Machine Learning in Python - Project 1

Due Friday, March 11th by 5 pm.

Yin Lau (s1735270)

Ivan Lim (s1657846)

Aditya Mardjikoen (s2264710)

0. Setup

Put all libraries up here

In [1]:
```
# Add any additional libraries or submodules below
import os
# to delete after final review.
from google.colab import drive

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80

# sklearn, yellowbrick modules
import sklearn
from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold, train_test_split
from sklearn.linear_model import LinearRegression, LassoCV, Lasso
from sklearn.metrics import mean_squared_error
from yellowbrick.regressor import AlphaSelection

# Miscellaneous.
import datetime
```

In [ ]:
```
# Load data

# We first read the CSV data file into a Pandas dataframe.
d = pd.read_csv("the_office.csv")
```

1. Introduction

This section should include a brief introduction to the task and the data (assume this is a report you are delivering to a client). If you use any additional data sources, you should introduce them here and discuss why they were included.

Briefly outline the approaches being used and the conclusions that you are able to draw.

In this report, we are interested in understanding what factors made some episodes in The Office more popular than other ones. Moreover, we wish to find out which variables or factors have a non-negligible effect on the audience rating (based on IMDb rating) for The Office.

We conducted a thorough analysis of the dataset, removing 6 irrelevant or uncontrollable variables, and 89 variables via correlation analysis, from a total of 126 numerical and categorical variables.

We then selected 5 potential models, which were tested and cross-validated before the best model was chosen by ranking them with several metrics.

We then built a predictive model that attempted to capture the underlying relationships between these features and the audience ratings, and then used the insights gained from this model to provide recommendations to NBC Universal in order to produce the best reunion episode possible.

The data that we will used in this report is stored in a file called `the_office.csv` . This dataset consists of 12 columns and 188 rows scraped from IMDb. The description of each column in the dataset are as follows:

- `season` - Season number of the episode
- `episode` - Episode number within a season

- `episode_name` - Episode name
- `director` - Episode director(s), names are separated by `;`
- `writer` - Episode writer(s), names are separated by `;`
- `imdb_rating` - Episode rating on IMDB
- `total_votes` - Number of ratings for episode on IMDB
- `air_date` - Original air date of episode
- `n_lines` - Number of spoken lines in episode
- `n_directions` - Number of lines containing a stage direction
- `n_words` - Number of dialog words in episide
- `n_speak_char` - Number of different characters with spoken lines in episode
- `main_chars` - Main characters appearing in episode (main characters were determined to be characters appearing in more than 1/2 of the episodes)

The dataset itself is sourced from https://www.kaggle.com/nehaprabhavalkar/the-office-dataset.

2. Exploratory Data Analysis and Feature Engineering

Include a detailed discussion of the data with a particular emphasis on the features of the data that are relevant for the subsequent modeling. Including visualizations of the data is strongly encouraged - all code and plots must also be described in the write up. Think carefully about whether each plot needs to be included in your final draft - your report should include figures but they should be as focused and impactful as possible.

Additionally, this section should also implement and describe any preprocessing / feature engineering of the data. Specifically, this should be any code that you use to generate new columns in the data frame `d`. All of this processing is explicitly meant to occur before we split the data in to training and testing subsets. Processing that will be performed as part of an sklearn pipeline can be mentioned here but should be implemented in the following section.

All code and figures should be accompanied by text that provides an overview / context to what is being done or presented.

Exploratory Data Analysis

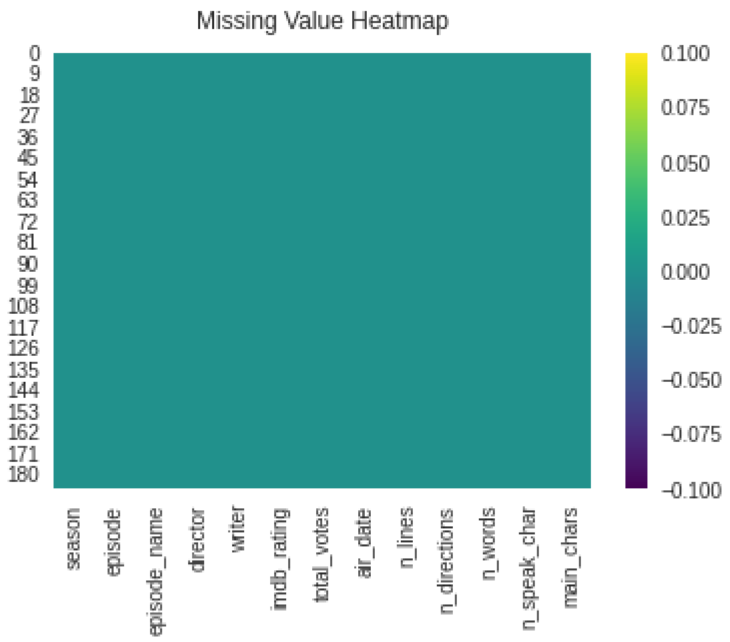
First, we examine the nature of the provided data, and check for missing data.

In [3]:
```
d.head()
```

Out[3]:

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_directions	n_words	n_speak_char	
0	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	7.6	3706	2005-03-24	229	27	2757	15	Angela;D
1	1	2	Diversity Day	Ken Kwapis	B.J. Novak	8.3	3566	2005-03-29	203	20	2808	12	Angela;
2	1	3	Health Care	Ken Whittingham	Paul Lieberstein	7.9	2983	2005-04-05	244	21	2769	13	Angela;D
3	1	4	The Alliance	Bryan Gordon	Michael Schur	8.1	2886	2005-04-12	243	24	2939	14	Angela;D
4	1	5	Basketball	Greg Daniels	Greg Daniels	8.4	3179	2005-04-19	230	49	2437	18	Angela;

In [4]:
```
# Check for missing data
missing_map = sns.heatmap(d.isnull(),cmap='viridis')
missing_map.set_title('Missing Value Heatmap', fontdict={'fontsize':12}, pad=12);
```



In this project, one of our main goals is to advise what NBC Universal should do to produce the highest rated reunion episode possible. Therefore, we must focus our model on only the relevant variables that can be controlled by our client.

`season` , `episode` - By definition, a reunion episode would not have either of these attributes. Therefore, we ignore them in our analysis.

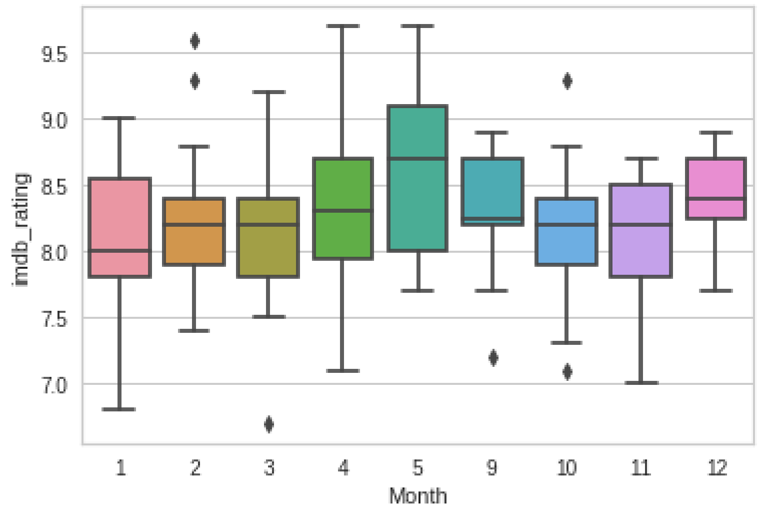
`episode_name` - This should be selected based on the content of the reunion episode; therefore, we do not consider this.

`total_votes` - While there are methods to influence the number of total votes cast (e.g. publicity campaigns), this is not directly controllable by our client; therefore, we ignore this.

`air_date` - The 'year' component of this variable is useless for our purposes; we analyze the 'month' and 'day' components below.

In [5]:
```
# Identify rating trend by month
month = pd.DatetimeIndex(d['air_date']).month
sns.boxplot(x=month, y='imdb_rating',data=d)
plt.xlabel('Month')
```

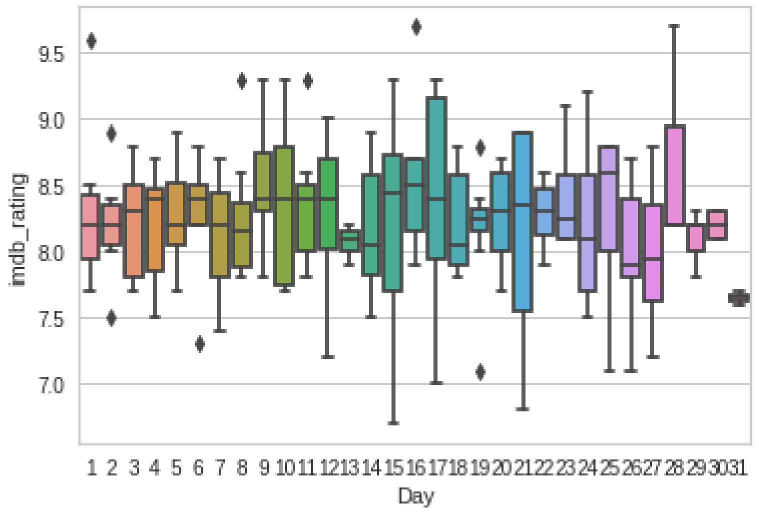
Out[5]:
```
Text(0.5, 0, 'Month')
```



Since the show is a series, it will be only shown during certain months. More specifically in this dataset, episodes are released from September up until May. Since we are aiding in the production of a reunion episode, the plot shown above will have no significant value to the IMDb rating as we will only be releasing a single episode. To further elaborate, this series usually follows the trend of starting off with a high rating, but then as the months progress, we see a consistent variation in the rating. Therefore, a single episode would not be affected by the month it is released in compared to if a new series were to be released.

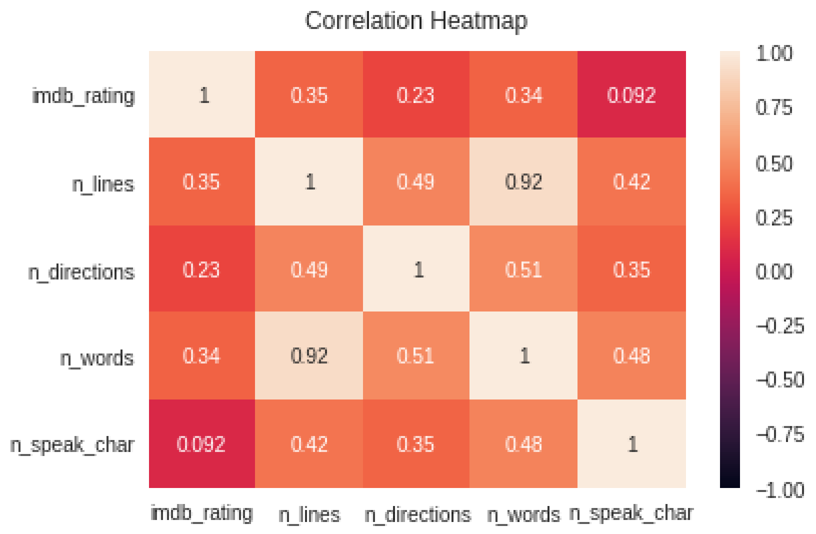
In [6]:
```
day = pd.DatetimeIndex(d['air_date']).day
sns.boxplot(x=day, y='imdb_rating',data=d)
plt.xlabel('Day')
```

Out[6]:
```
Text(0.5, 0, 'Day')
```



Looking at the average IMDb rating for each day, there is not a significant difference based on the day chosen - a line of best fit would be near horizontal. Therefore, we can also disregard which day of the month the episode should be released.

In [7]:
```
# We use a copy of the original data
data_copy = d.copy()
data_copy = data_copy.drop(columns=['season','episode','total_votes','air_date',
                                    'episode_name'])

# Correlation heatmap
heatmap = sns.heatmap(data_copy.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);
```



We observe in the correlation heatmap that the pairwise correlation coefficient for `n_words, n_lines` and `n_words, n_directions` are relatively high (0.92 and 0.51 respectively).

Given that the greater the number of spoken lines, the greater the number of dialog words in an episode's script, it is reasonable to remove `n_words` in order to reduce multicollinearity.

In [8]:
```
data_copy = data_copy.drop(columns=['n_words'])
```

Categorical variables.

In this section, we explore the relevant categorical variables: `director`, `writer`, and `main_chars`.

We one-hot encode each variable, as ordinal encoding would induce an unnecessary ordering on the levels of each of these variables.

We first explore **`main_chars`**.

In [9]:
```
dd = d.copy()

main_chars_df = dd['main_chars'].str.get_dummies(sep=';')
print("Frequency list of main characters.")
print(main_chars_df.sum(axis=0).sort_values(ascending=False), "\n")

# This is the dataframe for main characters that we will work with.
```

```
mcdf = main_chars_df.drop(columns=['Dwight','Jim','Pam','Kevin'])
# 13 characters left.

# Correlation analysis.
mcdf_corr = abs(mcdf.corr())

upper = np.triu(mcdf_corr) #  We only need upper triangular of correlation matrix.
n = 0.5  #  If correlation exceeds n, assume highly correlated.

print("Pairs of characters with a high correlation coefficient.")
for i in range(12): #  Iterate through each element.
  for j in range(12):
    if upper[i,j] > n and i != j: #  If exceeds n AND not a diagonal element,

      # Print the corresponding variable names.
      print(mcdf_corr.columns[i], " ", mcdf_corr.columns[j], " ", upper[i,j])
```

```
Frequency list of main characters.
Dwight      186
Jim         185
Pam         182
Kevin       180
Angela      171
Stanley     165
Phyllis     165
Oscar       164
Andy        143
Kelly       142
Ryan        139
Michael     137
Meredith    131
Creed       131
Toby        108
Darryl      105
Erin         98
dtype: int64

Pairs of characters with a high correlation coefficient.
Erin   Michael   0.5667167574826768
Kelly   Ryan   0.5497024438014416
```

main_chars

We note from the frequency list that the characters Dwight, Jim, Pam and Kevin each appear in ≥ 180 episodes - we therefore assume that they will be included in the reunion episode, and therefore exclude them.

Moreover, we note that there are two pairs of characters that have a relatively high correlation coefficient, in that they seem to be more likely to appear with each other in an episode than with other characters. This is at least partially due to the in-show relationships between each pair. However, this coefficient is not high enough to treat each pair as a singular 'character'; therefore, we do nothing.

Next, we examine `director`.

In [10]:

```
director_df = dd['director'].str.get_dummies(sep=';')
# display top 20 directors in terms of episodes directed.
print(director_df.sum(axis=0).sort_values(ascending=False).head(20))
# problem is that only 11 out of 60 directors with >= 5 credits to their name.

ddf = director_df[director_df.columns[director_df.sum()>=5]]
# 11 directors.
```

```
Randall Einhorn      15
Paul Feig            14
Greg Daniels         14
Ken Kwapis           12
Jeffrey Blitz        11
David Rogers          9
Matt Sohn             8
Ken Whittingham       8
Charles McDougall     6
Paul Lieberstein      6
B.J. Novak            5
Harold Ramis          4
Tucker Gates          4
Brent Forrester       4
Jennifer Celotta      3
Troy Miller           3
Steve Carell          3
Rainn Wilson          3
John Krasinski        3
Lee Eisenberg         2
dtype: int64
```

`director`

We note that only 11 out of 60 directors have directed ≥ 5 episodes. We therefore focus on these directors only, as we have insufficient data on the other directors to accurately determine their effect on the IMDB ratings.

In [11]:
```
# Correlation analysis.
ddf_corr = abs(ddf.corr())

upper = np.triu(ddf_corr) #  We only need upper triangular of correlation matrix.
n = 0.5  #  If correlation exceeds n, assume highly correlated.

for i in range(11): #  Iterate through each element.
  for j in range(11):
    if upper[i,j] > n and i != j: #  If exceeds n AND not a diagonal element,
      print(ddf_corr.columns[i], " ", ddf_corr.columns[j], " ", upper[i,j])
      # print the corresponding variable names.

# No highly correlated features for directors.
```

In our correlation analysis for `director` , we observe that there are no highly correlated features. We therefore cannot remove any features on this basis.

In [12]:
```
imdb_ddf = ddf.copy()
imdb_ddf.insert(0, 'imdb_rating', dd['imdb_rating'])

print('Directors and IMDB rating')
print(imdb_ddf.corr()['imdb_rating'][1:].sort_values(ascending=False))
# Randall Einhorn, Paul Lieberstein, B.J. Novak, David Rogers, Matt Sohn are negative.

delete_director = ['Randall Einhorn', 'Paul Lieberstein', 'B.J. Novak',
                   'David Rogers', 'Matt Sohn']

ddf = ddf.drop(columns = delete_director)
```

```
Directors and IMDB rating
Paul Feig            0.232619
Greg Daniels         0.144802
Ken Kwapis           0.143246
Ken Whittingham      0.044472
Jeffrey Blitz        0.023234
Charles McDougall    0.016920
Randall Einhorn     -0.046541
Paul Lieberstein    -0.068599
```

```
B.J. Novak          -0.084247
David Rogers        -0.124627
Matt Sohn           -0.159096
Name: imdb_rating, dtype: float64
```

If we examine the correlation between each of the directors and the IMDB rating, we observe that Randall Einhorn, Paul Lieberstein, B.J. Novak, David Rogers, and Matt Sohn are associated with negative coefficients. Therefore, in order to reduce dimensionality, we remove them as variables.

Finally, we examine **`writer`** .

In [13]:

```
writer_df = dd['writer'].str.get_dummies(sep=';')
# display first 25 writers.
print(writer_df.sum(axis=0).sort_values(ascending=False).head(25))
# There are 40 writers in total, but only 15 with >= 5 credits to their name.

wdf = writer_df[writer_df.columns[writer_df.sum()>=5]]
# 15 writers.
```

```
Mindy Kaling            21
Paul Lieberstein        16
Gene Stupnitsky         15
B.J. Novak              15
Lee Eisenberg           15
Greg Daniels            12
Brent Forrester         11
Justin Spitzer          11
Jennifer Celotta        11
Michael Schur           10
Charlie Grandy           8
Warren Lieberstein       7
Halsted Sullivan         7
Aaron Shure              6
Daniel Chun              6
Robert Padnick           4
Carrie Kemper            4
Owen Ellickson           4
Steve Hely               3
Allison Silverman        3
Graham Wagner            2
Nicki Schwartz-Wright    2
Steve Carell             2
Stephen Merchant         2
```

```
Ryan Koh                   2
dtype: int64
```

writer

Similarly, we note that only 15 writers out of 40 have written for at least 5 episodes each; we therefore remove the other writers, and focus on these writers.

In [14]:
```
# Correlation analysis.
wdf_corr = abs(wdf.corr())

upper = np.triu(wdf_corr) #  We only need upper triangular of correlation matrix.
n = 0.5  #  If correlation exceeds n, assume highly correlated.

for i in range(15): #  Iterate through each element.
  for j in range(15):
    if upper[i,j] > n and i != j: #  If exceeds n AND not a diagonal element,

      # Print the corresponding variable names.
      print(wdf_corr.columns[i], " ", wdf_corr.columns[j], " ", upper[i,j])

# 'High' correlation.
# Gene Stupnitsky   Lee Eisenberg   1.0
# Halsted Sullivan   Warren Lieberstein   1.0
# Presumably a pair of writers working on the same episodes.

# drop redundant columns and rename for pairs.
wdf = wdf.drop(columns=['Lee Eisenberg','Warren Lieberstein'])
wdf = wdf.rename(columns={"Gene Stupnitsky": "Gene Stupnitsky;Lee Eisenberg",
                          "Halsted Sullivan": "Halsted Sullivan;Warren Lieberstein"})
# left with 13 unique writers/pairs of writers.
```

```
Gene Stupnitsky   Lee Eisenberg   1.0
Halsted Sullivan   Warren Lieberstein   1.0
```

In our correlation analysis, we note that two pairs of writers, G. Stupnitsky and L. Eisenberg, and H. Sullivan and W. Lieberstein, each have a correlation coefficient of 1.0. This occurs because more than one writer can collaborate on an episode together, and these writers have only ever written for an Office episode together. We can therefore treat each of these pairs as a singular writer, thereby removing two redundant categorical variables.

In [15]:
```
imdb_wdf = wdf.copy()
imdb_wdf.insert(0, 'imdb_rating', dd['imdb_rating'])
```

```
print('Writers and IMDB rating')
print(imdb_wdf.corr()['imdb_rating'][1:].sort_values(ascending=False))
# Daniel Chun, Halsted Sullivan;Warren Lieberstein, Aaron Shure, Charlie Grandy are negative.

delete_writer = ['Daniel Chun', 'Halsted Sullivan;Warren Lieberstein',
                 'Aaron Shure', 'Charlie Grandy']

wdf = wdf.drop(columns = delete_writer)
```

```
Writers and IMDB rating
Greg Daniels                           0.233452
Paul Lieberstein                       0.168538
Michael Schur                          0.160538
Jennifer Celotta                       0.129994
Gene Stupnitsky;Lee Eisenberg          0.123633
Mindy Kaling                           0.090354
B.J. Novak                             0.068141
Brent Forrester                        0.061668
Justin Spitzer                         0.010423
Daniel Chun                           -0.034391
Halsted Sullivan;Warren Lieberstein   -0.045190
Aaron Shure                           -0.091404
Charlie Grandy                        -0.114410
Name: imdb_rating, dtype: float64
```

Similar to `director`, the writers Daniel Chun, 'Halsted Sullivan;Warren Lieberstein' (a pair of writers), Aaron Shure, and Charlie Grandy have negative correlation with IMDB ratings. We therefore remove them.

Summary of categorical variables.

In total, we have reduced the number of categorical variables for these three categories from $17 + 60 + 40 = 117$ to $13 + 9 + 6 = 28$ by correlation analysis.

Further preprocessing.

We can now conduct any final proprocessing required before we pass the data to our model of choice. First, we examine our current data, and divide it into a data matrix, X, and a vector of IMDB ratings, y.

In [16]:
```
data_copy.info()

X = data_copy.drop(['imdb_rating', 'director', 'writer', 'main_chars'],axis=1)
```

```
y = data_copy['imdb_rating']

print(data_copy.shape)
print(X.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 186 entries, 0 to 185
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   director      186 non-null    object 
 1   writer        186 non-null    object 
 2   imdb_rating   186 non-null    float64
 3   n_lines       186 non-null    int64  
 4   n_directions  186 non-null    int64  
 5   n_speak_char  186 non-null    int64  
 6   main_chars    186 non-null    object 
dtypes: float64(1), int64(3), object(3)
memory usage: 10.3+ KB
(186, 7)
(186, 3)
(186,)
```

We then scale these numerical variables, and concatenate it with the dataframes containing our previously-discussed categorical variables, `writer`, `main_chars` and `director`.

In [17]:

```
# Just to demonstrate what X contains.
X.columns
```

Out[17]:

```
Index(['n_lines', 'n_directions', 'n_speak_char'], dtype='object')
```

In [18]:

```
# Scale numerical predictor variables.
X_scaled = scale(X)

# Add categorical variables to scaled X by first converting them to Numpy arrays.

wdf_np = wdf.to_numpy()
mcdf_np = mcdf.to_numpy()
ddf_np = ddf.to_numpy()
X_scaled_cat = np.concatenate((X_scaled, ddf_np, wdf_np, mcdf_np), axis = 1)
```

Principal Component Analysis

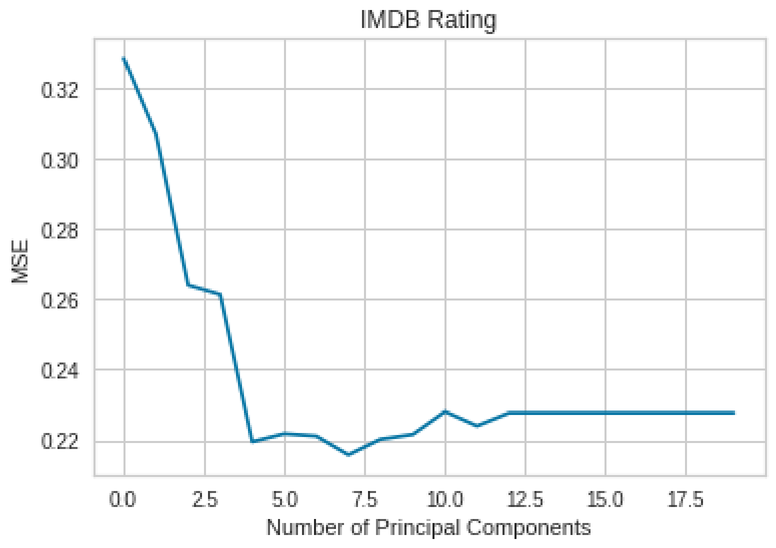
Our next step is to conduct Principal Component Analysis to further reduce dimensionality and retrieve only the most relevant components. We do this in two ways: by maximising the maximum likelihood estimate (MLE), and by minimising the Mean Squared Error (MSE).

The first is done by specifying the parameter `n_components = 'mle'` , while the second is done by analysing the graph of number of principal components versus MSE.

Applying PCA to maximise the maximum likelihood estimate (MLE):

In [19]:
```
# Split dataset into training and testing
X_train,X_test,y_train,y_test = train_test_split(X_scaled_cat,y,test_size=0.3,
                                                 random_state = 42)
pca = PCA(n_components = 'mle', svd_solver = 'full')
X_train_reduced = pca.fit_transform(X_train)
regr = LinearRegression()
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

mse = []

# Calculate MSE with only the intercept
score = -1*model_selection.cross_val_score(regr,
           np.ones((len(X_train_reduced),1)), y_train, cv=cv,
           scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using cross-validation, adding one component at a time
for i in np.arange(1, 20):
    score = -1*model_selection.cross_val_score(regr,
               X_train_reduced[:,:i], y_train, cv=cv, scoring='neg_mean_squared_error').mean()
    mse.append(score)

# Plot cross-validation results
plt.plot(mse)
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.title('IMDB Rating')
```

Out[19]:
```
Text(0.5, 1.0, 'IMDB Rating')
```



Based on the graph above, the number of principal components that minimises the MSE is 7.

In [20]:
```
print(pca.n_components_)
```

```
12
```

However, the maximum likelihood estimate for the number of principal number of components is 12. We therefore attempt to fit and transform our training data with these two different parameters.

We therefore implement Principal Component Analysis (PCA) in two different ways – by first minimising Mean Squared Error (MSE) in order to select the optimal number of components, and secondly, by selecting the Maximum Likelihood Estimator (MLE) for the optimal number of components.

3. Model Fitting and Tuning

In this section you should detail your choice of model and describe the process used to refine and fit that model. You are strongly encouraged to explore many different modeling methods (e.g. linear regression, regression trees, lasso, etc.) but you should not include a detailed narrative of all of these attempts. At most this section should mention the methods explored and why they were rejected - most of your effort should go into describing the model you are using and your process for tuning and validatin it.

For example if you considered a linear regression model, a classification tree, and a lasso model and ultimately settled on the linear regression approach then you should mention that other two approaches were tried but do not include any of the code or any in depth discussion of these models beyond why *they were rejected. This section should then detail is the development of the linear regression model in terms of features used, interactions considered, and any additional tuning and validation which ultimately led to your final model.*

This section should also include the full implementation of your final model, including all necessary validation. As with figures, any included code must also be addressed in the text of the document.

Model Selection

We focused our efforts on five selected model types - Linear Regression, Ridge Regression, Lasso Regression, Support Vector Regression and Decision Tree Regression.

We then determined the best parameters for each model using grid search and cross-validation separately for three sets of data $-$ PCA-transformed with 7 components, PCA-transformed with 12 components, and the untransformed data.

We then represented each model's fit with the root mean squared error (RMSE) and R^2 score for both the training and test data in the table below.

Below is a table showing our evaluation metrics for the models with PCA that minimises the MSE, i.e. 7 components.

index	Model	Training RMSE	Test RMSE	Training R-Square	Test R-Square
0	LinearRegression()	0.433828	0.3875041397183159	0.41488860085642787	0.23204869416471652
1	Ridge(alpha=12.06)	0.437077	0.3798712113628861	0.40609286357410745	0.26200443202082035
2	Lasso(alpha=0.01)	0.435583	0.38135783618541974	0.41014550600209976	0.25621684248927745
3	SVR(C=100, kernel='linear')	0.43638	0.38406487577151016	0.40798531913358227	0.24561998871288648
4	DecisionTreeRegressor(min_samples_split=8)	0.164884	0.5660756078302345	0.9154795958725283	-0.6388147851322408

Below is a table showing evaluation metrics for the models with PCA that uses the maximum likelihood estimate for the number of components, i.e. 12 components.

index	Model	Training RMSE	Test RMSE	Training R-Square	Test R-Square
0	LinearRegression()	0.421272	0.3962175811665945	0.4482665763915772	0.197124001667363
1	Ridge(alpha=15.98)	0.429088	0.378893767943658	0.42760473811761257	0.26579740627831894
2	Lasso(alpha=0.016681005372000558)	0.431274	0.37775568029536416	0.42175737317962214	0.27020144795313594
3	SVR(C=0.1, kernel='linear')	0.430777	0.3851488615749271	0.42309076650475774	0.24135565075222665

index	Model	Training RMSE	Test RMSE	Training R-Square	Test R-Square
4	DecisionTreeRegressor(min_samples_split=8)	0.160542	0.573705437676258	0.9198724476499699	-0.6832899199787121

Below is a table showing our evaluation metrics for the models without PCA.

index	Model	Training RMSE	Test RMSE	Training R-Square	Test R-Square
0	LinearRegression()	0.384725	0.3963947530451784	0.5398451295086462	0.19640581621011122
1	Ridge(alpha=17.487)	0.416302	0.3675736330926979	0.46120922395030606	0.3090132938628951
2	Lasso(alpha=0.016681005372000558)	0.426981	0.36743366145922696	0.43321256956755994	0.30953944761334184
3	SVR(C=0.1, kernel='linear')	0.411268	0.3697348531967613	0.4741610449858048	0.3008638265591439
4	DecisionTreeRegressor(min_samples_split=8)	0.198717	0.4905563367415925	0.8772347393349775	-0.23071861773080005

By looking at the test R-Squared values for the models with and without PCA, we observe that the models without PCA are generally more accurate in predicting the test data due to having a higher R^2. **Therefore, we do not use PCA to fit and transform the data, and proceed with fitting the model on untransformed data**.

We thus select the Lasso Regression model with specified parameter `alpha` above as the most accurate model.

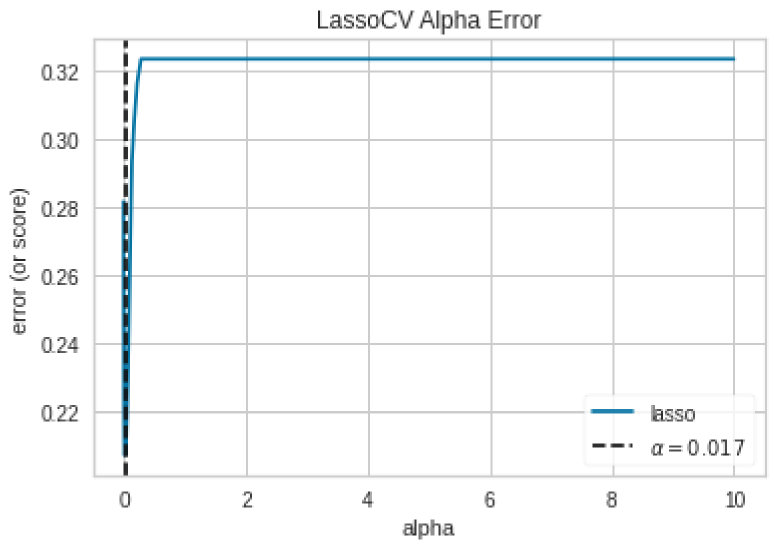
Model Tuning

We first demonstrate how we obtained the optimal value of the parameter `alpha` . The procedure is as follows:

1. Generate a list of possible `alpha` values.
2. Iterate over each of these values using Lasso Cross Validation function from `sklearn` .
3. Obtain the optimal value.

In [21]:
```
# Create a list of alphas to cross-validate against
alphas = np.logspace(-10, 1, 100)

# Instantiate the linear model and visualizer
# LassoCV iterates over all defined alpha values.
model_lcv = LassoCV(alphas=alphas, max_iter = 100000)
visualizer = AlphaSelection(model_lcv)
visualizer.fit(X_train, y_train)
visualizer.show()
```

```
print('\n', model_lcv.alpha_)
```



```
 0.016681005372000558
```

We then fit the Lasso regression model onto the training data, and obtain the coefficients corresponding to each variable.

In [22]:
```
# Fit model.
lasso_best = Lasso(model_lcv.alpha_).fit(X_train, y_train)

# Display coefficients.
lasso_best.coef_
```

Out[22]:
```
array([ 0.15154645,  0.0400675 ,  0.03884847,  0.        ,  0.05542537,
       -0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        , -0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        , -0.        ,
        0.01574135, -0.        , -0.21726889,  0.04288747, -0.        ,
        0.40305407, -0.        ,  0.        , -0.        ,  0.        ,
       -0.        ])
```

The code in the next cell creates a dataframe to better display the weights.

In [23]:
```
# Creating dataframe to visualize weights, name of variables, and their respective categories.
weights = lasso_best.coef_
```

```
names = np.concatenate([X.columns, ddf.columns, wdf.columns, mcdf.columns])
category = np.concatenate([['Numerical']*len(X.columns), ['Director']*len(ddf.columns),
                          ['Writer']*len(wdf.columns), ['Main Character']*len(mcdf.columns)])

model_df_final = pd.DataFrame({'Name': names, 'Category': category, 'Weight': weights})

# An additional dataframe that sorts the rows by weight.
model_df_final_sorted = model_df_final.sort_values(by = 'Weight', ascending = False)

# This dataframe contains only the rows with non-zero weights.
final_df_nonzero = model_df_final.loc[model_df_final_sorted.Weight != 0]
final_df_nonzero_sorted = model_df_final_sorted.loc[model_df_final_sorted.Weight != 0]
final_df_nonzero_sorted.head(8)

display(model_df_final_sorted.head(9))
display(model_df_final_sorted.tail(7))
final_df_nonzero.head(8)
```

	Name	Category	Weight
25	Michael	Main Character	0.403054
0	n_lines	Numerical	0.151546
4	Greg Daniels	Director	0.055425
23	Kelly	Main Character	0.042887
1	n_directions	Numerical	0.040067
2	n_speak_char	Numerical	0.038848
20	Creed	Main Character	0.015741
17	Paul Lieberstein	Writer	0.000000
29	Stanley	Main Character	0.000000

	Name	Category	Weight
8	Paul Feig	Director	0.000000
7	Ken Whittingham	Director	0.000000
6	Ken Kwapis	Director	0.000000

	Name	Category	Weight
5	Jeffrey Blitz	Director	-0.000000
3	Charles McDougall	Director	0.000000
30	Toby	Main Character	-0.000000
22	Erin	Main Character	-0.217269

Out[23]:

	Name	Category	Weight
0	n_lines	Numerical	0.151546
1	n_directions	Numerical	0.040067
2	n_speak_char	Numerical	0.038848
4	Greg Daniels	Director	0.055425
20	Creed	Main Character	0.015741
22	Erin	Main Character	-0.217269
23	Kelly	Main Character	0.042887
25	Michael	Main Character	0.403054

4. Discussion & Conclusions

In this section you should provide a general overview of your final model, its performance, and reliability. You should discuss what the implications of your model are in terms of the included features, predictive performance, and anything else you think is relevant.

This should be written with a target audience of a NBC Universal executive who is with the show and university level mathematics but not necessarily someone who has taken a postgraduate statistical modeling course. Your goal should be to convince this audience that your model is both accurate and useful.

Finally, you should include concrete recommendations on what NBC Universal should do to make their reunion episode a popular as possible.

Keep in mind that a negative result, i.e. a model that does not work well predictively, that is well explained and justified in terms of why it failed will likely receive higher marks than a model with strong predictive performance but with poor or incorrect explinations / justifications.

Model overview

$y = \sum_{n=0}^{8} a_n X_n$ where

y denotes the predicted IMDB rating,

X_n is the variable in $(1,$ 'n_lines', 'n_directions', 'n_speak_char', 'Greg Daniels', 'Creed', 'Erin', 'Kelly', 'Michael'$)$,

and a_n is the weight of the corresponding variable or intercept in $(8.031079, 0.151546, 0.040067, 0.038848, 0.055425.0.015741, -0.217269, 0.042887, 0.403054)$.

Our final model is a Lasso regression model - the IMDB rating for an episode is modeled as a weighted sum of the relevant components - the number of dialogue lines, the number of speaking characters, the number of stage directions, the characters in the episode and the writer for the episode. The last three variables are categorical in nature; we therefore represent a person with 1 if he/she is involved, and 0 otherwise.

However, unlike a linear regression model, the regularisation parameter `alpha` acts as a penalty, which forces some of the weights of the independent variables to be equal to 0. This means that our model also performs variable selection, giving non-zero weights only to the most appropriate variables.

Therefore, for each individual person or numerical variable, there is a weight associated with each of them. This weight can be positive or negative $-$ if positive, this implies that the variable can improve the IMDB rating; if negative, this implies that the variable has a negative impact on the rating.

Model performance and reliability

In general, our model is poor at predicting the IMDB rating for an episode, given the relevant data. We can see this from the cross-validated R^2 squared value based on the test data (≈ 0.309539), which indicates that only approximately 30.1% of the variance of the IMDB rating can be explained by the model.

However, if we examine the weights associated with each variable, we can determine with some amount of certainty which variables would contribute to the IMDB rating, and which would only be a detriment. We can therefore use these weights to provide recommendations to improve the IMDB rating of the reunion episode.

The variable selection done by the Lasso regression model, while simplifying our resulting model, also prevents us from examining the impact of variables on the IMDB rating whose weights have been set to zero. This is one limitation of our selected model. If a variable's weight is set to zero, the most we can say about variables with non-zero weights is that they affect the IMDB rating more significantly than the zero-weight variables.

Model implications and recommendations

In [24]:
```
model_df_final_sorted.loc[model_df_final['Category'] == 'Director'].head(len(ddf.columns))
```

Out[24]:

	Name	Category	Weight
4	Greg Daniels	Director	0.055425
8	Paul Feig	Director	0.000000
7	Ken Whittingham	Director	0.000000
6	Ken Kwapis	Director	0.000000
5	Jeffrey Blitz	Director	-0.000000
3	Charles McDougall	Director	0.000000

Greg Daniels is the only director whose weight is non-zero - we therefore recommend him.

In [25]:
```
model_df_final_sorted.loc[model_df_final['Category'] == 'Main Character'].head(len(mcdf.columns))
```

Out[25]:

	Name	Category	Weight
25	Michael	Main Character	0.403054
23	Kelly	Main Character	0.042887
20	Creed	Main Character	0.015741
29	Stanley	Main Character	0.000000
28	Ryan	Main Character	-0.000000
27	Phyllis	Main Character	0.000000
26	Oscar	Main Character	-0.000000

	Name	Category	Weight
24	Meredith	Main Character	-0.000000
21	Darryl	Main Character	-0.000000
19	Angela	Main Character	-0.000000
18	Andy	Main Character	0.000000
30	Toby	Main Character	-0.000000
22	Erin	Main Character	-0.217269

Michael, Kelly, and Creed each have non-zero and positive weights - they should therefore be included in the reunion episode. Moreover, we observe that Erin has a significant negative weight, and therefore should not be included.

Dwight, Jim, Pam and Kevin are the main characters who have appeared in over 180 episodes - this makes it difficult to determine what impact they have on the IMDB ratings, as they would have appeared in both good and bad episodes. On this basis, we recommend that these four characters should appear, as they appear to be mainstays of the Office series.

In [26]:
```
model_df_final_sorted.loc[model_df_final['Category'] == 'Writer'].head(len(wdf.columns))
```

Out[26]:

	Name	Category	Weight
17	Paul Lieberstein	Writer	0.0
15	Michael Schur	Writer	0.0
16	Mindy Kaling	Writer	0.0
14	Justin Spitzer	Writer	0.0
13	Jennifer Celotta	Writer	-0.0
12	Greg Daniels	Writer	0.0
11	Gene Stupnitsky;Lee Eisenberg	Writer	0.0
10	Brent Forrester	Writer	0.0
9	B.J. Novak	Writer	0.0

The weights for all writers is zero.

We therefore must rely on the correlation analysis done in the previous section.

In [27]:
```
print('Correlation between Writers and IMDB rating')
print(imdb_wdf.corr()['imdb_rating'][1:].sort_values(ascending=False))
```

```
Correlation between Writers and IMDB rating
Greg Daniels                       0.233452
Paul Lieberstein                   0.168538
Michael Schur                      0.160538
Jennifer Celotta                   0.129994
Gene Stupnitsky;Lee Eisenberg      0.123633
Mindy Kaling                       0.090354
B.J. Novak                         0.068141
Brent Forrester                    0.061668
Justin Spitzer                     0.010423
Daniel Chun                       -0.034391
Halsted Sullivan;Warren Lieberstein   -0.045190
Aaron Shure                       -0.091404
Charlie Grandy                    -0.114410
Name: imdb_rating, dtype: float64
```

We then recommend Greg Daniels as the Writer - this is a plus as he is also the recommended director. If additional writers should be required to assist him, then the following writers are also recommended (in order of general effect on the IMDB rating): Paul Lieberstein, Michael Schur, Jennifer Celotta, and the pair of writers Gene Stupnitsky and Lee Eisenberg.

In [28]:
```
model_df_final_sorted.loc[model_df_final['Category'] == 'Numerical'].head(len(X.columns))
```

Out[28]:

	Name	Category	Weight
0	n_lines	Numerical	0.151546
1	n_directions	Numerical	0.040067
2	n_speak_char	Numerical	0.038848

We observe that the number of speaking characters and the number of stage directions in a script are not very significant, whereas the number of dialogue lines does seem to affect the IMDB rating - this may be because longer episodes tend to be more popular. Since a reunion episode would normally be longer in length, these three variables are not very relevant.

In [29]:
```
print(d.n_lines.median())
print(d.n_directions.median())
print(d.n_speak_char.median())
```

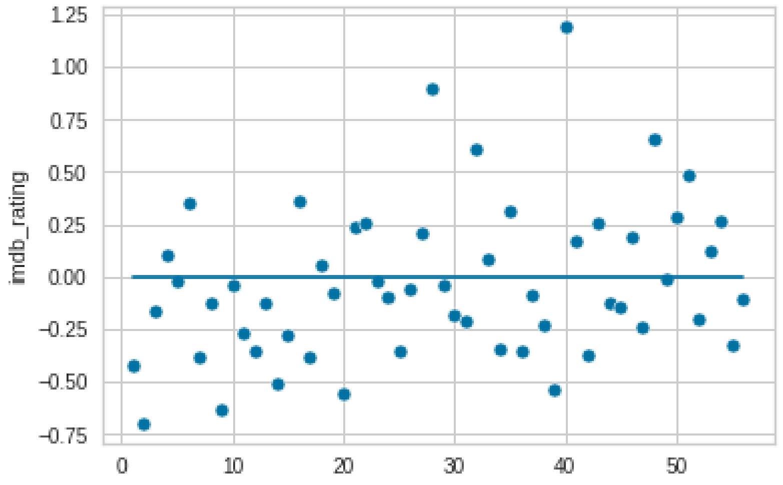
```
281.0
46.0
20.0
```

Our recommendation is then simple – that the number of lines, speaking directions and speaking characters should be greater than their median values (281, 46, 20 respectively). However, this should not be adhered to strictly, as the monetary cost of increasing any of these variables may very well outweigh the potential increase in IMDB rating.

Prediction

In [30]:
```
# Plot of test data residuals versus test episode index.
sns.scatterplot(x = np.linspace(1,56,56), y = y_test - lasso_best.predict(X_test))

# Zero line.
sns.lineplot(x = np.linspace(1,56,56), y = np.zeros(56))

print((y_test - lasso_best.predict(X_test) < 0).sum())
```

```
36
```



The above plot is a plot of the residuals (i.e. difference between the test data outputs and the predicted data outputs based on the test inputs) versus the test data index. We observe that in general, the residuals are more likely to be negative (for the test data in this model, roughly 64.3% are negative).

This indicates that using our lasso regression model to predict the IMDB rating would be more likely to result in a prediction that is lower than the actual rating.

We demonstrate a prediction with an example below; based on our recommendations above, we can use our model to predict the IMDB rating of the reunion episode by setting all categorical variables with positive weights (Director, Writer, Main Character) to 1 (selecting only one director and writer), setting the rest to 0, and setting `n_lines`, `n_speak_char` and `n_directions` to their maximum values in the original dataset, but scaled.

In [31]:
```
# Predicting the rating of the reunion episode, based on 'optimal' parameters.
# As the weights of writers are 0, we set their corresponding values to 0.

opt_par = np.array([scale(d.n_lines).max(),scale(d.n_speak_char).max(),scale(d.n_directions).max(),
                    0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,0,0,0])

print("The predicted rating is: ", lasso_best.predict(opt_par.reshape(1,-1))[0])
```

```
The predicted rating is:  9.608388533079899
```

In [32]:
```
1-(d.imdb_rating >= 9.6).sum()/186
```

Out[32]:
```
0.9838709677419355
```

The predicted best rating of ≈ 9.6 is better than roughly 98% of all Office episodes' IMDB ratings. However, we must note that there is non-negligible uncertainty in any prediction, as our model is not very robust, but given that it still remains somewhat accurate, this still demonstrates the effects of each variable on the IMDB rating.

Summary

In summary, we have the following recommendations for the reunion episode:

1. **Director and Writer:** Greg Daniels

   Supporting writers (if necessary): Paul Lieberstein, Michael Schur, Jennifer Celotta, the pair Gene Stupnitsky & Lee Eisenberg.

2. **Main Characters:** Dwight, Jim, Pam, Kevin, Michael, Kelly, Creed

   This is only a list of main characters that should be included, and does not mean that other main characters should be excluded.

3. **Number of dialog lines:** At least 281.

4. **Number of stage directions:** At least 46.

5. **Number of speaking characters:** At least 20.

Our recommendations should not be taken as absolute rules, as it is entirely possible that characters associated with a small or even negative weighting can be involved in a storyline that proves to be extremely popular, which would more than negate their non-significant or negative impact on the predicted IMDB rating.

For example, Dwight and Angela's relationship has been present in all but one season of the Office, and has been the focus of some of the episodes[1] – but her weight is set to 0 here. Stanley, who has appeared in several lists of characters by popularity[2][3][4], also does not appear in our final model.

This is why our criteria for recommending main characters are less strict than for the other categories of variables.

[1] see https://theoffice.fandom.com/wiki/Dwight-Angela_Relationship

[2] see https://www.cinemablend.com/television/2475607/the-29-best-characters-from-the-office (Ranked 8^{th})

[3] see https://www.thetoptens.com/best-characters-from-the-office/ (Ranked 8^{th})

[4] see https://www.ranker.com/crowdranked-list/best-the-office-u-s-characters (Ranked 7^{th})

Limitations.

Variable selection prevents examining impact of variables with zero weighting

As mentioned above, the variable selection done by the Lasso regression model, while simplifying our resulting model, also prevents us from examining the impact of variables on the IMDB rating whose weights have been set to zero.

Non-quantifiable factors

There are several reasons for why our model cannot be extremely accurate: episodes of the Office have several non-quantifiable aspects such as the acting quality, quality of the story, set design, and other such variables. These all contribute to the popularity of any one episode, and cannot be captured and modeled mathematically. As our approach towards this problem is purely statistical in nature, we cannot make any definite conclusions for these factors.

Change in audience preferences

The IMDB ratings and data were drawn from all 186 episodes of the Office, which ran from 2005 to 2013. It is now 2022. This model has been fitted and trained on that dataset, and implictly reflects the tastes and preferences of that period of time. It is therefore inherently limited in how well it can model current audiences' preferences - this only increases the uncertainty present in any predictions it makes.