project1

March 6, 2021

1 Machine Learning in Python - Project 1

Contributors: Rachel Dance, Silvia Cecilia Hernandez Vargas, Finlay Young

1.1 0. Setup

The following packages are required to run this notebook

[1]:
```
# Install required packages
!pip install -q -r requirements.txt
```

[2]:
```
# Modules to install:
# Display plots inline
%matplotlib inline

#Import Functions file to avoid code in notebook
import ml_functions

# Data libraries
import pandas as pd
import numpy as np
from numpy.random import uniform

import copy
from fuzzywuzzy import process
from fuzzywuzzy import fuzz

# 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
import sklearn
from sklearn.linear_model import LinearRegression, Ridge, Lasso, RidgeCV
```

```python
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures, StandardScaler,␣
↪OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score,␣
↪train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
```

1.2 1. Introduction

Analysis of the provided data describing NBC Universal's series 'The Office' is performed to provide advice to NBC Universal, on the factors important to the success of a special reunion episode. No other datasets were included. Herein, we refer to 'the_office.csv' as the data. The data extracted for this analysis is available publicly. Each of the columns of the data are referred to as 'features', e.g. episode name, number of votes, list of main characters appearing etc.

The feature 'imdb_rating' will be used as a proxy for success or popularity of an episode that NBC seeks to maximise. Therefore, we are interested in features that are most predictive of imdb_rating. To begin extracting this insight, a short data exploration is carried out in Section 2. We also recognise that there is an important relationship between imdb_rating and total_votes, as we not only want the episode to be highly rated, but viewed/voted on by a large number of people. To illustrate, an episode rated 10/10 by 2 people is not considered 'successful'. However, as on average for The Office an episode has 2100 votes, with a minimum 1394 votes, we are not too concerned by this. We also do not have information on how the number of votes varies with number of viewers to estimate a viewing rate prediction. This is left for further work.

Both linear and polynomial regression models are used trialled here, with Lasso and Ridge regularisation techniques to shrink the dataset. These techniques highlight features that do not have high impact on imdb_rating, so they can be removed.

We saw that the most predictive of our models was the linear regression model, in which we used lasso regularisation to carry out variable selection. Features that have been removed before modeling are detailed in Section 2, and model results in Section 4 describe the relative importance of those that remain which form the basis of our recommendations. Analysis found that episode number and total votes are significant to the popularity, as measured by our proxy. However, these variables are not under the control of NBC Universal, and as such will not form part of our recommendations. Key recommendations NBC for the production of a reunion episode are as follows:

1. Include characters Kelly and Michael as they had the most impact on imdb_rating. We would also include Dwight, as although he was removed from our analysis, he has appeared in every single other episode.
2. Ensure as many different characters as possible speak in the episode…?
3. Use any director - there is no preference indicated by the data. ?
4. Put this episode out as part of a long season…. ?
5. The month of July is a good time to air new episodes… ?

[3]:
```
df = pd.read_csv('the_office.csv')

ml_functions.typo_cleaner('director',df)
ml_functions.typo_cleaner('writer',df)
```

```
/work/ml_functions.py:61: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[col_name][i] = matches[1][0]
Typo cleaner finished - if no other output, no changes were made
Typo cleaner finished - if no other output, no changes were made
```

[4]:
```
df_split_test = copy.deepcopy(df)
# Split the main_chars col into individuals and
for i in range(len(df_split_test['main_chars'])):
    df_split_test['main_chars'][i] = df_split_test['main_chars'][i].split(';')

#split the actors into columns (dummies)
character_cols = df_split_test.main_chars.apply(lambda x: pd.Series([1] *␣
 ↪len(x), index=x)).fillna(0, downcast='infer')
```

```
/shared-libs/python3.7/py-core/lib/python3.7/site-
packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
```

1.3 2. Exploratory Data Analysis and Feature Engineering

1.3.1 2.0 Data cleaning

Typos: The data provided is publicly available, and as expected, there are some discrepancies. In order to have a 'correct' dataset as possible, we used "fuzzywuzzy" [https://pypi.org/project/fuzzywuzzy] to correct typos in the names of the directors & writers. From inspection, some names had one letter incorrect, which was causing them to be seen by the dataframes as a separate unique name, and caused double counting. Using fuzzywuzzy (process) we produced a statistic on how good one entry (string) matches other members of the column. On this basis, we were able to identify errors based on the following assumptions:

Each erroneous name is no less than a 90% match to its closest match in the column.

Erroneous names appear exactly once.

We do not run this typo cleaner on entries that contain ';' as this corresponds to entries where multiple names are listed.

For our purposes, this captures all of our typos although we respect that this will not work for every data set ever - particularly assumption 2. Other methods were also considered namely:

Web scraping (using Google & imdb advanced search) - if a name posesses a page on the imdb website which contains an exact match to the spelling of that name, then the name is legitimately a writer or director. This was eliminated on two counts: (1) sheer complexity of returning aAccurate* results, and (2) the *correct* names of other directors/actors etc matching the *incorrect* ones in our dataset, e.g. Charles McDougal and Charles McDougall both have pages, and both are legitimate names.

Compare with additional imdb data, "name.basics.tsv.gz" at (https://datasets.imdbws.com/), updated daily - we can compare our names to entries on a list of all possible names in imdb. However, this was eliminated on two counts: (1) the dataset was 700MB and computationally infeasible in this case, and (2) the file is also authored by imdb and is likely to *also* contain the error we are trying to find so comparison saw no results.

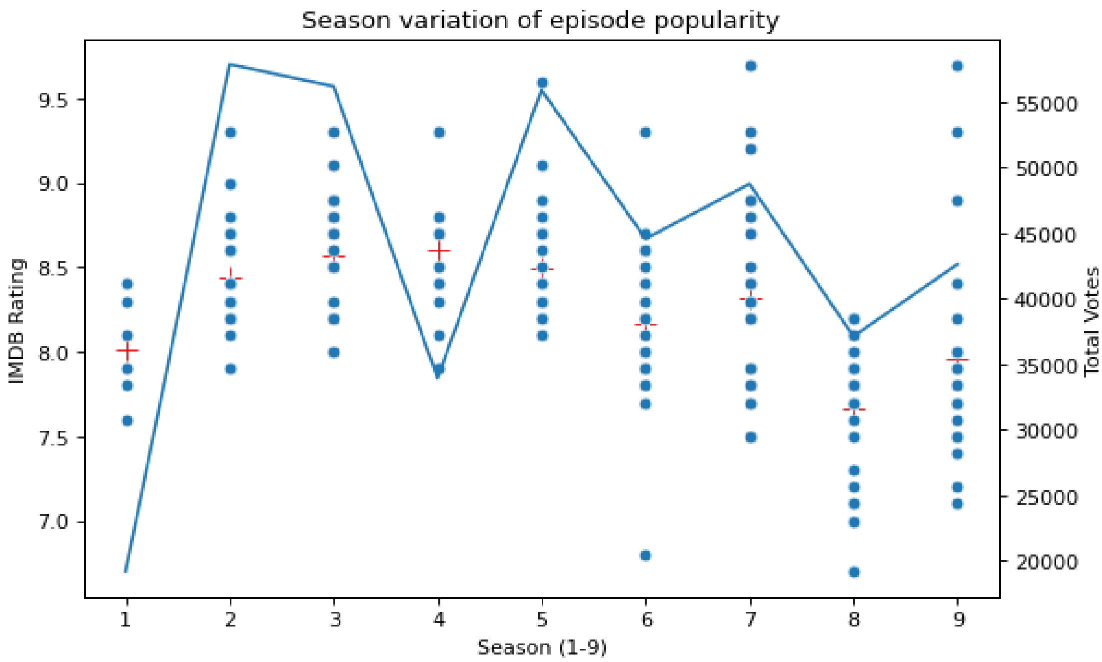
The "typo_clearner function can be fully inspected in the ml_functions.py file.

Main Characters Column: The column in the data where main charachters in an episode are listed is formatted as a single string, e.g. "Andy;Angela;Dwight;Jim;Michael;Pam;Phyllis;Stanley". This is not indexable so this was split into a 'list of lists', where unique names were identified and dummy columns were created to show whether or not a character appeared in an episode. This effectively performs the 'pandas.get_dummies()' method on all unique entries in all of the lists concurrently.

1.3.2 2.1 Initial analysis pairplot interpretation.

As a first step, a pairplot was created on the whole dataset as recieved to inspect any possible features which might have a relation with each other, specially for the variable for interest. The pairplot has been omitted as it was only used as a first indication of underlying trends and did not influence the modelling in any tangible way. ### 2.2 Exploratory Analysis #### 2.2.1 Ratings skew with season As the show has aired for 9 seasons, we can see from the data that the average rating has decreased (see red crossed data point) since season 4, where the average peaked >8.5, however this also comes with a vast increase in the varience of ratings given per episode, and reduction in the number of total votes submitted per episode. This is likely due to the popularity of the show itself, and therefore attracting a wider audience. From this observation, we do not expect the "season" feature itself to have an influence on predicting a high rating episode: as we can see here the last season has some of the highest number of votes received, but still a low average rating. Also, any additonal episode(s) will be part of the next sequencial season.

[5]:
```
ml_functions.season_plot(df)
```

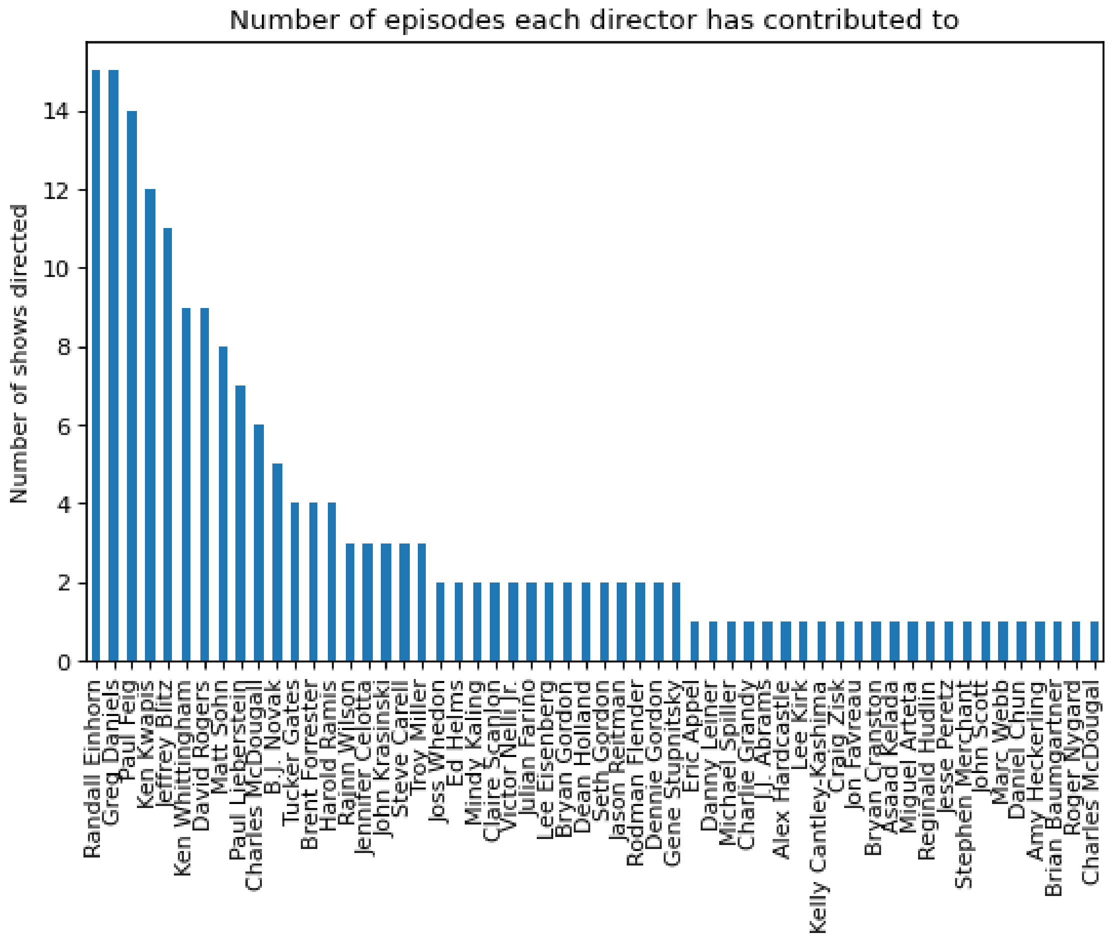


On this basis, the season feature has been removed from data. A prediction that the most popular episodes in a particular season may be present in the data, but holds no predictive power for this application. We have chosen to keep the episode number feature as this may indicate to producers whether an episode is likely to be successful following a small or large number of episodes. #### 2.2.2 Categorial data ##### Peoples Names There are 3 text columns which are lists of names, and these did not feature in a pairplot analysis. To quantify characters, writers and directors we will apply one hot encoding to allow us to assess these as separate features. This will allow for the study progress with eliminating writers, characters and directors at a feature level. ##### Episode names Further, we have episode names which could potentially be explored using a 'bag of words' analysis to classify episodes into sucessful or not (based on an imdb rating threshold). We could also explore the influence of verbs, nouns or adjectives. However, in this study we are are not going to do such analysis, as each of the episodes names is unique and is unlikely to be predictive in itself. Therefore, we drop the column in further analysis.

Directors & writers There are a large number of episodes which were written or directed by multiple people in collabouration with one another. Also, as seasons go on, some new writers/directors appear.

There are many unique writers and directors, and some combinations in the data set, i.e. on one episode, director A and director B worked together. As it is impossible to determine the influence of the individual in these cases as we do not who 'who did what', it is assumed here that director AB is a different 'person', and extra credit is not givento directors A and B individually. This further ensures that only one director is attributed to any episode. In order to work with finding out whether a writer or director is correlated with an increasing imdb rating, more than one data point is needed. As such, we have chosen to omit from the data, any writer or director that appears only once.

[6]:
```
ml_functions.director_episode_plot(df)
```



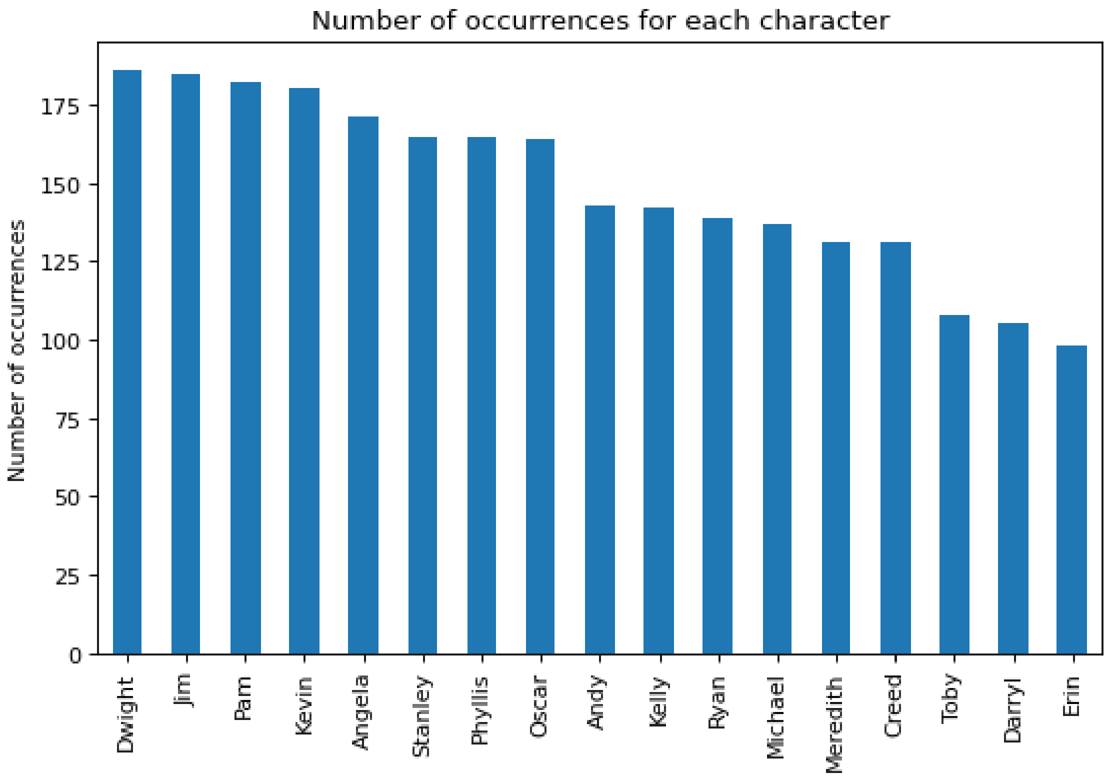
1.3.3 2.3 Feature Engineering

Sparse data points Data which appears sparsely represents a rare event. In the instance where a director (say) is seen only once, they will only appear in *either* the training or the test data, and in only one of the folds where a k-fold method is used. This may cause numerical issues with the modelling and is not deemed to provide any predictive influence. As such, any writer director or character that appears only once in the dataset is removed from the data, as part of feature reduction.

Further, as it is not possible to draw any correlation between two variables with less than (realistically) 5 datapoints, any writer, director or character that appears less than 5 times is also removed due to lack of predictive power.

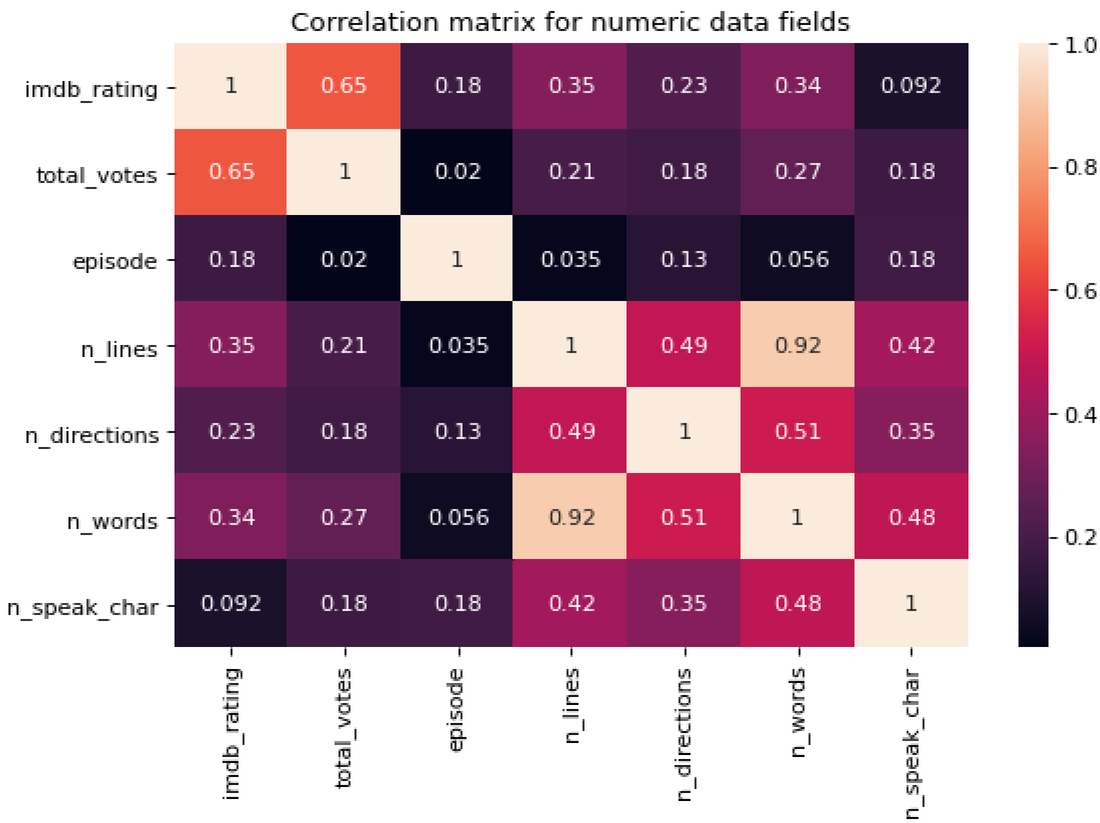
Other non predictive data points Conversely, we also observe that some features occur in the data for *every* episode. For example, the character 'Dwight' appears in every episode. A feature that appears in all episodes cannot provide any predictive power, as it represents a constant feature that will not correlate with others. We have extended this notion and eliminated directors, characters and writers that appear in 95% or *more* of the episodes (i.e. they appear in >176 episodes our of 186) by a similar justification that this also represents a 'constant' features, and any predictive insight is unlikely. On this basis we delete: the charaters Dwight,Jim,Pam & Kevin.

[7]:
```
ml_functions.character_occurrences(df)
```



Correlation Matrices The following correlation matrices are used to give an insight into which features to drop from the model. The matrix below illustrates correlation of our numeric data with imdb_rating. We can see here that the features which are closest to zero are not correlated - we are interested here principally in the imdb_rating.

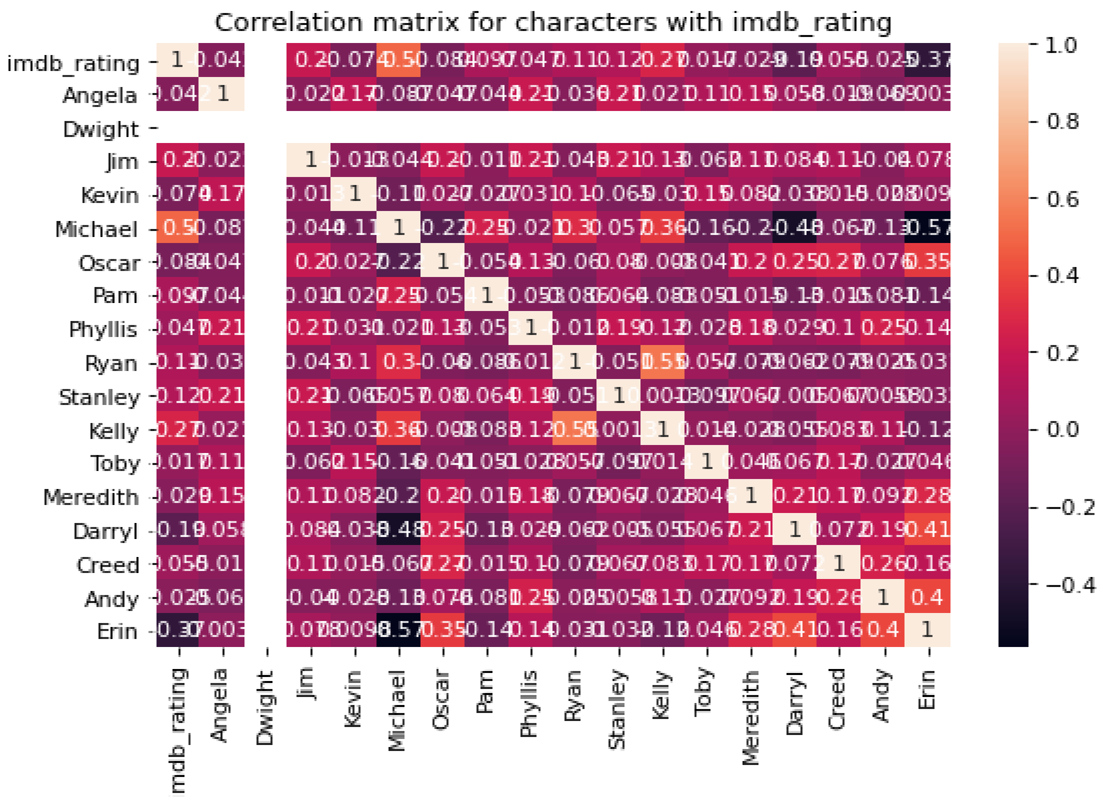
[8]:
```
ml_functions.create_corr_matrix_numeric(df)
```



The matrix shows that each of these features has either a direct, or secondary correlation with imdb_rating. Therefore we did not choose to delete any of these features.

Correlation matrix - characters The matrix below shows the correlation between imdb_rating and the characters. We can immediately see that the character ‘Dwight’ has a correlation of 1.0 for imdb_rating as it appears in all episodes, and this is the only character to do this. This reinforces our decision to remove this feature. We also observe that characters Jim,Pam,Kevin have correlation values close to zero. This indicates they have little or no influence on the imdb_rating.

[9]:
```
ml_functions.create_corr_matrix_characters(df,character_cols)
```



Correlation Matrices - writer/director The final correlation matrix we consider is that of writers and directors. This matrix shows broadly that the effect of the writer or director with imdb_rating is small. We see the largest contribution to the rating from the writers Greg Daniels and charlie Gandy. All other writers had a much smaller contributions so there are omitted. Running this for directors by the same reasoning means that 46 distinct directors (single names or combinations of individual) and 36 writers were also eliminated.

[10]:
```
# Function to create a list of writers and directors to delete
to_elim_dirs, to_elim_writers = ml_functions.
 ↪elim_writers_directors(df_split_test)

# Final list of columns to eliminate in feature engineering steps
to_elim = to_elim_writers + to_elim_dirs + ['Dwight','Jim','Pam','Kevin']

# Concatenate the columns
df_full = ml_functions.create_df_full(df_split_test, character_cols)
# Concatenate the dataframes where actors are already split, remove duplicates
df_full_dummies = pd.get_dummies(df_full)
df_model_data = df_full_dummies.drop(to_elim, axis=1)
```

```
/work/ml_functions.py:224: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  single_dirs['director'] = 'director_' + single_dirs['director'].astype(str)
/work/ml_functions.py:236: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  single_writers['writer'] = 'writer_' + single_writers['writer'].astype(str)
```

[11]:
```
ml_functions.create_corr_matrix_writers(df_model_data, writer_director='writer')
```

View Date The date the show aired at was altered from a date to a month to give a more general categorisation. This was done to allow for more episodes to have a month in common, and allowed for dummy month features to be constucted for 9 months.

1.4 3. Model Fitting and Tuning

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

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

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

1.4.1 3.1 Model Functions

The below functions were consistently used accross all of the models within this sections.

1.4.2 3.2 Linear regression

Two linear regression models were constructed: one standard Linear regression, and one with standartdisation. The Model with standardisation provided an RMSE of >8, and negative predicted ratings. This is of course so inaccurate that this model serves no putpose, and therefore we will only consider the standard Linear regression model.

[12]:
```
# Run the linear reg for dataframe df_model_data (one with directors missing␣
↪etc)
lin_reg, lin_reg_train,rmse_train_lin_reg,rmse_test_lin_reg = ml_functions.
↪run_linear_regression(df_model_data, show_output = False)
```

1.4.3 3.3 Polynomial

Polynomial Regression with no interaction Each of the numerical features within this model were processed as polynomial features without interactions, which will ensure that the model does not increade the number of coeficients. All of the binomial features which only include a 1/0 can be "passed through", as a ploynomial processing on these fields is negligible, as the result will simply be 1 or 0.

[13]:
```
poly_reg_noint, poly_reg_noint_train,rmse_train_poly_reg_noint,␣
↪rmse_test_poly_reg_noint = ml_functions.run_poly_noint(df_model_data,␣
↪show_output = False)
```

Regular Polynomial regression A standard model which allowed for interactions; this slighly outperformed the non-interacting model with a lower "best fit" and RMSE. However, the differences are so slight that models can be interpreted as being equivalent. Both polynomial models are consistent with suggesting that a polynomial degree of 1 is the best fit for the model; given this consistent result, Linar regression should be the predictive model for this dataset.

[14]:
```
poly_reg, poly_reg_train, rmse_train_poly_reg, rmse_test_poly_reg =␣
↪ml_functions.run_polynomial_regression(df_model_data, show_output = False)
```

1.4.4 3.4 Regularisation

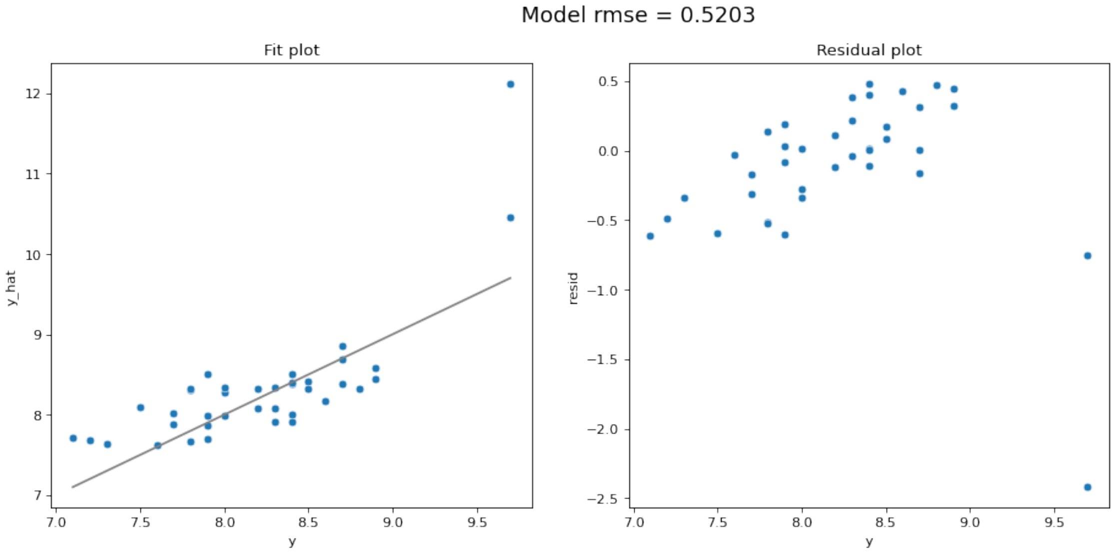
Lasso Polynomial The lasso model has provided the lowest RMSE of all models, and again the best polynomial degree recommended was 1. The lasso model was of particular interest as this will further reduce the number of features in the model, in a dataset which has an already reduced number of features.

[15]:
```
def run_lasso_(dataframe, show_output:bool):
    '''
    Runs polynomial regression with Lasso regularization, including predictions␣
↪for the test data.

    Inputs:
        dataframe == pandas dataframe on which to run lasso model
        show_output == boolean set to true or false; true gives terminal
                        and plotting outputs
    Returns:
        third_grid == results from GridSearch Cross-validation
        rmse_train == RMSE from train data
        rmse_test  == RMSE from test data
        results == dataframe containing the true values, estimated values and␣
↪residuals (from test data)
        results_train == == dataframe containing the true values, estimated␣
↪values and residuals (from train data)
    '''
    X_train, X_test, y_train, y_test = ml_functions.
↪dataframe_prep(dataframe,'imdb_rating')

    alpha_list = np.linspace(0.01, 15, num=100)

    third = make_pipeline(
            StandardScaler(),
            PolynomialFeatures(),
            Lasso(fit_intercept= False)
        )

    parameters = {'polynomialfeatures__degree': np.arange(1,3,1),
        'lasso__alpha': alpha_list}
```

```
    kf = KFold(n_splits=5, shuffle=True, random_state=0)

    third_grid = GridSearchCV(third, parameters, cv=kf,␣
↪scoring="neg_root_mean_squared_error").fit(X_train, y_train)

    y_hat = third_grid.predict(X_test)
    y_hat_train = third_grid.predict(X_train)

    ml_functions.model_fit(third_grid, X_test, y_test, plot = show_output)
    rmse_test = mean_squared_error(y_test, y_hat, squared=False)
    rmse_train = mean_squared_error(y_train, y_train, squared=False)

    results = pd.DataFrame(data = {'y': y_test, 'y_hat': y_hat,
                                   'resid': round(y_test - y_hat,1)})

    results_train = pd.DataFrame(data={'y_train': y_train, 'y_hat_train':␣
↪y_hat_train})

    if show_output == True:
        print("best param: ", third_grid.best_params_)
        print("best score: ", third_grid.best_score_ *-1)
        print("number of coefficients:",len(third_grid.best_estimator_.
↪named_steps["lasso"].coef_))
        print("intercept == ", third_grid.best_estimator_.named_steps["lasso"].
↪intercept_)

    return third_grid, rmse_train, rmse_test, results, results_train
```

[16]:
```
model, lasso_rmse_train, lasso_rmse_test, results_table, results_table_train  =␣
↪run_lasso_(df_model_data, show_output = True)
coefs = model.best_estimator_.named_steps["lasso"].coef_
Features = pd.DataFrame({'Feature':df_model_data.columns.drop('imdb_rating'),␣
↪'coefficients': coefs[1:] })
results_table
```

```
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.0667510001447253, tolerance: 0.8094089999999998
  positive)
```




```
best param:  {'lasso__alpha': 0.01, 'polynomialfeatures__degree': 1}
best score:  0.38235067661304317
number of coefficients: 56
intercept ==  0.0
```

[16]:

	y	y_hat	resid
106	8.4	8.384924	0.0
45	8.9	8.577038	0.3
158	7.2	7.685957	-0.5
63	7.9	8.503321	-0.6
135	9.7	10.456779	-0.8
66	8.3	7.918583	0.4
18	8.3	8.335011	-0.0
109	8.6	8.175420	0.4
141	7.3	7.634256	-0.3
7	8.2	8.319642	-0.1
5	7.8	8.311438	-0.5
162	7.8	8.326384	-0.5
153	7.9	7.864186	0.0
176	7.6	7.627820	-0.0
118	7.9	7.985662	-0.1
97	8.2	8.087104	0.1
37	8.7	8.860470	-0.2
93	8.0	7.985846	0.0
134	8.9	8.453192	0.4
126	8.3	8.081038	0.2
55	8.8	8.328757	0.5
83	8.4	8.002262	0.4
56	8.5	8.413943	0.1

```
149  7.9   7.706242   0.2
167  7.1   7.711942  -0.6
163  7.7   7.875138  -0.2
74   8.4   8.512809  -0.1
111  8.0   8.276219  -0.3
171  8.4   7.917445   0.5
33   8.0   8.337500  -0.3
4    8.4   8.395527   0.0
121  7.5   8.089872  -0.6
168  7.8   7.663808   0.1
61   8.7   8.384017   0.3
44   8.5   8.328345   0.2
26   8.7   8.691231   0.0
185  9.7  12.121106  -2.4
136  7.7   8.013019  -0.3
```

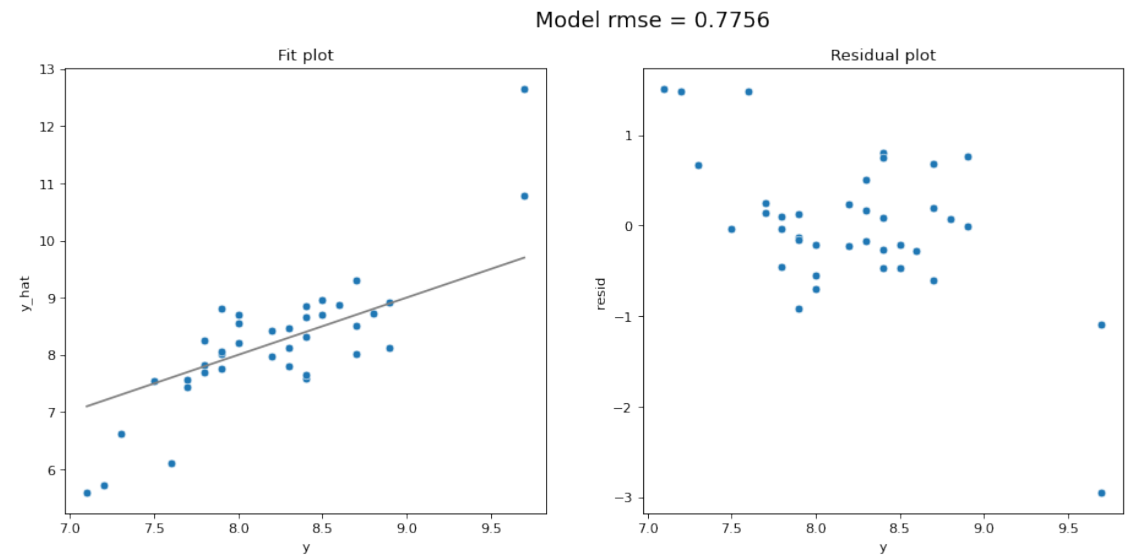
Ridge Regression A Ridge regression to shrink the les important features was also completed. This model was not as effective as the Lasso model. All binomial features have similar magnitude, and this appears to add noise when compared to the lasso model.

[17]:
```
poly_reg_ridge, poly_reg_ridge_train, rmse_train_poly_reg_ridge,␣
↪rmse_test_poly_reg_ridge = ml_functions.run_ridge_(df_model_data,␣
↪show_output = False)
```

1.4.5 3.5 Baseline models

The models above have been run on the full data set for baseline comparison. The is being used to justfiy some of the features which have been eliminated, and es=nsure no high coefficients appear in the unprocessed data.

[18]:
```
lin_reg_full, lin_reg_train_full, rmse_train_lin_reg_full,␣
↪rmse_test_lin_reg_full = ml_functions.run_linear_regression(df_full_dummies,␣
↪show_output = True)
```




```
best index:  0
best param:  {'linearregression__normalize': True}
best neg_root_mean_squared_error (score):  1.308421100830616
number of coefficients: 139
rmse_test ==  0.7756383994082595
rmse_train ==  0.0
[ 4.66264773e-02  6.97821133e-04  2.60324933e-03 -7.00330040e-04
 -6.16893853e-05 -1.15638825e-02 -4.89503592e-02  3.25090531e+00
  2.03986184e+00 -1.90116803e-01  2.33285041e-01 -3.65284620e-02
 -1.66074550e+00 -8.96545303e-02 -1.56434538e-01  9.30736313e-02
  2.25452298e-01 -8.94270796e-02  5.34509998e-02 -5.15408184e-02
  1.12643491e-01  1.27203615e-01 -5.13402390e-02 -1.72417636e-13
  2.46913601e-13 -1.80541529e-01  1.42534000e-01  3.29161877e-01
  4.27377456e-01  8.71855777e-01  1.89518651e-01 -2.27974959e-01
 -1.22715410e-01 -2.40916938e+00 -9.52680009e-01 -3.96086953e-01
  1.07021406e-03 -1.36473341e-01  4.58650217e-01  2.07484711e-01
 -1.85966918e-01 -1.54114571e+00  1.10585797e-01 -8.69924371e-02
 -5.10136589e-02  8.33268963e-02 -9.43689571e-16  3.06571246e-01
 -5.06687705e-02  7.25930036e-01  6.39212225e-01 -9.53527862e-02
  6.12263921e-01  3.45909172e-01  6.14505375e-01  3.24751829e-01
  1.05634344e+00  1.77242416e-02 -9.99753782e-02  7.60060419e-01
  0.00000000e+00  2.65551686e-01  1.87041630e-01  8.82985857e-01
  5.55111512e-17  0.00000000e+00 -5.25957471e-02 -1.63916997e-01
 -3.64803128e-01 -1.50309221e-02  6.10622664e-16  1.02085761e-01
 -2.26666956e-01  3.44440857e-01  1.49880108e-15  6.61029660e-03
  4.36591173e-01  2.93681500e-01  1.02184790e-01 -2.35334747e-01
 -5.08851518e-02 -2.61674693e-01 -2.46986726e-01 -4.71759558e-01
  1.70009727e-01 -2.47521660e-01  2.04438357e-01  2.45033984e-01
  3.03729811e-01 -4.00241253e-01  7.81948027e-01  4.19929993e-01
```

```
 2.63536260e-01 -1.66533454e-16 -1.36955939e-01  1.57024017e-02
 3.89082504e-02  1.54477711e-01  4.16431482e-02  1.21104348e+00
 1.76513204e-01 -2.24055411e-02  1.38588477e-01 -2.67302643e-02
 4.98966908e-02 -2.70065107e-01  4.01302371e-01 -4.61211867e-02
-1.85946103e-01  8.09272625e-01  1.24653091e-01  3.31841658e-01
 5.27489422e-01 -2.94509589e-02  7.02134892e-01 -7.23706779e-01
 3.29750253e-01 -4.93958497e-01  5.55111512e-17  2.85161014e-02
-6.51141927e-01  7.03860365e-03  3.75432226e-01 -2.38692410e-01
-1.19796797e-01  2.94595263e-02 -7.34442452e-03  5.71041502e-02
 1.60824486e-01  1.81871004e-01  1.85730320e-03  1.39369811e-01
 1.00264518e+00  4.10643582e-01  5.85789159e-01  7.10800635e-01
-1.19314621e-01  1.75040014e+00  1.61981979e+00]
intercept ==  0.0
```

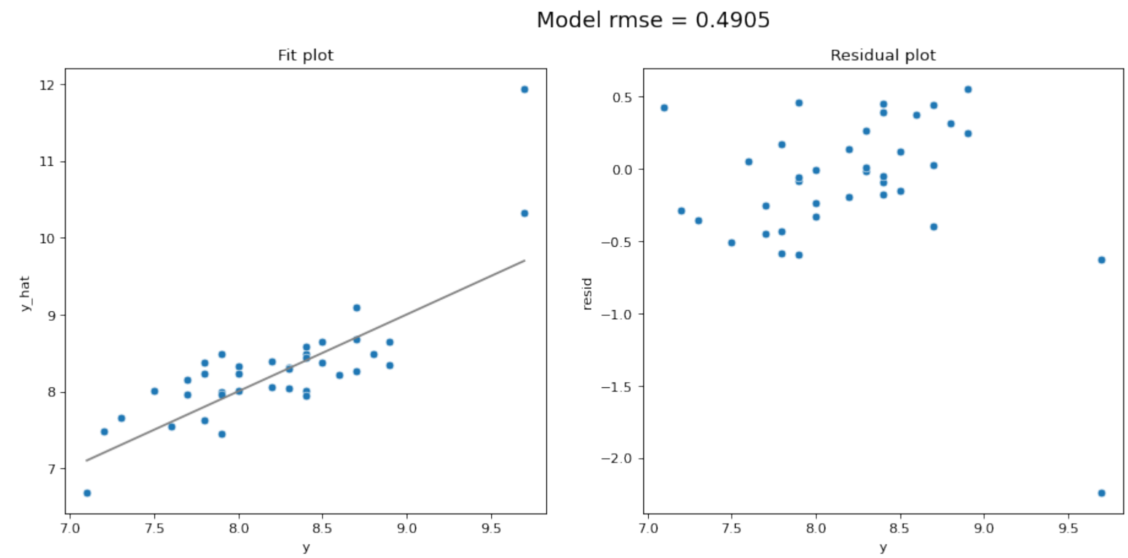
[19]:
```
poly_reg_noint_full, poly_reg_noint_train_full, rmse_train_poly_reg_noint_full,␣
↪rmse_test_poly_reg_noint_full = ml_functions.run_poly_noint(df_full_dummies,␣
↪show_output = False)
```

[20]:
```
poly_reg_full, poly_reg_train_full, rmse_train_poly_reg_full,␣
↪rmse_test_poly_reg_full = ml_functions.
↪run_polynomial_regression(df_full_dummies, show_output = False)
```

[21]:
```
model_full, lasso_rmse_train_full, lasso_rmse_test_full, results_table_full,␣
↪results_table_train_full = run_lasso_(df_full_dummies, show_output=True)
```

```
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8843016104762647, tolerance: 0.812053
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8555540463703626, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1.167279173615297, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8560315968242271, tolerance: 0.8151970000000001
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
```

```
Duality gap: 1.0122483064734311, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8340528027218284, tolerance: 0.8151970000000001
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.979976283504147, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.9518871918507301, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.9224710567691119, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8929122733969876, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8633539930737015, tolerance: 0.808377
  positive)
/shared-libs/python3.7/py/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:532: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.8337957127502307, tolerance: 0.808377
  positive)
```




```
best param:  {'lasso__alpha': 0.01, 'polynomialfeatures__degree': 1}
best score:  0.3963974069809105
number of coefficients: 140
intercept ==  0.0
```

[22]:
```
poly_reg_ridge_full, poly_reg_ridge_train_full, rmse_train_poly_reg_ridge_full,␣
↪rmse_test_poly_reg_ridge_full = ml_functions.run_ridge_(df_full_dummies,␣
↪show_output = False)
```

1.4.6 3.6 Model Error Comparisons

We can evaluate the models erros and compare against one another. The confidence interval is calculated for all of the models (see hidden code) from all models test and training ratings.

[23]:
```
# Confidence intervals for reduced (engineered) data sets

CI_Lasso_test = ml_functions.get_CI(results_table['y'], results_table['y_hat']␣
↪)     # Lasso Model
CI_Lasso_train = ml_functions.get_CI(results_table_train['y_train'],␣
↪results_table_train['y_hat_train'] )     # Lasso Model

CI_Ridge_test = ml_functions.get_CI(poly_reg_ridge['y'],␣
↪poly_reg_ridge['y_hat'] )     # Lasso Model
CI_Ridge_train = ml_functions.get_CI(poly_reg_ridge_train['y_train'],␣
↪poly_reg_ridge_train['y_hat_train'] )     # Lasso Model

CI_Lin_Reg_test = ml_functions.get_CI(lin_reg['y'], lin_reg['y_hat'] )   #␣
↪Linear Regression Model
```

```
CI_Lin_Reg_train = ml_functions.get_CI(lin_reg_train['y_train'],␣
↪lin_reg_train['y_hat_train'] )  # Linear Regression Model

CI_Poly_noint_test = ml_functions.get_CI(poly_reg_noint['y'],␣
↪poly_reg_noint['y_hat'] ) # Polynomial Model with No interationsl
CI_Poly_noint_train = ml_functions.get_CI(poly_reg_noint_train['y_train'],␣
↪poly_reg_noint_train['y_hat_train'] ) # Polynomial Model with No interationsl

CI_Poly_test = ml_functions.get_CI(poly_reg['y'], poly_reg['y_hat'] ) #␣
↪Polynomial Model with No interationsl
CI_Poly_train = ml_functions.get_CI(poly_reg_train['y_train'],␣
↪poly_reg_train['y_hat_train'] ) # Polynomial Model with No interationsl
```

[24]:
```
# Confidence intervals for whole data set

CI_Lasso_test_full = ml_functions.get_CI(results_table_full['y'],␣
↪results_table_full['y_hat'] )    # Lasso Model
CI_Lasso_train_full = ml_functions.get_CI(results_table_train_full['y_train'],␣
↪results_table_train_full['y_hat_train'] )    # Lasso Model

CI_Ridge_test_full = ml_functions.get_CI(poly_reg_ridge_full['y'],␣
↪poly_reg_ridge_full['y_hat'] )    # Lasso Model
CI_Ridge_train_full = ml_functions.get_CI(poly_reg_ridge_train_full['y_train'],␣
↪poly_reg_ridge_train_full['y_hat_train'] )    # Lasso Model

CI_Lin_Reg_test_full = ml_functions.get_CI(lin_reg_full['y'],␣
↪lin_reg_full['y_hat'] )  # Linear Regression Model
CI_Lin_Reg_train_full = ml_functions.get_CI(lin_reg_train_full['y_train'],␣
↪lin_reg_train_full['y_hat_train'] )  # Linear Regression Model

CI_Poly_noint_test_full = ml_functions.get_CI(poly_reg_noint_full['y'],␣
↪poly_reg_noint_full['y_hat'] ) # Polynomial Model with No interationsl
CI_Poly_noint_train_full = ml_functions.
↪get_CI(poly_reg_noint_train_full['y_train'],␣
↪poly_reg_noint_train_full['y_hat_train'] ) # Polynomial Model with No␣
↪interationsl

CI_Poly_test_full = ml_functions.get_CI(poly_reg_full['y'],␣
↪poly_reg_full['y_hat'] ) # Polynomial Model with No interationsl
CI_Poly_train_full = ml_functions.get_CI(poly_reg_train_full['y_train'],␣
↪poly_reg_train_full['y_hat_train'] ) # Polynomial Model with No interationsl
```

