Project 1.3

March 17, 2023

1 Machine Learning in Python - Group Project 1

Due Friday, March 10th by 16.00 pm.

Em Belanger, Alex Chen, Richelle Lee Wirharjanto, Robin Lin

1.1 General Setup

[1]:
```
%matplotlib inline

# Data libraries
import numpy as np
import pandas as pd

# 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 modules that are necessary
import sklearn

# For sentiment analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# ML processing libraries
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LassoCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Q-Q plot
from statsmodels.api import qqplot
```

[2]:
```python
# Load data
data = pd.read_csv("the_office.csv")
# Dialogue data for sentiment analysis
lines = pd.read_csv("The-Office-Lines.csv")
```

1.2 1. Introduction

In the project, we will investigate what factors are important for a high IMDB rating episode of the TV show "The Office". Using the insights we found we will shed light on how to produce the highest rated reunion episode. In addition to the historical performance of each episode, we utilised the dialogue in every episode to measure the activeness of characters and to conduct sentiment analysis, because we assumed that both measurements in an episode have an influence on the rating. To deliver a reasonable explanation of the model results, we chose Lasso linear regression as it gave us a model with strong interpretability. Applying Lasso linear regression can help select influential and important factors for the rating. To make the results robust, we conducted cross validation to select the best parameters and understand the model performance. In the end, we interpreted the coefficients of the model to find out which covariates can improve the ratings.

1.3 2. Exploratory Data Analysis and Feature Engineering

1.3.1 2.1 Data Quality Check

Before performing data visualisation, it is good to check whether there is missing data or not.

[3]:
```python
# Have a look at the dataset
data.head()
```

[3]:
```
   season  episode   episode_name          director  \
0       1        1          Pilot        Ken Kwapis
1       1        2  Diversity Day        Ken Kwapis
2       1        3    Health Care  Ken Whittingham
3       1        4   The Alliance      Bryan Gordon
4       1        5     Basketball      Greg Daniels

                                      writer  imdb_rating  total_votes  \
0  Ricky Gervais;Stephen Merchant;Greg Daniels          7.6         3706
1                                B.J. Novak          8.3         3566
2                           Paul Lieberstein          7.9         2983
3                              Michael Schur          8.1         2886
4                               Greg Daniels          8.4         3179

     air_date  n_lines  n_directions  n_words  n_speak_char  \
0  2005-03-24      229            27     2757            15
1  2005-03-29      203            20     2808            12
2  2005-04-05      244            21     2769            13
3  2005-04-12      243            24     2939            14
4  2005-04-19      230            49     2437            18
```

```
                                           main_chars
0  Angela;Dwight;Jim;Kevin;Michael;Oscar;Pam;Phyl…
1  Angela;Dwight;Jim;Kelly;Kevin;Michael;Oscar;Pa…
2  Angela;Dwight;Jim;Kevin;Meredith;Michael;Oscar…
3  Angela;Dwight;Jim;Kevin;Meredith;Michael;Oscar…
4  Angela;Darryl;Dwight;Jim;Kevin;Michael;Oscar;P…
```

[4]:
```
data.isna().any()
```

[4]:
```
season          False
episode         False
episode_name    False
director        False
writer          False
imdb_rating     False
total_votes     False
air_date        False
n_lines         False
n_directions    False
n_words         False
n_speak_char    False
main_chars      False
dtype: bool
```

From the summary above, there is no N/A value in the data.

[5]:
```
data.shape
```

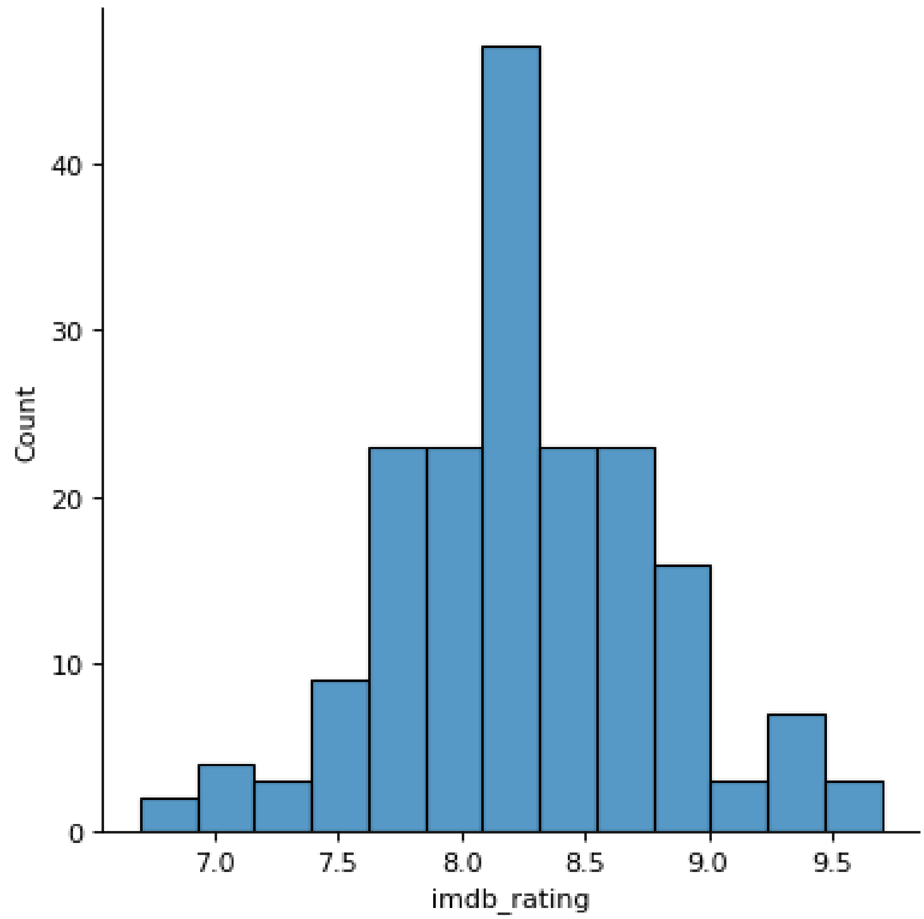
[5]:
```
(186, 13)
```

There are only 186 rows, indicating it's a small dataset. Keeping our model simple to keep from overfitting is necessary.

1.3.2 2.2 Preliminary Visualisation

[6]:
```
sns.displot(data.imdb_rating)
```

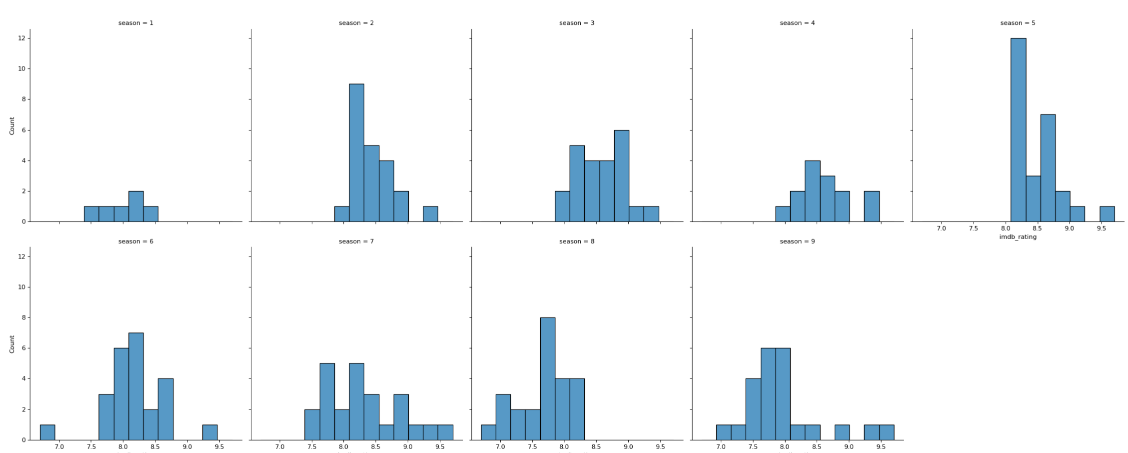
[6]:
```
<seaborn.axisgrid.FacetGrid at 0x11a4a5130>
```



From the plot above, the distribution of IMDB rating looks like a normal distribution.

[7]:
```
sns.displot(data=data, x='imdb_rating', col='season', col_wrap=5)
```

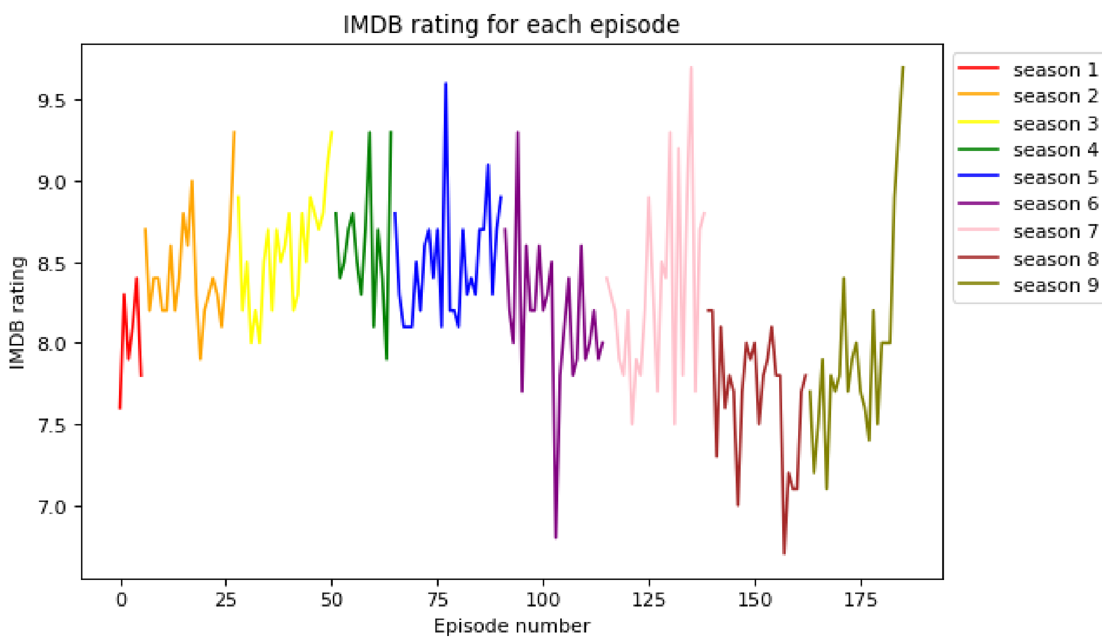
[7]: <seaborn.axisgrid.FacetGrid at 0x169028b50>



Based on the above plots, overall, the audience liked seasons 2, 5, and 7 because the distributions are shifted to right, and didn't like season 8 as well as the other seasons as the distribution has higher density on the left side.

[8]:
```
season_color = ['red', 'orange', 'yellow', 'green', 'blue', 'purple', 'pink',␣
 ↪'brown', 'olive']

for i in range(1,10):
    plt.plot(data[data.season == i].index,
             data[data.season == i].imdb_rating,
             c=season_color[i-1], label=f'season {i}')
    plt.legend(bbox_to_anchor=(1, 1))

plt.title('IMDB rating for each episode')
plt.ylabel('IMDB rating')
plt.xlabel('Episode number')
```

[8]: Text(0.5, 0, 'Episode number')

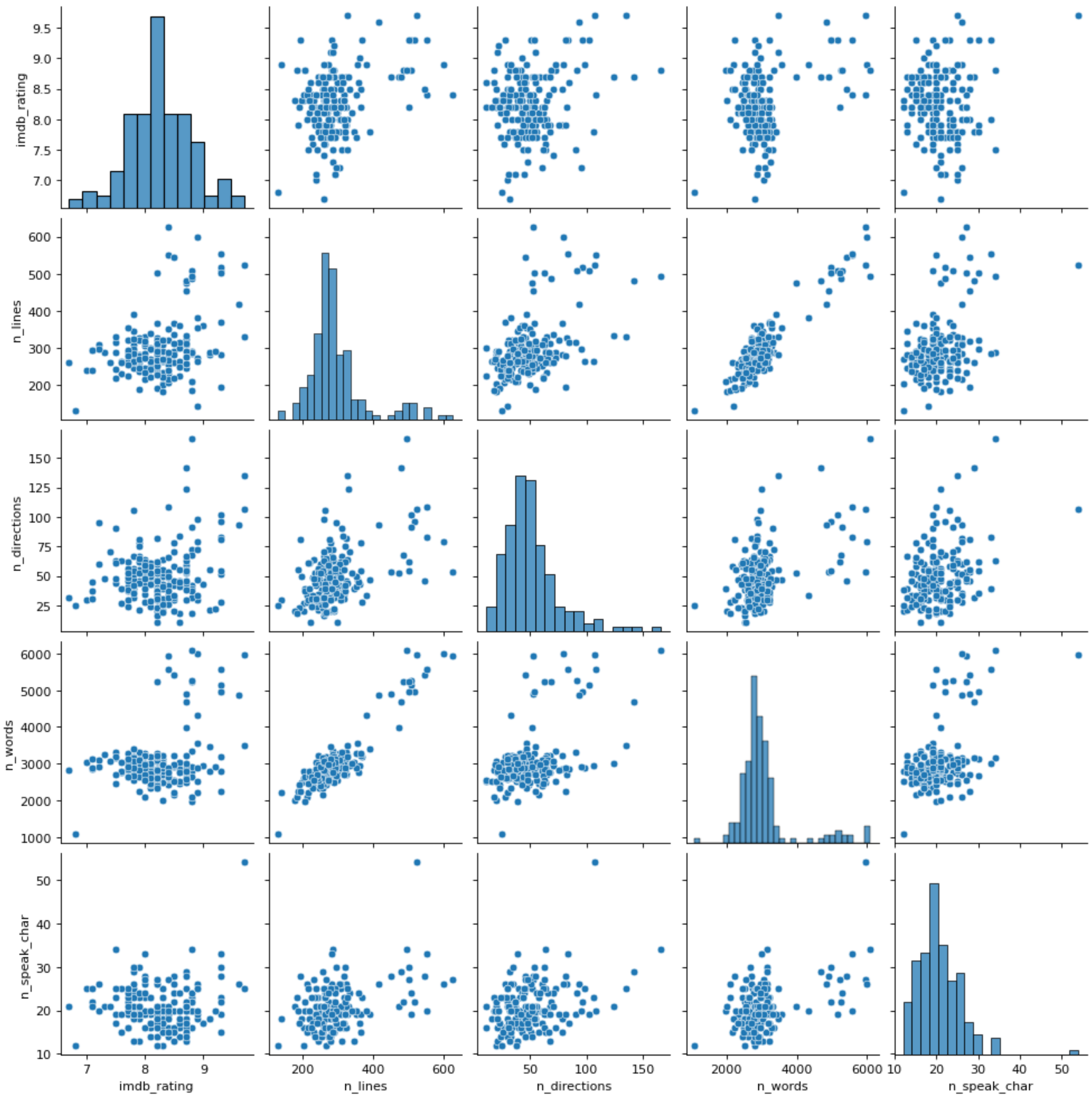


Based on the plot above, it could be identified that there is one episode from season 6 with low rating, and that on average, the ratings from season 8 are low. As for the high ratings, the graph shows that season 5 episodes are generally ranked in the higher range, and season 7 and the finale, while some of the episodes in these seasons are rated lower than any of that in season 5, has episodes which are ranked higher than that of any in season 5.

The graph made it visually easier to interpret which episodes within which season are performing better.

[9]:
```
sns.pairplot(data=data.loc[:
 ↪,['imdb_rating','n_lines','n_directions','n_words','n_speak_char']])
```

[9]:
```
<seaborn.axisgrid.PairGrid at 0x1695c5670>
```



From the pairplot, it seems that the number of lines, the number of lines containing stage directions, the number of words in an episode, and the number of speaking characters have a positive correlation with IMDB ratings. Moreover, there is a high positive correlation between the number of lines and the number of words in an episode. This is intuitive as with higher number of lines, a higher number of words is expected. A positive correlation could also be seen between number of directions and number of speaking characters, number of directions and number of words, number of directions and number of lines, and number of lines and number of speaking characters.

With these positively correlated variables, it is necessary to keep only one of them in the model to avoid redundancy.

1.3.3 2.3 Variable Tidying

There are some columns containing categorical data. Before conducting further analysis, it is essential to perform data preprocessing techniques on them.

[10]:
```
data.columns
```

[10]:
```
Index(['season', 'episode', 'episode_name', 'director', 'writer',
       'imdb_rating', 'total_votes', 'air_date', 'n_lines', 'n_directions',
       'n_words', 'n_speak_char', 'main_chars'],
      dtype='object')
```

`director`, `writer`, `main_chars` are useful categorical data, hence we worked on these variables.

director

[11]:
```
np.sort(data.director.unique())
```

[11]:
```
array(['Alex Hardcastle', 'Amy Heckerling', 'Asaad Kelada', 'B.J. Novak',
       'Brent Forrester', 'Brian Baumgartner', 'Bryan Cranston',
       'Bryan Gordon', 'Charles McDougal', 'Charles McDougall',
       'Charlie Grandy', 'Claire Scanlon', 'Claire Scanlong',
       'Craig Zisk', 'Daniel Chun', 'Danny Leiner', 'David Rogers',
       'Dean Holland', 'Dennie Gordon', 'Ed Helms', 'Eric Appel',
       'Gene Stupnitsky;Lee Eisenberg', 'Greg Daneils', 'Greg Daniels',
       'Harold Ramis', 'J.J. Abrams', 'Jason Reitman', 'Jeffrey Blitz',
       'Jennifer Celotta', 'Jesse Peretz', 'John Krasinski', 'John Scott',
       'Jon Favreau', 'Joss Whedon', 'Julian Farino',
       'Kelly Cantley-Kashima', 'Ken Kwapis', 'Ken Whittingham',
       'Ken Wittingham', 'Lee Eisenberg;Gene Stupnitsky', 'Lee Kirk',
       'Marc Webb', 'Matt Sohn', 'Michael Spiller', 'Miguel Arteta',
       'Mindy Kaling', 'Paul Feig', 'Paul Lieberstein', 'Paul Lieerstein',
       'Rainn Wilson', 'Randall Einhorn', 'Reginald Hudlin',
       'Rodman Flender', 'Roger Nygard', 'Seth Gordon',
       'Seth Gordon;Harold Ramis', 'Stephen Merchant', 'Steve Carell',
       'Troy Miller', 'Tucker Gates', 'Victor Nelli Jr.'], dtype=object)
```

From the results above, there are some misspellings, (the first is the correct one)

‘Charles McDougall’=‘Charles McDougal’

‘Claire Scanlon’=‘Claire Scanlong’

‘Greg Daniels’=‘Greg Daneils’

‘Ken Whittingham’=‘Ken Wittingham’

‘Paul Lieberstein’=‘Paul Lieerstein’

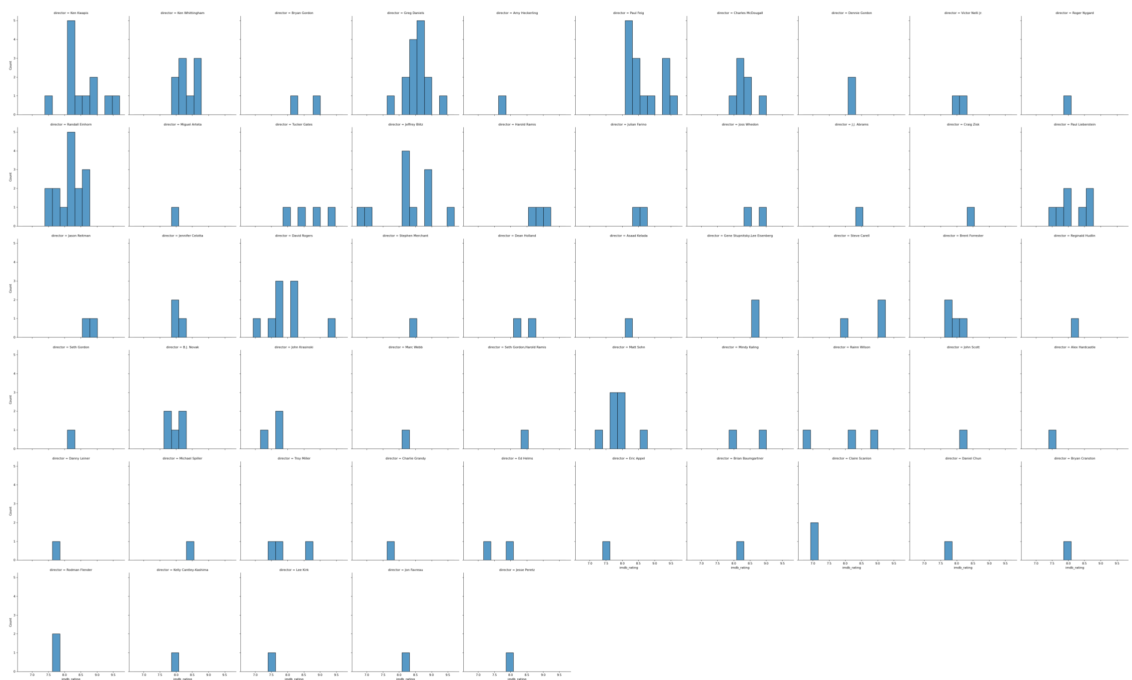
‘Gene Stupnitsky;Lee Eisenberg’=‘Lee Eisenberg;Gene Stupnitsky’ (here we used the order of the first letter)

Hence we needed to unify them.

[12]:
```
data.loc[data['director'] == 'Charles McDougal', 'director'] = 'Charles␣
↪McDougall'
data.loc[data['director'] == 'Claire Scanlong', 'director'] = 'Claire Scanlon'
data.loc[data['director'] == 'Greg Daneils', 'director'] = 'Greg Daniels'
data.loc[data['director'] == 'Ken Wittingham', 'director'] = 'Ken Whittingham'
data.loc[data['director'] == 'Paul Lieerstein', 'director'] = 'Paul Lieberstein'
data.loc[data['director'] == 'Lee Eisenberg;Gene Stupnitsky', 'director'] =␣
↪'Gene Stupnitsky;Lee Eisenberg'
```

[13]:
```
sns.displot(data=data, x='imdb_rating', col='director', col_wrap=10)
```

[13]:
```
<seaborn.axisgrid.FacetGrid at 0x16b0e6400>
```



From the plot above, it is observed that lots of directors only directed the show once. Directors which directed the show more than once tend to produce higher rating episodes than those that only directed the show once. Ken Kwapis, Greg Daniels, Paul Feig, Tucker Gates, Jeffery Blitz, and David Rogers are some of the directors that has directed high ratings episodes.

Therefore, suggesting that the director’s familiarity with the show, due to exposure in directing it, plays a role in how the episode is received by the audience.

writer

[14]:
```
np.sort(data.writer.unique())
```

[14]:
```
array(['Aaron Shure', 'Allison Silverman', 'Amelie Gillette',
       'Anthony Q. Farrell', 'B.J. Novak', 'Brent Forrester',
       'Brent Forrester;Justin Spitzer', 'Caroline Williams',
       'Carrie Kemper', 'Charlie Grandy', 'Dan Greaney', 'Dan Sterling',
       'Daniel Chun', 'Daniel Chun;Charlie Grandy',
       'Gene Stupnitsky;Lee Eisenberg', 'Graham Wagner', 'Greg Daniels',
       'Greg Daniels;Mindy Kaling', 'Halsted Sullivan;Warren Lieberstein',
       'Jason Kessler', 'Jennifer Celotta',
       'Jennifer Celotta;Greg Daniels',
       'Jennifer Celotta;Paul Lieberstein', 'Jon Vitti',
       'Jonathan Green;Gabe Miller', 'Jonathan Huges', 'Justin Spitzer',
       'Larry Willmore', 'Lee Eisenberg;Gene Stupnitsky',
       'Lee Eisenberg;Gene Stupnitsky;Michael Schur', 'Lester Lewis',
       'Michael Schur', 'Michael Schur;Lee Eisenberg;Gene Stupnitsky',
       'Mindy Kaling', 'Nicki Schwartz-Wright', 'Owen Ellickson',
       'Paul Lieberstein', 'Paul Lieberstein;Michael Schur', 'Peter Ocko',
       'Ricky Gervais;Stephen Merchant',
       'Ricky Gervais;Stephen Merchant;Greg Daniels', 'Robert Padnick',
       'Ryan Koh', 'Steve Carell', 'Steve Hely', 'Tim McAuliffe',
       'Warren Lieberstein;Halsted Sullivan'], dtype=object)
```

There are some identical pairs but with different orders,

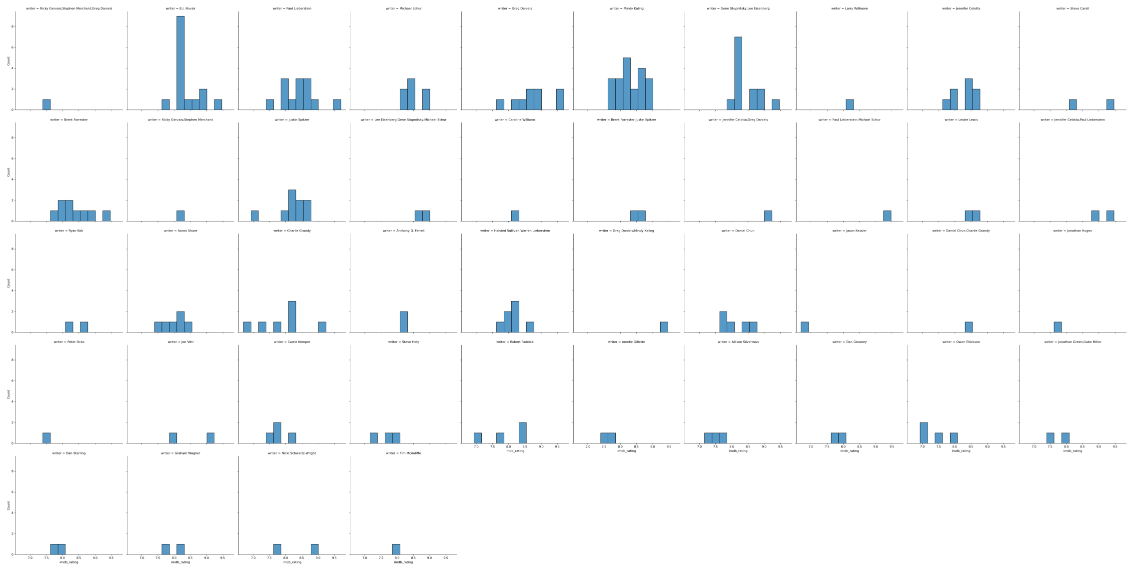
> 'Gene Stupnitsky;Lee Eisenberg'='Lee Eisenberg;Gene Stupnitsky'
>
> 'Halsted Sullivan;Warren Lieberstein'='Warren Lieberstein;Halsted Sullivan'
>
> 'Lee Eisenberg;Gene Stupnitsky;Michael Schur'='Michael Schur;Lee Eisenberg;Gene Stupnitsky'

Here we adopted the first one in each pair.

[15]:
```
data.loc[data['writer'] == 'Lee Eisenberg;Gene Stupnitsky', 'writer'] = 'Gene␣
 ↪Stupnitsky;Lee Eisenberg'
data.loc[data['writer'] == 'Warren Lieberstein;Halsted Sullivan',
         'writer'] = 'Halsted Sullivan;Warren Lieberstein'
data.loc[data['writer'] == 'Michael Schur;Lee Eisenberg;Gene Stupnitsky',
         'writer'] = 'Lee Eisenberg;Gene Stupnitsky;Michael Schur'
```

[16]:
```
sns.displot(data=data, x='imdb_rating', col='writer', col_wrap=10)
```

[16]:
```
<seaborn.axisgrid.FacetGrid at 0x16b0f62b0>
```



From the plot above, we observed that most episodes are written by the small number of writers. Some writers, such as B.J. Novak, Paul Lieberstein, Greg Daniels, Gene Stupnitsky; Lee Eisenberg, Brent Forrester, Jennifer Celotta; Paul Lieberstein, and Paul Lieberstein; Michael Schur had wrote high ratings episodes.

main_chars

[17]:
```
len(np.sort(data.main_chars.unique()))
```

[17]: 122

There are too many combinations of the main characters. It is unrealistic to put all of them into the model due to the small data size. We need another way to represent the character data.

The ultimate goal is to find a combination of features to produce the highest ratings and interpret the results. In this case, some variables are useless, such as `season`, `episode`, `episode_name`(it could be useful but hard to interpret, i.e., what kind of title is eye-attractive), and `air_date`. Some need to be further processed, such as `director`, `writer`, `main_chars`. However, due to the limited dataset, it is impractical to build a model without any selection process to predict what kind of combinations of the directors, the writers, and the main characters produce high-rating episodes.

To solve the problem, we used two methods, first is to analyse whether the numbers of directors/writers/characters have impact on the ratings. Another is using Lasso regression to select the appropriate covariates, such as what directors/writers could be important factors and should be put in the model, which would be implemented in Section 3.

Count the number of directors/writers/characters

[18]:
```
data.loc[:,'n_director'] = data.director.str.count(';')+1
data.loc[:,'n_writer'] = data.writer.str.count(';')+1
data.loc[:,'n_main_chars'] = data.main_chars.str.count(';')+1
```

Because all directors, writers, and main characters are seperated by ";", counting the number of ";" and adding 1 is an easy way to calculate the number of them.

Create dummy variables for directors and writers

[19]:
```
dir_dummies = pd.get_dummies(data, columns = ["director"])
all_dummies = pd.get_dummies(dir_dummies, columns = ["writer"])
```

Using these dummy variables for Lasso regression to select the appropriate covariates.

[20]:
```
all_dummies.head()
```

[20]:
```
   season  episode   episode_name  imdb_rating  total_votes    air_date  \
0       1        1          Pilot          7.6         3706  2005-03-24
1       1        2  Diversity Day          8.3         3566  2005-03-29
2       1        3    Health Care          7.9         2983  2005-04-05
3       1        4   The Alliance          8.1         2886  2005-04-12
4       1        5     Basketball          8.4         3179  2005-04-19

   n_lines  n_directions  n_words  n_speak_char  … writer_Paul Lieberstein  \
0      229            27     2757            15  …                       0
1      203            20     2808            12  …                       0
2      244            21     2769            13  …                       1
3      243            24     2939            14  …                       0
4      230            49     2437            18  …                       0

   writer_Paul Lieberstein;Michael Schur  writer_Peter Ocko  \
0                                      0                  0
1                                      0                  0
2                                      0                  0
3                                      0                  0
4                                      0                  0

   writer_Ricky Gervais;Stephen Merchant  \
0                                      0
1                                      0
2                                      0
3                                      0
4                                      0

   writer_Ricky Gervais;Stephen Merchant;Greg Daniels  writer_Robert Padnick  \
0                                                  1                      0
1                                                  0                      0
2                                                  0                      0
3                                                  0                      0
4                                                  0                      0

   writer_Ryan Koh  writer_Steve Carell  writer_Steve Hely  \
0                0                    0                  0
```

```
1                   0                         0                       0
2                   0                         0                       0
3                   0                         0                       0
4                   0                         0                       0

   writer_Tim McAuliffe
0                     0
1                     0
2                     0
3                     0
4                     0

[5 rows x 113 columns]
```

1.3.4 2.4 Integrating with dialogue data

Calculate the lines every character speaks in an episode Instead of using all the main charaters with their occurence, we decided to look at the number of lines per character, in this way, not only could we know about the occurence of the characters but also about the activeness of the character in an episode.

[21]:
```python
# Get count of number of lines per character per episode
count_lines = lines.groupby(by = ["season", "episode", "speaker"],
                            as_index = False).count()\
                      .loc[:,["season", "episode", "speaker", "line"]]
```

[22]:
```python
count_lines.head()
```

[22]:
```
   season  episode speaker  line
0       1        1  Angela     1
1       1        1  Dwight    29
2       1        1     Jan    12
3       1        1     Jim    36
4       1        1   Kevin     1
```

Here we have the number of lines every character speaks in an episode. The next step is to integrate them to original data.

[23]:
```python
# get all the main characters in main_chars column for further use

char_set = set()

# get all the combination of the main characters in main_chars column
for i in data.main_chars.unique():
    temp = i.split(';')
    if char_set == None:
        # use set to store the main characters in the show
        char_set = set(temp)
```

```python
    else:
        # combine two sets to eliminate the duplicated items
        char_set = char_set | set(temp)
```

[24]:
```python
for i in char_set: # loop through all characters in main_chars
    # for each character, select the corresponding entries
    # from count_lines and merge them to all_dummies
    temp = count_lines.loc[count_lines["speaker"] == i]\
        .drop("speaker", axis=1).rename({'line':f'{i}_lines'},
                                        axis='columns')
    all_dummies = pd.merge(all_dummies, temp, on=["season", "episode"],
                           how="outer")
```

[25]:
```python
# there must be some N/A values when using outer joins in this case,
# fill N/A values with 0 to indicate
# there is no line for certain character in an episode
all_dummies.fillna(0, inplace=True)
```

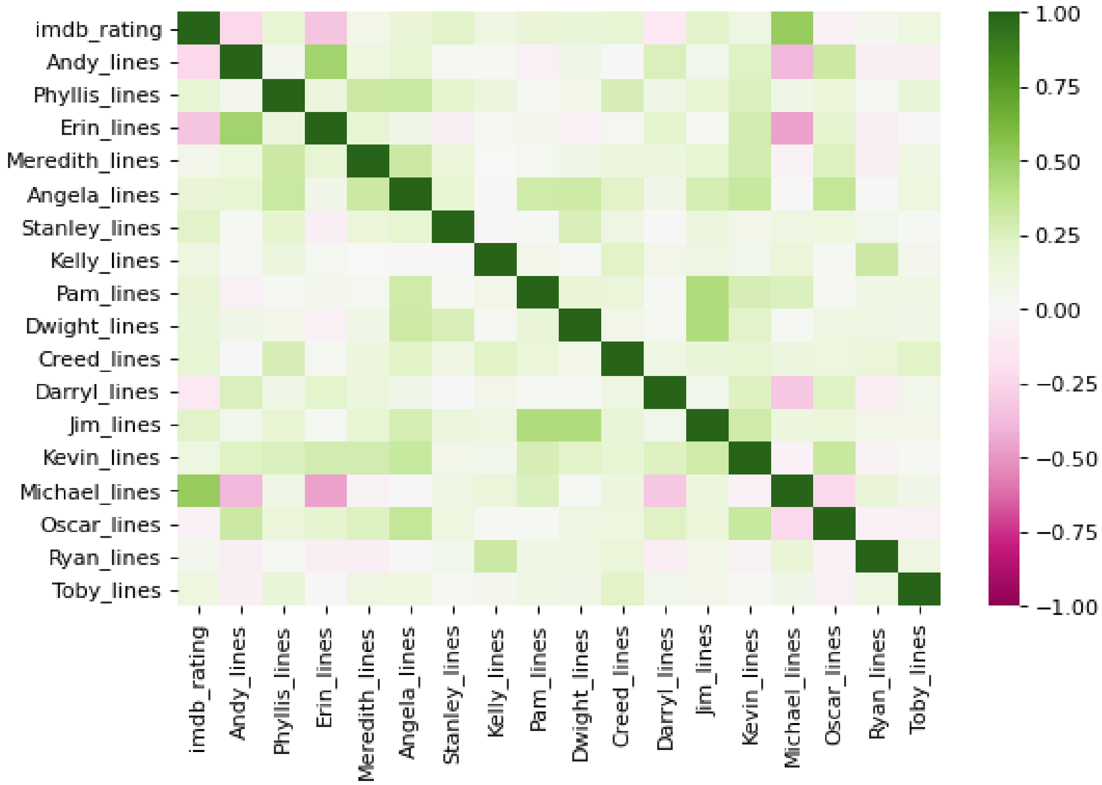
[26]:
```python
all_dummies.head()
```

[26]:
```
   season  episode   episode_name  imdb_rating  total_votes    air_date  \
0       1        1          Pilot          7.6         3706  2005-03-24
1       1        2  Diversity Day          8.3         3566  2005-03-29
2       1        3    Health Care          7.9         2983  2005-04-05
3       1        4   The Alliance          8.1         2886  2005-04-12
4       1        5     Basketball          8.4         3179  2005-04-19

   n_lines  n_directions  n_words  n_speak_char  … Kevin_lines  Jim_lines  \
0      229            27     2757            15  …         1.0       36.0
1      203            20     2808            12  …         8.0       25.0
2      244            21     2769            13  …         6.0       42.0
3      243            24     2939            14  …         3.0       49.0
4      230            49     2437            18  …         1.0       21.0

   Meredith_lines  Angela_lines  Ryan_lines  Michael_lines  Darryl_lines  \
0             0.0           1.0         8.0           81.0           0.0
1             0.0           4.0         4.0           75.0           0.0
2             3.0           5.0         1.0           55.0           0.0
3            10.0           7.0         4.0           68.0           0.0
4             0.0           3.0         8.0          104.0          15.0

   Andy_lines  Kelly_lines  Pam_lines
0         0.0          0.0       40.0
1         0.0          2.0       12.0
2         0.0          0.0       32.0
3         0.0          0.0       22.0
4         0.0          0.0       14.0
```

[5 rows x 130 columns]

[27]:
```
sns.heatmap(all_dummies.loc[:,["imdb_rating", 'Andy_lines', 'Phyllis_lines',␣
↪'Erin_lines', 'Meredith_lines',
       'Angela_lines', 'Stanley_lines', 'Kelly_lines', 'Pam_lines',
       'Dwight_lines', 'Creed_lines', 'Darryl_lines', "Jim_lines",␣
↪"Kevin_lines",
        "Michael_lines", "Oscar_lines", "Ryan_lines", "Toby_lines"]].corr(),
            vmin = -1, vmax = 1, cmap = 'PiYG')
```

[27]: <AxesSubplot:>



From the plot above, it is obvious that there are negative correlations between the activeness of Andy/Erin and the ratings. However, there is positive correlation between the activeness of Michael and the ratings.

This suggests that the audience enjoys when Michael speaks a lot in an episode, and when Andy/Erin speaks less in an episode.

1.3.5 2.5 Feature selection

We used correlation to select the directors and writers that would be considered for the model. Because we want to focus on "What directors/writers can produce better episodes? What directors/writers should the team hire?" rather than "What directors/writers can produce worse episodes? What directors/writers the team should not hire?", it is intuitive to know whether there are positive correlations between the directors/writers and the ratings. We only want to consider the ones with positive correlations.

[28]:
```
dir_corr = all_dummies.loc[:, ["imdb_rating", 'director_Alex Hardcastle',␣
↪'director_Amy Heckerling',
       'director_Asaad Kelada', 'director_B.J. Novak',
       'director_Brent Forrester', 'director_Brian Baumgartner',
       'director_Bryan Cranston', 'director_Bryan Gordon',
       'director_Charles McDougall', 'director_Charlie Grandy',
       'director_Claire Scanlon', 'director_Craig Zisk',
       'director_Daniel Chun', 'director_Danny Leiner',
       'director_David Rogers', 'director_Dean Holland',
       'director_Dennie Gordon', 'director_Ed Helms',
       'director_Eric Appel', 'director_Gene Stupnitsky;Lee Eisenberg',
       'director_Greg Daniels', 'director_Harold Ramis',
       'director_J.J. Abrams', 'director_Jason Reitman',
       'director_Jeffrey Blitz', 'director_Jennifer Celotta',
       'director_Jesse Peretz', 'director_John Krasinski',
       'director_John Scott', 'director_Jon Favreau',
       'director_Joss Whedon', 'director_Julian Farino',
       'director_Kelly Cantley-Kashima', 'director_Ken Kwapis',
       'director_Ken Whittingham', 'director_Lee Kirk',
       'director_Marc Webb', 'director_Matt Sohn',
       'director_Michael Spiller', 'director_Miguel Arteta',
       'director_Mindy Kaling', 'director_Paul Feig',
       'director_Paul Lieberstein', 'director_Rainn Wilson',
       'director_Randall Einhorn', 'director_Reginald Hudlin',
       'director_Rodman Flender', 'director_Roger Nygard',
       'director_Seth Gordon', 'director_Seth Gordon;Harold Ramis',
       'director_Stephen Merchant', 'director_Steve Carell',
       'director_Troy Miller', 'director_Tucker Gates',
       'director_Victor Nelli Jr.']].corr()
```

[29]:
```
to_drop = set()

for i in range(len(dir_corr.columns)):
    # Drop all directors with negative correlation with IMDB rating
    if (dir_corr.iloc[0, i]) < 0:
        colname = dir_corr.columns[i]
        to_drop.add(colname)

all_dummies = all_dummies.drop(labels = to_drop, axis = 1)
```

[30]:
```
writer_corr = all_dummies.loc[:, ["imdb_rating", 'writer_Aaron Shure',
       'writer_Allison Silverman', 'writer_Amelie Gillette',
       'writer_Anthony Q. Farrell', 'writer_B.J. Novak',
       'writer_Brent Forrester', 'writer_Brent Forrester;Justin Spitzer',
       'writer_Caroline Williams', 'writer_Carrie Kemper',
       'writer_Charlie Grandy', 'writer_Dan Greaney',
       'writer_Dan Sterling', 'writer_Daniel Chun',
       'writer_Daniel Chun;Charlie Grandy',
       'writer_Gene Stupnitsky;Lee Eisenberg', 'writer_Graham Wagner',
       'writer_Greg Daniels', 'writer_Greg Daniels;Mindy Kaling',
       'writer_Halsted Sullivan;Warren Lieberstein',
       'writer_Jason Kessler', 'writer_Jennifer Celotta',
       'writer_Jennifer Celotta;Greg Daniels',
       'writer_Jennifer Celotta;Paul Lieberstein', 'writer_Jon Vitti',
       'writer_Jonathan Green;Gabe Miller', 'writer_Jonathan Huges',
       'writer_Justin Spitzer', 'writer_Larry Willmore',
       'writer_Lee Eisenberg;Gene Stupnitsky;Michael Schur',
       'writer_Lester Lewis', 'writer_Michael Schur',
       'writer_Mindy Kaling', 'writer_Nicki Schwartz-Wright',
       'writer_Owen Ellickson', 'writer_Paul Lieberstein',
       'writer_Paul Lieberstein;Michael Schur', 'writer_Peter Ocko',
       'writer_Ricky Gervais;Stephen Merchant',
       'writer_Ricky Gervais;Stephen Merchant;Greg Daniels',
       'writer_Robert Padnick', 'writer_Ryan Koh', 'writer_Steve Carell',
       'writer_Steve Hely', 'writer_Tim McAuliffe']].corr()
```

[31]:
```
to_drop = set()

for i in range(len(writer_corr.columns)):
    # Drop all writers with negative correlation with IMDB rating
    if (writer_corr.iloc[0, i]) < 0:
        colname = writer_corr.columns[i]
        to_drop.add(colname)

all_dummies = all_dummies.drop(labels = to_drop, axis = 1)
```

1.3.6 2.6 Sentiment analysis

We thought that the sentiment, or the mood of each episode is a key factor of ratings. Hence, we conducted the sentiment analysis.

[32]:
```
# apply the pre-trained sentiment analyzer
sia = SentimentIntensityAnalyzer()
```

[33]:
```
# combine the sentiment results to the dialogue dataframe
for i, j in enumerate(lines.line):
    pol_score = sia.polarity_scores(j)
    lines.loc[i, 'neg'] = pol_score['neg']
```

```
    lines.loc[i, 'neu'] = pol_score['neu']
    lines.loc[i, 'pos'] = pol_score['pos']
    lines.loc[i, 'compound'] = pol_score['compound']
```

[34]:
```
lines.head()
```

[34]:
```
   season  episode  title  speaker  \
0       1        1  Pilot  Michael
1       1        1  Pilot      Jim
2       1        1  Pilot  Michael
3       1        1  Pilot      Jim
4       1        1  Pilot  Michael

                                              line  neg    neu    pos  \
0  All right Jim. Your quarterlies look very good…  0.0  0.803  0.197
1         Oh, I told you. I couldn't close it. So…  0.0  1.000  0.000
2  So you've come to the master for guidance? Is …  0.0  1.000  0.000
3         Actually, you called me in here, but yeah.  0.0  0.714  0.286
4    All right. Well, let me show you how it's done.  0.0  0.811  0.189

   compound
0    0.4927
1    0.0000
2    0.0000
3    0.4215
4    0.2732
```

Here, we want to know the effects of negative and positive sentiment on the rating. So we used the mean value of these two sentiments to represent the overall negative and positive feelings in an episode.

[35]:
```
lines_sent = lines.groupby(['season', 'episode'])[['neg', 'pos']]\
    .mean().reset_index()
```

[36]:
```
# merge the sentiment results to the original dataset
all_df = pd.merge(left=all_dummies, right=lines_sent,
                  on=['season','episode'], how='left')
```

[37]:
```
all_df[['imdb_rating', 'neg', 'pos']].corr()
```

[37]:
```
             imdb_rating       neg       pos
imdb_rating     1.000000 -0.105865  0.025255
neg            -0.105865  1.000000 -0.252272
pos             0.025255 -0.252272  1.000000
```

The correlation between the rating and the sentiment of each episode is low. It can be related to the overall style of the series (the sentiment of an comedy series should be positive most of time), or the accuracy of the sentiment analysis [1], as it is observed that the sentiment analysis algorithm that was utilised does not accurately capture the sentiment of conversational dialogue (i.e. sarcasm, underlying tone of worry). Therefore, the accuracy of the overall sentiment analysis is low, and thus not as reliable to conclude whether the tone of each episode effects the ratings of the episodes.

1.4 3. Model Fitting and Tuning

In this project, we emphasized the interpretability of the model. Hence, even though some models such as K-Neighbors regression and Decision Tree regression can perform better than Lasso regression in terms of R^2 score and mean squared error, we chose the model that is easy to interpret. For the former models, the interpretation of the feature importance is difficult, so it is hard to deliver meaningful results.

From the last section, we noticed that `n_words` and `n_lines` are extremely highly correlated, so we removed n_words to avoid redundancy. Also, writing a specific number of words in an episode seems more challenging than writing a specific number of lines, so `n_words` was a less useful explanatory variable for our purposes. We did an initial Lasso fit with cross validation using all of the directors and writers who we were considering, and then out of the covariates selected by Lasso, we chose the writer and director with the highest magnitude to consider in our model. We made this decision because we are only interested in conveying which writer and which director are the best, not explaining that certain writers and directors contribute to IMDB score by x amount. We then ran LASSO again, with a dataset that only included the most effective writer and the most effective director, to choose our final mdoel.

Hence, in the final dataset, the columns we used are `n_lines`, `n_directions`, `n_speak_char`, `n_director`, `n_writer`, `n_main_chars`, the covariates related to activeness (`_lines` columns), sentiment data (`neg`, `pos`), and the writer `writer_Jennifer Celotta;Paul Lieberstein` and director `director_Tucker Gates`. We determined the most effective write and director by running a lasso regression considering all of the writers and directors which we had consired potentially appropriate in our model, and then selected the most effective ones based on the magnitude and direction of their estimated coefficients. We chose only to include the most effective ones because we are interested in being able to give advice like "you should use this director and this writer", rather than make statements like "this writer increases the imdb rating by x amount".

We also considered average words per line, however the models did not have improved performance in terms of r squared and mean squared error, and words per line is harder to interpret, so we chose to stick with the number of lines model. We attempted to add interaction terms between main characters and add the dummy variables of `main_chars` from the original dataset as well, but they didn't improve the model. As a result, we didn't include them in our final model.

Besides the K-Neighbors regression and Decision Tree regression we mentioned before, we also tried linear regression, but it tended to overfit the data due to the small dataset and a large number of variables. So we decided to use Lasso regression to strike a balance between interpretability and accuracy.

1.4.1 3.1 Preparing the final dataset

[38]:
```
X_df = all_df[['n_lines', 'n_directions', 'n_speak_char','n_director',
               'n_writer', 'n_main_chars', 'Andy_lines', 'Erin_lines',
               'Kevin_lines', 'Pam_lines','Phyllis_lines', 'Toby_lines',
               'Kelly_lines', 'Jim_lines','Darryl_lines', 'Creed_lines',␣
 ↪'Ryan_lines',
```

```
                'Oscar_lines', 'Dwight_lines', 'Angela_lines', 'Meredith_lines',
                'Stanley_lines', 'Michael_lines', 'neg','pos',
                'director_Tucker Gates', 'writer_Jennifer Celotta;Paul␣
↪Lieberstein']]
```

[39]:
```
# intialise the response variable
y = all_df['imdb_rating']
```

[40]:
```
# separate data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X_df.values, y.values,
                                                    test_size=0.2,
                                                    random_state=50)
```

1.4.2 3.2 Fit the model

Before fitting the data using Lasso regression, we needed to standardised the data, and use the function `LassoCV` provided by sklearn to choose the appropriate parameters α. Here we used the default settings, 5-fold cross validation to search for optimised α.

[41]:
```
# Create pipeline for lasso model
pipe = Pipeline([('scaler', StandardScaler()),
                 ('lasso', LassoCV(random_state=50,
                                   max_iter=50000))])
```

[42]:
```
pipe.fit(X_train, y_train)
```

[42]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('lasso', LassoCV(max_iter=50000, random_state=50))])
```

1.4.3 3.3 Model Summary

To choose the value of α we explored both the BIC and the AIC. The AIC gave us a model with significantly more covariates than the BIC criterion, and so we chose to use the BIC model for the sake of parsimony.

[43]:
```
from sklearn.linear_model import LassoLarsIC
from sklearn.pipeline import make_pipeline
import warnings
warnings.simplefilter("ignore")
lasso_lars_ic = make_pipeline(StandardScaler(), LassoLarsIC(criterion = "bic")).
↪fit(X_train, y_train)
```

Choosing α based on BIC criteria.

[44]:
```
lasso_lars_ic.fit(X_train, y_train)
alpha_bic = lasso_lars_ic[-1].alpha_
```

Parameters of Model with α Selected Based on BIC

```
[45]: clf = sklearn.linear_model.Lasso(alpha = alpha_bic) # Alpha is selected based␣
      ↪on BIC
      clf.fit(X_train, y_train)
      lcv_coef = pd.DataFrame(data = clf.coef_, index = X_df.columns,
                              columns = ['coef'])
      lcv_intercept = pd.DataFrame(data = clf.intercept_,
                                   index = ['Intercept'], columns = ['coef'])
      lcv_c = pd.concat([lcv_coef, lcv_intercept])
      lcv_c.loc[lcv_c.coef != 0, :]
```

[45]:

	coef
n_lines	0.001812
n_directions	0.001588
n_speak_char	0.005560
n_main_chars	0.016573
Andy_lines	-0.004653
Erin_lines	-0.007734
Kevin_lines	0.003651
Pam_lines	-0.002723
Phyllis_lines	-0.000394
Jim_lines	0.000772
Darryl_lines	-0.000777
Ryan_lines	-0.004755
Oscar_lines	-0.005169
Dwight_lines	-0.000246
Angela_lines	-0.001133
Meredith_lines	-0.001486
Stanley_lines	0.012527
Michael_lines	0.003188
Intercept	7.339649

The Lasso method did not choose any of the sentiment analysis data for explaining the imdb rating, which corresponds to the findings above because of the weak correlation between IMDB rating and the sentiment results, and the fact that the sentiment analysis have not captured the sentiment of the conversational dialogue appropriately.

Based on the above, it is intuitive to aim to have a similar number of lines and directions as the longest episode, max(n_lines) or max(n_directions), as there is a positive relationship between n_lines and the rating, and n_directions and the ratings. It is also intuitive to include as many main chracters as possible, and to give all characters lines to boost the rating. However, it is crucial to identify which characters need to be given more lines and which should have less lines, as they contribute differently to the overall rating of an episode.

```
[46]: max(all_df["n_lines"])
```

[46]: 625

```
[47]: max(all_df["n_directions"])
```

[47]: 166

[48]:
```
lcv_d = lcv_c.loc[lcv_c.coef != 0, :]
```

[49]:
```
lcv_d.sort_values('coef')
```

[49]:

	coef
Erin_lines	-0.007734
Oscar_lines	-0.005169
Ryan_lines	-0.004755
Andy_lines	-0.004653
Pam_lines	-0.002723
Meredith_lines	-0.001486
Angela_lines	-0.001133
Darryl_lines	-0.000777
Phyllis_lines	-0.000394
Dwight_lines	-0.000246
Jim_lines	0.000772
n_directions	0.001588
n_lines	0.001812
Michael_lines	0.003188
Kevin_lines	0.003651
n_speak_char	0.005560
Stanley_lines	0.012527
n_main_chars	0.016573
Intercept	7.339649

R^2

[50]:
```
clf.score(X_train, y_train)
```

[50]: 0.3602785266875218

MSE

[51]:
```
y_pred = clf.predict(X_train)
mean_squared_error(y_train, y_pred)
```

[51]: 0.1726602532154785

Looking at the R^2 score, the model can explain the around 36% of variation of the training data. And the mean squared error is around 0.173. To know the ability of prediciton on the unseen dataset, we did the cross validation.

[52]:
```
# use the alpha chosen by cross-validation
clf_cv = Pipeline([('scaler', StandardScaler()),
                    ('lasso', Lasso(random_state = 50, max_iter = 50000,
                                    alpha = clf.alpha))])
cross_validate(clf_cv, X_train, y_train,
```

```
                scoring = ['r2', 'neg_mean_squared_error'])
```

[52]:
```
{'fit_time': array([0.00168705, 0.00132394, 0.00115871, 0.00103331,
0.00127268]),
 'score_time': array([0.00073266, 0.00053954, 0.00052333, 0.00061488,
0.00055218]),
 'test_r2': array([-0.00345902,  0.19380495,  0.1366306 , -0.05407384,
-0.01716202]),
 'test_neg_mean_squared_error': array([-0.22927924, -0.23405634, -0.23134463,
-0.27967424, -0.25079514])}
```

Based on the results above, the BIC model has a good performance in predicting the test data. While the R^2 is unstable across the folds, which suggests that we may be overfitting the data, the MSE is reasonably stable, and the instability may be because we are working with a small dataset.

1.4.4 3.4 Model assumption check

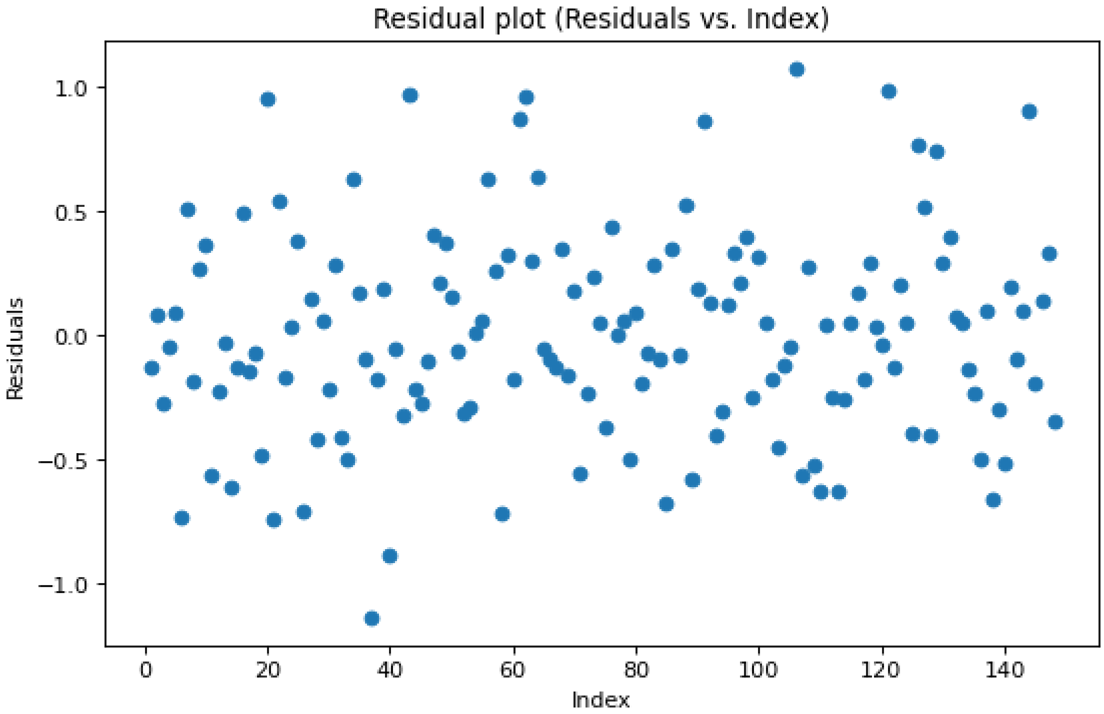
Using the linear model, we needed to check the residuals are normally distributed, have constant variance and mean 0.

[53]:
```
y_pred = clf.predict(X_train)
```

[54]:
```
# Calculate the residuals
res = y_train - y_pred
```

[55]:
```
plt.scatter(range(1, len(y_train)+1), res)
plt.xlabel('Index')
plt.ylabel('Residuals')
plt.title('Residual plot (Residuals vs. Index)')
```

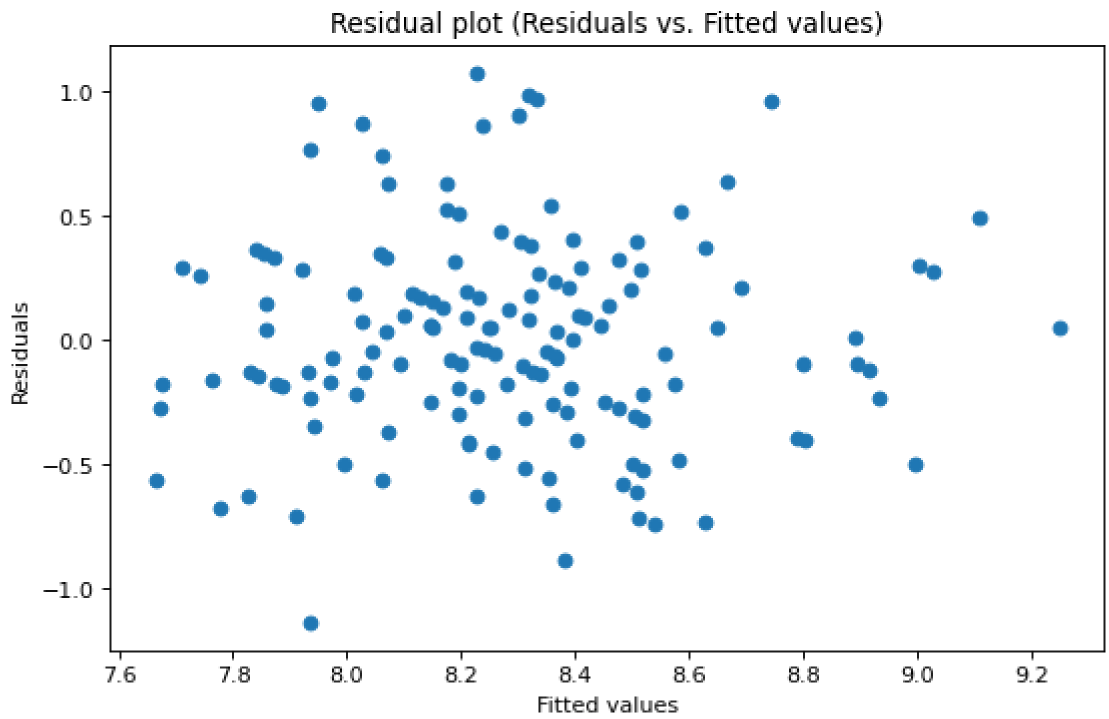
[55]:
```
Text(0.5, 1.0, 'Residual plot (Residuals vs. Index)')
```



The residuals are scattered around 0, indicating the mean of residuals are close to 0.

[56]:
```
plt.scatter(y_pred, res)
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residual plot (Residuals vs. Fitted values)')
```

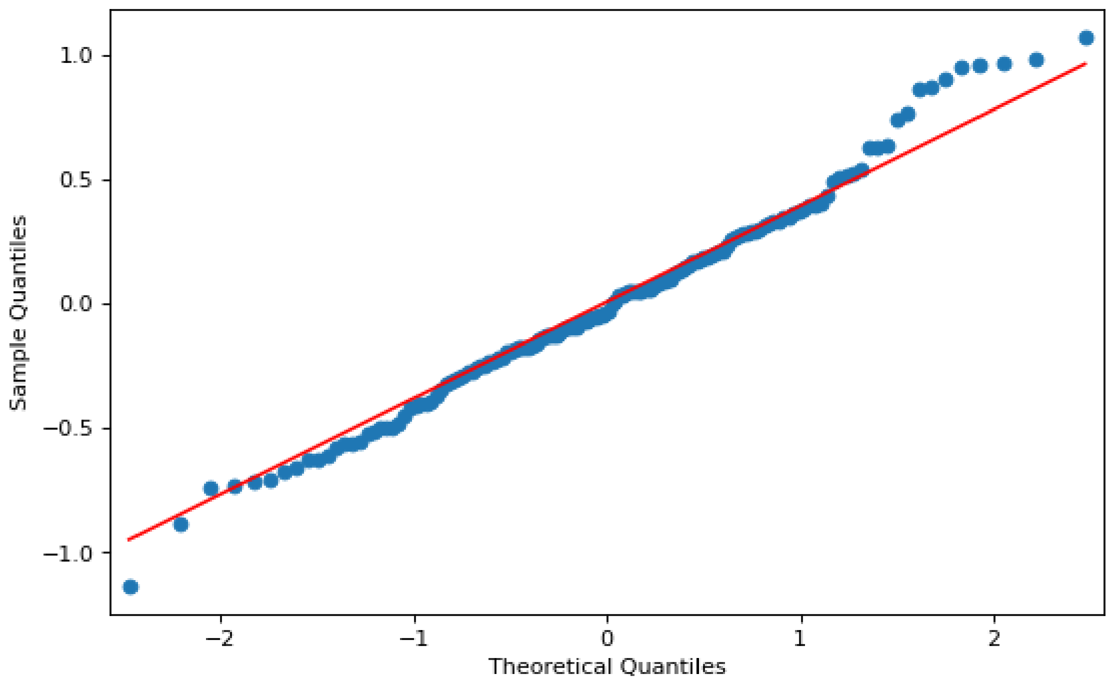
[56]:
```
Text(0.5, 1.0, 'Residual plot (Residuals vs. Fitted values)')
```

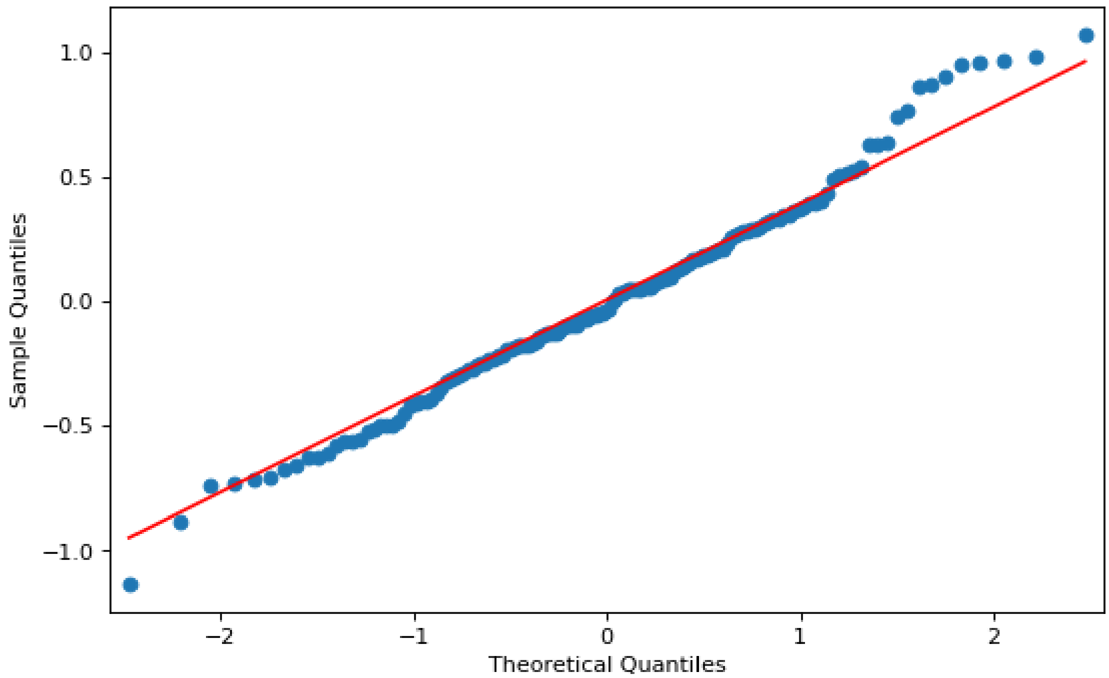


From the plot above, no pattern could be observed between the residuals and fitted values, and the points don't spread or shrink, so the assumption of constant variance holds.

[57]:
```
qqplot(res, line='q')
```

[57]:





From Q-Q plot, the residuals are close to normal distribution. Therefore, concluding that the residuals meet all the assumptions of linear regression, and it is reasonable to use linear regression to fit the data.

1.4.5 3.5 Prediction on the testing data

[58]:
```
# R-squared for pipe
clf.score(X_test, y_test)
```

[58]: 0.39531604025722955

[59]:
```
y_pred_t = clf.predict(X_test)
```

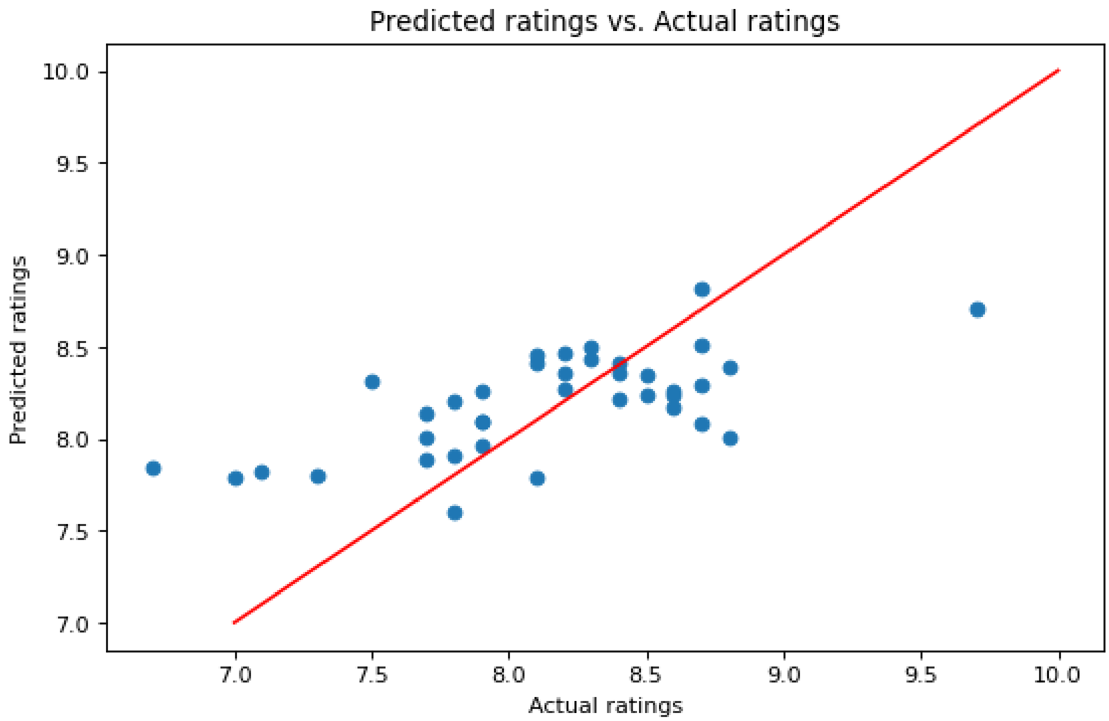
[60]:
```
# Mean squared error for pipe
mean_squared_error(y_test, y_pred_t)
```

[60]: 0.19812612318081502

Applying our model to the testing data, it can now explain 40% of variation in the data and the mean square error stays at 0.198, which exactly the same as the result from the training data, indicating there is no overfitting problem.

[61]:
```python
plt.scatter(y_test, y_pred_t)
plt.plot([7,10],[7,10], c='r')
plt.xlabel('Actual ratings')
plt.ylabel('Predicted ratings')
plt.title('Predicted ratings vs. Actual ratings')
```

[61]: `Text(0.5, 1.0, 'Predicted ratings vs. Actual ratings')`



From the plot above, although our model tends to underestimate the ratings when the actual ratings are high and overestimates the ratings when it is actually lower, the direction is correct, that is, the model still can assign higher ratings to the good episodes even when it underestimates the scores, and lower ratings when the episode is not as good.

As observed, the model does not assign extreme rating values to the episodes, instead the ratings stay between approximately 7.5 to 8.9 range. The actual ratings have a wider range than that of the predicted model.

1.5 4. Discussion and Conclusions

In our final model, we used the variables `n_lines`, `n_directions`, `n_speak_char`, `n_main_chars`, `Andy_lines`, `Erin_lines`, `Kevin_lines`, `Pam_lines`, `Phyllis_lines`, `Jim_lines`,`Darryl_lines`, `Ryan_lines`,`Oscar_lines`, `Dwight_lines`, `Angela_lines`, `Meredith_lines`,`Stanley_lines`, and `Michael_lines` to predict the IMDB ratings. The testing R-squared score is around 0.40 and the mean squared error is around 0.198. This model is choosen as it has the higher R-squared value and the lower mean squared error. In normal setting, the R-squared in the model choosen is not impressive, but considering the small dataset and the complicated interaction, including the social factors and psychological factors, in the real world, the results are acceptable. Although it would provide a conservative prediction, it can still provide the true tendency of the ratings. In other words, it can help predict what kind of episodes can receive higher ratings than others, with meaningful results which the showrunners can implement. Because the model is most valid for values which lie within the range of values in our dataset, and more lines and directions tend to lead to higher ratings, we would suggest aiming for the same number of lines and directions as the maximum values in the dataset. We recommend that writers aim to have 625 lines and 166 directions. We would advise the showrunners to hire as many of the main characters as possible, and to have all of the main characters speak. based on the coefficients of the lines for each character we have ranked the characters to determine who should have the most lines and who should have the least lines. In order from most lines to least lines the characters are ranked as follows: Stanley, Kevin, Michael, Jim, Dwight, Phyllis, Darryl, Angela, Meredith, Pam, Andy, Ryan, Oscar, Erin.

1.6 5. References

[1] The accuracy of the sentiment analysis based on vader, https://towardsdatascience.com/the-best-python-sentiment-analysis-package-1-huge-common-mistake-d6da9ad6cdeb

[2] The technique of sentiment analysis, https://realpython.com/python-nltk-sentiment-analysis/#installing-and-importing

[3] The dataset(the lines in every episode), https://www.kaggle.com/code/nilimajauhari/the-office-sentiment-analysis/data

[4] Official Documentation of `LassoCV`, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.La