Diya'sVersion

March 21, 2024

1 Your Title Here

Name(s): Diya Lakhani

Website Link: https://djlakhani.github.io/Recipes-and-Reviews-Reserach/

[1]:
```
import pandas as pd
import numpy as np
from pathlib import Path

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

import re

from dsc80_utils import * # Feel free to uncomment and use this.
```

[2]:
```
import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook"
```

1.1 Step 1: Introduction

The question this project will address from Steps 1-4 is if health and diet based recipes are more convenient to make. The significance of this question is mentioned on the website.

1.2 Step 2: Data Cleaning and Exploratory Data Analysis

1.2.1 Data Cleaning

[3]:
```
#Loading the data
recipes = pd.read_csv('food_data/RAW_recipes.csv')
ratings = pd.read_csv('food_data/RAW_interactions.csv')
```

[4]:
```
print("Ratings columns", ratings.columns)
print("Ratings shape", ratings.shape)
print("Recipes columns", recipes.columns)
print("Recipes shape", recipes.shape)
```

```
Ratings columns Index(['user_id', 'recipe_id', 'date', 'rating', 'review'],
dtype='object')
```

```
Ratings shape (731927, 5)
Recipes columns Index(['name', 'id', 'minutes', 'contributor_id', 'submitted',
'tags',
       'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
       'n_ingredients'],
      dtype='object')
Recipes shape (83782, 12)
```

[5]:
```
#Individual
rat_copy = ratings.copy()
rat_copy['rating'].replace(0.0, float('nan'), inplace=True)
rec_copy = recipes.copy()
```

[6]:
```
#Data cleaning
merged = recipes.merge(ratings, left_on = 'id', right_on = 'recipe_id', how =␣
 ↪'left')
merged['rating'].replace(0.0, float('nan'), inplace=True)

# Computing the average rating
rating_avg = ratings.groupby('recipe_id').mean(). \
             drop(columns=['user_id']). \
             reset_index(). \
             rename(columns={'rating':'avg_rating'})

cleaned = merged.merge(rating_avg, left_on='recipe_id', right_on='recipe_id',␣
 ↪how='left')

cleaned.head()
```

[6]:

	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	412 broccoli casserole	306168	40	50969	…	
4	412 broccoli casserole	306168	40	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]
```

[7]:
```
#Indicating recipes that contain the term 'low'
#We will require this column for the hypothesis test
cleaned = cleaned.assign(is_low=cleaned['tags'].apply(lambda x : len(re.
 ↪findall('low-', x)) > 0))
```

[8]:
```
cleaned.head()
```

[8]:

	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	412 broccoli casserole	306168	40	50969	…	
4	412 broccoli casserole	306168	40	50969	…	

	rating	review	avg_rating	is_low
0	4.0	These were pretty good, but took forever to ba…	4.0	False
1	5.0	Originally I was gonna cut the recipe in half …	5.0	False
2	5.0	This was one of the best broccoli casseroles t…	5.0	False
3	5.0	I made this for my son's first birthday party …	5.0	False
4	5.0	Loved this. Be sure to completely thaw the br…	5.0	False

```
[5 rows x 19 columns]
```

[9]:
```
# Adding columns for the nutrition information
all_categories = {0:'calories', 1:'total_fat', 2:'sugar', \
                  3:'sodium', 4:'protein', 5:'saturated_fat', \
                  6:'carbohydrates'}