Reduced models The results are tabulated below for all of the models which have used the reduced dataset, which have had many features removed as part of the preprocessing

```
[25]: # Create summary data frame for model data (features engineered)
model_data =␣
 ↪[['linear_reg',rmse_train_lin_reg,CI_Lin_Reg_train,rmse_test_lin_reg,CI_Lin_Reg_train],
        ␣
 ↪['poly_noint_reg',rmse_train_poly_reg_noint,CI_Poly_noint_train,rmse_test_poly_reg_noint,CI
        ['poly_reg',␣
 ↪rmse_train_poly_reg,CI_Poly_train,rmse_test_poly_reg,CI_Poly_test],
        ['lasso',␣
 ↪lasso_rmse_train,CI_Lasso_train,lasso_rmse_test,CI_Lasso_test],
        ␣
 ↪['ridge',rmse_train_poly_reg_ridge,CI_Ridge_train,rmse_test_poly_reg_ridge,CI_Ridge_test]]

model_data_summary = pd.DataFrame(model_data, columns = ['model', 'train_RMSE',␣
 ↪'train_RMSE_CI', 'test_RMSE', 'test_RMSE_CI'])

model_data_summary
```

```
[25]:             model  train_RMSE                             train_RMSE_CI  \
0      linear_reg         0.0  [0.14295985105068332, 0.22663173138162424]
1  poly_noint_reg         0.0   [0.13961787432342443, 0.2226556142322073]
2        poly_reg         0.0   [0.14295985105063358, 0.2266317313815745]
3           lasso         0.0   [0.15497149769345536, 0.2454879540715279]
4           ridge         0.0    [0.1509744688156157, 0.2347207490622968]

   test_RMSE                               test_RMSE_CI
0   0.637631  [0.14295985105068332, 0.22663173138162424]
1   0.643452    [0.1803697034543761, 0.5960750788172267]
2   0.637631   [0.17742128388122627, 0.5890929091061786]
3   0.520286   [0.16779727126958036, 0.5019627842558426]
4   0.611648   [0.17912584586134875, 0.5779777622463982]
```

Full data models The results are tabulated below for all of the models with no pre-processing.

```
[26]: # Create summary data frame for whole data
model_data_full = [['linear_reg',rmse_train_lin_reg_full,␣
 ↪CI_Lin_Reg_train_full, rmse_test_lin_reg_full, CI_Lin_Reg_train_full],
        ['poly_noint_reg',rmse_train_poly_reg_noint_full,␣
 ↪CI_Poly_noint_train_full, rmse_test_poly_reg_noint_full,␣
 ↪CI_Poly_noint_test_full],
        ['poly_reg', rmse_train_poly_reg_full, CI_Poly_train_full,␣
 ↪rmse_test_poly_reg_full, CI_Poly_test_full],
        ['lasso', lasso_rmse_train_full, CI_Lasso_train_full,␣
 ↪lasso_rmse_test_full, CI_Lasso_test_full],
        ['ridge', rmse_train_poly_reg_ridge_full, CI_Ridge_train_full,␣
 ↪rmse_test_poly_reg_ridge_full, CI_Ridge_test_full]]
```

```
model_data_summary_full = pd.DataFrame(model_data_full, columns = ['model',␣
↪'train_RMSE', 'train_RMSE_CI', 'test_RMSE', 'test_RMSE_CI'])

model_data_summary_full
```

[26]:
```
            model  train_RMSE  \
0      linear_reg         0.0
1  poly_noint_reg         0.0
2        poly_reg         0.0
3           lasso         0.0
4           ridge         0.0

                                   train_RMSE_CI  test_RMSE  \
0       [0.05334542510607752, 0.09126592305724601]   0.775638
1       [0.05235466664556533, 0.08751345099253478]   0.889433
2  [1.7370968183719121e-12, 2.706411805743506e-12]   1.238651
3          [0.10948062610872387, 0.16900711213849]   0.490460
4        [0.1764674430088074, 0.21838927305417632]   0.574062

                                test_RMSE_CI
0  [0.05334542510607752, 0.09126592305724601]
1   [0.21408629856171696, 0.7813370801859065]
2   [0.23083164207634327, 1.0442917612269853]
3   [0.1721177046539218, 0.48834951829099554]
4    [0.2358754584540893, 0.6035642282839611]
```

[27]:
```
#y_hat = model.predict(X)
#residuals = y.values - y_hat
#residual_sum_of_squares = residuals.T @ residuals
#sigma_squared_hat = residual_sum_of_squares[0, 0] / (N - p)
#var_beta_hat = np.linalg.inv(X_with_intercept.T @ X_with_intercept) *␣
↪sigma_squared_hat
#for p_ in range(p):
#    standard_error = var_beta_hat[p_, p_] ** 0.5
#    print(f"SE(beta_hat[{p_}]): {standard_error}")
```

1.5 4. Discussion & Conclusions

1.5.1 4.1 Model evaluation and interpretation

A Lasso Regression model was selected for this report. The decision to select Lasso regression was a result of various combinations of models and datasets experimented on to provide the most accurate model for predicting the rated episode, as we wish to maintain those features of the data which contribute the most to the feature to explain.

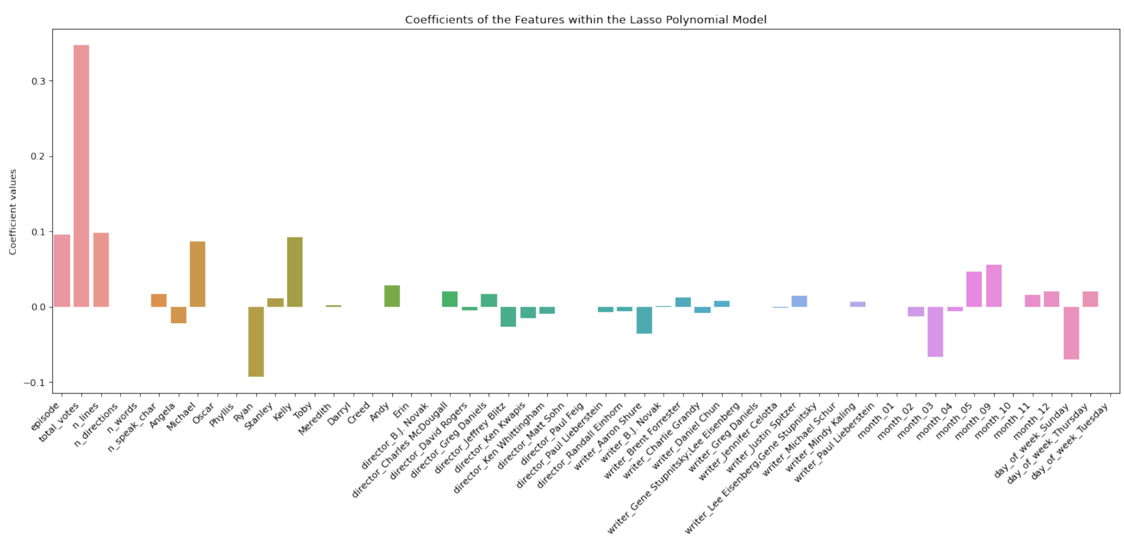
4.1.1 Datasets As explained in the feature engineering section of the report, there were features eliminated from the dataset because these would not help with model accuracy or could be considered as noise data, for example those features which appear in every episode. The predictive models were run with the full dataset as a baseline, and with the features removed as identified in the feature engineering stage of the report. This gave reassurance to assumptions made in the featuring engineering stage when tuning the model.

4.1.2 Model exploration A Regular linear regression model produced a reasonable model with an RMSE of 0.6376, this outperformed polynomial models which were interrogated both with and without interactions between features. The Polynomial models provided a best fit of 1 degree for all features a part from 1 within the reduced dataset. This suggested that a linear regression model was the superior model to further explore.

A Lasso & Ridge regression model were both investigated. Lasso provided the lowest Alpha, as well as the lowest RMSE of all models explored, and gace clarity on many columns which were non-influential to predicting a highly rated episode. Because Lasso Regression was found to be the most accurate and best performing model, this is the chosen model for the report.

1.5.2 4.2 Analysis of Coefficients

[28]:
```
fig, ax = plt.subplots(figsize=(20, 7))
feature_plot = sns.barplot(data = Features, y='coefficients',x = 'Feature' ,␣
 ↪ax=ax)
feature_plot.set_xticklabels(feature_plot.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
ax.set_xlabel('Features within the Lasso Model')
ax.set_ylabel('Coefficient values')
plt.title('Coefficients of the Features within the Lasso Polynomial Model')
plt.show()
```



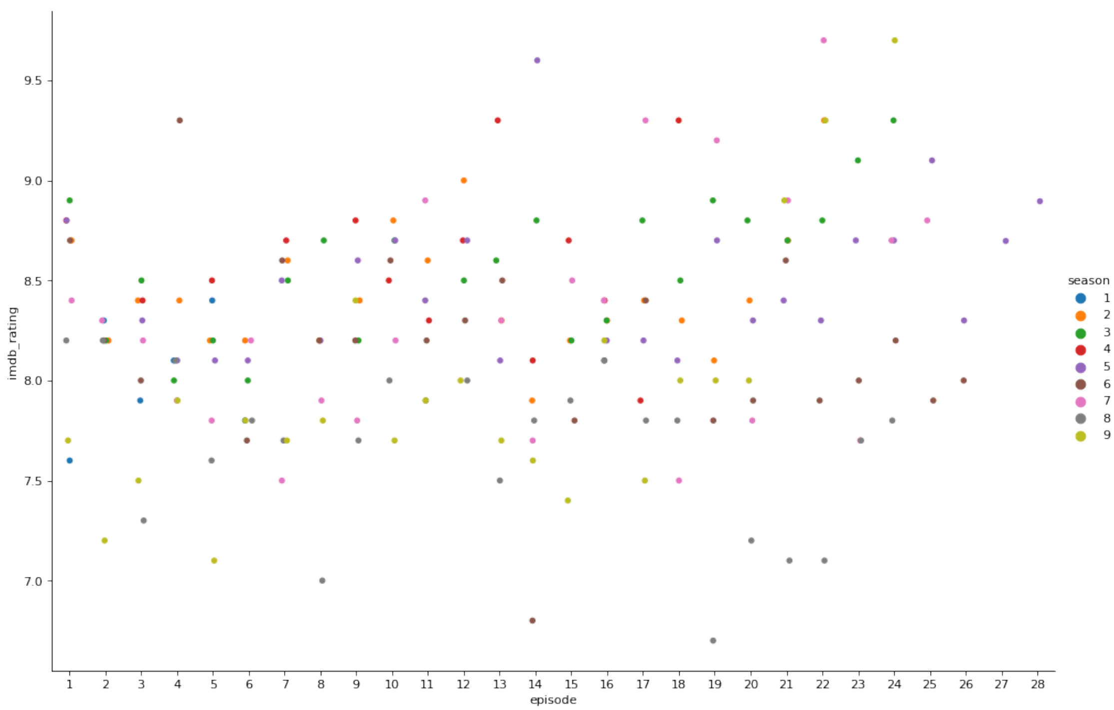
4.2.1 Numerical Features The number of words & number of directions within an episode (n_words & n_directions) were found to not have an impact to the IMDB rating of an episode.

Therefore, Lasso model reduced the coefficients of these features to zero. In the

for the number of speaking characters in an episode(n_speak_chars); this has a positive relationship with a right rating. Despite some characters removed in the preprocessing stage, this coefficient is powerful to know as this can encourage NBC producers to ensure many characters are included in an episode, regardless of the number of lines or directions which these characters have. A special edition episode that aim to have lots of characters "brought back", but should not focus on making each of these characters have significant lines or actions within the episode.

The number of episode, which increase throughout a season, was shown to be highly influential within the model. This is logical to explain, as often writers will ensure that the season finale "ends on a high", and ensures interest in the next season to come. We can see from the below plot that the highest rated episodes within a given season come toward the end of the season (note, not all seasons are of the same length). If one could entertain the idea, it would therefore be recommended that rather than NBC produce one single special edition, that a short special edition season was created, with the finale of this season expected to outperform all others.

[29]:
```
season_plot = sns.catplot(data = df, hue = 'season', x = 'episode', y =
 ↪'imdb_rating',height=8 , aspect=1.5)
plt.show()
```



Total votes is the largest coefficient provided by the Lasso predictive model: it should be noted that NBC will not be in control of the number of votes cast on IMDB, therefore this feature is not a reliable recommendation. However, what can be recommended is for NBC encourage viewers to make reviews, as a high number of reviews appears to result in an overall higher rating.

4.2.2 Binomial Features These are the "dummy" fields which have been direived from text.

It is suggested that NBC show the special edition episode on a Thursday; as its been shown that Sundays viewing tend to provide a negative relationship with how the shows are rated. There is no coefficient for viewing on a Tuesday, and a negative coefficient for Sunday viewings, therefore the small influence that Thursdays do have shall gives the episode a marginal advantage.

There is no correlation to airing a show in January or October, however the model suggests that airing the show between February and April results in less favourable rating - in particular March. Therefore the show should be aired withing either May or September (which has the highest positive coefficient), but not necessarily within the month's in-between as there have not been any episodes aired within these summer months.

[30]:
```
df.head(5)
```

[30]:
```
   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…
```

The following characters were found to have had no influence on an episode, and therefore could be dropped from the model are Phlylis, Ryan, Meredith, Creed and Andy. Therefore, it makes no difference whether these characters should be included in a special edition episode. NBC should note that Michael and Stanley both have negative coefficient resulting from this model; Given that Michael is one of the most recognised characters within the whole "Office" franchise, this result may be challenged and we would not recommend strictly adhering to this recommendation. Screenrant have written an article suggesting (The Office: Every Character, Ranked By Likability | ScreenRant)that Michael is the top liked character in the series, therefore additional feature engineering would explored the popularity of characters more rigorously if further time was available.

4.2.3 Dropped Fields

[31]:
```
Features
```

[31]:

	Feature	coefficients
0	episode	0.095255
1	total_votes	0.346642
2	n_lines	0.097397
3	n_directions	0.000000
4	n_words	0.000000
5	n_speak_char	0.017229
6	Angela	-0.021697
7	Michael	0.086504
8	Oscar	0.000000
9	Phyllis	-0.000000
10	Ryan	-0.092792
11	Stanley	0.010734
12	Kelly	0.091947
13	Toby	0.000000
14	Meredith	0.002338
15	Darryl	-0.000000
16	Creed	0.000000
17	Andy	0.028316
18	Erin	-0.000000
19	director_B.J. Novak	-0.000039
20	director_Charles McDougall	0.019856
21	director_David Rogers	-0.005333
22	director_Greg Daniels	0.016698
23	director_Jeffrey Blitz	-0.026736
24	director_Ken Kwapis	-0.015726
25	director_Ken Whittingham	-0.009836
26	director_Matt Sohn	-0.000000
27	director_Paul Feig	-0.000000
28	director_Paul Lieberstein	-0.007779
29	director_Randall Einhorn	-0.006114
30	writer_Aaron Shure	-0.036400
31	writer_B.J. Novak	0.000480
32	writer_Brent Forrester	0.011899
33	writer_Charlie Grandy	-0.008005
34	writer_Daniel Chun	0.007829
35	writer_Gene Stupnitsky;Lee Eisenberg	-0.000000
36	writer_Greg Daniels	0.000000
37	writer_Jennifer Celotta	-0.001526
38	writer_Justin Spitzer	0.014082

```
39  writer_Lee Eisenberg;Gene Stupnitsky      0.000000
40                     writer_Michael Schur      0.000000
41                      writer_Mindy Kaling      0.006688
42                  writer_Paul Lieberstein      0.000000
43                                 month_01     -0.000000
44                                 month_02     -0.013379
45                                 month_03     -0.066885
46                                 month_04     -0.006484
47                                 month_05      0.046501
48                                 month_09      0.056106
49                                 month_10      0.000000
50                                 month_11      0.015210
51                                 month_12      0.020622
52                      day_of_week_Sunday     -0.070705
53                    day_of_week_Thursday      0.020120
54                     day_of_week_Tuesday     -0.000000
```

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

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

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

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

1.6 5. Convert Document

[ ]:
```
# # Run the following to render to PDF
# !jupyter nbconvert --to markdown project1.ipynb # I dont think this one will␣
↪work?
!jupyter nbconvert --to pdf project1.ipynb
# To hide code:
# jupyter nbconvert --to pdf --TemplateExporter.exclude_input=True my_notebook.
↪ipynb
```

```
[NbConvertApp] Converting notebook project1.ipynb to pdf
[NbConvertApp] Support files will be in project1_files/
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
```

```
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Making directory ./project1_files
[NbConvertApp] Writing 181470 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 682709 bytes to project1.pdf
```