Predicting General Health Using BRFSS 2021

CS7389G Final Project, By Javad Mokhtari Koushyar

August 2023

Contents

List of Figures

List of Tables

1 Introduction

The Behavioral Risk Factor Surveillance System (BRFSS) 2021 dataset is a valuable resource, providing comprehensive data that sheds light on critical health-related factors affecting the U.S. population. This dataset is produced annually by the Centers for Disease Control and Prevention (CDC) as part of its ongoing efforts to monitor health conditions and risk behaviors.

The 2021 edition of the BRFSS dataset, like its predecessors, uses data derived from telephone surveys, including both landlines and cellphones, to collect information from adults aged 18 and over from all 50 states, the District of Columbia, and three U.S. territories.

In this experimental project, my goal is to predict the general health of patients using BRFSS 2021 dataset with the help of Machine-Learning. Also, I will contribute a PUI to this project that helps in managing the ML model easier through a web interface.

1.1 Motivation

My motivation for this project is to experiment with the things that I've learned through the course and apply them to a real-world dataset, which is BRFSS 2021. Also, developing a PUI (Perceptual User Interface) can be challenging by its nature. Because handling big data on user interfaces is often challenging work since most of the user interface programs run on a client which has a low amount of computational resources.

For the data-science part, I will use Python, R, scikit-learn, imbalance-learn, etc. tools. Which is a great opportunity for me to get familiar with these technologies and learn by doing.

1.2 Problem Definition

The problem I want to address in this project and solve is to **Predict General Health Using BRFSS**. I believe solving this problem can be interesting for several reasons. Just to mention a few:

1. People may be uncomfortable to express their situation
2. Can be useful in medical decision makings (e.g., Prescription, Hospitalization)
3. Public Health Planning
4. Researchers can study new hypotheses on public health

2 Data Pre-processing

2.1 Understanding BRFSS

Before starting the data pre-processing, it's good to have a look at BRFSS dataset and understand the data inside it. So, first I describe the BRFSS dataset and then I will jump into my data pre-processing steps. First, I downloaded the SAS version of BRFSS 2021 from the official CDC website. To study the dataset, I opened it in the R console. The downloaded file was in a zip format, so the initial step involved unzipping it. Once unzipped, a file named LLCP2021.XPT was obtained. The "LL" in the file name represents Land Line, while "CP" stands for Cell-Phone. This naming convention reflects the evolution of data collection methods, as prior to 2011, BRFSS data was exclusively collected through landlines. Therefore, the dataset we are working with includes data collected from both landlines and cell phones. I use the "rio" package in the R console to import the XPT data file and take a look at it.

```
> llcp2021 <- import("LLCP2021.XPT")
```

Figure 1: Importing BRFSS 2021 dataset in R console using rio package

Now that we have successfully imported it, let's have a look at the list of columns in this dataset:

No.	Column Name
1	_STATE
2	FMONTH
3	IDATE
4	IMONTH
...	...
300	_FRT16A
301	_VEG23A
302	_FRUITE1
303	_VGETE1

Table 1: Columns of BRFSS 2021

As you can see, this dataset has 303 variables (or columns) and it has 438,693 observations (or rows). **Some columns have been removed in Table 1, just for readability purposes, if you want to check the full list of columns you can use the commands 'names' in R console.** The columns that start with an underscore, are calculated variables. For understanding how these columns are calculated we can refer to the official CDC guidelines.

The other thing that was interesting to me, was to see how actual data look like inside this dataset:

_STATE	FMONTH	IDATE	IMONTH	IDAY	...	_VEGETE1
1	1	01192021	01	19	...	0
1	1	01212021	01	21	...	0
1	1	01212021	01	21	...	0
1	1	01172021	01	17	...	0

Table 2: First four rows in BRFSS 2021

As you can see in Table 1, the data we have is numerical. However, some of the values are NA or empty.

Let's have a look at General Health (GENHLTH) data distribution in BRFSS 2021 (after dropping NA and empty strings). We use matplotlib for plottings in this report. Also, General Health Has this definition in the BRFSS report:

Value	Value Label	Frequency
1	Excellent	77,741
2	Very Good	149,112
3	Good	137,938
4	Fair	54,736
5	Poor	18,005
7	Don't Know/Not Sure	788
9	Refused	369
BLANK	Not asked/Missing	4

Table 3: Definition of GENHLTH in BRFSS 2021

From what I understood in the course, I found that it's a good idea to clean the values that are not useful for us. For example in the GENHLTH column values 7, 9 and BLANK are not useful for us, so we can either replace them with a constant value or the median value. After applying these changes, our data distribution over the GENHLTH feature will be like Figure 2:

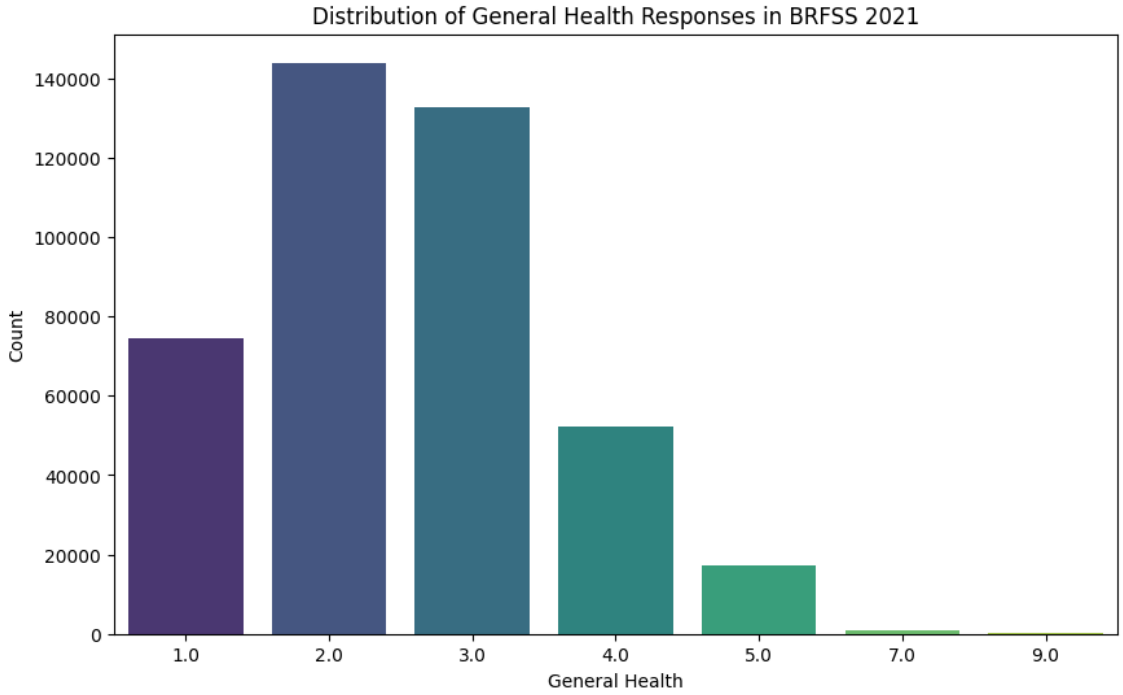
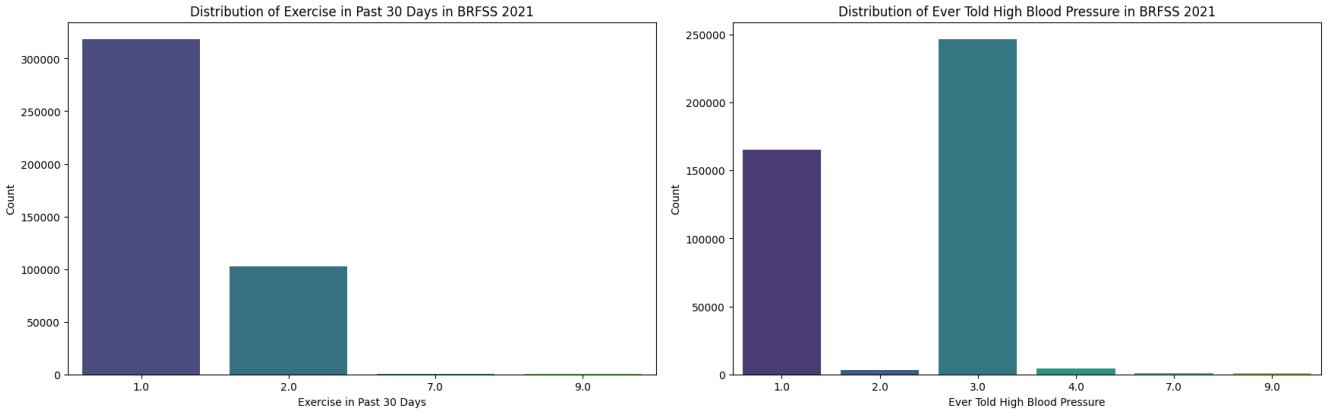


Figure 2: Distribution of General Health Responses in BRFSS 2021

The distribution of responses to the question represented by the feature '**EXERANY2**', namely "Have you participated in any physical activity during the last month?", is as follows. Additionally, we also have the distribution for the feature '**BPHIGH6**', which represents the question "Has the individual ever been informed that they have high blood pressure?".



(a) Had Exercise in Last 30 Days

(b) Ever Told High Blood Pressure

Figure 3: EXERANY2 and BPHIGH6 Responses in BRFSS 2021

Another step we can take is to use heatmaps for visualizing correlations between numeric variables that we like to use for solving our problem. I'll describe later how we can select a list of features efficiently, but for now just assume we have a list of features and we want to find the correlation rate between them::

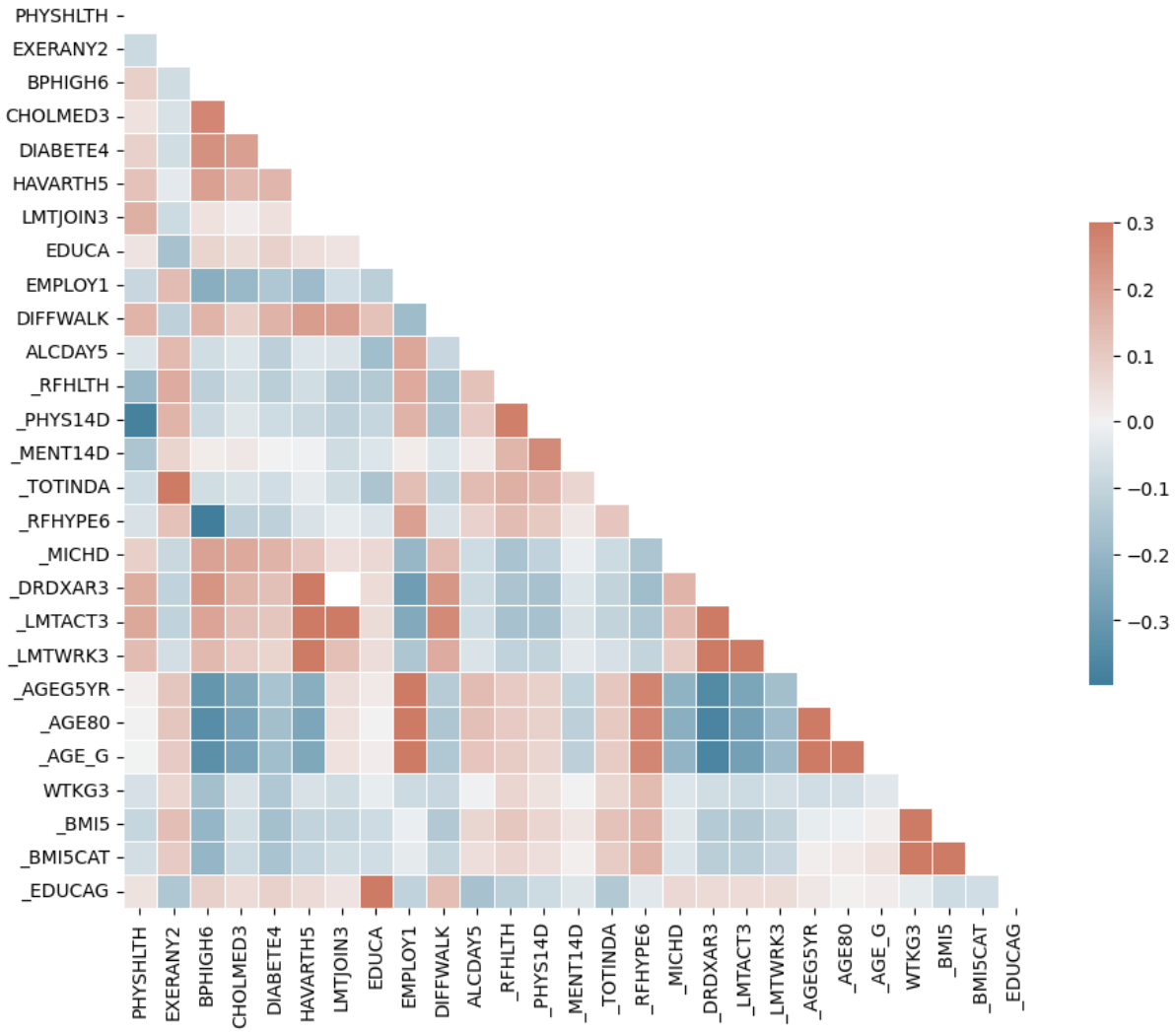


Figure 4: Correlation Rate Between BRFSS Features

Using these figures we can discover more information about the BRFSS dataset.

2.2 Pre-processing Steps

The first Step I did for Pre-processing was to convert the official dataset from XPT to CSV using the R Console. Then, I worked on the dataset to handle the missing data. As I described earlier, I found three general ways for dealing missing data:

1. Remove the samples with missing data
2. Replace the missing data with median value of column
3. Replace the missing data with a constant value

Among these ways, I selected the 3rd one. Because first one makes us to lose a large amount of information. Also, if we use the second one, we may help the data to become more imbalanced. By selecting the last option, it will be easier to balance the dataset using SMOTE technique.

The other step we should take, is to recode some of the columns we want to use. As you can see in Table 3, we have some values like 7 or 9 that are not useful for us. So we convert these values to a constant value like 3.

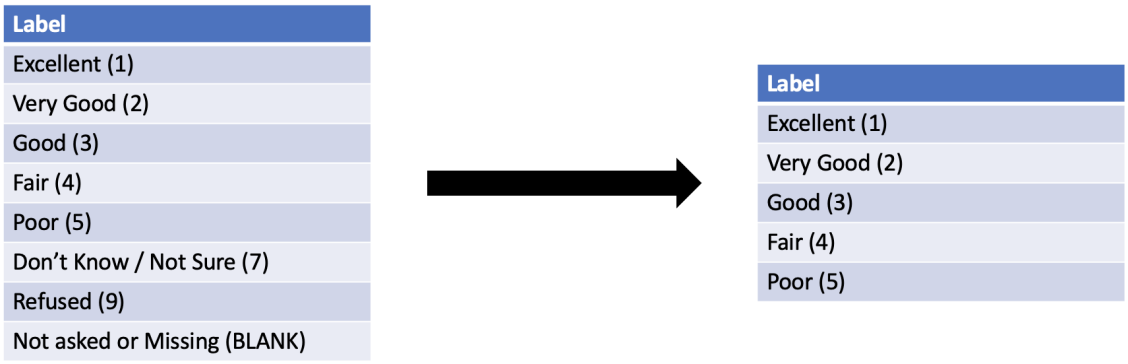


Figure 5: Recoding the GENHLTH column

After cleaning the data, it's time to make it more balanced. The solution I found through my research was to use the SMOTE technique from the imbalance-learn library in Python.

imbalanced-learn, is a Python library that provides methods for dealing with imbalanced datasets. Imbalance in the dataset can cause difficulties for many machine learning algorithms, which often assume balanced class distribution or equal misclassification costs. Imbalanced-learn provides many methods for over-sampling the minority class, under-sampling the majority class, or a combination of both.

The Synthetic Minority Over-sampling Technique, or SMOTE, is one of the over-sampling methods provided by the imbalanced-learn library. The goal of SMOTE is to create new synthetic examples in the dataset that increase the number of minority class examples.

```python
# Define the resampling strategy
over = SMOTE(sampling_strategy='auto')
under = RandomUnderSampler(sampling_strategy='auto')

# Apply the resampling to the training data
X_train_resampled, y_train_resampled = over.fit_resample(X_train, y_train)
X_train_resampled, y_train_resampled = under.fit_resample(X_train_resampled, y_train_resampled)
```

Figure 6: Applying SMOTE on data

3 Methods

3.1 General Design

Since we want to develop a PUI for our project, the architecture design of the project will be different from the classic data-science projects. The image 7 shows a simplified design of the project architecture:

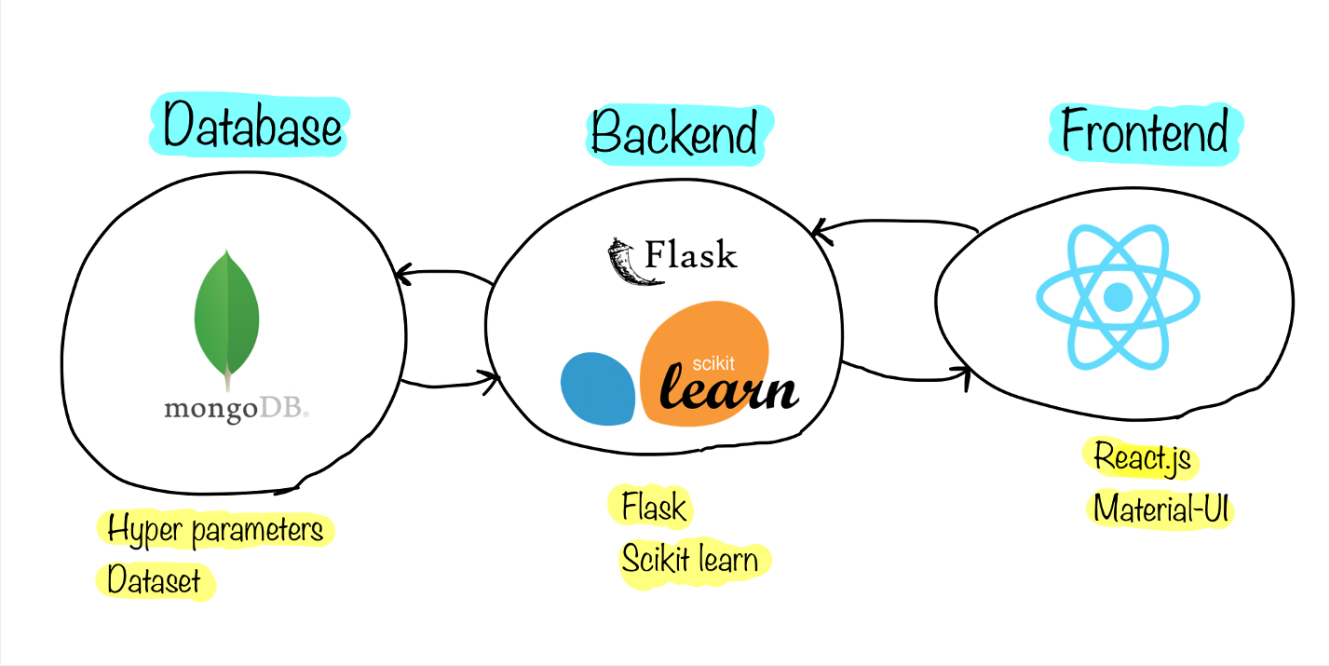


Figure 7: Simplified Architecture of Project

We have three main components here which are:

1. **Database**: loads/stores the dataset and hyper-parameters
2. **Back-End**: contains ML model and Web services
3. **Front-End**: contains PUI and can interact with Back-End

MongoDB is a source-available cross-platform document-oriented database program, classified as a NoSQL database. MongoDB's document model allows for complex nested objects and array storage that align well with object-oriented programming, making it a popular choice for big data and real-time web applications.

Flask is a micro web framework written in Python. Despite its minimalist and modular design, it's very powerful and flexible, allowing developers to build web applications the way they want to, without im-

posing any dependencies or project layout.

React.js, often simply called React, is a popular open-source JavaScript library for building user interfaces, particularly single-page applications where you need a fast, interactive user interface. Developed by Facebook, React allows developers to create large web applications that can change data without reloading the page.

Material-UI is a widely used React UI framework that implements Google's Material Design. It provides a robust set of pre-made React components that follow the Material Design guidelines, helping developers to build consistent, attractive, and user-friendly web applications more quickly and easily.

3.2 Machine-Learning Model

The first thing I did was to store the dataset in the MongoDB collection so we can have access to it more easily rather than reading it from a CSV file. That's because in this project we should think of every functionality as a service, not like a script that starts and finishes. Because of that, we need to have services that can be called by clients and they can repeat the tasks as many times as they want.

After loading the dataset into MongoDB, it's time to select some features that are useful for solving our problem. We don't want to apply our Machine-Learning algorithm on every feature that exists in the dataset. Because it takes a lot of time and also it can make inaccurate results. Therefore, we need a way to select the best features that can be useful to solve our problem. We can do this by different ways.

The simplest approach is to try to add/remove the features to the features list until we reach to the desired accuracy. But in this project, since we deal with a real-world dataset with about half a million samples, it's kind of not achievable. So, I started looking for a way that can handle the feature extraction in a more convenient way. The method I found is called Pearson's Correlation Algorithm which can be used to find the correlation between a set of features based on the threshold we specify for it. For example, I want to find the features that have a correlation with GENHLTH feature of about 30%.

Pearson's correlation is a statistical method that measures the strength and direction of the relationship between two variables. Think of it as a way to quantify how closely two sets of data move together. The output of Pearson's correlation is a number between -1 and 1. A value of 1 means a perfect positive correlation - as one variable increases, so does the other. A value of -1 means a perfect negative correlation - as one variable increases, the other decreases. A value of 0 suggests there's no linear relationship between the variables. So, it's a handy tool to see how things relate to each other in the world of data.

Here is the set of features extracted using Pearson's Correlation Algorithm for our project:

```
'PHYSHLTH', 'EXERANY2', 'BPHIGH6', 'CHOLMED3', 'DIABETE4', 'HAVARTH5',
'LMTJOIN3', 'EDUCA', 'EMPLOY1', 'DIFFWALK', 'ALCDAY5', '_RFHLTH',
'_PHYS14D', '_MENT14D', '_TOTINDA', '_RFHYPE6', '_MICHD', '_DRDXAR3',
'_LMTACT3', '_LMTWRK3', '_AGEG5YR', '_AGE80', '_AGE_G', 'WTKG3',
'_BMI5', '_BMI5CAT', '_EDUCAG'
```

Figure 8: Features With a Correlation of More Than 15% But Less Than 70%

The reason that we don't like features with more than 70% correlation is that those features lead to over-fitting since they kind of explain the same thing that GENHLTH explains. So, if we include those features, we'll get really good but fake results. Because the model is over-fitted in that case and cannot act well when it deals with real-world inputs that have not been seen before.

We use the RandomForestClassifier from scikit-learn for making our ML model. The RandomForestClassifier is like a team of decision trees where each member gives their opinion and the final decision is made based on the majority vote. Each decision tree is trained on a different set of data, and this diversity helps make the team (or "forest") more robust and less prone to mistakes than a single decision tree.

Also, tuning the hyper-parameters for this classifier, made a good difference in the final accuracy we got. The initial test accuracy I got without applying the techniques I describe was about 55% (Precision, Recall and F1-Score). But after applying the techniques, It reached around 80% in average for different classes.

The other technique I used to make sure my dataframe is more balanced was to use Cross-Validation. Cross-validation is a technique used in machine learning to check how well a model will work in the real world. Imagine you have a set of questions (your data) and you know the correct answers (the labels). You could teach (or "train") a model on all of these questions and answers, but then you wouldn't have a good way to check how well it has learned. So, instead, you hold back some of the questions and answers (a "validation" set). You train the model on the rest, and then test it on the held-back questions. If the model does well on these, you can be more confident it has really learned effectively. In cross-validation, you do this several times with different parts of your data, to make sure your model does well on different kinds of questions and not just a particular subset it has seen before.

I separated the dataset, like this:

1. **Training: 90%**
2. **Evaluation: 5%**
3. **Testing: 5%**

*** Note: all of these percentages can be changed using the UI. So, you don't need to change these values from the source code.**

3.3 Back-End

As I mentioned, our Back-End software is developed by Flask. The back-End is responsible for many important parts of this project. The first important service is that it makes functionalities available for the Front-End. The second important service is that it can interact with the Database.

For running this project, you need to connect the back-End to a MongoDB database using a URI. This MongoDB database can be on your local machine or on a remote server. It's important to note that you should modify the source code to successfully connect to the database. Also, you have to create a database called **brfss_project** and inside it you should create these collections:

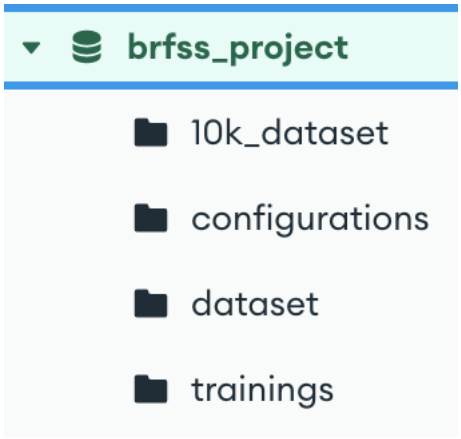


Figure 9: MongoDB Required Collections

For running the project, I suggest to first create a virtual environment using the **virtualenv** package and then install the packages using **pip install -r requirements.txt** . By entering command **python app.run** in the backend directory, the back-End server lunch on http://localhost:5000:

```
(app_env) sh-3.2$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
 deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 129-688-770
```

Figure 10: Running Back-End on Local Machine

My assumption is that the BRFSS 2021 is stored in the MongoDB database. I can't upload the database dump because of its size, but I've included the code that can store the dataset in MongoDB in the back-end source code. You should have BRFSS 2021 CSV file in a folder called raw_dataset inside your back-End folder, so MongoDB can load it into its collection. **Please refer to load_csv_to_mongo.py file for doing this part.**

3.4 Front-End

For running the Front-End side, you need to have Node.js and Node Package Manager (NPM) installed on your machine. Because React.js applications rely on Node.js.

After that, you can open a terminal and change the directory to the frontend folder and install the project dependencies using **npm install –force**. After execution of this command finished, you are ready to run the application. For running the application enter **npm start** in the command line.

*** Note: First you should run the database, then the Back-End and after that you are ready to run the Front-End**

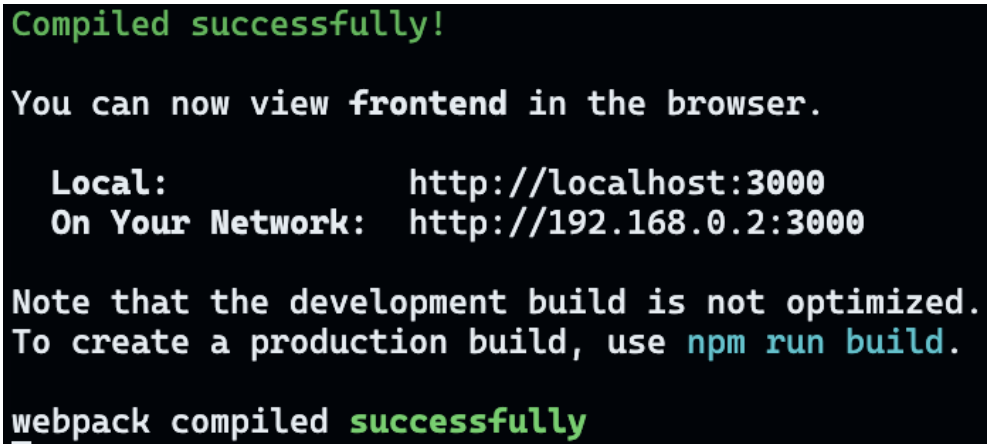


Figure 11: Running Front-End(PUI) on Local Machine

After running this command, a new will open on your default browser:

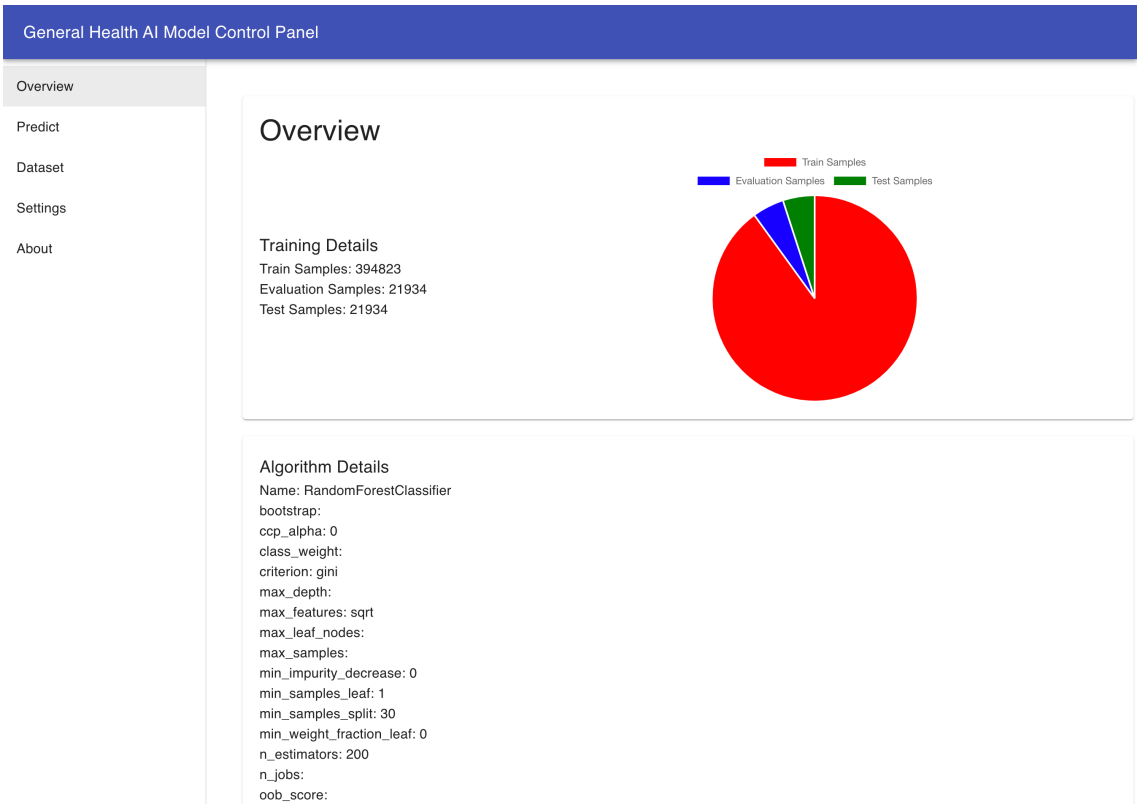


Figure 12: General view of PUI

As you can see, on the left side of the PUI we have list of sections and by clicking on them we can navigate to those sections. Here is a brief description of what each section does:

1. **Overview**: Gives general information about the count of data, split rates, ML model used for training, hyper-parameters used for tuning the ML model and etc.
2. **Predict**: Gets model input as a form from the machine teacher and sends them to the model, shows the model prediction at the bottom of the page.
3. **Dataset**: A visual look of the whole dataset, which allows machine teacher to look at the data.

4. **Settings**: Allows the machine teacher to modify the hyper-parameters of the system easily.
5. **About**: Gives a brief information about the project.

4 Results & Findings

4.1 Evaluation

I used Precision, F1-score and Recall as evaluation metrics.

Here is the **best** result for test accuracy after repeating the training with different tuning:

Class	Precision	Recall	F1-Score
1	0.69	0.64	0.65
2	0.76	0.76	0.76
3	0.93	0.94	0.93
4	0.85	0.88	0.81
5	0.81	0.79	0.80

Table 4: My Model Test Accuracy

From these results, it seems my model is performing best in class 3 and class 4, while it's struggling more with classes 1, 2, and 5. This could potentially indicate that there still exists an imbalance in the dataset or that my model is not complex enough to capture the distinctions between certain classes.

4.2 Baseline

For the baseline model, I used **Multinomial Logistic Regression** model. which is an extension of logistic regression that adds native support for multi-class classification problems. It changes the loss function to cross-entropy loss and predict probability distribution to a multinomial probability distribution to support multi-class classification problems.

Here is the best result for test accuracy using my baseline:

Class	Precision	Recall	F1-Score
1	0.53	0.49	0.51
2	0.62	0.62	0.62
3	0.73	0.77	0.85
4	0.80	0.83	0.84
5	0.68	0.62	0.64

Table 5: Baseline Model Test Accuracy

In Figure 13 you can see the precision metric comparison between my model (Random-Forest classifier) vs the baseline model:

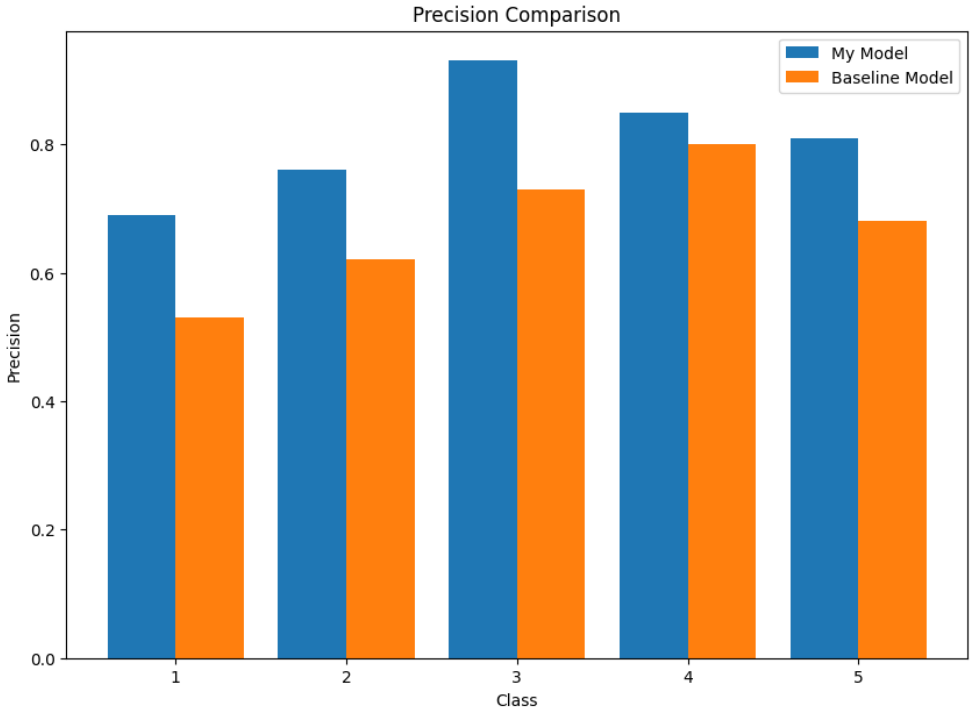


Figure 13: Precision Metric Comparison Between My Model and Baseline Model

In Figure 14 you can see the recall metric comparison between my model (Random-Forest classifier) vs the baseline model:

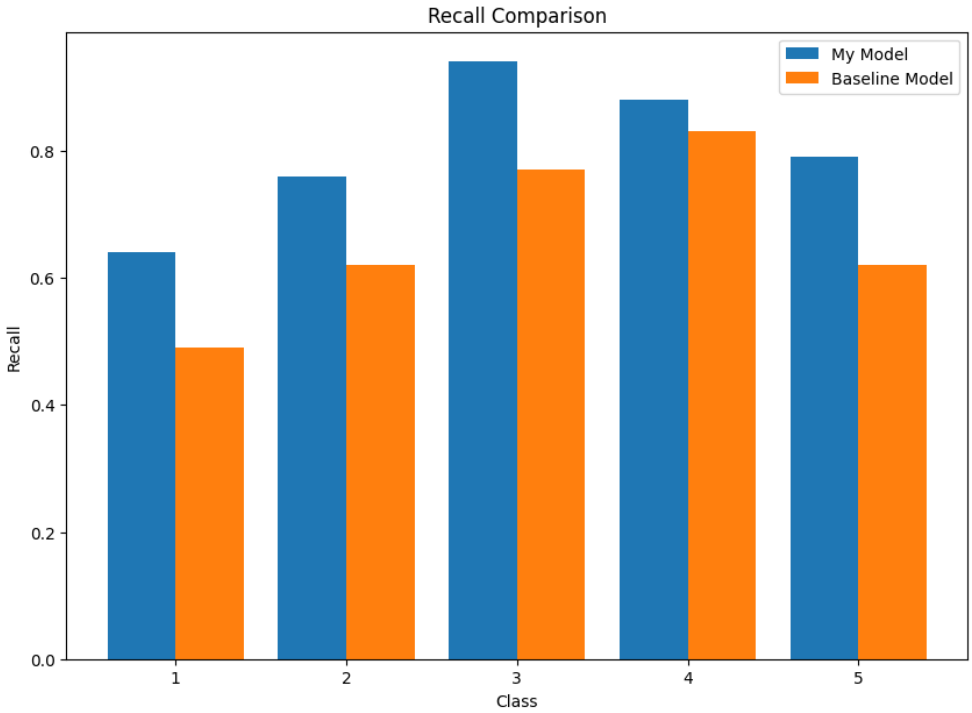


Figure 14: Recall Metric Comparison Between My Model and Baseline Model

In Figure 15 you can see the f1-score metric comparison between my model (Random-Forest classifier) vs the baseline model:

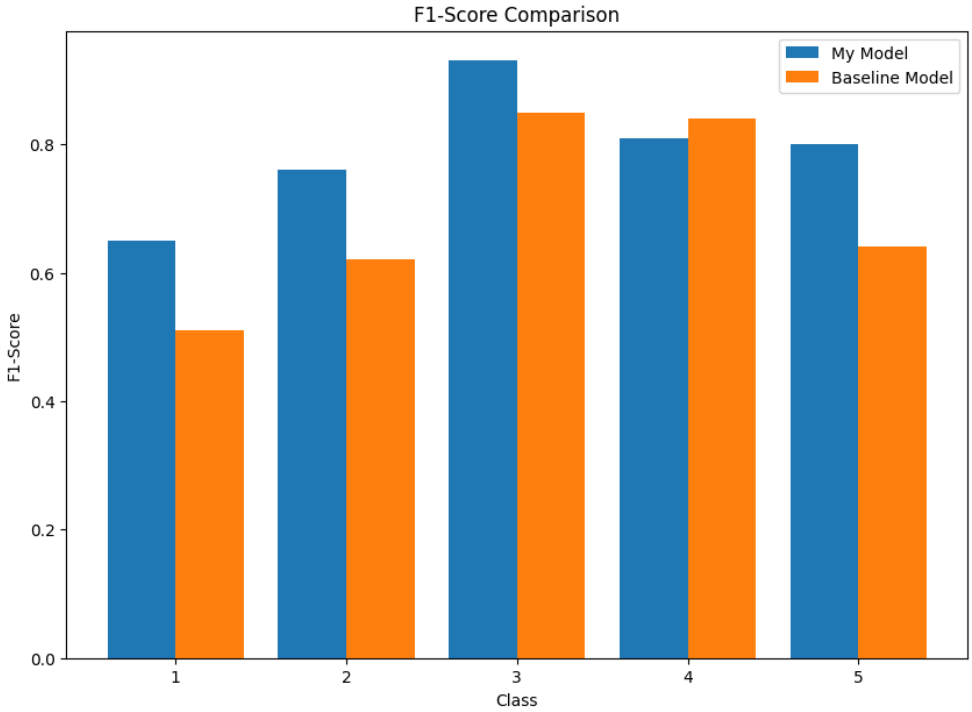


Figure 15: F1-Score Metric Comparison Between My Model and Baseline Model

5 Discussion & Conclusion

During this project, I learned that creating a stable human-centred data-science software solution has lots of challenges. being stable is a really important factor that is hard to achieve without having enough knowledge. Also, cleaning the data and hyper-parameter tuning plays a really important role on the model behavior. Also, from the experience I gained, I think creating a PUI can be quite challenging because of the size of data is big and UIs almost run on local machines which don't have very strong computing resources.

5.1 Advantages

5.1.1 Configurable Through Online PUI

Creating an online user interface (UI) for managing your machine learning (ML) model, particularly one that's configurable, has numerous benefits. This kind of "Configurable Through Online PUI" allows you to adjust, control, and monitor your ML model's behavior in a user-friendly manner.

5.1.2 Outperforms the baseline

Our model outperforms the baseline by a good span which means it has a better accuracy. However, this can't be necessarily true because it may be because of data-imbalance which I described in the report completely.

5.2 Limitations

5.2.1 Data Imbalance

If a model is trained on this imbalanced data, it might perform well on the majority class but poorly on the minority class. An imbalance in training data can lead to biased and inaccurate predictions in machine learning models.

5.2.2 Training Through PUI

Right now, training the model through the PUI is not possible because it takes about 20 minutes and leads to an HTTP request timeout. A better approach was to use real-time network protocols that don't have this limitation.

5.3 Error analysis Suggestions for future improvements

We can start by looking at where the model has performed poorly and trying to understand why it's failing. Then we can categorize the errors our model makes (e.g., false positives, false negatives) and then deep dive into some of these examples to see if we can find common features or patterns.

1. Check for errors, outliers, or biases in the data that could be affecting the model's performance.
2. Trying different models since different models have different strengths and can yield better results depending on the specific task.
3. Using techniques like grid search or randomized search to find the optimal hyperparameters for the model.
4. Handling class imbalance in a more efficient way.
5. Using ensemble methods to combine the results of different models in order to gain better accuracy.

5.4 Takeaways

1. Hyperparameter tuning can make a lot of difference in model accuracy.
2. Designing UI for ML model monitoring can be challenging due to large amount of data but also time consumption.
3. Documenting the dataset can help machine trainers to work easier with the system

5.5 Future Works

I think **trying Neural Network models** can be useful. They may lead to better results however, they usually take more time for training than classical ML algorithms.

In addition, if can **port the source code to be GPU-compatible**, it can make a good performance improvement not on model accuracy but at least for the training duration.

Also, adding **a work offloading feature to make PUI more reliable and able to handle model training** can be a good improvement on this project.

Finally, I think the most important future work can be **making sure that our data pre-processing is correct and it's in a good position so that we can build a reliable system on it**.

6 Showcase

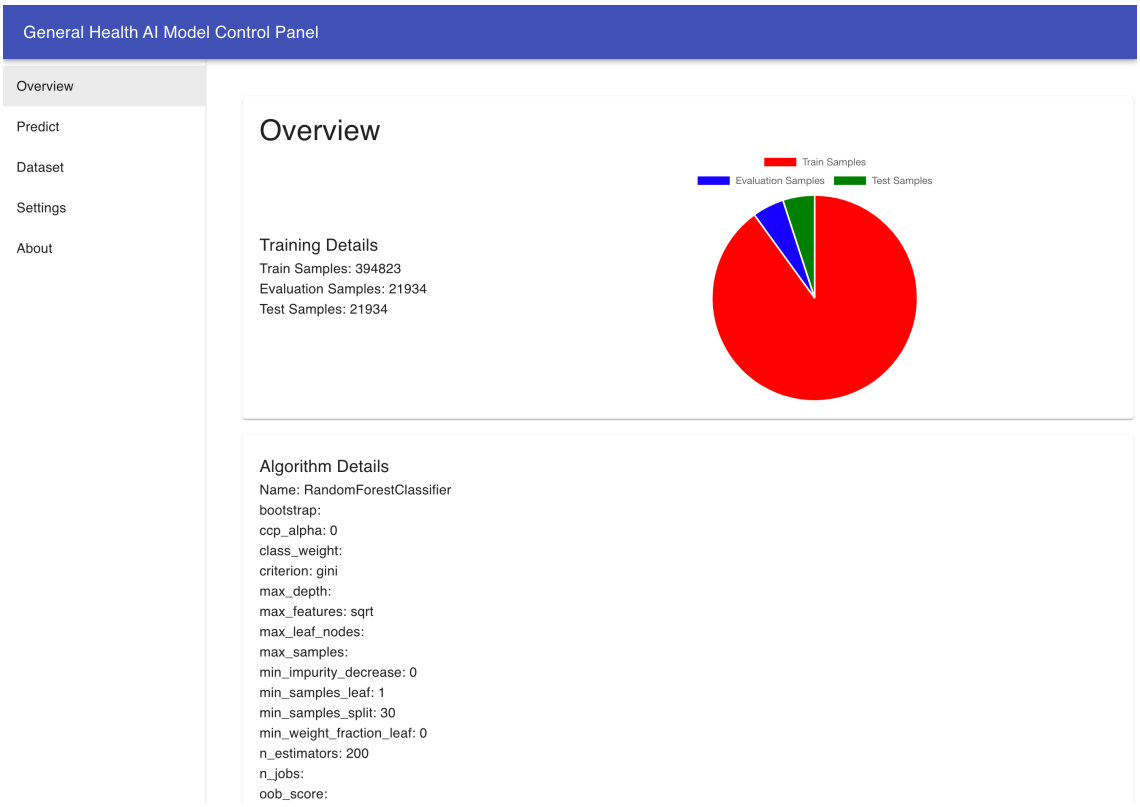


Figure 16: Overview Section of PUI

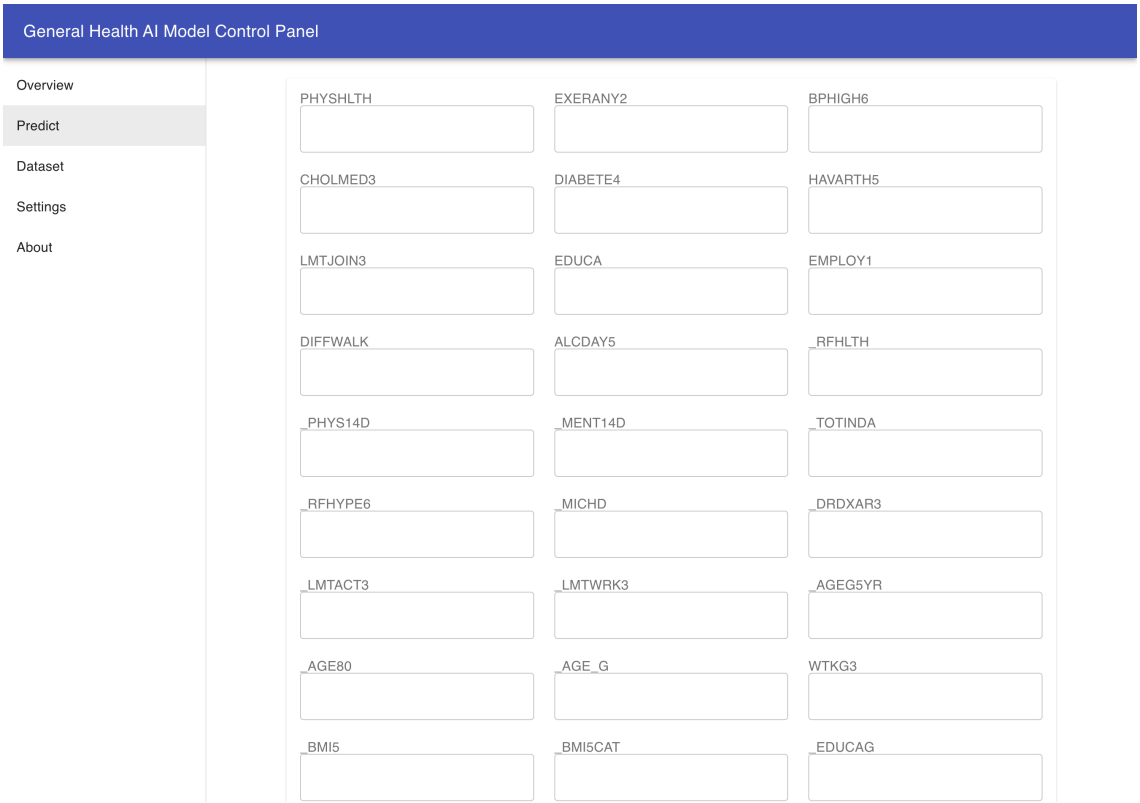


Figure 17: Predict Section of PUI

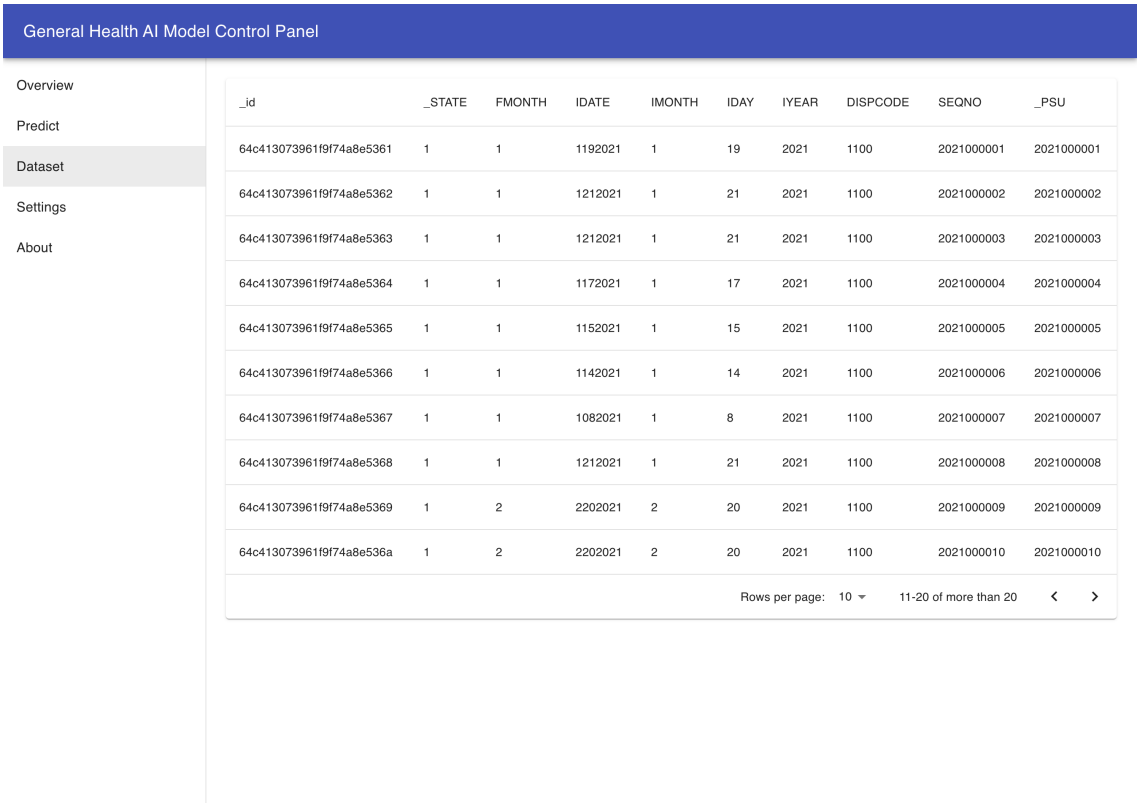


Figure 18: Dataset Section of PUI

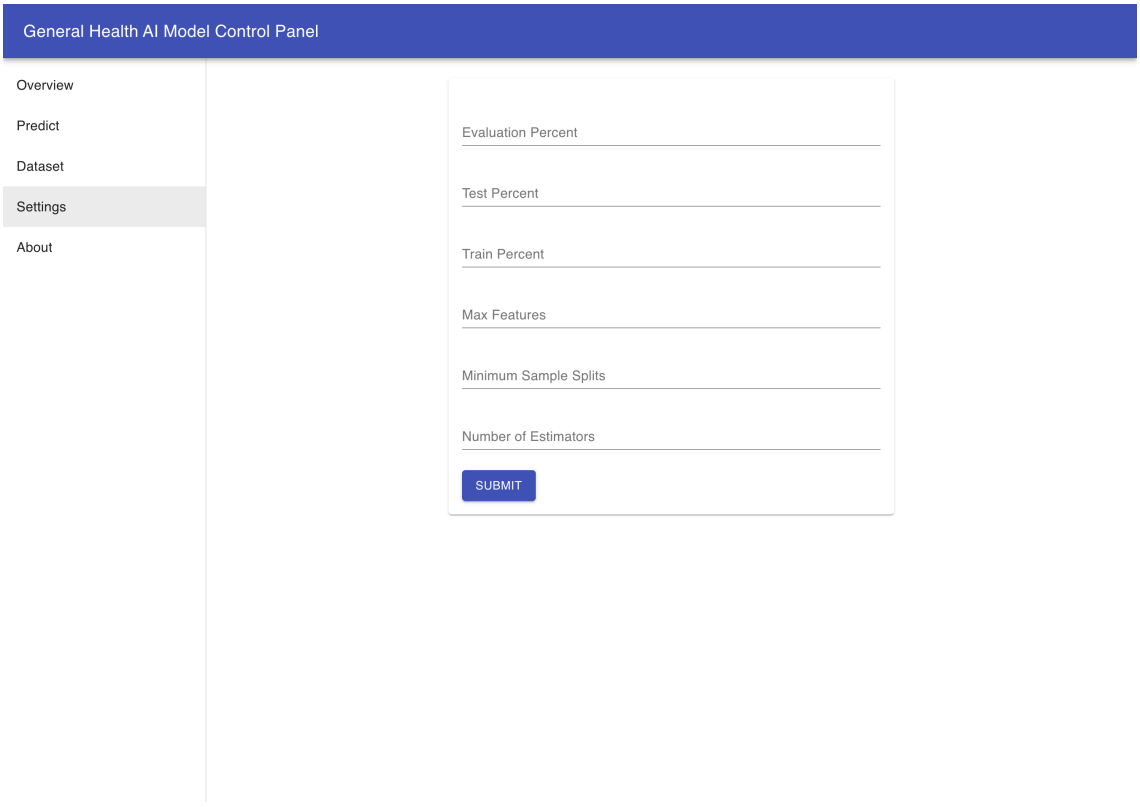


Figure 19: Settings Section of PUI

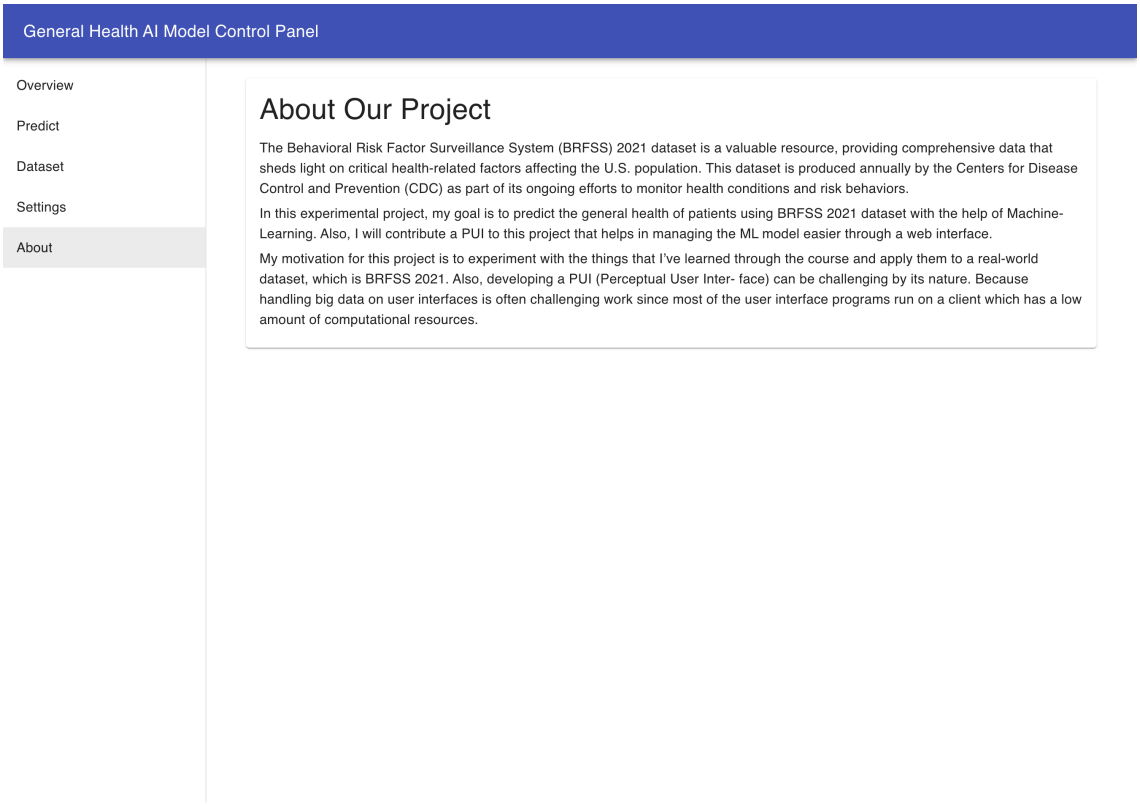


Figure 20: About Section of PUI