Tasty Bytes

August 18, 2024

1 'Tasty Bytes' (EST. 2020) and Featured Recipe's Influence on Site Traffic

2 Data Validation

The data set consisted of 947 rows and 8 columns. Values with unnecessary text were cleaned for features: Category/Servings. 52 rows had missing data for the same 4 features: Calories/Carbohydrate/Sugar/Protein; after insights garnered, records dropped resulting in 895 complete rows for the whole dataframe.

- Recipe: 895 integers of Unique IDs after 52 records dropped due to null values in other features.
- Calories/Carbohydrate/Sugar/Protein: 895 floats, 52 records with null values dropped.
- Category: 10 unique categories; 98 records labelled 'Chicken Breast' converted to 'Chicken'. No missing values.
- Servings: 4 unique integers [1,2,4,6]; 3 records had additional text dropped to allow conversion of dtype to integer. No missing values.
- High_traffic: Target variable; feature changed into binary integers for easier processing. After nulls dropped, 535 successes of 895 recipes.

[1]:
```python
# Start coding here...
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, roc_auc_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.model_selection import GridSearchCV
plt.style.use('ggplot')
```

[2]:
```python
df = pd.read_csv('recipe_site_traffic_2212.csv', )
print(df.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
None
```

[3]:

```
df.isna().sum()
```

[3]:

```
recipe            0
calories         52
carbohydrate     52
sugar            52
protein          52
category          0
servings          0
high_traffic    373
dtype: int64
```

[4]:

```
null_in_calories = df[df['calories'].isna()]
null_in_calories.info()
# whichever records are null for calories, are also null for carbs/sugars/
 ↪proteins
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 52 entries, 0 to 943
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   recipe        52 non-null     int64
 1   calories      0 non-null      float64
 2   carbohydrate  0 non-null      float64
 3   sugar         0 non-null      float64
 4   protein       0 non-null      float64
 5   category      52 non-null     object
 6   servings      52 non-null     object
 7   high_traffic  39 non-null     object
dtypes: float64(4), int64(1), object(3)
memory usage: 3.7+ KB
```

[5]:
```
df['category'].unique()
# 'Chicken Breast' shouldn't be there
```

[5]:
```
array(['Pork', 'Potato', 'Breakfast', 'Beverages', 'One Dish Meal',
       'Chicken Breast', 'Lunch/Snacks', 'Chicken', 'Vegetable', 'Meat',
       'Dessert'], dtype=object)
```

[6]:
```
df['servings'].unique()
# need to remove str 'as a snack'
```

[6]:
```
array(['6', '4', '1', '2', '4 as a snack', '6 as a snack'], dtype=object)
```

[7]:
```
df['high_traffic'].unique()
```

[7]:
```
array(['High', nan], dtype=object)
```

[8]:
```
df.describe()
#no negative values for the numeric features
```

[8]:

	recipe	calories	carbohydrate	sugar	protein
count	947.000000	895.000000	895.000000	895.000000	895.000000
mean	474.000000	435.939196	35.069676	9.046547	24.149296
std	273.519652	453.020997	43.949032	14.679176	36.369739
min	1.000000	0.140000	0.030000	0.010000	0.000000
25%	237.500000	110.430000	8.375000	1.690000	3.195000
50%	474.000000	288.550000	21.480000	4.550000	10.800000
75%	710.500000	597.650000	44.965000	9.800000	30.200000
max	947.000000	3633.160000	530.420000	148.750000	363.360000

[9]:
```
# Data Validation and Cleaning
cdf = df.dropna(subset=['calories', 'carbohydrate', 'sugar', 'protein']).copy()
cdf.loc[:, 'category'] = cdf['category'].str.replace('Chicken Breast',␣
 ↪'Chicken', regex=True)
cdf.loc[:,'servings'] = cdf['servings'].str.replace('4 as a snack', '4',␣
 ↪regex=True)
cdf.loc[:,'servings'] = cdf['servings'].str.replace('6 as a snack', '6',␣
 ↪regex=True)
cdf.loc[:,'servings']= cdf['servings'].astype(int)
cdf.loc[:,'high_traffic'] = cdf['high_traffic'].notnull() #converting feature␣
 ↪to bool
cdf.loc[:,'high_traffic'] = cdf['high_traffic'].astype(int) #bool to binary␣
 ↪integer
cdf.reset_index(drop=True, inplace=True)
print(cdf.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    object
 6   servings      895 non-null    object
 7   high_traffic  895 non-null    object
dtypes: float64(4), int64(1), object(3)
memory usage: 56.1+ KB
None
```

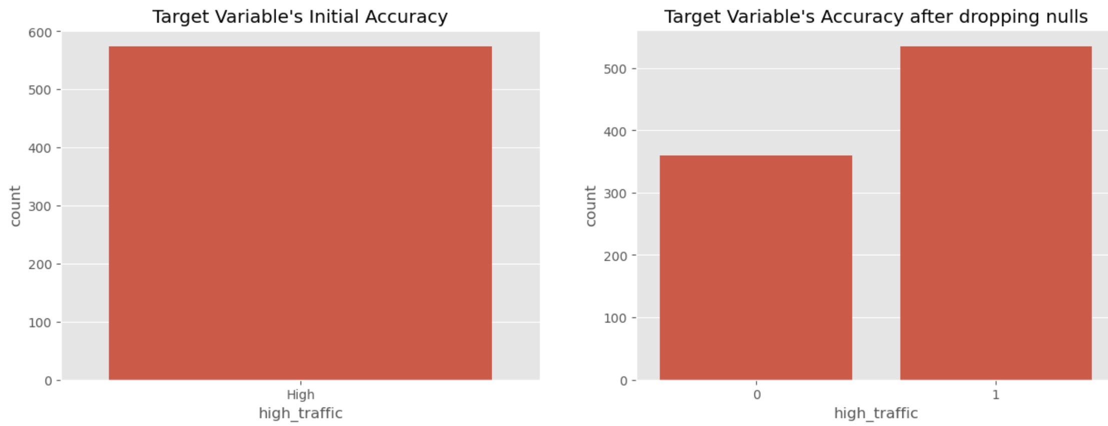
3 Exploratory Analysis

All the variables included within the dataset have had their interrelationships examined. The focus starts with the target variable and exploring the successes prior to any model implementation. After a heatmap shows the relationship between all the numeric variables and how there is little to no correlation between any of the variables. Two univariate histograms are then shown, one showing the distribution of the recipes based on calorie content and the second on protein content. Lastly, the stacked bar graph shows a good picture of the importance of food category's effect on the target variable.

3.1 Target Variable - high_traffic

From the plots shown below: - 574 recipes of the original 947 recipes pulled high traffic when featured (60.6% seen in the left plot) - After nulls dropped, 535 success from the remaining 895 recipes (count plot on the right) - Both equate to about 60% accuracy without application of any model or algorithm

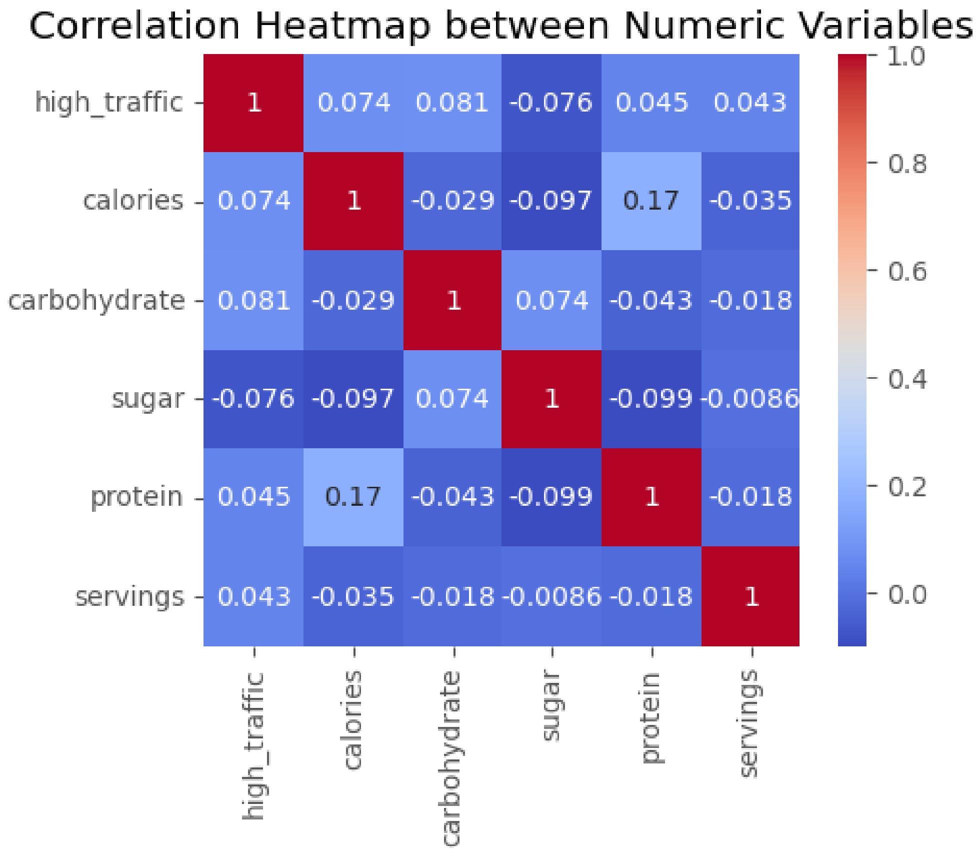
[10]:
```
# target var - Y - high_traffic
fig, axes = plt.subplots(1,2,figsize=(15,5))
sns.countplot(df,x='high_traffic',ax=axes[0]).set(title="Target Variable's␣
 ↪Initial Accuracy")
sns.countplot(cdf,x='high_traffic',ax=axes[1]).set(title="Target Variable's␣
 ↪Accuracy after dropping nulls");
```



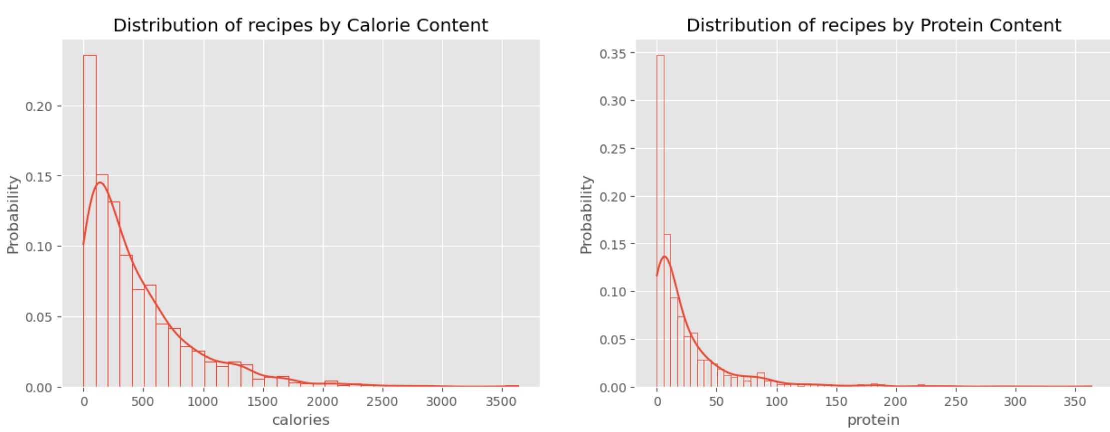
3.2 Numerical Variables - Calories, carbohydrate, sugar, protein, servings

- 39 of the 52 recipes that initially contained missing data had high_traffic when featured; this suggests it is not imperative to have those features labeled for a successfull recipe, maybe even the opposite for the case of comfort foods
- Once the dataset was cleaned from the nulls, it shows all the numeric features superficially have no correlation and especially not with the target variable: high_traffic
- The strongest correlation seen on the heatmap below, although still very weak, is between calories and protein. This is to be expected as it is well-known that protein is a calorie-rich macronutrient.

[11]:
```
#correlation heatmap
numf = ['high_traffic','calories', 'carbohydrate', 'sugar',␣
↪'protein','servings']
cdf[numf] = cdf[numf].apply(pd.to_numeric, errors='coerce')
plt.figure(figsize=(5, 4))
heatmap = sns.heatmap(cdf[numf].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap between Numeric Variables")
plt.show()
```



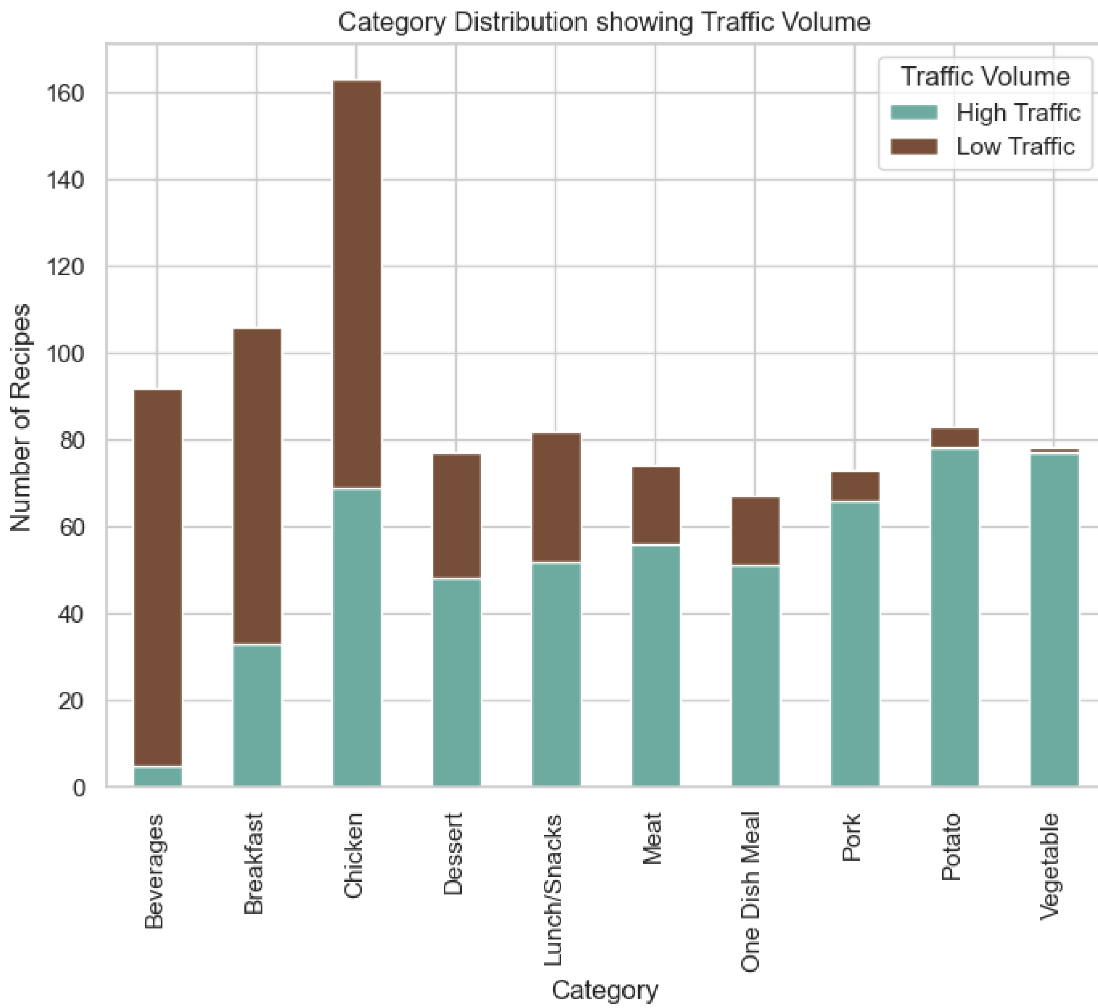
[12]:
```python
# Calorie and Protein Histograms
fig, axes = plt.subplots(1,2,figsize=(15,5))
sns.
 ↪histplot(cdf,x='calories',stat='probability',fill=False,kde=True,ax=axes[0]).
 ↪set(title="Distribution of recipes by Calorie Content")
sns.histplot(cdf,x='protein',stat='probability',fill=False,kde=True,ax=axes[1]).
 ↪set(title="Distribution of recipes by Protein Content");
```



3.3 Categorical Variable - category

- Of the 84 recipes initially submitted for Pork, 11 had null info yet were still part of the 77 pork recipes considered high_traffic
- It appears the category may be the most influential feature affecting a recipe's popularity
- The stacked bar graph below shows how Pork/Potato/Vegetable recipes seem to always be a hit
- (although the feature 'servings' could be treated as a categorical variable, it was grouped with other numeric variables for better processing)

[13]:
```
# bivariate stacked bar graph showing bars of category stacked by high_traffic
cdf['high_traffic'] = cdf['high_traffic'].map({0: 'Low Traffic', 1: 'High␣
↪Traffic'})
plot_data = cdf.groupby(['category', 'high_traffic']).size().
↪unstack(fill_value=0)
sns.set(style="whitegrid")
plt.figure(figsize=(8,6))
plot_data.plot(kind='bar', stacked=True, color=['#6daa9f', '#774f38'], ax=plt.
↪gca())
plt.title('Category Distribution showing Traffic Volume')
plt.xlabel('Category')
plt.ylabel('Number of Recipes')
plt.legend(title='Traffic Volume', loc='upper right')
plt.show()
```



4 Model Development

Since a binary outcome is sought after from the predictions and there is data regarding the target variable available, this is a supervised classification problem in machine learning. A logisitic regression model will first be fit to the data. The comparison model will be done via Random Forest Classification.

To start modeling, calories/carbohydrate/sugar/protein/category/servings will be the features and high_traffic the target variable. In addition, the following was adjusted: - Used one-hot encoding for the categorical features - Numeric features (besides servings) scaled using PowerTransformer() - Data was split into train and test sets - GridSearchCV used to find the best hyperparameters

[14]:
```
onehot = OneHotEncoder()
category_encoded = onehot.fit_transform(cdf[['category']]).toarray()
category_encoded_df = pd.DataFrame(category_encoded, columns=onehot.
  ↪get_feature_names_out(['category']))
```

```
cdf = pd.concat([cdf, category_encoded_df], axis=1)
cdf.drop('category', axis=1, inplace=True)
```

[15]:
```
feature_cols = ['calories', 'carbohydrate', 'sugar', 'protein','servings']
encoded_category_cols = list(category_encoded_df.columns)
feature_cols.extend(encoded_category_cols)
X= cdf[feature_cols] # Features
y= cdf['high_traffic'] # Target Variable
y= y.map({'Low Traffic': 0, 'High Traffic': 1})
```

[16]:
```
scaler = PowerTransformer()
numf = ['calories', 'carbohydrate', 'sugar', 'protein']
X.loc[:, numf] = scaler.fit_transform(X[numf])
```

[17]:
```
X.head()
```

[17]:
```
   calories  carbohydrate     sugar   protein  servings  category_Beverages  \
0 -1.392371      0.555369 -1.369492 -1.339950         4                 0.0
1  1.156281      0.648748 -0.268283 -0.812251         1                 0.0
2 -0.809689      0.345711  1.752953 -1.812703         4                 1.0
3 -1.525153     -1.625851 -1.262119 -1.509840         4                 1.0
4  0.848294     -1.292229 -0.773013  1.193668         2                 0.0

   category_Breakfast  category_Chicken  category_Dessert  \
0                 0.0               0.0               0.0
1                 1.0               0.0               0.0
2                 0.0               0.0               0.0
3                 0.0               0.0               0.0
4                 0.0               0.0               0.0

   category_Lunch/Snacks  category_Meat  category_One Dish Meal  \
0                    0.0            0.0                     0.0
1                    0.0            0.0                     0.0
2                    0.0            0.0                     0.0
3                    0.0            0.0                     0.0
4                    0.0            0.0                     1.0

   category_Pork  category_Potato  category_Vegetable
0            0.0              1.0                 0.0
1            0.0              0.0                 0.0
2            0.0              0.0                 0.0
3            0.0              0.0                 0.0
4            0.0              0.0                 0.0
```

[18]:
```
# Preliminary Checks
print(X.shape)
print(y.shape)
```

```
print(X.isnull().sum())
print(X.isin([np.inf, -np.inf]).sum())
print(y.isnull().sum())
print(y.isin([np.inf, -np.inf]).sum())
```

```
(895, 15)
(895,)
calories                  0
carbohydrate              0
sugar                     0
protein                   0
servings                  0
category_Beverages        0
category_Breakfast        0
category_Chicken          0
category_Dessert          0
category_Lunch/Snacks     0
category_Meat             0
category_One Dish Meal    0
category_Pork             0
category_Potato           0
category_Vegetable        0
dtype: int64
calories                  0
carbohydrate              0
sugar                     0
protein                   0
servings                  0
category_Beverages        0
category_Breakfast        0
category_Chicken          0
category_Dessert          0
category_Lunch/Snacks     0
category_Meat             0
category_One Dish Meal    0
category_Pork             0
category_Potato           0
category_Vegetable        0
dtype: int64
0
0
```

[19]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
    ↪random_state=42)
```

[20]:

```
models_params = {'LogisticRegression': {'model': LogisticRegression(),'params':␣
    ↪{'C': [0.001, 0.01, 0.1, 1, 10, 100],
```

```
↪'penalty': ['l1', 'l2'],
↪'solver': ['liblinear']}},
                'RandomForest': {'model': RandomForestClassifier(),'params':
↪{'n_estimators': [10, 50, 100, 200],
↪'max_features': ['auto', 'sqrt', 'log2'],
                                                                    'max_depth':
↪ [None, 10, 20, 30, 40, 50],
↪'min_samples_split': [2, 5, 10]}}}
```

('For loop' below may cause small lag when processing and partly due to insufficient data, the results are not stable and consecutive)

[21]:
```
#for model_name, mp in models_params.items():
 #   grid_search = GridSearchCV(mp['model'], mp['params'], cv=5,
 ↪scoring='precision')
  #  grid_search.fit(X_train, y_train)
   # print(f"{model_name} best parameters:", grid_search.best_params_)
    #print(f"{model_name} best cross-validation score: {grid_search.best_score_:
 ↪.2f}")
```

4.1 Logistic Regression Model

[22]:
```
log = LogisticRegression(penalty= 'l2', class_weight='balanced',C=
 ↪1,solver='liblinear')
log.fit(X_train, y_train)
```

[22]:
```
LogisticRegression(C=1, class_weight='balanced', solver='liblinear')
```

4.2 Random Forest Classifier Model

[23]:
```
rf =
 ↪RandomForestClassifier(n_estimators=50,max_depth=10,max_features='log2',min_samples_split=2
rf.fit(X_train, y_train)
```

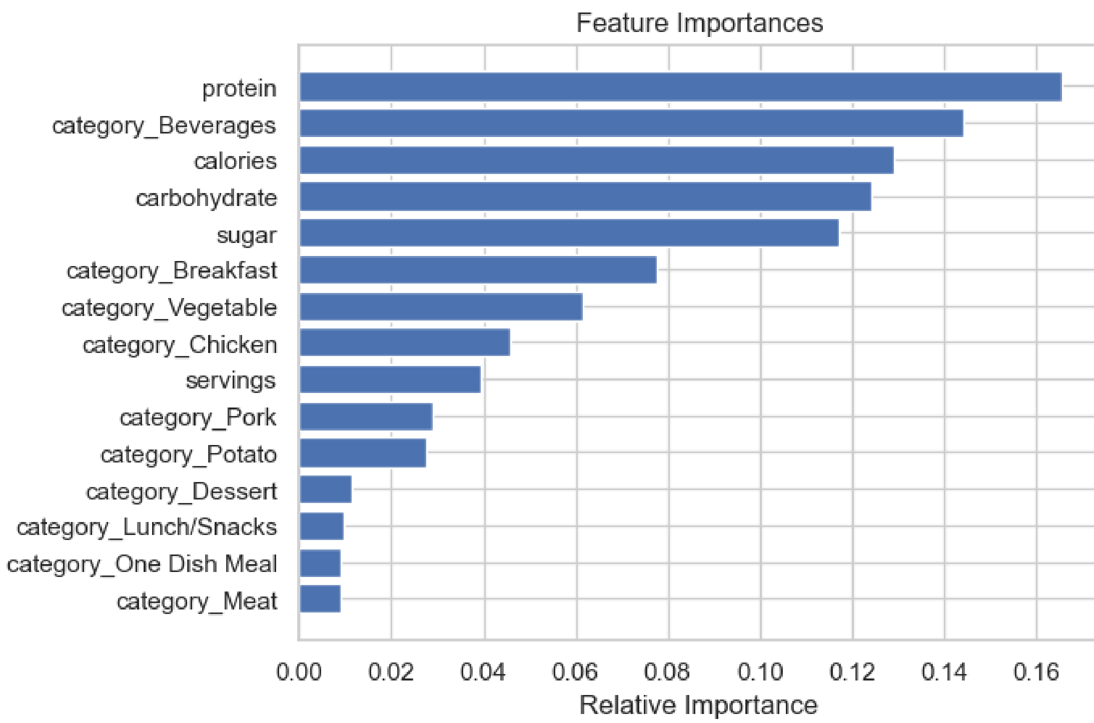
[23]:
```
RandomForestClassifier(max_depth=10, max_features='log2', n_estimators=50,
                       random_state=36)
```

4.2.1 Feature Importances

- Based off the analysis seen below, it appears more important to keep an eye on factors in a recipe that people dislike rather than like; as seen in the stacked bar graph before, 'Beverages' and 'Breakfast' are the least popular of the category feature and are now shown to be an important factor potentially turning clients to other avenues.

- Grouping the numerical variables to turn them into categorical variables will be saved for future analysis once more data is collected; this would most-likely show the opposing stories of the zero-calorie trend versus the protein-focused trend.

[24]:
```
importances = rf.feature_importances_
features = X_train.columns
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



5 Model Evaluation

For evaluation, **Precision** and **ROC-AUC scores** were used. Precision is chosen instead of Accuracy in order to maximize avoidance of false positives; a false positive represents having a recipe that is predicted to drive high traffic featured for 24 hours, when it in fact does not. A false negative of having a popular recipe not featured is less detrimental as long as there is another popular recipe in place. ROC AUC score will show the predicitive power of the model and its actual usefullness.
- Precision score of 1 represents 100% precision - ROC AUC score of 1 shows the model's positive effectiveness in the measure of separability with 0.5 representing random-guessing - **The metrics**

show the Logistic Regression Model to be more precise as well as more effective in its prediction capabilities

[25]:
```
pred_lr = log.predict(X_test)
pred_rf = rf.predict(X_test)

precision_lr = precision_score(y_test, pred_lr)
precision_rf = precision_score(y_test, pred_rf)
roc_auc_lr = roc_auc_score(y_test, pred_lr)
roc_auc_rf = roc_auc_score(y_test, pred_rf)
accuracy_lr = accuracy_score(y_test, pred_lr)
accuracy_rf = accuracy_score(y_test, pred_rf)

#print("Logistic Regression Model Accuracy Score: ", accuracy_lr)
#print("Random Forest Model Accuracy Score: ", accuracy_rf)
print("Logistic Regression Model Precision Score: ", precision_lr)
print("Logistic Regression Model ROC AUC Score: ", roc_auc_lr)
print('--------------------------')
print("Random Forest Model Precision Score: ", precision_rf)
print("Random Forest Model ROC AUC Score: ", roc_auc_rf)
```

```
Logistic Regression Model Precision Score:  0.7948717948717948
Logistic Regression Model ROC AUC Score:  0.7482609191469951
--------------------------
Random Forest Model Precision Score:  0.7219251336898396
Random Forest Model ROC AUC Score:  0.6929809556391835
```

6 Business Metrics

6.0.1 Current Metrics

Primary mission of the project is to predict which recipes will produce 'high_traffic'. This discrete binary status is determined by whether the website experiences a significant increase in visits — specifically, a 40% rise — on the day a recipe is featured. This metric was chosen based on observations from the product manager.

6.0.2 Analysis of Current Metric

Although binary classification problems are usually a lot simpler to solve with various models, it still leaves out key factors that can help lead to business growth. Already without building any algorithm, the starting accuracy was 60%. Once the models were built, a precision score of nearly 80% was achieved as desired. There is still potential for a more refined approach that can enhance the prediction reliability and granularity. From completing this work, identifying a popular recipe is now easier and should help increase traffic to the company's website, subscriptions, and revenue.

6.0.3 Proposed Improvements

1 - **Adding New Features**: many simple feature additions could be affecting high_traffic as well as popularity of the recipes. - 'time spent on page' - measuring how engaging the actual

content is - ‘bounce rate’ - measuring how “repulsive” the company’s landing page is - ‘time to make’ - measuring approximately how long it would take to prepare the recipe - ‘cost per serving’ - an important variable for cost-conscious consumers - ‘fat’ - another macro-nutrient to be familiar with next to protein and carbohydrate - ‘ingredients’ - list of the recipe’s main ingredients - ‘date featured’ - to check for possible correlations of recipe popularity due to calendar events

2 - **From Discrete to Continuous Target Variable**: if actual site visits per day were recorded, it would then turn this from a classification problem to a regression task, making it easier to acquire more precise predictions.

3 - **New KPI - Average Hourly Revenue Rate**: the new metric for the business to monitor. - Would refer to each of the main pages of the company’s site - Will be an aggregate of the most important features to better reflect user-engagement - Ambitious metric that measures rate of success and can be scaled when necessary

7 Recommendations

There are many paths moving forward that can lead to a more sophisticated machine learning model; it is recommended to roll out the following changes in a multiphase process: - Phase 1 : Increase granularity of the current model by adding more of the simpler features listed above - Phase 2 : Run a pilot model of the new regression model treating the target variable as the actual number of site visits - Phase 3 : Evaluate the possibility of using feature engineering to create more meaningful revenue-based metrics

7.1 Conclusion

The objective of 80% success in predicting popular recipes was practically reached; more important than this are the insights gained that point out various paths that can further increase company profits moving forward. It will be important for there to be good communication between the company’s data scientists and businessmen in order to refine the important metrics to better represent the company’s goals.