CookSmart: Insights into Recipe Complexity and Ratings

Name(s): Max Chiu

Website Link: https://github.com/maxxyhc/maxchiu.github.io-recipe-and-rating

In [1]:
```
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.
```

In [2]:
```
!pip install tabulate
```

```
Requirement already satisfied: tabulate in /Users/maxchiu/miniforge3/envs/
dsc80/lib/python3.12/site-packages (0.9.0)
```

In [ ]:

Step 1: Introduction

The merged dataset combines detailed information about recipes with user interactions, creating a comprehensive resource. This dataset includes key details such as minutes, rating, n_steps, and n_ingredients. The project focuses on a critical question: What factors influence the average rating of a recipe? We could build a predictive model to estimate recipe ratings based on recipe features. This question is important because they provide actionable insights into how recipe complexity and preparation time affect user satisfaction, and they help predict user preferences to recommend recipes more effectively. This analysis is designed to benefit recipe creators, home cooks, and anyone interested in optimizing the recipe experience.

Step 2: Data Cleaning and Exploratory Data Analysis

In [3]:
```
recipes_path = "/Users/maxchiu/Desktop/dsc80-2024-fa/projects/project04/R
interactions_path = "/Users/maxchiu/Desktop/dsc80-2024-fa/projects/projec

#Data cleaning for the two given files
recipes = pd.read_csv(recipes_path)
interactions = pd.read_csv(interactions_path)

avg_ratings = interactions.groupby('recipe_id')['rating'].mean()
recipes['average_rating'] = round(avg_ratings,2)
recipes['average_rating'] = recipes['average_rating'].replace(np.nan,0)
```

```
def parse_nutrition(nutrition_str):
    """
    Convert a string representation of a list into a Python list using ba
    """
    if isinstance(nutrition_str, str):
        nutrition_str = nutrition_str.strip("[]")
        return [float(x.strip()) for x in nutrition_str.split(",")]
    return None

recipes['nutrition'] = recipes['nutrition'].apply(parse_nutrition)
nutrition_columns = ['calories', 'total_fat', 'sugar', 'sodium', 'protein
for i, col in enumerate(nutrition_columns):
    recipes[col] = recipes['nutrition'].apply(lambda x: x[i] if isinstanc
recipes = recipes.drop(columns=['nutrition'])

merged = recipes.merge(interactions,how="left", left_on="id", right_on="r
merged = merged.drop(columns= ["recipe_id"])
merged
```

Out[3]:

	name	id	minutes	contributor_id	submitted	tags	n_step
0	1 brownies in the world best ever	333281	40	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	1
1	1 in canada chocolate chip cookies	453467	45	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	1
2	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
3	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
4	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
...	...	...	...	...	...	...	.
234424	zydeco ya ya deviled eggs	308080	40	37779	2008-06-07	['60-minutes-or-less', 'time-to-make', 'course...	
234425	cookies by design cookies on a stick	298512	29	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	
234426	cookies by design sugar shortbread cookies	298509	20	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	
234427	cookies by design sugar	298509	20	506822	2008-04-15	['30-minutes-or-less',	

	name	id	minutes	contributor_id	submitted	tags	n_step
	shortbread cookies					'time-to-make', 'course...	
234428	cookies by design sugar shortbread cookies	298509	20	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	

234429 rows × 23 columns

In [4]:

```
# plt.figure(figsize=(10, 6))
# plt.hist(merged['n_steps'], bins=50, color='skyblue', edgecolor='black'
# plt.title('Distribution of Recipe Preparation Times (n_steps)')
# plt.xlabel('n_steps')
# plt.ylabel('Frequency')
# plt.grid(axis='y', linestyle='--', alpha=0.7)
# plt.show()
```

In [5]:

```
threshold = merged['minutes'].quantile(0.95)
cleaned_df = merged[merged['minutes'] <= threshold]
cleaned_df
```

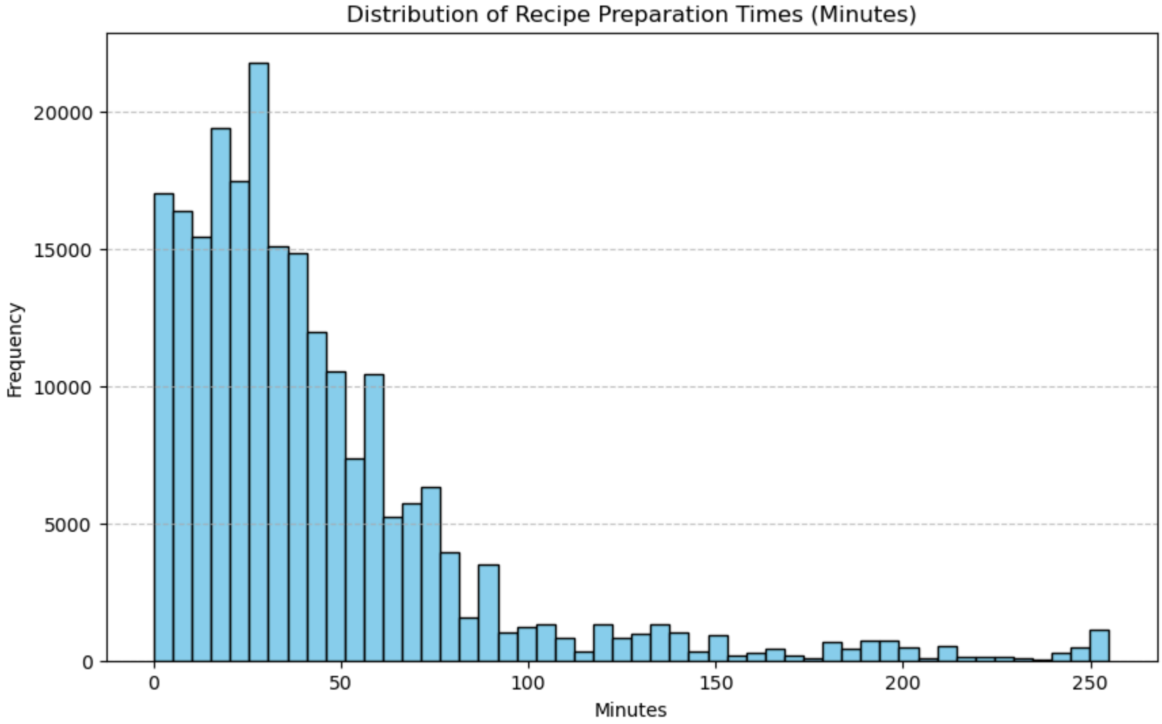
Out[5]:

	name	id	minutes	contributor_id	submitted	tags	n_step
0	1 brownies in the world best ever	333281	40	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	1
1	1 in canada chocolate chip cookies	453467	45	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	1
2	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
3	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
4	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	
...	...	...	...	...	...	...	.
234424	zydeco ya ya deviled eggs	308080	40	37779	2008-06-07	['60-minutes-or-less', 'time-to-make', 'course...	
234425	cookies by design cookies on a stick	298512	29	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	
234426	cookies by design sugar shortbread cookies	298509	20	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	
234427	cookies by design sugar	298509	20	506822	2008-04-15	['30-minutes-or-less',	

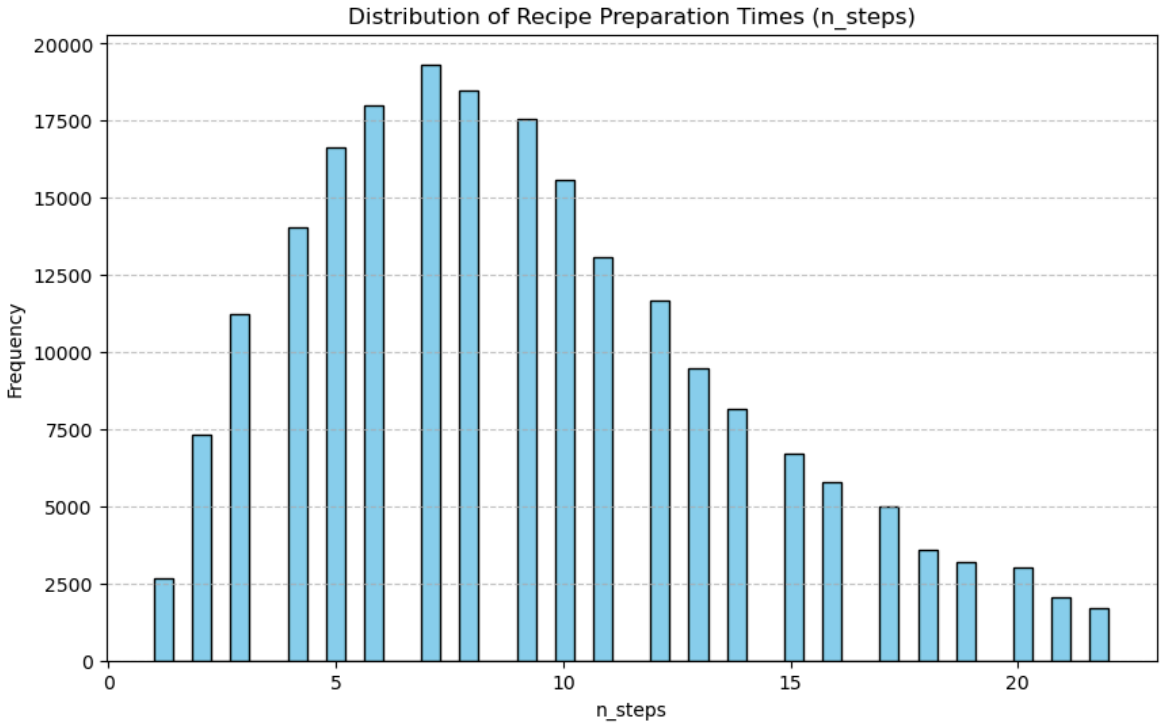
	name	id	minutes	contributor_id	submitted	tags	n_step
	shortbread cookies					'time-to-make', 'course...	
234428	cookies by design sugar shortbread cookies	298509	20	506822	2008-04-15	['30-minutes-or-less', 'time-to-make', 'course...	

222856 rows × 23 columns

In [6]:
```
# Plot a histogram of the 'minutes' column from the cleaned DataFrame
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(cleaned_df['minutes'], bins=50, color='skyblue', edgecolor='blac
plt.title('Distribution of Recipe Preparation Times (Minutes)')
plt.xlabel('Minutes')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



In [7]:
```
threshold = merged['n_steps'].quantile(0.95)
cleaned_df = cleaned_df[cleaned_df['n_steps'] <= threshold]
plt.figure(figsize=(10, 6))
plt.hist(cleaned_df['n_steps'], bins=50, color='skyblue', edgecolor='blac
plt.title('Distribution of Recipe Preparation Times (n_steps)')
plt.xlabel('n_steps')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



In [8]:

```
#Univariate Analysis
# Plotly histogram for average ratings
rating_distribution = cleaned_df['average_rating'].dropna()
fig = px.histogram(
    cleaned_df,
    x=rating_distribution,
    nbins=20,
    title='Distribution of Average Ratings',
    labels={'average_rating': 'Average Rating'},
    template='plotly_white'
)
fig.update_layout(
    yaxis_title="Frequency",
    xaxis_title="Average Rating"
)
fig.show(renderer="iframe")


# Plotly histogram for cooking time
outlier_threshold = cleaned_df['minutes'].quantile(0.99)

# Filter the data to remove outliers
filtered_data = cleaned_df[cleaned_df['minutes'] <= outlier_threshold]

fig_cooktime = px.histogram(
    filtered_data,
    x='minutes',
    nbins=20,
    title='Distribution of Cooking Time (minutes) - Outliers Removed',
    labels={'minutes': 'Cooking Time (minutes)'},
    template='plotly_white'
)
fig_cooktime.update_layout(yaxis_title="Frequency")
fig_cooktime.show(renderer="iframe")
fig_cooktime.write_html('Distribution-of-Cooking-Time-(minutes)-Outliers-
```

In [9]:
```
#remove 0
removed_zero = cleaned_df[cleaned_df['average_rating'] != 0.0]
```

In [10]:
```
# removed_zero.iloc[:5]
```

In [11]:
```
# print(removed_zero.iloc[:,:5].head().to_markdown(index=False))
```

In [12]:
```
rating_distribution = removed_zero['average_rating'].dropna()
fig = px.histogram(
    removed_zero,
    x=rating_distribution,
    nbins=20,
    title='Distribution of Average Ratings',
    labels={'average_rating': 'Average Rating'},
    template='plotly_white'
)
fig.update_layout(
    yaxis_title="Frequency",
    xaxis_title="Average Rating"
)
fig.show(renderer="iframe")
```

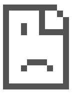
In [51]:

```python
fig = px.scatter(
    removed_zero,
    x='minutes',
    y='n_steps',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    #color='n_steps',
    #color_continuous_scale='Blues',
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```

In [52]:
```
fig = px.scatter(
    removed_zero,
    x='minutes',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
#may need to sqrt or log transform
```

In [53]:
```
fig = px.scatter(
    removed_zero,
    x='n_steps',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
#may not need change or maybe x^2
```

In [54]:
```
fig = px.scatter(
    removed_zero,
    x='calories',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```



In [17]:
```
removed_zero[['total_fat', 'sugar', 'sodium', 'protein',
         'saturated_fat', 'carbohydrates']]
```

Out[17]:

	total_fat	sugar	sodium	protein	saturated_fat	carbohydrates
82	41.0	28.0	13.0	67.0	51.0	3.0
83	0.0	70.0	0.0	2.0	0.0	7.0
84	0.0	70.0	0.0	2.0	0.0	7.0
85	0.0	70.0	0.0	2.0	0.0	7.0
86	0.0	70.0	0.0	2.0	0.0	7.0
...	...	...	...	...	...	...
234421	6.0	2.0	3.0	6.0	5.0	0.0
234422	6.0	2.0	3.0	6.0	5.0	0.0
234423	6.0	2.0	3.0	6.0	5.0	0.0
234424	6.0	2.0	3.0	6.0	5.0	0.0
234425	11.0	57.0	11.0	7.0	21.0	9.0

72193 rows × 6 columns

In [18]:
```
to_plot = ['total_fat', 'carbohydrates']
for t in to_plot:

    fig = px.scatter(
        removed_zero,
        x=t,
        y='average_rating',
        title=f"Scatter plot of {t}",

        size='n_steps',

    )

    # Show the figure
    fig.show(renderer="iframe")
```

In [19]:
```
#to do  'sugar', 'sodium', 'protein', 'saturated_fat'

fig = px.scatter(
    removed_zero,
    x='sugar',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```

In [20]:
```
fig = px.scatter(
    removed_zero,
    x='sodium',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```

In [21]:
```
fig = px.scatter(
    removed_zero,
    x='protein',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```

In [22]:
```
fig = px.scatter(
    removed_zero,
    x='saturated_fat',
    y='average_rating',
    title='Scatter Plot of Steps Over Time',
    labels={'minutes': 'Minutes', 'n_steps': 'Number of Steps'},
    size='n_steps',
    hover_name='minutes'
)

# Show the figure
fig.show(renderer="iframe")
```

In [23]:

```
# Calories vs. Average Ratings
fig2 = px.scatter(
    removed_zero,
    x='calories',
    y='average_rating',
    title='Calories vs. Average Recipe Ratings',
    labels={'calories': 'Calories', 'average_rating': 'Average Rating'},
    template='plotly_white'
)
fig2.show(renderer="iframe")
fig2.write_html('Calories-vs.-Average-Recipe-Ratings.html', include_plotl
```

In [24]:

```
merged['year'] = pd.to_datetime(merged['submitted'], errors='coerce').dt.


bins = [0, 30, 60, 90, 120, 150, 180, 210, 240, float('inf')]
labels = ["0-30", "30-60", "60-90", "90-120", "120-150", "150-180", "180-

# Create the 'minutes_bin' column based on bins
merged['minutes_bin'] = pd.cut(merged['minutes'], bins=bins, labels=label


pivot_table = pd.pivot_table(
    merged,
    values='rating',
    index='minutes_bin',
    columns='year',
    aggfunc='mean'
)

pivot_table
```

```
/var/folders/2h/80xj55qs223gk3hzcqk03zxh0000gn/T/ipykernel_11036/87920842.
py:11: FutureWarning:

The default value of observed=False is deprecated and will change to obser
ved=True in a future version of pandas. Specify observed=False to silence
this warning and retain the current behavior
```

Out[24]:

year	2008	2009	2010	2011	2012	2013	2
minutes_bin							
0-30	4.451862	4.463126	4.486388	4.492542	4.442878	4.448097	4.212
30-60	4.396222	4.395263	4.362192	4.426637	4.404619	4.334935	4.351
60-90	4.332100	4.327358	4.299153	4.347342	4.319891	4.314498	4.130
90-120	4.321290	4.313666	4.391917	4.265560	4.309771	4.060423	4.387
120-150	4.240640	4.349365	4.427273	4.385057	4.281457	4.147186	4.525
150-180	4.257745	4.263400	3.995413	4.168675	4.384000	4.370370	4.052
180-210	4.330435	4.207526	4.329384	4.245421	4.403670	3.654930	4.296
210-240	4.269231	4.082192	4.323810	4.400000	4.622642	4.304348	3.888
240+	4.190138	4.227747	4.192568	4.072345	4.051630	4.024845	3.745

In [25]:
```
# print(pivot_table.iloc[:,:].head().to_markdown(index=True))
```

In [26]:
```
#features to use
# to_use = ['minutes','n_steps','calories', 'total_fat', 'sugar', 'sodium
```

Step 3: Assessment of Missingness

In [27]:
```
copy_df = removed_zero.copy()
copy_df['description_missing'] = copy_df['description'].isna().astype(int

#column1 = 'minutes'
#column2 = 'n_ingredients'

# test statistics
def compute_mean_difference(means):
    return abs(means[1] - means[0])

observed_means = copy_df.groupby('description_missing')['minutes'].mean()
observed_mean_diff = compute_mean_difference(observed_means)

observed_means2 = copy_df.groupby('description_missing')['n_ingredients']
observed_mean_diff2 = compute_mean_difference(observed_means2)

# Permutation test
n_perm = 1000
mean_diff_null = []
mean_diff_null2 = []

for _ in range(n_perm):
    #shuffling
```

```
    copy_df['missing_description'] = np.random.permutation(copy_df['descr

    null_means = copy_df.groupby('missing_description')['minutes'].mean()
    null_means2 = copy_df.groupby('missing_description')['n_ingredients']
    mean_diff_null.append(compute_mean_difference(null_means))
    mean_diff_null2.append(compute_mean_difference(null_means2))

# Calculate p-values
mean_diff_null = np.array(mean_diff_null)
mean_diff_null2 = np.array(mean_diff_null2)

mean_diff_p_value = np.mean(mean_diff_null >= observed_mean_diff)
mean_diff_p_value2 = np.mean(mean_diff_null2 >= observed_mean_diff2)

mean_diff_p_value,mean_diff_p_value2
```

Out[27]:

```
(np.float64(0.31), np.float64(0.026))
```

In [28]:

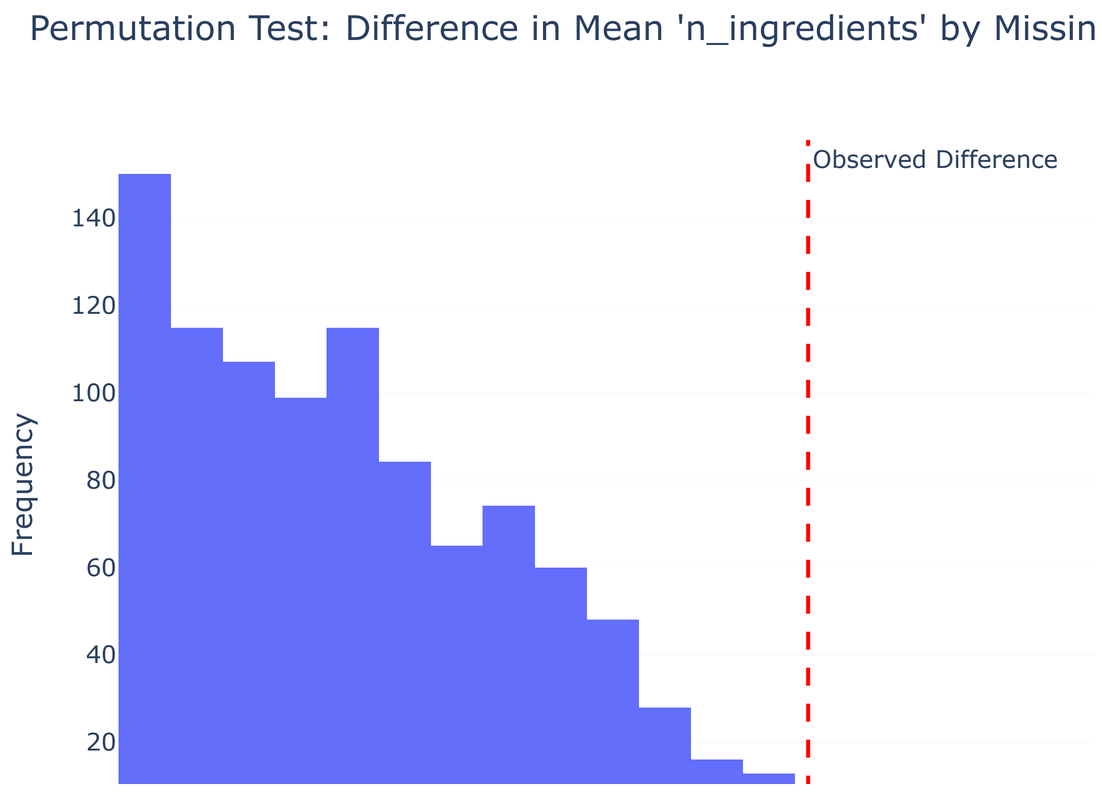
```
missing = copy_df[copy_df['description_missing']== True]
not_missing = copy_df[copy_df['description_missing'] == False]

#The distribution of column Y when column X is missing
fig_missing = px.histogram(
    missing,
    x='minutes',
    nbins=30,
    title="Distribution of Minutes when Description is Missing",
    labels={'x': 'Minutes'},
    template='plotly_white'
)
fig_missing.show(renderer="iframe")
fig_missing.write_html('Distribution-of-Minutes-when-Description-is-Missi
```

In [29]:

```
#the distribution of column Y when column X is not missing
fig_not_missing = px.histogram(
    not_missing,
    x='minutes',
    nbins=30,
    title="Distribution of Minutes when Description is not Missing",
    labels={'x': 'Minutes'},
    template='plotly_white'
)
fig_not_missing.show(renderer="iframe")
fig_not_missing.write_html('Distribution-of-Minutes-when-Description-is-n
```

In [30]:
```
# Plot 1: Empirical distribution of the test statistic for 'minutes'
fig1 = px.histogram(
    x=mean_diff_null,
    nbins=30,
    title="Permutation Test: Difference in Mean 'Minutes' by Missing Desc
    labels={'x': 'Difference in Mean Minutes'},
    template='plotly_white'
)
fig1.add_vline(
    x=observed_mean_diff,
    line_dash="dash",
    line_color="red",
    annotation_text="Observed Difference",
    annotation_position="top right"
)
fig1.update_layout(xaxis_title="Difference in Mean Minutes", yaxis_title=
fig1.show(renderer="iframe")
fig1.write_html("Permutation-Test:-Difference-in-Mean-'Minutes'-by-Missin
```

In [31]:
```
# Plot 2: Empirical distribution of the test statistic for 'n_ingredients
fig2 = px.histogram(
    x=mean_diff_null2,
    nbins=30,
    title="Permutation Test: Difference in Mean 'n_ingredients' by Missin
    labels={'x': 'Difference in Mean n_Ingredients'},
    template='plotly_white'
)
fig2.add_vline(
    x=observed_mean_diff2,
    line_dash="dash",
    line_color="red",
    annotation_text="Observed Difference",
    annotation_position="top right"
)
fig2.update_layout(xaxis_title="Difference in Mean n_Ingredients", yaxis_
#fig2.write_html("Permutation-Test:-Difference-in-Mean-'n_ingredients'-by
fig2
```



Step 4: Hypothesis Testing

Null Hypothesis: The mean average rating of recipes with short cooking times is equal to the mean average rating of recipes with long cooking times.

Alternative Hypothesis: The mean average rating of recipes with short cooking times is not equal to the mean average rating of recipes with long cooking times.

Test statistic: This is the difference between the mean ratings of the two groups (short vs. long cooking times). This choice directly compares the means, making it an intuitive measure for identifying any difference.

Significance Level: 0.05(5%)

Permutation Test Results: Observed Difference: 0.0177 p-Value: 0.005

Conclusion: Since the p-value (0.005) is less than the significance level, we reject the null hypothesis. This suggests that there is a statistically significant difference in mean ratings between recipes with short and long cooking times.

The permutation test was chosen because I want to test whether the two distributions,mean ratings of recipes with short and long cooking times, differ significantly. The test statistic is the observed difference in mean ratings between the two groups. I chose this measure because I want to see whether one distribution is greater, less, or significantly different from the other distribution. This difference directly quantifies the relationship between the two groups, making it an intuitive and interpretable measure for this analysis. A significance level of 0.05 was selected as it is a standard threshold.

In [32]:
```
df_copy = removed_zero.copy()
# Step 1: split into "short" and "long" cooking times based on the median
cooking_time_median = df_copy['minutes'].median()
df_copy['cooking_time_group'] = np.where(df_copy['minutes'] <= cooking_ti

# Step 2: Compute the observed difference in mean ratings
short_group_mean = df_copy[df_copy['cooking_time_group'] == 'short']['ave
long_group_mean = df_copy[df_copy['cooking_time_group'] == 'long']['avera
observed_diff = short_group_mean - long_group_mean

#Step 3: permutation test
def perm_test_mean_difference(data, group_col, value_col, num_permutation
    """
    Perform a permutation test for the difference in means between two gr
    """
    observed_diff = data[data[group_col] == 'short'][value_col].mean() -
                    data[data[group_col] == 'long'][value_col].mean()

    perm_diffs = []
    for _ in range(num_permutations):
        shuffled_labels = np.random.permutation(data[group_col])
        perm_diff = data[shuffled_labels == 'short'][value_col].mean() -
                    data[shuffled_labels == 'long'][value_col].mean()
        perm_diffs.append(perm_diff)

    p_value = np.mean(np.abs(perm_diffs) >= np.abs(observed_diff))
    return observed_diff, perm_diffs, p_value

# Run the permutation test
observed_diff, perm_diffs, p_value = perm_test_mean_difference(
    df_copy, 'cooking_time_group', 'average_rating', num_permutations=100
)

# Step 4: Visualize results

fig = px.histogram(
    x=perm_diffs,
    nbins=30,
    title="Permutation Test: Difference in Mean Ratings by Cooking Time",
    labels={'x': 'Difference in Mean Ratings'},
    template='plotly_white'
)
fig.update_layout(
    xaxis_title="Difference in Mean Ratings",
    yaxis_title="Frequency"
)
fig.show(renderer="iframe")
observed_diff, p_value
```

```
Out[32]: (np.float64(0.01771754319570551), np.float64(0.005))
```

Step 5: Framing a Prediction Problem

Predict Problem: We aim to predict the average rating of recipes based on recipe features. This is a regression problem, as the target variable, average_rating, is continuous.

Response Variable: average_rating the reason chosing this variable because It directly reflects user satisfaction, serving as a meaningful metric for recipe recommendation systems, while understanding the factors influencing ratings can help recipe creators improve their recipes and assist users in finding recipes they are more likely to enjoy.

Features: minutes, n_steps, n_ingredients, calories, total_fat, sugar, sodium, protein, saturated_fat, carbohydrates

Metric: Mean Squared Error (MSE) MSE is used because it provides a straightforward interpretation of prediction error by measuring the average squared magnitude of errors in the same units as the target variable (ratings).

best_r2_score, mse_tes

Step 6: Baseline Model

In [33]:
```
# # Define features and target variable
# from sklearn.model_selection import train_test_split
# from sklearn.pipeline import Pipeline
# from sklearn.preprocessing import StandardScaler
# from sklearn.linear_model import LinearRegression
# from sklearn.metrics import mean_absolute_error

# X = merged[['minutes', 'n_steps', 'n_ingredients']]
# y = merged['average_rating']

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# pipeline = Pipeline([
#     ('scaler', StandardScaler()),  # Standardize features
#     ('model', LinearRegression())  # Linear Regression model
# ])

# #Train the baseline model
# pipeline.fit(X_train, y_train)

# #Evaluate the model
# y_train_pred = pipeline.predict(X_train)
# y_test_pred = pipeline.predict(X_test)

# train_mae = mean_absolute_error(y_train, y_train_pred)
# test_mae = mean_absolute_error(y_test, y_test_pred)

# train_mae, test_mae




#The baseline model is a Linear Regression model designed to predict the
#using three quantitative features: minutes (total cooking time), n_steps
#(number of ingredients). All the features are quantitative, and since th
#this model, no encoding was required. To ensure the features are on a co
#StandardScaler before fitting the regression model. The performance of t
#Error (MAE). On the training set, the MAE was approximately 1.96, and on
#indicates that, on average, the model's predictions deviate from the tru
#Given that the target variable (average_rating) ranges from 1 to 5, this
#that the model’s predictive accuracy is limited. The results suggest tha
#While there is no sign of overfitting (the training and testing errors a
#assumption of linearity likely fail to capture the complexity of the dat
#nutritional information, or exploring non-linear models could improve pe
```

In [34]:
```
removed_zero[['minutes', 'n_steps', 'calories', 'total_fat', 'sugar','sod
```

Out[34]:

	minutes	n_steps	calories	total_fat	sugar	sodium	protein	saturated_f
82	75	9	429.9	41.0	28.0	13.0	67.0	5
83	5	2	94.7	0.0	70.0	0.0	2.0	(
84	5	2	94.7	0.0	70.0	0.0	2.0	(
85	5	2	94.7	0.0	70.0	0.0	2.0	(
86	5	2	94.7	0.0	70.0	0.0	2.0	(
...	...	...	...	...	...	...	...	
234421	40	7	59.2	6.0	2.0	3.0	6.0	!
234422	40	7	59.2	6.0	2.0	3.0	6.0	!
234423	40	7	59.2	6.0	2.0	3.0	6.0	!
234424	40	7	59.2	6.0	2.0	3.0	6.0	!
234425	29	9	188.0	11.0	57.0	11.0	7.0	2

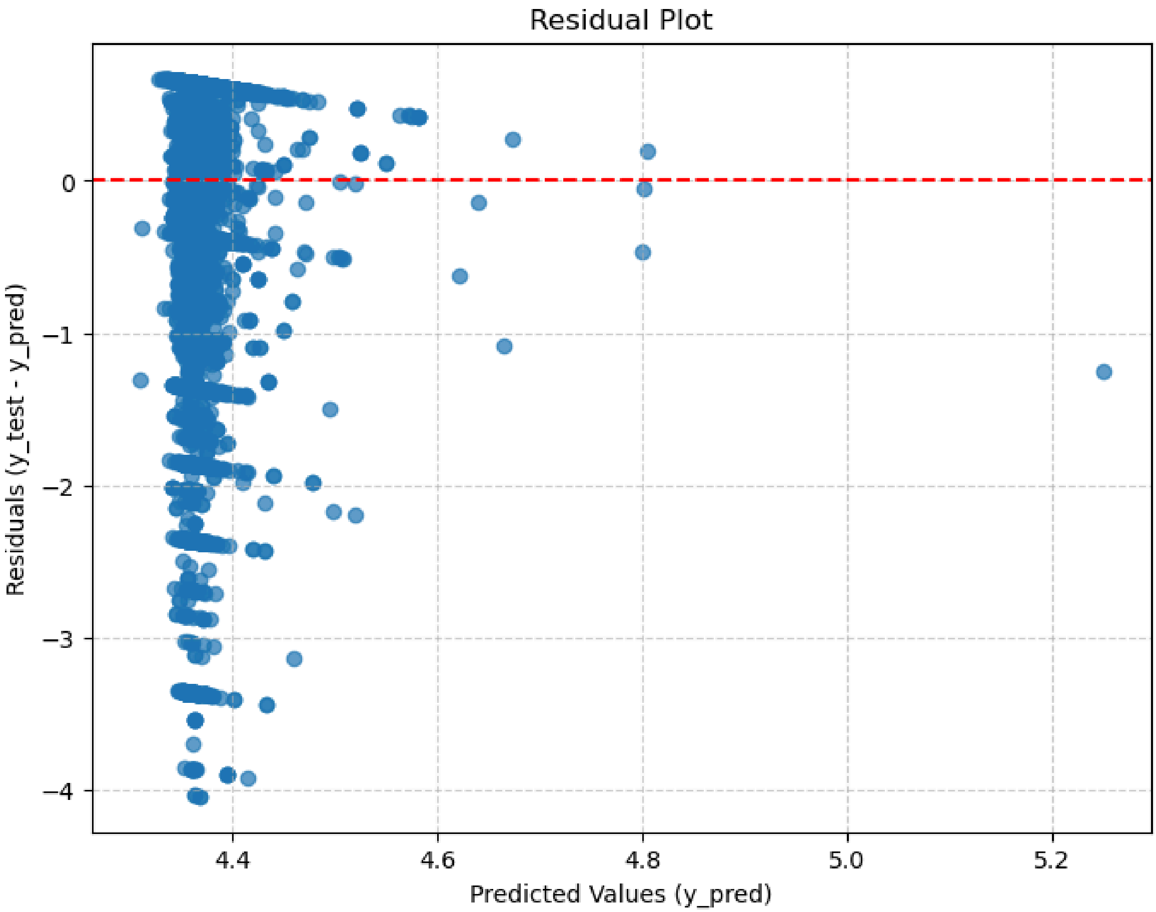
72193 rows × 9 columns

In [35]:

```
#to_use = ['minutes','n_steps','calories', 'total_fat', 'sugar', 'sodium'
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar','sodium'
target = 'average_rating'

# Splitting the data into features and target
X = removed_zero[to_use]
y = removed_zero[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Creating the pipeline
pipeline = Pipeline([
    ('regressor', LinearRegression())
])

# Fitting the model
pipeline.fit(X_train, y_train)

# Making predictions
y_pred = pipeline.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

Out[35]: (np.float64(0.6744103743883941), -4.711417292746489e-05)

In [36]:
```
import matplotlib.pyplot as plt

# Calculating residuals
residuals = y_test - y_pred

# Creating the residual plot
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("Predicted Values (y_pred)")
plt.ylabel("Residuals (y_test - y_pred)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



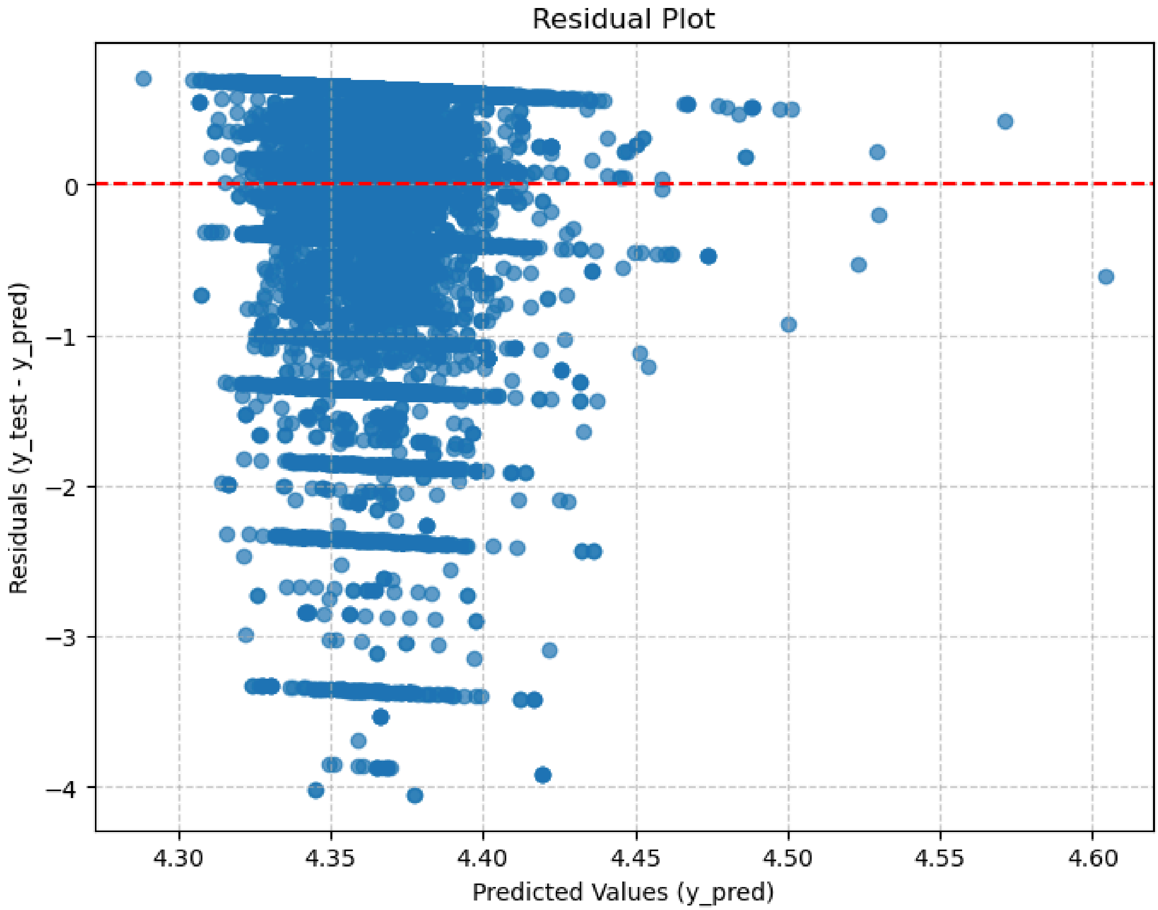
In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer

to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar','sodium'
target = 'average_rating'

# Splitting the data into features and target
X = removed_zero[to_use]
y = removed_zero[target]
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

preproc = ColumnTransformer(
    transformers = [
        ('sqrt modifier', FunctionTransformer(np.sqrt),to_use)
    ]
)

# Creating the pipeline
pipeline = Pipeline([
    ('preprocessor', preproc),
    ('regressor', LinearRegression())
])

# Fitting the model
pipeline.fit(X_train, y_train)

# Making predictions
y_pred = pipeline.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

Out[37]: (np.float64(0.674370999486436), 1.127276560342061e-05)

In [38]:

```
import matplotlib.pyplot as plt

# Calculating residuals
residuals = y_test - y_pred

# Creating the residual plot
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("Predicted Values (y_pred)")
plt.ylabel("Residuals (y_test - y_pred)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



In [39]:
```
# from sklearn.model_selection import train_test_split
# from sklearn.pipeline import Pipeline
# from sklearn.preprocessing import StandardScaler
# from sklearn.linear_model import LinearRegression
# from sklearn.metrics import mean_squared_error, r2_score
# from sklearn.preprocessing import FunctionTransformer
# from sklearn.compose import ColumnTransformer

# to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar','sodiu
# target = 'average_rating'

# # Splitting the data into features and target
# X = removed_zero[to_use]
# y = removed_zero[target]

# # Train-test split
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
# def safe_log_transform(X):
#     return np.log(X + 1e-6)

# preproc = ColumnTransformer(
#     transformers = [
#         ('sqrt modifier', FunctionTransformer(safe_log_transform),to_us
#     ]
# )

# # Creating the pipeline
# pipeline = Pipeline([
#     ('preprocessor', preproc),
#     ('regressor', LinearRegression())
# ])
```

```
# # Fitting the model
# pipeline.fit(X_train, y_train)

# # Making predictions
# y_pred = pipeline.predict(X_test)

# # Evaluating the model
# mse = mean_squared_error(y_test, y_pred)
# r2 = r2_score(y_test, y_pred)

# mse, r2
```

In [40]:
```
# # Calculating residuals
# residuals = y_test - y_pred

# # Creating the residual plot
# plt.figure(figsize=(8, 6))
# plt.scatter(y_pred, residuals, alpha=0.7)
# plt.axhline(y=0, color='r', linestyle='--')
# plt.title("Residual Plot")
# plt.xlabel("Predicted Values (y_pred)")
# plt.ylabel("Residuals (y_test - y_pred)")
# plt.grid(True, linestyle='--', alpha=0.6)
# plt.show()
```

In [ ]:
```

```

Step 7: Final Model

In [41]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
import numpy as np

# Define the features and target column
to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar', 'sodium
target = 'average_rating'

# Splitting the data into features and target
X = removed_zero[to_use]
y = removed_zero[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Safe logarithmic transformation to handle non-positive values
def safe_log_transform(X):
    return np.log(X + 1e-6)

# Preprocessing: Log-transform the features
preproc = ColumnTransformer(
    transformers=[
        ('log_modifier', FunctionTransformer(safe_log_transform), to_use)
```

```
    ]
)

# Base pipeline with LinearRegression
pipeline_poly = Pipeline([
    ('preprocessor', preproc),
    ('poly_features', PolynomialFeatures(include_bias=False)),  # Add pol
    ('scaler', StandardScaler()),  # Scale the polynomial features
    ('regressor', LinearRegression())  # Linear regression
])

# Parameter grid for GridSearchCV
param_grid = {
    'poly_features__degree': range(1, 6)  # Test polynomial degrees from
}

# GridSearchCV
grid_search = GridSearchCV(
    estimator=pipeline_poly,
    param_grid=param_grid,
    cv=5,  # 5-fold cross-validation
    scoring='r2',  # Use R2 as the evaluation metric
    n_jobs=-1,  # Use all processors
    verbose=1
)

# Fitting the GridSearchCV
grid_search.fit(X_train, y_train)

# Best parameters and best score
best_degree = grid_search.best_params_['poly_features__degree']
best_r2_score = grid_search.best_score_

# Evaluating on the test set
best_model = grid_search.best_estimator_
y_pred_test = best_model.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred_test)
r2_test = r2_score(y_test, y_pred_test)

best_degree, best_r2_score, mse_test, r2_test
```

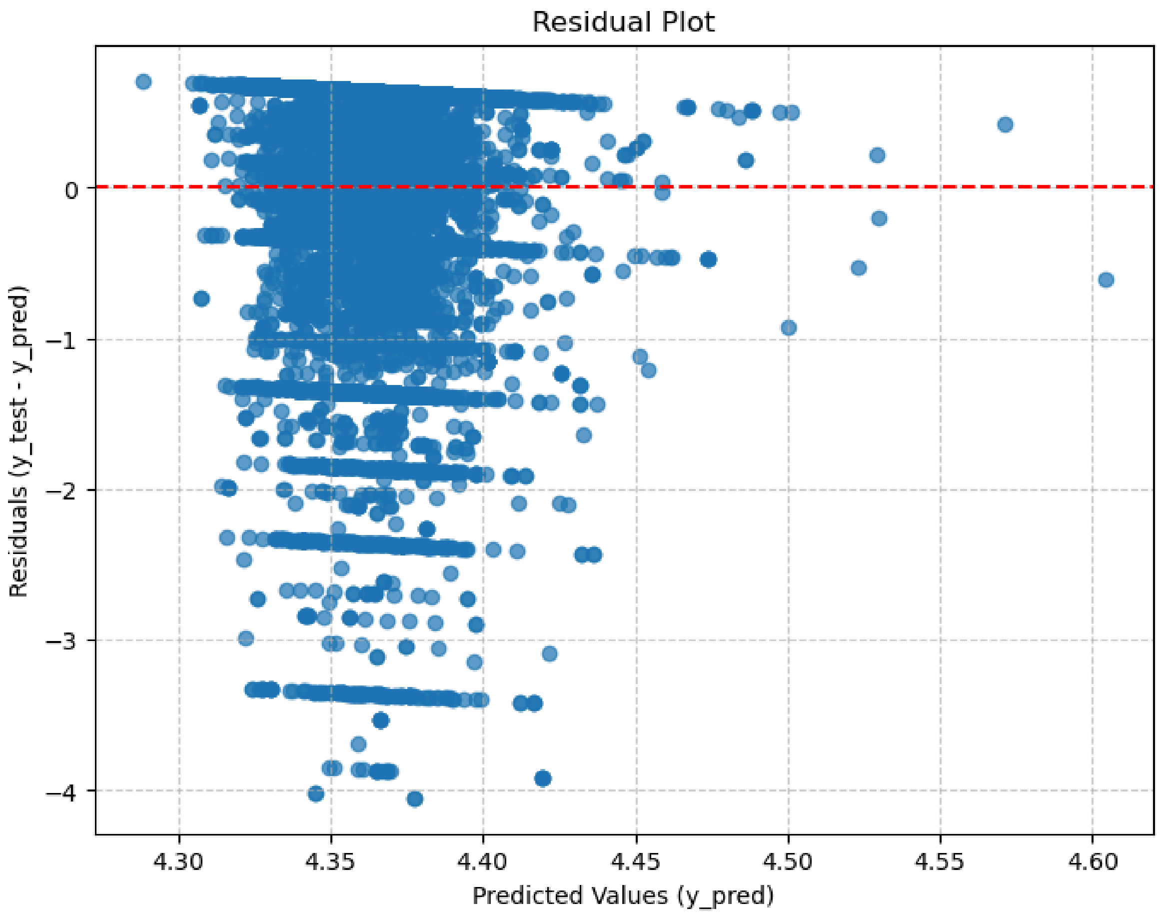
```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[41]:
```
(3,
 np.float64(0.015578416697516561),
 np.float64(0.6611808945666113),
 0.019570174677026486)
```

In [42]:
```
# from sklearn.model_selection import GridSearchCV
# from sklearn.pipeline import Pipeline
# from sklearn.preprocessing import StandardScaler, PolynomialFeatures
# from sklearn.linear_model import LinearRegression
# from sklearn.metrics import mean_squared_error, r2_score
# from sklearn.compose import ColumnTransformer
# from sklearn.preprocessing import FunctionTransformer
# import numpy as np

# # Define the features and target column
# to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar', 'sodi
# target = 'average_rating'
```

```
# # Splitting the data into features and target
# X = removed_zero[to_use]
# y = removed_zero[target]

# # Train-test split
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# # Safe logarithmic transformation to handle non-positive values
# def safe_log_transform(X):
#     return np.log(X + 1e-6)

# # Preprocessing: Log-transform the features
# preproc = ColumnTransformer(
#     transformers=[
#         ('log_modifier', FunctionTransformer(safe_log_transform), to_us
#     ]
# )

# # Base pipeline with LinearRegression
# pipeline_poly = Pipeline([
#     ('preprocessor', preproc),
#     ('poly_features', PolynomialFeatures(degree=3, include_bias=False))
#     ('scaler', StandardScaler()),  # Scale the polynomial features
#     ('regressor', LinearRegression())  # Linear regression
# ])

# pipeline_poly.fit(X_train,y_train)
```

In [43]:
```
residuals = y_test - y_pred

# Creating the residual plot
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color='r', linestyle='--')
plt.title("Residual Plot")
plt.xlabel("Predicted Values (y_pred)")
plt.ylabel("Residuals (y_test - y_pred)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



Step 8: Fairness Analysis

Null Hypothesis: the performance of the mertric for below-median-calories and above-median-calories are roughly the same,and any observed difference in performance is due to random chance.

Alternative Hypothesis: The performance metric for below-median-calories and above-median-calories are significantly different, suggesting that the model performs better for one group over the other.

In [44]:
```
removed_zero.columns
```

Out[44]:
```
Index(['name', 'id', 'minutes', 'contributor_id', 'submitted', 'tags',
       'n_steps', 'steps', 'description', 'ingredients', 'n_ingredient
s',
       'average_rating', 'calories', 'total_fat', 'sugar', 'sodium', 'pr
otein',
       'saturated_fat', 'carbohydrates', 'user_id', 'date', 'rating',
       'review'],
      dtype='object')
```

In [45]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score

#copy df
df_copy = removed_zero.copy()
```

```
#the median of the calories
median_calories = df_copy['calories'].median()
df_copy['calories_group'] = np.where(df_copy['calories'] > median_calorie
```

In [46]:
```
# Define the features and target column
to_use = ['minutes', 'n_steps', 'calories', 'total_fat', 'sugar', 'sodium
target = 'average_rating'

# Splitting the data into features and target
X = df_copy[to_use]
y = df_copy[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,



# Safe logarithmic transformation to handle non-positive values
def safe_log_transform(X):
    return np.log(X + 1e-6)

# Preprocessing: Log-transform the features
preproc = ColumnTransformer(
    transformers=[
        ('log_modifier', FunctionTransformer(safe_log_transform), to_use)
    ]
)

# Base pipeline with LinearRegression
pipeline_poly = Pipeline([
    ('preprocessor', preproc),
    ('poly_features', PolynomialFeatures(degree=3, include_bias=False)),
    ('scaler', StandardScaler()),  # Scale the polynomial features
    ('regressor', LinearRegression())  # Linear regression
])

pipeline_poly.fit(X_train,y_train)



X_test['is_fat'] = df_copy['calories'] > median_calories
test_df = pd.merge(X_test, y_test, left_index=True, right_index=True, how



# Predictions on the test set
test_df['pred'] = pipeline_poly.predict(X_test)

test_df
```

Out[46]:

	minutes	n_steps	calories	total_fat	sugar	sodium	protein	saturated_f
25304	60	11	641.4	47.0	242.0	25.0	15.0	62
82366	30	8	33.6	0.0	4.0	2.0	2.0	C
197958	15	7	380.7	24.0	20.0	30.0	31.0	24
14498	30	10	108.1	9.0	13.0	20.0	9.0	9
137343	50	14	443.3	43.0	10.0	20.0	51.0	48
...	...	...	...	...	...	...	...	
219331	5	3	181.9	8.0	89.0	2.0	8.0	18
127684	55	12	430.0	41.0	17.0	26.0	74.0	57
71648	55	7	277.5	14.0	4.0	19.0	69.0	18
157052	60	10	256.6	17.0	79.0	8.0	8.0	13
40601	25	9	37.9	0.0	2.0	0.0	1.0	C

14439 rows × 12 columns

In [47]:

```
# Separate fat and nonfat players in the test set
fat_indices = test_df['is_fat']
non_fat_indices = (test_df['is_fat'] == False)

# RMSE calculation for "fat" group
rmse_fat = np.sqrt(mean_squared_error(
    test_df.loc[fat_indices, 'average_rating'],  # True values
    test_df.loc[fat_indices, 'pred']  # Predicted values
))

# RMSE calculation for "non-fat" group
rmse_non_fat = np.sqrt(mean_squared_error(
    test_df.loc[non_fat_indices, 'average_rating'],  # True values
    test_df.loc[non_fat_indices, 'pred']  # Predicted values
))

# Calculate the observed difference in RMSE
observed_diff = rmse_fat - rmse_non_fat

print(f"Observed RMSE Difference: {observed_diff:.4f}")
```

```
Observed RMSE Difference: 0.0299
```

In [48]:

```
#permutation testing
n_permutations = 1000
permuted_diffs = []

# Perform permutation test
for _ in range(n_permutations):
    # Shuffle predicted labels
    shuffled_labels = np.random.permutation(test_df['is_fat'])

    # Split shuffled predictions into LPL and non-LPL
    shuffled_fat = shuffled_labels
    shuffled_non_fat = (shuffled_labels == False)
```

```
    # Calculate precision for shuffled groups
    rmse_shuffled_fat = np.sqrt(mean_squared_error(test_df.loc[shuffled_f
                                          test_df.loc[shuffled_fat, 'p

    rmse_shuffled_non_fat = np.sqrt(mean_squared_error(test_df.loc[shuffl
                                              test_df.loc[shuffled_non

    # Store the difference
    permuted_diff = rmse_shuffled_fat - rmse_shuffled_non_fat
    permuted_diffs.append(permuted_diff)

# Convert to NumPy array
permuted_diffs = np.array(permuted_diffs)

# Calculate p-value (two-tailed test)
p_value = (permuted_diffs >= observed_diff).mean()

# Print results
print(f"Observed rmse Difference: {observed_diff:.4f}")
print(f"P-value: {p_value:.4f}")
#not fair
```

```
Observed rmse Difference: 0.0299
P-value: 0.0260
```

In [ ]:

In [ ]: