Machine Learning in Python - Group 14

Contributors: Sondre Wikberg (s1863042), Tomas Maksimovic (s1903823), Emily Georgiadou (s1946112), Fraser Singh (s2268606)

In [1]:
```
# Downloading libraries and required packages
# Data libraries
import numpy as np
import pandas as pd


# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import linregress

# Plotting and Figure defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
styles = [dict(selector="caption", props=[("font-size","110%"),("font-weight", "b
old")])]

# sklearn modules that are necessary
import sklearn
from collections import Counter
from fuzzywuzzy import fuzz
from sklearn.model_selection import train_test_split, cross_validate, KFold, Grid
SearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler
from matplotlib.colors import ListedColormap

#Suppress specific warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Abstract

This project consisted of analysis of pre-existing data on the US rendition of the TV mockumentary "The Office", combined with a dataset containing information on the duration and names of guest stars in each episode. The full set of data was then fit to various regression models with `imdb_rating` as a response, with an aim to determine what combination of episode features might result in a high rating in the event of a reunion episode. We find a strong positive correlation between `imdb_rating` and episode duration, number of lines spoken in the episode, number of directed cuts, and Michael appearing as a character.

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

The aim of this work is to assess a public dataset which details the US TV show 'The Office' and to ascertain the main influencing factors on a given episode's 'IMDb rating', which is determined by public vote through IMDB's website (an abbreviation of Internet Movie Database). The Office, produced by NBC, is one of the most watched and critically acclaimed shows of American TV over the past 20 years, potentially making a reunion episode highly lucrative for NBC. Given a dataset of all episodes for "The Office", we set out to determine the factors which will maximise the IMDb rating for an upcoming reunion episode of the show. Previewed below, the provided dataset contained 186 observations (episodes) with the following variables describing various metrics of the episodes row by row:

`season:` Which season the episode was in

`episode`: The episode number within the season

`episode_name`

`director`: director(s) on the given episode

`writer`: writer(s) on the given episode

`imdb_rating`: average IMDb rating for the given episode

`total_votes`: number of votes on IMDb for the given episode

`air_date`

`n_lines`: Number of spoken lines within episode

`n_directions`: number of scripted actor directions (e.g. [actor1 looks at actor2])

`n_words`: number of spoken words within the episode

`n_speak_char`: number of characters who spoke within the episode

`main_chars`: main characters in the episode

In [2]:
```
# Load standard data
data = pd.read_csv("the_office.csv"); data.head(2).style.set_caption('Original da
ta').set_table_styles(styles)
```

Out[2]:

Original data

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_dir
0	1	1	Pilot	Ken Kwapis	Ricky Gervais;Stephen Merchant;Greg Daniels	7.600000	3706	2005-03-24	229	
1	1	2	Diversity Day	Ken Kwapis	B.J. Novak	8.300000	3566	2005-03-29	203	

1.1 Introducing and additional data:

In addition to the provided dataset, we have decided to include another dataset found online (subsequently referred to as the 'Saleem dataset ' [4] with additional metrics about 'The Office', (found at: [https://www.datainsightonline.com/post/exploring\-interesting\-information\-from\-the\-office\-dataset (https://www.datainsightonline.com/post/exploring-interesting-information-from-the-office-datasetta)] (https://www.datainsightonline.com/post/exploring\-interesting\-information\-from\-the\-office\-dataset (https://www.datainsightonline.com/post/exploring-interesting-information-from-the-office-dataset))).

In [3]:
```
#Load in the Saleem dataset found online
edata = pd.read_csv("the_office_series.csv"); edata.head(2).style.set_caption('Saleem data').set_table_styles(styles)
```

Out[3]:

Saleem data

	Unnamed: 0	Season	EpisodeTitle	About	Ratings	Votes	Viewership	Duration	Date	GuestStars	D
0	0	1	Pilot	The premiere episode introduces the boss and staff of the Dunder-Mifflin Paper Company in Scranton, Pennsylvania in a documentary about the workplace.	7.500000	4936	11.200000	23	24 March 2005	nan	
1	1	1	Diversity Day	Michael's off color remark puts a sensitivity trainer in the office for a presentation, which prompts Michael to create his own.	8.300000	4801	6.000000	23	29 March 2005	nan	

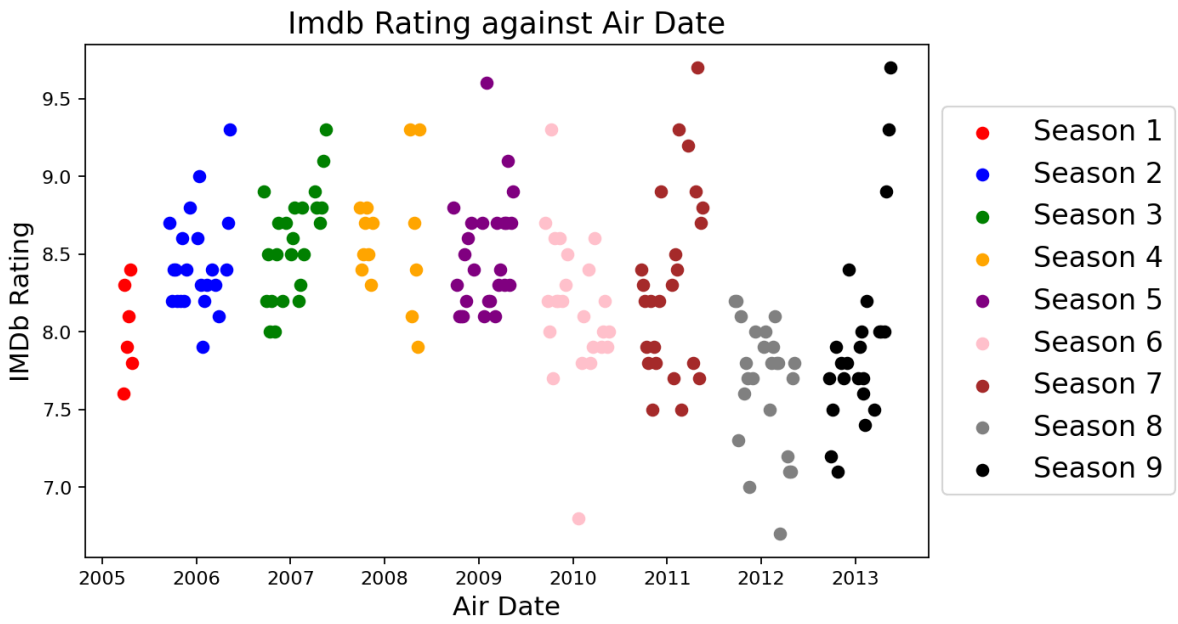
2.Exploratory Data Analysis and Feature Engineering

2.1 Exploratory Data Analysis

We start by plotting the episodes by air date and imdb rating, with the colours representing the season in which each episode appeared. It seems the median rating for each season was slowly increasing for the first 4 seasons while declining more rapidly from season 6. However, we note that seasons 5,6,7,8,9 each have some large outliers in the in terms of ratings. We aim to make recommendations for a reunion episode of "The Office" airing on an unspecified future date and so the air date for each episode is likely to be irrelevant to further exploration in this report. The plot is useful, however, as a way of visualising how ratings varied with season throughout the lifetime of the show, while also illuminating the fact that there were outliers, in terms of ratings, in some of the later seasons.

In [4]:
```
data['air_date_datetime'] = pd.to_datetime(data['air_date'])

# Create a dictionary that maps each season to a color
colors = {1: 'red', 2: 'blue', 3: 'green', 4: 'orange', 5: 'purple', 6: 'pink',
7: 'brown', 8: 'gray', 9: 'black'}

# Plot data for each season
fig, ax = plt.subplots()
for season in range(1, 10):
    season_data = data[data['season'] == season]
    ax.scatter(season_data['air_date_datetime'], season_data['imdb_rating'], colo
r=colors[season], label=f'Season {season}')

# Add labels and legend
ax.set_xlabel('Air Date', size=14)
ax.set_ylabel('IMDb Rating', size=14)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), prop={'size': 15})
ax.set_title("Imdb Rating against Air Date", size=16)

# Show
plt.show()

#Remove 'air_date_datetime' column from df in order to ensure we are working only
on the data we actually imported
data = data.drop(['air_date_datetime'], axis=1)
```

Out[4]:



From the Saleem dataset we selected the following columns:

- *GuestStars* - extra stars that appeared on that episode
- *Duration* - duration of episodes in minutes

While `Viewership` was one of the variables in the Saleem data set that may be correlated we `imdb_rating`, we chose not to include it in our dataset for several reasons, including the fact that NBC is unlikely to have direct control over viewership prior to an episode's release. As seen below, some data cleaning was required as the indexing from each data source was conflicting, necessitating manual corrections.

In [5]:

```
#Select only columns of Saleem data frame that contains new, relevant informatio
n.
#Include date and EpisodeTitle for now for easier interpretation
e_col = edata[["EpisodeTitle", "Date", "Duration","GuestStars"]]

#Check if number of rows in main dataset are same as in Saleem set
print(f"Number of rows in main data frame:{data.shape[0]} \nNumber of rows in Sal
eem data frame:{edata.shape[0]}\n")

# count the number of occurrences of each date in each dataset
date_counts_edata = e_col['Date'].value_counts()
date_counts_data = data['air_date'].value_counts()

# filter the dates that appear more than once
duplicates_edata = date_counts_edata[date_counts_edata > 1].index.tolist()
duplicates_data = date_counts_data[date_counts_data > 1].index.tolist()

#Print observations with a repeated date in Saleem dataset
Extra_duped=pd.DataFrame(columns=['EpisodeTitle','Date'])
for date in duplicates_edata:
        dupedI=edata[edata["Date"]==date][["Date","EpisodeTitle"]]
        Extra_duped=Extra_duped.append(dupedI, ignore_index=True)


#Print observations with a repeated date in main dataset
duped=pd.DataFrame(columns=["episode_name","air_date"])
for date in duplicates_data:
        dupedI=data[data["air_date"]==date][["air_date","episode_name"]]
        duped=duped.append(dupedI, ignore_index=True)

#Show the data in question
display(Extra_duped.style.set_caption('Episode names of episodes that were releas
ed on same day in Saleem data:').set_table_styles(styles))
display(duped.style.set_caption('Episode names of episodes that were released on
same day in Original data:').set_table_styles(styles))
```

```
Number of rows in main data frame:186
Number of rows in Saleem data frame:188
```

Out[5]:

Episode names of episodes that were released on same day in Saleem data:

	EpisodeTitle	Date
0	Niagara: Part 1	8 October 2009
1	Niagara: Part 2	8 October 2009
2	The Delivery: Part 1	4 March 2010
3	The Delivery: Part 2	4 March 2010
4	Dream Team	9 April 2009
5	Michael Scott Paper Company	9 April 2009
6	Junior Salesman	31 January 2013
7	Vandalism	31 January 2013

Out[5]:

Episode names of episodes that were released on same day in Original data:

	episode_name	air_date
0	Dream Team	2009-04-09
1	The Michael Scott Paper Company	2009-04-09
2	Junior Salesman	2013-01-31
3	Vandalism	2013-01-31

The first observation we make is that the Saleem data has two more rows than the main data frame, which disrupts the indexing of said episodes. We need both datasets to have the same number of entries (episodes) in order to properly combine them. The code above investigates this discrepancy by printing entries containing a repeated date. We note that both datasets confirm that the episodes "Dream Team" and "The Michael Scott Paper Company" shared release date (9 April 2009). Similarly, both datasets confirm that the episodes "Junior Salesman" and "Vandalism" were released on the same date (31 January 2013), thus, no cleaning is needed for these dates. In the Saleem dataset, the two-part episodes "Niagara" and "The Delivery" have been split into two observations. We check what these dates correspond to in the main dataset.

In [6]:
```
((data[data["air_date"]=='2009-10-08'][["episode_name", "air_date"]]).append((dat
a[data["air_date"]=='2010-03-04'][["episode_name", "air_date"]]), ignore_index=Tr
ue)).style.set_caption('Multipart episodes in Saleem data').set_table_styles(styl
es)
```

Out[6]:

Multipart episodes in Saleem data

	episode_name	air_date
0	Niagara (Parts 1&2)	2009-10-08
1	The Delivery (Parts 1&2)	2010-03-04

In the main dataset, we note that these two-part episodes have been identified with one observation each in the main dataset. From above, we recall that the Saleem dataset had two additional observations, compared to the main one. In order to properly combine the data, we thus need to combine the entries corresponding to "Niagara" and "The Delivery" in the Saleem dataset. To do so, we first print the entries corresponding to these episodes for the relevant variables in the Saleem dataset.

In [7]:
```
((e_col[e_col["Date"]==' 8 October 2009']).append((e_col[e_col["Date"]==' 4 March 2010'])))).style.set_caption('Multipart episodes in Original Data').set_table_styles(styles)
```

Out[7]:

Multipart episodes in Original Data

	EpisodeTitle	Date	Duration	GuestStars
94	Niagara: Part 1	8 October 2009	30	nan
95	Niagara: Part 2	8 October 2009	19	nan
107	The Delivery: Part 1	4 March 2010	30	nan
108	The Delivery: Part 2	4 March 2010	30	Mellisa Rauch

We see that "Niagara: Part 1" and "Niagara: Part 2" both had no guest stars. However, the duration of each part was different. This raises the question of what duration to assign to the observation (episode) in which we will combine these two. A quick look at the main dataset confirms that there were some episodes, such as "Fun Run", "Dunder Mifflin Infinity", and "Launch Party", which had two parts, but that were treated as a single observation in both sets of data. The Saleem dataset indicates that each of these episodes had a total duration of 42 minutes. The median duration of each episode is checked with

In [8]:
```
print("Median episode duration:", e_col['Duration'].median())
```

```
Median episode duration: 23.0
```

A duration of 42 minutes is almost 20 minutes longer than the median duration of each episode. We thus conclude that the Saleem data, when it combines data for two-part episodes, simply adds the duration of each part. Hence, to combine the data for episodes "Niagara: Part 1" and "Niagara: Part 2" into a single entry "Niagara", we simply add the durations (as opposed to, for example, taking an average). The viewership and Guest Stars entries are left untouched as they were the same for both parts.

In [9]:
```
e_col.at[94, "EpisodeTitle"] = "Niagara (Parts 1&2)"
e_col.at[94, "Duration"] = e_col["Duration"][94:96].sum()
e_col = e_col.drop([95])
```

Similarly, we note that "The Delivery: Part 1" and "The Delivery: Part 2", both released 4 March 2010, have the same viewership and duration. However, part two features guest star Mellisa Rauch. As for "Niagara", we simply add up the duration of each part when combining these two episodes into a single observation "The Delivery". We conclude that Mellisa Rauch was a guest star in the overall episode, and so we include her name in the "GuestStars" column for the new observation. Finally, we reset the index of the e_col dataset as we have deleted some rows.

In [10]:
```
e_col.at[108, "EpisodeTitle"] = "The Delivery (Parts 1&2)"
e_col.at[108, "Duration"] = e_col["Duration"][107:109].sum()
e_col = e_col.drop([107])
e_col= e_col.reset_index()
e_col = e_col.drop(["index"], axis=1)
```

We are now interested to see whether there are any remaining two-part episodes that have not been combined to a single observation. We want all such episode to be combined for uniformity of the data. In both datasets, we print any observations for which the episode_name contains "Part 1" or "Part 2".

In [11]:
```
two_part_index1 = []
two_part_index2 =[]
for i in range(data.shape[0]):
    if ("Part 1" in data["episode_name"][i]) or ("Part 2" in data["episode_name"]
[i]):
        two_part_index1.append(i)
    if ("Part 1" in e_col["EpisodeTitle"][i]) or ("Part 2" in e_col["EpisodeTitl
e"][i]):
        two_part_index2.append(i)

#Show the split multi part episodes
display(pd.DataFrame(data.iloc[two_part_index1]).style.set_caption('Split multi p
art episodes in Original Data').set_table_styles(styles))
display(pd.DataFrame(e_col.iloc[two_part_index2]).style.set_caption('Split multi
part episodes in Saleem Data').set_table_styles(styles))
```

Out[11]:

Split multi part episodes in Original Data

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_directions
78	5	16	Lecture Circuit (Part 1)	Ken Kwapis	Mindy Kaling	8.200000	1838	2009-02-05	237	52
79	5	17	Lecture Circuit (Part 2)	Ken Kwapis	Mindy Kaling	8.200000	1808	2009-02-12	252	60

Out[11]:

Split multi part episodes in Saleem Data

	EpisodeTitle	Date	Duration	GuestStars
78	Lecture Circuit: Part 1	5 February 2009	30	nan
79	Lecture Circuit: Part 2	12 February 2009	30	nan

In each dataset, we combine observations 78 and 79, corresponding to "Lecture Circuit" parts 1 and 2, into a new observation by considering each variable separately. We shall give the new observation the name "Lecture Circuit (Parts 1&2)". In the main dataset, both episodes have the same season number, director, writer, and imdb_rating. Hence, these variables do not need to be changed for observation "Lecture Circuit (Parts 1&2)". Downstream, we will exclude variable "episode", indicating the episode number within a season. Hence, we will set the "episode" variable to 16 for "Lecture Circuit (Parts 1&2)" (same as for "Lecture Circuit: Part 1" in original data). For each of total_votes, n_lines, n_directions, n_words, and n_speak_char, the entry in "Lecture Circuit (Parts 1&2)" is set to the average of the corresponding entries in "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2". Later in the project, we will transform the air_date column into dummy-variables for the day of the week on which the episode was released. "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2" were released exactly a week apart (same weekday), and so we may assign the air_date column of "Lecture Circuit (Parts 1&2)" the date "2009-02-05" as a placeholder (corresponding to the day on which "Lecture Circuit: Part 1" was released). Finally, we compare the entries in main_char for "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2":

In [12]:
```
#print main characters for each part of "The Circuit"
print(data["main_chars"][78])
print(data["main_chars"][79])
```

```
Andy;Creed;Dwight;Jim;Kelly;Michael;Oscar;Pam;Phyllis;Stanley
Angela;Creed;Dwight;Jim;Kelly;Kevin;Meredith;Michael;Oscar;Pam;Phyllis;Stanley
```

We note that each part had a different set of main characters. However, in line with considering "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2" two parts of the same episode, we assign the union of their main_char entries to the corresponding entry in "Lecture Circuit (Parts 1&2)". By the Saleem data, we note that no guest stars appeared in "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2". It follows that we may set the GuestStars column of "Lecture Circuit (Parts 1&2)" to NaN. Lastly, in line with the way that other two-part episodes have been combined in the Saleem dataset, the duration of "Lecture Circuit (Parts 1&2)" is set as the sum of durations for the two parts. The cell below creates the new entry "Lecture Circuit (Parts 1&2)" as described above, while removing the entries corresponding to "Lecture Circuit: Part 1" and "Lecture Circuit: Part 2".

In [13]:
```
#get list of main characters from each part
main_chars_part1 = set(data["main_chars"][78].split(';'))
main_chars_part2 = set(data["main_chars"][79].split(';'))

#Create string with union of set of characters from each part
main_chars_union = main_chars_part1.union(main_chars_part2)
main_chars_combined = ';'.join(main_chars_union)

#Create list of features for "Lecture Circuit (Parts 1&2)" in main data
combined_features_data = [data["season"][78], data["episode"][78], "Lecture Circu
it (Parts 1&2)", data["director"][78], data["writer"][78], data["imdb_rating"][7
8], np.mean(data["total_votes"][78:80]), data["air_date"][78], np.mean(data["n_li
nes"][78:80]), np.mean(data["n_directions"][78:80]), np.mean(data["n_words"][78:8
0]), np.mean(data["n_speak_char"][78:80]), main_chars_combined]

#Create list of features for "Lecture Circuit (Parts 1&2)" in Saleem data
combined_features_saleem = ["Lecture Circuit (Parts 1&2)", e_col["Date"][78], e_c
ol["Duration"][78:80].sum(), e_col["GuestStars"][78]]

data.loc[78, list(data.columns)] = combined_features_data
data = data.drop([79], axis=0)
data=data.reset_index()
data = data.drop(["index"], axis = 1)

e_col.loc[78, list(e_col.columns)] = combined_features_saleem
e_col = e_col.drop([79], axis=0)
e_col=e_col.reset_index()
e_col = e_col.drop(["index"], axis = 1)
```

The resulting combined column, in each set of data, looks as so

In [14]:
```
display(data[data["episode_name"]=='Lecture Circuit (Parts 1&2)'].style.set_capti
on('Lecture Circuit (Parts 1&2) in Original data').set_table_styles(styles))
display(e_col[e_col["EpisodeTitle"]=='Lecture Circuit (Parts 1&2)'].style.set_cap
tion('Lecture Circuit (Parts 1&2) in Saleem data').set_table_styles(styles))
```

Out[14]:

Lecture Circuit (Parts 1&2) in Original data

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_directio
78	5	16	Lecture Circuit (Parts 1&2)	Ken Kwapis	Mindy Kaling	8.200000	1823	2009-02-05	244.500000	

Out[14]:

Lecture Circuit (Parts 1&2) in Saleem data

	EpisodeTitle	Date	Duration	GuestStars
78	Lecture Circuit (Parts 1&2)	5 February 2009	60	nan

We may now combine the two sets of data

In [15]:
```
#Combine data
e_col_ = e_col.drop(["Date", "EpisodeTitle"], axis = 1)
df = pd.concat([data.reset_index(drop=True), e_col_.reset_index(drop=True)], axis
=1)

df #This is the full dataframe containing the data from original dataset + column
s from the Saleem data found online ("GuestStars", "Duration")

#Describe data
#df.info(verbose=False)
df.describe().round(2).style.format('{:.2f}').set_caption('Descriptive Statistics
of Combined Dataframe').set_table_styles(styles)
```

Out[15]:

Descriptive Statistics of Combined Dataframe

	season	episode	imdb_rating	total_votes	n_lines	n_directions	n_words	n_speak_char	Duration
count	185.00	185.00	185.00	185.00	185.00	185.00	185.00	185.00	185.00
mean	5.46	12.45	8.25	2131.20	296.68	50.12	3055.95	20.72	27.49
std	2.40	7.25	0.54	792.61	82.13	24.00	800.50	5.09	7.91
min	1.00	1.00	6.70	1393.00	131.00	11.00	1098.00	12.00	21.00
25%	3.00	6.00	7.90	1627.00	256.00	34.00	2667.00	17.00	22.00
50%	6.00	12.00	8.20	1954.00	281.00	46.00	2876.00	20.00	23.00
75%	8.00	18.00	8.60	2388.00	315.00	60.00	3141.00	23.00	30.00
max	9.00	28.00	9.70	7934.00	625.00	166.00	6076.00	54.00	60.00

The description shows that, prior to any pre-processing, the data is highly heterogenous across features, varying by data type (dtype of float, int64 and object). In addition, some of the numerical continuous features tended to have a high variation (total_votes, n_lines, n_directions, n_words had high std values), which we identified would hinder model performance later on. We will thus have to appropriately scale our variables before model fitting.

2.2 Feature engineering:

2.2.1. Cleaning the writers/directors columns

We want to use the directors and writers as predictors in our model, however it was apparent from initial data exploration that there were inconsistent name spellings:

In [16]:
```
#We create a new dataframe df_ on which we will later do our feature engineering,
allowing backwards comparison to the original dataframe (df)  if necessary
df_ = df.copy()

print("An example of inconsistent spelling in the 'director' column of the update
d dataframe")
df[df['director'].str.contains('Greg')].iloc[1:3, 3:4]
```

```
An example of inconsistent spelling in the 'director' column of the updated data
frame
```

Out[16]:

	director
6	Greg Daniels
12	Greg Daneils

As we can see in the two rows printed above, in the column `director,` the name - Greg Daniels, is spelled in two different ways. This type of error happens for `writers` as well and for multiple other names. Therefore we decided to write a pipeline which would automatically compare very similar names and correct the misspelled ones. We used a package from the library `fuzzywuzzy,` which uses a Levenshtein distance to calculate distances between strings. We used **90% similarity threshold** between two strings as it is enough to detect even bigger typing errors, but not small enough to categorize two different names as the same.

When the algorithm comes across two very similar names it decides which one is correct by counting the occurrences of each option and choosing the more common one. Clearly, this approach cannot correct misspelled names if they appear only once or twice. For these cases the function will return one of the similar names arbitrarily.

Some episodes also contain multiple writers/directors, therefore we decided to explode these rows, but kept the old index in order to merge the rows back together later.

In [17]:
```
#Step 0. Explode the rows with multiple writers/directors, but save the old index
for when we want to merge the rows for each episode back together
df_['writer'] = df_['writer'].str.split(';')
df_['director'] = df_['director'].str.split(';')

df_exp = df_.explode('writer').explode('director')
df_exp = df_exp.rename_axis('old_index')
df_exp = df_exp.reset_index(drop=False)

#Step 1. Create lists of correct names by comparing them to each other and if som
e are very similar pick the ones that appear more often
full_writers = df_exp['writer']
full_directors = df_exp['director']

def is_similar(a, b):
    return fuzz.ratio(a, b) >= 90

def correct_names(names):
    name_count = Counter(names)
    max_len = max(len(name) for name in names)
    correct_names = []
    for name in names:
        similar_names = [other_name for other_name in name_count.keys() if other_
name != name and is_similar(name, other_name)] #if two names are not identical, b
ut are similar add them to variable.
        if len(similar_names) == 0: #if there is no similar names, append the nam
e to the list of correct names
            correct_names.append(name)
        else:
            counts = {other_name: name_count.get(other_name, 0) for other_name in
similar_names + [name]} #compare the occurences of each variant and append the mo
re common one.
            max_count = max(counts.values())
            if counts[name] == max_count:
                correct_names.append(name)
            else:
                correct_names.append(max(counts, key=counts.get))
    return correct_names

directors = correct_names(full_directors) #These are the lists containing only th
e correct names, as we got rid of the misspelled ones
writers = correct_names(full_writers)


