Predicting Missing Ratings

Name: Aaron Rasin

Website Link: https://aaron-m-r.github.io/PredictingRecipeRatings/ (https://aaron-m-r.github.io/PredictingRecipeRatings/)

Code

In [1]:
```
import pandas as pd
import numpy as np
import os
from pathlib import Path
import nltk
import string
import re
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import Binarizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import GridSearchCV

import plotly.express as px
pd.options.plotting.backend = 'plotly'
```

Framing the Problem

Goal: In this project, we would like to predict the ratings for reviews that are missing a rating score.

Cleaning the Data

We're copying and pasting our data cleaning code from project 3. However, we're adding a custom aggregator to our groupby in order to binarize plurality of number of reviews, and to put reviews in lists in order to analyze their sentiment later.

In [2]:
```
# Read in recipe and interactions data
reviews_path = Path('food_data') / 'RAW_interactions.csv'
recipes_path = Path('food_data') / 'RAW_recipes.csv'
raw_reviews = pd.read_csv(reviews_path)[['recipe_id', 'rating', 'review']]
raw_recipes = pd.read_csv(recipes_path)[['name', 'id', 'minutes', 'nutrition', 'n_steps
```

In [3]:
```
# Merge recipes with interactions
recipes = raw_recipes.merge(raw_reviews, left_on='id', right_on='recipe_id', how='left'
recipes = recipes.rename(columns = {'submitted': 'recipe_date', 'date': 'review_date'})
recipes = recipes.assign(bin_reviews = recipes.review)

# Replace 0 with null value
# (rating of 0 represents lack of a rating, not a 0 out of 5, so we shouldn't average th
recipes = recipes.replace(0, np.nan)

# Calculate average rating, number of ratings, and list of reviews per recipe
columns_of_interest = ['minutes', 'nutrition', 'n_steps', 'n_ingredients', 'bin_reviews
recipes = recipes.groupby('id')[columns_of_interest].agg({'minutes': lambda x: x.iloc[0
                                                          'nutrition': lambda x: x.iloc
                                                          'n_steps': lambda x: x.iloc[0
                                                          'n_ingredients': lambda x: x.
                                                          'bin_reviews': lambda x: 0 if
                                                          'rating': np.mean,
                                                          'review': lambda x: list(x)})
```

In [4]:
```
# Convert any strings to lists if necessary
def str2list(lst):
    lst = lst.strip('][').split(', ')
    return [item.replace("'", "") for item in lst]

for column in ['nutrition']:
    recipes[column] = recipes[column].apply(str2list)

# Define a list of nutrition facts
nutrition_facts = ['calories', 'total fat (PDV)', 'sugar (PDV)', 'sodium (PDV)', 'protei

# Split nutrition list into 7 nutrition columns
recipes[nutrition_facts] = pd.DataFrame(recipes.nutrition.tolist(), index= recipes.index

# Convert nutrition facts from strings to flaots
for fact in nutrition_facts:
    recipes[fact] = recipes[fact].astype(float)
```

In [5]:
```
# Function for analyzing review sentiment
def sent_analyze(review_list):
    sent_list = [sentiment.polarity_scores(re.sub(r'[^\w ]', '', text).lower())['compoun
    if len(sent_list)==0:
        return 0
    return np.mean(sent_list)
```

In [6]:
```
# Instantiating sentiment intensity analyzer
sentiment = SentimentIntensityAnalyzer()

# Creating data to use for model training and testing
final = recipes[['minutes', 'n_steps', 'n_ingredients', 'bin_reviews', 'review', 'rating
final = final.dropna()
final = final.assign(review_sentiment=final['review'].apply(sent_analyze))
```

In [7]:
```
final.head()
```

Out[7]:

id	minutes	n_steps	n_ingredients	bin_reviews	review	rating	calories	total fat (PDV)	sugar (PDV)	review_sentiment
275022	50.0	11	7	1	[Easy comfort food! I definitely thought it wa...	3.0	386.1	34.0	7.0	0.551233
275024	55.0	6	8	0	[When I found myself needing a dessert and hav...	3.0	377.1	18.0	208.0	0.913900
275026	45.0	7	9	1	[Sorry, this one didn't work out so well....	3.0	326.6	30.0	12.0	0.864100
275030	45.0	11	9	1	[This was the first cheesecake I'd ever made. ...	5.0	577.7	53.0	149.0	0.927200
275032	25.0	8	9	0	[This needs at least 10 stars. The recipe was...	5.0	386.9	0.0	347.0	0.946800

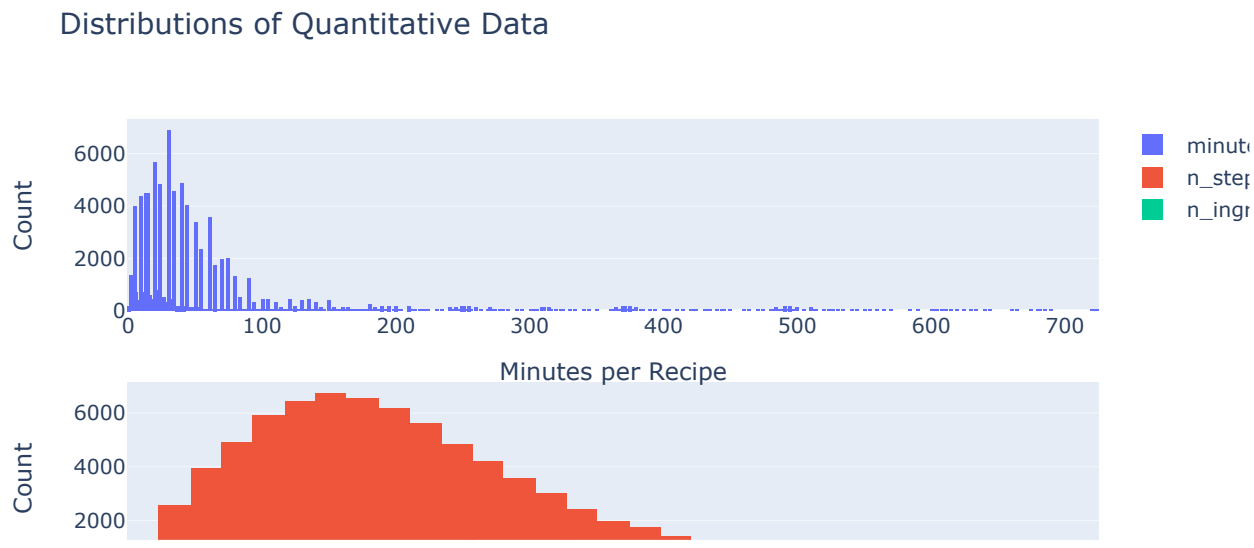
Baseline Model

Regression Modeling

In [8]:
```
# Store quantitative data and remove outliers (top half of a percent percentile) for the
quant = recipes[['minutes', 'n_steps', 'n_ingredients', 'rating']].dropna()
vis = quant[(quant['minutes']<=np.percentile(quant['minutes'], 99)) &
                (quant['n_steps']<=np.percentile(quant['n_steps'], 99)) &
                (quant['n_ingredients']<=np.percentile(quant['n_ingredients'], 99))]
```

Pipeline and Transformations

In [9]:
```
# Original Data
fig = make_subplots(rows=3, cols=1)

fig.add_trace(
    go.Histogram(name='minutes', x=vis['minutes']),
    row=1, col=1
)

fig.add_trace(
    go.Histogram(name='n_steps', x=vis['n_steps']),
    row=2, col=1
)

fig.add_trace(
    go.Histogram(name='n_ingredients', x=vis['n_ingredients']),
    row=3, col=1
)

fig.update_xaxes(title_text="Minutes per Recipe", row=1, col=1)
fig.update_xaxes(title_text="Steps per Recipe", row=2, col=1)
fig.update_xaxes(title_text="Ingredients per Recipe", row=3, col=1)

fig.update_yaxes(title_text="Count", row=1, col=1)
fig.update_yaxes(title_text="Count", row=2, col=1)
fig.update_yaxes(title_text="Count", row=3, col=1)

fig.update_layout(height=600, width=800, title_text="Distributions of Quantitative Data"
fig.write_html('quant_dists.html', include_plotlyjs='cdn')
fig.show()
```



In [10]:
```
# Log Transformed Data
fig = make_subplots(rows=3, cols=1)

fig.add_trace(
    go.Histogram(name='minutes', x=vis['minutes'].apply(np.log1p)),
    row=1, col=1
)

fig.add_trace(
    go.Histogram(name='n_steps', x=vis['n_steps'].apply(np.log1p)),
    row=2, col=1
)

fig.add_trace(
    go.Histogram(name='n_ingredients', x=vis['n_ingredients'].apply(np.log1p)),
    row=3, col=1
)

fig.update_xaxes(title_text="Minutes per Recipe", row=1, col=1)
fig.update_xaxes(title_text="Steps per Recipe", row=2, col=1)
fig.update_xaxes(title_text="Ingredients per Recipe", row=3, col=1)

fig.update_yaxes(title_text="Count", row=1, col=1)
fig.update_yaxes(title_text="Count", row=2, col=1)
fig.update_yaxes(title_text="Count", row=3, col=1)

fig.update_layout(height=600, width=800, title_text="Distributions of Log Transformed Qu
fig.write_html('log_quant_dists.html', include_plotlyjs='cdn')
fig.show()
```

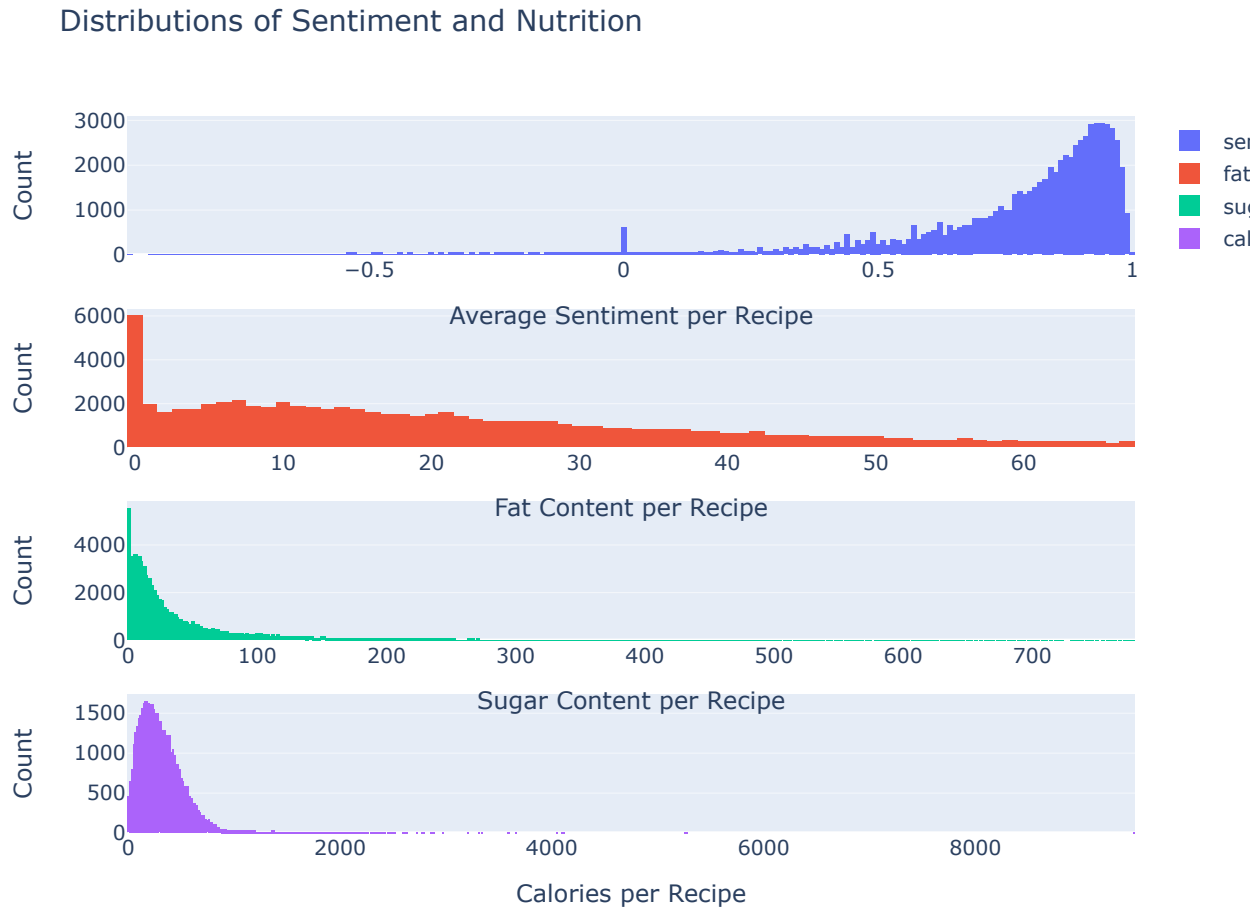
In [11]:
```
# Separate response variable and split all data into train and test sets
X, y = final.drop(columns=['rating', 'review']), final.rating
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Column transformer to take natural log of all numbers
col_transform = ColumnTransformer([
        ('log', FunctionTransformer(np.log1p), ['minutes', 'n_steps', 'n_ingredients'])])

# Create pipeline
pl = Pipeline([
        ('ct', col_transform),
        ('lr', RandomForestRegressor())])

# Fit pipeline model
pl.fit(X_train, y_train)

# Make and assess predictions
y_pred = pl.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_pred, y_test))
print(f"The root mean squared error is {rmse: .3f}.")
```

```
The root mean squared error is  0.693.
```

Final Model

In [12]:
```
# Remove outliers from nutrition for visualization purposes only
vis = final[(final['total fat (PDV)']<=np.percentile(final['total fat (PDV)'], 90)) &
                 (final['total fat (PDV)']<=np.percentile(final['total fat (PDV)'], 90)) &
                 (final['sugar (PDV)']<=np.percentile(final['calories'], 90))]
```

In [14]:
```
# Nutrition and Sentiment Distributions
fig = make_subplots(rows=4, cols=1)

fig.add_trace(
    go.Histogram(name='sentiment', x=vis['review_sentiment']),
    row=1, col=1
)

fig.add_trace(
    go.Histogram(name='fat', x=vis['total fat (PDV)']),
    row=2, col=1
)

fig.add_trace(
    go.Histogram(name='sugar', x=vis['sugar (PDV)']),
    row=3, col=1
)

fig.add_trace(
    go.Histogram(name='calories', x=vis['calories']),
    row=4, col=1
)

fig.update_xaxes(title_text="Average Sentiment per Recipe", row=1, col=1)
fig.update_xaxes(title_text="Fat Content per Recipe", row=2, col=1)
fig.update_xaxes(title_text="Sugar Content per Recipe", row=3, col=1)
fig.update_xaxes(title_text="Calories per Recipe", row=4, col=1)

fig.update_yaxes(title_text="Count", row=1, col=1)
fig.update_yaxes(title_text="Count", row=2, col=1)
fig.update_yaxes(title_text="Count", row=3, col=1)
fig.update_yaxes(title_text="Count", row=4, col=1)

fig.update_layout(height=600, width=800, title_text="Distributions of Sentiment and Nut
fig.write_html('nutrition.html', include_plotlyjs='cdn')
fig.show()
```



In [15]:
```
# Column transformer to take natural log of all numbers
col_transform = ColumnTransformer([
        ('log', FunctionTransformer(np.log1p), ['minutes', 'n_steps', 'n_ingredients',
    ])

# Create pipeline
pl = Pipeline([
        ('ct', col_transform),
        ('rf', RandomForestRegressor())])

# Fit pipeline model
pl.fit(X_train, y_train)

# Make and assess predictions
y_pred = pl.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_pred, y_test))
print(f"The root mean squared error is {rmse: .3f}.")
```

```
The root mean squared error is  0.606.
```

In [16]:
```
param_grid = {'rf__max_depth': [2, 3, 5, 7, 10],
              'rf__n_estimators': [10, 20, 50, 100]}

# Instantiating and fitting the GridSearchCV object
rf_cv = GridSearchCV(estimator=pl, param_grid=param_grid, cv = 5, scoring='neg_mean_squa
rf_cv.fit(X_train, y_train)
rf_cv.best_params_
```

Out[16]:
```
{'rf__max_depth': 5, 'rf__n_estimators': 50}
```

In [17]:
```
# Create pipeline with ideal parameters
pl = Pipeline([
        ('ct', col_transform),
        ('rf', RandomForestRegressor(max_depth=5, n_estimators=50))])

# Fit pipeline model
pl.fit(X_train, y_train)

# Make and assess predictions
y_pred = pl.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_pred, y_test))
print(f"The root mean squared error is {rmse: .3f}.")
```

```
The root mean squared error is  0.590.
```

Fairness Analysis

Null Hypothesis: Our model is fair and will predict ratings for recipes with 1 review equally as accurately as for recipes with multiple reviews.

Alternative Hypothesis: Our model is unfair, and predicts ratings for recipes with 1 review with less accuracy than for recipes with multipel reviews.

In [18]:
```
missing = recipes[recipes['rating'].isna()]['review'].apply(len).mean()
present = recipes[recipes['rating'].isna()==False]['review'].apply(len).mean()
print(f"The mean number of reviews for recipes with ratings is {present:.2f}, and withou
```

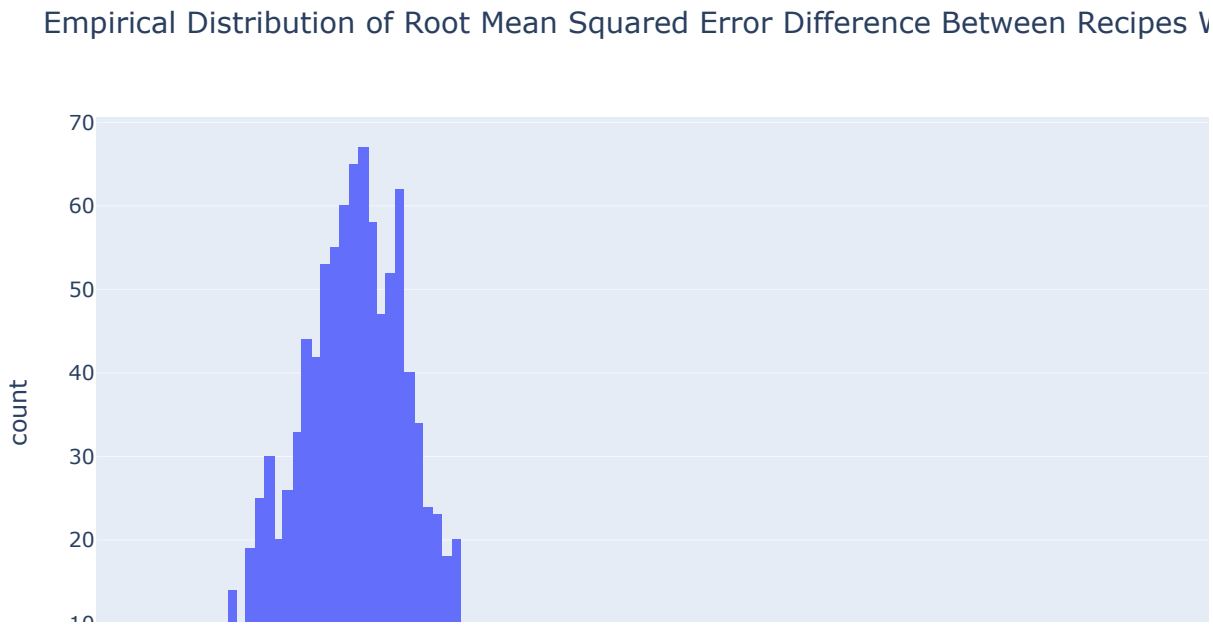
```
The mean number of reviews for recipes with ratings is 2.85, and without ratings is 1.
06
```

In [19]:
```
# Function to calculate the difference in accuracy of our model between recipes with 1 a
def prop_diff(df, model):

    # Split data into recipes with 1 review and more than 1 review
    X1 = df[df['bin_reviews'] == 0].drop(columns='bin_reviews')
    X2 = df[df['bin_reviews']== 1].drop(columns='bin_reviews')

    # Extract ratings
    y1 = X1.rating
    y2 = X2.rating

    # Remove rating column from design matrix
    X1 = X1.drop(columns='rating')
    X2 = X2.drop(columns='rating')

    # Predict ratings
    y1_pred = model.predict(X1)
    y2_pred = model.predict(X2)

    # Return the difference in scores
    rmse1 = np.sqrt(mean_squared_error(y1_pred, y1))
    rmse2 = np.sqrt(mean_squared_error(y2_pred, y2))
    return rmse1-rmse2
```

In [20]:
```
# Copy and concatenate the test set to keep original unshuffled
X_copy, y_copy = X_test, y_test
df = pd.concat([X_copy, y_copy], axis=1)

# Create empty array to store test statistics
results = np.array([])

# Compute and record observed test statistic
observed = prop_diff(df, pl)

for i in np.arange(1000):

    # Shuffle the response variable
    df['rating'] = np.random.permutation(df['rating'])

    # Compute and record test statistic
    stat = prop_diff(df, pl)
    results = np.append(results, stat)

pval = (observed<=results).mean()
print(f"The p-value of this permutation test is {pval: .2f}")
```

```
The p-value of this permutation test is  0.00
```

In [21]:
```
fig = px.histogram(results, x=0, nbins=50,
                   title='Empirical Distribution of Root Mean Squared Error Difference B
                   labels={'0': 'Difference in RMSE Score'})
fig.add_vline(x=observed, line_color='red', annotation_text='Observed Difference')
fig.write_html('permutation.html', include_plotlyjs='cdn')
fig.show()
```



In [ ]: