    project_name='ann_model'
)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it
  warnings.warn(

# Splitting training data into train and validation sets for ANN tuning
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train_pad, y_train, test_size=0.2, random_state=42)

# Run the tuner
tuner_ann.search(X_train_split, y_train_split, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Trial 5 Complete [00h 02m 45s]
val_accuracy: 1.0

Best val_accuracy So Far: 1.0
Total elapsed time: 00h 13m 43s

# Get the best model
ann_best_model = tuner_ann.get_best_models(num_models=1)[0]
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713: UserWarning: Skipping variable loading for optimizer 'adam', because
  saveable.load_own_variables(weights_store.get(inner_path))

# Train the best ANN model
ann_best_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Epoch 1/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 34s 69ms/step - accuracy: 1.0000 - loss: 1.0970e-13 - val_accuracy: 1.0000 - val_loss: 7.7367e-18
Epoch 2/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 30s 46ms/step - accuracy: 1.0000 - loss: 4.7124e-16 - val_accuracy: 1.0000 - val_loss: 7.7367e-18
Epoch 3/5
```

```
[ ] # Evaluate the ANN model
    test_loss, test_accuracy = ann_best_model.evaluate(X_test_pad, y_test)
    print(f"ANN Model Test Accuracy: {test_accuracy:.4f}")

170/170 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 1.0000 - loss: 3.7855e-24
ANN Model Test Accuracy: 1.0000

# Generate predictions and classification report
y_pred_probs_ann = ann_best_model.predict(X_test_pad)
y_pred_ann = (y_pred_probs_ann > 0.5).astype(int)
print("\nANN Classification Report:")
print(classification_report(y_test, y_pred_ann))

170/170 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step

ANN Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      5416

    accuracy                           1.00      5416
   macro avg       1.00      1.00      1.00      5416
weighted avg       1.00      1.00      1.00      5416
```

CNN

```
2. CNN

def build_cnn_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=hp.Int('embedding_dim', 64, 256, step=64), input_length=max_sequence_length))
    model.add(Conv1D(filters=hp.Int('filters', 32, 128, step=32), kernel_size=hp.Int('kernel_size', 3, 5, step=1), activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(units=hp.Int('dense_units', 32, 128, step=32), activation='relu'))
    model.add(Dropout(hp.Float('dropout_rate', 0.2, 0.5, step=0.1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Tuning and training CNN model
tuner_cnn = kt.RandomSearch(
    build_cnn_model,
    objective='val_accuracy',
    max_trials=5,
    executions_per_trial=1,
    directory='cnn_tuning',
    project_name='cnn_model'
)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it

tuner_cnn.search(X_train_split, y_train_split, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)
cnn_best_model = tuner_cnn.get_best_models(num_models=1)[0]
cnn_best_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val, y_val), class_weight=class_weights_dict)

Trial 5 Complete [00h 05m 41s]
val_accuracy: 1.0

Best val_accuracy So Far: 1.0
Total elapsed time: 00h 19m 08s
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713: UserWarning: Skipping variable loading for optimizer 'adam', beca
  saveable.load_own_variables(weights_store.get(inner_path))
477/477 ━━━━━━━━━━━━━━━━━━━━ 72s 104ms/step - accuracy: 1.0000 - loss: 3.2439e-06 - val_accuracy: 1.0000 - val_loss: 1.5969e-09
Epoch 2/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 71s 106ms/step - accuracy: 1.0000 - loss: 2.5446e-08 - val_accuracy: 1.0000 - val_loss: 4.8670e-10
Epoch 3/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 70s 103ms/step - accuracy: 1.0000 - loss: 9.0670e-09 - val_accuracy: 1.0000 - val_loss: 2.3031e-10
Epoch 4/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 82s 102ms/step - accuracy: 1.0000 - loss: 3.8188e-08 - val_accuracy: 1.0000 - val_loss: 6.8441e-11
Epoch 5/5
477/477 ━━━━━━━━━━━━━━━━━━━━ 82s 103ms/step - accuracy: 1.0000 - loss: 2.7175e-09 - val_accuracy: 1.0000 - val_loss: 5.5722e-11
<keras.src.callbacks.history.History at 0x7e8b0b9cf7c0>

cnn_best_model.summary()

Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 200, 256)	5,120,000
conv1d (Conv1D)	(None, 98, 128)	98,432
max_pooling1d (MaxPooling1D)	(None, 49, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 96)	602,208
dropout (Dropout)	(None, 96)	0
dense_1 (Dense)	(None, 1)	97

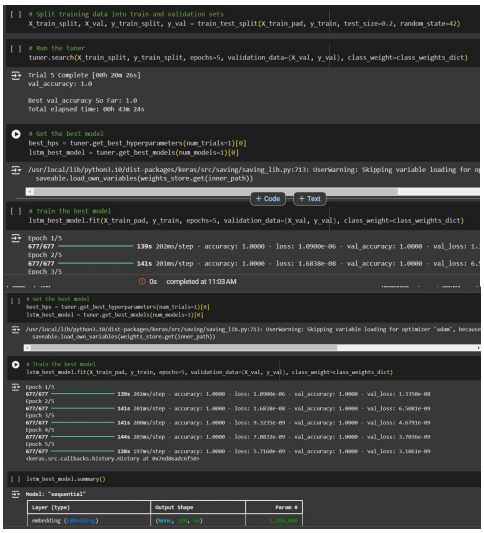
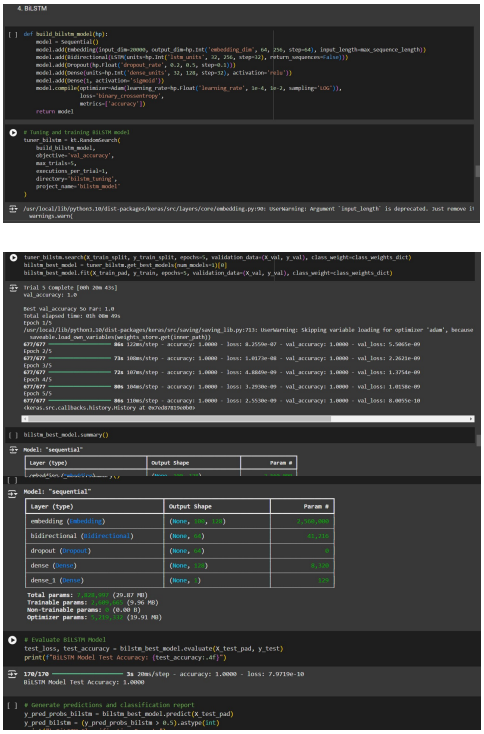
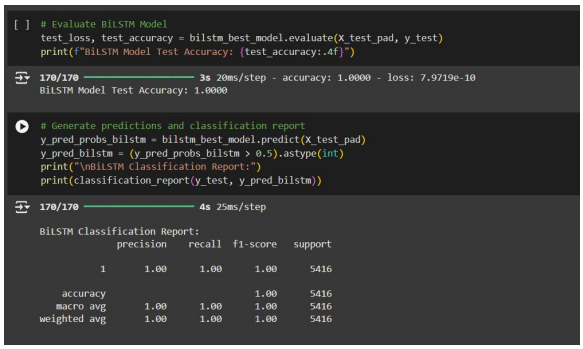
```
Total params: 16,157,485 (61.64 MB)
Trainable params: 5,719,193 (21.82 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 10,438,292 (39.82 MB)
```

```
[ ] test_loss, test_accuracy = cnn_best_model.evaluate(X_test_pad, y_test)
    print(f"CNN Model Test Accuracy: {test_accuracy:.4f}")

170/170 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 1.0000 - loss: 2.8217e-11
CNN Model Test Accuracy: 1.0000

# Generate predictions and classification report
y_pred_probs_cnn = cnn_best_model.predict(X_test_pad)
y_pred_cnn = (y_pred_probs_cnn > 0.5).astype(int)
print("\nCNN Classification Report:")
print(classification_report(y_test, y_pred_cnn))

170/170 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step

CNN Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      5416

    accuracy                           1.00      5416
   macro avg       1.00      1.00      1.00      5416
weighted avg       1.00      1.00      1.00      5416
```

LSTM

```
3. lstm

def build_lstm_model(hp):
    model = Sequential()
    model.add(Embedding(input_dim=20000, output_dim=hp.Int('embedding_dim', 64, 256, step=64), input_length=max_sequence_length))
    model.add(LSTM(units=hp.Int('lstm_units', 32, 256, step=32), return_sequences=False))
    model.add(Dropout(hp.Float('dropout_rate', 0.2, 0.5, step=0.1)))
    model.add(Dense(units=hp.Int('dense_units', 32, 128, step=32), activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate', 1e-4, 1e-2, sampling='LOG')),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Hyperparameter tuning with Keras Tuner
tuner = kt.RandomSearch(
    build_lstm_model,
    objective='val_accuracy',
    max_trials=5,
    executions_per_trial=1,
    directory='lstm_tuning',
    project_name='lstm_model'
)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it
  warnings.warn(
```

```
[ ] # Evaluate the model
    test_loss, test_accuracy = lstm_best_model.evaluate(X_test_pad, y_test)
    print(f"Test Accuracy: {test_accuracy:.4f}")

170/170 ━━━━━━━━━━━━━━━━━━━━ 11s 66ms/step - accuracy: 1.0000 - loss: 3.1054e-09
Test Accuracy: 1.0000

#Generate predictions and classification report
y_pred_probs = lstm_best_model.predict(X_test_pad)
y_pred = (y_pred_probs > 0.5).astype(int)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

170/170 ━━━━━━━━━━━━━━━━━━━━ 11s 64ms/step

Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      5416

    accuracy                           1.00      5416
   macro avg       1.00      1.00      1.00      5416
weighted avg       1.00      1.00      1.00      5416
```

<table>
<tr>
<td></td>
<td> 
</td>
<td></td>
</tr>
<tr>
<td>BiLSTM</td>
<td> 
</td>
<td> 
</td>
</tr>
</table>