#Step 2. Use the lists of only the correct names to fix the misspelled names in t
he dataframe.
for i, name in enumerate(df_exp['director']):
    if name not in set(directors):
        similar_names = [other_name for other_name in directors if fuzz.ratio(nam
e, other_name) >= 90]
        if len(similar_names) > 0:
            df_exp.at[i, 'director'] = max(similar_names, key=lambda x: fuzz.rati
o(name, x))

for i, name in enumerate(df_exp['writer']):
    if name not in set(writers):
        similar_names = [other_name for other_name in writers if fuzz.ratio(name,
other_name) >= 90]
        if len(similar_names) > 0:
            df_exp.at[i, 'writer'] = max(similar_names, key=lambda x: fuzz.ratio
(name, x))


# Step 3. Encoding writers and directos
writer_dummies = pd.get_dummies(df_exp['writer'], prefix='w')
director_dummies = pd.get_dummies(df_exp['director'], prefix='dir')
df_encoded = pd.concat([df_exp, writer_dummies, director_dummies], axis=1)

# Step 4. Use the old_index to merge the rows of the same episodes and return the
```

```
corrected dataframe df_
merged_df = df_encoded.groupby(df_encoded.old_index).agg('max')
df_ = merged_df.reset_index(drop=True)
```

2.2.2. `air_date` to `air_day` feature engineering

We decided to change the column containing the dates of the release of an episode into day of the week, which will be more informative if we are looking to see if the day of the week has an effect on the `imdb_rating`. We then introduced dummy variables for each day of the week on which an episode was released.

In [18]:
```
# Adressing air_date
from datetime import datetime
def date_to_day(date):
        date_obj = datetime.strptime(date, "%Y-%m-%d")
        day_of_week = date_obj.strftime("%A")
        return day_of_week

df_['air_day'] = df_['air_date'].apply(date_to_day)

one_hot = pd.get_dummies(df_['air_day'])
df_ = pd.concat([df_, one_hot], axis=1)
```

2.3 Encoding

2.3.1 Encoding Main Characters:

The main_chars column in the original dataset gives the names of all main characters that appeared in each episode. We are interested in the effect that selection of main characters has on the `imdb_rating` of an episode. To investigate this, we engineered one-hot-encoded features for each main character into the dataset. The string type feature of `main_chars`, which lists the main characters of each episode, was first exploded into several features (one for each main character) that were subsequently assigned one-hot encoding.

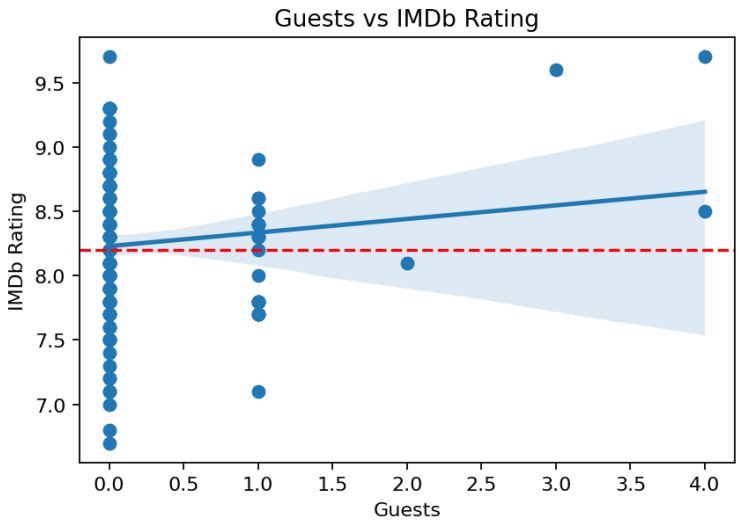
In [19]:
```
#Split main character strings and find unique characters
chars_split = df_['main_chars'].str.split(';')
all_chars = set(chars_split.explode().unique())

#Create one-hot encoding for each character
for char in all_chars:
    col_name = "mc_" + str(char)
    df_[col_name] = df_["main_chars"].str.contains(char, regex=False)
    df_[col_name] = df_[col_name].replace({True: 1, False: 0})
```

2.3.2 Encoding Guest Stars (`Guests`):

The Saleem data provided the names of guest stars appearing in each episode (if any), presenting an opportunity to investigate the effect of such appearances on ratings . On a few episodes there were more than one guest star present, and so we investigate if there is any correlation between the number of *guest stars* and the *imdb_rating.* We encoded the number of guest stars into a new temporary column *Guests* and plotted a graph of number of guest stars against the imdb_rating.

In [20]:
```
# Seeing if there is a correlation between no. of guest stars and imdb_rating
df_['Guests'] = df_['GuestStars'].apply(lambda x: 0 if pd.isnull(x) else x.count
(",") + 1)

# Create a smaller plot
fig, ax = plt.subplots(figsize=(6,4))

# Plot the scatter plot
ax.scatter(df_['Guests'], df_['imdb_rating'])


# Plot the horizontal line
ax.axhline(y=8.2, color='r', linestyle='--')

sns.regplot(x=df_['Guests'], y=df_['imdb_rating'], ax=ax)

# Set labels and title
ax.set_xlabel('Guests')
ax.set_ylabel('IMDb Rating')
ax.set_title('Guests vs IMDb Rating')

plt.show()
```

Out[20]:



As we can see in the figure above, there is a slight positive correlation between the number of guest stars and the imdb_rating. Unfortunately, there are only 4 data points for episodes with more than 1 guest star. There is no clear indication of whether having guest star increases the IMDB rating or not, given the small number of episodes in which guest stars were featured. We can infer from the scatter plot that the range of IMDB ratings for episodes with more than one guest star has a greater lower bound than the corresponding range for star-less episodes. In other words, episodes with one or more guest stars have ratings ranging from 7.2 to the highest rating of 9.8, while episodes not featuring a guest star have a larger spread, with IMDB ratings in the range [6.5, 9.8].

It is worth noting that, among the episodes featuring a guest star, the highest IMDB rating of 9.8 occurred for the "Finale," which is the final episode of the last season. Moreover, this episode has received 7934 votes, which is an outlier in the `total_votes` dimension, as Q_3 is at 2385. Therefore, we cannot necessarily attribute the high rating solely to the guest stars, and we should be cautious about placing too much weight on their presence.

We encoded the `GuestStars` column into *0s,* for episodes with no guest stars, and *1s,* for episodes with at least 1 guest star. We opted for a binary classification as there are few episodes with more than one guest and since we are not able to measure individual guest's fame (their individual impact/resonance they have on the viewer) i.e. different guests might have different effects on the episode rating.

In [21]:
```
df_['Guests'] = df_['GuestStars'].apply(lambda x: 0 if pd.isnull(x) else 1)
```

2.3.3 Encoding stand alone episodes vs two part episodes (`is_part`)

We suspected that the reason *duration* has such a strong positive correlation with the *imdb_rating* is because the longest episodes will probably be the ones that are a part of two-series episodes. To test that theory, we have created a new feature *is_part* which assigns 0s to independent episodes and 1s to episodes which are multi-part.

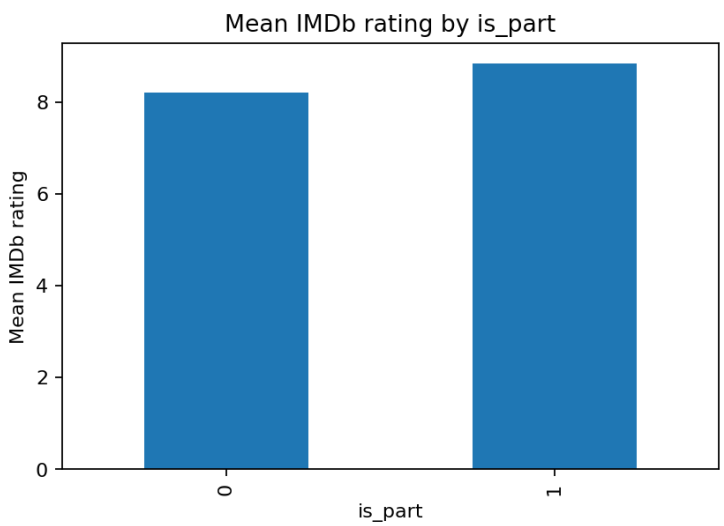
In [22]:
```
#Regex is causing a warning, which suggests to use str.extract instead of str.con
tains, but it is not what I needed so I muted the warning
import warnings
warnings.filterwarnings("ignore", message="This pattern is interpreted as a regul
ar expression, and has match groups.")
import re

#Create a new column for encoding part1&2 as a feature
df_['is_part'] = pd.Series([0] * len(df))

# This code is creating a column which contains binary information. Whether an ep
isode is a 2 parter (1), or a single individual episode (0)
pattern = r'\b(?<!party\s)(Parts?|Part \d+)\b' #Regular expression which includes
'Part' and 'Parts', but ignores 'party'.
for index, row in df_.iterrows():
    if re.search(pattern, row['episode_name']):
        df_.loc[index, 'is_part'] = 1
```

To check whether the stand alone episodes get rated lower on average than the episodes which consist of several parts we plotted a graph and calculated the % difference of the means. As we can see, the independent episodes on average have 7% lower rating than the multi-part episodes.

In [23]:
```
# group the data by 'is_part' and calculate the mean IMDb rating
grouped = df_.groupby('is_part')['imdb_rating'].mean()

percent_diff = abs(grouped.loc[0] - grouped.loc[1]) / ((grouped.loc[0] + grouped.
loc[1]) / 2) * 100
print(f"The independent episodes get on average {percent_diff:.2f}% lower ratin
g")

# create a bar plot of the mean IMDb ratings
plt.figure(figsize=(6,4))
grouped.plot(kind='bar')
plt.title('Mean IMDb rating by is_part')
plt.xlabel('is_part')
plt.ylabel('Mean IMDb rating')
plt.show()
```

```
The independent episodes get on average 7.43% lower rating
```

Out[23]:



2.4 Dropping (unnecessary) features:

We may now drop unnecessary features from the dataframe. Additional features are dropped as they caused issues during the model fitting stage downstream. Choosing which features to drop is a crucial part of any implementation of supervised learning and as such we will thoroughly justify our decisions during this step of the analysis. We motivate our discussion of rank deficiency issues by considering the dummy variables for the most commonly appearing main characters.

Below, we print the names and number of appearances for main characters that appeared in more than 180 of the 185 episodes in the dataset. Note that "Dwight" appears as a main character in every episode from the dataset, "Jim" appears as a main character in all but one episode, and "Pam" in all but four :

In [24]:
```
for name in list(df_.columns):
    if ("mc" in name and df_[name].sum() > 180):
        print(f"Number of episodes in which {name[3:]} appeared as a main charact
er: ", df_[name].sum())
print("Total nr of episodes: ", df_.shape[0])

display(df_[df_["mc_Jim"]==0].style.set_caption('Episode in which Jim does not ap
pear as main character').set_table_styles(styles))
```

```
Number of episodes in which Jim appeared as a main character:  184
Number of episodes in which Dwight appeared as a main character:  185
Number of episodes in which Pam appeared as a main character:  181
Total nr of episodes:  185
```

Out[24]:

Episode in which Jim does not appear as main character

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_direc
102	6	14	The Banker	Jeffrey Blitz	Jason Kessler	6.800000	2288	2010-01-21	131.000000	

In [25]:
```
display(df_[df_["mc_Pam"]==0].style.set_caption('Episodes in which Pam does not a
ppear as main character').set_table_styles(styles))
```

Out[25]:

Episodes in which Pam does not appear as main character

	season	episode	episode_name	director	writer	imdb_rating	total_votes	air_date	n_lines	n_d
146	8	9	Mrs. California	Charlie Grandy	Dan Greaney	7.700000	1553	2011-12-01	278.000000	
147	8	10	Christmas Wishes	Ed Helms	Mindy Kaling	8.000000	1547	2011-12-08	310.000000	
148	8	11	Trivia	B.J. Novak	Steve Hely	7.900000	1488	2012-01-12	237.000000	
149	8	12	Pool Party	Charles McDougall	Owen Ellickson	8.000000	1612	2012-01-19	273.000000	

In order to avoid rank-deficiency of our design matrix during model fitting, we see immediately that we must remove the column "mc_Dwight".

Consider then the entries in column "mc_Jim". Before model fitting a the dataset will be split into a training and a test set, respectively. During cross-validation, the training set will be sub-split into 5 folds, with one fold being assigned for validation. This process, of sub-splitting the training set, is repeated 5 times, such that each of the folds from the training set, acts a validation set exactly once.

See first that if "The Banker" is included in the test set, the training set will have all 1's in the "mc_Jim"-column. Consequently, the training set will be rank deficient when an intercept is included for model fitting. Conversely, assume now that "The Banker" is NOT assigned to the test set. During cross-validation, we see that "The Banker" will be assigned to the validation fold exactly once. In this case, the remaining 4 folds will have all 1's in the "mc_Jim"-column, again causing linear dependence with the intercept during model fitting (that is, model fitting within the cross-validation). It follows that we must drop the "mc-Jim" feature. The consequence is that we cannot quantify the effect of "Jim" appearing as a main character on imdb ratings. By our reasoning, this is justified, however, as the size of the subset within which "Jim" does NOT appear is too small to inform any discussion of the effect of his presence as a main character.

In a similar vein to the discussion for "mc_Jim", note that if all episodes in which "Pam" does not appear are assigned to the testing set, the remaining training set will be rank-deficient when an intercept column is included. Furthermore, note that if only one of these episodes are in the training set, rank-deficiency will occur during cross validation in the same way as was explained for "mc_Jim". Hence, when the data is split, we must ensure that in the training set at least two contain entries episodes in which "Pam" did not appear as a main character.

By the same reasoning with which we concluded that "mc_Jim" must be excluded from the dataset, we see that the dummy-variable for every writer and director appearing in only one episode must also be dropped from the dataset. (In this case there will be rank-deficiency due to a majority of 0's rather than 1s). We have chosen, however, to exclude the one-hot variables for every director and writer who contributed on less than 5 episodes. While less than ideal, this approach ensures we have enough data to assess individual writer/director effects. The reasoning is that the director/writer variables which are excluded would not contain enough non-zero entries for cross-validation to be effective in optimisation of model hyper-parameters and for the fitted models to have good predictive power.

The last step we must take to avoid linear dependence in our training data is to remove the one-hot variables for episodes released on a Sunday and on a Tuesday.

Below we have printed the number of episodes released on a Sunday, Tuesday, and Thursday, respectively

In [26]:
```
days=['Sunday','Tuesday','Thursday']
print(f"Number of episodes released on a specific day:")
for day in days:
    print(f'{day}: {df_[day].sum()}')
```

```
Number of episodes released on a specific day:
Sunday: 1
Tuesday: 15
Thursday: 169
```

Note first that at least one of the one-hot variables indicating weekday of release must be removed since their sum would be linearly dependent with the intercept column in the design matrix for the training data. It makes sense to drop the "Sunday" variable since cross validation would lead to rank-deficiency when the single non-zero entry ("Stress Relief") is included in the validation or test sets. Consider then the case where the "Sunday" variable is removed. If "Stress Relief" is assigned to the test set, the sum of columns "Tuesday" and "Thursday" will equal the intercept column, causing rank-deficiency for the remaining data. On the other hand, if "Stress Relief" is not in the test set, it will be assigned to the validation fold during cross-validation at least once. In the design matrix corresponding to the remaining collection of folds, the sum of "Tuesday" and "Thursday" columns will again equal the intercept, causing rank-deficiency. It follows that we must also eliminate either variable "Thursday" or variable "Tuesday". We choose to drop the "Tuesday" variable so as to allow a comparison between episodes released on a Thursday and episodes released on any other day (the vast majority of episodes were released on a Thursday).

As we shall see in the next section on fitting regression models, the result of keeping only director/writer variables with at least 5 non-zero entries is that the influence on rating of other directors/writers will be collectively included in an intercept term. Similarly, keeping only the "Thursday" dummy-variable for day of release causes the effect of other release-days to be contained in the intercept.

With regards to the other variables in our dataset, note that we have no way of encoding "episode_name" as a numerical variable, and so this column is also dropped. We encoded the writer and director above, so these variables may also be dropped now. Our work aims at finding the most important features to maximise ratings of a reunion episode, and so the "season" and "episode" variables are also tossed out. We also remove the 'total_votes' column, as we expect NBC to have little direct control over the number of votes cast. Finally, "GuestStars", "air_date", and "air_day" have all been encoded in one-hot or dummy variables, and so may be tossed out at this point.

The cell directly below effectuates all changes discussed above in our dataset.

In [27]:
```
# Select columns starting with 'w_' or 'dir_'
cols_to_sum = df_.filter(regex='^(w_|dir_)')

# Calculate the sum of each individual column
sum_cols = cols_to_sum.sum()

# Select columns where the sum is greater than or equal to 5
selected_cols = df_[sum_cols[sum_cols >= 5].index]

# Concatenate the selected columns with the other columns in the original datafra
me
df_ = pd.concat([df_.drop(cols_to_sum.columns, axis=1), selected_cols], axis=1)

#remove sunday variable in order to avaid colinearity with intercept term
drop_df = df_.drop(columns = ["Sunday", "Tuesday", "director", "writer", "main_ch
ars", "episode_name", "season", "episode", "GuestStars", 'air_date', 'air_day',
'total_votes', "mc_Dwight", "mc_Jim"])

#print number of columns and rank in remaining data (excluding the response) when
concatenated with an intercept column
check_rank = np.c_[np.ones(185), np.array(drop_df.drop("imdb_rating", axis = 1))]
print('Nr of Columns: ', check_rank.shape[1])
print('Rank: ', np.linalg.matrix_rank(check_rank))
```

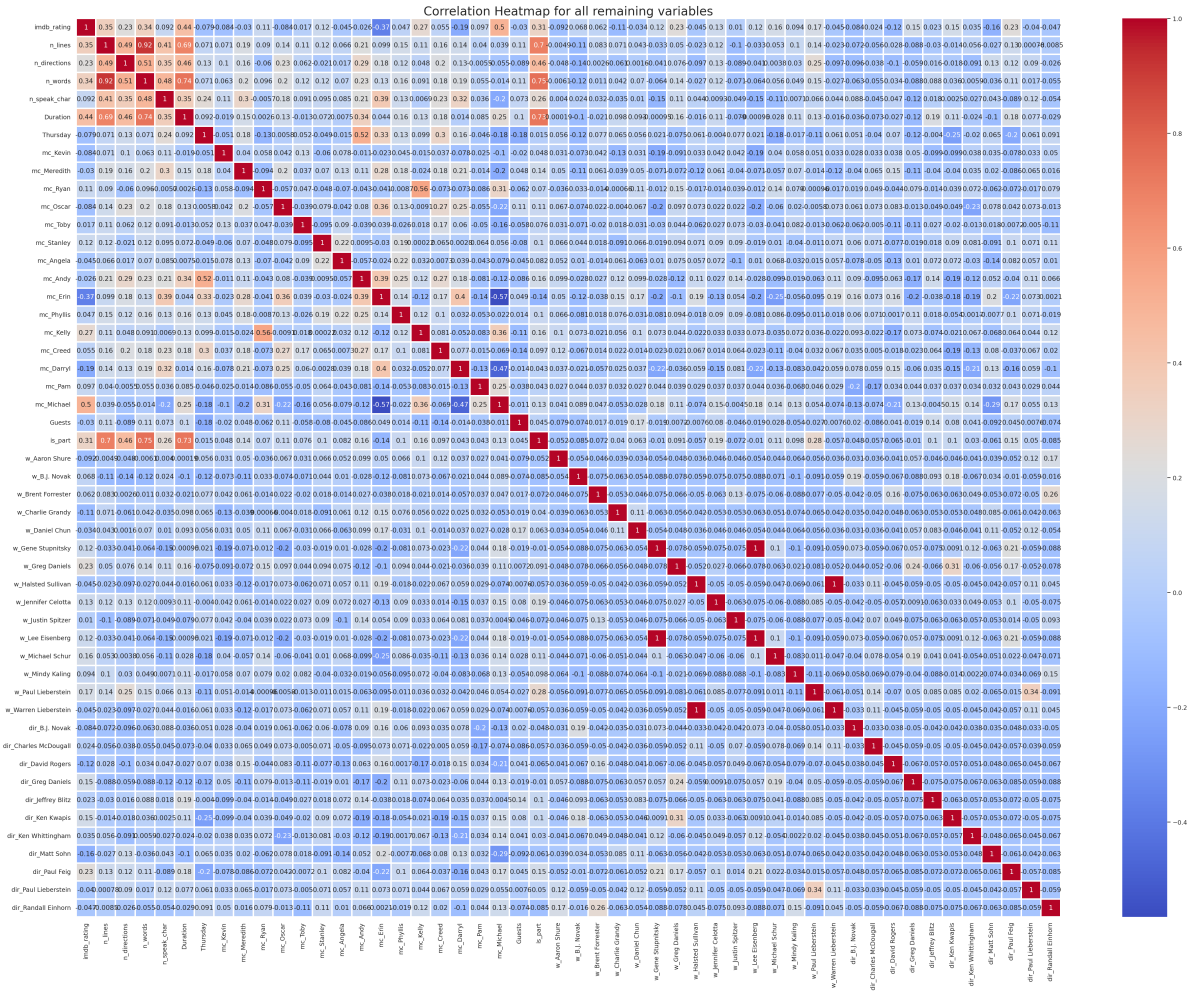
```
Nr of Columns:  50
Rank:  48
```

2.5 Feature Correlation

We have reduced the number of columns in the data to 50 (when including an intercept column of all 1's\ and exluding the imdb_rating). However, the rank is only 48. Below, we make a correlation plot to investigate any further interactions among the remaining variables.

In [28]:
```
# Produce a Heat Map
explore_set = drop_df.copy()
sns.set(rc={'figure.figsize': (40, 30)})
sns.heatmap(explore_set.corr(),cmap='coolwarm', annot = True, linewidths = 2)
plt.title("Correlation Heatmap for all remaining variables", size = 22)
plt.show()
```

Out[28]:



We note that the entries in "w_Lee Eisenberg" are perfectly correlated with the entries in "w_Gene Stupnitsky". Similarly, the entries in "w_Warren Lieberstein" are perfectly correlated with the entries in "w_Halsted Sullivan". This suggests that each of the writers in these two pairs always worked with the other writer in the same pair. To avoid linear dependence we must combine the columns for each pair into a single variable, respectively.
These changes are enacted below.

In [29]:
```
drop_df = drop_df.rename(columns = {"w_Lee Eisenberg": "w_ Eisenberg & Stupnitsk
y", "w_Warren Lieberstein": "w_ Lieberstein & Sullivan"})
drop_df = drop_df.drop(["w_Gene Stupnitsky", "w_Halsted Sullivan"], axis=1)
```

3. Model Fitting and Tuning

3.1 Splitting data

We split our data, as is good practice when developing an ML model. Splitting data is integral to interpreting the information on which the model is built and tested.

We pass an integer (42) to `random_state` for reproducible output across executions of the script. We also pass a parameter to specify the size of the test data, which we set as 20% of the whole dataset, in line with industry standard. Such splitting provides 37 observations in the test set, so that 5-fold cross-validation may be performed on the remaining data, with either 29 or 30 observations in each fold, respectively. Before fitting each model, we applied min-max scaling to the explanatory variables. There were several reasons why we made this choice of scaling. Firstly, min-max scaling reduced the range of every variable to the interval [0,1], making it easier to compare the varying coefficients for different variables. Second, min-max scaling leaves binary (dummy) variables unchanged, allowing for very simple interpretation of the resulting coefficients.

We considered using several regression models for our analysis, including lasso regression, ridge regression, polynomial regression, and decision tree regression. We have created some functions in order to standardise the process of calculating the RMSE, MSE and R^2 values, and of getting an array of coefficients for the regression models, including the intercept term if available. These functions were partially drawn from the workshop material and are shown below.

In [30]:
```
#Split data

X = drop_df.drop("imdb_rating", axis = 1) # Set of features
y = np.array(drop_df["imdb_rating"]) #response

#Split Data into test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42, test
_size=0.2, shuffle = True)

#Setting a fixed variable for KFold so that cross validation is standardised acro
ss model evaluations
kf=KFold(5)
```

In [31]:
```
#Use part of model_fit function from workshop to extract information on goodness
of fit
def model_fit(m, X, y):
    """Returns the mean squared error, root mean squared error and R^2 value of a
fitted model based
    on provided X and y values."""

    y_hat = m.predict(X)
    MSE = mean_squared_error(y, y_hat)
    RMSE = np.sqrt(mean_squared_error(y, y_hat))
    Rsqr = r2_score(y, y_hat)

    Metrics = (round(MSE, 4), round(RMSE, 4), round(Rsqr, 4))

    return Metrics
```

In [32]:
```
#Use function from workshops to extract coefficients from models
def get_coefs(m):
    """Returns the model coefficients from a Scikit-learn model object as an arra
y,
    includes the intercept if available.
    """

    # If pipeline, use the last step as the model
    if (isinstance(m, sklearn.pipeline.Pipeline)):
        m = m.steps[-1][1]

    if m.intercept_ is None:
        return m.coef_

    return np.concatenate([[m.intercept_], m.coef_])
```

To optimise the hyperparameters for the lasso, ridge, and polynomial regression, a 5-fold cross-validation method was applied to the training set for each model. During cross-validation for each model, we split the training data into 5 folds (split in the same way for each model). We mentioned above that when each fold acts as a validation set once, we must ensure that the design matrix as constructed from the remaining four folds is of full rank. The code below confirms that our training data does indeed maintain full rank during cross-validation.

In [33]:
```
print("The number of columns in the training data when combined with an intercep
t:", X_train.shape[1]+1)

for i, (train_index, test_index) in enumerate(kf.split(X_train)):
    validation_length = len(train_index)
    remainder = np.c_[np.ones(validation_length), np.array(X_train)[train_index]]
    print(f"When fold {i} acts as validation set, remaining trainng data (includi
ng intercept column) has rank", np.linalg.matrix_rank(remainder))
```

```
The number of columns in the training data when combined with an intercept: 48
When fold 0 acts as validation set, remaining trainng data (including intercept
column) has rank 48
When fold 1 acts as validation set, remaining trainng data (including intercept
column) has rank 48
When fold 2 acts as validation set, remaining trainng data (including intercept
column) has rank 48
When fold 3 acts as validation set, remaining trainng data (including intercept
column) has rank 48
When fold 4 acts as validation set, remaining trainng data (including intercept
column) has rank 48
```

After fitting each model, the RMSE when predicting for the test set was used as the final measure of performance.

In testing each of the four models mentioned above, we followed a fixed set of steps: hyperparameter optimisation, fitting, RMSE for test For example, in Terms of Lasso regression model we:

1. Performed optimisation of the hyperparameters (α) using 5-fold cross-validation on the training data
2. Train the Lasso regression model on the entirety of the training data using the optimal hyperparameter found in (1)
3. Measure the final performance of the model by finding RMSE with respect to test set

Recall that we dropped the one-hot variables for: "Dwight" as a main character, an episode airing on a Tuesday or Sunday, directors and writers who worked on less than 5 episodes. It follows that these variables form a base for each model in the sense that the intercept gives information about the predicted rating when "Dwight" is included as a main character, the episode airs on a Tuesday or Sunday, and the episode is written by a team of less experienced directors and writers.

For each model, we chose the root mean squared error, relative to a prediction on the test set, as a measure of performance.

We found that:

Lasso regression: For an optimal hyperparameter, α, of 0.01535; $RMSE = 0.3607$

Ridge regression: For an optimal hyperparameter, α, of 5.91466; $RMSE = 0.3709$

Polynomial regression: For an optimal degree of 1 (in each variable, equivalent to a linear model); $RMSE = 0.4516$

Decision tree: For optimal maximum depth of 2, and minimum observations at each leaf 8; $RMSE = 0.3855$

The model with the lowest root mean squared error was the Lasso Regression model. The code implementation for Lasso, including hyperparameter determination via cross-validation, is included below.

3.3 Lasso model

In [34]:
```
#Array of possible hyperparameters alpha
alphas = np.logspace(-3, 0, num=1000)

#perform gridsearch for optimal alpha on training data
gs = GridSearchCV(
    make_pipeline(
        MinMaxScaler(),
        Lasso()
    ),
    param_grid = {'lasso__alpha': alphas},
    cv = kf,
    scoring = "neg_root_mean_squared_error"
)

gs_lasso = gs.fit(X_train,y_train)
```

In [35]:
```
#print optimal hyperparameter
print(np.round(gs_lasso.best_params_['lasso__alpha'], 5))

#print corresponding best RMSE from cross-validation
print(abs(gs_lasso.best_score_))
```

```
0.01535
0.4685144992953799
```

In [36]:
```
#Make pipeline to fit model given hyperparameter
lasso_full = make_pipeline(
       MinMaxScaler(),
       Lasso(alpha = 0.01535)
    )

#Fit the model on the entire set of training data
full_lasso = lasso_full.fit(X_train,y_train)

#print MSE, RMSE, and R2 values when model is evaluated against test set
metrics = model_fit(full_lasso, X_test, y_test)
print("MSE: ", metrics[0], ", RMSE: ", metrics[1], ", R^2: ", metrics[2])
print()

model_coefs = get_coefs(full_lasso)

coef_dict ={}
coef_dict["Intercept"] = model_coefs[0]
for i in range(1, X.shape[1]):
    if model_coefs[i] != 0:
        coef_dict[X.columns[i-1]] = model_coefs[i]

print("Lasso Model: Nonzero coefficients and corresponding variables:")
print(coef_dict)
```

```
MSE:  0.1301 , RMSE:  0.3607 , R^2:  0.4426

Lasso Model: Nonzero coefficients and corresponding variables:
{'Intercept': 7.901377534167155, 'n_lines': 0.08051401219899766, 'Duration': 0.5
751568617477143, 'mc_Toby': 0.022320035222264743, 'mc_Erin': -0.1503855534464696
2, 'mc_Kelly': 0.08999416759956178, 'mc_Creed': 0.003017431447907244, 'mc_Michae
l': 0.30391438427028084, 'w_Greg Daniels': 0.09956655662093768, 'w_Paul Lieberst
ein': 0.10588468261369222, 'dir_Paul Feig': 0.018292388599823295}
```

We note that for several of the variables, the coefficients have been suppressed to zero by the Lasso model. The features that had non-zero coefficients, i.e. the most significant effect on `imbd_rating` , are listed below:

- Duration
- mc_Michael
- mc_Erin
- w_Paul Lieberstein
- w_Greg Daniels
- mc_Kelly
- n_lines
- mc_Toby
- dir_Paul Feig
- mc_Creed

In **2.5 Feature Correlation**, `duration`, `is_part` and `n_lines` were strongly correlated in the heatmap, which could contribute to fact that the lasso model supressed their coefficients entirely here. Logically, the `duration` parameter and `n_lines` features are intrinsically linked to an episode's status as a multi-part episode, so the model is unable to determine that part status is not important, as the suppression might be purely because of said correlation.

It would be preferable to split the part 1&2 episodes instead of merging them, as this would allow us to decouple the effects of the `duration` and `is_part` features. However, in the original dataset, the majority of the two-part episodes were collapsed into a single entry, respectively. For these two-part episodes, we would not be able to determine the features of parts 1 and 2 separately.

4. Discussion and Conclusions

Model implications, performance and drawbacks

To reiterate, the optimal model was a Lasso regression with optimal hyperparameter, α, of 0.01535. When applied to an unseen test set, this model's performance (root mean squared error) was measured as $RMSE = 0.3607$.

As we can observe, the greatest coefficient in the Lasso model corresponds to the `duration` of an episode. As outline below, however, we must be careful in making statements about the relative $importance$ of variabled based on the magnitude of their coefficients. We note that characters such as Michael, Kelly, Toby, and Creed have a positive impact on the rating, while Erin has a negative impact. Similarly, the number of lines also appears to play a positive role in determining the rating of an episode. Among the writers, we expect Paul Lieberstein and Greg Daniels to contribute positively to the `imdb_rating` of an episode relative to the writers who contributed on fewer than 5 episodes. In the same manner, Paul Feig directing an episode si expected to slightly increase the rating relative to one of the directors who worked on less than 5 episodes.

Writer and director caveats

The individual effects of numerous directors and writers were ultimately indiscernible in our final model. Several factors contributed to this:

- Due to their perfect correlation shown in **2.5 Feature Correlation**, we cannot ascertain the effects of the writers Lee Eisenberg, Gene Stupnitsky, Warren Lieberstein or Halsted Sullivan as they always worked in pairs, making their individual influence on IMD rating impossible to determine in any of our models. This may be a point of further investigation in subsequent analyses.
- As discussed in **3. Model Fitting and Tuning** several directors and writers who worked on few episodes, would have created rank deficiency in our statistical modelling, weakening the ability to make predictions during the model cross-validation stages. Their collective effect had to be included in the intercept instead, so their individual effect could not be calculated.
    - In fact, some of these "less experienced" writers/directors might have had good performance and improved `imdb_rating.` However, we are unable to individually discern any such effect given low number of episodes on which they contributed.

Conversely, of our recommended directors, director Paul Feig completed three episodes with Greg Daniels as the writer and these episodes have received high `imdb_scores` of 9.3, 8.2, and 9.7. This reinforces the idea that they work well together to deliver the best possible episode. Although both writers are also directors, they seem to perform better when focusing solely on writing or directing, respectively.

Our model's implementation of minmax scaling is a key component of managing the dataset's heterogeneity, however it's interpretation should be handled carefully. Whilst we can state that `duration` is highly influential, its stronger coefficient only shows the change in `imdb_rating` that we would expect if `duration` were increased by an entire increment of its range found in the training data (i.e. about 39 minutes as per the cell below). Extrapolations regarding imdb_rating based on such an extreme increase in duration are likely to be unrealistic, however. A better interpretation of the coefficient on duration is that an increase in the duration of an episode by 1 minute should lead to a $0.015 (\approx 0.575/39)$ increase in imdb ratings, based on the trainin data. Care should also be taken to avoid comparisons between minmax-scaled continuous features such as `duration` and similarly scaled binary features such as `mc_michael.` Note that the binary dummy-variables are invariant to the min-max scaling. Hence, for the dummy-features, we may indeed interpret the coefficient as a reflection of the expected increment to imdb_rating if the respective variable is activated (takes a value a of 1). For example, we would expect Michael's appearance as a main character in an episode to boost rankings by about ≈ 0.3 ranking points compared to the scenario in which he does not appear.

Recommendations

Based on the final Lasso regression model, we can anticipate that a good imdb_rating score is more likely to occur when Michael is included as a main character. To achieve an even higher score, we advise excluding Erin as a character and incorporating as many spoken lines as possible into the episode. Additionally, utilizing Paul Lieberstein or Greg Daniels as the writer and Paul Fieg as the director would yield the best results. Moreover, maximising episode duration and the number

of lines, as well as including Kelly and Toby in the episode would be beneficial. In the case that not all the corresponding staff can be employed for the reunion (i.e. if actors or writers not working anymore), we recommend asking them in order of decreasing coefficient (given that previous is unavailable).

Though we did not include Dwight and Jim in our features, their appearance in all or most of the episodes allows us to logically assume that viewers would expect them to be part of the reunion as part of the show's identity. Furthermore, we reasoned that there is no mathematical basis to exclude them as there was no, or too little data showing the influence of their absence on an episode's rating. Hence, even without relying on the model, it is safe to say that they should be included in the reunion episode of 'The Office'.

In [37]:
```
print(f"Min-max range of duration in training data: ", X_train["Duration"].max()-
X_train["Duration"].min())
```

```
Min-max range of duration in training data:  39
```

5. References

(1) Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in python. Journal of machine learning research 2011 Oct 1,.

(2) Linblad B. schrutepy version 0.1.3. 2022 Jan 23,;0.1.3.

(3) Cohen A. fuzzywuzzy version 0.18.0. 2020 Feb 13,;0.18.0.

(4) Saleem MM. Exploring Interesting Information From The Office Dataset. 2021; Available at: https://www.datainsightonline.com/post/exploring%5C-interesting%5C-information%5C-from%5C-the%5C-office%5C-dataset (https://www.datainsightonline.com/post/exploring%5C-interesting%5C-information%5C-from%5C-the%5C-office%5C-dataset). Accessed Feb 25, 2023.

6.Appendix

We applied the following overall process for the other models which were tested:

- Minmax scale the explanatory data
- Cross validate the fitted model for optimal hyperparameter
- Fit the model to entirety of the training data
- Test the efficiency of the model's predictive power against test data.

The code for these models and their associated MSE, RMSE, and R^2 scores are listed below.

Ridge Model

In [38]:
```
#Possible hyperparameters
alphas = np.logspace(-3, 4, num=400)

#Grid search cross-validation on training data for optimal hyperparameter determi
nation
gs = GridSearchCV(
    make_pipeline(
        MinMaxScaler(),
        Ridge()
    ),
    param_grid = {'ridge__alpha': alphas},
    cv = kf,
    scoring = "neg_root_mean_squared_error"
)

gs_ridge = gs.fit(X_train,y_train)

#Print optimal hyperparameter alpha and best RMSE score from cross-validation
print("Optimal paramter", np.round(gs_ridge.best_params_['ridge__alpha'], 5))
print("Best mean RMSE score for this value of alpha: ", abs(gs_ridge.best_score
_))
```

```
Optimal paramter 5.91466
Best mean RMSE score for this value of alpha:  0.4573758902482
```

In [39]:
```
#Make pipeline for ridge regression with minmax scaling and alpha as above
ridge_full = make_pipeline(
       MinMaxScaler(),
       Ridge(alpha = 5.91466)
    )

#Fit the model
full_ridge = ridge_full.fit(X_train,y_train)

#print metrics when sed to predict for test set
metrics = model_fit(full_ridge, X_test, y_test)
print("Ridge regression goodness of fit metrics when pplied to test set:")
print("MSE: ", metrics[0], ", RMSE: ", metrics[1], ", R^2: ", metrics[2])
```

```
Ridge regression goodness of fit metrics when pplied to test set:
MSE:  0.1376 , RMSE:  0.3709 , R^2:  0.4107
```

Decision Tree Regression

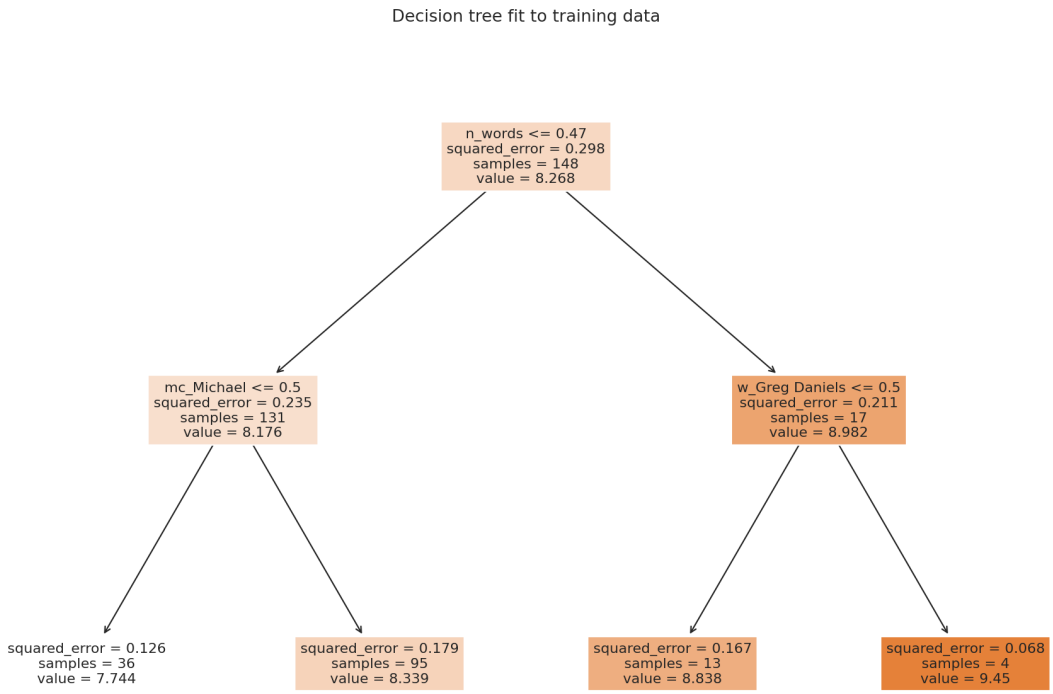
In [40]:
```
#Scale training data
X_train_minmaxscaled = MinMaxScaler().fit_transform(X_train)
X_test_minmaxscaled = MinMaxScaler().fit_transform(X_test)

#Perform grid search to determine optimal depth of tree and minimum nr of samples
that must be present at each leaf (tip of each branch)
g_cv = GridSearchCV(DecisionTreeRegressor(random_state = 10),
                param_grid={'min_samples_split': range(2, 40), 'max_depth': range
(2, 10), "min_samples_leaf":range(3,10)},
                scoring="neg_root_mean_squared_error", cv=kf, refit=True)

g_cv.fit(X_train_minmaxscaled, y_train)

#print best parameters
print(g_cv.best_params_)
```

```
{'max_depth': 2, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

In [41]:
```
#Create decision tree function for given best
dtr2 = DecisionTreeRegressor(max_depth = 2, min_samples_split = 8, min_samples_le
af= 2)

#fir the model
tree_fit = dtr2.fit(X_train_minmaxscaled, y_train)

#make prediction for test data
y_hat = tree_fit.predict(X_test_minmaxscaled)

#calculate root mean squared error
print("RMSE for decision tree model: ", np.sqrt(mean_squared_error(y_hat, y_tes
t)))

#Plot the tree
fig, ax = plt.subplots(figsize=(15, 10))
plot_tree(tree_fit, ax=ax, feature_names=X_train.columns, filled=True, fontsize=1
0)
plt.title("Decision tree fit to training data")
plt.show()
```

```
RMSE for decision tree model:  0.35788045283876874
```

Out[41]:



Polynomial model

In [42]:
```
#Adapted From workshop 5
#Building and cross validating a polynomial model

#Make pipeline
poly_pipe = make_pipeline(MinMaxScaler(),PolynomialFeatures(),LinearRegression())

parameters = {'polynomialfeatures__degree': np.arange(1,4,1)} #test polynomials f
rom degree 0 to 3

#Apply cross-validation to determine best degree of polynimial
Poly_grid_search = GridSearchCV(poly_pipe, parameters, cv = kf, scoring = "neg_ro
ot_mean_squared_error").fit(X_train, y_train)

#print best value for the degree and mean RMSE for that value
print(f"best param: {Poly_grid_search.best_params_}\nbest cross val score: {Poly_
grid_search.best_score_}")

# extract the cv_results dictionary
cv_results = Poly_grid_search.cv_results_; display(pd.DataFrame(cv_results))

#Setting degree to 1 and fitting
poly_pipe.set_params(polynomialfeatures__degree=1)
final_poly = poly_pipe.fit(X_train,y_train)
model_coefs=get_coefs(final_poly)
print("RMSE of fitted linear regression model of degree 1 when compared to test:
" , model_fit(final_poly, X_test, y_test)[1])
```

```
best param: {'polynomialfeatures__degree': 1}
best cross val score: -0.5251183715175356
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_polynomialfeatures__degree	
0	0.006864	0.001431	0.002349	0.000388	1	{'polynomia
1	0.213870	0.116052	0.058137	0.036235	2	{'polynomia
2	2.029933	0.577899	0.084291	0.025919	3	{'polynomia

```
RMSE of fitted linear regression model of degree 1 when compared to test:  0.451
6
```