# 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..*

## 1. Data Validation: Data survey and cleaning

- A very first step is to load the data into a `pandas` dataframe and observe the first 5 rows using `.head()`

| index | recipe | calories | carbohydrate | sugar | protein | category |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | Pork |
| 1 | 2 | 35.48 | 38.56 | 0.66 | 0.92 | Potato |
| 2 | 3 | 914.28 | 42.68 | 3.09 | 2.88 | Breakfas |
| 3 | 4 | 97.03 | 30.56 | 38.63 | 0.02 | Beverage |
| 4 | 5 | 27.05 | 1.85 | 0.8 | 0.53 | Beverage |

Rows: 5 Expand Table

- By using `.info()`, we can quickly check:

1. If the there is any `nan` values in the data
2. The current data types of each columns

- Comments:

1. Columns `calories`, `carbohydrate`, `sugar`, `protein` and `high_traffic` contains `nan` values
2. `servings` should be numeric but detected as `object` / `str`

- Use `describe(include = 'all')` to show statistics of each columns
- By default, `describe()` only shows statistics of numeric columns. To include all columns, need to specicfy the argument `inlcude = all`
- Comments:

1. `recipe` is just a identifier, we will not need this column when analyzing and building the model
2. For other `numeric` columns (i.e., `calories`, `carbohydrate`, `sugar`, `protein`)

- By observing `min` and `max` of : Seems ok, there is no `abnormality` (e.g., negative values, too large values) in those columns
- By observing `mean` and `std` of `calories`, `carbohydrate`, `sugar`, `protein`: the data of each columns ranges a lot (e.g., in `calories`, `std` (453.02) is even larger than the `mean` (435.93))

2. For `non-numeric` columns (i.e., `category`, `servings`, `high_traffic`)

- For `category` column: `Breakfast` is the most frequent class
- For `servings` column: `4` is the most frequent class
- For `high_traffic` column: there is only one class `High`

| in... | recipe | calories | carbohydrate | sugar | protein |
|---|---|---|---|---|---|
| count | 947 | 895 | 895 | 895 | |
| unique | | | | | |
| top | | | | | |
| freq | | | | | |
| mean | 474 | 435.9391955307 | 35.0696759777 | 9.046547486 | 24.: |
| std | 273.5196519448 | 453.0209971775 | 43.9490319812 | 14.6791758036 | 36.: |
| min | 1 | 0.14 | 0.03 | 0.01 | |
| 25% | 237.5 | 110.43 | 8.375 | 1.69 | |
| 50% | 474 | 288.55 | 21.48 | 4.55 | |
| 75% | 710.5 | 597.65 | 44.965 | 9.8 | |
| max | 947 | 3633.16 | 530.42 | 148.75 | |

Rows: 11 Expand Table

- Next, the next cell is to compute the percentage of missing values of each columns
- `calories`, `carbohydrate`, `sugar`, `protein` have a same number of missing values. This is a signal of Missing Completely at Random (MCAR). We'll need to have a further investigation for confirmation.
- `high_traffic`: absolutely Missing Not at Random (MNAR) because the missing value (null) is intentionally left blank when the traffic is not high.

- In the next cell, we can see that the data indices of missing values of all columns `calories`, `carbohydrate`, `sugar`, `protein` are the same.
- We can conclude that the missing rows are in type of MCAR (i.e, by somehow, the reporter skips inputing the information for those rows). Therefore, removing them is a reasonable solution

- The next cell is to confirm the `recipe` is just a identifier
- Number of unique values is exactly equal to the number of rows of the given data (i.e., 947 rows)

- Now, let's examine the others `non_numeric` columns: `servings` and `category`:

- For `servings`: `4 as a snack` and `6 as a snack` are the reasons why the data type of this column is `object`/`str`. We will need to clean this
- For `category`: it seems okay but there might be overlapsed group such as `Chicken Breast`/`Chicken` or `Meat`/`Pork`/`Chicken`. This will introduce unnessary complexity to the model later

- Now, I think we have some ideas to clean the data before going to the stage EDA. The cleaning process is conducted as following:

1. Step 1: remove the `recipe` column using `.drop()` because it's just a identifier which does not contain any useful information for analyzing
2. Step 2: remove missing rows in `calories`, `carbohydrate`, `sugar`, `protein` by using `.dropna()`
3. Step 3: use `.apply()` to get the number only and then convert the column type to `int` by using `.astype(int)`
4. Step 4: again, also use `.astype(str)` to convert the data type of `category` column to `str`
5. Step 5: finally, replace missing values of `high_traffic` column by 0, and convert `High` text to 1 for more convenience in the EDA step

- After cleaning the data, we again use `.describe()` to have a look at the cleaned one

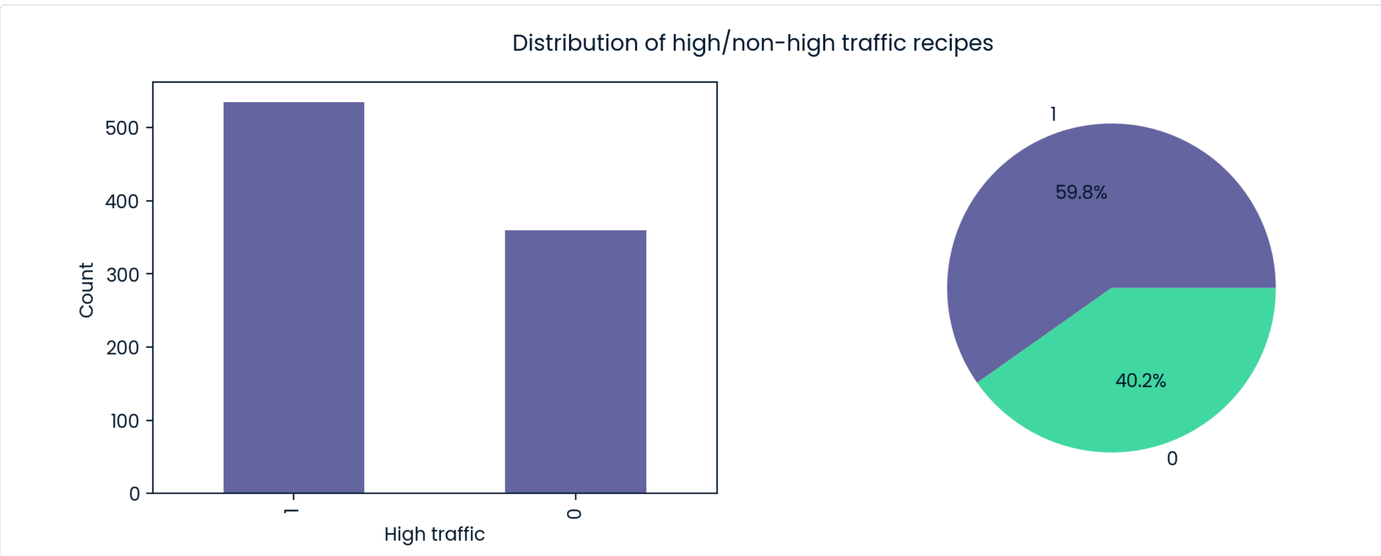
| | calories | carbo... | sugar | protein | c. | servi... | high... |
|---|---|---|---|---|---|---|---|
| count | 895 | 895 | 895 | 895 | 895 | 895 | 895 |
| unique | | | | | 11 | | |
| top | | | | | Breakfast | | |
| freq | | | | | 106 | | |
| mean | 435.9391955307 | 35.0696759777 | 9.046547486 | 24.1492960894 | null | 3.4581005587 | 0.5977653631 |
| std | 453.0209971775 | 43.9490319812 | 14.6791758036 | 36.3697385865 | null | 1.735978913 | 0.4906229555 |
| min | 0.14 | 0.03 | 0.01 | 0 | null | 1 | 0 |
| 25% | 110.43 | 8.375 | 1.69 | 3.195 | null | 2 | 0 |
| 50% | 288.55 | 21.48 | 4.55 | 10.8 | null | 4 | 1 |
| 75% | 597.65 | 44.965 | 9.8 | 30.2 | null | 4 | 1 |
| max | 3633.16 | 530.42 | 148.75 | 363.36 | null | 6 | 1 |

Rows: 11 Expand Table

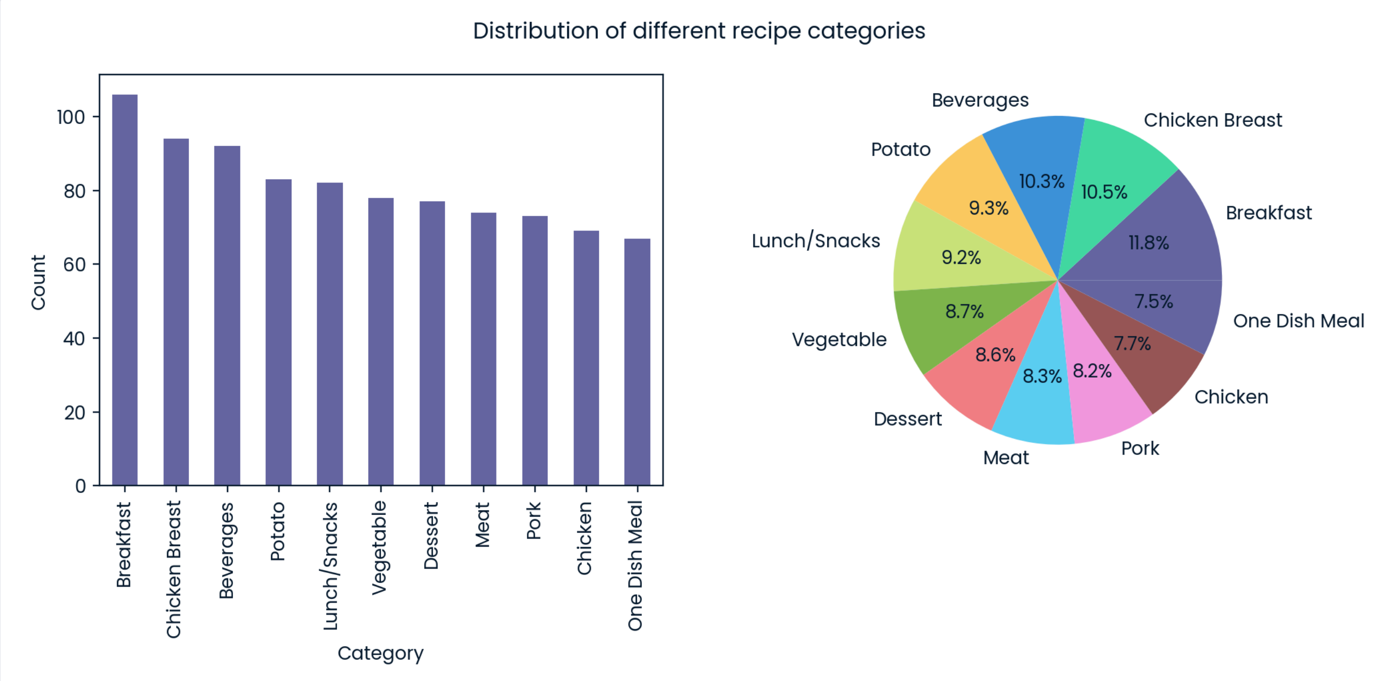
## 2. EDA

- Let's first examine our target column `high_traffic` to see the distribution of each class
- By using both bar chart and pie chart, we can see that:

-> Luckily, the class 0 and 1 seems balanced. It means we might dont need to use special techiniques to handle the class-imbalance problem during constructing an ML model

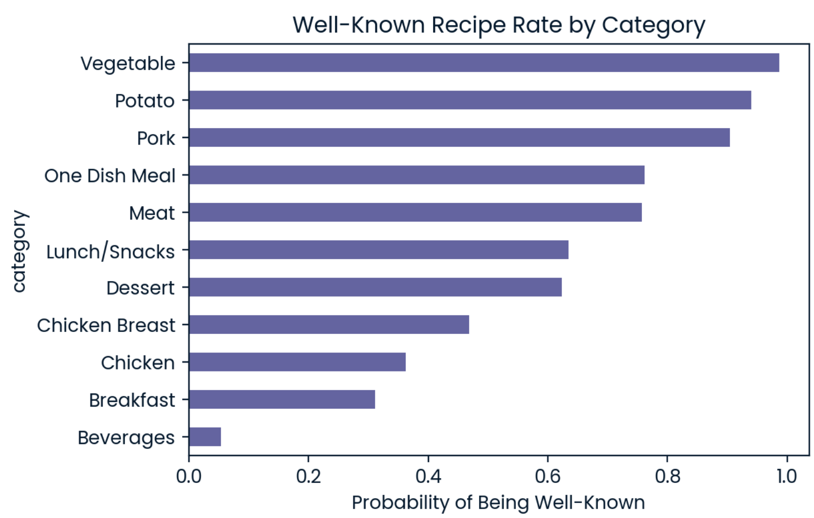



- A same stratergy applied to the column `category`, we see that:
- Each category group (e.g., Potato, Beverages,..) share a more or less similar contributions around 8% ~ 9%
- Therefore, we dont need to have any further preprocess before the modelling.



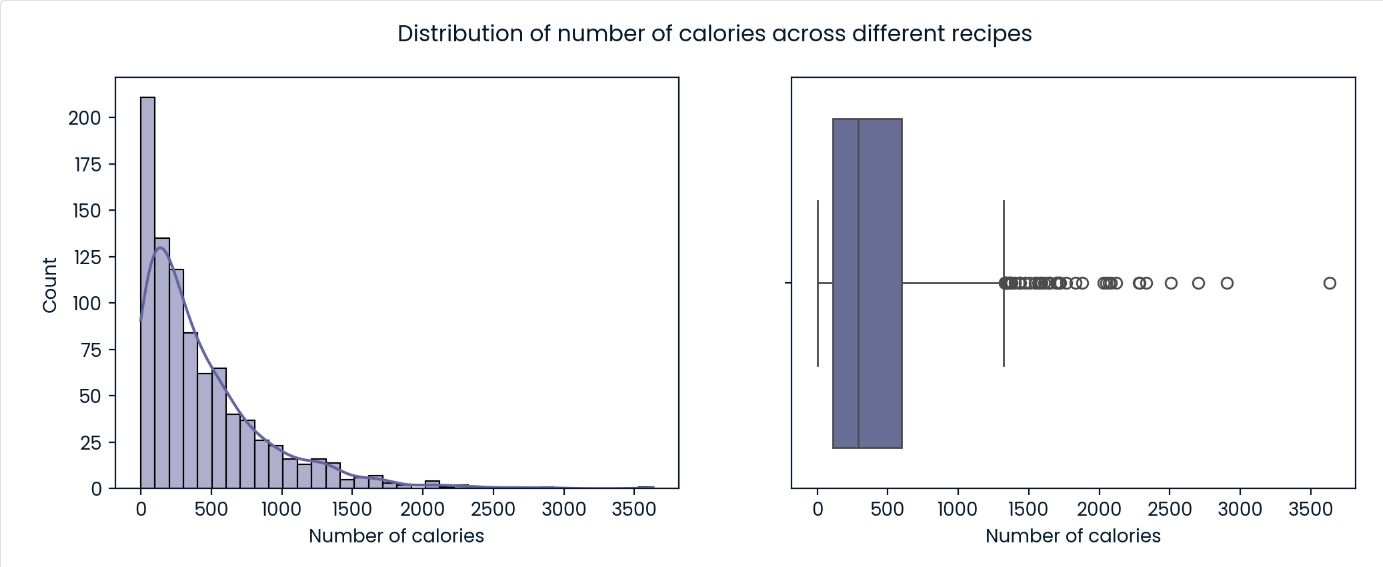

Subsequently, we might wonder which categories are more likely to be well-known. To answer this question, we can observe the high_traffic rate for each category and see that:

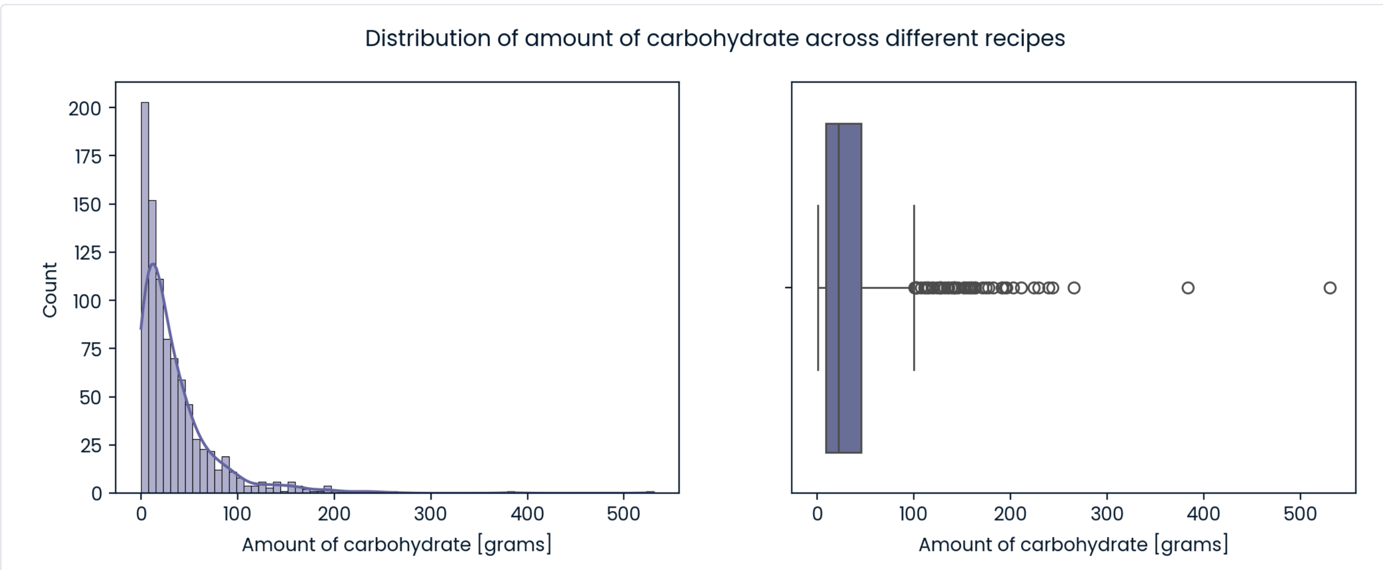
1. Vegetable, Potato, and Pork are the top three categories that easily capture the interest of viewers. (i.e., rate above 0.8)
2. In contrast, Chicken, Breakfast, and Beverages are categories that tend to be ignored. (i.e., rate below 0.4)
3. The remaining categories fall somewhere in between.

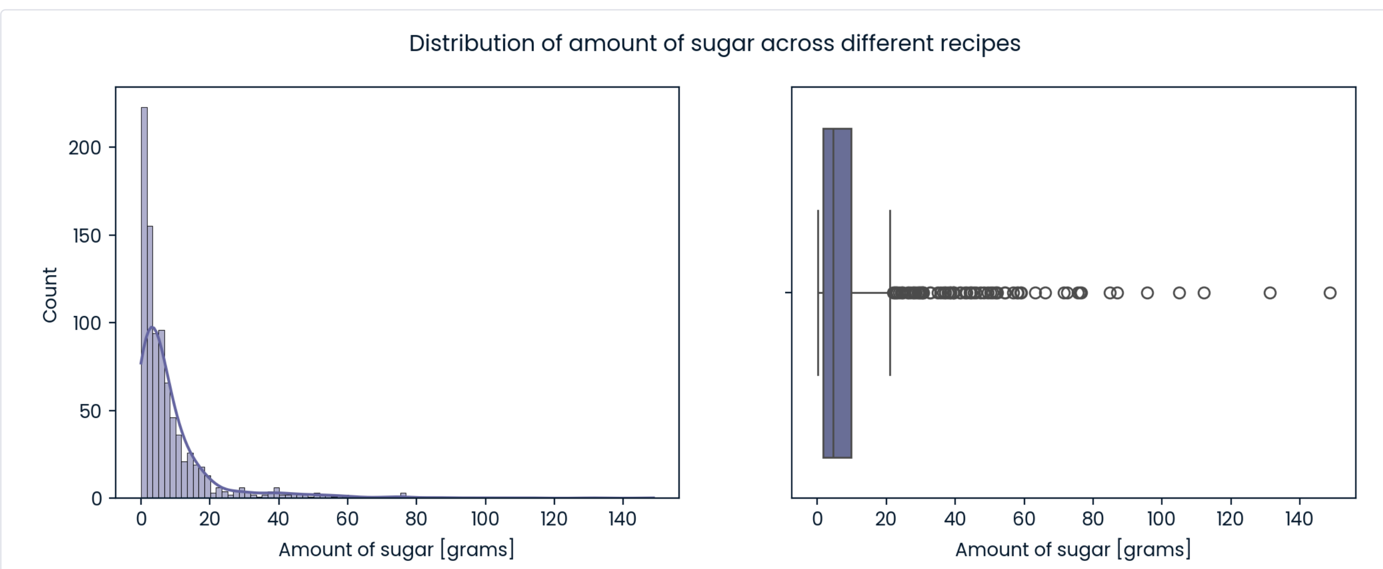


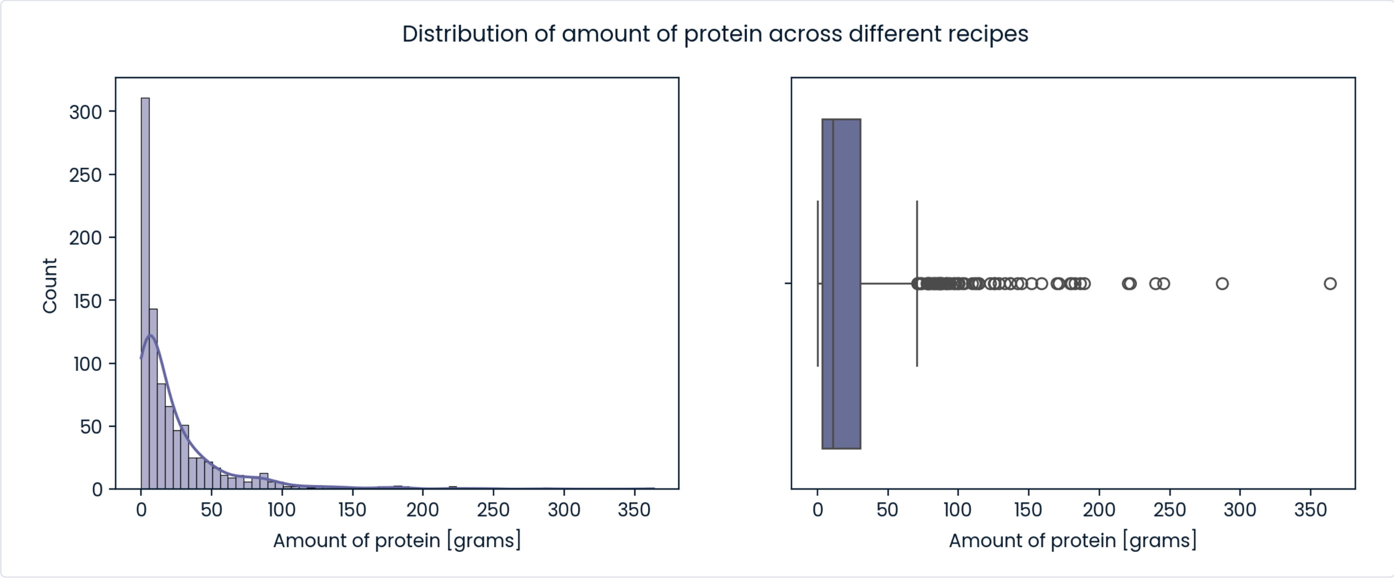

- The next four figures is to analyze the distribution of the four columns: `calories`, `carbohydrate`, `sugar` and `protein`
- In general, the distribution of these four features share some same chacteristics as follows:
  - Have long tail distributions
  - Contain outliers

-> We will need to standardize/normalize/transform them to make the model learn easier

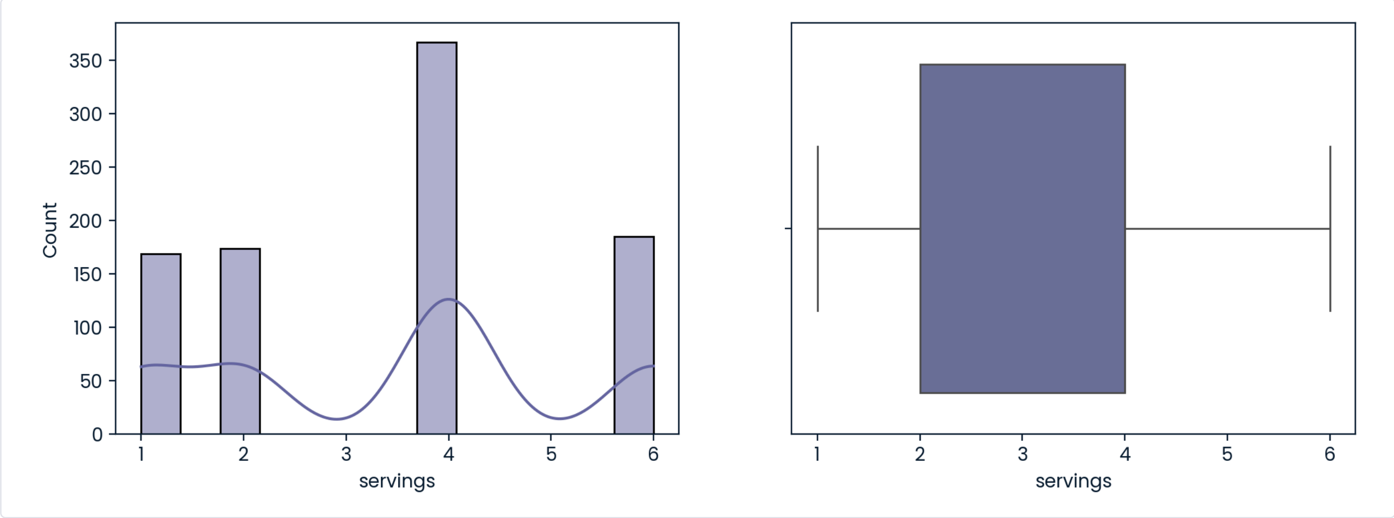

Distribution of number of calories across different recipes
Count
Number of calories
Number of calories


Distribution of amount of carbohydrate across different recipes
Count
Amount of carbohydrate [grams]
Amount of carbohydrate [grams]

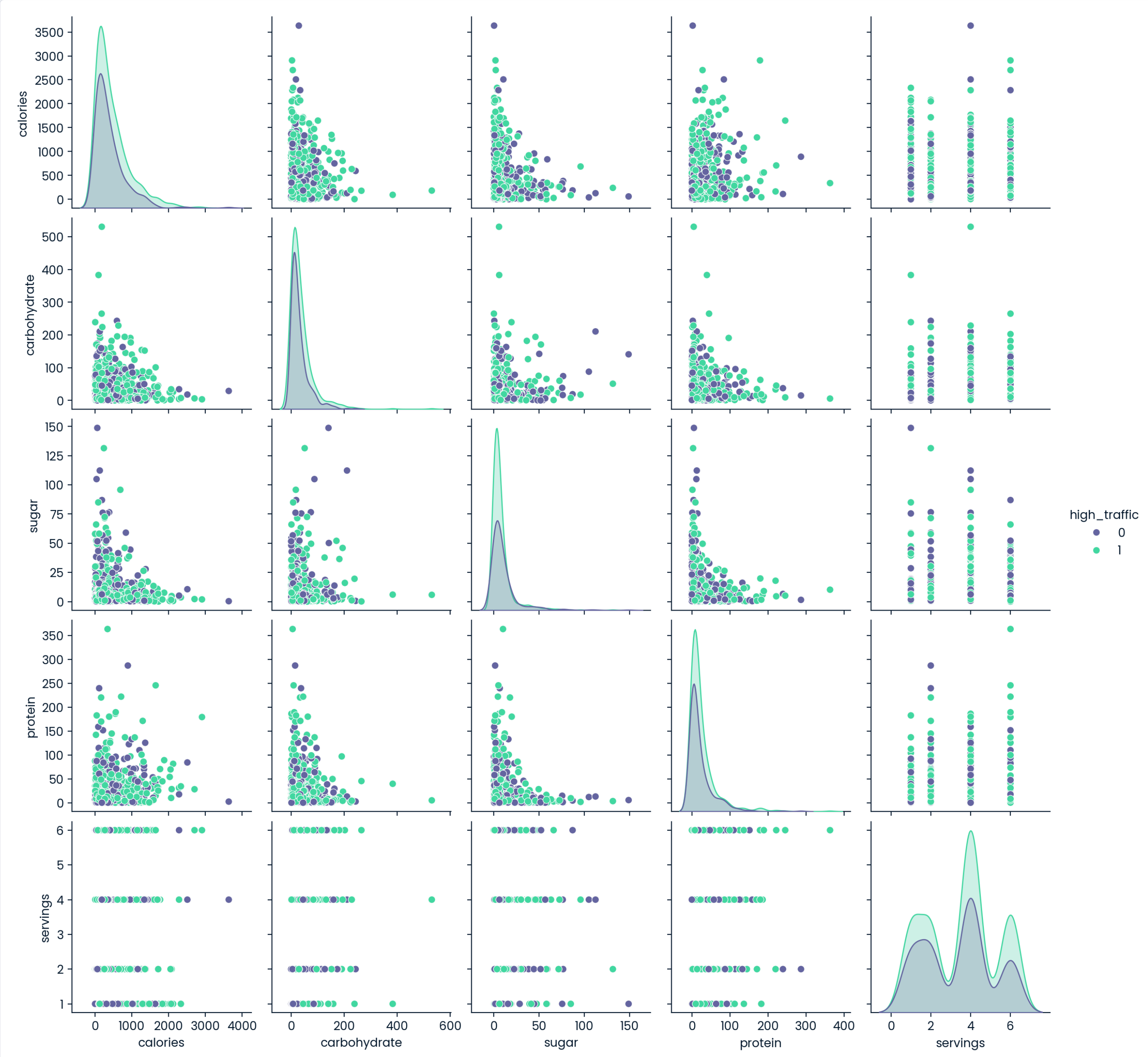

Distribution of amount of sugar across different recipes
Count
Amount of sugar [grams]
Amount of sugar [grams]




- Similarly, a histogram and a boxplot are also used to investigate values of number of `servings`
- We see that a larger concentration is on 4 serving




- Next, a pair plot is also an useful tool to analyze the correlation between each input features w.r.t different target classes:
- In details:
  - Regarding the scatter plots (i.e., the off-diagonal ones), there do not appear to be strong linear correlations between the input features. Additionally, it would be helpful to confirm this observation using a specific metric (e.g., Pearson correlation).
  - Another remark about the pair plot is the distribution of each input feature with respect to the target classes. It appears that these distributions are nearly identical for both class 0 and class 1. This suggests that there may be no easily separable patterns between the classes.



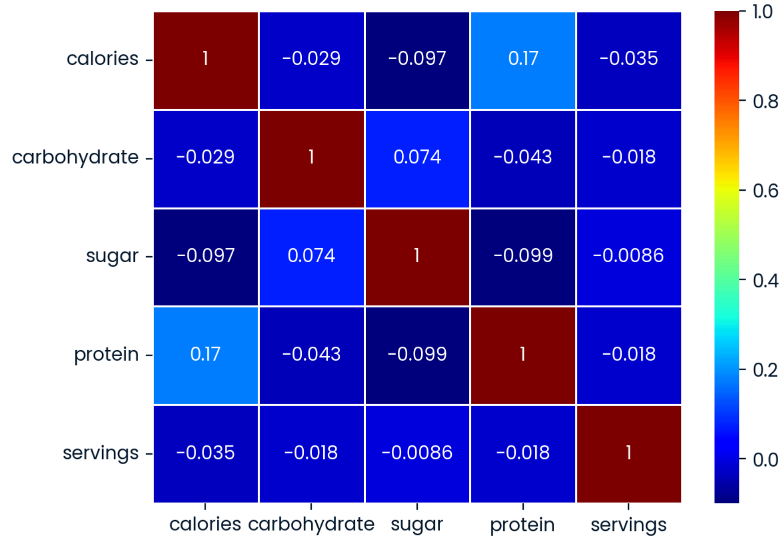

- As mentioned earlier, it's better to confirm whether there is any linear correlation between the input features.
- To do this, we compute the Pearson correlation using `.corr()` and visualize it with a heatmap to easily identify any strong patterns.

From the heatmap, we can see that

- There is no significant linear correlation between the input features, as the Pearson values are all around 0.1.

-> Therefore, we will need keep all the input features when modeling the machine learning model

```
<AxesSubplot: >
```




# 3. Model Development, Evaluation and Definition of Business metrics

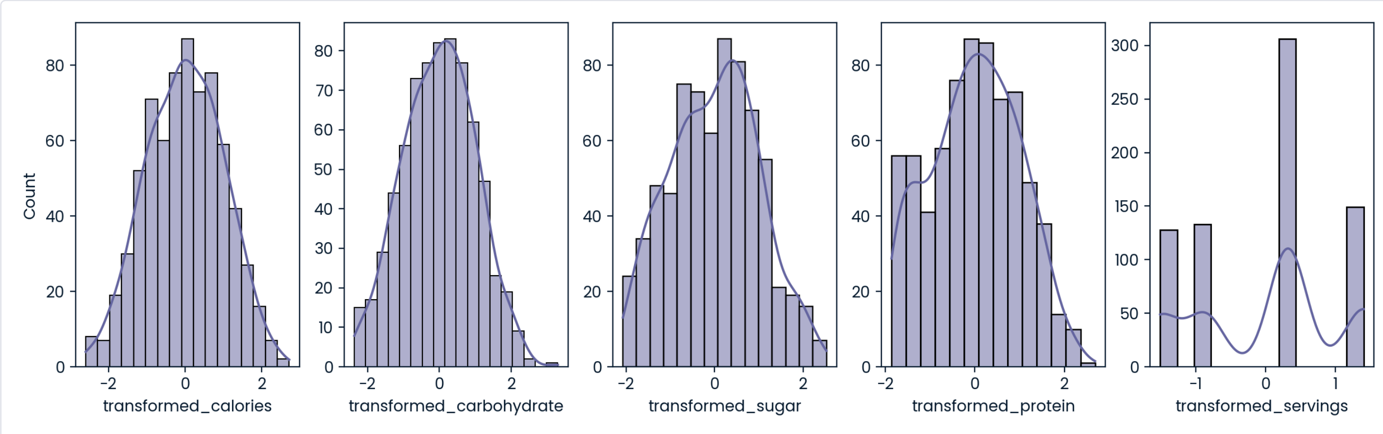
- So far, it's clear that we are working on a **binary classification problem**.
- As a result, there are two key business metrics we need to focus on:
  1. **Accuracy** – to understand the overall prediction performance of the model.
  2. **Precision** – to measure how accurate the model is when predicting the positive class. This is especially important in our case because we want to **minimize the chance of recommending unpopular recipes**.
- These two metrics will serve as the basis for comparing the performance of different models.

- Next, before constructing any ML model, we will need to define a baseline model
- A common and naive way to set up a baseline classifier is to use a model that always predicts the majority class of the observations.
- In this case, the baseline model always predicts `1` because `1` is the majority class.
- The accuracy of this model is therefore equal to the proportion of class `1` in the dataset. We will use this accuracy as the baseline metric to evaluate the performance of subsequent models.

- The code below is used to create the training and test datasets for constructing and evaluating the model.
- The train-test split ratio is set to 80/20, and stratified sampling is applied based on the target variable.
- By using `.value_counts(normalize=True)`, we can confirm that the proportion of each target class is similar in both the training and test datasets.

- We need to transform all the columns before constructing the model:
  - For `str` (categorical) columns, apply `OneHotEncoder()` to convert texts to numbers
  - For `num` (numerical) columns, apply `PowerTransformer()` to standardize the distributions.

- Again, it's a good idea to double-check that the transformations worked correctly.
- As we can see, the distributions of all numeric columns are standardized as expected.




### Summary of the Model Training Strategy:

The strategy involves training and tuning multiple classification models using **random search** over a range of hyperparameters. The goal is to identify the best-performing model for the given classification task (e.g., predicting whether a recipe is attractive or not).

**Key Elements of the Strategy:**

1. **Model Variety:**
    - A wide range of classifiers are tested, including:
        - Linear models: `LinearSVC`, `LogisticRegression`
        - Nonlinear models: `SVC` (with kernel options), `KNeighborsClassifier`
        - Tree-based models: `RandomForestClassifier`, `GradientBoostingClassifier`, `HistGradientBoostingClassifier`
2. **Dimensionality Reduction:**
    - Most models are tested **with and without PCA**, using different values for the number of components ( `pca__n_components` ), to evaluate whether reducing dimensionality improves performance.
3. **Hyperparameter Tuning:**
    - Each model has its own set of hyperparameters defined in a dictionary.
    - These hyperparameters are selected to influence regularization, complexity, learning rates, and distance metrics depending on the model type.

### ✅ Summary of the Strategy

The below code follows a structured **model evaluation strategy** using the following steps:

### 1. Preprocessing & Dimensionality Reduction

- **Column Transformer** ( `column_trans` ) handles data preprocessing.
- **PCA** is optionally applied to reduce feature dimensionality before model training.

### 2. Model Selection & Tuning

- A variety of models (linear, tree-based, ensemble) are considered.
- Each model is wrapped in a **pipeline** (preprocessing → PCA → model).
- **RandomizedSearchCV** is used to find the best hyperparameters:
    - 50 random combinations
    - 4-fold **Stratified K-Fold** cross-validation
    - Scored using **F1-score** (i.e., the **F1-score** is a robust metric used to avoid imbalance in the dataset (if have))

### 3. Training & Evaluation

- Each tuned model is trained on the **training data**.
- Predictions are made on both:
    - **Training set** (to check overfitting)
    - **Test set** (to evaluate generalization)
- **Accuracy** and **Precision** are calculated and logged.

### 4. Performance Logging

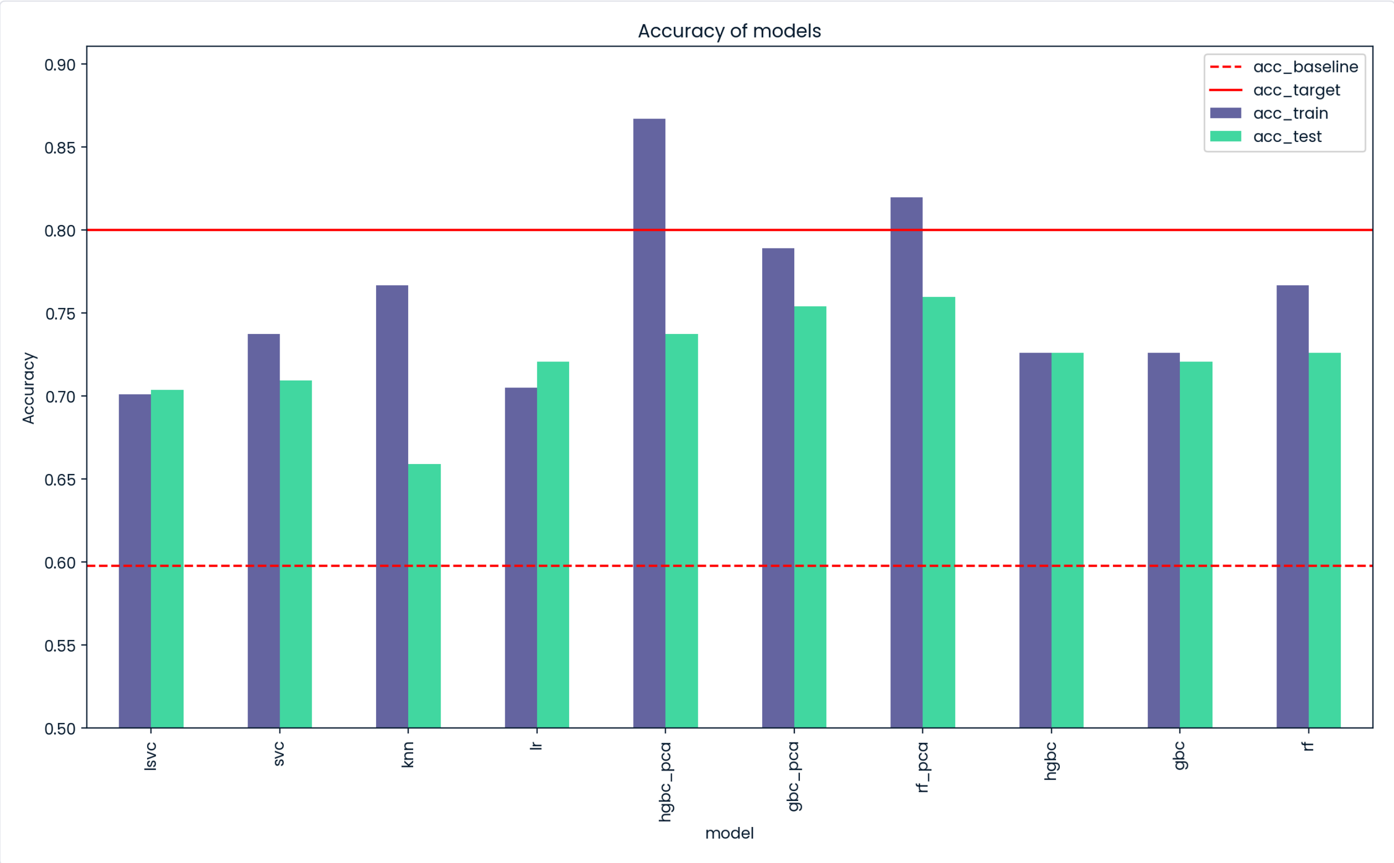
- Results (train/test accuracy and precision) are stored in a dictionary for all models.
- This allows easy comparison across models and configurations.

### 🔍 Observations:

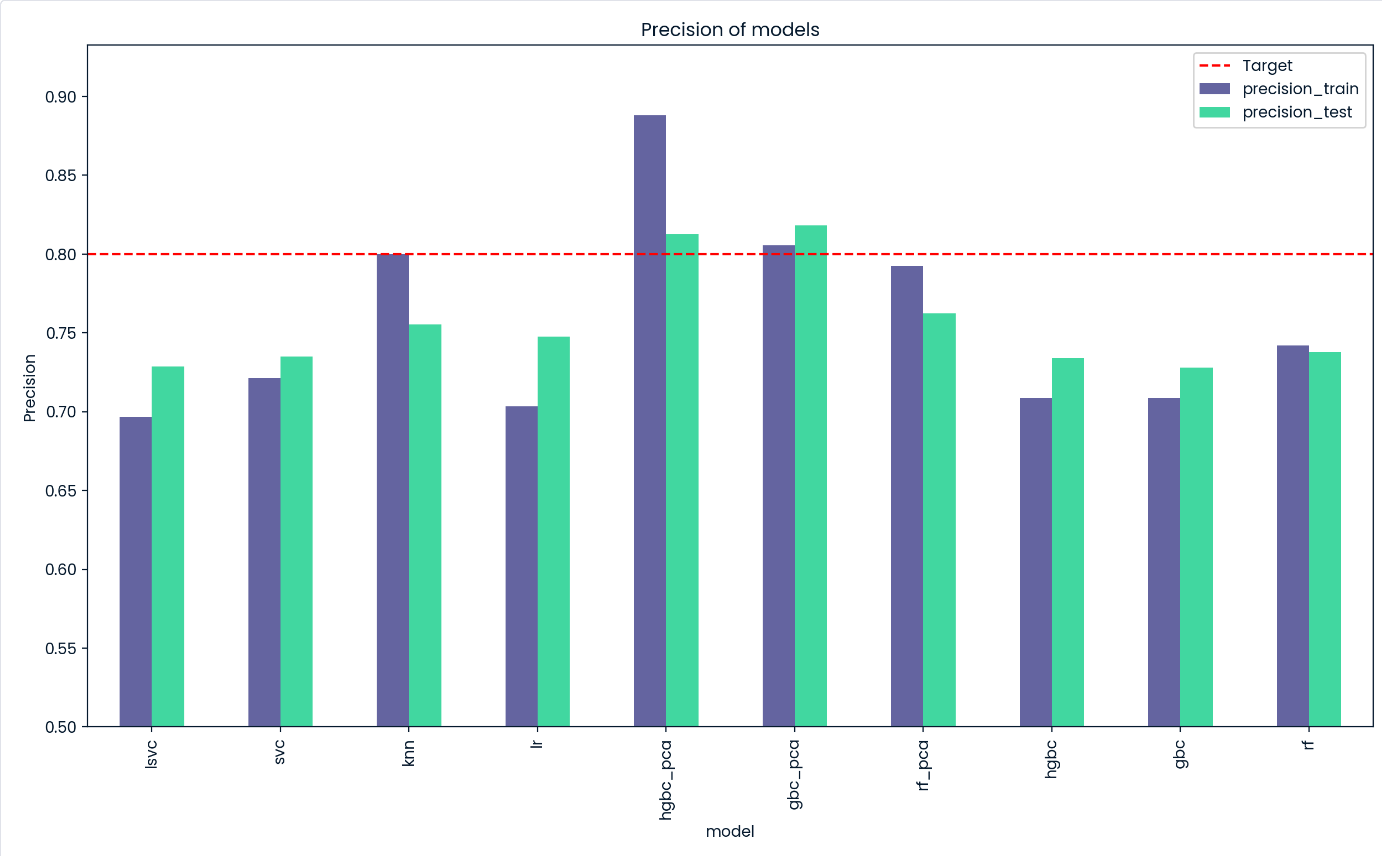
1. **Models Evaluated:**
    - Models include basic classifiers ( `lsvc` , `svc` , `knn` , `lr` ) and ensemble methods with and without PCA ( `hgbc_pca` , `gbc_pca` , `rf_pca` , `hgbc` , `gbc` , `rf` ).
2. **Accuracy Bars:**
    - **Dark purple bars**: Training accuracy ( `acc_train` )
    - **Light teal bars:** Test accuracy ( `acc_test` )
3. **Reference Lines:**
    - 🔴 **Red solid line** ( `acc_target` ): Accuracy target threshold (0.80)
    - 🔴 **Red dashed line** ( `acc_baseline` ): Baseline accuracy (~0.60)
4. **Model Performance:**
    - **Top performers on test data:** `rf_pca` , `gbc_pca` , and `hgbc_pca` are the only ones approaching or exceeding the target threshold (especially `rf_pca` ). This is an interesting finding because the tree-based model normally does not need PCA to perform good predictions
    - **Overfitting signs:** Some models (e.g., `hgbc_pca` , `rf_pca` ) show a noticeable gap between train and test accuracy, which may indicate overfitting.
    - **Underperformers:** `knn` , `lsvc` , and `lr` have lower test accuracy, barely above the baseline.

### 📌 Conclusion:

- The best generalization (both high accuracy on train and test data) seems to come from PCA-applied ensemble models.
- Along these models, we can conclude that `gbc_pca` is the most robust one because it has a balanced accuracy on both training and testing datasets compared to others in the same group.

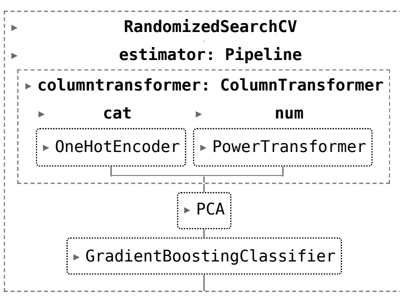



- Another metric we need to examine is **precision**. By leveraging precision, we can reduce the number of false positives, which may lead to higher costs.
- As shown in the figure below, we can again confirm the robustness of `gbc_pca` , as its metric values are consistently above 0.8.



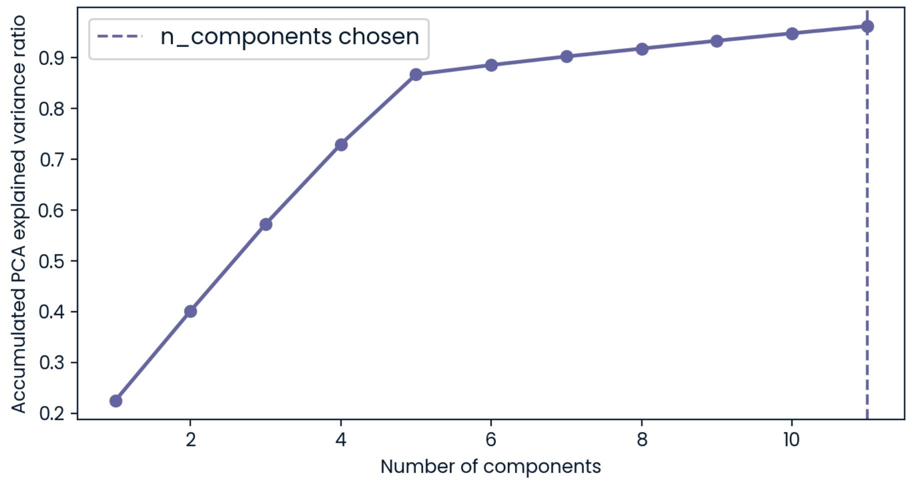

## 4. Post-processing

- Now, we extract the selected `gbc_pca` model and investigate further




- We can use `.best_score_` and `.best_params_` to access the score (i.e., F1-score) of the best model found during the random search, along with its associated hyperparameters.
- One point to clarify in the hyperparameter list is that the parameter `pca__n_components` refers to the amount of variance that needs to be explained, not the number of PCA components.
- In this case, 95% variance is choosen.

- The figure below shows the number of `PCA` components (i.e., 11) corresponding to 95% explained variance.
- We can see that the accumulated PCA explained variance ratio tends to plateau after 5 components.

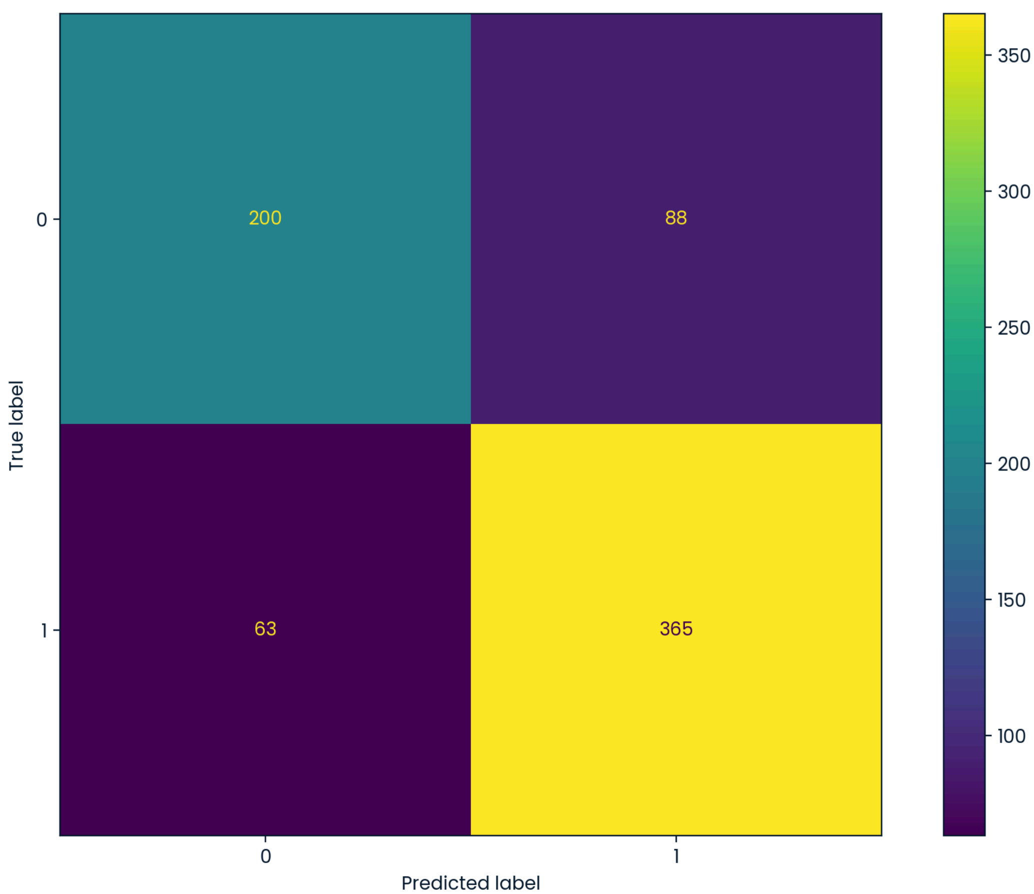



- Using `classification_report()` and `confusion_matrix()` are two effective ways to generate comprehensive reports on a classifier's performance using various metrics such as `precision`, `recall`, and `f1-score`.
- The report below shows the evaluation of the selected model on the training dataset.

```
train accuracy: 0.7891061452513967
train precision: 0.8057395143487859
                      precision    recall  f1-score   support

Not high traffic (0)       0.76      0.69      0.73       288
    High traffic (1)       0.81      0.85      0.83       428

            accuracy                           0.79       716
           macro avg       0.78      0.77      0.78       716
        weighted avg       0.79      0.79      0.79       716
```

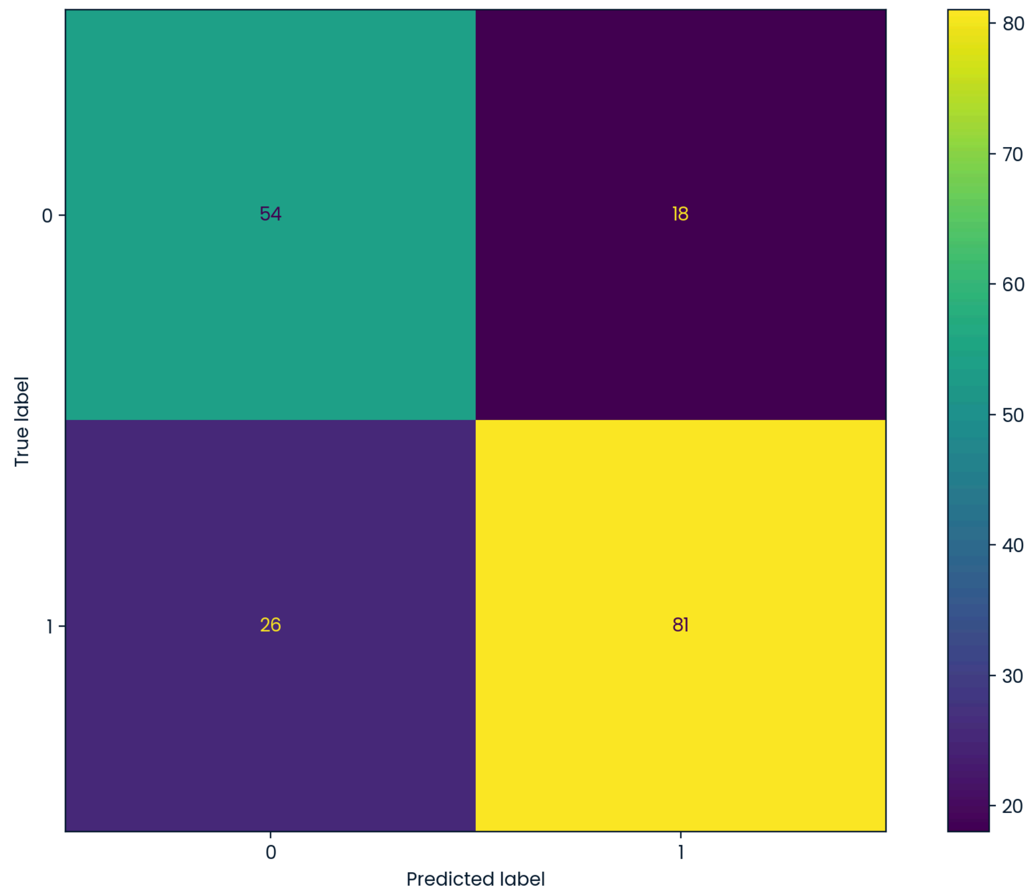


- Similarly, the below shows the evaluation of the selected model on the testing dataset.

```
train accuracy: 0.7541899441340782
train precision: 0.8181818181818182
                      precision    recall  f1-score   support

Not high traffic (0)       0.68      0.75      0.71        72
    High traffic (1)       0.82      0.76      0.79       107

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




## 5. Final Thoughts

## Summary:

Choose `gbc_pca` for deployment:

- Meet or exceed the 0.80 precision target.
- Low generalization gap → less overfitting.
- Acceptable test accuracy too.

# Other Recommendations

1. Threshold Tuning:

- Consider adjusting the classification threshold to maximize precision even further (e.g., using precision-recall curve).
- This can help drive ultra-conservative predictions, reducing false positives even more.

2. Post-Prediction Strategy:

- For recipes predicted to be high traffic, run a secondary check or business rule before promoting them (e.g., based on ingredient popularity, seasonality).

3. Monitoring and Retraining:

- Set up regular model evaluation, as user tastes and traffic patterns can change.

4. Use PCA?

- PCA is helping with precision here. So keep PCA in the pipeline, but monitor if it causes performance drift over time (e.g., as recipe trends shift).

### ✅ When you have finished...

- Publish your Workspace using the option on the left
- Check the published version of your report:
  - Can you see everything you want us to grade?
  - Are all the graphics visible?
- Review the grading rubric. Have you included everything that will be graded?
- Head back to the **Certification Dashboard** to submit your practical exam report and record your presentation