Predicting Recipe Ratings

Name(s):

- Richard Lin
- Francisco Downey

Website Link: https://rhlin2001.github.io/predicting-recipe-ratings/

In [1]:
```
# Basic imports
import pandas as pd
import numpy as np
from pathlib import Path

# Analysis imports
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, QuantileTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

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

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

!pip install tabulate
```

```
Requirement already satisfied: tabulate in /opt/anaconda3/envs/dsc80/lib/python3.
8/site-packages (0.9.0)
```

Step 1: Introduction

Question: What type of recipes have high ratings?

Step 2: Data Cleaning and Exploratory Data Analysis

Data Cleaning:

In [2]:
```
# Import data
interactions = pd.read_csv('food_data/RAW_interactions.csv')
recipes = pd.read_csv('food_data/RAW_recipes.csv')
```

In [3]:
```
# Merge recipes and interactions
merge_df = recipes.merge(
    interactions,
    how='left',
    left_on='id',
    right_on='recipe_id'
)

# Convert to correct data types
merge_df['rating'] = (
    merge_df['rating']
    .apply(lambda x: np.nan if x == 0 else x)
)
merge_df['nutrition'] = (
    merge_df['nutrition']
    .apply(lambda x: list(map(float, x[1:-1].split(', '))))
)
merge_df['date'] = pd.to_datetime(merge_df['date'])
merge_df['submitted'] = pd.to_datetime(merge_df['submitted'])

# Create new columns
merge_df['avg_rating'] = (
    merge_df.groupby('recipe_id')['rating']
    .transform('mean')
)

nutrition = np.array(merge_df['nutrition'].tolist())
merge_df['calories'] = nutrition[:, 0]
merge_df['total_fat'] = nutrition[:, 1]
merge_df['sugar'] = nutrition[:, 2]
merge_df['sodium'] = nutrition[:, 3]
merge_df['protein'] = nutrition[:, 4]
merge_df['saturated_fat'] = nutrition[:, 5]
merge_df['carbohydrates'] = nutrition[:, 6]
```

In [4]:
```
merge_df.head()
```

Out[4]:

	name	id	minutes	contributor_id	...	sodium	protein	saturated_fat	carbohyd
0	1 brownies in the world best ever	333281	40	985201	...	3.0	3.0	19.0	
1	1 in canada chocolate chip cookies	453467	45	1848091	...	22.0	13.0	51.0	
2	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	
3	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	
4	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	

5 rows × 25 columns

In [5]:

```
# Drop outliers for numerical columns using IQR method
numerical_cols = [
    'minutes',
    'n_steps',
    'n_ingredients',
    'calories',
    'total_fat',
    'sugar',
    'sodium',
    'protein',
    'saturated_fat',
    'carbohydrates'
]

clean_merge = merge_df.copy()
for col in numerical_cols:
    q1 = merge_df[col].quantile(0.25)
    q3 = merge_df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    clean_merge = clean_merge[
        (clean_merge[col] >= lower_bound) & (clean_merge[col] <= upper_bound)
    ]
```

In [6]:

```
clean_merge.head()
```

Out[6]:

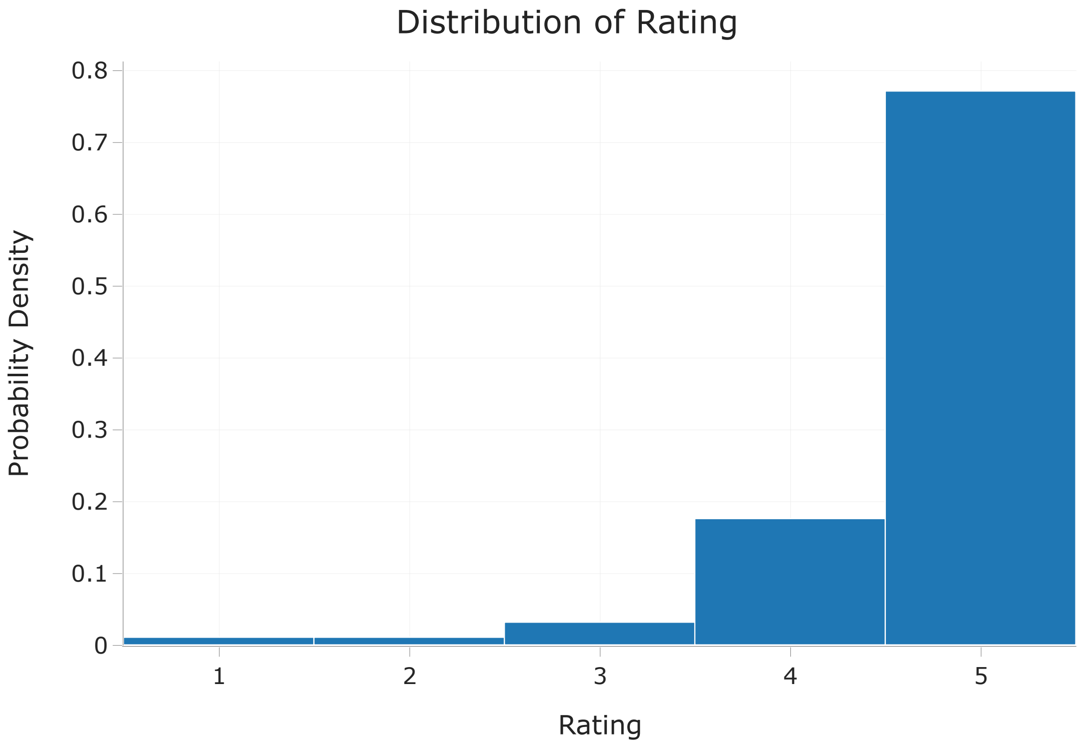
	name	id	minutes	contributor_id	...	sodium	protein	saturated_fat	carbohydr
0	1 brownies in the world best ever	333281	40	985201	...	3.0	3.0	19.0	
2	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	
3	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	
4	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	
5	412 broccoli casserole	306168	40	50969	...	32.0	22.0	36.0	

5 rows × 25 columns

Univariate Analysis:

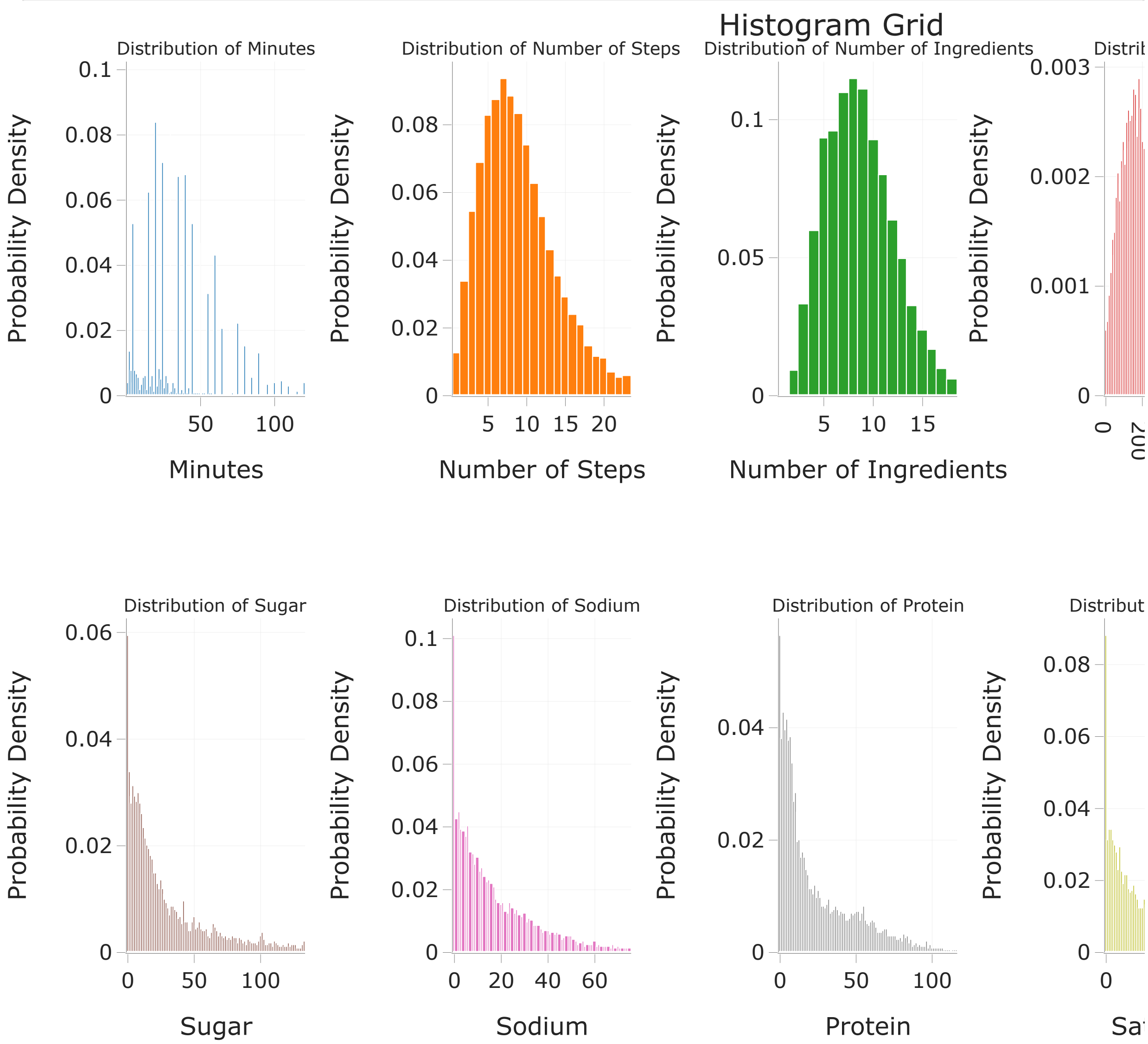
In [7]:

```
fig = px.histogram(
    clean_merge,
    x='rating',
    histnorm='probability density',
    title='Distribution of Rating'
)

fig.update_layout(
    xaxis_title='Rating',
    yaxis_title='Probability Density'
)

fig.show()

fig.write_html('assets/rating_hist.html', include_plotlyjs='cdn')
```



In [8]:
```
col_titles = [
    'Minutes',
    'Number of Steps',
    'Number of Ingredients',
    'Calories',
    'Total Fat',
    'Sugar',
    'Sodium',
    'Protein',
    'Saturated Fat',
    'Carbohydrates'
]
subplot_titles = tuple(
    [f"Distribution of {col_title}" for col_title in col_titles]
)

fig = make_subplots(
    rows=2,
    cols=5,
    subplot_titles=subplot_titles,
    horizontal_spacing=0.1
)

num_col = 0
for row in [1, 2]:
    for col in [1, 2, 3, 4, 5]:
        fig.add_trace(
            go.Histogram(
                x=clean_merge[numerical_cols[num_col]],
                histnorm='probability density'
            ),
            row=row,
            col=col
        )
        fig.update_xaxes(title_text=col_titles[num_col], row=row, col=col)
```

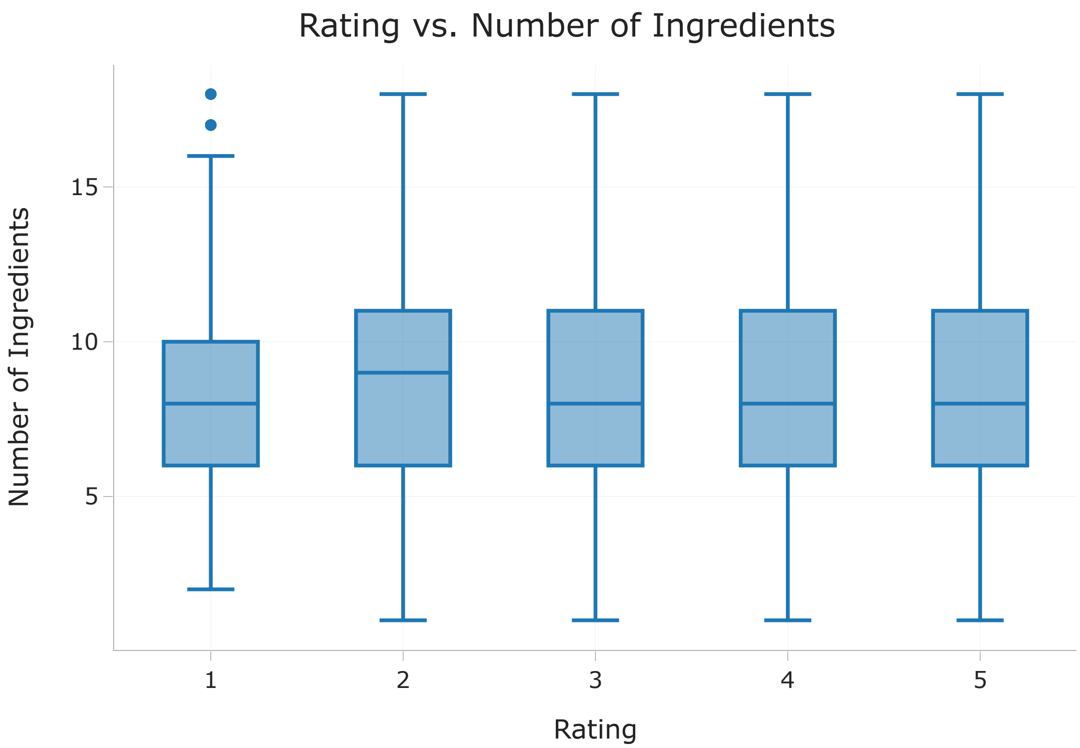
```
        fig.update_yaxes(title_text='Probability Density', row=row, col=col)
        num_col += 1

fig.update_layout(
    title_text='Histogram Grid',
    annotations=[
        {'font': {'size': 10}} if annotation['text'] in subplot_titles
        else annotation for annotation in fig['layout']['annotations']
    ],
    height=600,
    width=950,
    showlegend=False
)

fig.show()

fig.write_html('assets/hist_grid.html', include_plotlyjs='cdn')
```



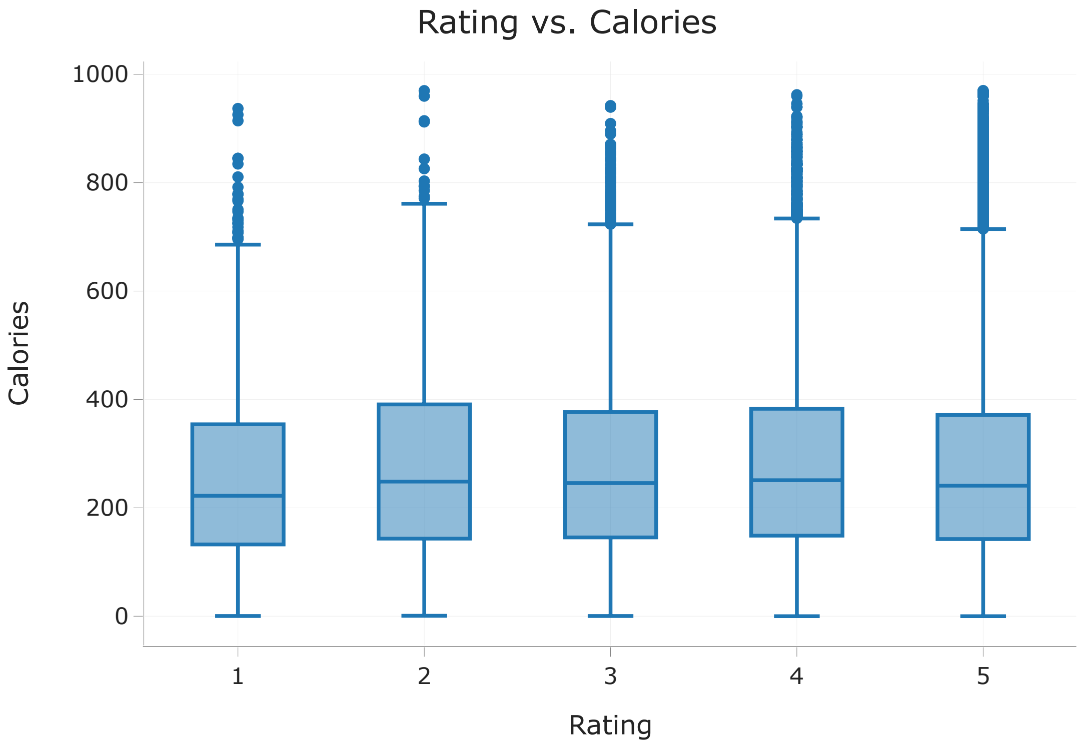
Bivariate Analysis:

In [9]:
```
fig = px.box(
    clean_merge,
```

```
    x='rating',
    y='n_ingredients',
    title='Rating vs. Number of Ingredients'
)

fig.update_layout(
    xaxis_title='Rating',
    yaxis_title='Number of Ingredients'
)

fig.show()

fig.write_html('assets/rating_vs_ingredients.html', include_plotlyjs='cdn')
```



In [10]:
```
fig = px.box(
    clean_merge,
    x='rating',
    y='calories',
    title='Rating vs. Calories'
)

fig.update_layout(
    xaxis_title='Rating',
    yaxis_title='Calories'
)

fig.show()

fig.write_html('assets/rating_vs_calories.html', include_plotlyjs='cdn')
```



Interesting Aggregates:

In [11]:
```
avg_numeric_by_rating = clean_merge.groupby('rating')[numerical_cols].mean()
avg_numeric_by_rating
```

Out[11]:

rating	minutes	n_steps	n_ingredients	calories	...	sodium	protein	saturated_fat	carbol
1.0	36.28	9.33	8.43	253.14	...	15.16	20.22	23.40	
2.0	37.26	9.44	8.74	278.02	...	16.62	24.04	24.51	
3.0	36.08	9.02	8.76	275.18	...	17.03	25.15	23.77	
4.0	34.85	8.84	8.73	279.74	...	17.45	26.12	23.78	
5.0	34.23	8.91	8.60	270.35	...	16.92	23.77	24.46	

5 rows × 10 columns

Step 3: Assessment of Missingness

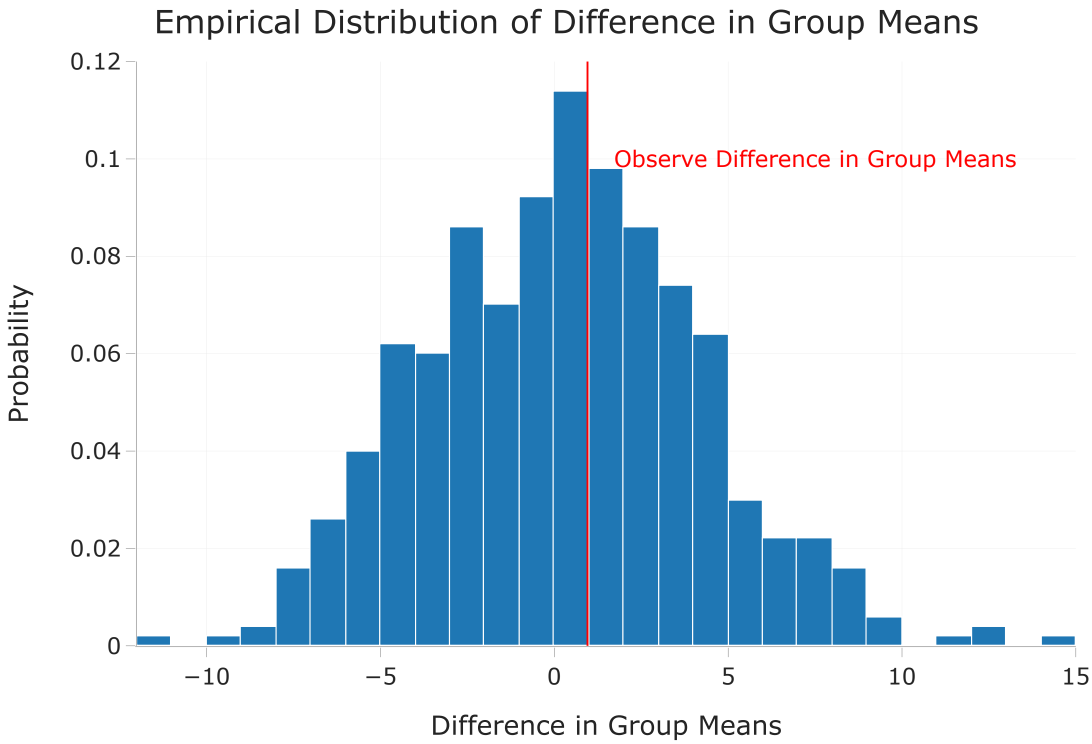
Permutation Test 1:

Null: The distribution of 'minutes' when 'review' is missing is the same as the distribution of 'minutes' when 'review' is not missing.

Alternative: The missingness of 'review' depends on 'minutes'.

In [12]:
```
missing_assess = clean_merge.assign(missing_review=clean_merge['review'].isna())

diffs = []
for _ in range(500):
    shuffle = (
        missing_assess
        .assign(shuffle_minutes=np.random.permutation(missing_assess['minutes']))
    )
    group_means = (
        shuffle
        .groupby('missing_review')
        .mean()
        .loc[:, 'shuffle_minutes']
    )
    diff = group_means.loc[True] - group_means.loc[False]
    diffs.append(diff)
obs_group_means = (
    missing_assess
    .groupby('missing_review')
    .mean()
    .loc[:, 'minutes']
)
obs = obs_group_means.loc[True] - obs_group_means.loc[False]
p_value = (np.array(diffs) >= obs).mean()
print(f"P-Value: {p_value}")

fig = px.histogram(
    pd.DataFrame(diffs),
    x=0,
    nbins=50,
    histnorm='probability',
    title='Empirical Distribution of Difference in Group Means'
)

fig.add_vline(x=obs, line_color='red', line_width=1, opacity=1)
fig.add_annotation(
    text=f"<span style='color:red'>Observe Difference in Group Means</span>",
    x=7.5,
    showarrow=False,
    y=0.1
)

fig.update_layout(
    xaxis_title='Difference in Group Means',
    yaxis_title='Probability'
)

fig.show()

fig.write_html('assets/emp_dist_1.html', include_plotlyjs='cdn')
```

```
P-Value: 0.438
```



Permutation Test 2:

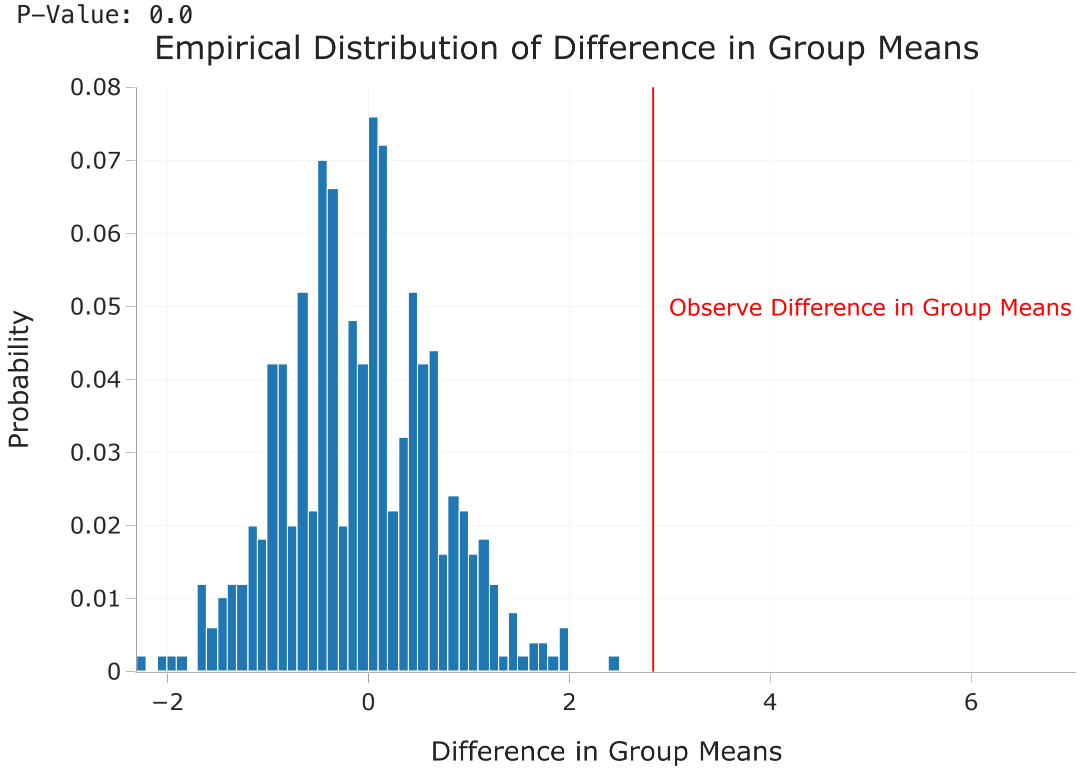
Null: The distribution of 'n_steps' when 'review' is missing is the same as the distribution of 'minutes' when 'review' is not missing.

Alternative: The missingness of 'review' depends on 'n_steps'.

In [13]:
```
missing_assess = clean_merge.assign(missing_review=clean_merge['review'].isna())

diffs = []
for _ in range(500):
    shuffle = (
        missing_assess
        .assign(shuffle_steps=np.random.permutation(missing_assess['n_steps']))
    )
    group_means = (
        shuffle
        .groupby('missing_review')
        .mean()
        .loc[:, 'shuffle_steps']
    )
    diff = group_means.loc[True] - group_means.loc[False]
    diffs.append(diff)
obs_group_means = (
    missing_assess
    .groupby('missing_review')
    .mean()
    .loc[:, 'n_steps']
)
obs = obs_group_means.loc[True] - obs_group_means.loc[False]
p_value = (np.array(diffs) >= obs).mean()
print(f"P-Value: {p_value}")
```

```
fig = px.histogram(
    pd.DataFrame(diffs),
    x=0,
    nbins=50,
    histnorm='probability',
    title='Empirical Distribution of Difference in Group Means'
)

fig.add_vline(x=obs, line_color='red', line_width=1, opacity=1)
fig.add_annotation(
    text=f"<span style='color:red'>Observe Difference in Group Means</span>",
    x=5.0,
    showarrow=False,
    y=0.05
)

fig.update_layout(
    xaxis_title='Difference in Group Means',
    yaxis_title='Probability'
)

fig.show()

fig.write_html('assets/emp_dist_2.html', include_plotlyjs='cdn')
```

```
P-Value: 0.0
```



Step 4: Hypothesis Testing

Null: High rating and low rating labels have no relationship to 'calories'.

Alternative: The 'calories' of high rating recipes is greater than that of low rating recipes.

In [14]:
```
permutation_test = clean_merge.assign(high_rating=clean_merge['rating'] >= 4.0)

diffs = []
for _ in range(500):
    shuffle = (
        permutation_test
        .assign(
            shuffle_calories=np.random.permutation(permutation_test['calories'])
        )
    )
    group_means = (
        shuffle
        .groupby('high_rating')
        .mean()
        .loc[:, 'shuffle_calories']
    )
    diff = group_means.loc[True] - group_means.loc[False]
    diffs.append(diff)
obs_group_means = (
    permutation_test
    .groupby('high_rating')
    .mean()
    .loc[:, 'calories']
)
obs = obs_group_means.loc[True] - obs_group_means.loc[False]
p_value = (np.array(diffs) >= obs).mean()
print(f"P-Value: {p_value}")
```

```
P-Value: 0.322
```

Step 5: Framing a Prediction Problem

Prediction Problem: Predict ratings of recipes.

Step 6: Baseline Model

In [15]:
```
X = clean_merge.dropna()
y = clean_merge.dropna()['rating']
X_train, X_test, y_train, y_test = (
    train_test_split(X, y, test_size=0.25, random_state=80)
)

preproc = ColumnTransformer(
    transformers=[(
        'quantile',
        QuantileTransformer(output_distribution='normal'),
        ['total_fat', 'sugar', 'carbohydrates']
    )],
    remainder='drop'
)

pipe = Pipeline([
    ('preproc', preproc),
    ('log-reg', LogisticRegression())
])
```

```
pipe.fit(X_train, y_train)
```

Out[15]:

```
Pipeline(steps=[('preproc',
                 ColumnTransformer(transformers=[('quantile',
                                                  QuantileTransformer(output_dist
ribution='normal'),
                                                  ['total_fat', 'sugar',
                                                   'carbohydrates'])])),
                ('log-reg', LogisticRegression())])
```

In [16]:

```
print(
    f"Training Accuracy: {pipe.score(X_train, y_train)}",
    f"Testing Accuracy: {pipe.score(X_test, y_test)}"
)
```

```
Training Accuracy: 0.772878795770613 Testing Accuracy: 0.7701502643298003
```

In [17]:

```
y_pred = pipe.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, zero_division=0))
```

```
[[    0     0     0     0   415]
 [    0     0     0     0   360]
 [    0     0     0     0  1226]
 [    0     0     0     0  6825]
 [    0     0     0     0 29573]]
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00       415
         2.0       0.00      0.00      0.00       360
         3.0       0.00      0.00      0.00      1226
         4.0       0.00      0.00      0.00      6825
         5.0       0.77      1.00      0.87     29573

    accuracy                           0.77     38399
   macro avg       0.15      0.20      0.17     38399
weighted avg       0.59      0.77      0.67     38399
```

Step 7: Final Model

In [18]:

```
preproc = ColumnTransformer(
    transformers=[
        ('tfidf', TfidfVectorizer(), 'review'),
        ('scaler', StandardScaler(), ['minutes', 'n_steps']),
        ('quantile',
         QuantileTransformer(output_distribution='normal'),
         ['total_fat', 'sugar', 'carbohydrates'])
    ],
    remainder='drop'
)

# Best parameter by grid search
pipe = Pipeline([
    ('preproc', preproc),
    ('log-reg', LogisticRegression(max_iter=1000))
])
```

```
# hyperparameters = {
#     'log-reg__max_iter': [500, 1000, 1500]
# }

# grid = GridSearchCV(
#     pipe,
#     n_jobs=-1,
#     param_grid=hyperparameters,
#     cv=3,
# )

# grid.fit(X_train, y_train)
# grid.best_params_

pipe.fit(X_train, y_train)
```

Out[18]:
```
Pipeline(steps=[('preproc',
                 ColumnTransformer(transformers=[('tfidf', TfidfVectorizer(),
                                                  'review'),
                                                 ('scaler', StandardScaler(),
                                                  ['minutes', 'n_steps']),
                                                 ('quantile',
                                                  QuantileTransformer(output_dist
ribution='normal'),
                                                  ['total_fat', 'sugar',
                                                   'carbohydrates'])])),
                ('log-reg', LogisticRegression(max_iter=1000))])
```

In [19]:
```
print(
    f"Training Accuracy: {pipe.score(X_train, y_train)}",
    f"Testing Accuracy: {pipe.score(X_test, y_test)}"
)
```

```
Training Accuracy: 0.8383335937635641 Testing Accuracy: 0.811531550300789
```

In [20]:
```
y_pred = pipe.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[  184    16    41    38   136]
 [   39    25   107    86   103]
 [   19    24   323   491   369]
 [   10     2   126  2142  4545]
 [    8     2    30  1045 28488]]
              precision    recall  f1-score   support

         1.0       0.71      0.44      0.55       415
         2.0       0.36      0.07      0.12       360
         3.0       0.52      0.26      0.35      1226
         4.0       0.56      0.31      0.40      6825
         5.0       0.85      0.96      0.90     29573

    accuracy                           0.81     38399
   macro avg       0.60      0.41      0.46     38399
weighted avg       0.78      0.81      0.78     38399
```

Step 8: Fairness Analysis

Null: Our model is fair. Its accuracy for reviews before 2013 and after 2013 are roughly the same, and any differences are due to random chance.

Alternative: Our model is unfair. Its accuracy for reviews before 2013 is lower than that of after 2013.

In [21]:
```
results = X_test
results['before_2013'] = results['date'] < pd.Timestamp('2013-01-01')
results['prediction'] = y_pred
results['rating'] = y_test

compute_acc = lambda x: accuracy_score(x['rating'], x['prediction'])

diff_acc = []
for _ in range(500):
    shuffle = (
        results[['before_2013', 'prediction', 'rating']]
        .assign(before_2013=np.random.permutation(results['before_2013']))
        .groupby('before_2013')
        .apply(compute_acc)
        .diff()
        .iloc[-1]
    )
    diff_acc.append(shuffle)
obs = (
    results[['before_2013', 'prediction', 'rating']]
    .groupby('before_2013')
    .apply(compute_acc)
    .diff()
    .iloc[-1]
)
p_value = (np.array(diff_acc) >= obs).mean()
print(f"P-Value: {p_value}")
```

```
P-Value: 0.104
```

In [ ]: