Audience Reviews

The Last Airbender Season 1 (Netflix)

- This project involves analyzing audience reviews from Rotten Tomatoes for season 1 of The Last Airbender. At the time of writing

this, the audience score is 71%. That's like 3.5 stars. That could mean people generally liked it, or overall people think season 1 was pretty good. However, this metric alone is not very informative.

- The goal is to try and find out what people liked or disliked about the first season. The idea here is to find words of high interest

from the reviews, select some of those words as topics to be explored, and analyze only the parts of the reviews containing those topics.

- I noticed that in these reviews, thoughts and opinions are often split up with periods, exclamation points and question marks. So that is how strings containing our topics will be split.

- The sentiments are being determined with a Logistic Regression pipeline which was created in the IMDB_Logistic_Regression.

notebook

- I scraped reviews from Rotten Tomatoes and that is the data being used.

Contents

2. Select Topics of High Interest
3. Subset Reviews Around Selected Topics
4. Sentiment Analysis
5. Results
6. Conclusion

Libraries

```
In [1]: import pandas as pd
        import joblib

        import matplotlib.pyplot as plt
        from wordcloud import WordCloud

        from langdetect import detect

        import re
        import string
        from nltk.corpus import stopwords
        from collections import Counter
        import itertools
        from nltk.tokenize import word_tokenize
        from nltk.stem.porter import PorterStemmer
        import nltk
```

Data

```
In [2]: df = pd.read_csv(r'C:\Users\baile\Downloads\last_airbender.csv')
```

```
In [3]: df.rename(columns={'audience-reviews__review':'reviews'}, inplace=True)
        print(f'length of df: {len(df)}')
        df.head()
```

```
length of df: 1360
```

Out[3]:

	reviews
0	It was well done as compared to the anime. Cas...
1	For the most part this hits the spot. Don’t ge...
2	Me a encantado es difícil recrear una serie de...
3	Overall, it's a pretty meh show that had a lot...
4	I think the show did a lot of things well. I l...

Only Want Reviews in English

In [4]:

```
in_english = []
for i in df.reviews:
    if detect(i) == 'en':
        in_english.append(i)
    else:
        pass

data = {'reviews':in_english}
df = pd.DataFrame(data)
print(f'length of df: {len(df)}')
df.head()
```

```
length of df: 1278
```

Out[4]:

	reviews
0	It was well done as compared to the anime. Cas...
1	For the most part this hits the spot. Don’t ge...
2	Overall, it's a pretty meh show that had a lot...
3	I think the show did a lot of things well. I l...
4	I understand it can't be exactly the same as t...

