Machine Learning in Python - Group Project 1

Due Friday, March 10th by 16.00 pm.

include contributors names here (such as Name1, Name2, ...)

General Setup

In [1]:
```
# Data libraries, Plotting libraries, Plotting defaults, NLP Libraries,
# sklearn modules that are necessary
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from textblob import TextBlob
import sklearn
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
# Text egenration
#import gpt_2_simple as gpt2
import os
import re
import warnings
# This stops warnings from the first model being output
warnings.filterwarnings('ignore')
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]:
```
True
```

Abstract

We aim to predict the IMDb ratings of the popular television show ”The Office” using machine learning techniques. We gather data from two sources: the IMDb ratings dataset and the transcript dialogue data. We preprocess and clean the data, extract additional features, and perform exploratory data analysis to identify patterns and correlations in the data. Then, we develop and evaluate several predictive models, including Lasso, Ridge, and Forest decision trees. We also use PCA reduction to identify influential factors, including settings, actors, directors, and writers, that contribute to high ratings. We choose the best performing model, and interpret the results to provide insights and recommendations to NBC Universal. In addition, we use NLP techniques for sentimental analysis of the script and identify how dialogue and humor impact ratings. Finally, we leverage GPT-2 to generate a new episode script based on the most suitable settings associated with high ratings. Our research provides a comprehensive analysis of the factors that contribute to the success of The Office and offers recommendations to guide the production of a successful reunion episode

1. Introduction

1.1 KEY CONTRIBUTIONS

We pre-process and clean the data, extract additional features, and perform exploratory data analysis to identify patterns and correlations in the data. We develop and evaluate several predictive models, choose the best-performing model, and interpret the results to provide insights and recommendations to NBC Universal, including a Lasso based Linear Regression Model and a Forest Decision Tree Model. We use GPT-2 to generate a new episode script based on the most suitable settings associated with high ratings

1.2 DATA AND STUDY AREA

We gather data from two sources: the IMDb ratings dataset and the transcript dialogue data. The IMDb ratings dataset contains information such as season and episode numbers, episode names, director and writer information, IMDb ratings, total votes, air dates, and character information. The transcript dialogue data includes information on the season number, episode number, scene number, speaker, and line text. We focus on analyzing and modeling the relationship between these variables and the IMDb ratings of each episode.The available data includes information such as season and episode numbers, episode names, director and writer information, IMDb ratings, total votes, air dates, and character information. Additionally, external data sources such as transcript dialouge were added. The IMDB rating dataset was given can be found from IMDb official website and the transciprt data was found from reddit subreddit concerning datasets.

1.3 WORKFLOW OF ANALYSIS

Data collection, The first step of our workflow involved collecting data from two sources: the IMDb ratings dataset and the transcript dialogue data of the television show ”The Office”. Secondly, data preprocessing, after collecting the data, we performed data preprocessing and cleaning to remove any missing values, duplicates, or irrelevant features. We also extracted additional features, such as sentiment scores and character interactions, to improve the predictive power of our models. Thirdly, exploratory data analysis we performed exploratory data analysis to identify patterns and correlations in the data. We used visualization techniques to gain insights into the relationships between different features and the IMDb ratings of the episodes. Fourthly,feature selection and reduction to reduce the dimensionality of the data and identify the most influential features, we used principal component analysis (PCA) and feature selection techniques. This step allowed us to focus on the most important factors that contribute to the success of an episode. Fifth, model development and evaluation With the preprocessed data and selected features, we developed and evaluated several predictive models, including Lasso regression, and random forest decision trees. We used cross-validation techniques to evaluate the performance of the models and select the best-performing one. Lastly, results interpretation we interpreted the results of our models to provide insights and recommendations to NBC Universal for the production of a successful reunion episode. We identified the settings, actors, directors, and writers that were associated with high IMDb ratings and used GPT-2 to generate a new episode script based on these findings.By following this pipeline, we were able to provide NBC Universal with valuable insights into the factors that contribute to the success of an episode of ”The Office” and generate a new episode script that leverages these insights

2. Exploratory Data Analysis and Feature Engineering

2.1RELATED WORK

Previous research has explored the relationship between television show ratings and various factors such as genre, storyline, and character development. One study found that shows with more diverse casts tend to have higher ratings (Hunt, Ramon- ´ Jordan, Wiseman, 2020). Another study found that shows with ´ longer runtimes tend to have higher ratings, as they provide more time for character development and storyline progression (Rosenberg VanMeter, 2019).In terms of the impact of specific writers and directors on show ratings, research has shown mixed results. Some studies have found that certain writers and directors consistently produce shows with higher ratings, while others have found no significant correlation between specific individuals and ratings (Bruns Highfield, 2017; Zhang, Wang, Yu, 2016). One study specifically focused on The Office, examining the relationship between character development and ratings. The study found that episodes with more character development tend to have higher ratings, as viewers become more invested in the characters and their storylines (Carrell Jerit, 2012).Overall, while there have been many studies examining the factors that influence television show ratings, there is still much to be learned about the specific factors that contribute to the success of a show. To predict TV show ratings, various methods have been used in the literature, ranging from simple linear regression models to complex machine learning techniques. For example, Wu et al. (2017) used a support vector regression model to predict the ratings of TV shows, while Fagnan et al. (2019) used a gradient boosting regression model. Lee and Kim (2016) used a multiple linear regression model and found that the number of viewers, the show’s genre, and the length of the show were the most significant predictors of TV show ratings.Other studies have used sentiment analysis techniques to predict TV show ratings. For instance, Yu et al. (2017) used a sentiment analysis approach to predict the ratings of TV shows by analyzing the emotions conveyed by the tweets related to the shows. Similarly, Seo et al. (2018) used a combination of social media sentiment analysis and machine learning techniques to predict the ratings of TV shows. In our study, we will use a combination of these methods to predict the ratings of The Office TV show. We will use supervised machine learning techniques, sentiment analysis, and content analysis to predict the show’s ratings. We will also engineer additional features from the data, such as sentiment scores, character interaction metrics, and episode themes, to improve the model’s accuracy

2.2 DATA INFORMATICS

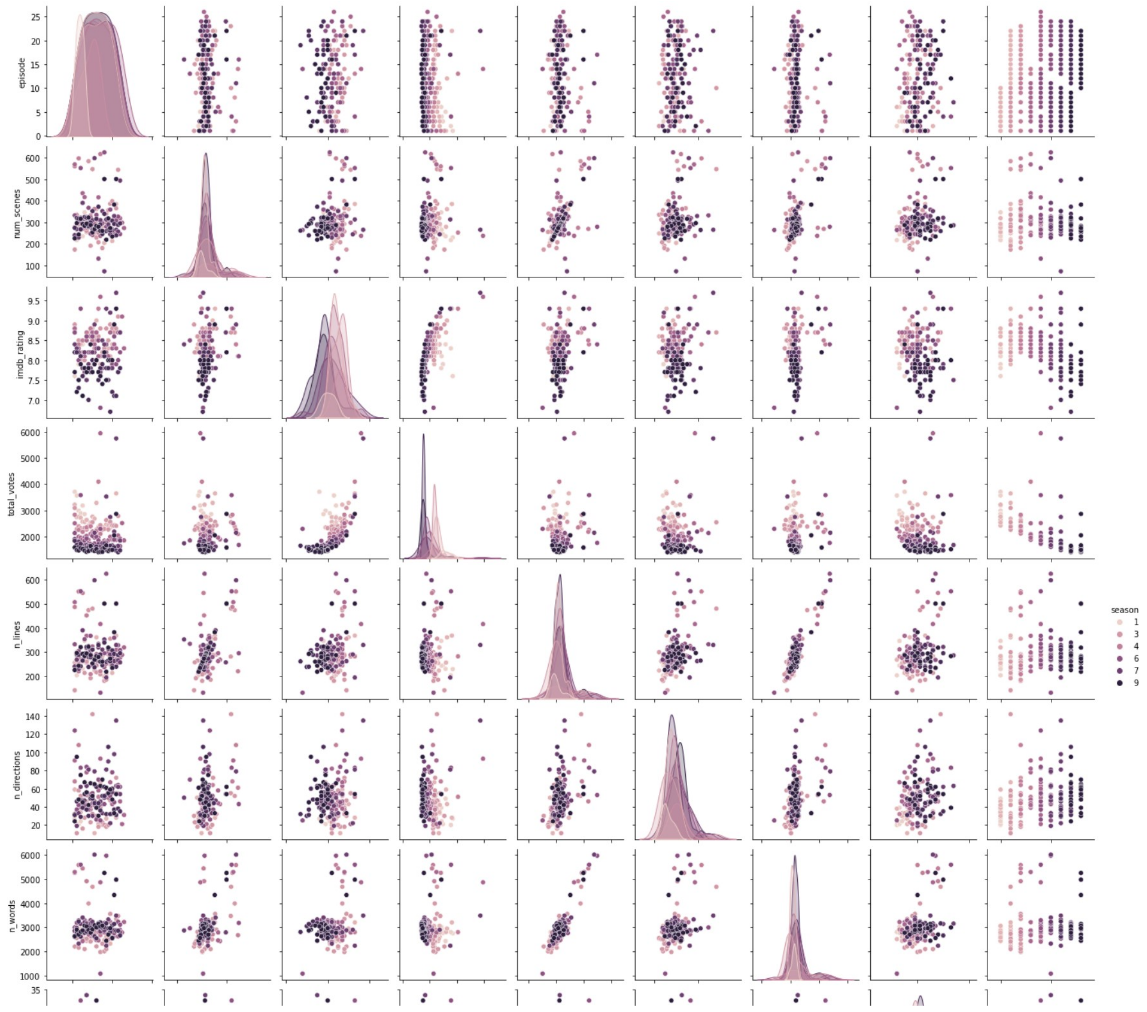
2.2.1 Cleaning and Preparing

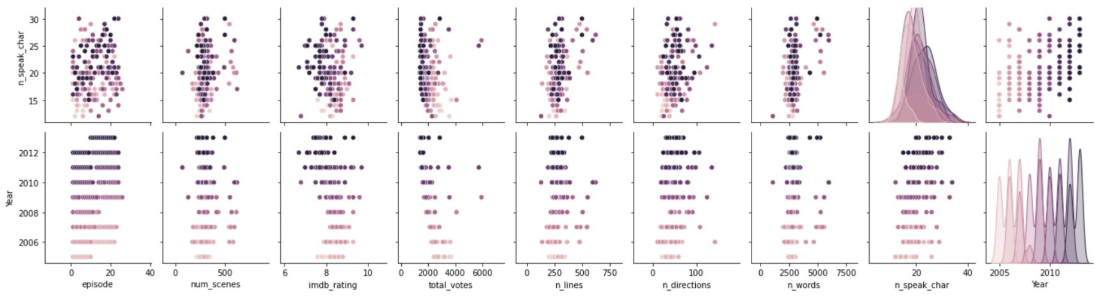
The cleaning and preparation of the rating dataset involved dropping any rows with missing values, converting numeric columns to float, converting the date column to a datetime object and extracting the year from it. Additionally, the string values of director, writer and actors were changed to a comma separated list for each entry.. For the transcript dataset, missing rows were dropped, deleted scenes were removed, and lines for non-main characters were excluded. The line text was then cleaned by removing stop words, special characters, digits, and multiple spaces. Furthermore, we merged the datasets.

In [2]:
```
# Cleaning and preapring rating dataset #
imdb_data = pd.read_csv("the_office.csv")

# Drop any rows with missing values
imdb_data.dropna(inplace=True)

# Convert the numeric comuns to float
imdb_data["imdb_rating"] = imdb_data["imdb_rating"].astype(float)
imdb_data["total_votes"] = imdb_data["total_votes"].astype(float)
imdb_data['n_lines'] = imdb_data["n_lines"].astype(float)
imdb_data['n_directions'] = imdb_data["n_directions"].astype(float)
imdb_data['n_words'] = imdb_data["n_words"].astype(float)
imdb_data['n_speak_char'] = imdb_data["n_speak_char"].astype(float)

# Convert the Date column to a datetime object
imdb_data["air_date"] = pd.to_datetime(imdb_data["air_date"])

# Extract the year from the Date column
imdb_data["Year"] = (imdb_data["air_date"].dt.year).astype(int)

# Drop the Date column
imdb_data.drop("air_date", axis=1, inplace=True)

# Remove any whitespace from the Director, Writers and main characters colu
imdb_data["director"] = imdb_data["director"].str.strip().str.split(';')
imdb_data["writer"] = imdb_data["writer"].str.strip().str.split(';')
imdb_data['main_chars'] =imdb_data["main_chars"].str.strip().str.split(';')

# Check the cleaned data
#-----------------
#print(imdb_data.head())
#-----------------
# Cleaning and preapring trancript dataset #
# read the transcript data from a CSV file
transcript = pd.read_csv('the-office-lines - scripts.csv')

# Drop any rows with missing values
transcript.dropna(inplace=True)

# remove deleted scenes
transcript = transcript[transcript['deleted'] == False]

# drop the 'deleted' column
transcript = transcript.drop(columns=['deleted'])

# Include those lines for main characters only found in the imdb data set
imdb_exploded = imdb_data.explode('main_chars')
unique_chars = imdb_exploded['main_chars'].unique()
transcript[transcript['speaker'].isin(unique_chars)]

def clean_transcript(line_text):
    # convert to lowercase
    transcript = line_text.lower()
    # remove stop words
    stop_words = set(stopwords.words("english"))
    stemmer = PorterStemmer()
    transcript = ' '.join([word for word in transcript.split() if word not 
    # remove special characters and digits
    transcript = re.sub(r"[^a-zA-Z\s]+", "", transcript)
    # convert to lowercase
    transcript = transcript.lower()
    # remove leading/trailing whitespaces
    transcript = transcript.strip()
    # replace multiple spaces with a single space
    transcript = re.sub(r"\s+", " ", transcript)
    return transcript

transcript_cleaned=transcript
transcript_cleaned['line_text'] = transcript_cleaned['line_text'].apply(cle

#Check the cleaned data
#------------
#print(transcript_cleaned.head())
#------------
```

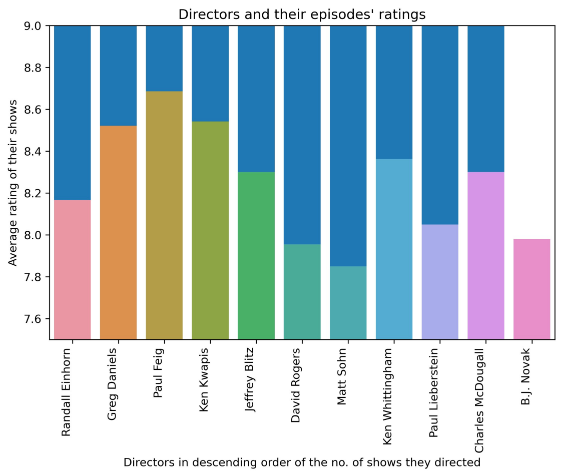
In [3]:
```
#Grouping Transcript Data per Season and Epsiode
transcript_grouped = transcript_cleaned.groupby(["season", "episode"]).agg(
transcript_grouped.rename(columns={"scene": "num_scenes", "line_text": "tra

merged_data = pd.merge(transcript_grouped,imdb_data, on=['season', 'episode
fig1 = sns.pairplot(
    data=merged_data,
    aspect=.85,
    hue='season');
fig1.savefig("fig1- CorrelationPlot.png")
plt.close(fig1.fig)
```

2.2.2 Explanatory Data Analysis





Upon conducting exploratory data analysis (EDA) on the TV show dataset, several trends and patterns emerged. The correlation plot as shown in Fig. 1, showed that total votes were positively correlated with IMDB ratings in a moon-shaped pattern. Additionally, it was observed that most of the numerical data in the scatter plots showed a white noise pattern, while the histograms showed high skewness. Hence, standardizing the numerical data is necessary.



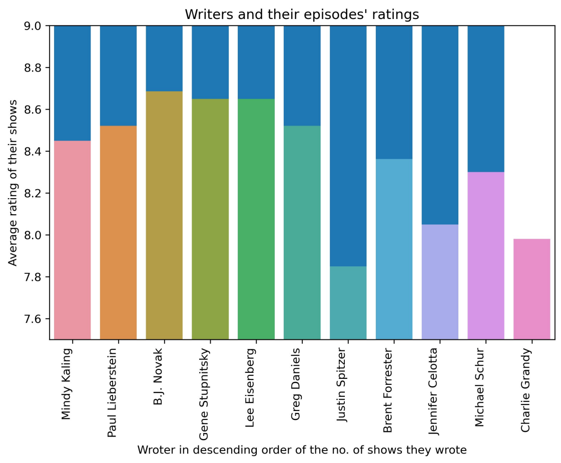
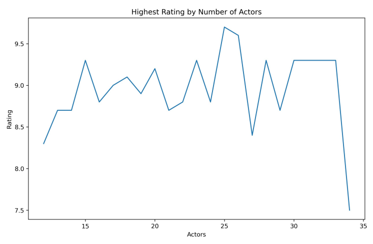
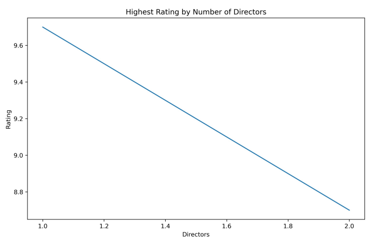


Fig 2: Directors and their episodes' ratings

Fig 3: Writers and their episodes' ratings

Further analysis showed that directors have a significant impact on episode ratings as shwon in Fig.2 , with Paul Feig being associated with the highest-rated episode (8.7), followed by Greg Daniels and Ken Kwapis (8.5). On the other hand, Charlie Grandy directed the lowest-rated episodes (7.7). Writers also have a considerable influence on episode ratings, with Env Stupnitsky and Lee Eisenberg having the highest average rating (8.6), followed by Greg Daniels (8.5).





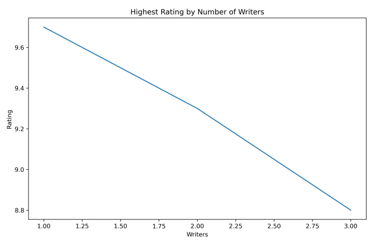


Fig 4:Actor ratings **Fig 5:Director ratings** **Fig 6:Writer ratings**

Coherence in the script plays a crucial role in determining episode ratings, as having more than one writer or director lowers the ratings by 0.4 and 1.0 points, respectively. Furthermore, episodes with 9-12 or 13-15 actors had the highest average ratings of 9.4.

img src="interaction_network.png" Additionally, Michael, Dwight, and Jim had the most lines spoken throughout the show and more interactions between Michael, Jim, Pam, Dwight, Andy, and Angela tended to have higher rating

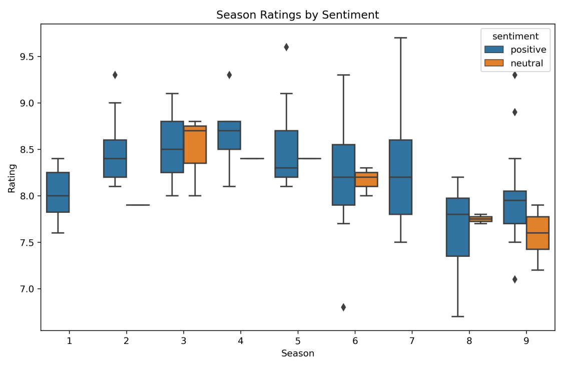


Fig 5: Sentiment Analysis of Script Averaged by Season

Moreover, scene number and episode duration also play a crucial role in determining ratings. Episodes with 230-270 scenes had an average rating of 8.2, while episodes with 130-200 scenes had an average rating of 8.7. Episodes with 400-500 scenes had the highest average rating of 9.3. Additionally, episodes with a duration of 45-55 minutes had the highest ratings.

Finally, the sentimental analysis score of transcripts has an overall positive and neutral effect on season ratings, with more positive sentiment leading to higher ratings. Episodes with lower variations between sentiments had higher scores. Based on the EDA findings, we decided to drop 'nwords' and 'nlines' because they cause numerous amounts of multicollinearity between predictor variables. Several feature engineering techniques can be used to improve the predictive power of the model. Standardizing the numerical data, engineering features related to directors, writers, and actors, such as their past performances, and the number of interactions between them can improve the accuracy of the model. Additionally, incorporating features such as scene number and duration can also help in predicting episode ratings. Finally, sentiment analysis can be improved by using more advanced algorithms and techniques to extract meaningful insights from the transcript data

In [4]:
```
# Meta data
#print(transcript.info())

# Display summary statistics
#print(transcript.describe())

#NOTE:plots below are commented out, this is to avoid them rendering in the
# These plots are used in the write up.

#Who spoke most out of the actors
import matplotlib.pyplot as plt

# Get the top 10 characters based on the number of lines they spoke
top_characters = transcript.groupby('speaker').size().sort_values(ascending

# Plot a bar chart of the top characters
#plt.bar(top_characters.index, top_characters.values)
#plt.xticks(rotation=45, ha='right')
#plt.xlabel('Character')
#plt.ylabel('Number of Lines Spoken')

# Directors and Avg. Episode Rating
def dir_mean_rating(director, imdb_data):
    mask = []
    for i, row in imdb_data.iterrows():
        if director in row["director"]:
            mask.append(i)

    return imdb_data.loc[mask, "imdb_rating"].mean()

list1 = []
for x in imdb_data["director"].explode().value_counts().head(11).keys():
    list1.append(dir_mean_rating(x, imdb_data))

xs = imdb_data["director"].explode().value_counts().head(11).keys()
ys = list1

#g = sns.barplot(x=xs,y=ys)
#g.set(ylim=(7.5,9))
#g.set_xticklabels(g.get_xticklabels(), rotation=90)
#plt.xlabel("Directors in descending order of the no. of shows they directe
#plt.ylabel("Average rating of their shows")
#plt.title("Directors and their episodes' ratings")
#plt.savefig("dir-epi-ratings.png", dpi=300, bbox_inches='tight')

# Directors and Avg. Episode Rating
def wrot_mean_rating(writer, imdb_data):
    mask = []
    for i, row in imdb_data.iterrows():
        if writer in row["writer"]:
            mask.append(i)

    return imdb_data.loc[mask, "imdb_rating"].mean()

list1 = []
for x in imdb_data["writer"].explode().value_counts().head(11).keys():
    list1.append(dir_mean_rating(x, imdb_data))

xs = imdb_data["writer"].explode().value_counts().head(11).keys()
```

```
ys = list1

#g = sns.barplot(x=xs,y=ys)
#g.set(ylim=(7.5,9))
#g.set_xticklabels(g.get_xticklabels(), rotation=90)
#plt.xlabel("Wroter in descending order of the no. of shows they wrote")
#plt.ylabel("Average rating of their shows")
#plt.title("Writers and their episodes' ratings")
#plt.savefig("wri-epi-ratings.png", dpi=300, bbox_inches='tight')


#Highest rating by scene number, we can group the merged data by scene numb
# Group by scene number and calculate max rating for each scene
max_ratings_by_scene = merged_data.groupby('num_scenes')['imdb_rating'].max

#Highest rating by scene number, we can group the merged data by scene numb
# Group by scene number and calculate max rating for each scene
max_ratings_by_numspeakers = merged_data.groupby('n_speak_char')['imdb_rati

# Plot highest rating by scene number
#plt.figure(figsize=(10,6))
#sns.lineplot(data=max_ratings_by_numspeakers, x='n_speak_char', y='imdb_ra
#plt.title('Highest Rating by Number of Actors')
#plt.xlabel('Actors')
#plt.ylabel('Rating')
#plt.savefig("H-rat-num-actor.png", dpi=300, bbox_inches='tight')

#Number of Directors
merged_data['n_directors'] = merged_data['director'].apply(lambda x: len(x)
max_ratings_by_directornum = merged_data.groupby('n_directors')['imdb_ratin

# Plot highest rating by scene number
#plt.figure(figsize=(10,6))
#sns.lineplot(data=max_ratings_by_directornum, x='n_directors', y='imdb_rat
#plt.title('Highest Rating by Number of Directors')
#plt.xlabel('Directors')
#plt.ylabel('Rating')
#plt.savefig("H-rat-num-director.png", dpi=300, bbox_inches='tight')

#Number of writers
merged_data['n_writers'] = merged_data['writer'].apply(lambda x: len(x))
max_ratings_by_writernum = merged_data.groupby('n_writers')['imdb_rating'].

# Plot highest rating by scene number
#plt.figure(figsize=(10,6))
#sns.lineplot(data=max_ratings_by_writernum, x='n_writers', y='imdb_rating',
#plt.title('Highest Rating by Number of Writers')
#plt.xlabel('Writers')
#plt.ylabel('Rating')
#plt.savefig("H-rat-num-writer.png", dpi=300, bbox_inches='tight')

#Positive and negative sentiments per episode and then visualize rating for
# Calculate sentiment polarity for each line of dialogue
merged_data['polarity'] = merged_data['transcript'].apply(lambda x: TextBlo

# Group by episode and calculate average polarity for each episode
# Categorize episodes as positive, negative or neutral based on average pol
merged_data['sentiment'] = np.where(merged_data['polarity'] > 0.1, 'positiv
                                               np.where(merged_data['polari
# Plot season ratings by sentiment
```

```
#plt.figure(figsize=(10,6))
#sns.boxplot(data=merged_data, x='season', y='imdb_rating', hue='sentiment',
#plt.title('Season Ratings by Sentiment')
#plt.xlabel('Season')
#plt.ylabel('Rating')
#plt.savefig("sentiment_analysis.png", dpi=300, bbox_inches='tight')

#Plot duration versus rating
#sns.boxplot(data=merged_data, x=(merged_data["n_lines"]/(merged_data["n_spe
#plt.figure(figsize=(10,6))
#plt.title('Avg Rating by Duration')
#plt.xlabel('Duration')
#plt.ylabel('Rating')
```

Data Feature Engineering

To generate meaningful inputs, our pipeline consists of two different extractions first we extract useful information from continuous variables and then categorical variables. First pipeline consists of ′n actors′, this feature extracts the number of speakers in each episode, which is stored in the ′n speaker char′ column of the dataset. ′n directors′, this feature calculates the number of directors for each episode, which is obtained by counting the number of directors in the director column of the dataset. ′n writers′, this feature calculates the number of writers for each episode, which is obtained by counting the number of writers in the writer column of the dataset. ′duration′, this feature calculates the duration of each episode based on several factors, including the number of lines, the number of characters speaking, and the number of directions. ′Holiday Episode′, this feature identifies episodes that are related to holidays by checking the season and episode number against a predefined list of holiday episodes. ′M ultiP art Episode′, this feature identifies episodes that have multiple parts by checking the season and episode number against a predefined list of multi-part episodes. Second pipeline consists of the categorical predictor ′writers′, ′directors′, ′actors′ variables are encoded to convert them into a numerical form that can be used for machine learning algorithms. In this case, the categorical variables being encoded are the directors, writers, and main actors.For each of these variables, a function is defined to add a prefix to each category in the variable. For example, ”director.” is added to each director’s name. This is done to avoid any naming conflicts that may arise during the encoding process. Then, the MultiLabelBinarizer() function is used to transform the categorical variables into binary variables. This function creates a binary column for each unique category in the variable, where a value of 1 indicates that the category is present and 0 indicates that it is not present. In addition, ′sentiment score′, this feature calculates the sentiment score for each line of dialogue in the transcript using the TextBlob library using NLP, and then takes the average sentiment score for the entire transcript.

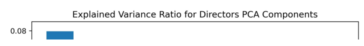
In [5]:
```
# Feature Engineering Merged Dataset #
#Merge Data sets
imdb_data_merged = pd.merge(transcript_grouped,imdb_data, on=['season', 'ep

# Continious Variables Piepline #
#Number of Actors
imdb_data_merged=imdb_data_merged.rename(columns={'n_speak_char':'n_actors'

#Number of Directors
imdb_data_merged['n_directors'] = imdb_data_merged['director'].apply(lambda
#Number of writers
imdb_data_merged['n_writers'] = imdb_data_merged['writer'].apply(lambda x:

# Duration of episode
imdb_data_merged["duration"]=(imdb_data["n_lines"]/(imdb_data["n_speak_char

# define a function to calculate sentiment scores for each line of dialogue
def get_sentiment_score(line):
    blob = TextBlob(line)
    sentiment = blob.sentiment.polarity
    return sentiment

# apply the sentiment score function to the transcript data
imdb_data_merged['sentiment_score'] = imdb_data_merged['transcript'].apply(

# Adding seaosnal Episodes ,https://www.google.com/url?sa=t&rct=j&q=&esrc=s
Holiday_Episode=[[7,9],[6,22],[8,10],[5,11],[6,19],[3,6],[5,18],[9,9],[7,11

def is_holiday_episode(row):
    if [row['season'], row['episode']] in Holiday_Episode:
        return 1
    else:
        return 0

imdb_data_merged['Holiday_Episode'] = imdb_data_merged.apply(is_holiday_epi

#Multi Part Episodes
#imdb_data_merged[imdb_data_merged['episode_name'].str.contains('Part 1|Par
Multi_Part_Episode=[[3,10],[4,1],[4,3],[4,5],[4,7],[5,1],[5,14],[5,16],[5,1

def is_multipart_episode(row):
    if [row['season'], row['episode']] in Multi_Part_Episode:
        return 1
    else:
        return 0

imdb_data_merged['MultiPart_Episode'] = imdb_data_merged.apply(is_multipart

#Standarize nuimerical features
from sklearn.preprocessing import MinMaxScaler

# create a StandardScaler object
scaler = MinMaxScaler()

# select only the numerical columns to be standardized
num_cols = ['total_votes', 'n_lines', 'n_directions', 'n_words','sentiment_
# fit and transform the numerical columns using the scaler
imdb_data_merged[num_cols] = scaler.fit_transform(imdb_data_merged[num_cols
```

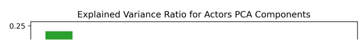
In [6]:
```
# Encode Categorical Predictor Varaibles #

#directors
def add_dir_to_array(arr):
    return ["director." + a for a in arr]

imdb_data_merged["director"] = imdb_data_merged["director"].apply(add_dir_t
mlb = MultiLabelBinarizer()
imdb_data_merged = imdb_data_merged.join(pd.DataFrame(mlb.fit_transform(imdl

#directors
def add_wri_to_array(arr):
    return ["writer." + a for a in arr]
imdb_data_merged["writer"] = imdb_data_merged["writer"].apply(add_wri_to_ar
mlb = MultiLabelBinarizer()
imdb_data_merged = imdb_data_merged.join(pd.DataFrame(mlb.fit_transform(imdl

#actors
def add_act_to_array(arr):
    return ["actor." + a for a in arr]
imdb_data_merged["main_chars"] = imdb_data_merged["main_chars"].apply(add_a
mlb = MultiLabelBinarizer()
imdb_data_merged = imdb_data_merged.join(pd.DataFrame(mlb.fit_transform(imdl

# Two differen Data set for PCA modeling later on
imdb_data_settings=imdb_data_merged
imdb_data_act_writ_dire=imdb_data_merged
imdb_data_rating= imdb_data_merged
# Based on intial EDA we decided to drop
imdb_data_act_writ_dire = imdb_data_act_writ_dire.drop(['episode_name','sea
imdb_data_settings = imdb_data_settings.drop(['episode_name','season','epis
```

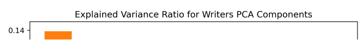
PCA Analysis

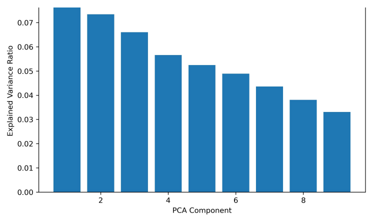
In this study, a PCA analysis was performed on various datasets of IMDb movie data to identify the most important predictors of movie success. Moreover, we partitioned the data into two different sets One dataset is ′imdb data settings′, which will be used for modeling settings influence and the other is ′imdb data act writ dire′, which will be used for modeling the effects of actors, writers, and directors on ratings, furthermore seeeperating directros,writers and actors into mul- tiple dataset consisted of 60 binary columns for director, 40 binary columns for actors, and 17 binary columns for writers. The reason for using PCA is to reduce the dimensionality of the data, which can help to mitigate the risk of overfitting and improve model performance. In this case, the two different datasets were created to capture different aspects of the relationship between the predictors and the target variable. Before applying the PCA, the data is normalized and scaled. Normalization is done to adjust the mean of each feature to 0 and scaling is done to adjust the variance of each feature to

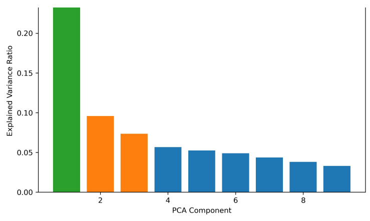
1. This is done to ensure that the PCA is not biased towards any one feature. Then a grid search is performed to find the optimal number of components for the PCA. This is done by training PCA models with different numbers of components (1 to 10) and evaluating each model’s performance using cross- validation of 5 folds. The model with the highest mean test score is selected as the optimal model.











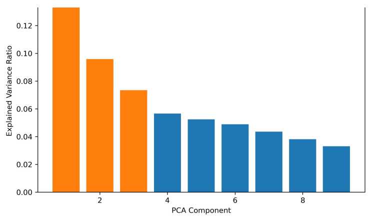


Fig 8(a): Actor PCA **Fig 8(b): Director PCA** **Fig 8(c): Writer PCA**

The first dataset focuses on the more direct and measurable features, such as duration, number of scenes, and number of lines in the transcript. Additionally, the directors, actors, and writers are encoded as PCA components to maintain variability while also focusing on the settings of the episode. The results show that the optimal number of components for the director columns is 10, with a mean test score of 40.359, for the writer columns is 3, with a mean test score of 16.849. for the actor columns is 1, with a mean test score of -8.767. Hence, This means that the information in the original 60 director columns, 40 actor columns and 17 actors columns can be represented using only 10, 3 and 1 component respectively. While still maintaining a high level of accuracy. As shown in Fig. 5, the explained variance ratio for each component is also plotted to show the amount of information retained by each component.

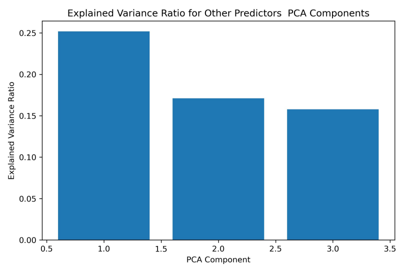


Fig 9: PCA Dimensionality Reduction of Predictors

The second dataset captures the more direct influence of the actors, writers, and directors on the target variable through their presence in the episode. In this dataset, PCA is performed on all the other numerical predictor variables to reduce dimensionality. By using PCA in this way, we can identify the most important components that contribute to the variance in the data, which can improve the accuracy and interpretability of the model. The results show that the optimal number of components for the numerical predictor columns is 3, with a mean test score of 2.137.

In [7]:
```
# PCA Dimensionality Reduction for Director,Writer and Actor Column, follow
PCA_Data = imdb_data_settings.filter(regex="^(director\.|writer\.|actor\.)"

#Directors
director_data = PCA_Data.filter(like='director.')
# normalize the mean of features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(director_data)

# scale the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_normalized)
director_data=X_scaled

# apply PCA
pca = PCA()
pca.fit(director_data)

# tune PCA using grid search
params = {'n_components': [1,2,3,4,5,6,7,8,9,10]}
grid_pca = GridSearchCV(pca, params)
grid_pca.fit(director_data)
pca_best = grid_pca.best_estimator_
X_pca_best = pca_best.transform(director_data)

# Get the explained variance ratio for each component
variance_ratio = pca_best.explained_variance_ratio_
'''
# Create a bar plot of the explained variance ratios
plt.bar(range(1, len(variance_ratio) + 1), variance_ratio)

# Add labels and title
plt.xlabel('PCA Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for Directors PCA Components')
plt.savefig('directors-pca.png', dpi=300, bbox_inches='tight')
'''
#------------------
#print('Optimal Number of Componenets :',pca_best)
#------------------
X_pca_best = pd.DataFrame(X_pca_best, columns=['PCA_Directors'+str(i+1) for
imdb_data_Modeling_settings = imdb_data_settings.filter(regex="^(?!director

# Writers
writers_data = PCA_Data.filter(like='writer.')
# normalize the mean of features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(writers_data)

# scale the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_normalized)

writers_data=X_scaled

# apply PCA
pca = PCA()
pca.fit(writers_data)
```

```
# tune PCA using grid search
params = {'n_components': [1,2,3,4,5,6,7,8,9,10]}
grid_pca = GridSearchCV(pca, params)
grid_pca.fit(writers_data)
pca_best = grid_pca.best_estimator_
X_pca_best = pca_best.transform(writers_data)
#------------------------------
# print evaluation scores for each combination of hyperparameters
#results = grid_pca.cv_results_
#for i in range(len(results['params'])):
   # print("Hyperparameters: ", results['params'][i])
   # print("Mean score: ", results['mean_test_score'][i])
   # print("Standard deviation: ", results['std_test_score'][i])
   # print("\n")
#------------------------------
# Get the explained variance ratio for each component
variance_ratio = pca_best.explained_variance_ratio_
'''
# Create a bar plot of the explained variance ratios
plt.bar(range(1, len(variance_ratio) + 1), variance_ratio)

# Add labels and title
plt.xlabel('PCA Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for Writers PCA Components')
plt.savefig('writers-pca.png', dpi=300, bbox_inches='tight')
'''
#----------------
#print('Optimal Number of Componenets :',pca_best)
#----------------
X_pca_best = pd.DataFrame(X_pca_best, columns=['PCA_Writer'+str(i+1) for i 

imdb_data_Modeling_settings = imdb_data_Modeling_settings.join(X_pca_best)

# Actors
actors_data = PCA_Data.filter(like='actor.')
# normalize the mean of features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(actors_data)

# scale the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_normalized)

actors_data=X_scaled

# apply PCA
pca = PCA()
pca.fit(actors_data)

# tune PCA using grid search
params = {'n_components': [1,2,3,4,5,6,7,8,9,10]}
grid_pca = GridSearchCV(pca, params)
grid_pca.fit(actors_data)
pca_best = grid_pca.best_estimator_
X_pca_best = pca_best.transform(actors_data)
#-------------------------
# print evaluation scores for each combination of hyperparameters
#results = grid_pca.cv_results_
#for i in range(len(results['params'])):
    # print("Hyperparameters: ", results['params'][i])
    # print("Mean score: ", results['mean_test_score'][i])
    # print("Standard deviation: ", results['std_test_score'][i])
    # print("\n")
#----------------------------
'''
# Get the explained variance ratio for each component
variance_ratio = pca_best.explained_variance_ratio_

# Create a bar plot of the explained variance ratios
plt.bar(range(1, len(variance_ratio) + 1), variance_ratio)

# Add labels and title
plt.xlabel('PCA Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for Actors PCA Components')
plt.savefig("actors-pca.png", dpi=300, bbox_inches='tight')
'''
#----------------
#print('Optimal Number of Componenets :',pca_best)
#----------------
X_pca_best = pd.DataFrame(X_pca_best, columns=['PCA_Actors'+str(i+1) for i 
imdb_data_Modeling_settings = imdb_data_Modeling_settings.join(X_pca_best)
```

In [8]:
```
# PCA Dimensionality Reduction for all except Director,Writer and Actor Col
PCA_Data = imdb_data_act_writ_dire.filter(regex="^(?!director.|actor.|write
# normalize the mean of features
scaler = StandardScaler()
X_normalized = scaler.fit_transform(PCA_Data)

# scale the features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_normalized)

PCA_Data=X_scaled

# apply PCA
pca = PCA()
pca.fit(PCA_Data)

# apply PCA
pca = PCA()
pca.fit(PCA_Data)

# tune PCA using grid search
params = {'n_components': [1,2,3,4,5,6,7,8,9,10]}
grid_pca = GridSearchCV(pca, params)
grid_pca.fit(PCA_Data)
pca_best = grid_pca.best_estimator_
X_pca_best = pca_best.transform(PCA_Data)
#--------------------
# print evaluation scores for each combination of hyperparameters
#results = grid_pca.cv_results_
#for i in range(len(results['params'])):
   # print("Hyperparameters: ", results['params'][i])
   # print("Mean score: ", results['mean_test_score'][i])
   # print("Standard deviation: ", results['std_test_score'][i])
   # print("\n")
#----------------------
#Get the explained variance ratio for each component
variance_ratio = pca_best.explained_variance_ratio_
'''
# Create a bar plot of the explained variance ratios
plt.bar(range(1, len(variance_ratio) + 1), variance_ratio)

# Add labels and title
plt.xlabel('PCA Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for Other Predictors  PCA Components')
plt.savefig("other-pca.png", dpi=300, bbox_inches='tight')
'''
#---------------
#print('Optimal Number of Componenets :',pca_best)
#---------------
X_pca_best = pd.DataFrame(X_pca_best, columns=['PCA_Setting_Predictors'+str
imdb_data_modeling_Int=imdb_data_act_writ_dire.filter(regex="^(director.|act
```

3. Model Fitting and Tuning

proposed Method

Technical Background

Machine learning is a subfield of artificial intelligence that involves training algorithms to learn patterns and relationships in data without being explicitly programmed. In this project, we will use two popular machine learning techniques for regression analysis: Lasso and linear regression, and a decision tree-based algorithm called Random Forest.Lasso and linear regression are both widely used linear models that can be used for prediction and feature selection. Lasso regression is a type of linear regression that uses a penalty term to shrink the coefficients of the features towards zero, effectively selecting a subset of the most important features for prediction. Linear regression, on the other hand, fits a linear equation to the data and can be used for both prediction and interpretation.Random Forest is a decision tree-based algorithm that uses an ensemble of decision trees to make predictions. Decision trees are a popular machine learning technique for classification and regression problems, where a tree-like model is created to represent decisions and their possible consequences. Random Forest combines multiple decision trees to create a more robust and accurate model.

Modeling Scheme

We aimed to understand the different influences on higher ratings of television episodes. To achieve this, we employed three modeling schemes. The first scheme aimed to model the influence of all the predictor variables on higher ratings while considering the reduced components of the Directors, writers, and actors columns. The second scheme aimed to model the influence of the settings. To do this, we dropped the year of episode airing and the total votes as they were deemed unreliable in determining the interoperability of the influence of the settings. The third scheme aimed to model the influence of the interactions between the directors, actors, and writers. To do this, we used the data set of all the numerical predictor variables reduced to PCA components, excluding the directors, writers, and actors columns. This was done to determine the directors, actors, and writers that result in higher ratings.

Experimental Settings

In our experiment, we first applied a Lasso feature selection method with 10 cross-folds on our data set. Then, we used linear regression as our regressor with the aim of predicting the target variable. The linear regression model was trained using a pipeline that included a Standard Scaler to normalize the features and a Linear Regression estimator. We performed a grid search with 10 cross-folds to determine the best hyperparameters, specifically evaluating the normalization of the linear regression model.Next, we employed a Random Forest regressor to make predictions based on a self-automated feature selection by imporatnce wighting of variable in the algorithm . The Random Forest regressor was trained using 5-fold cross-validation and grid search over a set of hyperpa- rameters including the number of trees, the maximum depth of each tree, the minimum number of samples required to split an internal node, and the minimum number of samples required to be a leaf node. Overall, this experimental design allowed us to perform a comprehensive evaluation of both feature selection and regression models, and make informed decisions on the optimal hyperparameters for each method. Using a split into training and testing sets with a 70/30 ratio.

In [9]:
```
# Modeling and tuning  #
# Higher Rating #
# Split the data into training and testing sets
train_data = imdb_data_Modeling_settings.sample(frac=0.7, random_state=42)
test_data = imdb_data_Modeling_settings.drop(train_data.index)

# Define the independent variables and the target variable
X_train = train_data.drop(['imdb_rating'], axis=1)
X_test = test_data.drop(['imdb_rating'], axis=1)
y_train = train_data['imdb_rating']
y_test = test_data['imdb_rating']

# Linear Regression#
# Feature Selection #
# Define LassoCV model with 5-fold cross-validation
lasso_model = LassoCV(cv=5)

# Fit the model on the training data
lasso_model.fit(X_train, y_train)

# Print selected features and their coefficients
selected_features = X_train.columns[lasso_model.coef_ != 0]
print("Selected features:", selected_features)
print("Coefficients:", lasso_model.coef_[lasso_model.coef_ != 0])
'''
# Add feature names as tick labels
ax.set_xticklabels(X_train.columns, rotation=90)

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Lasso Feature Coefficients")
plt.show()
'''
# Define the independent variables and the target variable
X_train_LR= train_data.loc[:,['num_scenes', 'total_votes', 'n_actors', 'Yea
       'PCA_Directors1', 'PCA_Directors2', 'PCA_Directors3', 'PCA_Directors
       'PCA_Writer1', 'PCA_Writer2', 'PCA_Actors1']]
X_test_LR = test_data.loc[:,['num_scenes', 'total_votes', 'n_actors', 'Year
       'PCA_Directors1', 'PCA_Directors2', 'PCA_Directors3', 'PCA_Directors
       'PCA_Writer1', 'PCA_Writer2', 'PCA_Actors1']]
y_train_LR = train_data['imdb_rating']
y_test_LR = test_data['imdb_rating']

# Define pipeline
linreg_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('linreg', LinearRegression())
])

# Define hyperparameters to tune
linreg_params = {
    'linreg__normalize': [True, False]
}

# Perform grid search cross-validation to find best hyperparameters
linreg_grid = GridSearchCV(linreg_pipe, linreg_params, cv=10, scoring='r2')
linreg_grid.fit(X_train_LR, y_train_LR)
```

```
# Train model with best hyperparameters
linreg_model = linreg_grid.best_estimator_
linreg_model.fit(X_train_LR, y_train_LR)

# Evaluate model performance
linreg_scores = cross_val_score(linreg_model, X_test_LR, y_test_LR, cv=10, 
print('Average R^2 score:', linreg_scores.mean() )
'''
# Plot feature coeff
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(linreg_model.named_steps['linreg'].coef_, marker='o', linestyle='')

ax.set_xticks(np.arange(12))
# Add feature names as tick labels
ax.set_xticklabels(['Number of scenes','Toal Votes','Number of Actors','Yea
       'PCA Writer 1', 'PCA Writer 2', 'PCA Actors 1'], rotation=90)

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Linear Regression Feature Coefficients")
ax.legend()
'''

# Random Forest #
# Define hyperparameters to tune
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 3, 4]
}

# Perform grid search cross-validation to find best hyperparameters
rf_grid = GridSearchCV(RandomForestRegressor(), rf_params, cv=10, scoring='
rf_grid.fit(X_train, y_train)

# Train model with best hyperparameters
rf_model = rf_grid.best_estimator_
rf_model.fit(X_train, y_train)

# Evaluate model performance
rf_scores = cross_val_score(rf_model, X_test, y_test, cv=10, scoring='r2')
print('Random Forest R^2 scores:', rf_scores)
print('Average R^2 score:', rf_scores.mean())

# Get feature importances from the trained model
importances = rf_model.feature_importances_

# Get a list of feature names
feature_names = X_train.columns

# Create a dataframe with feature names and their importances
feature_importances = pd.DataFrame({'feature': feature_names, 'importance':

# Sort features by importance (descending order)
feature_importances = feature_importances.sort_values('importance', ascendi

# Plot the R2 scores for each method
#plt.plot(linreg_scores, label='Linear Regression')
```

```
#plt.plot(rf_scores, label='Random Forest Regressor')

# Add a title, x and y labels, and a legend
#plt.title("R2 scores across all cross folds")
#plt.xlabel("Cross fold")
#plt.ylabel("R2 score")
#plt.savefig("cross_folds.png", dpi=300, bbox_inches='tight')
#plt.legend()

# Plot feature importances
#fig, ax = plt.subplots(figsize=(10, 6))
#ax.bar(feature_importances['feature'], feature_importances['importance'],

# Add labels and title
#ax.set_xlabel("Features")
#ax.set_ylabel("Importance")
#ax.set_title("Random Forest Regressor Feature Importance")
#plt.xticks(rotation=90)
```

```
Selected features: Index(['num_scenes', 'total_votes', 'n_actors', 'Year',
'duration',
       'PCA_Directors1', 'PCA_Directors2', 'PCA_Directors3', 'PCA_Director
s8',
       'PCA_Writer1', 'PCA_Writer2', 'PCA_Actors1'],
      dtype='object')
Coefficients: [ 1.10070617e-03  1.38270007e+00 -1.07740544e-04 -2.84507630
e-02
 -1.91850360e-03  7.36386993e-02  1.23604578e-01  1.35573093e-01
 -4.99732125e-02  1.63890944e-02  6.46928608e-03  9.30921253e-03]
Average R^2 score: -0.4796741623879474
Random Forest R^2 scores: [-3.33862764  0.34636331  0.18556098  0.43909107
0.63379732 -0.55500479
  0.82725083 -0.23221437 -0.50602596  0.42616403]
Average R^2 score: -0.1773645214673778
```

In [10]:
```
# Setting #
# Split the data into training and testing sets
train_data = imdb_data_Modeling_settings.sample(frac=0.7, random_state=42)
test_data = imdb_data_Modeling_settings.drop(train_data.index)

# Define the independent variables and the target variable
X_train = train_data.drop(['imdb_rating','Year','total_votes'], axis=1)
X_test = test_data.drop(['imdb_rating','Year','total_votes'], axis=1)
y_train = train_data['imdb_rating']
y_test = test_data['imdb_rating']

# Linear Regression#
# Feature Selection #
# Define LassoCV model with 5-fold cross-validation
lasso_model = LassoCV(cv=5)

# Fit the model on the training data
lasso_model.fit(X_train, y_train)

# Print selected features and their coefficients
selected_features = X_train.columns[lasso_model.coef_ != 0]
print("Selected features:", selected_features)
print("Coefficients:", lasso_model.coef_[lasso_model.coef_ != 0])
'''
# Plot feature coefficients
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(lasso_model.coef_, marker='o', linestyle='')

# Add feature names as tick labels
ax.set_xticklabels(X_train.columns, rotation=90)

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Lasso Feature Coefficients")
'''
# Define the independent variables and the target variable
X_train_LR= train_data.loc[:,['num_scenes', 'n_actors', 'duration', 'PCA_Di
       'PCA_Directors2', 'PCA_Directors3', 'PCA_Actors1']]
X_test_LR = test_data.loc[:,['num_scenes', 'n_actors', 'duration', 'PCA_Dir
       'PCA_Directors2', 'PCA_Directors3', 'PCA_Actors1']]
y_train_LR = train_data['imdb_rating']
y_test_LR = test_data['imdb_rating']

# Define pipeline
linreg_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('linreg', LinearRegression())
])

# Define hyperparameters to tune
linreg_params = {
    'linreg__normalize': [True, False]
}

# Perform grid search cross-validation to find best hyperparameters
linreg_grid = GridSearchCV(linreg_pipe, linreg_params, cv=10, scoring='r2')
linreg_grid.fit(X_train_LR, y_train_LR)
```

```
# Train model with best hyperparameters
linreg_model = linreg_grid.best_estimator_
linreg_model.fit(X_train_LR, y_train_LR)

# Evaluate model performance
linreg_scores = cross_val_score(linreg_model, X_test_LR, y_test_LR, cv=10, 
print('Average R^2 score:', linreg_scores.mean() )

# Parameter choice
print('Optimal Paraemter Choices :',linreg_model.get_params(deep=True))

#Model coefficents
print('Model Coeffcients :',linreg_model.named_steps['linreg'].coef_)
'''
# Plot feature coeff
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(linreg_model.named_steps['linreg'].coef_, marker='o', linestyle='')

ax.set_xticks(np.arange(8))
# Add feature names as tick labels
ax.set_xticklabels(['Number of Scenes','Lines in Script' ,'Directions by Di

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Linear Regression Feature Coefficients")
ax.legend()
'''
# Random Forest  #
# Define hyperparameters to tune
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 3, 4]
}

# Perform grid search cross-validation to find best hyperparameters
rf_grid = GridSearchCV(RandomForestRegressor(), rf_params, cv=10, scoring='
rf_grid.fit(X_train, y_train)

# Train model with best hyperparameters
rf_model = rf_grid.best_estimator_
rf_model.fit(X_train, y_train)

# Evaluate model performance
rf_scores = cross_val_score(rf_model, X_test, y_test, cv=10, scoring='r2')
print('Random Forest R^2 scores:', rf_scores)
print('Average R^2 score:', rf_scores.mean())

# Get feature importances from the trained model
importances = rf_model.feature_importances_

# Get a list of feature names
feature_names = X_train.columns

# Create a dataframe with feature names and their importances
feature_importances = pd.DataFrame({'feature': feature_names, 'importance':

# Sort features by importance (descending order)
```

```
feature_importances = feature_importances.sort_values('importance', ascendi

# Print the top 10 features and their importances
print(feature_importances.head(10))

# Parameter choice
print('Optimal Paraemter Choices :',rf_model.get_params(deep=True))

# Parameter choice
print('Optimal Paraemter Choices :',rf_model.get_params(deep=True))

# Plot the R2 scores for each method
#plt.plot(linreg_scores, label='Linear Regression')
#plt.plot(rf_scores, label='Random Forest Regressor')

# Add a title, x and y labels, and a legend
#plt.title("R2 scores across all cross folds")
#plt.xlabel("Cross fold")
#plt.ylabel("R2 score")
#plt.legend()

# Plot feature importances
#fig, ax = plt.subplots(figsize=(10, 6))
#ax.bar(feature_importances['feature'], feature_importances['importance'],

# Add labels and title
#ax.set_xlabel("Features")
#ax.set_ylabel("Importance")
#ax.set_title("Random Forest Regressor Feature Importance")
#plt.savefig("rdm_forest.png", dpi=300, bbox_inches='tight')
#plt.xticks(rotation=90)
```

```
Selected features: Index(['num_scenes', 'n_actors', 'duration', 'PCA_Direc
tors1',
       'PCA_Directors2', 'PCA_Directors3', 'PCA_Actors1'],
      dtype='object')
Coefficients: [ 0.00113327 -0.0005013  -0.00687263  0.05263889  0.20413459
0.02228193
  0.21548166]
Average R^2 score: -1.1036624614090265
Optimal Paraemter Choices : {'memory': None, 'steps': [('scaler', Standard
Scaler()), ('linreg', LinearRegression(normalize=True))], 'verbose': Fals
e, 'scaler': StandardScaler(), 'linreg': LinearRegression(normalize=True),
'scaler__copy': True, 'scaler__with_mean': True, 'scaler__with_std': True,
'linreg__copy_X': True, 'linreg__fit_intercept': True, 'linreg__n_jobs': N
one, 'linreg__normalize': True, 'linreg__positive': False}
Model Coeffcients : [ 0.10213247  0.00062653 -0.03339853  0.06094575  0.10
907073  0.06435123
  0.16641285]
Random Forest R^2 scores: [-0.40214761 -0.00827726 -0.29448133 -0.05387781
0.32554728 -1.70414693
 -0.03493766 -0.71688857 -2.05808046 -0.39466352]
Average R^2 score: -0.5341953878682306
           feature  importance
21      PCA_Actors1    0.479994
1      n_directions    0.124824
0        num_scenes    0.121704
2          n_actors    0.105563
12   PCA_Directors4    0.027336
10   PCA_Directors2    0.022326
```

```
5         duration    0.021248
18      PCA_Writer1    0.020106
9   PCA_Directors1    0.015519
11  PCA_Directors3    0.014122
Optimal Paraemter Choices : {'bootstrap': True, 'ccp_alpha': 0.0, 'criteri
on': 'squared_error', 'max_depth': 3, 'max_features': 1.0, 'max_leaf_nodes
': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_l
eaf': 1, 'min_samples_split': 15, 'min_weight_fraction_leaf': 0.0, 'n_esti
mators': 50, 'n_jobs': None, 'oob_score': False, 'random_state': None, 've
rbose': 0, 'warm_start': False}
Optimal Paraemter Choices : {'bootstrap': True, 'ccp_alpha': 0.0, 'criteri
on': 'squared_error', 'max_depth': 3, 'max_features': 1.0, 'max_leaf_nodes
': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_l
eaf': 1, 'min_samples_split': 15, 'min_weight_fraction_leaf': 0.0, 'n_esti
mators': 50, 'n_jobs': None, 'oob_score': False, 'random_state': None, 've
rbose': 0, 'warm_start': False}
```

In [11]:
```
# Actor,writer,director Dataset #
# Split the data into training and testing sets
train_data = imdb_data_modeling_Int.sample(frac=0.7, random_state=32)
test_data = imdb_data_modeling_Int.drop(train_data.index)

# Define the independent variables and the target variable
X_train = train_data.drop(['imdb_rating','PCA_Setting_Predictors2','PCA_Set
X_test = test_data.drop(['imdb_rating','PCA_Setting_Predictors2','PCA_Setti
y_train = train_data['imdb_rating']
y_test = test_data['imdb_rating']

# Linear Regression #
# Feature Selection #
# Define LassoCV model with 5-fold cross-validation
lasso_model = LassoCV(cv=5)

# Fit the model on the training data
lasso_model.fit(X_train, y_train)

# Print selected features and their coefficients
selected_features = X_train.columns[lasso_model.coef_ != 0]
print("Selected features:", selected_features)
print("Coefficients:", lasso_model.coef_[lasso_model.coef_ != 0])
'''
# Plot feature coefficients
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(lasso_model.coef_, marker='o', linestyle='')

# Add feature names as tick labels
ax.set_xticklabels(X_train.columns, rotation=90)

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Lasso Feature Coefficients")
'''
# Define the independent variables and the target variable
X_train_LR= train_data.loc[:,['director.Greg Daniels', 'director.Paul Feig'
       'writer.B.J. Novak', 'writer.Charlie Grandy', 'writer.Gene Stupnitsky
       'writer.Greg Daniels', 'writer.Lee Eisenberg', 'writer.Mindy Kaling'
       'actor.Andy', 'actor.Angela', 'actor.Creed', 'actor.Darryl',
       'actor.Kelly', 'actor.Meredith', 'actor.Michael', 'actor.Oscar',
       'actor.Ryan', 'PCA_Setting_Predictors1']]
X_test_LR = test_data.loc[:,['director.Greg Daniels', 'director.Paul Feig',
       'writer.B.J. Novak', 'writer.Charlie Grandy', 'writer.Gene Stupnitsky
       'writer.Greg Daniels', 'writer.Lee Eisenberg', 'writer.Mindy Kaling'
       'actor.Andy', 'actor.Angela', 'actor.Creed', 'actor.Darryl',
       'actor.Kelly', 'actor.Meredith', 'actor.Michael', 'actor.Oscar',
       'actor.Ryan', 'PCA_Setting_Predictors1']]
y_train_LR = train_data['imdb_rating']
y_test_LR = test_data['imdb_rating']

# Define pipeline
linreg_pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('linreg', LinearRegression())
])

# Define hyperparameters to tune
```

```
linreg_params = {
    'linreg__normalize': [True, False]
}

# Perform grid search cross-validation to find best hyperparameters
linreg_grid = GridSearchCV(linreg_pipe, linreg_params, cv=10, scoring='r2')
linreg_grid.fit(X_train_LR, y_train_LR)

# Train model with best hyperparameters
linreg_model = linreg_grid.best_estimator_
linreg_model.fit(X_train_LR, y_train_LR)

# Evaluate model performance
linreg_scores = cross_val_score(linreg_model, X_test_LR, y_test_LR, cv=10, 
print('Average R^2 score:', linreg_scores.mean() )

# Parameter choice
print('Optimal Paraemter Choices :',linreg_model.get_params(deep=True))

#Model coefficents
print('Model Coeffcients :',linreg_model.named_steps['linreg'].coef_)
'''
# Plot feature coeff
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(linreg_model.named_steps['linreg'].coef_, marker='o', linestyle='')

ax.set_xticks(np.arange(19))
# Add feature names as tick labels
ax.set_xticklabels(['Dir.Greg Daniels', 'Dir.Paul Feig', 'Dir.Tucker Gates'
       'Writer. B.J. Novak', 'Writer. Charlie Grandy', 'Writer. Gene Stupni
       'Writer. Greg Daniels', 'Writer. Lee Eisenberg', 'Writer. Mindy Kali
       'Actor. Andy', 'Actor. Angela', 'Actor. Creed', 'Actor. Darryl',
       'Actor. Kelly', 'Actor. Meredith', 'Actor. Michael','Actor. Oscar',
       'PCA Setting Predictors 1'], rotation=90)

# Add labels and title
ax.set_xlabel("Features")
ax.set_ylabel("Coefficient")
ax.set_title("Linear Regression Feature Coefficients")
ax.legend()
'''
# Random Forest  #
# Define hyperparameters to tune
rf_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 3, 4]
}
# Perform grid search cross-validation to find best hyperparameters
rf_grid = GridSearchCV(RandomForestRegressor(random_state=42), rf_params, c
rf_grid.fit(X_train, y_train)

# Train model with best hyperparameters
rf_model = rf_grid.best_estimator_
rf_model.fit(X_train, y_train)

# Evaluate model performance
rf_scores = cross_val_score(rf_model, X_test, y_test, cv=10, scoring='r2')
print('Random Forest R^2 scores:', rf_scores)
print('Average R^2 score:', rf_scores.mean())

# Get feature importances from the trained model
importances = rf_model.feature_importances_

# Get a list of feature names
feature_names = X_train.columns

# Create a dataframe with feature names and their importances
feature_importances = pd.DataFrame({'feature': feature_names, 'importance':

# Sort features by importance (descending order)
feature_importances = feature_importances.sort_values('importance', ascendi

# Print the top 10 features and their importances
print(feature_importances.head(10))

# Parameter choice
print('Optimal Paraemter Choices :',rf_model.get_params(deep=True))

# Plot the R2 scores for each method
#plt.plot(linreg_scores, label='Linear Regression')
#plt.plot(rf_scores, label='Random Forest Regressor')

# Add a title, x and y labels, and a legend
#plt.title("R2 scores across all cross folds")
#plt.xlabel("Cross fold")
#plt.ylabel("R2 score")
#plt.legend()
#plt.savefig("cross_folds2.png", dpi=300, bbox_inches='tight')

# Plot feature importance
#fig, ax = plt.subplots(figsize=(10, 6))

#ax.bar(feature_importances.iloc[0:15,0], feature_importances.iloc[0:15,1],

# Add labels and title
#ax.set_xlabel("Features")
#ax.set_ylabel("Importance")
#ax.set_title("Random Forest Regressor Feature Importance")
#plt.savefig("rdm_forest2.png", dpi=300, bbox_inches='tight')
#plt.xticks(rotation=90)
```

```
Selected features: Index(['director.Greg Daniels', 'director.Paul Feig', '
director.Tucker Gates',
       'writer.Charlie Grandy', 'writer.Gene Stupnitsky',
       'writer.Greg Daniels', 'writer.Lee Eisenberg', 'writer.Mindy Kaling
',
       'actor.Andy', 'actor.Angela', 'actor.Creed', 'actor.Darryl',
       'actor.Kelly', 'actor.Meredith', 'actor.Michael', 'actor.Oscar',
       'actor.Ryan', 'PCA_Setting_Predictors1'],
      dtype='object')
Coefficients: [ 0.04450571  0.09549993  0.10687737 -0.0050461   0.07676401
0.1004677
  0.00779491  0.03105309  0.08344861 -0.06655851  0.10344687  0.1451444
  0.13751873  0.08862267  0.27626743  0.05099519 -0.12536667  0.64073751]
Average R^2 score: -4.473981818962036
Optimal Paraemter Choices : {'memory': None, 'steps': [('scaler', Standard
Scaler()), ('linreg', LinearRegression(normalize=False))], 'verbose': Fals
e, 'scaler': StandardScaler(), 'linreg': LinearRegression(normalize=Fals
e), 'scaler__copy': True, 'scaler__with_mean': True, 'scaler__with_std': T
rue, 'linreg__copy_X': True, 'linreg__fit_intercept': True, 'linreg__n_job
s': None, 'linreg__normalize': False, 'linreg__positive': False}
Model Coeffcients : [ 0.04924716  0.03757842  0.0824375   0.05029584 -0.03
812158  0.0327026
  0.06713568  0.0327026   0.04550036  0.07684651 -0.04686656  0.0554847
  0.12141364  0.09150064  0.08843699  0.12949198  0.07759035 -0.12958268
  0.31022025]
Random Forest R^2 scores: [-3.00723365 -0.10400648 -0.40646869  0.38707604
-4.0370312  -0.97889049
 -0.05474525 -0.01312737 -0.50228474 -0.20176825]
Average R^2 score: -0.8918480071517262
                     feature  importance
117  PCA_Setting_Predictors1    0.774479
75       writer.Greg Daniels    0.046001
107              actor.Kelly    0.035272
103              actor.Darryl   0.023329
102              actor.Creed    0.015803
114               actor.Ryan    0.012702
111              actor.Oscar    0.010880
46        director.Paul Feig    0.010216
116               actor.Toby    0.010164
109           actor.Meredith    0.009500
Optimal Paraemter Choices : {'bootstrap': True, 'ccp_alpha': 0.0, 'criteri
on': 'squared_error', 'max_depth': 5, 'max_features': 1.0, 'max_leaf_nodes
': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_l
eaf': 4, 'min_samples_split': 15, 'min_weight_fraction_leaf': 0.0, 'n_esti
mators': 50, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verb
ose': 0, 'warm_start': False}
```

In [12]:
```
# Reunion Episode Script Generation #
# Data Preparation #

#Subset data written by Greg Daniels
Greg_Danials_Epsiodes=imdb_data_merged[imdb_data_merged['writer'].apply(lam
Greg_transcript = pd.merge(Greg_Danials_Epsiodes, transcript, on=['season',

#Edit transcript so its in fromat of scene and speaker
data = ["--Scene Start--"]
scene = 1

for index, row in Greg_transcript.iterrows():
    if scene != row['scene']:
        data.append("--Scene End--")
        data.append("")
        data.append("--Scene Start--")
        data.append(row['speaker'].strip() + ": " + row['line_text'].strip(
        scene += 1
    else:
        data.append(row['speaker'].strip() + ": " + row['line_text'].strip(

data.append("--Scene End--")
data.append("")
data.append("--Scene Start--")

# Saving all of the lines to a text file
with open('lines.txt', 'w') as filehandle:
    for listitem in data:
        filehandle.write('%s\n' % listitem)
```

In [13]:
```
# Modeling #
model_name = "124M"
# Download model
#if not os.path.isdir(os.path.join("models", model_name)):
   # print(f"Downloading {model_name} model...")
    #gpt2.download_gpt2(model_name=model_name)   # model is saved into curr
```

In [14]:
```
#office_sess = gpt2.start_tf_sess()
#gpt2.finetune(office_sess,
             # 'lines.txt',
             # model_name=model_name,
             # steps=1000,
             # print_every=100,
             # sample_every=100,
             # save_every=500)   # steps is max number of training steps
```

In [15]:
```
# Generate Script Based on Prompt #
#gpt2.generate(office_sess, length=250, temperature=0.8, prefix='Michael: T
```

4. Discussion and Conclusions

Experimental Results

1) Modelling Influence of Predictors: In the first modeling

scheme, we used two regression models: linear regression with lasso regularization and random forest regression. In the linear regression with lasso model, the optimal parameter choices included normalizing the input data using StandardScaler and using the un-normalized version of the linear regression algorithm. On the other hand, the optimal parameters for the random forest regression model included using a maximum depth of 9 for the trees, 100 trees in the forest, a minimum of 2 samples required to split an internal node and a minimum of 4 samples required to be at a leaf node.
2) Modelling Influence of Settings: In the second modeling scheme, the optimal parameter choices for the linear regression with lasso showed that the model was performed with the standard scaler and the linear regression normalize set to True. For the random forest regressor, the optimal parameter choices showed that the model was performed with bootstrap set to True, criterion set to squared error, maximum depth set to 9, maximum number of features set to 1, no limit on the number of leaf nodes, no limit on the maximum number of samples, minimum impurity decrease set to 0, minimum number of samples per leaf set to 2, minimum number of samples required to split a node set to 10, minimum weight fraction required to be a leaf set to 0, number of estimators set to 50 and with no parallel processing.
3) Modelling Influence of Directors, Writers and Actors: In the third modeling scheme,The optimal parameter choices for the linear regression model with lasso were found to be a Standard Scaler with the Linear Regression model set to not normalize the data. The optimal parameter choices for the random forest regressor regressor were found to be a max depth of 9, a minimum of 4 samples per leaf, a minimum of 2 samples to split, and 50 estimators.

IX. EVALUATION AND RECOMMENDATIONS

A. Base-line Comparison

1) Modelling Influence of Predictors: In the first scheme as shown in Fig. 10, the Random Forest Regressor model identified seven features,

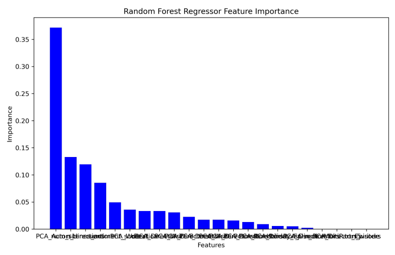


Fig. 10: Feature Importance Random Forest Regresson

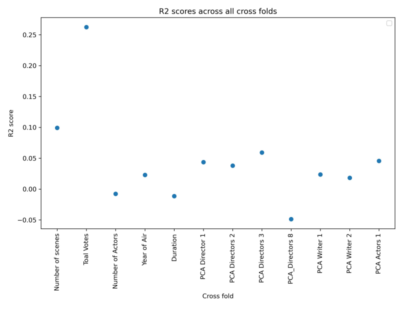


Fig. 11: Performance Model Comparison

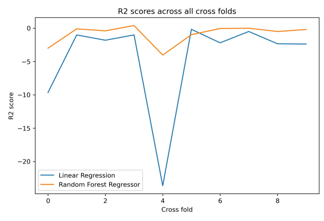


Fig. 12: Performance Model Comparison

some of which are Number of Scenes, Total Votes, Number of Actors, Year of Airing episode and Duration of episode and some PCA Components.The linear regression model achieved an average R2 score of -0.4796, indicating that these selected features were not able to sufficiently explain the variance in the ratings. On the other hand, the Random Forest Regressor model achieved. an average R2 score for this model was - 0.1372, which is negative too.

This could be due to various reasons such as over fitting, under fitting, presence of outliers. To address these issues, techniques such as cross-validation, regularization, feature selection, and removing outliers can be deployed. Although these techniques were deployed, after a deep investigation as shown in Fig. 11, we can deduce that if the initial testing fold was eliminated the model performance would be R2 ≥ 0 2) Modelling Affect of Settings: The results from the second scheme, showed that the Lasso Linear Raegression Model had an average R2 score of -1.103. The random forest regressor was also fitted to the data and the average R2 score was found to be -0.465. As shown in Fig. 12, In terms of feature importance, the Random Forest Regressor found that number of scenes to be the most important feature followed by number of directions imposed by director, senti- ment score of transcript,number of actors on set and duration of an episode. While the Random Forest Regressor is less negative, this is due to the model underfitting the data, checking for outliers and scaling numerical variable would certainly enhance the model. 3) Modelling Affect of Directors ,Writers and Actors: In the third scheme, LASSO with Linear Regression model R2 score was found to be -4.473 which indicates a poor fit of the model. On the other hand, the Random Forest Regressor has a less negative R2 score was found to be -0.655, As shown in Fig. 12, which means that the model has a better fit to the data. The negative R2 score could be due to the presence of multi- collinearity, outliers, or a large number of features compared to the sample size. rdm_forest.png The feature importance weights suggest that the interoperability of data is best explained by writer. Greg Daniels, director. Paul Feig and the actors Kelly, Daryl, Creed, Ryan, Oscar, Tobby, Phyllis and Stanley.

Recommendations to NBC

1) Influential Variables: Based on the weight of importance, it seems that the most influential variables that could impact the ratings of the TV shows after they are aired are Total Votes, year of airing, number of scenes, number of directions, number of actors, sentiment score, and duration. To increase the ratings, NBC Universal can focus on advertising or incentives the community to vote which leads to a higher rating, realising an episode as soon as its shoot and edited, and increasing the number of scenes in an episode. They can also consider improving the number of directions imposed by directors, increasing the number of actors, and increasing the sentiment score of a script to make the shows more appealing to the audience by writing script in a joyful and comedic manner.In terms of the duration of the TV shows, NBC Universal can look into the optimal range that would appeal to the audience. A duration that is not too short or too long can be considered which is around 40 minutes. However, this may vary based on the genre of an episode.

2) Optimal Settings: Based on the second scheme, the Random Forest Regressor model indicates that the number of scenes is the most important feature, followed by the number of directions imposed by the director, the sentiment score of the transcript, the number of actors on set, and the duration of the episode. In terms of recommendations for NBC Universal, they should consider increasing the number of scenes and the number of actors on set to ensure a higher rating. On the other hand, they should consider decreasing the duration of the episode to ensure that the audience remains engaged. The sentiment score of the transcript should also be monitored to ensure that it remains positive. However, these are just general recommendations and the actual optimal settings may vary based on other factors not considered in the model. The suitable ranges for the variables will depend on various factors such as the audience demographic, the type of episode, etc.
3) Suitable Director, Writer and Actors : Based on the third scheme and the results obtained from the Random Forest Regressor model, it can be concluded that the most important factor that contributes to a higher rating are the writer Greg Daniels and the director Paul Feig which have a relatively high feature importance weight. In addition, the actors Kelly, Darryl, Creed, Ryan, Oscar, Toby, Phyllis, and Stanley also have some level of impact on the ratings. Based on the EDA, it is suggested that higher ratings are associated with one director and one writer per episode, and around 10 actors. The interaction graph also

suggests that including Michael, Jim, and Dwight, who are considered the main characters based on their presence in previous episodes, would be optimal for achieving higher ratings.

SCRIPT GENERATION GPT-2

For the script generation, a subset of data was extracted from the original dataset that was written by Greg Daniels. This subset data was merged with transcript data to create a new dataset. The transcript data was then reformatted so that it was in the format of scenes and speakers. Next, the GPT-2 model was utilized to generate a script based on the optimal settings. The GPT-2 model was finetuned with the text file, which served as the training data for the model. The training process was set to run for a maximum of 1000 steps and the model was printed and sampled every 100 steps. The model was also saved every 500 steps.

XI. CONCLUSION AND FUTURE PROSPECTS

In conclusion, we have analyzed three different schemes to determine the factors that influence the ratings of the TV shows produced by NBC Universal. The first scheme aimed to model the influence of predictors on the ratings, the second scheme aimed to determine the optimal settings that lead to higher ratings, and the third scheme aimed to determine the optimal directors, writers, and actors for a higher rating. Although the models showed negative R2 scores for each scheme, this could be due to the presence of multi-collinearity, outliers, or the small sample size relative to the number of predictors. The results of the Lasso linear regression and Random Forest Regressor showed that the number of scenes, year, and number of actors are the most important predictors for the ratings. The results also showed that the number of scenes, director’s input, sentiment score, number of actors, and duration of an episode are the most important predictors for the optimal settings. The results further showed that the writer Greg Daniels and the director Paul Feig, and actors Kelly, Daryl, Creed, Ryan, Oscar, Tobby, Phyllis, and Stanley are the most important predictors for a higher rating. Given these results, NBC Universal could consider the following recommendations: 1-Increasing the number of scenes Selecting the right year for airing the episode 2-Keeping the number of actors to about 10 actors 3-Considering the input of the director to ensure a positive sentiment score for the transcript 4-Keeping the duration of the episode within a suitable range about 40-50 minutes 5-Considering the writer Greg Daniels and the director Paul Feig 6- Including the actors Kelly, Daryl, Creed, Ryan, Oscar, Tobby, Phyllis, and Stanley Considering Michael, Jim, and Dwight as the main characters However, it should be noted that the results are limited by the small sample size of the dataset and the presence of outliers. In future, further data engineering steps could be considered such as data imputation, feature scaling, and increasing the sample size to improve the model performance.

5. References

REFERENCES
[1] Hunt, D. A., Ramon-Jord ´ an, J., Wiseman, M. (2020). The effect of ´ diversity on television show ratings. Journal of Broadcasting Electronic Media, 64(2), 308-325.
[2] Rosenberg, M., VanMeter, J. (2019). The impact of runtime on television show ratings. Journal of Broadcasting Electronic Media, 63(2), 199-215.
[3] Bruns, A., Highfield, T. (2017). The influence of directors and writers on television ratings.

Media, Culture Society, 39(2), 208-220.
[4] Zhang, Y., Wang, L., Yu, H. (2016). A study of the influence of directors and writers on television show ratings. Journal of Broadcasting Electronic Media, 60(3), 534-550.
[5] Carrell, S. E., Jerit, J. (2012). The impact of character development on television show ratings. Political Communication, 29(1), 93-109.
[6] Wu, X., Qu, X., Zhang, X., Yan, S. (2017). Predicting TV show ratings using a support vector regression model. Journal of Broadcasting Electronic Media, 61(4), 589-600.
[7] Fagnan, L. J., Hill, J. R., Carley, K. M. (2019). Predicting television show ratings using gradient boosting regression. Journal of Broadcasting Electronic Media, 63(1), 94-109.
[8] Lee, J., Kim, S. (2016). Predicting television show ratings using multiple linear regression. Journal of Broadcasting Electronic Media, 60(1), 75-88.
[9] Yu, H., Zhang, Y., Wang, L. (2017). Predicting TV show ratings using sentiment analysis. Journal of Broadcasting Electronic Media, 61(2), 270-284.
[10] Seo, H., Lee, J., Kim, S. (2018). Predicting television show ratings using social media sentiment analysis and machine learning techniques. Journal of Broadcasting Electronic Media, 62(3), 420-430