# Converts a series of lists into a dataframe
nutrition = cleaned['nutrition'].apply(lambda x : re.findall(r'([0-9\.]+)', x))
nutri_df = pd.DataFrame.from_dict(dict(zip(nutrition.index, nutrition.values))).
 ↪T
nutri_df.head()
```

[9]:

	0	1	2	3	4	5	6
0	138.4	10.0	50.0	3.0	3.0	19.0	6.0
1	595.1	46.0	211.0	22.0	13.0	51.0	26.0
2	194.8	20.0	6.0	32.0	22.0	36.0	3.0
3	194.8	20.0	6.0	32.0	22.0	36.0	3.0
4	194.8	20.0	6.0	32.0	22.0	36.0	3.0

[10]:
```
# Converting to float and adding columns to the cleaned dataframe
nutri_df = nutri_df.astype(float).rename(columns=all_categories)
cleaned = cleaned.merge(nutri_df, left_index = True, right_index= True)
cleaned.head()
```

[10]:

	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	412 broccoli casserole	306168	40	50969	…	
4	412 broccoli casserole	306168	40	50969	…	

	sodium	protein	saturated_fat	carbohydrates
0	3.0	3.0	19.0	6.0
1	22.0	13.0	51.0	26.0
2	32.0	22.0	36.0	3.0
3	32.0	22.0	36.0	3.0
4	32.0	22.0	36.0	3.0

[5 rows x 26 columns]

[11]:
```
print(cleaned[['id', 'name', 'n_ingredients', 'avg_rating', 'calories',␣
 ↪'is_low', ]].head().to_markdown(index=False))
```

id	name	n_ingredients	avg_rating	calories	is_low
333281	1 brownies in the world best ever	9	4	138.4	False
453467	1 in canada chocolate chip cookies	11	5	595.1	False
306168	412 broccoli casserole	9	5	194.8	False
306168	412 broccoli casserole	9	5	194.8	False
306168	412 broccoli casserole	9	5	194.8	False

1.2.2 Univariate Analysis

[12]:
```
#COMMENT BEFORE SUBMISSION
#pio.renderers.default = "browser"
```

[13]:
```
fig = px.histogram(cleaned[cleaned['calories'] < 1000], x="calories",␣
 ↪title='Calories Dist.', color='is_low')

# fig.update_layout(
```

```
#     autosize=False,
#     width=1000,
#     height=1000,
# )

fig.show()
#fig.write_html('assets/calories-dist.html', include_plotlyjs='cdn')
```

The figure suggests that the calories distribution of reciped with and without a 'low-something' tag are quite similar.

[14]:
```
fig = px.histogram(cleaned, x="n_ingredients", title='Number of Ingrdients␣
↪Distribution', labels={'n_ingredients':'Number of ingredients'})

fig.show()
```

[15]:
```
fig = px.bar(cleaned.groupby('is_low').mean().reset_index(), x='is_low',␣
↪y=['n_ingredients', 'n_steps'], \
             labels={'is_low':'Health and Diet Recipe', 'n_ingredients':'Number␣
↪of Ingredients'})

#fig.write_html('assets/ingre-steps.html', include_plotlyjs='cdn')

fig.show()
```

[16]:
```
cleaned.columns
```

[16]:
```
Index(['name', 'id', 'minutes', 'contributor_id', 'submitted', 'tags',
       'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
       'n_ingredients', 'user_id', 'recipe_id', 'date', 'rating', 'review',
       'avg_rating', 'is_low', 'calories', 'total_fat', 'sugar', 'sodium',
       'protein', 'saturated_fat', 'carbohydrates'],
      dtype='object')
```

1.2.3 Bivariate Analysis

[17]:
```
fig = px.box(cleaned, x="is_low", y="n_steps", labels={'is_low':'Health and␣
↪Diet Recipe', 'n_ingredients':'Number of Ingredients'})

fig.update_layout(
    autosize=False,
    width=1000,
    height=600,
)

#fig.write_html('assets/steps.html', include_plotlyjs='cdn')
fig.show()
```

[18]:
```
fig = px.scatter(cleaned[cleaned['minutes'] < 180], x="minutes", y="calories",␣
 ↪title='Minutes v/s Calories')
#fig.write_html('assets/calorie-min.html', include_plotlyjs='cdn')
fig.show()
```

1.2.4 Interesting Aggregates

[19]:
```
# Examine the distribution of calories across the average rating and whether␣
 ↪the recipe is a health and diet recipe
#cleaned['avg_rating'].dropna().apply(lambda x : int(x // 1))
agg_df = cleaned[~cleaned['avg_rating'].isna()].assign(rating_cate =␣
 ↪cleaned['avg_rating'].dropna(). \
                                    apply(lambda x : int(x // 1))).
 ↪pivot_table(index='rating_cate',
                                    columns='is_low', values='calories',␣
 ↪aggfunc='mean')
agg_df
#print(agg_df.reset_index().to_markdown(index=False))
```

[19]:

is_low	False	True
rating_cate		
0	522.54	493.99
1	459.96	518.00
2	498.09	455.56
3	489.97	482.29
4	408.08	370.19
5	437.03	392.85

1.3 Step 3: Assessment of Missingness

To begin, we need to find the columns that contain missing values.

[20]:
```
cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 234429 entries, 0 to 234428
Data columns (total 26 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            234428 non-null  object
 1   id              234429 non-null  int64
 2   minutes         234429 non-null  int64
 3   contributor_id  234429 non-null  int64
 4   submitted       234429 non-null  object
 5   tags            234429 non-null  object
 6   nutrition       234429 non-null  object
 7   n_steps         234429 non-null  int64
 8   steps           234429 non-null  object
```

```
 9   description     234315 non-null  object
 10  ingredients     234429 non-null  object
 11  n_ingredients   234429 non-null  int64
 12  user_id         234428 non-null  float64
 13  recipe_id       234428 non-null  float64
 14  date            234428 non-null  object
 15  rating          219393 non-null  float64
 16  review          234371 non-null  object
 17  avg_rating      234428 non-null  float64
 18  is_low          234429 non-null  bool
 19  calories        234429 non-null  float64
 20  total_fat       234429 non-null  float64
 21  sugar           234429 non-null  float64
 22  sodium          234429 non-null  float64
 23  protein         234429 non-null  float64
 24  saturated_fat   234429 non-null  float64
 25  carbohydrates   234429 non-null  float64
dtypes: bool(1), float64(11), int64(5), object(9)
memory usage: 54.8+ MB
```

The three columns that contain missing values are rating, review, and description. We will analyze the missing values in the 'rating' column.

[21]:
```
#To check if the rating entry is NaN
cleaned['has_rating'] = cleaned['rating'].isna()
```

1.3.1 Column that missingness is dependent

[22]:
```
#Comparing against number of ingredients #CHANGE TO USE UNMERGED DATASET
fig = px.histogram(cleaned[['n_ingredients', 'has_rating', 'rating']],␣
 ↪x='n_ingredients', \
                   color='has_rating', histnorm='probability', marginal='box', \
                   title="Number of ingredients when rating is NaN",␣
 ↪barmode='overlay', opacity=0.5)
fig.show()
```

[23]:
```
#Simulating a permutation test to see if missingness in 'rating' column is␣
 ↪dependent on number of ingredients
miss_ingredients = cleaned[['n_ingredients', 'has_rating']]

#The chosen test statistic is absolute difference in group means
grouped_rating = cleaned[['n_ingredients', 'has_rating']].groupby('has_rating').
 ↪mean()
observed_stat = grouped_rating.diff().loc[True]['n_ingredients']
print("The observed statistic is:", observed_stat)

n_rep = 1000
test_stats = []
```

```
for i in range(n_rep):
    miss_ingredients['shuffle'] = np.random.
 ↪permutation(miss_ingredients['has_rating'])
    grouped_rating = miss_ingredients.groupby('shuffle').mean()
    test_stat = np.abs(grouped_rating.diff().loc[True]['n_ingredients'])
    test_stats.append(test_stat)

print("The p-value is:", np.mean(test_stats >= observed_stat))
```

```
The observed statistic is: 0.1607379066254797
The p-value is: 0.0
```

A p-value of $0.0 < 0.05$ implies that the number of ratings is MAR with respect to the number of ingredients.

[24]:
```
fig = px.histogram(test_stats, title='Test Statistic Distribution')
fig.update_xaxes(range=[-0.05, 0.2])
fig.add_vline(x=observed_stat, line_width=3, line_dash="dash",␣
 ↪line_color="green", opacity=1)
#fig.write_html('assets/perm-test.html', include_plotlyjs='cdn')
fig.show()
```

1.3.2 Column that missingness is not dependent

[25]:
```
#Simulating a permutation test to see if missingness in 'rating' column is␣
 ↪dependent on number of ingredients
miss_calories = cleaned[['minutes', 'has_rating']]

#The chosen test statistic is absolute difference in group means
grouped_rating = cleaned[['minutes', 'has_rating']].groupby('has_rating').mean()
observed_stat = grouped_rating.diff().loc[True]['minutes']
print("The observed statistic is:", observed_stat)

n_rep = 1000
test_stats = []

for i in range(n_rep):
    miss_calories['shuffle'] = np.random.
 ↪permutation(miss_calories['has_rating'])
    grouped_rating = miss_calories.groupby('shuffle').mean()
    test_stat = np.abs(grouped_rating.diff().loc[True]['minutes'])
    test_stats.append(test_stat)

print("The p-value is:", np.mean(test_stats >= observed_stat))
```

```
The observed statistic is: 51.45237039852127
The p-value is: 0.124
```

[26]:
```
fig = px.histogram(test_stats, title='Test Statistic Distribution')
fig.update_xaxes(range=[-1, 100])
fig.add_vline(x=observed_stat, line_width=3, line_dash="dash",␣
 ↪line_color="green", opacity=1)
fig.write_html('assets/perm-test2.html', include_plotlyjs='cdn')
fig.show()
```

Since the p-value is not lesser than 0.05, we fail to reject the null hypothesis which is that the missingness of the average ratings column depends on the minutes column.

1.4 Step 4: Hypothesis Testing

Question: Do health and diet recipes take lesser steps to make? For this question a health and diet recipe is defined as a recipe that contains a tag "low-(something)". The distribution of the different "low-something" tags is below.

[27]:
```
#low-tags
print(cleaned['tags']. \
      apply(lambda x : re.findall(r"low\-[aA-zZ]+", x)). \
      explode(). \
      value_counts(). \
      to_string())
print(pd.DataFrame(cleaned['tags']. \
      apply(lambda x : re.findall(r"low\-[aA-zZ]+", x)). \
      explode(). \
      value_counts()).reset_index().to_markdown(index=False))
```

```
low-in             88247
low-carb           44903
low-sodium         41317
low-cholesterol    38836
low-calorie        38473
low-protein        33914
low-saturated      32576
low-fat            22071
low-carbs          10135
low-cooker          7958
low-beans           1402
| index           |   tags |
|:----------------|-------:|
| low-in          |  88247 |
| low-carb        |  44903 |
| low-sodium      |  41317 |
| low-cholesterol |  38836 |
| low-calorie     |  38473 |
| low-protein     |  33914 |
| low-saturated   |  32576 |
| low-fat         |  22071 |
| low-carbs       |  10135 |
```

```
| low-cooker      |   7958 |
| low-beans       |   1402 |
```

Null Hypothesis: Health and diet recipes take the same number of steps as non-health and diet recipes. Alternative Hypothesis: Health and diet recipes take less number of steps to make than non-health and diet recipes. Test Statistic: Difference in group means.

[28]:
```
for_hypotest = cleaned.groupby('id').mean()
#groupby because cleaned dataframe has repeated recipes
for_hypotest = for_hypotest.assign(is_low = for_hypotest['is_low'].apply(lambda␣
 ↪x : x > 0)). \
               reset_index()[['id', 'n_steps', 'is_low']]
for_hypotest.head()
```

[28]:

	id	n_steps	is_low
0	275022	11.0	False
1	275024	6.0	False
2	275026	7.0	True
3	275030	11.0	False
4	275032	8.0	False

[29]:
```
observed = for_hypotest.groupby('is_low').mean().diff()['n_steps'][True]
# True - False
print ("The observed test statistic is:", observed)
```

```
The observed test statistic is: -1.381904964259407
```

[30]:
```
# Conducting a permutation test

n_rep = 1000
test_stats = []

for i in range(n_rep):
    for_hypotest['shuffle'] = np.random.permutation(for_hypotest['is_low'])
    test_stat = for_hypotest.groupby('shuffle').mean().diff()['n_steps'][True]
    test_stats.append(test_stat)

print("The p-value is:", np.sum(test_stats <= observed) / n_rep)
```

```
The p-value is: 0.0
```

[31]:
```
fig = px.histogram(test_stats)
fig.add_vline(x=-1.38, line_width=1.5, line_color="firebrick", opacity=1)
fig.update_layout(xaxis_range=[-1.5, 0.2])
fig.write_html('assets/hypo-test.html', include_plotlyjs='cdn')
fig.show()
```

1.5 Step 5: Framing a Prediction Problem

Prediction Problem: We will predict the number of calories in the recipe. I have chosen this question since it complements the hypothesis test that works with diet and health recipes. - This is a regression task. - The response variable is the number of calories.

1.6 Step 6: Baseline Model

[32]:
```
cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 234429 entries, 0 to 234428
Data columns (total 27 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            234428 non-null  object
 1   id              234429 non-null  int64
 2   minutes         234429 non-null  int64
 3   contributor_id  234429 non-null  int64
 4   submitted       234429 non-null  object
 5   tags            234429 non-null  object
 6   nutrition       234429 non-null  object
 7   n_steps         234429 non-null  int64
 8   steps           234429 non-null  object
 9   description     234315 non-null  object
 10  ingredients     234429 non-null  object
 11  n_ingredients   234429 non-null  int64
 12  user_id         234428 non-null  float64
 13  recipe_id       234428 non-null  float64
 14  date            234428 non-null  object
 15  rating          219393 non-null  float64
 16  review          234371 non-null  object
 17  avg_rating      234428 non-null  float64
 18  is_low          234429 non-null  bool
 19  calories        234429 non-null  float64
 20  total_fat       234429 non-null  float64
 21  sugar           234429 non-null  float64
 22  sodium          234429 non-null  float64
 23  protein         234429 non-null  float64
 24  saturated_fat   234429 non-null  float64
 25  carbohydrates   234429 non-null  float64
 26  has_rating      234429 non-null  bool
dtypes: bool(2), float64(11), int64(5), object(9)
memory usage: 55.0+ MB
```

For the baseline model we will attempt to predict the number of calories using the features 'is_low', 'sugar', 'total_fat', 'protein', and 'saturated_fat'. #EXPLAIN WHY

[33]:
```
# Importing required packages
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer, make_column_transformer,␣
 ↪make_column_selector
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, FunctionTransformer,␣
 ↪StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

[34]:
```
X = cleaned[['is_low', 'sugar', 'total_fat', 'carbohydrates', 'protein',
             'sodium', 'n_ingredients', 'saturated_fat', 'avg_rating']]
y = cleaned['calories']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)
```

[35]:
```
baseline_tranform = ColumnTransformer(
    transformers=[
        ('quanitative', FunctionTransformer(lambda x: x), ['sugar',␣
 ↪'carbohydrates', 'total_fat']),
        ('is_low', OneHotEncoder(drop='first'), ['is_low'])
    ],
    remainder='drop'
)
```

[36]:
```
pl = Pipeline([
    ('tranform', baseline_tranform),
    ('lin-reg', LinearRegression())
])
```

[37]:
```
#X_train
```

[38]:
```
pl.fit(X_train, y_train)
```

[38]:
```
Pipeline(steps=[('tranform',
                 ColumnTransformer(transformers=[('quanitative',
FunctionTransformer(func=<function <lambda> at 0x2c5761670>),
                                                  ['sugar', 'carbohydrates',
                                                   'total_fat']),
                                                 ('is_low',
                                                  OneHotEncoder(drop='first'),
                                                  ['is_low'])])),
                ('lin-reg', LinearRegression())])
```

[39]:
```
print("The coeffcients of the model are:", pl.named_steps['lin-reg'].coef_)
```

```
The coeffcients of the model are: [-0.21 13.32  6.61  9.65]
```

[40]:
```
# RMSE on training set
pred_train = pl.predict(X_train)
rmse_train = mean_squared_error(y_train, pred_train, squared=False)
rmse_train
```

[40]:
```
93.12890383163167
```

[41]:
```
# RMSE on test set
pred_test = pl.predict(X_test)
rmse_test = mean_squared_error(y_test, pred_test, squared=False)
rmse_test
```

[41]:
```
96.85226741944645
```

[42]:
```
print('The training set R2 is:', pl.score(X_train, y_train))
print('The test set R2 is:', pl.score(X_test, y_test))
```

```
The training set R2 is: 0.9743288400178577
The test set R2 is: 0.9731476686904615
```

The RMSE of the test set and the training set is quite high which suggests that perhaps we have not chosen suitable features to make our prediction. The RMSE of the training set is quite similar to that of the test set, which means the baseline model is able to generalize well to unseen data. Lastly, the R2 value for both datasets is close to 1, suggesting a good linear fit.

1.7 Step 7: Final Model

[43]:
```
cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 234429 entries, 0 to 234428
Data columns (total 27 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   name            234428 non-null  object
 1   id              234429 non-null  int64
 2   minutes         234429 non-null  int64
 3   contributor_id  234429 non-null  int64
 4   submitted       234429 non-null  object
 5   tags            234429 non-null  object
 6   nutrition       234429 non-null  object
 7   n_steps         234429 non-null  int64
 8   steps           234429 non-null  object
 9   description     234315 non-null  object
 10  ingredients     234429 non-null  object
 11  n_ingredients   234429 non-null  int64
 12  user_id         234428 non-null  float64
 13  recipe_id       234428 non-null  float64
 14  date            234428 non-null  object
```

```
 15  rating          219393 non-null  float64
 16  review          234371 non-null  object
 17  avg_rating      234428 non-null  float64
 18  is_low          234429 non-null  bool
 19  calories        234429 non-null  float64
 20  total_fat       234429 non-null  float64
 21  sugar           234429 non-null  float64
 22  sodium          234429 non-null  float64
 23  protein         234429 non-null  float64
 24  saturated_fat   234429 non-null  float64
 25  carbohydrates   234429 non-null  float64
 26  has_rating      234429 non-null  bool
dtypes: bool(2), float64(11), int64(5), object(9)
memory usage: 55.0+ MB
```

[44]:
```
pipes = {
    'low + sugar + total + carbs' : make_pipeline(
        make_column_transformer(
            (FunctionTransformer(lambda x: x), ['sugar', 'carbohydrates',␣
 ↪'total_fat']),
            (OneHotEncoder(drop='first'), ['is_low']),
        ),
        LinearRegression(),
    ),
    'low + sugar + total + carbs + pro': make_pipeline(
        make_column_transformer(
            (FunctionTransformer(lambda x: x), ['sugar', 'carbohydrates',␣
 ↪'total_fat', 'protein']),
            (OneHotEncoder(drop='first'), ['is_low']),
        ),
        LinearRegression(),
    ),
    'low + sugar + total + carbs + pro3': make_pipeline(
        make_column_transformer(
            (FunctionTransformer(lambda x: x), ['sugar', 'carbohydrates',␣
 ↪'total_fat']),
            (FunctionTransformer(lambda x: x ** 3), ['protein']),
            (OneHotEncoder(drop='first'), ['is_low']),
        ),
        LinearRegression(),
    ),
    'low + sugar + total + carbs + ingre + pro + sodium + satfat':␣
 ↪make_pipeline(
        make_column_transformer(
            (StandardScaler(), ['sugar', 'carbohydrates', 'total_fat',␣
 ↪'n_ingredients', \
                                'protein', 'saturated_fat', 'sodium']),
```

```
            (OneHotEncoder(drop='first'), ['is_low']),
        ),
        LinearRegression(),
    ),
    'low + sugar + total + carbs + pro + sodium + satfat' : make_pipeline(
        make_column_transformer(
            (FunctionTransformer(lambda x: x), ['sugar', 'carbohydrates',␣
 ↪'total_fat', \
                                                'protein', 'saturated_fat',␣
 ↪'sodium']),
            (OneHotEncoder(drop='first'), ['is_low']),
        ),
        LinearRegression(),
    )
}
```

[ ]:
```
px.scatter(cleaned, x=cleaned['protein'], y=cleaned['calories'], title='Protein␣
 ↪v/s Calories')
```

[47]:
```
pipe_df = pd.DataFrame()

for pipe in pipes:
    errs = cross_val_score(pipes[pipe], X_train, y_train,
                           cv=5, scoring='neg_root_mean_squared_error')
    pipe_df[pipe] = -errs

pipe_df.index = [f'Fold {i}' for i in range(1, 6)]
pipe_df.index.name = 'Validation Fold'
```

[48]:
```
pipe_df
```

[48]:

Validation Fold	low + sugar + total + carbs	low + sugar + total + carbs + pro
Fold 1	93.76	38.68
Fold 2	86.54	42.35
Fold 3	98.48	41.64
Fold 4	86.29	33.66
Fold 5	101.19	31.38

```
                 low + sugar + total + carbs + pro3  \
Validation Fold
Fold 1                                        88.77
Fold 2                                        84.68
Fold 3                                        92.39
Fold 4                                        83.24
Fold 5                                        84.06

                 low + sugar + total + carbs + ingre + pro + sodium + satfat  \
Validation Fold
Fold 1                                                              38.67
Fold 2                                                              42.31
Fold 3                                                              41.59
Fold 4                                                              33.63
Fold 5                                                              31.40

                 low + sugar + total + carbs + pro + sodium + satfat
Validation Fold
Fold 1                                                          38.70
Fold 2                                                          42.35
Fold 3                                                          41.64
Fold 4                                                          33.67
Fold 5                                                          31.42
```

[49]:
```
pipe_df.mean()
```

[49]:
```
low + sugar + total + carbs                                  93.25
low + sugar + total + carbs + pro                            37.54
low + sugar + total + carbs + pro3                           86.63
low + sugar + total + carbs + ingre + pro + sodium + satfat  37.52
low + sugar + total + carbs + pro + sodium + satfat          37.56
dtype: float64
```

[50]:
```
pipe_df.mean().idxmin()
```

[50]:
```
'low + sugar + total + carbs + ingre + pro + sodium + satfat'
```

While including more nutrition columns did decrease our RMSE it is evident that the number of ingredients did not significantly impact this number. For this reason, although 'low + sugar + total + carbs + ingre + pro + sodium + satfat' is proven to be the best combination, we will consider 'low + sugar + total + carbs + pro' for simplicity since the RMSE value differs by 0.02.

[51]:
```
final_tranform = ColumnTransformer(
    transformers=[
        ('quanitative', FunctionTransformer(lambda x: x), ['sugar',␣
↪'carbohydrates', 'total_fat', 'protein']),
        ('is_low', OneHotEncoder(drop='first'), ['is_low'])
    ],
```

```
    remainder='drop'
)
```

[52]:
```
pl_final = Pipeline([
    ('tranform', final_tranform),
    ('lin-reg', LinearRegression())
])
```

[53]:
```
pl_final.fit(X_train, y_train)
```

[53]:
```
Pipeline(steps=[('tranform',
                 ColumnTransformer(transformers=[('quanitative',
FunctionTransformer(func=<function <lambda> at 0x2b81668b0>),
                                                  ['sugar', 'carbohydrates',
                                                   'total_fat', 'protein']),
                                                 ('is_low',
                                                  OneHotEncoder(drop='first'),
                                                  ['is_low'])])),
                ('lin-reg', LinearRegression())])
```

[54]:
```
print("The coeffcients of the model are:", pl_final.named_steps['lin-reg'].
↪coef_)
```

```
The coeffcients of the model are: [-0.   11.67  5.71  2.16 -1.27]
```

[55]:
```
# RMSE on training set
pred_train = pl_final.predict(X_train)
rmse_train = mean_squared_error(y_train, pred_train, squared=False)
rmse_train
```

[55]:
```
37.758171255859104
```

[56]:
```
# RMSE on test set
pred_test = pl_final.predict(X_test)
rmse_test = mean_squared_error(y_test, pred_test, squared=False)
rmse_test
```

[56]:
```
39.87494754942134
```

[57]:
```
print('The training set R2 is:', pl.score(X_train, y_train))
print('The test set R2 is:', pl.score(X_test, y_test))
```

```
The training set R2 is: 0.9743288400178577
The test set R2 is: 0.9731476686904615
```

1.8 Step 8: Fairness Analysis

For the fariness analysis I plan to compare two interesting groups which is groups of high rating and low rating. High rating is a group with a rating above 4.

[58]:
```
X_test
```

[58]:
```
        is_low   sugar  total_fat  carbohydrates  …  sodium  n_ingredients  \
36396     True   131.0        4.0           22.0  …     6.0              5
66906     True    21.0       17.0            3.0  …    30.0             10
213212    True    84.0       28.0            9.0  …     8.0              8
…          …     …         …             …  …     …           …
195166   False  2440.0      594.0          299.0  …   162.0              7
203130   False    69.0        1.0            9.0  …     3.0              4
99911    False    25.0       17.0            5.0  …     4.0              4

        saturated_fat  avg_rating
36396             8.0        5.00
66906            33.0        5.00
213212           56.0        3.00
…                 …          …
195166         1143.0        3.24
203130            2.0        4.43
99911            36.0        4.89

[46886 rows x 9 columns]
```

[59]:
```
y_test
```

[59]:
```
36396      326.1
66906      184.3
213212     281.0
           …
195166    7371.9
203130     128.6
99911      166.2
Name: calories, Length: 46886, dtype: float64
```

[60]:
```
fairness_df = X_test.assign(y_test=y_test, y_predic=pred_test)
```

[61]:
```
fairness_df = fairness_df.assign(high_rating=fairness_df['avg_rating'] >= 4)
```

[62]:
```
def compute_rmse(df):
    return mean_squared_error(df['y_test'], df['y_predic'], squared=False)
```

[63]:
```
observed_stat = fairness_df.groupby('high_rating').apply(compute_rmse).
 ↪diff()[True]
```

[64]:
```
n_rep = 1000
test_stats = []

for i in range(n_rep):
    fairness_df['shuffle'] = np.random.permutation(fairness_df['high_rating'])
```

```
    test_stat = fairness_df.groupby('shuffle').apply(compute_rmse).diff()[True]
    test_stats.append(test_stat)

print("The p-value is:", np.mean(test_stats >= observed_stat))
```

```
The p-value is: 0.309
```

[65]:
```
fairness_df.groupby('shuffle').apply(compute_rmse)
```

[65]:
```
shuffle
False    29.39
True     41.87
dtype: float64
```

[ ]:
```

```