final_code_submit

April 21, 2025

1 Recipe Ratings Analysis

Name(s): Colin Czarnik and Arthur Yang

Website Link: https://colin-czarnik.github.io/RecipeRatingsTextAnalysis/

[101]:
```
import numpy as np
import plotly
import pandas as pd
import plotly.express as px
from sklearn.pipeline import FunctionTransformer
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
import sys
import os
warnings.filterwarnings('ignore')
pd.options.plotting.backend = 'plotly'
from lec_utils import *
```

1.1 Step 1: Introduction

[18]:
```
recipes = pd.read_csv('RAW_recipes.csv')
interactions = pd.read_csv('RAW_interactions.csv')
```

[19]:
```
interactions.head()
```

[19]:

	user_id	recipe_id	date	rating	\
0	1293707	40893	2011-12-21	5	
1	126440	85009	2010-02-27	5	

```
2       57222      85009  2011-10-01       5
3      124416     120345  2011-08-06       0
4  2000192946     120345  2015-05-10       2

                                              review
0  So simple, so delicious! Great for chilly fall…
1  I made the Mexican topping and took it to bunk…
2  Made the cheddar bacon topping, adding a sprin…
3  Just an observation, so I will not rate.  I fo…
4  This recipe was OVERLY too sweet.  I would sta…
```

[20]:
```
interactions.columns, interactions.shape
```

[20]:
```
(Index(['user_id', 'recipe_id', 'date', 'rating', 'review'], dtype='object'),
 (731927, 5))
```

[21]:
```
recipes.head()
```

[21]:
```
                                name      id  minutes  contributor_id  …  \
0  1 brownies in the world    best ever  333281       40          985201  …
1    1 in canada chocolate chip cookies  453467       45         1848091  …
2               412 broccoli casserole  306168       40           50969  …
3               millionaire pound cake  286009      120          461724  …
4                        2000 meatloaf  475785       90         2202916  …

                                             steps  \
0  ['heat the oven to 350f and arrange the rack i…
1  ['pre-heat oven the 350 degrees f', 'in a mixi…
2  ['preheat oven to 350 degrees', 'spray a 2 qua…
3  ['freheat the oven to 300 degrees', 'grease a …
4  ['pan fry bacon , and set aside on a paper tow…

                                         description  \
0  these are the most; chocolatey, moist, rich, d…
1  this is the recipe that we use at my school ca…
2  since there are already 411 recipes for brocco…
3  why a millionaire pound cake?  because it's su…
4  ready, set, cook! special edition contest entr…

                                         ingredients  n_ingredients
0  ['bittersweet chocolate', 'unsalted butter', '…              9
1  ['white sugar', 'brown sugar', 'salt', 'margar…             11
2  ['frozen broccoli cuts', 'cream of chicken sou…              9
3  ['butter', 'sugar', 'eggs', 'all-purpose flour…              7
4  ['meatloaf mixture', 'unsmoked bacon', 'goat c…             13

[5 rows x 12 columns]
```

[22]:
```
recipes.columns, recipes.shape
```

[22]:
```
(Index(['name', 'id', 'minutes', 'contributor_id', 'submitted', 'tags',
        'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
        'n_ingredients'],
       dtype='object'),
 (83782, 12))
```

[23]:
```
# merge tables on recipe id
merged = recipes.merge(interactions, how='left', left_on="id",␣
 ↪right_on="recipe_id")
merged.shape
```

[23]:
```
(234429, 17)
```

[24]:
```
merged.columns
```

[24]:
```
Index(['name', 'id', 'minutes', 'contributor_id', 'submitted', 'tags',
       'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
       'n_ingredients', 'user_id', 'recipe_id', 'date', 'rating', 'review'],
      dtype='object')
```

1.2 Step 2: Data Cleaning and Exploratory Data Analysis

[25]:
```
print(merged[merged['rating']== 0].iloc[244, -1])
```

```
I'm not rating this because I have not made it but I am suggesting that the
references to quantities be taken out of the method because anyone who wants to
scale the servings down to say 12 can be very confused about references to
adding 12 cups of flour!
```

[26]:
```
merged.replace(0, np.nan, inplace = True)
```

[27]:
```
avg_ratings = merged.groupby('recipe_id')['rating'].mean().sort_values()
avg_ratings = avg_ratings.to_frame().rename(columns={'rating':'avg_rating'})
avg_ratings.head()
```

[27]:

recipe_id	avg_rating
469990.0	1.0
423015.0	1.0
416845.0	1.0
468835.0	1.0
289197.0	1.0

[28]:
```
merged = merged.merge(avg_ratings,on='recipe_id')
merged.head()
```

[28]:
```
                                  name      id  minutes  contributor_id  …  \
0  1 brownies in the world    best ever  333281     40.0          985201  …
1    1 in canada chocolate chip cookies  453467     45.0         1848091  …
2                412 broccoli casserole  306168     40.0           50969  …
3                412 broccoli casserole  306168     40.0           50969  …
4                412 broccoli casserole  306168     40.0           50969  …

         date rating                                             review  \
0  2008-11-19    4.0  These were pretty good, but took forever to ba…
1  2012-01-26    5.0  Originally I was gonna cut the recipe in half …
2  2008-12-31    5.0  This was one of the best broccoli casseroles t…
3  2009-04-13    5.0  I made this for my son's first birthday party …
4  2013-08-02    5.0  Loved this.  Be sure to completely thaw the br…

   avg_rating
0         4.0
1         5.0
2         5.0
3         5.0
4         5.0

[5 rows x 18 columns]
```

[29]:
```
# Change column nutrition to calories, total fat, sugar, sodium, protein,␣
 ↪saturated fat, carbohydrates
import re
# convert a string resembling a list of floats into an actual list of floats␣
 ↪(for nutrition column)
def string_to_float_list(s):
    return [float(i) for i in re.findall('\d+\.\d', s)]

# convert a string resembling a list of strings into a list of strings (for␣
 ↪tags and steps columns)
def string_to_string_list(s):
    return re.findall('\'(.+?)\'', s)
```

[30]:
```
# apply string to float list to "nutrition", "steps", "tags"
merged['nutrition'] = merged['nutrition'].apply(string_to_float_list)
merged['steps'] = merged['steps'].apply(string_to_string_list)
merged['tags'] = merged['tags'].apply(string_to_string_list)
```

[31]:
```
merged = merged.assign(**{
    'calories (#)' : merged['nutrition'].str[0],
    'total fat (PDV)' : merged['nutrition'].str[1],
    'sugar (PDV)' : merged['nutrition'].str[2],
    'sodium (PDV)' : merged['nutrition'].str[3],
    'protein (PDV)' : merged['nutrition'].str[4],
```

```
    'saturated fat (PDV)' : merged['nutrition'].str[5],
    'carbohydrates (PDV)' : merged['nutrition'].str[6],
    })
```

[32]:
```
def join_reviews(series):
    return ' '.join(series.dropna().astype(str))

merged_grouped = merged.groupby('id').agg({
    'name': 'first',
    'minutes': 'first',
    'submitted': 'first', # omit contributor_id
    'tags': 'first',
    'nutrition': 'first',
    'n_steps': 'first',
    'steps': 'first',
    'description': 'first',
    'ingredients': 'first',
    'n_ingredients': 'first', # Omit user_id
    'recipe_id': 'first',
    'calories (#)': 'first',
    'total fat (PDV)': 'first',
    'sugar (PDV)': 'first',
    'sodium (PDV)': 'first',
    'protein (PDV)': 'first',
    'saturated fat (PDV)': 'first',
    'carbohydrates (PDV)': 'first',
    'review': join_reviews, # potentially add custom agg function, adding ' ' ␣
 ↪for each string
    'avg_rating': 'first'
}).reset_index()
merged_grouped.head()
```

[32]:

	id	name	minutes	submitted	…	\
0	275022	impossible macaroni and cheese pie	50.0	2008-01-01	…	
1	275024	impossible rhubarb pie	55.0	2008-01-01	…	
2	275026	impossible seafood pie	45.0	2008-01-01	…	
3	275030	paula deen s caramel apple cheesecake	45.0	2008-01-01	…	
4	275032	midori poached pears	25.0	2008-01-01	…	

	saturated fat (PDV)	carbohydrates (PDV)	\
0	62.0	8.0	
1	30.0	20.0	
2	51.0	5.0	
3	67.0	21.0	
4	0.0	33.0	

	review	avg_rating

```
0  Easy comfort food! I definitely thought it was…        3.0
1  When I found myself needing a dessert and havi…        3.0
2  Sorry, this one didn&#039;t work out so well. …        3.0
3  This was the first cheesecake I'd ever made.  …        5.0
4  This needs at least 10 stars.  The recipe was …        5.0

[5 rows x 21 columns]
```

[33]:
```
print(merged_grouped.shape[0])
print(merged_grouped[['minutes']].isna().sum()) # basically nothing. Safe to␣
 ↪just get rid of.
print(merged_grouped[['avg_rating']].isna().sum()) # 2608/83781 missing values.␣
 ↪Much more but no reason to fill in na
```

```
83781
minutes    1
dtype: int64
avg_rating    2608
dtype: int64
```

[34]:
```
[i for i in merged_grouped[merged_grouped['minutes'].isna()].iloc[0]]
```

[34]:
```
[282837,
 'vegan parmesan',
 nan,
 '2008-01-29',
 ['15-minutes-or-less',
  'time-to-make',
  'course',
  'preparation',
  'healthy',
  '5-ingredients-or-less',
  'condiments-etc',
  'easy',
  'vegan',
  'vegetarian',
  'food-processor-blender',
  'dietary',
  'low-cholesterol',
  'high-fiber',
  'high-protein',
  'inexpensive',
  'healthy-2',
  'high-in-something',
  'low-in-something',
  'equipment',
  'small-appliance',
```

```
 '3-steps-or-less'],
 [653.1, 46.0, 9.0, 34.0, 118.0, 13.0, 19.0],
 1,
 ['grind the almonds into a fine powder using a coffee , nut , or spice
grinder'],
 "after finding that nowhere in new zealand has vegan parmesan in stock i
decided to make my own and it tastes great! just only use when serving or for
ingredients. if it's left on a moist dish as a garnish it tends to make the
yeast soggy.",
 "['nutritional yeast', 'whole almond', 'salt']",
 3,
 282837.0,
 653.1,
 46.0,
 9.0,
 34.0,
 118.0,
 13.0,
 19.0,
 "Easy and so good for you, not to mention yummy! I used this over stuffed
peppers last night, and will use my smoothies too! Thanks!! This is really nice
tasting, and so much healthier for you than the real thing!  I used a coffee
grinder to grind the almonds.  Used this over pasta and also in Recipe #242383.
Worked out great both times.  Thanks!  Made for Aussie/Kiwi Swap #19. This
doesn't take exactly like parm, but it's awesome to put over italian-style meals
for added flavor. Thanks so much!",
 5.0]
```

[99]:
```
fig = px.histogram(merged.groupby('recipe_id')['avg_rating'].mean(),␣
 ↪x='avg_rating', nbins=10, title='Histogram of Average Ratings of Recipes',␣
 ↪labels={'avg_rating':'Average Rating', 'percent':'Count'},␣
 ↪histnorm='probability')
fig.write_html('rate_hist.html', include_plotlyjs='cdn')
#fig.show()
```

[36]:
```
(merged_grouped['avg_rating'] >= 4).sum() / merged_grouped.shape[0]
```

[36]:
```
0.9050142633771138
```

[37]:
```
merged_grouped['cal_bins'] = pd.qcut(merged_grouped['calories (#)'], q=5).
 ↪astype(str)
mean_cal_bins = merged_grouped.groupby('cal_bins')['avg_rating'].mean().
 ↪reset_index()
```

[38]:
```
fig1 = px.box(merged_grouped, x='cal_bins', y='avg_rating', title='Average␣
 ↪Rating vs. Calories',
```

```
                category_orders={'cal_bins':['(-0.001, 146.5]','(146.5, 248.9]',␣
 ↪'(248.9, 370.6]', '(370.6, 563.3]', '(563.3, 45609.0]']})
fig1.add_trace(go.Scatter(x=mean_cal_bins['cal_bins'],␣
 ↪y=mean_cal_bins['avg_rating'], mode='markers',
                marker=dict(color='red', size=10), name='Mean'))
fig1.write_html('cal_box.html', include_plotlyjs='cdn')
#fig1
```

[39]:
```
merged_grouped['min_bins'] = pd.qcut(merged_grouped['minutes'], q=5).astype(str)
mean_min_bins = merged_grouped.dropna().groupby('min_bins')['avg_rating'].
 ↪mean().reset_index()
```

[40]:
```
fig1 = px.box(merged_grouped.dropna(), x='min_bins', y='avg_rating',␣
 ↪title='Average Rating vs. Minutes',
                 category_orders={'min_bins':['(0.999, 16.0]', '(16.0, 30.
 ↪0]','(30.0, 45.0]','(45.0, 75.0]','(75.0, 1051200.0]']})
fig1.add_trace(go.Scatter(x=mean_min_bins['min_bins'],␣
 ↪y=mean_min_bins['avg_rating'], mode='markers',
                marker=dict(color='red', size=10), name='Mean'))
fig1.write_html('min_box.html', include_plotlyjs='cdn')
#fig1
```

[77]:
```
merged_grouped['fat_ratio'] = (merged_grouped['saturated fat (PDV)'] /␣
 ↪merged_grouped['total fat (PDV)']).where(X['total fat (PDV)'] != 0, 0)
print(merged_grouped['fat_ratio'].isna().sum())
merged_grouped['fat_bins'] = pd.qcut(merged_grouped['fat_ratio'], q=5).
 ↪astype(str)
mean_sat_bins = merged_grouped.groupby('fat_bins')['avg_rating'].mean().
 ↪reset_index()
```

```
0
```

[81]:
```
merged_grouped['fat_bins'].unique()
```

[81]:
```
array(['(1.776, 5.0]', '(1.242, 1.776]', '(-0.001, 0.438]',
       '(0.824, 1.242]', '(0.438, 0.824]'], dtype=object)
```

[98]:
```
fig1 = px.box(merged_grouped.dropna(), x='fat_bins', y='avg_rating', title='Fat␣
 ↪Ratio vs. Average Ratio',
                 category_orders={'fat_bins':['(-0.001, 0.438]', '(0.438, 0.
 ↪824]','(0.824, 1.242]','(1.242, 1.776]','(1.776, 5.0]']},
                 labels={'avg_rating':"Average Rating", 'fat_bins':'saturated fat␣
 ↪(PDV) / total fat (PDV)'})
fig1.add_trace(go.Scatter(x=mean_sat_bins['fat_bins'],␣
 ↪y=mean_sat_bins['avg_rating'], mode='lines+markers',
```

```
                          line=dict(color='red'), marker=dict(color='red',␣
 ↪size=10), name='Mean'))
fig1.write_html('fat_box.html', include_plotlyjs='cdn')
#fig1
```

[97]:
```
merged_grouped.dropna().pivot_table(index='cal_bins', columns='min_bins',␣
 ↪values='avg_rating', aggfunc='mean')
```

[97]:
```
min_bins          (0.999, 16.0]  (16.0, 30.0]  (30.0, 45.0]  (45.0, 75.0]  \
cal_bins
(-0.001, 146.5]            4.69          4.61          4.57          4.63
(146.5, 248.9]             4.68          4.63          4.61          4.62
(248.9, 370.6]             4.65          4.63          4.60          4.61
(370.6, 563.3]             4.66          4.63          4.61          4.61
(563.3, 45609.0]           4.63          4.62          4.64          4.61

min_bins          (75.0, 1051200.0]
cal_bins
(-0.001, 146.5]                4.62
(146.5, 248.9]                 4.61
(248.9, 370.6]                 4.58
(370.6, 563.3]                 4.60
(563.3, 45609.0]               4.62
```

[96]:
```
merged_grouped.dropna().pivot_table(index='cal_bins', columns='min_bins',␣
 ↪values='avg_rating', aggfunc='mean').to_markdown()
```

[96]:
```
'| cal_bins         |   (0.999, 16.0] |   (16.0, 30.0] |   (30.0, 45.0] |
(45.0, 75.0] |   (75.0, 1051200.0] |\n|:-----------------|----------------:|----
-----------:|---------------:|---------------:|--------------------:|\n|
(-0.001, 146.5]  |         4.69033 |        4.6081  |        4.56837 |
4.63054 |             4.62424 |\n| (146.5, 248.9]   |         4.67717 |
4.62631 |        4.60803 |        4.61518 |            4.61388 |\n| (248.9,
370.6]   |         4.65487 |        4.62798 |        4.59572 |        4.60872 |
4.58442 |\n| (370.6, 563.3]   |         4.66003 |        4.63333 |
4.61248 |        4.60986 |            4.60022 |\n| (563.3, 45609.0] |
4.63045 |        4.62148 |        4.63718 |        4.61324 |             4.62323
|'
```

[41]:
```
# make copy with dropped NA values (just ratings we converted from 0 to NA and␣
 ↪1 mssing minutes value)
merged_copy = merged_grouped.dropna().copy()
merged_copy['tags'] = merged_copy['tags'].apply(lambda x: ' '.join(x))  #␣
 ↪Convert tag list to string
```

1.3 Step 3: Framing a Prediction Problem

[42]:
```
# Perform proper train-test split
X = merged_copy.drop('avg_rating', axis=1)
y = merged_copy['avg_rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=214)
```

1.4 Step 4: Baseline Model

[44]:
```
# make baseline model
# 2 quantitative, 0 ordinal, 0 nominal
def create_baseline_model(): # Concatenating average
    # transform numerical columns
    log_transformer = FunctionTransformer(lambda x: np.log1p(x))

    preprocessing = make_column_transformer(
        (log_transformer, ['calories (#)', 'minutes']),  # Apply log␣
↪transformation to 'calories' and 'minutes'
        remainder='drop'  # Drop other columns not being used
    )

    # The pipeline
    model = make_pipeline(preprocessing, LinearRegression()) # apply l1␣
↪regularization

    # Define feature columns
    feature_cols = ['calories (#)', 'minutes']
    return model, feature_cols

baseline_model, feature_cols = create_baseline_model()
display(baseline_model)
```

```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('functiontransformer',
                                                  ␣
↪FunctionTransformer(func=<function create_baseline_model.<locals>.<lambda> at␣
↪0x7ff610d788b0>),
                                                  ['calories (#)',
                                                   'minutes'])])),
                ('linearregression', LinearRegression())])
```

[45]:
```
baseline_model.fit(X_train[feature_cols], y_train)
baseline_model.predict(pd.DataFrame([{
    'calories (#)': 1, # slight positive coefficient, more caloric recipes more␣
↪favorable
```

```
    'minutes': 8000000 # slight negative coefficient, longer recipes less␣
 ↪favorable
}]))[0]
```

[45]: 4.388951347586458

[46]:
```
# Predict on training and test data
y_train_pred = baseline_model.predict(X_train[feature_cols])
y_test_pred = baseline_model.predict(X_test[feature_cols])

# Compute MSE
baseline_train_mse = mean_squared_error(y_train, y_train_pred)
baseline_test_mse = mean_squared_error(y_test, y_test_pred)

print(f"Baseline Model - Train MSE: {baseline_train_mse:.4f}")
print(f"Baseline Model - Test MSE: {baseline_test_mse:.4f}")
```

```
Baseline Model - Train MSE: 0.4084
Baseline Model - Test MSE: 0.4153
```

[47]:
```
# On test data with two mean squared error estimates (mse of avg_rating > 4 and␣
 ↪avg_rating < 4)
high_ratings_test = y_test > 4
low_ratings_test = y_test <= 4
high_ratings_train = y_train > 4
low_ratings_train = y_train <= 4
baseline_hr_train_predictions = baseline_model.
 ↪predict(X_train[high_ratings_train])
baseline_lr_train_predictions = baseline_model.
 ↪predict(X_train[low_ratings_train])
baseline_hr_test_predictions = baseline_model.predict(X_test[high_ratings_test])
baseline_lr_test_predictions = baseline_model.predict(X_test[low_ratings_test])

baseline_hr_train_mse = mean_squared_error(y_train[high_ratings_train],␣
 ↪baseline_hr_train_predictions)
baseline_lr_train_mse = mean_squared_error(y_train[low_ratings_train],␣
 ↪baseline_lr_train_predictions)
baseline_hr_test_mse = mean_squared_error(y_test[high_ratings_test],␣
 ↪baseline_hr_test_predictions)
baseline_lr_test_mse = mean_squared_error(y_test[low_ratings_test],␣
 ↪baseline_lr_test_predictions)
print("Baseline Model - Train MSE (avg_rating > 4):", baseline_hr_train_mse)
print("Baseline Model - Train MSE (avg_rating  4):", baseline_lr_train_mse)
print("Baseline Model - Test MSE (avg_rating > 4):", baseline_hr_test_mse)
print("Baseline Model - Test MSE (avg_rating  4):", baseline_lr_test_mse)
```

```
Baseline Model - Train MSE (avg_rating > 4): 0.11438249692296486
```

```
Baseline Model - Train MSE (avg_rating   4): 1.4705086351731933
Baseline Model - Test MSE (avg_rating > 4): 0.11486969453245344
Baseline Model - Test MSE (avg_rating   4): 1.515076841709836
```

[48]:
```python
labels = [
    "Baseline (Train)",
    "Baseline (Test)",
    "Baseline (>4 stars) Train",
    "Baseline (>4 stars) Test",
    "Baseline ( 4 stars) Train",
    "Baseline ( 4 stars) Test"
]

mse_values = [
    baseline_train_mse,
    baseline_test_mse,
    baseline_hr_train_mse,
    baseline_hr_test_mse,
    baseline_lr_train_mse,
    baseline_lr_test_mse
]
# Create a bar plot using Plotly
fig = go.Figure()

# Add bars to the plot
fig.add_trace(go.Bar(
    x=labels,
    y=mse_values,
    marker=dict(color=['blue', 'green', 'blue', 'green', 'blue', 'green']),
    text=[f'{val:.4f}' for val in mse_values],  # Annotate bars with MSE values
    textposition='outside',
))

# Update layout for better presentation
fig.update_layout(
    title="Baseline Model - MSE Comparison",
    xaxis_title="Model",
    yaxis_title="Mean Squared Error (MSE)",
    template="plotly_dark",  # Choose a dark theme for the plot
    showlegend=False,
    xaxis_tickangle=-45,
    margin=dict(l=40, r=40, b=60, t=60),
    height=500
)

# Show the plot
#fig.show()
```

```
# Save to html file
fig.write_html('baseline_model_mse_performance.html', include_plotlyjs = 'cdn')
```

1.5 Step 5: Final Model

[49]:
```
# make experimental model (Vector of vector spaces, each index is a date,␣
 ↪manipulate so its reflective of time series)
# 3 quantitative +  a million more quantitative, 0 nominal, 0 ordinal.
def create_experimental_model(): # For average rating, we concatenate all tags␣
 ↪into a large string, and then use that string to predict avg rating. Avg␣
 ↪calories and minutes
    # transform numerical columns
    log_transformer = FunctionTransformer(lambda x: np.log1p(x))

    # Fat ratio function transformer
    fat_ratio_transformer = FunctionTransformer(
        lambda X: ((X['saturated fat (PDV)'] / X['total fat (PDV)']).
 ↪where(X['total fat (PDV)'] != 0, 0)).to_frame(name='fat_ratio')
    )

    # Pipeline to process the ratio: transformer + scaler
    fat_ratio_pipeline = make_pipeline(
        fat_ratio_transformer,
        StandardScaler()
    )

    preprocessing = make_column_transformer(
        (log_transformer, ['calories (#)', 'minutes']),  # Apply log␣
 ↪transformation to 'calories' and 'minutes'
        (TfidfVectorizer(), 'tags'),  # Apply TF-IDF vectorization to 'tags'
        (TfidfVectorizer(), 'review'),
        (fat_ratio_pipeline, ['saturated fat (PDV)', 'total fat (PDV)']),
        remainder='drop'  # Drop other columns not being used
    )

    # Parameter Grid For Grid Search
    param_grid = {'lasso__alpha': [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1,␣
 ↪1], # testing on values 10^-5 to 10^0.
                  'columntransformer__tfidfvectorizer-2__max_features': [1000,␣
 ↪2000, 3000, 4000, 5000]} # testing on values 1000 to 5000.

    # The model pipeline
    pipe = make_pipeline(preprocessing, Lasso()) # l1 regularization
    # print(sorted(pipe.get_params().keys()))

    grid_search = GridSearchCV(
```

```
        pipe,
        param_grid,
        cv=5,  # 5-fold cross-validation
        scoring='neg_mean_squared_error',
        n_jobs=-1  # Need this bc the model with only 1 cpu at a time is super␣
↪slow to run grid search
    )

    # Define feature columns
    feature_cols = ['calories (#)', 'minutes', 'tags', 'review', 'saturated fat␣
↪(PDV)', 'total fat (PDV)']
    return grid_search, feature_cols
```

[50]:
```
experimental_model, feature_cols = create_experimental_model()
display(experimental_model)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('columntransformer',
                                        ␣
↪ColumnTransformer(transformers=[('functiontransformer',
                                                                        ␣
↪FunctionTransformer(func=<function create_experimental_model.<locals>.<lambda>␣
↪at 0x7ff610e22ef0>),
                                                                        ␣
↪['calories '
                                                                          '(#)',
                                                                         ␣
↪'minutes']),
                                                                       ␣
↪('tfidfvectorizer-1',
                                                                        ␣
↪TfidfVectorizer(),
                                                                          'tags'),
                                                                       ␣
↪('tfidfvectorizer-2',
                                                                        ␣
↪TfidfVectorizer(),
                                                                        ␣
↪'review'),
                                                                       ␣
↪('pipelin…
                                                                              ␣
↪          FunctionTransformer(func=<function create_experimental_model.<locals>.
↪<lambda> at 0x7ff610e22e60>)),
                                                                              ␣
↪         ('standardscaler',
```

```
                                                                          ␣
↪          StandardScaler())]),
                                                                          ␣
↪['saturated '
                                                                            'fat '
                                                                          ␣
↪'(PDV)',
                                                                            'total␣
↪'
                                                                            'fat '
                                                                          ␣
↪'(PDV)'])])),
                                        ('lasso', Lasso())]),
             n_jobs=-1,
             param_grid={'columntransformer__tfidfvectorizer-2__max_features':␣
↪[1000,
                                                                            ␣
↪2000,
                                                                            ␣
↪3000,
                                                                            ␣
↪4000,
                                                                            ␣
↪5000],
                         'lasso__alpha': [1e-06, 1e-05, 0.0001, 0.001, 0.01,
                                          0.1, 1]},
             scoring='neg_mean_squared_error')
```

[102]:
```
if not sys.warnoptions:
    warnings.simplefilter("ignore")
    os.environ["PYTHONWARNINGS"] = "ignore" # only way I found to remove␣
 ↪ConvergenceWarning
experimental_model.fit(X_train[feature_cols], y_train) # took about an hour to␣
 ↪run

experimental_model.predict(pd.DataFrame([{
    'calories (#)': 10,
    'minutes': 70000,
    'tags': '60-minutes-or-less chicken-stew', # FOR NOW MUST BE A STRING NOT A␣
 ↪LIST
    'review': 'So delicious!',
    'saturated fat (PDV)': 100,
    'total fat (PDV)': 100
}]))[0]
```

[102]:
```
5.829939850437746
```

[103]:
```
# ON ALL TEST DATA UNCONDITIONAL OF GROUPING BY MSE OF AVG RATINGS
final_train_predictions = experimental_model.predict(X_train)
final_train_mse = mean_squared_error(y_train, final_train_predictions)
final_test_predictions = experimental_model.predict(X_test)
final_test_mse = mean_squared_error(y_test, final_test_predictions)

# printing mse on test
print("Baseline Model - Train MSE:", baseline_train_mse)
print("Final Model - Train MSE:", final_train_mse)
print("Baseline Model - Test MSE:", baseline_test_mse)
print("Final Model - Test MSE:", final_test_mse)
```

```
Baseline Model - Train MSE: 0.4084146545104165
Final Model - Train MSE: 0.2186820725437408
Baseline Model - Test MSE: 0.4153333290611471
Final Model - Test MSE: 0.25128459328522157
```

[104]:
```
# On test data with two mean squared error estimates (mse of avg_rating > 4 and␣
 ↪avg_rating < 4)
high_train_ratings = y_train > 4
low_train_ratings = y_train <= 4
high_test_ratings = y_test > 4
low_test_ratings = y_test <= 4

final_hr_train_predictions = experimental_model.
 ↪predict(X_train[high_train_ratings])
final_lr_train_predictions = experimental_model.
 ↪predict(X_train[low_train_ratings])
final_hr_train_mse = mean_squared_error(y_train[high_train_ratings],␣
 ↪final_hr_train_predictions)
final_lr_train_mse = mean_squared_error(y_train[low_train_ratings],␣
 ↪final_lr_train_predictions)

final_hr_test_predictions = experimental_model.
 ↪predict(X_test[high_test_ratings])
final_lr_test_predictions = experimental_model.predict(X_test[low_test_ratings])
final_hr_test_mse = mean_squared_error(y_test[high_test_ratings],␣
 ↪final_hr_test_predictions)
final_lr_test_mse = mean_squared_error(y_test[low_test_ratings],␣
 ↪final_lr_test_predictions)

print("Final Model - Train MSE (avg_rating > 4):", final_hr_train_mse)
print("Final Model - Train MSE (avg_rating  4):", final_lr_train_mse)
print("Final Model - Test MSE (avg_rating > 4):", final_hr_test_mse)
print("Final Model - Test MSE (avg_rating  4):", final_lr_test_mse)
```

```
Final Model - Train MSE (avg_rating > 4): 0.10824568253570038
```

```
Final Model - Train MSE (avg_rating  4): 0.6175970280916924
Final Model - Test MSE (avg_rating > 4): 0.12598657690128942
Final Model - Test MSE (avg_rating  4): 0.7098947704606539
```

[105]:
```
# MSE values for all recipes (Train vs Test)
labels_all = ["Baseline (Train)", "Baseline (Test)", "Final (Train)", "Final␣
 ↪(Test)"]
mse_values_all = [baseline_train_mse, baseline_test_mse, final_train_mse,␣
 ↪final_test_mse]
colors_all = ['blue', 'blue', 'orange', 'orange']

fig1 = go.Figure()

fig1.add_trace(go.Bar(
    x=labels_all,
    y=mse_values_all,
    marker_color=colors_all,
    text=[f'{val:.4f}' for val in mse_values_all],
    textposition='outside'
))

fig1.update_layout(
    title="MSE Comparison: All Recipes (Train vs Test)",
    xaxis_title="Model",
    yaxis_title="Mean Squared Error",
    showlegend=False,
    xaxis_tickangle=-15,
    template="plotly_white",
    margin=dict(l=40, r=40, t=60, b=60)
)

#fig1.show()
# Save to html file
fig1.write_html('baseline_vs_final_all_mse_performance.html', include_plotlyjs␣
 ↪= 'cdn')
```

[106]:
```
# Labels and values
groups = ['> 4 Stars', '  4 Stars']
splits = ['Train', 'Test']
models = ['Baseline', 'Final']

# Example MSE values - replace these with your actual computed MSEs
mse_values = {
    '> 4 Stars': {
        'Baseline': {'Train': baseline_hr_train_mse, 'Test':␣
 ↪baseline_hr_test_mse},
        'Final': {'Train': final_hr_train_mse, 'Test': final_hr_test_mse}
```

```
    },
    '  4 Stars': {
        'Baseline': {'Train': baseline_lr_train_mse, 'Test':␣
 ↪baseline_lr_test_mse},
        'Final': {'Train':  final_lr_train_mse, 'Test': final_lr_test_mse}
    }
}

# Flatten data for plotting
x = []
y = []
colors = []
texts = []

for group in groups:
    for model in models:
        for split in splits:
            x.append(f"{group}<br>{model} ({split})")
            y.append(mse_values[group][model][split])
            colors.append('blue' if model == 'Baseline' else 'green')
            texts.append(f"{mse_values[group][model][split]:.3f}")
# Create Plotly bar chart
fig = go.Figure(data=[
    go.Bar(
        x=x,
        y=y,
        marker_color=colors,
        text=texts,
        textposition='auto'
    )
])

fig.update_layout(
    title='Train and Test MSE by Rating Group and Model',
    yaxis_title='Mean Squared Error',
    xaxis_tickangle=45,
    plot_bgcolor='white',
    margin=dict(l=40, r=40, t=60, b=100),
    height=500
)

fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='lightgray')
#fig.show()
fig.write_html('baseline_vs_final_grouped_mse_performance.html',␣
 ↪include_plotlyjs = 'cdn')
```

[107]:
```
print("Best alpha:", experimental_model.best_params_['lasso__alpha']) # best␣
 ↪alpha is 0.0001
print("Best word limit for reviews tf idf vectorizer:", experimental_model.
 ↪best_params_['columntransformer__tfidfvectorizer-2__max_features']) # best␣
 ↪word limit is 3000
```

```
Best alpha: 1e-05
Best word limit for reviews tf idf vectorizer: 5000
```

[108]:
```
results = experimental_model.cv_results_

# Extract data
alphas = np.array(results['param_lasso__alpha'].data, dtype=float)
max_features = np.
 ↪array(results['param_columntransformer__tfidfvectorizer-2__max_features'].
 ↪data, dtype=int)
val_mse = -np.array(results['mean_test_score'])  # Convert to positive MSE

log_alphas = np.log10(alphas)

# Prepare a grid of (log_alpha, max_features)
unique_log_alphas = np.unique(log_alphas)
unique_max_features = np.unique(max_features)

# Create a grid for the MSE values
mse_grid = np.zeros((len(unique_max_features), len(unique_log_alphas)))

# Fill the grid with mean MSE for each (alpha, max_features) combination
for i, alpha in enumerate(unique_log_alphas):
    for j, feature in enumerate(unique_max_features):
        mask = (log_alphas == alpha) & (max_features == feature)
        mse_grid[j, i] = np.mean(val_mse[mask])

# Create the surface plot
fig = go.Figure(data=[
    go.Surface(
        z=mse_grid,
        x=unique_log_alphas,  # log10(alpha)
        y=unique_max_features,  # max_features
        colorscale='Viridis',
        showscale=True,
        colorbar=dict(title='Validation MSE')
    )
])

fig.update_layout(
    title='Validation MSE Surface',
```

```
    scene=dict(
        xaxis=dict(title='log10(Alpha)'),
        yaxis=dict(title='Max Features'),
        zaxis=dict(title='Validation MSE')
    ),
    margin=dict(l=0, r=0, t=40, b=0)
)

#fig.show()

# Save to html file
fig.write_html('final_model_val_mse_surface_plot.html', include_plotlyjs =␣
 ↪'cdn')
```