Complete Project PDF

Hanna Halfinger

DTSC-691

TABLE OF CONTENTS

Project Overview

DTSC691: Applied Data Science

Hanna Halfinger

Project Objective and Scope

This project aimed to develop a machine learning algorithm to predict the likelihood of developing allergies based on socioeconomic and lifestyle factors. The primary objective was to build an interactive web application using Flask, deployed on AWS, allowing users to input data and receive personalized predictions on allergy risk. The model explored key patterns and relationships within a comprehensive dataset to provide valuable insights into the determinants of allergies, assisting healthcare providers, researchers, and policymakers.

Significance: Allergies are a growing public health concern, with the U.S. Centers for Disease Control and Prevention (CDC) reporting a 50% increase in food allergy prevalence since the 1990s ("Digging up the Roots," 2023). By addressing socioeconomic and lifestyle factors, this project provides actionable insights to mitigate allergy prevalence and promote better public health outcomes.

Scope: The project covered the development and deployment of a machine learning model, user-interface integration, and exploratory analysis to uncover key correlations. It focused on analyzing dietary habits, physical activity, living conditions, smoking behaviors, and prior health history as predictors of allergies. Limitations included the lack of other types of factors that may influence allergies, such as genetic, environmental, or other medical factors. The region assessed was also limited to only European countries, meaning results may look different in other parts of the world.

Data Acquisition

The data source used for this project was the ESS11 from the European Social Survey, an organization conducting academically driven surveys in Europe measuring change over time in living conditions, social structure, public opinion and health conditions. This dataset included information collected from 31 European countries and addressed social conditions and indicators, social behavior and attitudes, general health and wellbeing, political ideology, religion and values, diet and nutrition, alcohol and smoking, and more (European Social Survey, 2024).

Information was collected from participants aged 15 and older from an hour-long face-to-face interview. The dataset included 22,190 participants and 534 features, which were originally narrowed down to 46, then reduced to 17 key features which were used for model training and deployment.

This data source was chosen because of the broad range of health and socioeconomic indicators covered, as well as the size of the dataset and substantial number of data points. The dataset was also organized in an adequate format that I found easy to work with. The diversity of features available allowed for a more comprehensive exploration of potential factors influencing allergies, which had the potential to improve the model's predictive capabilities and enable the identification of more insightful correlations.

The initial features chosen for this project were hand-picked based on various lifestyle and socioeconomic factors that I hypothesized to be potentially linked to allergies, and were assessed using a correlation matrix to exclude features with negligible scores or those considered irrelevant.

Exploratory Data Analysis

Comprehensive exploratory data analysis was conducted to uncover patterns and relationships within the dataset. I began by reviewing the survey documentation to understand its structure, types of variables, and their measurement scale, which included summarizing missing values and handling any inconsistencies. Next, I calculated summary statistics for each variable, including frequency counts, proportions, median and mode to assess the central tendency and variability for each indicator. I also conducted a chi-square test for each feature and a Spearman correlation matrix to identify significant relationships between the variables.

The visualizations included a range of charts to analyze and interpret the data effectively. Radar charts and bar charts were used to examine the distribution of individual features, while percent-stacked bar charts and boxplots illustrated the relationships between allergy cases and categorical variables. Additionally, a Cramer's V bar chart highlighted the strength of correlations and identified the most significant features. To further support the exploratory analysis, I created a table to show the percentage of individuals with allergies compared to those without for each category across all features.

During the exploration of this dataset, it was determined that addressing outliers was unnecessary, as the data consisted exclusively of nominal values derived from categorical survey responses.

The following features were hand-picked for analysis and initial model training:

1. etfruit - How often eat fruit, excluding drinking juice
2. eatveg - How often eat vegetables or salad, excluding potatoes
3. dosprt - Do sports or other physical activity, how many of last 7 days
4. cgtsmok - Cigarette smoking behavior
5. alcfreq - How often drink alcohol
6. trhltacu - Treatments used for own health, last 12 months: acupuncture
7. trhltacp - Treatments used for own health, last 12 months: acupressure
8. trhltcm - Treatments used for own health, last 12 months: chinese medicine
9. trhltch - Treatments used for own health, last 12 months: chiropractics
10. trhltos - Treatments used for own health, last 12 months: osteopathy
11. trhltho - Treatments used for own health, last 12 months: homeopathy
12. trhltht - Treatments used for own health, last 12 months: herbal treatment
13. trhlthy - Treatments used for own health, last 12 months: hypnotherapy
14. trhltmt - Treatments used for own health, last 12 months: massage therapy
15. trhltpt - Treatments used for own health, last 12 months: physiotherapy
16. trhltre - Treatments used for own health, last 12 months: reflexology
17. trhltsh - Treatments used for own health, last 12 months: spiritual healing
18. trhltnt - Treatments used for own health, last 12 months: none of these
19. hltprhc - Health problems, last 12 months: heart or circulation problem
20. hltprhb - Health problems, last 12 months: high blood pressure
21. hltprbp - Health problems, last 12 months: breathing problems
22. hltprbn - Health problems, last 12 months: back or neck pain
23. hltprpa - Health problems, last 12 months: muscular or joint pain in hand or arm
24. hltprpf - Health problems, last 12 months: muscular or joint pain in foot or leg
25. hltprsd - Health problems, last 12 months: stomach or digestion related
26. hltprsc - Health problems, last 12 months: skin condition related
27. hltprsh - Health problems, last 12 months: severe headaches
28. hltprdi - Health problems, last 12 months: diabetes
29. hltprnt - Health problems, last 12 months: none of these
30. happy - How happy are you
31. health - Subjective general health
32. hlthhmp - Hampered in daily activities by illness/disability/infirmity/mental problem
33. height - Height of respondent (cm)
34. weighta - Weight of respondent (kg)
35. jbexevh - In any job, ever exposed to: very hot temperatures
36. jbexevc - In any job, ever exposed to: very cold temperatures
37. jbexera - In any job, ever exposed to: radiation such as X-rays
38. jbexecp - In any job, ever exposed to: contact with chemical products, vapors, substances
39. jbexebs - In any job, ever exposed to: breathing in other types of smoke, fumes, powder, dust
40. domicil - Domicile, respondent's description (big city, suburbs, town, country village, etc.)
41. paccmoro - Problems with accommodation: mold or rot in windows, doors and floors

42. paccocrw - Problems with accommodation: overcrowding
43. paccxhoc - Problems with accommodation: extremely hot or extremely cold
44. paccinro - Problems with accommodation: presence of insects or rodents
45. isco08 - Occupation
46. nacer2 - Type of industry participant works in
47. hinctnta - Household's total net income, all sources

The following target label was selected:

1. hltpral - Health problems, last 12 months: allergies

Below is an example of a survey question and the corresponding category options that participants could select from:

Using this card, please tell me how often you eat fruit, excluding drinking juice?

Value	Category
1	Three times or more a day
2	Twice a day
3	Once a day
4	Less than once a day but at least 4 times a week
5	Less than 4 times a week but at least once a week
6	Less than once a week
7	Never
77	Refusal*
88	Don't know*
99	No answer*

Data Preparation and Cleaning

To prepare the data, I began by identifying missing values. After testing various imputation methods, I ultimately decided to impute missing values with the mode due to its simplicity and the best results in maintaining model performance. While testing alternative approaches, I observed that dropping all missing values significantly reduced model performance. I also experimented with K-Nearest Neighbors (KNN) imputation; however, the process proved too time-intensive given the size of the dataset.

Given the categorical nature of the data, there was no need to identify or address outliers or anomalies, as the dataset only included nominal values derived from categorical survey responses.

The majority of the features in the dataset were already represented in a numeric format suitable for machine learning models, so minimal data transformation was required. For nominal variables, I applied One-Hot encoding to ensure unbiased representation of all numeric categories. In some sampling techniques, such as SMOTE-NC, Ordinal Encoding was applied to ensure appropriate data formatting. A Standard Scaler was also applied to normalize the dataset for a consistent scale.

For feature engineering, I leveraged the feature importance scores generated by the best-performing Random Forest model. These scores identified the most impactful features contributing to prediction outcomes, enabling a more focused selection of variables. This approach enhanced the model's interpretability and efficiency by prioritizing the most significant predictors.

The original list of features was reduced to the following:

1. hltprnt - Health problems, last 12 months: none of these
2. chldhhe - Ever had children living in household
3. hltprbn - Health problems, last 12 months: back or neck pain
4. domicil - Domicile, respondent's description
5. hltprsc - Health problems, last 12 months: skin condition related
6. alcbnge - Frequency of binge drinking for men and women, last 12 months
7. cgtsmok - Cigarette smoking behavior
8. alcfreq - How often drink alcohol
9. dshltms - Discussed health, last 12 months: medical specialist
10. dosprt - Do sports or other physical activity, how many of last 7 days
11. health - Subjective general health
12. fnsdfml - Severe financial difficulties in family when growing up, how often
13. hltprbp - Health problems, last 12 months: breathing problems
14. eatveg - How often eat vegetables or salad, excluding potatoes
15. rlgdgr - How religious are you
16. hltprhb - Health problems, last 12 months: high blood pressure
17. happy - How happy are you

After preparing the dataset, I assessed the distribution of the target variable (likelihood of developing allergies) to check for class imbalance. I found a substantial imbalance between the two classes, with 13,590 data points for the positive class and only 1,943 cases for the negative class within the training set.

To address this imbalance, I experimented with several resampling techniques to improve the model’s performance on the minority class while preserving data integrity. The techniques included Random Oversampling, SMOTE (Synthetic Minority Over-sampling Technique), SMOTE-Tomek, SMOTE-NC, as well as ADASYN. These methods are described in further detail in the following section.

Model Training

The model training phase involved selecting and evaluating multiple machine learning algorithms to find the most effective model for predicting the likelihood of developing allergies. The following models and hyperparameters were used:

1. **Logistic Regression**: c, max_iter, solver, class_weight
2. **Random Forest Classifier**: n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features, class_weight
3. **Support Vector Classifier**: probability, c, kernel
4. **XGB Classifier**: n_estimators, learning_rate, max_depth, min_child_weight, gamma

These models were chosen to due their suitability for binary classification tasks, specifically predicting the likelihood of developing allergies. Each type of model is effective with a moderate number of features, making for easier interpretability and for identifying key patterns in the data. They are also relatively simple in design, making them appropriate for this dataset and predictive task.

To train each model, the dataset was divided into 70% training data and 30% testing data. Promising models were further optimized using Grid Search CV and Randomized Search CV. While K-fold cross-validation (with K set to 5 or 10) was initially considered to enhance model reliability, it was ultimately replaced by grid search techniques due to the time-intensive nature of combining cross-validation with other steps in the workflow.

Logistic Regression

The first version of the Logistic Regression model I trained used Random Oversampling. This technique selects samples from the minority class at random and duplicates them, then adds the duplicated samples back into the dataset. I used pandas' get_dummies to One-Hot encode the features that weren't ordinal, and then applied a Standard Scaler to the training and test sets. I set the max_iter to 5,000 to allow the model enough time to train, then fit the model to the training set.

The first version had the following classification report (which is analyzed in the next section).

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.66	0.78	5825
1	0.22	0.67	0.34	832
accuracy			0.67	6657
macro avg	0.58	0.67	0.56	6657
weighted avg	0.85	0.67	0.72	6657

ROC-AUC Score: 0.7437561901617695

The second version of the Logistic Regression model was trained using SMOTE-Tomek, which is an oversampling technique short for synthetic minority oversampling technique, and it works by generating synthetic samples for the minority class and adding them to the dataset. The new data points are created by interpolating between existing minority class examples and their nearest neighbors. Tomek links are also used in this technique, which removes pairs of neighboring samples that are in opposite classes, to remove noisy data and borderline samples. For this version, I went through all the same steps as before, and obtained the following classification report:

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.96	0.92	5825
1	0.30	0.12	0.18	832
accuracy			0.85	6657
macro avg	0.59	0.54	0.55	6657
weighted avg	0.81	0.85	0.83	6657

ROC-AUC Score: 0.7271776989105315

For the third version of the model, I used SMOTE-NC (Synthetic Minority Oversampling Technique for Nominal and Continuous data). This approach is similar to regular SMOTE, which interpolates between randomly chosen minority instances and their k-nearest neighbors for continuous data. However, for categorical data, SMOTE-NC assigns the most frequent category among the nearest neighbors to the synthetic data points. The method then combines the synthetic samples for both continuous and categorical features to generate a balanced dataset.

To accommodate the requirements of SMOTE-NC, I applied an Ordinal Encoder to ensure all features were formatted appropriately. I also implemented a Grid Search CV to optimize the model's hyperparameters, which took about 20 minutes to run. Specifically, I tested the following:

C: [0.01, 0.1, 1, 10]
Solver: ['liblinear', 'saga', 'newton-cg']
Class Weight: ['balanced'], to address class imbalances.

The Grid Search determined the best parameters to be:

C: 10

Solver: newton-cg.

I then retrained the model using these optimal parameters, resulting in the following classification performance:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.76      0.83      5796
           1       0.24      0.49      0.32       861

    accuracy                           0.73      6657
   macro avg       0.57      0.63      0.57      6657
weighted avg       0.82      0.73      0.76      6657
```

ROC-AUC Score: 0.7237740153207506

Random Forest Classifier

The next model I trained was a Random Forest Classifier. For the initial Random Forest, I decided to continue with the SMOTE-NC technique for oversampling, due to a more balanced accuracy and recall score for the last Logistic Regression model. I set the class_weight to balanced, and the n_jobs equal to -1. I then fit and trained the model on the scaled data, and got the results below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      5796
           1       0.30      0.20      0.24       861

    accuracy                           0.84      6657
   macro avg       0.59      0.57      0.57      6657
weighted avg       0.81      0.84      0.82      6657
```

AUC-ROC: 0.7504232162995987

Next, I trained the Random Forest Classifier using another oversampling technique known as ADASYN (Adaptive Synthetic Sampling). Similar to SMOTE, ADASYN generates synthetic data points for the minority class. However, it goes a step further by identifying minority class samples that are harder to classify. These samples are given higher weights, leading to the creation of more synthetic data points in areas where the model struggles most to differentiate between classes. I repeated the same process as above, first scaling the data using a Standard Scaler, putting the dataset through ADASYN, and then training and fitting the model on the scaled data. The following results are shown below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      5796
           1       0.59      0.07      0.13       861

    accuracy                           0.87      6657
   macro avg       0.73      0.53      0.53      6657
weighted avg       0.84      0.87      0.83      6657
```

AUC-ROC: 0.7842403427731408

I also performed a Randomized Search CV for the Random Forest model, testing the following:

N_estimators: 50, 100, 200, 300
Max_depth: None, 10, 20, 30
Min_samples_split: 2, 5, 10
Min_samples_leaf: 1, 2, 4
Max_features: sqrt, log2, None
Class_weight: None, balanced, balanced_subsample

I decided to try a few different class weights to see if one would perform better when paired with the oversampling method for addressing the class imbalance. The Randomized Search CV took my computer about 40 minutes to run, given the size of the dataset and number of features. It generated the following results for the best hyperparameters:

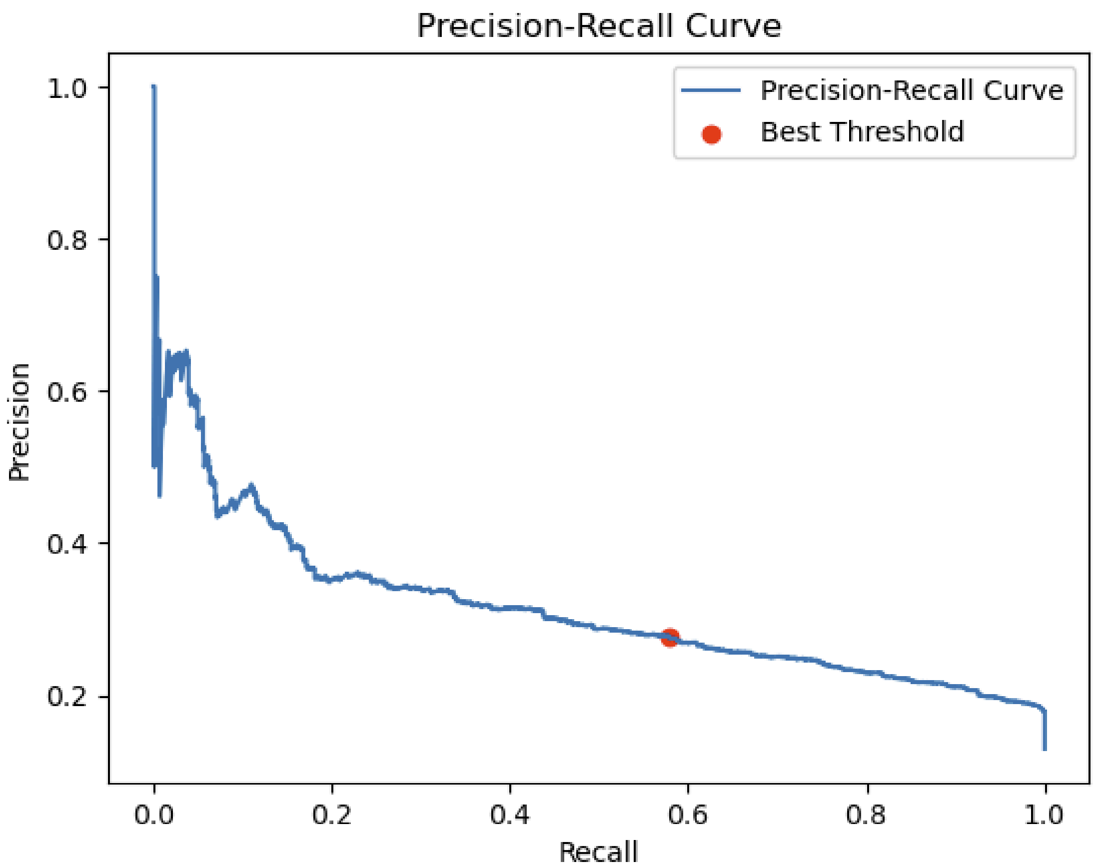
N_estimators: 200
Max_depth: 10
Min_samples_split: 2
Min_samples_leaf: 2
Max_features: sqrt
Class_weight: balanced_subsample

I trained my Random Forest Classifier on these hyperparameters and got the results below:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      5796
           1       0.34      0.29      0.31       861

    accuracy                           0.84      6657
   macro avg       0.62      0.60      0.61      6657
weighted avg       0.82      0.84      0.83      6657
```

AUC-ROC: 0.7761123655306356

Next, I decided to adjust the threshold for the model, to see if I could get the precision and recall more balanced. I also plotted the precision-recall curve, to give a better visual for where the threshold is. The optimal threshold turned out to be 0.4057. I adjusted the Random Forest to this threshold and obtained the following results:



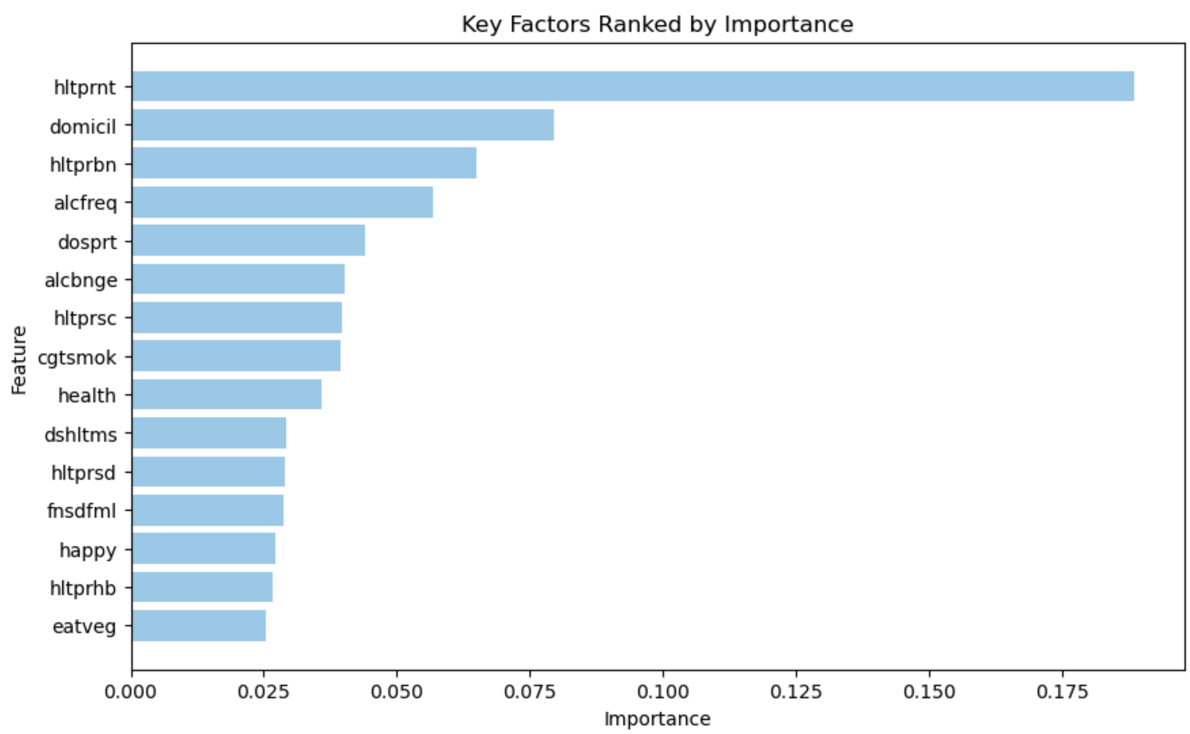
Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.93	0.78	0.84	5796
1	0.28	0.58	0.38	861
accuracy			0.75	6657
macro avg	0.60	0.68	0.61	6657
weighted avg	0.84	0.75	0.78	6657

ROC-AUC: 0.7761123655306356

Feature Engineering: Next, I printed the list of features considered most important by the Random Forest. I ended up using the first 15 as well as some of the next few listed in the top 20 as my features for my final model and in my web application. The scores for the first 15 are as follows:

	Feature	Importance
32	hltprnt	0.188452
47	domicil	0.079502
25	hltprbn	0.064863
4	alcfreq	0.056703
2	dosprt	0.044146
5	alcbnge	0.040126
29	hltprsc	0.039590
3	cgtsmok	0.039548
34	health	0.035814
7	dshltms	0.029138
28	hltprsd	0.029029
40	fnsdfml	0.028680
33	happy	0.027066
23	hltprhb	0.026792
1	eatveg	0.025307



I then retrained my Random Forest using the reduced list of features (see Data Preparation and Cleaning), which led to the following results:

Classification Report (ADASYN + Reduced Features):

precision recall f1-score support

0	0.91	0.81	0.86	3876
1	0.26	0.45	0.33	562
accuracy			0.77	4438
macro avg	0.58	0.63	0.59	4438
weighted avg	0.83	0.77	0.79	4438

ROC-AUC (ADASYN + Reduced Features): 0.7504

I also adjusted the threshold for this reduced model, and got the following scores:

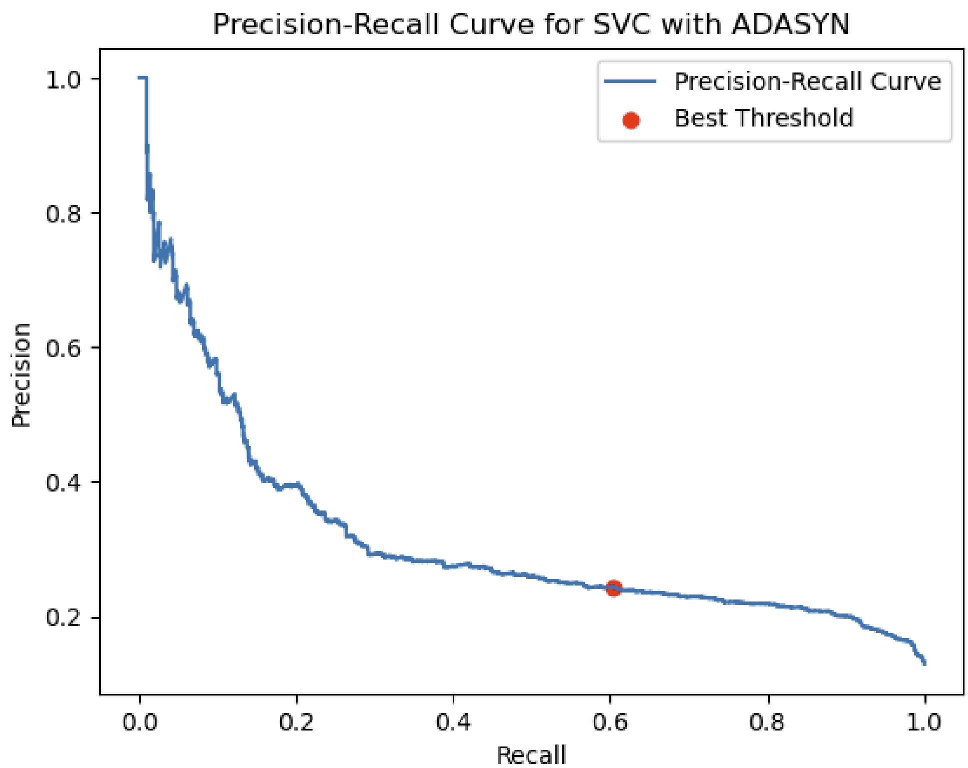
Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.94	0.65	0.77	3876
1	0.23	0.70	0.34	562
accuracy			0.66	4438
macro avg	0.58	0.68	0.56	4438
weighted avg	0.85	0.66	0.72	4438

ROC-AUC (based on probabilities): 0.7504

Support Vector Classifier

The third model I trained was a Support Vector Classifier, which I tested with ADASYN, and an adjusted threshold. I chose minority for the sampling_strategy, then fit and trained the model on the scaled data. I also set the probability to True, to allow the model to give probability scores for allergy cases, which was necessary for adjusting the precision-recall threshold. The optimal threshold was 0.0949, which I adjusted the model to. I then printed the results, which can be seen below:



Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.92	0.72	0.81	5796
1	0.24	0.60	0.35	861
accuracy			0.71	6657
macro avg	0.58	0.66	0.58	6657
weighted avg	0.84	0.71	0.75	6657

ROC-AUC: 0.74771619499691

Due to the model's moderate performance and time constraints, I only created one version of this model, and decided to move on to testing and creating my final model.

XGBoost Classifier

The final model that I trained and tested was the XGBoost Classifier. The first version of this model I trained using SMOTE for oversampling. I chose regular SMOTE over the other types due to its simplicity and supposed better compatibility with XGBoost, as XGBoost is already a complex model. This time, I split the data into an 80/20 training and test set. (I also experimented with 70/30, but found that 80/20 worked better for the XGBoost.) I applied the

Standard Scaler on this newly split data, then SMOTE, and created the model. I set the use_label_encoder to False, to avoid the model trying to re-encode my features, and set eval_metric equal to logloss, which optimizes the model for probability predictions. I then ran the model through a pipeline to merge the SMOTE and the classifier into one, and fit the pipeline to my training data, then trained the model. The following results were obtained:

Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.89	0.98	0.93	3876
1	0.51	0.15	0.24	562
accuracy			0.87	4438
macro avg	0.70	0.57	0.58	4438
weighted avg	0.84	0.87	0.84	4438

ROC-AUC: 0.7881598228352964

For the next version of my XGBoost model, I tried ADASYN for oversampling. I repeated the same steps as above, and obtained the following results:

Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.89	0.97	0.93	3876
1	0.45	0.16	0.24	562
accuracy			0.87	4438
macro avg	0.67	0.57	0.58	4438
weighted avg	0.83	0.87	0.84	4438

ROC-AUC: 0.7800861400937973

I then performed a Grid Search CV for this model, and tested the following hyperparameters:

Max_depth: 3, 5, 7
Learning_rate: 0.01, 0.1, 0.2
N_estimators: 50, 100, 200

The best parameters produced by the Grid Search CV were:

Max_depth: 7
Learning_rate: 0.1
N_estimators: 100

I retrained my model with these parameters, and the results were as follows:

Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.88	0.99	0.93	3876
1	0.49	0.06	0.10	562
accuracy			0.87	4438
macro avg	0.69	0.52	0.52	4438
weighted avg	0.83	0.87	0.83	4438

ROC-AUC: 0.7983374282471932

Another method I tried was setting the scale_pos_weight equal to 4, to see if combining oversampling with another method of handling class imbalances would improve performance. After adjusting these class weights, I obtained the following results:

Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.92	0.84	0.88	3876
1	0.32	0.53	0.40	562
accuracy			0.80	4438
macro avg	0.62	0.68	0.64	4438
weighted avg	0.85	0.80	0.82	4438

ROC-AUC: 0.7975579255864174

I ended up going with this model, and retrained it on the reduced list of 17 features that I picked out from the Random Forest generated list. After retraining the model on the reduced features, I got the following scores:

Classification Report at Adjusted Threshold:

	precision	recall	f1-score	support
0	0.91	0.80	0.85	3883
1	0.25	0.47	0.33	555
accuracy			0.76	4438
macro avg	0.58	0.63	0.59	4438
weighted avg	0.83	0.76	0.79	4438

ROC-AUC: 0.7541354901128273

Model Evaluation

Models	Version	Accuracy	Precision	Recall	F1-Score	ROC-AUC Score
Logistic Regression	Random Oversampling	0.67	0.22	0.67	0.34	0.7438
Logistic Regression	SMOTE-Tomek	0.85	0.30	0.12	0.18	0.7272
Logistic Regression	SMOTE-NC + Best Parameters	0.73	0.24	0.49	0.32	0.7238
Random Forest	SMOTE-NC	0.84	0.30	0.20	0.24	0.7504
Random Forest	ADASYN	0.87	0.59	0.07	0.13	0.7842
Random Forest	ADASYN + Best Parameters	0.84	0.34	0.29	0.31	0.7761
Random Forest	ADASYN + Best Parameters + Adjusted Threshold	0.75	0.28	0.58	0.38	0.7761
Random Forest	ADASYN + Reduced Features	0.77	0.26	0.45	0.33	0.7504
Random Forest	ADASYN + Reduced Features + Adjusted Threshold	0.66	0.23	0.70	0.34	0.7504
SVM Classifier	ADASYN + Adjusted Threshold	0.71	0.24	0.60	0.35	0.7477
XGBoost Classifier	SMOTE	0.87	0.51	0.15	0.24	0.7882
XGBoost Classifier	ADASYN	0.87	0.45	0.16	0.24	0.7801
XGBoost Classifier	ADASYN + Best Parameters	0.87	0.49	0.06	0.10	0.7983
XGBoost Classifier	ADASYN + Best Parameters + Scale_pos_weight	0.80	0.32	0.53	0.40	0.7976
XGBoost Classifier	ADASYN + Reduced Features + Scale_pos_weight	0.76	0.25	0.47	0.33	0.7541

Metrics Used for Assessment

The scores for each model are summarized in the table above. The primary objective was to identify the best-performing model for predicting and identifying allergy cases. Key metrics used to evaluate the models included precision, recall, ROC-AUC score, and the F1-score, with accuracy considered as a supplementary measure. While oversampling techniques helped address class imbalance during training, accuracy can still be misleading in such datasets. A high accuracy score may reflect a bias toward the majority class, which remains a concern when deploying the model on real-world data where the imbalance persists. Therefore, accuracy alone is not the most reliable indicator of model performance.

Precision was emphasized to assess how accurately the positive cases (allergy predictions) were identified, while recall measured the model's ability to capture all true positive cases within the dataset. Balancing and maximizing both precision and recall was a key focus, which is why thresholds were adjusted for some of the top-performing models to achieve better results.

The ROC-AUC score was another critical metric, offering an overall measure of the model's performance across all thresholds. It provided a more holistic view of the model's ability to distinguish between classes, making it particularly useful for comparing models and handling datasets with class imbalances like ours. This score is considered a reliable metric for evaluating discrimination ability and model performance.

The F1-score, as the harmonic mean of precision and recall, was also considered an important metric, as it reflects the balance between these two measures. Including the F1-score in the evaluation helped ensure that models performed well in capturing true positives while minimizing false positives.

Overall, the ROC-AUC score was prioritized as the most significant metric, followed by precision and recall, with the F1-score ranked third. Accuracy was considered the least important metric due to its potential for bias in imbalanced datasets, though it was still included for a comprehensive evaluation of the models.

Best Performing Models

The top-performing model selected for the web application was the XGBoost Classifier trained with ADASYN and scaled class weights, following feature reduction. This model was chosen due to its strong overall performance, particularly its balance between key metrics. It achieved the second-highest ROC-AUC score among all models at 0.7976, just slightly lower than the previous version of the same model, which had a score of 0.7983. However, the selected version significantly outperformed its predecessor in other critical areas, such as F1-score (0.40 vs. 0.10) and recall (0.53 vs. 0.06).

While its precision was slightly lower, the higher recall score made it the more favorable choice, as well as the significantly higher F1-score. Prioritizing recall ensures that fewer allergy cases are missed, and identifying as many true positives as possible is important for aiding in preventative measures. This trade-off aligned with the project's goal of maximizing the model's utility for predicting allergy risk effectively.

The next comparable model was the Random Forest Classifier trained with ADASYN and an adjusted threshold. Before feature reduction, this model achieved an ROC-AUC score of 0.7761, which was the second-highest among all models and third-highest across all versions tested. It also demonstrated a strong recall score of 0.58 and a comparable F1-score of 0.38.

However, it fell slightly short of the XGBoost model in overall performance. The Random Forest model had a lower accuracy, less balanced precision and recall, and a slightly lower ROC-AUC score. These differences carried over to its performance after feature reduction, solidifying it as a strong contender but ultimately not the top choice for the web application.

Moderately Performing Models

Random Forest Classifier (SMOTE-NC): The version trained with SMOTE-NC achieved an ROC-AUC of 0.7504, a precision of 0.34, and a recall of 0.29. While it showed balanced performance, it lacked the higher recall and precision seen in the ADASYN versions, making it less competitive.

Support Vector Classifier (ADASYN): This model achieved a reasonable ROC-AUC score of 0.7477 with a recall of 0.60 and a precision of 0.24 after adjusting the threshold. While its recall was comparable to the Random Forest ADASYN version, its precision and overall F1-score were slightly lower, indicating it struggled more with balancing predictions.

Logistic Regression (SMOTE-NC): With a recall of 0.49, precision of 0.24, and an ROC-AUC score of 0.7238, this version outperformed earlier Logistic Regression iterations. However, its overall balance across metrics was not as strong as the top-performing models, limiting its utility.

Lower Performing Models

Logistic Regression (SMOTE-Tomek): This version achieved an ROC-AUC of 0.7272 and a recall of 0.12. The precision was slightly higher at 0.30, but the recall was insufficient for identifying allergy cases effectively.

Random Forest Classifier (ADASYN + Default Threshold): While it had a relatively high ROC-AUC score of 0.7842, its recall of 0.07 and F1-score of 0.13 were among the lowest across all versions, making it a poor choice for detecting positive cases despite its good precision.

Logistic Regression (Random Oversampling): The first version of Logistic Regression had an ROC-AUC score of 0.7438 and a recall of 0.67, but its precision of 0.22 and overall balance of metrics were weaker than those of later iterations.

Worst Performing Model

XGBoost Classifier (ADASYN + Best Parameters): Although it achieved the highest ROC-AUC score (0.7983), its recall of 0.06 and F1-score of 0.10 severely limited its practical utility. This version was highly imbalanced in its predictions, making it unsuitable despite its excellent discrimination ability.

After the best model was selected, it was exported as a pickle file using Joblib, and deployed to be used in the web application.

User-Interface Integration

The model was deployed through a web application built using Flask and hosted on AWS via App Runner, with packaging handled through a Docker file. This application enables users to input socioeconomic and lifestyle factors into a form consisting of 17 questions corresponding to the 17 features selected during the feature engineering phase. These features were also used to train the final model. Upon submission, the application generates a personalized probability score, ranging from 0-100%, representing the user's likelihood of developing allergies based on their inputs. Additionally, the interface provides a clear summary explaining the score and highlights the factors influencing their risk level. A visualization is included to display these contributing features in order of importance, illustrating their magnitude of impact.

The following link can be used to access the web app:
https://nd2m9pys4b.us-east-1.awsapprunner.com/

Capstone Complexity

This project achieves complexity through the high number of features analyzed, number of predictive machine learning models used, as well as an interactive user interface deployed on AWS. The interface provides personalized allergy predictions using probability percentages, including a visualization ranking feature importances with corresponding descriptions. Additionally, a short slideshow presentation was included for introduction to the topic.

Software

The software used for this project was Python through Jupyter Notebook for handling data manipulation, exploration, as well as model training and analysis. Libraries included Pandas, Numpy, Scipy, and Scikit-Learn. For data visualization, Seaborn and Matplotlib were used to provide insights into data distributions, correlations, and model performance.

For the UI, Flask was used to build the web application that integrated the final machine learning model. Flask handled the backend processes, user inputs, model predictions, and rendering of results to the frontend. HTML and CSS were used to design the user interface. The application was created in VS code, containerized using Docker, then hosted on AWS using App Runner.

Presentation Plan

For the presentation portion, a comprehensive video walkthrough was created, showcasing the problem, solution, and methodology. The video included PowerPoint slides, a demonstration of the code and user interface, and was structured into the following key sections:

- **Introduction** - Overview of topic, problem being addressed, and project objectives
- **Data sources and preparation -** Walkthrough of data sources and code for data prep
- **Model development** - Code walkthrough for model development, training and selection
- **User interface demonstration -** Demonstration of user interface in use
- **Insights and outcomes** - Walkthrough of web app's predicted outcome and key features important for determining risk

Conclusion

One area of this project that I found challenging was the model training portion. The dataset proved to be fairly large, which made training the models and running all the cells take a substantial amount of time. To run the entire notebook of code all the way through took about an hour and a half, even after extra data cleaning, and I was unsure whether to reduce the size of the dataset, or how much of a negative impact a smaller dataset may have on the models' outcome. I wanted to maximize potential performance by providing a large enough dataset, so I decided to keep it at a size I believed was adequate.

I also felt that more model exploration could have been done to further refine performance, had time permitted. Retrospectively, it may be better to first perform feature engineering as a separate step prior to model training and to fully reduce features before training each model, then comparing model results. This would likely allow for faster training and possibly better results as well, since the final model's performance would be the one being analyzed, not the models' before feature reduction. On the other hand, I chose to perform feature engineering on the basis of the best Random Forest's performance and the features it selected as most important, so this approach didn't allow for the steps to be easily reversed. With this approach, I figured the results created by the model may be more accurate than other calculations, though that may not actually be the case.

Another area I struggled with was deploying the model on AWS. Originally, I tried using Elastic Beanstalk, but had difficulty getting it to work, as it automatically resorted to using Launch Configuration for its setup, which was outdated and deprecated by AWS. Amazon now requires the use of Launch Templates, which I tried to set up with additional corresponding files, but the persistent error messages led me to choose a different option altogether. I tried App Runner next, using Docker to containerize the project, and found the process much quicker and smoother.

The final area I surprisingly found a bit challenging was fitting everything into the video presentation. I planned on walking through the front and backend of the code used for developing the web app and the process of deploying it on AWS, in addition to the code created in Jupyter Notebook for the data cleaning, visualizations, and model training. Even excluding the Flask code in the video however, still put me slightly above the 30 minute time limit, and I had to trim some parts of the video to make it fit within the time frame.

Overall, this project provided valuable learning experiences and practical insights into building and deploying a machine learning model and developing a fully functional web application. I felt that the project objectives and depth gave me critical opportunities to refine my technical skills, learn new ways to handle challenges, and obtain a clearer understanding of end-to-end project development, which will be essential in my future career.

Resources

"Allergies Are Getting More Common. Playing in the Dirt Could Help." Memorialhermann, 28 July 2023, memorialhermann.org/health-wellness/health/allergies-getting-more-common.

Bloom, Dave. "Private Insurance Claims Related to Anaphylaxis from Food Allergy Have Nearly Quadrupled since 2007." SnackSafely.Com, 21 Aug. 2017, snacksafely.com/2017/08/private-insurance-claims-related-to-food-allergy-induced-anaphylaxis-have-nearly-quadrupled-since-2007/.

Davies, Dave. "Why Our Allergies Are Getting Worse -and What to Do about It." NPR, NPR, 30 May 2023, www.npr.org/sections/health-shots/2023/05/30/1178433166/theresa-macphail-allergic-allergies.

"Digging up the Roots of Food Allergies." National Institutes of Health, U.S. Department of Health and Human Services, irp.nih.gov/blog/post/2023/05/digging-up-the-roots-of-food-allergies#:~:text=If%20it%20se ems%20like%20food,a%20serious%20public%20health%20concern. Accessed 31 Oct. 2024.

European Social Survey European Research Infrastructure (ESS ERIC). (2024). ESS11 Data Documentation. Sikt - Norwegian Agency for Shared Services in Education and Research. https://doi.org/10.21338/ess11-2023

Final Capstone Project Code

December 12, 2024

1 Final Capstone Project Code

2 Importing Dataset

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
cd desktop
```

```
[WinError 2] The system cannot find the file specified: 'desktop'
C:\Users\Emilia\Desktop\Capstone Project
```

[3]:
```
cd desktop\Capstone Project
```

```
[WinError 3] The system cannot find the path specified: 'desktop\\Capstone
Project'
C:\Users\Emilia\Desktop\Capstone Project
```

[4]:
```
cd ESS11
```

```
C:\Users\Emilia\Desktop\Capstone Project\ESS11
```

[5]:
```
df = pd.read_csv('ESS11.csv')
```

```
C:\Users\Emilia\AppData\Local\Temp\ipykernel_9752\70637920.py:1: DtypeWarning:
Columns (548) have mixed types. Specify dtype option on import or set
low_memory=False.
  df = pd.read_csv('ESS11.csv')
```

3 Data Cleaning & Exploration

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22190 entries, 0 to 22189
Columns: 558 entries, name to psu
dtypes: float64(142), int64(392), object(24)
memory usage: 94.5+ MB
```

[7]:
```python
df.describe()
```

[7]:
```
       essround  edition          idno       dweight       pweight  \
count   22190.0  22190.0  22190.000000  22190.000000  22190.000000
mean       11.0      1.0  68365.483912      1.000000      0.868026
std         0.0      0.0  10531.037907      0.334933      1.094132
min        11.0      1.0  50003.000000      0.086694      0.144186
25%        11.0      1.0  59224.000000      0.906116      0.211267
50%        11.0      1.0  68428.500000      1.000000      0.330915
75%        11.0      1.0  77588.750000      1.026732      0.889906
max        11.0      1.0  86480.000000      4.001857      3.324269

             nwspol      netusoft       netustm       ppltrst       pplfair  \
count  22190.000000  22190.000000  22190.000000  22190.000000  22190.000000
mean     179.455385      4.250293   1607.815683      5.579630      6.363091
std      867.165476      1.388776   2674.494402      4.417469      5.937601
min        0.000000      1.000000      0.000000      0.000000      0.000000
25%       30.000000      4.000000    120.000000      4.000000      5.000000
50%       60.000000      5.000000    240.000000      6.000000      6.000000
75%       90.000000      5.000000    600.000000      7.000000      8.000000
max     9999.000000      9.000000   9999.000000     99.000000     99.000000

       …       symtc19      symtnc19        vacc19         recon  \
count  …  22190.000000   22190.00000  22190.000000  22190.000000
mean   …      3.306084       5.16877      1.203560      4.517801
std    …      2.132578       1.74266      0.624781      3.684603
min    …      1.000000       1.00000      1.000000      1.000000
25%    …      2.000000       6.00000      1.000000      1.000000
50%    …      2.000000       6.00000      1.000000      2.000000
75%    …      6.000000       6.00000      1.000000      9.000000
max    …      9.000000       9.00000      9.000000      9.000000

              inwtm          mode        domain          prob       stratum  \
count  21956.000000  22190.000000  11262.000000  22190.000000  22190.000000
mean      56.851476      1.065300      1.679275      0.000757    496.070437
std       23.097540      0.348725      0.538009      0.000563    271.796266
min        4.000000      1.000000      1.000000      0.000028      1.000000
25%       43.000000      1.000000      1.000000      0.000342    193.000000
50%       54.000000      1.000000      2.000000      0.000735    534.000000
75%       66.000000      1.000000      2.000000      0.001046    716.000000
max      788.000000      9.000000      3.000000      0.008759    877.000000

                psu
count  22190.000000
mean    5729.373276
std     3543.031610
min        1.000000
```

```
25%     2371.000000
50%     5628.500000
75%     8268.750000
max    11887.000000

[8 rows x 534 columns]
```

[9]:
```
df.dtypes
```

[9]:
```
name          object
essround       int64
edition        int64
proddate      object
idno           int64
               …
mode           int64
domain       float64
prob         float64
stratum        int64
psu            int64
Length: 558, dtype: object
```

[10]:
```
df.head(25)
```

[10]:
```
        name  essround  edition    proddate   idno cntry   dweight   pweight  \
0   ESS11e01        11        1  20.06.2024  50014    AT  1.185115  0.330915
1   ESS11e01        11        1  20.06.2024  50030    AT  0.609898  0.330915
2   ESS11e01        11        1  20.06.2024  50057    AT  1.392330  0.330915
3   ESS11e01        11        1  20.06.2024  50106    AT  0.556061  0.330915
4   ESS11e01        11        1  20.06.2024  50145    AT  0.722795  0.330915
5   ESS11e01        11        1  20.06.2024  50158    AT  0.992605  0.330915
6   ESS11e01        11        1  20.06.2024  50211    AT  0.540318  0.330915
7   ESS11e01        11        1  20.06.2024  50212    AT  0.814622  0.330915
8   ESS11e01        11        1  20.06.2024  50213    AT  1.364956  0.330915
9   ESS11e01        11        1  20.06.2024  50235    AT  0.872949  0.330915
10  ESS11e01        11        1  20.06.2024  50236    AT  0.698986  0.330915
11  ESS11e01        11        1  20.06.2024  50238    AT  0.832037  0.330915
12  ESS11e01        11        1  20.06.2024  50240    AT  0.873454  0.330915
13  ESS11e01        11        1  20.06.2024  50248    AT  0.902551  0.330915
14  ESS11e01        11        1  20.06.2024  50270    AT  0.581818  0.330915
15  ESS11e01        11        1  20.06.2024  50281    AT  0.693522  0.330915
16  ESS11e01        11        1  20.06.2024  50310    AT  2.324511  0.330915
17  ESS11e01        11        1  20.06.2024  50311    AT  0.992605  0.330915
18  ESS11e01        11        1  20.06.2024  50324    AT  1.070298  0.330915
19  ESS11e01        11        1  20.06.2024  50339    AT  0.556061  0.330915
20  ESS11e01        11        1  20.06.2024  50341    AT  0.482949  0.330915
21  ESS11e01        11        1  20.06.2024  50349    AT  0.992605  0.330915
```

22	ESS11e01	11	1	20.06.2024	50355	AT	0.736608	0.330915
23	ESS11e01	11	1	20.06.2024	50358	AT	1.950199	0.330915
24	ESS11e01	11	1	20.06.2024	50363	AT	1.956047	0.330915

	nwspol	netusoft	…	rinwe	inwde	jinws	\
0	90	5	…	NaN	11/12/2023 15:26	11/12/2023 15:21	
1	90	5	…	NaN	10/18/2023 10:44	10/18/2023 10:42	
2	30	5	…	NaN	9/30/2023 14:13	9/30/2023 14:08	
3	15	1	…	NaN	6/30/2023 15:11	6/30/2023 15:08	
4	60	5	…	NaN	7/11/2023 11:14	7/11/2023 11:10	
5	120	5	…	NaN	10/16/2023 9:42	10/16/2023 9:38	
6	45	1	…	NaN	10/20/2023 18:46	10/20/2023 18:44	
7	120	5	…	NaN	10/26/2023 16:29	10/26/2023 16:19	
8	20	5	…	NaN	9/9/2023 15:59	9/9/2023 15:57	
9	120	1	…	NaN	10/8/2023 12:00	10/8/2023 11:57	
10	60	5	…	NaN	11/3/2023 12:21	11/3/2023 12:17	
11	5	5	…	NaN	10/16/2023 10:46	10/16/2023 10:42	
12	60	5	…	NaN	9/30/2023 11:08	9/30/2023 11:06	
13	90	1	…	NaN	11/11/2023 11:01	11/11/2023 11:01	
14	60	2	…	NaN	9/28/2023 10:20	9/28/2023 10:18	
15	180	5	…	NaN	10/20/2023 13:22	10/20/2023 13:17	
16	20	5	…	NaN	11/11/2023 18:36	11/11/2023 18:33	
17	45	3	…	NaN	7/15/2023 20:48	7/15/2023 20:46	
18	5	5	…	NaN	11/6/2023 15:56	11/6/2023 15:54	
19	30	4	…	NaN	10/21/2023 11:46	10/21/2023 11:43	
20	90	2	…	NaN	7/14/2023 17:55	7/14/2023 17:53	
21	60	5	…	NaN	9/16/2023 15:08	9/16/2023 15:02	
22	30	5	…	NaN	11/19/2023 10:04	11/19/2023 10:02	
23	60	5	…	NaN	9/9/2023 12:34	9/9/2023 12:30	
24	30	5	…	NaN	9/1/2023 18:05	9/1/2023 18:03	

	jinwe	inwtm	mode	domain	prob	stratum	psu
0	11/12/2023 15:26	30.0	1	2.0	0.000579	107	317
1	10/18/2023 10:44	40.0	1	1.0	0.001124	69	128
2	9/30/2023 14:13	42.0	1	2.0	0.000493	18	418
3	6/30/2023 15:11	34.0	1	1.0	0.001233	101	295
4	7/11/2023 11:14	57.0	1	2.0	0.000949	115	344
5	10/16/2023 9:42	57.0	1	2.0	0.000691	7	373
6	10/20/2023 18:46	25.0	1	2.0	0.001269	58	86
7	10/26/2023 16:29	68.0	1	2.0	0.000842	38	3
8	9/9/2023 15:59	40.0	1	2.0	0.000502	62	108
9	10/8/2023 12:00	36.0	1	2.0	0.000786	105	314
10	11/3/2023 12:21	24.0	1	3.0	0.000981	26	622
11	10/16/2023 10:46	43.0	1	2.0	0.000824	94	242
12	9/30/2023 11:08	47.0	1	2.0	0.000785	100	270
13	11/11/2023 11:01	41.0	1	2.0	0.000760	98	255
14	9/28/2023 10:20	46.0	1	1.0	0.001179	5	368

```
15  10/20/2023 13:22   39.0      1     3.0  0.000989       35  783
16  11/11/2023 18:36   66.0      1     2.0  0.000295       75  165
17   7/15/2023 20:48   78.0      1     2.0  0.000691        7  373
18   11/6/2023 15:56   40.0      1     2.0  0.000641       56   73
19  10/21/2023 11:46   51.0      1     1.0  0.001233      101  283
20   7/14/2023 17:55   56.0      1     2.0  0.001420       17  414
21   9/16/2023 15:08   43.0      1     2.0  0.000691        7  373
22  11/19/2023 10:04   44.0      1     3.0  0.000931       21  610
23    9/9/2023 12:34   47.0      1     2.0  0.000352       23  429
24    9/1/2023 18:05   28.0      1     2.0  0.000351       45   31

[25 rows x 558 columns]
```

[11]:
```
# Select 47 features of the dataset to keep for model training

columns_to_keep = [
    'etfruit', 'eatveg', 'dosprt', 'cgtsmok', 'alcfreq', 'alcbnge',
    'dshltgp', 'dshltms', 'dshltnt', 'trhltacu', 'trhltacp', 'trhltcm',
    'trhltch', 'trhltos', 'trhltho', 'trhltht', 'trhlthy', 'trhltmt',
    'trhltpt', 'trhltre', 'trhltsh', 'trhltnt', 'hltprhc', 'hltpral', 'hltprhb',
    'hltprbp', 'hltprbn', 'hltprpa', 'hltprpf', 'hltprsd', 'hltprsc',
    'hltprsh', 'hltprdi', 'hltprnt', 'happy', 'health', 'hlthhmp',
    'rlgdgr', 'pray', 'height', 'weighta', 'fnsdfml', 'jbexevh',
    'jbexevc', 'jbexera', 'jbexecp', 'jbexebs', 'chldhhe', 'domicil',
    'paccmoro', 'paccocrw', 'paccxhoc', 'paccinro', 'isco08',
    'nacer2', 'hinctnta']
```

[12]:
```
df_cleaned = df[columns_to_keep]
```

[13]:
```
df_cleaned.head(25)
```

[13]:
```
    etfruit  eatveg  dosprt  cgtsmok  alcfreq  alcbnge  dshltgp  dshltms  \
0         3       3       3        4        3        5        1        1
1         1       1       5        5        3        2        1        0
2         4       3       3        1        4        5        1        1
3         2       2       3        6        7        6        1        1
4         3       3       3        1        2        4        0        1
5         5       3       4        5        2        2        1        1
6         4      99       4        6        7        6        1        1
7         3       2       2        4        3        5        1        1
8         2       3       2        5        5        5        1        1
9         5       3       1        4        7        6        1        1
10        3       3       5        5        4        5        1        1
11        2       2       5        6        4        4        1        0
12        4       2       3        1        7        6        1        0
13        3       3       2        6        4        5        1        1
14        1       1      77        3       77        4        1        1
```

15	3	3	5	4	7	6	1	1
16	3	3	1	6	3	5	1	1
17	3	4	4	6	4	4	1	1
18	4	3	7	6	4	5	1	1
19	2	2	7	4	2	2	1	0
20	3	3	5	6	2	5	1	1
21	3	2	7	6	5	5	0	1
22	5	5	4	1	2	2	0	0
23	2	3	5	6	2	4	1	0
24	3	4	3	5	4	5	0	1

	dshltnt	trhltacu	…	jbexebs	chldhhe	domicil	paccmoro	paccocrw	\
0	0	0	…	0	1	3	0	0	
1	0	0	…	0	2	1	1	0	
2	0	0	…	0	6	3	0	0	
3	0	0	…	1	1	1	0	0	
4	0	0	…	0	1	4	0	0	
5	0	0	…	0	1	4	0	0	
6	0	0	…	1	1	3	0	0	
7	0	0	…	0	1	4	0	0	
8	0	0	…	0	6	4	0	0	
9	0	0	…	0	1	4	0	0	
10	0	0	…	0	2	1	0	0	
11	0	0	…	1	2	3	1	0	
12	0	0	…	0	1	1	0	0	
13	0	0	…	0	1	5	0	0	
14	0	1	…	0	2	1	0	0	
15	0	0	…	0	1	1	0	0	
16	0	0	…	1	6	4	0	0	
17	0	0	…	0	1	3	0	0	
18	0	0	…	0	2	4	0	0	
19	0	0	…	0	1	2	0	0	
20	0	0	…	0	1	4	0	0	
21	0	0	…	0	1	5	0	0	
22	1	0	…	0	2	1	0	0	
23	0	0	…	0	6	4	0	0	
24	0	0	…	0	6	3	0	0	

	paccxhoc	paccinro	isco08	nacer2	hinctnta
0	0	0	66666	666	6
1	0	0	5249	14	1
2	0	0	2635	88	5
3	0	0	2221	87	2
4	0	0	5223	47	77
5	0	0	1112	21	9
6	0	0	8219	25	3
7	0	0	2330	85	10

8	0	0	2221	85	8
9	0	0	5249	10	3
10	0	0	4226	46	4
11	0	0	9629	25	6
12	0	0	99999	90	77
13	0	0	6130	1	77
14	0	0	2642	63	8
15	0	0	3312	64	5
16	0	0	2422	25	8
17	0	0	5311	14	6
18	0	0	3513	63	88
19	0	0	5131	56	1
20	0	0	66666	666	2
21	0	0	5223	46	2
22	0	0	5131	55	2
23	0	0	4312	72	88
24	0	0	3359	85	8

[25 rows x 56 columns]

[14]:
```
df_cleaned.describe()
```

[14]:

	etfruit	eatveg	dosprt	cgtsmok	alcfreq	\
count	22190.000000	22190.000000	22190.000000	22190.000000	22190.000000	
mean	3.387247	3.389860	4.365119	4.511086	4.854078	
std	3.831477	4.684408	8.403213	1.749884	4.953001	
min	1.000000	1.000000	0.000000	1.000000	1.000000	
25%	2.000000	3.000000	1.000000	4.000000	3.000000	
50%	3.000000	3.000000	3.000000	5.000000	5.000000	
75%	4.000000	4.000000	7.000000	6.000000	7.000000	
max	99.000000	99.000000	99.000000	9.000000	99.000000	

	alcbnge	dshltgp	dshltms	dshltnt	trhltacu	\
count	22190.000000	22190.000000	22190.000000	22190.000000	22190.00000	
mean	4.336503	0.745246	0.441550	0.182830	0.03114	
std	1.387614	0.435733	0.496583	0.386536	0.17370	
min	1.000000	0.000000	0.000000	0.000000	0.00000	
25%	3.000000	0.000000	0.000000	0.000000	0.00000	
50%	5.000000	1.000000	0.000000	0.000000	0.00000	
75%	5.000000	1.000000	1.000000	0.000000	0.00000	
max	9.000000	1.000000	1.000000	1.000000	1.00000	

	…	jbexebs	chldhhe	domicil	paccmoro	\
count	…	22190.000000	22190.000000	22190.000000	22190.000000	
mean	…	0.154845	2.896485	3.022983	0.052231	
std	…	0.361765	2.165649	1.233449	0.222497	
min	…	0.000000	1.000000	1.000000	0.000000	

```
25%    …       0.000000       1.000000       2.000000       0.000000
50%    …       0.000000       2.000000       3.000000       0.000000
75%    …       0.000000       6.000000       4.000000       0.000000
max    …       1.000000       9.000000       9.000000       1.000000

           paccocrw      paccxhoc      paccinro        isco08        nacer2  \
count  22190.000000  22190.000000  22190.000000  22190.000000  22190.000000
mean       0.014917      0.034250      0.036773  10462.581703    133.852636
std        0.121222      0.181874      0.188209  19669.034152    228.610905
min        0.000000      0.000000      0.000000    110.000000      1.000000
25%        0.000000      0.000000      0.000000   2522.000000     43.000000
50%        0.000000      0.000000      0.000000   5112.000000     64.000000
75%        0.000000      0.000000      0.000000   7480.500000     86.000000
max        1.000000      1.000000      1.000000  99999.000000    999.000000

           hinctnta
count  22190.000000
mean      18.996755
std       28.954341
min        1.000000
25%        4.000000
50%        7.000000
75%       10.000000
max       99.000000

[8 rows x 56 columns]
```

[15]:

```python
# Define the mapping for values to be replaced with NaN
general_nan_values = {77: np.nan, 88: np.nan, 99: np.nan, 8: np.nan, 9: np.nan,
 ↪666: np.nan,
                      777: np.nan, 888: np.nan, 999: np.nan, 66666: np.nan,
 ↪77777: np.nan,
                      88888: np.nan, 99999: np.nan}

# Define exceptions for columns where specific values shouldn't be removed
exception_columns = ['hinctnta', 'happy', 'rlgdgr', 'height', 'weight',
 ↪'nacer2']

# Columns to remove '7'
remove_7_columns = ['cgtsmok', 'health', 'hlthhmp', 'fnsdfml', 'domicil']

# Columns to remove '6' and '7'
remove_6_7_columns = ['alcbnge', 'chldhhe']

# Apply general cleaning to all columns except exceptions
for col in df_cleaned.columns:
    if col not in exception_columns:
```

```
        df_cleaned.loc[:, col] = df_cleaned[col].replace(general_nan_values)

# Handle exception columns (preserve 8 and 9)
for col in exception_columns:
    if col in df_cleaned.columns:
        df_cleaned.loc[:, col] = df_cleaned[col].replace({
            77: np.nan, 88: np.nan, 99: np.nan,
            666: np.nan, 777: np.nan, 888: np.nan, 999: np.nan,
            66666: np.nan, 77777: np.nan, 88888: np.nan, 99999: np.nan
        })  # Keep 8 and 9 intact

# Handle columns where '7' should also be removed
for col in remove_7_columns:
    if col in df_cleaned.columns:
        df_cleaned.loc[:, col] = df_cleaned[col].replace({7: np.nan})

# Handle columns where '6' and '7' should be removed
for col in remove_6_7_columns:
    if col in df_cleaned.columns:
        df_cleaned.loc[:, col] = df_cleaned[col].replace({6: np.nan, 7: np.nan})
```

[16]:
```
print(df_cleaned.dtypes)
```

```
etfruit     float64
eatveg      float64
dosprt      float64
cgtsmok     float64
alcfreq     float64
alcbnge     float64
dshltgp       int64
dshltms       int64
dshltnt       int64
trhltacu      int64
trhltacp      int64
trhltcm       int64
trhltch       int64
trhltos       int64
trhltho       int64
trhltht       int64
trhlthy       int64
trhltmt       int64
trhltpt       int64
trhltre       int64
trhltsh       int64
trhltnt       int64
hltprhc       int64
hltpral       int64
hltprhb       int64
```

```
hltprbp       int64
hltprbn       int64
hltprpa       int64
hltprpf       int64
hltprsd       int64
hltprsc       int64
hltprsh       int64
hltprdi       int64
hltprnt       int64
happy       float64
health      float64
hlthhmp     float64
rlgdgr      float64
pray        float64
height      float64
weighta     float64
fnsdfml     float64
jbexevh       int64
jbexevc       int64
jbexera       int64
jbexecp       int64
jbexebs       int64
chldhhe     float64
domicil     float64
paccmoro      int64
paccocrw      int64
paccxhoc      int64
paccinro      int64
isco08      float64
nacer2      float64
hinctnta    float64
dtype: object
```

[17]:
```python
# See all null values
df_cleaned.isnull().sum()
```

[17]:
```
etfruit       39
eatveg        57
dosprt       213
cgtsmok       47
alcfreq       78
alcbnge     5125
dshltgp        0
dshltms        0
dshltnt        0
trhltacu       0
trhltacp       0
```

```
trhltcm        0
trhltch        0
trhltos        0
trhltho        0
trhltht        0
trhlthy        0
trhltmt        0
trhltpt        0
trhltre        0
trhltsh        0
trhltnt        0
hltprhc        0
hltpral        0
hltprhb        0
hltprbp        0
hltprbn        0
hltprpa        0
hltprpf        0
hltprsd        0
hltprsc        0
hltprsh        0
hltprdi        0
hltprnt        0
happy         70
health        24
hlthhmp       68
rlgdgr       140
pray         451
height       558
weighta     1805
fnsdfml      296
jbexevh        0
jbexevc        0
jbexera        0
jbexecp        0
jbexebs        0
chldhhe     6944
domicil       41
paccmoro       0
paccocrw       0
paccxhoc       0
paccinro       0
isco08      1856
nacer2      2975
hinctnta    3984
dtype: int64
```

[18]:
```python
# Replace all null values with mode
for column in df_cleaned.columns:
    if df_cleaned[column].isna().sum() > 0:  # Check if the column has missing
 ↪values
        mode_value = df_cleaned[column].mode()[0]  # Get the mode of the column
        df_cleaned[column].fillna(mode_value, inplace=True)  # Fill missing
 ↪values with the mode
```

[19]:
```python
# No more null values
df_cleaned.isnull().sum()
```

[19]:
```
etfruit     0
eatveg      0
dosprt      0
cgtsmok     0
alcfreq     0
alcbnge     0
dshltgp     0
dshltms     0
dshltnt     0
trhltacu    0
trhltacp    0
trhltcm     0
trhltch     0
trhltos     0
trhltho     0
trhltht     0
trhlthy     0
trhltmt     0
trhltpt     0
trhltre     0
trhltsh     0
trhltnt     0
hltprhc     0
hltpral     0
hltprhb     0
hltprbp     0
hltprbn     0
hltprpa     0
hltprpf     0
hltprsd     0
hltprsc     0
hltprsh     0
hltprdi     0
hltprnt     0
happy       0
health      0
```

```
hlthhmp     0
rlgdgr      0
pray        0
height      0
weighta     0
fnsdfml     0
jbexevh     0
jbexevc     0
jbexera     0
jbexecp     0
jbexebs     0
chldhhe     0
domicil     0
paccmoro    0
paccocrw    0
paccxhoc    0
paccinro    0
isco08      0
nacer2      0
hinctnta    0
dtype: int64
```

Next, a Spearman correlation was run. This type of correlation method was chosen as it works better with categorical data, whereas a traditional Pearson correlation matrix works better with continuous/numeric data.

[20]:
```
df_cleaned.corr(method='spearman')
```

[20]:
```
           etfruit    eatveg    dosprt   cgtsmok   alcfreq   alcbnge  \
etfruit   1.000000  0.500002 -0.157651 -0.139357 -0.028294 -0.065702
eatveg    0.500002  1.000000 -0.167929 -0.080705  0.043639  0.010882
dosprt   -0.157651 -0.167929  1.000000  0.030287 -0.083396 -0.050668
cgtsmok  -0.139357 -0.080705  0.030287  1.000000  0.220329  0.208332
alcfreq  -0.028294  0.043639 -0.083396  0.220329  1.000000  0.619598
alcbnge  -0.065702  0.010882 -0.050668  0.208332  0.619598  1.000000
dshltgp  -0.038094 -0.039290 -0.051196 -0.014411 -0.014164  0.007936
dshltms  -0.051405 -0.045035 -0.021486 -0.034467 -0.045518  0.048035
dshltnt   0.048873  0.056970  0.036106  0.014889  0.030329 -0.007142
trhltacu -0.034777 -0.050927  0.019255 -0.000942 -0.022068 -0.025446
trhltacp -0.009431 -0.020355  0.019690 -0.009875 -0.006031 -0.006618
trhltcm  -0.019240 -0.045756  0.027815 -0.012071 -0.005279 -0.002855
trhltch  -0.033324 -0.029929  0.012340 -0.006920 -0.047278 -0.052220
trhltos  -0.045484 -0.065182  0.035694  0.006019 -0.051231 -0.026178
trhltho  -0.046268 -0.066325  0.041885 -0.011662 -0.028758 -0.005335
trhltht  -0.033279 -0.059154 -0.000319 -0.011478  0.027957  0.012182
trhlthy  -0.006239 -0.015518  0.012505 -0.014544  0.001422  0.003316
trhltmt  -0.061107 -0.064815  0.051790 -0.017298 -0.057175 -0.044500
trhltpt  -0.081500 -0.080023  0.045299  0.001577 -0.073224 -0.023297
```

trhltre	-0.026154	-0.029385	0.015747	0.010546	-0.004636	-0.004436
trhltsh	-0.028732	-0.034999	0.028812	-0.015827	0.025511	0.007745
trhltnt	0.106902	0.118130	-0.067625	0.005807	0.081263	0.044329
hltprhc	-0.005015	0.035915	-0.103404	-0.012773	0.039883	0.067516
hltpral	-0.004182	-0.050246	0.018043	0.004471	-0.021354	-0.020715
hltprhb	0.014055	0.054168	-0.114756	-0.017867	-0.006382	0.046075
hltprbp	0.020671	0.019681	-0.061339	-0.052493	-0.011851	-0.002764
hltprbn	-0.001078	-0.028940	-0.019771	-0.058890	-0.064003	-0.031143
hltprpa	-0.011530	-0.013705	-0.045873	-0.033694	-0.001922	0.031563
hltprpf	-0.021787	-0.012113	-0.049112	-0.028182	-0.021947	0.020676
hltprsd	0.009226	-0.025212	-0.023998	-0.032448	-0.027642	-0.025405
hltprsc	-0.011838	-0.032188	0.006848	-0.036536	-0.031052	-0.038272
hltprsh	0.002658	-0.037784	-0.003502	-0.025480	0.031714	0.002434
hltprdi	0.002130	0.041187	-0.083189	-0.019121	0.056908	0.057293
hltprnt	0.014897	0.040084	0.044417	0.058297	0.098931	0.042650
happy	-0.104935	-0.130565	0.130141	0.068639	-0.058722	0.002293
health	0.057973	0.097560	-0.214492	-0.092356	0.073997	0.088677
hlthhmp	-0.009757	-0.040273	0.158750	0.066842	-0.066964	-0.086604
rlgdgr	-0.073137	-0.022540	-0.048174	0.121308	0.153739	0.153477
pray	0.077590	0.028672	0.059056	-0.117056	-0.171414	-0.162996
height	0.093786	0.069456	0.082114	-0.125182	-0.236828	-0.212506
weighta	0.103818	0.106475	-0.058544	-0.117371	-0.139084	-0.143772
fnsdfml	-0.026398	-0.050886	0.069184	0.093580	-0.056928	-0.004687
jbexevh	0.050241	0.045560	0.035245	-0.132031	-0.080451	-0.071389
jbexevc	0.041978	0.030388	0.026034	-0.102293	-0.068737	-0.075404
jbexera	-0.020419	-0.040091	0.026527	-0.012428	-0.037614	-0.024184
jbexecp	0.009118	-0.005838	0.034855	-0.087447	-0.072241	-0.060276
jbexebs	0.050258	0.036778	0.024200	-0.118925	-0.095056	-0.085070
chldhhe	0.041868	0.013620	0.074678	0.037459	0.004909	-0.083490
domicil	0.024024	0.046126	-0.012778	0.022936	-0.007089	0.021628
paccmoro	0.023669	0.000073	-0.019365	-0.042926	0.001766	-0.019801
paccocrw	0.003664	0.007330	-0.016304	-0.017680	0.016086	-0.011697
paccxhoc	0.009842	-0.003383	0.012425	-0.046980	0.010424	-0.021306
paccinro	0.002865	-0.019672	-0.001157	-0.034350	-0.014996	-0.032525
isco08	0.120424	0.169997	-0.087795	-0.083925	0.147612	0.089580
nacer2	-0.088577	-0.118462	0.057361	0.056866	-0.009676	0.007445
hinctnta	-0.028610	-0.085953	0.077004	0.032589	-0.136218	-0.119429

	dshltgp	dshltms	dshltnt	trhltacu	…	jbexebs	chldhhe	\
etfruit	-0.038094	-0.051405	0.048873	-0.034777	…	0.050258	0.041868	
eatveg	-0.039290	-0.045035	0.056970	-0.050927	…	0.036778	0.013620	
dosprt	-0.051196	-0.021486	0.036106	0.019255	…	0.024200	0.074678	
cgtsmok	-0.014411	-0.034467	0.014889	-0.000942	…	-0.118925	0.037459	
alcfreq	-0.014164	-0.045518	0.030329	-0.022068	…	-0.095056	0.004909	
alcbnge	0.007936	0.048035	-0.007142	-0.025446	…	-0.085070	-0.083490	
dshltgp	1.000000	0.198509	-0.807409	0.047061	…	0.010960	-0.104669	
dshltms	0.198509	1.000000	-0.420127	0.074657	…	0.033324	-0.075181	

dshltnt	-0.807409	-0.420127	1.000000	-0.053252	…	-0.012957	0.100559
trhltacu	0.047061	0.074657	-0.053252	1.000000	…	0.010042	-0.002416
trhltacp	0.011762	0.027540	-0.019747	0.068847	…	0.008309	0.001473
trhltcm	0.015196	0.045036	-0.015843	0.184860	…	-0.007248	0.002191
trhltch	0.051979	0.046725	-0.050453	0.104867	…	0.032765	-0.009211
trhltos	0.052198	0.087628	-0.059641	0.126777	…	0.000360	-0.014518
trhltho	0.045141	0.076682	-0.051492	0.124818	…	0.000140	-0.008334
trhltht	0.059508	0.057943	-0.060713	0.054140	…	0.007054	-0.025268
trhlthy	0.010006	0.012524	-0.008461	0.041273	…	0.001772	0.013376
trhltmt	0.099989	0.127272	-0.105183	0.140307	…	0.019549	-0.012820
trhltpt	0.155717	0.235307	-0.171751	0.127678	…	0.039484	-0.019769
trhltre	0.028617	0.047307	-0.027385	0.107154	…	0.003254	-0.003214
trhltsh	0.021370	0.021133	-0.014806	0.058328	…	-0.002728	0.004673
trhltnt	-0.166191	-0.210248	0.188768	-0.227378	…	-0.035790	0.029812
hltprhc	0.119848	0.187502	-0.129510	0.012310	…	0.048813	-0.091922
hltpral	0.060953	0.075914	-0.071676	0.049874	…	0.021582	0.052002
hltprhb	0.169146	0.165231	-0.166793	-0.000295	…	0.045514	-0.149022
hltprbp	0.104370	0.130102	-0.103013	0.033644	…	0.067827	-0.023294
hltprbn	0.151830	0.178366	-0.163628	0.090833	…	0.095762	-0.051576
hltprpa	0.131233	0.178188	-0.135019	0.054473	…	0.094081	-0.078210
hltprpf	0.135906	0.207294	-0.143325	0.049793	…	0.082966	-0.061815
hltprsd	0.115474	0.146922	-0.122989	0.062332	…	0.053756	0.005058
hltprsc	0.079364	0.124157	-0.092200	0.027110	…	0.044017	0.031404
hltprsh	0.078298	0.099480	-0.086676	0.052731	…	0.031262	0.037272
hltprdi	0.086051	0.124337	-0.091895	0.001028	…	0.026036	-0.077872
hltprnt	-0.260147	-0.265820	0.300131	-0.067627	…	-0.105646	0.080863
happy	-0.044514	-0.011489	0.023398	0.019624	…	-0.029948	-0.032081
health	0.231318	0.262598	-0.242302	0.029405	…	0.068065	-0.148418
hlthhmp	-0.184315	-0.278634	0.195868	-0.050175	…	-0.078035	0.079247
rlgdgr	0.079703	0.046074	-0.080577	0.002732	…	-0.046788	-0.125036
pray	-0.090999	-0.055887	0.088633	-0.003686	…	0.062477	0.122661
height	-0.102020	-0.068374	0.105784	-0.020090	…	0.157896	0.102924
weighta	-0.000227	0.006811	0.011926	-0.010002	…	0.156340	-0.085414
fnsdfml	-0.067934	-0.036006	0.055071	0.018314	…	-0.076690	0.102247
jbexevh	0.025022	0.051219	-0.028818	0.007634	…	0.355690	-0.010694
jbexevc	0.013861	0.016822	-0.011923	0.003985	…	0.329816	-0.008646
jbexera	-0.004038	0.043044	-0.014878	0.005554	…	0.113218	-0.005086
jbexecp	0.019309	0.055103	-0.034603	0.019641	…	0.409181	-0.020695
jbexebs	0.010960	0.033324	-0.012957	0.010042	…	1.000000	-0.016455
chldhhe	-0.104669	-0.075181	0.100559	-0.002416	…	-0.016455	1.000000
domicil	-0.007140	-0.022381	0.013830	0.002081	…	0.061598	-0.080282
paccmoro	0.008953	0.012744	-0.006236	0.015053	…	0.028855	0.035050
paccocrw	-0.000577	-0.014339	0.001427	-0.002798	…	0.011044	-0.000819
paccxhoc	0.007173	0.015180	-0.013431	0.010462	…	0.050905	0.060303
paccinro	0.013672	0.014319	-0.018702	0.011841	…	0.049409	0.010066
isco08	0.002388	-0.064604	0.027171	-0.043693	…	0.135125	0.018663
nacer2	0.002921	0.047417	-0.022237	0.036902	…	-0.146302	0.005026

hinctnta	-0.037689	-0.004820	0.016811	0.026644	…	-0.021964	-0.022957

	domicil	paccmoro	paccocrw	paccxhoc	paccinro	isco08	\
etfruit	0.024024	0.023669	0.003664	0.009842	0.002865	0.120424	
eatveg	0.046126	0.000073	0.007330	-0.003383	-0.019672	0.169997	
dosprt	-0.012778	-0.019365	-0.016304	0.012425	-0.001157	-0.087795	
cgtsmok	0.022936	-0.042926	-0.017680	-0.046980	-0.034350	-0.083925	
alcfreq	-0.007089	0.001766	0.016086	0.010424	-0.014996	0.147612	
alcbnge	0.021628	-0.019801	-0.011697	-0.021306	-0.032525	0.089580	
dshltgp	-0.007140	0.008953	-0.000577	0.007173	0.013672	0.002388	
dshltms	-0.022381	0.012744	-0.014339	0.015180	0.014319	-0.064604	
dshltnt	0.013830	-0.006236	0.001427	-0.013431	-0.018702	0.027171	
trhltacu	0.002081	0.015053	-0.002798	0.010462	0.011841	-0.043693	
trhltacp	-0.000883	0.025991	-0.010015	0.009193	0.016678	-0.022182	
trhltcm	-0.016664	0.011247	-0.003351	0.015580	0.000399	-0.031028	
trhltch	0.004255	0.008659	0.001781	0.016713	0.013763	-0.048950	
trhltos	0.014246	0.017598	0.000546	0.020055	0.018275	-0.062468	
trhltho	-0.004263	0.009363	-0.007668	0.018474	0.011401	-0.057320	
trhltht	-0.045288	0.015254	-0.008329	0.005102	0.027181	-0.019983	
trhlthy	0.002692	0.026138	-0.007896	-0.000453	0.024930	-0.010829	
trhltmt	-0.050466	-0.005699	0.002030	0.017172	0.013184	-0.084755	
trhltpt	-0.009974	0.012999	0.006843	0.025814	0.014183	-0.094298	
trhltre	0.003397	0.001405	0.010334	0.004795	0.004777	-0.025407	
trhltsh	-0.014584	0.020534	-0.001915	0.011431	0.016659	-0.018492	
trhltnt	0.030151	-0.012913	0.001165	-0.023708	-0.021725	0.127082	
hltprhc	0.014357	0.015697	-0.018467	0.008735	0.008896	0.042006	
hltpral	-0.036917	0.051472	0.014168	0.052405	0.057154	-0.066585	
hltprhb	0.035136	-0.001658	-0.019886	-0.009903	0.025708	0.052812	
hltprbp	0.011125	0.050010	0.008000	0.042636	0.042139	0.020796	
hltprbn	0.002888	0.054069	0.004665	0.049710	0.043556	-0.027192	
hltprpa	0.027237	0.051992	0.002475	0.049147	0.055279	0.029333	
hltprpf	0.007122	0.041367	-0.003756	0.047617	0.054031	0.006846	
hltprsd	-0.016973	0.066171	0.009110	0.062663	0.069568	-0.029388	
hltprsc	-0.047585	0.060311	0.005189	0.046791	0.046668	-0.053408	
hltprsh	-0.031457	0.058930	0.023893	0.085624	0.061341	-0.017061	
hltprdi	0.007901	0.002609	-0.010794	0.004637	0.010786	0.044422	
hltprnt	-0.009413	-0.057272	-0.009504	-0.062065	-0.069176	0.032054	
happy	0.011262	-0.059001	-0.033755	-0.055865	-0.051414	-0.137063	
health	0.023641	0.041495	0.008110	0.035298	0.056915	0.146262	
hlthhmp	-0.023094	-0.047886	-0.015870	-0.047144	-0.063295	-0.104061	
rlgdgr	0.073095	-0.019542	0.009879	-0.008073	-0.011527	0.068527	
pray	-0.073472	0.000839	-0.012404	0.017091	0.007761	-0.077446	
height	-0.017801	-0.014590	-0.015917	-0.004810	-0.000757	-0.056442	
weighta	0.047724	-0.009409	-0.006128	-0.012889	0.009633	0.026860	
fnsdfml	0.003705	-0.065679	-0.028101	-0.061100	-0.059825	-0.133564	
jbexevh	0.061380	0.041014	0.014766	0.070147	0.047173	0.148017	
jbexevc	0.058725	0.045978	0.020995	0.056097	0.051553	0.124562	

jbexera	-0.009328	0.013212	0.001714	0.016592	0.013642	-0.086703
jbexecp	0.028254	0.044630	0.012163	0.055096	0.049637	0.061310
jbexebs	0.061598	0.028855	0.011044	0.050905	0.049409	0.135125
chldhhe	-0.080282	0.035050	-0.000819	0.060303	0.010066	0.018663
domicil	1.000000	0.002113	-0.031278	-0.051135	0.045118	0.121286
paccmoro	0.002113	1.000000	0.058000	0.154030	0.171526	0.031551
paccocrw	-0.031278	0.058000	1.000000	0.056548	0.031266	0.028723
paccxhoc	-0.051135	0.154030	0.056548	1.000000	0.110663	0.023799
paccinro	0.045118	0.171526	0.031266	0.110663	1.000000	0.015345
isco08	0.121286	0.031551	0.028723	0.023799	0.015345	1.000000
nacer2	-0.126961	0.003123	-0.002576	0.013168	-0.006600	-0.341363
hinctnta	-0.053566	-0.070891	-0.022854	-0.063413	-0.031774	-0.284425

	nacer2	hinctnta
etfruit	-0.088577	-0.028610
eatveg	-0.118462	-0.085953
dosprt	0.057361	0.077004
cgtsmok	0.056866	0.032589
alcfreq	-0.009676	-0.136218
alcbnge	0.007445	-0.119429
dshltgp	0.002921	-0.037689
dshltms	0.047417	-0.004820
dshltnt	-0.022237	0.016811
trhltacu	0.036902	0.026644
trhltacp	0.021382	0.005615
trhltcm	0.024020	0.011168
trhltch	0.025734	0.039102
trhltos	0.046084	0.046344
trhltho	0.040743	0.043920
trhltht	0.011981	0.005681
trhlthy	0.018478	-0.002098
trhltmt	0.045784	0.084043
trhltpt	0.059489	0.035631
trhltre	0.027610	-0.008180
trhltsh	0.037670	-0.012934
trhltnt	-0.087442	-0.083114
hltprhc	-0.042141	-0.139087
hltpral	0.058994	0.029389
hltprhb	-0.050574	-0.143797
hltprbp	0.006379	-0.071881
hltprbn	0.028611	-0.003628
hltprpa	0.005356	-0.092522
hltprpf	0.010681	-0.083911
hltprsd	0.050144	-0.021175
hltprsc	0.044928	0.004444
hltprsh	0.040050	0.003717
hltprdi	-0.029970	-0.108655

```
hltprnt  -0.026967  0.061472
happy     0.083028  0.180186
health   -0.093246 -0.228068
hlthhmp   0.052759  0.208845
rlgdgr   -0.011202 -0.103366
pray      0.000414  0.134465
height   -0.112778  0.189369
weighta  -0.131059  0.070832
fnsdfml   0.060768  0.162480
jbexevh  -0.131749 -0.033131
jbexevc  -0.131159 -0.041705
jbexera   0.087854  0.052661
jbexecp  -0.048725 -0.001265
jbexebs  -0.146302 -0.021964
chldhhe   0.005026 -0.022957
domicil  -0.126961 -0.053566
paccmoro  0.003123 -0.070891
paccocrw -0.002576 -0.022854
paccxhoc  0.013168 -0.063413
paccinro -0.006600 -0.031774
isco08   -0.341363 -0.284425
nacer2    1.000000  0.084305
hinctnta  0.084305  1.000000

[56 rows x 56 columns]
```

The data shows several notable correlations. A positive correlation exists between health and dshltms (whether health was discussed with a medical professional in the last 12 months): 0.263. There's also a correlation between isco08 (occupation) and eatveg (how often vegetables were eaten): 0.17. Hltprbn (high blood pressure) and hltprbn (back or neck pain) have a positive correlation as well, indicating overall health deterioration for those with chronic conditions: 0.169.

Next, I created frequency counts to see how many respondents chose each category for each feature/question.

[21]:
```
all_columns = df_cleaned

for column in all_columns:
    print(f"Frequency counts for {column}:")
    print(df_cleaned[column].value_counts())
    print(f"Percentage distribution for {column}:")
    print(df_cleaned[column].value_counts(normalize=True) * 100)
    print("-" * 50)
```

```
Frequency counts for etfruit:
etfruit
3.0    8674
2.0    4452
4.0    4027
```

```
5.0    2161
1.0    1506
6.0    1131
7.0     239
Name: count, dtype: int64
Percentage distribution for etfruit:
etfruit
3.0    39.089680
2.0    20.063091
4.0    18.147814
5.0     9.738621
1.0     6.786841
6.0     5.096890
7.0     1.077062
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for eatveg:
eatveg
3.0    10467
4.0     4262
2.0     3904
5.0     1779
1.0     1167
6.0      495
7.0      116
Name: count, dtype: int64
Percentage distribution for eatveg:
eatveg
3.0    47.169896
4.0    19.206850
2.0    17.593511
5.0     8.017125
1.0     5.259126
6.0     2.230735
7.0     0.522758
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for dosprt:
dosprt
7.0    5718
0.0    4295
3.0    2836
2.0    2575
5.0    2201
4.0    2088
1.0    1551
6.0     926
Name: count, dtype: int64
```

```
Percentage distribution for dosprt:
dosprt
7.0    25.768364
0.0    19.355566
3.0    12.780532
2.0    11.604326
5.0     9.918882
4.0     9.409644
1.0     6.989635
6.0     4.173051
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for cgtsmok:
cgtsmok
6.0    10226
4.0     4838
1.0     2699
5.0     2293
2.0     1242
3.0      892
Name: count, dtype: int64
Percentage distribution for cgtsmok:
cgtsmok
6.0    46.083822
4.0    21.802614
1.0    12.163137
5.0    10.333484
2.0     5.597116
3.0     4.019829
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for alcfreq:
alcfreq
7.0    5897
3.0    3606
6.0    3476
2.0    3369
4.0    2944
5.0    1892
1.0    1006
Name: count, dtype: int64
Percentage distribution for alcfreq:
alcfreq
7.0    26.575034
3.0    16.250563
6.0    15.664714
2.0    15.182515
4.0    13.267237
```

```
5.0     8.526363
1.0     4.533574
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for alcbnge:
alcbnge
5.0    11201
4.0     5202
3.0     2864
2.0     2463
1.0      460
Name: count, dtype: int64
Percentage distribution for alcbnge:
alcbnge
5.0    50.477693
4.0    23.442992
3.0    12.906715
2.0    11.099594
1.0     2.073006
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for dshltgp:
dshltgp
1    16537
0     5653
Name: count, dtype: int64
Percentage distribution for dshltgp:
dshltgp
1    74.524561
0    25.475439
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for dshltms:
dshltms
0    12392
1     9798
Name: count, dtype: int64
Percentage distribution for dshltms:
dshltms
0    55.844975
1    44.155025
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for dshltnt:
dshltnt
0    18133
1     4057
Name: count, dtype: int64
```

```
Percentage distribution for dshltnt:
dshltnt
0    81.71699
1    18.28301
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltacu:
trhltacu
0    21499
1      691
Name: count, dtype: int64
Percentage distribution for trhltacu:
trhltacu
0    96.885985
1     3.114015
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltacp:
trhltacp
0    22044
1      146
Name: count, dtype: int64
Percentage distribution for trhltacp:
trhltacp
0    99.342046
1     0.657954
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltcm:
trhltcm
0    21923
1      267
Name: count, dtype: int64
Percentage distribution for trhltcm:
trhltcm
0    98.796755
1     1.203245
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltch:
trhltch
0    21508
1      682
Name: count, dtype: int64
Percentage distribution for trhltch:
trhltch
0    96.926543
1     3.073457
```

```
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltos:
trhltos
0    21470
1      720
Name: count, dtype: int64
Percentage distribution for trhltos:
trhltos
0    96.755295
1     3.244705
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltho:
trhltho
0    21397
1      793
Name: count, dtype: int64
Percentage distribution for trhltho:
trhltho
0    96.426318
1     3.573682
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltht:
trhltht
0    20900
1     1290
Name: count, dtype: int64
Percentage distribution for trhltht:
trhltht
0    94.186571
1     5.813429
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhlthy:
trhlthy
0    22099
1       91
Name: count, dtype: int64
Percentage distribution for trhlthy:
trhlthy
0    99.589905
1     0.410095
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltmt:
trhltmt
```

```
0    18505
1     3685
Name: count, dtype: int64
Percentage distribution for trhltmt:
trhltmt
0    83.39342
1    16.60658
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltpt:
trhltpt
0    17981
1     4209
Name: count, dtype: int64
Percentage distribution for trhltpt:
trhltpt
0    81.031996
1    18.968004
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltre:
trhltre
0    21824
1      366
Name: count, dtype: int64
Percentage distribution for trhltre:
trhltre
0    98.350608
1     1.649392
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltsh:
trhltsh
0    21810
1      380
Name: count, dtype: int64
Percentage distribution for trhltsh:
trhltsh
0    98.287517
1     1.712483
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for trhltnt:
trhltnt
1    13708
0     8482
Name: count, dtype: int64
Percentage distribution for trhltnt:
```

```
trhltnt
1    61.775575
0    38.224425
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprhc:
hltprhc
0    19656
1     2534
Name: count, dtype: int64
Percentage distribution for hltprhc:
hltprhc
0    88.580442
1    11.419558
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltpral:
hltpral
0    19415
1     2775
Name: count, dtype: int64
Percentage distribution for hltpral:
hltpral
0    87.494367
1    12.505633
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprhb:
hltprhb
0    17358
1     4832
Name: count, dtype: int64
Percentage distribution for hltprhb:
hltprhb
0    78.224425
1    21.775575
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprbp:
hltprbp
0    20251
1     1939
Name: count, dtype: int64
Percentage distribution for hltprbp:
hltprbp
0    91.26183
1     8.73817
Name: proportion, dtype: float64
```

```
--------------------------------------------------
Frequency counts for hltprbn:
hltprbn
0    14280
1     7910
Name: count, dtype: int64
Percentage distribution for hltprbn:
hltprbn
0    64.353312
1    35.646688
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprpa:
hltprpa
0    17809
1     4381
Name: count, dtype: int64
Percentage distribution for hltprpa:
hltprpa
0    80.256872
1    19.743128
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprpf:
hltprpf
0    17078
1     5112
Name: count, dtype: int64
Percentage distribution for hltprpf:
hltprpf
0    76.962596
1    23.037404
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprsd:
hltprsd
0    18824
1     3366
Name: count, dtype: int64
Percentage distribution for hltprsd:
hltprsd
0    84.831005
1    15.168995
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprsc:
hltprsc
0    20116
```

```
1     2074
Name: count, dtype: int64
Percentage distribution for hltprsc:
hltprsc
0    90.653447
1     9.346553
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprsh:
hltprsh
0    19669
1     2521
Name: count, dtype: int64
Percentage distribution for hltprsh:
hltprsh
0    88.639027
1    11.360973
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprdi:
hltprdi
0    20776
1     1414
Name: count, dtype: int64
Percentage distribution for hltprdi:
hltprdi
0    93.62776
1     6.37224
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hltprnt:
hltprnt
0    15643
1     6547
Name: count, dtype: int64
Percentage distribution for hltprnt:
hltprnt
0    70.495719
1    29.504281
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for happy:
happy
8.0     6794
9.0     4471
7.0     3830
10.0    2667
6.0     1664
```

```
5.0     1467
4.0      549
3.0      412
2.0      205
0.0       75
1.0       56
Name: count, dtype: int64
Percentage distribution for happy:
happy
8.0     30.617395
9.0     20.148716
7.0     17.260027
10.0    12.018927
6.0      7.498873
5.0      6.611086
4.0      2.474087
3.0      1.856692
2.0      0.923840
0.0      0.337990
1.0      0.252366
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for health:
health
2.0    9683
3.0    5552
1.0    5347
4.0    1369
5.0     239
Name: count, dtype: int64
Percentage distribution for health:
health
2.0    43.636773
3.0    25.020279
1.0    24.096440
4.0     6.169446
5.0     1.077062
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for hlthhmp:
hlthhmp
3.0    15566
2.0     5095
1.0     1529
Name: count, dtype: int64
Percentage distribution for hlthhmp:
hlthhmp
3.0    70.148716
```

```
2.0    22.960793
1.0     6.890491
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for rlgdgr:
rlgdgr
0.0     4235
5.0     3059
7.0     2492
8.0     2214
6.0     2107
3.0     1653
2.0     1643
10.0    1335
4.0     1279
1.0     1274
9.0      899
Name: count, dtype: int64
Percentage distribution for rlgdgr:
rlgdgr
0.0     19.085174
5.0     13.785489
7.0     11.230284
8.0      9.977467
6.0      9.495268
3.0      7.449301
2.0      7.404236
10.0     6.016224
4.0      5.763858
1.0      5.741325
9.0      4.051374
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for pray:
pray
7.0    8826
1.0    4024
6.0    3560
2.0    1741
4.0    1412
3.0    1383
5.0    1244
Name: count, dtype: int64
Percentage distribution for pray:
pray
7.0    39.774673
1.0    18.134295
6.0    16.043263
```

```
2.0     7.845877
4.0     6.363227
3.0     6.232537
5.0     5.606129
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for height:
height
170.0    2084
165.0    1394
168.0    1245
180.0    1143
175.0    1094
         …
106.0       1
122.0       1
130.0       1
137.0       1
204.0       1
Name: count, Length: 72, dtype: int64
Percentage distribution for height:
height
170.0    9.391618
165.0    6.282109
168.0    5.610635
180.0    5.150969
175.0    4.930149
           …
106.0    0.004507
122.0    0.004507
130.0    0.004507
137.0    0.004507
204.0    0.004507
Name: proportion, Length: 72, dtype: float64
--------------------------------------------------
Frequency counts for weighta:
weighta
80.0     2956
70.0     1052
75.0      864
90.0      802
85.0      797
         …
129.0       1
133.0       1
139.0       1
148.0       1
143.0       1
```

```
Name: count, Length: 101, dtype: int64
Percentage distribution for weighta:
weighta
80.0     13.321316
70.0      4.740874
75.0      3.893646
90.0      3.614241
85.0      3.591708
           …
129.0     0.004507
133.0     0.004507
139.0     0.004507
148.0     0.004507
143.0     0.004507
Name: proportion, Length: 101, dtype: float64
--------------------------------------------------
Frequency counts for fnsdfml:
fnsdfml
5.0    7526
3.0    6012
4.0    5280
2.0    2684
1.0     688
Name: count, dtype: int64
Percentage distribution for fnsdfml:
fnsdfml
5.0    33.916178
3.0    27.093285
4.0    23.794502
2.0    12.095539
1.0     3.100496
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for jbexevh:
jbexevh
0    18186
1     4004
Name: count, dtype: int64
Percentage distribution for jbexevh:
jbexevh
0    81.955836
1    18.044164
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for jbexevc:
jbexevc
0    19587
1     2603
```

```
Name: count, dtype: int64
Percentage distribution for jbexevc:
jbexevc
0    88.269491
1    11.730509
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for jbexera:
jbexera
0    21506
1      684
Name: count, dtype: int64
Percentage distribution for jbexera:
jbexera
0    96.91753
1     3.08247
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for jbexecp:
jbexecp
0    19428
1     2762
Name: count, dtype: int64
Percentage distribution for jbexecp:
jbexecp
0    87.552952
1    12.447048
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for jbexebs:
jbexebs
0    18754
1     3436
Name: count, dtype: int64
Percentage distribution for jbexebs:
jbexebs
0    84.515548
1    15.484452
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for chldhhe:
chldhhe
1.0    15082
2.0     7108
Name: count, dtype: int64
Percentage distribution for chldhhe:
chldhhe
1.0    67.967553
```

```
2.0    32.032447
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for domicil:
domicil
4.0    7952
3.0    6132
1.0    4057
2.0    2529
5.0    1520
Name: count, dtype: int64
Percentage distribution for domicil:
domicil
4.0    35.835962
3.0    27.634069
1.0    18.283010
2.0    11.397026
5.0     6.849932
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for paccmoro:
paccmoro
0    21031
1     1159
Name: count, dtype: int64
Percentage distribution for paccmoro:
paccmoro
0    94.776927
1     5.223073
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for paccocrw:
paccocrw
0    21859
1      331
Name: count, dtype: int64
Percentage distribution for paccocrw:
paccocrw
0    98.508337
1     1.491663
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for paccxhoc:
paccxhoc
0    21430
1      760
Name: count, dtype: int64
Percentage distribution for paccxhoc:
```

```
paccxhoc
0    96.575034
1     3.424966
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for paccinro:
paccinro
0    21374
1      816
Name: count, dtype: int64
Percentage distribution for paccinro:
paccinro
0    96.322668
1     3.677332
Name: proportion, dtype: float64
--------------------------------------------------
Frequency counts for isco08:
isco08
5223.0    2790
4110.0     514
9112.0     405
2221.0     304
2330.0     289
          …
3150.0       1
1220.0       1
9400.0       1
7420.0       1
4130.0       1
Name: count, Length: 521, dtype: int64
Percentage distribution for isco08:
isco08
5223.0    12.573231
4110.0     2.316359
9112.0     1.825146
2221.0     1.369986
2330.0     1.302388
            …
3150.0     0.004507
1220.0     0.004507
9400.0     0.004507
7420.0     0.004507
4130.0     0.004507
Name: proportion, Length: 521, dtype: float64
--------------------------------------------------
Frequency counts for nacer2:
nacer2
47.0    4764
```

```
85.0    1753
86.0    1560
84.0    1278
56.0     719
        …
5.0       13
9.0        7
12.0       3
7.0        3
39.0       1
Name: count, Length: 85, dtype: int64
Percentage distribution for nacer2:
nacer2
47.0    21.469130
85.0     7.899955
86.0     7.030194
84.0     5.759351
56.0     3.240198
          …
5.0      0.058585
9.0      0.031546
12.0     0.013520
7.0      0.013520
39.0     0.004507
Name: proportion, Length: 85, dtype: float64
--------------------------------------------------
Frequency counts for hinctnta:
hinctnta
6.0     6016
7.0     1954
8.0     1946
3.0     1936
5.0     1879
4.0     1831
2.0     1784
10.0    1741
9.0     1608
1.0     1495
Name: count, dtype: int64
Percentage distribution for hinctnta:
hinctnta
6.0     27.111311
7.0      8.805768
8.0      8.769716
3.0      8.724651
5.0      8.467778
4.0      8.251465
2.0      8.039658
```

```
10.0     7.845877
9.0      7.246507
1.0      6.737269
Name: proportion, dtype: float64
--------------------------------------------------
```

I also looked at the modes to see which category was most commonly selected for each feature.

[22]:
```python
numerical_features = df_cleaned[['height', 'weighta']]
medians = numerical_features.median()
modes = all_columns.mode().iloc[0]
print("Medians:")
print(medians)
print("Modes:")
print(modes)
```

```
Medians:
height     170.0
weighta     76.0
dtype: float64
Modes:
etfruit       3.0
eatveg        3.0
dosprt        7.0
cgtsmok       6.0
alcfreq       7.0
alcbnge       5.0
dshltgp       1.0
dshltms       0.0
dshltnt       0.0
trhltacu      0.0
trhltacp      0.0
trhltcm       0.0
trhltch       0.0
trhltos       0.0
trhltho       0.0
trhltht       0.0
trhlthy       0.0
trhltmt       0.0
trhltpt       0.0
trhltre       0.0
trhltsh       0.0
trhltnt       1.0
hltprhc       0.0
hltpral       0.0
hltprhb       0.0
hltprbp       0.0
hltprbn       0.0
hltprpa       0.0
```

```
hltprpf        0.0
hltprsd        0.0
hltprsc        0.0
hltprsh        0.0
hltprdi        0.0
hltprnt        0.0
happy          8.0
health         2.0
hlthhmp        3.0
rlgdgr         0.0
pray           7.0
height       170.0
weighta       80.0
fnsdfml        5.0
jbexevh        0.0
jbexevc        0.0
jbexera        0.0
jbexecp        0.0
jbexebs        0.0
chldhhe        1.0
domicil        4.0
paccmoro       0.0
paccocrw       0.0
paccxhoc       0.0
paccinro       0.0
isco08      5223.0
nacer2        47.0
hinctnta       6.0
Name: 0, dtype: float64
```

Next, I ran a chi-square test with each selected feature against the allergy label, to check for any correlation between them. The results show a strong correlation between allergies and efruit, (eating fruit), eatveg (eating vegetables), dosprt (doing sports), and a number of other features to do with cigarette smoking, drinking, religion, and other health problems. Some features that showed little to no correlation were certain treatments used including acupressure and hypnotherapy, or the presence of diabetes.

[23]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency

target = 'hltpral'

for feature in all_columns:
    if feature != target:
        crosstab = pd.crosstab(df_cleaned[target], df_cleaned[feature])
        chi2, p, dof, expected = chi2_contingency(crosstab)
        print(f"Chi-square test for {feature}:")
```

```python
    print(f"Chi2 statistic: {chi2}, p-value: {p}")
    print(crosstab)
```

```
Chi-square test for etfruit:
Chi2 statistic: 21.521795066091137, p-value: 0.001477682167613517
etfruit   1.0   2.0   3.0   4.0   5.0  6.0  7.0
hltpral
0        1298  3855  7671  3539  1855  996  201
1         208   597  1003   488   306  135   38
Chi-square test for eatveg:
Chi2 statistic: 66.09439244627502, p-value: 2.5780553339634884e-12
eatveg   1.0   2.0   3.0   4.0   5.0  6.0  7.0
hltpral
0        966  3319  9199  3786  1603  440  102
1        201   585  1268   476   176   55   14
Chi-square test for dosprt:
Chi2 statistic: 40.74805463271075, p-value: 9.049359411958455e-07
dosprt    0.0   1.0   2.0   3.0   4.0   5.0  6.0   7.0
hltpral
0        3866  1329  2237  2440  1806  1946  814  4977
1         429   222   338   396   282   255  112   741
Chi-square test for cgtsmok:
Chi2 statistic: 96.83845472355647, p-value: 2.4494620201691707e-19
cgtsmok   1.0   2.0  3.0   4.0   5.0   6.0
hltpral
0        2455  1094  761  4193  1889  9023
1         244   148  131   645   404  1203
Chi-square test for alcfreq:
Chi2 statistic: 82.47230344584344, p-value: 1.1014097681606256e-15
alcfreq  1.0   2.0   3.0   4.0   5.0   6.0   7.0
hltpral
0        915  2938  3133  2504  1632  2974  5319
1         91   431   473   440   260   502   578
Chi-square test for alcbnge:
Chi2 statistic: 66.6128707961237, p-value: 1.176452751748156e-13
alcbnge  1.0   2.0   3.0   4.0   5.0
hltpral
0        423  2194  2434  4437  9927
1         37   269   430   765  1274
Chi-square test for dshltgp:
Chi2 statistic: 82.02050094935404, p-value: 1.3468240603352096e-19
dshltgp     0      1
hltpral
0        5141  14274
1         512   2263
Chi-square test for dshltms:
Chi2 statistic: 127.41892565249103, p-value: 1.5042186306402113e-29
dshltms      0     1
```

```
hltpral
0        11119  8296
1         1273  1502
Chi-square test for dshltnt:
Chi2 statistic: 113.43890531565575, p-value: 1.7292835716302229e-26
dshltnt      0     1
hltpral
0        15662  3753
1         2471   304
Chi-square test for trhltacu:
Chi2 statistic: 54.33031593504486, p-value: 1.69467080230606e-13
trhltacu      0    1
hltpral
0         18874  541
1          2625  150
Chi-square test for trhltacp:
Chi2 statistic: 3.3045961282128116, p-value: 0.0690863260205883
trhltacp      0    1
hltpral
0         19295  120
1          2749   26
Chi-square test for trhltcm:
Chi2 statistic: 11.3630582890458, p-value: 0.0007491935900520524
trhltcm      0    1
hltpral
0        19200  215
1         2723   52
Chi-square test for trhltch:
Chi2 statistic: 18.129345016949586, p-value: 2.0639602997436312e-05
trhltch      0    1
hltpral
0        18855  560
1         2653  122
Chi-square test for trhltos:
Chi2 statistic: 109.73977630271463, p-value: 1.1173705817690696e-25
trhltos      0    1
hltpral
0        18877  538
1         2593  182
Chi-square test for trhltho:
Chi2 statistic: 54.183483563272354, p-value: 1.8261537807822842e-13
trhltho      0    1
hltpral
0        18789  626
1         2608  167
Chi-square test for trhltht:
Chi2 statistic: 4.396957634066954, p-value: 0.03600310364026719
trhltht      0     1
```

```
hltpral
0        18311  1104
1         2589   186
Chi-square test for trhlthy:
Chi2 statistic: 0.9815932626934929, p-value: 0.3218057715859046
trhlthy      0   1
hltpral
0        19339  76
1         2760  15
Chi-square test for trhltmt:
Chi2 statistic: 81.62423964247307, p-value: 1.64583118870615e-19
trhltmt      0     1
hltpral
0        16357  3058
1         2148   627
Chi-square test for trhltpt:
Chi2 statistic: 118.32800245876933, p-value: 1.4696071860071707e-27
trhltpt      0     1
hltpral
0        15943  3472
1         2038   737
Chi-square test for trhltre:
Chi2 statistic: 7.980742723232706, p-value: 0.0047277541575029575
trhltre      0    1
hltpral
0        19113  302
1         2711   64
Chi-square test for trhltsh:
Chi2 statistic: 8.81374194691826, p-value: 0.0029897027224918887
trhltsh      0    1
hltpral
0        19102  313
1         2708   67
Chi-square test for trhltnt:
Chi2 statistic: 210.9545381558471, p-value: 8.503835152003555e-48
trhltnt     0      1
hltpral
0        7073  12342
1        1409   1366
Chi-square test for hltprhc:
Chi2 statistic: 7.04868187720574, p-value: 0.007932360215167389
hltprhc      0     1
hltpral
0        17240  2175
1         2416   359
Chi-square test for hltprhb:
Chi2 statistic: 0.012000501269886416, p-value: 0.9127688044974189
hltprhb      0     1
```

```
hltpral
0        15190  4225
1         2168   607
Chi-square test for hltprbp:
Chi2 statistic: 569.3307926025976, p-value: 7.850321217953867e-126
hltprbp      0     1
hltpral
0        18051  1364
1         2200   575
Chi-square test for hltprbn:
Chi2 statistic: 329.36082254421297, p-value: 1.3243459973735211e-73
hltprbn      0     1
hltpral
0        12923  6492
1         1357  1418
Chi-square test for hltprpa:
Chi2 statistic: 154.28151298462134, p-value: 2.0100530206685109e-35
hltprpa      0     1
hltpral
0        15826  3589
1         1983   792
Chi-square test for hltprpf:
Chi2 statistic: 108.59338941568132, p-value: 1.9923807623312883e-25
hltprpf      0     1
hltpral
0        15159  4256
1         1919   856
Chi-square test for hltprsd:
Chi2 statistic: 333.0178886469475, p-value: 2.1159047003318312e-74
hltprsd      0     1
hltpral
0        16793  2622
1         2031   744
Chi-square test for hltprsc:
Chi2 statistic: 793.907923085484, p-value: 1.1391859078097676e-174
hltprsc      0     1
hltpral
0        18005  1410
1         2111   664
Chi-square test for hltprsh:
Chi2 statistic: 323.4811065403386, p-value: 2.5272806493513786e-72
hltprsh      0     1
hltpral
0        17491  1924
1         2178   597
Chi-square test for hltprdi:
Chi2 statistic: 0.12940955985092492, p-value: 0.7190450145683268
hltprdi      0     1
```

```
hltpral
0        18173  1242
1         2603   172
Chi-square test for hltprnt:
Chi2 statistic: 1325.7895403115183, p-value: 2.8108330237830844e-290
hltprnt      0     1
hltpral
0        12868  6547
1         2775     0
Chi-square test for happy:
Chi2 statistic: 39.520873195763684, p-value: 2.0572822156394768e-05
happy    0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0   10.0
hltpral
0         64    43   177   358   496  1293  1481  3371  5874  3864  2394
1         11    13    28    54    53   174   183   459   920   607   273
Chi-square test for health:
Chi2 statistic: 36.906432925776826, p-value: 1.883047764086483e-07
health    1.0   2.0   3.0   4.0  5.0
hltpral
0        4777  8483  4798  1157  200
1         570  1200   754   212   39
Chi-square test for hlthhmp:
Chi2 statistic: 80.7837120231286, p-value: 2.8710438152096566e-18
hlthhmp   1.0   2.0    3.0
hltpral
0        1266  4334  13815
1         263   761   1751
Chi-square test for rlgdgr:
Chi2 statistic: 65.56672676370714, p-value: 3.157448234930856e-10
rlgdgr   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0   10.0
hltpral
0        3651  1054  1433  1411  1113  2732  1847  2213  1988   790  1183
1         584   220   210   242   166   327   260   279   226   109   152
Chi-square test for pray:
Chi2 statistic: 87.1636834330165, p-value: 1.175306398313408e-16
pray      1.0   2.0   3.0   4.0   5.0   6.0   7.0
hltpral
0        3563  1580  1235  1253  1151  3043  7590
1         461   161   148   159    93   517  1236
Chi-square test for height:
Chi2 statistic: 93.37441165835972, p-value: 0.03879335319003395
height   106.0  108.0  120.0  122.0  130.0  135.0  137.0  140.0  142.0  143.0  \
hltpral
0            1      1      1      1      1      3      0      2     10      2
1            0      0      0      0      0      0      1      1      1      0

height   …  196.0  197.0  198.0  199.0  200.0  201.0  202.0  203.0  204.0  \
hltpral  …
```

```
0        …      24     14     17      5     15      2      2      1      1
1        …       1      2      0      0      2      0      0      1      0

height   205.0
hltpral
0            2
1            0

[2 rows x 72 columns]
Chi-square test for weighta:
Chi2 statistic: 108.15807019426238, p-value: 0.2713506036600929
weighta  30.0   39.0   40.0   41.0   42.0   43.0   44.0   45.0   46.0   47.0   \
hltpral
0            1      2      5      7      6      3     23     28     22     28
1            0      0      3      0      1      1      6      8      6      6

weighta  …  133.0  134.0  135.0  136.0  138.0  139.0  140.0  143.0  145.0  \
hltpral  …
0        …      1      2      8      4      2      1      5      1      2
1        …      0      1      2      0      0      0      0      0      0

weighta  148.0
hltpral
0            1
1            0

[2 rows x 101 columns]
Chi-square test for fnsdfml:
Chi2 statistic: 6.994501968097959, p-value: 0.13617905662315938
fnsdfml  1.0   2.0   3.0   4.0   5.0
hltpral
0        594  2358  5304  4623  6536
1         94   326   708   657   990
Chi-square test for jbexevh:
Chi2 statistic: 13.950611668257988, p-value: 0.00018767658074180044
jbexevh      0     1
hltpral
0        15983  3432
1         2203   572
Chi-square test for jbexevc:
Chi2 statistic: 5.148875224326842, p-value: 0.023261436029307882
jbexevc      0     1
hltpral
0        17174  2241
1         2413   362
Chi-square test for jbexera:
Chi2 statistic: 7.268672889160196, p-value: 0.00701676300729221
jbexera      0    1
```

```
hltpral
0        18840  575
1         2666  109
Chi-square test for jbexecp:
Chi2 statistic: 50.06430960723819, p-value: 1.4878877963150697e-12
jbexecp      0     1
hltpral
0        17114  2301
1         2314   461
Chi-square test for jbexebs:
Chi2 statistic: 10.15592934111358, p-value: 0.0014383786818635855
jbexebs      0     1
hltpral
0        16466  2949
1         2288   487
Chi-square test for chldhhe:
Chi2 statistic: 59.66904788561761, p-value: 1.1222777792765835e-14
chldhhe    1.0   2.0
hltpral
0        13374  6041
1         1708  1067
Chi-square test for domicil:
Chi2 statistic: 64.39925127563997, p-value: 3.4436059753073183e-13
domicil   1.0   2.0   3.0   4.0   5.0
hltpral
0        3552  2142  5259  7100  1362
1         505   387   873   852   158
Chi-square test for paccmoro:
Chi2 statistic: 58.09276256529061, p-value: 2.5004453791952638e-14
paccmoro      0    1
hltpral
0         18485  930
1          2546  229
Chi-square test for paccocrw:
Chi2 statistic: 4.108090416964615, p-value: 0.04267852509403052
paccocrw      0    1
hltpral
0         19138  277
1          2721   54
Chi-square test for paccxhoc:
Chi2 statistic: 60.07173197048796, p-value: 9.146253929152913e-15
paccxhoc      0    1
hltpral
0         18820  595
1          2610  165
Chi-square test for paccinro:
Chi2 statistic: 71.56919164366832, p-value: 2.6770407455677704e-17
paccinro      0    1
```

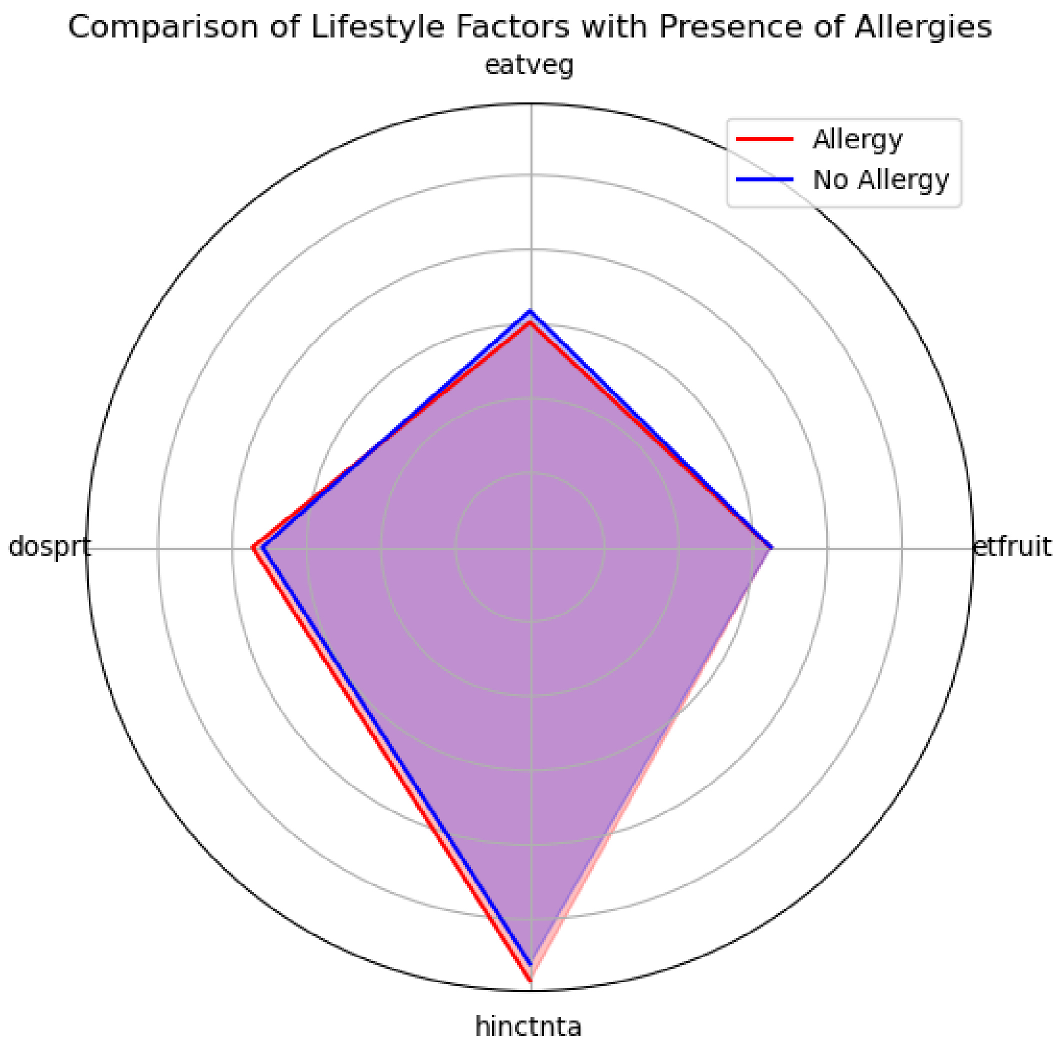
```
hltpral
0         18780  635
1          2594  181
Chi-square test for isco08:
Chi2 statistic: 847.6006602375436, p-value: 4.1551186786293866e-18
isco08   110.0   210.0   310.0   1000.0  1110.0  1111.0  1112.0  1113.0  \
hltpral
0           23      17      23      14       4      10      91       7
1            6       0       4       4       0       0      11       0

isco08   1114.0  1120.0  …  9510.0  9520.0  9611.0  9612.0  9613.0  9620.0  \
hltpral                  …
0             7     233  …       2       1      19       6      26       2
1             0      29  …       0       0       0       2       0       0

isco08   9621.0  9622.0  9623.0  9629.0
hltpral
0            53      17       1     138
1             9       4       0      14

[2 rows x 521 columns]
Chi-square test for nacer2:
Chi2 statistic: 263.3989473347392, p-value: 2.1980998484912272e-20
nacer2   1.0   2.0   3.0   5.0   6.0   7.0   8.0   9.0   10.0  11.0  …  \
hltpral                                                              …
0         652    68    15    11    25     2    29     7   380    31  …
1          55     4     2     2     5     1     2     0    49     3  …

nacer2   87.0  90.0  91.0  92.0  93.0  94.0  95.0  96.0  97.0  98.0
hltpral
0         291   124    89    27   108   124    27   264    33    16
1          60    23    14     5    31    18     0    28     3     3

[2 rows x 85 columns]
Chi-square test for hinctnta:
Chi2 statistic: 104.36123598599413, p-value: 2.0578900642487005e-18
hinctnta  1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0   10.0
hltpral
0         1313  1579  1702  1599  1606  5450  1670  1659  1377  1460
1          182   205   234   232   273   566   284   287   231   281
```

3.1 Visualizations

The following visualizations are radar charts showing the relationship between four selected features and their relationship to allergies. The closer the corner of the quadrilateral gets to the edge, the higher the average numerical response for the survey out of the given options. These visuals help us see the distribution of responses for each feature, and how they differ for those with allergies vs. those without.

[24]:

```
df_cleaned['hltpral_mapped'] = df_cleaned['hltpral'].map({0: 'No', 1: 'Yes'})

categories = ['etfruit', 'eatveg', 'dosprt', 'hinctnta']
values_allergy = df_cleaned[df_cleaned['hltpral_mapped'] == 'Yes'][categories].
 ↪mean()
values_no_allergy = df_cleaned[df_cleaned['hltpral_mapped'] ==␣
 ↪'No'][categories].mean()

angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False).tolist()

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.plot(angles, values_allergy, label='Allergy', color='red')
ax.plot(angles, values_no_allergy, label='No Allergy', color='blue')

ax.fill(angles, values_allergy, color='red', alpha=0.25)
ax.fill(angles, values_no_allergy, color='blue', alpha=0.25)

ax.set_yticklabels([])
ax.set_xticks(angles)
ax.set_xticklabels(categories)

plt.title('Comparison of Lifestyle Factors with Presence of Allergies')
plt.legend(loc='upper right')
plt.show()
```



etfruit = How often the respondent eats fruit

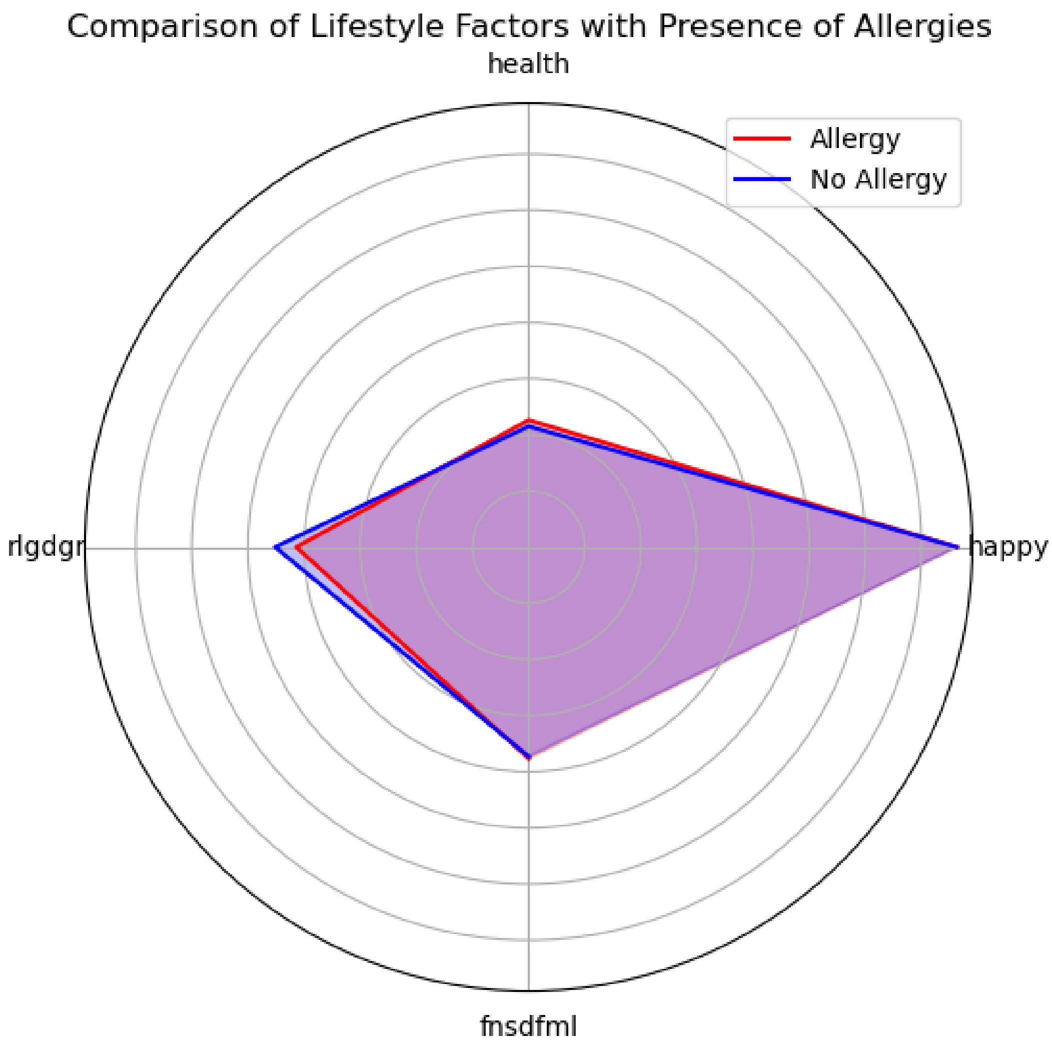
eatveg = How often the respondent eats vegetables or salads, excluding potatoes

dosprt = How often the respondent does sports

hinctnta = Total household net income

[25]:
```
df_cleaned['hltpral_mapped'] = df_cleaned['hltpral'].map({0: 'No', 1: 'Yes'})

categories = ['happy', 'health', 'rlgdgr', 'fnsdfml']
values_allergy = df_cleaned[df_cleaned['hltpral_mapped'] == 'Yes'][categories].
 ↪mean()
values_no_allergy = df_cleaned[df_cleaned['hltpral_mapped'] ==␣
 ↪'No'][categories].mean()
```

```
angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False).tolist()

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.plot(angles, values_allergy, label='Allergy', color='red')
ax.plot(angles, values_no_allergy, label='No Allergy', color='blue')

ax.fill(angles, values_allergy, color='red', alpha=0.25)
ax.fill(angles, values_no_allergy, color='blue', alpha=0.25)

ax.set_yticklabels([])
ax.set_xticks(angles)
ax.set_xticklabels(categories)

plt.title('Comparison of Lifestyle Factors with Presence of Allergies')
plt.legend(loc='upper right')
plt.show()
```



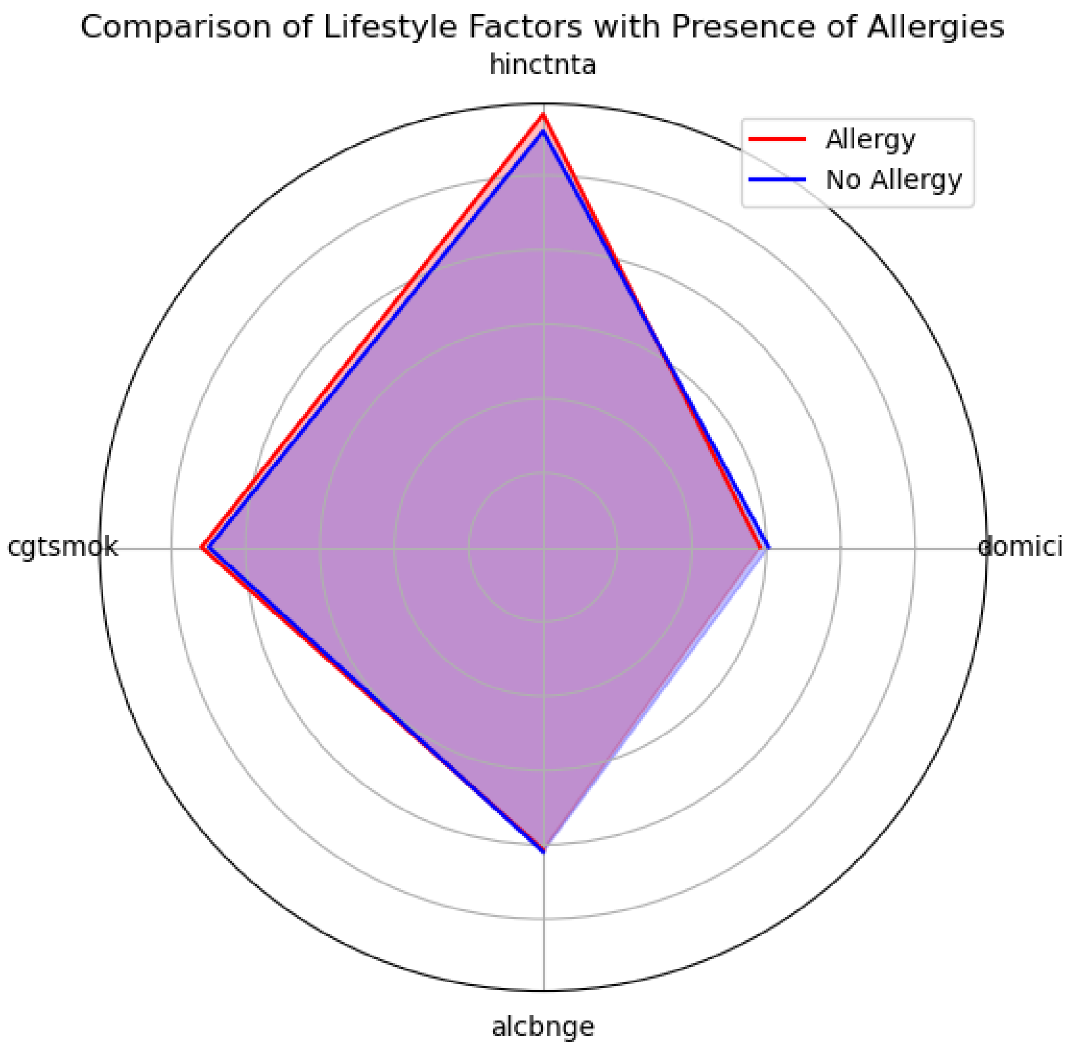
health = Subjective general health

happy = How happy respondent is overall

rlgdgr = How religious respondent is

fnsdfml = How often respondent faced severe financial difficulties growing up

[26]:
```
df_cleaned['hltpral_mapped'] = df_cleaned['hltpral'].map({0: 'No', 1: 'Yes'})

categories = ['domicil', 'hinctnta', 'cgtsmok', 'alcbnge']
values_allergy = df_cleaned[df_cleaned['hltpral_mapped'] == 'Yes'][categories].
 ↪mean()
values_no_allergy = df_cleaned[df_cleaned['hltpral_mapped'] ==␣
 ↪'No'][categories].mean()

angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False).tolist()

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))
ax.plot(angles, values_allergy, label='Allergy', color='red')
ax.plot(angles, values_no_allergy, label='No Allergy', color='blue')

ax.fill(angles, values_allergy, color='red', alpha=0.25)
ax.fill(angles, values_no_allergy, color='blue', alpha=0.25)

ax.set_yticklabels([])
ax.set_xticks(angles)
ax.set_xticklabels(categories)

plt.title('Comparison of Lifestyle Factors with Presence of Allergies')
plt.legend(loc='upper right')
plt.show()
```



hinctnta = Total household net income

cgtsmok = How often respondent smokes cigarettes

domicil = Type of home respondent lives in

alcbnge = How often respondent binge drinks alcohol

The next visualization is a Cramer's V correlation heatmap, showing the correlation of each feature with the presence of allergies, calculated using confusion matrices. The visual is in the form of a bar chart, with features listed in descending order of correlation found. This gives us an idea for which features are most predictive of allergies, and can help us narrow down our list during feature engineering.

[27]:
```
def cramers_v(confusion_matrix):
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()  # Ensure it's the total sum of the matrix
```

```
    phi2 = chi2 / n
    r, k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min((kcorr-1), (rcorr-1)))

# Create a new DataFrame for analysis
df_analysis = df_cleaned.copy()

# Specify the features and target variable
features = ['eatveg', 'dosprt', 'hinctnta', 'etfruit', 'hltprnt', 'chldhhe',␣
↪'hltprbn', 'domicil', 'happy',
            'health', 'alcfreq', 'cgtsmok', 'fnsdfml', 'rlgdgr', 'pray']  #␣
↪Replace with your categorical feature list
target = 'hltpral_mapped'  # Replace with your allergy status column name

# Ensure features and target are categorical in the new DataFrame
for col in features + [target]:
    if not pd.api.types.is_categorical_dtype(df_analysis[col]):
        df_analysis[col] = df_analysis[col].astype('category')

# Calculate Cramér's V for each feature
cramers_v_results = {}
for feature in features:
    confusion_matrix = pd.crosstab(df_analysis[feature], df_analysis[target])
    print(f"Confusion matrix for {feature}:\n{confusion_matrix}\n")  # Debugging
    cramers_v_results[feature] = cramers_v(confusion_matrix)

# Convert results to a DataFrame for visualization
cramers_v_df = pd.DataFrame(list(cramers_v_results.items()),␣
↪columns=['Feature', 'Cramers_V'])

# Sort by Cramér's V values
cramers_v_df = cramers_v_df.sort_values(by='Cramers_V', ascending=False)

# Plot the results
plt.figure(figsize=(8, 5))
sns.barplot(x='Cramers_V', y='Feature', data=cramers_v_df, palette='coolwarm')
plt.title("Cramér's V for Features vs Allergy Status", fontsize=16)
plt.xlabel("Cramér's V (Strength of Association)", fontsize=12)
plt.ylabel("Feature", fontsize=12)
plt.tight_layout()
plt.show()
```

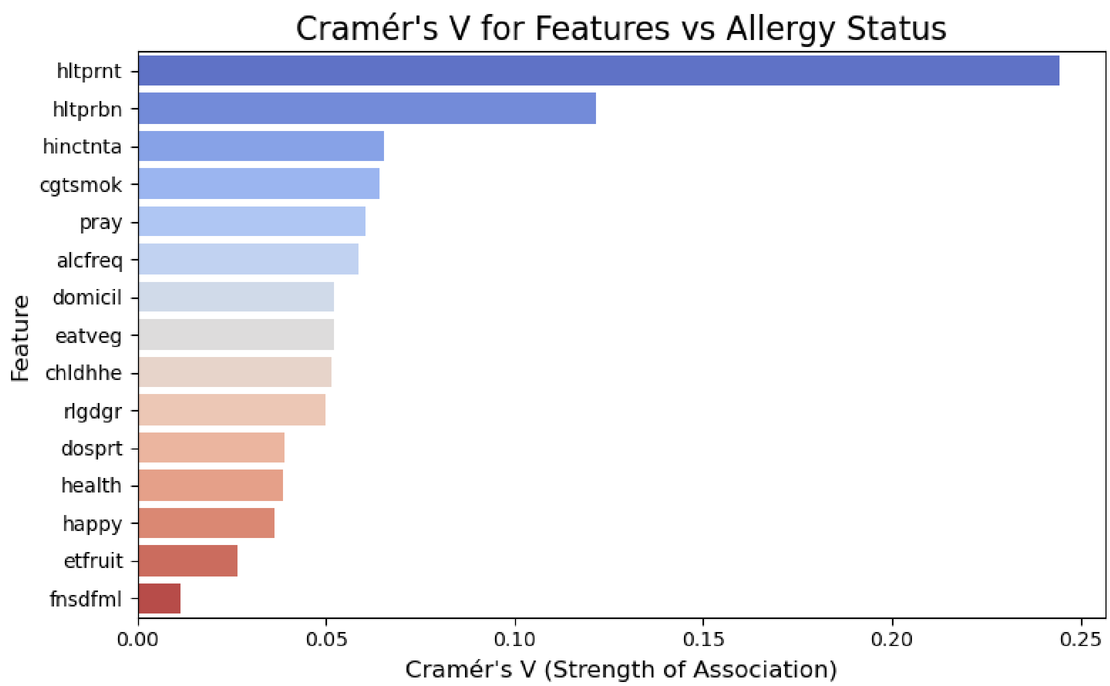
```
C:\Users\Emilia\AppData\Local\Temp\ipykernel_9752\3031511525.py:21:
DeprecationWarning: is_categorical_dtype is deprecated and will be removed in a
```

```
future version. Use isinstance(dtype, pd.CategoricalDtype) instead
  if not pd.api.types.is_categorical_dtype(df_analysis[col]):

Confusion matrix for eatveg:
hltpral_mapped    No   Yes
eatveg
1.0              966   201
2.0             3319   585
3.0             9199  1268
4.0             3786   476
5.0             1603   176
6.0              440    55
7.0              102    14

Confusion matrix for dosprt:
hltpral_mapped    No  Yes
dosprt
0.0             3866  429
1.0             1329  222
2.0             2237  338
3.0             2440  396
4.0             1806  282
5.0             1946  255
6.0              814  112
7.0             4977  741

Confusion matrix for hinctnta:
hltpral_mapped    No  Yes
hinctnta
1.0             1313  182
2.0             1579  205
3.0             1702  234
4.0             1599  232
5.0             1606  273
6.0             5450  566
7.0             1670  284
8.0             1659  287
9.0             1377  231
10.0            1460  281

Confusion matrix for etfruit:
hltpral_mapped    No   Yes
etfruit
1.0             1298   208
2.0             3855   597
3.0             7671  1003
4.0             3539   488
5.0             1855   306
```

```
6.0               996   135
7.0               201    38

Confusion matrix for hltprnt:
hltpral_mapped     No   Yes
hltprnt
0               12868  2775
1                6547     0

Confusion matrix for chldhhe:
hltpral_mapped     No   Yes
chldhhe
1.0             13374  1708
2.0              6041  1067

Confusion matrix for hltprbn:
hltpral_mapped     No   Yes
hltprbn
0               12923  1357
1                6492  1418

Confusion matrix for domicil:
hltpral_mapped    No  Yes
domicil
1.0             3552  505
2.0             2142  387
3.0             5259  873
4.0             7100  852
5.0             1362  158

Confusion matrix for happy:
hltpral_mapped    No  Yes
happy
0.0               64   11
1.0               43   13
2.0              177   28
3.0              358   54
4.0              496   53
5.0             1293  174
6.0             1481  183
7.0             3371  459
8.0             5874  920
9.0             3864  607
10.0            2394  273

Confusion matrix for health:
hltpral_mapped    No   Yes
health
```

```
1.0            4777   570
2.0            8483  1200
3.0            4798   754
4.0            1157   212
5.0             200    39

Confusion matrix for alcfreq:
hltpral_mapped    No  Yes
alcfreq
1.0             915   91
2.0            2938  431
3.0            3133  473
4.0            2504  440
5.0            1632  260
6.0            2974  502
7.0            5319  578

Confusion matrix for cgtsmok:
hltpral_mapped    No   Yes
cgtsmok
1.0            2455   244
2.0            1094   148
3.0             761   131
4.0            4193   645
5.0            1889   404
6.0            9023  1203

Confusion matrix for fnsdfml:
hltpral_mapped    No  Yes
fnsdfml
1.0             594   94
2.0            2358  326
3.0            5304  708
4.0            4623  657
5.0            6536  990

Confusion matrix for rlgdgr:
hltpral_mapped    No  Yes
rlgdgr
0.0            3651  584
1.0            1054  220
2.0            1433  210
3.0            1411  242
4.0            1113  166
5.0            2732  327
6.0            1847  260
7.0            2213  279
8.0            1988  226
```

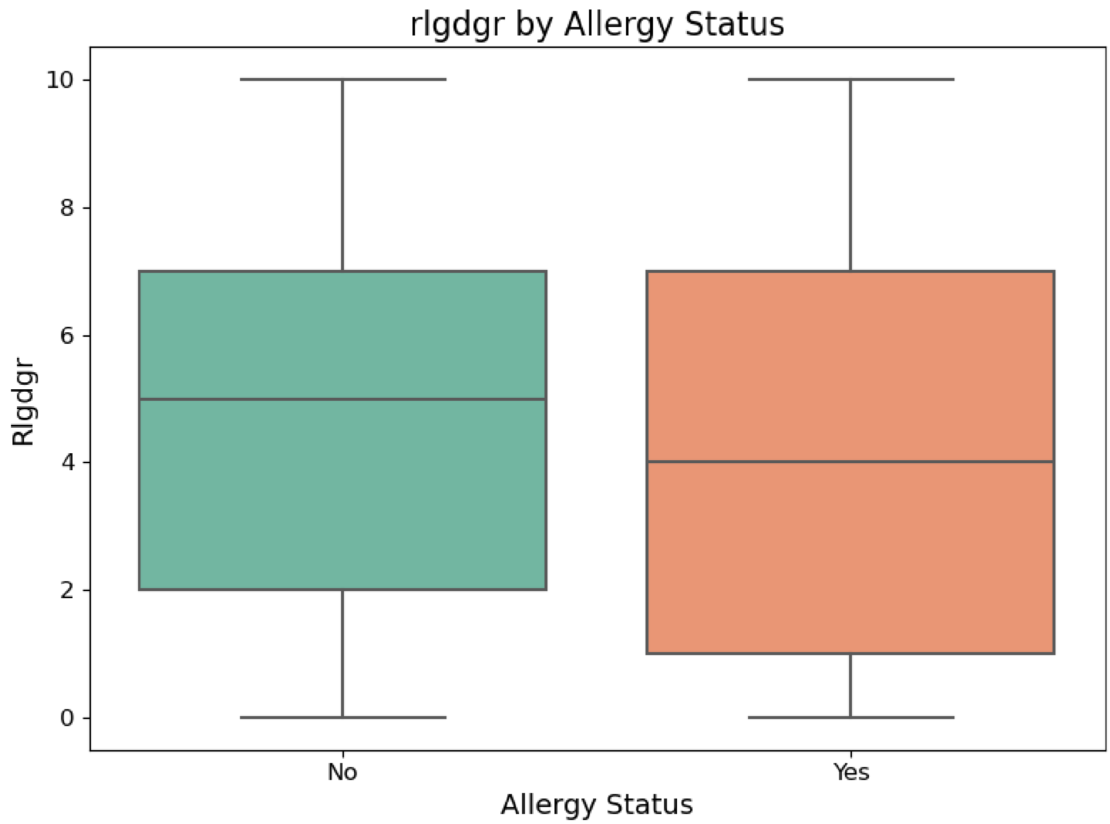
```
9.0               790  109
10.0             1183  152

Confusion matrix for pray:
hltpral_mapped     No   Yes
pray
1.0              3563   461
2.0              1580   161
3.0              1235   148
4.0              1253   159
5.0              1151    93
6.0              3043   517
7.0              7590  1236
```

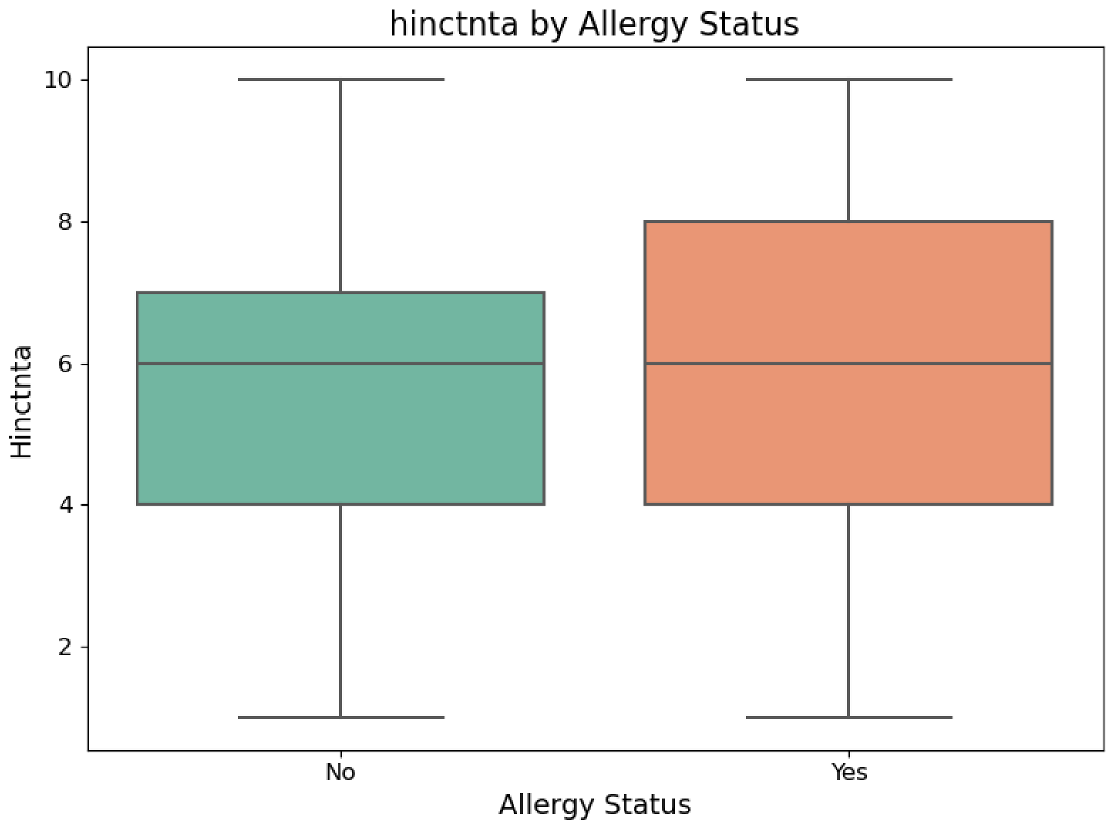


As we can see from the chart, the features with the highest correlations were hltprnt (presence of health issues), hltprbn (back or neck pain), hinctnta (total household income), cgtsmok (smoking habits), alcohol frequency, frequency of prayer, and type of area lived in (e.g. big city, country village).

The boxplots below show the distribution between various features and whether there was a difference between those who had allergies and those who did not. As we can see below, there's little to no difference between overall happiness, but there was a larger variation among rlgdgr (religiousness) and among hinctnta (total household income). Those with allergies have a larger range of respondents and more of whom identified as non-religious, and there were more respondents in the allergy category who identified as having larger incomes.

[28]:
```
selected_features = ['happy', 'rlgdgr', 'hinctnta']

# Plot boxplots for each feature
for feature in selected_features:
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='hltpral_mapped', y=feature, data=df_cleaned, palette='Set2')
    plt.title(f"{feature} by Allergy Status", fontsize=16)
    plt.xlabel("Allergy Status", fontsize=14)
    plt.ylabel(feature.capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.tight_layout()
    plt.show()
```


rlgdgr by Allergy Status
Rlgdgr
0
2
4
6
8
10
No
Yes
Allergy Status



happy = How happy respondent is overall

rlgdgr = How religious respondent is

hinctnta = Total household net income

Categories are ordinal on scale of 1-10

The next visualizations are bar charts showing the distribution of survey responses for all categories, for a selected group of features that were deemed important (determined from previous visualizations). I included legends to show what each numeric category represents, for easier interpretation.

[29]:
```
import matplotlib.patches as mpatches

value_mappings = {
    'etfruit': {  # How often eat fruit
        1: 'Three times or more a day',
        2: 'Twice a day',
        3: 'Once a day',
        4: 'Less than once a day but at least 4 times a week',
        5: 'Less than 4 times a week but at least once a week',
        6: 'Less than once a week',
        7: 'Never'
```

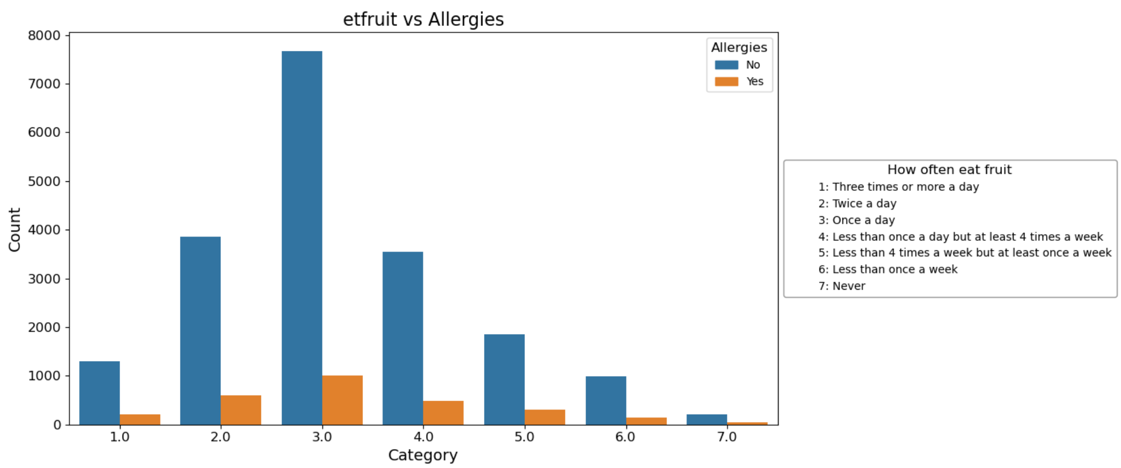
```
    },
    'eatveg': {  # How often eat vegetables or salad, excluding potatoes
        1: 'Three times or more a day',
        2: 'Twice a day',
        3: 'Once a day',
        4: 'Less than once a day but at least 4 times a week',
        5: 'Less than 4 times a week but at least once a week',
        6: 'Less than once a week',
        7: 'Never'
    },
    'dosprt': {  # How often do sports
        1: 'One day per week',
        2: 'Two days per week',
        3: 'Three days per week',
        4: 'Four days per week',
        5: 'Five days per week',
        6: 'Six days per week',
        7: 'Seven days per week'
    },
    'health': {  # Subjective general health
        1: 'Very good',
        2: 'Good',
        3: 'Fair',
        4: 'Bad',
        5: 'Very bad'
    },
    'happy': {  # General happiness level
        0: 'Extremely unhappy',
        1: '1',
        2: '2',
        3: '3',
        4: '4',
        5: '5',
        6: '6',
        7: '7',
        8: '8',
        9: '9',
        10: 'Extremely happy'
    },
    'cgtsmok': {  # Cigarette smoking behavior
        1: 'I smoke daily, 10 or more cigarettes',
        2: 'I smoke daily, 9 or fewer cigarettes',
        3: 'I smoke but not every day',
        4: 'I don't smoke now but I used to',
        5: 'I have only smoked a few times',
        6: 'I have never smoked',
    },
```

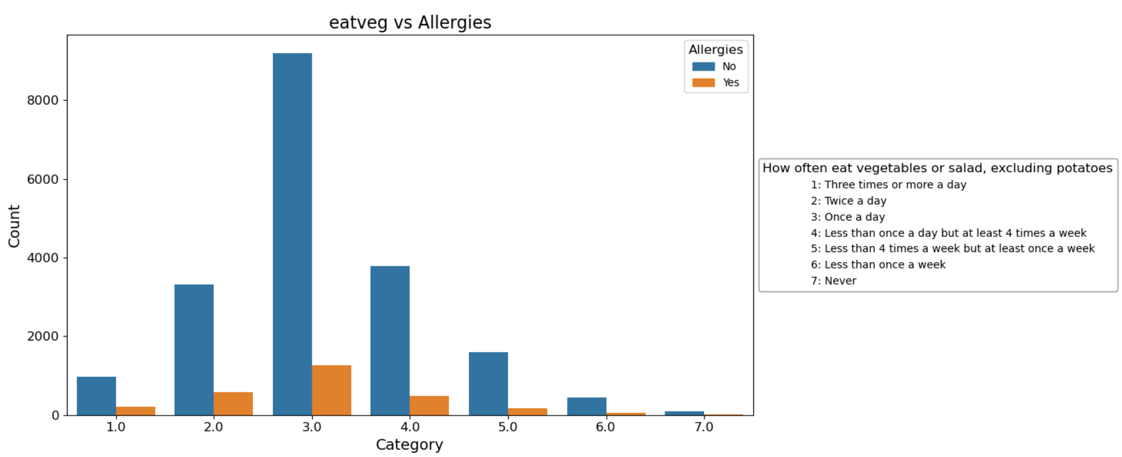
```
'alcfreq': {  # Frequency of alcohol consumption
    1: 'Every day',
    2: 'Several times a week',
    3: 'Once a week',
    4: '2-3 times a month',
    5: 'Once a month',
    6: 'Less than once a month',
    7: 'Never'
},
'alcbnge': {  # Frequency of alcohol bingeing
    1: 'Daily or almost daily',
    2: 'Weekly',
    3: 'Monthly',
    4: 'Less than monthly',
    5: 'Never',
},
'dshltms': {  # Discussed health with medical specialist
    0: 'Not marked',
    1: 'Marked'
},
'hltprnt': {  # No health problems
    0: 'Not marked',
    1: 'Marked'
},
'hltprbn': {  # Back or neck pain
    0: 'Not marked',
    1: 'Marked'
},
'hltprhb': {  # High blood pressure
    0: 'Not marked',
    1: 'Marked'
},
'hltprsc': {  # Skin condition
    0: 'Not marked',
    1: 'Marked'
},
'hltprbp': {  # Breathing problems
    0: 'Not marked',
    1: 'Marked'
},
'domicil': {  # Type of home
    1: 'A big city',
    2: 'Suburbs or outskirts of big city',
    3: 'Town or small city',
    4: 'Country village',
    5: 'Farm or home in countryside'
},
```

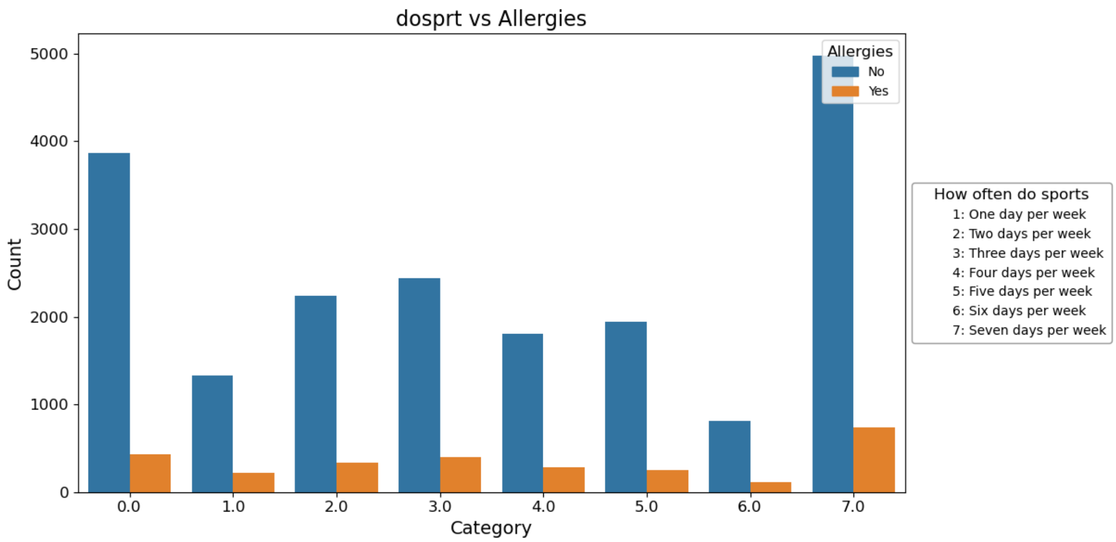
```python
    'chldhhe': {  # Ever had children living at home
        1: 'Yes',
        2: 'No',
    },
    'fnsdfml': {  # Severe financial difficulty growing up
        1: 'Always',
        2: 'Often',
        3: 'Sometimes',
        4: 'Hardly ever',
        5: 'Never'
    },
    'rlgdgr': {  # How religious are you
        0: 'Not at all religious',
        1: '1',
        2: '2',
        3: '3',
        4: '4',
        5: '5',
        6: '6',
        7: '7',
        8: '8',
        9: '9',
        10: 'Very religious'
    }
}

legend_titles = {
    'etfruit': 'How often eat fruit',
    'eatveg': 'How often eat vegetables or salad, excluding potatoes',
    'dosprt': 'How often do sports',
    'health': 'Subjective general health',
    'happy': 'General happiness level',
    'cgtsmok': 'Cigarette smoking behavior',
    'alcfreq': 'Frequency of alcohol consumption',
    'alcbnge': 'Frequency of alcohol bingeing',
    'dshltms': 'Discussed health with medical specialist',
    'hltprnt': 'No health problems',
    'hltprbn': 'Back or neck pain',
    'hltprhb': 'High blood pressure',
    'hltprsc': 'Skin condition',
    'hltprbp': 'Breathing problems',
    'domicil': 'Type of home',
    'chldhhe': 'Ever had children living at home',
    'fnsdfml': 'Severe financial difficulty growing up',
    'rlgdgr': 'How religious are you'
}
```

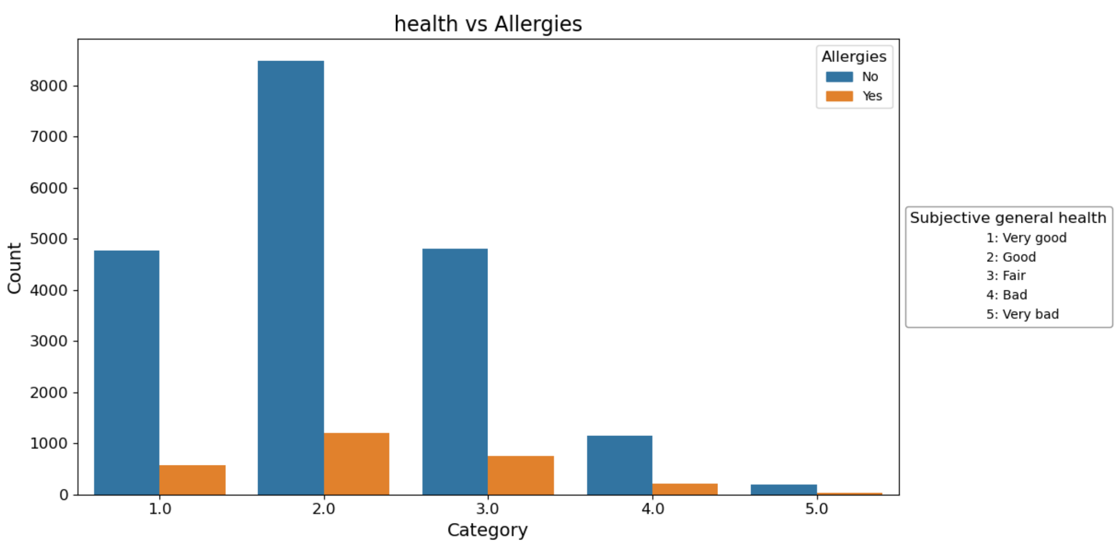
```
# Use the mapped column as the hue for the plots
for column in ['etfruit', 'eatveg', 'dosprt', 'health', 'happy', 'cgtsmok',␣
↪'alcfreq',
               'alcbnge', 'dshltms', 'hltprnt', 'hltprbn', 'hltprhb',␣
↪'hltprsc', 'hltprbp',
               'domicil', 'chldhhe', 'fnsdfml', 'rlgdgr']:  # Adjust this list␣
↪to focus on relevant columns
    if column != 'hltpral':  # Skip the target column itself
        # Create the figure and axes
        fig, ax = plt.subplots(figsize=(10, 6))

        # Create the countplot
        sns.countplot(x=column, hue='hltpral_mapped', data=df_cleaned, ax=ax)
        ax.set_title(f"{column} vs Allergies", fontsize=16)  # Increase title␣
↪font size
        ax.set_xlabel("Category", fontsize=14)  # Increase x-axis label font␣
↪size
        ax.set_ylabel("Count", fontsize=14)  # Increase y-axis label font size
        ax.tick_params(axis='both', which='major', labelsize=12)  # Increase␣
↪tick label size

        # Remove the default legend (to fully customize it)
        ax.legend_.remove()

        # Extract the unique colors for "No" and "Yes" from the first set of␣
↪bars
        bars = ax.patches  # Get all the bars in the plot
        no_color = bars[0].get_facecolor()  # First set of bars for "No"
        yes_color = bars[len(bars) // 2].get_facecolor()  # Second set of bars␣
↪for "Yes"

        # Define the primary legend for x-axis categories
        x_categories = value_mappings[column]  # Get the dictionary for the␣
↪current column
        x_handles = [mpatches.Patch(color='white', label=f"{k}: {v}") for k, v␣
↪in x_categories.items()]
        primary_legend = plt.legend(handles=x_handles, title=legend_titles.
↪get(column, "Categories"),
                                    fontsize=10, title_fontsize=12, loc='center␣
↪left', bbox_to_anchor=(1.0, 0.5),
                                    frameon=True, edgecolor='gray')

        # Define the mini legend for allergies with matching colors
        allergy_handles = [
            mpatches.Patch(color=no_color, label='No'),  # Match "No" color
            mpatches.Patch(color=yes_color, label='Yes')  # Match "Yes" color
```

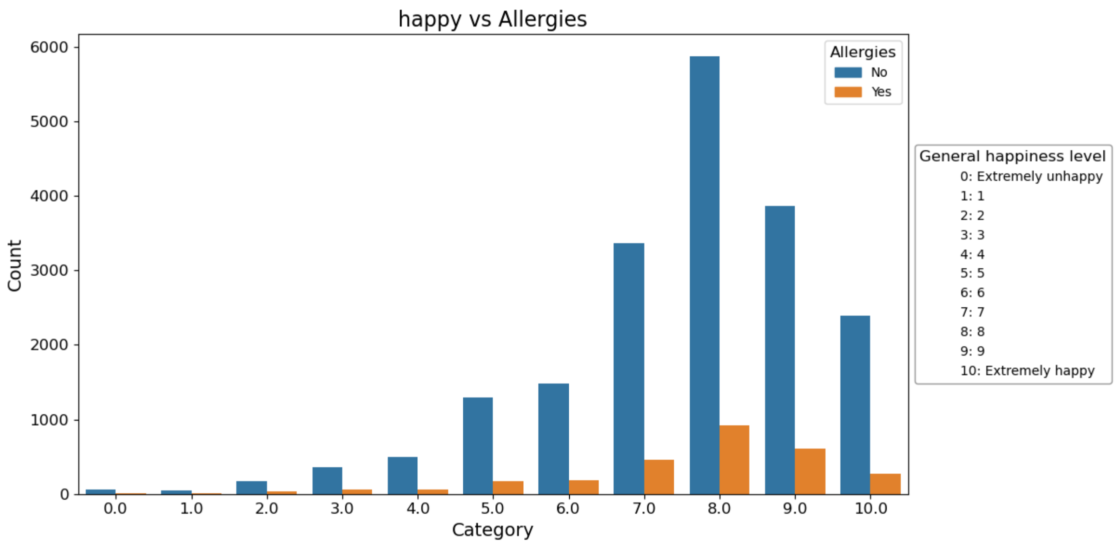
```
        ]
        mini_legend = ax.legend(handles=allergy_handles, title="Allergies",
                                fontsize=10, title_fontsize=12, loc='upper␣
↪right')

        # Re-add the primary legend after adding the mini legend
        fig.add_artist(primary_legend)

        # Adjust the layout and display the plot
        plt.tight_layout()
        plt.show()
```

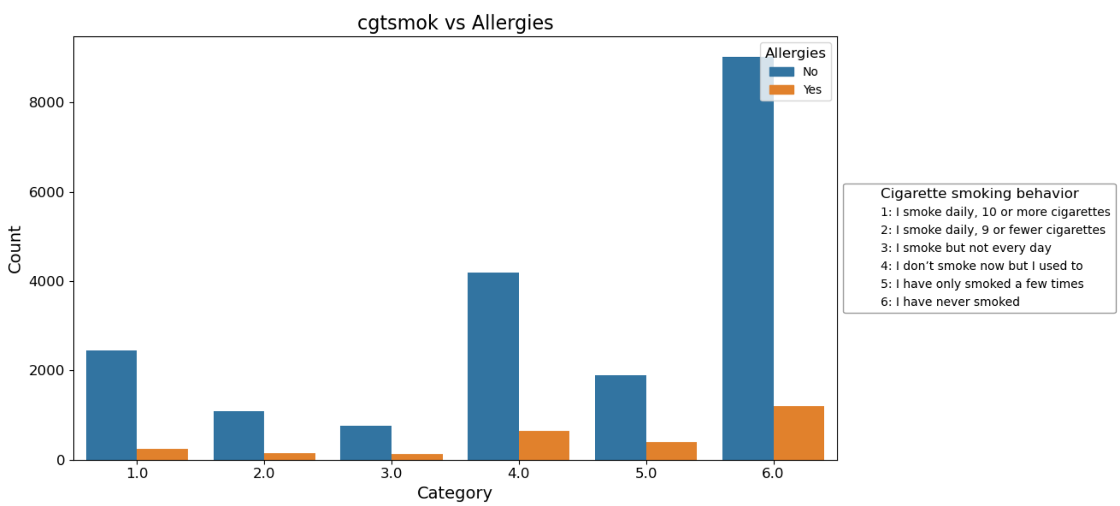


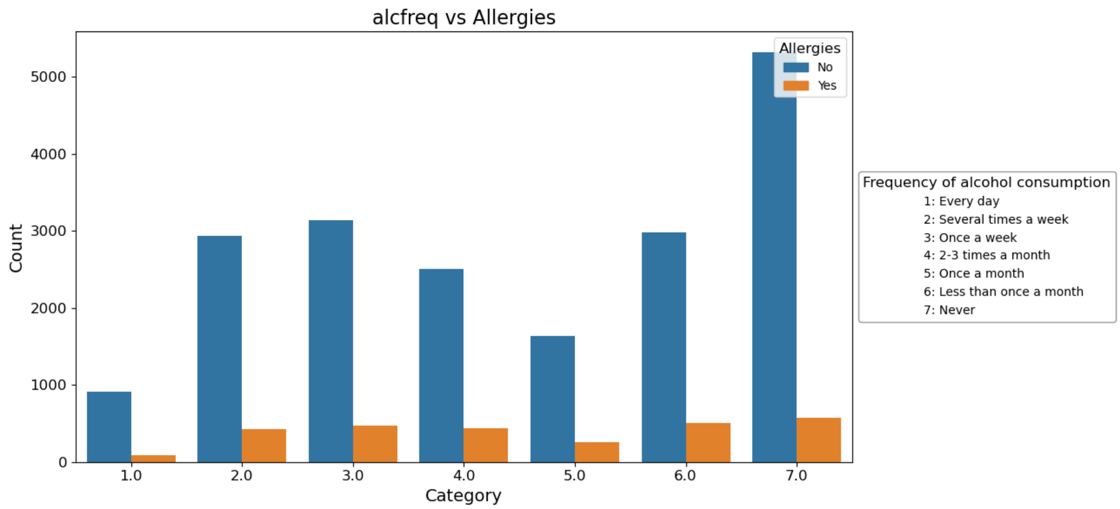


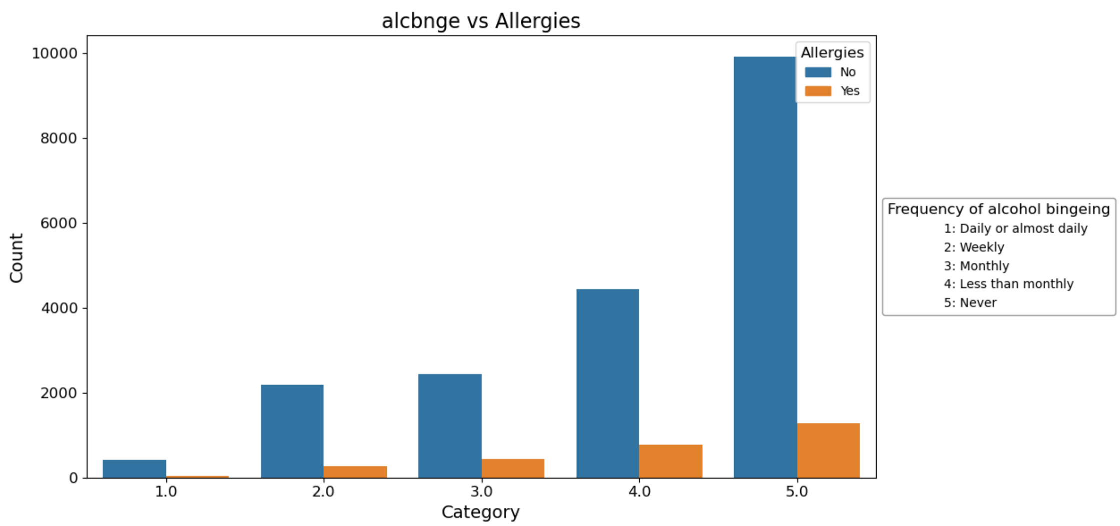

dosprt vs Allergies
Allergies
No
Yes
5000
4000
3000
2000
1000
0
Count
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
Category
How often do sports
1: One day per week
2: Two days per week
3: Three days per week
4: Four days per week
5: Five days per week
6: Six days per week
7: Seven days per week

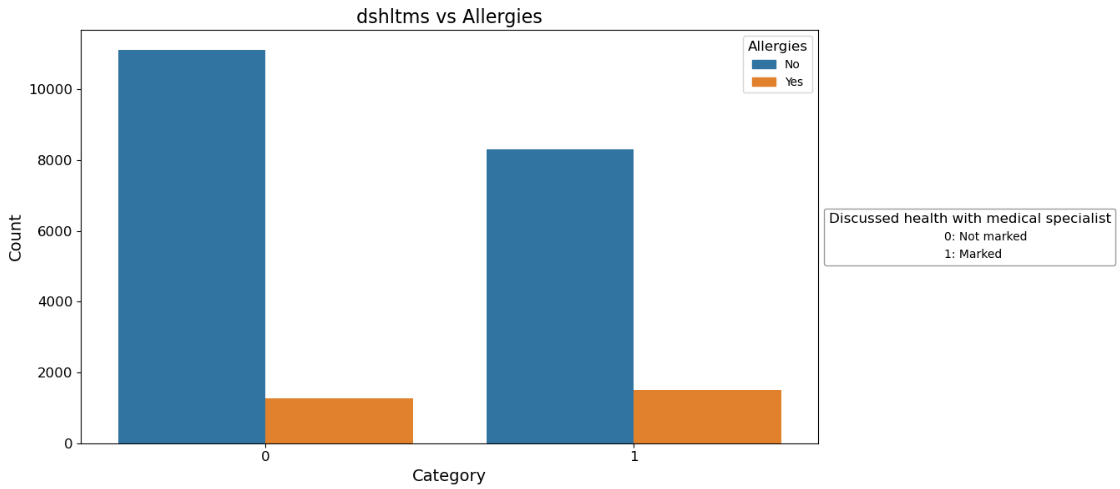

health vs Allergies
Allergies
No
Yes
8000
7000
6000
5000
4000
3000
2000
1000
0
Count
1.0
2.0
3.0
4.0
5.0
Category
Subjective general health
1: Very good
2: Good
3: Fair
4: Bad
5: Very bad

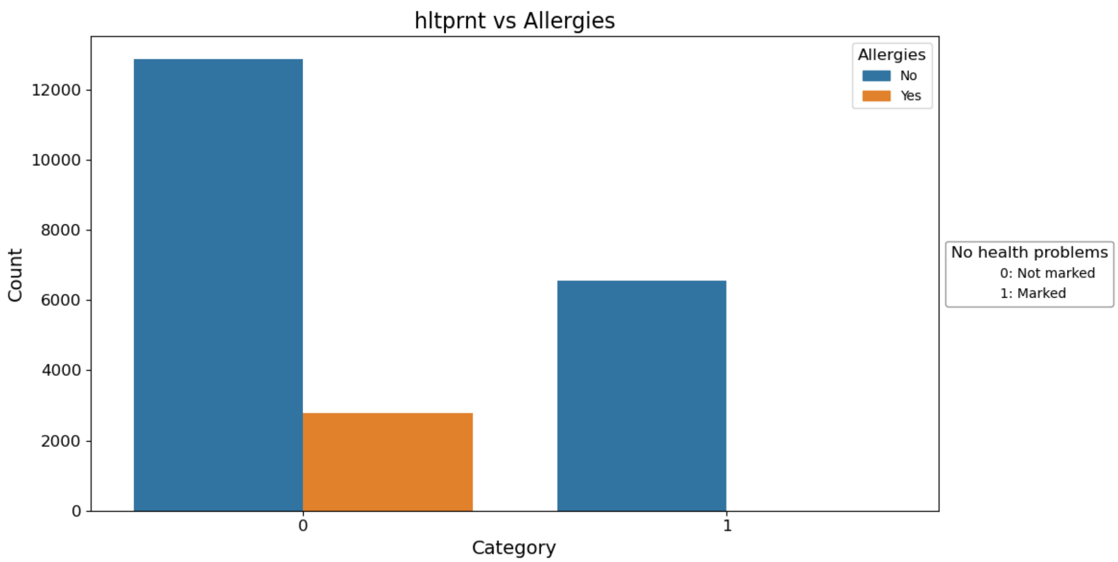

happy vs Allergies
Allergies
No
Yes
General happiness level
0: Extremely unhappy
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
10: Extremely happy
Count
Category
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0

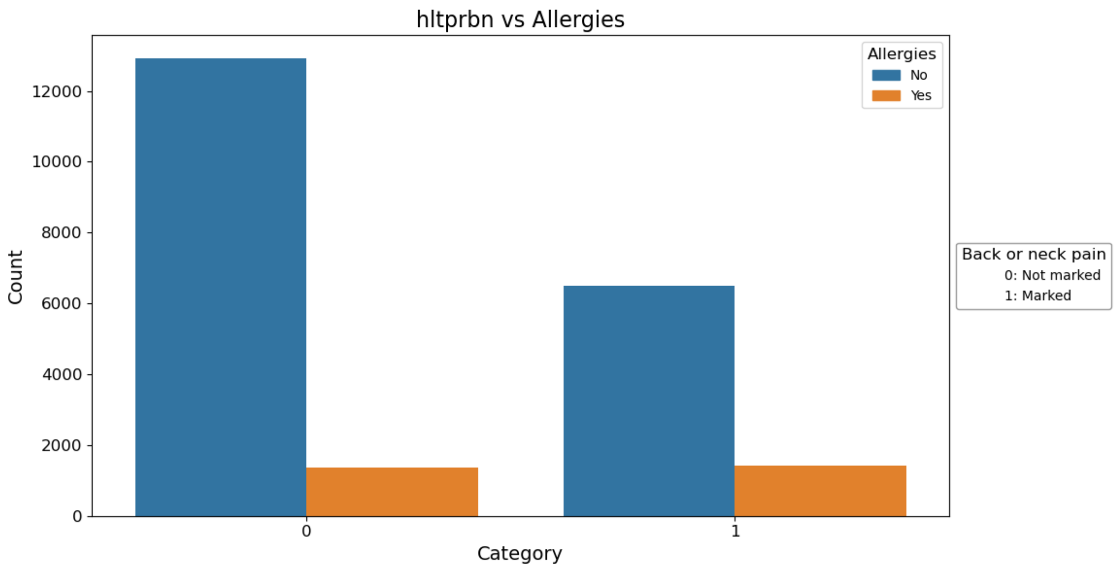

cgtsmok vs Allergies
Allergies
No
Yes
Cigarette smoking behavior
1: I smoke daily, 10 or more cigarettes
2: I smoke daily, 9 or fewer cigarettes
3: I smoke but not every day
4: I don't smoke now but I used to
5: I have only smoked a few times
6: I have never smoked
Count
Category
1.0
2.0
3.0
4.0
5.0
6.0

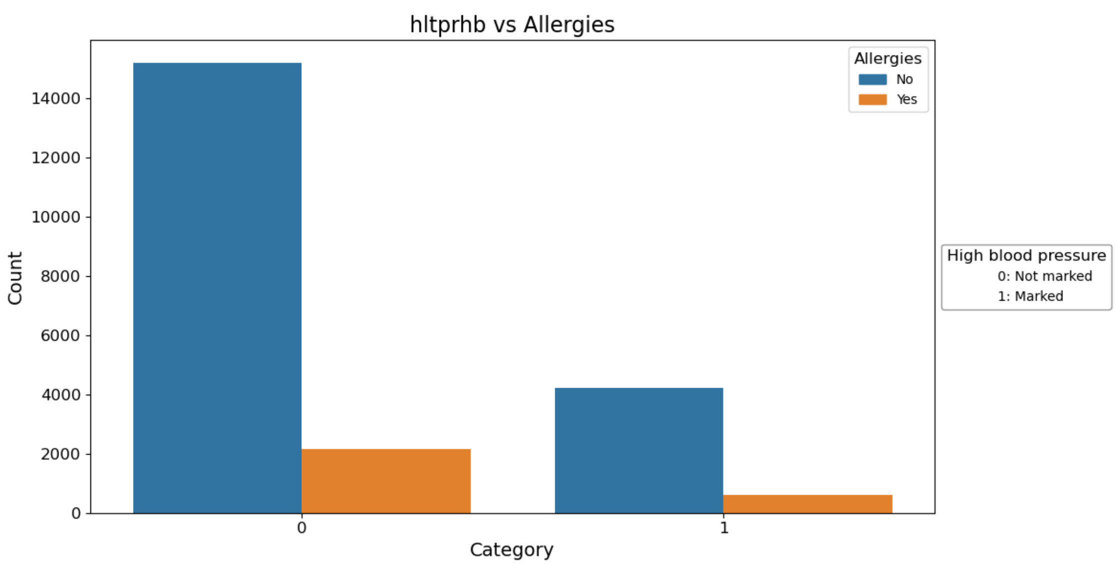

alcfreq vs Allergies
Allergies
No
Yes
Count
Category
1.0
2.0
3.0
4.0
5.0
6.0
7.0
0
1000
2000
3000
4000
5000
Frequency of alcohol consumption
1: Every day
2: Several times a week
3: Once a week
4: 2-3 times a month
5: Once a month
6: Less than once a month
7: Never

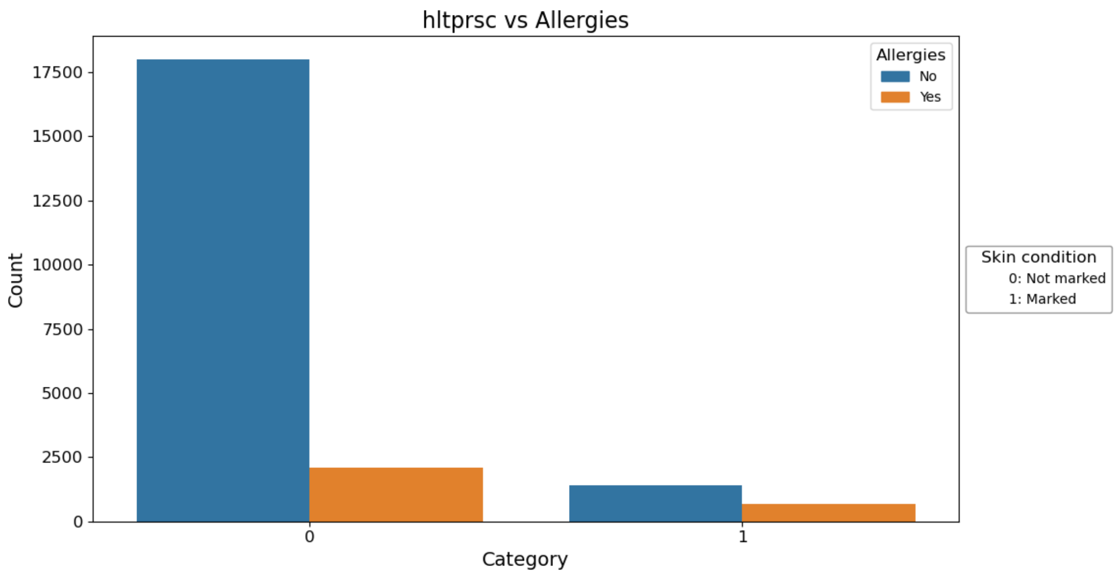

alcbnge vs Allergies
Allergies
No
Yes
Count
Category
1.0
2.0
3.0
4.0
5.0
0
2000
4000
6000
8000
10000
Frequency of alcohol bingeing
1: Daily or almost daily
2: Weekly
3: Monthly
4: Less than monthly
5: Never

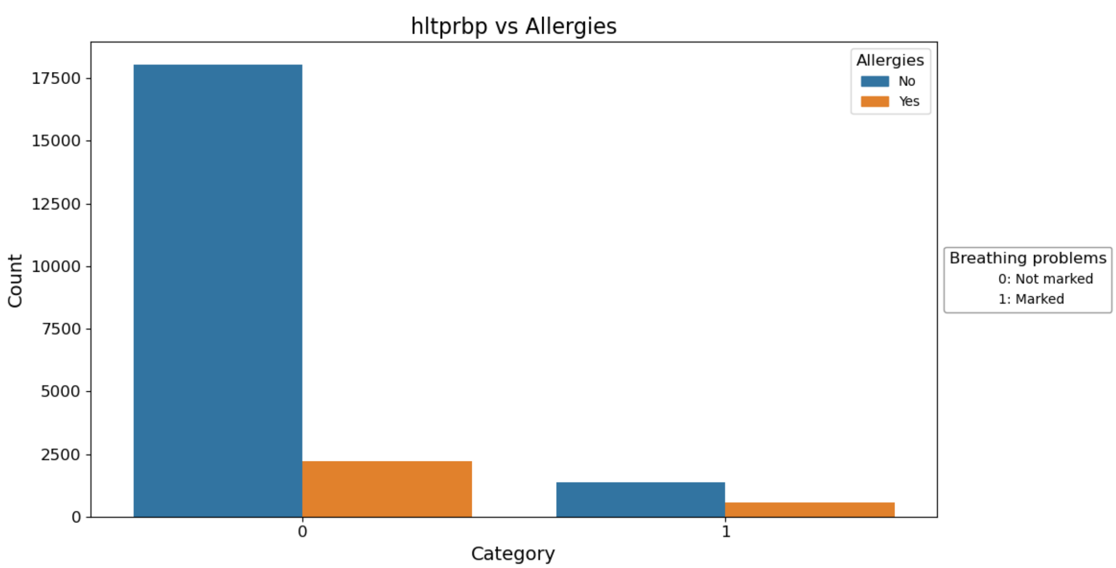

dshltms vs Allergies
Allergies
No
Yes
Count
Category
0
1
Discussed health with medical specialist
0: Not marked
1: Marked

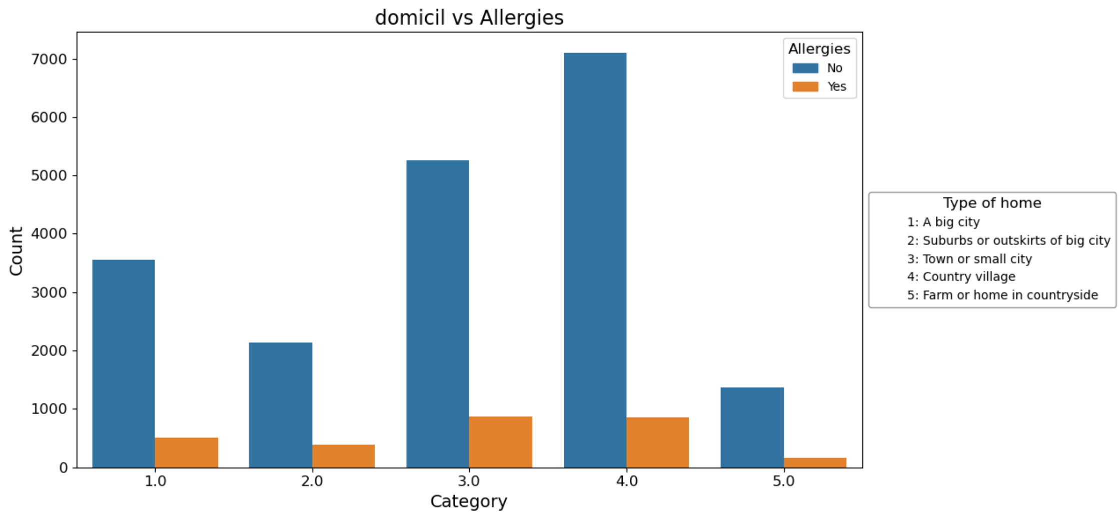

hltprnt vs Allergies
Allergies
No
Yes
Count
Category
0
1
No health problems
0: Not marked
1: Marked

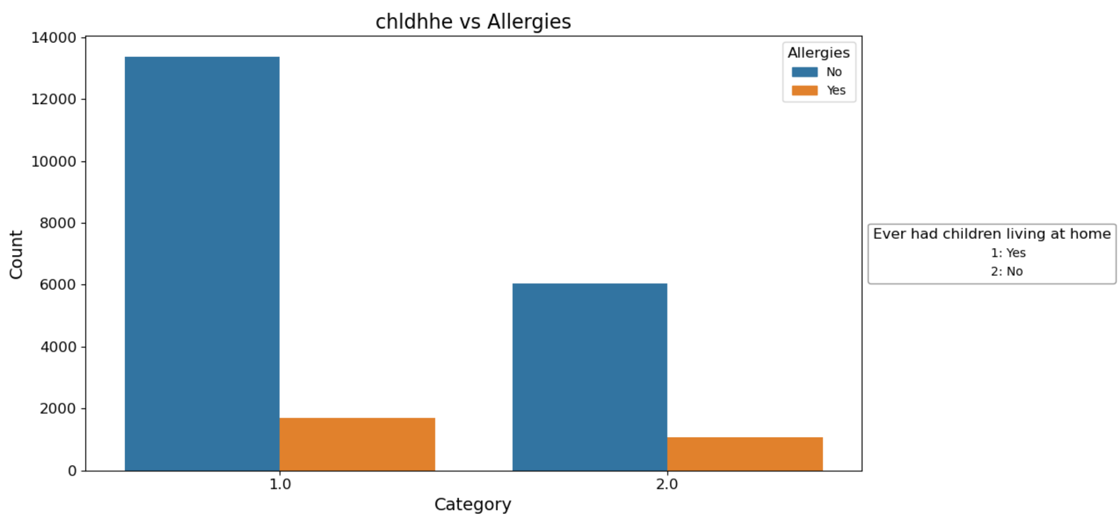

hltprbn vs Allergies
Allergies
No
Yes
Back or neck pain
0: Not marked
1: Marked
Count
0
2000
4000
6000
8000
10000
12000
0
1
Category

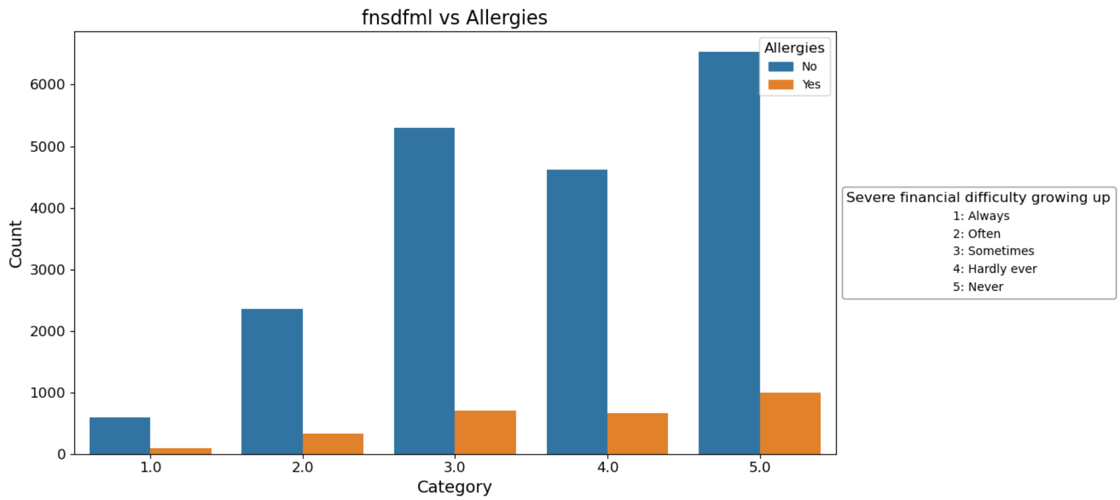

hltprhb vs Allergies
Allergies
No
Yes
High blood pressure
0: Not marked
1: Marked
Count
0
2000
4000
6000
8000
10000
12000
14000
0
1
Category

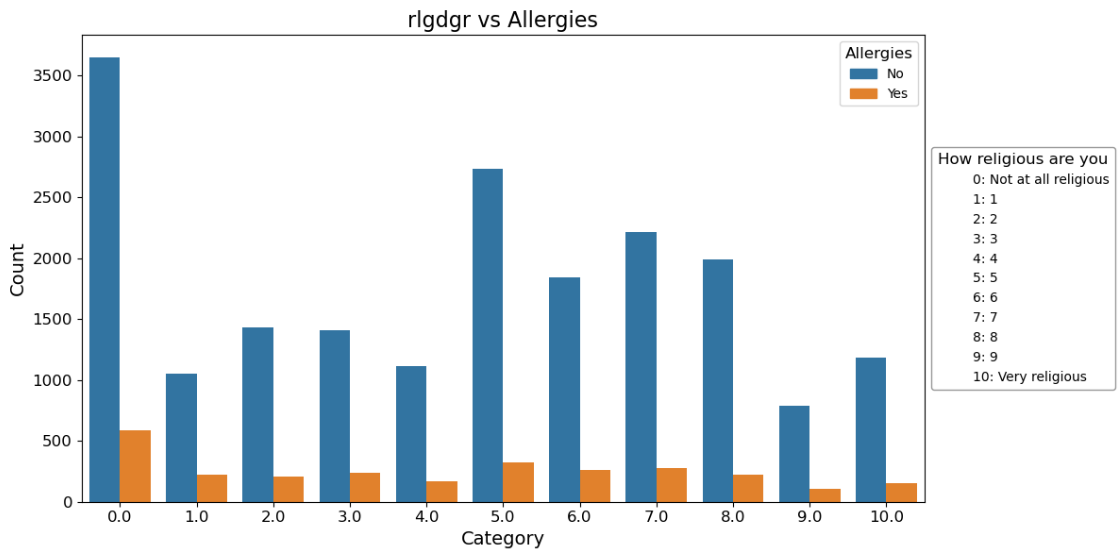

hltprsc vs Allergies
Allergies
No
Yes
Skin condition
0: Not marked
1: Marked
Count
17500
15000
12500
10000
7500
5000
2500
0
0
1
Category


hltprbp vs Allergies
Allergies
No
Yes
Breathing problems
0: Not marked
1: Marked
Count
17500
15000
12500
10000
7500
5000
2500
0
0
1
Category


domicil vs Allergies
Allergies
No
Yes
Count
Category
1.0
2.0
3.0
4.0
5.0
0
1000
2000
3000
4000
5000
6000
7000
Type of home
1: A big city
2: Suburbs or outskirts of big city
3: Town or small city
4: Country village
5: Farm or home in countryside


chldhhe vs Allergies
Allergies
No
Yes
Count
Category
1.0
2.0
0
2000
4000
6000
8000
10000
12000
14000
Ever had children living at home
1: Yes
2: No





I created a table summarizing the counts for each category of every feature, showcasing the percentage of individuals within each category who have allergies versus those who do not.

[30]:
```
df_cleaned['hltpral_mapped'] = df_cleaned['hltpral'].map({0: 'No', 1: 'Yes'}) ␣
↪# Modify if necessary

# List of feature columns to iterate over
feature_columns = ['etfruit', 'eatveg', 'dosprt', 'cgtsmok', 'alcfreq',␣
↪'alcbnge', 'dshltgp']  # Add more if needed

# Create an empty list to store the results
all_data = []
```

```
# Iterate through each feature
for column in all_columns:
    # Calculate the total count and allergy counts (Yes and No) for each␣
↪category in the feature
    feature_counts = df_cleaned.groupby([column, 'hltpral_mapped']).size().
↪unstack(fill_value=0)
    total_counts = df_cleaned.groupby([column]).size()

    # Iterate over the unique categories in the feature
    for category in feature_counts.index:
        # Get the counts for allergy Yes and No
        allergy_yes = feature_counts.loc[category, 'Yes'] if 'Yes' in␣
↪feature_counts.columns else 0
        allergy_no = feature_counts.loc[category, 'No'] if 'No' in␣
↪feature_counts.columns else 0
        total_count = total_counts.get(category, 0)

        # Calculate the allergy percentages
        allergy_yes_percentage = (allergy_yes / total_count) * 100 if␣
↪total_count > 0 else 0
        allergy_no_percentage = (allergy_no / total_count) * 100 if total_count␣
↪> 0 else 0

        # Append the row with the calculated data
        all_data.append({
            'Feature': column,
            'Category': category,
            'Count': total_count,
            'Allergy Yes': allergy_yes,
            'Allergy No': allergy_no,
            'Allergy Yes %': allergy_yes_percentage,
            'Allergy No %': allergy_no_percentage
        })

# Create a DataFrame from the collected data
df_summary = pd.DataFrame(all_data)

# Display the resulting DataFrame
df_summary
```

[30]:

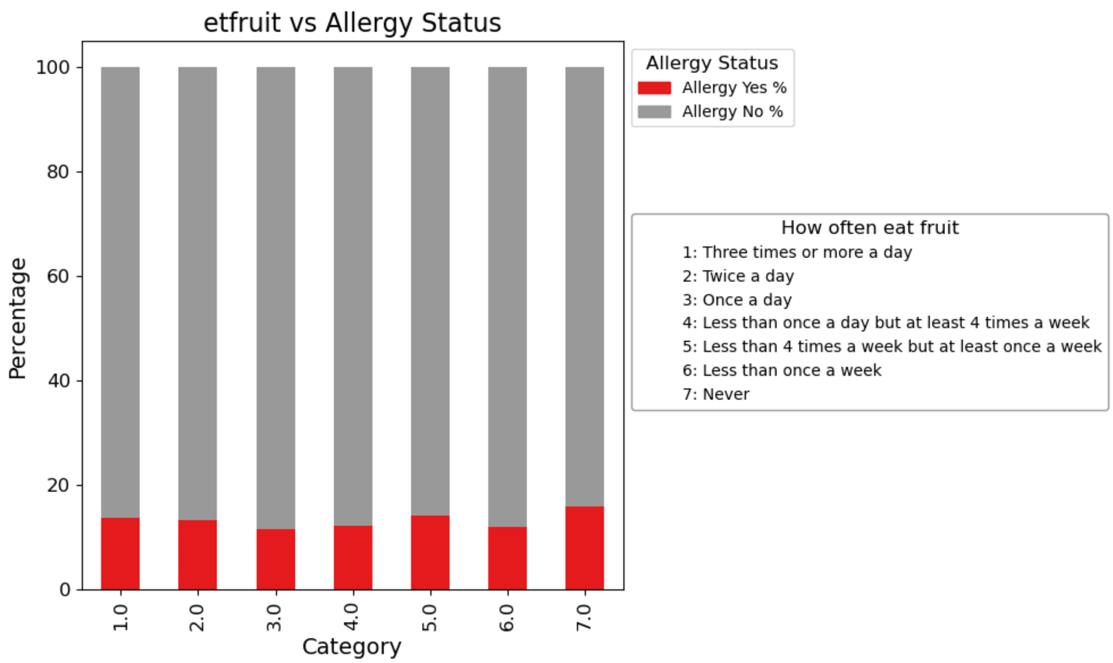
	Feature	Category	Count	Allergy Yes	Allergy No	Allergy Yes %	\
0	etfruit	1.0	1506	208	1298	13.811421	
1	etfruit	2.0	4452	597	3855	13.409704	
2	etfruit	3.0	8674	1003	7671	11.563293	
3	etfruit	4.0	4027	488	3539	12.118202	
4	etfruit	5.0	2161	306	1855	14.160111	

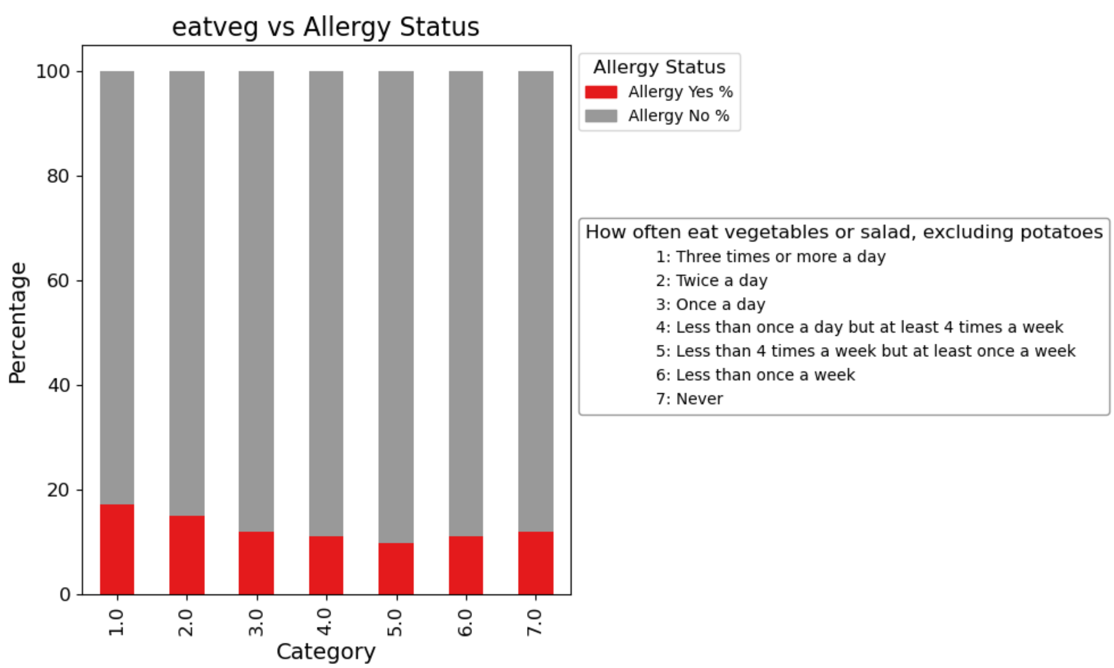
```
..              …      …   …            …         …          …
949         hinctnta     8.0   1946          287       1659      14.748201
950         hinctnta     9.0   1608          231       1377      14.365672
951         hinctnta    10.0   1741          281       1460      16.140149
952   hltpral_mapped      No  19415            0      19415       0.000000
953   hltpral_mapped     Yes   2775         2775          0     100.000000

     Allergy No %
0       86.188579
1       86.590296
2       88.436707
3       87.881798
4       85.839889
..            …
949     85.251799
950     85.634328
951     83.859851
952    100.000000
953      0.000000

[954 rows x 7 columns]
```

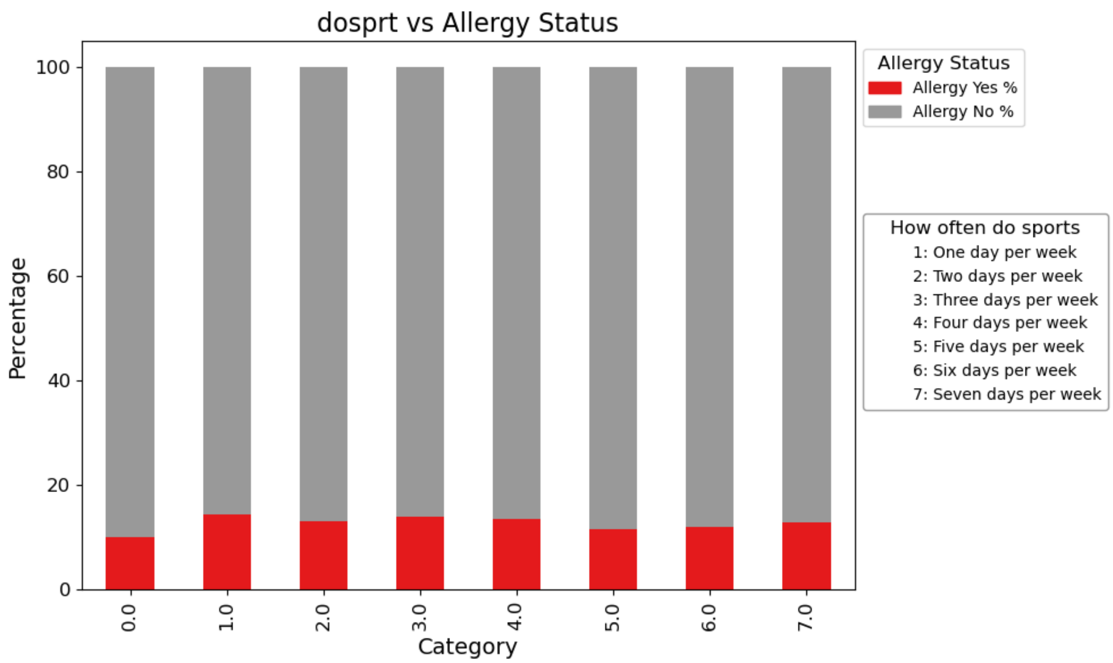
My final visualization is a stacked bar chart illustrating the percentage distribution of individuals with and without allergies across the categories for each feature. This provides a clear comparison of how allergy prevalence varies within each category of the features.

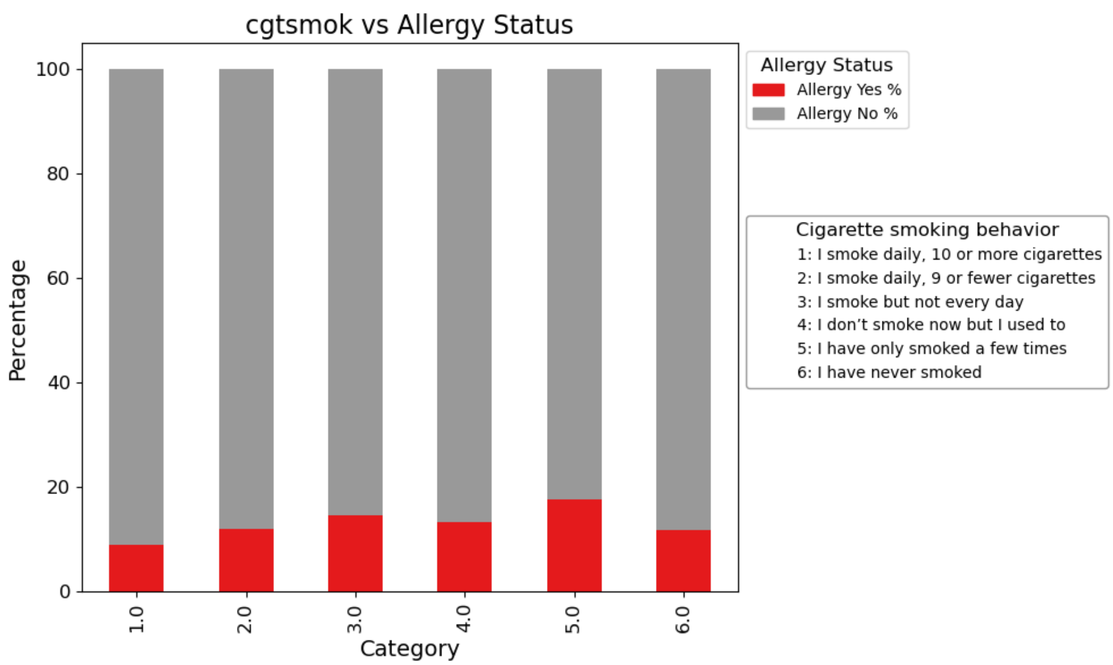
[31]:
```python
features_with_mappings = [feature for feature in df_summary['Feature'].unique()␣
↪if feature in value_mappings]

for feature in features_with_mappings:
    # Filter the rows for the current feature
    feature_data = df_summary[df_summary['Feature'] == feature]

    # Only select the percentage columns for plotting
    percentage_data = feature_data[['Category', 'Allergy Yes %', 'Allergy No␣
↪%']].set_index('Category')

    # Plot the stacked bar chart
    ax = percentage_data.plot(kind='bar', stacked=True, figsize=(10, 6),␣
↪colormap='Set1')

    # Customize the plot
    plt.title(f"{feature} vs Allergy Status", fontsize=16)
    plt.xlabel('Category', fontsize=14)
    plt.ylabel('Percentage', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
```

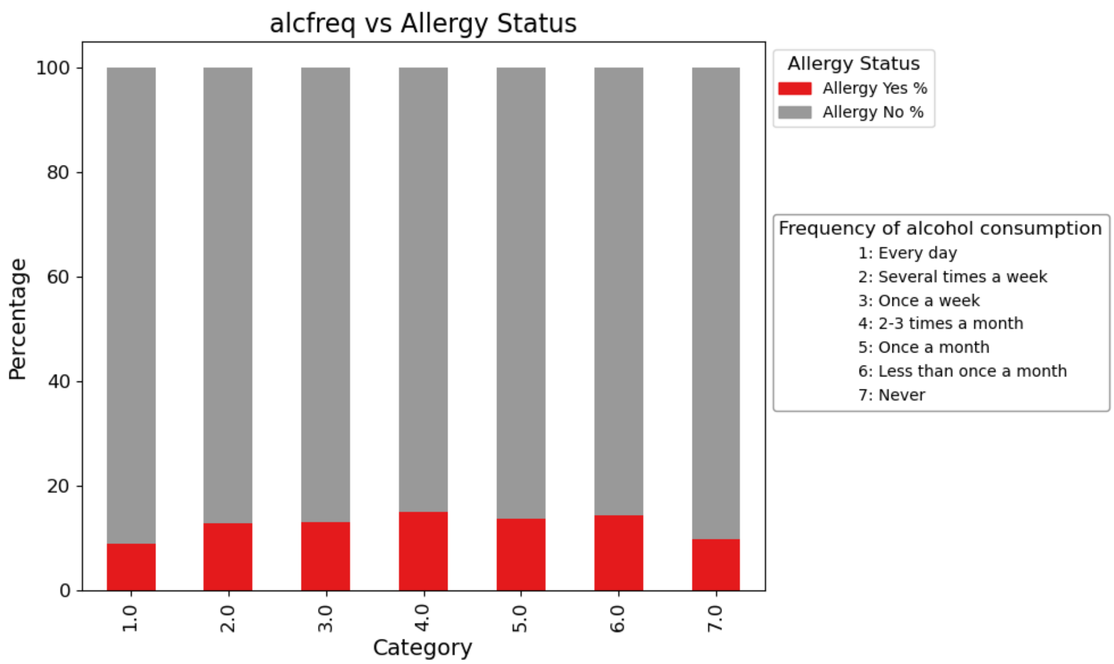
```python
    # Dynamically extract the colors for "Allergy Yes %" and "Allergy No %"␣
↪from the graph
    bars = ax.patches  # Get all the bar objects in the plot
    yes_color = bars[0].get_facecolor()  # First group corresponds to "Allergy␣
↪Yes %"
    no_color = bars[len(percentage_data)].get_facecolor()  # Second group␣
↪corresponds to "Allergy No %"

    # Define the "Allergy Status" legend with dynamically matched colors
    allergy_handles = [
        mpatches.Patch(color=yes_color, label='Allergy Yes %'),
        mpatches.Patch(color=no_color, label='Allergy No %')
    ]
    allergy_legend = ax.legend(handles=allergy_handles, title='Allergy Status',
                               loc='upper left', bbox_to_anchor=(1.0, 1.0),␣
↪fontsize=10, title_fontsize=12)

    # Fetch the legend title from the dictionary for the primary legend
    primary_legend_title = legend_titles.get(feature, "Category Descriptions") ␣
↪# Use "Category Descriptions" as default

    # Define the primary legend for x-axis categories
    x_categories = value_mappings[feature]  # Get the dictionary for the␣
↪current feature
    x_handles = [mpatches.Patch(color='white', label=f"{k}: {v}") for k, v in␣
↪x_categories.items()]
    primary_legend = ax.legend(handles=x_handles, title=primary_legend_title,
                               loc='upper left', bbox_to_anchor=(1.0, 0.7),␣
↪fontsize=10, title_fontsize=12,
                               frameon=True, edgecolor='gray')

    # Manually add the "Allergy Status" legend back
    plt.gca().add_artist(allergy_legend)  # Re-add the allergy status legend␣
↪after adding primary legend

    # Adjust the layout and display the plot
    plt.tight_layout()
    plt.show()
```

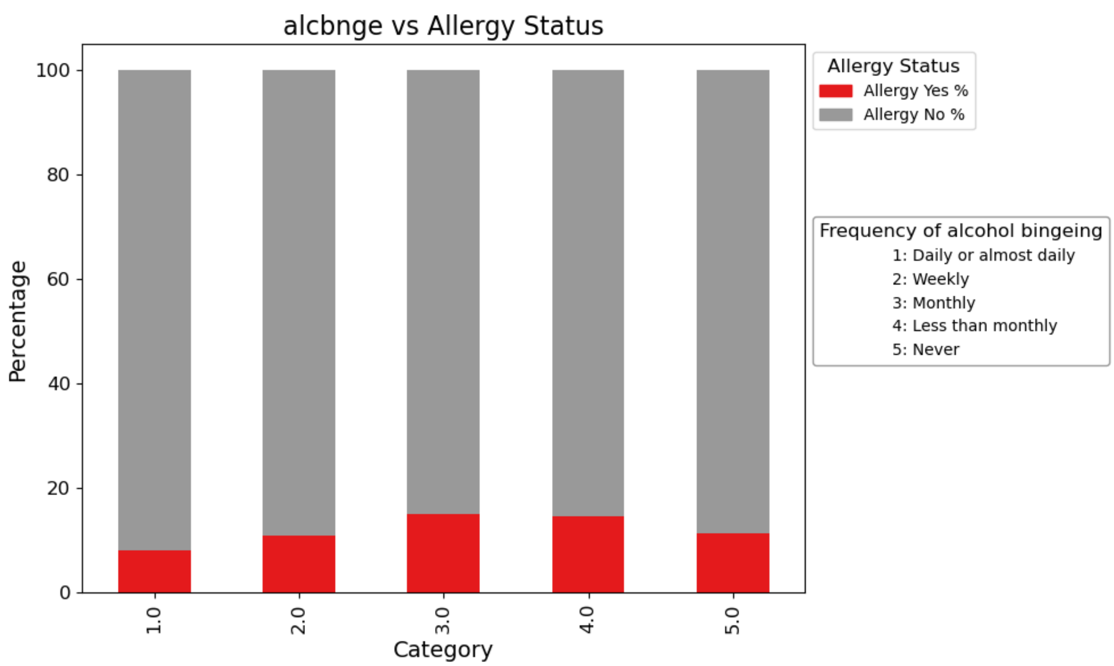

etfruit vs Allergy Status
Percentage
100
80
60
40
20
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
Category
Allergy Status
Allergy Yes %
Allergy No %
How often eat fruit
1: Three times or more a day
2: Twice a day
3: Once a day
4: Less than once a day but at least 4 times a week
5: Less than 4 times a week but at least once a week
6: Less than once a week
7: Never

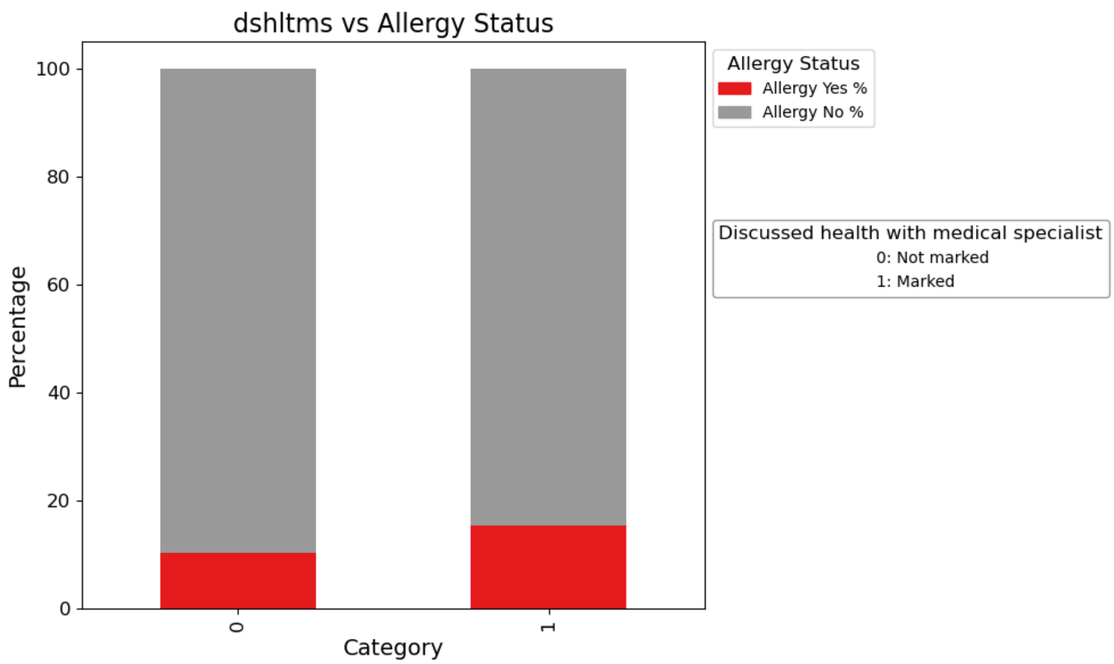

eatveg vs Allergy Status
Percentage
100
80
60
40
20
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
Category
Allergy Status
Allergy Yes %
Allergy No %
How often eat vegetables or salad, excluding potatoes
1: Three times or more a day
2: Twice a day
3: Once a day
4: Less than once a day but at least 4 times a week
5: Less than 4 times a week but at least once a week
6: Less than once a week
7: Never

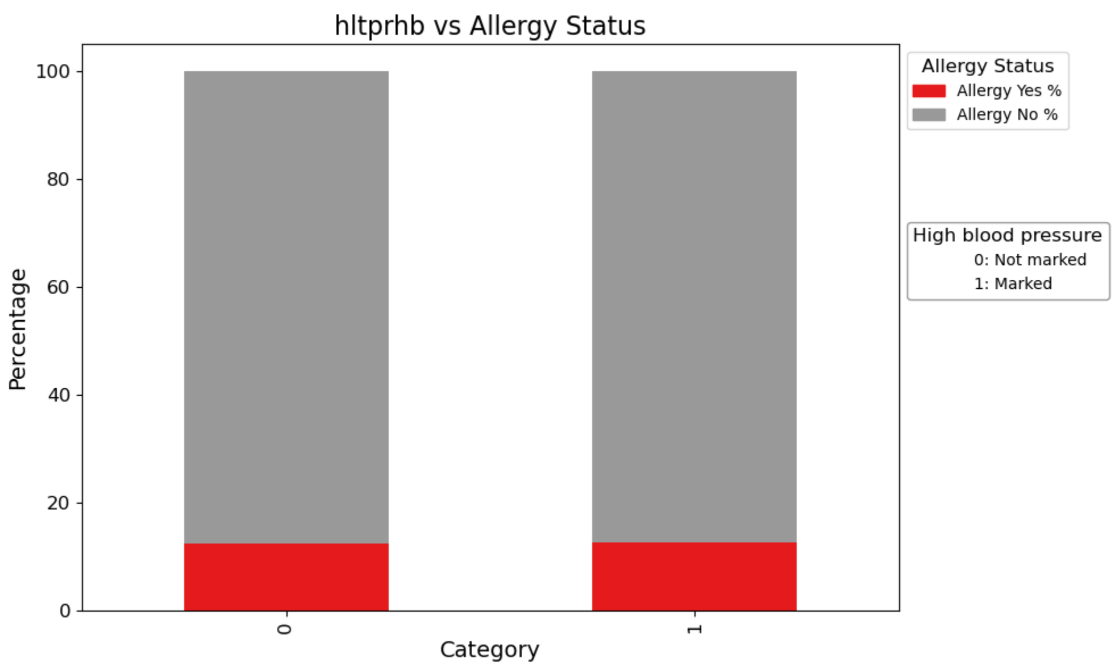

dosprt vs Allergy Status
Percentage
Category
Allergy Status
Allergy Yes %
Allergy No %
How often do sports
1: One day per week
2: Two days per week
3: Three days per week
4: Four days per week
5: Five days per week
6: Six days per week
7: Seven days per week

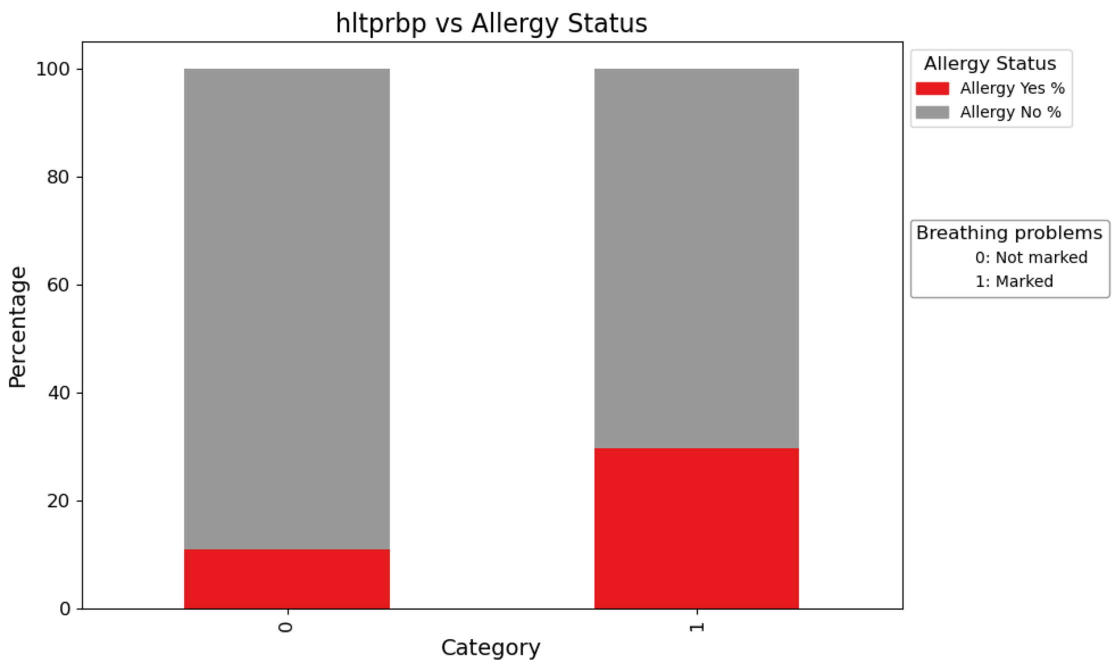

cgtsmok vs Allergy Status
Percentage
Category
Allergy Status
Allergy Yes %
Allergy No %
Cigarette smoking behavior
1: I smoke daily, 10 or more cigarettes
2: I smoke daily, 9 or fewer cigarettes
3: I smoke but not every day
4: I don't smoke now but I used to
5: I have only smoked a few times
6: I have never smoked

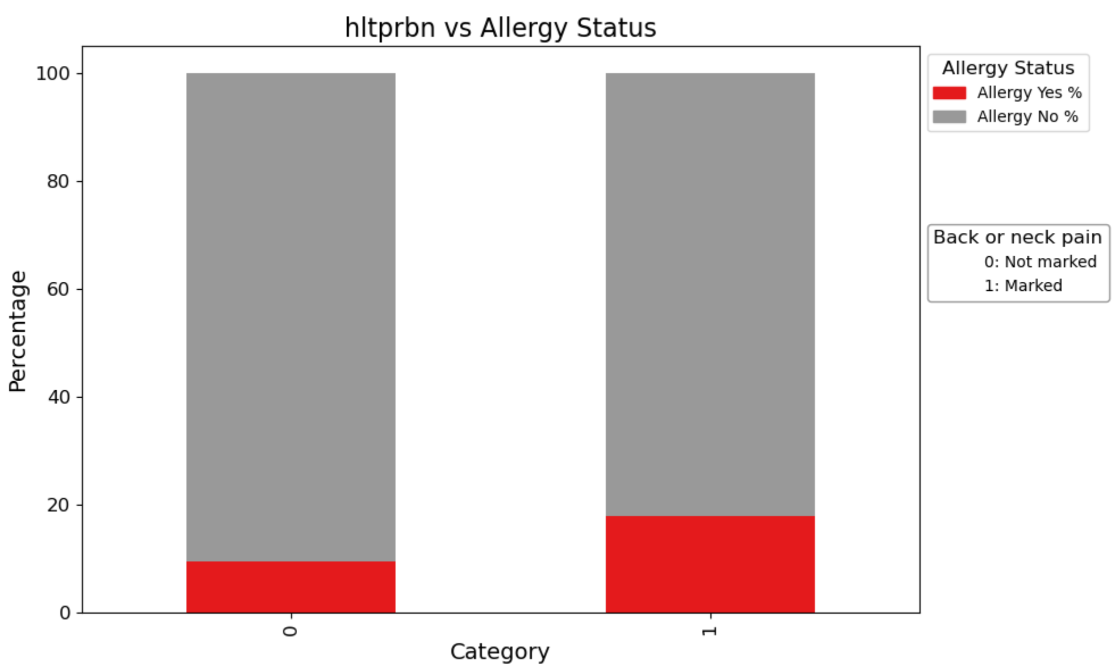

alcfreq vs Allergy Status
Allergy Status
Allergy Yes %
Allergy No %
Frequency of alcohol consumption
1: Every day
2: Several times a week
3: Once a week
4: 2-3 times a month
5: Once a month
6: Less than once a month
7: Never
Percentage
Category
0
20
40
60
80
100
1.0
2.0
3.0
4.0
5.0
6.0
7.0

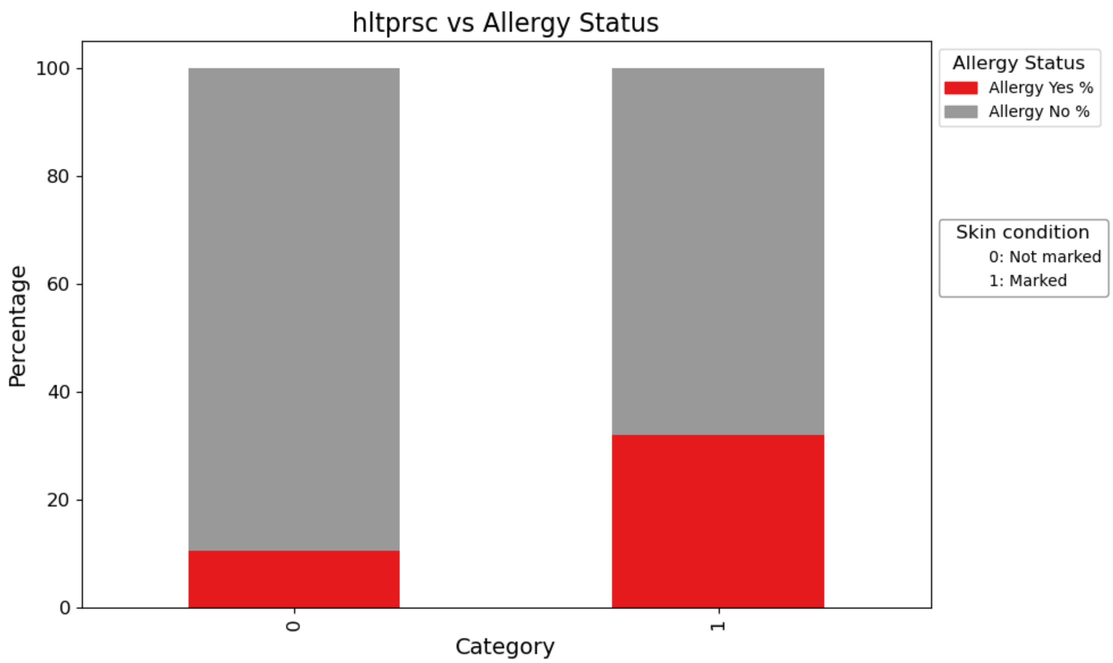

alcbnge vs Allergy Status
Allergy Status
Allergy Yes %
Allergy No %
Frequency of alcohol bingeing
1: Daily or almost daily
2: Weekly
3: Monthly
4: Less than monthly
5: Never
Percentage
Category
0
20
40
60
80
100
1.0
2.0
3.0
4.0
5.0

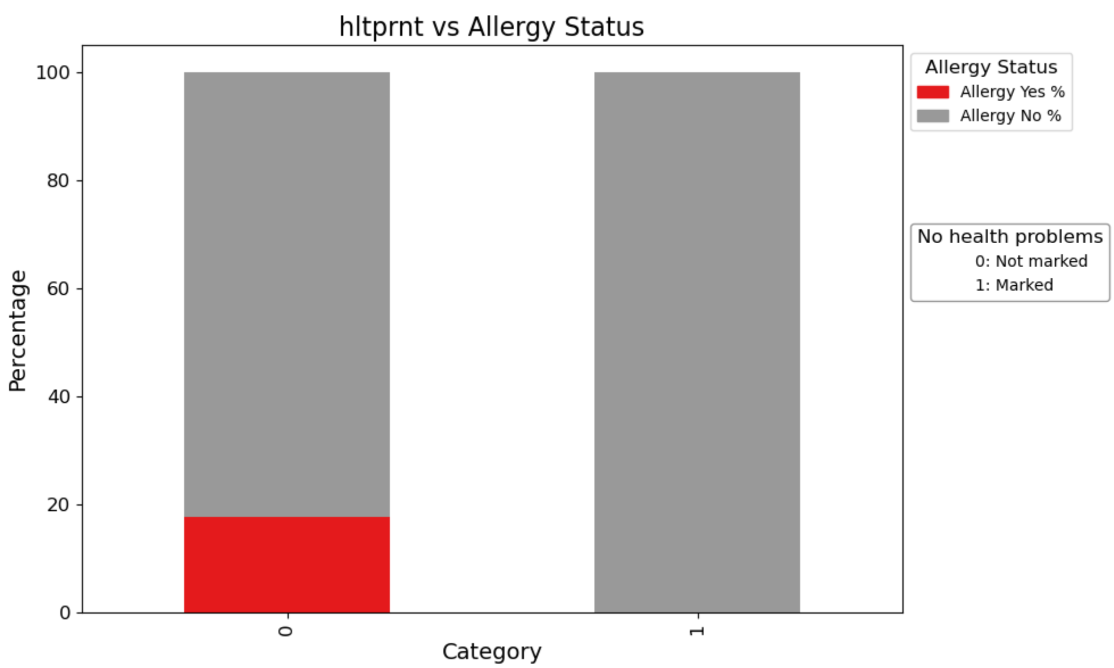

dshltms vs Allergy Status
Percentage
100
80
60
40
20
0
0
1
Category
Allergy Status
Allergy Yes %
Allergy No %
Discussed health with medical specialist
0: Not marked
1: Marked

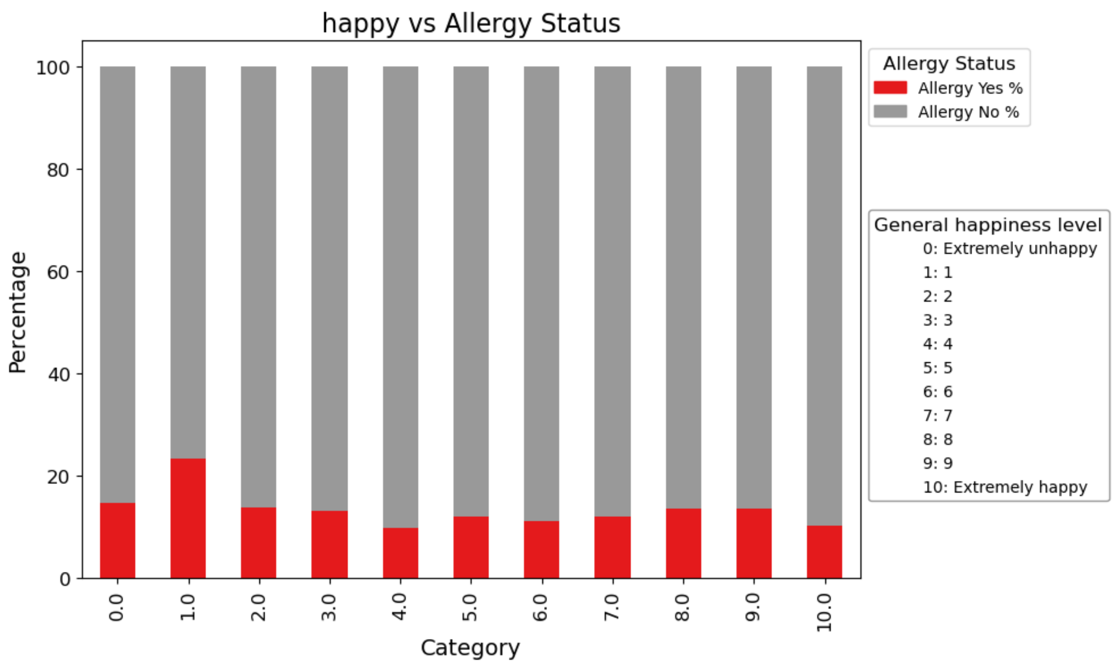

hltprhb vs Allergy Status
Percentage
100
80
60
40
20
0
0
1
Category
Allergy Status
Allergy Yes %
Allergy No %
High blood pressure
0: Not marked
1: Marked

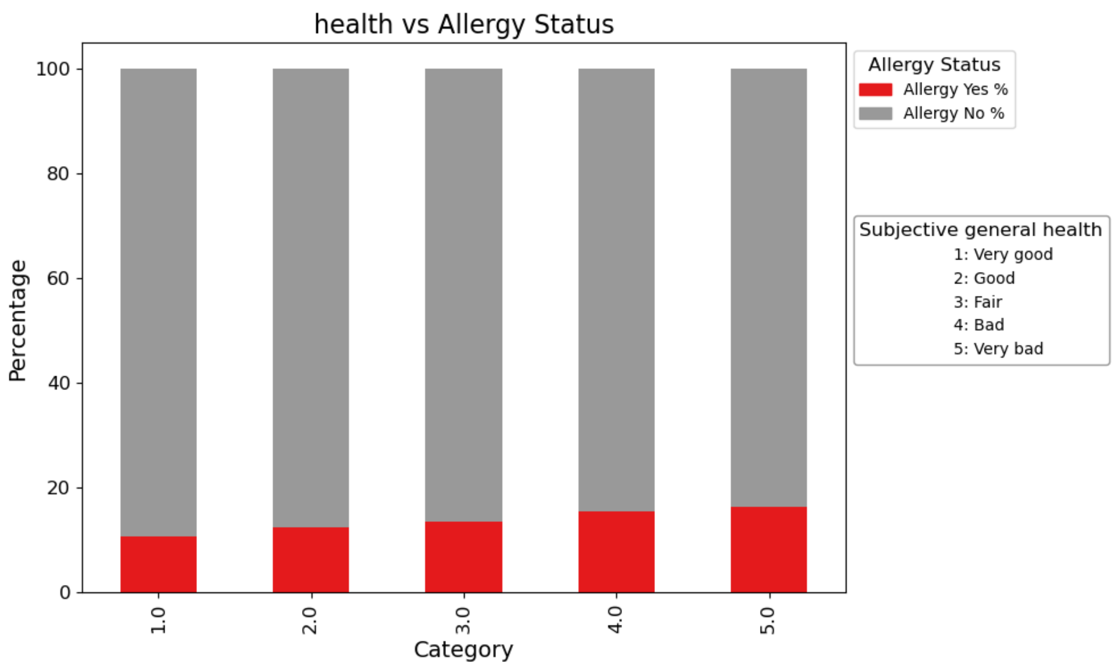

hltprbp vs Allergy Status
Percentage
Category
Allergy Status
Allergy Yes %
Allergy No %
Breathing problems
0: Not marked
1: Marked
0
20
40
60
80
100
0
1

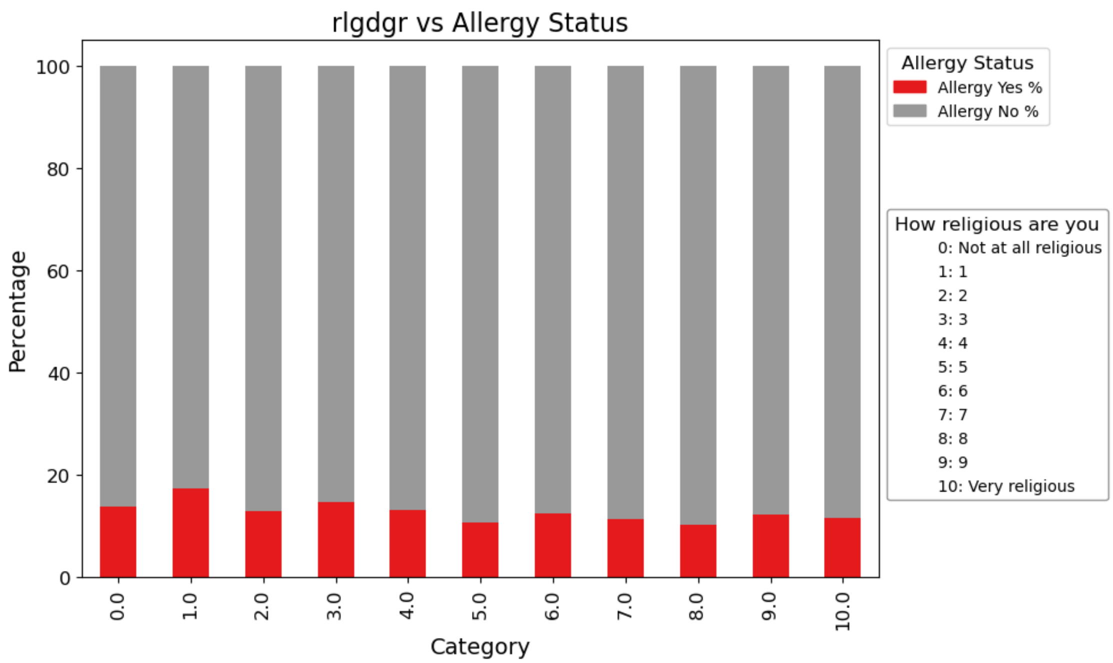

hltprbn vs Allergy Status
Percentage
Category
Allergy Status
Allergy Yes %
Allergy No %
Back or neck pain
0: Not marked
1: Marked
0
20
40
60
80
100
0
1

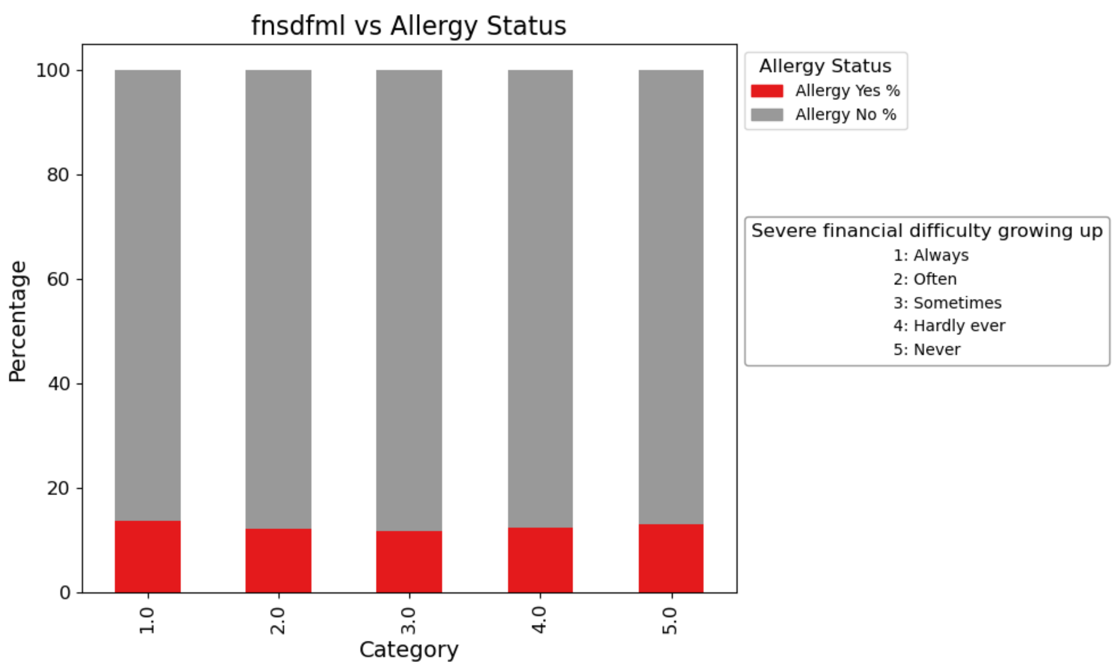

hltprsc vs Allergy Status
Percentage
Category
0
20
40
60
80
100
0
1
Allergy Status
Allergy Yes %
Allergy No %
Skin condition
0: Not marked
1: Marked

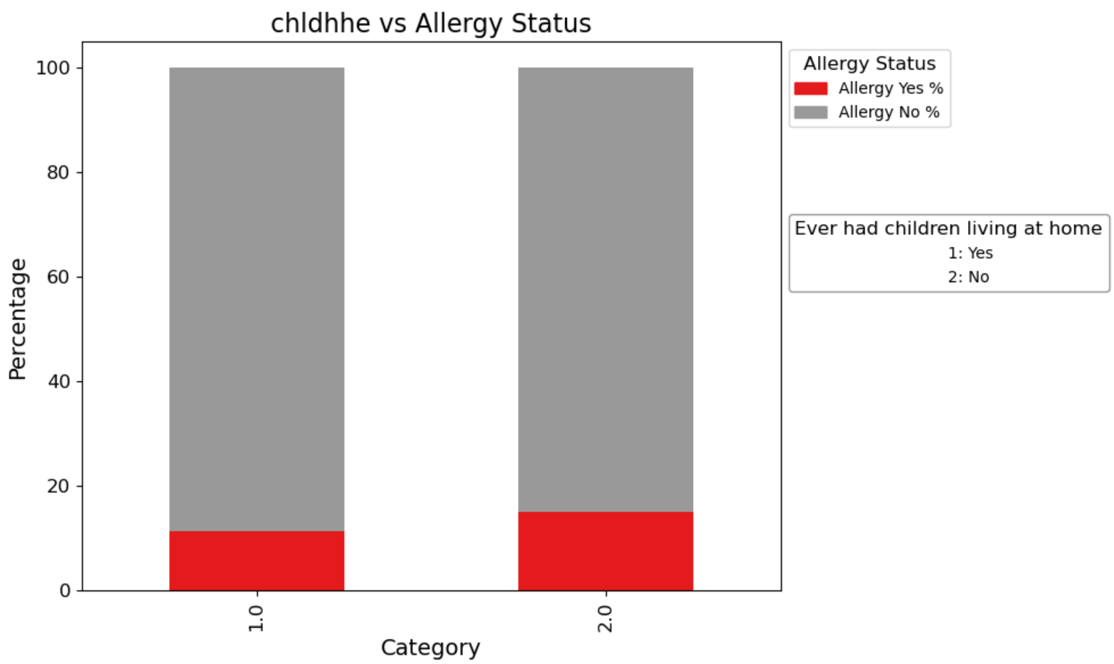

hltprnt vs Allergy Status
Percentage
Category
0
20
40
60
80
100
0
1
Allergy Status
Allergy Yes %
Allergy No %
No health problems
0: Not marked
1: Marked

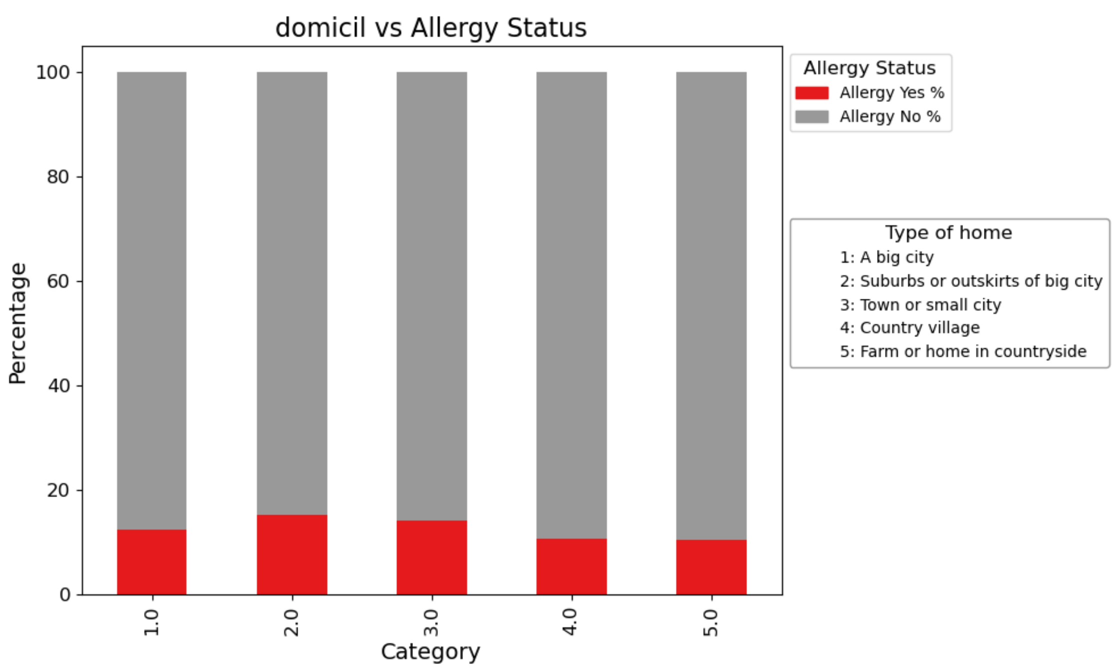

happy vs Allergy Status
Percentage
Category
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
0
20
40
60
80
100
Allergy Status
Allergy Yes %
Allergy No %
General happiness level
0: Extremely unhappy
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
10: Extremely happy


health vs Allergy Status
Percentage
Category
1.0
2.0
3.0
4.0
5.0
0
20
40
60
80
100
Allergy Status
Allergy Yes %
Allergy No %
Subjective general health
1: Very good
2: Good
3: Fair
4: Bad
5: Very bad


rlgdgr vs Allergy Status
Percentage
100
80
60
40
20
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
Category
Allergy Status
Allergy Yes %
Allergy No %
How religious are you
0: Not at all religious
1: 1
2: 2
3: 3
4: 4
5: 5
6: 6
7: 7
8: 8
9: 9
10: Very religious


fnsdfml vs Allergy Status
Percentage
100
80
60
40
20
0
1.0
2.0
3.0
4.0
5.0
Category
Allergy Status
Allergy Yes %
Allergy No %
Severe financial difficulty growing up
1: Always
2: Often
3: Sometimes
4: Hardly ever
5: Never





[32]:
```
df_cleaned.columns
```

[32]:
```
Index(['etfruit', 'eatveg', 'dosprt', 'cgtsmok', 'alcfreq', 'alcbnge',
       'dshltgp', 'dshltms', 'dshltnt', 'trhltacu', 'trhltacp', 'trhltcm',
       'trhltch', 'trhltos', 'trhltho', 'trhltht', 'trhlthy', 'trhltmt',
       'trhltpt', 'trhltre', 'trhltsh', 'trhltnt', 'hltprhc', 'hltpral',
       'hltprhb', 'hltprbp', 'hltprbn', 'hltprpa', 'hltprpf', 'hltprsd',
       'hltprsc', 'hltprsh', 'hltprdi', 'hltprnt', 'happy', 'health',
       'hlthhmp', 'rlgdgr', 'pray', 'height', 'weighta', 'fnsdfml', 'jbexevh',
       'jbexevc', 'jbexera', 'jbexecp', 'jbexebs', 'chldhhe', 'domicil',
       'paccmoro', 'paccocrw', 'paccxhoc', 'paccinro', 'isco08', 'nacer2',
       'hinctnta', 'hltpral_mapped'],
      dtype='object')
```

4 Model Training

For the model training portion of the project, I chose four models to train, and used a variety of oversampling techniques to account for the class imbalance between individuals with and without allergies. I split the dataset into 70% training data and 30% testing data, to ensure more accurate scores for analyzing each model. This split was chosen to maximize the reliability of the model's evaluation metrics, given the substantial size of the dataset.

[33]:
```python
from imblearn.over_sampling import RandomOverSampler, SMOTENC, SMOTE, ADASYN
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪roc_auc_score, roc_curve, make_scorer, recall_score, precision_recall_curve
from imblearn.combine import SMOTETomek
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from imblearn.pipeline import Pipeline
from xgboost import XGBClassifier

# Split the data
X = df_cleaned.drop(columns=['hltpral', 'hltpral_mapped'])  # Features
y = df_cleaned['hltpral']  # Target (0 = No Allergy, 1 = Allergy)

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

4.1 Logistic Regression - Random Oversampling

The first model I chose was a logistic regression model, which was first tested with random oversampling. Random oversampling selects samples from the minority class at random and duplicates them, then adds the duplicated samples back into the dataset.

[34]:
```python
# Apply Random Oversampling
ros = RandomOverSampler(random_state=42)
X_train_ros, y_train_ros = ros.fit_resample(X_train, y_train)

print("Before Oversampling:", y_train.value_counts())
print("After Oversampling:", y_train_ros.value_counts())
```

```
Before Oversampling: hltpral
0    13590
1     1943
Name: count, dtype: int64
After Oversampling: hltpral
0    13590
1    13590
Name: count, dtype: int64
```

[35]:
```python
## Model training using Random Sampler

X_train_ros_encoded = pd.get_dummies(X_train_ros, columns=['domicil', 'isco08',␣
 ↪'nacer2'])

# Also One-Hot Encode X_test for consistency
X_test_encoded = pd.get_dummies(X_test, columns=['domicil', 'isco08', 'nacer2'])

# Ensure both train and test dataframes have the same columns
X_train_ros_encoded, X_test_encoded = X_train_ros_encoded.align(X_test_encoded,␣
 ↪join='left', axis=1, fill_value=0)

# Checking the new shape
print(X_train_ros_encoded.shape)
print(X_test_encoded.shape)
```

```
(27180, 650)
(6657, 650)
```

[36]:
```python
# Scale dataset
scaler = StandardScaler()
X_train_ros_scaled = scaler.fit_transform(X_train_ros_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

# Initialize the model
model = LogisticRegression(max_iter=5000, random_state=42)

# Fit the model to the resampled training data
model.fit(X_train_ros_scaled, y_train_ros)

# Make predictions on the test data
y_pred = model.predict(X_test_scaled)
```

```
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
roc_auc = roc_auc_score(y_test, y_pred_proba)
print("\nROC-AUC Score:", roc_auc)
```

```
Accuracy: 0.665765359771669
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.66      0.78      5825
           1       0.22      0.67      0.34       832

    accuracy                           0.67      6657
   macro avg       0.58      0.67      0.56      6657
weighted avg       0.85      0.67      0.72      6657


ROC-AUC Score: 0.7437561901617695
```

As shown, this model's performance was suboptimal, with a relatively low accuracy score and notably low precision and F1-score for the minority class. While the recall score was relatively high overall, the ROC-AUC score of 0.7437 indicates room for improvement in balancing sensitivity and specificity.

4.2 Logistic Regression - SMOTE-Tomek

The next oversampling techinque I tried was SMOTE-Tomek. SMOTE stands for synthetic minority oversampling technique, and it works by generating synthetic samples for the minority class and adding them to the dataset. The new data points are created by interpolating between existing minority class examples and their nearest neighbors. Tomek links are also used in this technique, which removes pairs of neighboring samples that are in opposite classes, to remove noisy data and borderline samples.

[37]:
```
# Split the data
X = df_cleaned.drop(columns=['hltpral', 'hltpral_mapped'])  # Features
y = df_cleaned['hltpral']  # Target (0 = No Allergy, 1 = Allergy)

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

[38]:
```
# Apply SMOTE-Tomek (combines oversampling and undersampling)
smt = SMOTETomek(random_state=42)
X_train_smt, y_train_smt = smt.fit_resample(X_train, y_train)
```

```
print("Before SMOTE-Tomek:", y_train.value_counts())
print("After SMOTE-Tomek:", y_train_smt.value_counts())
```

```
Before SMOTE-Tomek: hltpral
0    13590
1     1943
Name: count, dtype: int64
After SMOTE-Tomek: hltpral
0    13523
1    13523
Name: count, dtype: int64
```

[39]:
```
## Model training using SMOTE-tomek

X_train_smt_encoded = pd.get_dummies(X_train_smt, columns=['domicil', 'isco08',
 ↪'nacer2'])

# Also One-Hot Encode X_test for consistency
X_test_encoded = pd.get_dummies(X_test, columns=['domicil', 'isco08', 'nacer2'])

# Ensure both train and test dataframes have the same columns
X_train_smt_encoded, X_test_encoded = X_train_smt_encoded.align(X_test_encoded,
 ↪join='left', axis=1, fill_value=0)

# Checking the new shape
print(X_train_smt_encoded.shape)
print(X_test_encoded.shape)
```

```
(27046, 22370)
(6657, 22370)
```

[40]:
```
scaler = StandardScaler()
X_train_smt_scaled = scaler.fit_transform(X_train_smt_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

# Initialize the model
model = LogisticRegression(max_iter=5000, random_state=42)

# Fit the model to the resampled training data
model.fit(X_train_smt_scaled, y_train_smt)

# Make predictions on the test data
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
```

```
print(classification_report(y_test, y_pred))
roc_auc = roc_auc_score(y_test, y_pred_proba)
print("\nROC-AUC Score:", roc_auc)
```

```
Accuracy: 0.854138500826198
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      5825
           1       0.30      0.12      0.18       832

    accuracy                           0.85      6657
   macro avg       0.59      0.54      0.55      6657
weighted avg       0.81      0.85      0.83      6657


ROC-AUC Score: 0.7271776989105315
```

The results for this model show an overall accuracy of 85.4%, which is better than the last model. However, the performance for the minority class (those with allergies) is poor, with a precision of 30% and a recall of only 12%, resulting in a low F1-score of 0.18. The ROC-AUC score of 0.727 is also lower. There is room for improvement, particularly in predicting the minority class more effectively.

4.3 Logistic Regression - SMOTE-NC (best parameters)

The third oversampling technique used was SMOTE-NC, which is synthetic minority oversampling for nominal and continuous data. This is similar to regular SMOTE, where the algorithm interpolates between randomly chosen minority instances and k-nearest neighbors for continuous data, but for categorical data, it chooses the most frequent category among the nearest neighbors to assign to synthetic data points. It then combines the synthetic samples for both continuous and categorical features to generate a balanced dataset.

For this model, I also performed a grid search (shown below), for which I applied the best parameters to see if the model's performance would be any higher.

[41]:
```
# Split the data
X = df_cleaned.drop(columns=['hltpral', 'hltpral_mapped'])  # Features
y = df_cleaned['hltpral']  # Target (0 = No Allergy, 1 = Allergy)

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

[42]:
```
## SmoteNC

# Encode categorical features
encoder = OrdinalEncoder()
X_encoded = encoder.fit_transform(X)
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.
↪3, random_state=42)

# All columns are categorical, but SMOTENC needs some numerical features
# Choose one or more ordinal features to keep as "numerical"
categorical_indices = list(range(X_train.shape[1]))  # Assume all are␣
↪categorical

# Apply SMOTENC
smote_nc = SMOTENC(categorical_features=categorical_indices[:-1],␣
↪random_state=42)
X_train_smote, y_train_smote = smote_nc.fit_resample(X_train, y_train)

print("Before SMOTENC:", y_train.value_counts())
print("After SMOTENC:", y_train_smote.value_counts())
```

```
Before SMOTENC: hltpral
0    13619
1     1914
Name: count, dtype: int64
After SMOTENC: hltpral
0    13619
1    13619
Name: count, dtype: int64
```

[43]:
```
X_train_smote_encoded = pd.DataFrame(X_train_smote, columns=X.columns)

# Apply pd.get_dummies on categorical columns
X_train_smote_encoded = pd.get_dummies(X_train_smote_encoded,␣
↪columns=['domicil', 'isco08', 'nacer2'])

# Also One-Hot Encode X_test for consistency
X_test_encoded = pd.DataFrame(X_test, columns=X.columns)

# Apply pd.get_dummies on test set
X_test_encoded = pd.get_dummies(X_test_encoded, columns=['domicil', 'isco08',␣
↪'nacer2'])

# Ensure both train and test dataframes have the same columns
X_train_smote_encoded, X_test_encoded = X_train_smote_encoded.
↪align(X_test_encoded, join='left', axis=1, fill_value=0)

# Checking the new shape
print(X_train_smote_encoded.shape)
print(X_test_encoded.shape)
```

```
(27238, 649)
```

```
(6657, 649)
```

[44]:
```
scaler = StandardScaler()
X_train_smote_scaled = scaler.fit_transform(X_train_smote_encoded)
X_test_scaled = scaler.transform(X_test_encoded)

# Initialize the model
best_model = LogisticRegression(
    C=10,
    class_weight='balanced',
    solver='newton-cg',
    max_iter=10000
)

# Fit the model to the resampled training data
best_model.fit(X_train_smote_scaled, y_train_smote)

# Make predictions on the test data
y_pred = best_model.predict(X_test_scaled)
y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
roc_auc = roc_auc_score(y_test, y_pred_proba)
print("\nROC-AUC Score:", roc_auc)
```

```
Accuracy: 0.7278053177106805
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.76      0.83      5796
           1       0.24      0.49      0.32       861

    accuracy                           0.73      6657
   macro avg       0.57      0.63      0.57      6657
weighted avg       0.82      0.73      0.76      6657


ROC-AUC Score: 0.7237740153207506
```

Compared to previous models, this one achieves higher recall for the minority class (49%), but at the cost of lower precision (24%) and a modest F1-score (0.32). The accuracy score of 72.8% and the ROC-AUC score are similar to the previous model.

[45]:
```
# Grid Search
param_grid = {
```

```
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'saga', 'newton-cg'],
    'class_weight': ['balanced']
}

grid_search = GridSearchCV(
    LogisticRegression(max_iter=10000),
    param_grid,
    scoring='roc_auc',
    cv=5,
    verbose=1
)
grid_search.fit(X_train_smote_scaled, y_train_smote)

best_model = grid_search.best_estimator_
print("Best parameters:", grid_search.best_params_)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best parameters: {'C': 10, 'class_weight': 'balanced', 'solver': 'newton-cg'}
```

4.4 Random Forest Classifier - SMOTE-NC

Next, I trained a random forest classifier using SMOTE-NC, with the class weights balanced.

[46]:
```
# Create the Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42, class_weight='balanced',
 ↪n_jobs=-1)

# Train the Model
rf_clf.fit(X_train_smote_scaled, y_train_smote)

# Make Predictions
y_pred_rf = rf_clf.predict(X_test_scaled)
y_pred_prob_rf = rf_clf.predict_proba(X_test_scaled)[:, 1]  # Probabilities for
 ↪AUC-ROC

# Evaluate the Model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)
auc_roc_rf = roc_auc_score(y_test, y_pred_prob_rf)

print(f"Accuracy: {accuracy_rf}")
print("Classification Report:")
print(classification_rep_rf)
print(f"AUC-ROC: {auc_roc_rf}")
```

```
Accuracy: 0.8355114916629113
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      5796
           1       0.30      0.20      0.24       861

    accuracy                           0.84      6657
   macro avg       0.59      0.57      0.57      6657
weighted avg       0.81      0.84      0.82      6657

AUC-ROC: 0.7504232162995987
```

The first random forest classifier achieved an accuracy of 83.6% and an AUC-ROC of 0.750, marking an improvement over previous models in overall performance. While its recall for the minority class is lower at 20%, precision remains comparable. This model has a higher overall accuracy for the majority class with a slightly better AUC-ROC score, indicating improved discrimination between classes compared to earlier models, though minority class performance still needs improvement.

4.5 Random Forest Classifier - ADASYN

For this next Random Forest classifier, I applied an oversampling technique called ADASYN (Adaptive Synthetic Sampling). Similar to SMOTE, ADASYN generates synthetic data points for the minority class. However, it goes a step further by identifying minority class samples that are harder to classify. These samples are given higher weights, leading to the creation of more synthetic data points in areas where the model struggles most to differentiate between classes.

[47]:
```
# Scale before sampling for ADAYSN
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[48]:
```
# Apply ADASYN
adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train_scaled, y_train)

# Train Random Forest Classifier
rf_clf_adasyn = RandomForestClassifier(random_state=42,␣
 ↪class_weight='balanced', n_jobs=-1)
rf_clf_adasyn.fit(X_train_adasyn, y_train_adasyn)

# Make Predictions
y_pred_rf_adasyn = rf_clf_adasyn.predict(X_test_scaled)
y_pred_prob_rf_adasyn = rf_clf_adasyn.predict_proba(X_test_scaled)[:, 1]

# Evaluate the Model
accuracy_rf_adasyn = accuracy_score(y_test, y_pred_rf_adasyn)
classification_rep_rf_adasyn = classification_report(y_test, y_pred_rf_adasyn)
auc_roc_rf_adasyn = roc_auc_score(y_test, y_pred_prob_rf_adasyn)
```

```
print(f"Accuracy: {accuracy_rf_adasyn}")
print("Classification Report:")
print(classification_rep_rf_adasyn)
print(f"AUC-ROC: {auc_roc_rf_adasyn}")
```

```
Accuracy: 0.8735165990686495
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      5796
           1       0.59      0.07      0.13       861

    accuracy                           0.87      6657
   macro avg       0.73      0.53      0.53      6657
weighted avg       0.84      0.87      0.83      6657

AUC-ROC: 0.7842403427731408
```

This model achieved a higher accuracy score than the previous models, with a higher AUC-ROC score as well. However, the recall and f1-score for this model are very low, meaning it will perform poorly for identifying positive allergy cases. However, this can be adjusted by changing the model's threshold.

4.6 Random Forest Classifier - ADASYN, best parameters

The next Random Forest classifier also utilized ADASYN for oversampling but was optimized using the best parameters obtained through a Randomized Search CV.

[49]:
```
# Define the Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Define the Parameter Grid
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
    'class_weight': [None, 'balanced', 'balanced_subsample']
}

# Define Scorer to Focus on Recall
recall_scorer = make_scorer(recall_score)

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=rf_clf,
    param_distributions=param_dist,
```

```
    n_iter=50,  # Number of parameter settings sampled
    scoring=recall_scorer,  # Optimizing for recall
    cv=3,  # 3-fold cross-validation
    verbose=2,
    random_state=42
)

# Fit RandomizedSearchCV
random_search.fit(X_train_adasyn, y_train_adasyn)

# Display Best Parameters and Refit Model
print("Best Parameters:", random_search.best_params_)
rf_best = random_search.best_estimator_

# Evaluate the Optimized Model
y_pred_rf_best = rf_best.predict(X_test_scaled)
y_pred_prob_rf_best = rf_best.predict_proba(X_test_scaled)[:, 1]
accuracy_rf_best = accuracy_score(y_test, y_pred_rf_best)
classification_rep_rf_best = classification_report(y_test, y_pred_rf_best)
auc_roc_rf_best = roc_auc_score(y_test, y_pred_prob_rf_best)

print(f"Accuracy: {accuracy_rf_best}")
print("Classification Report:")
print(classification_rep_rf_best)
print(f"AUC-ROC: {auc_roc_rf_best}")
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   3.6s
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.9s
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=200; total time=   0.9s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=200; total time=   1.6s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=200; total time=   1.5s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=10, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   4.0s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   3.5s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   3.3s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.6s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
```

```
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.5s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.6s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   8.8s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   7.3s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   7.1s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.4s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.3s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.3s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   1.3s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   1.3s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   1.2s
[CV] END class_weight=balanced, max_depth=10, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   1.8s
[CV] END class_weight=balanced, max_depth=10, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   1.5s
[CV] END class_weight=balanced, max_depth=10, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   1.5s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.7s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=200; total time=   1.5s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.2s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.6s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.3s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   2.7s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   2.6s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=50; total time=   2.2s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=log2,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.1s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=log2,
```

```
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.3s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=log2,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.3s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.6s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.2s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   2.0s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=   1.6s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=50; total time=   2.2s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=50; total time=   2.4s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=50; total time=   1.9s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   0.4s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   0.4s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   0.3s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   0.7s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   0.8s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   0.6s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  13.0s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  10.3s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  11.2s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.9s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.7s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=300; total time=   2.0s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=1,
```

```
min_samples_split=2, n_estimators=300; total time=   2.0s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=300; total time=   2.2s
[CV] END class_weight=None, max_depth=30, max_features=None, min_samples_leaf=4,
min_samples_split=5, n_estimators=100; total time=   4.2s
[CV] END class_weight=None, max_depth=30, max_features=None, min_samples_leaf=4,
min_samples_split=5, n_estimators=100; total time=   3.6s
[CV] END class_weight=None, max_depth=30, max_features=None, min_samples_leaf=4,
min_samples_split=5, n_estimators=100; total time=   3.4s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=   0.7s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=   0.8s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=sqrt,
min_samples_leaf=1, min_samples_split=5, n_estimators=100; total time=   0.7s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   6.8s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   5.3s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   6.5s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   4.4s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   3.7s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=4, min_samples_split=5, n_estimators=100; total time=   3.8s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=   2.4s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=   2.0s
[CV] END class_weight=balanced, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=   2.1s
[CV] END class_weight=None, max_depth=20, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100; total time=   0.7s
[CV] END class_weight=None, max_depth=20, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100; total time=   0.7s
[CV] END class_weight=None, max_depth=20, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100; total time=   0.8s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=sqrt,
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   1.1s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=sqrt,
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.9s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=sqrt,
min_samples_leaf=2, min_samples_split=10, n_estimators=100; total time=   0.8s
[CV] END class_weight=None, max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=200; total time=   2.0s
[CV] END class_weight=None, max_depth=20, max_features=sqrt, min_samples_leaf=1,
```

```
min_samples_split=2, n_estimators=200; total time=   1.9s
[CV] END class_weight=None, max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=200; total time=   1.9s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   5.4s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   3.6s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=5, n_estimators=100; total time=   3.7s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   1.6s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   1.6s
[CV] END class_weight=None, max_depth=30, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.3s
[CV] END class_weight=balanced_subsample, max_depth=30, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.4s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   0.6s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   0.6s
[CV] END class_weight=None, max_depth=30, max_features=log2, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   0.7s
[CV] END class_weight=balanced, max_depth=30, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced, max_depth=30, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced, max_depth=30, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=  13.2s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=  11.6s
[CV] END class_weight=balanced, max_depth=30, max_features=None,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=  10.4s
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time=   0.7s
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time=   0.5s
[CV] END class_weight=balanced, max_depth=10, max_features=log2,
min_samples_leaf=4, min_samples_split=10, n_estimators=100; total time=   0.5s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=   1.9s
[CV] END class_weight=None, max_depth=None, max_features=log2,
```

```
min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=   1.8s
[CV] END class_weight=None, max_depth=None, max_features=log2,
min_samples_leaf=4, min_samples_split=2, n_estimators=300; total time=   1.7s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   2.1s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   2.0s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   1.9s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time=  13.1s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time=  10.2s
[CV] END class_weight=balanced, max_depth=20, max_features=None,
min_samples_leaf=4, min_samples_split=10, n_estimators=300; total time=   9.5s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.3s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.3s
[CV] END class_weight=balanced, max_depth=None, max_features=sqrt,
min_samples_leaf=2, min_samples_split=2, n_estimators=300; total time=   2.2s
[CV] END class_weight=balanced, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   9.1s
[CV] END class_weight=balanced, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   8.5s
[CV] END class_weight=balanced, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=200; total time=   6.9s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   8.7s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   7.2s
[CV] END class_weight=None, max_depth=None, max_features=None,
min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   7.7s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=1, min_samples_split=5, n_estimators=50; total time=   0.4s
[CV] END class_weight=None, max_depth=20, max_features=None, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   3.9s
[CV] END class_weight=None, max_depth=20, max_features=None, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   3.4s
[CV] END class_weight=None, max_depth=20, max_features=None, min_samples_leaf=2,
min_samples_split=10, n_estimators=100; total time=   3.6s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
```

```
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=2, n_estimators=50; total time=   0.4s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  12.0s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  11.3s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=300; total time=  10.5s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.5s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.5s
[CV] END class_weight=balanced_subsample, max_depth=20, max_features=sqrt,
min_samples_leaf=4, min_samples_split=10, n_estimators=50; total time=   0.5s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   1.8s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   1.4s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=None,
min_samples_leaf=1, min_samples_split=10, n_estimators=50; total time=   1.4s
[CV] END class_weight=None, max_depth=10, max_features=None, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   6.5s
[CV] END class_weight=None, max_depth=10, max_features=None, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   5.0s
[CV] END class_weight=None, max_depth=10, max_features=None, min_samples_leaf=2,
min_samples_split=5, n_estimators=200; total time=   5.1s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time=   1.6s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time=   1.6s
[CV] END class_weight=balanced_subsample, max_depth=10, max_features=log2,
min_samples_leaf=2, min_samples_split=10, n_estimators=300; total time=   1.6s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.5s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.4s
[CV] END class_weight=balanced_subsample, max_depth=None, max_features=log2,
min_samples_leaf=2, min_samples_split=2, n_estimators=200; total time=   1.4s
Best Parameters: {'n_estimators': 200, 'min_samples_split': 2,
'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_depth': 10, 'class_weight':
'balanced_subsample'}
Accuracy: 0.8358119272945771
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      5796
           1       0.34      0.29      0.31       861
```

```
    accuracy                           0.84      6657
   macro avg       0.62      0.60      0.61      6657
weighted avg       0.82      0.84      0.83      6657

AUC-ROC: 0.7761123655306356
```

After applying the best parameters the Randomized Search CV chose for this model, it had a balanced improvement in the recall and f1-score, while the accuracy and AUC-ROC score were slightly lower. However, the model is a bit more balanced overall and would likely perform better than the previous one.

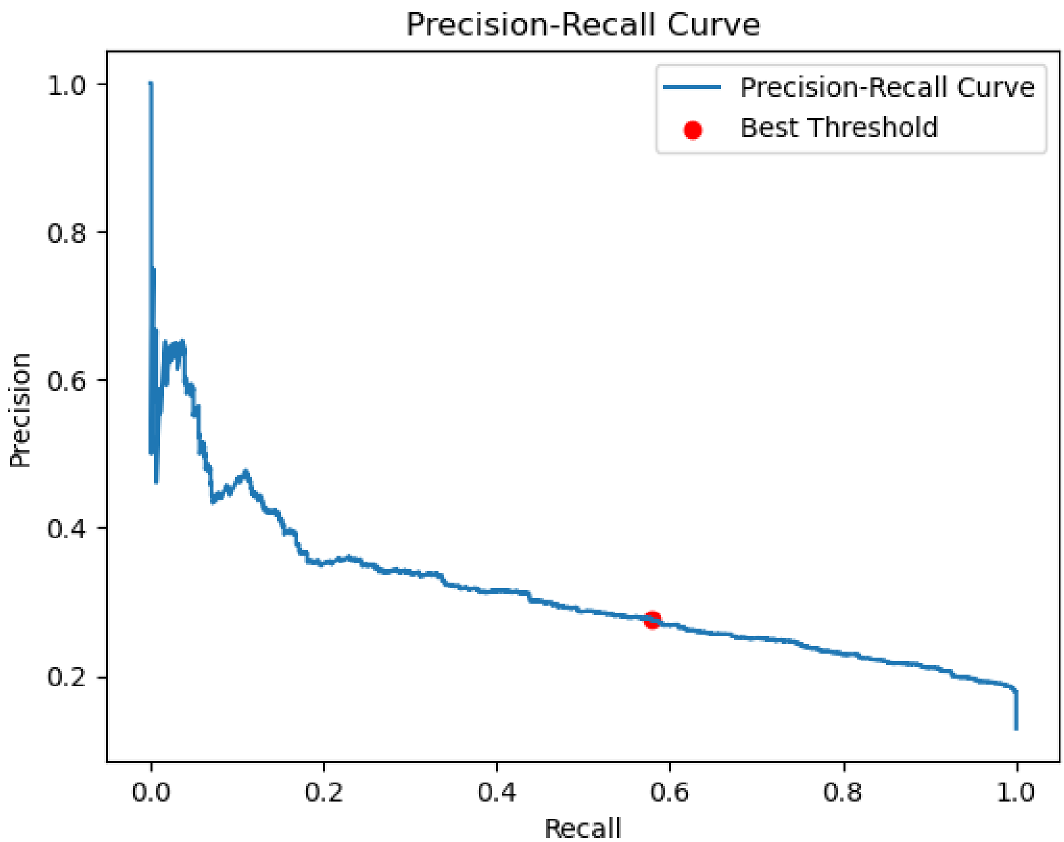
4.7 Random Forest Classifier - ADASYN, best parameters, adjusted threshold

For this next model, I combined the ADASYN oversampling technique with the best parameters, and also decided to adjust the threshold for the model, to see if a more balanced precision-recall relationship could be achieved, aiming to improve the model's performance on the minority class.

[50]:
```python
# Get prediction probabilities
y_pred_prob = rf_best.predict_proba(X_test_scaled)[:, 1]

# Compute precision-recall values
precisions, recalls, thresholds = precision_recall_curve(y_test, y_pred_prob)

# Compute F1-scores for each threshold
f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-10)  #␣
 ↪Avoid division by zero
best_threshold_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_threshold_idx]

print(f"Optimal Threshold: {best_threshold}")

# Plot Precision-Recall Curve
plt.plot(recalls, precisions, label='Precision-Recall Curve')
plt.scatter(recalls[best_threshold_idx], precisions[best_threshold_idx],␣
 ↪color='red', label='Best Threshold')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.show()

# Make predictions using the new threshold
y_pred_adjusted = (y_pred_prob >= best_threshold).astype(int)

# Evaluate the new predictions
print("Classification Report at Adjusted Threshold:")
print(classification_report(y_test, y_pred_adjusted))
```

```python
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
Optimal Threshold: 0.405737058909893
```




```
Classification Report at Adjusted Threshold:
              precision    recall  f1-score   support

           0       0.93      0.78      0.84      5796
           1       0.28      0.58      0.38       861

    accuracy                           0.75      6657
   macro avg       0.60      0.68      0.61      6657
weighted avg       0.84      0.75      0.78      6657

ROC-AUC: 0.7761123655306356
```

After adjusting the threshold for this model, the ROC-AUC score didn't change, but the recall and f1-score were a bit higher for the minority class, which means this model would perform better than the previous ones for identifying allergy cases. The accuracy score is lower than for the previous model, but that is expected as part of the trade-off. Overall, there's still room for improvement.

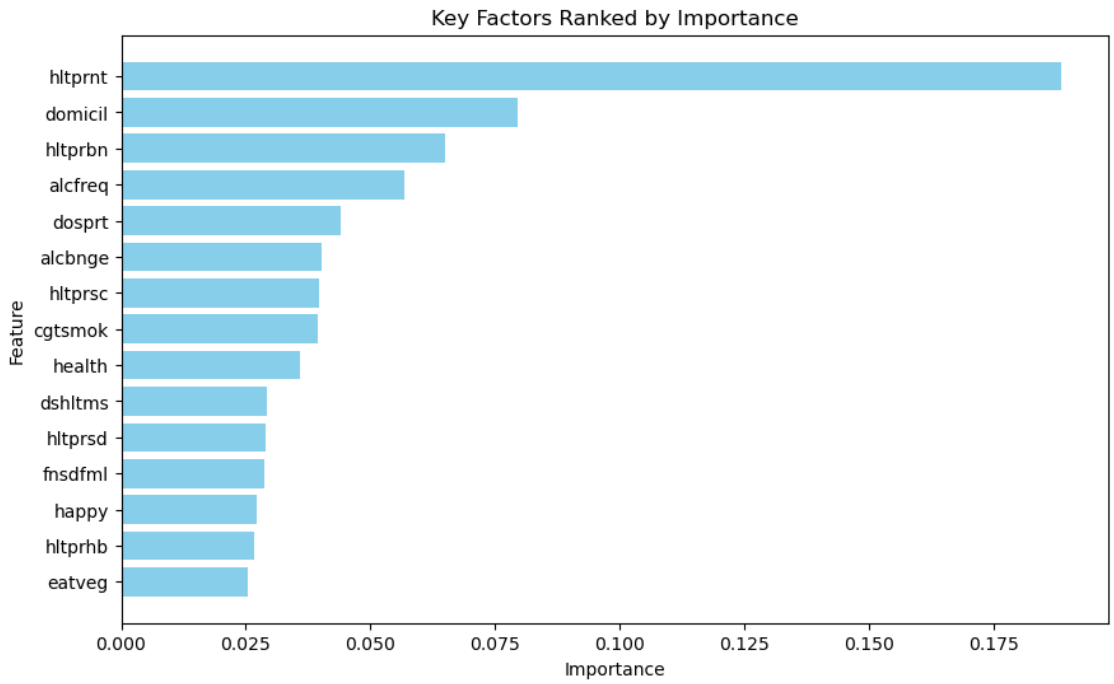
4.8 Feature Engineering

After creating the Random Forest model that performed the best so far, I generated the list of features that were considered most important by the Random Forest. I then took those first 15 as well as some of the next few listed, and used those as my features for my final model and in my web app.

[69]:
```
# Drop 'hltpral' from the original dataframe
df_cleaned_without_label = df_cleaned.drop(columns=['hltpral',␣
↪'hltpral_mapped'])

# Get the column names (features) after dropping 'hltpral'
columns_to_keep_without_label = df_cleaned_without_label.columns

# Calculate feature importances
importances = rf_best.feature_importances_

# Create a DataFrame for feature importance and sort it
feature_importance_df = pd.DataFrame({
    'Feature': columns_to_keep_without_label,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Display the top 15 most important features
top_features = feature_importance_df.head(15)
print(top_features)

# Plot the top 15 feature importances
plt.figure(figsize=(10, 6))
plt.barh(top_features['Feature'], top_features['Importance'], color='skyblue')
plt.gca().invert_yaxis()  # Highest importance at the top
plt.title("Key Factors Ranked by Importance")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

```
    Feature  Importance
32  hltprnt    0.188452
47  domicil    0.079502
25  hltprbn    0.064863
4   alcfreq    0.056703
2    dosprt    0.044146
5   alcbnge    0.040126
29  hltprsc    0.039590
3   cgtsmok    0.039548
34   health    0.035814
7   dshltms    0.029138
28  hltprsd    0.029029
```

```
40  fnsdfml    0.028680
33    happy    0.027066
23  hltprhb    0.026792
1    eatveg    0.025307
```



I also created this bar chart to better visualize the level of importance for each feature, and included this in my final web app.

Feature Descriptions

hltprnt: Indicates whether the respondent has had specific health problems in the last 12 months.

domicil: The type of area where the respondent lives (e.g., big city, countryside).

hltprbn: Back or neck pain reported by the respondent.

alcfreq: Frequency of alcohol consumption in the last 12 months.

dosprt: Days per week the respondent engages in sports activities.

alcbnge: Frequency of binge drinking in the last 12 months.

hltprsc: Presence of skin conditions reported by the respondent.

cgtsmok: Smoking habits of the respondent.

health: General health status of the respondent.

dshltms: Whether health was discussed with a medical professional in the last 12 months.

hltprsd: Stomach or digestion-related issues reported by the respondent.

fnsdfml: Frequency of financial difficulties during childhood.

happy: General happiness level of the respondent (scale of 1 to 10).

hltprhb: High blood pressure reported by the respondent.

eatveg: Frequency of vegetable consumption.ncy of vegetable consumption.

4.9 Random Forest Classifier - ADASYN, best parameters, feature reduction

After the feature engineering, I trained a new Random Forest on just the best features that the model picked out, and used the same oversampling technique and best parameters. For an even better model, I could've completed another grid search or randomized search to optimize the model more.

[52]:
```
top_features = ['hltprnt', 'chldhhe', 'hltprbn', 'domicil', 'hltprsc',
                'alcbnge', 'cgtsmok', 'alcfreq', 'dshltms', 'dosprt',
                'health', 'fnsdfml', 'hltprbp', 'eatveg', 'rlgdgr', 'hltprhb',␣
 ↪'happy']
X_reduced = df_cleaned[top_features]
y = df_cleaned['hltpral']

# Split the dataset
X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced =␣
 ↪train_test_split(
    X_reduced, y, test_size=0.2, random_state=42
)

# Apply scaling
scaler = StandardScaler()
X_train_reduced_scaled = scaler.fit_transform(X_train_reduced)
X_test_reduced_scaled = scaler.transform(X_test_reduced)

# Apply ADASYN to the reduced feature set
adasyn = ADASYN(random_state=42)
X_train_reduced_adasyn, y_train_reduced_adasyn = adasyn.
 ↪fit_resample(X_train_reduced_scaled, y_train_reduced)

# Train the Random Forest on ADASYN data
rf_best_reduced = RandomForestClassifier(**rf_best.get_params())
rf_best_reduced.fit(X_train_reduced_adasyn, y_train_reduced_adasyn)

# Evaluate the reduced model
y_pred_reduced = rf_best_reduced.predict(X_test_reduced_scaled)
y_pred_prob_reduced = rf_best_reduced.predict_proba(X_test_reduced_scaled)[:, 1]

# Classification report
print("Classification Report (ADASYN + Reduced Features):")
print(classification_report(y_test_reduced, y_pred_reduced))
```

```python
# ROC-AUC
roc_auc_reduced = roc_auc_score(y_test_reduced, y_pred_prob_reduced)
print(f"ROC-AUC (ADASYN + Reduced Features): {roc_auc_reduced:.4f}")
```

```
Classification Report (ADASYN + Reduced Features):
              precision    recall  f1-score   support

           0       0.91      0.81      0.86      3876
           1       0.26      0.45      0.33       562

    accuracy                           0.77      4438
   macro avg       0.58      0.63      0.59      4438
weighted avg       0.83      0.77      0.79      4438

ROC-AUC (ADASYN + Reduced Features): 0.7504
```

The model with reduced features only performed slightly worse, with a lower ROC-AUC score of 0.75, and slightly lower precision, recall, and f1-scores. The accuracy score improved slightly however, from 75% to 77%.

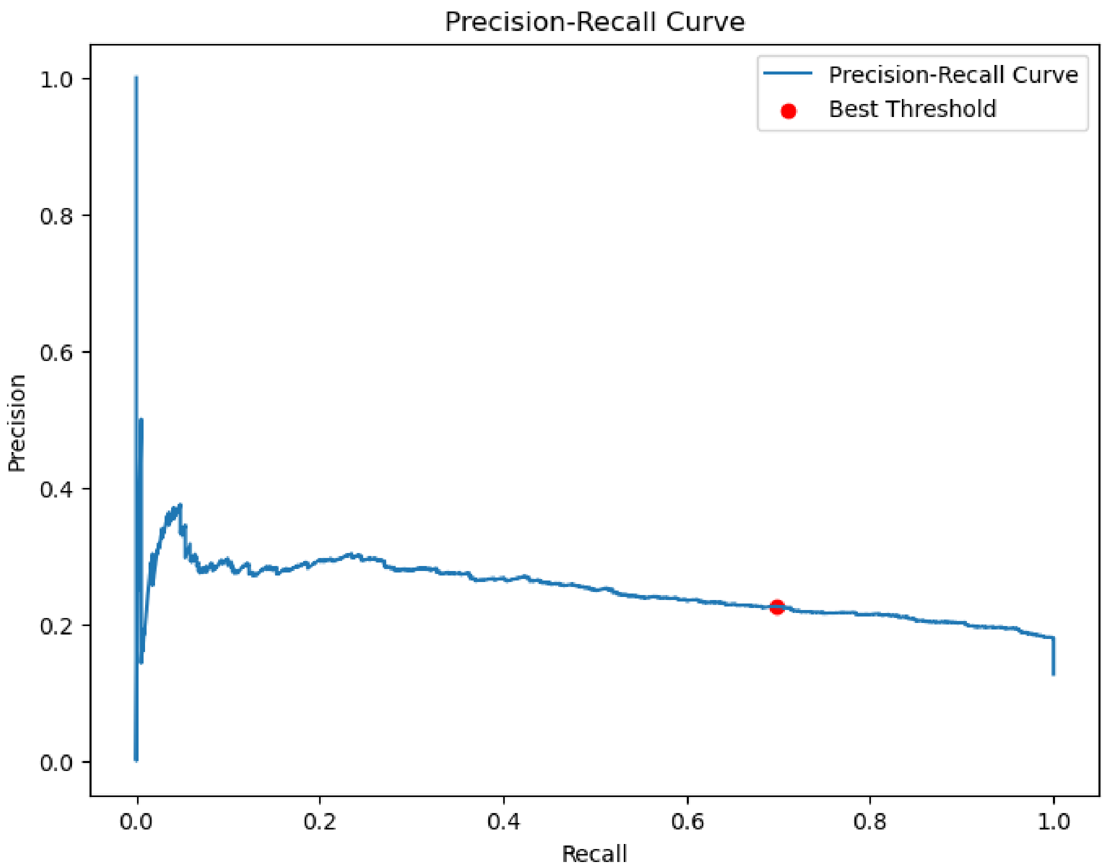
4.10 Random Forest Classifier - ADASYN, best parameters, adjusted threshold, feature reduction

I also decided to adjust the threshold of this model just to see how much more I could balance the precision-recall scores, and to see how much the threshold had changed.

[53]:
```python
y_pred_prob_reduced = rf_best_reduced.predict_proba(X_test_reduced_scaled)[:, 1]

# Compute Precision-Recall values
precisions, recalls, thresholds = precision_recall_curve(y_test_reduced,␣
 ↪y_pred_prob_reduced)

# Find the optimal threshold based on F1 score
f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-10)  #␣
 ↪Avoid division by zero
best_threshold_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_threshold_idx]

print(f"Optimal Threshold: {best_threshold:.4f}")

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recalls, precisions, label='Precision-Recall Curve')
plt.scatter(recalls[best_threshold_idx], precisions[best_threshold_idx],␣
 ↪color='red', label='Best Threshold')
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```python
plt.title("Precision-Recall Curve")
plt.legend()
plt.show()

# Make predictions using the new threshold
y_pred_adjusted = (y_pred_prob_reduced >= best_threshold).astype(int)

# Evaluate predictions with the adjusted threshold
print("Classification Report at Adjusted Threshold:")
print(classification_report(y_test_reduced, y_pred_adjusted))

# Compute and display ROC-AUC
roc_auc_reduced = roc_auc_score(y_test_reduced, y_pred_prob_reduced)
print(f"ROC-AUC (based on probabilities): {roc_auc_reduced:.4f}")
```

```
Optimal Threshold: 0.3955
```




```
Classification Report at Adjusted Threshold:
              precision    recall  f1-score   support
```

```
              0       0.94      0.65      0.77      3876
              1       0.23      0.70      0.34       562

       accuracy                           0.66      4438
      macro avg       0.58      0.68      0.56      4438
   weighted avg       0.85      0.66      0.72      4438

ROC-AUC (based on probabilities): 0.7504
```

With an adjusted threshold, this model has an improved recall score of 70%, which is significantly higher than before, with only a slightly reduced precision score for the minority class. The ROC-AUC score doesn't change with this method, so it's still at a moderate score of 0.75, and could still be improved.

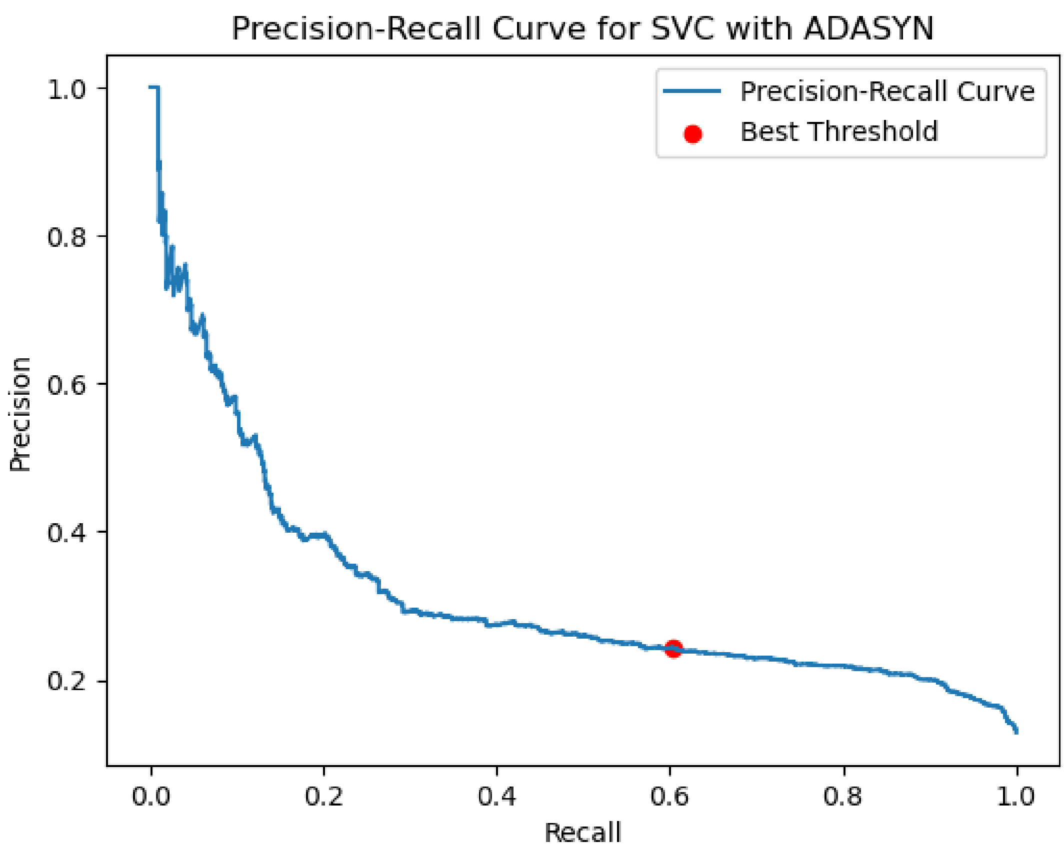
4.11 Support Vector Classifier - ADASYN, adjusted threshold

I decided to use ADAYSN as the oversampling technique for the support vector machine, because of the higher AUC-ROC score the Random Forests saw with this technique.

[54]:
```python
# Apply ADASYN to balance the dataset
adasyn = ADASYN(sampling_strategy='minority', random_state=42)
X_train_resampled, y_train_resampled = adasyn.fit_resample(X_train_scaled,␣
 ↪y_train)

# Initialize the Support Vector Classifier with probability estimation
svc = SVC(probability=True, random_state=42)

# Train the SVC on the resampled data
svc.fit(X_train_resampled, y_train_resampled)

# Get prediction probabilities for the test set
y_pred_prob = svc.predict_proba(X_test_scaled)[:, 1]

# Compute the precision-recall curve and the best threshold
precisions, recalls, thresholds = precision_recall_curve(y_test, y_pred_prob)
f1_scores = 2 * (precisions * recalls) / (precisions + recalls + 1e-10)  #␣
 ↪Avoid division by zero
best_threshold_idx = np.argmax(f1_scores)
best_threshold = thresholds[best_threshold_idx]

print(f"Optimal Threshold: {best_threshold}")

# Plot Precision-Recall Curve
plt.plot(recalls, precisions, label='Precision-Recall Curve')
plt.scatter(recalls[best_threshold_idx], precisions[best_threshold_idx],␣
 ↪color='red', label='Best Threshold')
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve for SVC with ADASYN")
plt.legend()
plt.show()

# Make predictions using the best threshold
y_pred_adjusted = (y_pred_prob >= best_threshold).astype(int)

# Evaluate the model with the adjusted threshold
print("Classification Report at Adjusted Threshold:")
print(classification_report(y_test, y_pred_adjusted))

# Evaluate the ROC-AUC
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
Optimal Threshold: 0.09491121131447976
```




```
Classification Report at Adjusted Threshold:
              precision    recall  f1-score   support

           0       0.92      0.72      0.81      5796
```

```
           1       0.24      0.60      0.35       861

    accuracy                           0.71      6657
   macro avg       0.58      0.66      0.58      6657
weighted avg       0.84      0.71      0.75      6657

ROC-AUC: 0.74771619499691
```

The SVC model with an adjusted threshold has a high recall, but pretty low precision score for the minority class, and a lower accuracy score of 71% as well. The ROC-AUC score is a bit lower than the Random Forest models as well, meaning it overall performed worse than the Random Forests did, but better than the Logistic Regression models.

4.12 XGB Classifier - SMOTE

For my first XGB Classifier, I decided to try SMOTE for the oversampling technique.

[55]:
```
# Split the data
X = df_cleaned.drop(columns=['hltpral', 'hltpral_mapped'])  # Features
y = df_cleaned['hltpral']  # Target (0 = No Allergy, 1 = Allergy)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Apply scaling (if necessary)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create the SMOTE oversampling and XGBClassifier pipeline
smote = SMOTE(sampling_strategy='auto', random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

pipeline = Pipeline([
    ('smote', smote),
    ('classifier', xgb)
])

# Train the model
pipeline.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = pipeline.predict(X_test_scaled)
y_pred_prob = pipeline.predict_proba(X_test_scaled)[:, 1]

from sklearn.metrics import classification_report, roc_auc_score

# Print classification report
```

```
print(classification_report(y_test, y_pred))

# Print ROC-AUC score
roc_auc = roc_auc_score(y_test, y_pred_prob)
print(f"ROC-AUC: {roc_auc}")
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:20] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

              precision    recall  f1-score   support

           0       0.89      0.98      0.93      3876
           1       0.51      0.15      0.24       562

    accuracy                           0.87      4438
   macro avg       0.70      0.57      0.58      4438
weighted avg       0.84      0.87      0.84      4438

ROC-AUC: 0.7881598228352964
```

The XGB Classifier performed better overall than all the other models, with the highest ROC-AUC score so far of 78.8, and an accuracy of 87%. The recall score was low however, at 0.15, and the f1-score could be higher as well.

4.13 XGB Classifier - ADAYSN

This time, I tried using ADAYSN, to see if the model would perform any better.

[56]:
```
# Assuming your dataset is prepared and the features and labels are ready
X = df_cleaned.drop(columns=['hltpral', 'hltpral_mapped'])  # Features
y = df_cleaned['hltpral']  # Target (0 = No Allergy, 1 = Allergy)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize ADASYN for resampling
adasyn = ADASYN(sampling_strategy='minority', random_state=42)

# Create the XGBClassifier model
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Create a pipeline with ADASYN and the classifier
pipeline = Pipeline([
    ('adasyn', adasyn),
```

```python
    ('classifier', xgb)
])

# Train the model
pipeline.fit(X_train, y_train)

# Predict and evaluate
y_pred = pipeline.predict(X_test)

# Evaluation metrics
from sklearn.metrics import classification_report, roc_auc_score
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, pipeline.predict_proba(X_test)[:, 1]))
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:21] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      3876
           1       0.45      0.16      0.24       562

    accuracy                           0.87      4438
   macro avg       0.67      0.57      0.58      4438
weighted avg       0.83      0.87      0.84      4438

ROC-AUC: 0.7800861400937973
```

This model performed very similarly to the previous one, with only a minor difference in the ROC-AUC score, and a slight drop in precision for the minority class, but otherwise was almost identical.

I decided to perform a grid search to see if the model could be improved with better parameters.

[57]:
```python
# Grid Search for XGB Classifier
param_grid = {
    'classifier__max_depth': [3, 5, 7],
    'classifier__learning_rate': [0.01, 0.1, 0.2],
    'classifier__n_estimators': [50, 100, 200]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='roc_auc')
grid_search.fit(X_train, y_train)

print(f"Best Params: {grid_search.best_params_}")
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:21] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:22] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:22] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:23] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:23] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:24] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:24] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:25] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:26] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:26] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:27] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:27] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:28] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:28] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:29] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:30] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:30] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:31] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:32] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:32] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:33] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:33] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:34] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:35] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:35] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:36] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:38] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:38] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:39] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:39] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:40] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:40] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:41] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:41] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:42] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:43] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:43] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:44] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:44] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:45] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:45] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:46] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:46] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:47] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:48] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:49] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:49] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:50] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:50] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:51] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:52] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:53] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:54] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:54] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:56] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:56] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:56] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:57] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:57] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:58] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:59] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:23:59] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:00] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:01] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:01] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:02] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:02] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:03] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:03] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:04] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:05] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:05] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:06] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:07] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:07] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:08] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:09] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:09] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:10] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:11] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:12] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\Emilia\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarning:
[21:24:13] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-
group-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Best Params: {'classifier__learning_rate': 0.1, 'classifier__max_depth': 7,
'classifier__n_estimators': 100}
```

4.14 XGB Classifier - ADASYN, best parameters

[58]:
```python
# Define the XGBoost classifier with the best parameters
xgb = XGBClassifier(learning_rate=0.1, max_depth=7, n_estimators=100,␣
 ↪random_state=42)

# Define the pipeline
pipeline = Pipeline([
    ('sampling', ADASYN(sampling_strategy='auto', random_state=42)),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False)),
    ('classifier', xgb)
])

# Train the model
pipeline.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, pipeline.predict_proba(X_test)[:, 1]))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      3876
           1       0.49      0.06      0.10       562

    accuracy                           0.87      4438
   macro avg       0.69      0.52      0.52      4438
weighted avg       0.83      0.87      0.83      4438

ROC-AUC: 0.7983374282471932
```

After applying the grid search, the model had a higher ROC score, but a lower recall score, with the accuracy still the same.

4.15 XGB Classifier - ADASYN, best parameters, class weights adjusted

[59]:
```
# Define the XGBoost classifier with the best parameters
xgb = XGBClassifier(learning_rate=0.1, max_depth=7, n_estimators=100,␣
↪random_state=42, scale_pos_weight=4)
# Define the pipeline
pipeline = Pipeline([
    ('sampling', ADASYN(sampling_strategy='auto', random_state=42)),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False)),
    ('classifier', xgb)
])

# Train the model
pipeline.fit(X_train, y_train)

# Predict and evaluate the model
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, pipeline.predict_proba(X_test)[:, 1]))
```

```
              precision    recall  f1-score   support

           0       0.92      0.84      0.88      3876
           1       0.32      0.53      0.40       562

    accuracy                           0.80      4438
   macro avg       0.62      0.68      0.64      4438
weighted avg       0.85      0.80      0.82      4438

ROC-AUC: 0.7975579255864174
```

I adjusted the class weights, and managed to get a better model with a much higher recall score and f1-score. The accuracy was reduced slightly, same with the precision score for minority classes, due to the trade-off, but overall it was a more balanced model with a higher ROC-AUC score.

[61]:
```
X_reduced = df_cleaned[top_features]
y = df_cleaned['hltpral']

# Split the data into training and testing sets
X_train_reduced, X_test_reduced, y_train_reduced, y_test_reduced =␣
↪train_test_split(
    X_reduced, y, test_size=0.2, random_state=42, stratify=y)
```

4.16 XGB Classifier - ADASYN, best parameters, class weights adjusted, feature reduction

[62]:
```
xgb_best = XGBClassifier(
    learning_rate=0.1,
    max_depth=7,
    n_estimators=100,
    random_state=42,
    scale_pos_weight=4
)

# Define the pipeline
pipeline = Pipeline([
    ('sampling', ADASYN(sampling_strategy='auto', random_state=42)),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False)),
    ('classifier', xgb_best)
])

# Train the model on the reduced dataset
pipeline.fit(X_train_reduced, y_train_reduced)

# Predict and evaluate the model
y_pred_reduced = pipeline.predict(X_test_reduced)
print(classification_report(y_test_reduced, y_pred_reduced))
print("ROC-AUC:", roc_auc_score(y_test_reduced, pipeline.
 ↪predict_proba(X_test_reduced)[:, 1]))
```

```
              precision    recall  f1-score   support

           0       0.91      0.80      0.85      3883
           1       0.25      0.47      0.33       555

    accuracy                           0.76      4438
   macro avg       0.58      0.63      0.59      4438
weighted avg       0.83      0.76      0.79      4438

ROC-AUC: 0.7541354901128273
```

Next, I refined the model by reducing the features to only the top predictors identified by the Random Forest. While this slightly reduced the model's performance, it still outperformed most of the previous models overall. Due to its strong balance of simplicity and effectiveness, I selected this version for my final project.

5 Exporting Best Model

[63]:
```
import joblib

model_data = {
    'model': xgb_best,
    'best_threshold': best_threshold
}

# Save the model data as a .pkl file
model_filename = 'allergy_model.pkl'
joblib.dump(model_data, model_filename)

print(f"Model and threshold saved as {model_filename}")
```

```
Model and threshold saved as allergy_model.pkl
```

Project Files for Flask Web App

app.py

```python
from flask import Flask, request, render_template
import pandas as pd
import numpy as np
import joblib
import warnings
from sklearn.exceptions import DataConversionWarning

warnings.filterwarnings(action='ignore', category=UserWarning, module='sklearn')

app = Flask(__name__)

@app.route('/')
@app.route('/about')
def about():
    return render_template("about.html")

@app.route('/allergyPredictor')
def allergyPredictor():
    return render_template("allergyPredictor.html")

def preprocessDataAndPredict(hltprnt, chldhhe, hltprbn, domicil, hltprsc, alcbnge, cgtsmok,
alcfreq,
                             dshltms, dosprt, health, fnsdfml, hltprbp, eatveg, rlgdgr,
hltprhb, happy):
    # Create DataFrame from inputs
    data = pd.DataFrame({
        'hltprnt': [hltprnt],
        'chldhhe': [chldhhe],
        'hltprbn': [hltprbn],
        'domicil': [domicil],
        'hltprsc': [hltprsc],
        'alcbnge': [alcbnge],
        'cgtsmok': [cgtsmok],
        'alcfreq': [alcfreq],
        'dshltms': [dshltms],
        'dosprt': [dosprt],
        'health': [health],
        'fnsdfml': [fnsdfml],
        'hltprbp': [hltprbp],
        'eatveg': [eatveg],
        'rlgdgr': [rlgdgr],
        'hltprhb': [hltprhb],
        'happy': [happy]
    })

    file = open("allergy_model.pkl", "rb")
    model_data = joblib.load(file)
    file.close()

    # Extract model and threshold
    trained_model = model_data['model']
    best_threshold = model_data['best_threshold']

    # Use the model to predict probabilities
    proba = trained_model.predict_proba(data)[0][1]

    # Apply threshold to determine the prediction
    prediction = 1 if proba >= best_threshold else 0

    return np.round(proba * 100, 1).tolist()

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == "POST":
        try:
            # Collect form data
            form_data = {key: request.form.get(key) for key in request.form.keys()}
```

```python
            print("Form Data Received:", form_data)  # Log received data for debugging

            # Convert inputs to integers
            hltprnt = int(form_data['hltprnt'])
            chldhhe = int(form_data['chldhhe'])
            hltprbn = int(form_data['hltprbn'])
            domicil = int(form_data['domicil'])
            hltprsc = int(form_data['hltprsc'])
            alcbnge = int(form_data['alcbnge'])
            cgtsmok = int(form_data['cgtsmok'])
            alcfreq = int(form_data['alcfreq'])
            dshltms = int(form_data['dshltms'])
            dosprt = int(form_data['dosprt'])
            health = int(form_data['health'])
            fnsdfml = int(form_data['fnsdfml'])
            hltprbp = int(form_data['hltprbp'])
            eatveg = int(form_data['eatveg'])
            rlgdgr = int(form_data['rlgdgr'])
            hltprhb = int(form_data['hltprhb'])
            happy = int(form_data['happy'])

            # Call prediction function
            prediction = preprocessDataAndPredict(hltprnt, chldhhe, hltprbn, domicil, hltprsc,
                                                  alcbnge, cgtsmok, alcfreq, dshltms, dosprt,
                                                  health, fnsdfml, hltprbp, eatveg, rlgdgr,
                                                  hltprhb, happy)

            # Return prediction to the template
            return render_template('predict.html', prediction=prediction)

        except ValueError as ve:
            print("ValueError:", str(ve))
            return render_template('predict.html', message="Please enter valid numerical values
for all fields.")

        except Exception as e:
            print("General Error:", str(e))
            return render_template('predict.html', message=f"An unexpected error occurred:
{str(e)}")

    # Render predict page if method is GET
    return render_template('predict.html')

@app.route('/resume')
def resume():
    return render_template('resume.html')

@app.route('/projects')
def projects():
    return render_template('projects.html')

# Run on the correct port
if __name__ == '__main__':
    app.debug = True
    app.run(host="0.0.0.0", port=8080, debug=True)
```

About Me Allergy Predictor Resume Projects
{% block header %}{% endblock %}
{% for message in get_flashed_messages() %}
{{ message }}
{% endfor %} {% block content %}{% endblock %}

{% extends "base.html" %} {% block title %}About Me{% endblock %} {% block content %}

About Me

Welcome to my project!

Hi! My legal first name is Hanna, but I go by my middle name, Emilia.

Iâ€™m originally from Hungary. I immigrated to the U.S. when I was almost two, and grew up in New Hampshire, where I still live today. I have two older brothers as well as two younger sistersâ€”whom were born here.

I completed my Bachelor's of Science in Business Administration last year, with a concentration in Marketing, and am excited to be now finishing up my Master's in Data Science! Once I graduate, I aspire to work as a data scientist in a hybrid remote role, where I can hopefully combine the technical skills I've learned with my creative thinking to make a positive impact.

Beyond my academic and professional pursuits, I enjoy expressing my creativity through arts and crafts like drawing, painting, and crocheting. I also love reading and writing, and do competitive gymnastics, which I train for 12 hours a week.

Feel free to look around and explore the other pages of my project to see more!

{% endblock %}

{% extends 'base.html' %} {% block title %}Allergy Risk Predictor{% endblock %} {% block content %}

Allergy Risk Predictor

Allergies are a common health issue that affects billions worldwide, with about 1 in 3 adults in the U.S. experiencing allergies. Similar trends are observed globally, and reports indicate that the prevalence of food allergies in children increased by 50% between 1997 and 2011 ("Allergies, 2023"). Approximately 30â€“40% of the world's population now suffers from some form of allergy (Davies, 2023).

This project leverages data collected through the European Social Survey, an international study conducted in 2023. The survey included over 22,000 participants across 31 European countries and involved hour-long face-to-face interviews. Using this comprehensive dataset, we trained a machine learning model to analyze lifestyle and socioeconomic factors to predict the likelihood of developing or currently having allergies.

Our approach examines patterns in various factors such as diet, exercise, and living environment to better understand how these contribute to allergy risks. This model can help raise awareness of these contributing factors and empower individuals to take preventative actions.

How to Use This Tool

To use this tool, fill out the form below with information about your lifestyle and socioeconomic background. Based on the inputs you provide, the model will calculate a percentage likelihood of having allergies.

How would you describe your health in general?
Choose an option

How happy would you say you are in general? 1 being extremely unhappy, and 10 being extremely happy

Regardless of whether you belong to a religion, how religious would you say you are? 0 being not religious at all, and 10 being very religious

How often do you eat vegetables or salads (excluding potatoes)?
Choose an option

How many days a week do you do sports? (1-7)

In the last 12 months, how often have you had a drink containing alcohol?
Choose an option

In the last 12 months, how often have you drank to the point of feeling intoxicated?
Choose an option

Which of the following best describes your smoking habits?
Choose an option

Which of the following best describes the area where you live?
Choose an option

Have you ever had any children of your own, step-children, adopted children, foster children, or a partner's child living in your household?
Choose an option

How often did you or your family experience severe financial difficulties when you were growing up?
Choose an option

Have you discussed your health with a medical professional in the last 12 months?
Choose an option

In the last 12 months, have you experienced any of the following health problems?

High blood pressure
Choose an option

Breathing problems
Choose an option

Back or neck pain
Choose an option

Skin conditions
Choose an option

If you have NOT experienced any of the above health problems, or the following: heart or circulation problems, muscular joint pain in arms, legs, hands or feet, stomach or digestion related issues, severe headaches, diabetes, or allergies, then select No. Otherwise, select Yes.

Choose an option

Get Allergy Risk Prediction

References

- â€œAllergies Are Getting More Common. Playing in the Dirt Could Help.â€ Memorialhermann, 28 July 2023, memorialhermann.org/health-wellness/health/allergies-getting-more-common. .
- Davies, Dave. â€œWhy Our Allergies Are Getting Worse -and What to Do about It.â€ NPR, NPR, 30 May 2023, www.npr.org/sections/health-shots/2023/05/30/1178433166/theresa-macphail-allergic-allergies.

{% endblock %}

{% extends 'base.html' %} {% block title %}Allergy Risk Prediction{% endblock %} {% block content %}

Allergy Risk Prediction

{% if prediction %}

According to our calculations, you have a {{ prediction }}% chance of developing or having allergies.

Our prediction is based on an analysis of the most important factors influencing allergy risk, the top 15 of which are shown below.

Characteristics (also known as features) with higher importance contribute more significantly to the model's predictions. For example, 'hltprnt' has the greatest influence on determining allergy risk.

Feature Descriptions

hltprnt: Indicates whether the respondent has had specific health problems in the last 12 months.

domicil: The type of area where the respondent lives (e.g., big city, countryside).

hltprbn: Back or neck pain reported by the respondent.

alcfreq: Frequency of alcohol consumption in the last 12 months.

dosprt: Days per week the respondent engages in sports activities.

alcbnge: Frequency of binge drinking in the last 12 months.

hltprsc: Presence of skin conditions reported by the respondent.

cgtsmok: Smoking habits of the respondent.

health: General health status of the respondent.

dshltms: Whether health was discussed with a medical professional in the last 12 months.

hltprsd: Stomach or digestion-related issues reported by the respondent.

fnsdfml: Frequency of financial difficulties during childhood.

happy: General happiness level of the respondent (scale of 1 to 10).

hltprhb: High blood pressure reported by the respondent.

eatveg: Frequency of vegetable consumption.

{% elif message %}

{{ message }}

{% endif %}
{% endblock %}

{% extends "base.html" %} {% block title %}Resume{% endblock %} {% block content %}

My Resume

Hanna Emilia Halfinger
Londonderry, NH 03053
603-425-8850
ehalfinger@icloud.com

Professional Summary
Dedicated and detail-oriented data science graduate student with expertise in statistical analysis, Python, R, SQL, and data visualization. Skilled at leveraging data-driven insights to solve complex business challenges and improve organizational decision-making. Passionate about research, data storytelling, and delivering innovative solutions to real-world problems. Fluent in Hungarian and committed to continuous learning and professional growth.

Education

Master of Science in Data Science
Eastern University â€” Expected Graduation: December 2024
- Advanced coursework in machine learning, statistical modeling, and data visualization.
- Current capstone project focuses on developing predictive models for allergies using socioeconomic data, leveraging Flask and AWS for web app deployment.

Bachelor of Science in Business Administration, Marketing Concentration
Southern New Hampshire University â€” 2021â€“2023
- Graduated with distinction; emphasis on data-driven marketing strategies and consumer behavior analysis.

Dual Enrollment Program
Nashua & Manchester Community College â€” 2019â€“2021
- Completed over 60 college credits while attending high school.

Technical Skills
- Programming & Data Tools: Python, R, SQL, Flask, AWS, Tableau, Microsoft Excel
- Data Science Expertise: Statistical analysis, predictive modeling, data cleaning, feature engineering
- Soft Skills: Problem-solving, analytical thinking, strong work ethic, effective communication, teamwork
- Languages: Fluent in Hungarian

Professional Experience

Customer Success Coordinator
Reliable Respiratory â€” January 2024 â€“ May 2024
- Facilitated communication between patients, insurance companies, and medical staff to ensure timely documentation and approval of critical medical services.
- Processed patient orders and maintained accurate medical records, improving service efficiency.

Server
Sky Meadow Country Club â€” November 2021 â€“ Present
- Delivered high-quality service across dining, event hosting, and bar operations, consistently exceeding member expectations.
- Developed strong client relationships by anticipating needs in a fast-paced environment.

{% endblock %}

{% extends "base.html" %} {% block title %}Projects{% endblock %} {% block content %}

My Projects

Ethics Video Project

How AI Impacts Recruitment - DTSC 690

Github

{% endblock %}

Test.py

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, World!"

if __name__ == '__main__':
    app.run()
```

Style.css

```
.c1 {
padding: 0;
margin: 0;
}
.c2 {
font-size: 11pt;
font-family: "Calibri";
font-weight: 700;
}
.c3 {
font-size: 10pt;
font-family: "Calibri";
font-weight: 700;
}
.c4 {
color: #000000;
text-decoration: none;
vertical-align: baseline;
font-style: normal;
}
.c5 {
font-size: 10pt;
font-family: "Calibri";
font-weight: 400;
}
.c6 {
margin-left: 36pt;
padding-top: 0pt;
padding-left: 0pt;
padding-bottom: 0pt;
line-height: 1.15;
orphans: 2;
widows: 2;
text-align: left;
}
.c7 {
margin-left: 54pt;
padding-top: 0pt;
padding-left: 0pt;
padding-bottom: 0pt;
line-height: 1;
orphans: 2;
widows: 2;
text-align: left;
}
.c8 {
padding-top: 0pt;
padding-bottom: 0pt;
line-height: 1;
orphans: 2;
widows: 2;
text-align: center;
margin-right: 18pt;
}
```

```
.c10 {
font-style: italic;
}
.c21 {
padding-top: 4pt;
padding-bottom: 0pt;
line-height: 1;
orphans: 2;
widows: 2;
text-align: left;
}
.c23 {
margin-left: 18pt;
padding-top: 0pt;
padding-bottom: 0pt;
line-height: 1;
orphans: 2;
widows: 2;
text-align: justify;
height: 12pt;
}
.c26 {
margin-left: 18pt;
}
.c28 {
height: 12pt;
}
.c30 {
background-color: #ffffff;
}
.c31 {
font-weight: 700;
font-size: 20pt;
font-family: "Calibri";
}
.c32 {
padding-top: 0pt;
padding-bottom: 0pt;
line-height: 1.15;
orphans: 2;
widows: 2;
text-align: left;
}
.c34 {
padding-top: 0pt;
border-bottom-color: #000000;
border-bottom-width: 0.5pt;
padding-bottom: 1pt;
line-height: 1;
border-bottom-style: solid;
orphans: 2;
widows: 2;
text-align: left;
}
```

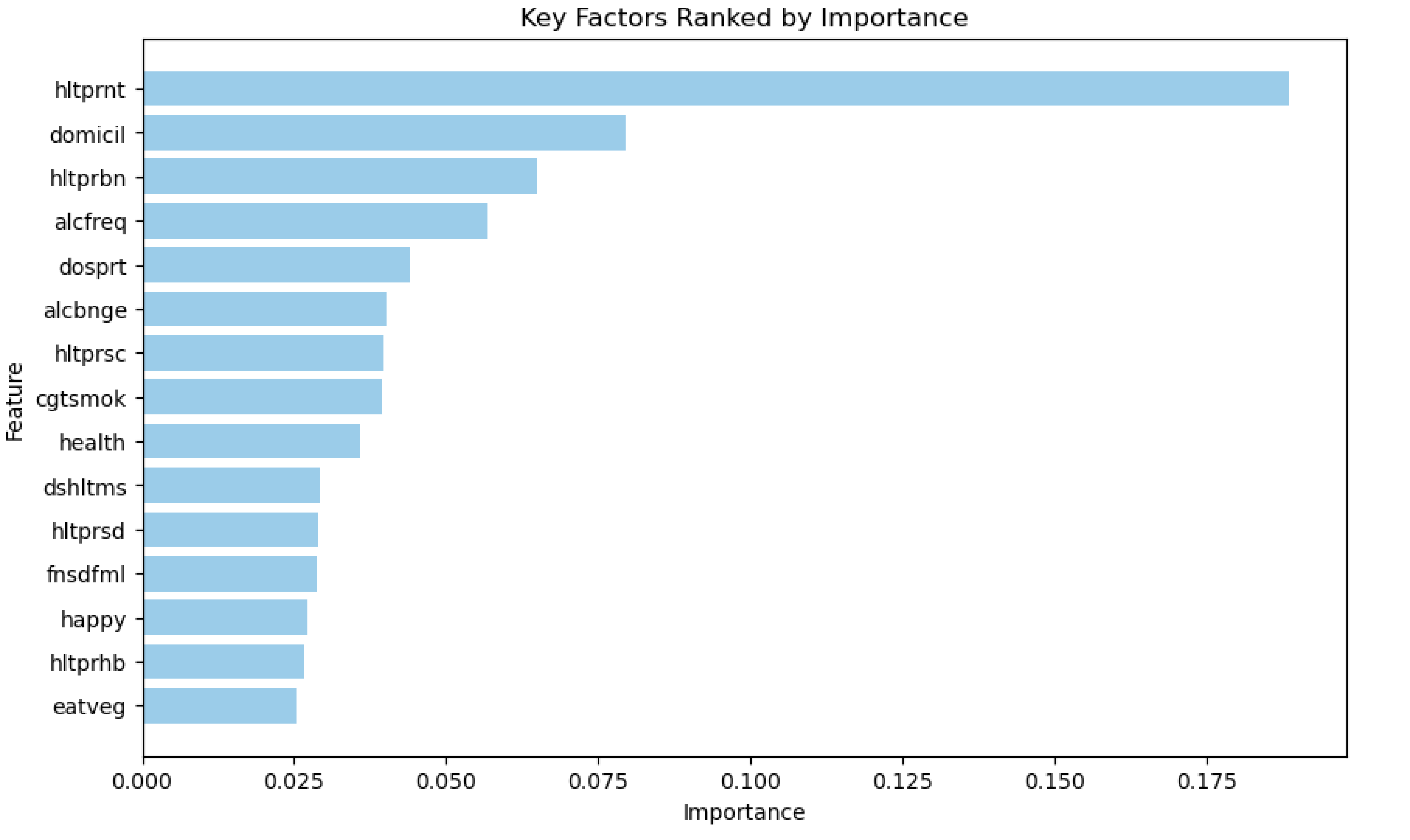
```
.c35 {
text-indent: -18pt;
}
.c36 {
margin-left: 36pt;
}
.c38 {
padding-top: 4pt;
padding-bottom: 4pt;
line-height: 1;
orphans: 2;
widows: 2;
text-align: left;
}
ol {
margin: 0;
padding: 0;
}
table td,
table th {
padding: 0;
}
.title {
padding-top: 24pt;
color: #000000;
font-weight: 700;
font-size: 36pt;
padding-bottom: 6pt;
font-family: "Cambria";
line-height: 1;
page-break-after: avoid;
orphans: 2;
widows: 2;
text-align: left;
}
.subtitle {
padding-top: 18pt;
color: #666666;
font-size: 24pt;
padding-bottom: 4pt;
font-family: "Georgia";
line-height: 1;
page-break-after: avoid;
font-style: italic;
orphans: 2;
widows: 2;
text-align: left;
}
li {
color: #000000;
font-size: 12pt;
font-family: "Cambria";
}
p {
```

```
margin: 0;
color: #000000;
font-size: 12pt;
font-family: "Cambria";
}

/* ------------------------------ NAVIGATION BAR STYLING ------------------------------ */

.topnav {
background-color: #333;
overflow: hidden;
}

/* Style the links inside the navigation bar */
.topnav a {
float: left;
color: #f2f2f2;
text-align: center;
padding: 14px 16px;
text-decoration: none;
font-size: 17px;
}

/* Change the color of links on hover */
.topnav a:hover {
background-color: #ddd;
color: black;
}

/* Add a color to the active/current link */
.topnav a.active {
background-color: #04AA6D;
color: white;
}
/* ------------------------- SIDE NAVIGATION BAR STYLING FOR DISSERTATION PAGE -------------------------
*/

/* The sidebar menu */
.sidenav {
height: 100%; /* Full-height: remove this if you want "auto" height */
width: 160px; /* Set the width of the sidebar */
position: fixed; /* Fixed Sidebar (stay in place on scroll) */
z-index: 1; /* Stay on top */
top: 0; /* Stay at the top */
left: 0;
background-color: #111; /* Black */
overflow-x: hidden; /* Disable horizontal scroll */
padding-top: 20px;
}

/* The navigation menu links */
.sidenav a {
padding: 6px 8px 6px 16px;
text-decoration: none;
```

```
font-size: 25px;
color: #818181;
display: block;
}

/* When you mouse over the navigation links, change their color */
.sidenav a:hover {
color: #f1f1f1;
}

/* Style page content */
.main {
margin-left: 160px; /* Same as the width of the sidebar */
padding: 0px 10px;
}

/* On smaller screens, where height is less than 450px, change the style of the sidebar (less
padding and a smaller font size) */
@media screen and (max-height: 450px) {
.sidenav {padding-top: 15px;}
.sidenav a {font-size: 18px;}
}


#wrapper {
width: 920px;
height: auto;
margin: 0 auto;
}
#home1 {
width: 47.5%;
height: 300px;
float: left;
margin-right: 5%;
}

#home2 {
width: 47.5%;
height: 300px;
float: left;
}

.clear{
clear: both;
}

@media (max-width:767px) {
#wrapper{
width: 100%;
height: auto;
}
#home1 {
width: 100%;
height: auto;
```

```
float: none;
}
#home2 {
width: 100%;
height: auto;
float: none;
}
}

.content-wrapper {
margin: 0 auto; /* Centers the content */
max-width: 800px; /* Restricts the width for readability */
padding: 20px; /* Adds inner padding */
background-color: #f9f9f9; /* Optional: Background for contrast */
border-radius: 10px; /* Optional: Rounded corners */
box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1); /* Optional: Shadow effect */
}

.hungary-picture {
display: block;
margin: 20px auto; /* Centers the picture */
max-width: 800px; /* Restricts the size */
box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1); /* Optional: Shadow effect */
}

p {
margin-bottom: 20px; /* Adds space below each paragraph */
line-height: 1.6; /* Adjusts line spacing for better readability */
}

/* General styles for form elements */
.form-group {
margin-bottom: 20px; /* Adds space between form groups */
}

.disclaimer {
font-size: 0.9em;
font-style: italic;
color: #555; /* Optional: Slightly gray color for the text */
}

h1 {
font-size: 2.5rem; /* Adjust this size as desired */
font-weight: bold; /* Optional: make it bold */
margin-bottom: 20px; /* Optional: spacing below */
}

h2 {
font-size: 2rem; /* Slightly smaller than h1 */
font-weight: bold; /* Optional */
margin-bottom: 15px; /* Optional */
}

.references {
```

```
line-height: 2; /* Adds space between lines */
margin-top: 10px; /* Adds space before the references list */
}

.references li {
margin-bottom: 15px; /* Adds space between references */
}
```


Key Factors Ranked by Importance
hltprnt
domicil
hltprbn
alcfreq
dosprt
alcbnge
hltprsc
cgtsmok
health
dshltms
hltprsd
fnsdfml
happy
hltprhb
eatveg
Feature
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Importance

Requirements.txt

```
awscli==1.36.17
awsebcli==3.21.0
blinker==1.9.0
botocore==1.35.76
cement==2.10.14
certifi==2024.8.30
charset-normalizer==3.4.0
click==8.1.7
colorama==0.4.6
contourpy==1.3.1
cycler==0.12.1
docutils==0.16
Flask==3.1.0
fonttools==4.55.0
gunicorn==23.0.0
idna==3.10
imbalanced-learn==0.12.4
imblearn==0.0
itsdangerous==2.2.0
Jinja2==3.1.4
jmespath==1.0.1
joblib==1.4.2
kiwisolver==1.4.7
MarkupSafe==3.0.2
matplotlib==3.9.3
numpy==1.24.3
packaging==24.2
pandas==2.2.3
pathspec==0.10.1
patsy==1.0.1
pillow==11.0.0
pyasn1==0.6.1
pyparsing==3.2.0
python-dateutil==2.9.0.post0
pytz==2024.2
PyYAML==6.0.2
requests==2.32.3
rsa==4.7.2
s3transfer==0.10.4
scikit-learn==1.2.2
scipy==1.10.1
seaborn==0.13.2
semantic-version==2.10.0
six==1.16.0
statsmodels==0.14.4
termcolor==2.5.0
threadpoolctl==3.5.0
tzdata==2024.2
urllib3==1.26.20
wcwidth==0.2.13
Werkzeug==3.1.3
xgboost==2.1.3
```

Dockerfile

Use the official Python image as a base

FROM python:3.11-slim

Set the working directory

WORKDIR /app

Copy project files to the container

COPY . /app

Install dependencies

RUN pip install –no-cache-dir -r requirements.txt

Expose the port Flask runs on

EXPOSE 8080

Command to run your app

CMD [“gunicorn”, “-w”, “4”, “-b”, “0.0.0.0:8080”, “app:app”]

PREDICTING ALLERGY DEVELOPMENT WITH MACHINE LEARNING

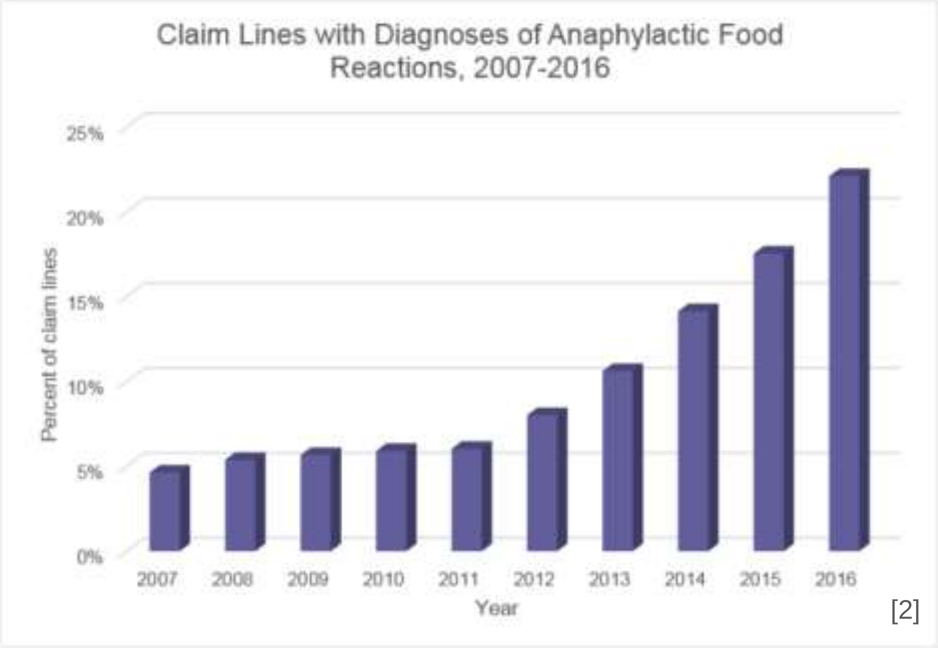
DTSC 691 – HANNA HALFINGER

WHAT WE WILL COVER

- The Rise of Allergies
- Potential Causes and Factors
- Machine Learning Web App
- Code Walkthrough
- Web App Demonstration
- Insights and Outcomes

THE RISE OF ALLERGIES

- Allergies affect 1 in 3 adults in the U.S.
- Food allergies in children increased by 50% between 1997 and 2011 [1]
- Approximately 30–40% of the world's population now suffers from some form of allergy [3]



[2]

POTENTIAL CAUSES AND FACTORS

- Dietary changes towards processed foods
- Too little exposure to potential allergens early in life
- Chemical exposure from cleaning & hygiene products

THE SOLUTION

MACHINE LEARNING WEB APP

Trained to predict the likelihood of developing allergies based on potential causes and factors

DATASET USED

The dataset was obtained by the European Social Survey, an international survey conducted in 2023, that included over 22,000 participants across 31 European countries. The survey involved an hour-long face-to-face interview

FEATURES ANALYZED

- 47 features including lifestyle factors such as:
    - Diet
    - Physical activity
    - Smoking & drinking habits
    - Other health problems
    - Treatments used in last 12 months
    - Exposure to pollutants growing up and in job
    - Type of area lived in (e.g. big city, town, farm)
    - Occupation & industry worked in
    - Total household income

MACHINE LEARNING MODELS

- Logistic Regressor
- Random Forest Classifier
- Support Vector Classifier
- XG Boost Classifier

REFERENCES

- [1] “Allergies Are Getting More Common. Playing in the Dirt Could Help.” Memorialhermann, 28 July 2023, memorialhermann.org/health-wellness/health/allergies-getting-more-common.
- [2] Bloom, Dave. “Private Insurance Claims Related to Anaphylaxis from Food Allergy Have Nearly Quadrupled since 2007.” *SnackSafely.Com*, 21 Aug. 2017, snacksafely.com/2017/08/private-insurance-claims-related-to-food-allergy-induced-anaphylaxis-have-nearly-quadrupled-since-2007/.
- [3] Davies, Dave. “Why Our Allergies Are Getting Worse -and What to Do about It.” NPR, NPR, 30 May 2023, www.npr.org/sections/health-shots/2023/05/30/1178433166/theresa-macphail-allergic-allergies.