Your Title Here

Name(s): So Hirota

Website Link: https://soh09.github.io/Recipes-Dataset-Modeling/

Code

In [3]:
```
import pandas as pd
import numpy as np
import os

from sys import platform

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

import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
```

In [4]:
```
# reading in the datasets
mac = (platform == 'darwin')

if mac:
    interactions_fp = '/Users/hiro002/Documents/RAW_interactions.csv'
    recipes_fp = '/Users/hiro002/Documents/RAW_recipes.csv'
else:
    interactions_fp ='C:\\Users\\So\\Documents\\RAW_interactions.csv'
    recipes_fp ='C:\\Users\\So\\Documents\\RAW_recipes.csv'


og_interactions = pd.read_csv(interactions_fp)
og_recipes = pd.read_csv(recipes_fp)
```

In [5]:
```
print(og_interactions.shape)
og_interactions.head(3)#.loc[interactions['rating'] == 5].iloc[0].loc['review']
```

```
(731927, 5)
```

Out[5]:

	user_id	recipe_id	date	rating	review
0	1293707	40893	2011-12-21	5	So simple, so delicious! Great for chilly fall...
1	126440	85009	2010-02-27	5	I made the Mexican topping and took it to bunk...
2	57222	85009	2011-10-01	5	Made the cheddar bacon topping, adding a sprin...

In [6]:
```
print(og_recipes.shape)
og_recipes.head(3)
```

```
(83782, 12)
```

Out[6]:

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps	desc
0	1 brownies in the world best ever	333281	40	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	[138.4, 10.0, 50.0, 3.0, 3.0, 19.0, 6.0]	10	['heat the oven to 350f and arrange the rack i...	th th choc moi
1	1 in canada chocolate chip cookies	453467	45	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	[595.1, 46.0, 211.0, 22.0, 13.0, 51.0, 26.0]	12	['pre-heat oven the 350 degrees f', 'in a mixi...	thi reci we my
2	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	[194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0]	6	['preheat oven to 350 degrees', 'spray a 2 qua...	sinc are 411 for b

Framing the Problem

Question: Can we predict the rating of a recipe?

To answer this question,

1. I'll make a regression model
2. This regression model will predict the average rating of a certain recipe, given a few features from the recipes and interactions dataset.
3. This regression model will be evaluted using the **R^2** score. I chose this metric because R^2 is an easily interpretable metric that tells me the proportion (from 0.0 to 1.0) of variation in the data that the model is able to explain. 1.0 would means that model's predictions are identitcal to actual response variable.

Data Cleaning

- Recipes data
  1. Converting "fake lists" from strings to actual lists
  2. Breaking up the nutrition column into its individual components
  3. Dropping `name`, `contributor_id`, `steps`, `submitted`, and `description`. I'm keeping tags because maybe I could use them for feature engineering down the line.
  4. Adding the average ratings column, which is derived from the interactions data. This will be the response variable (y) for the model.
  5. Adding a column that contains all the review comments stored as a list for each recipe
- Interactions data
  1. Drop rows with nan in `review` column

In [7]:
```
# this code is adapted from part 1 of this project
recipes = og_recipes.copy()

# first convert columns into real lists (as is, they are strings that look like lists)
recipes['nutrition'] = recipes['nutrition'].transform(lambda x: x.str[1:-1].str.split(
recipes['tags'] = recipes['tags'].transform(lambda x: x.str[1:-1].str.split(', ')).app

# braek up the column into the individual components
for i, col in enumerate(['calories (#)', 'total fat (%)', 'sugar (%)', 'sodium (%)', '
    recipes[col] = recipes['nutrition'].transform(lambda x: x[i])
recipes = recipes.drop(columns = ['nutrition'])

#

# drop specified columns
recipes = recipes.drop(columns = ['name', 'contributor_id', 'steps', 'description', 'i
```

In [8]:
```
print(recipes.shape)
recipes.head(3)
```

```
(83782, 12)
```

Out[8]:

	id	minutes	tags	n_steps	n_ingredients	calories (#)	total fat (%)	sugar (%)	sodium (%)	protein (%)	sat fats (%)	c
0	333281	40	['60-minutes-or-less', 'time-to-make', 'course...	10	9	138.4	10.0	50.0	3.0	3.0	19.0	
1	453467	45	['60-minutes-or-less', 'time-to-make', 'cuisin...	12	11	595.1	46.0	211.0	22.0	13.0	51.0	
2	306168	40	['60-minutes-or-less', 'time-to-make', 'course...	6	9	194.8	20.0	6.0	32.0	22.0	36.0	

In [9]:
```
np.corrcoef([recipes['minutes'], recipes['n_ingredients'], recipes['n_steps'], recipes
```

Out[9]:
```
array([[ 1.        , -0.00820395,  0.00781249,  0.00453115],
       [-0.00820395,  1.        ,  0.43015751,  0.12430594],
       [ 0.00781249,  0.43015751,  1.        ,  0.14423246],
       [ 0.00453115,  0.12430594,  0.14423246,  1.        ]])
```

In [10]:
```
interactions = og_interactions.copy()

# dropping user_id and date column in interactions data
interactions = interactions.drop(columns = ['user_id','date'])

# dropping rows that have a rating of 0
interactions = interactions.loc[interactions['rating'] != 0]

# dropping rows with nan reviews
interactions = interactions.dropna(subset = ['review'])
```

In [11]:
```
from transformers import GPT2Tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# this function filters out most of reviews that are too long to perform sentiment anc
def filter_reviews(X):
    def get_okay_comments(comment):
        # print(comment)
        try:
            if len(tokenizer.encode(comment)) <= 128:
                return comment
            else:
                return np.nan
        except:
            return np.nan

    return X['review'].apply(get_okay_comments)
```

```
interactions['review'] = filter_reviews(interactions[['review']])

# this belwo snippet would help me read in the csv that has gone through this filter.
# This step was necessary because ths filter function would take about 10 minutes to r

# interactions = pd.read_csv('C:\\Users\\So\\Documents\\interactions_filtered.csv')
# interactions = interactions.drop(columns = ['Unnamed: 0'])
```

In [12]:
```
interactions
```

Out[12]:

	recipe_id	rating	review
0	79222	4	Oh, how wonderful! I doubled the crab, and ad...
1	79222	5	Along with the onions we added in a square of ...
2	79222	4	I made this last nite and it was pretty good. ...
3	79222	5	Delish! Part of my New Years resolution.... ...
4	79222	5	Fantastic! I used a little bit of cream since ...
...	...	...	...
375982	82303	5	5 stars for taste! I had a hard time getting m...
375983	82303	5	This was amazingly delicious! The only change...
375984	82303	1	I've improvised sauces that turned out be...
375985	82303	5	This is the best and easiest hot fudge ever. I...
375986	82303	5	Delicious quick thick chocolate sauce with ing...

375987 rows × 3 columns

In [13]:
```
# keeping only rows corresponding to recipes with 5 or more reviews in total
interactions = interactions.dropna(subset = ['review'])
num_recipes = interactions['recipe_id'].value_counts()
interactions = interactions.loc[interactions['recipe_id'].isin(num_recipes[num_recipes
print(interactions.shape)
interactions.head(3)
```

```
(375987, 3)
```

Out[13]:

	recipe_id	rating	review
0	79222	4	Oh, how wonderful! I doubled the crab, and ad...
1	79222	5	Along with the onions we added in a square of ...
2	79222	4	I made this last nite and it was pretty good. ...

In [14]:
```
# setting recipe index to id
recipes = recipes.set_index('id')

# adding average rating column to recipes
# average rating is our response variable (variable that we are trying to predict)
recipes = recipes.merge(interactions.groupby('recipe_id').mean(), how = 'inner', left_
```

```
print(recipes.shape)
recipes.head(3)
```

```
(9707, 12)
```

Out[14]:

	minutes	tags	n_steps	n_ingredients	calories (#)	total fat (%)	sugar (%)	sodium (%)	protein (%)	sat fats (%)	c
353171	75	['time-to-make', 'course', 'main-ingredient', ...	6	6	1582.6	88.0	402.0	27.0	96.0	156.0	
423875	5	['15-minutes-or-less', 'time-to-make', 'course...	2	4	94.7	0.0	70.0	0.0	2.0	0.0	
315537	40	['60-minutes-or-less', 'time-to-make', 'course...	8	5	377.8	48.0	15.0	51.0	29.0	96.0	

In [15]:

```
# add review column to recipes dataframe
reviews = interactions.groupby('recipe_id')['review'].agg(lambda x: list(x))
recipes = recipes.merge(reviews, how = 'left', left_index = True, right_index = True)
print(recipes.shape)
recipes.head(3)
```

```
(9707, 13)
```

Out[15]:

	minutes	tags	n_steps	n_ingredients	calories (#)	total fat (%)	sugar (%)	sodium (%)	protein (%)	sat fats (%)	c
353171	75	['time-to-make', 'course', 'main-ingredient', ...	6	6	1582.6	88.0	402.0	27.0	96.0	156.0	
423875	5	['15-minutes-or-less', 'time-to-make', 'course...	2	4	94.7	0.0	70.0	0.0	2.0	0.0	
315537	40	['60-minutes-or-less', 'time-to-make', 'course...	8	5	377.8	48.0	15.0	51.0	29.0	96.0	

Baseline Model

The mode will be as described below

- Features: 10 quantitative/numerical features
  1. minutes
  2. n_steps, n_ingredients
  3. all of the nutrition columns
- Response variable: the rating column
- No encoding is necessary at this stage, since all the input features will be numerical
- Transformation
  - I won't use any transformations at this point

R^2 score: close to 0

This word is a very poor model because the r^2 is 0. This means the model is unable to explain any of the variance present in the data. I think that this model has a very low R^2 score because the features that are fed into it are not correlated the rating very well. I will try to transform columns to create these relationships for the final model.

In [16]:
```
X = recipes[['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'total fat (%)', '
y = recipes['rating']

X_train, X_test, y_train, y_test, = train_test_split(X, y, test_size = 0.25, shuffle =

pl = Pipeline([
    ('lin_reg', LinearRegression())
    ])
```

```
pl.fit(X_train, y_train)
pl.score(X_test, y_test)
```

Out[16]: 0.0019820002870738485

Final Model

Findings and Possible Improvements

- The nutrition columns
    1. calories has high correlation with total fat, saturated fats, and carbs. Perhaps, if we include calories, saturated fats and carbs column are redundant.
    2. sodium has a low correlation with all columns
    3. sugar could also be included because it has relativly low correlation with other columns besides carbs
    4. To simplify the mode, we can probably drop the other nutrition labels
- Incorporating data from the interactions dataset
    1. Create two columns, n_good_review and n_bad_review
        - I will derive this feature by performing sentiment analysis on the reviews. The two aforementioned columns will contain the number of positive and negative reviews, respectively.

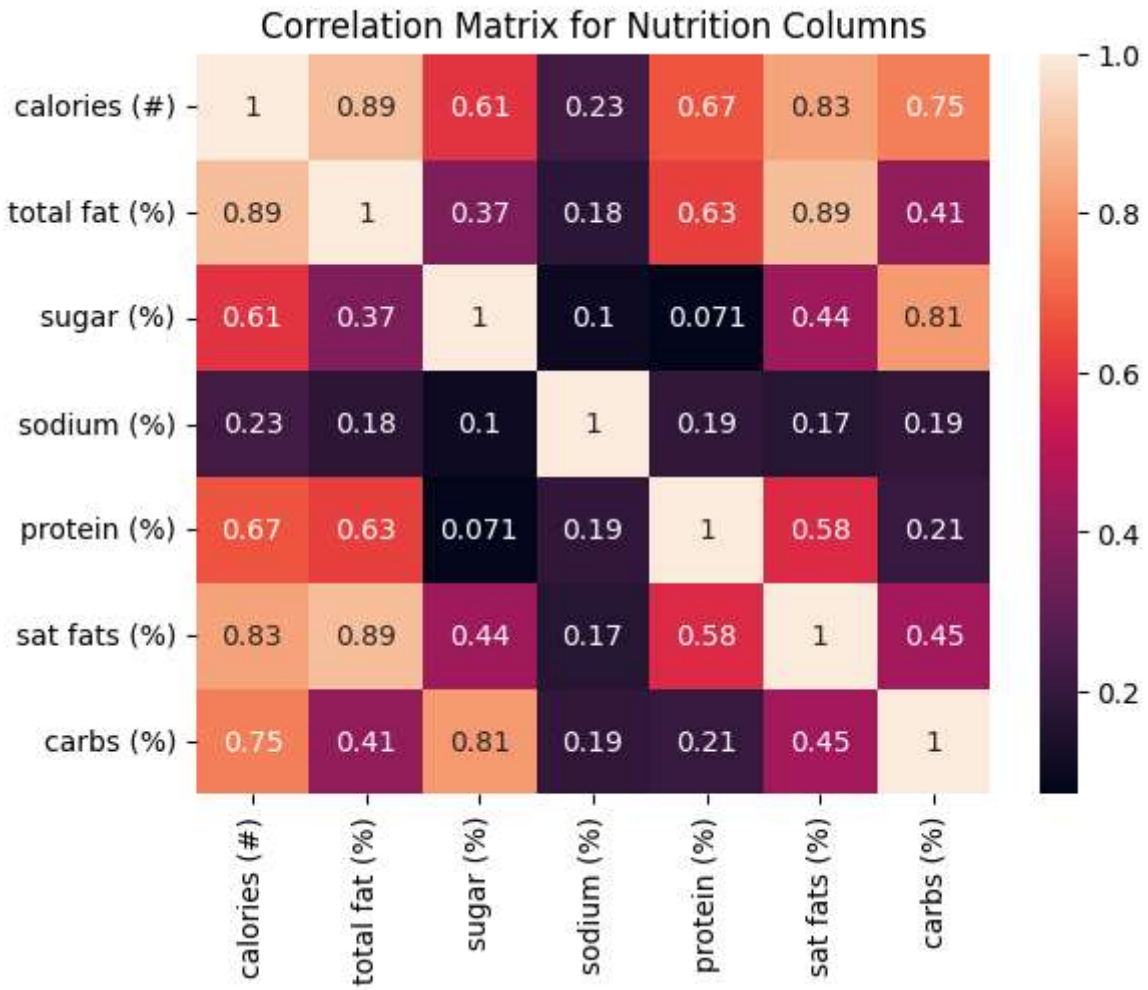
Models

1. Linear Regression model
    - Rationale: if there are any simple linear relationships that the model can pick up on, then the R^2 score could be pretty high. The sentiment analysis columns are good candidates for linear relationships.
    - Hyperparamters: combinations of standarizing minutes, n_ingredients, n_steps, and calories
2. K-Nearest Neighbor Regression model
    - Rationale: since rating is a categorical variable in some sense, then it could make sense to use a KNN algorithm to figure out the "clusters" (which are the rating categories, like 1 star, 2 stars, etc). This could maybe lead to accurate predictions.
    - hyperparamters: n_neighbors, p (how distance is calculated)
3. Decision Tree Regressor model
    - Rationale: Individuals giving the recipe a ratings is a decision that a user has to make, so perhaps a decision tree model will be able to capture that nuance well.
    - hyperparameters: depth, mininimum sample splits, scoring criterion

In [18]:
```
# from this heatmap, we can see that some of the columns are highly correlated to each
# by only including calories, sodium, and sugar, we can reduce the dimensionlity of th

corr_matrix = recipes[['calories (#)', 'total fat (%)', 'sugar (%)', 'sodium (%)', 'pr
sn.heatmap(corr_matrix, annot=True)
```

```
plt.title('Correlation Matrix for Nutrition Columns')
plt.show()
```



Finding the optimal model

Transformation #1: Converting `review` column into a list of length 3, where the 1st element is the number of positive reviews, 2nd element the number of neutral reviews, and the third element the number of negative reviews.

(This transformation must be done outside of the other transformations because it takes over an hour to run. I ran it once and saved it into a csv file, so I would not need to run it more than once.)

In [64]:
```
# I will use the bertweet sentiment analysis model for the analysis of the reviews
from transformers import pipeline
sentiment_task = pipeline(model = 'finiteautomata/bertweet-base-sentiment-analysis')

# I performed this transformation outside of the pipeline because this takes over an h
# it goes through all the comments and assigns it a sentiment: positve, neutral, or ne

def sentiment_analysis(X):
    # X is a dataframe that contains one column: the review comments
    def convert_to_sentiment(row):
        # global i
        # if (i % 100 == 0):
```

```
        #     print(i)
        # i += 1
        d = {
            'POS': 0,
            'NEG': 0,
            'NEU': 0
            }
        for comment in row:
            try:
                senti = sentiment_task(comment)
                # print(senti)
                d[senti[0]['label']] += 1

            except:
                print("EXCEPTION: token too long")
                pass
        return list(d.values())

    return pd.DataFrame(X['review'].apply(convert_to_sentiment))

sentimentize = ColumnTransformer(
    transformers = [
        ('convert_sentiment', FunctionTransformer(sentiment_analysis), ['review'])
    ],
    remainder = 'passthrough'
)

sentiment_recipes = sentimentize.fit_transform(recipes)
colnames = ['review' ,'minutes', 'tags', 'n_steps', 'n_ingredients', 'calories (#)',
sentiment_recipes = pd.DataFrame(sentiment_recipes, columns = colnames)
```

In [15]:
```
# code to read in sentiment csv and get it running
# this cell was used to read in the saved sentiment_recipes csv after the above transf
# it was necessary for this transformation to be performed outside of the unified pipe

# colnames = ['a', 'review' ,'minutes', 'tags', 'n_steps', 'n_ingredients', 'calories
# sentiment_recipes = pd.read_csv('C:\\Users\\So\\Documents\\recipes_sentiment.csv', r
# sentiment_recipes = sentiment_recipes.iloc[1:]
# sentiment_recipes['review'] = sentiment_recipes['review'].transform(lambda x: x.str[
# sentiment_recipes['review'] = sentiment_recipes['review'].apply(lambda x: np.array(x
# sentiment_recipes['tags'] = sentiment_recipes['tags'].transform(lambda x: x.str[1:-1
# sentiment_recipes = sentiment_recipes.drop(columns = ['a'])
```

Transformation #2: Split the list outputted from transformation #1 into three seperate columns, where each column holds the number of positive, neutral, and negative number of reviews for a given recipe.

In [20]:
```
def split_sentiment(X):
    # X is a dataframe
    X['pos'] = X['review'].apply(lambda x: x[0])
    X['neu'] = X['review'].apply(lambda x: x[1])
    X['neg'] = X['review'].apply(lambda x: x[2])
    return X[['pos', 'neu', 'neg']]
```

Transformation #3: Reduce certain columns to the power of n. Especially the `minutes` column has a large range with some very large outliers, so this transformation can effectively standardize the values by penalizing the especially large values more than the smaller values.

In [21]:
```
def reduce(X, n):
    return X**n
```

Final Pipeline for final model

In [35]:
```
col_trans = ColumnTransformer(
    transformers = [
        # ('standardize', StandardScaler(), list(comb)),
        ('split_sentiment', FunctionTransformer(split_sentiment), ['review']),
        ('root_4', FunctionTransformer(func = reduce, kw_args={'n': 0.25}), ['minutes'
    ],
    remainder = 'passthrough'
)

# I will find the optimal model using GridSearchCV
final_pipeline = Pipeline(
    [
        ('column_transformer', col_trans)
        # ('some_model', SomeMode())
    ]
)
```

Hyperparameter Tuning - Liner Regression, Decision Tree Regressor, KNN Regressor

Hyperparamter tuning for Liner Regression Model

- Since Liner Regression models don't have hyperparamters like the other models, I will try differnt combinations of columns to standardize.
- I picked `['minutes', 'n_steps', 'n_ingredients']` for standardization because these values have the most outliers, compared to other columns, and therefore think that standraizing would be the most beneficial.
- The function defined below take a list of columns and tries every combinations of transforming columns using the StandardScaler transformer, and returns a dictionary containing the R^2 score for each combination.

Conclusion

- Different combinations of standarized columns does not change the R^2 score of the linear model at all. This seems like an obvious conclusion because standardization is just a linear transformation, so it would just change the weight of the column, but would not actually change the prediction at all.

In [189…
```
from itertools import combinations
from sklearn.preprocessing import StandardScaler

# this function can test all the combinations of `columns` and returns a dictionary wi
def std_combinations(columns, X, y):

    cols = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiu
    X = sentiment_recipes[cols]
    y = sentiment_recipes['rating']
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_s
    # print(X_train)

    d = {}

    combs = []
    for i in range(1, (len(columns) + 1)):
        combs.append(list(combinations(columns, i)))

    # flattens the list
    combs = [item for comb in combs for item in comb]

    for comb in combs:

        std_cols = ColumnTransformer(
            transformers = [
                ('standardize', StandardScaler(), list(comb)),
                ('split_sentiment', FunctionTransformer(split_sentiment), ['review'])
            ],
            remainder = 'passthrough'
        )

        pl = Pipeline(
            [
                ('std', std_cols),
                ('lin_reg', LinearRegression())
            ]
        )

        pl.fit(X_train.copy(), y_train)
        d[comb] = pl.score(X_test.copy(), y_test)
    return d
```

In [190…

```
columns = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiu
X = sentiment_recipes[columns]
y = sentiment_recipes['rating']

std_combinations(['minutes', 'n_steps', 'n_ingredients'], X, y)
```

Out[190]:

```
{('minutes',): 0.3472597678655802,
 ('n_steps',): 0.34725976786557977,
 ('n_ingredients',): 0.3472597678655833,
 ('minutes', 'n_steps'): 0.3472597678655792,
 ('minutes', 'n_ingredients'): 0.34725976786558066,
 ('n_steps', 'n_ingredients'): 0.34725976786558654,
 ('minutes', 'n_steps', 'n_ingredients'): 0.34725976786558066}
```

Hyperparamter tuning for Decision Tree Regressor

- depth
- mininimum sample splits
- scoring criterion
- I chose these criteria because depth and minimum sample split can prevent overfitting, which is a common issue with decision trees. Scoring criterion can also influence the way the tree finds the optimal point to branch out, so I included it as a hyperparamter.

Conclusion

- {'criterion': 'friedman_mse', 'max_depth': 5, 'min_samples_split': 50}
- R^2: 0.5373138645841257
- This is the highest accuracy I have seen so far

In [23]:
```
# this cell performs a gridsearch for the best hyperpararmters for the decision tree r

columns = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiur
X = sentiment_recipes[columns]
y = sentiment_recipes['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_size

hyperparameters = {
    'max_depth': [3, 4, 5, 7, 10, 15, 18, 20, 22, 25, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10, 20, 50, 100, 200],
    'criterion': ['squared_error', 'friedman_mse', 'absolute_error', 'poisson']
}

seperate_sentiments = ColumnTransformer(
    transformers = [
        ('split_sentiment', FunctionTransformer(split_sentiment), ['review']),
        ('root_4', FunctionTransformer(func = reduce, kw_args={'n': 0.25}), ['minutes'
    ],
    remainder = 'passthrough'
)

X_train = seperate_sentiments.fit_transform(X_train)
X_test = seperate_sentiments.fit_transform(X_test)

searcher = GridSearchCV(DecisionTreeRegressor(), hyperparameters, cv = 5, n_jobs = -1)
searcher.fit(X_train, y_train)

print(searcher.score(X_test, y_test))
searcher.best_params_
```

Out[23]:
```
0.5373138645841257
{'criterion': 'friedman_mse', 'max_depth': 5, 'min_samples_split': 50}
```

Hyperparamter Tuning for KNN Regressor

- n_neighbors (the algorithm will find a point where n_neighbors are the closest)
- p (how distance is calculated)
- I chose these hyperparamters because n_neighbors will affect the ability for the algorithm to find "clusters". p changes the way distance is calculated between the points (euclidean vs manhattan distance), which also directly affects where the algorithm will make the prediction.

Result

- {'n_neighbors': 64, 'p': 1}
- R^2 score: 0.019942176024953295
- the KNN regressor is not a very good model for this prediction problem

In [33]:
```
# this cell performs a gridsearch for the best hyperpararmters for the nearest neighbo

columns = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiu
X = sentiment_recipes[columns]
y = sentiment_recipes['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_size


hyperparameters = {
    'n_neighbors': [i for i in range(1, 200)],
    'p': [1, 2]
}

seperate_sentiments = ColumnTransformer(
    transformers = [
        ('split_sentiment', FunctionTransformer(split_sentiment), ['review']),
        ('root_4', FunctionTransformer(func = reduce, kw_args={'n': 0.25}), ['minutes'
    ],
    remainder = 'passthrough'
)

X_train = seperate_sentiments.fit_transform(X_train)
X_test = seperate_sentiments.fit_transform(X_test)

searcher = GridSearchCV(KNeighborsRegressor(), hyperparameters, cv = 5, n_jobs = -1)
searcher.fit(X_train, y_train)

print(searcher.score(X_test, y_test))
searcher.best_params_
```

Out[33]:
```
0.019942176024953295
{'n_neighbors': 64, 'p': 1}
```

Final Model - Decision Tree Regressor

Hyperparamters

- scoring criterion: friedman mean squared error

- max depth: 5
- minimum sample split: 50
- R^2 score: 0.5373138645841267

In [135…

```
columns = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiu
X = sentiment_recipes[columns]
y = sentiment_recipes['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, test_size

col_trans = ColumnTransformer(
    transformers = [
        ('split_sentiment', FunctionTransformer(split_sentiment), ['review']),
        ('root_4', FunctionTransformer(func = reduce, kw_args={'n': 0.25}), ['minutes'
    ],
    remainder = 'passthrough'
)

# pipeline to transform and fit the best decision tree mode l
final_model = Pipeline(
    [
        ('transformer', col_trans),
        ('dec_tree', DecisionTreeRegressor(criterion = 'friedman_mse', max_depth = 5,
    ]
)

final_model.fit(X_train, y_train)
print(f"Final Model Score: {final_model.score(X_test, y_test)}")
```

```
Final Model Score: 0.5373138645841257
```

Fairness Analysis

Permutation Test Setup

I will perform a permtuation test to determine if the model can predict the average ratings of recipes that were published before year 2009 as well the average rating of recipes published after year 2009.

Null Hypothesis: Our model is fair. It's r^2 score for recipes published before and after 2009 are roughly the same, and any difference is due to random chance. Alternative Hypothesis: Our model is NOT fair. It's r^2 score is higher for recipes that were published after 2009 than the r^2 score for recipes published before 2009.

Cutoff date: 2009 Test statistic: r^2 score for recipes after 2009 - r^2 score for recipes before 2009 Significance leve: 0.05 Observed Statistic: 0.1420083050836909

Steps I will take to prepare data for this permutation test

1. Feed the model the recipes dataframe to get a prediction.
2. Then, I need to re-merge the date column, which I dropped during the data cleaning phase.
3. Then, I need to use the Binarizer transformer to transform the dates column to 0 for before 2009 and 1 for after 2009.

Permutation Test Result

With a p-value of 0.0, I will reject the null hypothesis. It is likely that our model is unfair, and has a higher r^2 score for predicting recipes that were submitted after 2009, compared to recipes published before 2009. We can infer from the p-value of 0 that the model is significantly better at predicting average ratings for recipes that were published afer 2009. A possible reason for this could be due to the sentiment classifer I used being trained on tweets from 2012 to 2019. Perhaps internet "lingo" has changed overtime, and so the sentiment classifer was able to make more accurate predictions about the sentiment of the review comment for recipes created after 2009 than comments from before 2009.

In [159…

```
columns = ['minutes', 'n_steps', 'n_ingredients', 'calories (#)', 'sugar (%)', 'sodiu
X = sentiment_recipes[columns]

y_pred = final_model.predict(X)

pt_df = sentiment_recipes[['rating']].reset_index()[['rating']]

# concatting the id back to dataframe so I can merge dates
pt_df['predicted'] = y_pred
pt_df = pd.concat([pt_df, recipes.reset_index()[['index']]], axis = 1).set_index('inde

# merging the submitted date and converting submitted to timestamps
pt_df = pt_df.merge(og_recipes[['id', 'submitted']], how = 'left', left_index = True,
pt_df['submitted'] = pd.to_datetime(pt_df['submitted'])

# creating the labels
pt_df['before_2009'] = pt_df['submitted'].dt.year <= 2009

pt_df = pt_df.drop(columns = ['id', 'submitted'])
pt_df.head(5)
```

Out[159]:

	rating	predicted	before_2009
35	4.400000	4.862173	True
41	4.800000	4.881972	False
50	4.818182	4.553786	True
53	5.000000	4.367439	True
63	4.000000	4.612413	True

In [134…

```
# this function will give us the test statistic
from sklearn.metrics import r2_score

def r_squared_test_stat(shuffled):
    before = shuffled[shuffled['before_2009']]
    after = shuffled[~shuffled['before_2009']]

    return r2_score(after['rating'], after['predicted']) - r2_score(before['rating'],

# computing the observed statistic
obs_stat = r_squared_test_stat(pt_df)
print(f'Observed statistic: {r_squared_test_stat(pt_df)}')
```

```
Observed statistic: 0.14200830508368645
```

In [123…

```python
# function that takes in a df and N number of permutations
def permutation_test(df, N):
    perm = df.copy()
    test_stats = []
    for _ in range(N):
        perm['before_2009'] = np.random.permutation(perm['before_2009'])
        test_stats.append(r_squared_test_stat(perm))
    return test_stats
```

In [ ]:

```python
result = permutation_test(pt_df, 50000)
```

In [154…

```python
# plotting the results of the permutation test

fig = px.histogram(pd.DataFrame(result), x=0, nbins=50, histnorm='probability',
                   title='Empirical Distribution of Difference in R^2 Score')
fig.add_vline(x=obs_stat, line_color='red')
fig.add_annotation(text = f'<span style="color:red">Observed Stat = {round(obs_stat, 4
fig.add_annotation(text = f'<span style="color:blue">P value = {round(np.mean(result >

fig.update_layout(yaxis_range=[0, 0.2])
```