Explore the ralationship between calories and ratings in diet

Name(s): YAJIGNG FANG

Website Link: https://yaf008.github.io/Explore_recipes/

In [38]:
```
# Importing essential libraries for data manipulation and visualization
import pandas as pd
import numpy as np
from pathlib import Path

# Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
pd.options.plotting.backend = 'plotly'
import matplotlib.pyplot as plt
import plotly.tools as mpl_tools
import plotly.figure_factory as ff

# Statistical analysis
from scipy.stats import pearsonr

# Machine learning libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, QuantileTransformer, Funct
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# Miscellaneous
import random
```

Step 1: Introduction

Data set introduction

My dataset comes from Food.com, which contains recipes and user ratings. This dataset records various types of recipes and their corresponding ratings and calorie information. At the heart of my project was the question: What is the relationship between a recipe's rating and its calorie content? I wanted to explore modern eating habits and see if people tend to give high marks to low-calorie recipes or whether high-calorie recipes are more popular.

Why is this a concern?

This question is especially important in these days of health and eating habits. By analyzing this relationship, we can better understand current food preferences and help the restaurant industry, health advocates, and everyday users fine-tune diet recommendations and choices.

Step 2: Data Cleaning and Exploratory Data Analysis

Data Cleaning

Clean the data appropriately. For instance, you may need to replace data that should be missing with NaN or create new columns out of given ones (e.g. compute distances, scale data, or get time information from time stamps).

In [2]:
```
#read RAW_interactions file
file_path = Path('data') / 'RAW_interactions.csv'
reviews = pd.read_csv(file_path)


#read RAW_recipes file
file_path = Path('data') / 'RAW_recipes.csv'
recipes = pd.read_csv(file_path)
```

In [3]:
```
#merge these two raw df together, and fill rating nan with 0
rate_recipes = recipes.merge(reviews, left_on = 'id', right_on = 'recipe_id'
fill_rate = rate_recipes.copy()
fill_rate['rating'] = fill_rate['rating'].fillna(0)
avg_rating = fill_rate.groupby('id')['rating'].mean().reset_index()
avg_rating = avg_rating.rename(columns={'rating': 'average rating'})
avg_rating.head()
```

Out[3]:

	id	average rating
0	275022	3.0
1	275024	3.0
2	275026	3.0
3	275030	5.0
4	275032	5.0

In [4]:
```
#average rating for each recipe, and put average rating in to recipes df
avg_recipes = rate_recipes.merge(avg_rating, left_on = 'id', right_on = 'id'
```

In [5]:
```
#put each nutrition into a col
#change sting to list
avg_recipes['nutrition'] = avg_recipes['nutrition'].apply(lambda x : x.strip
```

```
#chang string in list to num in list
avg_recipes['nutrition'] = avg_recipes['nutrition'].apply(lambda x : [float(
# put each nutrition value to be a col
avg_recipes[['calories', 'total_fat', 'sugar', 'sodium', 'protein', 'saturat
#drop nutrition col
avg_recipes= avg_recipes.drop('nutrition',axis=1)
avg_recipes.head()
```

Out[5]:

	name	id	minutes	contributor_id	submitted	tags	n_steps	ste
0	impossible macaroni and cheese pie	275022	50	531768	2008-01-01	['60-minutes-or-less', 'time-to-make', 'course...	11	['h over 4 degr fahrenh 'grea
1	impossible macaroni and cheese pie	275022	50	531768	2008-01-01	['60-minutes-or-less', 'time-to-make', 'course...	11	['h over 4 degr fahrenh 'grea
2	impossible macaroni and cheese pie	275022	50	531768	2008-01-01	['60-minutes-or-less', 'time-to-make', 'course...	11	['h over 4 degr fahrenh 'grea
3	impossible rhubarb pie	275024	55	531768	2008-01-01	['60-minutes-or-less', 'time-to-make', 'course...	6	['h over 3 degre 'gre 10" pan
4	impossible seafood pie	275026	45	531768	2008-01-01	['60-minutes-or-less', 'time-to-make', 'course...	7	['preh over 40 'ligl gre larg

5 rows × 24 columns

Univariate Analysis

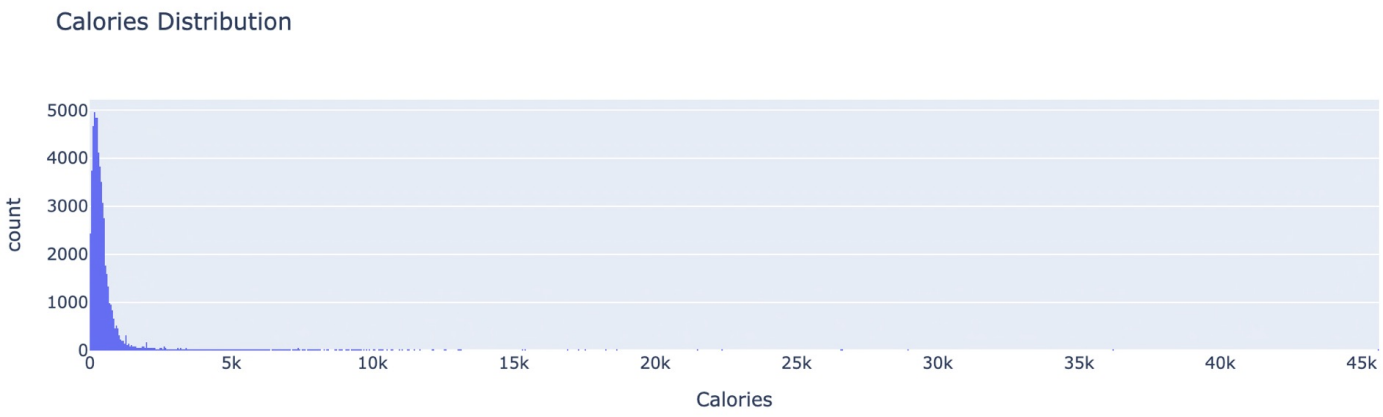
In a univariate analysis, I would look at the distribution of calories and recipe scores separately.

Calories Distribution

In [6]:
```
##Calories distribution
print(avg_recipes['calories'].describe())
```

```
count    234429.00000
mean        419.52887
std         583.22359
min           0.00000
25%         170.70000
50%         301.10000
75%         491.10000
max       45609.00000
Name: calories, dtype: float64
```

In [11]:
```
fig1 = px.histogram(avg_recipes, x='calories', title='Calories Distribution'
fig1.show()
import plotly.express as px
fig1.write_html('assets/calories_dis.html', include_plotlyjs='cdn')
```



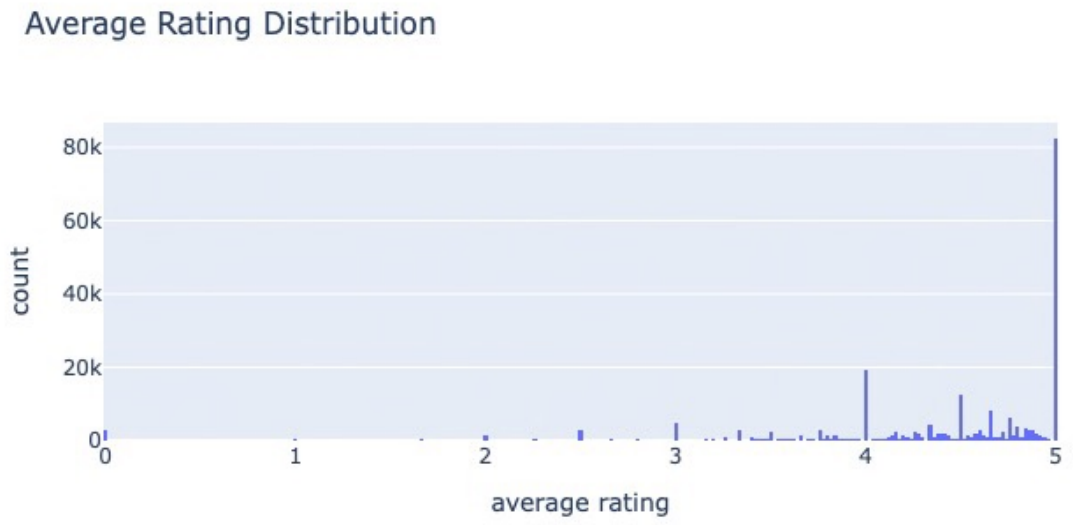
As you can see, the distribution is skewed to the right. Some recipes have unimaginable calorie content, but these are very few. The vast majority of calories are in the 200-500 calorie range.

Average Rating Distribution

In [12]:
```
#average rating
print(avg_recipes['average rating'].describe())
```

```
count    234429.000000
mean          4.379706
std           0.855994
min           0.000000
25%           4.000000
50%           4.666667
75%           5.000000
max           5.000000
Name: average rating, dtype: float64
```

In [13]:
```
fig2 = px.histogram(avg_recipes, x='average rating', title='Average Rating D
fig2.show()
fig2.write_html('assets/avgrating_dis.html', include_plotlyjs='cdn')
```



From this average rating distribution map, most of the ratings are concentrated near the position of 5 points, especially the number of 5 points is far more than other ratings, which indicates that the overall satisfaction of users is high, or there is a phenomenon of

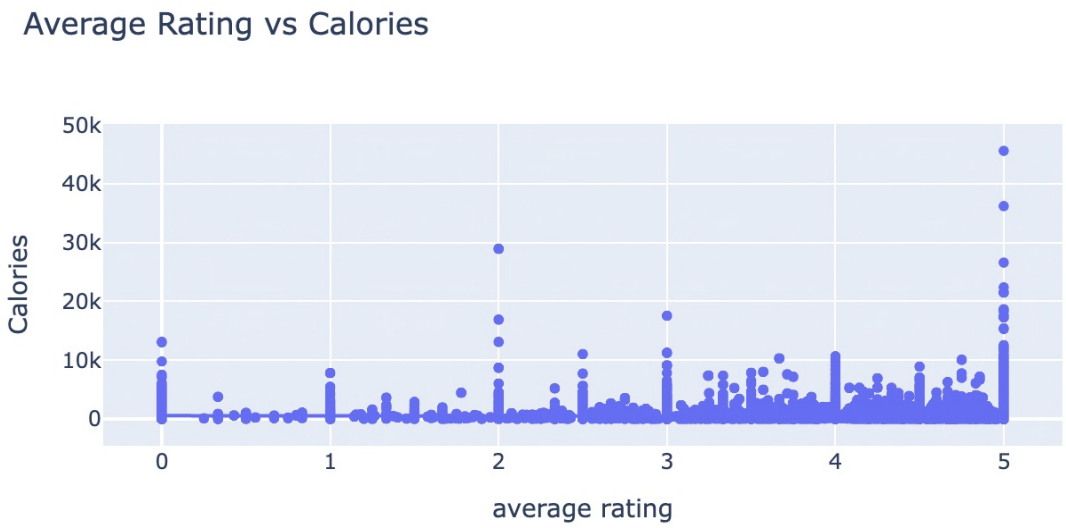
high ratings. Low ratings, such as below 2, are relatively rare, indicating that very few users give extremely low scores. This distribution may reflect the influence of rating bias or scoring mechanism. In addition, there were small spikes in certain scores such as 4 and 3. It is recommended to further analyze the time trend of review content or ratings to gain a deeper understanding of user feedback and optimize the rating mechanism to more accurately reflect the user experience.

Bivariate Analysis

Now let's take a look at the relationship between menu average ratings and calories.

Scatter of Average rating and Calories

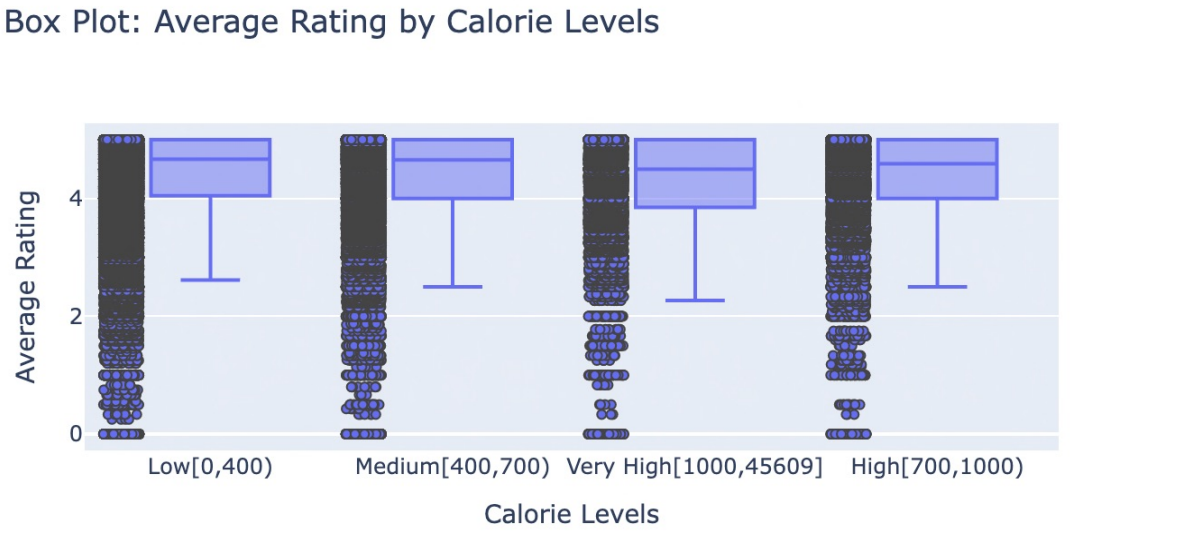
In [15]:
```
# average rating and calories
fig3 = px.scatter(avg_recipes,
                  x='average rating',
                  y='calories',
                  title='Average Rating vs Calories',
                  labels={'rating': 'Average Rating', 'calories': 'Calories'
                  trendline='ols'
                 )
fig3.show()
fig3.write_html('assets/avgrating_calories.html', include_plotlyjs='cdn')
```



Observation results: Most recipes are low in calories, concentrated in the 0 to 10,000 range. A large number of ratings are concentrated between 4 and 5 points. As the score approached 5, some recipes appeared that were very high in calories (over 20,000 calories). Recipes with low ratings (near 0 to 1) usually have lower calorie values. Conclusion: Highly rated recipes are sometimes accompanied by extremely high calorie content. Low-rated recipes are rarely seen in the high-calorie range, which may indicate that people are less receptive to low-rated, high-calorie recipes. The scatter plot shows that recipes with scores above 4 are mainstream, but the calorie distribution is very wide.

In [16]:
```
avg_recipes['calorie_bins'] = pd.cut(avg_recipes['calories'], bins=[0, 400,
fig4 = px.box(
    avg_recipes,
    x='calorie_bins',
    y='average rating',
    title='Box Plot: Average Rating by Calorie Levels',
    labels={'calorie_bins': 'Calorie Levels', 'average rating': 'Average Rat
    template='plotly',
    points='all'  # Show all points
)
fig4.update_traces(marker=dict(size=5, line=dict(width=1)))
```

```
fig4.show()
fig4.write_html('assets/avgrating_calories_box.html', include_plotlyjs='cdn'
```



Observation results: The recipes were divided into four calorie levels: Low (0-400), Medium (400-700), High (700-1000), and Very High (1000 +). The distribution of scores for each calorie category was concentrated in the higher rating range, with most scores around 4 points. The median scores for each calorie category were very close, with little difference in scores. Across all calorie levels, there were some low-scoring outliers. Conclusion: The relationship between calorie content and ratings did not appear to be significant, and the distribution of ratings for recipes with different calorie levels was relatively consistent. Both low - and high-calorie recipes are likely to receive high ratings, suggesting that ratings may depend more on other aspects of the recipe (e.g. taste, ingredients, cooking methods, etc.).

Interesting polymerization

Cooking Time outlier

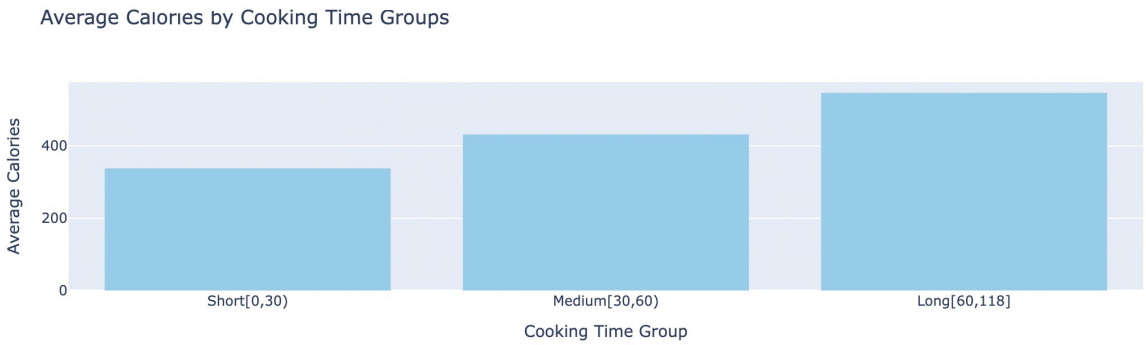
In [17]:
```
q_1 = avg_recipes['minutes'].quantile(0.25)
q_3 = avg_recipes['minutes'].quantile(0.75)
iqr= q_3 - q_1
lower_bound = q_1 - 1.5 * iqr
upper_bound = q_3 + 1.5 * iqr
no_outliers = avg_recipes[(avg_recipes['minutes'] > lower_bound) & (avg_reci
```

Fractional analysis

In [30]:
```
bins = [0, 30, 60, no_outliers['minutes'].max()]
labels = ['Short[0,30)', 'Medium[30,60)', 'Long[60,118]']
no_outliers['time_group'] = pd.cut(no_outliers['minutes'], bins=bins, labels

# Average calories by group
grouped = no_outliers.groupby('time_group')['calories'].mean().reset_index()

# Create a bar chart using Plotly
fig5 = px.bar(
    grouped,
    x='time_group',
    y='calories',
    title="Average Calories by Cooking Time Groups",
    labels={'time_group': 'Cooking Time Group', 'calories': 'Average Calorie
    color_discrete_sequence=['skyblue']  # Optional: Set color
)
fig5.show()
# Show the chart in the notebook
fig5.write_html('assets/avgrating_minutes.html', include_plotlyjs='cdn')
```



Possible conclusion:

1.The longer the cooking time, the higher the average calorie: This may indicate that recipes that take longer to cook often contain richer or more complex ingredients, resulting in higher calories.

2.Recipes that cook for shorter periods of time are lower in calories: Recipes with short cooking times may favor light meals, simple dishes, or healthy recipes.

Step 3: Assessment of Missingness

Missingness Dependency

In [31]:
```
avg_recipes['rating_missing'] = avg_recipes['rating'].isna()
```

In [32]:
```
def permutation_test(df, column, num_permutations=1000):
    # Calculate the mean difference of the actual group
    actual_mean_diff = df[df['rating_missing']][column].mean() - df[~df['rat
```

```
    # Store the mean difference of the replacement sample

    perm_diffs = []

    for _ in range(num_permutations):
        # replaces the 'rating_missing' flag
        shuffled_missing = df['rating_missing'].sample(frac=1, replace=False
        perm_diff = df[shuffled_missing][column].mean() - df[~shuffled_missi
        perm_diffs.append(perm_diff)

    return actual_mean_diff, perm_diffs
```

Rating and Cooking time

In [45]:
```
actual_mean_diff, perm_diffs = permutation_test(avg_recipes, 'minutes')

print(f"Actual Mean Difference: {actual_mean_diff}")
```

```
Actual Mean Difference: -61.78992697118092
```

In [46]:
```
p_values = np.mean(np.abs(perm_diffs) >= np.abs(actual_mean_diff))
p_values
```

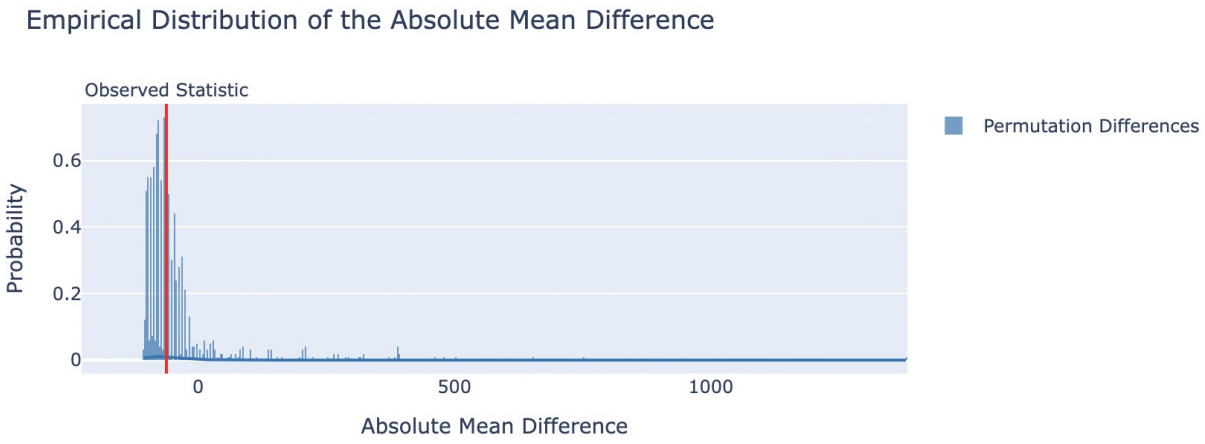
Out[46]:
```
np.float64(0.72)
```

In [40]:
```
fig6 = ff.create_distplot([perm_diffs], group_labels=['Permutation Differenc
fig6.add_vline(x=actual_mean_diff, line_color='red', line_width=2, annotatic


fig6.update_layout(
    title="Empirical Distribution of the Absolute Mean Difference",
    xaxis_title="Absolute Mean Difference",
    yaxis_title="Probability"
)


fig6.show()


fig6.write_html('assets/rating_minutes.html', include_plotlyjs='cdn')
```



Rating and saturated_fat

In [53]:
```
actual_mean_diff, perm_diffs = permutation_test(avg_recipes, 'saturated_fat'

print(f"Actual Mean Difference: {actual_mean_diff}")
```

```
Actual Mean Difference: 85.501731875032
```

In [54]:
```
p_values = np.mean(np.abs(perm_diffs) >= np.abs(actual_mean_diff))
p_values
```

Out[54]:
```
np.float64(0.049)
```

In [55]:
```
fig7 = px.histogram(
    x=perm_diffs,
    nbins=50,
    title="Empirical Distribution of the<br>Absolute Mean Difference in Prop
    labels={'x': "Absolute Mean Difference", 'y': "Probability"},
    color_discrete_sequence=['blue']
)
```

```
fig7.add_vline(x=actual_mean_diff, line_color='red', line_width=2, annotatic

fig7.update_layout(showlegend=True)

fig7.show()

fig7.write_html('assets/rating_fat.html', include_plotlyjs='cdn')
```

Step 4: Hypothesis Testing

Pearson correlation coefficient was used for hypothesis testing

We'll examine the relationship between calories and rating.

The hypothesis is as follows:

Null Hypothesis: Calories have nothing to do with average ratings.

Alternative Hypothesis: Calories are related to average ratings.

In [57]:
```
# Delete missing values in 'calories' and 'rating'
cleaned_data = avg_recipes[['calories', 'rating']].dropna()
```

In [58]:
```
corr, p_value = pearsonr(cleaned_data['calories'], cleaned_data['rating'])

print(f"Pearson Correlation Coefficient: {corr}")
print(f"P-Value: {p_value}")
```

```
Pearson Correlation Coefficient: -0.029353990804797694
P-Value: 7.33898944571452e-46
```

In [59]:
```
p_value <=0.05
```

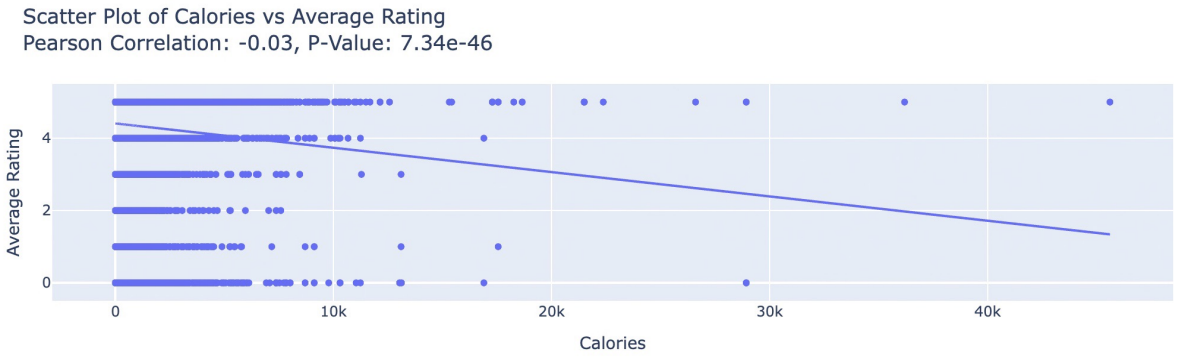
Out[59]: np.True_

In [60]:
```
fig8 = px.scatter(
    cleaned_data,
    x='calories',
    y='rating',
    trendline='ols',
    title=f"Scatter Plot of Calories vs Average Rating<br>Pearson Correlatic
    labels={'calories': 'Calories', 'rating': 'Average Rating'}
)


fig8.update_xaxes(showgrid=True)
fig8.update_yaxes(showgrid=True)


fig8.show()


fig8.write_html('assets/calories_vs_rating.html', include_plotlyjs='cdn')
```



Conclusion

1.Reject the null hypothesis (Calories and Average Ratings are independent) :

Since the P-value is very small, we reject the null hypothesis and suggest that there is some statistical correlation between calories and ratings.

2.The correlation is weak:

Although statistically significant, the correlation coefficient is only -0.029, indicating that the association is insignificant in practice. Calorie content had a negligible effect on ratings.

Step 5: Framing a Prediction Problem

Prediction Problem： the calorie content of a recipe

We plan to predict the calorie content of a recipe, which is a regression problem since calories are a continuous variable. Our goal is to build a regression model that predicts calorie values based on the ingredients of a recipe.

Prediction Problem Type: Regression 1.Response Variable: The variable we aim to predict is the calorie content of the recipe. 2.Reason for Selection: Calorie content directly reflects the energy level of a recipe, which is crucial for assessing its healthiness. Understanding the calories in a recipe helps people make informed dietary choices and avoid excessive calorie intake.

Evaluation Metrics We will use the following metrics to evaluate the performance of the regression model: 1.Mean Squared Error (MSE)

Reason: MSE measures the average squared difference between predicted and actual values. It is effective in capturing large errors, penalizing predictions that deviate significantly from actual values, and helps optimize the model's accuracy.

2.Root Mean Squared Error (RMSE)

Reason: RMSE is the square root of MSE, maintaining the same unit as the response variable, making it easier to interpret. It provides a clear measure of the magnitude of prediction errors. We chose these regression metrics instead of classification metrics (e.g., accuracy or F1 score) because our response variable is continuous rather than categorical. These metrics effectively quantify the difference between continuous predicted values and actual values.

By building this regression model, we aim to help users quickly understand the calorie content of recipes, enabling them to make healthier dietary choices.

Step 6: Baseline Model

In [67]:
```
def classify_cooking_time(minutes):
    if minutes < 30:
        return 'short'
    elif 30 <= minutes <= 60:
        return 'medium'
    else:
        return 'long'

avg_recipes['cooking_duration'] = avg_recipes['minutes'].apply(classify_cook
avg_recipes.head()
print(avg_recipes[['minutes', 'cooking_duration']].head())
```

```
   minutes cooking_duration
0       50           medium
1       50           medium
2       50           medium
3       55           medium
4       45           medium
```

In [68]:
```
avg_recipes = pd.get_dummies(avg_recipes, columns=['cooking_duration'],drop_
print(avg_recipes.head())
```

```
                                name      id  minutes  contributor_id  \
0  impossible macaroni and cheese pie  275022       50          531768
1  impossible macaroni and cheese pie  275022       50          531768
2  impossible macaroni and cheese pie  275022       50          531768
3              impossible rhubarb pie  275024       55          531768
4              impossible seafood pie  275026       45          531768

    submitted                                               tags  n_steps  \
0  2008-01-01  ['60-minutes-or-less', 'time-to-make', 'course...       11
1  2008-01-01  ['60-minutes-or-less', 'time-to-make', 'course...       11
2  2008-01-01  ['60-minutes-or-less', 'time-to-make', 'course...       11
3  2008-01-01  ['60-minutes-or-less', 'time-to-make', 'course...        6
4  2008-01-01  ['60-minutes-or-less', 'time-to-make', 'course...        7

                                              steps  \
0  ['heat oven to 400 degrees fahrenheit', 'greas...
1  ['heat oven to 400 degrees fahrenheit', 'greas...
2  ['heat oven to 400 degrees fahrenheit', 'greas...
3  ['heat oven to 375 degrees', 'grease 10" pan ,...
4  ['preheat oven to 400f', 'lightly grease large...

                                         description  \
0  one of my mom's favorite bisquick recipes. thi...
1  one of my mom's favorite bisquick recipes. thi...
2  one of my mom's favorite bisquick recipes. thi...
3  a childhood favorite of mine. my mom loved it ...
4  this is an oldie but a goodie. mom's stand by ...

                                        ingredients  ...  sugar  sodium  \
0  ['cheddar cheese', 'macaroni', 'milk', 'eggs',...  ...    7.0    24.0
1  ['cheddar cheese', 'macaroni', 'milk', 'eggs',...  ...    7.0    24.0
2  ['cheddar cheese', 'macaroni', 'milk', 'eggs',...  ...    7.0    24.0
3  ['rhubarb', 'eggs', 'bisquick', 'butter', 'sal...  ...  208.0    13.0
4  ['frozen crabmeat', 'sharp cheddar cheese', 'c...  ...   12.0    27.0

   protein saturated_fat  carbohydrates calorie_bins  rating_missing  \
0     41.0          62.0            8.0   Low[0,400)           False
1     41.0          62.0            8.0   Low[0,400)           False
2     41.0          62.0            8.0   Low[0,400)           False
3     13.0          30.0           20.0   Low[0,400)           False
4     37.0          51.0            5.0   Low[0,400)           False

   cooking_duration_long  cooking_duration_medium  cooking_duration_short
0                  False                     True                   False
1                  False                     True                   False
2                  False                     True                   False
3                  False                     True                   False
4                  False                     True                   False

[5 rows x 29 columns]
```

In [69]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
avg_recipes['cooking_duration'] = avg_recipes['minutes'].apply(classify_cook
# Tag encoding for "cooking_duration"
avg_recipes['cooking_duration_encoded'] = le.fit_transform(avg_recipes['cook
```

```
print(avg_recipes[['cooking_duration', 'cooking_duration_encoded']].head())
```

```
  cooking_duration  cooking_duration_encoded
0           medium                         1
1           medium                         1
2           medium                         1
3           medium                         1
4           medium                         1
```

In [70]:
```
features = ['n_steps', 'cooking_duration_encoded','total_fat', 'sugar']
X = avg_recipes[features]
y = avg_recipes['calories']

# Divide the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Build Pipeline (standardized model training)

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Training model
pipeline.fit(X_train, y_train)

# Predictive test set
y_pred = pipeline.predict(X_test)

# Evaluate model performance
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 14117.43
Root Mean Squared Error (RMSE): 118.82
R² Score: 0.96
```

The baseline model uses a Random Forest Regressor to predict recipe calorie content based on four features: n_steps, total_fat, sugar, and cooking_duration_encoded (an ordinal feature representing cooking duration categorized as long, medium, or short). The features were scaled using StandardScaler within a Pipeline. The model achieved a Mean Squared Error (MSE) of 14117.43, a Root Mean Squared Error (RMSE) of 118.82, and an R^2 Score of 0.96, indicating that it explains 96% of the variance in calorie content. While the high R^2 score shows strong predictive performance, the RMSE suggests room for improvement, particularly for recipes with extreme calorie values. Adding more relevant features and tuning model hyperparameters could further enhance the model's accuracy.

Step 7: Final Model

Improve with adding more features

In [71]:
```
features = ['total_fat', 'sugar', 'sodium', 'protein', 'saturated_fat', 'car
X = avg_recipes[features]
y = avg_recipes['calories']

# Divide the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Build Pipeline (standardized model training)

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Training model
pipeline.fit(X_train, y_train)

# Predictive test set
y_pred = pipeline.predict(X_test)

# Evaluate model performance
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 1249.69
Root Mean Squared Error (RMSE): 35.35
R² Score: 1.00
```

Step 8: Fairness Analysis

In [72]:
```
## cal RMSE
short_duration = X_test[X_test['cooking_duration_encoded'] == 2]
long_duration = X_test[X_test['cooking_duration_encoded'] == 0]

# Extract the corresponding true value
y_short = y_test[short_duration.index]
y_long = y_test[long_duration.index]

# Use the final model to make predictions

y_pred_short = pipeline.predict(short_duration)
y_pred_long = pipeline.predict(long_duration)
```

```
# Calculate RMSE for both groups
rmse_short = np.sqrt(mean_squared_error(y_short, y_pred_short))
rmse_long = np.sqrt(mean_squared_error(y_long, y_pred_long))

print(f"RMSE for Short Duration Recipes: {rmse_short:.2f}")
print(f"RMSE for Long Duration Recipes: {rmse_long:.2f}")
```

```
RMSE for Short Duration Recipes: 33.18
RMSE for Long Duration Recipes: 55.22
```

Set null hypothesis and alternative hypothesis

Zero hypothesis (H_0) : The model has no significant difference in performance between short - and long-duration diets, and the difference in RMSE is due to random chance.

Alternative Hypothesis (H_1) : The model RMSE for short - and long-form recipes was significantly higher than the RMSE for long - form recipes, indicating a worse model performance for short - and long-form recipes.

The P-Value is less than 0.05, it indicates that we can reject the null hypothesis, and there is a significant difference in the performance of the model on short-long and long-long recipes, which may be unfair.

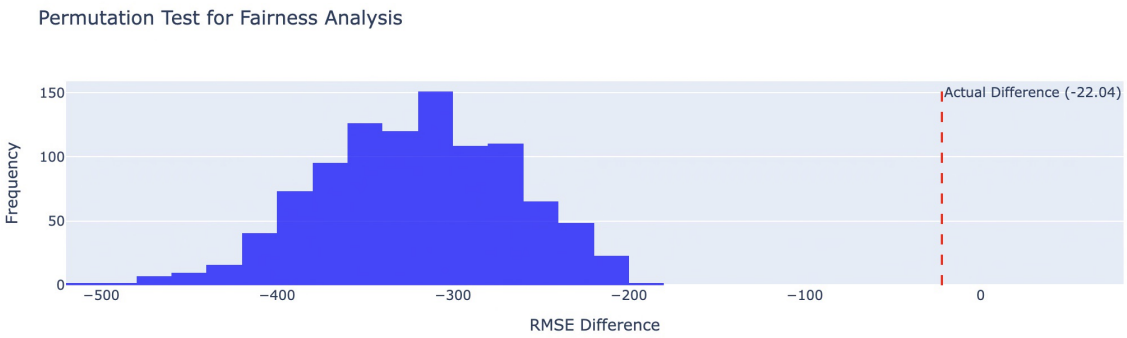
In [75]:
```
import plotly.graph_objects as go
fig9 = go.Figure()

fig9.add_trace(go.Histogram(
    x=permuted_differences,
    nbinsx=30,
    name='Permuted Differences',
    marker_color='blue',
    opacity=0.7
))


fig9.add_vline(
    x=actual_difference,
    line_dash='dash',
    line_color='red',
    line_width=2,
    annotation_text=f'Actual Difference ({actual_difference:.2f})',
    annotation_position='top right'
)


fig9.update_layout(
    title='Permutation Test for Fairness Analysis',
    xaxis_title='RMSE Difference',
    yaxis_title='Frequency',
    legend=dict(x=0.7, y=0.95),
    template='plotly'
)
```

```
fig9.show()
fig9.write_html('assets/permutation_test.html', include_plotlyjs='cdn')
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: