Classifying Recipe Popularity and Rating with Imbalanced Data

Name(s): Weijie Zhang

Website Link: https://wzhang3912.github.io/Classifying-Recipe-Popularity-and-Rating/

In [1]:
```
# for data wrangling
import pandas as pd
import numpy as np
from pathlib import Path

# for visualization
import plotly.express as px
pd.options.plotting.backend = 'plotly'
import plotly.graph_objects as go
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

# for permutation testing
from scipy.stats import ks_2samp

# for classification model
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier

# for feature engineering
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import RobustScaler, FunctionTransformer
from sklearn.preprocessing import OneHotEncoder, KBinsDiscretizer
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

# for model evaluation
from sklearn.metrics import precision_recall_fscore_support, classification_report
from sklearn.metrics import accuracy_score, precision_score
from sklearn.metrics import confusion_matrix

import warnings
warnings.filterwarnings("ignore")

import plotly.offline as pyo
pyo.init_notebook_mode(connected=False)
```

1. Introduction

In [2]:
```
# load the recipe dataset
recipe = pd.read_csv('data/RAW_recipes.csv')
recipe.head()
```

Out[2]:

	name	id	minutes	contributor_id	submitted	tags	nutrition	n_steps	steps
0	1 brownies in the world best ever	333281	40	985201	2008-10-27	['60-minutes-or-less', 'time-to-make', 'course...	[138.4, 10.0, 50.0, 3.0, 3.0, 19.0, 6.0]	10	['heat the oven to 350f and arrange the rack i...
1	1 in canada chocolate chip cookies	453467	45	1848091	2011-04-11	['60-minutes-or-less', 'time-to-make', 'cuisin...	[595.1, 46.0, 211.0, 22.0, 13.0, 51.0, 26.0]	12	['pre-heat oven the 350 degrees f', 'in a mixi...
2	412 broccoli casserole	306168	40	50969	2008-05-30	['60-minutes-or-less', 'time-to-make', 'course...	[194.8, 20.0, 6.0, 32.0, 22.0, 36.0, 3.0]	6	['preheat oven to 350 degrees', 'spray a 2 qua...
3	millionaire pound cake	286009	120	461724	2008-02-12	['time-to-make', 'course', 'cuisine', 'prepara...	[878.3, 63.0, 326.0, 13.0, 20.0, 123.0, 39.0]	7	['freheat the oven to 300 degrees', 'grease a ...
4	2000 meatloaf	475785	90	2202916	2012-03-06	['time-to-make', 'course', 'main-ingredient', ...	[267.0, 30.0, 12.0, 12.0, 29.0, 48.0, 2.0]	17	['pan fry bacon , and set aside on a paper tow...

In [3]:
```
# load the review dataset
review = pd.read_csv('data/RAW_interactions.csv')
review.head()
```

Out[3]:

	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...
3	124416	120345	2011-08-06	0	Just an observation, so I will not rate. I fo...
4	2000192946	120345	2015-05-10	2	This recipe was OVERLY too sweet. I would sta...

Research Question:

What are some of the characteristics of the recipe that are both popular and highly rated?

What category does a recipe fall into based on its predicted popularity level and average rating score?

Category 1: Low review count, low average rating

Category 2: Low review count, high average rating

Category 3: High review count, low average rating

Category 4: High review count, high average rating

We define the threshold for high and low review count as **10** reviews count and the threshold for high and low average rating score as **4.8**

2. Data Cleaning and Exploratory Data Analysis

2.1 Data Cleaning

In [4]:
```
'''
Convert string like objects to list or datetime
'''
def convert_dtype(recipe):
    df = recipe.copy()
    df['tags'] = df['tags'].str.replace('[\[\]\']', '', regex=True).str.split(', ')
    df['steps'] = df['steps'].str.replace('[\[\]\']', '', regex=True).str.split(', ')
    df['ingredients'] = (df['ingredients']
                         .str.replace('[\[\]\']', '', regex=True)
                         .str.split(', '))
    df['submitted'] = pd.to_datetime(df['submitted'])
    return df
```

In [5]:
```
'''
Expand values in nutrition columns into their indiviual columns
'''
def expand_nutrition(recipe):
    nutriton_df = (recipe['nutrition']
                   .str.replace('[\[\]\'\s]', '', regex=True)
                   .str.split(',', expand=True))

    col_name = ['calories',
                'total_fat',
                'sugar',
                'sodium',
                'protein',
                'saturated_fat',
                'carbohydrates']

    nutriton_df.columns = col_name

    for col in col_name:
        nutriton_df[col] = pd.to_numeric(nutriton_df[col], errors='coerce')

    return pd.concat([recipe, nutriton_df], axis=1).drop(columns='nutrition')
```

In [6]:
```
'''
Add average rating, the number of rating, and reviews comment in recipe dataset
'''
def add_rating_review(recipe, review):
```

```
    recipe_review = recipe.merge(review, how='left',
                                 left_on='id', right_on='recipe_id')

    recipe_review['rating'] = recipe_review['rating'].replace(0, np.NaN)

    rating = recipe_review.groupby('recipe_id')['rating'].mean()
    rating.name = 'rating'

    n_review = recipe_review.groupby('recipe_id')['user_id'].count()
    n_review.name = 'n_review'

    review_lst = (recipe_review.groupby('recipe_id')['review']
                  .apply(lambda df: df.values))
    review_lst.name = 'reviews'

    return (recipe
            .merge(rating, how='left', left_on='id', right_index=True)
            .merge(n_review, how='left', left_on='id', right_index=True)
            .merge(review_lst, how='left', left_on='id', right_index=True)
           )
```

In [7]:
```
'''
Add class label for classification task
'''
def add_class(recipe):
    '''
    helper function for creating class label
    '''
    def create_label(df):
        if df['n_review'] >= 10:
            if df['rating'] >= 4.8:
                return 4 # high rating, high count
            else:
                return 3 # low rating, high count
        else:
            if df['rating'] >= 4.8:
                return 2 # high rating, low count
            else:
                return 1 # low rating, low count

    # avoid if we run the same cell multiple times
    if 'class' not in recipe.columns:
        class_ser = recipe[['n_review', 'rating']].apply(create_label, axis=1)
        class_ser.name = 'class'

        return pd.concat([recipe, class_ser], axis=1)

    else:
        return recipe
```

In [8]:
```
'''
Only include n% of the data for quantative columns, excluding extreme values

recipe: the dataframe to perform outlier exclusion
outlier_col: the columnt to exclude outlier
n: the percentage of quantative data to keep
'''
def remove_outlier(recipe, outlier_col, n):

    df = recipe.copy()
    n = n/100
    upper = n+(1-n)/2
    lower = 1-upper
```

```
    for col in outlier_col:

        upper_lim = df[col].quantile(upper)
        lower_lim = df[col].quantile(lower)

        df = df[df[col].between(lower_lim, upper_lim)]

    return df
```

In [9]:
```
# data cleaning in action
recipe = (recipe
          .pipe(convert_dtype)
          .pipe(expand_nutrition)
          .pipe(add_rating_review, review)
         )
```

In [10]:
```
recipe.head()
```

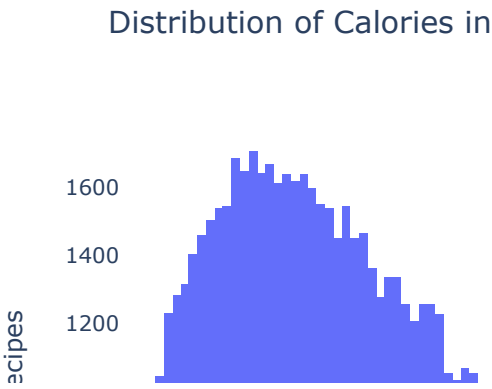
Out[10]:

	name	id	minutes	contributor_id	submitted	tags	n_steps	steps	descripti
0	1 brownies in the world best ever	333281	40	985201	2008-10-27	[60-minutes-or-less, time-to-make, course, mai...	10	[heat the oven to 350f and arrange the rack in...	these a the mo chocolat moist, ric c
1	1 in canada chocolate chip cookies	453467	45	1848091	2011-04-11	[60-minutes-or-less, time-to-make, cuisine, pr...	12	[pre-heat oven the 350 degrees f, in a mixing ...	this is t recipe th we use my scho ca
2	412 broccoli casserole	306168	40	50969	2008-05-30	[60-minutes-or-less, time-to-make, course, mai...	6	[preheat oven to 350 degrees, spray a 2 quart ...	since the are alrea 411 recip t brocc
3	millionaire pound cake	286009	120	461724	2008-02-12	[time-to-make, course, cuisine, preparation, o...	7	[freheat the oven to 300 degrees, grease a 10-...	why millionа pou cak because i su
4	2000 meatloaf	475785	90	2202916	2012-03-06	[time-to-make, course, main-ingredient, prepar...	17	[pan fry bacon , and set aside on a paper towe...	ready, s coc spec editi conte ent

5 rows × 21 columns

2.2 Univariate Analysis

In [11]:
```
import plotly.offline as pyo
pyo.init_notebook_mode(connected=False)
# remove extreme values for better visualization of the distribution
df_proc = remove_outlier(recipe, ['calories'], 98)['calories']

fig = px.histogram(df_proc,
                   x='calories',
                   title='Distribution of Calories in the Food Recipes',
                   nbins=300
                  ).update_layout(yaxis_title='Number of Recipes',
                                  xaxis_title='Calories')
fig.update_layout(
    plot_bgcolor='white'
)

fig.show()
```



A histogram of the distribution of the calories of food recipes in my dataset.

I only includes 98% of the data value for the `calories` column for better visualization. From the histogram, we see that the most of the values falls between 0 to 1000, with more values to the right of the plot. The plot is right-skewed, showing that most of the food receips have calories within a normal range, while there are some food recipes that have higher calories.

In [12]:
```
fig = px.histogram(recipe,
                   x='rating',
```

```
            title='Distribution of Rating of the Food Recipes',
            nbins=20
                 ).update_layout(yaxis_title='Number of Recipes',
                                 xaxis_title='Rating Range')
fig.show()
```

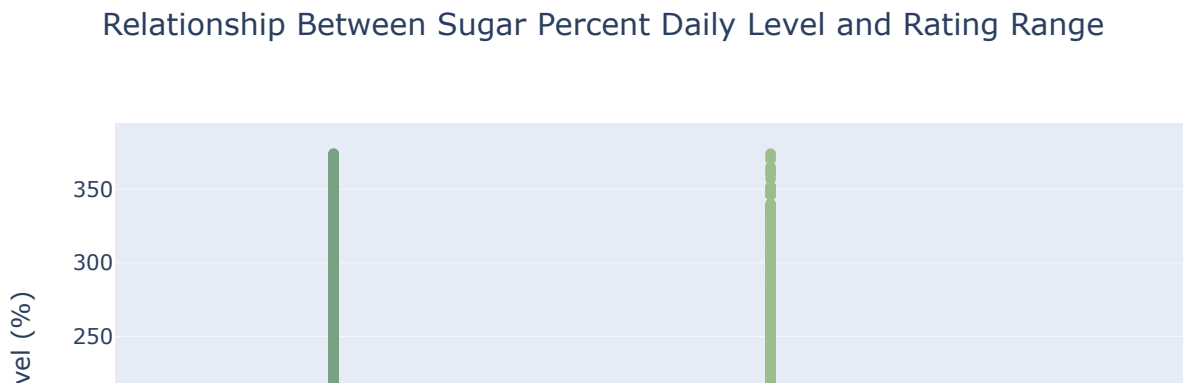
A histogram of the distribution of the rating of food recipes in my dataset.

From the histogram, we see that most of the recipe have a average rating above 3.0, with a very minority of the recipe has average rating below 3.0. If we want to do classification that discrminates recipes based on good rating and bad rating, we might encounter the issue of **imbalanced data**. This is somthing we should consider when working on our classification model.

2.3 Bivariate Analysis

In [13]:
```
# process dataframe for better visualization by removing outliers and assign ratings to
df_proc = remove_outlier(recipe,  ['sugar'], 95)[['rating', 'sugar']]
df_proc['rating_bin'] = pd.cut(df_proc['rating'],
                               [0.0, 1.0, 2.0, 3.0, 4.0, 5.0]).astype('string')


fig = px.box(df_proc,
             y='sugar',
             x='rating_bin',
             color='rating_bin',
             title='Relationship Between Sugar Percent Daily Level and Rating Range',
             category_orders={
                 'rating_bin': ['(4.0, 5.0]', '(3.0, 4.0]',
                                '(2.0, 3.0]', '(1.0, 2.0]',
```

```
                                '(0.0, 1.0]']
            },
            labels={
                'sugar': 'Sugar Daily Level (%)',
                'rating_bin': 'Rating Range'
            },
            color_discrete_map={
                '(4.0, 5.0]': '#7AA183',
                '(3.0, 4.0]': '#9DBE8B',
                '(2.0, 3.0]': '#EAB97B',
                '(1.0, 2.0]': '#D39470',
                '(0.0, 1.0]': '#CF775E'
            })


fig.show()
```



A box plot of the relationship between rating and sugar percent daily value.

I only include 95% of the data value for the `sugar` column to exclude extreme values and categorize rating into groups for better visualization. From the box plot, if we look at the third quarter and the upper fence, we can see that there is a slight increasing trend in the daily level of sugar in recipes as the recipe average rating decreases. This suggests that there might be a **negative correlation between the sugar level and the average rating** in recipes, indicating that high sugar level in the recipe might be a characteristics of reciving bad rating.

In [15]:
```
# process dataframe for better visualization by
# removing outliers, assigning ratings to bins, adding noise to the n_ingredients columr
df_proc = remove_outlier(recipe, ['calories'], 99)[['n_ingredients', 'calories']]
```

```
df_proc['n_ingredients'] = (
    df_proc['n_ingredients']
    .apply(lambda x: x+np.random.normal(0, 0.1))
)

fig = px.scatter(
    df_proc,
    x='n_ingredients',
    y='calories',
    title='Relationship Between Number of Ingredients and Calories Daily Level (%)',
    labels={
      'calories': 'Calories Daily Level (%)',
      'n_ingredients': 'Number of Ingredients'
    },
    opacity=0.9,
    trendline='ols',
    trendline_color_override='red',
    height=600,
    width=800
)

fig.show()
```

A scratter plot of the relationship between calories percent daily value and number of ingredients.

Before making the visualization, I only includes 99% of data value for `calories` column to exclude extreme vlaues, categorize rating into groups, and manualy add random noise to `n_ingredients` column which is discrete variable, for better visualization. From looking at the scatter plot, we can see that there is a **week positive correlation between the daily level of calories and number of ingredients**. It's to be expected that using more ingredients in a recipe would likely result in higher calorie content in the food.

2.4 Interesting Aggregates

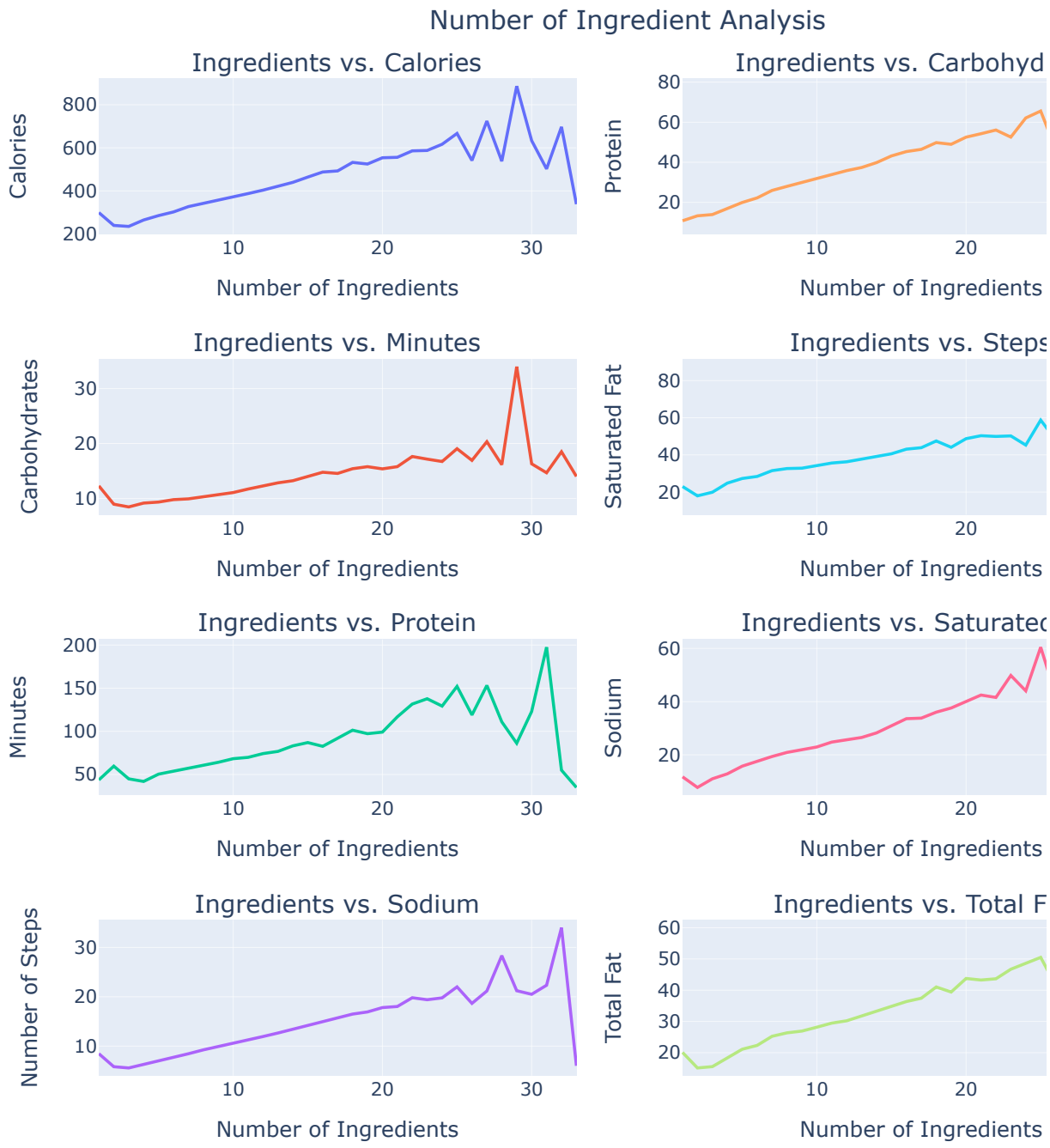
In [16]:
```
df_agg=recipe.copy()

# categorizes rating into 5 bins
quan_col = ['n_steps', 'minutes', 'calories', 'total_fat', 'sugar',
            'sodium', 'protein', 'saturated_fat','carbohydrates']

df_agg = remove_outlier(df_agg, quan_col[1:], 99)

df_agg.pivot_table(index='n_ingredients', values=quan_col, aggfunc='mean')
```

Out[16]:

n_ingredients	calories	carbohydrates	minutes	n_steps	protein	saturated_fat	so
1	299.172727	12.272727	43.727273	8.454545	10.818182	23.000000	11.72
2	239.645757	8.940100	59.710483	5.816972	13.286190	17.968386	7.73
3	235.040124	8.442646	44.954307	5.564017	13.915754	19.918610	10.98
4	264.826676	9.143439	41.946360	6.238985	17.064895	24.807232	12.83
5	285.283628	9.328884	50.417498	7.123512	19.947089	27.309424	15.73
6	302.164466	9.771556	53.587268	7.743418	22.229558	28.406881	17.66
7	326.971044	9.912910	57.201862	8.442063	25.963866	31.528172	19.36
8	341.125946	10.306710	59.943232	9.229858	27.985257	32.658579	20.90
9	354.512385	10.813796	64.115412	9.864930	30.076942	32.855282	21.97
10	368.668506	11.060865	68.302512	10.724553	32.065527	34.273245	22.94
11	387.523041	11.702582	69.841516	11.258320	34.142964	35.637218	24.78
12	403.407440	12.370748	74.213753	11.933782	35.845188	36.301801	25.75
13	421.273942	12.830983	76.750289	12.648555	37.400462	37.613179	26.51
14	439.602060	13.220148	83.167042	13.370776	39.842292	39.501448	28.24
15	464.068472	13.906550	86.988646	14.288210	43.180349	40.592576	30.76
16	487.721195	14.762169	82.646334	14.966112	45.347505	43.094886	33.59
17	492.720202	14.545455	92.386593	15.667585	46.461892	43.886134	33.79
18	532.243419	15.405699	101.328358	16.485753	49.824966	47.504749	36.03
19	524.739917	15.775468	97.291060	16.920998	48.983368	44.106029	37.60
20	553.788604	15.378917	99.105413	17.803419	52.552707	48.743590	40.17
21	555.859709	15.781553	116.800971	18.024272	54.262136	50.334951	42.47
22	585.853676	17.632353	131.595588	19.801471	56.102941	49.948529	41.57
23	587.696774	17.150538	137.752688	19.397849	52.580645	50.225806	49.83
24	616.487324	16.718310	129.253521	19.774648	62.126761	45.295775	44.05
25	666.739474	19.052632	152.052632	22.000000	65.578947	58.736842	60.47
26	541.234615	16.923077	118.961538	18.653846	46.615385	48.115385	42.50
27	724.388889	20.333333	153.166667	21.166667	67.666667	75.555556	44.
28	537.786667	16.133333	111.000000	28.333333	61.466667	38.466667	52.73
29	886.300000	34.000000	86.222222	21.222222	78.000000	72.333333	59.44
30	633.680000	16.300000	123.000000	20.500000	56.900000	64.500000	51.90
31	502.050000	14.666667	197.500000	22.333333	47.666667	45.000000	46.50
32	697.350000	18.500000	55.000000	34.000000	66.000000	87.500000	53.00
33	338.200000	14.000000	35.000000	6.000000	8.000000	12.000000	16.00

If we groupby number of ingredients in the dataset and calculate the respective mean, we can see from the pivot table as we look from top to down for each column, there is a increasing trend in other quantative columns, such as `calories`, `carbohydrates`, `minutes`, `n_steps`, `protein`, `saturated_fat`, `sodium`, and `total_fat`, as number of ingredients increase.

In [17]:
```
# columns to make line plot
viz_cols = ['calories', 'carbohydrates', 'minutes', 'n_steps', 'protein',
            'saturated_fat', 'sodium', 'total_fat']
name_cols = ['Calories', 'Carbohydrates', 'Minutes', 'Number of Steps',
             'Protein', 'Saturated Fat', 'Sodium', 'Total Fat']

# preprocss grouped dataframe for visualization
df_viz = df_agg.groupby('n_ingredients')[quan_col].mean().reset_index()

# make 8 line plots
fig = make_subplots(
    rows=4, cols=2,
    subplot_titles=("Ingredients vs. Calories",
                    "Ingredients vs. Carbohydrates",
                    "Ingredients vs. Minutes",
                    "Ingredients vs. Steps",
                    "Ingredients vs. Protein",
                    "Ingredients vs. Saturated Fat",
                    "Ingredients vs. Sodium",
                    "Ingredients vs. Total Fat"),
)

row, col= 1,1

for viz_col, name_col in zip(viz_cols, name_cols):
    fig.append_trace(go.Scatter(
        x=df_viz['n_ingredients'],
        y=df_viz[viz_col],
        name=name_col
    ), row=row, col=col)

    # rename x-axis and y-axis label
    fig.update_xaxes(title_text="Number of Ingredients", row=row, col=col)
    fig.update_yaxes(title_text=f"{name_col}", row=row, col=col)

    if row == 4:
        col+=1
        row=1
    else:
        row+=1

fig.update_layout(height=800, width=950,
                  title=dict(text="Number of Ingredient Analysis", x=0.45))

fig.update_layout(margin=dict(t=65))

fig.show()
```

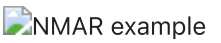


If we plot the line plot for the relationship between number of ingredients and other quantative columns such as `calories`, `carbohydrates`, `minutes`, `n_steps`, `protein`, `saturated_fat`, `sodium`, and `total_fat`, respectively, we can see there are indeed obvious increasing trends for each of the plottings, suggesting that on average, `n_ingredients` have positive correlation relationship with these quantative columns as above.

3. Assessment of Missingness

3.1 NMAR Analysis

We believe that the missing values in the `rating` column is Not Missing at Random (**NMAR**). This means that the chance that a value missing **depends on the actual missing values.** After browsering from the food.com website, we found that, in the comment section of a recipe, there are some reviews without any ratings. This is the case in which some users might only post a review comment, without posting a rating score. Therefore, during the data generation process, this will cause some recipes to have missing rating score value.



We decide to **drop the rows with missing values in the rating column** so that it will not influence our target class label variable, and thus, influence our classification problem.

In [18]:
```
recipe = recipe.dropna(subset=['rating'])
```

3.2 Missingness Dependency Analysis

In [19]:
```
recipe.isna().sum()
```

Out[19]:
```
name               0
id                 0
minutes            0
contributor_id     0
submitted          0
tags               0
n_steps            0
steps              0
description       66
ingredients        0
n_ingredients      0
calories           0
total_fat          0
sugar              0
sodium             0
protein            0
saturated_fat      0
carbohydrates      0
rating             0
n_review           0
reviews            0
dtype: int64
```

As we can see, there are 70 missing values in the `description`. We believe that the missingness of `description` is **Missing at Random (MAR)**. We will investigate the missing dependency of `description` using permutation testing as below.

In [20]:
```
'''
Helper Function

Make histogram plot based on given simulated statistics and observed statistic
'''
def make_permutation_plot(sim_stats, obs_stat, col, col_label):

    df = pd.DataFrame(sim_stats)
    df = df.rename(columns={0:f'shuffled_{col}'})

    fig = px.histogram(df, x=f'shuffled_{col}', nbins=50, histnorm='probability')

    fig.add_vline(x=obs_stat, line_color='red', line_width=1, opacity=1)
```

```
    fig.add_annotation(
        text=f'''<span style="color:red">Observed K-S =
{round(obs_stat, 2)}</span>''',
        x=obs_stat-0.017, showarrow=False, y=0.1
    )

    alpha_level = np.quantile(sim_stats, 0.95)

    fig.add_vline(x=alpha_level, line_color='blue', line_width=1, opacity=1)
    fig.add_annotation(
        text=f'''<span style="color:blue">Significance Level of 0.05 = {
round(alpha_level, 2)}</span>''',
        x=alpha_level-0.017, showarrow=False, y=0.12
    )

    fig.update_layout(xaxis_range=[0, 0.2])
    fig.update_layout(yaxis_range=[0, 0.2])

    fig.update_layout(
        title=dict(text=f'''Distribution of Permutation Test for {col}
by Missingness of {col_label}''',
                   font=dict(size=20))
    )

    fig.show()

'''
Helper Function

Perform permutation testing using K-S statistics
'''
def permutation_ks(recipe, n_rep, col, col_label):
    df = recipe.copy()

    simulated_stats = []

    # Computing and storing the K_S observed statistics.
    obs_stat = ks_2samp(recipe_mar.loc[recipe[col_label] == True, col],
                        recipe_mar.loc[recipe[col_label] == False, col]).statistic

    for _ in range(n_rep):

        # Shuffling the values, while keeping the group labels in place
        df[f'shuffled_{col_label}'] = np.random.permutation(df[col_label])

        # Computing and storing the K-S simulated statistic
        groups = df.groupby(f'shuffled_{col_label}')[col]
        sim_stat = ks_2samp(df.loc[df[f'shuffled_{col_label}'] == True, col],
                            df.loc[df[f'shuffled_{col_label}'] == False, col]
                           ).statistic

        simulated_stats.append(sim_stat)

    make_permutation_plot(simulated_stats, obs_stat, col, col_label)

    return np.mean(np.array(simulated_stats) >= obs_stat)

'''
Helper Function

Create kde distribution plot
'''
def create_kde_plotly(df, group_col, group1, group2, vals_col, x_range=None, title=''):
```

```
    fig = ff.create_distplot(
        hist_data=[df.loc[df[group_col] == group1, vals_col],
                   df.loc[df[group_col] == group2, vals_col]],
        group_labels=[group1, group2],
        show_rug=False, show_hist=False
    )
    if x_range:
        fig.update_layout(xaxis_range=x_range)
    return fig.update_layout(title=title, xaxis_range=x_range)
```

Does the missingness of `description` depend on the `sugar` column?

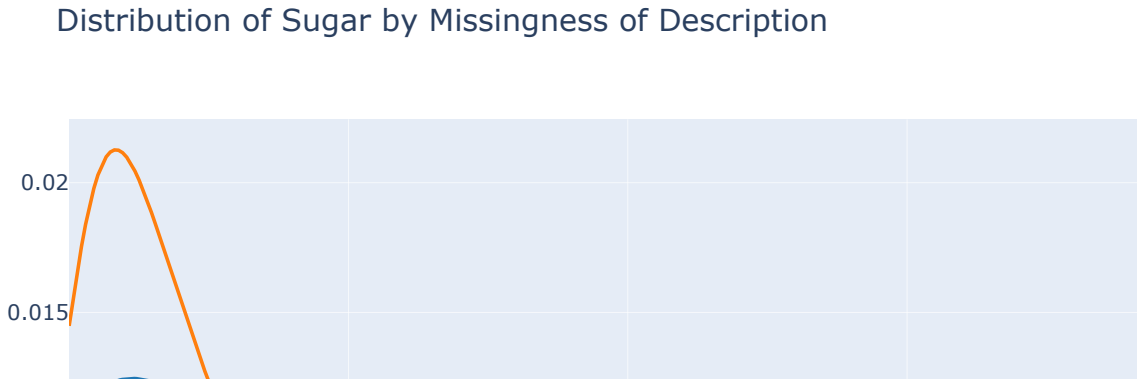
Null Hypothesis: the missingness of `description` does not depend on `sugar`.

Alternative Hypothesis: the missingness of `description` does depend on `sugar`.

Let's look at the distribution of `sugar`, conditional on the missingness of `description` column.

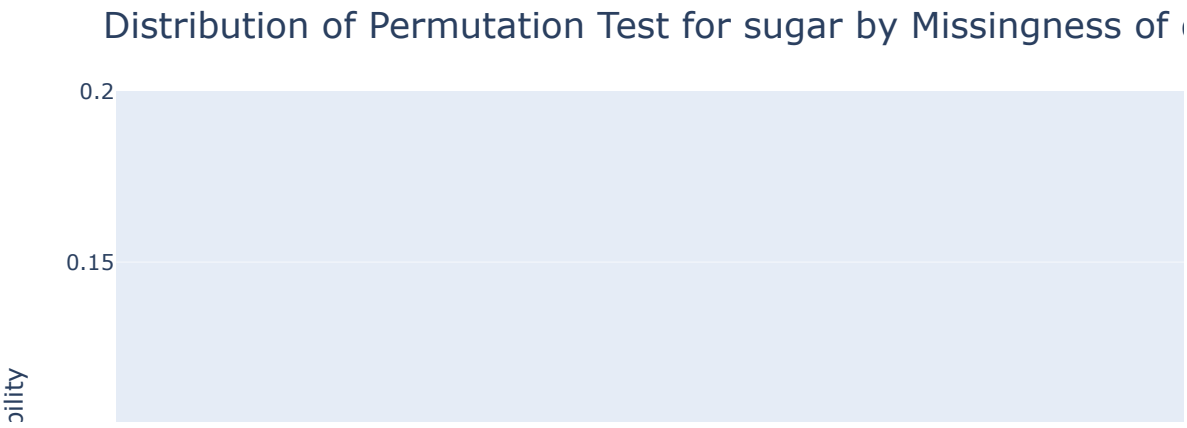
In [21]:
```
# create new dataframe for investigating missing dependency for description column
recipe_mar = recipe.copy()
recipe_mar['desc_missing'] = recipe_mar['description'].isna()
```

In [22]:
```
recipe_mar_sugar = recipe_mar[['sugar', 'desc_missing']]

recipe_vis = remove_outlier(recipe_mar_sugar, ['sugar'], 95)

# distribution of sugar, conditional on the missingness of description
fig = create_kde_plotly(recipe_vis, 'desc_missing', True, False, 'sugar', [0, 400],
                            f"Distribution of Sugar by Missingness of Description")

fig.show()
```



From the plotting, we see that the above two distribution of `sugar`, conditional on the missingness of `description` looks quite difference. I think using K-S statistics is better to capture this difference than absolute difference of mean. So we will use **K-S statistics** as our test statistics for the permutation test.

In [23]:
```
p_value = permutation_ks(recipe_mar_sugar, 1000, 'sugar', 'desc_missing')
print(f'''p_value for permutaiton test for sugar \
by missingness of descrpition is {p_value}''')
```




```
p_value for permutaiton test for sugar by missingness of descrpition is 0.023
```

From the graph above and our permutation test, we get a p-value of 0.023, which is less than our significance level of 0.05, so we reject the null hypothesis.

Thus, we conclude that **the missingness of `descrpition` does depends on the `sugar` column.**

Does the missingness of `description` depend on `minutes` column?

Null Hypothesis: the missingness of `description` does not depend on `minutes`.

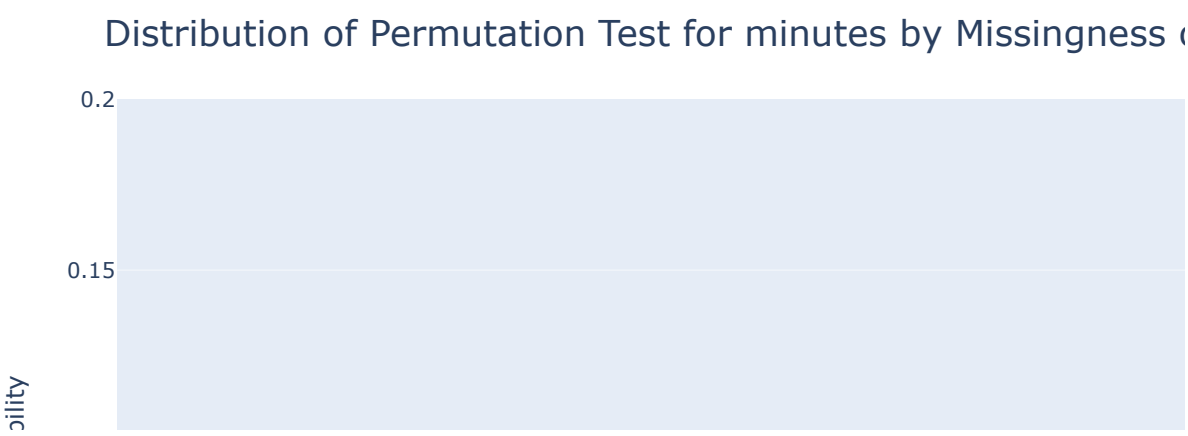
Alternative Hypothesis: the missingness of `description` does depend on `minutes`.

Let's look at the distribution of `minutes`, conditional on the missingness of `description` column.

In [24]:
```
recipe_mar_minute = recipe_mar[['minutes', 'desc_missing']]

recipe_vis = remove_outlier(recipe_mar_minute, ['minutes'], 93)

# distribution of sugar, conditional on the missingness of description
fig = create_kde_plotly(recipe_vis, 'desc_missing', True, False, 'minutes', [0, 200],
                        f"Distribution of Minutes by Missingness of Description")
fig.show()
```

From the plotting, we see that the distribution of `minutes` kinds have a smiliar shape, conditional on the missingness of `description`. We will use **K-S** statistics as our test statistics to capture this difference in shape for the permutation test.

In [25]:
```
p_value = permutation_ks(recipe_mar_minute, 1000, 'minutes', 'desc_missing')
print(f'''p_value for permutaiton test for minutes \
by missingness of descrpition is {p_value}''')
```




```
p_value for permutaiton test for minutes by missingness of descrpition is 0.411
```

From the graph above and our permutation test, we get a p-value of 0.411 that's greater than our significance level of 0.05, so we fail to reject the null hypothesis.

Thus, we conclude that **the missingness of `descrpition` does not depends on the `minutes` column.**

4. Hypothesis Testing

The question we are trying to answer in this section is that: **Do popular recipes (high review count) have a lower sugar level than those that aren't (low review count)** We define high review count threshold as having more than **10** reviews.

Null Hypothesis: The receipes with high review count does not have a lower sugar level than those with low review count, and the observed differences in our samples are due to random chance.

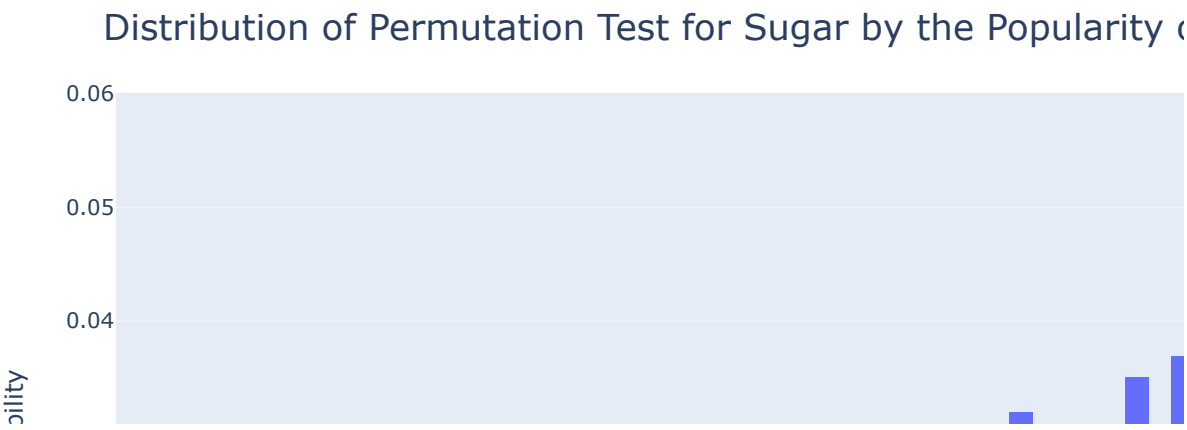
Alternative Hypothesis: The recipes with high review count do indeed have a lower sugar level than those with low review count, and the observed difference in our samples cannot be explained by random chance alone.

Test statistics Since our variable of interest are numerical and our test are one-tail test, meaning there is a direction in our alternative hypothesis, we will use difference in mean as our test statistics for the permutation test.

In [26]:
```
# create class label for whether the recipe is popular or not
df_hyp = recipe.copy()
df_hyp['is_popular'] = (df_hyp['n_review']
                        .apply(lambda n_review: True
                               if n_review >=10
                               else False))
```

In [27]:
```
'''
perform permutation testing using difference in mean test statistics
'''
def permutation_diff(recipe, n_rep, col, col_label):
    df = recipe.copy()

    simulated_stats = []

    # Computing and storing the absolute difference in means.
    grouped_stat = df.groupby(col_label)[col].mean()
    obs_stat = grouped_stat.loc[False] - grouped_stat.loc[True]

    for _ in range(n_rep):

        # Shuffling the values, while keeping the group labels in place.
        df[f'shuffled_{col_label}'] = np.random.permutation(df[col_label])

        # Computing and storing the difference in means.
        grouped_stat = df.groupby(f'shuffled_{col_label}')[col].mean()
        sim_stat = grouped_stat.loc[False] - grouped_stat.loc[True]

        simulated_stats.append(sim_stat)

    fig = px.histogram(pd.DataFrame(simulated_stats), x=0,
                       nbins=80, histnorm='probability')

    fig.add_vline(x=obs_stat, line_color='red', line_width=1, opacity=1)
    fig.add_annotation(
        text=f'''<span style="color:red">Observed Difference in Means =
{round(obs_stat, 2)}</span>''',
        x=obs_stat-3, showarrow=False, y=0.03)

    alpha_level = np.quantile(simulated_stats, 0.95)
    fig.add_vline(x=alpha_level, line_color='blue', line_width=1, opacity=1)
    fig.add_annotation(
        text=f'''<span style="color:blue">Significance Level of 0.05 =
{round(alpha_level, 2)}</span>''',
        x=alpha_level+3, showarrow=False, y=0.04)

    fig.update_layout(xaxis_range=[-25, 25])

    fig.update_layout(
        title=dict(text=f'''Distribution of Permutation Test for Sugar \
by the Popularity of Recipe''',
                   font=dict(size=20)))
    fig.show()

    return np.mean(np.array(simulated_stats) >= obs_stat)
```

In [28]:
```
p_value = permutation_diff(df_hyp, 1000, 'sugar', 'is_popular')
print(f'p_value for permutaiton test for sugar is {p_value}')
```




```
p_value for permutaiton test for sugar is 0.0
```

From the graph above, we see that our observed difference in mean is greater than the significance level of 0.05, indicating that the observed statistics in our sample is not merely coincidental. The p-value we obtained from performing our permutation testing is 0。0, which is less than our significance level of 0.05. So, we **reject our null hypothesis** and in favor of our alternative hypothesis that the recipes with high review count might have a lower sugar level than those with low review count.

Step 5: Framing a Prediction Problem

What are some of the characteristics of the recipe that are both popular and highly rated?

Prediction Problem: **What category does a recipe fall into based on its popularity level and average rating score?**

Category 1: Low review count, low average rating

Category 2: Low review count, high average rating

Category 3: High review count, low average rating

Category 4: High review count, high average rating

We define the threshold for high and low review count as **10** reviews count and the threshold for high and low average rating score as **4.8**

Step 6: Baseline Model

In [29]:
```
# add class
recipe = recipe.pipe(add_class);

# split data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(
    recipe.drop(['class', 'rating', 'n_review'], axis=1),
    recipe['class'], test_size=0.2, stratify=recipe['class'])
```

6.1 Feature Engineering

`minutes`, `protein`, `sodium`, `saturated_fat`, `total_fat`, `carbohydrates`

- Type: quantative continuous
- Feature Transformation: use RobustScaler to reduce the impact of outlier

`n_steps`, `n_ingredients`

- Type: quantative discrete
- Feature Transformation: passthrough

`calories`, `sugar`

- Type: quantative to nominal
- Feature Transformation: cateogrize calories and sugar into 8 bins and do one-hot encoding

`time`

- Type: quantative
- Feature Transformation: extract year, month, and day from `submitted` timestamp column

`recipe_complexity`

- Type: nominal
- Feature Transformation: binarize `n_steps` and `n_ingredients` to represent recipe complexity

In [30]:
```
# extract year, month, and day
pl_time = Pipeline([
    ('time', FunctionTransformer(lambda df:
                                 pd.concat([df['submitted'].dt.year,
                                            df['submitted'].dt.month,
                                            df['submitted'].dt.day], axis=1)))
])

# remove the impact of outlier, put numerical into bins, and OHE
pl_bins = Pipeline([
    ('outlier', RobustScaler()),
    ('to_bins', KBinsDiscretizer(n_bins=8))
])

# create new feature to represent recipe complexity
def recipe_complexity(df):
    return (((df['n_steps'] >= 10) & (df['n_ingredients'] >= 10))
            .astype(int)
            .to_numpy().reshape(-1, 1))
```

```
pl_complexity = Pipeline([
    ('complex', FunctionTransformer(recipe_complexity))
])

# transform existing columns to features
col_trans = ColumnTransformer(
    transformers=[
        ('outlier', RobustScaler(), ['minutes', 'protein', 'sodium',
                                     'saturated_fat', 'total_fat', 'carbohydrates']),
        ('pass', 'passthrough', ['n_steps', 'n_ingredients']),
        ('to_bin', pl_bins, ['calories', 'sugar']),
        ('time', pl_time, ['submitted']),
        ('complexity', pl_complexity, ['n_steps', 'n_ingredients']),
    ], remainder='drop'
)

pl_clf = Pipeline([
     ('col_trans', col_trans),
     ('clf', RandomForestClassifier())
])
```

6.2 Baseline Model Building and Performance Evaluation

In [31]:
```
baseline_model = pl_clf.fit(X_train, y_train)

test_pred = baseline_model.predict(X_test)

confusion_mtx = confusion_matrix(y_test, test_pred)
```

In [32]:
```
baseline_model.score(X_test, y_test)
```

Out[32]:
```
0.5943332306744687
```

Let's look at the confusion matrix to understand our result.

In [33]:
```
def display_confusion_mtx(confusion_mtx):

    fig = ff.create_annotated_heatmap(confusion_mtx,
                                      x=[1, 2, 3, 4],
                                      y=[1, 2, 3, 4],
                                      colorscale='Viridis',
                                      showscale=True)

    fig.update_layout(title=dict(text='<b>Confusion matrix</b>',
                                 font=dict(size=24), x=0.5),
                      )

    # add xaxis title
    fig.add_annotation(dict(font=dict(color="black",size=16),
                            x=0.5,
                            y=-0.15,
                            showarrow=False,
                            text="Predicted value",
                            xref="paper",
                            yref="paper"
                           ))

    # add yaxis title
    fig.add_annotation(dict(font=dict(color="black",size=16),
                            x=-0.08,
                            y=0.5,
```

```
                              showarrow=False,
                              text="Actual value",
                              textangle=-90,
                              xref="paper",
                              yref="paper"))

    fig.show()


display_confusion_mtx(confusion_mtx)
```

From the confusion matrix, we see that the baseline model misclassify lots of recipe into class lable 1 and 2. In addition, the baseline model have trouble classifying class label 3 and 4, which represents high review counts. This is caused by the imbalanced nature of data in our dataset. Recipes with high review count is rare and constitute a minority groups of data while recipes with low review count is more common and consitute a majority group of data. This is something we need to address in building the final model.

In [34]:
```
model_metr = pd.DataFrame(
    precision_recall_fscore_support(y_true=y_test,
                                    y_pred=test_pred)
).round(2)
model_metr = model_metr.rename(columns = {0:1, 1:2, 2:3, 3:4},
                               index={0:'precision', 1:'recall',
                                      2:'fscore', 3:'support'})
model_metr
```

Out[34]:

	1	2	3	4
precision	0.43	0.61	0.0	0.0
recall	0.13	0.91	0.0	0.0
fscore	0.20	0.73	0.0	0.0
support	5765.00	9796.00	307.0	367.0

In [35]:
```
print(classification_report(y_test, test_pred))
```

```
              precision    recall  f1-score   support

           1       0.43      0.13      0.20      5765
           2       0.61      0.91      0.73      9796
           3       0.00      0.00      0.00       307
           4       0.00      0.00      0.00       367

    accuracy                           0.59     16235
   macro avg       0.26      0.26      0.23     16235
weighted avg       0.52      0.59      0.51     16235
```

If we use a dummy classifier, a classifier that ignores the input feature while making prediction, and compare the model performance with our baseline model:

In [36]:
```
dummy_clf = DummyClassifier()

dummy_clf.fit(X_test, y_test)
y_pred = dummy_clf.predict(X_test)

model_metr = pd.DataFrame(
    precision_recall_fscore_support(y_true=y_test,
                                    y_pred=y_pred)
)
model_metr = model_metr.rename(columns = {0:1, 1:2, 2:3, 3:4},
                               index={0:'precision', 1:'recall',
                                      2:'fscore', 3:'support'})
model_metr
```

Out[36]:

	1	2	3	4
precision	0.0	0.603388	0.0	0.0
recall	0.0	1.000000	0.0	0.0
fscore	0.0	0.752641	0.0	0.0
support	5765.0	9796.000000	307.0	367.0

In [37]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00      5765
           2       0.60      1.00      0.75      9796
           3       0.00      0.00      0.00       307
           4       0.00      0.00      0.00       367

    accuracy                           0.60     16235
   macro avg       0.15      0.25      0.19     16235
weighted avg       0.36      0.60      0.45     16235
```

Our baseline model is better than dummy classifier in the sense that dummy classifier classify all recipes into one class.

7. Final Model

7.1 Feature Engineering

`minutes`, `protein`, `sodium`, `saturated_fat`, `total_fat`, `carbohydrates`

- Type: quantative continuous
- Feature Transformation: use RobustScaler to reduce the impact of outlier

`n_steps`, `n_ingredients`

- Type: quantative discrete
- Feature Transformation: passthrough

`time`

- Type: quantative
- Feature Transformation: extract year, month, and day from `submitted` timestamp column

`recipe_complexity`

- Type: nominal
- Feature Transformation: binarize `n_steps` and `n_ingredients` to represent recipe complexity

The following features are added for the final model:

`description`

- Type: text data
- Feature Transformation:
- Build a list of vocabulary from high Inverse Document Frequency(IDF) words from the `description` of **low reivew count recipes** (minority class label, class 3 and 4)
- Vectorize the description text column using TF-IDF and the vocabulary from previous step
- For each recipe, extract the top 5 highest TF-IDF values as the features

`steps`

- Type: text data
- Feature Transformation:

- Build a list of vocabulary from high Inverse Document Frequency(IDF) words from the `steps` of **low reivew count recipes** (minority class label, class 3 and 4)
- Vectorize the steps text column using TF-IDF and the vocabulary from previous step
- For each recipe, extract the top 5 highest TF-IDF values as the features

`reviews`

- Type: text data
- Feature Transformation:
- Manaually create a list of sentiment words such as `good`, `loved`, `hated`, etc that are relevant for extracting sentiment features
- For each word in sentiment word list, binarize review based on whether the review comments contains that particular word

In [38]:
```
'''
extract high inverse document frequency vocabulary
'''
def create_vocabulary(df, col, n):
    vectorizer = TfidfVectorizer(analyzer='word',
                                 max_features=int(df.shape[0]*0.8),
                                 ngram_range=(1, 3))

    vectorizer.fit(df)

    feature_names = vectorizer.get_feature_names_out()
    tfidf = vectorizer.idf_

    top_n_index = np.argsort(tfidf)[::-1][:n]

    top_n_features = [feature_names[i] for i in top_n_index]

    return top_n_features

'''
build vocabulary from minority class
'''
df = pd.concat([X_train, y_train], axis=1)

df = df[df['class'].isin([3, 4])][['description', 'steps']]

df_desc = df['description'].fillna('')

df_steps = df['steps'].str.join(' ')

df_desc_voca = create_vocabulary(df_desc, 'description', int(df_desc.shape[0]*0.07))
df_steps_voca = create_vocabulary(df_desc, 'steps', int(df_steps.shape[0]*0.07))


'''
CountVectorize the description text column, based on high-idf vocabulary
'''
def desc_vec(df_desc, vocabulary):
    df = df_desc['description'].fillna('')

    vectorizer = TfidfVectorizer(analyzer='word',
                                 vocabulary=vocabulary
                                )

    count_mtx = pd.DataFrame(vectorizer.fit_transform(df).toarray())
```

```
    # remove empty column
    count_mtx = count_mtx.loc[:, count_mtx.sum() > 0]

    # for each row(recipe), select the top values across the columns
    count_mtx = count_mtx.T.apply(lambda vals: vals.nlargest(5).tolist()).T

    return count_mtx.to_numpy()


pl_description = Pipeline([
    ('vectroize', FunctionTransformer(desc_vec,
                                      kw_args={'vocabulary': df_desc_voca}
                                     )
    )])

'''
CountVectorize the steps text column, based on high-idf vocabulary
'''
def steps_vec(df_steps, vocabulary):

    df = df_steps['steps'].str.join(' ')

    vectorizer = TfidfVectorizer(analyzer='word',
                                 vocabulary=vocabulary
                                )

    count_mtx = pd.DataFrame(vectorizer.fit_transform(df).toarray())

    # remove empty column
    count_mtx = count_mtx.loc[:, count_mtx.sum() > 0]

    # for each row(recipe), select the top values across the columns
    count_mtx = count_mtx.T.apply(lambda vals: vals.nlargest(5).tolist()).T

    return count_mtx.to_numpy()


pl_steps = Pipeline([
    ('vectroize', FunctionTransformer(steps_vec,
                                      kw_args={'vocabulary': df_steps_voca}
                                     )
    )])
```

In [39]:

```
# the list of words we manaually created that we believed
# might be relevant for feature engineering
word_list = ['but', 'next', 'loved', 'hate', 'dislike',
             'keeper', 'bland', 'think', 'nice', 'however',
             'delicious', 'wonderful', 'fantastic', 'fav',
             'perfect', 'maybe', 'bad', 'would', 'good',
             'excellent', 'very', 'sorry', 'liked',
             'great', 'well', 'pretty good', 'awful',
             'yet', 'made']
'''
Count the number of occurence of word in input list
'''
def review_to_count(lst, word):
    count=0
    for review in lst:
        if pd.notnull(review) and word in review:
            count+=1
    return count

'''
```

```
One hot encoding review column by the given word_list
'''
def ohe_review(df, word_list, threshold):

    ohe = pd.DataFrame()
    df['reviews'] = df['reviews'].fillna('')

    for word in word_list:
        df['word_count'] = df['reviews'].apply(lambda lst: review_to_count(lst, word))

        word_ohe = (df['word_count'] > threshold).astype(int)

        ohe = pd.concat([ohe, word_ohe], axis=1)

    return ohe.to_numpy()

pl_review = Pipeline([
    ('review', FunctionTransformer(ohe_review,
                                   kw_args={'word_list': word_list,
                                            'threshold': 0}
                                  )
    )])
```

7.2 Final Model Building and Performance Evaluation

In [40]:
```
y_train.value_counts()
```

Out[40]:
```
2    39183
1    23060
4     1467
3     1228
Name: class, dtype: int64
```

The approach we are trying to use for final model is to break the multi-class classification problem into two binary classification problem. We will build two classifiaion model, one for clasifying whether the recipe has high or low review count, and one for classifying whether the recipe has high or low average rating. We will them combine the two binary class result into 4 different class labels, making up of our multi-class classification problem.

We observe that there is a severe data imbalance issue in classifying recipes based on review count, where low review count constitutes the majority class and high review count constitutes the minority class. The model we are going to use is the **BalancedRandomForest** from imbalanced-learn library. The main difference between balanced random forest and random forest from sklearn is that it will bootstrap sample from the minority class and sample with replacement the same number of samples from the majority class while building individual decision tree. This model will help us increase our precision of classifying minority class.

In [41]:
```
'''
turn a mutl-class labels to two binary classl label
'''
def to_n_review(ser_class):
    # 1 means high review count, 0 means low review count
    return ser_class.replace([1, 2], 0).replace([3, 4], 1)

def to_rating(ser_class):
    # 1 means high average rating, 0 means low average rating
    return ser_class.replace([1, 3], 0).replace([2, 4], 1)
```

```
from sklearn.base import BaseEstimator, TransformerMixin, clone

'''
Custom classification model that takes in two model,
classifying the label seperately, and combine the result
'''
class ClassificationTransformer(BaseEstimator, TransformerMixin):

    def __init__(self, model_1, model_2):
        self.model_1 = clone(model_1)
        self.model_2 = clone(model_2)

    def fit(self, X, Y):
        self.model_2.fit(X, Y[1])
        self.model_1.fit(X, Y[0])

        return self

    def predict(self, X):
        # 1 means high review count, 0 means low review count
        y_1 = self.model_1.predict(X)

        # 1 means high average rating, 0 means low average rating
        y_2 = self.model_2.predict(X)

        y = np.zeros(len(X))

        # Find indices in which y_1 & y_2 are 0
        indices_00 = np.logical_and(y_1 == 0, y_2 == 0)

        # y_1 is 0 & y_2 is 1
        indices_01 = np.logical_and(y_1 == 0, y_2 == 1)

        # y_1 is 1 & y_2 is 0
        indices_10 = np.logical_and(y_1 == 1, y_2 == 0)

        # y_1 & y_2 are 1
        indices_11 = np.logical_and(y_1 == 1, y_2 == 1)

        # Assign values to y
        y[indices_00] = 1
        y[indices_01] = 2
        y[indices_10] = 3
        y[indices_11] = 4

        return y
```

In [42]:

```
# Feature Engineering
col_trans = ColumnTransformer(
    transformers=[
        ('outlier', RobustScaler(), ['minutes', 'protein', 'sodium',
                                     'saturated_fat', 'total_fat',
                                     'carbohydrates']), # 0-5
        ('pass', 'passthrough', ['n_steps', 'n_ingredients']), # 6-7
        ('time', pl_time, ['submitted']),# 8-10
        ('complexity', pl_complexity, ['n_steps', 'n_ingredients']), # 11
        ('description', pl_description, ['description']), # 12-16
        ('steps', pl_steps, ['steps']), # 16-20
        ('review', pl_review, ['reviews']) # 21-50
    ], remainder='drop'
)
```

```
# model for classifying recipes based on reiview count
clf_n_review = BalancedRandomForestClassifier(class_weight='balanced',
                                              n_estimators=180,
                                              max_depth=16,
                                              criterion='entropy'
                                             )
#sklearn.ensemble.ExtraTreesClassifier
# model for classifying recipes based on average rating
clf_rating = RandomForestClassifier(class_weight='balanced',
                                    n_estimators=150,
                                    max_depth=19,
                                    criterion='entropy'
                                   )


# pipeline that combines everything
pl_clf = Pipeline([
     ('col_trans', col_trans),
     ('clf', ClassificationTransformer(clf_n_review, clf_rating)),
])
```

Model Hyperparameter Tunning

We already found our best hyperparameter for the model. For the sake of time, I will comment out the code for search.

In [43]:
```
def custom_grid_search():
    n_estimators_lst = [50, 80, 130, 180, 200]
    max_depth_lst = [2, 10, 19, 20, 35]

    errs_df = pd.DataFrame()

    # we will use 5 folds here
    skf = StratifiedKFold(n_splits=5)

    for n_est in n_estimators_lst:
        for max_depth in max_depth_lst:

            accuracy_score_lst = []
            # train, validation split
            for train_index, test_index in skf.split(X_train, y_train):
                X_train_k, X_valid_k = (X_train.iloc[train_index],
                                        X_train.iloc[test_index])
                y_train_k, y_valid_k = (y_train.iloc[train_index],
                                        y_train.iloc[test_index])

                clf_n_review = BalancedRandomForestClassifier(class_weight='balanced',
                                                              n_estimators=n_est,
                                                              max_depth=max_depth
                                                             )
                clf_rating = RandomForestClassifier(class_weight='balanced',
                                                    n_estimators=n_est,
                                                    max_depth=max_depth
                                                   )

                pl_clf = Pipeline([
                     ('col_trans', col_trans),
                     ('clf', ClassificationTransformer(clf_n_review, clf_rating)),
                ])

                pl_clf.fit(X_train_k, [to_n_review(y_train_k), to_rating(y_train_k)])
```

```
                test_pred = pl_clf.predict(X_valid_k)

                accuracy = accuracy_score(y_valid_k, test_pred)

                accuracy_score_lst.append(accuracy)

            errs_df[f'({n_est}, {max_depth})'] = accuracy_score_lst

    errs_df.index = [f'Fold {i}' for i in range(0, 5)]
    errs_df.index.name = 'Validation Fold'

    return errs_df

# errs_df = custom_grid_search()
# print(errs_df.mean().idxmin())
```

In [44]:
```
# traning and predicting
final_model = pl_clf.fit(X_train, [to_n_review(y_train), to_rating(y_train)])

test_pred = final_model.predict(X_test)

confusion_mtx = confusion_matrix(y_test, test_pred)
```

In [45]:
```
display_confusion_mtx(confusion_mtx)
```

In [46]:
```
print(classification_report(y_test, test_pred))
```

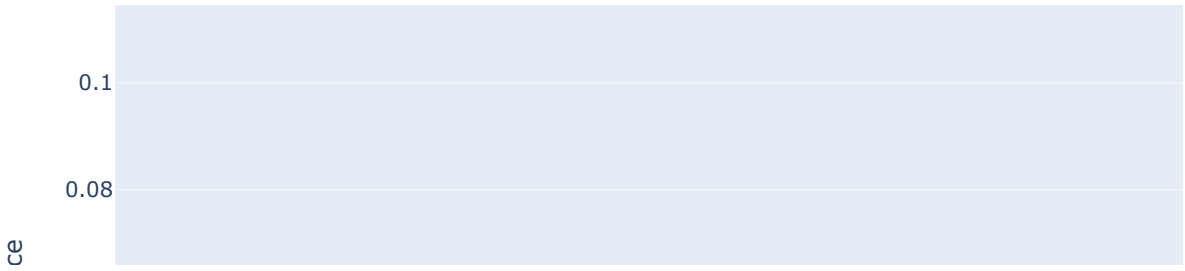
```
              precision    recall  f1-score   support

           1       0.59      0.47      0.52      5765
           2       0.74      0.73      0.74      9796
           3       0.21      0.55      0.30       307
           4       0.20      0.62      0.31       367

    accuracy                           0.63     16235
   macro avg       0.43      0.59      0.47     16235
weighted avg       0.66      0.63      0.64     16235
```

In [47]:
```
model_metr = pd.DataFrame(
    precision_recall_fscore_support(y_true=y_test,
                                    y_pred=test_pred)
).round(2)
model_metr = model_metr.rename(columns = {0:1, 1:2, 2:3, 3:4},
                               index={0:'precision', 1:'recall',
                                      2:'fscore', 3:'support'})
model_metr
```

Out[47]:

	1	2	3	4
precision	0.59	0.74	0.21	0.20
recall	0.47	0.73	0.55	0.62
fscore	0.52	0.74	0.30	0.31
support	5765.00	9796.00	307.00	367.00

In [48]:
```
tmp_mdl = pd.DataFrame(
    {'Feature': np.arange(0, 51),
     'Feature importance': (final_model.named_steps['clf']
                            .model_1.feature_importances_)
    })
tmp_mdl['Feature'] = tmp_mdl['Feature'].astype(str)

fig = go.Figure(data=[go.Bar(
    x=tmp_mdl['Feature'],
    y=tmp_mdl['Feature importance'],
    marker_color=tmp_mdl['Feature importance'],
    marker_colorscale='agsunset',
)])
fig.update_xaxes(tickangle=45)
fig.update_layout(
    title=dict(text="Final Model Feature Importance", font=dict(size=20)),
    xaxis_title="Features",
    yaxis_title='Importance'
)

fig.show()
```



Top 10 most "useful" sentiment words for our classification model

In [49]:
```
index = (tmp_mdl.iloc[21:]
         .sort_values('Feature importance', ascending=False)
         .iloc[:10]['Feature']
         .astype(int)-22)

pd.Series(word_list).iloc[index].reset_index(drop=True)
```

Out[49]:
```
0        great
1         good
2    delicious
3         very
4         made
5          but
6        would
7        loved
8      perfect
9    wonderful
dtype: object
```

Let's train our model using all available data.

In [50]:
```
final_model = pl_clf.fit(
    recipe.drop(['class', 'rating', 'n_review'], axis=1),
    [to_n_review(recipe['class']), to_rating(recipe['class'])])

final_pred = final_model.predict(
    recipe.drop(['class', 'rating', 'n_review'], axis=1))
```

8. Fairness Analysis

Are recipes with `vegetarian` tags more likely to be correctly classified as to the high average rating category by the model, compared to those without the `vegetarian` tags?" In the context of our modeling analysis, we will compare the **precision** acorss two groups. If the precision for recipes with the 'vegetarian' tag is statistically significantly higher than the precision for recipes without the 'vegetarian' tag, it could potentially indicate a bias towards classifying `vegetarian` recipes as high average rating more often, even when they should not be classified as such. In our dataset, high average rating means class category 2 and 4.

Null Hypothesis: Our model is fair. Our classifier's precision is the same for recipes with and without `vegetarian` tag, and any differences are due to random chance.

Alternative Hypothesis: Our model is not fair. Our classifier's precision is higher for recipes with `vegetarian` tag than those without, and any observed differences can not be explained by random chance alone.

Test statistic: Difference in average precision of class 2 and 4 (without `vegetarian` tag - with `vegetarian` tag).

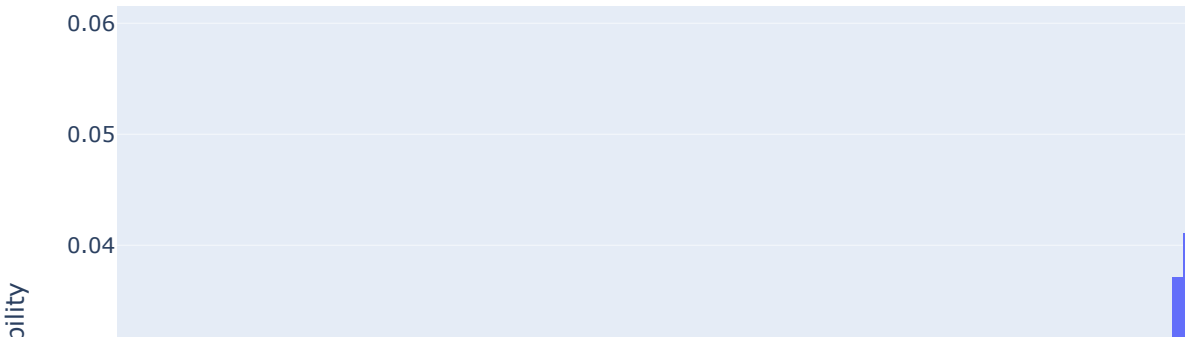
Significance level: 0.05.

In [51]:
```
tag = 'vegetarian'

# preprocessing data by coping from fitted model result
result = recipe.copy()

result['has_tags_vegetarian'] = result['tags'].apply(lambda lst: tag in lst)
result['prediction'] = final_pred

result = result[['has_tags_vegetarian', 'prediction',  'class']]

# function to compute average precision for predicting class 2 and 4
compute_precision = lambda df: precision_score(df['class'], df['prediction'],
                                               average=None,
                                               labels=[2, 4]).mean()
```

In [52]:
```
'''
perform permutation testing using difference in mean test statistics
'''
def permutation_diff(recipe, n_rep, col_label):
    df = recipe.copy()

    simulated_stats = []

    # Computing and storing the absolute difference in means.
    grouped_stat = df.groupby(col_label).apply(compute_precision)
    obs_stat = grouped_stat.loc[False].mean() - grouped_stat.loc[True].mean()

    for _ in range(n_rep):

        # Shuffling the values, while keeping the group labels in place.
        df[f'shuffled_{col_label}'] = np.random.permutation(df[col_label])

        # Computing and storing the difference in means.
        grouped_stat = df.groupby(f'shuffled_{col_label}').apply(compute_precision)
        sim_stat = grouped_stat.loc[False] - grouped_stat.loc[True]
```

```
        simulated_stats.append(sim_stat)

    #making permutation distribution plot
    fig = px.histogram(pd.DataFrame(simulated_stats), x=0,
                       nbins=80, histnorm='probability')

    fig.add_vline(x=obs_stat, line_color='red', line_width=1, opacity=1)
    fig.add_annotation(
        text=f'''<span style="color:red">Observed Difference in Means
= {round(obs_stat, 2)}</span>''',
        x=obs_stat+0.017, showarrow=False, y=0.06)

    alpha_level = np.quantile(simulated_stats, 0.95)
    fig.add_vline(x=alpha_level, line_color='blue', line_width=1, opacity=1)
    fig.add_annotation(
        text=f'''<span style="color:blue">Significance Level of 0.05
= {round(alpha_level, 2)}</span>''',
        x=alpha_level+0.017, showarrow=False, y=0.04)

    fig.update_layout(xaxis_range=[-0.1, 0.1])

    fig.update_layout(
        title=dict(text=f'''Distribution of Difference in Precision \
by the Vegetarian Tags in Recipe''',
                   font=dict(size=20)))
    fig.show()

    return np.mean(np.array(simulated_stats) >= obs_stat)
```

In [53]:
```
p_value = permutation_diff(result, 1000, 'has_tags_vegetarian')
print(f'p_value for permutaiton test for \
the difference in precision is {p_value}')
```



`p_value for permutaiton test for the difference in precision is 0.397`

From the graph above, we see that our observed difference in precision is lower than the significance level of 0.05, indicating that the observed statistics in our sample is by random chance alone. The p-value we obtained from performing our permutation testing is around 0.397, which is greater than our significance level of 0.05. So, we **fail to feject our null hypothesis** that our classifier's precision is likely the same for recipes with and without `vegetarian` tag, and any differences are due to random chance. Our model **achieves precision parity** for groups with `vegetarian` tag and groups without `vegetarian` tag.