TASTY BYTES

Site traffic boost

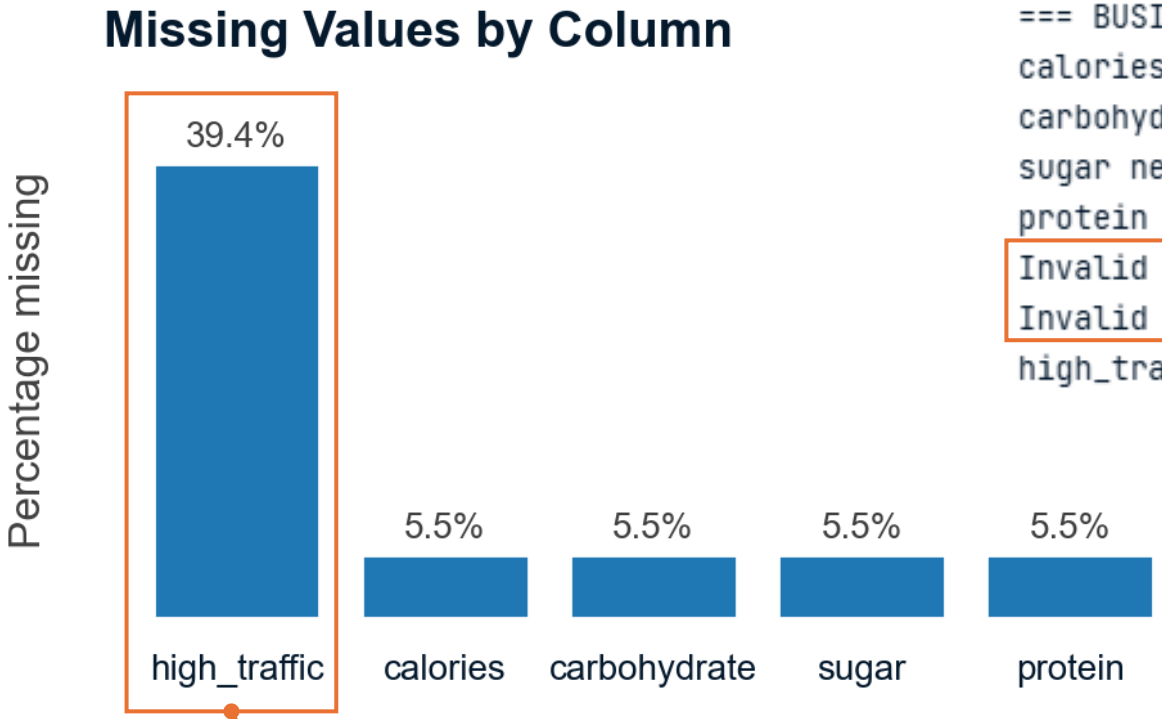
- Raise **site traffic** with **better recipes' selection** powered by machine learning
- Metrics to follow, improvements in data collection and **next steps**

André Canal July 09th, 2025

Data validation

Data Integrity — Missing values, business rules, data types

Dataset size = 947 registers




```
=== BUSINESS RULE VALIDATION ===
calories negative values: 0 (Business rule: >=0)
carbohydrate negative values: 0 (Business rule: >=0)
sugar negative values: 0 (Business rule: >=0)
protein negative values: 0 (Business rule: >=0)
Invalid categories: {'Chicken Breast'}
Invalid servings: ['4 as a snack' '6 as a snack']
high_traffic values: ['High' nan]
```

Missing values from the target represent the “0”

Invalid values were adjusted

Data validation: Summary 1

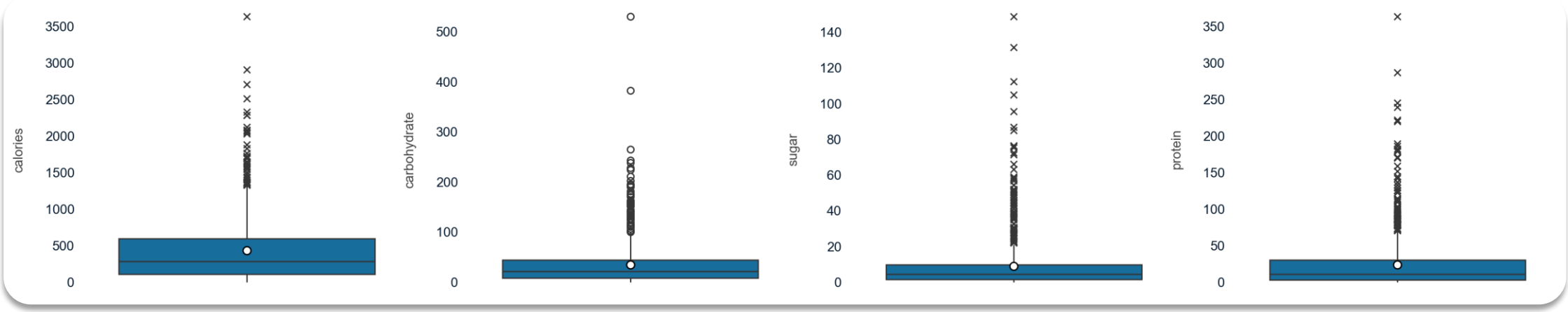
Data Integrity — Missing values, business rules, data types

Column	Validation	Issues	Cleaning/Handling	Business Rule
calories	Range, missing, type	5.49% missing	Dropped from the dataset	>= 0
carbohydrate	Range, missing, type	5.49% missing	Dropped from the dataset	>= 0
sugar	Range, missing, type	5.49% missing	Dropped from the dataset	>= 0
protein	Range, missing, type	5.49% missing	Dropped from the dataset	>= 0
category	Value check, missing, type	11 categories vs 10 in spec	Mapped 'Chicken Breast' to 'Chicken'	10 predefined groups
servings	Value check, missing, type	'4 as snack' format	Converted to integer	Must be 1, 2, 4 or 6
high_traffic	Missing, type	39.4% nulls	Converted to integer (1=High, 0=Low)	Target variable
recipe	Uniqueness, type	None	Converted to string	Unique identifier

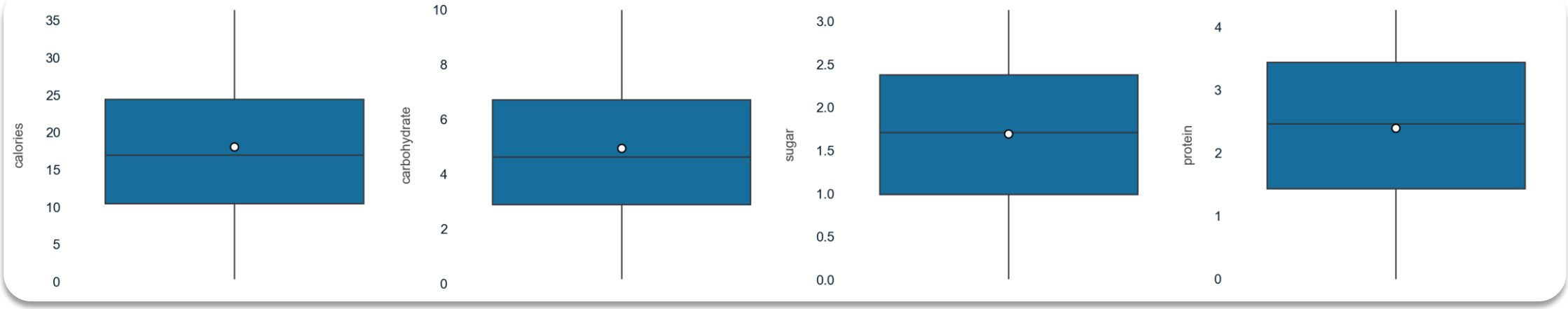
Data validation

Statistical properties

Outliers

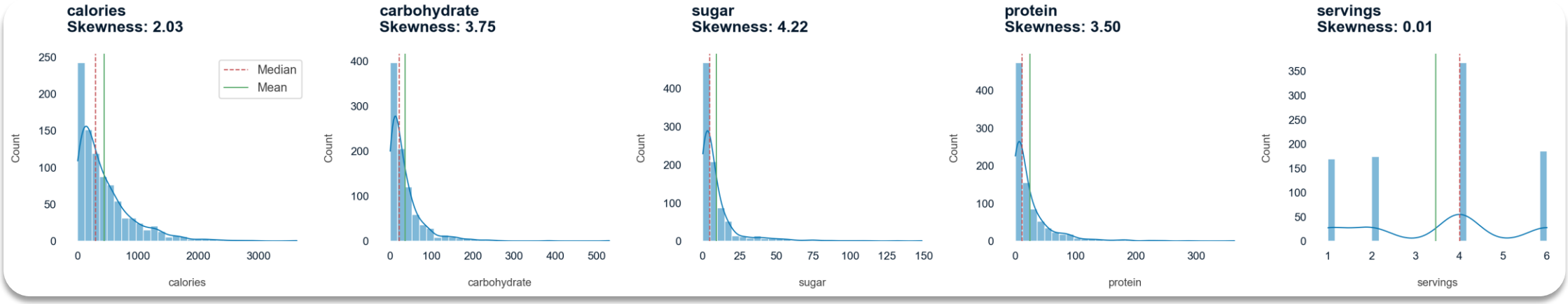


IQR caping

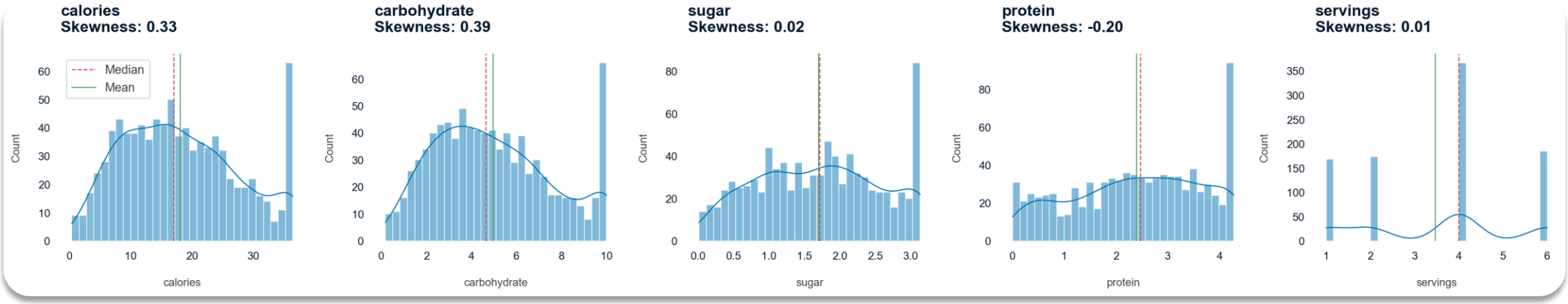


Data validation

Statistical properties | **Skewness**



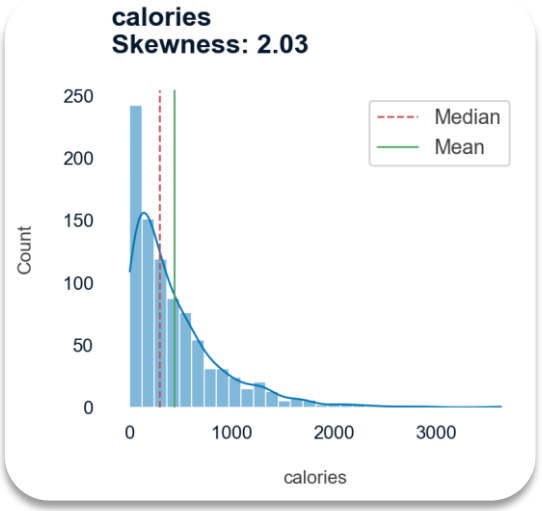
Log and sqrt transformations

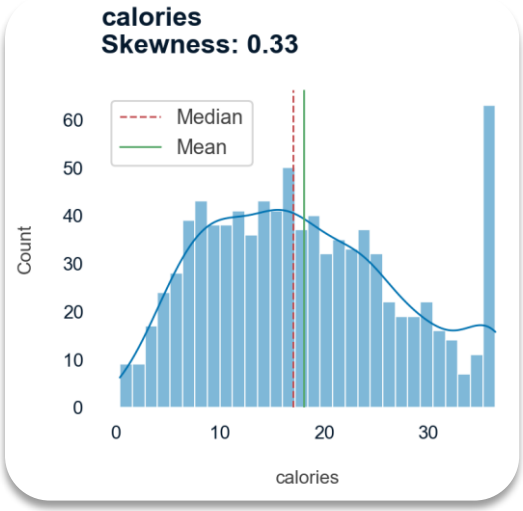


Data validation: Summary 2

Statistical properties | Outliers and skewness

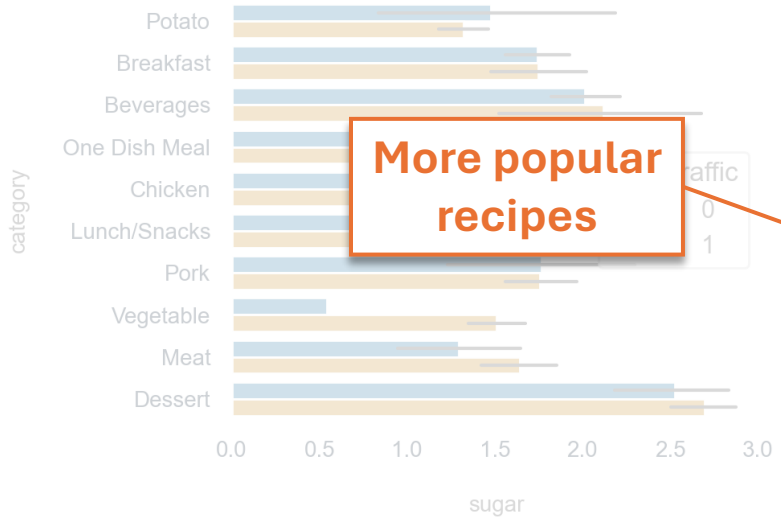
Column	Initial Skewness	Outlier Detection	Transformation Applied	Final Skewness
calories	2.03	IQR (Upper bound)	Log(x+1)	0.33
carbohydrate	3.75	IQR (Upper bound)	Log(x+1)	0.39
sugar	4.22	IQR (Upper bound)	Log(x+1)	0.02
protein	3.50	IQR (Upper bound)	Square root	-0.20
servings	0.01	None	None	0.01



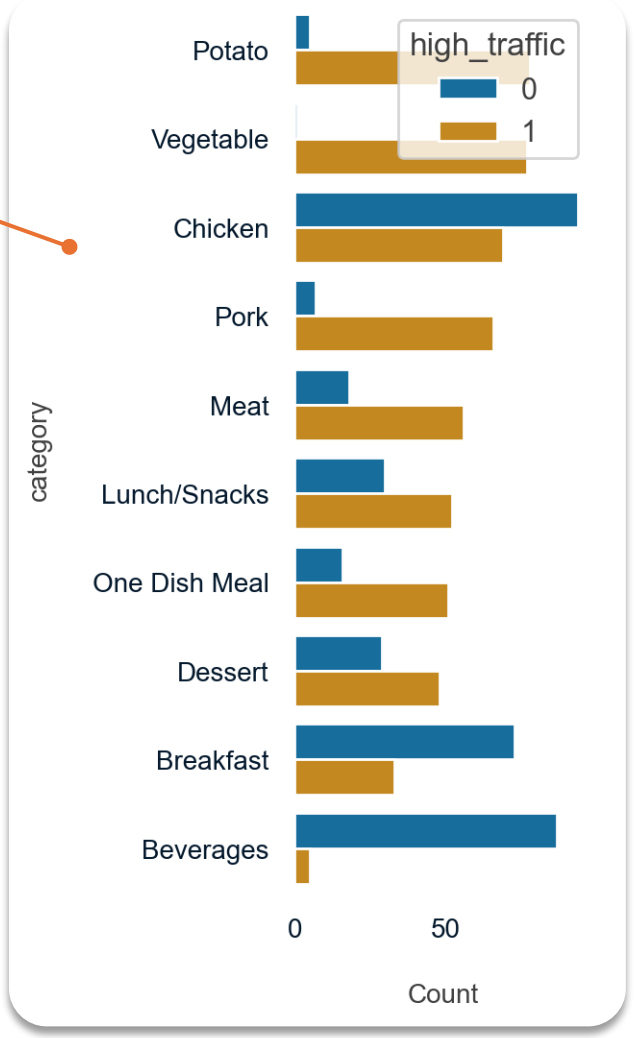


Feature engineering

Exploratory Data Analysis | **Patterns and correlations**



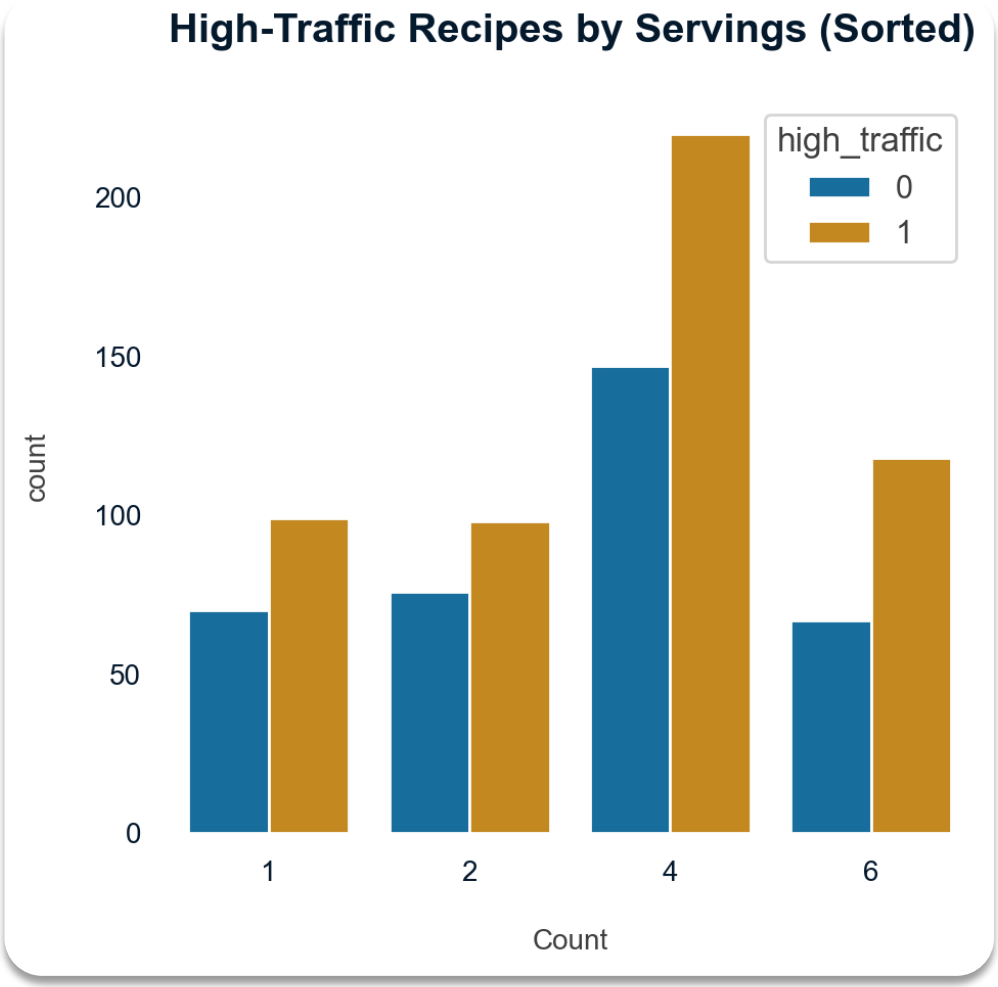
More popular recipes

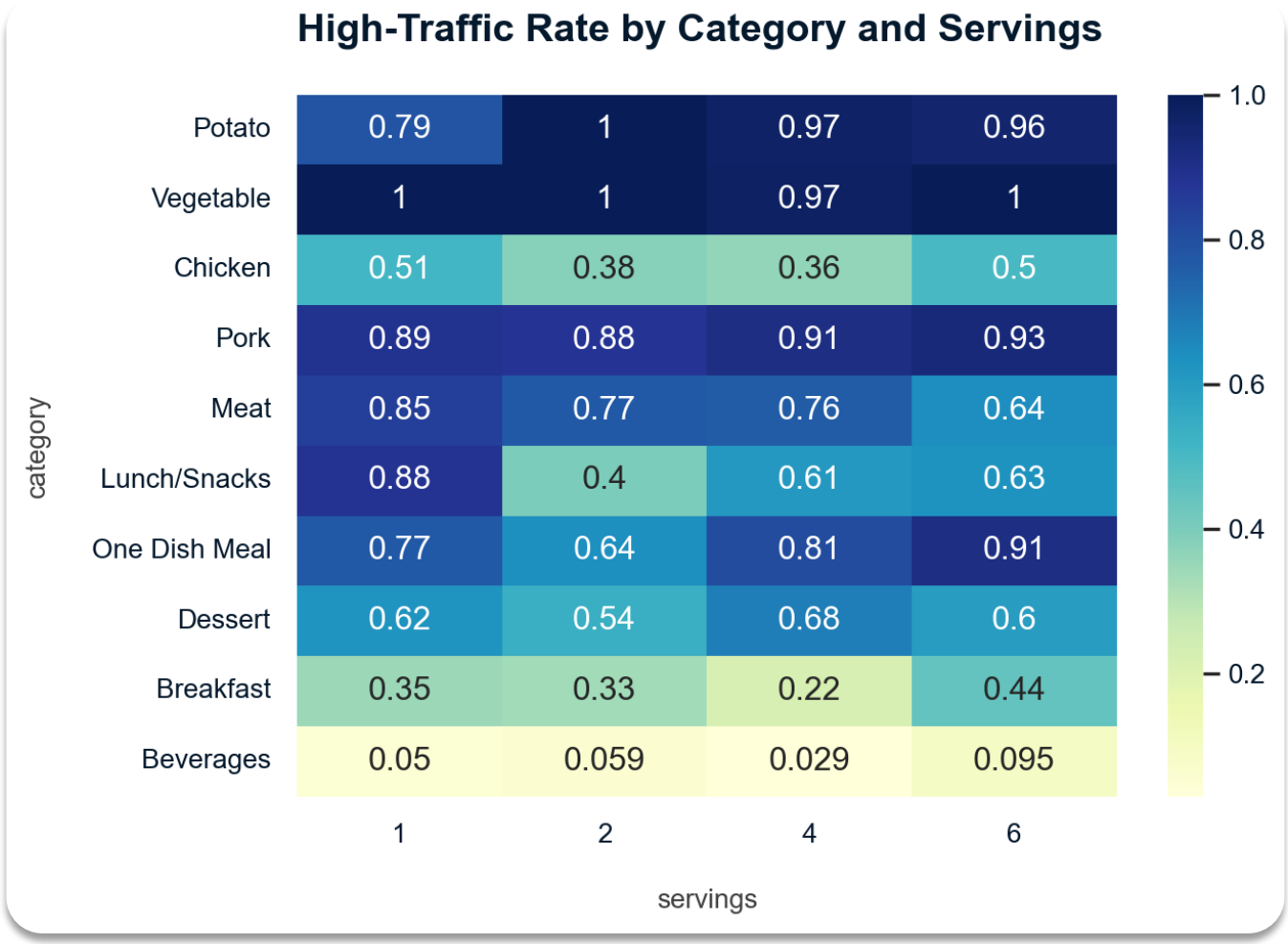


Feature engineering

Exploratory Data Analysis

Patterns and correlations

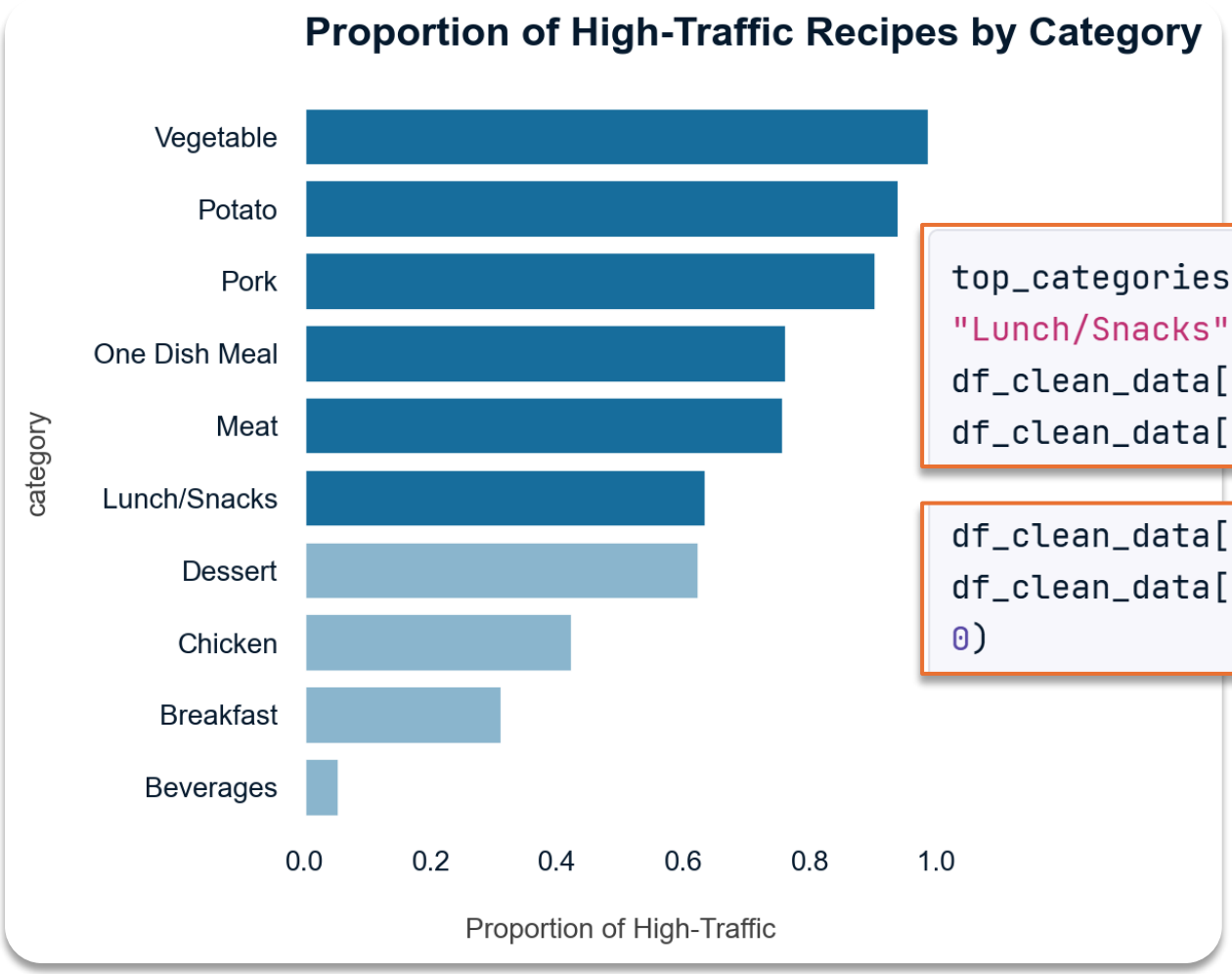




Feature engineering

Exploratory Data Analysis

Patterns and correlations




```
top_categories = ["Vegetable", "Potato", "Pork", "Meat"  [93]
"Lunch/Snacks", "One Dish Meal"]
df_clean_data["is_top_category"] =
df_clean_data["category"].isin(top_categories).astype(int)
```

```
df_clean_data["optimal_servings"] =
df_clean_data["servings"].apply(lambda x: 1 if x == 4 else
0)
```

Model selection

Modeling Ensemble | **Logistic Regression and Random Forest**

Preparing X's and y's

Spliting train, validation, and te rget.

```
X = df_clean_data.drop(col                fic']
y = df_clean_data['high_tr

X_train_, X_test, y_train_, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_, y_train_, test_size=0.2, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), ["calories", "carbohydrate", "sugar", "protein",
"servings"]),
        ("cat", OneHotEncoder(), ["category"])
        ,("interact", "passthrough", ["is_top_category"
                                      ,"meat_for_one"
                                      ,"lunch_snacks_for_one"
                                      ,"one_dish_for_six"
                                      ,"chicken_high_protein"
                                      ,"dessert_high_sugar"
                                      ,"dessert_high_carbo"
                                      ,"one_dish_high_carbo"
                                      ,"potato_high_carbo"
                                     ])
    ]
)

X_train_processed = preprocessor.fit_transform(X_train)
X_val_processed = preprocessor.transform(X_val)
```

X's and y's → **Pre-processor** → **Base model** — *Logistic Regression*

Comparison model 1 — *Logistic Regression trained with 10 k-fold cv*

Comparison model 2 — *Random Forest trained with top 10 features from Logistic Regression*

Comparison model 3 — *XGBoost* → **Ensemble model**

Model selection

Modeling Ensemble | **Logistic Regression and Random Forest**

Preparing X's and y's

Spliting train, validation, and t

```
X = df_clean_data.dro
y = df_clean_data['hi

X_train_, X_test, y_t
    X, y, test_size=0

X_train, X_val, y_train, y_val = train_test_split(
    X_train_, y_train_, test_size=0.2, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), ["calories", "carbohydrate", "sugar", "protein",
"servings"]),
        ("cat", OneHotEncoder(), ["category"])
        ,("interact", "passthrough", ["is_top_category"
                                      ,"meat_for_one"
                                      ,"lunch_snacks_for_one"
                                      ,"one_dish_for_six"
                                      ,"chicken_high_protein"
                                      ,"dessert_high_sugar"
                                      ,"dessert_high_carbo"
                                      ,"one_dish_high_carbo"
                                      ,"potato_high_carbo"
                                     ])
    ]
)

X_train_processed = preprocessor.fit_transform(X_train)
X_val_processed = preprocessor.transform(X_val)
```

X's and y's → **Pre-processor** → **Base model** — *Logistic Regression*

Comparison model 1 — *Logistic Regression trained with 10 k-fold cv*

Comparison model 2 — *Random Forest trained with the top 10 features from Logistic Regression*

Comparison model 3 — *XGBoost*

→ **Ensemble model**

Model selection

Modeling Ensemble	Logistic Regression and Random Forest

```
from sklearn.ensemble import VotingClassifier

ensemble = VotingClassifier(
    estimators=[("logreg", clf_tuned), ("rf", rf_pruned)],
    voting="hard"  # Uses probabilities
    , weights=[2, 1]
)
ensemble.fit(X_train_processed_final, y_train_)
y_test_pred = ensemble.predict(X_test_processed)
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      0.75      0.71        73
           1       0.81      0.75      0.78       106

    accuracy                           0.75       179
   macro avg       0.74      0.75      0.74       179
weighted avg       0.76      0.75      0.75       179
```

Ensemble model

81% *Precision*

75% *Recall*

An ensemble voting classifier combining **Logistic Regression** and **Random Forest**

Recommendations for the marketing team

Ensemble model

81% *Precision*

75% *Recall*

An ensemble voting classifier combining **Logistic Regression** and **Random Forest**

The metric

Precision > 80%

- **Deploy ensemble model**
  - Build **admin tool** (Python + Streamlit/Flask) to predict "POPULAR"/"NOT POPULAR" w/ probability.
- **Monitor Performance**
  - Track **% high-traffic recipes** weekly; retrain if precision <75%.
- **Hybrid human-model decision making**
  - Allow overrides from the product manager when model's confidence is between 60-80%.
- **Enhance data**
  - Collect user feedback (ratings, cooking time).
- **A/B test**
  - Run **2-week trial**: model vs. manual picks. Measure traffic & subscriptions.

TASTY BYTES

Site traffic boost

Thank you!

André Canal July 09th, 2025