TrueFork

Are guilty-pleasure dishes truly tastier?

Name(s): Stephanie Anshell and Ved Panse

Website Link: https://stephaniepatriciaans.github.io/TrueFork/

In [1]:
```
import pandas as pd
import numpy as np
from pathlib import Path
import plotly.io as pio

pio.renderers.default = 'iframe'

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

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

In [2]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.compose import make_column_transformer, ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, QuantileTra
from sklearn.metrics import root_mean_squared_error
from sklearn.ensemble import RandomForestRegressor
```

Step 1: Introduction

Data

In [3]:
```
# Data recipes
recipes = pd.read_csv('data/RAW_recipes.csv')
recipes
```

Out[3]:

	name	id	minutes	contributor_id	...	steps	description
0	1 brownies in the world best ever	333281	40	985201	...	['heat the oven to 350f and arrange the rack i...	these are the most; chocolatey, moist, rich, d...
1	1 in canada chocolate chip cookies	453467	45	1848091	...	['pre-heat oven the 350 degrees f', 'in a mixi...	this is the recipe that we use at my school ca...
2	412 broccoli casserole	306168	40	50969	...	['preheat oven to 350 degrees', 'spray a 2 qua...	since there are already 411 recipes for brocco...
...	...	...	...	...	...	...	...
83779	zydeco ya ya deviled eggs	308080	40	37779	...	['in a bowl , combine the mashed yolks and may...	deviled eggs, cajun-style
83780	cookies by design cookies on a stick	298512	29	506822	...	['place melted butter in a large mixing bowl a...	i've heard of the 'cookies by design' company,...
83781	cookies by design sugar shortbread cookies	298509	20	506822	...	['whip sugar and shortening in a large bowl , ...	i've heard of the 'cookies by design' company,...

83782 rows × 12 columns

In [4]:

```
# Data interactions
interactions = pd.read_csv('data/RAW_interactions.csv')
interactions
```

Out[4]:

	user_id	recipe_id	date	rating	review
0	1293707	40893	2011-12-21	5	So simple, so delicious! Great for chilly fall...
1	126440	85009	2010-02-27	5	I made the Mexican topping and took it to bunk...
2	57222	85009	2011-10-01	5	Made the cheddar bacon topping, adding a sprin...
...	...	...	...	...	...
731924	157126	78003	2008-06-23	5	WOW! Sometimes I don't take the time to rate ...
731925	53932	78003	2009-01-11	4	Very good! I used regular port as well. The ...
731926	2001868099	78003	2017-12-18	5	I am so glad I googled and found this here. Th...

731927 rows × 5 columns

Question Identification

> Do unhealthy recipes receive higher average ratings than healthy ones?

Step 2: Data Cleaning and Exploratory Data Analysis

Cleaning the data

We'll add 2 columns:

1. `num_calories` : contains the number of calories, extracted from nutrition
2. `is_unhealthy` : a simple boolean column which states if that recipe contains a lot of calories (500 hard limit)

Since we will extensively be working only with `avg_rating` and `is_unhealthy` for our hypothesis testing, we can filter out missing values in **Step 4**, right before hypothesis testing so that we can efficienly analyze the missingness types for this dataset.

In [5]:
```
# Replace ratings of 0 with NaN, as 0 may indicate missing or invalid data
interactions['rating'] = interactions['rating'].replace(0, np.nan)
```

In [6]:
```python
# Left merge interactions into recipes on recipe ID
merged = recipes.merge(interactions, left_on='id', right_on='recipe_id', how

# Compute average rating per recipe ID
avg_ratings = merged.groupby('id')['rating'].mean()

# Add the average rating back to the original recipes DataFrame
recipes = recipes.set_index('id')
recipes['avg_rating'] = avg_ratings
recipes = recipes.reset_index()
recipes
```

Out[6]:

	id	name	minutes	contributor_id	...	description	ingredien
0	333281	1 brownies in the world best ever	40	985201	...	these are the most; chocolatey, moist, rich, d...	['bitterswe chocolat 'unsalt butter',
1	453467	1 in canada chocolate chip cookies	45	1848091	...	this is the recipe that we use at my school ca...	['wh suga 'brov sugar', 'sa 'marga
2	306168	412 broccoli casserole	40	50969	...	since there are already 411 recipes for brocco...	['froz broccoli cut 'cream chicken sou
...	...	...	...	...	...	...	
83779	308080	zydeco ya ya deviled eggs	40	37779	...	deviled eggs, cajun-style	['hard-cook egg 'mayonnais 'dijon must
83780	298512	cookies by design cookies on a stick	29	506822	...	i've heard of the 'cookies by design' company,...	['butte 'eagle bra condens milk', 'ligh
83781	298509	cookies by design sugar shortbread cookies	20	506822	...	i've heard of the 'cookies by design' company,...	['granulat suga 'shortenin 'eggs', 'f

83782 rows × 13 columns

In [7]:
```python
num_rows, num_cols = recipes.shape
num_rows, num_cols
```

Out[7]: (83782, 13)

In [8]:
```python
recipes.columns
```

Out[8]:
```
Index(['id', 'name', 'minutes', 'contributor_id', 'submitted', 'tags',
       'nutrition', 'n_steps', 'steps', 'description', 'ingredients',
       'n_ingredients', 'avg_rating'],
      dtype='object')
```

In [9]:
```
# Extract calories from the nutrition column
# 'nutrition' is a string that represents a list like: "[calories, fat, suga
recipes['calories'] = recipes['nutrition'].apply(lambda x: eval(x)[0] if pd.

# Define a recipe as unhealthy if it has more than 500 calories
recipes['is_unhealthy'] = recipes['calories'] > 500
recipes.head()
```

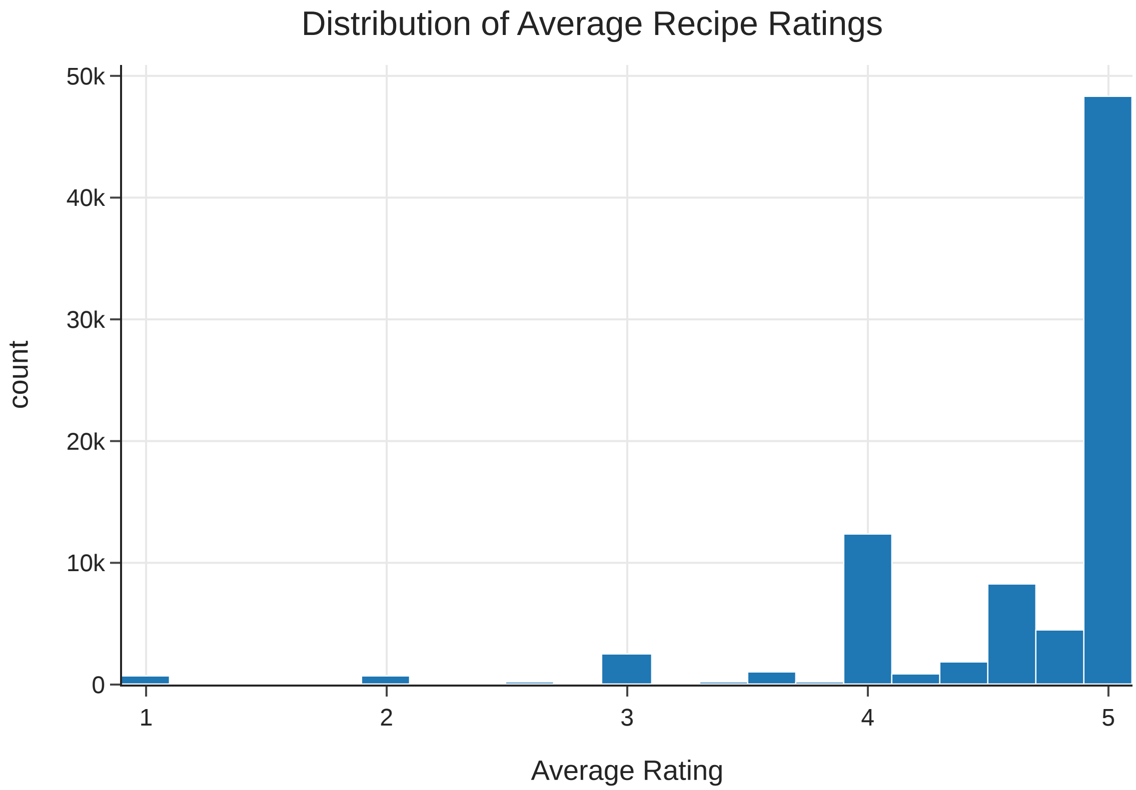
Out[9]:

	id	name	minutes	contributor_id	...	n_ingredients	avg_rating	c
0	333281	1 brownies in the world best ever	40	985201	...	9	4.0	
1	453467	1 in canada chocolate chip cookies	45	1848091	...	11	5.0	
2	306168	412 broccoli casserole	40	50969	...	9	5.0	
3	286009	millionaire pound cake	120	461724	...	7	5.0	
4	475785	2000 meatloaf	90	2202916	...	13	5.0	

5 rows × 15 columns

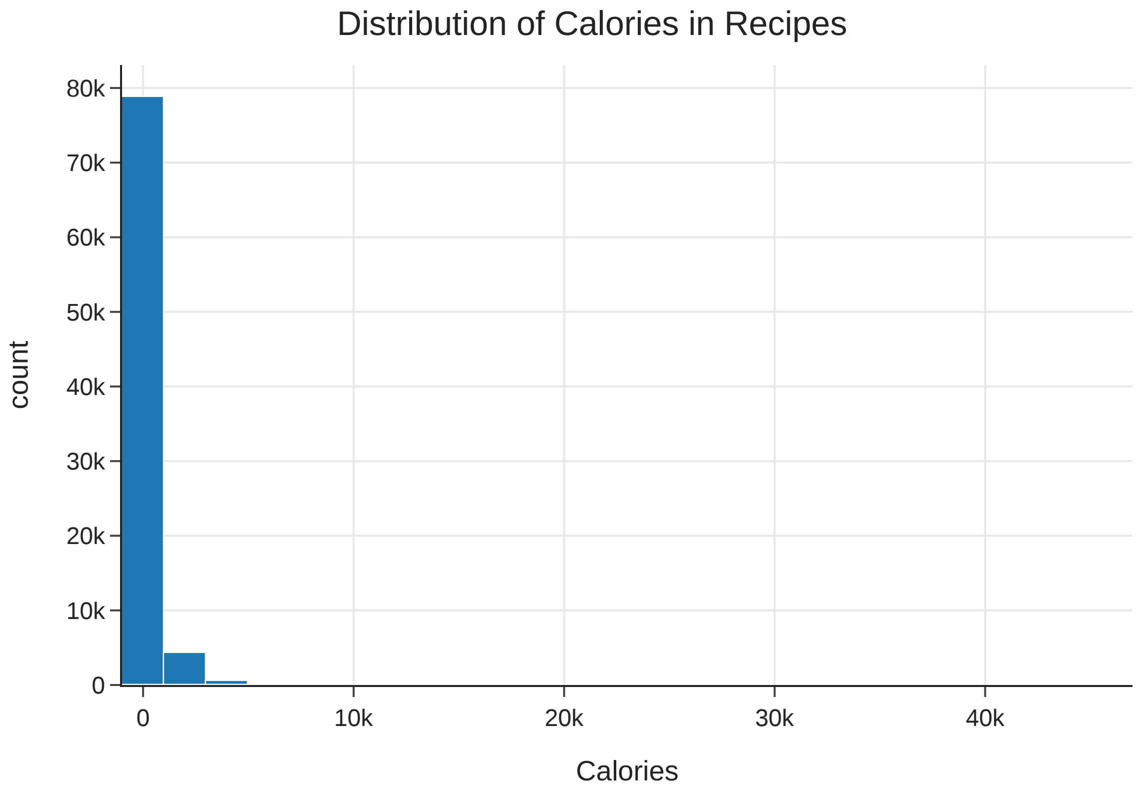
Univariate Analysis

In [10]:
```
fig1 = px.histogram(
    recipes,
    x='avg_rating',
    nbins=30,
    title='Distribution of Average Recipe Ratings',
    labels={'avg_rating': 'Average Rating'}
)
fig1.show()
```



The distribution of average ratings is extremely right-skewed, with most recipes receiving the maximum score of 5.0. Very few recipes fall below a rating of 4.0, suggesting that users on the site tend to give very high scores.

In [11]:
```
fig2 = px.histogram(
    recipes,
    x='calories',
    nbins=40,
    title='Distribution of Calories in Recipes',
    labels={'calories': 'Calories'}
)
fig2.show()
recipes['nutrition']
```

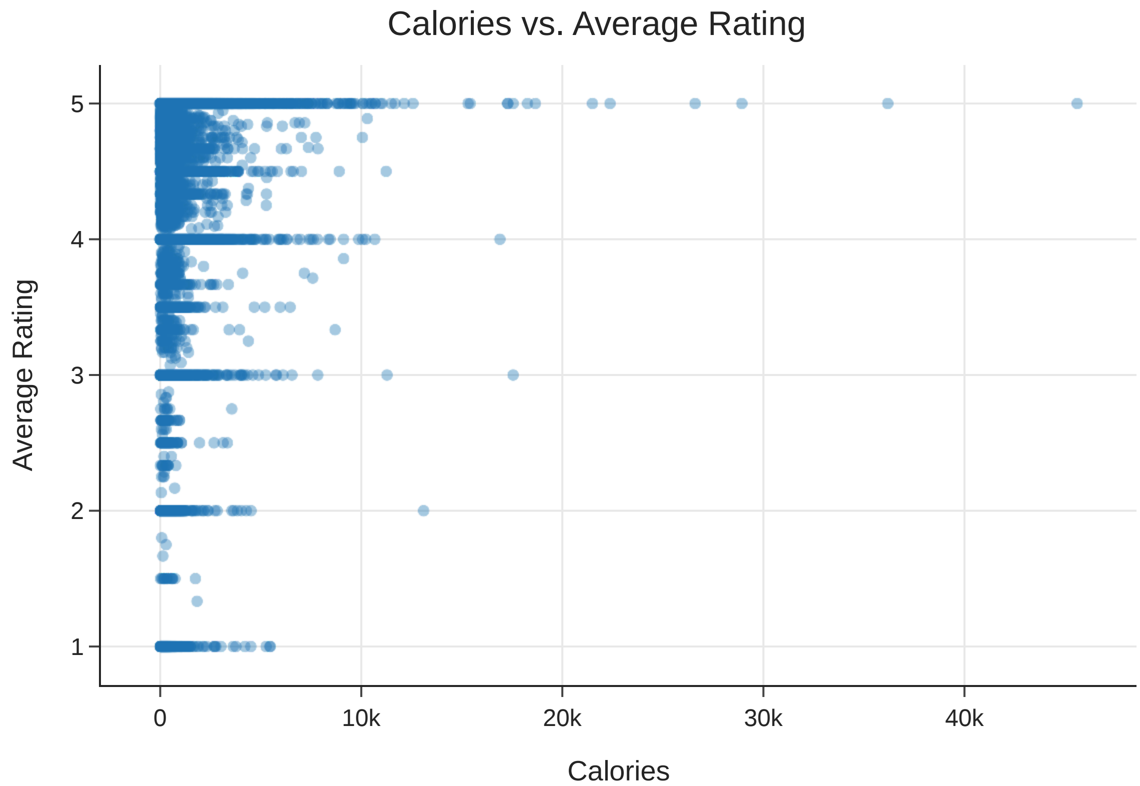


Out[11]:
```
0              [138.4, 10.0, 50.0, 3.0, 3.0, 19.0, 6.0]
1          [595.1, 46.0, 211.0, 22.0, 13.0, 51.0, 26.0]
2             [194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0]
                                ...
83779              [59.2, 6.0, 2.0, 3.0, 6.0, 5.0, 0.0]
83780           [188.0, 11.0, 57.0, 11.0, 7.0, 21.0, 9.0]
83781            [174.9, 14.0, 33.0, 4.0, 4.0, 11.0, 6.0]
Name: nutrition, Length: 83782, dtype: object
```

The distribution of calorie content is highly skewed, with most recipes containing fewer than 1,000 calories. There are significant outliers — some recipes exceed 30,000 calories — indicating the presence of large or indulgent dishes.

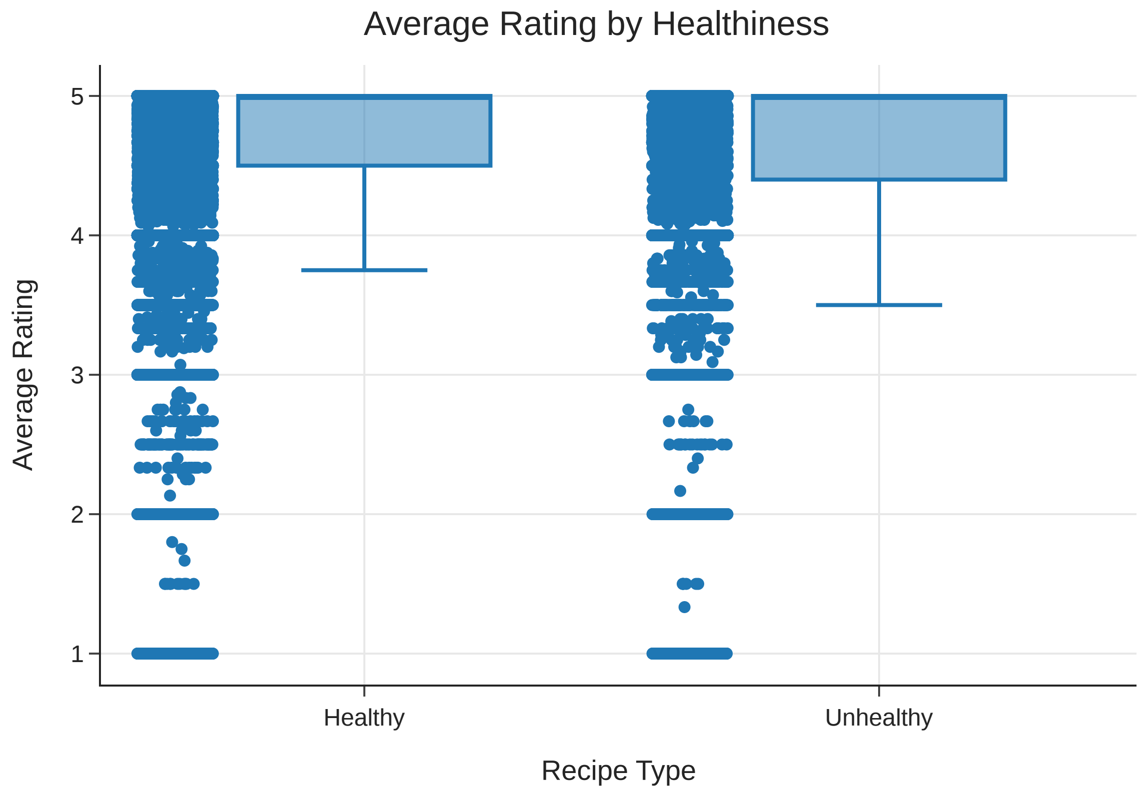
Bivariate analysis

In [12]:
```python
# Scatter Plot: Calories vs. Average Rating
fig3 = px.scatter(
    recipes,
    x='calories',
    y='avg_rating',
    opacity=0.4,
    title='Calories vs. Average Rating',
    labels={'calories': 'Calories', 'avg_rating': 'Average Rating'}
)
fig3.show()
```



This scatter plot shows no clear relationship between calories and average rating. Most recipes, regardless of calorie count, are rated highly.

In [13]:
```
recipes['label'] = recipes['is_unhealthy'].replace({True: 'Unhealthy', False
fig4 = px.box(
    recipes,
    x='label',
    y='avg_rating',
    points='all',
    title='Average Rating by Healthiness',
    labels={'label': 'Recipe Type', 'avg_rating': 'Average Rating'}
)
fig4.show()
```



This boxplot compares average ratings of healthy vs. unhealthy recipes. The distributions look very similar, with no strong indication that one group is rated better.

Interesting Aggregates

In [14]:
```
# Group by healthiness and compute aggregate stats on known columns
agg = recipes.groupby('is_unhealthy')[['avg_rating', 'calories', 'n_ingredie

# Flatten column MultiIndex
agg.columns = ['_'.join(col).strip() for col in agg.columns.values]
agg = agg.reset_index()
agg['is_unhealthy'] = agg['is_unhealthy'].map({True: 'Unhealthy', False: 'He

agg
```

Out[14]:

	is_unhealthy	avg_rating_mean	avg_rating_median	avg_rating_std	...	n_in
0	Healthy	4.63	5.0	0.64	...	
1	Unhealthy	4.62	5.0	0.64	...	

2 rows × 13 columns

From this summary table, we observe that both healthy and unhealthy recipes have nearly identical average ratings (4.63 vs. 4.62). However, unhealthy recipes tend to use more ingredients on average — about 10.43 compared to

8.81 for healthy ones. This may suggest that indulgent dishes often require more components or preparation, but users rate both recipe types similarly overall.

Step 3: Assessment of Missingness

Missingness Dependency

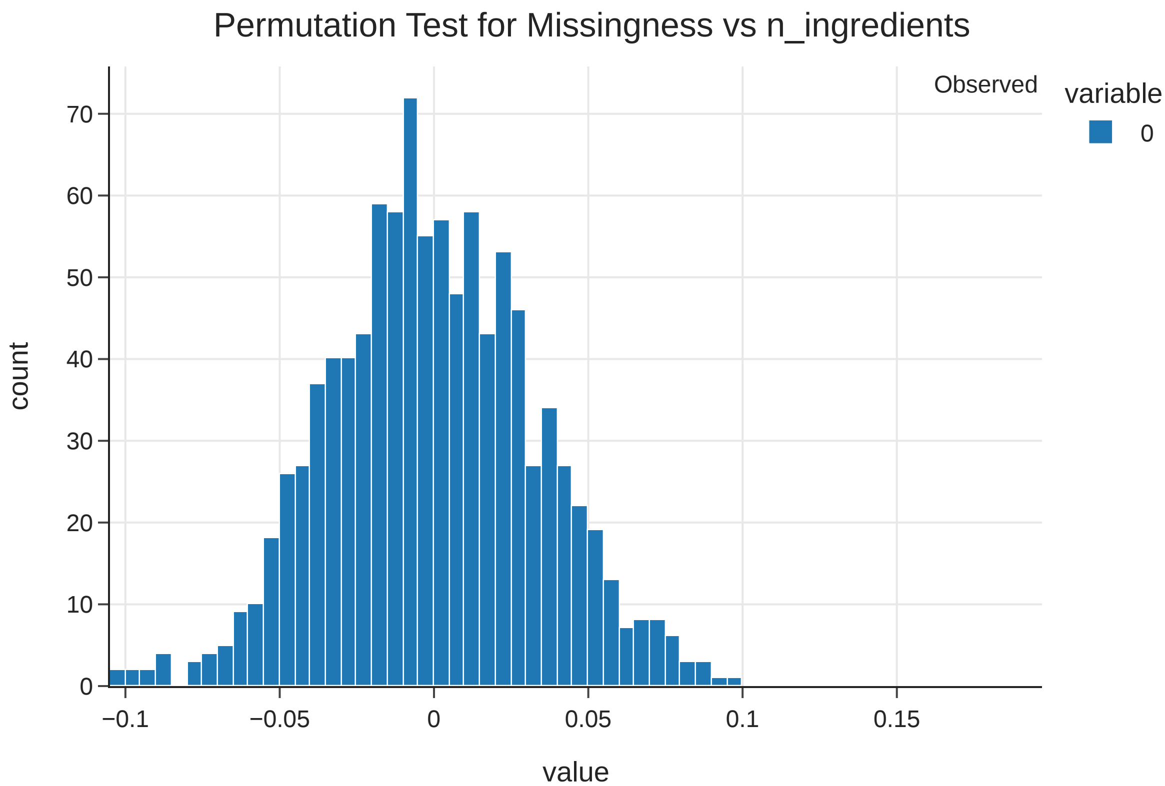
Identify missing columns

In [15]:
```
merged = recipes.merge(interactions, left_on='id', right_on='recipe_id', how
missing_summary = merged.isna().mean().sort_values(ascending=False)
missing_summary[missing_summary > 0]
```

Out[15]:
```
rating         6.41e-02
avg_rating     1.18e-02
description    4.86e-04
                 ...
recipe_id      4.27e-06
user_id        4.27e-06
name           4.27e-06
Length: 8, dtype: float64
```

In [16]:
```
def perm_test_stat(df, group_col, target_col, stat_func=np.mean, n_permutati
    """
    Returns:
        p-value, observed statistic, and plot
    """
    df = df.dropna(subset=[target_col])  # Drop rows where target is missing

    group1 = df[df[group_col]][target_col]
    group2 = df[~df[group_col]][target_col]
    observed = stat_func(group1) - stat_func(group2)

    diffs = []
    for _ in range(n_permutations):
        shuffled = df[group_col].sample(frac=1, replace=False).reset_index(d
        diff = stat_func(df[shuffled][target_col]) - stat_func(df[~shuffled]
        diffs.append(diff)

    diffs = np.array(diffs)
    if alternative == 'greater':
        p = np.mean(diffs >= observed)
    elif alternative == 'less':
        p = np.mean(diffs <= observed)
    else:
        p = np.mean(np.abs(diffs) >= np.abs(observed))

    fig = px.histogram(diffs, nbins=50, title=f'Permutation Test for Missing
    fig.add_vline(x=observed, line_color='red', line_dash='dash', annotatior

    return p, observed, fig
```

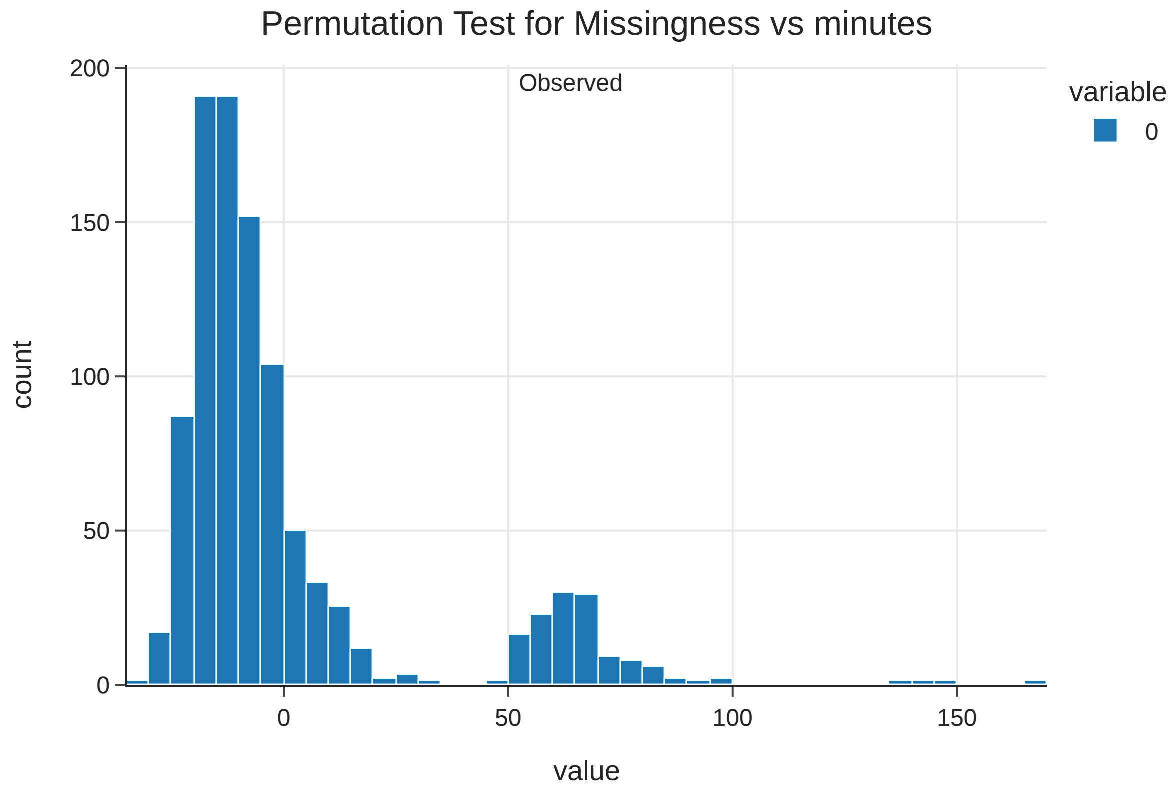
In [17]:
```
# Make missingness indicator
merged['missing_rating'] = merged['rating'].isna()

# Dependency test: Does missingness of rating depend on number of ingredient
p1, obs1, fig1 = perm_test_stat(
    merged,
    'missing_rating',
    'n_ingredients',
    n_permutations=1000
)
fig1.show()
print(f"Missingness depends on n_ingredients? p-value: {p1:.4f}")
```




```
Missingness depends on n_ingredients? p-value: 0.0000
```

Since the p-value is 0, it means missingness very strongly depends on number of ingredients

In [18]:
```
# Example: Does missingness of rating depend on contributor_id?
p3, obs3, fig3 = perm_test_stat(
    merged,
    'missing_rating',
    'minutes',
    n_permutations=1000
)
fig3.show()
print(f"Missingness depends on contributor_id? p-value: {p3:.4f}")
```




```
Missingness depends on contributor_id? p-value: 0.1280
```

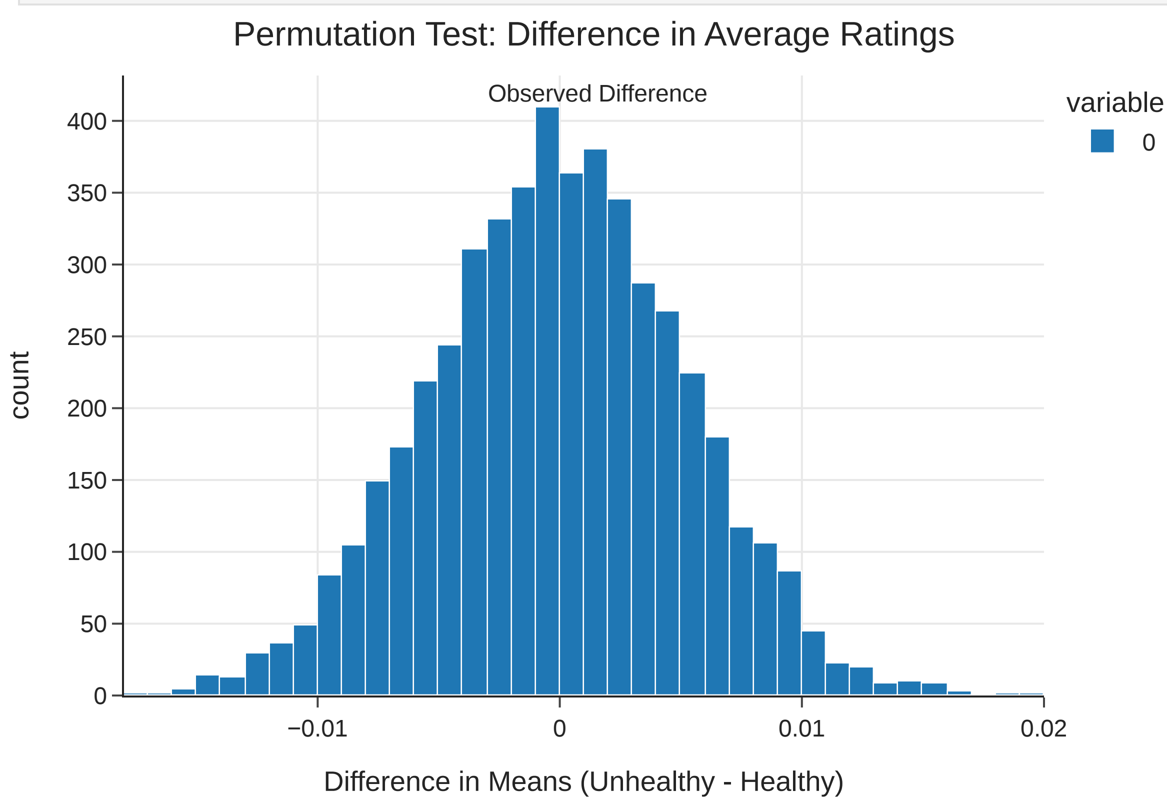
Since the p-value is greater than 0.05, we do not have enough evidence to support the claim that `missing_rating` and `minutes` depend on each other. So, they are probably **independent**.

Step 4: Hypothesis Testing

- **Null Hypothesis (H_0):** There is no difference in average ratings between healthy and unhealthy recipes.
- **Alternative Hypothesis (H_1):** Unhealthy recipes are rated higher than healthy ones.

In [19]:
```
# Drop rows where avg_rating is missing
recipes_clean = recipes.dropna(subset=['avg_rating'])

# Calculate observed difference in mean average rating
mean_unhealthy = recipes_clean.loc[recipes_clean['is_unhealthy'], 'avg_ratin
mean_healthy = recipes_clean.loc[~recipes_clean['is_unhealthy'], 'avg_rating
observed_diff = mean_unhealthy - mean_healthy

# Permutation test
n_permutations = 5000
diffs = []

for _ in range(n_permutations):
    shuffled = recipes_clean.copy()
    shuffled['is_unhealthy'] = np.random.permutation(shuffled['is_unhealthy'
```

```
    mean_unhealthy_perm = shuffled.loc[shuffled['is_unhealthy'], 'avg_rating
    mean_healthy_perm = shuffled.loc[~shuffled['is_unhealthy'], 'avg_rating'
    diffs.append(mean_unhealthy_perm - mean_healthy_perm)

# Convert to NumPy array for vectorized computation
diffs = np.array(diffs)

# One-sided p-value (we're testing if unhealthy recipes are rated higher)
p_value = np.mean(diffs >= observed_diff)

# Plot the distribution
fig = px.histogram(
    diffs,
    nbins=50,
    title='Permutation Test: Difference in Average Ratings',
    labels={'value': 'Difference in Means (Unhealthy - Healthy)'}
)
fig.add_vline(
    x=observed_diff,
    line_color='red',
    line_dash='dash',
    annotation_text='Observed Difference',
    annotation_position='top right'
)
fig.show()

print(f'Observed Difference: {observed_diff:.4f}')
print(f'p-value: {p_value:.4f}')
```




```
Observed Difference: -0.0031
p-value: 0.7234
```

Step 5: Framing a Prediction Problem

In this section, we define our prediction task.

Prediction Type: Regression

Prediction Goal:
We aim to predict the **average rating (`avg_rating`)** a recipe will receive based on its properties.

Response Variable: `avg_rating`
This variable represents the mean user rating for a given recipe, typically between 0 and 5 stars.

Rationale:
This prediction is useful for platforms like Food.com to estimate how well a newly submitted recipe might be received by users before it accumulates reviews.

Features to be used at time of prediction:
To ensure our model uses only information available at the time the recipe is posted, we will use the following features:

- `minutes` : total time to make the recipe
- `n_steps` : number of steps in the recipe
- `n_ingredients` : number of ingredients required

All of these are known before the recipe is rated by users, making them valid predictors.

Evaluation Metric: Root Mean Squared Error (RMSE)
RMSE is appropriate for regression tasks like this because it penalizes larger errors more heavily. It’s interpretable in the same units as our response variable (`avg_rating`), making it easy to assess prediction accuracy.

Step 6: Baseline Model

In [20]:
```
# Let's pick our numeric predictors for the baseline model. Fingers crossed!
X = recipes[['minutes', 'n_steps', 'n_ingredients']]
y = recipes['avg_rating']

# Drop rows where the target is missing. Because NaNs are the enemy of sciki
X = X[y.notna()]
y = y[y.notna()]

# Time to split the data into train and test sets. Please, random_state, be
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
# Scale those numeric features! Because raw minutes are scary.
preprocessor = make_column_transformer(
    (StandardScaler(), ['minutes', 'n_steps', 'n_ingredients']),
    remainder='passthrough'  # Just in case we missed something (we didn't,
)

# Build the pipeline. Linear regression, please save the day.
pipeline = make_pipeline(
    preprocessor,
    LinearRegression()
)

# Fit the model and predict. This is where the magic (or heartbreak) happens
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)

# Calculate RMSE. Lower is better REMEMBER THIS STEPHANIE!!!! LOWER IS BETTE
rmse = np.sqrt(root_mean_squared_error(y_test, y_pred))
print(f"Test RMSE: {rmse:.4f}")
```

```
Test RMSE: 0.7994
```

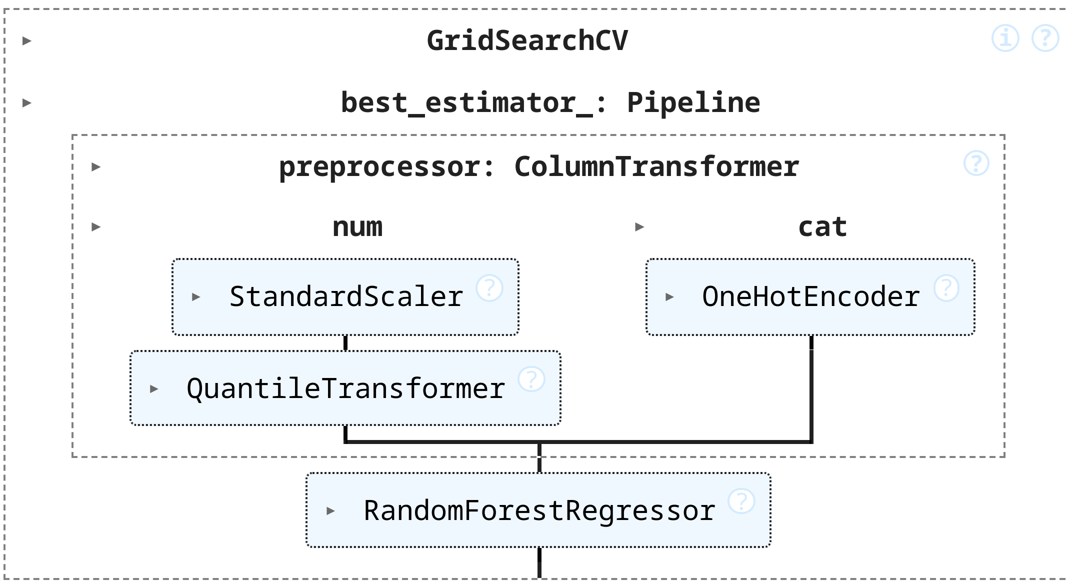
Step 7: Final Model

In [21]:

```
# Choose features for training
categorical_features = ['submitted_month']  # You can add more if needed
numerical_features = ['calories', 'n_ingredients']  # Base features
engineered_features = ['calories', 'n_ingredients']  # We'll apply transform

# Add month as a new categorical feature
X = recipes.copy()
X['submitted'] = pd.to_datetime(X['submitted'], errors='coerce')
X['submitted_month'] = X['submitted'].dt.month
y = X['avg_rating']

# Drop rows with missing target
df = X[['calories', 'n_ingredients', 'submitted_month', 'avg_rating']].dropr
X = df.drop(columns='avg_rating')
y = df['avg_rating']

# Train-test split (same as baseline)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Transformers
cat_transformer = OneHotEncoder(handle_unknown='ignore')
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('quantile', QuantileTransformer(output_distribution='normal'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, engineered_features),
    ('cat', cat_transformer, categorical_features)
])
```

```python
# Final pipeline with a Random Forest Regressor
final_model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(random_state=42))
])

# GridSearchCV for hyperparameter tuning
param_grid = {
    'regressor__n_estimators': [50, 100],
    'regressor__max_depth': [None, 5, 10]
}

grid_search = GridSearchCV(
    final_model_pipeline,
    param_grid,
    cv=5,
    scoring='neg_root_mean_squared_error'
)

# Fit grid_search yourself:
grid_search.fit(X_train, y_train)
```

Out[21]:



In [22]:
```python
best_model = grid_search.best_estimator_
best_model.score(X_test, y_test)
best_model.named_steps['regressor'].feature_importances_
```

Out[22]:
```
array([0.59, 0.19, 0.05, 0.02, 0.01, 0.01, 0.06, 0.01, 0.03, 0.  , 0.  ,
       0.01, 0.01, 0.01])
```

In [23]:
```python
y_pred = best_model.predict(X_test)
rmse = np.sqrt(root_mean_squared_error(y_test, y_pred))
print(f"Test RMSE: {rmse:.4f}")
```

```
Test RMSE: 0.7993
```

Step 8: Fairness Analysis

Hypotheses

- **Null Hypothesis (H_0):** The model is fair. Its RMSE is the same for short and long recipes; any observed difference is due to random chance.
- **Alternative Hypothesis (H_1):** The model is unfair. Its RMSE for short recipes is higher than that for long recipes.

We use **Root Mean Squared Error (RMSE)** as the evaluation metric, because our prediction task is a **regression problem** (predicting `avg_rating`). RMSE quantifies the average prediction error magnitude.

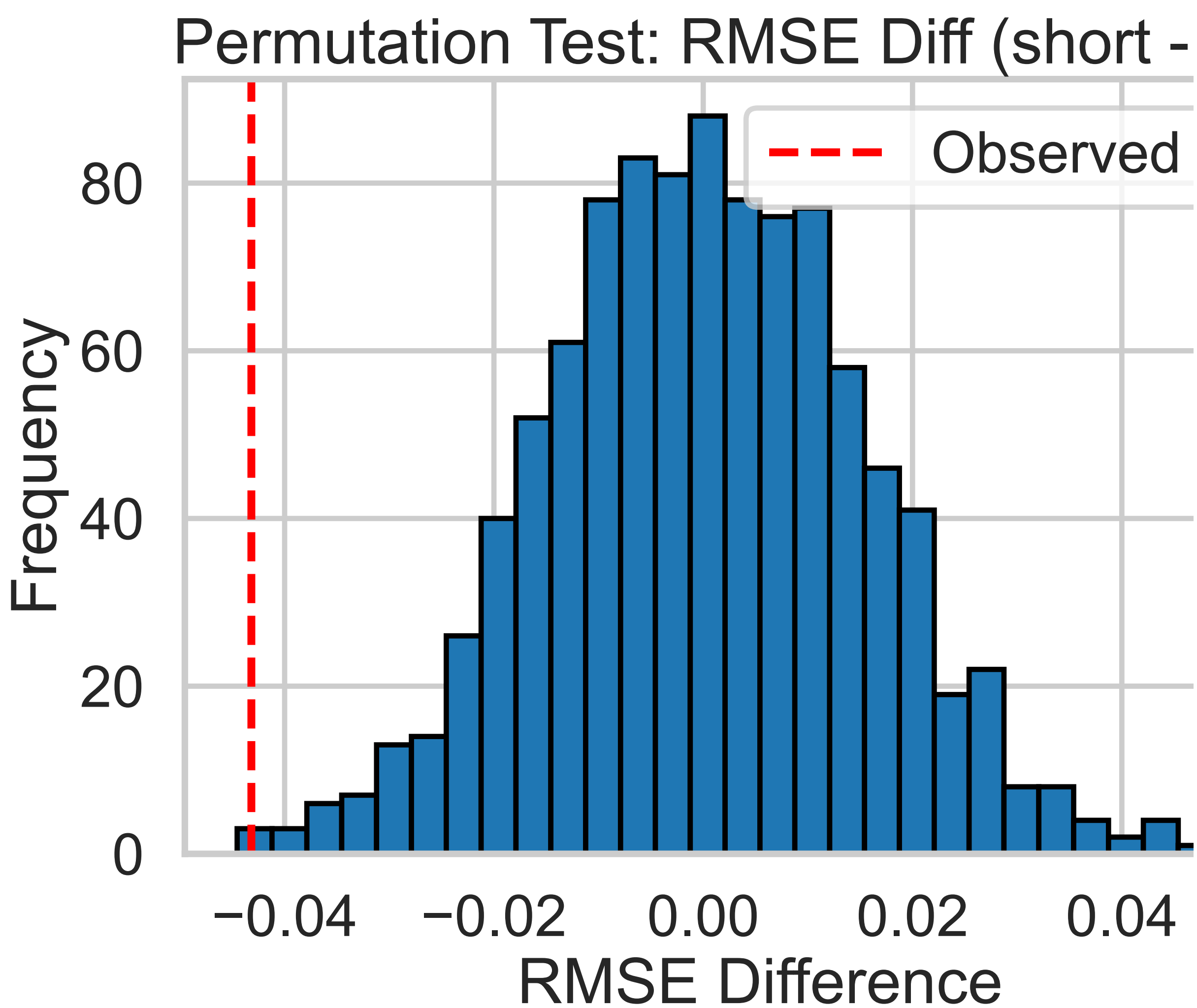
We binarize the `minutes` column to define two groups:

- **Group X (short recipes):** Recipes that take **less than 30 minutes**
- **Group Y (long recipes):** Recipes that take **30 minutes or more**

In [24]:
```python
from sklearn.metrics import root_mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Merge back with original 'minutes'
X_test_copy = X_test.copy()
X_test_copy['minutes'] = recipes.loc[X_test.index, 'minutes']
X_test_copy['group'] = (X_test_copy['minutes'] >= 30).astype(int)

# Predictions
y_pred = best_model.predict(X_test)

# RMSE by group
rmse_short = root_mean_squared_error(
    y_test[X_test_copy['group'] == 0],
    y_pred[X_test_copy['group'] == 0]
)
rmse_long = root_mean_squared_error(
    y_test[X_test_copy['group'] == 1],
    y_pred[X_test_copy['group'] == 1]
)

observed_diff = rmse_short - rmse_long
print(f"Observed RMSE difference (short - long): {observed_diff:.4f}")

# Permutation test
n_permutations = 1000
permuted_diffs = []

for _ in range(n_permutations):
    shuffled = np.random.permutation(X_test_copy['group'])
    rmse_short_perm = root_mean_squared_error(
        y_test[shuffled == 0],
        y_pred[shuffled == 0]
    )
    rmse_long_perm = root_mean_squared_error(
```

```
        y_test[shuffled == 1],
        y_pred[shuffled == 1]
    )
    permuted_diffs.append(rmse_short_perm - rmse_long_perm)

# Compute p-value
permuted_diffs = np.array(permuted_diffs)
p_val = np.mean(permuted_diffs >= observed_diff)
print(f"Permutation test p-value: {p_val:.4f}")

# Plot with matplotlib
plt.figure(figsize=(8, 6))
plt.hist(permuted_diffs, bins=30, edgecolor='black')
plt.axvline(observed_diff, color='red', linestyle='--', label='Observed Diff
plt.title('Permutation Test: RMSE Diff (short - long)')
plt.xlabel('RMSE Difference')
plt.ylabel('Frequency')
plt.legend()
plt.tight_layout()
plt.show()
```

```
Observed RMSE difference (short - long): -0.0432
Permutation test p-value: 0.9990
```



Results

We performed a permutation test by shuffling the group labels and computing the RMSE difference (short - long) 1000 times.

- **Observed RMSE Difference:** -0.0432
- **p-value:** 0.9990

The observed difference is small and in the opposite direction, and the p-value is high, so we **fail to reject the null hypothesis**.
Our model **does not appear to be unfair** with respect to recipe prep time.