# Data Scientist Professional Practical Exam Submission

**Use this template to write up your summary for submission. Code in Python or R needs to be included.**

## Task List

Your written report should include both code, output and written text summaries of the following:

- Data Validation:
  - Describe validation and cleaning steps for every column in the data
- Exploratory Analysis:
  - Include two different graphics showing single variables only to demonstrate the characteristics of data
  - Include at least one graphic showing two or more variables to represent the relationship between features
  - Describe your findings
- Model Development
  - Include your reasons for selecting the models you use as well as a statement of the problem type
  - Code to fit the baseline and comparison models
- Model Evaluation
  - Describe the performance of the two models based on an appropriate metric
- Business Metrics
  - Define a way to compare your model performance to the business
  - Describe how your models perform using this approach
- Final summary including recommendations that the business should undertake

*Start writing report here..*

# *Recipe Popularity Prediction Project*

## Objective and Approach

The objective is recipe prediction based on the ideal outcome to be an 80% popularity rate, thereby avoiding exposure to unpopular recipes.

Here is a step-by-step procedure to achieve that:

### Data Validation

1. **Remove Duplicates:**

- Ensure that the data does not contain duplicate entries. It can be done using the `drop_duplicates` method in pandas or through Excel's "Remove Duplicates" functionality under the Data tab.

2. **Handle Missing Values:**

- Identify the missing values in the dataset and handle them appropriately. This may involve the replacement of missing values with the mean, median, or mode, or might involve advanced imputation based on the nature of the data.

3. **Fix Data Types**: Ensure that the data types of columns in the dataframe are suitable for dataframe; modify them if not. Convert columns to numeric, categorical, datetime data type as appropriate.

### Exploratory Data Analysis

**EDA:**

- Carry out proper EDA to identify any pattern/insight from the data. Check among others distribution, correlation, and other statistical properties of the data. Visualize the data by plotting: histograms, box plots, scatter plots. This may provide insights into the trends and dependencies that exist within the data.

### Machine Learning Model Development

**Feature Engineering**:

- Standardize and create new features that could strengthen the predictability of a model about the popularity of a recipe.

**Model Selection and Training**:

- Choose a relevant machine learning model: regression or classification, based on the nature of the problem.
- Train the model on the prepared dataset and evaluate its performance based on appropriate metrics; examples include Accuracy, Precision, Recall, and the F1 score.

**Model Evaluation**:

- Assess the model's performance in finding areas of improvement. This could be through hyperparameter tuning, feature selection, or the testing of various models.

### Business Metric Alignment and Recommendations

**Key Metric Definition**:

- Define the business metrics aligned with the vision of attaining an 80% popularity rate of recipes. It could be done on the basis of metrics such as average user engagement, adoption rate of the recipes, and customer satisfaction.

**Performance Monitoring**:

- Set up a system to monitor these metrics constantly so that the model predictions are on par with the set business objectives.

**Recommendations:** Therefore, based on the insight from data analysis and performance of our models, do provide recommendations to business regarding optimization of recipe content, marketing strategies, or improving user experience that would help enhance the popularity of a recipe.

This structured approach reassures us that our solution will be data-driven, effective, and aligned with the objectives of the business: maximize exposure to popular recipes and minimize the exposure of unpopular recipes.

```
#importing relevant libraries
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import boxcox, yeojohnson
from scipy.stats.mstats import winsorize
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
```

```
#read recipe_site_traffic.csv file and convert it into dataframe
recipe_site_traffic = pd.read_csv("recipe_site_traffic_2212.csv")

#checking first 5 rows of the dataframe
recipe_site_traffic.head()
```

| | recipe | calories | carbohydrate | sugar |
|---|---|---|---|---|
| 0 | 1 | null | null | |
| 1 | 2 | 35.48 | 38.56 | |
| 2 | 3 | 914.28 | 42.68 | |
| 3 | 4 | 97.03 | 30.56 | |
| 4 | 5 | 27.05 | 1.85 | |

5 rows

```
#finding the number of duplicated recipes
recipe_site_traffic.duplicated(subset='recipe').sum()
```

```
0
```

## Data Exploration and Preparation

Since the dataset doesn't have any duplicate records, let's see some information about the structure of the dataset.

### Dimensions of Dataset and Column Info

First, let's find out how many rows and columns make up this dataset. We will also fetch details on column names and their respective data types.

#### Column-Level Analysis

To get a better idea of what constitutes this dataset, we will:

1. **Column names and data type assessment:** Column names clearly define the data they hold, and each column's data type is correctly set for the type of its values.

2. **Non-null value counts:** Use to get the count of non-null values within a column. You can also use it as an indication of possible missing values for columns.

**Handling Data Quality Issues**

In case your analysis uncovers any data quality issues, like:

1. **Data type mismatches:** Perform proper data type casting in such cases to make them both accurate and consistent.
2. **Missing value handling:** We will design a strategy for handling the missing values. It can be through imputation, interpolation, or any other technique. This will ensure our dataset is well-structured and accurate for further analysis and modeling.

```
recipe_site_traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 947 entries, 0 to 946
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   recipe        947 non-null    int64  
 1   calories      895 non-null    float64
 2   carbohydrate  895 non-null    float64
 3   sugar         895 non-null    float64
 4   protein       895 non-null    float64
 5   category      947 non-null    object 
 6   servings      947 non-null    object 
 7   high_traffic  574 non-null    object 
dtypes: float64(4), int64(1), object(3)
memory usage: 59.3+ KB
```

```
#checking the values of servings column
recipe_site_traffic['servings'].value_counts()
```

| |
|---|
| 4 |
| 6 |
| 2 |
| 1 |
| 4 as a snack |
| 6 as a snack |

6 rows

```
#replacing the rows including "as a snack" with their relevant numeric number
recipe_site_traffic['servings'] = recipe_site_traffic['servings'].str.replace(" as a snack",
"")

#checking the values of servings column again
```

```
recipe_site_traffic['servings'].value_counts()
```

| | servings |
|---|---|
| 4 | |
| 6 | |
| 2 | |
| 1 | |

4 rows

```
#converting data type of servings column to integer
recipe_site_traffic['servings'] = recipe_site_traffic['servings'].astype('int')
```

```
#checking the values of high_traffic column
recipe_site_traffic['high_traffic'].value_counts()
```

| | high_traffic |
|---|---|
| High | |

1 rows

```
#replacing the rows with value "High" with True, and null values with False
recipe_site_traffic['high_traffic'] = np.where(recipe_site_traffic['high_traffic'] == "High",
True, False)

#checking the values of high_traffic column again
recipe_site_traffic['high_traffic'].value_counts()
```

| | high_traffic |
|---|---|
| True | |
| False | |

2 rows

```
#checking the values of category column
recipe_site_traffic['category'].value_counts()
```

| |
|---|
| Breakfast |
| Chicken Breast |
| Beverages |
| Lunch/Snacks |
| Potato |
| Pork |
| Vegetable |
| Dessert |
| Meat |
| Chicken |
| One Dish Meal |

11 rows

```
#replacing the rows including "as a snack" with their relevant numeric number
recipe_site_traffic['category'] = recipe_site_traffic['category'].str.replace(" Breast", "")

#checking the values of servings column again
recipe_site_traffic['category'].value_counts()
```

| |
|---|
| Chicken |
| Breakfast |
| Beverages |
| Lunch/Snacks |
| Potato |
| Pork |
| Vegetable |
| Dessert |
| Meat |
| One Dish Meal |

10 rows

```
#converting data type of category column to category
recipe_site_traffic['category'] = recipe_site_traffic['category'].astype('category')
```

```
#checking the summary of dataframe's structure
recipe_site_traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 947 entries, 0 to 946
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   recipe        947 non-null    int64
 1   calories      895 non-null    float64
 2   carbohydrate  895 non-null    float64
 3   sugar         895 non-null    float64
 4   protein       895 non-null    float64
 5   category      947 non-null    category
 6   servings      947 non-null    int64
 7   high_traffic  947 non-null    bool
dtypes: bool(1), category(1), float64(4), int64(2)
memory usage: 46.7 KB
```

```
#checking the values of servings column
recipe_site_traffic['servings'].value_counts()
```

| | servings |
|---|---|
| 4 | |
| 6 | |
| 2 | |
| 1 | |

4 rows

```
#replacing the rows including "as a snack" with their relevant numeric number
recipe_site_traffic['servings'] = recipe_site_traffic['servings'].astype(str)
recipe_site_traffic['servings'] = recipe_site_traffic['servings'].str.replace(" as a snack",
"")

#checking the values of servings column again
recipe_site_traffic['servings'].value_counts()
```

| | servings |
|---|---|
| 4 | |
| 6 | |
| 2 | |
| 1 | |

4 rows

Data Type Optimization: Servings Column

The "servings" column is purely an integer in nature but is not represented in integer data type format at this point in time.

We cast the "serving" column to an integer format because it will provide consistency in the data.

**Key Concerns**

1. **Variable Nature:**
    - Numerical by nature but functions as a categorical feature.

- Each size of serving classifies into a different class and hence is not continuous, per-se.

2. **Strategy for Data Type:**
    - To retain numeric as of now
    - Shall be changed to integer in:
        - Model development phase
        - Model evaluation stage

This approach maintains data integrity while offering the flexibility needed later in modelling.

```
#converting data type of servings column to integer
recipe_site_traffic['servings'] = recipe_site_traffic['servings'].astype('int')
```

Analyzing the "High_Traffic" Column

We need to analyze the distribution of values within the "high_traffic" column, which at the moment is an object data type. This will help us in:

1. **Understanding Value Distribution**
    - Getting unique values
    - Checking inconsistency in the captured values
    - Verification of data patterns
2. **Determining Optimal Data Type**
    - Is the column better to convert to: Boolean Numeric Categorical
    - Choose right data type according to analysis

By doing this, that review will give us the guarantee that our data is properly treated for the highest performance in our future analysis.

```
#checking the values of high_traffic column
recipe_site_traffic['high_traffic'].value_counts()
```

| | high_traffic |
|---|---|
| True | |
| False | |

2 rows

Traffic Classification Analysis

**Column Transformation Strategy: High Traffic Indicator**

**Current Status of Column**

- "high_traffic" column is populated with primarily a single value: "High"
- Null values in the column will indicate "Low" traffic cases

**Transformation Recommended**

- The data type of the column will be boolean.
- Mapping Logic:
  - `True` = High Traffic
  - `False` = Low Traffic

**Rationale**

- Simplifies how data is represented
- Computational efficiency is enhanced
- Binary classification is explicit
- Integrity of original information preserved

**How to Achieve**

1. Replace null values with `False`
2. Convert "High" values to `True`
3. Normalize Boolean Representations

**Benefits**

- Saves on Processing Time
- Model Interpretations Are Easier
- Can use more efficient Machine Learning Algorithms

**How It Was Done: Critical Transformations**

Identify null/low traffic entries Map to `False` Verify boolean conversion was done correctly

Category Column Analysis and Conversion

**Current Status**

The "category" column is of object data type and needs to be optimized for our analytics.

**Activities to Perform**

1. **Value Distribution Analysis**
   - Get unique values from "category" column
   - Look for any pattern or inconsistencies
   - Understand the frequency distribution of categories
2. **Data Type Conversion

- Check if it is possible to convert it into a categorical data type
  - Also, there might be some hassles in conversion
  - Plan necessary cleaning steps to do

**Goal**

Ensure that "category" column is well formatted and changed to the appropriate category data type for best processing and model performances.

This process is very vital to:

- Enhance data processing performance
- Performing valid analysis
- Let models respect the categorical variables

```
#checking the values of category column
recipe_site_traffic['category'].value_counts()
```

| |
|---|
| Chicken |
| Breakfast |
| Beverages |
| Lunch/Snacks |
| Potato |
| Pork |
| Dessert |
| Vegetable |
| Meat |
| One Dish Meal |

10 rows

## Data Validation and Cleaning

This looks like there is an extra category, "Chicken Breast," in the "category" column. However, since the set criteria involve only a few categories like 'Lunch/Snacks', 'Beverages', 'Potato', 'Vegetable', 'Meat', 'Chicken', 'Pork', 'Dessert', 'Breakfast', and 'One Dish Meal,' it is not difficult to identify that the category 'Chicken Breast' does not align. Thus, it seems appropriate that this category go to the 'Chicken' category. Now, let's proceed and do the correction accordingly.

```
#replacing the rows including "as a snack" with their relevant numeric number
recipe_site_traffic['category'] = recipe_site_traffic['category'].str.replace(" Breast", "")

#checking the values of servings column again
recipe_site_traffic['category'].value_counts()
```

| |
|---|
| Chicken |
| Breakfast |
| Beverages |
| Lunch/Snacks |
| Potato |
| Pork |
| Vegetable |
| Dessert |
| Meat |
| One Dish Meal |

10 rows

```
#converting data type of category column to category
recipe_site_traffic['category'] = recipe_site_traffic['category'].astype('category')
```

```
#checking the summary of dataframe's structure
recipe_site_traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 947 entries, 0 to 946
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype   
---  ------        --------------  -----   
 0   recipe        947 non-null    int64   
 1   calories      895 non-null    float64 
 2   carbohydrate  895 non-null    float64 
 3   sugar         895 non-null    float64 
 4   protein       895 non-null    float64 
 5   category      947 non-null    category
 6   servings      947 non-null    int64   
 7   high_traffic  947 non-null    bool    
dtypes: bool(1), category(1), float64(4), int64(2)
memory usage: 46.7 KB
```

```
#checking missing numbers for each columns
recipe_site_traffic.isna().sum()
```

| |
|---|
| recipe |
| calories |
| carbohydrate |
| sugar |
| protein |
| category |
| servings |
| high_traffic |

8 rows

```
#showing all rows with missing values
recipe_site_traffic[recipe_site_traffic.isnull().any(axis=1)]
```

| | recipe | calories | carbohydrate | sugar |
|---|---|---|---|---|
| 0 | 1 | null | null | |
| 23 | 24 | null | null | |
| 48 | 49 | null | null | |
| 82 | 83 | null | null | |
| 89 | 90 | null | null | |
| 116 | 117 | null | null | |
| 121 | 122 | null | null | |
| 136 | 137 | null | null | |
| 149 | 150 | null | null | |
| 187 | 188 | null | null | |
| 209 | 210 | null | null | |
| 212 | 213 | null | null | |
| 221 | 222 | null | null | |
| 249 | 250 | null | null | |
| 262 | 263 | null | null | |

52 rows

```
#dropping missing values
recipe_site_traffic = recipe_site_traffic.dropna().reset_index(drop=True)
```

```
#checking missing numbers for each columns
recipe_site_traffic.isna().sum()
```

| |
|---|
| recipe |
| calories |
| carbohydrate |
| sugar |
| protein |
| category |
| servings |
| high_traffic |

8 rows

```
#checking the summary of dataframe's structure
recipe_site_traffic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 895 entries, 0 to 894
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   recipe        895 non-null    int64
 1   calories      895 non-null    float64
 2   carbohydrate  895 non-null    float64
 3   sugar         895 non-null    float64
 4   protein       895 non-null    float64
 5   category      895 non-null    category
 6   servings      895 non-null    int64
 7   high_traffic  895 non-null    bool
dtypes: bool(1), category(1), float64(4), int64(2)
memory usage: 44.2 KB
```

## Descriptive Statistics and Data Distribution

Now that we have completed data validation and cleaning on this dataframe, it is prepared for exploratory data analysis. First, we will compute descriptive statistics for all the numeric columns in the dataframe. This will give us an idea about how the data looks.

```
#generating descriptive statistic for each columns
recipe_site_traffic.describe()
```

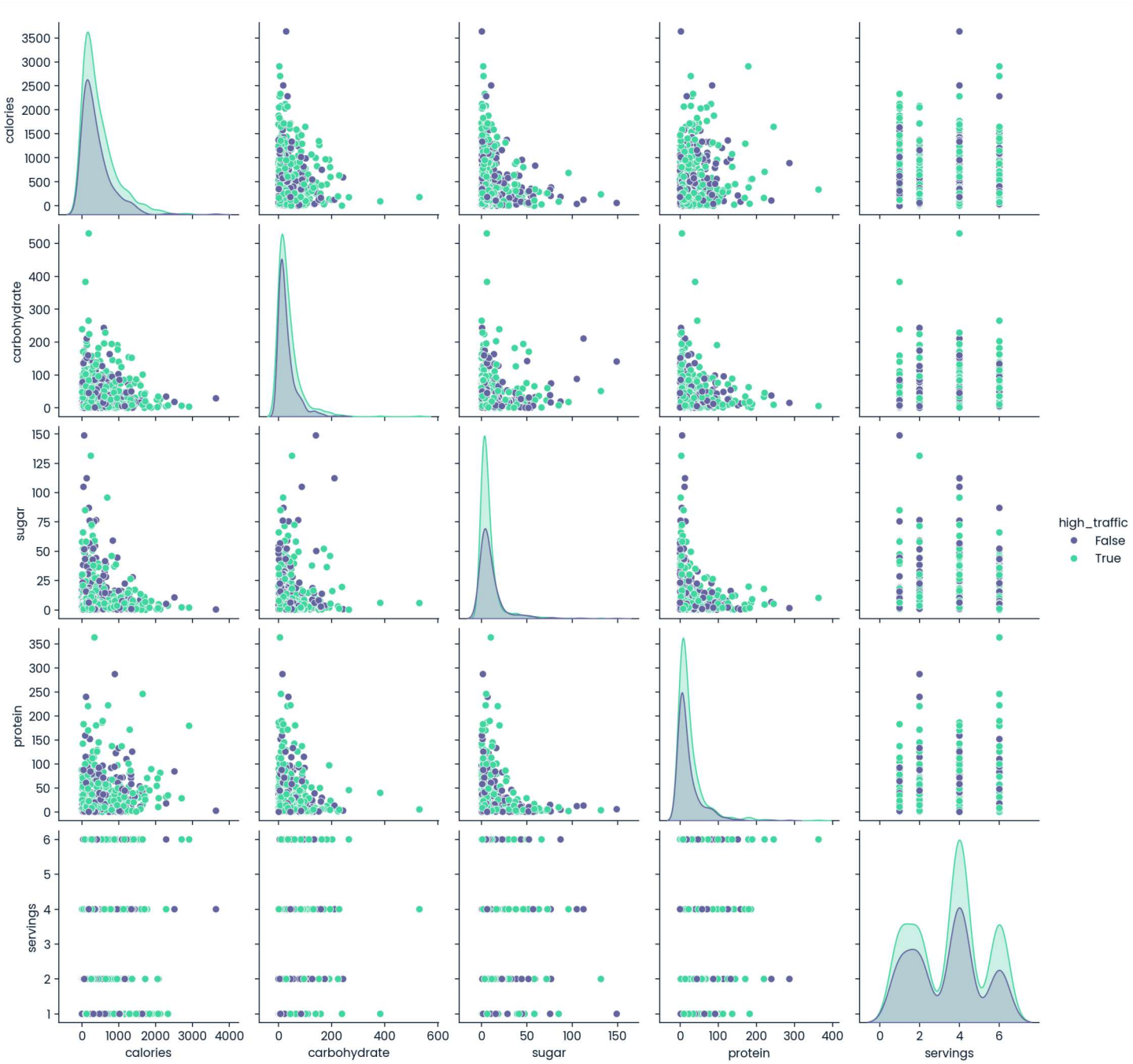
| | recipe | calories | carbohydrate |
|---|---|---|---|
| count | 895 | 895 | |
| mean | 473.6525139665 | 435.9391955307 | 3 |
| std | 272.7632831753 | 453.0209971775 | 4 |
| min | 2 | 0.14 | |
| 25% | 237.5 | 110.43 | |
| 50% | 475 | 288.55 | |
| 75% | 708.5 | 597.65 | |
| max | 947 | 3633.16 | |

8 rows

## Correlation Analysis

From the plot, we can easily infer that there is no significant correlation within these columns.

```
#filtering columns necessary for analysis.
filtered_columns = ['calories', 'carbohydrate', 'sugar', 'protein', 'category', 'servings',
'high_traffic']

sns.pairplot(recipe_site_traffic[filtered_columns], hue='high_traffic')
plt.show()
```

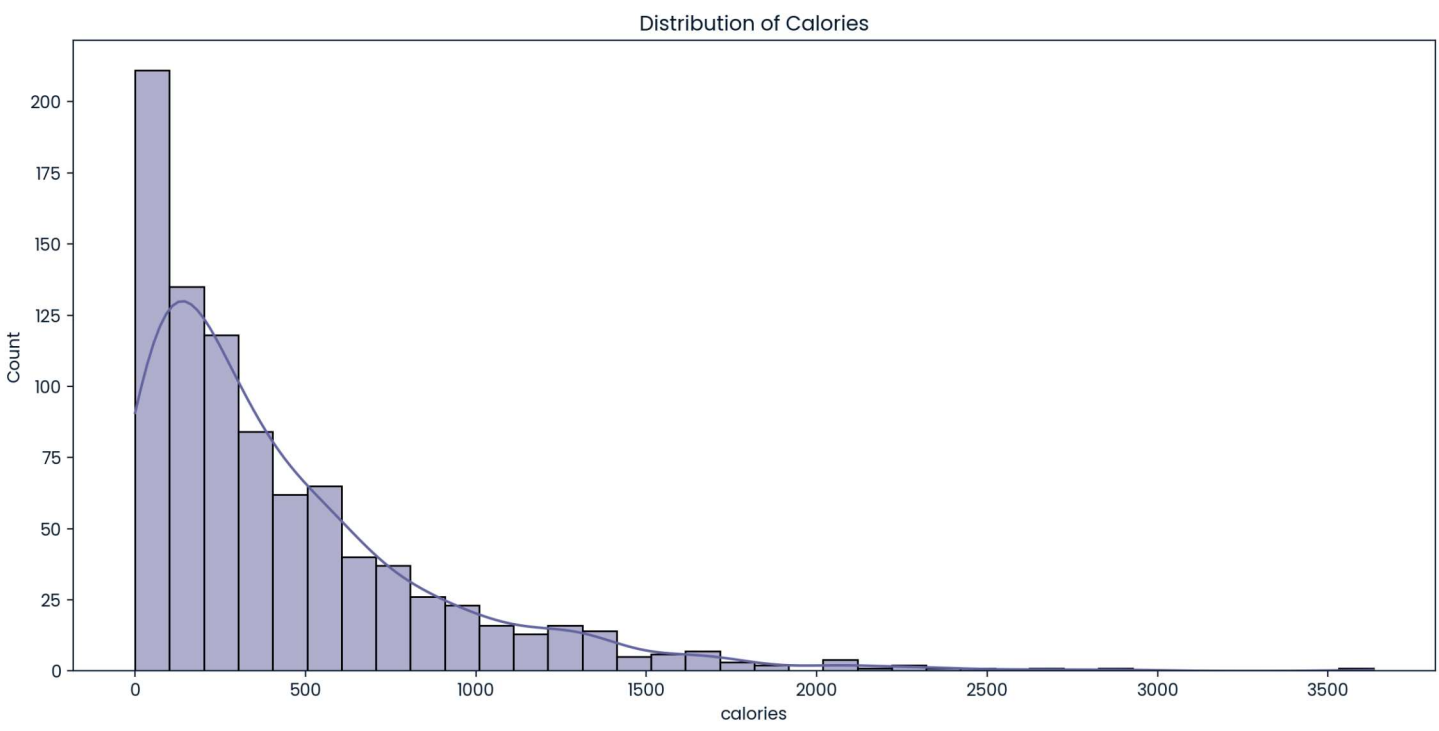


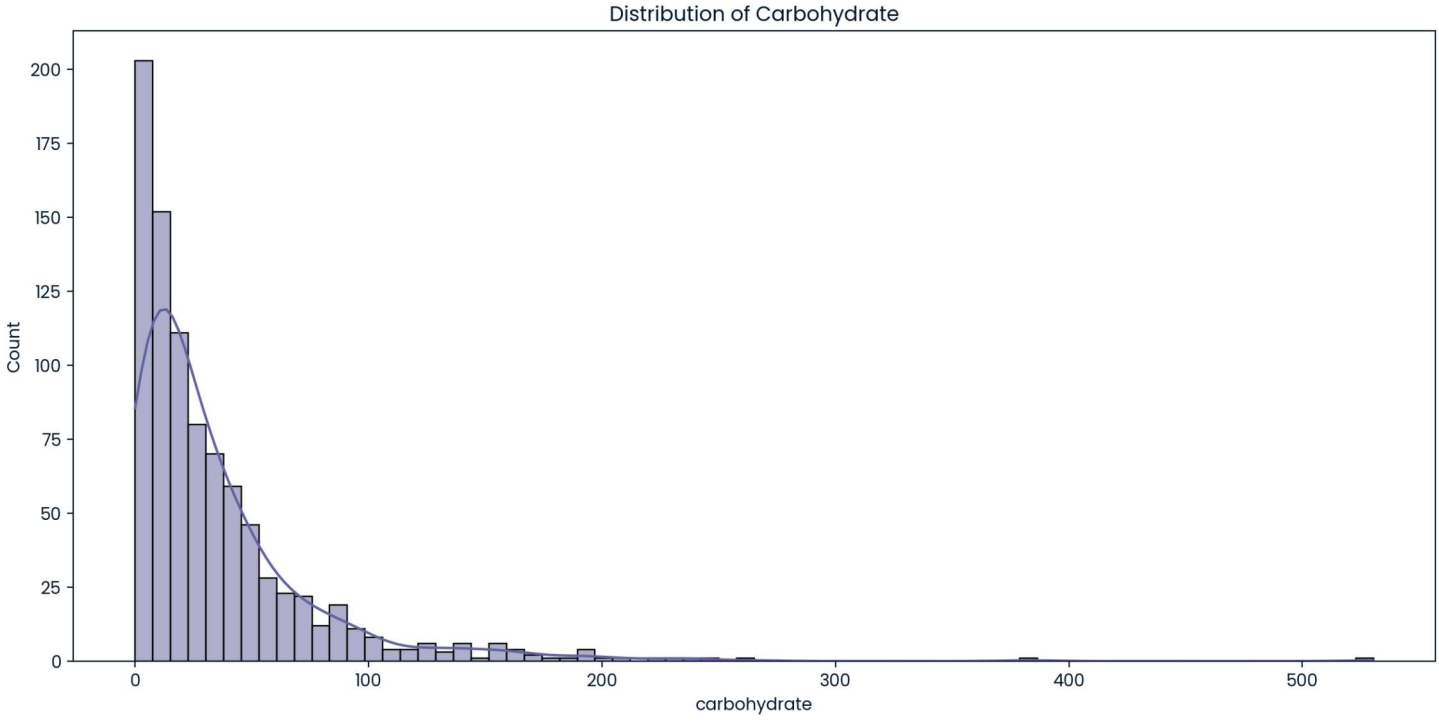

## Histograms and Kernel Density Estimation (KDE)

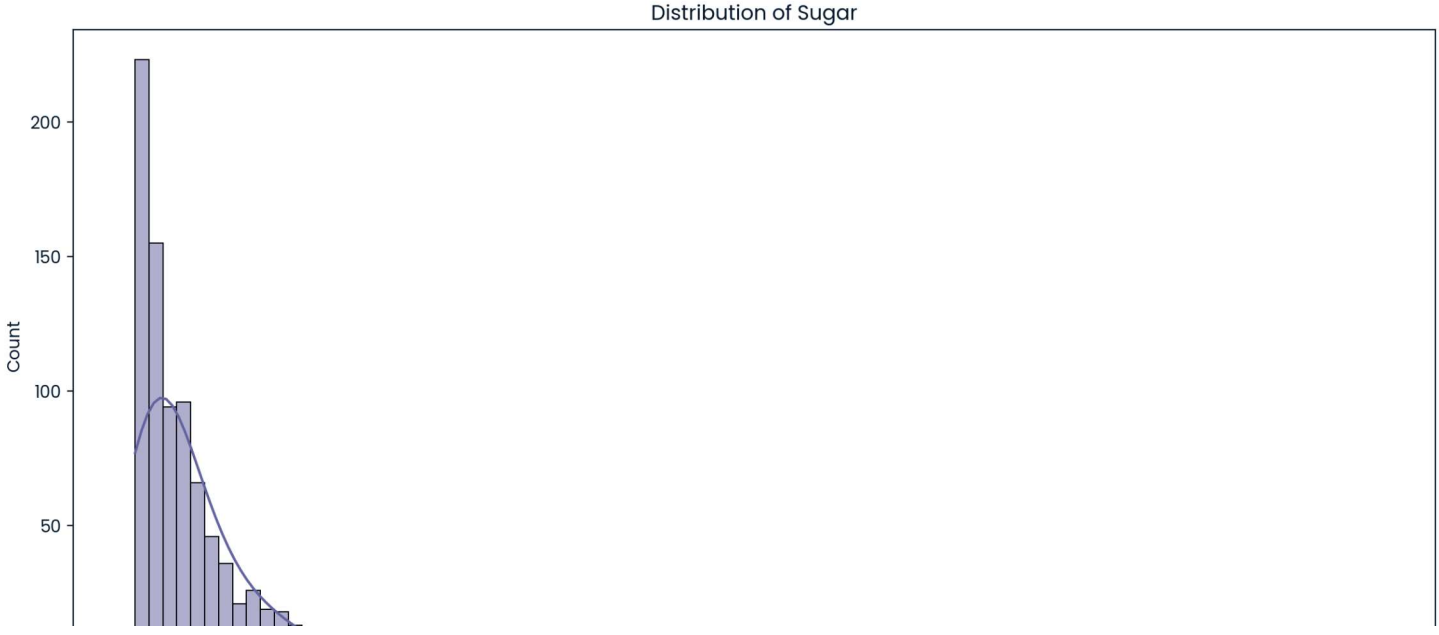
The existence of Kernel Density Estimation, KDE, in the plots provides a smooth curve showing the true distribution of data. From this, one realizes that the data is mainly composed of lower values with respect to the food calorie and nutritional contents of beverages or foods. The histograms confirm our assumption earlier about skewness. As a matter of fact, all the numerical columns are right-skewed.

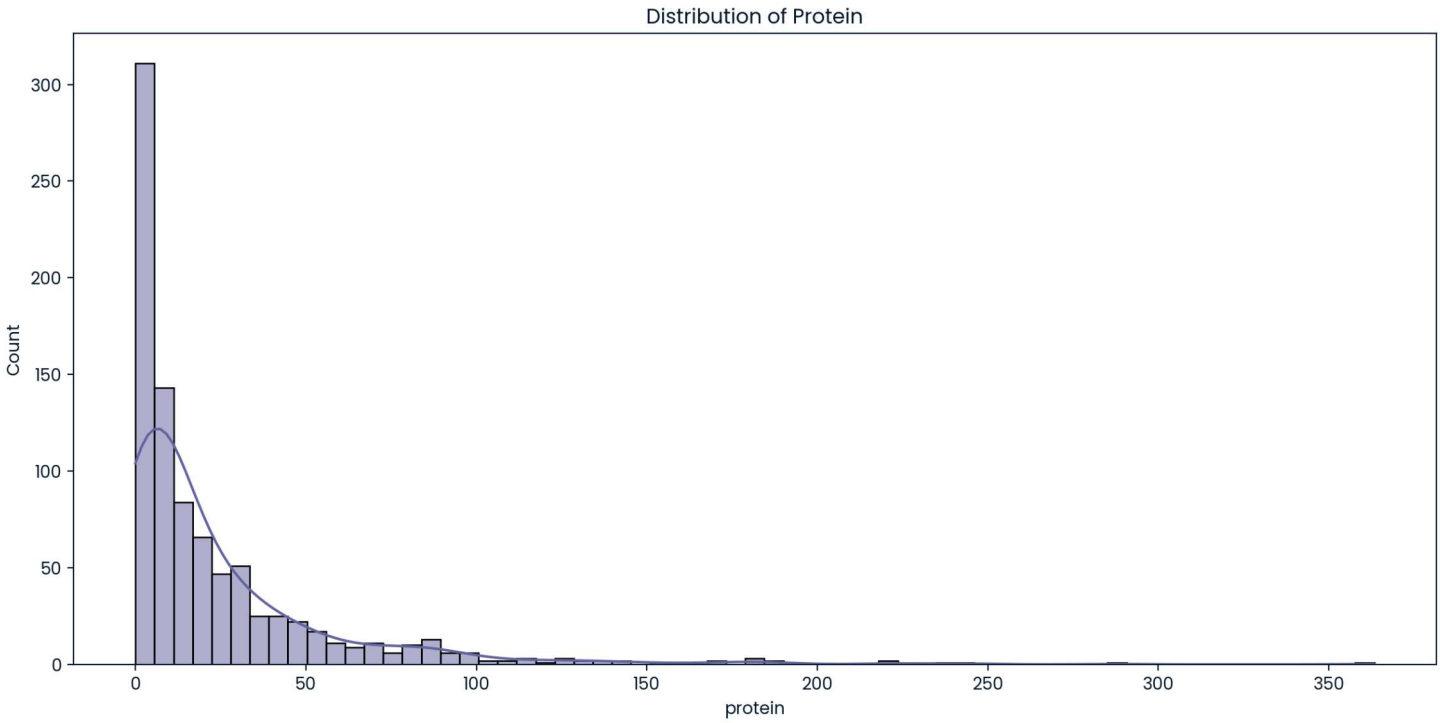
```
#specifying the numerical columns to plot
numerical_columns =  recipe_site_traffic.select_dtypes(include='float').columns

#creating subplots with the desired number of rows and columns
num_rows = len(numerical_columns)
fig, axes = plt.subplots(num_rows, 1, figsize=(12, 6 * num_rows))

#iterating over the numerical columns and plot the distributions
for i, column in enumerate(numerical_columns):
```

```
    ax = axes[i] if num_rows > 1 else axes
    sns.histplot(data=recipe_site_traffic, x=column, kde=True, ax=ax)
    ax.set_title(f"Distribution of {column.capitalize()}")

#adjust the spacing between subplots
plt.tight_layout()
plt.show()
```


Distribution of Calories
Count
calories
0
25
50
75
100
125
150
175
200
0
500
1000
1500
2000
2500
3000
3500


Distribution of Carbohydrate
Count
carbohydrate
0
25
50
75
100
125
150
175
200
0
100
200
300
400
500

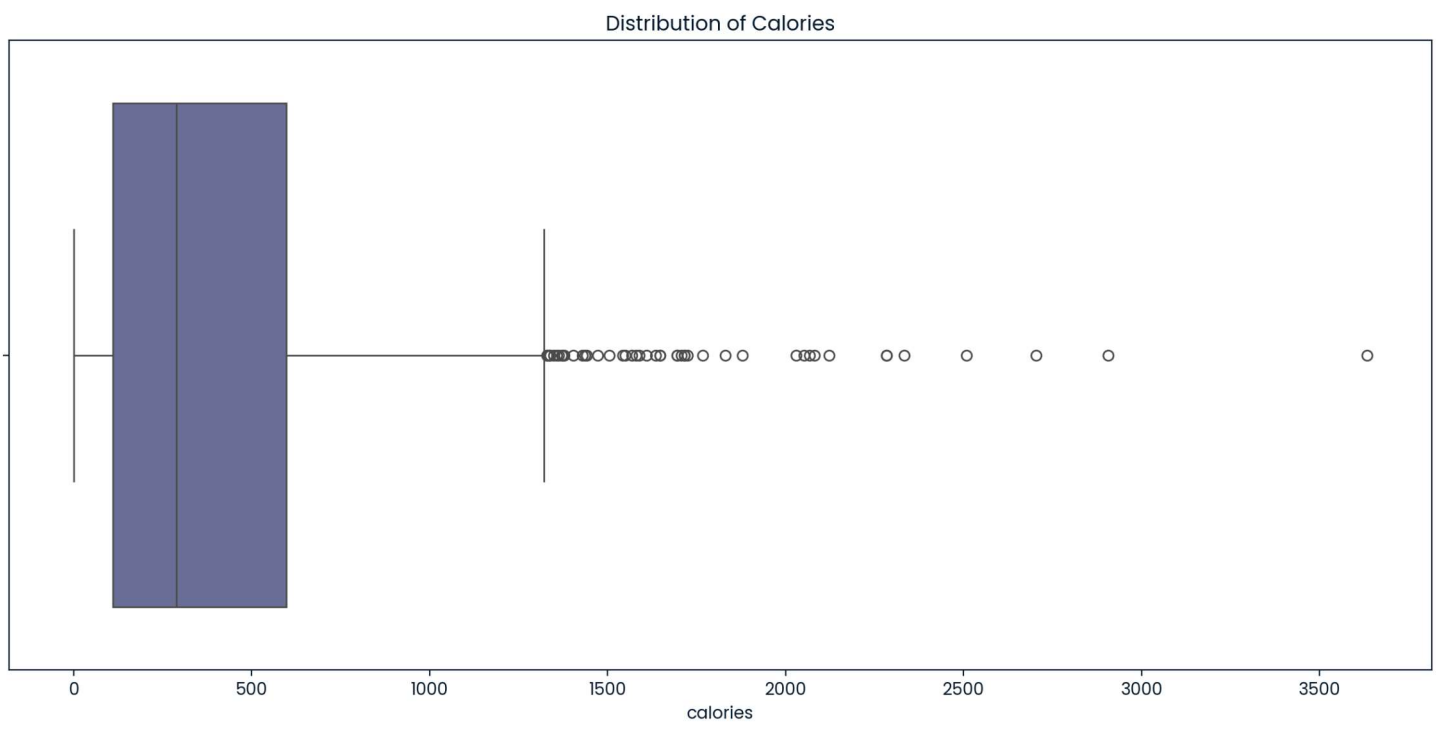

Distribution of Sugar
Count
50
100
150
200

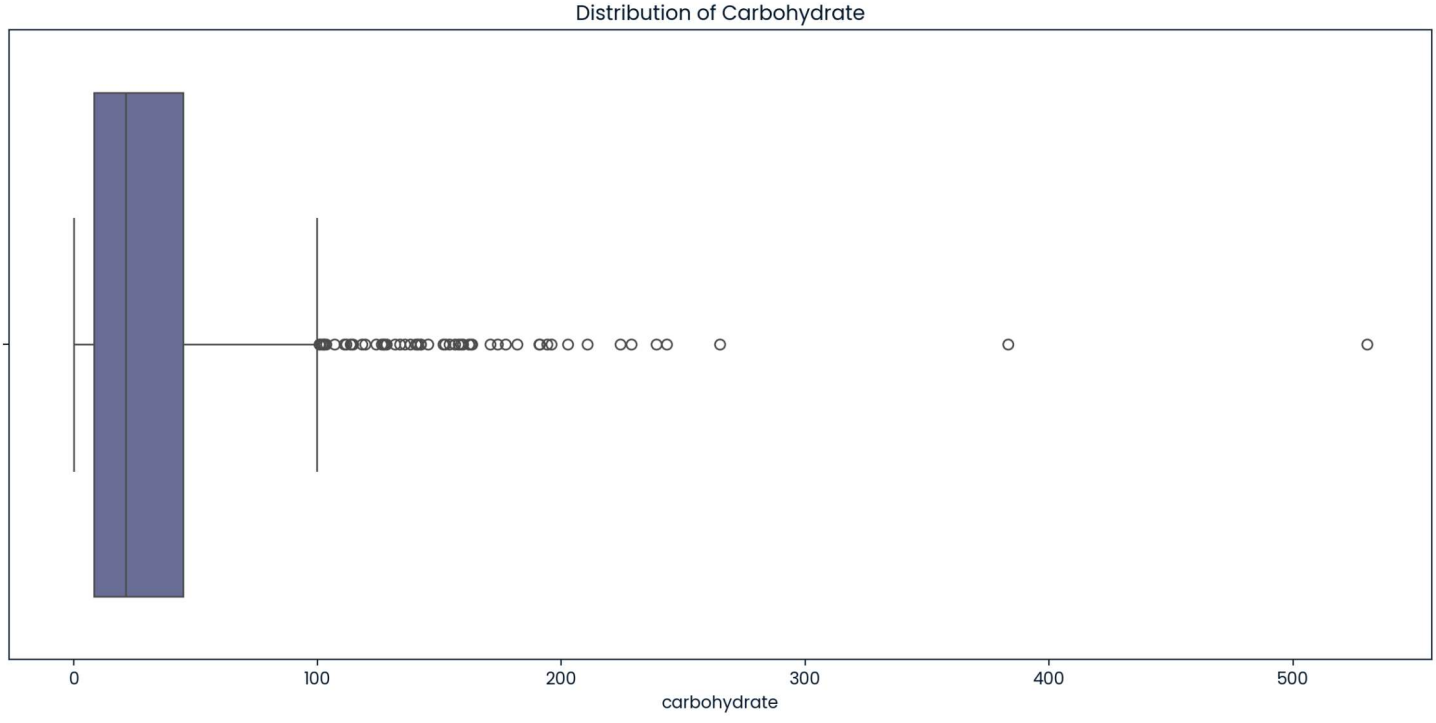



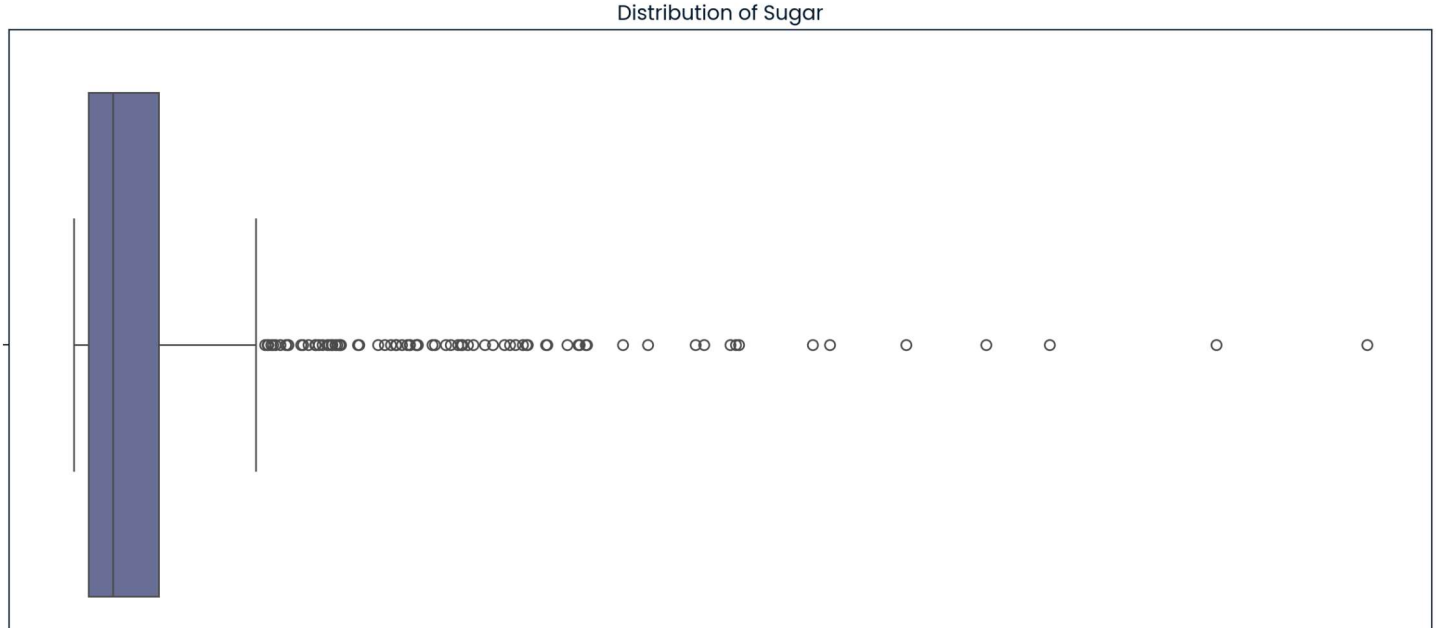

## Box Plots and Outlier Analysis

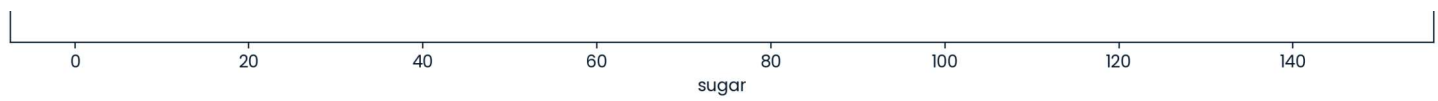
Again, above plots confirm the presence of outliers, right-skewness of those columns, and the consideration of median values while analyzing.

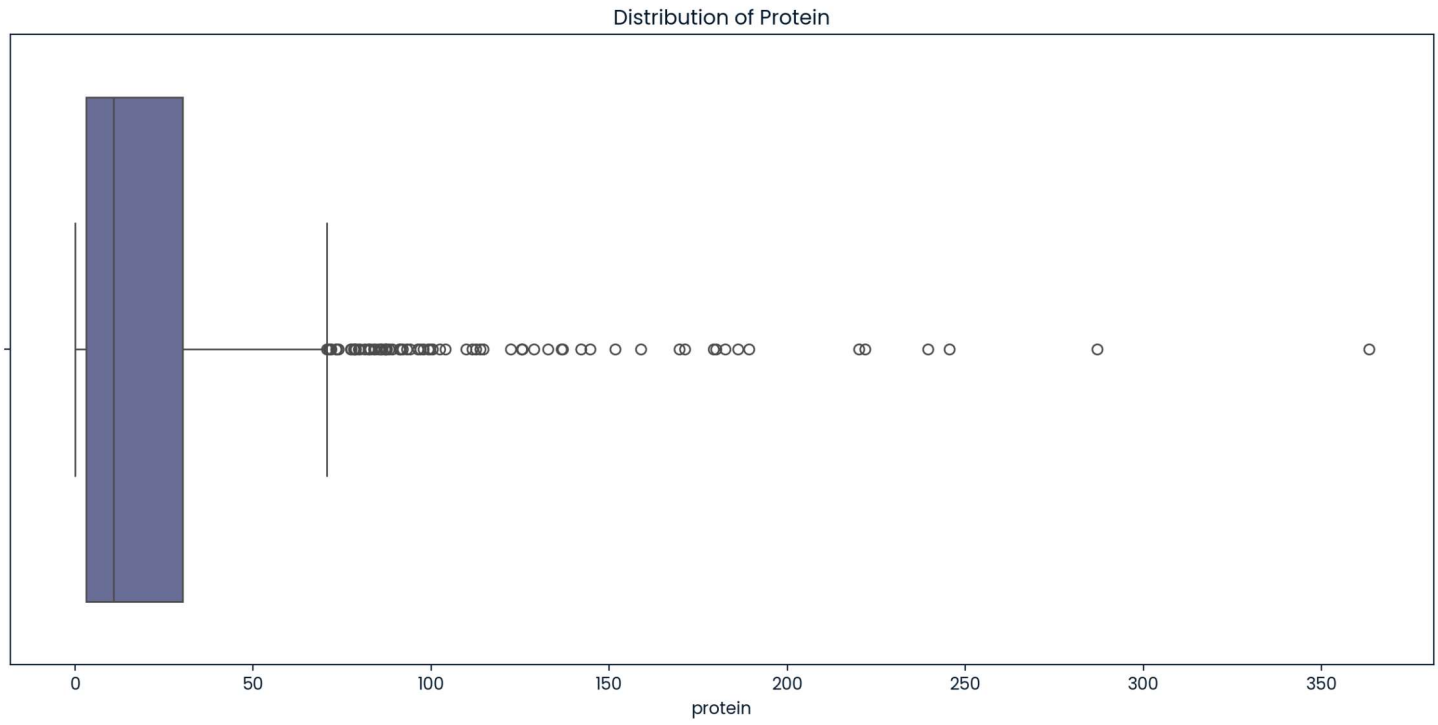
```
#specifying the numerical columns to plot
numerical_columns =  recipe_site_traffic.select_dtypes(include='float').columns

#creating subplots with the desired number of rows and columns
num_rows = len(numerical_columns)
fig, axes = plt.subplots(num_rows, 1, figsize=(12, 6 * num_rows))

#iterating over the numerical columns and plot the distributions
for i, column in enumerate(numerical_columns):
    ax = axes[i] if num_rows > 1 else axes
    sns.boxplot(data=recipe_site_traffic, x=column, ax=ax)
    ax.set_title(f"Distribution of {column.capitalize()}")

# Adjust the spacing between subplots
plt.tight_layout()
plt.show()
```


Distribution of Calories
0
500
1000
1500
2000
2500
3000
3500
calories


Distribution of Carbohydrate
0
100
200
300
400
500
carbohydrate


Distribution of Sugar



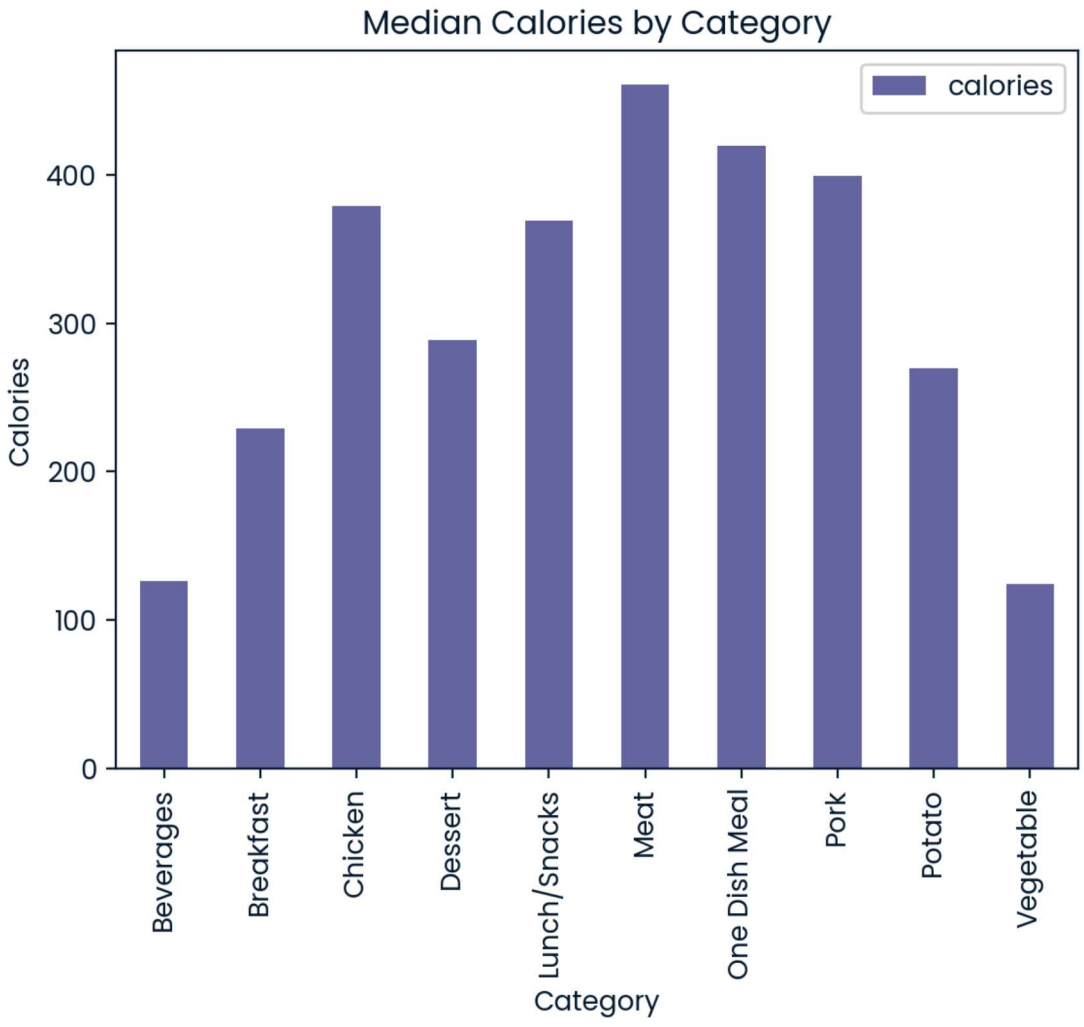

## Median Values Against Category

Next, we can analyze medians of calories and nutritional components against their category. We can do this by first grouping data by category and then aggregating with median.

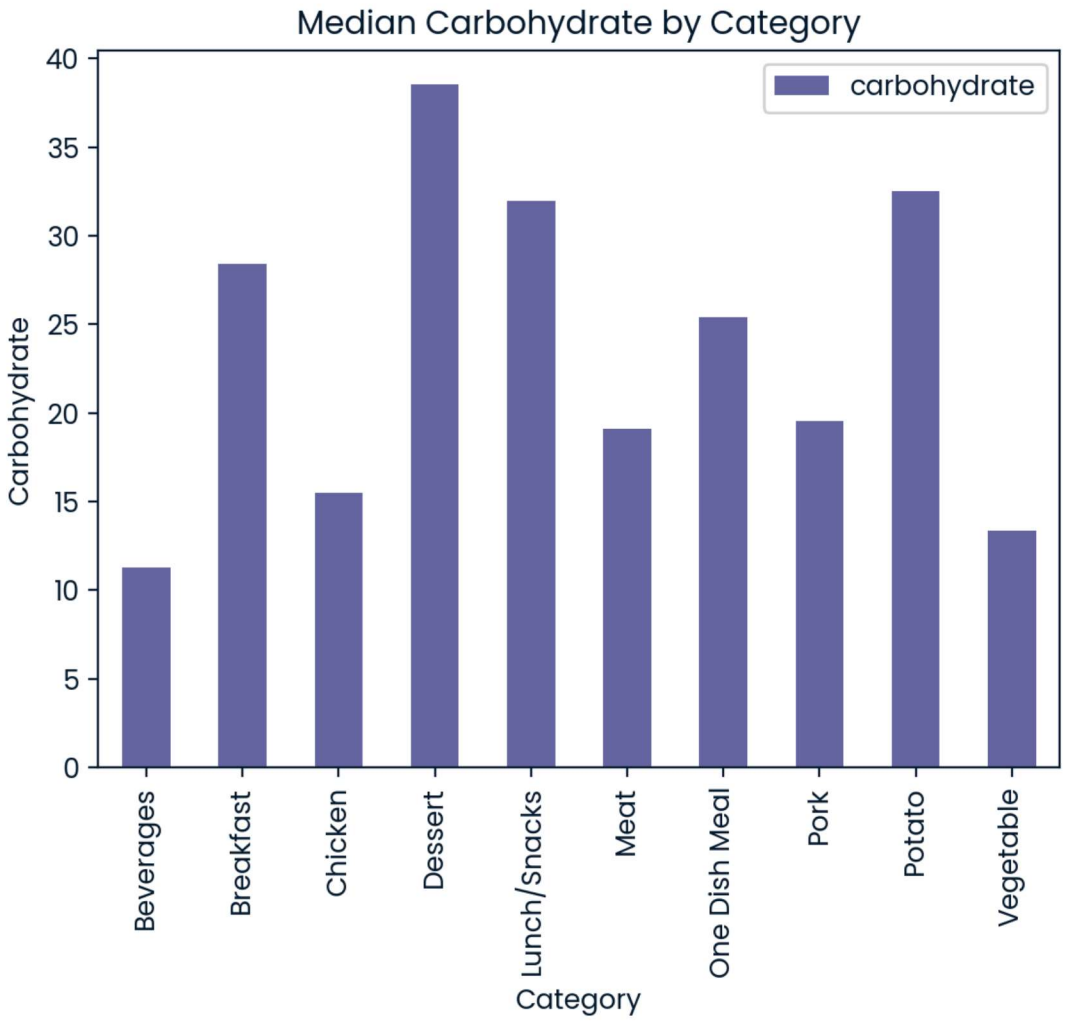
```
#group by category column and find total number of servings
category_medians = recipe_site_traffic.groupby("category")
[numerical_columns].median().reset_index()
print(category_medians)

for column in numerical_columns:
    #create a bar plot
    plt.figure()
    category_medians.plot(x="category", y=column, kind="bar")

    #set x and y axis labels
    plt.xlabel("Category")
    plt.ylabel(f"{column.capitalize()}")
    plt.title(f"Median {column.capitalize()} by Category")

#rotate x axis ticks
plt.xticks(rotation=90)
plt.show()
```

| | category | calories | carbohydrate | sugar | protein |
|---|---|---|---|---|---|
| 0 | Beverages | 126.455 | 11.250 | 8.075 | 0.410 |
| 1 | Breakfast | 229.380 | 28.420 | 4.830 | 11.685 |
| 2 | Chicken | 379.160 | 15.490 | 3.500 | 33.170 |
| 3 | Dessert | 288.550 | 38.550 | 24.650 | 4.810 |
| 4 | Lunch/Snacks | 369.505 | 31.985 | 2.750 | 12.965 |
| 5 | Meat | 460.870 | 19.115 | 3.385 | 28.530 |
| 6 | One Dish Meal | 419.280 | 25.430 | 4.040 | 28.480 |
| 7 | Pork | 399.260 | 19.560 | 5.230 | 29.820 |
| 8 | Potato | 269.460 | 32.540 | 2.640 | 5.900 |
| 9 | Vegetable | 124.305 | 13.335 | 3.700 | 4.835 |

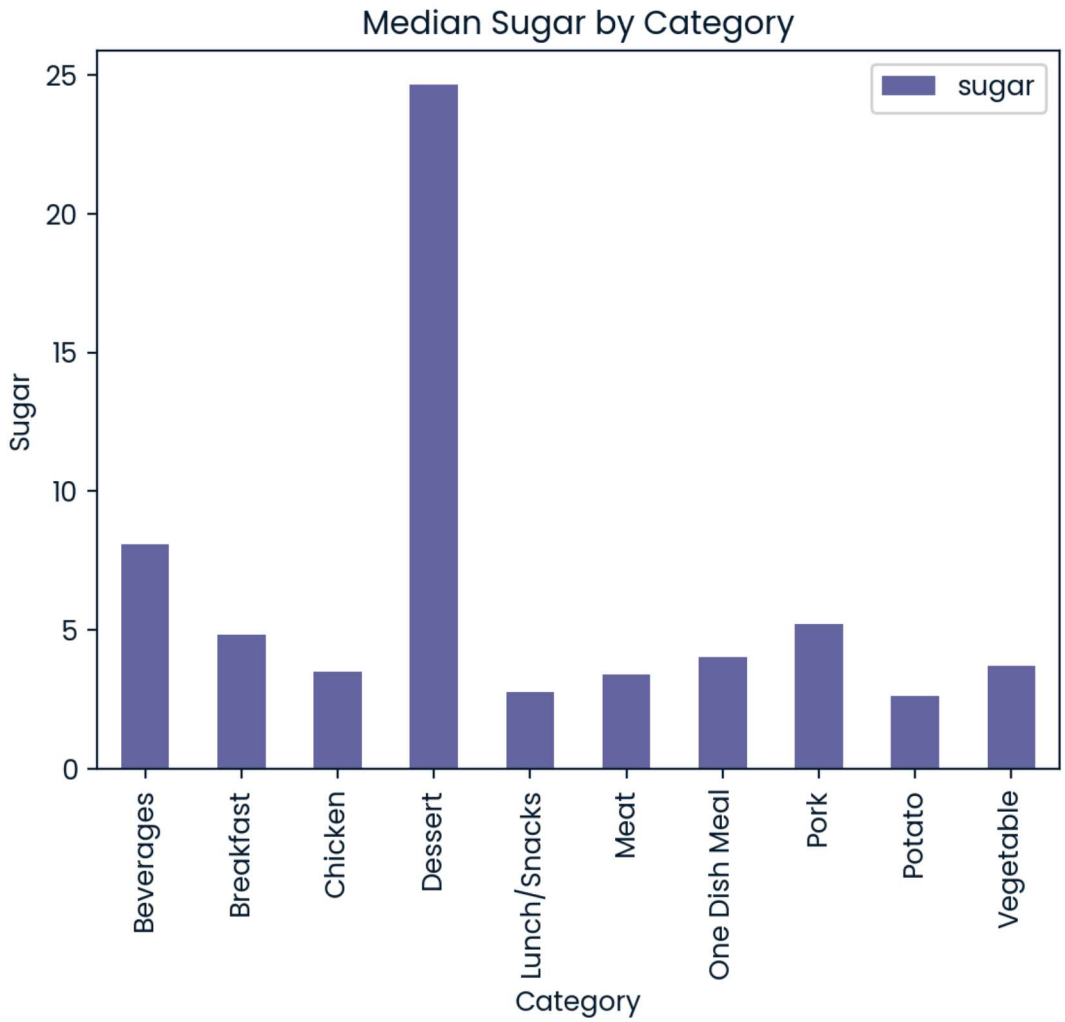
<Figure size 640x480 with 0 Axes>



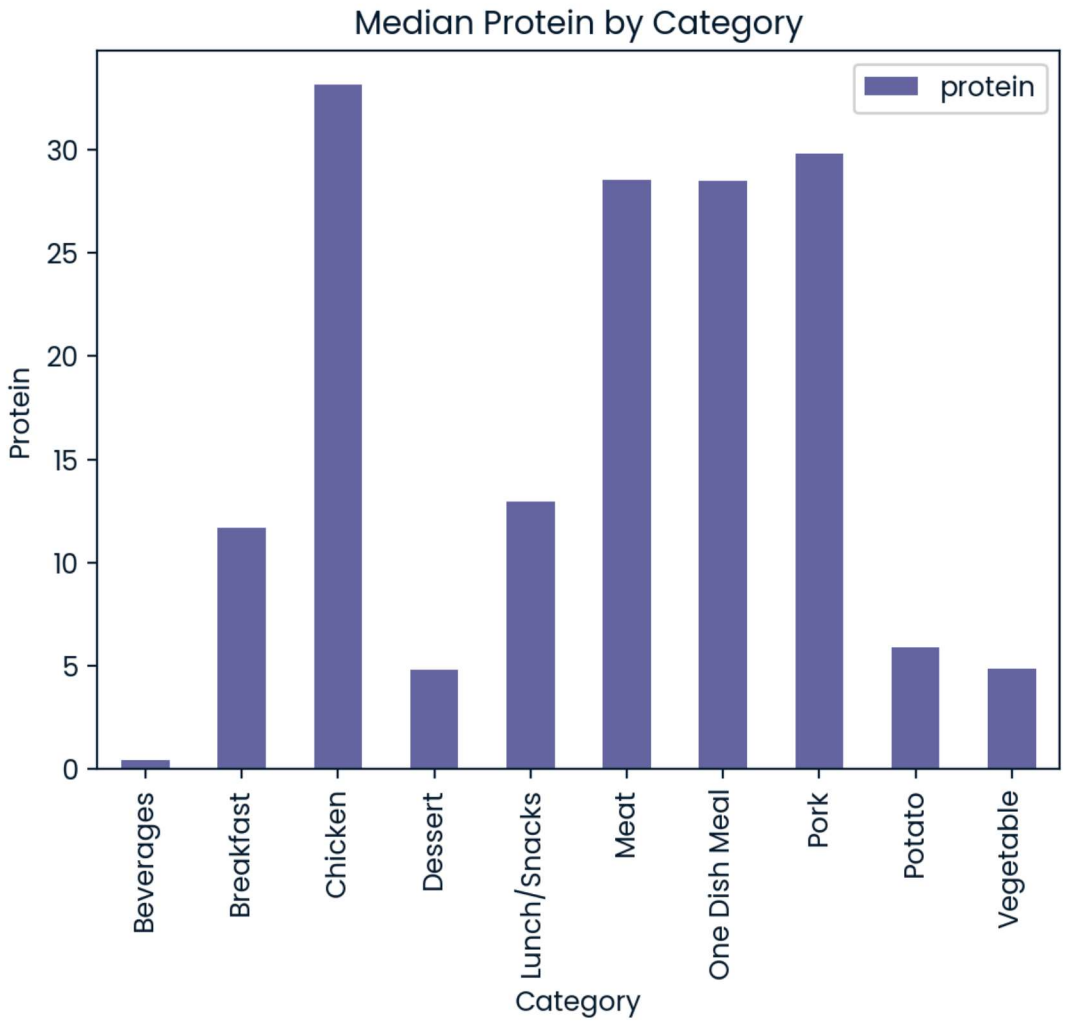

<Figure size 640x480 with 0 Axes>




```
<Figure size 640x480 with 0 Axes>
```



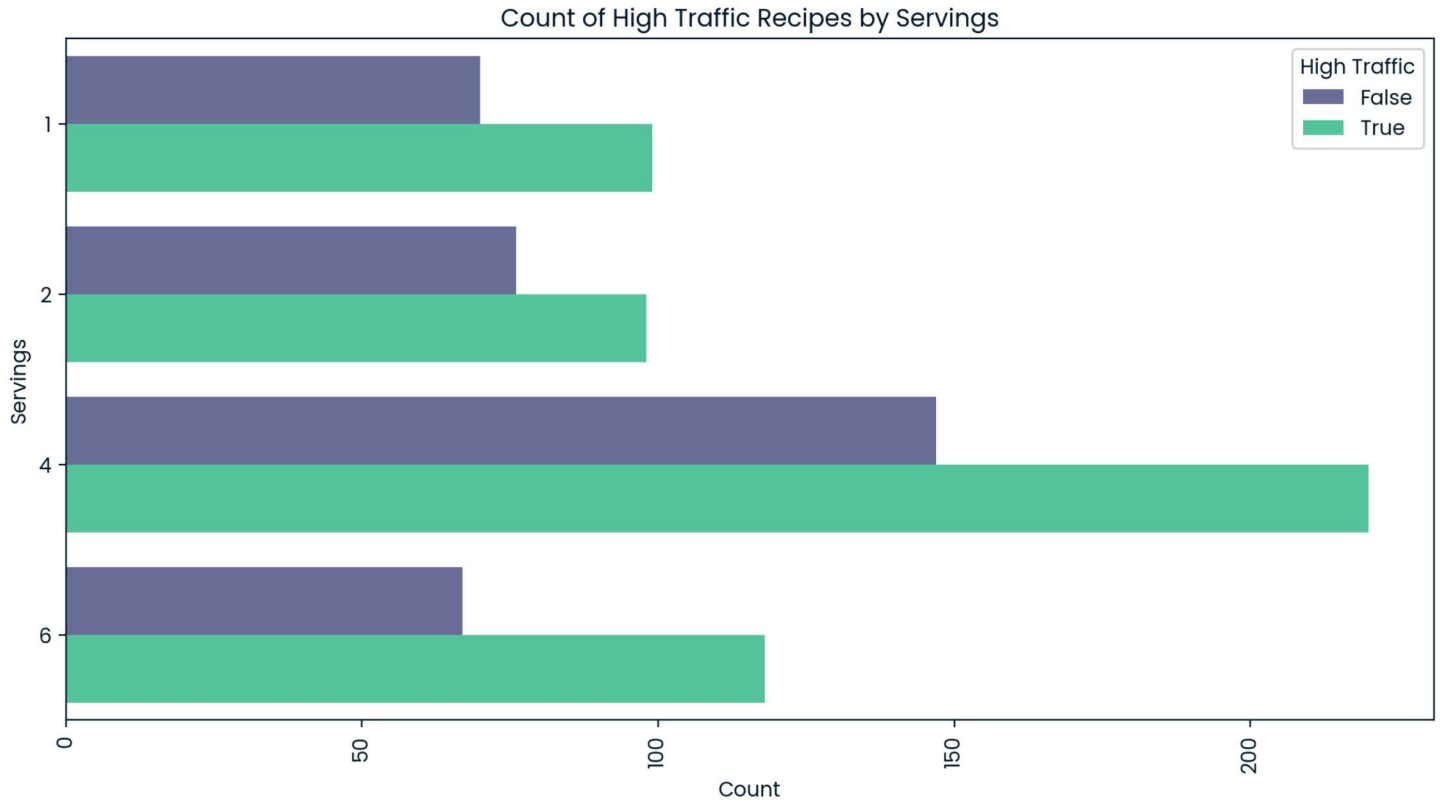

```
<Figure size 640x480 with 0 Axes>
```




## Relationship Between Servings and High Traffic Status

We note that the medians for calories and nutritional components are not symmetric as would be expected but are different by type of food and beverages. Continuing with the project now, let's first analyze the relationship of servings to high traffic status.

```
#create a count plot
plt.figure(figsize=(12, 6))
sns.countplot(y="servings", hue="high_traffic", data=recipe_site_traffic)

#set x and y axis labels
plt.xlabel("Count")
plt.ylabel("Servings")

#set title
plt.title("Count of High Traffic Recipes by Servings")

#set x axis ticks
plt.xticks(rotation=90)

#set legend title
plt.legend(title="High Traffic")
plt.show()
```




```
#group by high_trafic to show how total number of survings is distributed
high_traffic_servings = pd.pivot_table(recipe_site_traffic, index=["servings"], columns=
["high_traffic"], values="recipe", aggfunc='count')
high_traffic_servings_normalized = high_traffic_servings.div(high_traffic_servings.sum(axis=1),
axis=0)

print(high_traffic_servings_normalized)
```

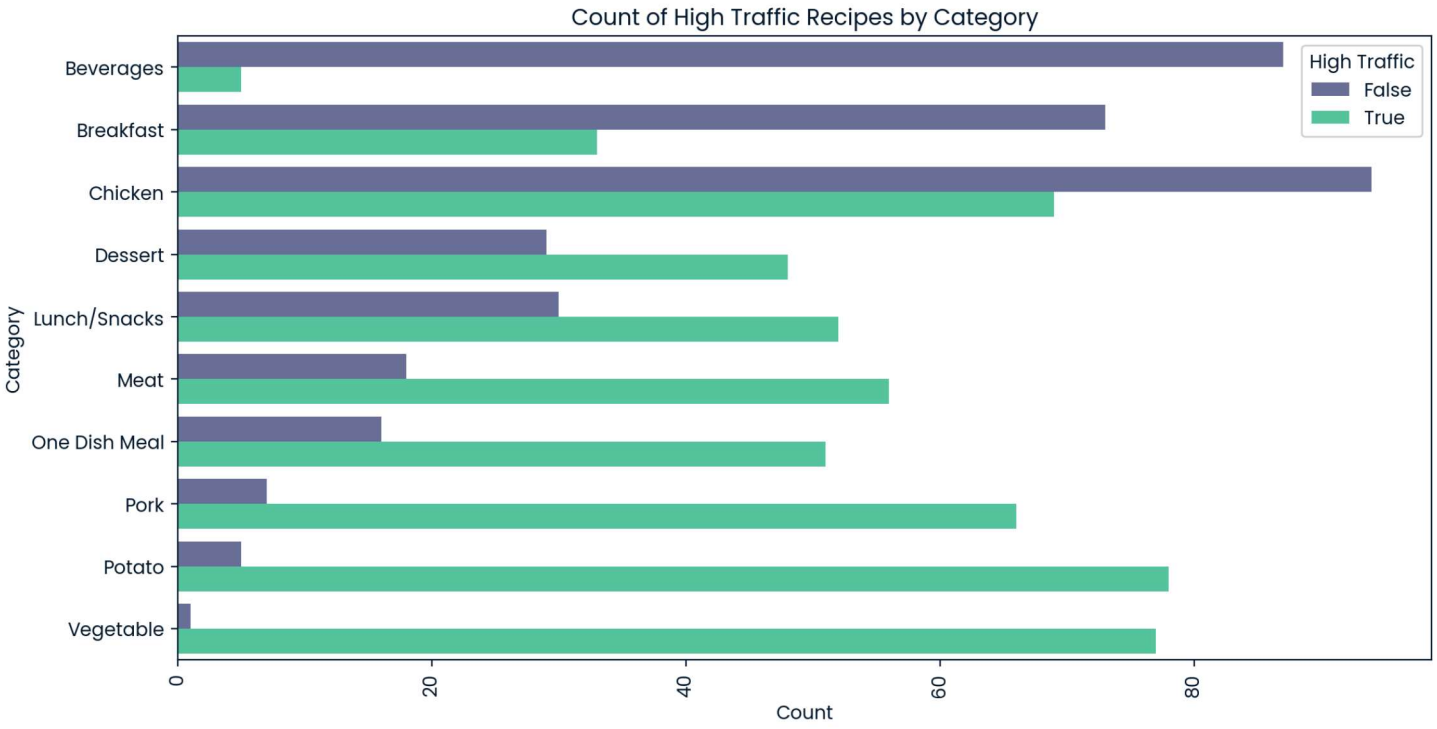
```
high_traffic     False     True
servings
1             0.414201  0.585799
2             0.436782  0.563218
4             0.400545  0.599455
6             0.362162  0.637838
```

## Relationship Between Categories and High Traffic Status

The line plot and the normalized values both suggest that recipes with 6 servings have a higher rate of high traffic. We continue now to further investigate the relationship between categories and high traffic status.

```
plt.figure(figsize=(12, 6))
sns.countplot(y="category", hue="high_traffic", data=recipe_site_traffic)

#set x and y axis labels
plt.xlabel("Count")
plt.ylabel("Category")

#set title
plt.title("Count of High Traffic Recipes by Category")
```

```
#set x axis ticks
plt.xticks(rotation=90)

#set legend title
plt.legend(title="High Traffic")
plt.show()
```




```
#create pivot table for aggregation
high_traffic_category = recipe_site_traffic.pivot_table(index="category",
columns="high_traffic", values="servings", aggfunc="sum")
normalized_table = high_traffic_category.div(high_traffic_category.sum(axis=1), axis=0)

#show the result
print(normalized_table)
```

```
high_traffic      False     True
category
Beverages      0.939873  0.060127
Breakfast      0.685294  0.314706
Chicken        0.581056  0.418944
Dessert        0.372263  0.627737
Lunch/Snacks   0.385965  0.614035
Meat           0.270270  0.729730
One Dish Meal  0.198068  0.801932
Pork           0.087452  0.912548
Potato         0.041139  0.958861
Vegetable      0.013986  0.986014
```

## Model Selection for Machine Learning

To deal with this, we will utilize supervised machine learning's binary classification algorithms. We can use any appropriate model to predict the high traffic status. Some popular choices could be logistic regression, decision tree, random forest, or support vector machines. In this work, we consider a logistic regression model as our baseline and the rest for comparisons.

## Outlier Handling

Before proceeding with Logistic Regression, let's address the issue of outliers in our dataset. To do so, we will calculate the IQR for each column and then calculate the upper and lower bounds to handle the outliers in a better way.

```
#function to calculate upper and lower boundaries of columns
def calculate_outlier_limits(data, factor=1.5):
    # Find 25% and 75% percentiles
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)

    #calculate IQR
    iqr = q3 - q1

    #calculate lower and upper boundaries
    lower_limit = q1 - (factor * iqr)
    upper_limit = q3 + (factor * iqr)
    return lower_limit, upper_limit
```

```
#select numerical columns
numerical_columns = ['calories', 'carbohydrate', 'sugar', 'protein']

#calculate outlier limits for each numerical column
outlier_limits = {}
for column in numerical_columns:
    lower_limit, upper_limit = calculate_outlier_limits(recipe_site_traffic[column])
    outlier_limits[column] = (lower_limit, upper_limit)

#check the calculated outlier limits for each column
for column, limits in outlier_limits.items():
    print(f"Column: {column}")
    print(f"Lower Limit: {limits[0]}")
    print(f"Upper Limit: {limits[1]}")
    print("--------------------------------")
```

```
Column: calories
Lower Limit: -620.3999999999999
Upper Limit: 1328.48
--------------------------------
Column: carbohydrate
Lower Limit: -46.510000000000005
Upper Limit: 99.85000000000001
--------------------------------
Column: sugar
Lower Limit: -10.475000000000003
Upper Limit: 21.965000000000003
--------------------------------
Column: protein
Lower Limit: -37.31250000000001
Upper Limit: 70.70750000000001
--------------------------------
```

## Data Transformation

As expected, there are no issues regarding low values; however, some high values need to be taken care of. There are multiple ways to handle this issue. Accordingly, upon analysis, a full exclusion of the outlies have caused a huge information loss. Thus it is not advisable. Capping outliers or winsorizing also does not give the desired distribution. Amongst many transformation techniques that were tried out as Logarithmic Transformation, Square Root Transformation, Yeo-Johnson Transformation, and Box-Cox Transformation-the most appropriate was Yeo-Johnson Transformation.

```
#specifying the numerical columns to plot
numerical_columns =  recipe_site_traffic.select_dtypes(include='float').columns

#filter the dataset based on the outlier limits
transformed_data = recipe_site_traffic.copy()
for column, limits in outlier_limits.items():
    lower_limit, upper_limit = limits

    #transformed_data = transformed_data[transformed_data[column] <= upper_limit]           #
Removing Outliers
    #transformed_data[column] = transformed_data[column].clip(upper=upper_limit)            #
Capping Outliers
    #transformed_data[column] = winsorize(transformed_data[column], limits=[0.05, 0.05])    #
Winsorization

    #transformed_data[column] = np.log1p(transformed_data[column])                          #
Logarithmic Transformation
    #transformed_data[column] = np.sqrt(transformed_data[column])                           #
Square Root Transformation

    #transformed_data.loc[transformed_data[column] == 0, column] = 0.00001
    #transformed_data[column] = boxcox(transformed_data[column])[0]                         #
Box-Cox Transformation
```

```
    transformed_data[column] = yeojohnson(transformed_data[column])[0]                    #
Yeo-Johnson Transformation

# Check the filtered dataset
```

| | recipe | calories | carbohydrate | sugar |
|---|---|---|---|---|
| 0 | 2 | 5.7427126968 | 4.4090234258 | 0.48 |
| 1 | 3 | 17.4377545219 | 4.5506122215 | 1.22 |
| 2 | 4 | 8.4164763598 | 4.0911202842 | 2.58 |
| 3 | 5 | 5.1334122659 | 1.1016611946 | 0.5 |
| 4 | 6 | 16.0244902729 | 1.607527938 | 0.88 |

5 rows

```
plt.figure(figsize=(16, 4))

#iterating over each column and create a bar plot
for i, column in enumerate(numerical_columns):
    plt.subplot(1, len(numerical_columns), i+1)
    sns.histplot(data=transformed_data, x=column, kde=True)
    plt.title(f"Distribution of {column.capitalize()}")

#adjust the spacing between subplots if needed
plt.tight_layout()
plt.show()
```