Select Topics of High Interest

```
In [5]: stop_words = set(stopwords.words('english'))

        filtered_words = [
            word.lower()
            for review in df.reviews
            for word in word_tokenize(review)
            if word.lower() not in stop_words and word not in string.punctuation]

        Counter(filtered_words).most_common(150)
```

```
Out[5]: [('show', 965),
         ('original', 827),
         ('series', 626),
         ('’', 568),
         ("'s", 522),
         ("n't", 501),
         ('like', 500),
         ('season', 461),
         ('good', 413),
         ('great', 410),
         ('story', 405),
         ('characters', 392),
         ('really', 364),
         ('adaptation', 362),
         ('action', 318),
         ('live', 302),
         ('animated', 290),
         ('better', 267),
         ('much', 265),
         ('acting', 248),
         ('character', 247),
         ('see', 241),
         ('aang', 234),
         ('2', 230),
         ('well', 224),
         ('made', 222),
         ('avatar', 222),
         ('cartoon', 211),
         ('watch', 207),
         ('would', 206),
         ('think', 205),
         ('one', 205),
         ('love', 198),
         ('also', 196),
         ('could', 196),
         ('episodes', 195),
         ('get', 181),
         ('overall', 180),
         ('loved', 179),
         ('changes', 176),
         ('time', 174),
         ('even', 164),
         ('actors', 160),
         ('bad', 158),
         ('make', 158),
         ('way', 155),
         ('feel', 152),
         ('amazing', 152),
         ('casting', 149),
         ('still', 149),
         ('movie', 146),
         ('enjoyed', 145),
         ('things', 144),
         ('episode', 141),
         ('lot', 140),
         ('first', 140),
         ('new', 138),
         ('perfect', 137),
         ('watching', 136),
         ('fan', 134),
         ('hope', 131),
         ('people', 131),
         ('zuko', 130),
         ('world', 126),
         ('cast', 124),
         ('bending', 124),
         ('scenes', 121),
         ('little', 116),
         ('job', 114),
         ('cgi', 114),
         ('katara', 110),
         ('sokka', 109),
         ("'m", 108),
         ('done', 107),
         ('effects', 107),
         ('many', 107),
         ('material', 106),
         ('plot', 106),
         ('animation', 102),
         ('version', 101),
         ('bit', 101),
         ('writing', 100),
         ('fun', 96),
         ('wait', 96),
         ('seasons', 93),
         ('moments', 93),
         ('fans', 93),
         ('felt', 92),
         ('watched', 92),
         ('source', 91),
         ('pretty', 91),
         ('iroh', 90),
         ('best', 90),
         ('know', 89),
         ('everything', 88),
         ('far', 88),
         ('next', 87),
         ('times', 84),
         ('looking', 82),
         ('need', 78),
         ('development', 78),
         ('``', 78),
         ("''", 78),
         ('forward', 78),
         ('definitely', 77),
         ('dialogue', 77),
         ('...', 77),
         ('second', 76),
         ('feels', 76),
         ('look', 76),
         ('remake', 75),
         ('life', 74),
         ('hard', 73),
         ('going', 72),
         ('visuals', 72),
         ('though', 72),
         ('thought', 72),
         ('want', 70),
         ('ca', 69),
         ('1', 68),
         ('never', 68),
         ('absolutely', 68),
         ('different', 68),
         ('understand', 67),
         ('changed', 67),
         ('however', 67),
         ('anime', 66),
         ('go', 66),
         ('true', 66),
         ('say', 66),
         ('every', 65),
         ('something', 65),
         ('give', 65),
         ('last', 65),
         ('two', 64),
         ('actually', 64),
         ('8', 64),
         ('main', 62),
         ('enjoy', 62),
         ('work', 61),
         ('live-action', 60),
         ('part', 59),
         ('sense', 59),
         ('azula', 59),
         ('rushed', 59),
         ('sure', 59),
         ('makes', 59),
         ('thing', 58),
         ('come', 58),
         ('without', 57)]
```

words commonly used together

In [7]:
```python
tokenized_reviews = df['reviews'].apply(lambda x: [word for word in word_tokenize(str(x))if word.lower() not in stop_words])

all_tokens = [token for tokens_list in tokenized_reviews for token in tokens_list]
text_object = nltk.Text(all_tokens)
```

In [8]:
```python
text_object.collocations()
```

```
live action; animated series; source material; character development;
Last Airbender; looking forward; special effects; second season;
original series; feel like; Looking forward; next season; great job;
wait season; really enjoyed; feels like; open mind; n't wait; fire
nation; 2010 movie
```

- Topics will be 'characters' (392), 'story' (405), 'acting' (248), 'dialogue' (77) and 'cgi' (114)

Split by Topics

In [11]:
```
reviews_split = [re.split('[.!?]', x) for x in df.reviews]

story      = [part for sublist in reviews_split for part in sublist if 'story' in part]
characters = [part for sublist in reviews_split for part in sublist if 'character' in part or 'characters' in part]
acting     = [part for sublist in reviews_split for part in sublist if 'acting' in part]
cgi        = [part for sublist in reviews_split for part in sublist if 'cgi' in part]
dialogue   = [part for sublist in reviews_split for part in sublist if 'dialogue' in part]
```

In [12]:
```
all_reviews = story + characters + acting + cgi + dialogue
df = pd.DataFrame({'reviews': all_reviews})
print(f'Length of df: {len(df)}')
```

```
Length of df: 1426
```

- The new df is actually longer than the original as some sentences are repeated. This happened because some of the sentences contain more than 1 of the selected topics.

- I am keeping these duplicates because each sentence that references multiple topics is relevant to each of those topics.

Manually Labeling Samples of the Data

In [13]:
```
characters_sample = characters[:5]
story_sample = story[:5]
acting_sample = acting[:5]
dialogue_sample = dialogue[:5]
cgi_sample = cgi[:5]
```

```
In [14]: cgi_sample
```

```
Out[14]: [' Great cgi and beautiful fights',
          ' However the cgi was pretty cringy in some parts',
          ' The cgi is great',
          " depth of characters, pace, the cgi, that's a few that need to improve",
          'Most effort is put into finding look a like actors and cgi']
```

characters_labels = [1,1,1,0,0]

story_labels = [1,1,1,0,1]

acting_labels = [0,0,0,1,1]

dialogue_labels = [0,1,1,0,0]

cgi_labels = [1,0,1,0,0]

- 1 for positive, 0 for negative

Class for Preparing Text

```
In [15]: class Prep_Text:
             def __init__(self, text):
                 self.text = text

             def to_lower(self):
                 self.text = [x.lower() for x in self.text]

             def remove_punc(self):
                 self.text = [''.join(char for char in x if char not in string.punctuation) for x in self.text]

             def remove_stop_words(self):
                 stop_words = set(stopwords.words('english'))
```

```
        self.text = [' '.join(x for x in word_tokenize(words) if x not in stop_words) for words in self.text]

    def get_stems(self):
        porter = PorterStemmer()
        self.text = [' '.join(porter.stem(word) for word in word_tokenize(char)) for char in self.text]
        return self.text
```

Check Scores for Manually Labeled Samples

In [16]:
```
my_samples = [characters_sample, story_sample, acting_sample, dialogue_sample, cgi_sample]
```

In [17]:
```
tuned_model = joblib.load('tuned_logreg_model_imdb.pkl')
```

In [18]:
```
predictions = []
for sample in my_samples:
    prep = Prep_Text(sample)
    prep.to_lower()
    prep.remove_punc()
    prep.remove_stop_words()
    cleaned_text = prep.get_stems()

    predictions.append(tuned_model.predict(cleaned_text))
```

In [19]:
```
predictions
```

Out[19]:
```
[array([1, 1, 1, 0, 0], dtype=int64),
 array([1, 1, 1, 0, 0], dtype=int64),
 array([0, 0, 0, 1, 1], dtype=int64),
 array([0, 0, 0, 0, 1], dtype=int64),
 array([1, 0, 1, 0, 0], dtype=int64)]
```

- Comapring these labels to my labels above, the model got 21/25. That is 84% accuracy.

- I feel confident in applying this model to get an idea of how the audience felt about the topics of interest.

Sentiment Analysis

prepare the lists of topics

In [20]:
```
topics_dict = {"characters": characters,
               "story": story,
               "acting": acting,
               "dialogue": dialogue,
               "cgi": cgi
              }
```

In [21]:
```
preprocessed_topics = {}

for name, topic_list in topics_dict.items():
    prep = Prep_Text(topic_list)
    prep.to_lower()
    prep.remove_punc()
    prep.remove_stop_words()
    cleaned = prep.get_stems()

    preprocessed_topics[name] = cleaned
```

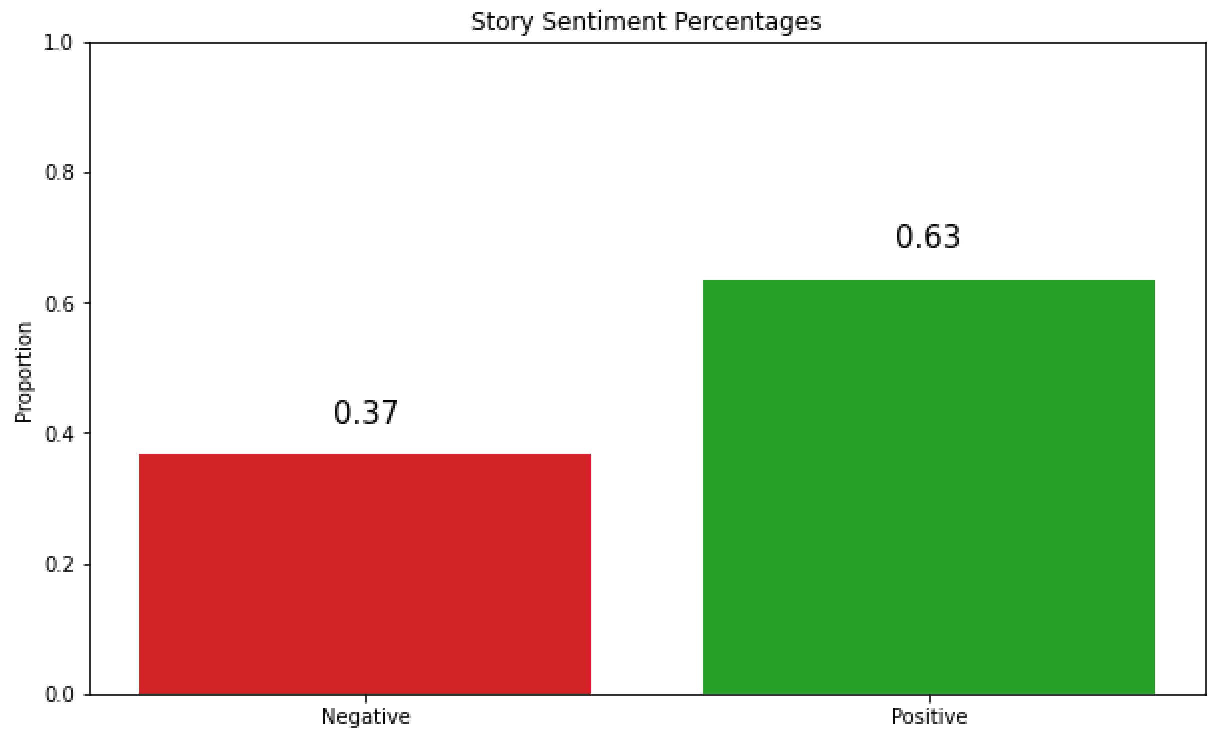
class to predict and plot sentiment scores

In [22]:
```
class sentiment:
    def  __init__(self, topic):
        self.topic = topic
        self.pipeline = joblib.load('tuned_logreg_model_imdb.pkl')

    def predict_sentiment(self):
        '''Function to determine if sentiment is negative(0) or positive(1)'''
        predictions = self.pipeline.predict(self.topic)
        return predictions

    def plot_scores(self):
        '''Function to plot sentiment scores'''
        scores = self.predict_sentiment()
        all_count = len(scores)
        negative_percent = scores.tolist().count(0) / all_count
        positive_percent = scores.tolist().count(1) / all_count
        plt.figure(figsize=(10,6))
        plt.text(0, negative_percent + 0.05, round(negative_percent, 2), fontsize=15, color='black', ha='center')
        plt.text(1, positive_percent + 0.05, round(positive_percent, 2), fontsize=15, color='black', ha='center')

        plt.bar(['Negative', 'Positive'], [negative_percent, positive_percent], color=['tab:red', 'tab:green'])
        plt.ylim(0, 1)
        plt.ylabel('Proportion')
```

Results

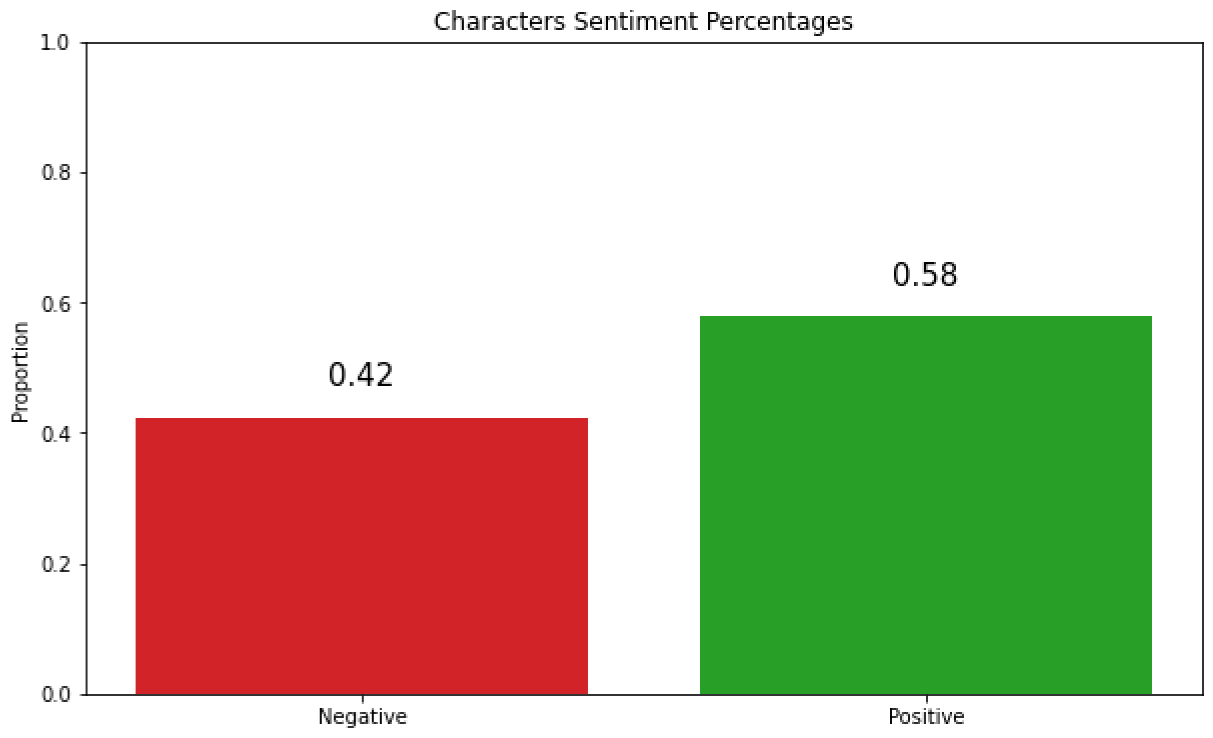
Story

```
In [23]: sentiment(preprocessed_topics["story"]).plot_scores()
         plt.title('Story Sentiment Percentages')
         plt.show()
```



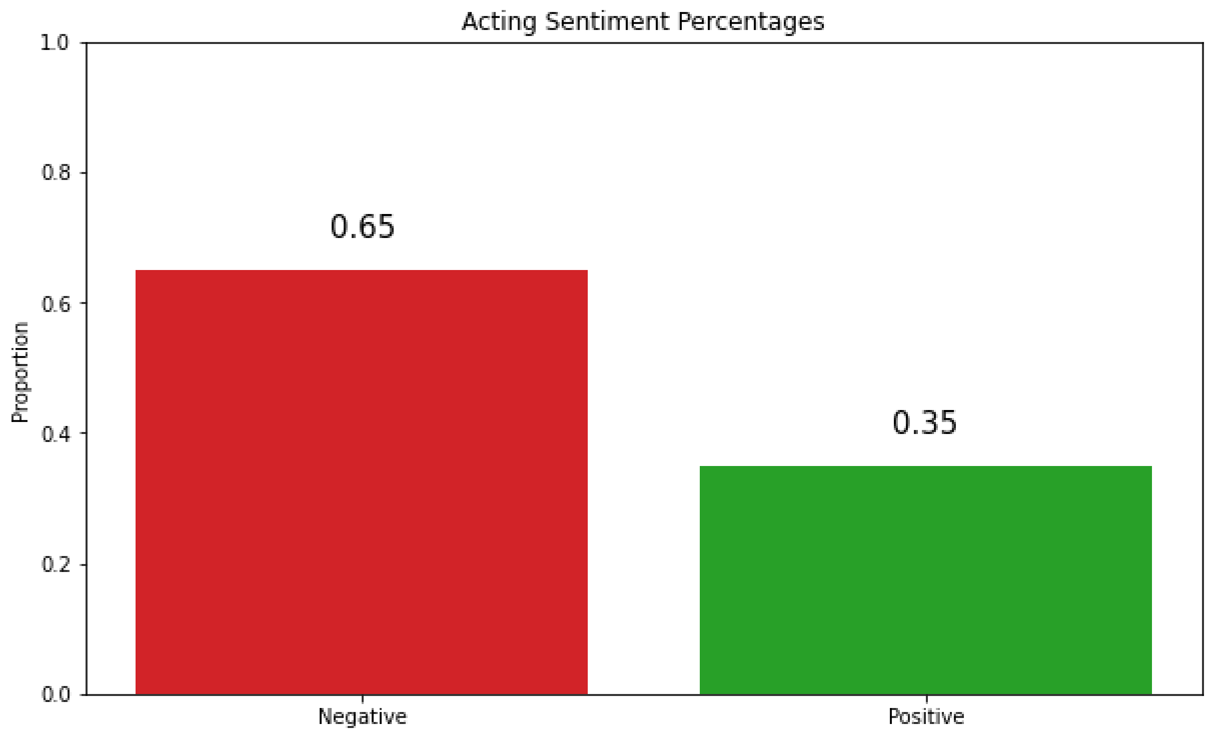
Characters

```
In [24]: sentiment(preprocessed_topics["characters"]).plot_scores()
         plt.title('Characters Sentiment Percentages')
         plt.show()
```



