CONSTRUCTION OF KNOWLEDGE GRAPHS AND LONG SHORT-TERM MEMORY MODELS USING MOVIE REVIEWS

Steve Desilets

MSDS 453: Natural Language Processing

Section 56 – Summer 2023

August 6, 2023

1. Introduction and Problem Statement

Some of the most cutting-edge natural language processing (NLP) algorithms produce insights not only by applying deep learning techniques to leverage information stored within a corpus of text, but also by drawing information from ontologies that define structured relationships between named entities in the real world. While data scientists can apply these advanced NLP techniques to a broad array of domains, in this paper, we focus on applying ontology creation and long short-term memory (LSTM) model development techniques to a corpus of movie reviews. The underlying movie review data (which contains 10 reviews for each of 19 movies of interest) leveraged in this study consists of 190 rows and 9 columns, as described in Table 1 below.

Table 1: Description of the Structure of the Movie Review Dataset

Variable Name	Python Data Type	Description
DSI_Title	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Submission File Name	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Student Name	Object	Initials of student who added review to the corpus
Genre of Movie	Object	Movie genre (action, comedy, horror, or sci-fi)
Review Type (pos or neg)	Object	Type of review (positive or negative)
Movie Title	Object	Movie title
Text	Object	First 500 words of the original movie review's text
Descriptor	Object	One-word summary of movie review (including genre, movie name, review type, and Doc ID)
Doc_ID	Integer	Unique ID for movie review document

In this paper, we first examine how to most effectively extract named entities and resolve entity co-references in order to generate useful knowledge graphs capable of summarizing information about the real world. Then, we turn our attention to identifying which LSTM model architectures are most effective at classifying our 190 movie reviews by movie genre and movie review sentiment. Through these analyses, we hope to learn how to most effectively construct NLP pipelines focused on empowering movie lovers via ontology construction and document classification applications.

2. Research Design and Modeling Methods

Our exploration of NLP pipelines throughout this study was divided into three sections: 1) Knowledge Graph Experiments, 2) Movie Genre Classification LSTM Model Experiments, and 3) Movie Review Sentiment Classification LSTM Model Experiments. We began the first phase of this study by importing the corpus of movie reviews into a Jupyter Notebook in Python and conducting exploratory data analysis. Then, we leverage the spaCy library to extract triplets of sources, edges, and targets from each sentence in the corpus of reviews of our primary movie of interest: "Sisters" (Moore 2015). After constructing an initial knowledge graph that contains all the triplets from the "Sisters" reviews, we then filter this knowledge graph to just display important subsets pertaining to important named entities like "sisters," "Tina Fey," and "Amy Poehler." We then turn our attention toward conducting entity co-reference resolution to determine whether consolidating named entities will improve the clarity of our knowledge graphs. After inspecting the dataframe of triplets, we consolidate named entities (pertaining to Tina Fey, Amy Poehler, sisters, Paula Pell, Jason Moore, Saturday Night Live, Maya Rudolph, Maura Ellis, Kate Ellis, Madison Davenport, and Dianne Wiest) and reconstruct each of the five original knowledge graphs to determine whether they're able to display named entity relationships more clearly than their original counterparts. The knowledge graphs resulting from these experiments are visualized in Appendix A, and the corresponding Python code associated with these experiments is provided in section 1 of Appendix D.

In the second section of our study, we turn our attention toward building LSTM models capable of predicting movie genres using movie review text. For this section, we utilize our entire corpus of 190 movie reviews and aim to categorize each review as corresponding to either the action, comedy, horror, or science fiction movie genre. We begin by performing data wrangling (including stop words removal and lemmatization) on the movie text; dividing the dataset into training, testing and validation subsets; and encoding the tokens in the reviews. We then constructed four LSTM models as described in the table below and assessed each model's performance using performance metrics, line graphs plotting training and validation accuracy and loss across epochs during model fitting, and confusion matrices. The outputs

resulting from these genre classification experiments are presented in Appendix B, and the full Python code leveraged to run these experiments is available in section 2.1 of Appendix D.

Table 2: Description of the LSTM Models Used for Movie Genre and Review Sentiment Classification

LSTM Model	Description of LSTM Model Layers
Model 1	1. Input layer of 64-dimensional text embeddings 2. Bidirectional LSTM layer with 64 neurons and 30% dropout 3. Bidirectional LSTM layer with 32 neurons and 30% dropout 4. Dense layer with 64 neurons and rectified linear unit activation function 5. Dense output layer using SoftMax to sort data into target variable's classes
Model 2	1. Input layer of 128-dimensional text embeddings 2. Bidirectional LSTM layer with 128 neurons and 30% dropout 3. Bidirectional LSTM layer with 64 neurons and 30% dropout 4. Dense layer with 128 neurons and rectified linear unit activation function 5. Dense output layer using SoftMax to sort data into target variable's classes
Model 3	1. Input layer of 64-dimensional text embeddings 2. Bidirectional LSTM layer with 64 neurons and 30% dropout 3. Bidirectional LSTM layer with 32 neurons and 30% dropout 4. Dense layer with 64 neurons and rectified linear unit activation function 5. Dense layer with 32 neurons and rectified linear unit activation function 6. Dense output layer using SoftMax to sort data into target variable's classes
Model 4	1. Input layer of 128-dimensional text embeddings 2. Bidirectional LSTM layer with 128 neurons and 30% dropout 3. Bidirectional LSTM layer with 64 neurons and 30% dropout 4. Dense layer with 128 neurons and rectified linear unit activation function 5. Dense layer with 64 neurons and rectified linear unit activation function 6. Dense output layer using SoftMax to sort data into target variable's classes

In our final section of the study, we construct sentiment classification LSTM models in a way that is very similar to the methodology leveraged in the previous section. Again, we leverage the entire corpus of movie reviews, but this time, the output variable we aim to predict – movie review sentiment – only has two classes: positive or negative. Like in the previous section, we begin by performing data wrangling (including stop words removal and lemmatization); splitting the dataset into training,

validation, and testing datasets; and encoding the movie review text. We then construct four LSTM models that leverage the same neural network architecture as the LSTM models previously described in Table 2. After fitting each of these four models, we assess their performance by constructing confusion matrices and receiving operating characteristic (ROC) curves; by plotting their accuracy and loss across epochs during model fitting; and by printing relevant performance metrics like precision, recall, accuracy, and area under the curve (AUC). The outputs associated with these sentiment classification experiments are provided in Appendix C, and the full Python code associated with these experiments is available in section 2.2 of Appendix D.

3. Results

The results from the ontology construction, genre classification, and sentiment classification analyses for this study are displayed in Appendices A – D. Appendix A displays the ontologies constructed using the "Sisters" movies reviews both before and after implementing entity co-reference resolution procedures, so that the readers can clearly assess the impacts of these exercises in side-by-side visualizations. Appendix B presents a suite of outputs from each of the four genre classification experiments, including line graphs that display the training and validation accuracy and loss for each epoch during model fitting, confusion matrices for each LSTM model, and a data table summarizing the accuracy of the four models when applied to the testing dataset. Appendix C presents similar outputs for each of the four sentiment classification LSTM models, including graphs that plot the training and validation accuracy and loss for each epoch during LSTM model creation, confusion matrices and receiving operating characteristic (ROC) curves for each model, and a table summarizing the accuracy, precision, recall, and area under the curve (AUC) for each of the four sentiment classification LSTM models. To promote transparency and reproducibility of our results, Appendix D contains the Python code leveraged to conduct our full analysis as well.

4. Analysis and Interpretation

Studying the knowledge graphs depicted in Appendix A illuminates the criticality of entity co-reference resolution for producing knowledge graphs that effectively summarize information regarding

real-world relationships. To illustrate this point, we can look at the two side-by-side knowledge graphs that display relationships containing information about Amy Poehler before and after conducting entity co-reference resolution. If a computer were to examine the six disjointed relationships displayed in the first Amy Poehler Ontology, the computer would fail to realize that the entities "poehler," "simply amy," "enough poehler," "dead sarah poehler," "ensuing poehler," and "amy poehler" all referred to the same person. After we conduct entity co-reference resolution, the machine is able to construct an ontology that clearly connects all six of these relationships via one, unified node for Amy Poehler. Entity co-reference resolution enables us to find very similar results for the Tina Fey ontologies, and this process actually reveals nine additional connections to the "Sisters" movie entity that were not initially identified in the original ontology for the movie title. While these benefits may not be immediately visible on the macro-level when examining the full (quite crowded) ontologies representing the entire corpus of "Sisters" reviews, the benefits of entity co-reference resolution are perhaps most profoundly actualized in the ontologies that contain nodes for either Amy Poehler, Tina Fey, or Sisters. Prior to consolidating named entities, the ontology contains 20 clusters of relationships that are disjointed from one another, but after entity co-reference resolution, these relationships are depicted via just two (overlapping) clusters! Thus, the entity co-reference resolution process empowered our algorithm to draw many important connections that would be necessary for a NLP pipeline focused on drawing insights from the real-world knowledge summarized in a concise knowledge graph.

Examination of the results summarized in Appendix B reveals that LSTM models can perform reasonably well at predicting movie genres using movie review text. In fact, the top performing model, Model 2, achieved an accuracy of 83.3% when applied to the testing dataset, which represents a lift of 3.3 over the accuracy of completely random genre classification model, which would have an accuracy of 25%. The results in Appendix B also suggest that LSTM neural network architecture plays a significant role in model accuracy, though. In fact, while both five-layer models achieved testing accuracies of 72% or higher, the two six-layer models only managed to attain testing accuracies of 55.6%. While the significance of LSTM model architecture is one important finding from these experiments, another key

message is that these genre classification LSTM models actually slightly underperformed compared to other genre classification machine learning models constructed during Assignment 2. In fact, six of the eight support vector machine classification, naïve Bayes classifier, and random forest classification models from assignment 2 achieved predictive accuracies for movie genres of 100%. This finding suggests that even though LSTM models may perform reasonably well at movie genre classification, they may not perform as well as other algorithms do.

While the findings regarding genre classification LSTM models were somewhat encouraging, the results displayed in Appendix C suggest that movie review sentiment classification LSTM models perform much less effectively. To be specific, even though the output variable (sentiment) contained two classes, only one of our four sentiment classification models managed to achieve an accuracy above 50% when applied to the testing dataset. In fact, the best model only achieved an accuracy, area under the curve, precision, and recall of 55.6%, 0.6, 0.5, and 0.75, respectively, which are not promising figures. While the concave up ROC curves and bleak confusion matrices in this appendix echo the message that these models aren't performing well, the line graphs plotting training and validation accuracy by epoch during model fitting suggest a possible root cause. The training accuracies achieved in the final epochs for these model reach as high as 1.0, while the validation accuracies for these models are notably lower, which suggests that these LSTM models could benefit from further regularization to prevent over-fitting to the training data. That said, examination of the Assignment 2 sentiment classification model results points to another possible root cause: that the text data may not be useful for predicting movie review sentiment. In fact, all eight of the sentiment analysis classification models from assignment 2 - which leveraged random forest, naïve bayes, and support vector machine classification methods – performed so poorly that the highest testing accuracy achieved had been just 52%. This finding means that the best sentiment classification LSTM model actually outperformed other machine learning methods. Still, the most prominent finding to emerge from the comparison of the LSTM sentiment classification model results with the assignment 2 sentiment classification model results is that the text in our corpus may not be particularly strongly predictive of movie review sentiment.

5. Conclusions

The analyses conducted throughout this study have helped us understand how to maximize the utility of ontologies and LSTM classification models constructed from movie reviews. The ontology experiments underscored the criticality of data scientists conducting thorough entity co-reference resolution processes when conducting knowledge graphs so that computers are capable of drawing important connections between named entities. The presence of these clear connections is particularly important for NLP algorithms that depend upon the knowledge stored in ontologies to return accurate information about real world relationships. While the movie genre classification LSTM models performed decently well in this study, our experience building genre classification machine learning models informs us that LSTM models may underperform at predicting movie genres compared to other machine learning methods. In contrast, the movie review sentiment classification LSTM models actually outperformed previously constructed sentiment classification models that leveraged other machine learning methods. Still, these sentiment classification LSTM models did not perform well overall, which suggests that further regularization or a much larger sample size (to compensate for the fact that the underlying data may not be particularly useful for predicting sentiments) might be needed to improve the performance of the sentiment classification LSTM models. This paper's findings regarding best practices for constructing ontologies and for developing LSTM classification models could be highly beneficial to data scientists interested in creating optimal NLP pipelines with movie review data.

6. Directions for Future Work

While this study resulted in many useful insights that could help data scientists better understand how to optimally build knowledge graphs and LSTM models that provide automated insights into named entity relationships, movie genres, and sentiments conveyed in movie reviews, there certainly exist exciting opportunities for further research related to these models.

One primary example of areas for further research would be for data scientists to conduct more extensive hyperparameter tuning for the genre and sentiment classification LSTM models. In this study, we primarily experimented with various neural network architectures (changing the number of layers and

nodes), but many more hyperparameters can influence model performance, including the L1 and L2 regularization parameters, the dropout rate, the activation functions utilized, batch sizes, and the number of epochs. By tuning these hyperparameters to the values that result in the optimal accuracy, researchers could potentially further improve the sentiment and genre classification LSTM models.

Another way in which researchers could build upon the findings from this paper would be by conducting even more thorough entity co-reference resolution analyses for the knowledge graphs than we completed. For example, future researchers could read through every row and identify how to translate specific instances of pronouns like “she”, “he”, and “they” into named entities like “Amy Poehler”, “Ike Barinholtz”, and “sisters”, so that the resulting knowledge graphs are even more consolidated and informative. This more thorough named entity disambiguation process, along with the LSTM model hyperparameter tuning process described above, could empower data scientists to more impactfully leverage information from movie reviews than even the existing analyses in this paper highlighted.

References

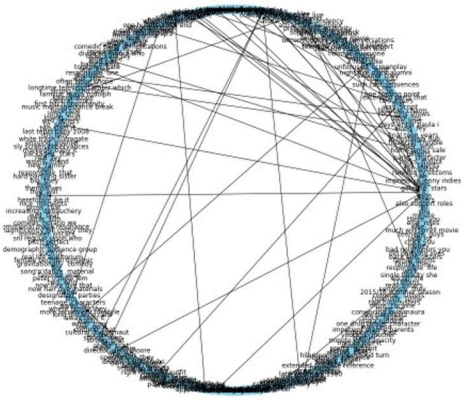
Moore, Jason. 2015. *Sisters*. United States: Universal Pictures.

Appendix A – Knowledge Graph Outputs

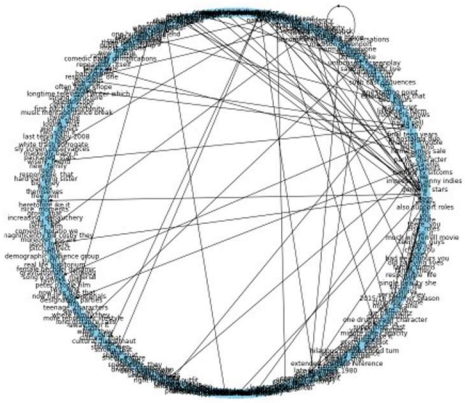
The images below visualize the full knowledge graph, as well as important subsets of the knowledge graph, generated from the ontology construction pipelines leveraging the corpus of movie reviews. While the full knowledge graph displays the source – target relationships for the entire corpus of "Sisters" review sentences, the subsets include only relationships related to either Tina Fey, Amy Poehler, "Sisters," or any of the above three terms. We created each of these five types of knowledge graphs before and after conducting entity co-reference resolution and present the corresponding knowledge graphs generated before and after that process side-by-side. Larger versions of each of the graphics presented in Appendix A are available in section 1.4 of Appendix D.

Full Knowledge Graph

Before Entity Co-Reference Resolution

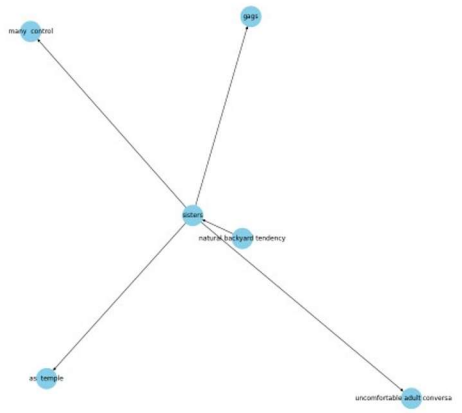


After Entity Co-Reference Resolution



Knowledge Graph of Relationships Related to “Sisters”

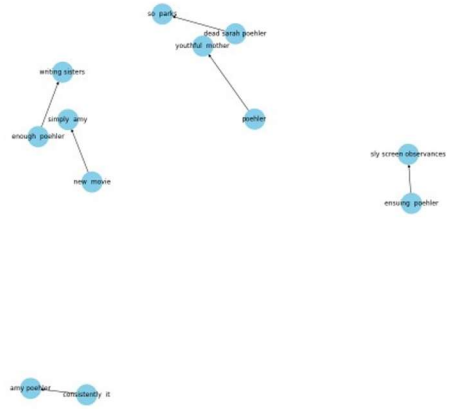
Before Entity Co-Reference Resolution



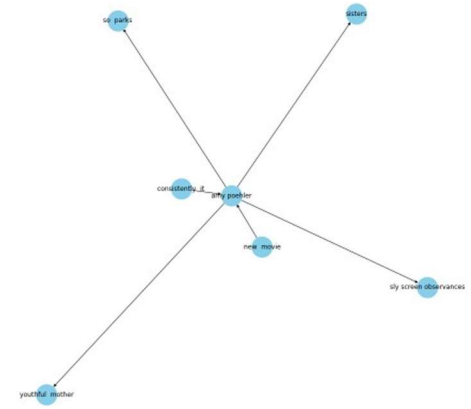
After Entity Co-Reference Resolution

Knowledge Graph of Relationships Related to Amy Poehler

Before Entity Co-Reference Resolution

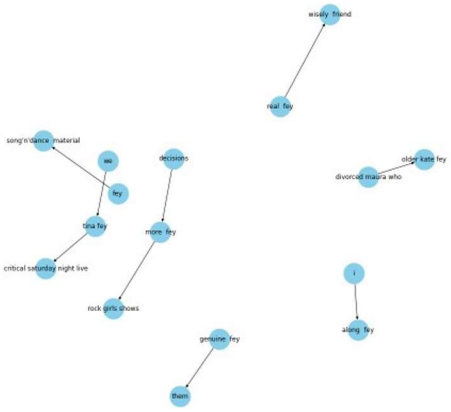


After Entity Co-Reference Resolution

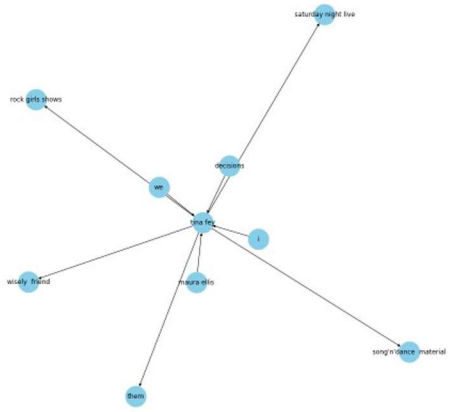


Knowledge Graph of Relationships Related to Tina Fey

Before Entity Co-Reference Resolution

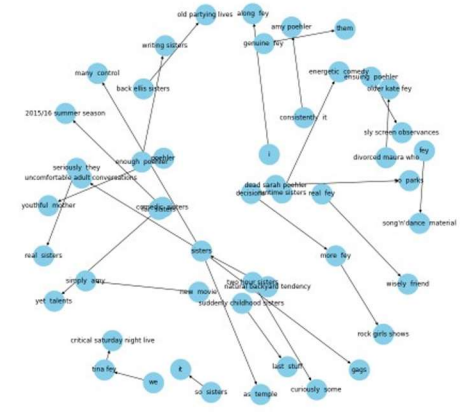


After Entity Co-Reference Resolution

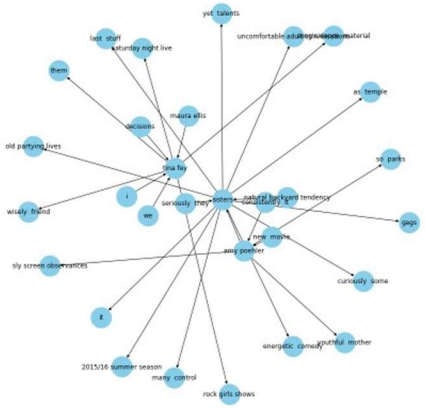


Knowledge Graph of Relationships Related to Amy Poehler, Tina Fey, or “Sisters”

Before Entity Co-Reference Resolution



After Entity Co-Reference Resolution

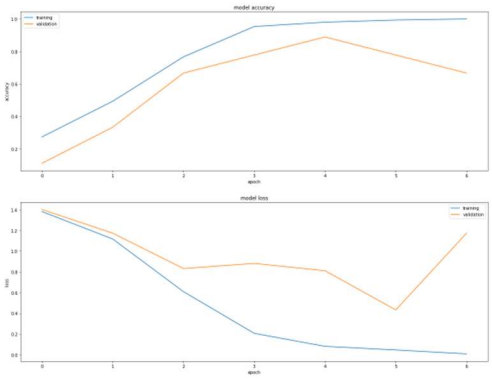


Appendix B – Genre Classification LSTM Model Outputs

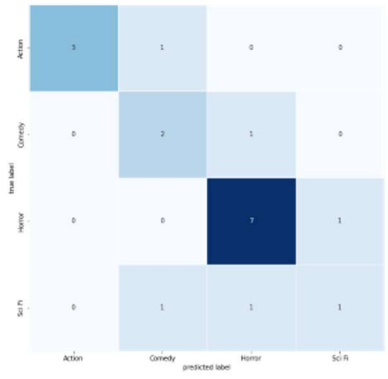
The images below display some of the most important outputs from the genre classification LSTM model experiments conducted upon the corpus of movie reviews. Specifically, the outputs include line graphs plotting the training and validation accuracy and loss by epoch during model creation, as well as the accuracy and the confusion matrix resulting from application of the genre classification LSTM models to the testing dataset. Larger versions of each of the visualizations presented in Appendix B are available in section 2.1 of Appendix D.

Genre Classification LSTM Model 1Results

Training and Validation Accuracy and Loss by Epoch

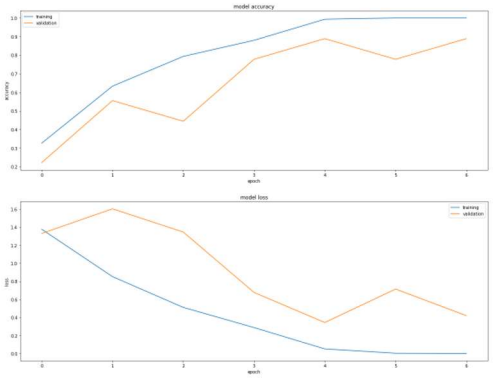


Testing Confusion Matrix

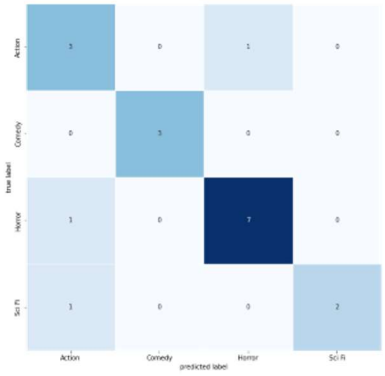


Genre Classification LSTM Model 2 Results

Training and Validation Accuracy and Loss by Epoch

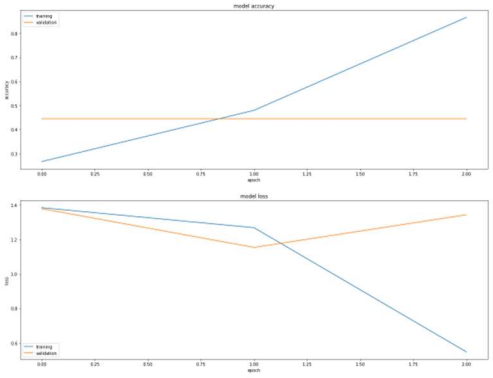


Testing Confusion Matrix

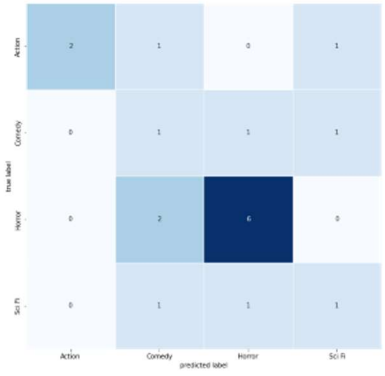


Genre Classification LSTM Model 3 Results

Training and Validation Accuracy and Loss by Epoch

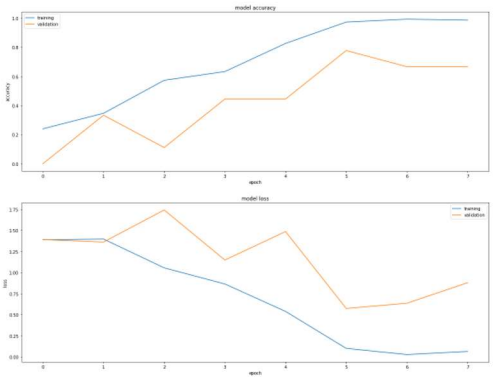


Testing Confusion Matrix

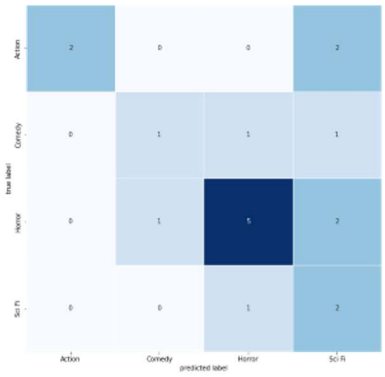


Genre Classification LSTM Model 4 Results

Training and Validation Accuracy and Loss by Epoch



Testing Confusion Matrix



Genre Classification LSTM Models – Testing Performance Metrics

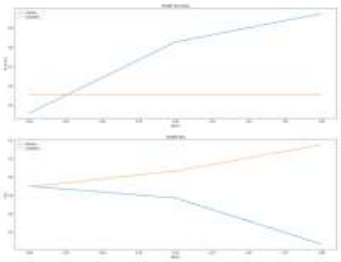
Testing Performance Metric	LSTM Model			
	Model 1	Model 2	Model 3	Model 4
Accuracy	72.2%	83.3%	55.6%	55.6%

Appendix C – Sentiment Analysis LSTM Model Outputs

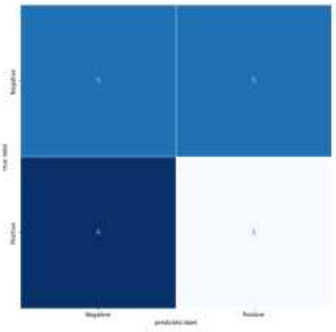
The images below display some of the most important outputs from the sentiment analysis / classification experiments conducted using the corpus of movie reviews. Specifically, the outputs include line graphs plotting the training and validation accuracy and loss by epoch during model creation; the confusion matrix and receiving operating characteristic (ROC) curves resulting from application of the genre classification LSTM models to the testing dataset; and a summary of the accuracy, precision, recall, and area under the curve (AUC) resulting from application of the genre classification LSTM models to the testing dataset. Larger versions of each of the graphics presented in Appendix C are available in section 2.2 of Appendix D.

Sentiment Analysis / Classification – LSTM Model 2 Results

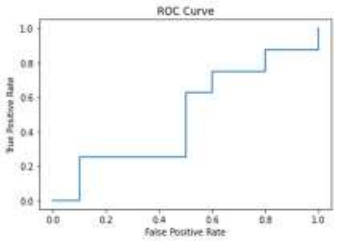
Training and Validation Accuracy and Loss By Epoch



Testing Confusion Matrix

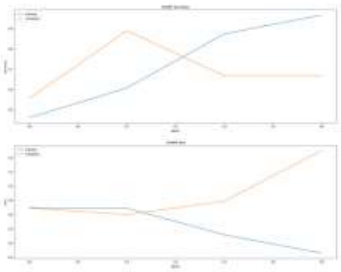


Testing ROC Curve

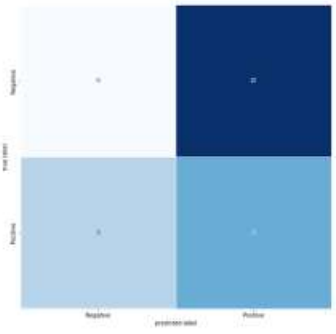


Sentiment Analysis / Classification – LSTM Model 3 Results

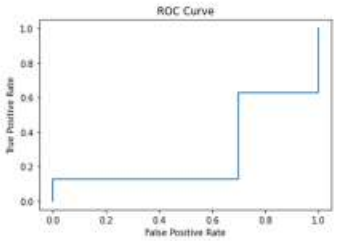
Training and Validation Accuracy and Loss By Epoch



Testing Confusion Matrix

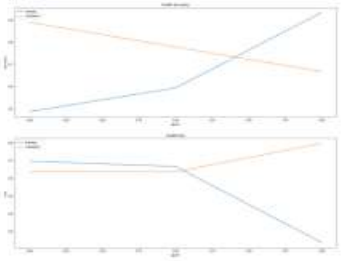


Testing ROC Curve

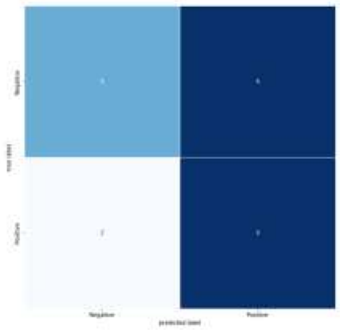


Sentiment Analysis / Classification – LSTM Model 4 Results

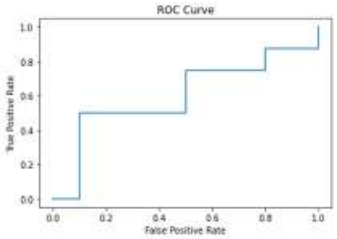
Training and Validation Accuracy and Loss By Epoch



Testing Confusion Matrix



Testing ROC Curve



Sentiment Analysis / Classification LSTM Models – Testing Performance Metrics

Testing Performance Metric	LSTM Model			
	Model 1	Model 2	Model 3	Model 4
Accuracy	33.3%	38.9%	27.8%	55.6%
Area Under the Curve (AUC)	0.312	0.487	0.275	0.6
Precision	0	0.286	0.333	0.5
Recall	0	0.25	0.625	0.75

Appendix D - Knowledge Graph and Deep Learning Python Code

1) Knowledge Graphs

Let's leverage the reviews of the movie "Sisters" in our corpus to generate knowledge graphs that can visualize relationships between named entities in the text.

1.1) Loading the Corpus

First, let's load the corpus into Python.

In [1]:
```
# !pip install -U pip setuptools wheel
# !pip install -U spacy
```

In [2]:
```
 conda install -c conda-forge g2p-en
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
# from g2p_en import G2p
```

In [4]:
```
# !pip install tensorflow_datasets
```

In [5]:
```
import pandas as pd
import numpy as np
import re
import string
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from collections import Counter
from dataclasses import dataclass
from timeit import default_timer as timer

from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize

import gensim
from gensim.models import Word2Vec

import spacy
from spacy import displacy

from spacy.matcher import Matcher
from spacy.tokens import Span

import networkx as nx
import matplotlib.pyplot as plt
from tqdm import tqdm
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, roc_auc_score, roc_curve, R

from IPython.display import display, HTML

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, f1_score

import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tfds
tf.random.set_seed(2022)
import tensorflow.keras.backend as k

from typing import List, Callable, Dict, Tuple, Set
```

```
C:\Users\steve\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected versio
n 1.25.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [6]:
```
# Requires Restart of Runtime after Installation

!python -m spacy download en_core_web_lg -q
```

```
[+] Download and installation successful
C:\Users\steve\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected versio
n 1.25.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
WARNING: Ignoring invalid distribution -rotobuf (c:\users\steve\anaconda3\lib\site-pa
ckages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\steve\anaconda3\lib\site-pa
ckages)
You can now load the package via spacy.load('en_core_web_lg')
```

In [7]:
```
nlp = spacy.load('en_core_web_lg')
```

In [8]:
```
# Only run this once, they will be downloaded.
nltk.download('stopwords',quiet=True)
nltk.download('wordnet',quiet=True)
nltk.download('punkt',quiet=True)
nltk.download('omw-1.4',quiet=True)
```

Out[8]:
```
True
```

Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. Target audience is the natural language processing (NLP) and information retrieval (IR) community: https://pypi.org/project/gensim/

In [9]:
```
import pkg_resources
#pkg_resources.require("gensim<=3.8.3");
```

In [10]:
```
print("Genism Version: ", gensim.__version__)
```

```
Genism Version:  4.1.2
```

In [11]:
```
# Suppress Warning Messages
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [12]:
```
def add_movie_descriptor(data: pd.DataFrame, corpus_df: pd.DataFrame):
    """
    Adds "Movie Description" to the supplied dataframe, in the form {Genre}_{P|N}_{Mov
    """
    review = np.where(corpus_df['Review Type (pos or neg)'] == 'Positive', 'P', 'N')
    data['Descriptor'] = corpus_df['Genre of Movie'] + '_' + corpus_df['Movie Title']

def get_corpus_df(path: str) -> pd.DataFrame:
    data = pd.read_csv(path,encoding="utf-8")
    add_movie_descriptor(data, data)
    sorted_data = data.sort_values(['Descriptor'])
    indexed_data = sorted_data.set_index(['Doc_ID'])
    indexed_data['Doc_ID'] = indexed_data.index
    return indexed_data

corpus_df = get_corpus_df('MSDS453_ClassCorpus_Final_Sec56_v5_20230720.csv')
```

1.2) Exploratory Data Analysis

Having loaded the appropriate Python libraries and loaded the corpus of movie reviews, let's conduct some initial exploratory data analysis.

In [13]:
```
# Tokenize sentences
def get_sentences(text: str) -> List[str]:
    return [str(x) for x in nlp(text).sents]

corpus_df['raw_sentences'] = corpus_df.Text.apply(get_sentences)
corpus_df.head(3).T
```

Out[13]:

Doc_ID	101	102	103
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Student Name	SAR	SAR	SAR
Genre of Movie	Action	Action	Action
Review Type (pos or neg)	Negative	Negative	Negative
Movie Title	Covenant	Covenant	Covenant
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103
Doc_ID	101	102	103
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...

In [14]:
```
corpus_df.shape
```

Out[14]:
```
(190, 10)
```

In [15]:
```
print(corpus_df.info());
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 190 entries, 101 to 217
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   DSI_Title                190 non-null    object
 1   Submission File Name     190 non-null    object
 2   Student Name             190 non-null    object
 3   Genre of Movie           190 non-null    object
 4   Review Type (pos or neg) 190 non-null    object
 5   Movie Title              190 non-null    object
 6   Text                     190 non-null    object
 7   Descriptor               190 non-null    object
 8   Doc_ID                   190 non-null    int64 
 9   raw_sentences            190 non-null    object
dtypes: int64(1), object(9)
memory usage: 16.3+ KB
None
```

In [16]:
```
print(corpus_df['Movie Title'].unique())
```

```
['Covenant' 'Inception' 'No time to die' 'Taken' 'The Dark Knight Rises'
 'Despicable Me 3' 'Holmes and Watson' 'Legally Blonde' 'Lost City'
 'Sisters' 'Drag Me to Hell' 'Fresh' 'It Chapter Two' 'The Toxic Avenger'
 'US' 'Annihilation' 'Minority Report' 'Oblivion' 'Pitch Black']
```

In [17]:
```
corpus_df.columns
```

Out[17]:
```
Index(['DSI_Title', 'Submission File Name', 'Student Name', 'Genre of Movie',
       'Review Type (pos or neg)', 'Movie Title', 'Text', 'Descriptor',
       'Doc_ID', 'raw_sentences'],
      dtype='object')
```

In [18]:
```
# Gather the number of reviews by genre
counts_df = corpus_df[['Genre of Movie']].copy()
counts_df['Count'] = 1
counts_df.groupby(['Genre of Movie']).count().reset_index()
```

Out[18]:

	Genre of Movie	Count
0	Action	50
1	Comedy	50
2	Horror	50
3	Sci-Fi	40

1.3) Data Wrangling

Let's conduct data wrangling to prepare the text as needed prior to the creation of our knowledge graphs.

1.3.1) Data Wrangling and Vectorization Method 1

Let's define data wrangling utility functions that we can leverage on our corpus.

In [19]:
```
def remove_punctuation(text: str) -> str:
    return re.sub('[^a-zA-Z]', '', str(text))

def remove_tags(text: str) -> str:
    return re.sub('&lt;/?.*?&gt;', '', text)

def remove_special_chars_and_digits(text: str) -> str:
    return re.sub('(\\d|\\W)+', '', text)

def get_coref_resolved_sentences(text: str) -> List[str]:
    return [str(x) for x in nlp(text).sents]

def get_lemmas(text: str, stopwords: Set[str]) -> List[str]:
    initial = [remove_tags(remove_special_chars_and_digits(remove_punctuation(x.lemma_
    return [x for x in initial if x not in stopwords]

def lemmatize_sentence(text: str, stopwords: Set[str]) -> str:
    return ' '.join(get_lemmas(text, stopwords))

def clean_doc(doc):
    #doc = remove_punctuation(doc)
    doc= ' '.join(remove_stop_words(doc))
    doc = apply_lemmatization(doc)
    return doc

def remove_stop_words(in_text):
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(in_text)
```

```
    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    return filtered_sentence

def apply_lemmatization(in_text):
    # Lemmatization
    lem = WordNetLemmatizer()
    word_list = nltk.word_tokenize(in_text)
    output = ' '.join([lem.lemmatize(w) for w in word_list])
    return output

def counter_word(text):
  count=Counter()
  for i in text.values:
    for word in i.split():
      count[word]=+1
  return count
```

Tokenize Words

In [20]:
```
default_stopwords=\
set(nltk.corpus.stopwords.words('english')).union(set(nlp.Defaults.stop_words)).union(
corpus_df['lemmas'] = corpus_df.Text.apply(lambda x: get_lemmas(x, default_stopwords))
corpus_df.lemmas.head()
```

Out[20]:
```
Doc_ID
101    [nearly, year, american, military, withdraw, a...
102    [guy, ritchie, jake, gyllenhaal, switch, place...
103    [guy, ritchie, covenant, notably, mark, featur...
104    [weird, throwback, chuck, norris, miss, action...
105    [little, odd, guy, ritchie, come, title, guy, ...
Name: lemmas, dtype: object
```

In [21]:
```
corpus_df.head(3).T
```

Out[21]:

Doc_ID	101	102	103
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Student Name	SAR	SAR	SAR
Genre of Movie	Action	Action	Action
Review Type (pos or neg)	Negative	Negative	Negative
Movie Title	Covenant	Covenant	Covenant
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103
Doc_ID	101	102	103
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...
lemmas	[nearly, year, american, military, withdraw, a...	[guy, ritchie, jake, gyllenhaal, switch, place...	[guy, ritchie, covenant, notably, mark, featur...

Reassemble Lemmatized Words

In [22]:
```
corpus_df['lemmas_joined'] = corpus_df.lemmas.apply(lambda x: ' '.join(x))
```

In [23]:
```
corpus_df['lemmas_joined'].head()
```

Out[23]:
```
Doc_ID
101    nearly year american military withdraw afghani...
102    guy ritchie jake gyllenhaal switch place peter...
103    guy ritchie covenant notably mark feature incl...
104    weird throwback chuck norris miss action film ...
105    little odd guy ritchie come title guy ritchie ...
Name: lemmas_joined, dtype: object
```

In [24]:
```
corpus_df.head(3).T
```

Out[24]:

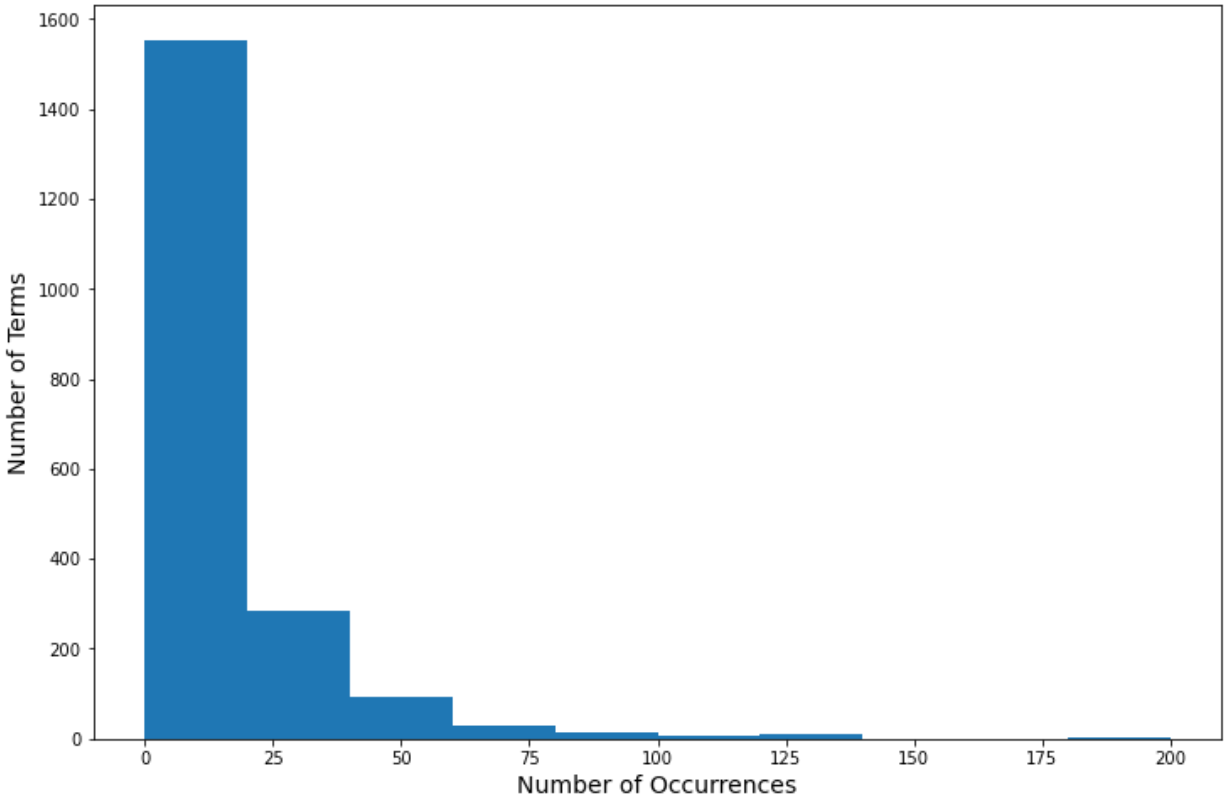
Doc_ID	101	102	103
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant
Student Name	SAR	SAR	SAR
Genre of Movie	Action	Action	Action
Review Type (pos or neg)	Negative	Negative	Negative
Movie Title	Covenant	Covenant	Covenant
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103
Doc_ID	101	102	103
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...
lemmas	[nearly, year, american, military, withdraw, a...	[guy, ritchie, jake, gyllenhaal, switch, place...	[guy, ritchie, covenant, notably, mark, featur...
lemmas_joined	nearly year american military withdraw afghani...	guy ritchie jake gyllenhaal switch place peter...	guy ritchie covenant notably mark feature incl...

Vocabulary Data Wrangling

In [25]:
```
vectorizer = CountVectorizer(ngram_range=(1, 1))
transformed_documents = vectorizer.fit_transform(corpus_df.lemmas_joined)
doc_term_matrix = transformed_documents.todense()
doc_term_df = pd.DataFrame(doc_term_matrix,
                           columns=vectorizer.get_feature_names_out(),
                           index=corpus_df.Descriptor)
print(f'All Word Vocabulary size: {doc_term_df.shape[1]}')
all_words = set(doc_term_df.columns)

vectorizer = CountVectorizer(ngram_range=(1, 1), min_df=5, max_df=.8)
transformed_documents = vectorizer.fit_transform(corpus_df.lemmas_joined)
doc_term_matrix = transformed_documents.todense()
doc_term_df = pd.DataFrame(doc_term_matrix,
                           columns=vectorizer.get_feature_names_out(),
                           index=corpus_df.Descriptor)
print(f'Curated Vocabulary size: {doc_term_df.shape[1]}')
vocabulary = set(doc_term_df.columns)

words_to_remove = default_stopwords.union(all_words - vocabulary)
```

```
All Word Vocabulary size: 9247
Curated Vocabulary size: 2000
```

In [26]:
```
plt.figure(figsize = (12, 8))
plt.hist(doc_term_df.sum(axis=0).T, range=(0, 200))
plt.xlabel('Number of Occurrences',fontsize=14)
plt.ylabel('Number of Terms',fontsize=14);
```



In [27]:
```
# Re-tokenize words, recreate joined documents
corpus_df['lemmas'] = corpus_df.Text.apply(lambda x: get_lemmas(x, words_to_remove))
corpus_df['lemmas_joined'] = corpus_df.lemmas.apply(lambda x: ' '.join(x))
corpus_df.lemmas.head()
```

Out[27]:
```
Doc_ID
101    [nearly, year, american, military, afghanistan...
102    [guy, ritchie, jake, gyllenhaal, switch, place...
103    [guy, ritchie, covenant, notably, mark, featur...
104    [weird, throwback, miss, action, guy, ritchie,...
105    [little, odd, guy, ritchie, come, title, guy, ...
Name: lemmas, dtype: object
```

Get Lemmatized and Filtered Sentences

In [28]:
```
corpus_df['sentences_lemmatized']=\
corpus_df.raw_sentences.apply(lambda x: [lemmatize_sentence(s, words_to_remove) for s
```

In [29]:
```
corpus_df.head().T
```

Out[29]:

Doc_ID	101	102	103	
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Student Name	SAR	SAR	SAR	
Genre of Movie	Action	Action	Action	
Review Type (pos or neg)	Negative	Negative	Negative	
Movie Title	Covenant	Covenant	Covenant	
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...	In a weird those
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103	Action_Cov
Doc_ID	101	102	103	
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...	[In a we to those
lemmas	[nearly, year, american, military, afghanistan...	[guy, ritchie, jake, gyllenhaal, switch, place...	[guy, ritchie, covenant, notably, mark, featur...	[weird, thr action,
lemmas_joined	nearly year american military afghanistan end ...	guy ritchie jake gyllenhaal switch place peter...	guy ritchie covenant notably mark feature incl...	weird th acti
sentences_lemmatized	[nearly year american military afghanistan end...	[guy ritchie jake gyllenhaal switch place pete...	[guy ritchie covenant notably mark feature inc...	[weird th acti

Review Documents By Movie Title

In [30]:
```
print(corpus_df['Movie Title'].unique())
```

```
['Covenant' 'Inception' 'No time to die' 'Taken' 'The Dark Knight Rises'
 'Despicable Me 3' 'Holmes and Watson' 'Legally Blonde' 'Lost City'
 'Sisters' 'Drag Me to Hell' 'Fresh' 'It Chapter Two' 'The Toxic Avenger'
 'US' 'Annihilation' 'Minority Report' 'Oblivion' 'Pitch Black']
```

In [31]:
```
movie_df = corpus_df[corpus_df['Movie Title'] == 'Sisters'].copy()
movie_df.head(3).T
```

Out[31]:

Doc_ID	136	137	138
DSI_Title	SJD_Doc6_Sisters	SJD_Doc7_Sisters	SJD_Doc8_Sisters
Submission File Name	SJD_Doc6_Sisters	SJD_Doc7_Sisters	SJD_Doc8_Sisters
Student Name	SJD	SJD	SJD
Genre of Movie	Comedy	Comedy	Comedy
Review Type (pos or neg)	Negative	Negative	Negative
Movie Title	Sisters	Sisters	Sisters
Text	Without its two leads (and some great supporti...	The chemistry between these two "Saturday Nigh...	Sisters plants the seeds of its own criticism ...
Descriptor	Comedy_Sisters_N_136	Comedy_Sisters_N_137	Comedy_Sisters_N_138
Doc_ID	136	137	138
raw_sentences	[Without its two leads (and some great support...	[The chemistry between these two "Saturday Nig...	[Sisters plants the seeds of its own criticism...
lemmas	[lead, great, support, notably, lee, comic, ti...	[chemistry, saturday, night, live, second, fea...	[sister, plant, scene, sister, kate, fey, maur...
lemmas_joined	lead great support notably lee comic timing lo...	chemistry saturday night live second feature s...	sister plant scene sister kate fey maura poehl...
sentences_lemmatized	[lead great support notably lee comic timing l...	[chemistry saturday night live second feature ...	[sister plant scene sister kate fey maura poeh...

1.4) Knowledge Graphs

1.4.1) Creation of Initial Knowledge Graphs

Let's define knowledge context graph functions.

In [32]:

```
def map_edges(map_to: str, map_from: Set[str], df: pd.DataFrame):
    print(f'Before mapping {", ".join(map_from)} -> {map_to}: {sum(df.edge == map_to)}')
    df['edge'] = np.where(kg_df.edge.isin(map_from), map_to, kg_df.edge)
    print(f'After mapping {", ".join(map_from)} -> {map_to}: {sum(df.edge == map_to)}')

def map_sources_and_targets(map_to: str, map_from: Set[str], df: pd.DataFrame):
    before = sum(df.source == map_to) + sum(df.target == map_to)
    print(f'Before mapping {", ".join(map_from)} -> {map_to}: {before}')

    df['source'] = np.where(kg_df.source.isin(map_from), map_to, kg_df.source)
    df['target'] = np.where(kg_df.target.isin(map_from), map_to, kg_df.target)

    after = sum(df.source == map_to) + sum(df.target == map_to)
    print(f'After mapping {", ".join(map_from)} -> {map_to}: {after}')

def get_neighborhood(sources: Set[str], edge_types: Set[str], depth: int, df: pd.DataF
    output = []
```

```
    for d in range(depth):
        if edge_types is not None:
            rows = df[(df.edge.isin(edge_types)) & ((df.source.isin(sources)) | (df.ta
        else:
            rows = df[(df.source.isin(sources)) | (df.target.isin(sources))].copy()

        output.append(rows)
        sources = set(rows.target).union(set(rows.source))

    return pd.concat(output).drop_duplicates()

def find_sources_and_targets_with_patterns(patterns: List[str], df: pd.DataFrame):
    mask = np.zeros(kg_df.shape[0])
    for pattern in patterns:
        mask = mask | (df.source.str.contains(pattern)) | (df.target.str.contains(patt

    return df[mask]
# Examples of how to use the function:
# find_sources_and_targets_with_patterns(['action'], kg_df)
# find_sources_and_targets_with_patterns(['terror'], kg_df)
# find_sources_and_targets_with_patterns(['novel'], kg_df)
# find_sources_and_targets_with_patterns(['director', 'campbell'], kg_df)


def plot_graph(df: pd.DataFrame, show_edges: bool = False, figsize: Tuple[int, int] =
    graph = nx.from_pandas_edgelist(df, "source", "target", edge_attr='edge', create_u

    plt.figure(figsize=figsize)
    if use_circular:
        pos = nx.circular_layout(graph)
    else:
        pos = nx.kamada_kawai_layout(graph)

    nx.draw(graph, with_labels=True, node_color='skyblue', edge_cmap=plt.cm.Blues, pos
    if show_edges:
        nx.draw_networkx_edge_labels(graph, pos=pos, font_size=8)

    plt.show()

def get_top_sources_and_targets(df: pd.DataFrame, top: int = 10):
    return (Counter(df.source) + Counter(df.target)).most_common(top)

def get_top_edges(df: pd.DataFrame, top: int = 10):
    return Counter(df.edge).most_common(top)

def get_dataset_partitions_pd(df, train_split=0.8, val_split=0.10, test_split=0.10):
       # Specify seed to always have the same split distribution between runs
    df_sample = df.sample(frac=1, random_state=12)
    indices_or_sections = [int(.8*len(df)), int(.9*len(df))]
    train_ds, val_ds, test_ds = np.split(df_sample, indices_or_sections)
    return train_ds, val_ds, test_ds
```

Let's define entity extraction functions.

In [33]:

```
def get_entities(sent):
  ## chunk 1
  ent1 = ""
  ent2 = ""
```

```
  prv_tok_dep = ""    # dependency tag of previous token in the sentence
  prv_tok_text = ""   # previous token in the sentence

  prefix = ""
  modifier = ""

  for tok in nlp(sent):
    ## chunk 2
    # if token is a punctuation mark then move on to the next token
    if tok.dep_ != "punct":
      # check: token is a compound word or not
      if tok.dep_ == "compound":
        prefix = tok.text
        # if the previous word was also a 'compound' then add the current word to it
        if prv_tok_dep == "compound":
          prefix = prv_tok_text + " "+ tok.text

      # check: token is a modifier or not
      if tok.dep_.endswith("mod") == True:
        modifier = tok.text
        # if the previous word was also a 'compound' then add the current word to it
        if prv_tok_dep == "compound":
          modifier = prv_tok_text + " "+ tok.text

      ## chunk 3
      if tok.dep_.find("subj") == True:
        ent1 = modifier +" "+ prefix + " "+ tok.text
        prefix = ""
        modifier = ""
        prv_tok_dep = ""
        prv_tok_text = ""

      ## chunk 4
      if tok.dep_.find("obj") == True:
        ent2 = modifier +" "+ prefix +" "+ tok.text

      ## chunk 5
      # update variables
      prv_tok_dep = tok.dep_
      prv_tok_text = tok.text

  return [ent1.strip(), ent2.strip()]

def get_relation(sent):
    try:
        doc = nlp(sent)

        # Matcher class object
        matcher = Matcher(nlp.vocab)

        #define the pattern
        pattern = [{'DEP':'ROOT'},
                {'DEP':'prep','OP':"?"},
                {'DEP':'agent','OP':"?"},
                {'POS':'ADJ','OP':"?"}]
        matcher.add("matching_1", [pattern])
        matches = matcher(doc)
        k = len(matches) - 1
        span = doc[matches[k][1]:matches[k][2]]
```

```
        return(span.text)
    except:
        pass

def get_subject_verb_object(sent):
  ent1 = ""
  ent2 = ""
  root = ""

  for tok in nlp(sent):
      if tok.dep_ == 'ROOT':
        root = tok.text
      elif tok.dep_ == "nsubj":
        ent1 = tok.text
      elif tok.dep_ == "dobj":
        ent2 = tok.text

      if ent1 != '' and ent2 != '' and root != '':
        break

  return [ent1, root, ent2]
```

Knowledge Graph Extraction

In [34]:
```
corpus_text_sentences = [y for x in movie_df.raw_sentences for y in x]
example_sentence = nlp(corpus_text_sentences[5])
corpus_text_sentences[5]
```

Out[34]:
```
"As with her Parks & Rec character, Poehler's Maura is an uptight forty-something do-gooder here."
```

In [35]:
```
entity_pairs = [get_entities(x) for x in tqdm(corpus_text_sentences)]
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 210/210 [00:01<00:00, 124.96it/s]
```

In [36]:
```
entity_pairs
```

Out[36]:
```
[['Sisters', 'gags'],
 ['it', 'genuine  stars'],
 ['likes', 'impressive Jenny indies'],
 ['we', 'running TV sitcoms'],
 ['far  Sisters', 'yet  talents'],
 ['Parks  Maura', 'Parks  character'],
 ['parents', 'home James sale'],
 ['older  who', 'financial  trouble'],
 ['in nostalgia duo', 'final teen years'],
 ['I', 'along  Fey'],
 ['days writer Paula I', 'one  liner'],
 ['more  Fey', 'Rock Girls shows'],
 ['likeable  charm', 'way  script'],
 ['that', 'two  hours'],
 ['extended party that', 'one Trading point'],
 ['us', 'her'],
 ['such cast sequences', 'cast list'],
 ['That', 'final  cut'],
 ['second  they', 'Night Live Night alumni'],
 ['also  what', 'unfocused  screenplay'],
 ['they', '2 hour joke'],
 ['material', 'strengths'],
 ['normal  everyone', 'cast'],
 ['', 'shoehorned character arcs'],
 ['teenage Madison who', 'teenage Madison Davenport'],
 ['suddenly childhood sisters', 'last  stuff'],
 ['Sisters', 'uncomfortable adult conversations'],
 ['single premise movie', 'energy'],
 ['when  intelligence', 'sophomoric  slapstick'],
 ['', 'extended party sequence'],
 ['Maya Rudolph co', 'pretentious  maturity'],
 ['Kate', 'attractive  neighbor'],
 ['she', 'party'],
 ['natural backyard tendency', 'Sisters'],
 ['', ''],
 ['own  Kate', 'high school friend'],
 ['that', 'movie'],
 ['stars', ''],
 ['Tina Fey', 'critical Saturday Night Live'],
 ['dead Sarah Poehler', 'so  Parks'],
 ['Maybe  all', 'heads'],
 ['them', 'it'],
 ['available  that', 'available  humor'],
 ['funny  material', 'camera'],
 ['genuine  Fey', 'them'],
 ['when  someone', 'music box'],
 ['time', ''],
 ['that', 'common  denominator'],
 ['Sisters', 'many  control'],
 ['Animal which', 'Animal House'],
 ['', ''],
 ['humor', 'it'],
 ['Guys', 'just D them'],
 ['They', 'wacky  personalities'],
 ['Animal House', 'when  spots'],
 ['It', 'relentless  pace'],
 ['It', 'big  laugh'],
 ['characters', ''],
 ['', 'actually  lunacy'],
 ['Sisters', 'as  temple'],
```

```
['parents', 'childhood home'],
['major  house', 'snooty  couple'],
['This', 'normal  people'],
['where  supplies', 'sight dollars'],
['how  they', 'one hundred one end'],
['that', 'an screen time'],
['Tina Fey', ''],
['personas', ''],
['television that', 'cultural  moment'],
['three peat hosting', 'public  friendship'],
['most  anyone', 'time'],
['new  movie', 'simply  Amy'],
['Rather  written', 'comedic party complications'],
['still  it', 'repeatedly  itself'],
['divorced Maura who', 'older Kate Fey'],
['parents', 'James sale'],
['lead pair', 'Baby Mama'],
['together  Kate', 'it'],
['one  Kate', 'responsible  one'],
['decisions', 'more  Fey'],
['well  she', 'often Leslie Knope'],
['I', 'frustrated'],
['longtime television writer which', 'strung  structure'],
['movie', 'familiar Maya Rudolph'],
['kaleidoscope', 'central  story'],
['first Pitch opportunity', 'music montage dance break'],
['she', 'movie'],
['Here  she', 'long  time'],
['We', 'Tina Fey'],
['loving  that', 'well  others'],
['so  Sisters', 'it'],
['', ''],
['two slam it', 'last tepid Baby 2008'],
['ish  lady', 'white trash surrogate'],
['ensuing  Poehler', 'sly screen observances'],
['markedly Baby it', 'packaged  stars'],
['real  Fey', 'wisely  friend'],
['', 'biographical Paula life'],
['new  family', 'room'],
['responsible  that', 'hard partying sister'],
['big  they', 'lives'],
['themselves', 'free  will'],
['it', 'while'],
['heretofore Ike it', 'nice  moments'],
['gross  that', 'increasing  debauchery'],
['wild  that', 'latter  film'],
['comedic Horatio we', 'emotional Maya resonance'],
['magnificently Bill Cosby they', 'moreover  Oscars'],
['SNL regular Jason who', 'Pitch Perfect'],
['critics', 'demographic audience group'],
['such  it', 'real life auditorium'],
['female Bechdel dynamic', 'gravitationally  comedy'],
['Fey', "song'n'dance  material"],
['Favorite Year', "Peter O'Toole film"],
['two  who', 'lives'],
['now hair dye that', 'now hair dye materials'],
['Maura', 'designated  parties'],
['how  they', 'film'],
['teenage  characters', 'awful  lot'],
['where family they', 'more retirement lifestyle'],
```

```
['They', 'long distance calls'],
['away teen it', 'trash'],
['when  they', 'few  odds'],
['cultural  that', 'cultural  juggernaut'],
['so  they', 'them'],
['strong  men', 'strong  that'],
['it', ''],
['what', 'them'],
['sheer  that', 'sheer  nature'],
['it', 'director Jason Moore'],
['well  we', 'dynamic'],
['specifically  they', '2008  comedy'],
['divorced sister who', 'own  enjoyment'],
['single  who', 'selfishly  smarts'],
['few stuff they', 'wild  future'],
['Maura', 'right rounding outfit'],
['that', 'novelty'],
['clothing that', 'laughter  tears'],
['Informer"-is  they', 'so  goofiness'],
['also core that', 'whole  movie'],
['little J. Fox posters', 'powerful diary reminder'],
['', ''],
['', ''],
['', ''],
['', ''],
['', ''],
['', ''],
['Women', ''],
['actually  40something', 'secretly  resistance'],
['responsible  men', 'fun'],
['', 'So  Sisters'],
['juvenile  men', 'this'],
['positive  that', 'approved  womanhood'],
['', 'appropriate  womanliness'],
['', ''],
['how  this', ''],
['how  this', 'how  women'],
['how  this', 'it'],
['someone', ''],
['', 'Pitch tone'],
['40something  men', 'stupid  ways'],
['titular  who', 'Admission Maura Out'],
['childhood they', 'even  this'],
['it', 'them'],
['kindly  movie', 'entitlement'],
['Hooray  it', ''],
['time  women', 'least  metaphysical'],
['consistently  it', 'Amy Poehler'],
['only  film', 'lesser  hands'],
['who', 'rated life territory'],
['runtime Sisters', 'energetic  comedy'],
['', 'Poehler Maura'],
['aged Madison who', 'competent  parent'],
['two Dianne race', 'them'],
['They', 'old  bedrooms'],
['then  they', 'late landmark 1980'],
['enough  Poehler', 'writing Sisters'],
['most', 'minimum'],
['extended Scarface reference', 'tendency'],
['always  that', 'always  corner'],
```

```
['budding  romance', 'hilarious neighborhood turn'],
['waking  they', 'senior  years'],
['pretty  subplot', 'daughter'],
['minute joke it', 'minute joke capacity'],
['Poehler', 'youthful  mother'],
['supporting  cast', 'impatient Dianne parents'],
['John Cena', 'one drug dealer character'],
['Maya Rudolph', 'stuck  realtor'],
['Ike Barinholtz', 'convincing love Maura'],
['more  he', 'more  cocaine'],
['two hour Sisters', 'curiously  some'],
['Comedic  sisters', '2015/16 summer season'],
['seriously  they', 'real  SISTERS'],
['', 'January 6th Universal Pictures Australia'],
['It', '118mins'],
['who', 'social  life'],
['single beauty she', 'work'],
['She', 'responsible  life'],
['family they', 'family rooms'],
['Back Ellis sisters', 'old partying lives'],
['Bad Neighbours you', 'money'],
['CHICK you', 'lady LOT'],
['teenage  guys', 'much Will Ferrell movie'],
['they', 'it'],
['They', ''],
['else  it', 'else  roles'],
['there  you', 'minute'],
['you', 'LOL'],
['you', 'also support roles'],
['', '']]
```

Create Dataframe of Sources, Edges, and Targets

In [37]:
```
relations = [get_relation(x) for x in corpus_text_sentences]
#extract subject and object
source = [i[0] for i in entity_pairs]
target = [i[1] for i in entity_pairs]
kg_df = pd.DataFrame({'source': source, 'target': target, 'edge': relations})
```

Knowledge Graph Preprocessing (Lowercase Transformation and Removal of Empty Spaces)

In [38]:
```
# Move everything to lower case
kg_df.source = kg_df.source.str.lower()
kg_df.target = kg_df.target.str.lower()
kg_df.edge = kg_df.edge.str.lower()

# Filter out empties
kg_df = kg_df[kg_df.source != '']
kg_df = kg_df[kg_df.target != '']
kg_df = kg_df[kg_df.edge != ''].copy()
```

In [39]:
```
kg_df.head(6).T
```

Out[39]:

	0	1	2	3	4	5
source	sisters	it	likes	we	far sisters	parks maura
target	gags	genuine stars	impressive jenny indies	running tv sitcoms	yet talents	parks character
edge	be	's	risen to	have	failed	is

In [40]:
```
kg_df.shape
```

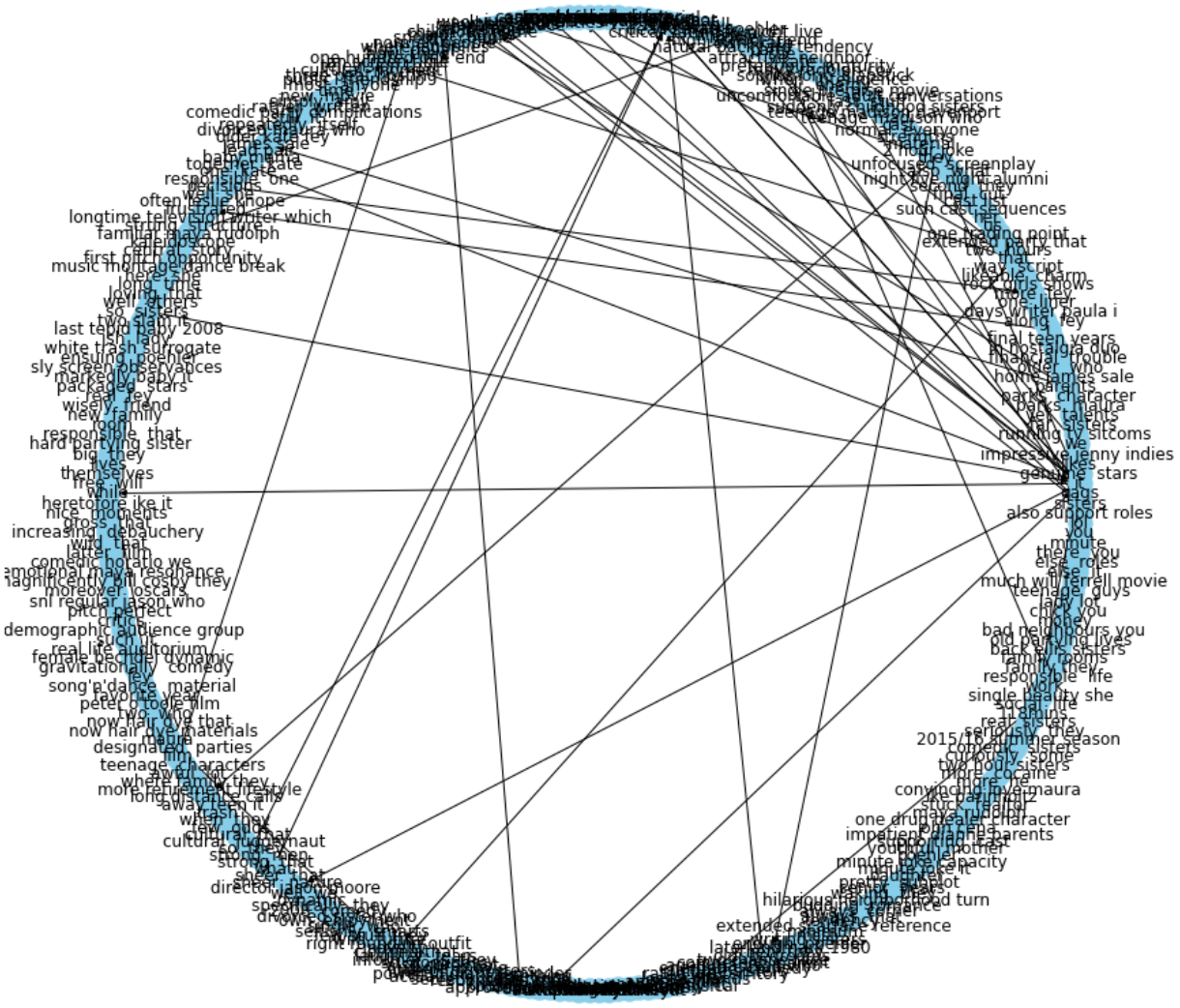
Out[40]:
```
(179, 3)
```

In [41]:
```
len(corpus_text_sentences)
```

Out[41]:
```
210
```

Plotting Knowledge Graph

In [42]:
```
plot_graph(kg_df, use_circular=True)
```



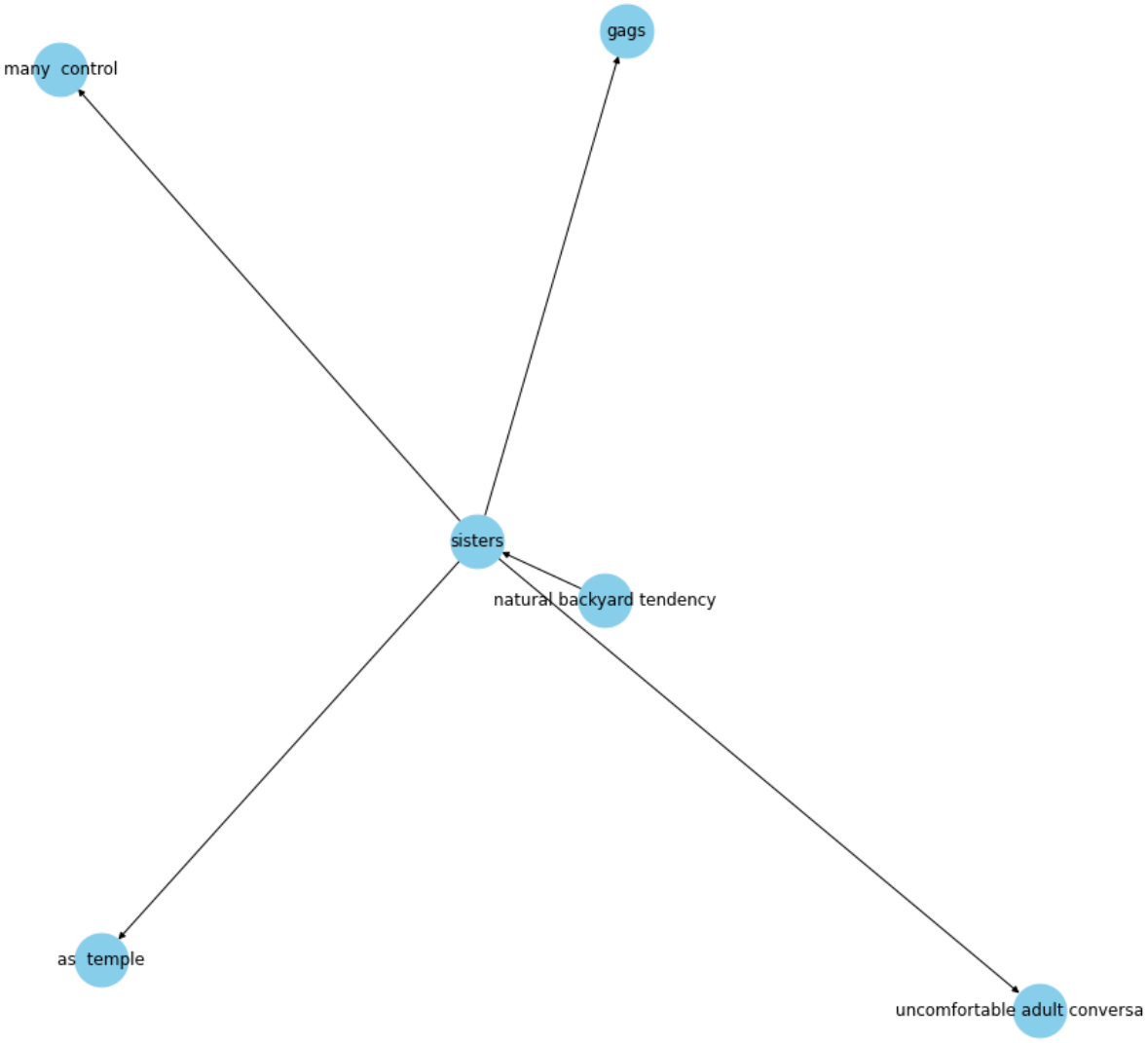
Plotting Knowledge Graph Subset 1

Let's examine the subset of our knowledge graph connected to nodes that correspond to the movie title, "Sisters."

In [43]:
```
RELATION_TO_EXPLORE = 'sisters'
```

In [44]:
```
G = nx.from_pandas_edgelist(kg_df[(kg_df['source'] == RELATION_TO_EXPLORE) | (kg_df['t
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)
```

```
plt.show()
```

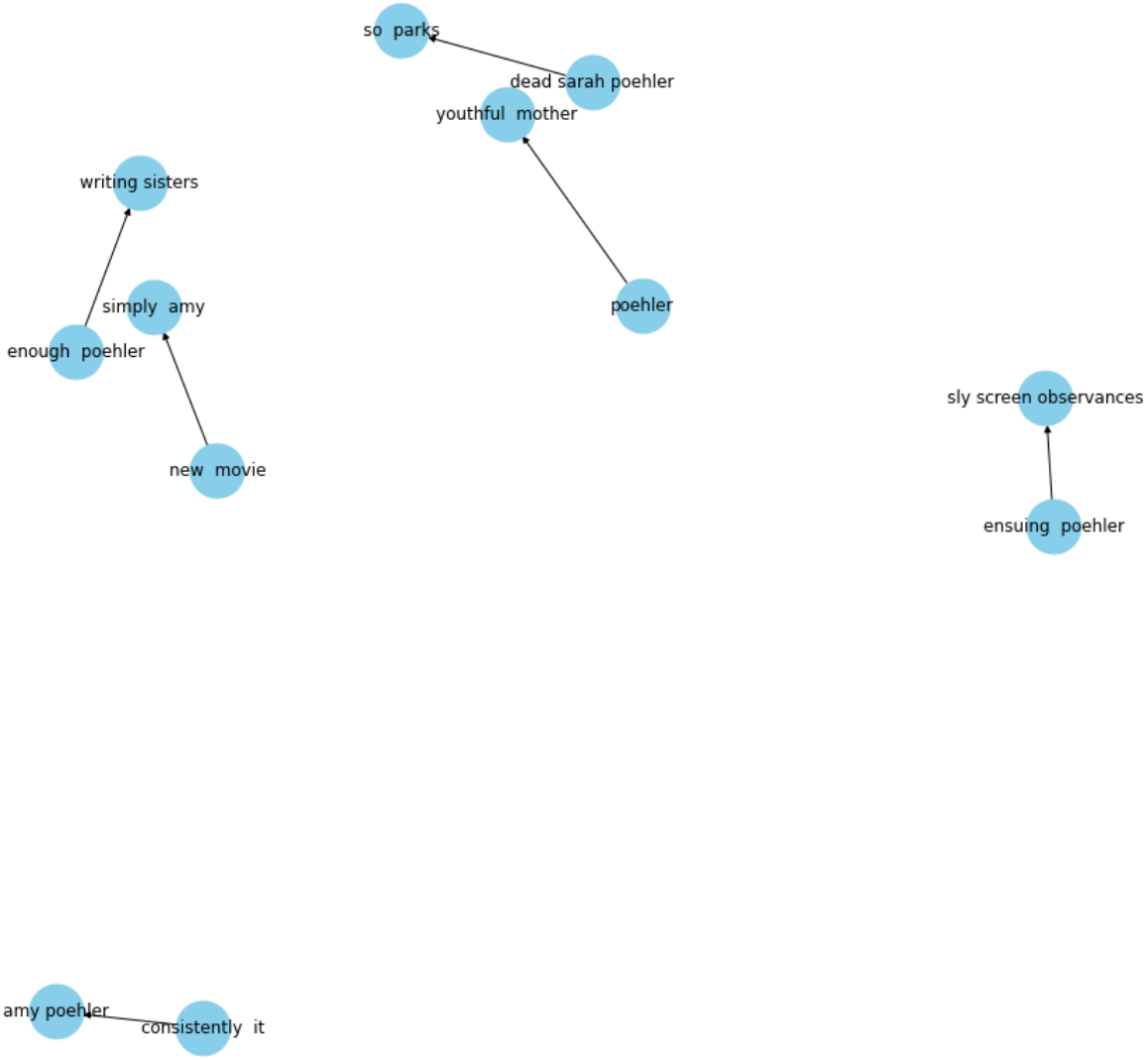


Plotting Knowledge Graph Subset 2

Let's examine the subset of our knowledge graph connected to nodes that correspond to one of the lead actresses, Amy Poehler.

In [45]:
```
G = nx.from_pandas_edgelist(kg_df[(kg_df['source'].str.contains("amy") == True) |
                                  (kg_df['target'].str.contains("amy") == True) |
                                  (kg_df['source'].str.contains("poehler") == True) |
                                  (kg_df['target'].str.contains("poehler") == True)
                                  ],
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes
```

```
nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```

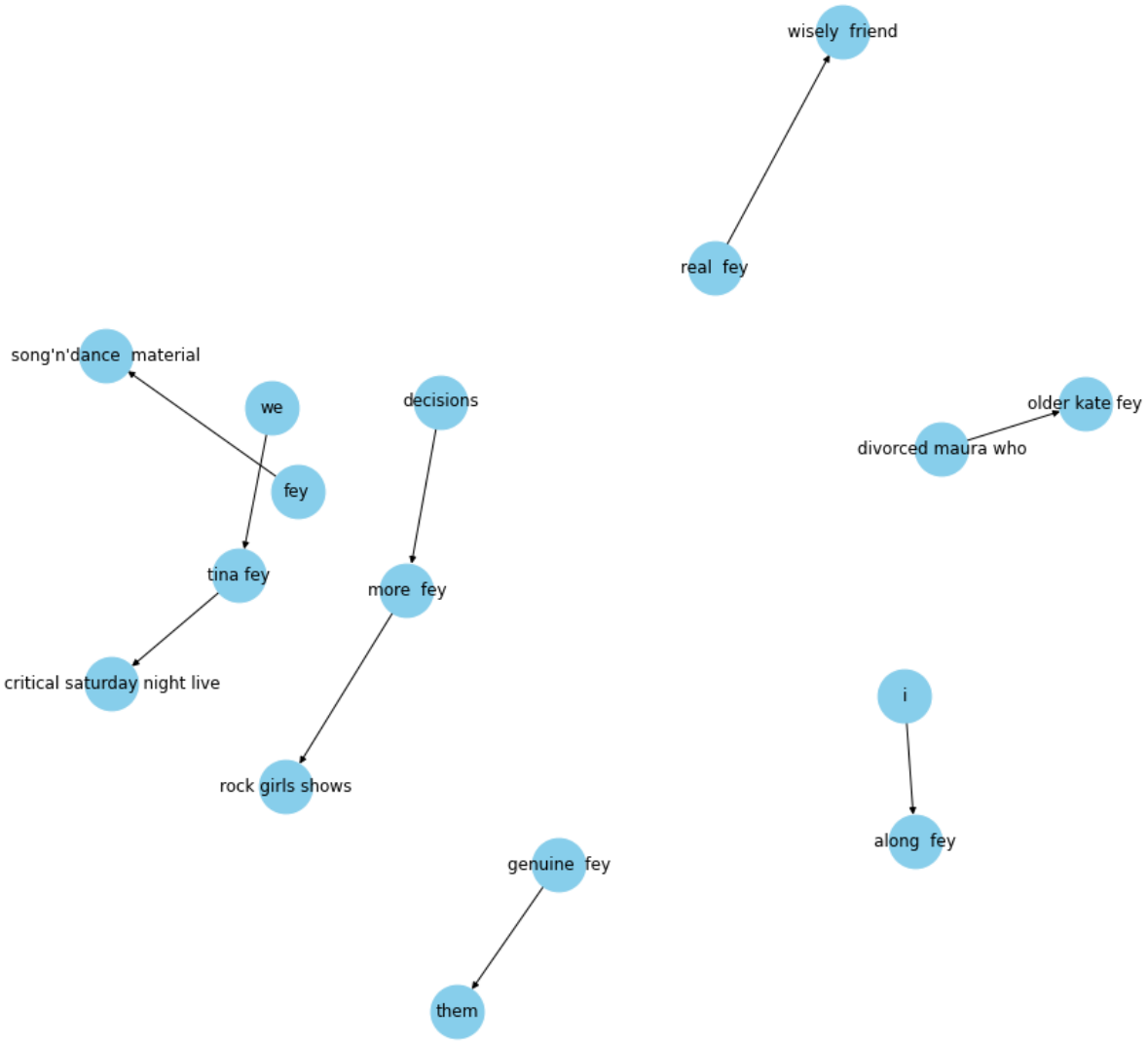


Plotting Knowledge Graph Subset 3

Let's examine the subset of our knowledge graph connected to nodes that correspond to one of the lead actresses, Tina Fey.

In [46]:
```
G = nx.from_pandas_edgelist(kg_df[(kg_df['source'].str.contains("tina") == True) |
                                  (kg_df['target'].str.contains("tina") == True) |
                                  (kg_df['source'].str.contains("fey") == True) |
                                  (kg_df['target'].str.contains("fey") == True)
                                  ],
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())
```

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```

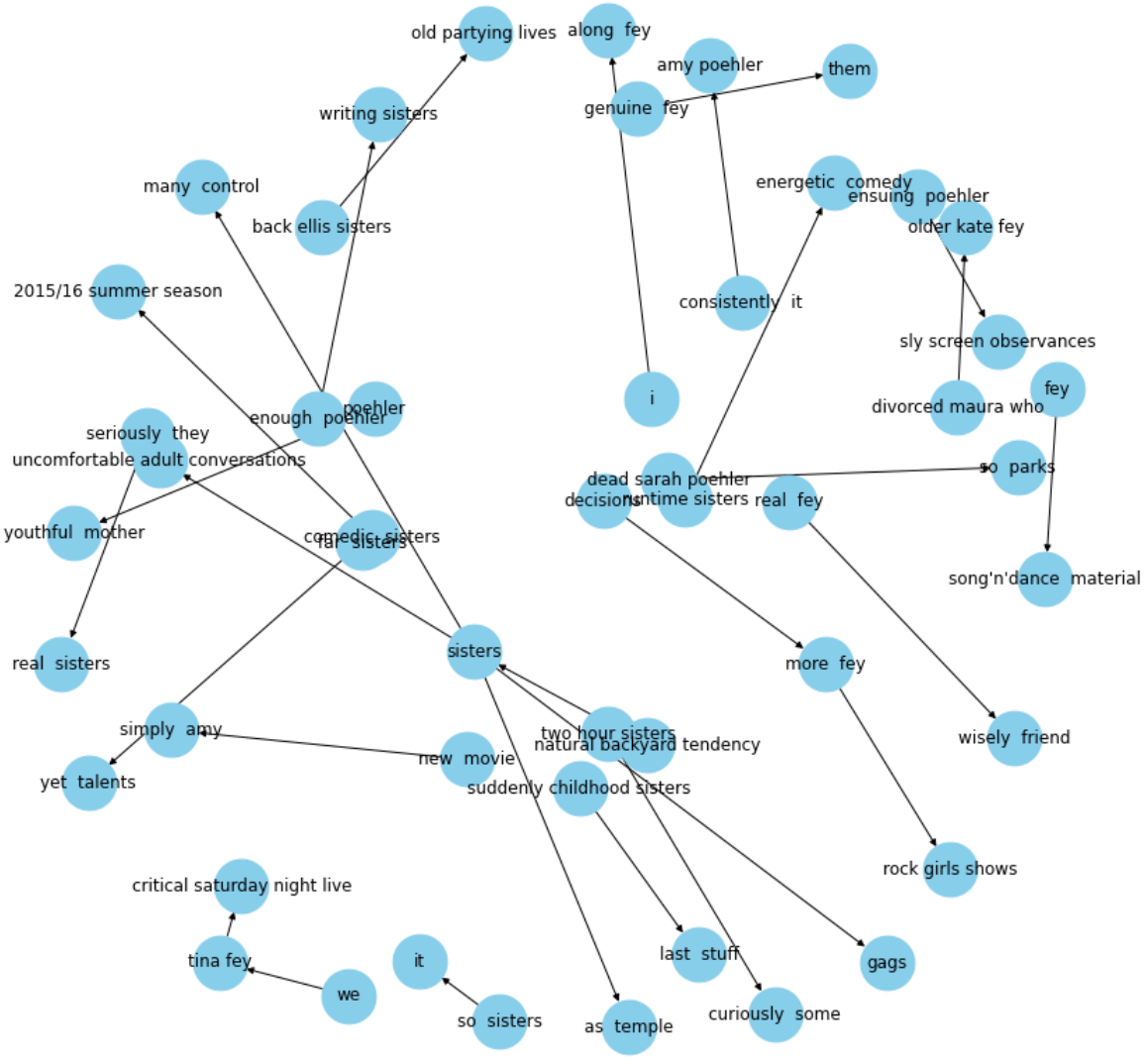


Let's look at a subset of the knowledge graph that contains nodes that refer to some of our key named entities: Tina Fey, Amy Poehler, and "Sisters."

In [210…

```
G = nx.from_pandas_edgelist(kg_df[(kg_df['source'].str.contains("tina") == True) |
                                  (kg_df['target'].str.contains("tina") == True) |
                                  (kg_df['source'].str.contains("fey") == True) |
                                  (kg_df['target'].str.contains("fey") == True) |
                                  (kg_df['source'].str.contains("amy") == True) |
                                  (kg_df['target'].str.contains("amy") == True) |
```

```
                                    (kg_df['source'].str.contains("poehler") == True) |
                                    (kg_df['target'].str.contains("poehler") == True) |
                                    (kg_df['source'].str.contains("sisters") == True) |
                                    (kg_df['target'].str.contains("sisters") == True)
                                    ],
                                "source", "target",
                                edge_attr=True,
                                create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```



1.4.2) Creation of Revised Knowledge Graphs After Consolidating Equivalence Classes

Let's comb through the sources, edges, and targets, so that we can consolidate equivalent classes, so that our knowledge graphs can better represent named entities that are in fact equivalent and better represent connections between terms in the corpus.

In [47]:
```
kg_df_consolidated = kg_df.copy(deep = False)
```

In [48]:
```
with pd.option_context("display.max_rows",1000):
    display(kg_df[['source', 'edge', 'target']].head(1000))
```

	source	edge	target
0	sisters	be	gags
1	it	's	genuine stars
2	likes	risen to	impressive jenny indies
3	we	have	running tv sitcoms
4	far sisters	failed	yet talents
5	parks maura	is	parks character
6	parents	learns	home james sale
7	older who	go	financial trouble
8	in nostalgia duo	decide	final teen years
9	i	had	along fey
10	days writer paula i	is devoid	one liner
11	more fey	is	rock girls shows
12	likeable charm	brings	way script
13	that	's	two hours
14	extended party that	is	one trading point
15	us	are	her
16	such cast sequences	gone	cast list
17	that	's	final cut
18	second they	is undeniable	night live night alumni
19	also what	is	unfocused screenplay
20	they	have	2 hour joke
21	material	is clear	strengths
22	normal everyone	feels like	cast
24	teenage madison who	is	teenage madison davenport
25	suddenly childhood sisters	decide	last stuff
26	sisters	forces	uncomfortable adult conversations
27	single premise movie	is	energy
28	when intelligence	is	sophomoric slapstick
30	maya rudolph co	-	pretentious maturity
31	kate	learn	attractive neighbor
32	she	invites to	party
33	natural backyard tendency	are occasional	sisters
35	own kate	plants	high school friend

	source	edge	target
36	that	think	movie
38	tina fey	received	critical saturday night live
39	dead sarah poehler	bad	so parks
40	maybe all	gone to	heads
41	them	believing	it
42	available that	mined	available humor
43	funny material	needed	camera
44	genuine fey	are	them
45	when someone	was	music box
47	that	want	common denominator
48	sisters	covers	many control
49	animal which	think of	animal house
51	humor	were funny	it
52	guys	performing	just d them
53	they	were	wacky personalities
54	animal house	knew	when spots
55	it	hurl verbal	relentless pace
56	it	took	big laugh
59	sisters	is	as temple
60	parents	are	childhood home
61	major house	fly to	snooty couple
62	this	is	normal people
63	where supplies	is	sight dollars
64	how they	imagine	one hundred one end
65	that	transpires at	an screen time
68	television that	created	cultural moment
69	three peat hosting	cemented	public friendship
70	most anyone	seemed	time
71	new movie	satisfied	simply amy
72	rather written	tends	comedic party complications
73	still it	like	repeatedly itself
74	divorced maura who	plays	older kate fey
75	parents	have	james sale

	source	edge	target
76	lead pair	play with	baby mama
77	together kate	has	it
78	one kate	flip	responsible one
79	decisions	work in	more fey
80	well she	gives	often leslie knope
81	i	am brassy	frustrated
82	longtime television writer which	is	strung structure
83	movie	stuffed with familiar	familiar maya rudolph
84	kaleidoscope	diffuses	central story
85	first pitch opportunity	missed	music montage dance break
86	she	is	movie
87	here she	plays	long time
88	we	love	tina fey
89	loving that	shown	well others
90	so sisters	's	it
92	two slam it	's	last tepid baby 2008
93	ish lady	anticipated	white trash surrogate
94	ensuing poehler	reunited on several	sly screen observances
95	markedly baby it	have	packaged stars
96	real fey	seemed	wisely friend
98	new family	tasked with	room
99	responsible that	is	hard partying sister
100	big they	decide	lives
101	themselves	promises	free will
102	it	takes	while
103	heretofore ike it	's	nice moments
104	gross that	get	increasing debauchery
105	wild that	knows	latter film
106	comedic horatio we	is	emotional maya resonance
107	magnificently bill cosby they	is	moreover oscars
108	snl regular jason who	is	pitch perfect
109	critics	's	demographic audience group
110	such it	need	real life auditorium

	source	edge	target
111	female bechdel dynamic	is	gravitationally comedy
112	fey	laughs with	song'n'dance material
113	favorite year	are	peter o'toole film
114	two who	play	lives
115	now hair dye that	makes	now hair dye materials
116	maura	is	designated parties
117	how they	's	film
118	teenage characters	shows	awful lot
119	where family they	triggered	more retirement lifestyle
120	they	are in	long distance calls
121	away teen it	informed	trash
122	when they	kept	few odds
123	cultural that	is	cultural juggernaut
124	so they	see	them
125	strong men	entertain through	strong that
127	what	is	them
128	sheer that	feels refreshing	sheer nature
129	it	keep	director jason moore
130	well we	know	dynamic
131	specifically they	is	2008 comedy
132	divorced sister who	is	own enjoyment
133	single who	is	selfishly smarts
134	few stuff they	learn	wild future
135	maura	are in	right rounding outfit
136	that	's	novelty
137	clothing that	is	laughter tears
138	informer"-is they	routine	so goofiness
139	also core that	's	whole movie
140	little j. fox posters	serves as	powerful diary reminder
148	actually 40something	love	secretly resistance
149	responsible men	's	fun
151	juvenile men	is	this
152	positive that	be	approved womanhood

	source	edge	target
156	how this	sees	how women
157	how this	sees	it
160	40something men	got	stupid ways
161	titular who	are	admission maura out
162	childhood they	sold	even this
163	it	has	them
164	kindly movie	thought	entitlement
166	time women	get	least metaphysical
167	consistently it	's	amy poehler
168	only film	cringe at	lesser hands
169	who	does	rated life territory
170	runtime sisters	is as rapid	energetic comedy
172	aged madison who	is impulsive	competent parent
173	two dianne race	stop	them
174	they	're	old bedrooms
175	then they	invite	late landmark 1980
176	enough poehler	had	writing sisters
177	most	feels like	minimum
178	extended scarface reference	have	tendency
179	always that	's	always corner
180	budding romance	feels fresh	hilarious neighborhood turn
181	waking they	works in	senior years
182	pretty subplot	is	daughter
183	minute joke it	exists as	minute joke capacity
184	poehler	are	youthful mother
185	supporting cast	is	impatient dianne parents
186	john cena	gets	one drug dealer character
187	maya rudolph	gets	stuck realtor
188	ike barinholtz	keeps	convincing love maura
189	more he	comes	more cocaine
190	two hour sisters	run	curiously some
191	comedic sisters	join	2015/16 summer season
192	seriously they	believe	real sisters

	source	edge	target
194	it	runs for	118mins
195	who	is responsible	social life
196	single beauty she	is	work
197	she	has	responsible life
198	family they	announce	family rooms
199	back ellis sisters	were famous	old partying lives
200	bad neighbours you	think	money
201	chick you	is	lady lot
202	teenage guys	see teenage	much will ferrell movie
203	they	tear	it
205	else it	been horrendous	else roles
206	there you	are fast	minute
207	you	written	lol
208	you	is	also support roles

In [49]:
```
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "tina fey" if "fey" in str(
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "tina fey" if "tina" in str
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "amy poehler" if "amy" in s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "amy poehler" if "poehler"
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "sisters" if "sisters" in s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "paula pell" if "paula" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "paula pell" if "pell" in s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "jason moore" if "jason" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "jason moore" if "moore" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "saturday night live" if "s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "saturday night live" if "s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "saturday night live" if "n
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "saturday night live" if "s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "maya rudolph" if "maya" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "maya rudolph" if "rudolph"
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "maya rudolph" if "maya" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "maura ellis" if "maura" in
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "kate ellis" if "kate" in s
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "madison davenport" if "mad
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "madison davenport" if "dav
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "dianne wiest" if "dianne"
kg_df_consolidated = kg_df_consolidated.applymap(lambda x: "dianne wiest" if "wiest" i
```

In [50]:
```
with pd.option_context("display.max_rows",1000):
    display(kg_df_consolidated[['source', 'edge', 'target']].head(1000))
```

	source	edge	target
0	sisters	be	gags
1	it	's	genuine stars
2	likes	risen to	impressive jenny indies
3	we	have	running tv sitcoms
4	sisters	failed	yet talents
5	maura ellis	is	parks character
6	parents	learns	home james sale
7	older who	go	financial trouble
8	in nostalgia duo	decide	final teen years
9	i	had	tina fey
10	paula pell	is devoid	one liner
11	tina fey	is	rock girls shows
12	likeable charm	brings	way script
13	that	's	two hours
14	extended party that	is	one trading point
15	us	are	her
16	such cast sequences	gone	cast list
17	that	's	final cut
18	second they	is undeniable	saturday night live
19	also what	is	unfocused screenplay
20	they	have	2 hour joke
21	material	is clear	strengths
22	normal everyone	feels like	cast
24	madison davenport	is	madison davenport
25	sisters	decide	last stuff
26	sisters	forces	uncomfortable adult conversations
27	single premise movie	is	energy
28	when intelligence	is	sophomoric slapstick
30	maya rudolph	-	pretentious maturity
31	kate ellis	learn	attractive neighbor
32	she	invites to	party
33	natural backyard tendency	are occasional	sisters
35	kate ellis	plants	high school friend

	source	edge	target
36	that	think	movie
38	tina fey	received	saturday night live
39	amy poehler	bad	so parks
40	maybe all	gone to	heads
41	them	believing	it
42	available that	mined	available humor
43	funny material	needed	camera
44	tina fey	are	them
45	when someone	was	music box
47	that	want	common denominator
48	sisters	covers	many control
49	animal which	think of	animal house
51	humor	were funny	it
52	guys	performing	just d them
53	they	were	wacky personalities
54	animal house	knew	when spots
55	it	hurl verbal	relentless pace
56	it	took	big laugh
59	sisters	is	as temple
60	parents	are	childhood home
61	major house	fly to	snooty couple
62	this	is	normal people
63	where supplies	is	sight dollars
64	how they	imagine	one hundred one end
65	that	transpires at	an screen time
68	television that	created	cultural moment
69	three peat hosting	cemented	public friendship
70	most anyone	seemed	time
71	new movie	satisfied	amy poehler
72	rather written	tends	comedic party complications
73	still it	like	repeatedly itself
74	maura ellis	plays	tina fey
75	parents	have	james sale

	source	edge	target
76	lead pair	play with	baby mama
77	kate ellis	has	it
78	kate ellis	flip	responsible one
79	decisions	work in	tina fey
80	well she	gives	often leslie knope
81	i	am brassy	frustrated
82	longtime television writer which	is	strung structure
83	movie	stuffed with familiar	maya rudolph
84	kaleidoscope	diffuses	central story
85	first pitch opportunity	missed	music montage dance break
86	she	is	movie
87	here she	plays	long time
88	we	love	tina fey
89	loving that	shown	well others
90	sisters	's	it
92	two slam it	's	last tepid baby 2008
93	ish lady	anticipated	white trash surrogate
94	amy poehler	reunited on several	sly screen observances
95	markedly baby it	have	packaged stars
96	tina fey	seemed	wisely friend
98	new family	tasked with	room
99	responsible that	is	hard partying sister
100	big they	decide	lives
101	themselves	promises	free will
102	it	takes	while
103	heretofore ike it	's	nice moments
104	gross that	get	increasing debauchery
105	wild that	knows	latter film
106	comedic horatio we	is	maya rudolph
107	magnificently bill cosby they	is	moreover oscars
108	jason moore	is	pitch perfect
109	critics	's	demographic audience group
110	such it	need	real life auditorium

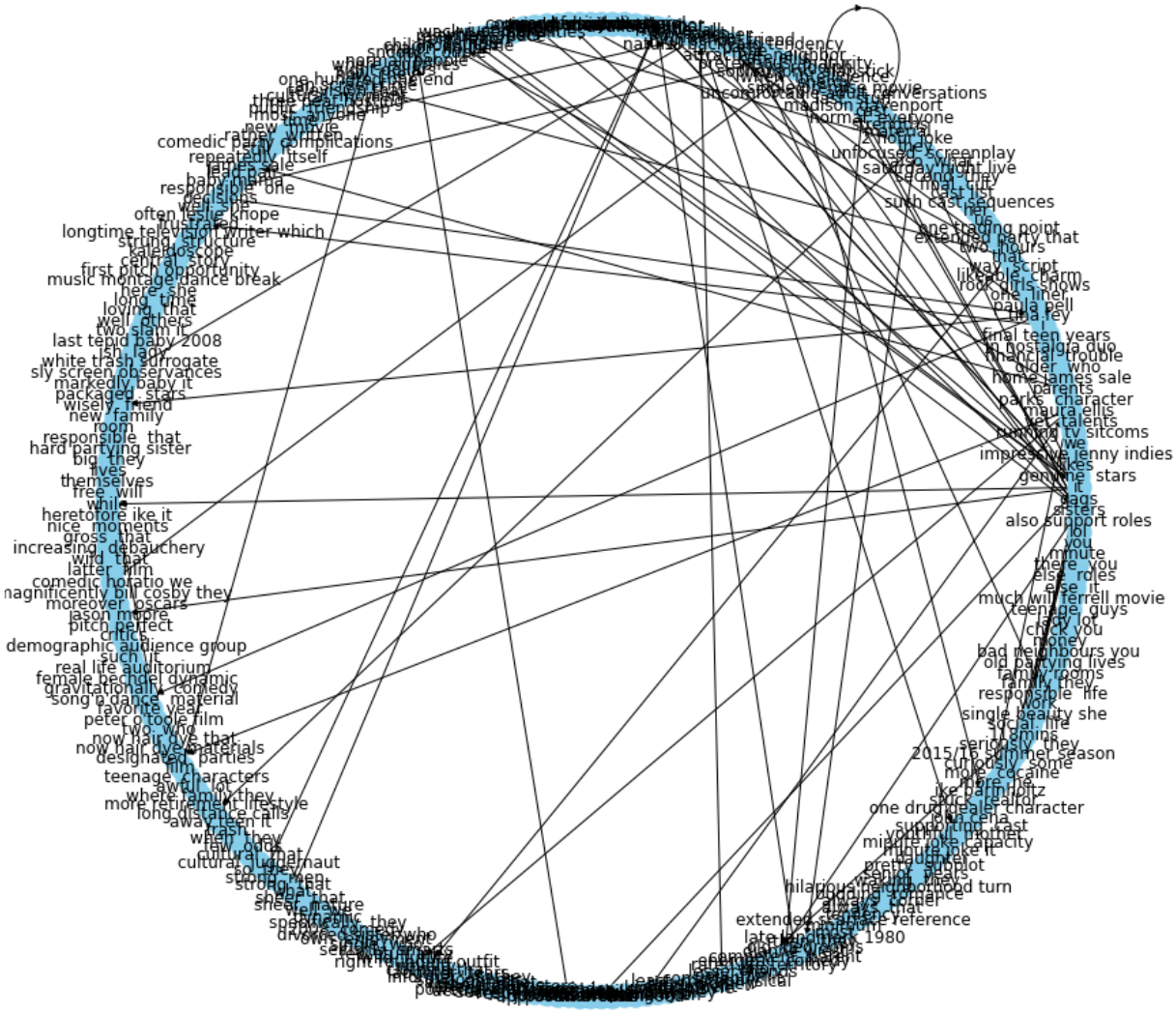
	source	edge	target
111	female bechdel dynamic	is	gravitationally comedy
112	tina fey	laughs with	song'n'dance material
113	favorite year	are	peter o'toole film
114	two who	play	lives
115	now hair dye that	makes	now hair dye materials
116	maura ellis	is	designated parties
117	how they	's	film
118	teenage characters	shows	awful lot
119	where family they	triggered	more retirement lifestyle
120	they	are in	long distance calls
121	away teen it	informed	trash
122	when they	kept	few odds
123	cultural that	is	cultural juggernaut
124	so they	see	them
125	strong men	entertain through	strong that
127	what	is	them
128	sheer that	feels refreshing	sheer nature
129	it	keep	jason moore
130	well we	know	dynamic
131	specifically they	is	2008 comedy
132	divorced sister who	is	own enjoyment
133	single who	is	selfishly smarts
134	few stuff they	learn	wild future
135	maura ellis	are in	right rounding outfit
136	that	's	novelty
137	clothing that	is	laughter tears
138	informer"-is they	routine	so goofiness
139	also core that	's	whole movie
140	little j. fox posters	serves as	powerful diary reminder
148	actually 40something	love	secretly resistance
149	responsible men	's	fun
151	juvenile men	is	this
152	positive that	be	approved womanhood

	source	edge	target
156	how this	sees	how women
157	how this	sees	it
160	40something men	got	stupid ways
161	titular who	are	maura ellis
162	childhood they	sold	even this
163	it	has	them
164	kindly movie	thought	entitlement
166	time women	get	least metaphysical
167	consistently it	's	amy poehler
168	only film	cringe at	lesser hands
169	who	does	rated life territory
170	sisters	is as rapid	energetic comedy
172	madison davenport	is impulsive	competent parent
173	dianne wiest	stop	them
174	they	're	old bedrooms
175	then they	invite	late landmark 1980
176	amy poehler	had	sisters
177	most	feels like	minimum
178	extended scarface reference	have	tendency
179	always that	's	always corner
180	budding romance	feels fresh	hilarious neighborhood turn
181	waking they	works in	senior years
182	pretty subplot	is	daughter
183	minute joke it	exists as	minute joke capacity
184	amy poehler	are	youthful mother
185	supporting cast	is	dianne wiest
186	john cena	gets	one drug dealer character
187	maya rudolph	gets	stuck realtor
188	ike barinholtz	keeps	maura ellis
189	more he	comes	more cocaine
190	sisters	run	curiously some
191	sisters	join	2015/16 summer season
192	seriously they	believe	sisters

	source	edge	target
194	it	runs for	118mins
195	who	is responsible	social life
196	single beauty she	is	work
197	she	has	responsible life
198	family they	announce	family rooms
199	sisters	were famous	old partying lives
200	bad neighbours you	think	money
201	chick you	is	lady lot
202	teenage guys	see teenage	much will ferrell movie
203	they	tear	it
205	else it	been horrendous	else roles
206	there you	are fast	minute
207	you	written	lol
208	you	is	also support roles

Plotting the Entire Knowledge Graph

In [51]:
```
plot_graph(kg_df_consolidated, use_circular=True)
```



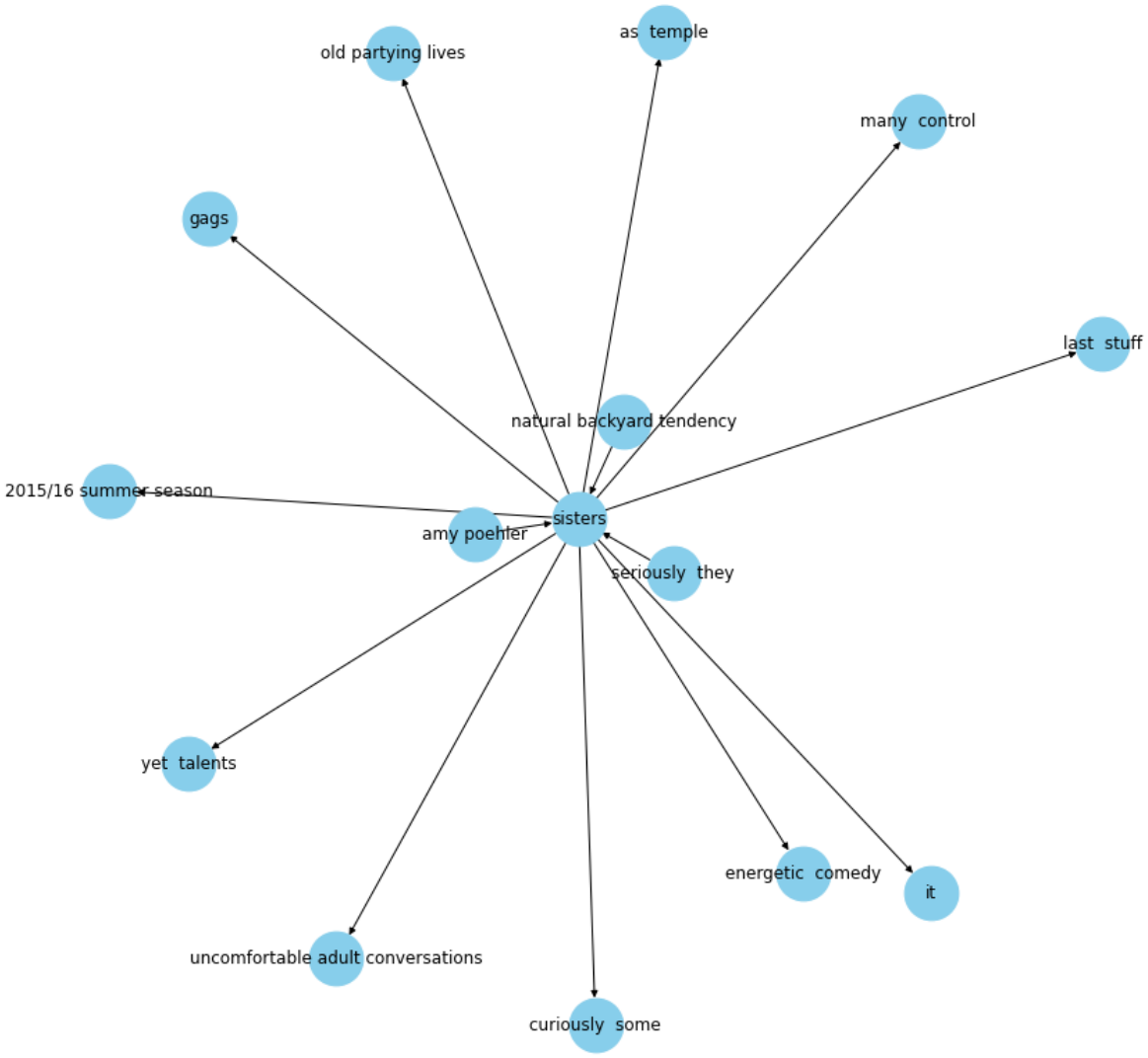
Plotting Knowledge Graph Subset 1

Let's examine the subset of our knowledge graph connected to nodes that correspond to the movie title, "Sisters."

In [52]:
```
RELATION_TO_EXPLORE = 'sisters'
```

In [53]:
```
G = nx.from_pandas_edgelist(kg_df_consolidated[(kg_df_consolidated['source'] == RELATI
                                    (kg_df_consolidated['target'] == RELATION_TO_EXPLORE
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
```

```
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```

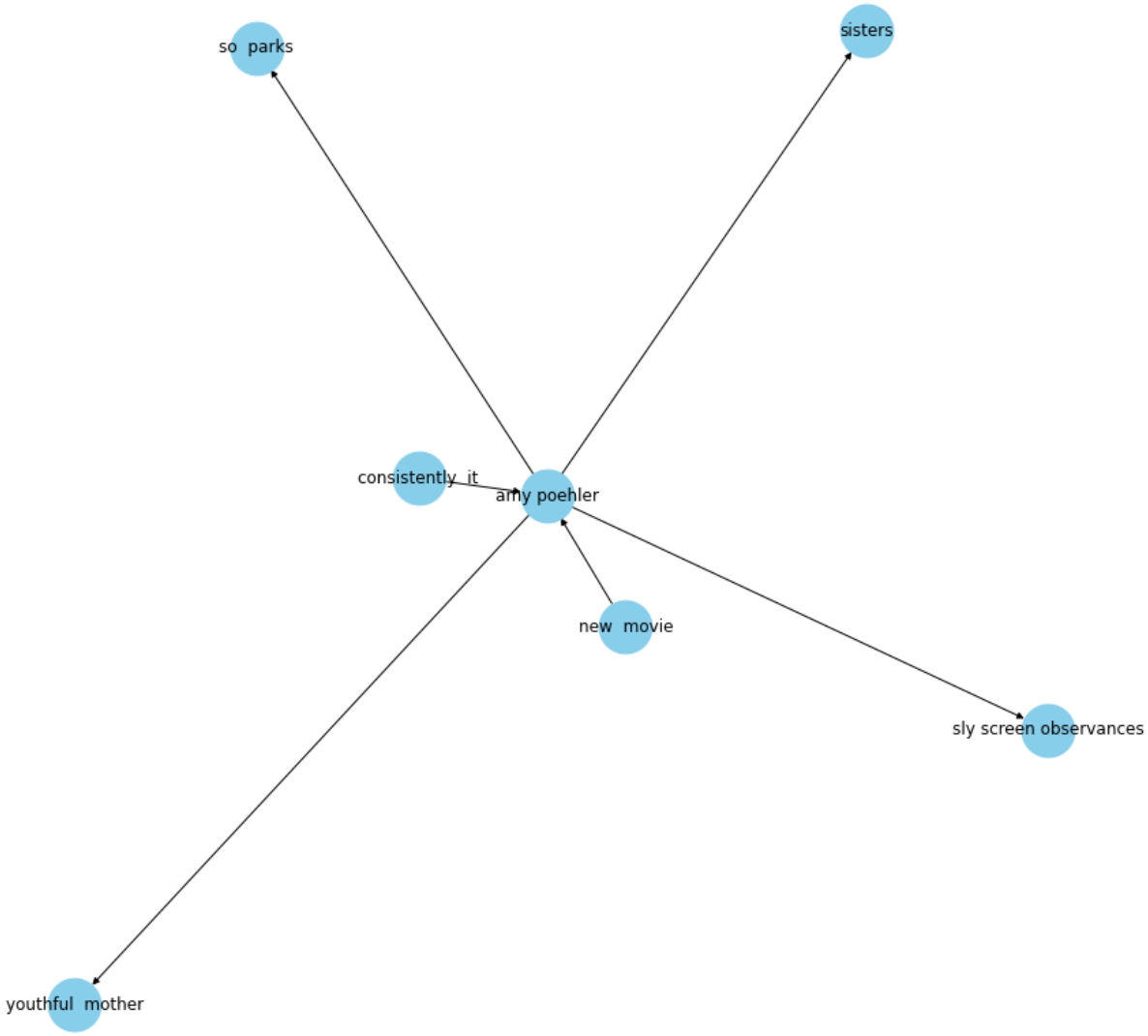


Plotting Knowledge Graph Subset 2

Let's examine the subset of our knowledge graph connected to nodes that correspond to one of the lead actresses, Amy Poehler.

In [54]:
```
G = nx.from_pandas_edgelist(kg_df_consolidated[(kg_df_consolidated['source'].str.conta
                                    (kg_df_consolidated['target'].str.contains("amy") ==
                                    (kg_df_consolidated['source'].str.contains("poehler"
                                    (kg_df_consolidated['target'].str.contains("poehler"
                                    ],
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())

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

```
pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```

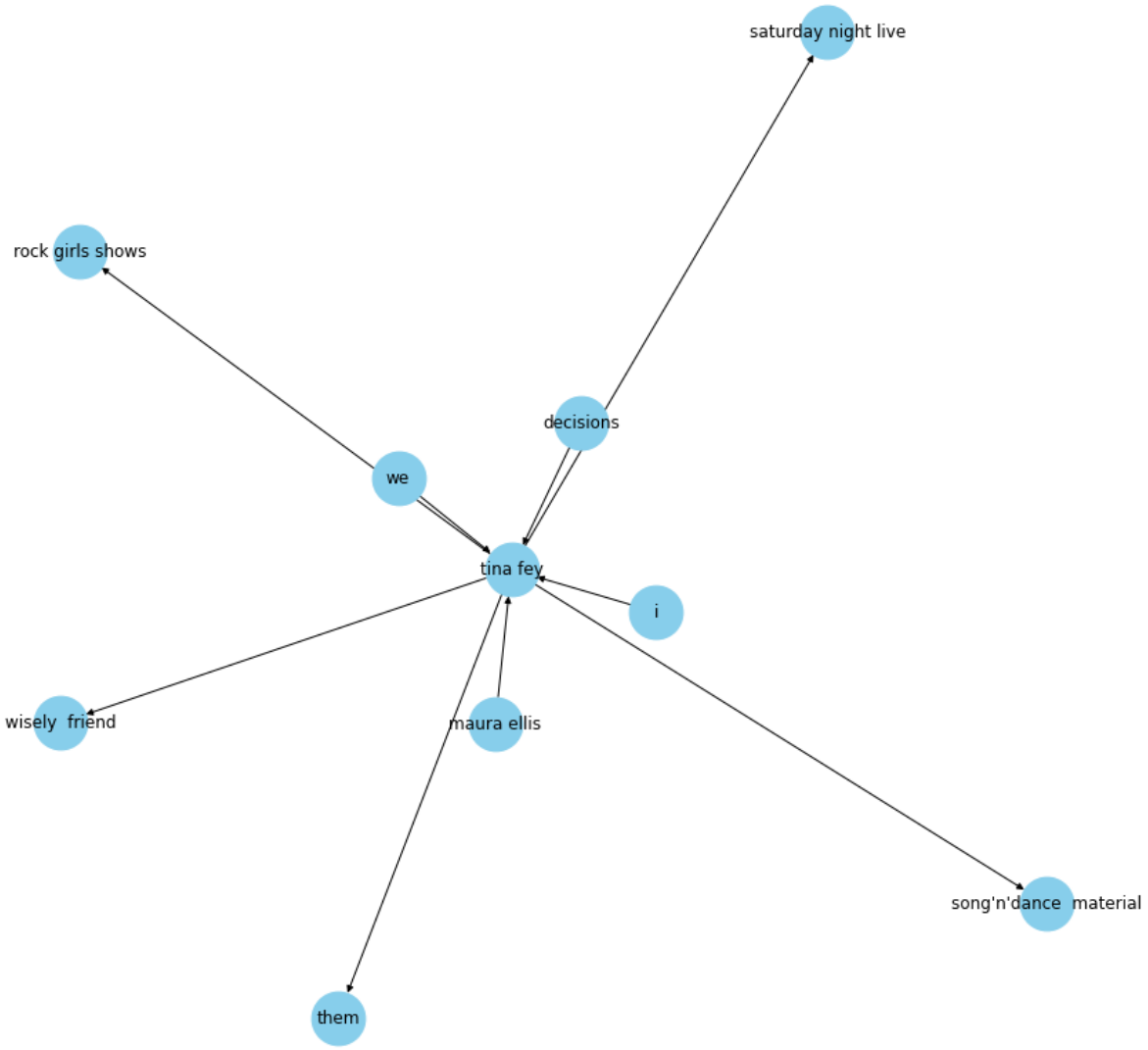


Plotting Knowledge Graph Subset 3

Let's examine the subset of our knowledge graph connected to nodes that correspond to one of the lead actresses, Tina Fey.

In [55]:
```
G = nx.from_pandas_edgelist(kg_df_consolidated[(kg_df_consolidated['source'].str.conta
                                    (kg_df_consolidated['target'].str.contains("tina") =
                                    (kg_df_consolidated['source'].str.contains("fey") ==
                                    (kg_df_consolidated['target'].str.contains("fey") ==
                                    ],
                            "source", "target",
                            edge_attr=True,
```

```
                              create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```

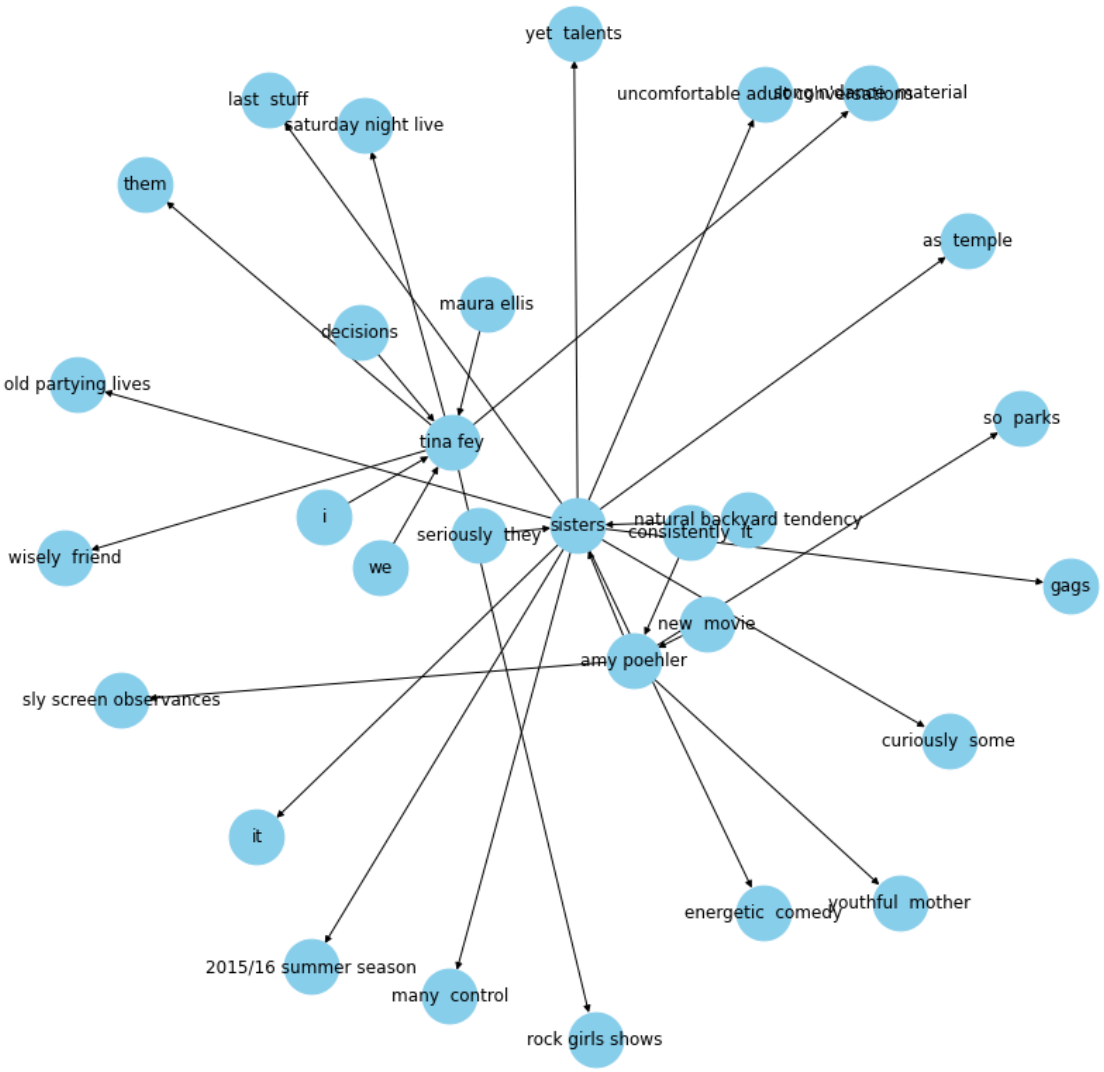


Plotting Knowledge Graph Subset 4

Let's examine the subset of our knowlege graph connected to nodes that correspond to Tina Fey, Amy Poehler, or "Sisters."

In [56]:
```
G = nx.from_pandas_edgelist(kg_df_consolidated[(kg_df_consolidated['source'].str.conta
                                    (kg_df_consolidated['target'].str.contains("tina") =
                                    (kg_df_consolidated['source'].str.contains("fey") ==
```

```
                                (kg_df_consolidated['target'].str.contains("fey") ==
                                (kg_df_consolidated['source'].str.contains("amy") ==
                                (kg_df_consolidated['target'].str.contains("amy") ==
                                (kg_df_consolidated['source'].str.contains("poehler"
                                (kg_df_consolidated['target'].str.contains("poehler"
                                (kg_df_consolidated['source'].str.contains("sisters"
                                (kg_df_consolidated['target'].str.contains("sisters"
                                ],
                            "source", "target",
                            edge_attr=True,
                            create_using=nx.MultiDiGraph())

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

pos = nx.spring_layout(G, k=0.5) # k regulates the distance between nodes

nx.draw(G, with_labels=True,
        node_color='skyblue',
        node_size=1500, edge_cmap=plt.cm.Blues, pos=pos)

plt.show()
```



2) Classification with Long Short-Term Memory (LSTM) Models

Let's create Long Short-Term Memory models that are capable of classifying movie reviews by genre and by movie review sentiment (i.e. positive or negative review).

2.1) Genre Classification with LSTM Models

2.1.1) Data Wrangling for Genre Classification LSTM Models

Let's conduct data wrangling and preprocessing as needed to format the corpus text for the genre classification LSTM models.

In [57]:
```
datafull=corpus_df.copy()
datafull.reset_index(drop=True, inplace=True)
datafull.head(4).T
```

Out[57]:

	0	1	2	
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Student Name	SAR	SAR	SAR	
Genre of Movie	Action	Action	Action	
Review Type (pos or neg)	Negative	Negative	Negative	
Movie Title	Covenant	Covenant	Covenant	
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...	In a weird those
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103	Action_Cov
Doc_ID	101	102	103	
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...	[In a we to those
lemmas	[nearly, year, american, military, afghanistan...	[guy, ritchie, jake, gyllenhaal, switch, place...	[guy, ritchie, covenant, notably, mark, featur...	[weird, thr action,
lemmas_joined	nearly year american military afghanistan end ...	guy ritchie jake gyllenhaal switch place peter...	guy ritchie covenant notably mark feature incl...	weird th acti
sentences_lemmatized	[nearly year american military afghanistan end...	[guy ritchie jake gyllenhaal switch place pete...	[guy ritchie covenant notably mark feature inc...	[weird th acti

In [58]:
```
datafull['Text'] = datafull['Text'].apply(lambda x :clean_doc(x))
```

In [59]:
```
data = datafull[['Text','Genre of Movie']].copy()
```

In [60]:
```
data['Genre of Movie'] = data['Genre of Movie'].astype("category")
data['Genre of Movie code'] = data['Genre of Movie'].cat.codes
data['Genre of Movie code'].tail().T
```

Out[60]:
```
185    3
186    3
187    3
188    3
189    3
Name: Genre of Movie code, dtype: int8
```

In [61]:
```
features, targets = data['Text'], data['Genre of Movie code']
data[["Genre of Movie","Genre of Movie code"]].value_counts()
```

Out[61]:
```
Genre of Movie  Genre of Movie code
Action          0                     50
Comedy          1                     50
Horror          2                     50
Sci-Fi          3                     40
dtype: int64
```

Let's create the training, validation, and testing datasets for construction of our LSTM models.

In [62]:
```
trainds, valds, testds = get_dataset_partitions_pd(data[['Text','Genre of Movie code']
trainds.shape, valds.shape, testds.shape
```

Out[62]:
```
((152, 2), (19, 2), (19, 2))
```

Let's convert the dataframe to a TensorFlow dataset.

In [63]:
```
# train X & y
train_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(trainds['Text'].values, tf.string)
)
train_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(trainds['Genre of Movie code'].values, tf.int64),
  )
# test X & y
test_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(testds['Text'].values, tf.string)
)
test_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(testds['Genre of Movie code'].values, tf.int64),
)
#val X & Y
val_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(valds['Text'].values, tf.string)
)
val_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(valds['Genre of Movie code'].values, tf.int64),
)
```

Create Datasets (X=Preprocessed Text, Y=Encoded Categories)

In [64]:
```
train_ds = tf.data.Dataset.zip(
    (
            train_text_ds_raw,
            train_cat_ds_raw
     )
)
test_ds = tf.data.Dataset.zip(
    (
            test_text_ds_raw,
            test_cat_ds_raw
     )
)
val_ds = tf.data.Dataset.zip(
    (
            val_text_ds_raw,
            val_cat_ds_raw
     )
)
```

Create Data Pipelines (Batching, Shuffling, and Optimizing)

In [65]:
```
batch_size = 3
AUTOTUNE = tf.data.experimental.AUTOTUNE
buffer_size=train_ds.cardinality().numpy()

train_ds = train_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)

test_ds = test_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)


val_ds = val_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)
```

In [66]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 190 entries, 0 to 189
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype   
---  ------               --------------  -----   
 0   Text                 190 non-null    object  
 1   Genre of Movie       190 non-null    category
 2   Genre of Movie code  190 non-null    int8    
dtypes: category(1), int8(1), object(1)
memory usage: 2.2+ KB
```

Create The Text Encoder

The raw text loaded by `tfds` needs to be processed before it can be used in a model. The simplest way to process text for training is using the `experimental.preprocessing.TextVectorization` layer. This layer has many capabilities, but this tutorial sticks to the default behavior.

Create the layer, and pass the dataset's text to the layer's `.adapt` method: The processing of each sample contains the following steps:

```
standardize each sample (usually lowercasing + punctuation stripping)
split each sample into substrings (usually words)
recombine substrings into tokens (usually ngrams)
index tokens (associate a unique int value with each token)
transform each sample using this index, either into a vector of ints
or a dense float vector.
```

In [67]:
```
VOCAB_SIZE=5000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE, standardize="lower_and_strip_punctuation", pad_to_max_toker
encoder.adapt(train_ds.map(lambda text, label: text), batch_size= None)
```

In [68]:
```
vocab = np.array(encoder.get_vocabulary())
len(vocab)
```

Out[68]:
```
5000
```

In [69]:
```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

Out[69]:
```
array(['', '[UNK]', 's', 'the', 'film', 'i', 'movie', 'nt', 'one', 'like',
       'it', 'get', 'time', 'character', 'even', 'two', 'but', 'make',
       'first', 'much'], dtype='<U30')
```

In [70]:
```
vocab[-20:]
```

Out[70]:
```
array(['victorianera', 'victorian', 'victimized', 'vicious', 'vice',
       'vicarious', 'vibrate', 'vibe', 'viable', 'vessel', 'verve',
       'vertigo', 'vertically', 'vertical', 'veritable',
       'verhoevenstyled', 'verdant', 'verbatim', 'vera', 'venturing'],
      dtype='<U30')
```

View Examples of Encoded Words

In [71]:
```
encoded_example = encoder('encanto we dont talk about bruno no no').numpy()
encoded_example[:]
```

Out[71]:
```
array([   1,  218,    1,  617, 2393,    1,  139,  139], dtype=int64)
```

In [72]:
```
len(encoder.get_vocabulary())
```

Out[72]:
```
5000
```

In [73]:
```
encoder(data['Text'][0])
```

Out[73]:
```
<tf.Tensor: shape=(315,), dtype=int64, numpy=
array([ 365,   15,   26,  395,  629, 3199,  484,  958, 2773,  192,   57,
        752,   41,  572,  439, 1620,   51,   66,  870,  202,  509,   89,
          2, 3129,   64, 4259,   61,  122,   89,   62,  104,    8,  518,
        183,  100,  572, 4087,  104,  132,    8,  103,    8,  976,  272,
       3401,   35, 2418, 1690,    1, 1118, 1884,    1,    1,    1,  139,
        159,   16,  547,  192,  690,    3,  264, 1172,  834,  276,   85,
        192,  484,    2,  235,   67,   45,   23,  581,   62,  104,   23,
       2477,   62,  104,   23, 4309,   62,  104,  125,    1,   62,  104,
         41,  177,  887,  100,  919,   67,  494,  233,   11, 4553,   76,
          1,  499,    4,    8,   92,  419,   14,   53,  564,  646,    9,
          6,  104,    2,  669, 1964,  117,    1,  230,  246,  504, 4483,
       1527, 1084,    4, 4667, 1107,  317,  419,   24,  518,  726,   76,
        457,  483,  247,  484, 4711,   57,  629,   15,  692,   29, 1264,
        799,   86, 1387, 1926, 1573,   56, 2449,  196,  238,  192,    1,
       1481,  197, 1167, 1385,    3,  264,  104,  798,  707,  478, 1239,
       1867, 2102, 2284,  683,   57,  483, 1641,  101,  106,  105,  276,
         32,  232, 1174,  797, 1339,  756,    1,  285, 1532,    1,   10,
          2,  767,   27,  470,  369,   31,  593,  106,    2,  457, 1174,
        159,   46,  247, 1091,   83,  854,  532,  236,    1,  247,   32,
        621,   83,   32,   20,  917, 1706, 4090,  989,   10,    2, 3290,
       1284,   25,  176, 1695,    2,   20,  954,    8,  104,    2,    4,
       3676,  260,  448,  874,  301,   10,  176,  676, 3709,   86, 1133,
        247,  196,  238,   57, 4839, 1940,   25,   50,  561, 2561,    3,
        264,    1,  542,   24,    2, 2043, 1699,  419,    9,  276,    2,
       1641,  106,  532,    2,   83,  720, 1985,  452,  113,   62,    1,
       3031,    1,   64, 4323,    1,  114,   62,   64,    1,   64, 4924,
        892,  113,  246,  191, 2854, 4478,    8,   82,    1, 1264, 1233,
       2715, 1985, 2616,    2, 2468,    1,   16,   11,   18,  328,  470,
       1005,    8,   82,  773, 1385,   18,  257], dtype=int64)>
```

In [74]:
```
for example, label in train_ds.take(1):
  print('texts: ', example.numpy()[:1])
  print()
```

```
texts:  [b"There 's recurring line No Time Die - Daniel Craig 's fifth final outing J
ames Bond - 's heartbreakingly prescient . Now retired life MI6 superspy , Bond say l
adylove Madeleine Swann ( Lea Seydoux ) , met last film , Spectre , `` All time world
. `` All time heal past wound , time write different ending . Of course , suave agent
legion fan embraced Craig 's 007 , mantra nothing wishful thinking . Maybe would bett
er line ending 2015 's Spectre , Bond Swann drove sunset Craig mused press 'd rather
`` slash wrist `` return character . Then could believed fanciful happy ending . As t
urn , idyllic life mirage . After opening tie film 's main villain Lyutsifer Safin (
Rami Malek ) Swann , action cut mountain road Italy Bond Madeleine race towards happi
ly-ever-after . But tranquillity short-lived SPECTRE agent hunt . How find easily ? B
ond immediately think 's betrayed , extended chase scene , unceremoniously chuck Made
leine train . No time goodbye . The action pick five year later , old pal earlier fil
m , CIA handler Felix Leiter ( Jeffrey Wright ) , interrupt Bond 's single life Jamai
ca ask help tracking missing scientist ( David Dencik ) behind dangerous nanotech , n
icknamed Heracles . As story ( director Cary Fukunaga , Neal Purvis , Robert Wade Pho
ebe Waller-Bridge ) continues Cuba , Bond team CIA newbie Paloma ( delightful Ana de
Armas ) cross path 00 replacement ( Lashana Lynch ) . Turns , mission n't simple snat
ch-and-grab . Bond 's double-crossed know , 's back front old nemesis Ernst Stavro Bl
ofeld ( Christoph Waltz ) , locked Belmarsh Prison . And oh look , 's Madeleine . It
turn 's one Blofeld talk . The mysterious Heracles project , old bos M ( Ralph Fienne
s ) hand creating , sends Bond Madeleine path life irrevocably changed Safin . To say
, would venture spoiler territory . But Fukunaga - first American-born filmmaker helm
Bond picture - manages tie many plot point together , bringing Craig 's run wholly sa
tisfying end . Unlike Pierce Brosnan 's film , Bond story require investment part vie
wer . You need know happened entry one full import denouement hit home . Craig 's Bon
d hero bruised year loss betrayal , see ramification . And"]
```

2.1.2) Genre Classification LSTM Model 1

Let's first create LSTM models that are capable of classifying movie reviews into one of four genres: 1) horror, 2) science fiction, 3) comedy, or 4) action.

Build the LSTM Model. For more information about layers in keras, refer to this resource: https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [75]:
```
k.clear_session()
num_classes=4
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True, dro
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,dropout=0.3))
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [76]:
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [77]:
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 32s 357ms/step - loss: 1.3828 - accuracy: 0.
2733 - val_loss: 1.4026 - val_accuracy: 0.1111
Epoch 2/200
50/50 [==============================] - 12s 238ms/step - loss: 1.1179 - accuracy: 0.
4933 - val_loss: 1.1739 - val_accuracy: 0.3333
Epoch 3/200
50/50 [==============================] - 11s 217ms/step - loss: 0.6095 - accuracy: 0.
7667 - val_loss: 0.8323 - val_accuracy: 0.6667
Epoch 4/200
50/50 [==============================] - 10s 206ms/step - loss: 0.2071 - accuracy: 0.
9533 - val_loss: 0.8820 - val_accuracy: 0.7778
Epoch 5/200
50/50 [==============================] - 10s 210ms/step - loss: 0.0813 - accuracy: 0.
9800 - val_loss: 0.8114 - val_accuracy: 0.8889
Epoch 6/200
50/50 [==============================] - 10s 203ms/step - loss: 0.0464 - accuracy: 0.
9933 - val_loss: 0.4331 - val_accuracy: 0.7778
Epoch 7/200
50/50 [==============================] - 10s 201ms/step - loss: 0.0088 - accuracy: 1.
0000 - val_loss: 1.1739 - val_accuracy: 0.6667
CPU times: total: 2min 14s
Wall time: 1min 35s
```

View a Summary of the LSTM Model Architecture

In [78]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 431,684
Trainable params: 431,684
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [79]:
```
def plot_confusion_matrix_labeled(y_true, y_pred, CLASSES_LIST):
    mtx = confusion_matrix(y_true, y_pred)
    # define classes
    classes = CLASSES_LIST
```

```
    temp_df = pd.DataFrame(data=mtx,columns=classes)
    temp_df.index = classes
    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(temp_df, annot=True, fmt='d', linewidths=.75,  cbar=False, ax=ax,cmap=
    #  square=True,
    plt.ylabel('true label')
    plt.xlabel('predicted label')

def plot_graphs(history, metric):
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])

def display_training_curves(training, validation, title, subplot):
  ax = plt.subplot(subplot)
  ax.plot(training)
  ax.plot(validation)
  ax.set_title('model '+ title)
  ax.set_ylabel(title)
  ax.set_xlabel('epoch')
  ax.legend(['training', 'validation'])
```

In [80]:
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 0s 40ms/step - loss: 1.1863 - accuracy: 0.7222
Test Loss: 1.1862927675247192
Test Accuracy: 0.7222222089767456
```

In [81]:
```
history_dict = history.history
history_dict.keys()
```

Out[81]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

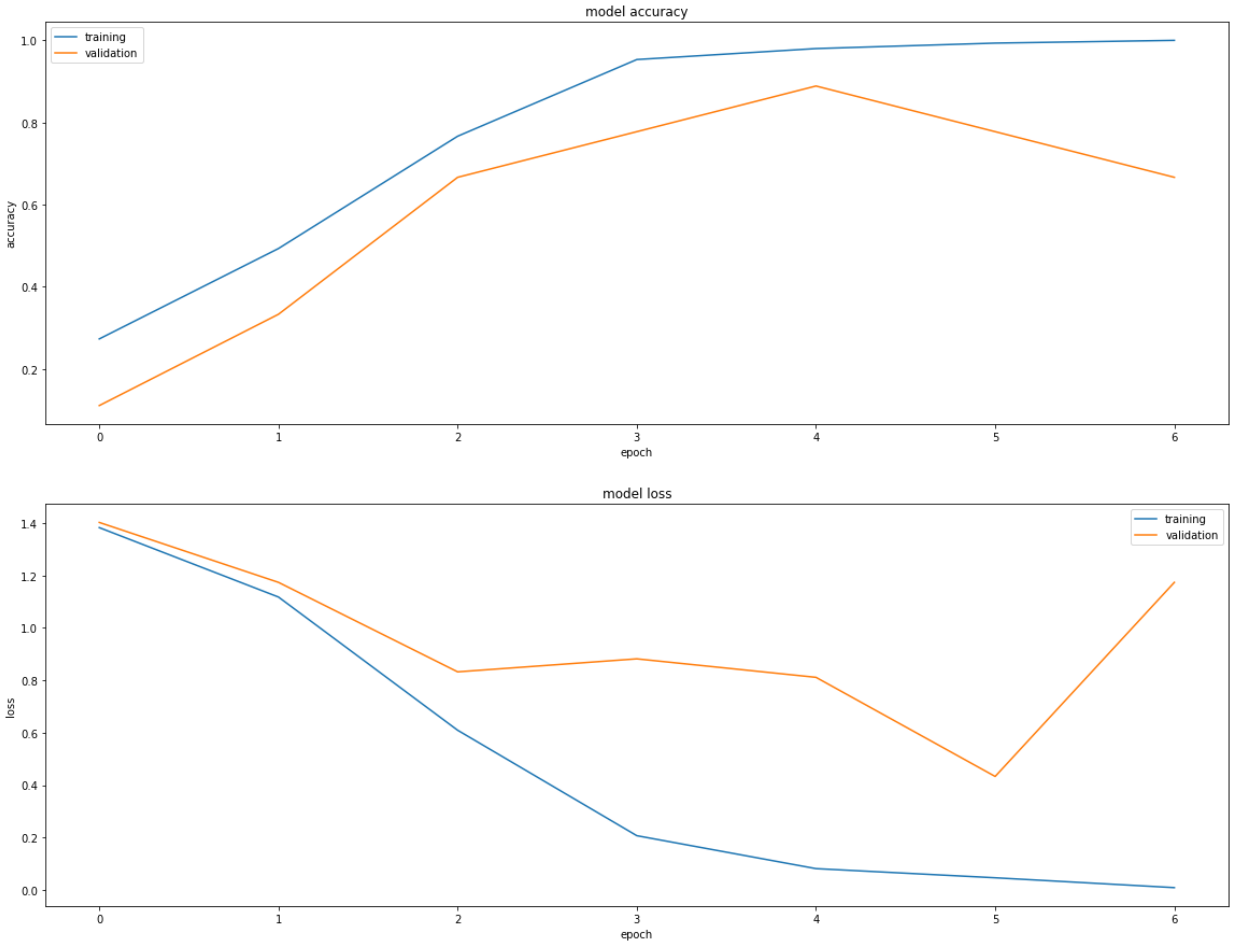
In [82]:
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

Out[82]:

	loss	accuracy	val_loss	val_accuracy
2	0.610	0.767	0.832	0.667
3	0.207	0.953	0.882	0.778
4	0.081	0.980	0.811	0.889
5	0.046	0.993	0.433	0.778
6	0.009	1.000	1.174	0.667

In [83]:
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [84]:
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```



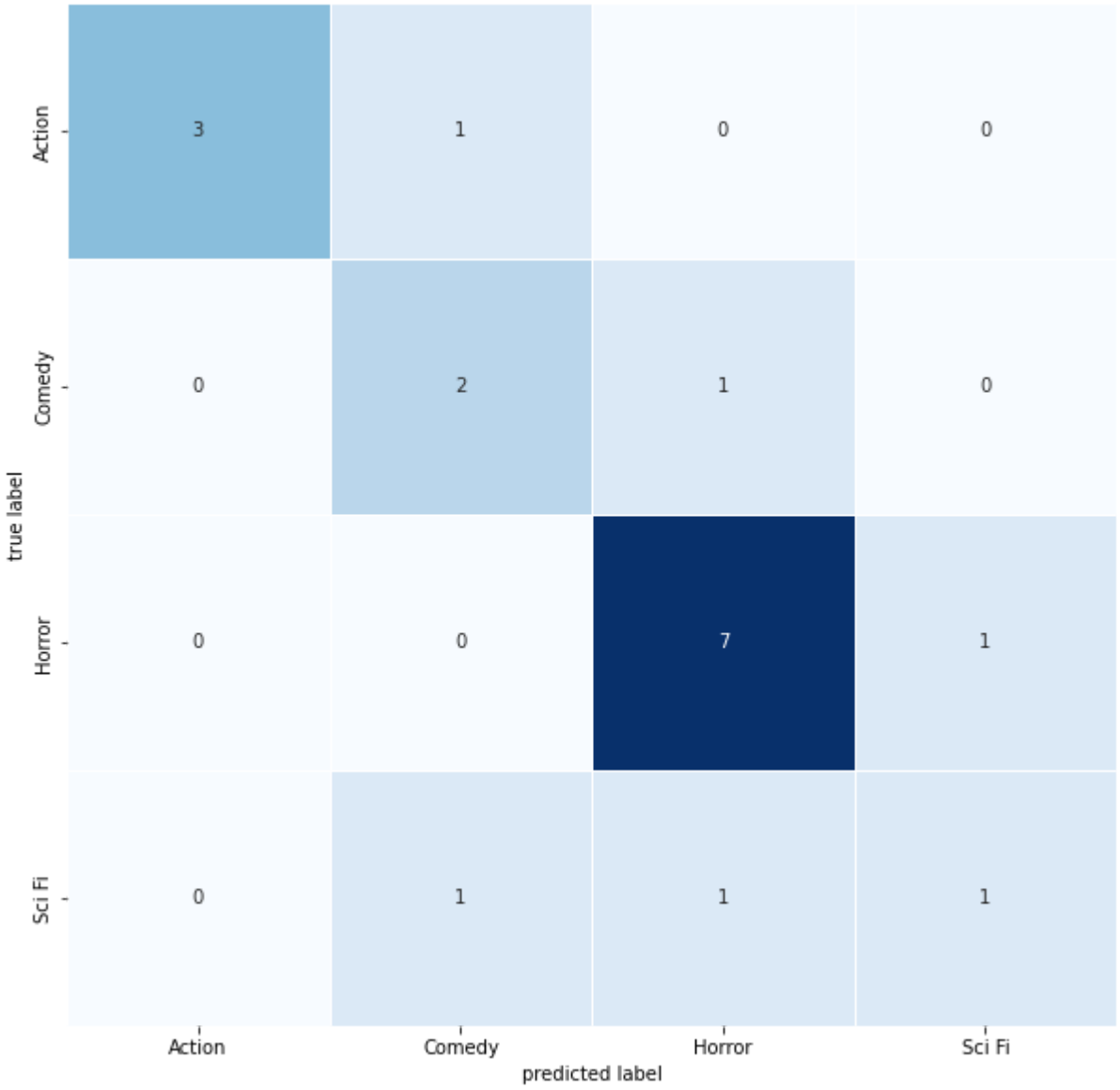
Assess the Model's Performance Using Confusion Matrix

In [85]:
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 6s 57ms/step
```

In [86]:
```
CLASSES_LIST = ['Action','Comedy','Horror','Sci Fi']
```

In [87]:
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [88]:
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [89]:
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

Out[89]:

	0	1	2	3	4	5	6	7	8	9	10
Action	0.28%	98.94%	0.56%	0.46%	0.28%	95.47%	0.03%	0.08%	96.44%	0.26%	0.26%
Comedy	0.52%	0.04%	0.40%	0.25%	0.43%	0.41%	99.68%	98.10%	1.15%	80.67%	0.30%
Horror	93.84%	1.02%	96.47%	97.81%	95.11%	3.88%	0.17%	0.87%	2.35%	16.22%	96.51%
Sci Fi	5.36%	0.00%	2.57%	1.48%	4.18%	0.24%	0.13%	0.94%	0.07%	2.84%	2.92%

2.1.3) Genre Classification LSTM Model 2

Build the LSTM Model. For more information about layers in keras, refer to this resource: https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [90]:
```
k.clear_session()
num_classes=4
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 128, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,  return_sequences=True, dr
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,dropout=0.3))
   ,tf.keras.layers.Dense(128, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [91]:
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [92]:
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 35s 424ms/step - loss: 1.3759 - accuracy: 0.
3267 - val_loss: 1.3297 - val_accuracy: 0.2222
Epoch 2/200
50/50 [==============================] - 20s 397ms/step - loss: 0.8513 - accuracy: 0.
6333 - val_loss: 1.6023 - val_accuracy: 0.5556
Epoch 3/200
50/50 [==============================] - 18s 362ms/step - loss: 0.5107 - accuracy: 0.
7933 - val_loss: 1.3450 - val_accuracy: 0.4444
Epoch 4/200
50/50 [==============================] - 17s 329ms/step - loss: 0.2874 - accuracy: 0.
8800 - val_loss: 0.6764 - val_accuracy: 0.7778
Epoch 5/200
50/50 [==============================] - 17s 332ms/step - loss: 0.0511 - accuracy: 0.
9933 - val_loss: 0.3430 - val_accuracy: 0.8889
Epoch 6/200
50/50 [==============================] - 16s 328ms/step - loss: 0.0033 - accuracy: 1.
0000 - val_loss: 0.7137 - val_accuracy: 0.7778
Epoch 7/200
50/50 [==============================] - 16s 319ms/step - loss: 0.0016 - accuracy: 1.
0000 - val_loss: 0.4192 - val_accuracy: 0.8889
CPU times: total: 5min 33s
Wall time: 2min 18s
```

View a Summary of the LSTM Model Architecture

In [93]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 128)         640000

 bidirectional (Bidirectiona  (None, None, 256)        263168
 l)

 bidirectional_1 (Bidirectio  (None, 128)              164352
 nal)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 1,084,548
Trainable params: 1,084,548
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [94]:
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 0s 63ms/step - loss: 0.6458 - accuracy: 0.8333
Test Loss: 0.6458274722099304
Test Accuracy: 0.8333333134651184
```

In [95]:
```
history_dict = history.history
history_dict.keys()
```

Out[95]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [96]:
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

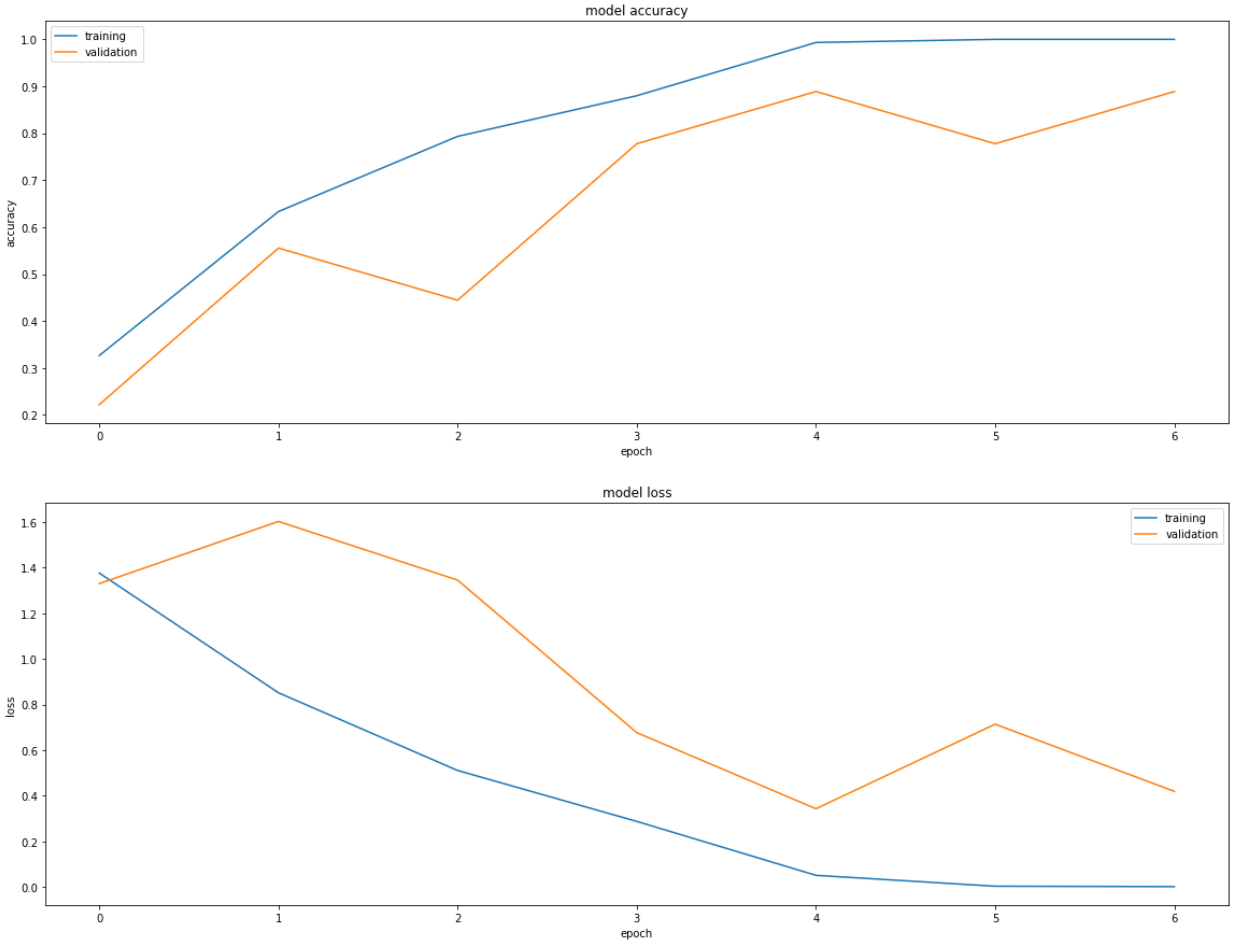
Out[96]:

	loss	accuracy	val_loss	val_accuracy
2	0.511	0.793	1.345	0.444
3	0.287	0.880	0.676	0.778
4	0.051	0.993	0.343	0.889
5	0.003	1.000	0.714	0.778
6	0.002	1.000	0.419	0.889

In [97]:
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
```

```
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [98]:
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```



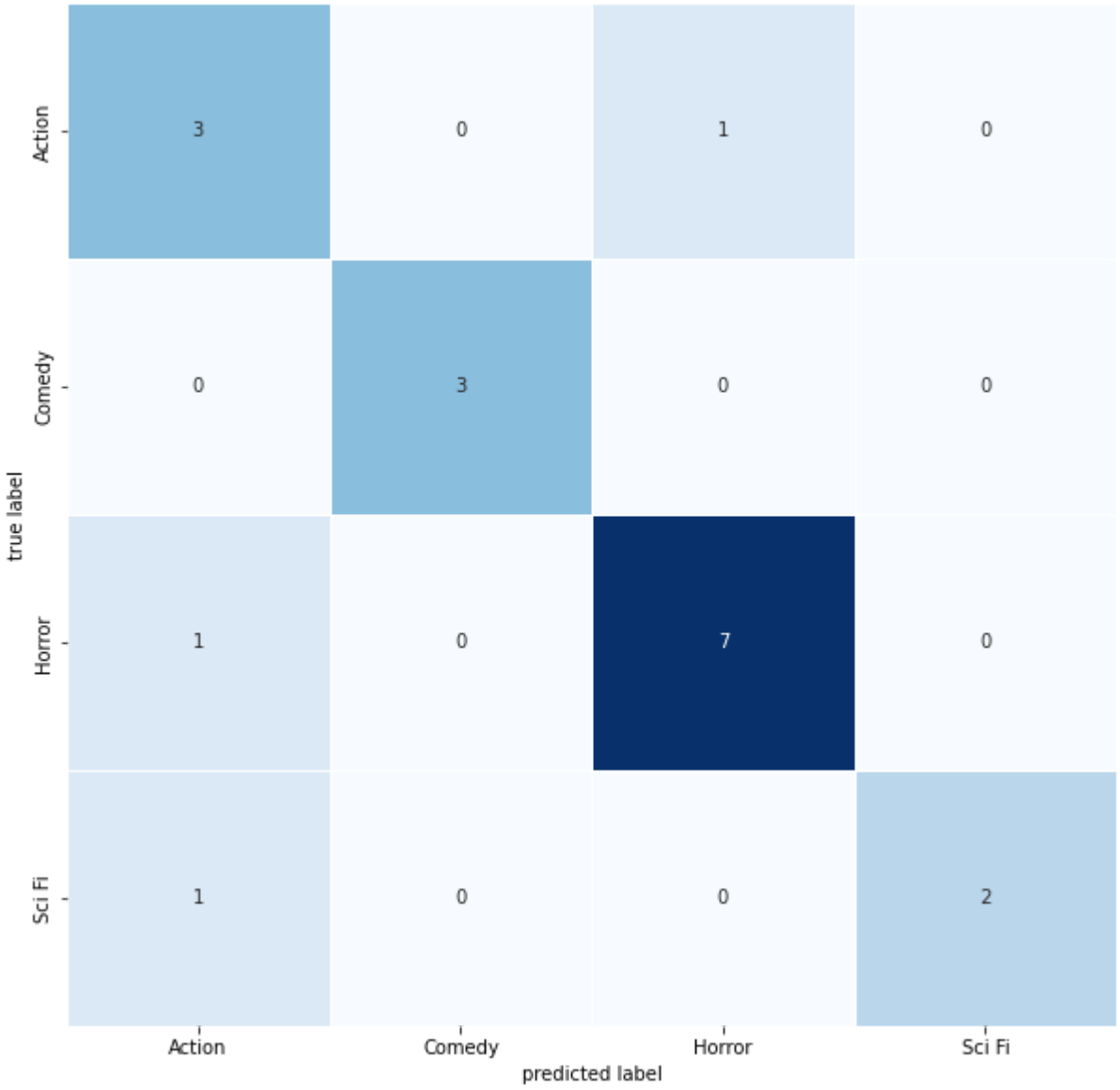
Assess the Model's Performance Using Confusion Matrix

In [99]:
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 5s 61ms/step
```

In [100…
```
CLASSES_LIST = ['Action','Comedy','Horror','Sci Fi']
```

In [101…
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [102…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [103…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

Out[103]:

	0	1	2	3	4	5	6	7	8	9	10
Action	0.30%	98.98%	0.77%	17.14%	1.98%	99.65%	60.41%	64.68%	13.13%	0.96%	2.10%
Comedy	0.01%	0.05%	0.03%	5.48%	0.04%	0.00%	2.07%	1.30%	0.28%	97.82%	0.03%
Horror	99.66%	0.61%	99.08%	54.85%	97.88%	0.29%	35.29%	24.10%	86.38%	0.57%	97.81%
Sci Fi	0.03%	0.35%	0.12%	22.53%	0.10%	0.06%	2.23%	9.92%	0.20%	0.65%	0.07%

2.1.4) Genre Classification LSTM Model 3

Build the LSTM Model. For more information about layers in keras, refer to this resource: https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [104…
```
k.clear_session()
num_classes=4
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True, dro
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,dropout=0.3))
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(32, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [105…
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [106…
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 37s 309ms/step - loss: 1.3848 - accuracy: 0.
2667 - val_loss: 1.3794 - val_accuracy: 0.4444
Epoch 2/200
50/50 [==============================] - 12s 238ms/step - loss: 1.2687 - accuracy: 0.
4800 - val_loss: 1.1543 - val_accuracy: 0.4444
Epoch 3/200
50/50 [==============================] - 14s 282ms/step - loss: 0.5481 - accuracy: 0.
8667 - val_loss: 1.3439 - val_accuracy: 0.4444
CPU times: total: 1min 7s
Wall time: 1min 2s
```

View a Summary of the LSTM Model Architecture

In [107…
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 4)                 132

=================================================================
Total params: 433,636
Trainable params: 433,636
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [108…
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 0s 47ms/step - loss: 0.9164 - accuracy: 0.5556
Test Loss: 0.9163747429847717
Test Accuracy: 0.5555555820465088
```

In [109…
```
history_dict = history.history
history_dict.keys()
```

Out[109]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [110…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

Out[110]:

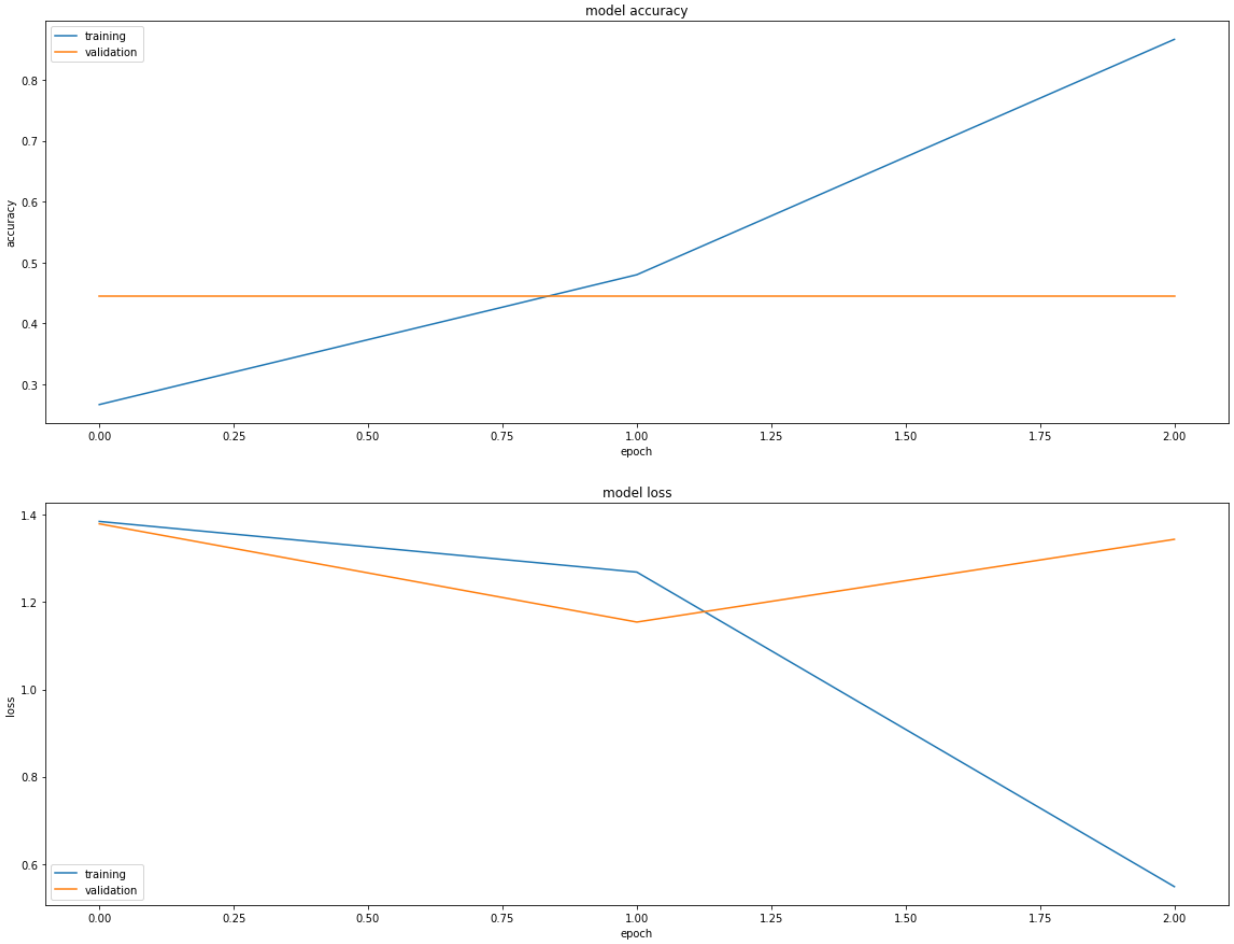
	loss	accuracy	val_loss	val_accuracy
0	1.385	0.267	1.379	0.444
1	1.269	0.480	1.154	0.444
2	0.548	0.867	1.344	0.444

In [111…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
```

```
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [112…

```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```



Assess the Model's Performance Using Confusion Matrix

In [113…

```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

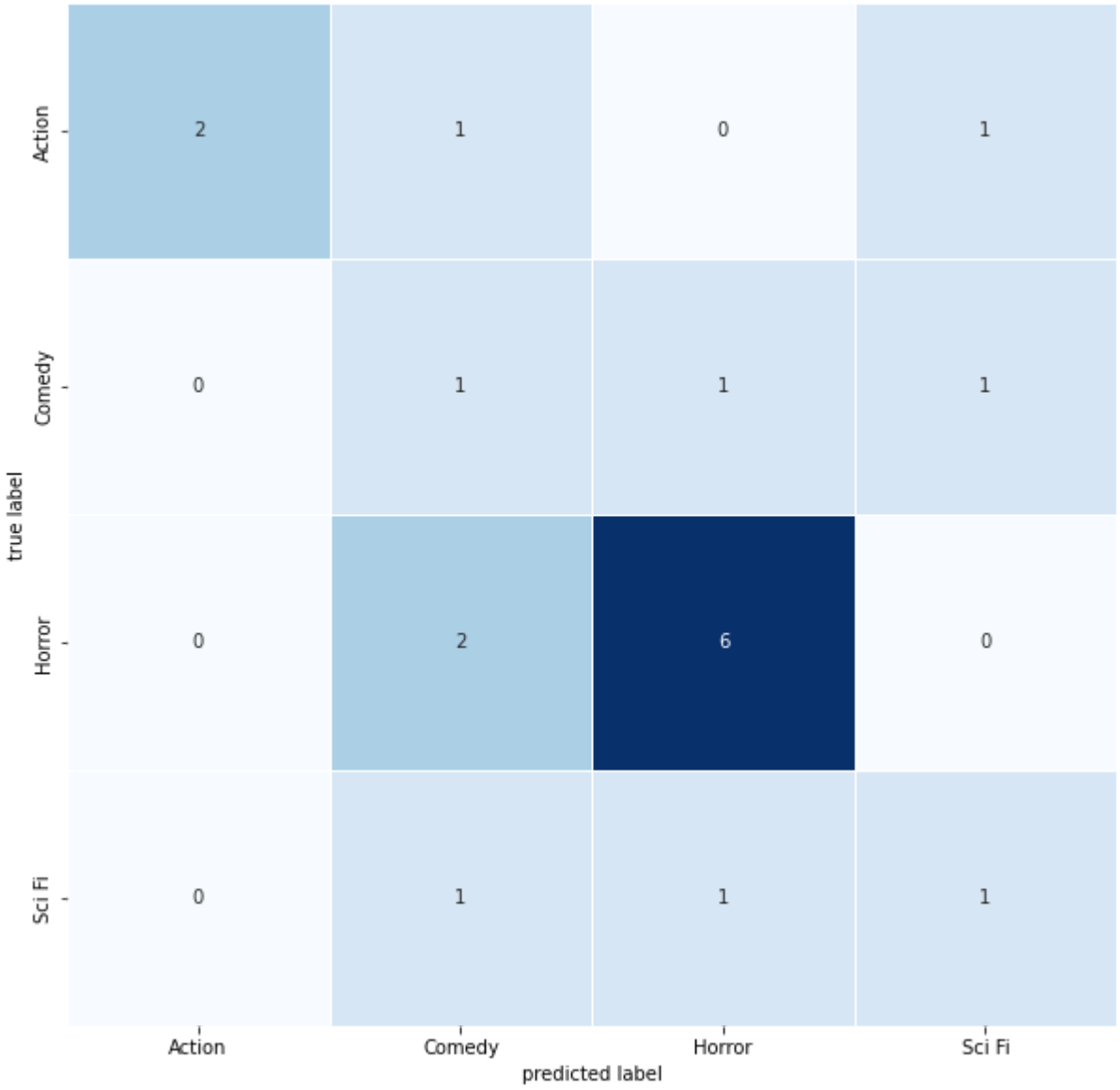
```
6/6 [==============================] - 4s 42ms/step
```

In [114…

```
CLASSES_LIST = ['Action','Comedy','Horror','Sci Fi']
```

In [115…

```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [116…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [117…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

Out[117]:

	0	1	2	3	4	5	6	7	8	9	10
Action	2.57%	35.11%	2.24%	2.45%	3.08%	81.61%	11.82%	81.55%	6.13%	3.31%	2.59%
Comedy	22.68%	10.73%	17.84%	75.69%	8.46%	6.28%	51.68%	8.48%	64.49%	71.42%	15.34%
Horror	51.98%	4.50%	69.58%	19.14%	57.22%	0.86%	15.86%	0.32%	22.51%	22.12%	66.08%
Sci Fi	22.77%	49.67%	10.35%	2.72%	31.24%	11.25%	20.64%	9.64%	6.87%	3.15%	15.99%

2.1.5) Genre Classification LSTM Model 4

Build the LSTM Model. For more information about layers in keras, refer to this resource: https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [118…
```
k.clear_session()
num_classes=4
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 128, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,  return_sequences=True, dr
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,dropout=0.3))
   ,tf.keras.layers.Dense(128, activation='relu')
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [119…
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [120…
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 71s 961ms/step - loss: 1.3888 - accuracy: 0.
2400 - val_loss: 1.3908 - val_accuracy: 0.0000e+00
Epoch 2/200
50/50 [==============================] - 52s 1s/step - loss: 1.3975 - accuracy: 0.346
7 - val_loss: 1.3623 - val_accuracy: 0.3333
Epoch 3/200
50/50 [==============================] - 56s 1s/step - loss: 1.0559 - accuracy: 0.573
3 - val_loss: 1.7424 - val_accuracy: 0.1111
Epoch 4/200
50/50 [==============================] - 49s 980ms/step - loss: 0.8645 - accuracy: 0.
6333 - val_loss: 1.1486 - val_accuracy: 0.4444
Epoch 5/200
50/50 [==============================] - 46s 930ms/step - loss: 0.5392 - accuracy: 0.
8267 - val_loss: 1.4865 - val_accuracy: 0.4444
Epoch 6/200
50/50 [==============================] - 61s 1s/step - loss: 0.1005 - accuracy: 0.973
3 - val_loss: 0.5741 - val_accuracy: 0.7778
Epoch 7/200
50/50 [==============================] - 53s 1s/step - loss: 0.0273 - accuracy: 0.993
3 - val_loss: 0.6362 - val_accuracy: 0.6667
Epoch 8/200
50/50 [==============================] - 51s 1s/step - loss: 0.0643 - accuracy: 0.986
7 - val_loss: 0.8795 - val_accuracy: 0.6667
CPU times: total: 3min 7s
Wall time: 7min 19s
```

View a Summary of the LSTM Model Architecture

In [121…
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 128)         640000

 bidirectional (Bidirectiona  (None, None, 256)        263168
 l)

 bidirectional_1 (Bidirectio  (None, 128)              164352
 nal)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 4)                 260

=================================================================
Total params: 1,092,548
Trainable params: 1,092,548
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [122…
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 2s 331ms/step - loss: 1.4488 - accuracy: 0.555
6
Test Loss: 1.4487601518630981
Test Accuracy: 0.5555555820465088
```

In [123…
```
history_dict = history.history
history_dict.keys()
```

Out[123]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

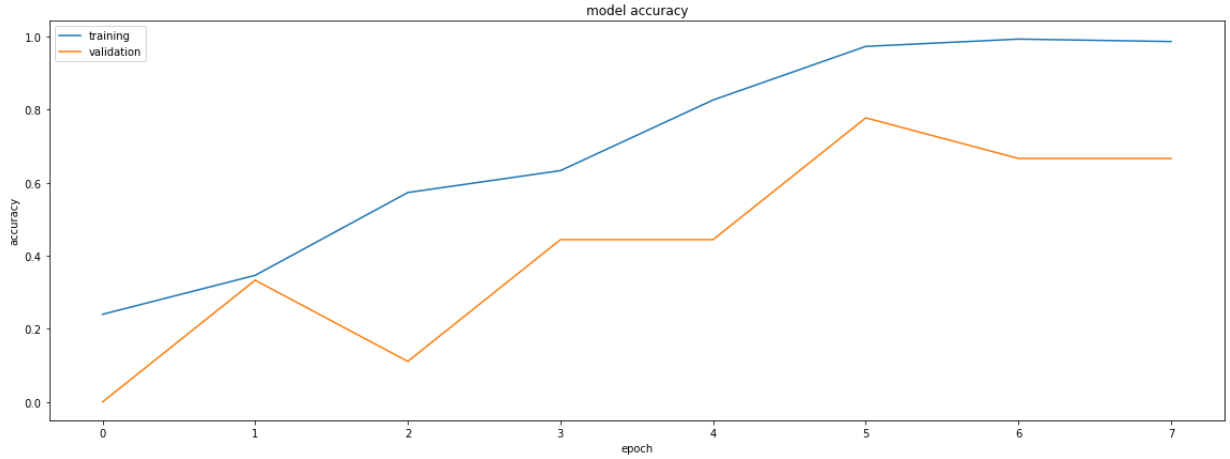
In [124…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

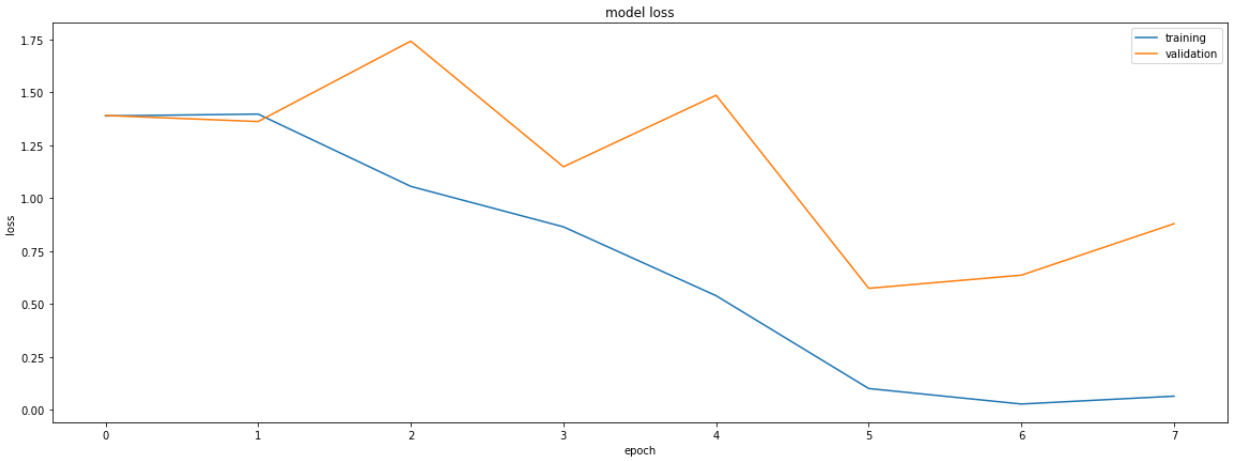
Out[124]:

	loss	accuracy	val_loss	val_accuracy
3	0.865	0.633	1.149	0.444
4	0.539	0.827	1.486	0.444
5	0.100	0.973	0.574	0.778
6	0.027	0.993	0.636	0.667
7	0.064	0.987	0.879	0.667

In [125…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [126…
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```





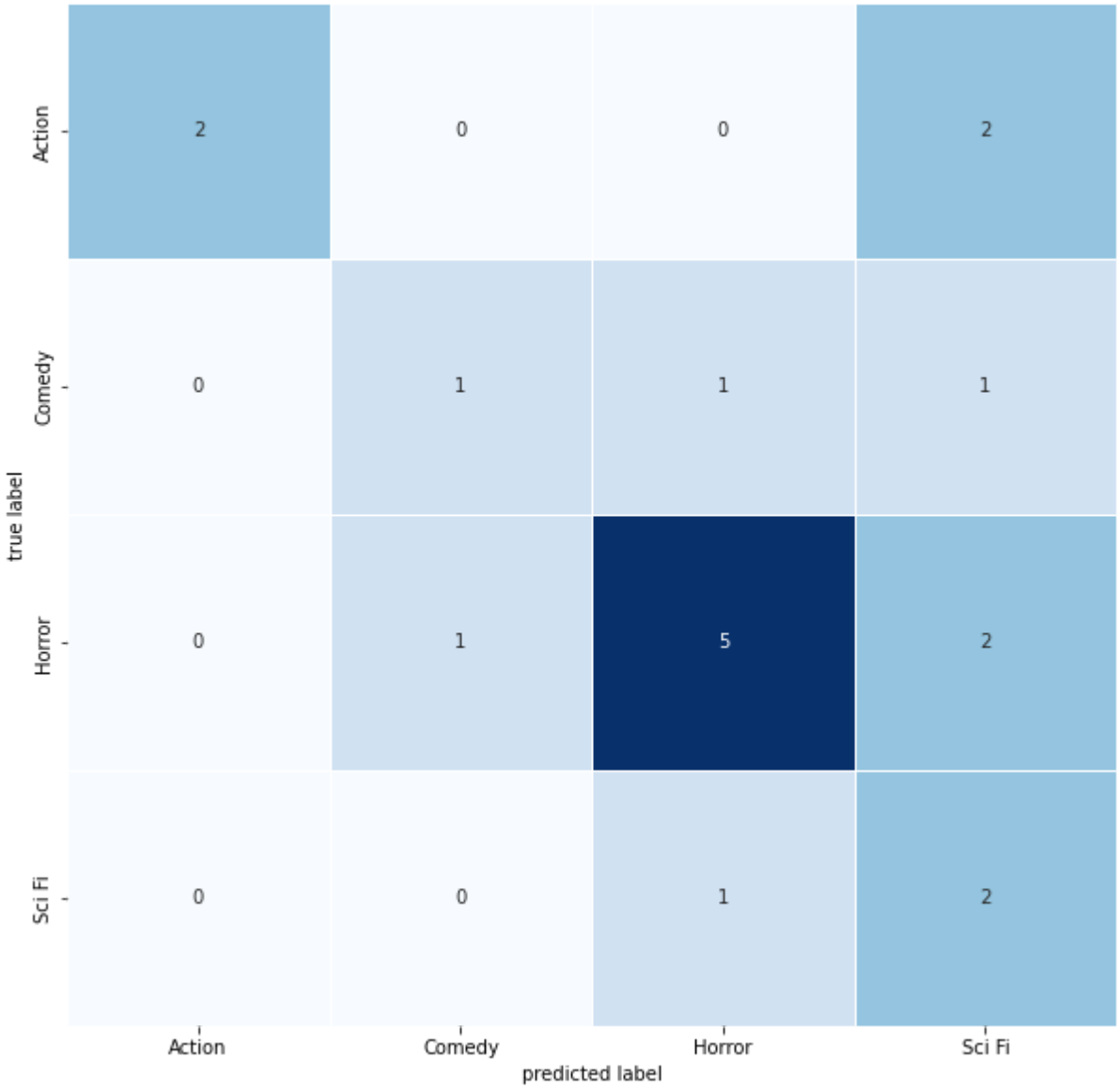
Assess the Model's Performance Using Confusion Matrix

In [127…
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 9s 339ms/step
```

In [128…
```
CLASSES_LIST = ['Action','Comedy','Horror','Sci Fi']
```

In [129…
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [130…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [131…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

Out[131]:

	0	1	2	3	4	5	6	7	8	9	10
Action	0.00%	46.22%	0.00%	0.00%	0.98%	67.65%	38.93%	0.13%	4.66%	0.48%	0.04%
Comedy	9.72%	5.57%	3.86%	1.30%	1.15%	3.31%	1.14%	3.34%	6.88%	10.75%	0.43%
Horror	89.77%	19.79%	95.72%	98.37%	5.37%	13.16%	14.62%	10.94%	5.29%	6.03%	99.27%
Sci Fi	0.51%	28.42%	0.42%	0.33%	92.50%	15.89%	45.32%	85.59%	83.17%	82.74%	0.26%

2.2) Sentiment Analysis and Classification with LSTM Models

Let's now turn our attention toward creating LSTM models capable of classifying movie reviews as having either positive or negative sentiments.

2.2.1) Data Wrangling for Sentiment Classification LSTM Models

Let's conduct data wrangling and preprocessing as needed to format the corpus text for the sentiment classification LSTM models.

In [132…
```
datafull=corpus_df.copy()
datafull.reset_index(drop=True, inplace=True)
datafull.head(4).T
```

Out[132]:

	0	1	2	
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_D
Student Name	SAR	SAR	SAR	
Genre of Movie	Action	Action	Action	
Review Type (pos or neg)	Negative	Negative	Negative	
Movie Title	Covenant	Covenant	Covenant	
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...	In a weird those
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103	Action_Co
Doc_ID	101	102	103	
raw_sentences	[Nearly two years after the American military ...	[Have Guy Ritchie and Jake Gyllenhaal switched...	[Guy Ritchie's The Covenant notably marks the ...	[In a we to those
lemmas	[nearly, year, american, military, afghanistan...	[guy, ritchie, jake, gyllenhaal, switch, place...	[guy, ritchie, covenant, notably, mark, featur...	[weird, thr action,
lemmas_joined	nearly year american military afghanistan end ...	guy ritchie jake gyllenhaal switch place peter...	guy ritchie covenant notably mark feature incl...	weird th acti
sentences_lemmatized	[nearly year american military afghanistan end...	[guy ritchie jake gyllenhaal switch place pete...	[guy ritchie covenant notably mark feature inc...	[weird th acti

In [133…
```
datafull['Text'] = datafull['Text'].apply(lambda x :clean_doc(x))
```

In [134…
```
data = datafull[['Text','Review Type (pos or neg)']].copy()
```

In [135…
```
data['Review Type (pos or neg)'] = data['Review Type (pos or neg)'].astype("category")
data['Review Type (pos or neg) code'] = data['Review Type (pos or neg)'].cat.codes
```

```
data['Review Type (pos or neg) code'].tail(10).T
```

Out[135]:
```
180    0
181    0
182    0
183    0
184    0
185    1
186    1
187    1
188    1
189    1
Name: Review Type (pos or neg) code, dtype: int8
```

In [136…
```
features, targets = data['Text'], data['Review Type (pos or neg) code']
data[["Review Type (pos or neg)","Review Type (pos or neg) code"]].value_counts()
```

Out[136]:
```
Review Type (pos or neg)  Review Type (pos or neg) code
Negative                  0                             95
Positive                  1                             95
dtype: int64
```

Let's create the training, validation, and testing datasets for construction of our LSTM models.

In [137…
```
trainds, valds, testds = get_dataset_partitions_pd(data[['Text','Review Type (pos or r
trainds.shape, valds.shape, testds.shape
```

Out[137]:
```
((152, 2), (19, 2), (19, 2))
```

Let's convert the dataframe to a TensorFlow dataset.

In [138…
```
# train X & y
train_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(trainds['Text'].values, tf.string)
)
train_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(trainds['Review Type (pos or neg) code'].values, tf.int64),
  )
# test X & y
test_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(testds['Text'].values, tf.string)
)
test_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(testds['Review Type (pos or neg) code'].values, tf.int64),
)
#val X & Y
val_text_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(valds['Text'].values, tf.string)
)
val_cat_ds_raw = tf.data.Dataset.from_tensor_slices(
            tf.cast(valds['Review Type (pos or neg) code'].values, tf.int64),
)
```

Create Datasets (X=Preprocessed Text, Y=Encoded Categories)

In [139…
```
train_ds = tf.data.Dataset.zip(
    (
            train_text_ds_raw,
```

```
            train_cat_ds_raw
    )
)
test_ds = tf.data.Dataset.zip(
    (
            test_text_ds_raw,
            test_cat_ds_raw
    )
)
val_ds = tf.data.Dataset.zip(
    (
            val_text_ds_raw,
            val_cat_ds_raw
    )
)
```

Create Data Pipelines (Batching, Shuffling, and Optimizing)

In [140…

```
batch_size = 3
AUTOTUNE = tf.data.experimental.AUTOTUNE
buffer_size=train_ds.cardinality().numpy()

train_ds = train_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)

test_ds = test_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)


val_ds = val_ds.shuffle(buffer_size=buffer_size)\
                   .batch(batch_size=batch_size,drop_remainder=True)\
                   .cache()\
                   .prefetch(AUTOTUNE)
```

In [141…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 190 entries, 0 to 189
Data columns (total 3 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Text                           190 non-null    object
 1   Review Type (pos or neg)       190 non-null    category
 2   Review Type (pos or neg) code  190 non-null    int8
dtypes: category(1), int8(1), object(1)
memory usage: 2.1+ KB
```

Create The Text Encoder

The raw text loaded by `tfds` needs to be processed before it can be used in a model. The simplest way to process text for training is using the `experimental.preprocessing.TextVectorization` layer. This layer has many capabilities, but this tutorial sticks to the default behavior.

Create the layer, and pass the dataset's text to the layer's `.adapt` method: The processing of each sample contains the following steps:

```
standardize each sample (usually lowercasing + punctuation stripping)
split each sample into substrings (usually words)
recombine substrings into tokens (usually ngrams)
index tokens (associate a unique int value with each token)
transform each sample using this index, either into a vector of ints
or a dense float vector.
```

In [142…
```
VOCAB_SIZE=5000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE, standardize="lower_and_strip_punctuation", pad_to_max_toker
encoder.adapt(train_ds.map(lambda text, label: text), batch_size= None)
```

In [143…
```
vocab = np.array(encoder.get_vocabulary())
len(vocab)
```

Out[143]:
```
5000
```

In [144…
```
vocab = np.array(encoder.get_vocabulary())
vocab[:20]
```

Out[144]:
```
array(['', '[UNK]', 's', 'the', 'film', 'i', 'movie', 'nt', 'one', 'like',
       'it', 'get', 'time', 'character', 'two', 'make', 'even', 'but',
       'first', 'much'], dtype='<U30')
```

In [145…
```
vocab[-20:]
```

Out[145]:
```
array(['vigilance', 'viewed', 'victory', 'victorianera', 'victorian',
       'victimized', 'vicious', 'vice', 'vicarious', 'vibrate', 'viable',
       'vessel', 'verve', 'vertigo', 'vertically', 'vertical',
       'veritable', 'verhoevenstyled', 'verdant', 'verbatim'],
      dtype='<U30')
```

View Examples of Encoded Words

In [146…
```
encoded_example = encoder('encanto we dont talk about bruno no no').numpy()
encoded_example[:]
```

Out[146]:
```
array([   1,  208,    1,  618, 2388,    1,  142,  142], dtype=int64)
```

In [147…
```
len(encoder.get_vocabulary())
```

Out[147]:
```
5000
```

In [148…
```
encoder(data['Text'][0])
```

Out[148]:
```
<tf.Tensor: shape=(315,), dtype=int64, numpy=
array([ 386,   14,   22,  401,  630, 3197,  487,  967, 2766,  193,   57,
        760,   40,  570,  420, 1621,   53,   62,  872,  203,  473,   89,
          2, 3124,   63, 4258,   58,  121,   89,   61,  103,    8,  518,
        185,   96,  570, 4081,  103,  128,    8,   99,    8,  864,  292,
       3402,   36, 2412, 1685,    1, 1117, 1874,    1,    1,    1,  142,
        154,   17,  549,  193,  696,    3,  265, 1167,  834,  276,   85,
        193,  487,    2,  235,   67,   51,   27,  728,   61,  103,   27,
       2469,   61,  103,   27, 4302,   61,  103,  130,    1,   61,  103,
         40,  179,  890,   96,  927,   67,  497,  250,   11, 4546,   68,
          1,  502,    4,    8,   93,  417,   16,   50,  565,  649,    9,
          6,  103,    2,  748, 2516,  116,    1,  231,  247,  506, 4476,
       1530, 1086,    4, 4664, 1107,  315,  417,   25,  518,  735,   68,
        432,  486,  249,  487, 4707,   57,  630,   14,  698,   29, 1257,
        800,   81, 1385, 1918, 1577,   55, 2441,  165,  239,  193,    1,
       1484,  198, 1163, 1190,    3,  265,  103,  799,  710,  482, 1231,
       1857, 2099, 2276,  688,   57,  486, 1639,   97,  104,  107,  276,
         35,  227, 1169,  798, 1334,  764,    1,  284, 1536,    1,   10,
          2,  771,   30,  471,  371,   23,  548,  104,    2,  432, 1169,
        154,   47,  249, 1091,   83,  854,  534,  229,    1,  249,   35,
        623,   83,   35,   20,  925, 1699, 4084, 1304,   10,    2, 3291,
       1279,   26,  191, 1689,    2,   20,  961,    8,  103,    2,    4,
       3675,  253,  453,  877,  300,   10,  191,  681, 3706,   81, 1131,
        249,  165,  239,   57, 4840, 1932,   26,   48,  562, 2554,    3,
        265,    1,  543,   25,    2, 2036, 1693,  417,    9,  276,    2,
       1639,  104,  534,    2,   83,  725, 1979,  458,  118,   61,    1,
       3031,    1,   63, 4318,    1,  113,   61,   63,    1,   63, 4928,
        897,  118,  247,  192, 2849, 4470,    8,   82,    1, 1257, 1225,
       2708, 1979, 2607,    2, 2459,    1,   17,   11,   18,  327,  471,
       1007,    8,   82,  779, 1190,   18,  251], dtype=int64)>
```

In [149…
```
for example, label in train_ds.take(1):
  print('texts: ', example.numpy()[:1])
  print()
```

```
texts:  [b"In Minority Report , futuristic thriller set 2054 , murder eliminated Wash
ington DC thanks division law enforcement known Precrime . Tapping mind Pre-Cogs , tr
io psychic foresee future , Precrime able track arrest would-be killer actually commi
t lethal deed . The system appears flawless . But police chief John Anderton ( Tom Cr
uise ) , head controversial department , accused future murder man n't even met , set
prove innocence , hunted fellow officer . Based 1956 short story renowned science fic
tion writer Philip K. Dick , Minority Report typical director Steven Spielberg deftly
meld flashy entertainment critical issue . The movie made three major ingredient : th
rilling action innocent man running law ; cautionary tale good intention go astray te
mpered wisdom ( specifically , infringing people 's civil liberty name protecting ) ;
old debate human free versus pre-ordained destiny . Unfortunately , movie 's philosop
hical element take backseat John 's mundane escape authority . Clearly , I understand
relentless manhunt important bunch metaphysical mumbo jumbo people . But without pre-
determined fate aspect , Minority Report really nothing standard crime drama , comple
te cliche ranging generic framing innocent man corruption rank police . I 'm asking s
tuffy dissertation bore audience like sterile Solaris -- comfortable balance give lit
tle substantial acknowledgement concept able control one 's future . As , film briefl
y engages notion pre-determination intelligently John debate Precrime issue Danny Wit
wer ( Colin Farrell ) , agent Justice Department ordered find flaw system . John 's a
rgument absolute future interesting thought-provoking moment whole movie , although s
cene quickly give way nonsense stretch even boundary science fiction ( claim Pre-Cogs
foresee homicide `` 's nothing destructive metaphysical fabric bind u untimely murder
one human another `` put arbitrary importance human life ) . Credibility shot story d
issolve trite cheerleading say , `` You ! You control destiny ! Go , John ! `` Cruise
's frequent failure convey emotion make character John Anderton le interesting . We s
ee none conviction idea justice , none sense betrayal system turn , none paranoia sud
denly hostile world . In attempt elicit sympathy establish John 's dedication"]
```

2.2.2) Sentiment Classification LSTM Model 1

Build the LSTM Model. For more information about layers in keras, refer to this resource:
https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [150…
```
k.clear_session()
num_classes=2
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True, dro
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,dropout=0.3))
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [151…
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [152…
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 58s 899ms/step - loss: 0.6955 - accuracy: 0.
4200 - val_loss: 0.6898 - val_accuracy: 0.5556
Epoch 2/200
50/50 [==============================] - 38s 754ms/step - loss: 0.7261 - accuracy: 0.
6600 - val_loss: 0.7099 - val_accuracy: 0.5556
Epoch 3/200
50/50 [==============================] - 38s 769ms/step - loss: 0.2698 - accuracy: 0.
9467 - val_loss: 1.4292 - val_accuracy: 0.6667
Epoch 4/200
50/50 [==============================] - 38s 770ms/step - loss: 0.1783 - accuracy: 0.
9600 - val_loss: 0.8286 - val_accuracy: 0.3333
Epoch 5/200
50/50 [==============================] - 38s 767ms/step - loss: 0.0137 - accuracy: 1.
0000 - val_loss: 0.6436 - val_accuracy: 0.7778
Epoch 6/200
50/50 [==============================] - 38s 758ms/step - loss: 0.0011 - accuracy: 1.
0000 - val_loss: 0.8744 - val_accuracy: 0.6667
Epoch 7/200
50/50 [==============================] - 38s 769ms/step - loss: 7.7478e-04 - accurac
y: 1.0000 - val_loss: 1.3474 - val_accuracy: 0.5556
CPU times: total: 1min 43s
Wall time: 4min 47s
```

View a Summary of the LSTM Model Architecture

In [153…

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 2)                 130

=================================================================
Total params: 431,554
Trainable params: 431,554
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [154…

```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 1s 200ms/step - loss: 1.5217 - accuracy: 0.333
3
Test Loss: 1.5216914415359497
Test Accuracy: 0.3333333432674408
```

In [155…
```
history_dict = history.history
history_dict.keys()
```

Out[155]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

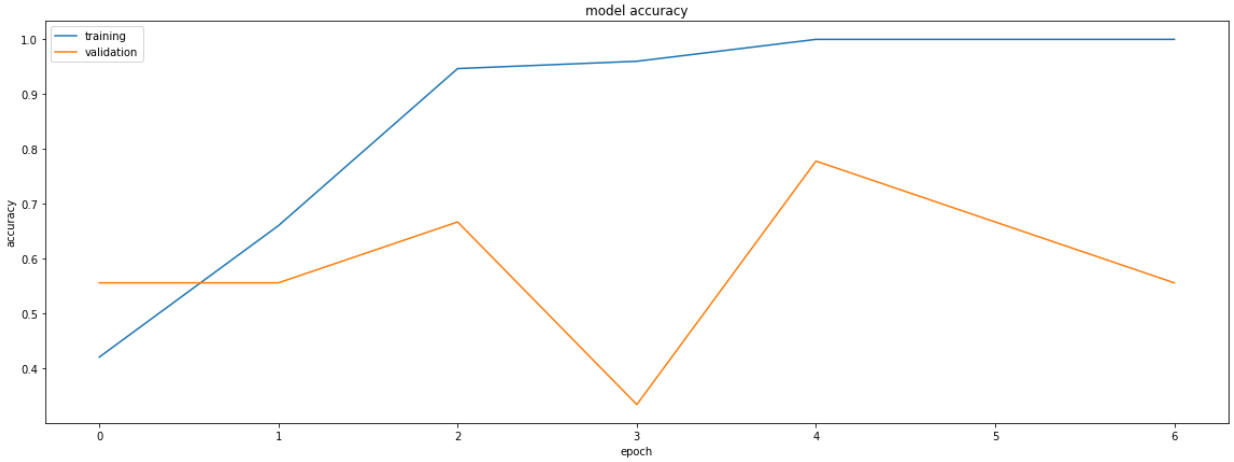
In [156…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

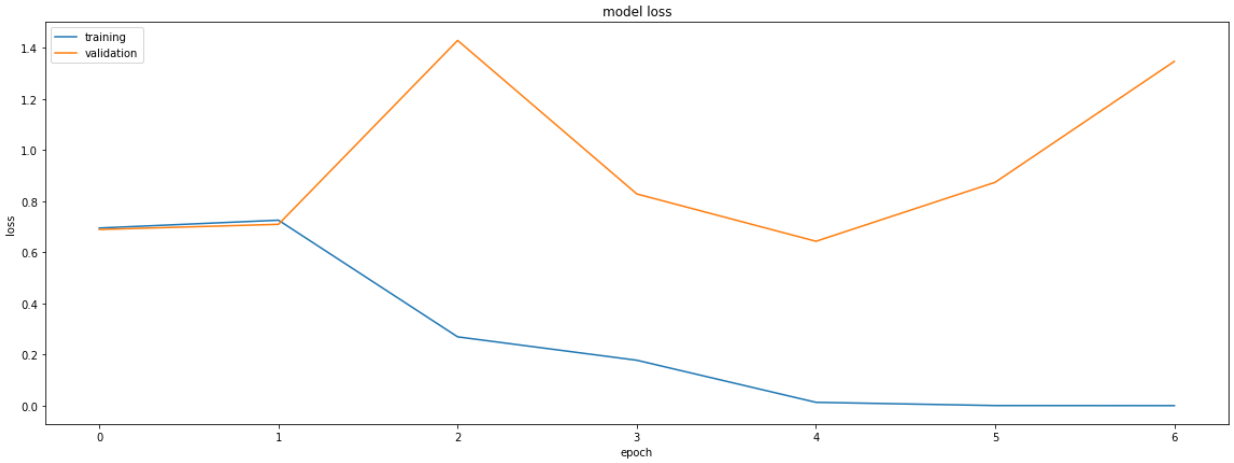
Out[156]:

	loss	accuracy	val_loss	val_accuracy
2	0.270	0.947	1.429	0.667
3	0.178	0.960	0.829	0.333
4	0.014	1.000	0.644	0.778
5	0.001	1.000	0.874	0.667
6	0.001	1.000	1.347	0.556

In [157…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [158…
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```





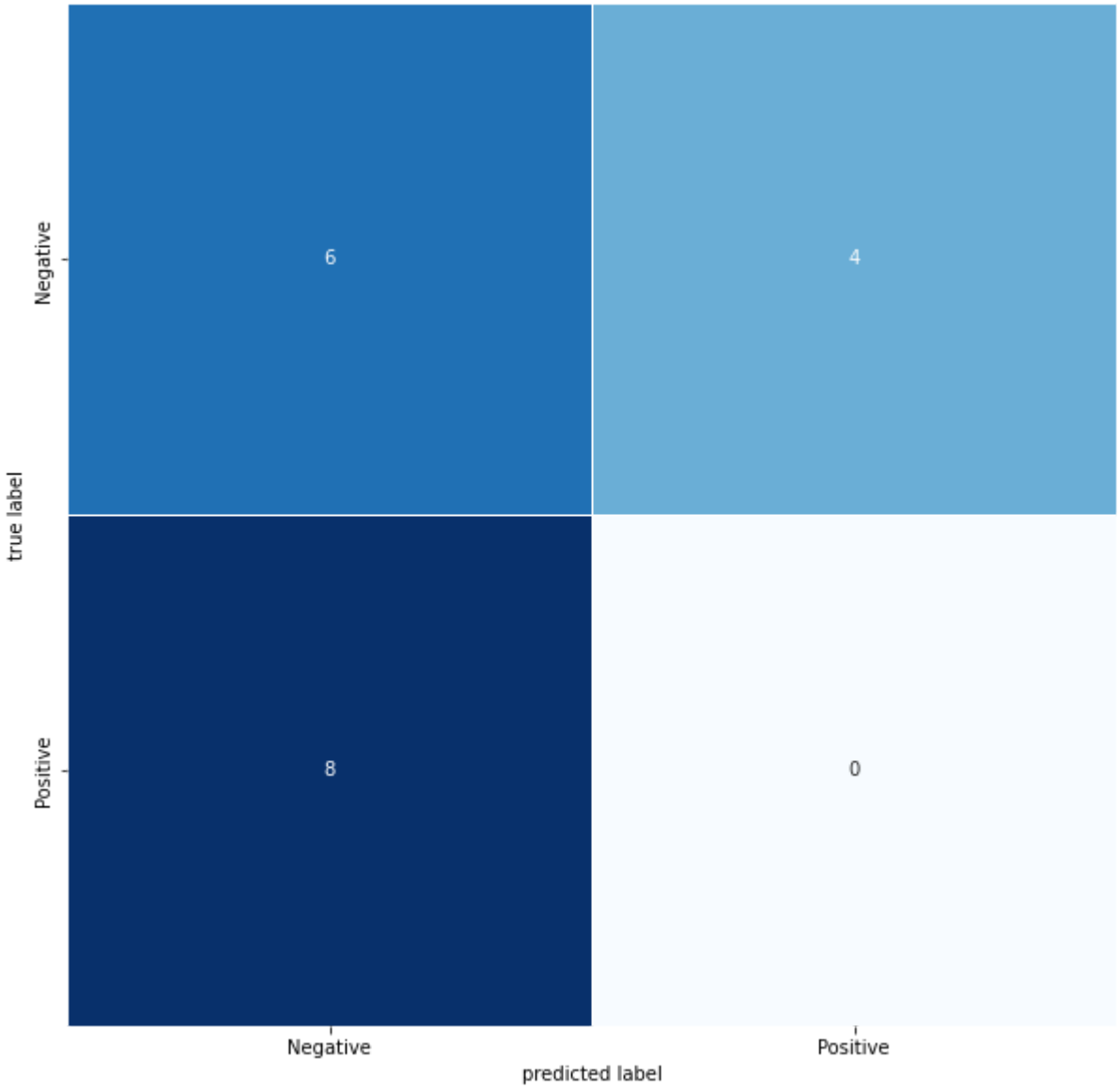
Assess the Model's Performance Using Confusion Matrix

In [159…
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 13s 168ms/step
```

In [160…
```
CLASSES_LIST = ['Negative','Positive']
```

In [161…
```
plot_confusion_matrix_labeled(y2, y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [162…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [163…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

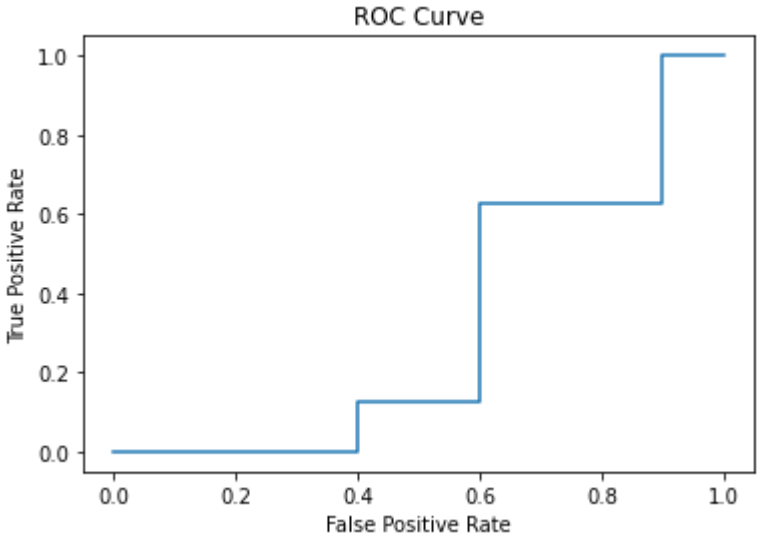
Out[163]:

	0	1	2	3	4	5	6	7	8	9	10
Negative	92.73%	69.28%	62.45%	77.66%	1.07%	96.21%	97.47%	15.91%	84.85%	88.44%	96.30%
Positive	7.27%	30.72%	37.55%	22.34%	98.93%	3.79%	2.53%	84.09%	15.15%	11.56%	3.70%

In [164…
```
fpr, tpr, _ = roc_curve(y2, preds2[:,1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('ROC Curve')

precision = precision_score(y2, y_pred2)
recall = recall_score(y2, y_pred2)
auc = roc_auc_score(y2, preds2[:,1])
print("AUC = ", round(auc, 3))
```

```
print("Precision = ", round(precision,3))
print("Recall = ", round(recall,3))
```

```
AUC =  0.312
Precision =  0.0
Recall =  0.0
```



2.2.3) Sentiment Classification LSTM Model 2

Build the LSTM Model. For more information about layers in keras, refer to this resource:
https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [165…
```
k.clear_session()
num_classes=2
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 128, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,  return_sequences=True, dr
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,dropout=0.3))
   ,tf.keras.layers.Dense(128, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [166…
```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [167…
```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 100s 2s/step - loss: 0.6984 - accuracy: 0.46
00 - val_loss: 0.6921 - val_accuracy: 0.5556
Epoch 2/200
50/50 [==============================] - 80s 2s/step - loss: 0.5700 - accuracy: 0.826
7 - val_loss: 0.8642 - val_accuracy: 0.5556
Epoch 3/200
50/50 [==============================] - 79s 2s/step - loss: 0.0648 - accuracy: 0.973
3 - val_loss: 1.1502 - val_accuracy: 0.5556
CPU times: total: 1min 13s
Wall time: 4min 19s
```

View a Summary of the LSTM Model Architecture

In [168…
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 128)         640000

 bidirectional (Bidirectiona  (None, None, 256)        263168
 l)

 bidirectional_1 (Bidirectio  (None, 128)              164352
 nal)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 1,084,290
Trainable params: 1,084,290
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [169…
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 3s 494ms/step - loss: 2.0868 - accuracy: 0.388
9
Test Loss: 2.086827278137207
Test Accuracy: 0.3888888955116272
```

In [170…
```
history_dict = history.history
history_dict.keys()
```

Out[170]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

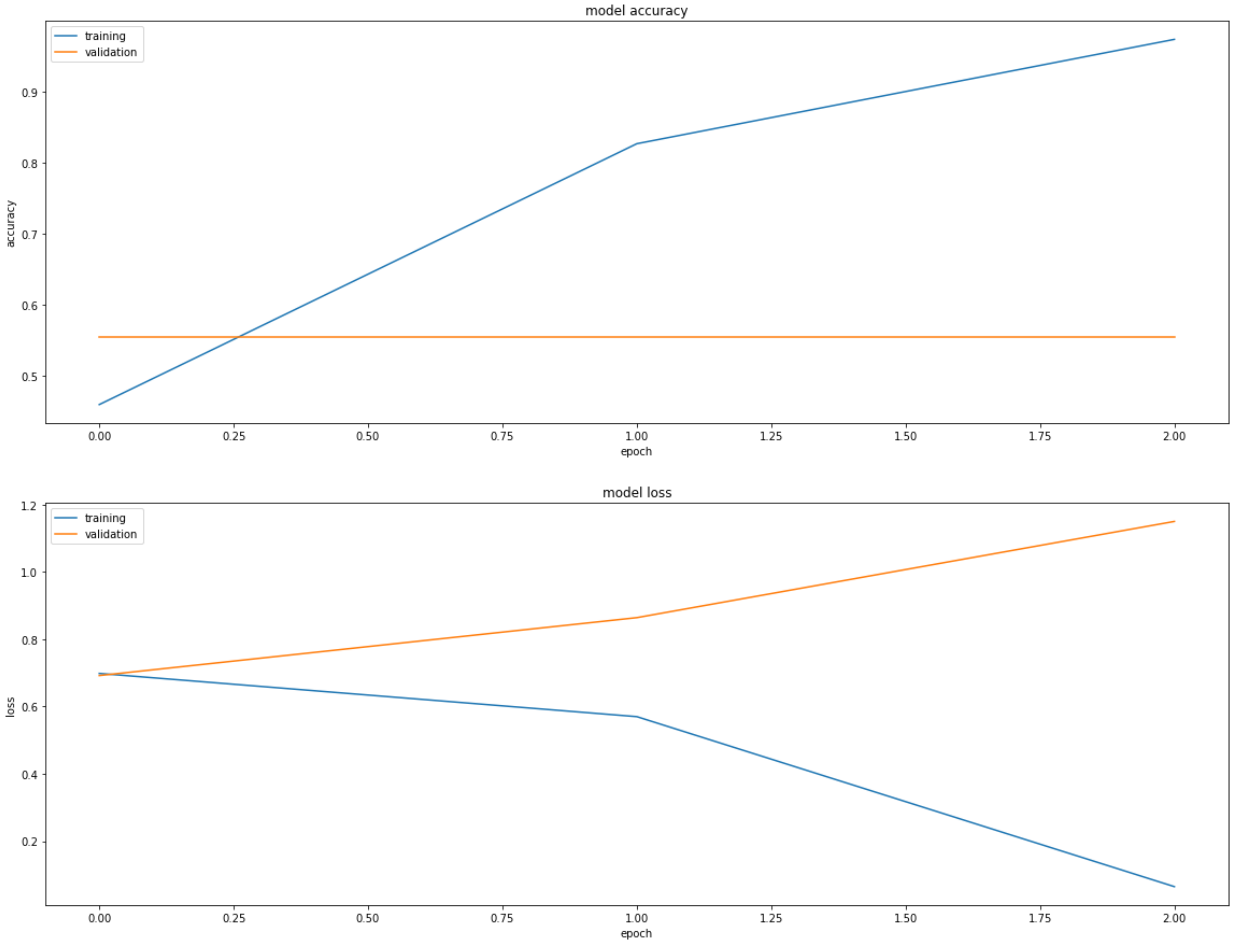
In [171…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

Out[171]:

	loss	accuracy	val_loss	val_accuracy
0	0.698	0.460	0.692	0.556
1	0.570	0.827	0.864	0.556
2	0.065	0.973	1.150	0.556

In [172…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [173…
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```



