A Data-Driven Approach to Estimating Calories in Recipes

Name(s): Keanu Ventura

Website Link: https://keanuventura.github.io/RecipeDataProject/

```
import pandas as pd
import numpy as np
from pathlib import Path

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

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

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
```

Step 1: Introduction

```
question = 'What is the relationship between the cooking time and
average rating of recipes?'
question
```

```
'What is the relationship between the cooking time and average rating
of recipes?'
```

Step 2: Data Cleaning and Exploratory Data Analysis

```
# Load and merge recipe and interaction data, clean and filter
relevant columns, and engineer time_bins and calorie features

recipes = pd.read_csv('RAW_recipes.csv')
interactions = pd.read_csv('interactions.csv')
merged = recipes.merge(interactions, left_on='id',
right_on='recipe_id', how='left').drop(columns='recipe_id')
merged['rating'] = merged['rating'].replace(0, np.nan)
merged['avg_rating'] = merged.groupby('id')
['rating'].transform('mean')
cleaned = merged[merged['minutes'] <= 500]
bin_edges = [0, 10, 20, 30, 40, 50, 60, 90, 120, 180, 240, 300, 400,
```

```
500]
bin_labels = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60',
'60-90', '90-120',
              '120-180', '180-240', '240-300', '300-400', '400-500']
cleaned['time_bins'] = pd.cut(cleaned['minutes'], bins=bin_edges,
labels=bin_labels, right=False)
cleaned['calories'] =
cleaned['nutrition'].str.strip('[]').str.split(',').str[0].astype(floa
t)
cleaned = cleaned[cleaned['calories'] <= 1000]
cleaned
```

```
                                                name      id  minutes
\
0                1 brownies in the world    best ever  333281       40

1                  1 in canada chocolate chip cookies  453467       45

2                                412 broccoli casserole  306168       40

...                                               ...     ...      ...

234426  cookies by design   sugar shortbread cookies  298509       20

234427  cookies by design   sugar shortbread cookies  298509       20

234428  cookies by design   sugar shortbread cookies  298509       20


        contributor_id  ...  \
0               985201  ...
1              1848091  ...
2                50969  ...
...                ...  ...
234426          506822  ...
234427          506822  ...
234428          506822  ...

                                                  review
avg_rating  \
0       These were pretty good, but took forever to ba...         4.0

1       Originally I was gonna cut the recipe in half ...         5.0

2       This was one of the best broccoli casseroles t...         5.0

...                                                   ...         ...

234426  This recipe tastes nothing like the Cookies by...         3.0

234427  yummy cookies, i love this recipe me and my sm...         3.0
```

```
234428  I work at a Cookies By Design and can say this...        3.0


       time_bins  calories
0          40-50     138.4
1          40-50     595.1
2          40-50     194.8
...          ...       ...
234426     20-30     174.9
234427     20-30     174.9
234428     20-30     174.9

[218885 rows x 19 columns]

# Plot a histogram showing the distribution of cooking times in
minutes for all recipes

plt.figure(figsize=(10,6))
sns.histplot(cleaned['minutes'], bins=50, kde=False, color='skyblue',
edgecolor='white')
plt.title('Distribution of Cooking Time (Minutes)', fontsize=18)
plt.xlabel('Cooking Time (Minutes)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()
```

```python
# Plot a histogram showing the distribution of average recipe ratings

plt.figure(figsize=(10,6))
sns.histplot(cleaned['avg_rating'], bins=30, kde=False,
color='skyblue', edgecolor='white')
plt.title('Distribution of Average Ratings', fontsize=18)
plt.xlabel('Average Rating', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()
```

```python
# Create a scatter plot to visualize the relationship between average
rating and cooking time using a random sample of 5000 recipes


sampled = cleaned.sample(n=5000, random_state=6)

plt.figure(figsize=(10,6))
sns.scatterplot(data=sampled, x='avg_rating', y='minutes', alpha=0.4,
color='skyblue', edgecolor=None)
plt.title('Scatter Plot: Average Rating vs. Cooking Time',
fontsize=18)
plt.xlabel('Average Rating', fontsize=14)
plt.ylabel('Cooking Time (Minutes)', fontsize=14)
plt.grid(True, linestyle='--', linewidth=0.5, alpha=0.7)
```

```
plt.tight_layout()
plt.show()
```

```
# Calculate and plot the average recipe rating for each cooking time
bin.


mean_ratings = cleaned.groupby('time_bins', observed=False)
['avg_rating'].mean().reset_index()

plt.figure(figsize=(12, 6))
sns.lineplot(data=mean_ratings, x='time_bins', y='avg_rating',
marker='o', color='skyblue', linewidth=2)

plt.title('Average Recipe Rating by Cooking Time Bins', fontsize=18)
plt.xlabel('Cooking Time (minutes)', fontsize=14)
plt.ylabel('Average Rating', fontsize=14)
plt.xticks(rotation=45)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
# Create a pivot table showing summary statistics of average ratings
for each cooking time bin.


cleaned.pivot_table(values='avg_rating',
                    index='time_bins',
                    aggfunc=['mean', 'median', 'min', 'max', 'count'],
                    observed=False)
```

```
                mean     median        min        max      count
          avg_rating avg_rating avg_rating avg_rating avg_rating
time_bins
0-10            4.71       4.89        1.0        5.0      20296
10-20           4.72       4.91        1.0        5.0      29686
20-30           4.68       4.86        1.0        5.0      34390
...              ...        ...        ...        ...        ...
240-300         4.58       4.69        1.0        5.0       2910
300-400         4.55       4.67        1.0        5.0       3712
400-500         4.57       4.73        1.0        5.0       2201

[13 rows x 5 columns]
```

```
# Compute summary statistics of average ratings grouped by exact
cooking time in minutes.


cleaned.groupby('minutes')['avg_rating'].agg(['mean', 'median', 'min',
'max', 'count'])
```

```
         mean  median  min  max  count
minutes
0        5.00    5.00  5.0  5.0      3
```

```
1        4.81    5.00  1.0  5.0    683
2        4.76    4.95  1.0  5.0   2668
...       ...     ...  ...  ...    ...
496      5.00    5.00  5.0  5.0      4
498      5.00    5.00  5.0  5.0      2
500      4.53    4.74  2.0  5.0    360

[295 rows x 5 columns]
```

Step 3: Assessment of Missingness

```
missingness_null_hypothesis_1 = 'The missingness of ratings does not depend on the user ID.'
missingness_alternative_hypothesis_1 = 'The missingness of ratings does depend on the user ID.'
missingness_test_statistic_1 = 'The variance of the proportions of missing ratings across different user IDs.'
print(missingness_null_hypothesis_1)
print(missingness_alternative_hypothesis_1)
print(missingness_test_statistic_1)
```

```
The missingness of ratings does not depend on the user ID.
The missingness of ratings does depend on the user ID.
The variance of the proportions of missing ratings across different user IDs.
```

```
# Define a permutation test function to assess if missingness in ratings depends on another column.

cleaned['missing_rating'] = cleaned['rating'].isna().astype(int)

def missingness_perm_test(df, col_to_test, missing_col='missing_rating', n_permutations=1000):

    df_with_two_cols = df[[col_to_test, missing_col]].dropna()

    missingness_proportions = df_with_two_cols.groupby(col_to_test, observed=False)[missing_col].mean()
    observed_variance = missingness_proportions.var()

    perm_variances = []

    for _ in range(n_permutations):

        shuffled_missing = np.random.permutation(df_with_two_cols[missing_col])

        df_with_two_cols_shuffled = df_with_two_cols.copy()
        df_with_two_cols_shuffled[missing_col] = shuffled_missing
```

```
        perm_missingness_proportions =
df_with_two_cols_shuffled.groupby(col_to_test, observed=False)
[missing_col].mean()

        perm_variances.append(perm_missingness_proportions.var())

    p_value = np.mean(np.array(perm_variances) >= observed_variance)


    return round(observed_variance, 4), round(p_value, 4)
```

```
observed_variance_1, p_value_1 = missingness_perm_test(cleaned,
'user_id', missing_col='missing_rating', n_permutations=1000)
observed_variance_1, p_value_1
```

```
(np.float64(0.1262), np.float64(0.0))
```

```
permutation_test_1_result = f'We reject the null, missingness in
rating DOES depend on user_id because p-value of {p_value_1} is less
than .05.'
permutation_test_1_result
```

```
'We reject the null, missingness in rating DOES depend on user_id
because p-value of 0.0 is less than .05.'
```

```
missingness_null_hypothesis_2 = 'The missingness of ratings does not
depend on the cooking time of the recipe in minutes.'
missingness_alternative_hypothesis_2 = 'The missingness of ratings
does depend on the cooking time of the recipe in minutes.'
missingness_test_statistic_2 = 'The variance of the proportions of
missing ratings across different cooking times.'
print(missingness_null_hypothesis_2)
print(missingness_alternative_hypothesis_2)
print(missingness_test_statistic_2)
```

```
The missingness of ratings does not depend on the cooking time of the
recipe in minutes.
The missingness of ratings does depend on the cooking time of the
recipe in minutes.
The variance of the proportions of missing ratings across different
cooking times.
```

```
observed_variance_2, p_value_2 = missingness_perm_test(cleaned,
'minutes', missing_col='missing_rating', n_permutations=1000)
observed_variance_2, p_value_2
```

```
(np.float64(0.0126), np.float64(0.343))
```

```
permutation_test_2_result = f'We fail to reject the null, missingness
in rating does NOT depend on cooking time because p-value of
{p_value_2} is greater than .05.'
permutation_test_2_result
```

```
'We fail to reject the null, missingness in rating does NOT depend on cooking time because p-value of 0.343 is greater than .05.'
```

Step 4: Hypothesis Testing

```python
null_hypothesis = 'There is no difference in average recipe ratings between a short cooking time(20-30 minutes bin) and a long cooking time(300-400 minutes bin).'
alternative_hypothesis = 'There is a difference in average recipe ratings between the two cooking times.'
test_statistic = 'The difference in means of average ratings between the two cooking time bins'
print(null_hypothesis)
print(alternative_hypothesis)
print(test_statistic)
```

```
There is no difference in average recipe ratings between a short cooking time(20-30 minutes bin) and a long cooking time(300-400 minutes bin).
There is a difference in average recipe ratings between the two cooking times.
The difference in means of average ratings between the two cooking time bins
```

```python
# Perform a permutation test comparing mean ratings between short and long cooking time groups.

short_cooking_time = cleaned.loc[cleaned['time_bins'] == '20-30', 'avg_rating'].dropna()
long_cooking_time = cleaned.loc[cleaned['time_bins'] == '300-400', 'avg_rating'].dropna()

hypothesis_observed_difference = short_cooking_time.mean() - long_cooking_time.mean()

combined = np.concatenate([short_cooking_time, long_cooking_time])

permutation_differences = []

for _ in range(1000):

    np.random.shuffle(combined)
    perm_short_cooking_time = combined[:len(short_cooking_time)]
    perm_long_cooking_time = combined[len(short_cooking_time):]

    perm_difference = perm_short_cooking_time.mean() - perm_long_cooking_time.mean()
    permutation_differences.append(perm_difference)

permutation_differences = np.array(permutation_differences)
```

```
hypothesis_p_value = np.mean(np.abs(permutation_differences) >=
np.abs(hypothesis_observed_difference))

hypothesis_observed_difference, hypothesis_p_value
```

```
(np.float64(0.12912984395712535), np.float64(0.0))
```

```
hypothesis_test_result = f'We reject the null, there is a difference
in average recipe ratings between the two cooking times because p-
value of {hypothesis_p_value} is less than .05.'
hypothesis_test_result
```

```
'We reject the null, there is a difference in average recipe ratings
between the two cooking times because p-value of 0.0 is less
than .05.'
```

Step 5: Framing a Prediction Problem

```
prediction_problem = 'Predict calories of recipes.'
type_of_problem = 'Regression'
evaluation_metric = 'R_squared'
print(prediction_problem)
print(type_of_problem)
print(evaluation_metric)
```

```
Predict calories of recipes.
Regression
R_squared
```

Step 6: Baseline Model

```
# Build and evaluate a baseline linear regression model predicting
calories from ingredients count and cooking time.


def rsquared(actual, pred):
    return np.var(pred) / np.var(actual)

def rmse(actual, pred):
    return np.sqrt(np.mean((actual - pred) ** 2))

two_features = ['n_ingredients', 'minutes']

preprocessor = ColumnTransformer(transformers=[
    ('num', 'passthrough', two_features)
])

baseline_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
```

```
    ('regressor', LinearRegression())
])

baseline_pipeline.fit(cleaned[two_features], cleaned['calories'])


baseline_model = baseline_pipeline.named_steps['regressor']
baseline_model

baseline_model_rsquared = rsquared(cleaned['calories'],
baseline_pipeline.predict(cleaned[two_features]))

print(f'rsquared: {round(baseline_model_rsquared, 4)}')
```

```
rsquared: 0.0856
```

Step 7: Final Model

```
# Tune and train a Random Forest model with three scaled features to
predict calories, then evaluate performance.

final_features = ['n_ingredients', 'minutes', 'n_steps']

preprocessor = ColumnTransformer(transformers=[
    ('n_ingredients_scaled', StandardScaler(), ['n_ingredients']),
    ('minutes_quantiled',
QuantileTransformer(output_distribution='normal'), ['minutes']),
    ('n_steps_scaled', StandardScaler(), ['n_steps'])
])

final_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor())
])

param_grid = {
    'regressor__max_depth': [5, 10, 15, 20, None],
    'regressor__n_estimators': [50, 100]
}

grid_search = GridSearchCV(final_pipeline, param_grid, cv=5,
                           scoring='neg_root_mean_squared_error',
n_jobs=-1)

grid_search.fit(cleaned[final_features], cleaned['calories'])

final_model = grid_search.best_estimator_
predictions = final_model.predict(cleaned[final_features])

final_model_rmse = rmse(cleaned['calories'], predictions)
```

```
final_model_rsquared = rsquared(cleaned['calories'], predictions)

print("Best parameters:", grid_search.best_params_)
print(f'R²: {round(final_model_rsquared, 4)}')
```

Step 8: Fairness Analysis

```
fairness_null_hypothesis ='My model is fair. Its RMSE for simple
recipes(n_steps<=9) and complex recipes(n_steps>9) are roughly the
same, and any differences are due to random chance.'
fairness_alternative_hypothesis ='My model is unfair. Its RMSE for
simple recipes is lower than complex recipes'
print(fairness_null_hypothesis)
print(fairness_alternative_hypothesis)

# Evaluate and test for fairness by comparing RMSE between simple and
complex recipes using a permutation test.

final_features = ['n_ingredients', 'minutes', 'n_steps']

X = cleaned[final_features]
y = cleaned['calories']

fairness_predictions = final_model.predict(X)

rmse_simple = rmse(y[cleaned['n_steps'] <= 9],
fairness_predictions[cleaned['n_steps'] <= 9])
rmse_complex = rmse(y[cleaned['n_steps'] > 9],
fairness_predictions[cleaned['n_steps'] > 9])

fairness_observed_difference = rmse_complex - rmse_simple

print(f'rmse_simple: {round(rmse_simple, 4)}')
print(f'rmse_complex: {round(rmse_complex, 4)}')
print(f'observed_difference: {round(fairness_observed_difference,
4)}')

fairness_permutation_differences = []

for _ in range(1000):
    shuffled_recipe_complexities =
np.random.permutation(cleaned['n_steps'] <= 9)

    simple_rmse_perm = rmse(
        cleaned['calories'][shuffled_recipe_complexities],
        fairness_predictions[shuffled_recipe_complexities]
    )
    complex_rmse_perm = rmse(
        cleaned['calories'][~shuffled_recipe_complexities],
        fairness_predictions[~shuffled_recipe_complexities]
```

```
    )

    fairness_permutation_differences.append(complex_rmse_perm - 
simple_rmse_perm)

fairness_permutation_differences = 
np.array(fairness_permutation_differences)
fairness_p_value = np.mean(fairness_permutation_differences >= 
fairness_observed_difference)

print(f'p_value: {fairness_p_value}')

fairness_hypothesis_test_result = f'We reject the null, there is 
strong evidence that my model is unfair as it is less accurate for 
complex recipes compared to simple ones because p-value of 
{fairness_p_value} is less than .05.'
fairness_hypothesis_test_result
```