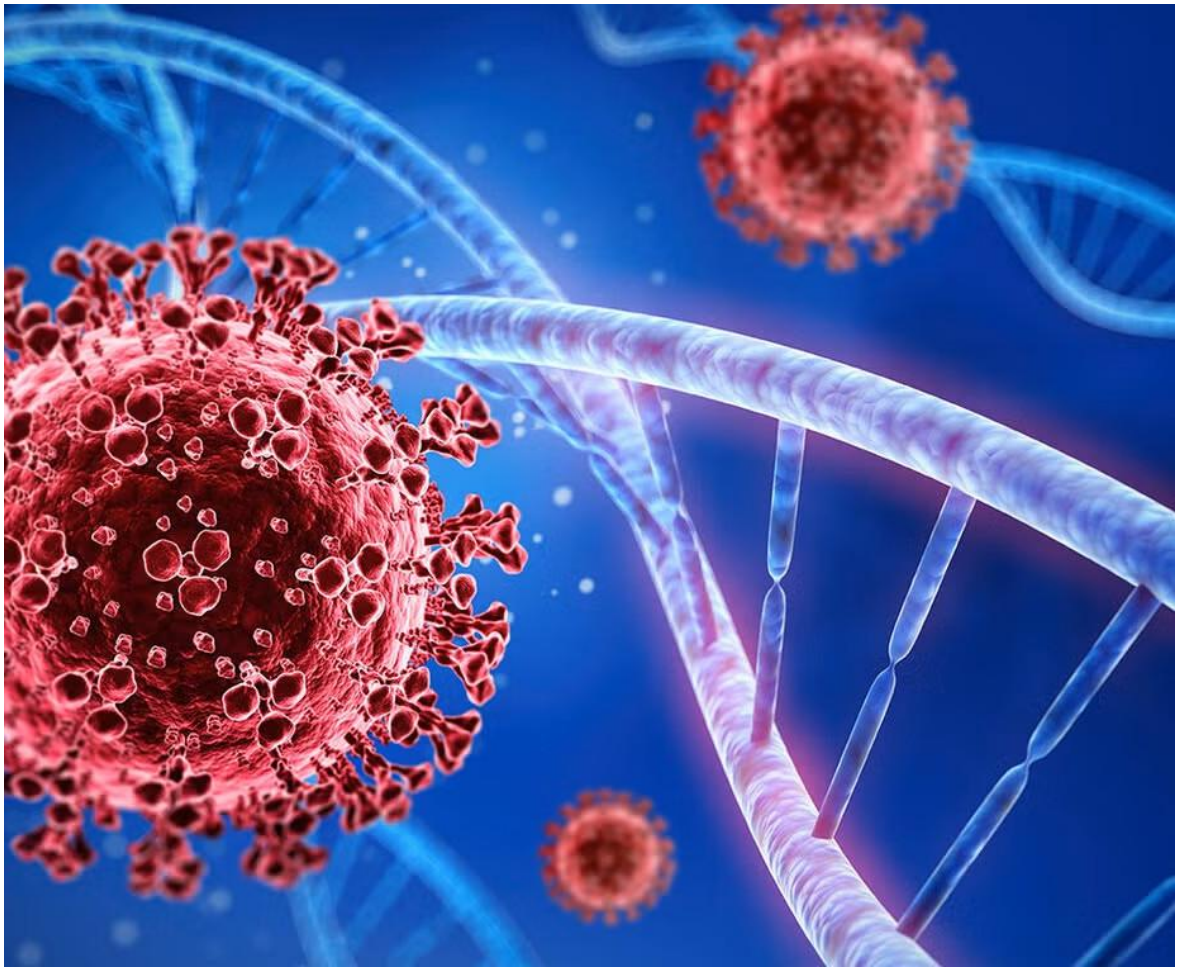

Voice Analysis for the Detection of Respiratory Diseases

Olga Kardampiki *(p2822414)*
Christiana Sofantzi *(p2822432)*

Machine Learning and Content Analytics

Table of Contents

Data Collection

In our study we used an open dataset, which came from GitHub[1], and aimed at developing non-invasive tools for the detection of COVID-19 and other respiratory illnesses. Dataset includes medical and demographic information combined with audio samples, from April 2020 to February 2022. Each entry corresponds to a participant and contains:

- Demographic details, such as age, gender, and geographic location
- Symptoms and pre-existing conditions, such as asthma, hypertension, diabetes, pneumonia, chronic lung disease etc.
- COVID test status, which can be positive, negative, or not taken
- Health status and self-reported health status, like healthy, COVID-positive with mild/moderate symptoms, or other respiratory illness
- Audio recordings in nine categories: deep/shallow breathing, heavy/shallow cough, sustained vowels (a, e, o), and speech (counting at normal and fast pace).

The gathering of data executed through an online platform, where participants voluntarily contributed their health information and audio recordings. To ensure quality, the audio files were manually reviewed and annotated on a quality scale. The careful collection process enabled the creation of a dataset that combines clinical, demographic, and audio evidence, providing a rich foundation for machine learning approaches to respiratory disease detection.

**For personal data protection, the dataset is anonymized and does not contain information that can identify individuals.*

[1] https://github.com/iiscleap/Coswara-Data

Data Overview

The dataset includes records from 2,635 participants, from whom their status was covid-positive, covid-negative or recovered subjects, but in this study, we take it a step further and search for the ones who are healthy or sick generally, by including other diseases.

Each record combines structured metadata with audio features. Specifically:

- Symptoms and pre-existing conditions encoded as binary features (e.g., asthma, cough, fever, fatigue, diabetes, pneumonia).
- COVID-19 test status is represented through one-hot encoded variables (test_status_Positive, test_status_Negative, test_status_na).
- Audio features extracted from the recordings, including MFCCs, spectral centroid, rolloff, bandwidth, ZCR, and duration across all nine sound categories.
- Health labels indicating whether an individual is healthy, has COVID virus, or suffers from another respiratory condition (like asthma, pneumonia, chronic lung disease, or other).

The dataset is imbalanced, with the larger proportion of participants to be healthy and COVID-negative compared to the COVID-positives and other respiratory disease cases. Nevertheless, it offers a diverse representation of both metadata and acoustic signatures, making it suitable for training and evaluating tasks of binary classification (healthy vs. sick) and multi-class disease classification.

Data Processing

Starting with the raw dataset, several processing steps were applied to ensure consistency, and to create a feature-rich input for training and evaluation of machine learning models.

First, we imported the CSV files that were contained in several date-named folders and merged them into a single DataFrame to build the dataset. From this DataFrame, only the relevant medical, demographic, and diagnostic attributes were kept, while non-essential fields were removed. The dataset was examined for anomalies, missing values, and inconsistent encodings. Some of the changes that occurred are:

- Gender was encoded into categories *(male = 0, female = 1, other = 2)*
- Boolean fields were converted into numeric values *(True/Yes = 1, False/No = 0, Missing/Unknown = -1)*
- Variables such as vaccination and testType were standardized into consistent labels or numeric values. More specifically:
  For vacc we have *y = 2 (both doses), p = 1 (one dose), n = 0 (no doses), missing = -1*
  For testType we have *False = Rapid* [2]*, rat = Rapid, rtpcr = PCR, missing = Not taken Test*
- Missing dates (ctDate, test_date) were replaced with explicit labels *("No ctScan", "Not Available")*
- test_status was normalized into four categories: *Positive / Negative / Not taken Test / Unknown*.

Secondly, going through the audio files, we extracted the audios and linked them to each participant based on the ID. Having done this matching step, we calculated the duration of each signal. Then, we excluded from the dataset the rows without all the required recordings, based on their durations. To be more specific, we had 2 rows that did not cover the required durations. After that, for each audio file *(breathing, cough, vowel sounds, and speech)*, we extracted a few simple acoustic features that are widely used in speech and sound analysis. The **MFCCs** *(Mel-Frequency Cepstral Coefficients)* describe the overall shape of the sound and are commonly used because they reflect how people perceive sound. The **Spectral Centroid** shows whether the sound energy is more in the high or low frequencies, while the **Spectral Rolloff** points to the frequency where most of the energy is concentrated. The **Spectral Bandwidth** tells us how wide the spread of the spectrum is, and the **ZCR** *(Zero Crossing Rate)* measures how often the waveform changes sign, which usually indicates how noisy or irregular the sound is. This process yielded a set of ~90 numerical audio features per participant, representing both spectral and temporal aspects of the respiratory sounds.

[2] This one is not clear at first, but it was chosen after the examination and the understanding of other parameters of the dataset.

For better data processing and execution, we created two new variables: the first one, **symptom_col**, consists of all the symptoms participants can report having, and the second, **symptom_sick**, contains those symptoms that indicate someone is ill. The variable *symptom_sick* is the one that will be used to examine if the participant is sick and what kind of disease he/she has.

Based on the type of columns, categorical attributes *(covid_status, smoker, test_status, testType, l_c)* were transformed into One-Hot Encoded dummy variables so they could be used as numerical features.

Having done the above steps, into a new DataFrame, named **df_encoded**, which contains all the columns, audios and the dummies variables, we created our target variable, **is_sick**. The binary target *is_sick* shows if someone is healthy *(is_sick = 0)* or is ill *(is_sick = 1)*. For someone to be sick, he/she should have at least one *symptom_sick* value equal 1 (True/Yes) or a positive COVID test.

Below, there is a plot that shows the frequency of the symptom:

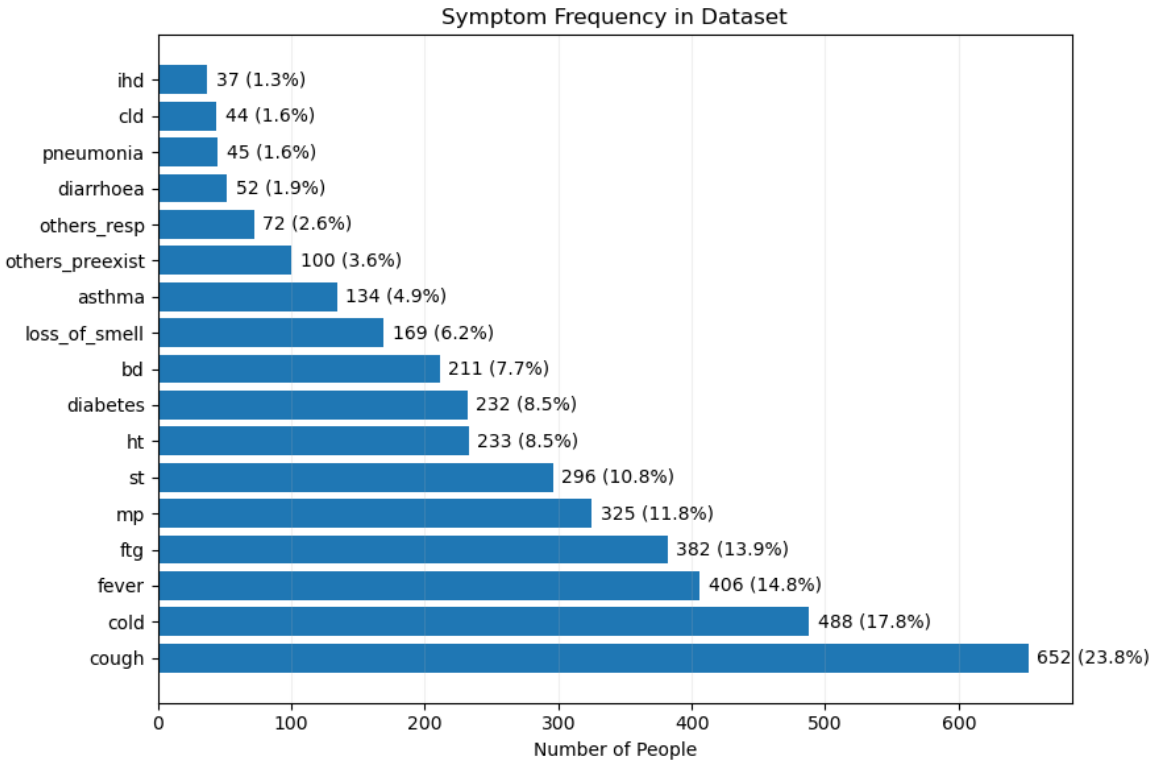

Figure 1 : Frequency of Symptoms in Dataset

In the new DataFrame, *df_encoded*, we also included the audio files. To have clean data, and to be sure we will work "correctly", we checked that we only have participants with both symptoms documented and audio recorded. In our case, all IDs have at least one valid audio to combine with their symptoms.

As the final step in this process, the features *(X)* were created for the CSV **(X_csv)**, the audio files **(X_audio)**, and the combination of these two **(X_train_combined** & **X_test_combined)**. Also, we created the label **(y)** containing the target variable *is_sick*, which will be used later in the execution of the models.

Methodology

Moving on to the methodology we followed in our study, we created three different classification models to evaluate separately the information provided by each feature group, the CSV with symptoms and the audio, as well as their combination.

First, we split the dataset into training and test sets, keeping the same IDs across all feature groups, CSV and audio, to ensure consistency in the results to compare the same participants. The audio features were normalized using *StandardScaler*, while the CSV features, symptoms, were used as they were after preprocessing.

About the models, firstly, we created the CSV-only Model, where we used only the symptoms and demographic characteristics that were present in the CSV files. Our goal was to check how well health data, such as fever, cough, and other symptoms, can predict if a participant is sick or not.

The model was trained in Random Forest with 200 trees and a balanced class. The results of the test's set evaluation are below:

Accuracy:	0.952
F1 - Score:	0.942
ROC - AUC:	0.945

Table 1 : Accuracy, F1-Score, ROC-AUC of CSV-only Model

Classification Report:				
	Precision	**Recall**	**F1 - Score**	**Support**
0	0.92	1.00	0.96	468
1	1.00	0.89	0.94	356
Accuracy			0.95	824
Macro Avg	0.96	0.95	0.95	824
Weighted Avg	0.96	0.95	0.95	824

Table 2: Classification Report of CSV-only Model

The results of the CSV-only model, *Table 1 & Table 2*, show that using only symptoms and demographic information already gives strong performance, with an overall accuracy of 95%. The model performs very well in identifying healthy participants, with a recall of 1.00, meaning it does not misclassify them as sick. However, for sick cases the recall drops to 0.89, indicating that some sick participants are incorrectly predicted as healthy.

As we wanted to examine all cases, we checked the false negatives and what symptoms they have. In total, we observed 39 false negatives out of 824 test samples. The analysis of their reported symptoms, *Table 3* : Symptoms of False Negatives on CSV-only Model, shows that the most frequent ones were cough, cold, and fever. This indicates that the model sometimes failed to recognize sick cases, even when participants had typical symptoms.

Symptom	Count	Fraction
cough	16	0.41
cold	13	0.33
fever	13	0.33
ftg	8	0.20
st	7	0.18
bd	7	0.18
mp	6	0.15
ht	3	0.08
other_resp	3	0.08
diarrhoea	3	0.08
loss_of_smell	3	0.08
other_preexist	2	0.05
diabetes	2	0.05
ihd	1	0.02
pneumonia	1	0.02

Table 3 : Symptoms of False Negatives on CSV-only Model

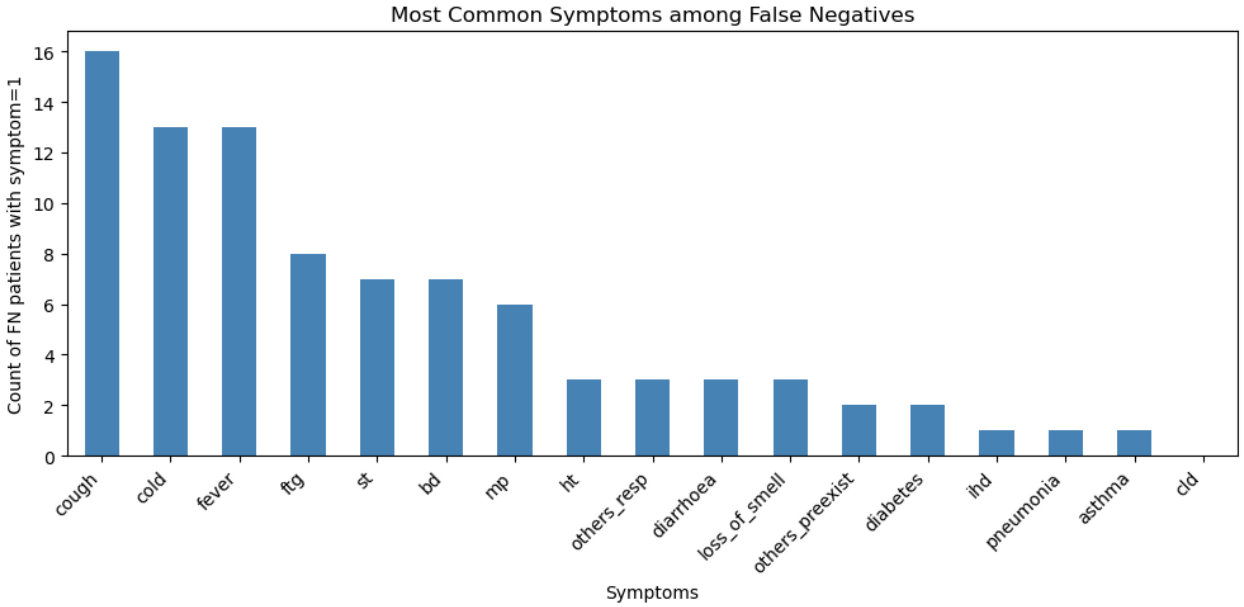

Figure 2: Barplot of False Negatives Symptoms on CSV-only Model

Next, we created the Audio-only Model, where we used only the audio data, specifically the features we extracted on the previous steps.

As before, we trained a Random Forest model, with also 200 trees and a balanced class, to see if the audio signals alone are sufficient for accurate diagnosis.

Accuracy:	0.683
F1 - Score:	0.549
ROC - AUC:	0.740

Table 4: Accuracy, F1-Score, ROC-AUC of Audio-only Model

Classification Report:				
	Precision	**Recall**	**F1 - Score**	**Support**
0	0.67	0.86	0.76	468
1	0.71	0.45	0.55	356
Accuracy			0.68	824
Macro Avg	0.69	0.65	0.65	824
Weighted Avg	0.69	0.68	0.67	824

Table 5: Classification Report of Audio-only Model

The Audio-only Model did not perform as well as the CSV-only Model. Its accuracy reached 68%, with an F1-score of 0.55 and ROC-AUC of 0.74, *Table 4 & Table 5*. Looking at the classification report, we see that the model is better at identifying healthy samples, with a recall of 0.86, but struggles with sick cases, with a recall of 0.45. This means that many sick patients were misclassified as healthy. Overall, the audio features alone give us a few useful information, but they are not strong enough to reliably separate healthy from sick cases. This shows why combining audio with symptoms and demographic data is necessary for better performance.

The Audio-only Model produced 197 false negatives out of 824 test samples (~24% of the data), meaning that these patients were incorrectly classified as healthy, with cough, cold, and fever being the most common symptoms, *Table 6*. This indicates that even when such symptoms were present, the model based only on audio features was not always able to capture the underlying health condition.

Symptom	Count	Fraction
cough	67	0.34
cold	56	0.28
fever	46	0.23
ftg	36	0.18
mp	35	0.17
st	30	0.15
ht	25	0.13
loss_of_smell	19	0.09
bd	17	0.08
asthma	14	0.07

Table 6: Symptoms of False Negatives on Audio-only Model

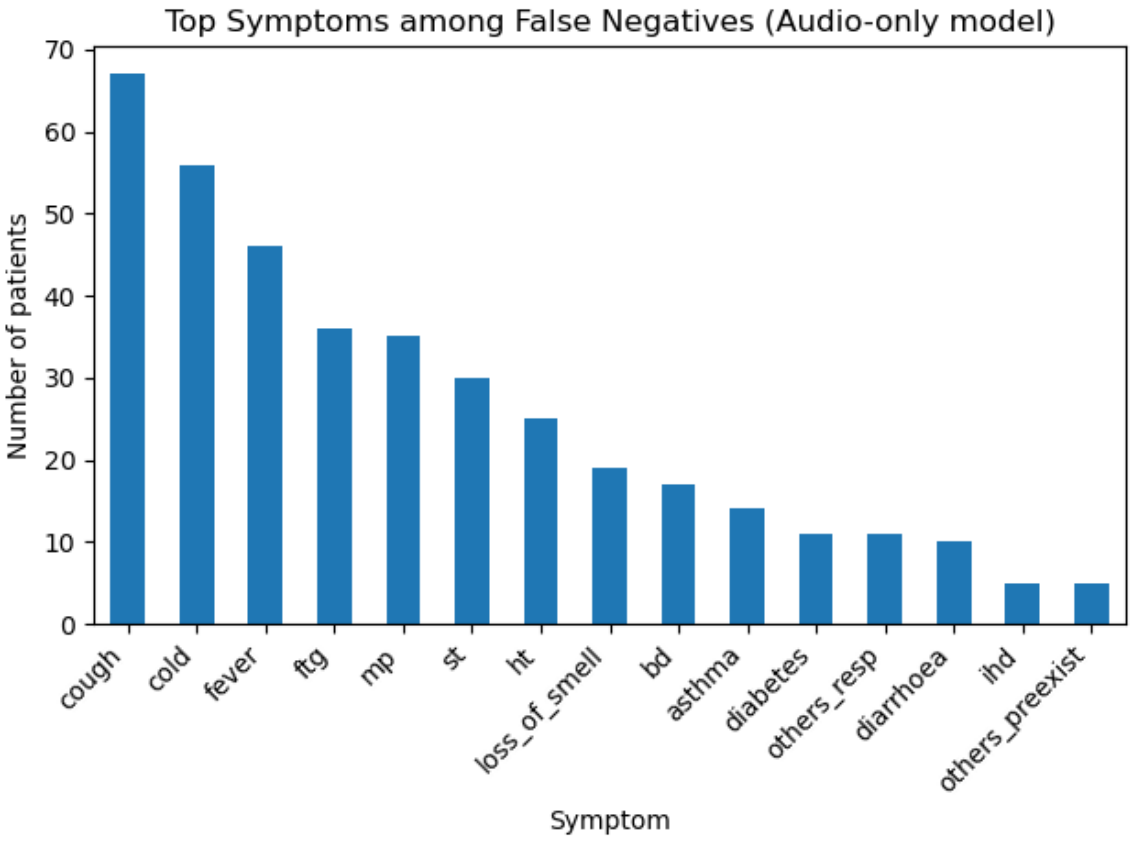

Figure 3: Barplot of False Negatives Symptoms on Audio-only Model

About the third model, we combined the symptoms and the metadata with audio features. So, we created a unified representation that contains both sources of information and we trained two different models, as we first wanted to identify patients based on the data combination, and then to determine which disease they had.

In the first step, we trained a Random Forest model with 300 trees and a balanced class, where the results show that:

Stage 1 Report (Binary Classification):				
	Precision	**Recall**	**F1 - Score**	**Support**
healthy	0.93	1.00	0.96	468
sick	0.99	0.90	0.94	356
Accuracy			0.95	824
Macro Avg	0.96	0.95	0.95	824
Weighted Avg	0.96	0.95	0.95	824

Table 7: Stage–1 Report for Binary Classification

In Stage–1, the model performed very well with an overall accuracy of 95%, *Table 7*. For the healthy class, recall reached 100%, meaning the model detected all healthy cases correctly, though precision was slightly lower at 0.93. For the sick class, precision was very high, 0.99, but recall dropped to 0.90, showing that a few sick cases were missed. Overall, the results show that Stage–1 is very strong and provides a reliable first filter before moving on to more detailed classification.

Next, we created labels for the main diseases that we have in the dataset and checked how many participants had declared each one:

Classes: *healthy, covid, asthma, pneumonia, cold, resp_other*	
Distribution (of all the datasets)**:**	
healthy	1752
covid	681
cold	174
asthma	108
pneumonia	15
resp_other	14

Table 8 : Classes for Diagnosed Diseases

To compete with the problem of imbalance between healthy and sick individuals, we applied sample weighting, giving more weight to the examples of the sick individuals *(parameter α=2)*. In this way, we enhanced the model's ability to correctly identify patient cases and reduce False Negatives.

Additionally, instead of using the classic threshold of 0.5 for prediction, we searched for the optimal threshold and selected the one that maximizes the F1-Score for the **sick** category, which is *0.450*.

Stage 1 Report (Binary - Tuned Threshold):				
	Precision	**Recall**	**F1 - Score**	**Support**
healthy	0.93	0.98	0.96	468
sick	0.97	0.91	0.94	356
Accuracy			0.95	824
Macro Avg	0.95	0.94	0.95	824
Weighted Avg	0.95	0.95	0.95	824

Table 9 : Stage–1 Report for Binary Classification with Tuned Threshold

The final Stage–1 binary classifier achieved very strong performance, with an overall accuracy of 95%, *Table 9*. Precision and recall were both high for the two classes, with healthy reaching 0.93 precision and 0.98 recall, and sick reaching 0.97 precision and 0.91 recall. The F1-Scores are also balanced, 0.96 for health and 0.94 for sickness, showing that the model performs well in both detecting sick patients and avoiding false alarms. This tuned threshold helped balance sensitivity and precision, resulting in a reliable first-stage screening step before moving on to multiclass classification.

Stage 1 Confusion Matrix:
[[457 11] [32 324]]

Figure 4 : Confusion Matrix of Stage–1

The confusion matrix of Stage–1, *Figure 4* : Confusion Matrix of Stage–1, shows that the model correctly classified most of the samples, with 457 true healthy and 324 true sick cases. Only 11 healthy cases were misclassified as sick *(false positives)*, and 32 sick cases were misclassified as healthy *(false negatives)*. This confirms the high accuracy observed in the classification report.

In the second stage, we applied the two-stage methodology. Initially, each sample goes through Stage–1, where it is classified as healthy or sick. Samples classified as sick are forwarded to Stage–2 for further categorization into a specific disease.

Two-stage Overall Report *(Stage–1 threshold=0.45)*:				
	Precision	**Recall**	**F1 - Score**	**Support**
healthy	0.93	0.88	0.90	526
covid	0.57	0.84	0.68	219
asthma	0.00	0.00	0.00	24
pneumonia	0.00	0.00	0.00	5
cold	0.00	0.00	0.00	48
resp_other	0.00	0.00	0.00	2
Accuracy			0.79	824
Macro Avg	0.25	0.29	0.26	824
Weighted Avg	0.74	0.79	0.76	824

Table 10 : Report of Stage–2

The two-stage model achieved an overall accuracy of 0.79, *Table 10*, but the performance across classes is very unbalanced. While the model performs reasonably well in detecting covid cases (recall 0.84), it fails to correctly classify the smaller classes like asthma, pneumonia, cold, and other respiratory illnesses, all with F1-Scores to 0. This suggests that the model is biased toward the majority classes (healthy and covid) and struggles with underrepresented categories.

Two-stage confusion matrix

```
[[ 464   62   0   0   0   0 ]
[  34  185   0   0   0   0 ]
[   3   21   0   0   0   0 ]
[   0    5   0   0   0   0 ]
[   0   48   0   0   0   0 ]
[   0    2   0   0   0   0 ]]
```

Figure 5 : Confusion Matrix of Stage–2

The confusion matrix confirms these findings, *Figure 5*. Most predictions are concentrated in the healthy and covid classes, while the smaller classes are almost completely misclassified into these categories. For example, asthma and cold cases are mostly predicted as covid or healthy, showing that the model cannot separate minority diseases well. This highlights the class imbalance problem.

Also, we performed an error analysis to better understand where the model is “struggling”, and we identified the False Positives (11) and the False Negatives (32).

The most common symptoms of the False Negatives, for example, are:

Symptom	Count
cough	67
cold	56
fever	46
ftg	36
mp	35
st	30
ht	25
loss_of_smell	19
bd	17
asthma	14

Table 11 : Symptoms of False Negatives in Stage–2

For the cases of False Negatives, we thoroughly examined the symptoms they exhibited and recorded the most frequent occurrences. We also examined the distribution of **covid_status** to see if the model systematically fails on positive cases with specific characteristics.

covid_status Distribution for False Negatives:	
covid_status_healthy	8
covid_status_no_resp_illness_exposed	2
covid_status_positive_asymp	1
covid_status_positive_mild	10
covid_status_positive_moderate	6
covid_status_recovered_full	3
covid_status_resp_illness_not_identified	2
covid_status_under_validation	0

Table 12 : covid_status Distribution of False Negatives in Stage–2

As shown in the table, *Table 12*, a significant part of the false negatives was actually covid–positive, especially mild and moderate cases. This indicates that while the model sometimes misses symptomatic covid patients, the problem is more evident in cases with less severe or unclear symptoms.

After the initial Two–Stage approach, we proceeded to optimize Stage–2. For this purpose, we used *RandomizedSearchCV* with Random Forest to find the best combination of hyperparameters.

Stage–2 Report (RandomizedSearchCV):				
	Precision	**Recall**	**F1 - Score**	**Support**
healthy	0.93	0.89	0.91	526
covid	0.61	0.61	0.61	219
asthma	0.69	1.00	0.81	24
pneumonia	0.33	0.40	0.36	5
cold	0.46	0.56	0.50	48
resp_other	0.00	0.00	0.00	2
Accuracy			0.79	824
Macro Avg	0.50	0.58	0.53	824
Weighted Avg	0.80	0.79	0.80	824

Table 13 : Report of Stage–2 with RandomizedSearchCV

After tuning Stage–2, *Table 13*, we observed some improvement in certain classes, such as asthma and cold, which achieved better balance between precision and recall. However, the overall accuracy, which is at 0.79, and macro average F1-Score, which is at 0.53, show that the model still struggles with the minority classes like pneumonia and other respiratory illnesses.

In our last move, we examined the role of **test_status** as an additional feature to our model. We created One-Hot Encoded columns for the test_status categories and added them to the CSV features, maintaining the same train/test splits in comparison with the previous model. Then, we retrained Stage–1 (Healthy vs Sick).

Stage–1 Report (with test_status in X):				
	Precision	**Recall**	**F1 - Score**	**Support**
healthy	1.00	0.99	0.99	468
sick	0.99	0.99	0.99	356
Accuracy			0.99	824
Macro Avg	0.99	0.99	0.99	824
Weighted Avg	0.99	0.99	0.99	824

Table 14: Report of Stage–1 including test_status

By including the test_status feature in Stage–1, the model's performance improved dramatically. The classification report, *Table 14*, shows almost perfect results, with precision, recall, and F1-Score close to 0.99 for both classes. In addition, we compared the false positives and false negatives of the new model with those of the previous one, while also checking the values in test_status and covid_status. When comparing error cases, the difference is significant. In the previous Stage–2, without test_status, we had 11 false positives and 32 false negatives, and after adding test_status the errors dropped to only 3 false positives and 2 false negatives.

This shows that the inclusion of test information strongly boosts the model's ability to distinguish between healthy and sick cases, significantly reducing misclassifications. However, it should also be noted that this improvement comes from using test results directly, which may not be realistic as the goal is to predict illness without prior test information.

Tools

For the implementation of this study, we used *Python* via *Visual Studio*, as it offers more reliable data processing, especially with audio, and more specifically audio in combination with CSV.

The libraries we used are the following:
For data handling and preprocessing:

- **pandas**, for data manipulation and analysis
- **numpy**, for numerical computing, arrays, and math operations
- **glob**, for matching file patterns
- **re**, for regular expressions for text/string matching
- **json**, for working with JSON files
- **shutil**, for file operations
- **tarfile**, for working with *.tar* archives
- and from **collection** the **defaultdict**, for advanced data structures

For audio processing

- **librosa**, for audio analysis and feature extraction

For machine learning and evaluation processes

- from **scipy.stats** the **randint**, for distributions for hyperparameter search
- from **sklearn** the:
  - **train_test_split**, to split data into train/test sets
  - **StandardScaler**, for feature scaling/normalization
  - **RandomForestClassifier**, for the classification model
  - **compute_sample_weight**, to handle class imbalance
  - **accuracy_score**, **f1_score**, **roc_auc_score**, **classification_report**, to get evaluation metrics
  - **confusion_matrix**, for the confusion matrix evaluation
  - **RandomizedSearchCV**, for hyperparameter tuning

Index of Tables & Figures

Members, Their Roles & The Time Plan

As someone sees on the cover page, the members of this project are two, and for the completion we worked together and simultaneously on the code and the report. As we were both very interested in every part of the procedure, we chose to have a group project with exchange and discussion of ideas.

Regarding the time plan, we initially worked on searching and understanding the data, designing and executing the code, for about 1 – 2.5 months, and then on writing the report, for about 2 weeks.

Inference & Model Loading

After training and tuning our models, and to avoid re-training the model every time, we saved the trained weights and other necessary artifacts *(Stage–1 binary model, Stage–2 multiclass model, the scaler for audio, feature names, class labels, and the chosen threshold)*. This way, we can test new samples and get predictions without re-training the whole pipeline.

During inference, we load these files, rebuild the input data, and normalize the audio using the saved scaler. The process is straightforward: load the saved models, preprocess any new samples with the same feature configuration, and then generate predictions.

```
import numpy as np
import pandas as pd
import joblib, json
from pathlib import Path

OUT = Path(r"…") #Path to our project

#Load artifacts
rf_bin2 = joblib.load(OUT / "rf_bin2.joblib")
rf_stage2 = joblib.load(OUT / "rf_stage2_best.joblib")
scaler_audio = joblib.load(OUT / "scaler_audio.joblib")
csv_cols = json.load(open(OUT / "csv_feature_names.json", "r", encoding="utf-8"))
class_names = json.load(open(OUT / "class_names.json", "r", encoding="utf-8"))
name2id = json.load(open(OUT / "name2id.json", "r", encoding="utf-8"))
best_thr = json.load(open(OUT / "stage1_threshold.json", "r"))["best_thr"]

#Load feature table and indices
full = pd.read_parquet(OUT / "full.parquet")
full.index = np.load(OUT / "full_index.npy", allow_pickle=True)
idx_test = np.load(OUT / "test_indices.npy", allow_pickle=True)

#Keep only indices that exist in 'full'
idx_test = np.asarray(idx_test)
exist_mask = np.isin(idx_test, full.index)
idx_valid = idx_test[exist_mask]
idx_missing = idx_test[~exist_mask]
if idx_missing.size:
    print(f"[WARN] Skipping {idx_missing.size} missing indices:
{idx_missing.tolist()}")

#Build CSV features
X_csv = full.loc[idx_valid, csv_cols]

#Build AUDIO features to match the scaler exactly
expected_audio = getattr(scaler_audio, "feature_names_in_", None)
if expected_audio is None:
    expected_audio = json.load(open(OUT / "audio_feature_names.json", "r",
encoding="utf-8"))
expected_audio = list(expected_audio)
present = full.columns.intersection(expected_audio)
X_audio = full.loc[idx_valid, present].copy()
for col in expected_audio:
    if col not in X_audio.columns:
```

```
        X_audio[col] = 0.0
X_audio = X_audio[expected_audio].astype(float)

#Scale audio and combine
X_audio_sc = scaler_audio.transform(X_audio)
X_test = np.hstack([X_csv.values, X_audio_sc])

#Stage 1
p_sick = rf_bin2.predict_proba(X_test)[:, 1]
y_pred_bin = (p_sick >= best_thr).astype(int)

#Stage 2
healthy_id = name2id["healthy"]
y_pred = np.full(len(idx_valid), healthy_id)
sick_mask = (y_pred_bin == 1)
if sick_mask.any():
    y_pred[sick_mask] = rf_stage2.predict(X_test[sick_mask])

#Convert to labels and show result counts
pred_labels = [class_names[i] for i in y_pred]
print(pd.Series(pred_labels).value_counts())
```