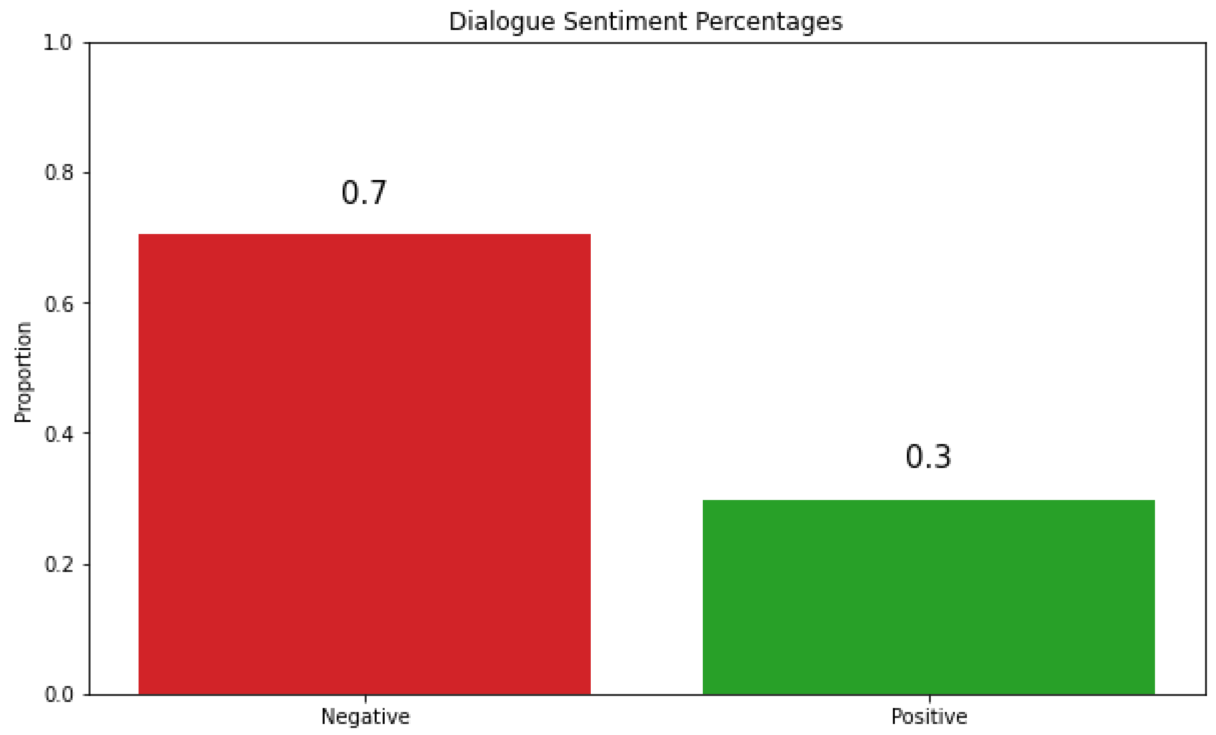
Acting

```
In [25]: sentiment(preprocessed_topics["acting"]).plot_scores()
         plt.title('Acting Sentiment Percentages')
         plt.show()
```



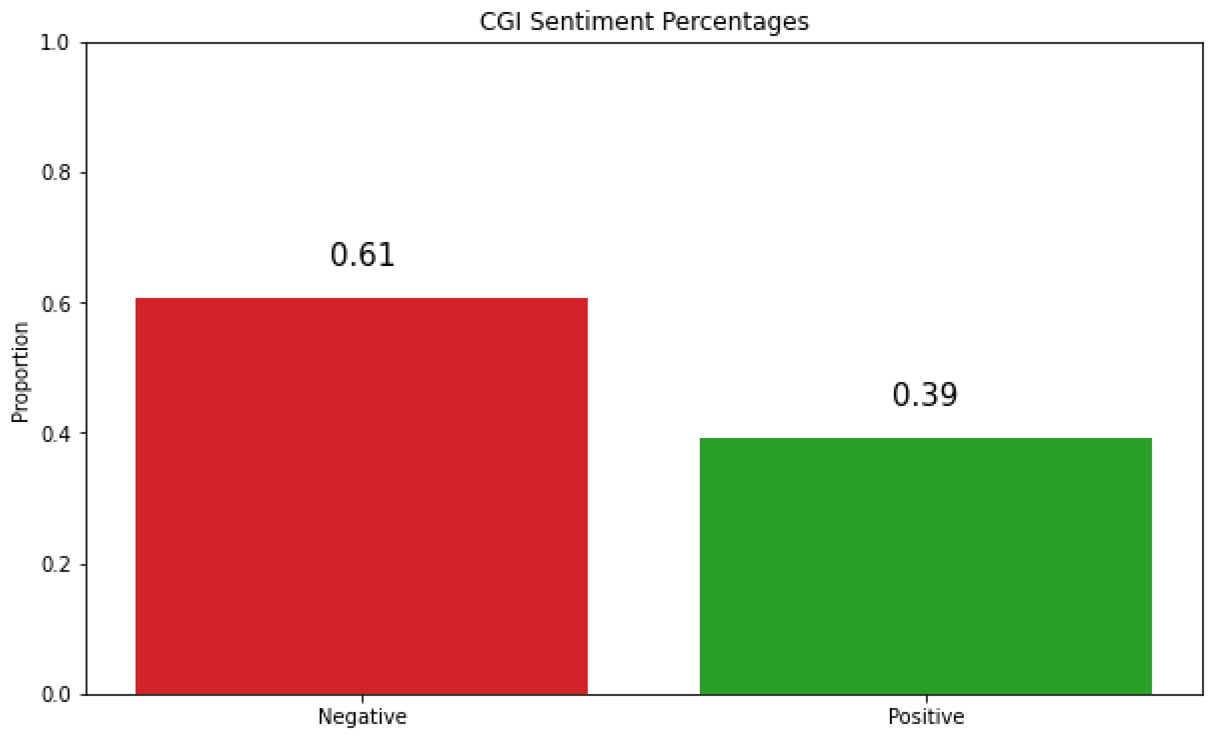
Dialogue

In [26]:
```
sentiment(preprocessed_topics["dialogue"]).plot_scores()
plt.title('Dialogue Sentiment Percentages')
plt.show()
```



CGI

```
In [27]: sentiment(preprocessed_topics["cgi"]).plot_scores()
         plt.title('CGI Sentiment Percentages')
         plt.show()
```



- It looks like the audience has mostly negative opinions about the acting, dialogue and CGI.

Top 5 Adjectives

In [28]:
```
excluded_words = {'more', 'same', 'less', 'most', 'many', 'few', 'like', 'original',
                'little', 'main', 'much', 'new'}

def adjectives(topic):
    all_adjectives = []
    for sentence in topic:
        tokens = nltk.word_tokenize(sentence)
        tagged = nltk.pos_tag(tokens)
        part_of_speech = [word.lower() for word, pos in tagged
                         if pos in ['JJ', 'JJR', 'JJS'] and word.lower() not in excluded_words]
        all_adjectives.extend(part_of_speech)
    return Counter(all_adjectives).most_common(5)
```

In [32]:
```
print(f'CGI: {adjectives(cgi)}\n')
print(f'Acting: {adjectives(acting)}\n')
print(f'Dialogue: {adjectives(dialogue)}\n')
print(f'Characters: {adjectives(characters)}\n')
print(f'Story: {adjectives(story)}\n')
```

```
CGI: [('good', 4), ('beautiful', 3), ('great', 3), ('unnecessary', 2), ('empathetic', 1)]

Acting: [('bad', 35), ('good', 26), ('great', 21), ('better', 14), ('terrible', 12)]

Dialogue: [('bad', 9), ('terrible', 8), ('good', 5), ('great', 4), ('beautiful', 3)]

Characters: [('great', 38), ('animated', 31), ('good', 28), ('live', 22), ('bad', 21)]

Story: [('good', 36), ('great', 35), ('animated', 33), ('live', 23), ('overall', 20)]
```

Conclusion

It appears that what could improve the show the most in the opinion of the audiesnce is better dialogue, acting and cgi. The top 5 adjectives for the topics were not informative.