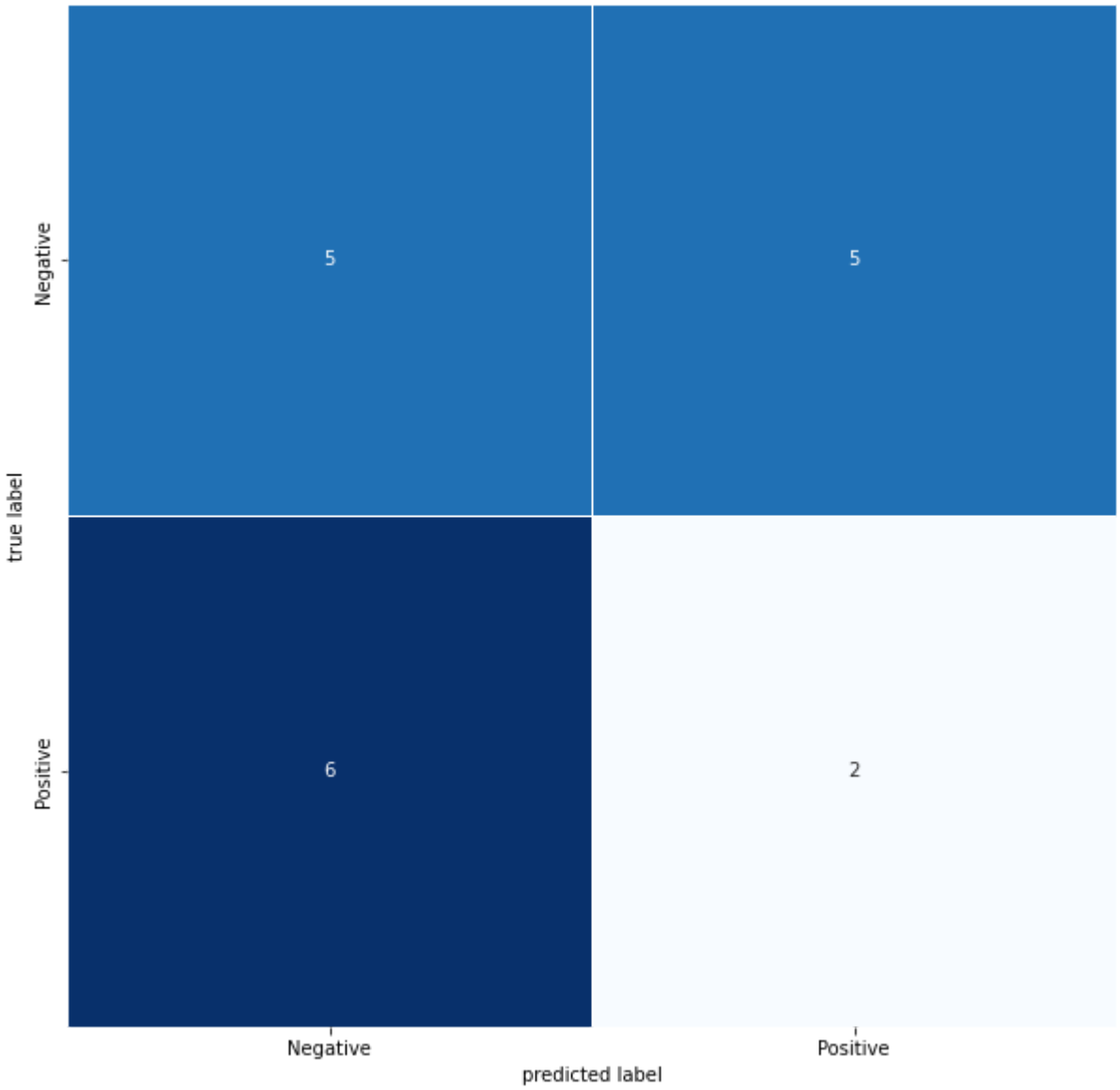
Assess the Model's Performance Using Confusion Matrix

In [174…
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 8s 741ms/step
```

In [175…
```
CLASSES_LIST = ['Negative','Positive']
```

In [176…
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [177…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [178…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

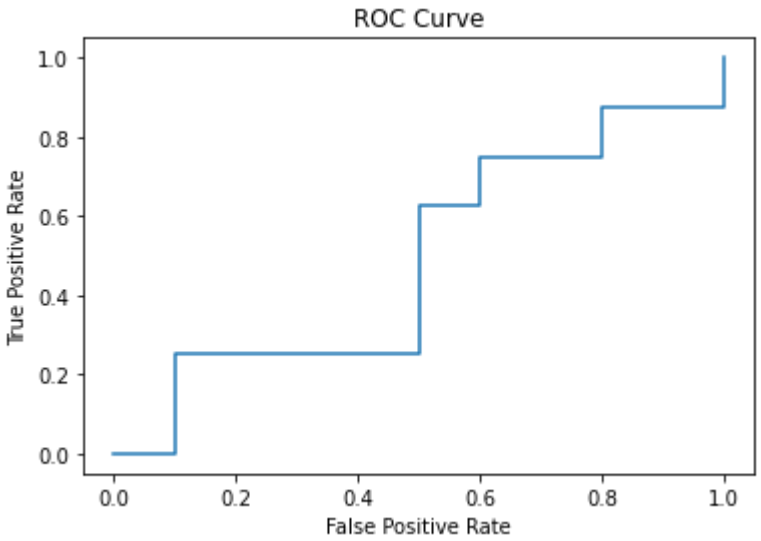
Out[178]:

	0	1	2	3	4	5	6	7	8	9	10
Negative	99.62%	98.69%	99.42%	3.10%	28.74%	5.18%	99.13%	48.08%	98.09%	99.09%	99.19%
Positive	0.38%	1.31%	0.58%	96.90%	71.26%	94.82%	0.87%	51.92%	1.91%	0.91%	0.81%

In [179…
```
fpr, tpr, _ = roc_curve(y2, preds2[:,1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
```

```
plt.title('ROC Curve')

precision = precision_score(y2, y_pred2)
recall = recall_score(y2, y_pred2)
auc = roc_auc_score(y2, preds2[:,1])
print("AUC = ", round(auc, 3))
print("Precision = ", round(precision,3))
print("Recall = ", round(recall,3))
```

```
AUC =  0.487
Precision =  0.286
Recall =  0.25
```



2.2.4) Sentiment Classification LSTM Model 3

Build the LSTM Model. For more information about layers in keras, refer to this resource:
https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [180…

```
k.clear_session()
num_classes=2
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 64, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True, dro
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,dropout=0.3))
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(32, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [181…

```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
              ,metrics=['accuracy'])
```

In [182…

```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 55s 789ms/step - loss: 0.6962 - accuracy: 0.
4600 - val_loss: 0.6917 - val_accuracy: 0.5556
Epoch 2/200
50/50 [==============================] - 35s 699ms/step - loss: 0.6884 - accuracy: 0.
6067 - val_loss: 0.5979 - val_accuracy: 0.8889
Epoch 3/200
50/50 [==============================] - 34s 674ms/step - loss: 0.3158 - accuracy: 0.
8733 - val_loss: 0.7867 - val_accuracy: 0.6667
Epoch 4/200
50/50 [==============================] - 34s 686ms/step - loss: 0.0570 - accuracy: 0.
9667 - val_loss: 1.4970 - val_accuracy: 0.6667
CPU times: total: 1min 6s
Wall time: 2min 38s
```

View a Summary of the LSTM Model Architecture

In [183…
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 64)          320000

 bidirectional (Bidirectiona  (None, None, 128)        66048
 l)

 bidirectional_1 (Bidirectio  (None, 64)               41216
 nal)

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 2)                 66

=================================================================
Total params: 433,570
Trainable params: 433,570
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [184…
```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 1s 176ms/step - loss: 3.2183 - accuracy: 0.277
8
Test Loss: 3.2183308601379395
Test Accuracy: 0.2777777910232544
```

In [185…
```
history_dict = history.history
history_dict.keys()
```

Out[185]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

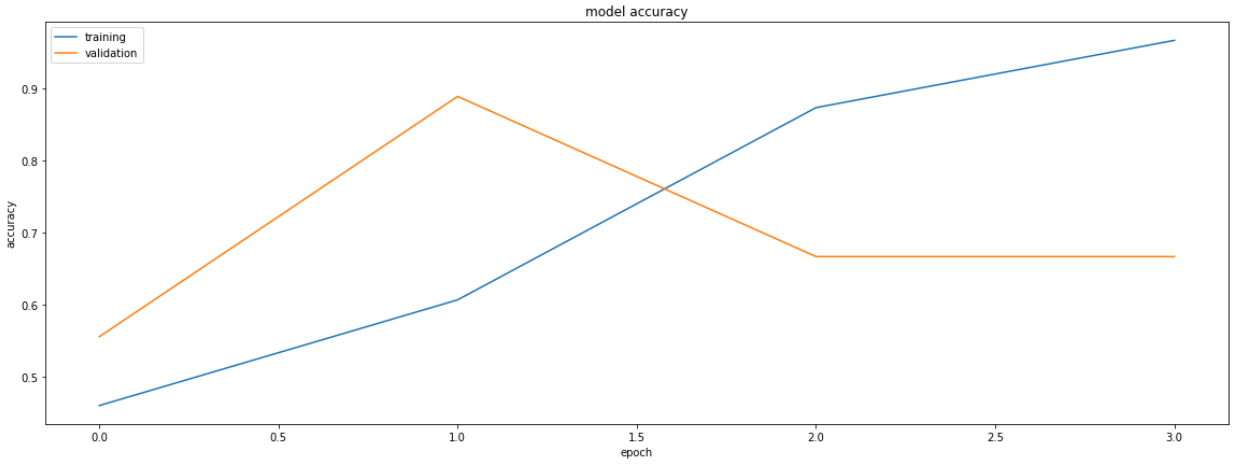
In [186…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

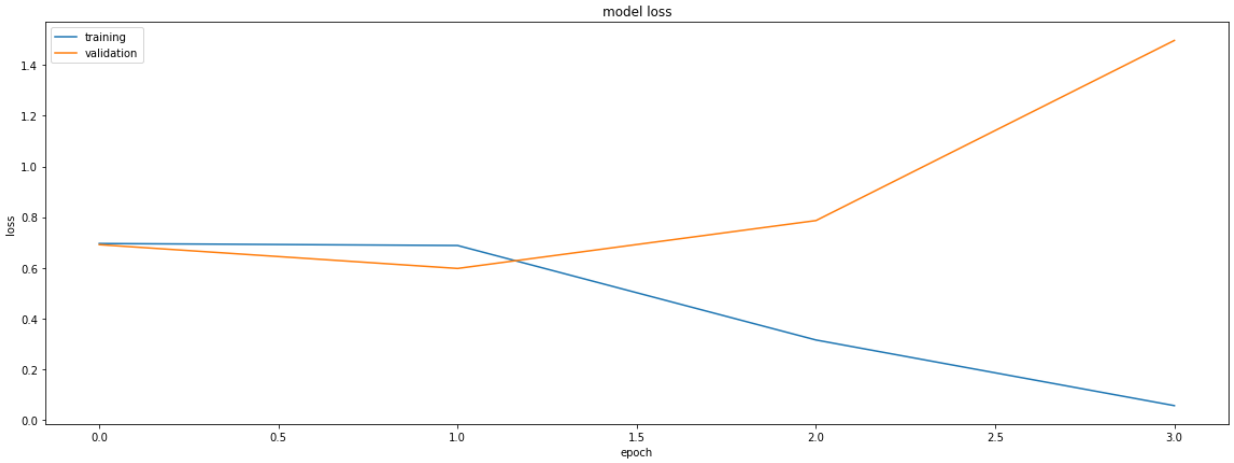
Out[186]:

	loss	accuracy	val_loss	val_accuracy
0	0.696	0.460	0.692	0.556
1	0.688	0.607	0.598	0.889
2	0.316	0.873	0.787	0.667
3	0.057	0.967	1.497	0.667

In [187…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [188…
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```





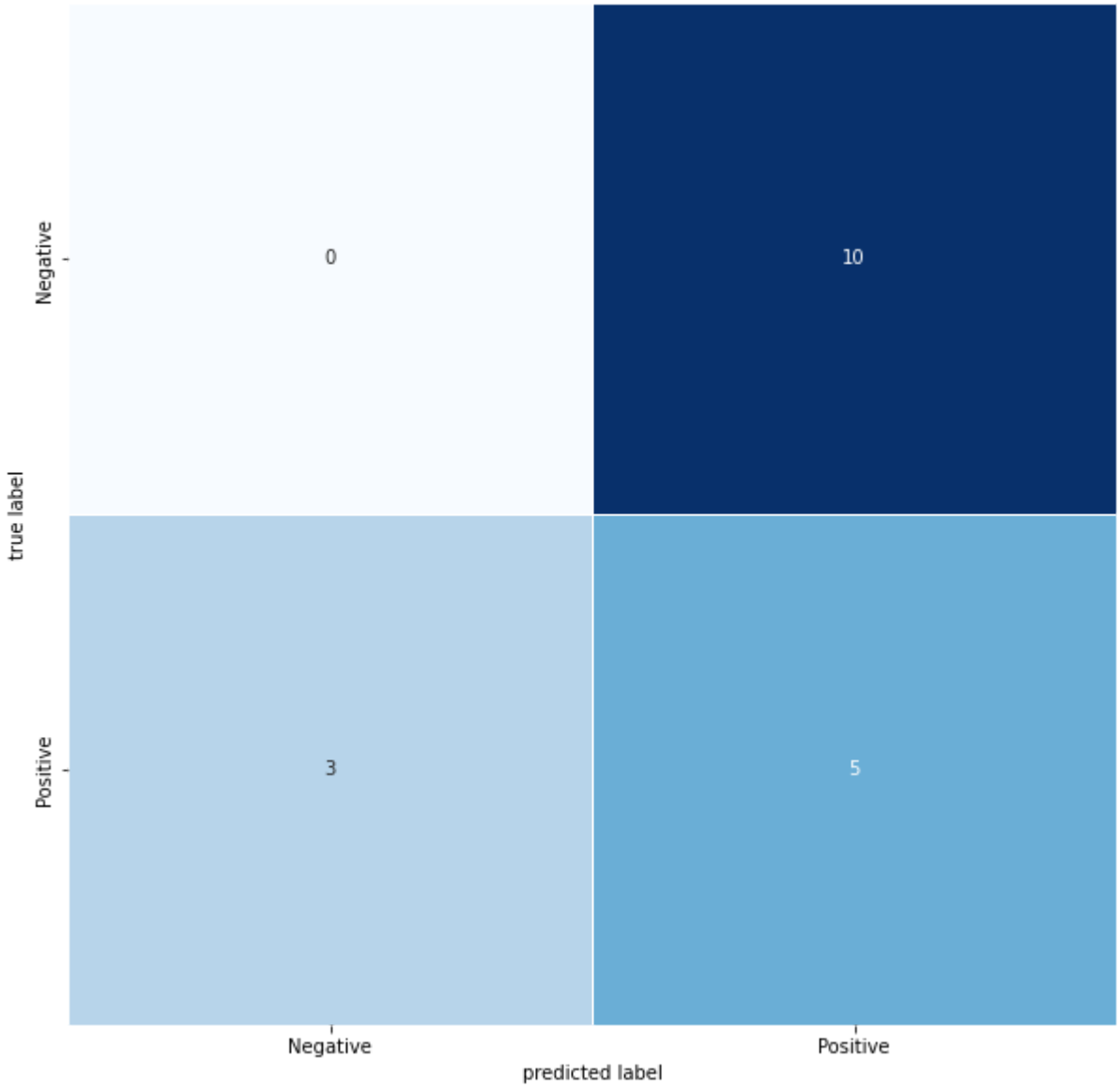
Assess the Model's Performance Using Confusion Matrix

In [189…
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 5s 198ms/step
```

In [190…
```
CLASSES_LIST = ['Negative','Positive']
```

In [191…
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [192…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [193…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

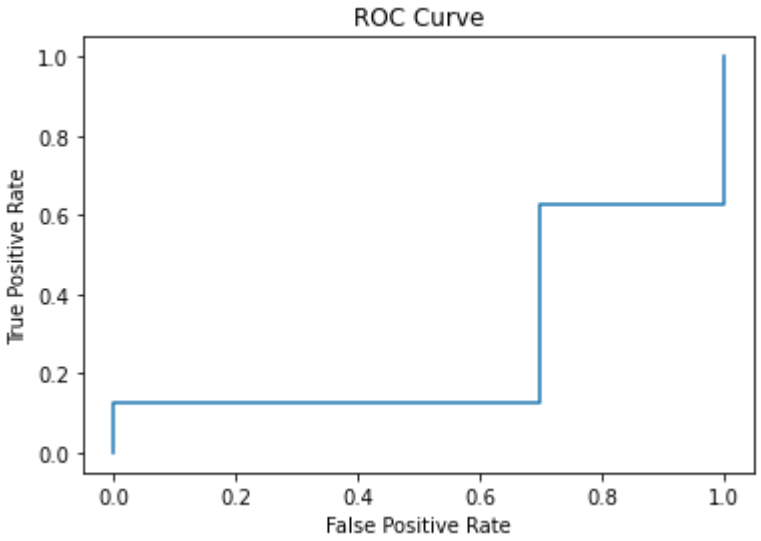
Out[193]:

	0	1	2	3	4	5	6	7	8	9	10
Negative	11.66%	0.75%	0.65%	2.11%	0.21%	0.80%	9.08%	0.50%	1.06%	0.27%	99.37%
Positive	88.34%	99.25%	99.35%	97.89%	99.79%	99.20%	90.92%	99.50%	98.94%	99.73%	0.63%

In [194…

```
fpr, tpr, _ = roc_curve(y2, preds2[:,1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('ROC Curve')

precision = precision_score(y2, y_pred2)
recall = recall_score(y2, y_pred2)
auc = roc_auc_score(y2, preds2[:,1])
print("AUC = ", round(auc, 3))
print("Precision = ", round(precision,3))
print("Recall = ", round(recall,3))
```

```
AUC =  0.275
Precision =  0.333
Recall =  0.625
```



2.2.5) Sentiment Classification LSTM Model 4

Build the LSTM Model. For more information about layers in keras, refer to this resource:
https://www.tensorflow.org/api_docs/python/tf/keras/layers

In [195…

```
k.clear_session()
num_classes=2
model = tf.keras.Sequential([encoder
   ,tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 128, mask_zero=True)
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,  return_sequences=True, dr
   ,tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,dropout=0.5))
   ,tf.keras.layers.Dense(128, activation='relu')
   ,tf.keras.layers.Dense(64, activation='relu')
   ,tf.keras.layers.Dense(num_classes,activation='softmax')
])
```

In [196…

```
model.compile(optimizer= tf.keras.optimizers.Adam( )
              ,loss=tf.keras.losses.SparseCategoricalCrossentropy()
```

```
              ,metrics=['accuracy'])
```

In [197…

```
%%time
history = model.fit(train_ds
         ,epochs=200
         ,validation_data=val_ds
         ,validation_steps=3
         ,callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience
```

```
Epoch 1/200
50/50 [==============================] - 130s 2s/step - loss: 0.6961 - accuracy: 0.48
67 - val_loss: 0.6352 - val_accuracy: 0.8889
Epoch 2/200
50/50 [==============================] - 111s 2s/step - loss: 0.6656 - accuracy: 0.59
33 - val_loss: 0.6351 - val_accuracy: 0.7778
Epoch 3/200
50/50 [==============================] - 109s 2s/step - loss: 0.2361 - accuracy: 0.93
33 - val_loss: 0.7962 - val_accuracy: 0.6667
CPU times: total: 1min 56s
Wall time: 5min 50s
```

View a Summary of the LSTM Model Architecture

In [198…

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization (TextVec  (None, None)             0
 torization)

 embedding (Embedding)       (None, None, 128)         640000

 bidirectional (Bidirectiona  (None, None, 256)        263168
 l)

 bidirectional_1 (Bidirectio  (None, 128)              164352
 nal)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 2)                 130

=================================================================
Total params: 1,092,418
Trainable params: 1,092,418
Non-trainable params: 0
_________________________________________________________________
```

Assess Model's Performance Using Performance Metrics

In [199…

```
test_loss, test_acc = model.evaluate(test_ds)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
6/6 [==============================] - 4s 684ms/step - loss: 1.0802 - accuracy: 0.555
6
Test Loss: 1.0802489519119263
Test Accuracy: 0.5555555820465088
```

In [200…
```
history_dict = history.history
history_dict.keys()
```

Out[200]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

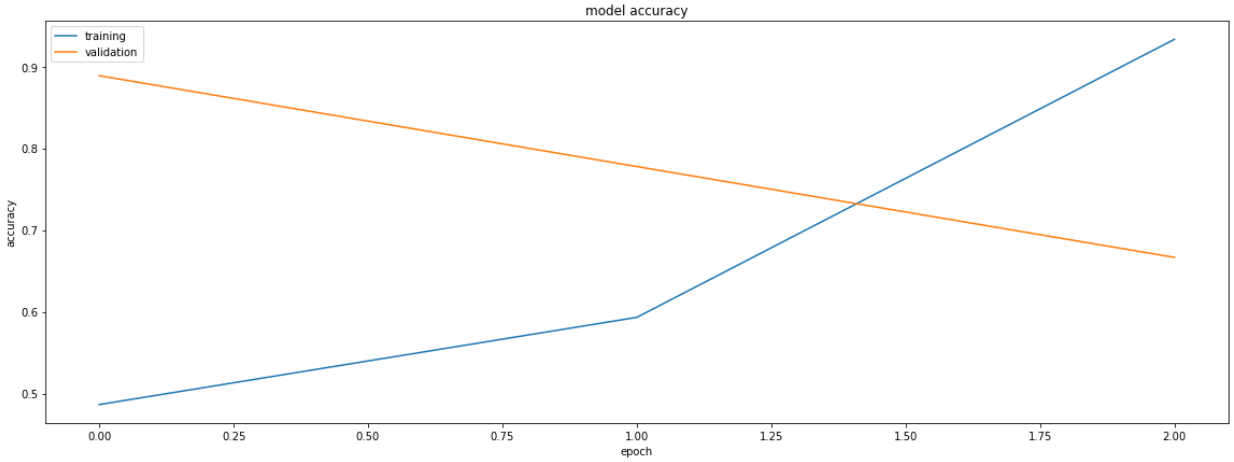
In [201…
```
history_df=pd.DataFrame(history_dict)
history_df.tail().round(3)
```

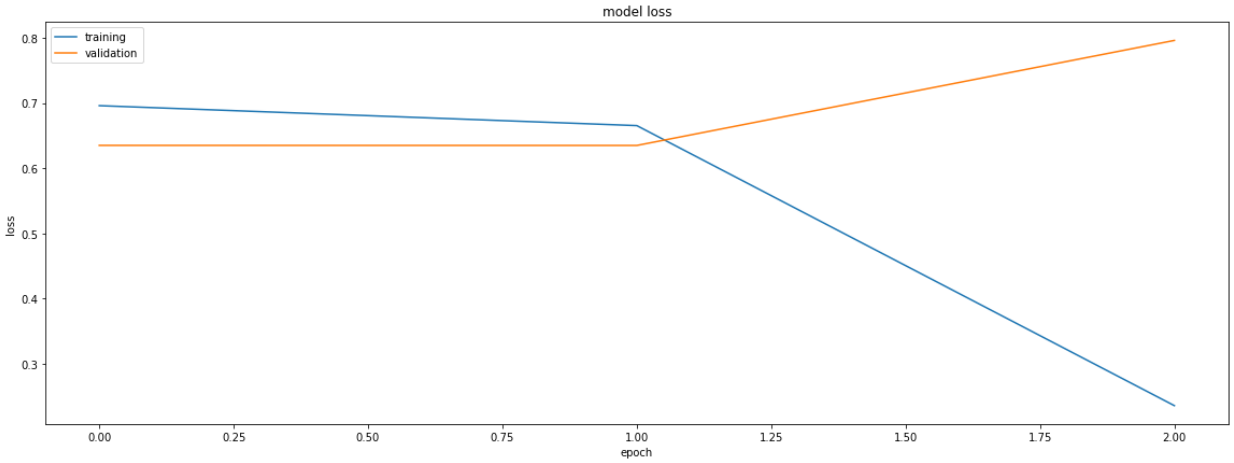
Out[201]:

	loss	accuracy	val_loss	val_accuracy
0	0.696	0.487	0.635	0.889
1	0.666	0.593	0.635	0.778
2	0.236	0.933	0.796	0.667

In [202…
```
losses = history.history['loss']
accs = history.history['accuracy']
val_losses = history.history['val_loss']
val_accs = history.history['val_accuracy']
epochs = len(losses)
```

In [203…
```
plt.subplots(figsize=(16,12))
plt.tight_layout()
display_training_curves(history.history['accuracy'], history.history['val_accuracy'],
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss',
```





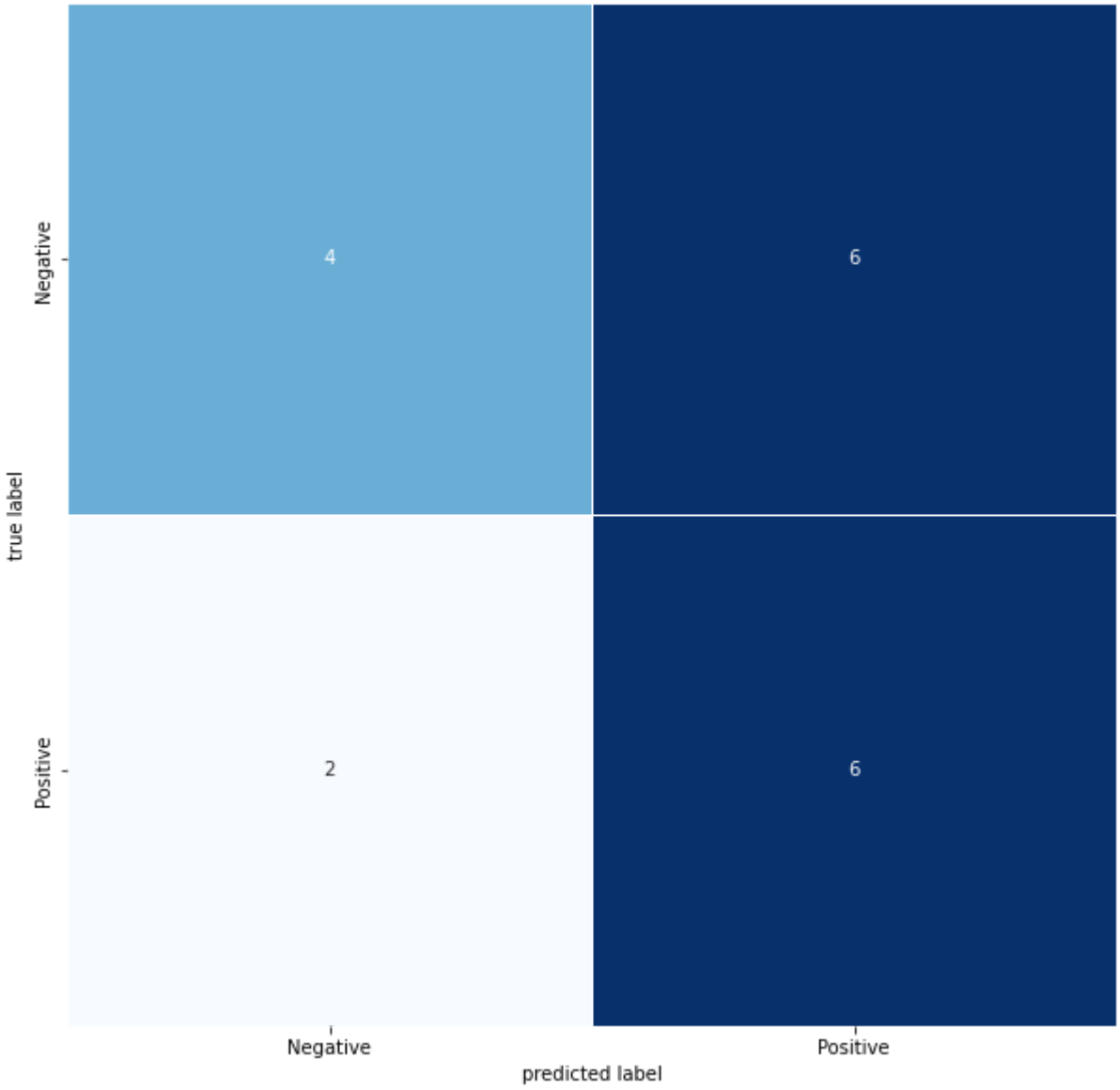
Assess the Model's Performance Using Confusion Matrix

In [204…
```
preds2 = model.predict(test_ds)
y_pred2 = np.argmax(preds2, axis=1)
y2 = np.concatenate([y for x, y in test_ds], axis=0)
```

```
6/6 [==============================] - 17s 769ms/step
```

In [205…
```
CLASSES_LIST = ['Negative','Positive']
```

In [206…
```
plot_confusion_matrix_labeled(y2,y_pred2, CLASSES_LIST=CLASSES_LIST)
```



In [207…
```
cm = sns.light_palette((260, 75, 60), input="husl", as_cmap=True)
```

In [208…
```
df2 = pd.DataFrame(preds2[0:15]
                   ,columns = CLASSES_LIST).T
df2.style.format("{:.2%}").background_gradient(cmap=cm)
```

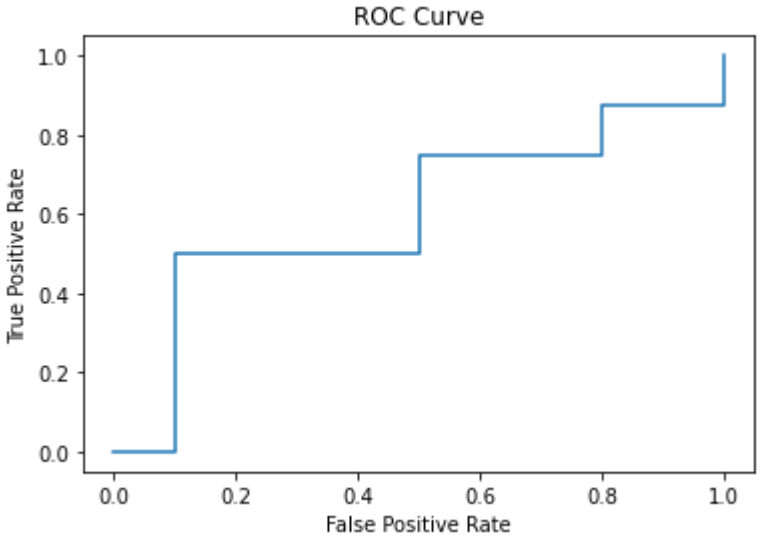
Out[208]:

	0	1	2	3	4	5	6	7	8	9	10
Negative	80.45%	3.08%	1.88%	41.83%	69.65%	3.70%	91.46%	20.16%	5.19%	9.42%	83.89%
Positive	19.55%	96.92%	98.12%	58.17%	30.35%	96.30%	8.54%	79.84%	94.81%	90.58%	16.11%

In [209…
```
fpr, tpr, _ = roc_curve(y2, preds2[:,1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.title('ROC Curve')

precision = precision_score(y2, y_pred2)
recall = recall_score(y2, y_pred2)
auc = roc_auc_score(y2, preds2[:,1])
print("AUC = ", round(auc, 3))
```

```
print("Precision = ", round(precision,3))
print("Recall = ", round(recall,3))
```

```
AUC =  0.6
Precision =  0.5
Recall =  0.75
```



In [ ]: