ClinicalBERT for ICU Phenotyping:
A Beginner's Tutorial

Introduction

Automated ICU phenotyping uses patients' electronic health records to infer clinical conditions—such as sepsis, heart failure, hypertension and diabetes—from free-text notes and structured data. This capability supports tasks like risk stratification and cohort building.

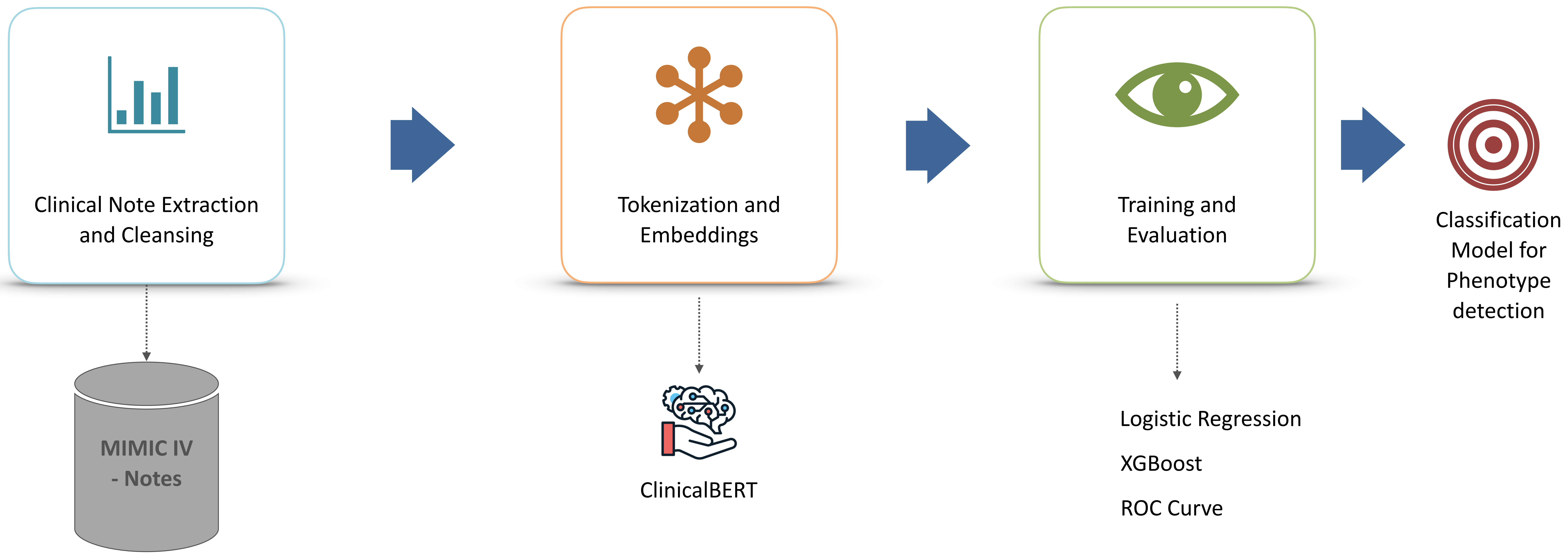
In this project, we leverage the MIMIC-IV ICU database, a large, publicly available repository of tens of thousands of ICU stays that includes both structured fields and de-identified narrative notes.

- We extract each patient's discharge note (one per hospital admission) and, via SQL queries on Google BigQuery, retrieve corresponding diagnosis codes.
- Phenotype labels are assigned by matching ICD-10 code prefixes—for example, any code beginning with "I50" denotes heart failure, "A41" indicates sepsis, and "E11" flags type 2 diabetes.
- Applying these rules yields a dataset in which every ICU admission is paired with its clinical note and a set of binary indicators for each phenotype of interest.
- We then apply NLP and Classification models to classify the phenotype based on the clinical notes

Project Goals and Architecture

Goal: Leverage AI/ML models to automatically infer patient phenotypes from ICU data by harnessing the rich detail in clinical notes—capturing observations, status descriptions, and clinician impressions.

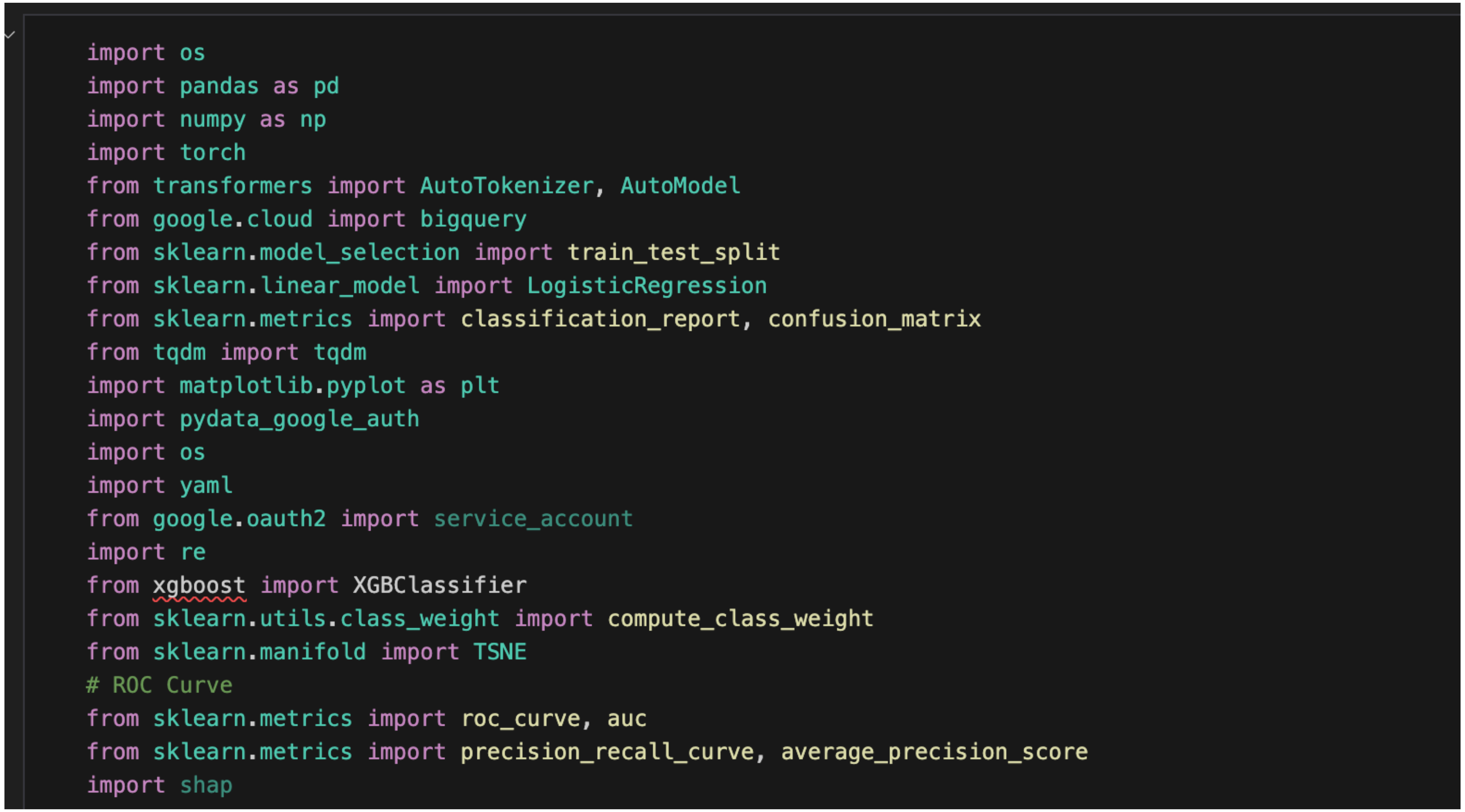
Because the notes are unstructured, we apply NLP techniques to extract meaningful features for phenotyping. For this project, we'll create a **Python** program to preprocess the text, use ClinicalBERT to generate rich embeddings, and train a classifier to predict patient phenotypes.


Clinical Note Extraction and Cleansing
MIMIC IV - Notes
Tokenization and Embeddings
ClinicalBERT
Training and Evaluation
Logistic Regression
XGBoost
ROC Curve
Classification Model for Phenotype detection

Importing Libraries

Import all necessary Python libraries for data processing, modeling, and visualization.

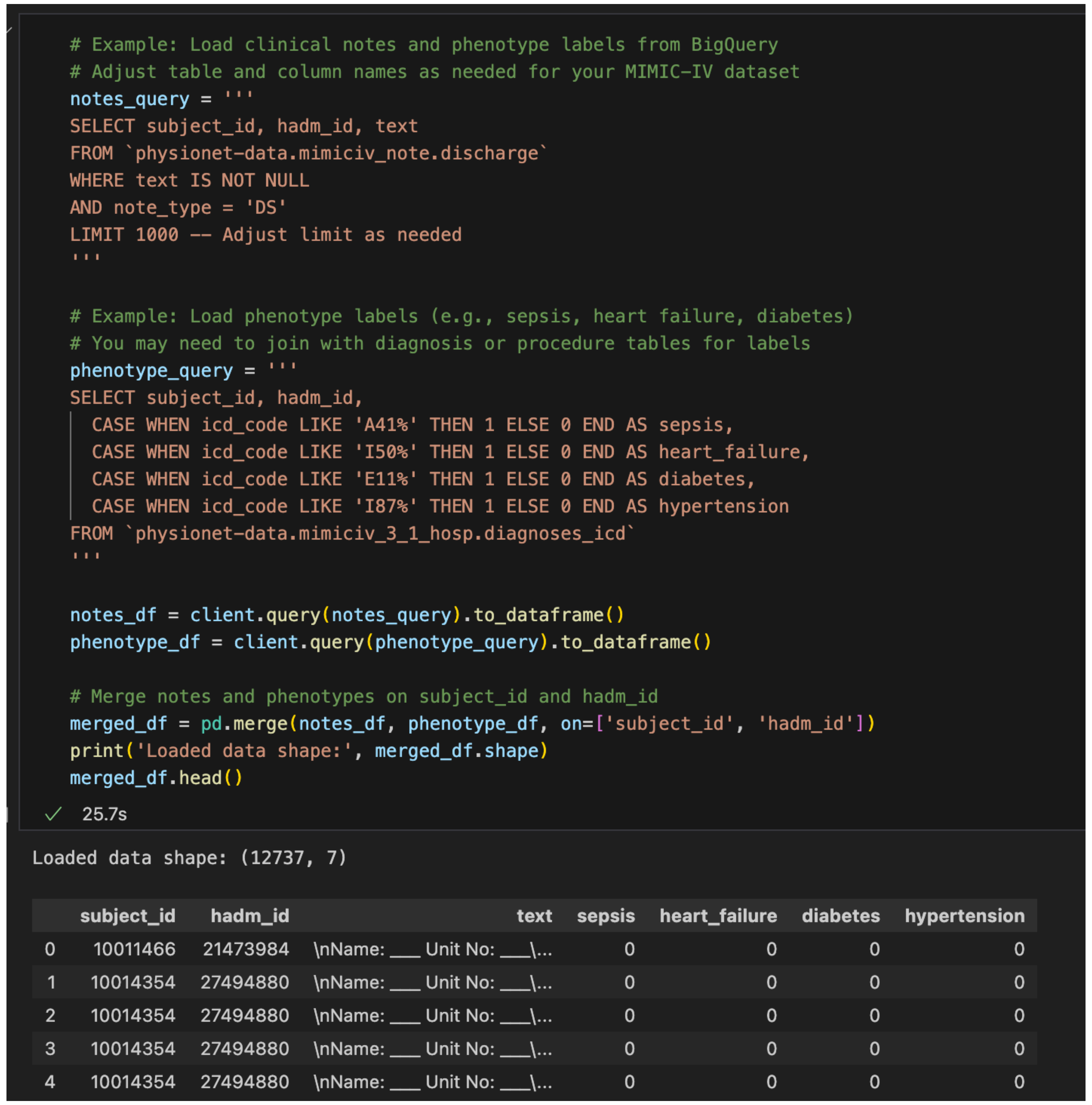
- Ensure you have Python 3.8+ and Jupyter installed.
- Install required packages: `transformers`, `torch`, `google-cloud-bigquery`, `pandas`, `scikit-learn`, `tqdm`, `matplotlib`.
- Set up your Google Cloud credentials (service account JSON) and set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.



```python
import os
import pandas as pd
import numpy as np
import torch
from transformers import AutoTokenizer, AutoModel
from google.cloud import bigquery
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from tqdm import tqdm
import matplotlib.pyplot as plt
import pydata_google_auth
import os
import yaml
from google.oauth2 import service_account
import re
from xgboost import XGBClassifier
from sklearn.utils.class_weight import compute_class_weight
from sklearn.manifold import TSNE
# ROC Curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score
import shap
```

Data Pre-Processing

- Before we can feed the data to BERT, we must preprocess it. First, the raw notes are loaded into a pandas DataFrame (after being queried from BigQuery). We also have the phenotype labels (0/1) for each note (for example, a column sepsis is 1 if that ICU stay had a sepsis diagnosis, else 0).
- Text Cleaning: The clinical text often contains a lot of noise (patient identifying info, formatting artifacts, etc.). We perform simple cleaning steps:
  - Convert all text to lowercase.
  - Remove special characters or punctuation, keeping only alphanumeric characters and spaces.
  - Collapse multiple spaces/newlines into a single space.
- This normalization makes the notes easier for the model to digest without extraneous symbols. For instance, in code we define a preprocess_text function that uses regex substitutions to remove non-alphanumeric characters and extra whitespace. After cleaning, we store the result in a new column like clean_note

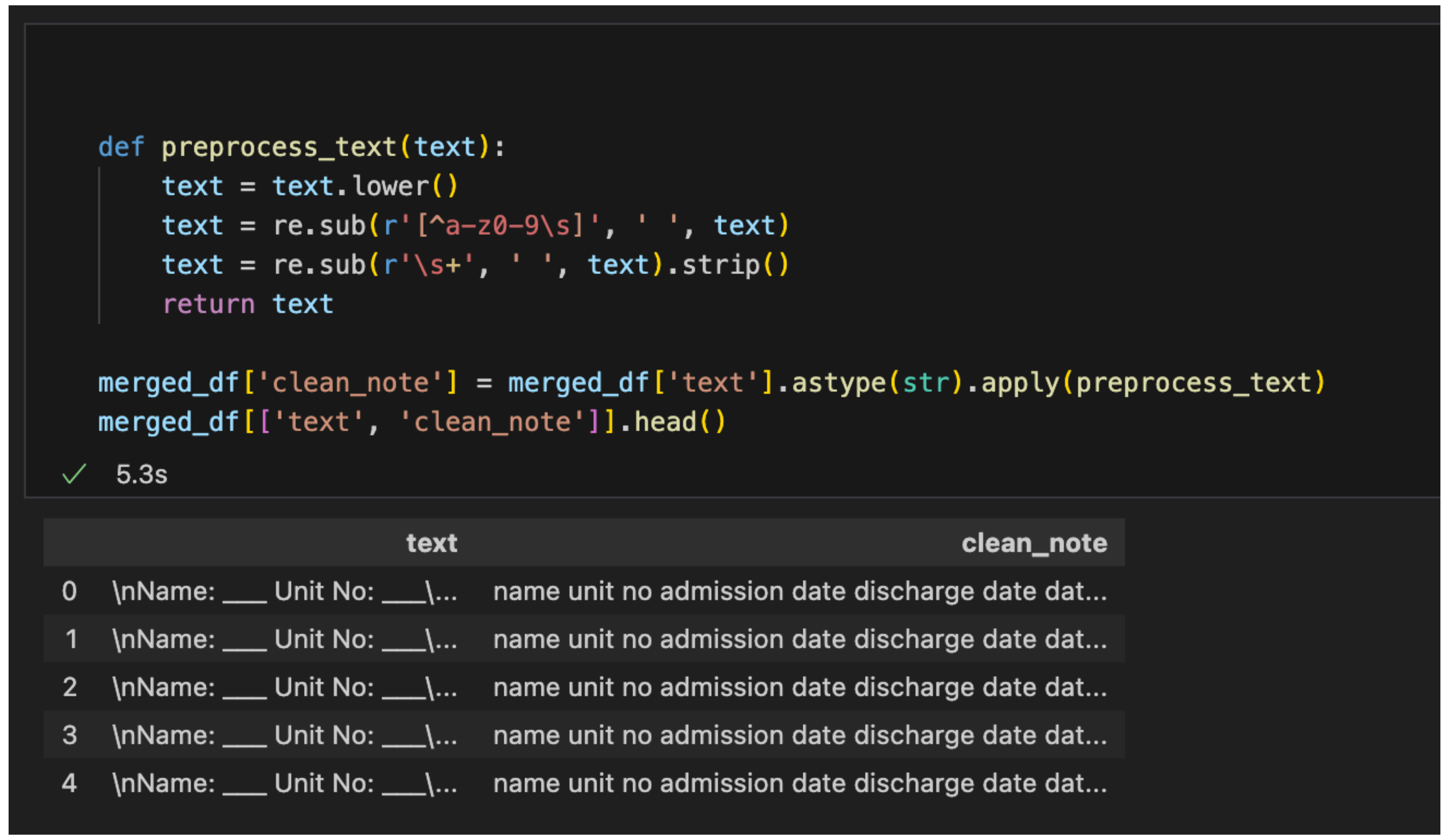


```python
# Example: Load clinical notes and phenotype labels from BigQuery
# Adjust table and column names as needed for your MIMIC-IV dataset
notes_query = '''
SELECT subject_id, hadm_id, text
FROM `physionet-data.mimiciv_note.discharge`
WHERE text IS NOT NULL
AND note_type = 'DS'
LIMIT 1000 -- Adjust limit as needed
'''

# Example: Load phenotype labels (e.g., sepsis, heart failure, diabetes)
# You may need to join with diagnosis or procedure tables for labels
phenotype_query = '''
SELECT subject_id, hadm_id,
  CASE WHEN icd_code LIKE 'A41%' THEN 1 ELSE 0 END AS sepsis,
  CASE WHEN icd_code LIKE 'I50%' THEN 1 ELSE 0 END AS heart_failure,
  CASE WHEN icd_code LIKE 'E11%' THEN 1 ELSE 0 END AS diabetes,
  CASE WHEN icd_code LIKE 'I87%' THEN 1 ELSE 0 END AS hypertension
FROM `physionet-data.mimiciv_3_1_hosp.diagnoses_icd`
'''

notes_df = client.query(notes_query).to_dataframe()
phenotype_df = client.query(phenotype_query).to_dataframe()

# Merge notes and phenotypes on subject_id and hadm_id
merged_df = pd.merge(notes_df, phenotype_df, on=['subject_id', 'hadm_id'])
print('Loaded data shape:', merged_df.shape)
merged_df.head()
```

✓ 25.7s

```
Loaded data shape: (12737, 7)
```

	subject_id	hadm_id	text	sepsis	heart_failure	diabetes	hypertension
0	10011466	21473984	\nName: ___ Unit No: ___\...	0	0	0	0
1	10014354	27494880	\nName: ___ Unit No: ___\...	0	0	0	0
2	10014354	27494880	\nName: ___ Unit No: ___\...	0	0	0	0
3	10014354	27494880	\nName: ___ Unit No: ___\...	0	0	0	0
4	10014354	27494880	\nName: ___ Unit No: ___\...	0	0	0	0



```python
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

merged_df['clean_note'] = merged_df['text'].astype(str).apply(preprocess_text)
merged_df[['text', 'clean_note']].head()
```

✓ 5.3s

	text	clean_note
0	\nName: ___ Unit No: ___\...	name unit no admission date discharge date dat...
1	\nName: ___ Unit No: ___\...	name unit no admission date discharge date dat...
2	\nName: ___ Unit No: ___\...	name unit no admission date discharge date dat...
3	\nName: ___ Unit No: ___\...	name unit no admission date discharge date dat...
4	\nName: ___ Unit No: ___\...	name unit no admission date discharge date dat...

- After this step, the note text is much cleaner (for example, headers like "Name: ___ Unit No: ___" are stripped out, leaving just meaningful words).
- It's a good practice to inspect a few cleaned notes to ensure the preprocessing did no harm.

Natural Language Processing: ClinicalBERT

- We leverage **ClinicalBERT**, a variant of the BERT model that is pre-trained on clinical text (specifically, on MIMIC-III clinical notes).
- BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based language model that produces contextual word embeddings.
- ClinicalBERT uses the same architecture as BERT-base (12-layer Transformer encoder) but has been initialized and further pre-trained on biomedical and clinical domain text.
- This gives it a better understanding of medical terminology, abbreviations, and chart note style language than a vanilla BERT trained on general text. In fact, studies have shown that ClinicalBERT outperforms general-purpose BERT on clinical tasks, validating the benefit of domain-specific pretraining[1]. Researchers have also found ClinicalBERT to outperform simpler text representations like bag-of-words or BiLSTM models on hospital tasks[2].

For our task of ICU phenotyping, we use the pre-trained ClinicalBERT and **fine-tune** it for classification. There are two possible approaches here:

- **Feature extraction approach (used in this tutorial):** load the pre-trained ClinicalBERT model and freeze its weights, use it to transform each note into a fixed vector embedding, and then train a separate classifier on those embeddings. This is what we do initially – we treat ClinicalBERT as a feature generator.
- **Fine-tuning approach (future work):** add a classification layer on top of BERT and train the entire model end-to-end on the phenotyping task. This updates BERT's weights slightly to better fit the task. Fine-tuning could potentially improve performance further, and indeed the authors suggest it as a next step

[1] ai.jmir.org

[2] medium.com

Tokenization & Embeddings

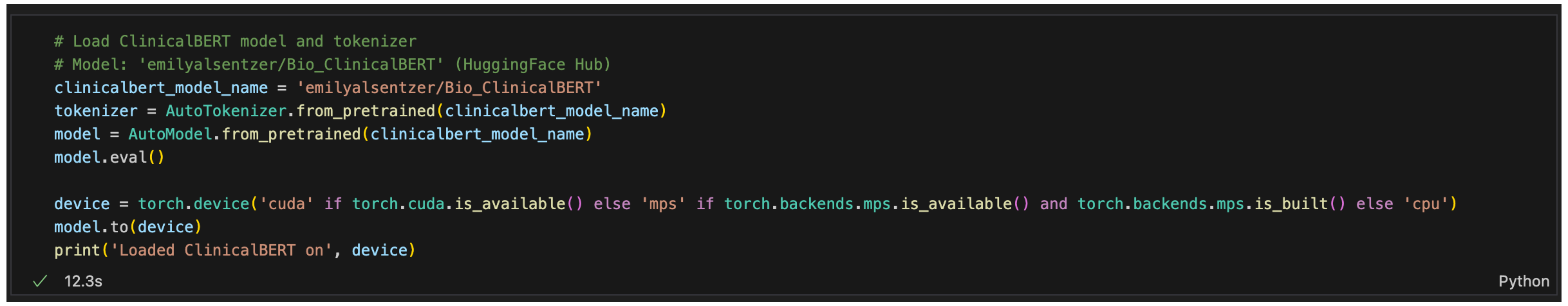
Tokenization:

Once we have cleaned text, we need to prepare it for input to BERT. BERT models expect tokenized input (subwords with special tokens).

- We use the **HuggingFace Transformers** library's tokenizer for ClinicalBERT to tokenize each note. This will break the text into word pieces, add the special `[CLS]` token at the beginning and `[SEP]` at the end, and pad/truncate to a fixed length. In this project, the maximum sequence length is set to 128 tokens for efficiency (notes longer than that are truncated). The tokenizer outputs tensors for the token IDs and attention mask which we will feed into the model.
- We also split our dataset into a training and test set (e.g. 80% train, 20% test). This allows us to train the model on one portion of the data and evaluate performance on unseen data to assess generalization. In code, this is done with scikit-learn's `train_test_split`.

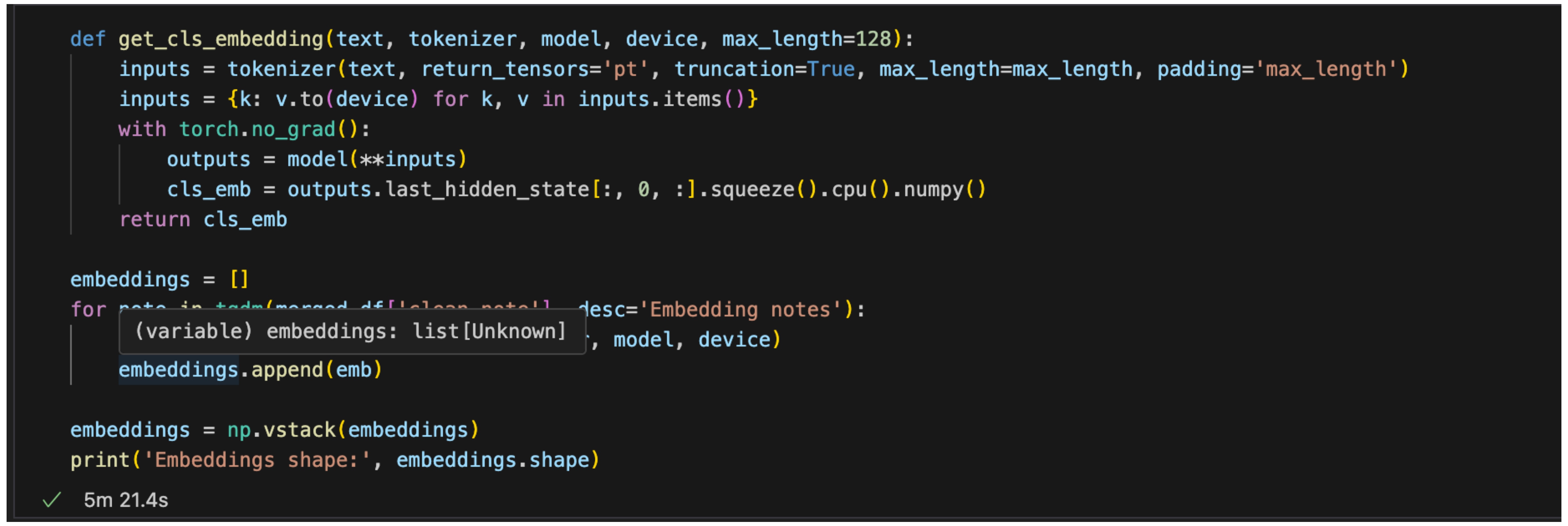
Embeddings:

- We load the pre-trained **`emilyalsentzer/Bio_ClinicalBERT`** model from HuggingFace Hub. This gives us a `tokenizer` and a `model` (an instance of `AutoModel` from Transformers).
- We then send the model to the available device (GPU if possible) and set it to eval mode (since we won't train BERT itself). For each note's cleaned text, we tokenize it and feed it through ClinicalBERT to obtain an embedding.



```
# Load ClinicalBERT model and tokenizer
# Model: 'emilyalsentzer/Bio_ClinicalBERT' (HuggingFace Hub)
clinicalbert_model_name = 'emilyalsentzer/Bio_ClinicalBERT'
tokenizer = AutoTokenizer.from_pretrained(clinicalbert_model_name)
model = AutoModel.from_pretrained(clinicalbert_model_name)
model.eval()

device = torch.device('cuda' if torch.cuda.is_available() else 'mps' if torch.backends.mps.is_available() and torch.backends.mps.is_built() else 'cpu')
model.to(device)
print('Loaded ClinicalBERT on', device)
```

✓ 12.3s Python



```
def get_cls_embedding(text, tokenizer, model, device, max_length=128):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, max_length=max_length, padding='max_length')
    inputs = {k: v.to(device) for k, v in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
        cls_emb = outputs.last_hidden_state[:, 0, :].squeeze().cpu().numpy()
    return cls_emb

embeddings = []
for note in tqdm(merged_df['clean_note'], desc='Embedding notes'):
    (variable) embeddings: list[Unknown] , model, device)
    embeddings.append(emb)

embeddings = np.vstack(embeddings)
print('Embeddings shape:', embeddings.shape)
```

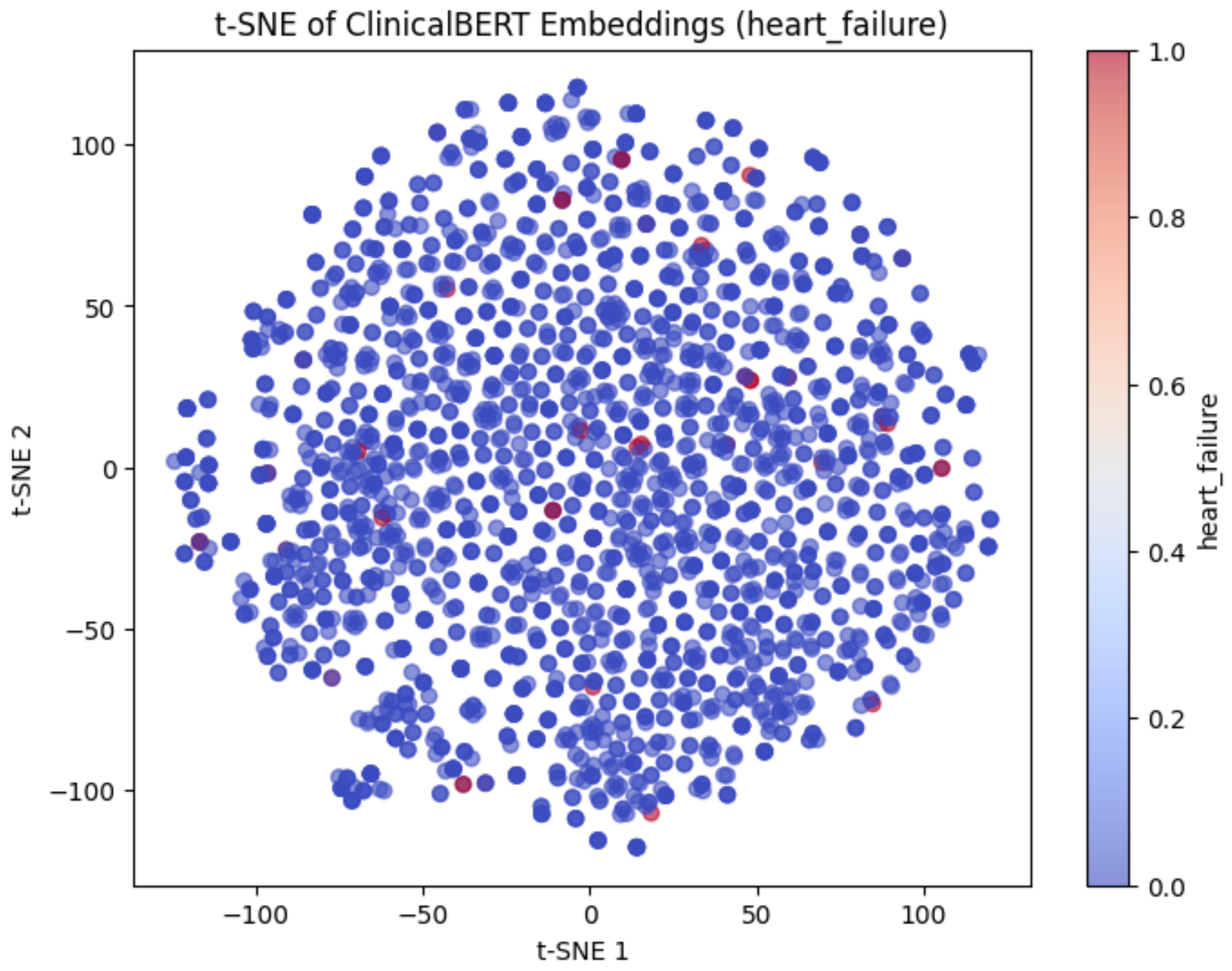
✓ 5m 21.4s

- The output is a 768-dimensional vector (because BERT-base hidden size is 768). Thus, for each patient note, ClinicalBERT produces a 768-dimension embedding capturing the semantic content of that note.
- **At this point, our textual data is converted into numerical features**: we have an array of shape (N, 768) where N is the number of notes (patients), and each row is that patient's note embedding. Now we can build a classification model on top of these embeddings to predict the phenotype labels.

Visualization of Embeddings

Visualization of embeddings: Visualize the high-dimensional ClinicalBERT embeddings of notes to see if patients naturally cluster by phenotype.

- Using a dimensionality reduction like t-SNE, we project the 768-dim embeddings to 2D and color the points by phenotype label.
- For example, plotting heart failure vs non-heart failure patients in embedding space did not show very distinct clusters (which is not surprising given the subtlety of the signal and class imbalance).
- This kind of plot can sometimes reveal if the model might easily separate the classes or if they heavily overlap in feature space.



Classification Model: Training and Evaluation

- **Classifier Training:** With embeddings as input features (X) and the phenotype label as the target (y), we train a classifier to predict the phenotype. In the notebook, a **logistic regression** model was used for demonstration. Logistic regression is a simple yet effective baseline for binary classification. We train the logistic regression model on the training data.
- Because some phenotypes are relatively rare (e.g., heart failure occurred in only a small fraction of patients in our sample), the classes are highly imbalanced. To address this, the classifier was configured with **class weights** so that mistakes on the minority class (positive cases) are penalized more during training. This helps the model not completely ignore the rare class. The logistic regression uses a binary cross-entropy loss (logistic loss) under the hood to optimize the classification boundary.
- After training, we evaluate on the test set. We also experimented with a more complex classifier, **XGBoost** (gradient boosted trees), on the same embeddings to see if it could capture nonlinear patterns. The XGBoost model was also trained on the training set after logistic regression. In practice, you could also try other classifiers like random forests, SVMs, or even a simple feed-forward neural network on the embeddings.

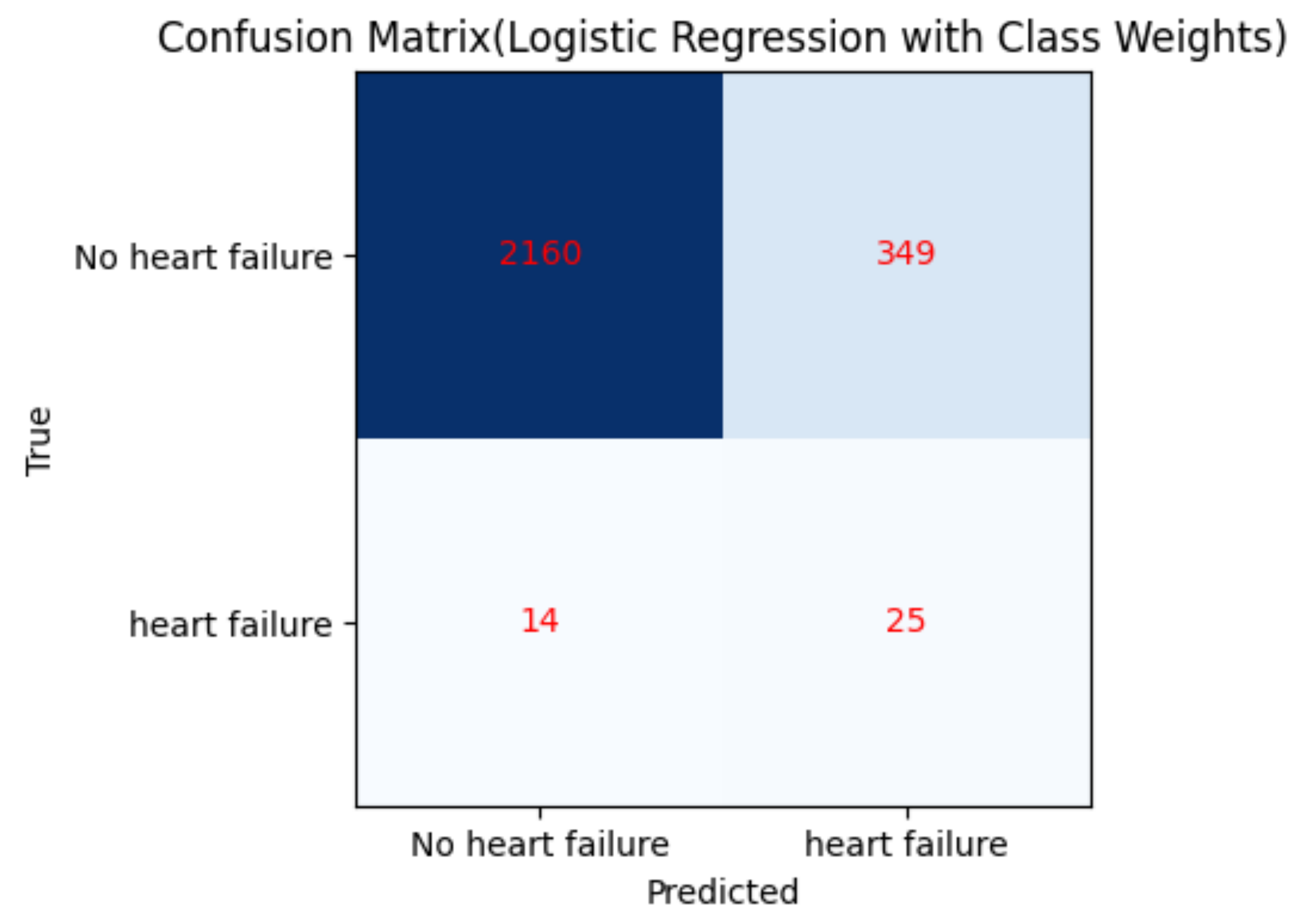
Evaluation Metrics: For evaluation, we look at several metrics:

- **Accuracy:** the overall fraction of correct predictions.
- **Precision:** among the cases predicted as positive (phenotype present), what fraction were truly positive.
- **Recall (Sensitivity):** among the true positive cases, what fraction the model correctly identified.
- **F1-score:** the harmonic mean of precision and recall (useful summary of classifier skill on the positive class).
- **Confusion matrix:** a 2x2 table showing counts of true negatives, false positives, false negatives, and true positives.

Measuring Model Performance

Classification with Class Weights

	precision	recall	f1-score	support
No heart failure	0.99	0.86	0.86	2509
heart failure	0.07	0.64	0.64	39
accuracy		**0.86**		2548
macro avg	0.53	0.75	0.75	2548
weighted avg	0.98	0.86	0.86	2548



Evaluation Metrics:

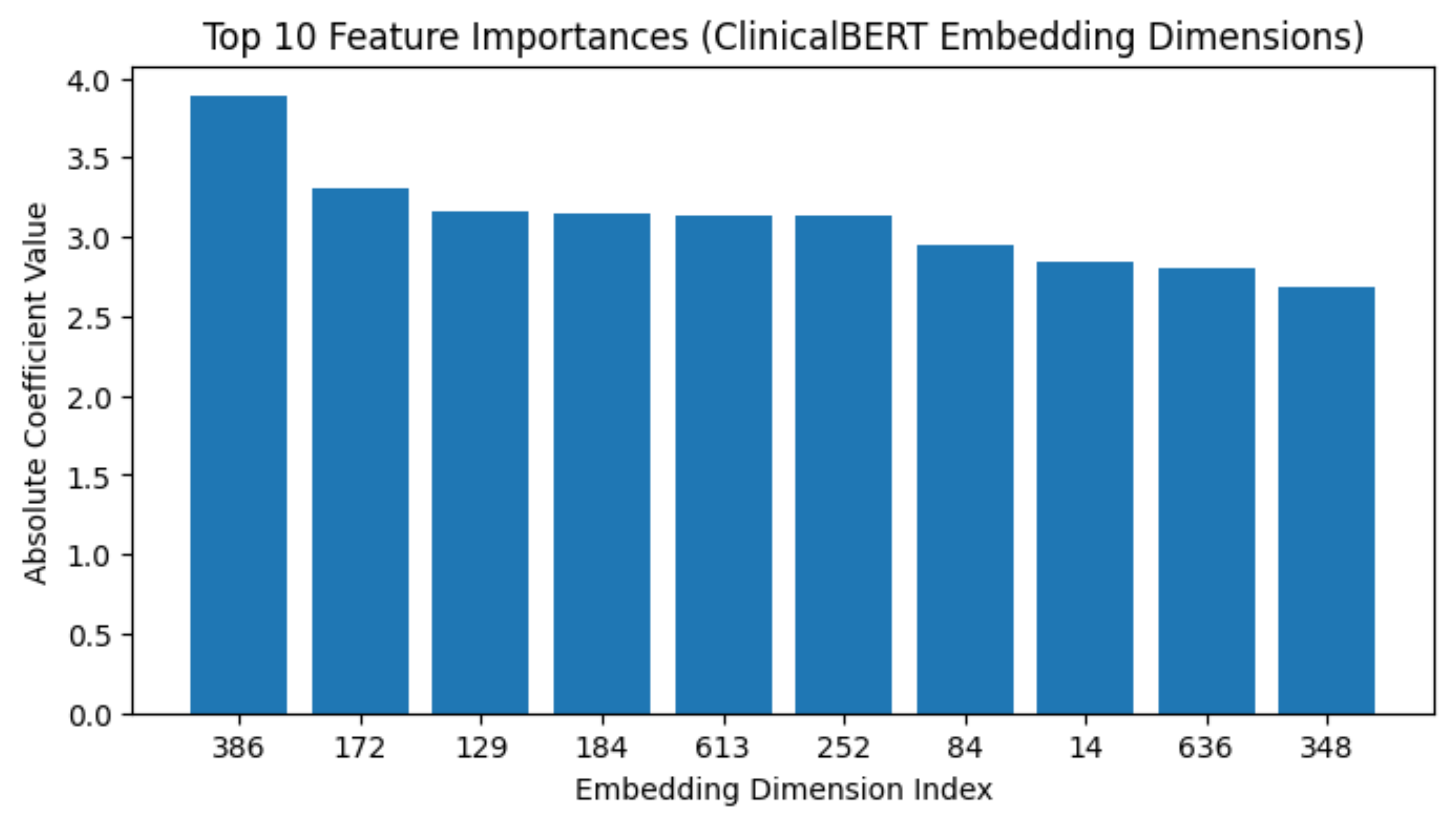
- Accuracy can be misleading when classes are imbalanced, so we pay close attention to precision, recall, and F1 for the positive class.
- We use scikit-learn's `classification_report` to get precision, recall, F1 for each class, as well as the overall (macro and weighted) averages.
- For example, in one of our phenotype tasks (heart failure prediction), the model achieved an **overall accuracy** of about 86%, but the **F1-score for the positive class** (heart failure) was much lower. This is because heart failure cases were very rare (only 39 positive cases out of ~12.7k), so the model could get many negatives right but struggled on the positives.

Confusion Matrix:

- This gives a snapshot of how the classifier is predicting relative to true labels. Each cell [i,j] shows the count of examples with true class i that were predicted as class j. In our heart failure example, the confusion matrix revealed the model's struggle with the positive class.
- *Confusion matrix for predicting heart failure (class 1 = heart failure, class 0 = no heart failure) using logistic regression with ClinicalBERT embeddings*. In this matrix, the bottom-right cell (25) are true heart failure cases correctly predicted (true positives), and the top-right cell (349) are false positives (cases without heart failure that were misclassified as positive).
- The model managed to recall about 25 of 39 true heart failure cases (~64% recall) but at the cost of flagging 349 false alarms, resulting in a precision around 7%. This imbalance is reflected in the poor F1-score for heart failure (around 0.12).

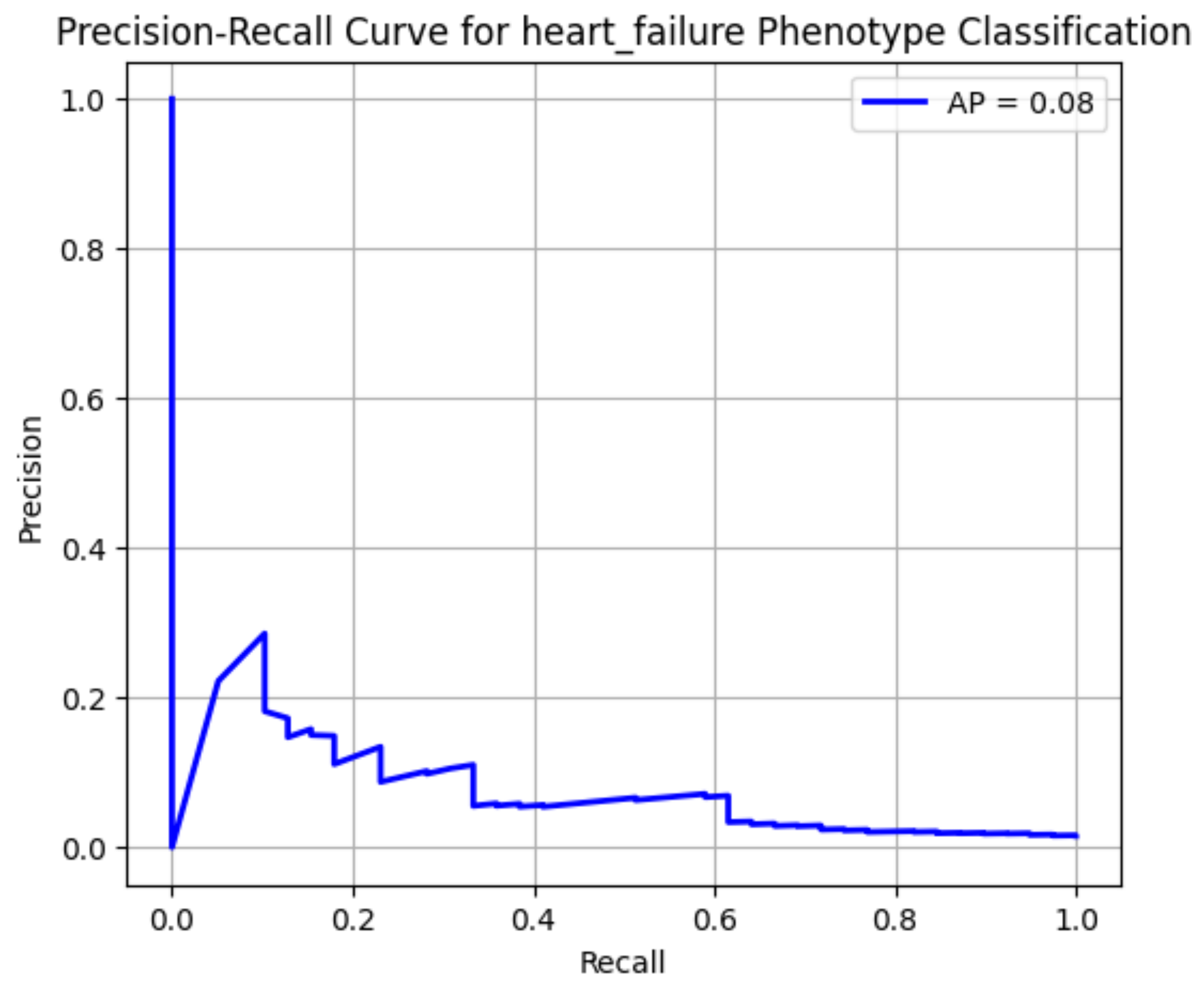
Additional Evaluation Techniques

Feature Importance

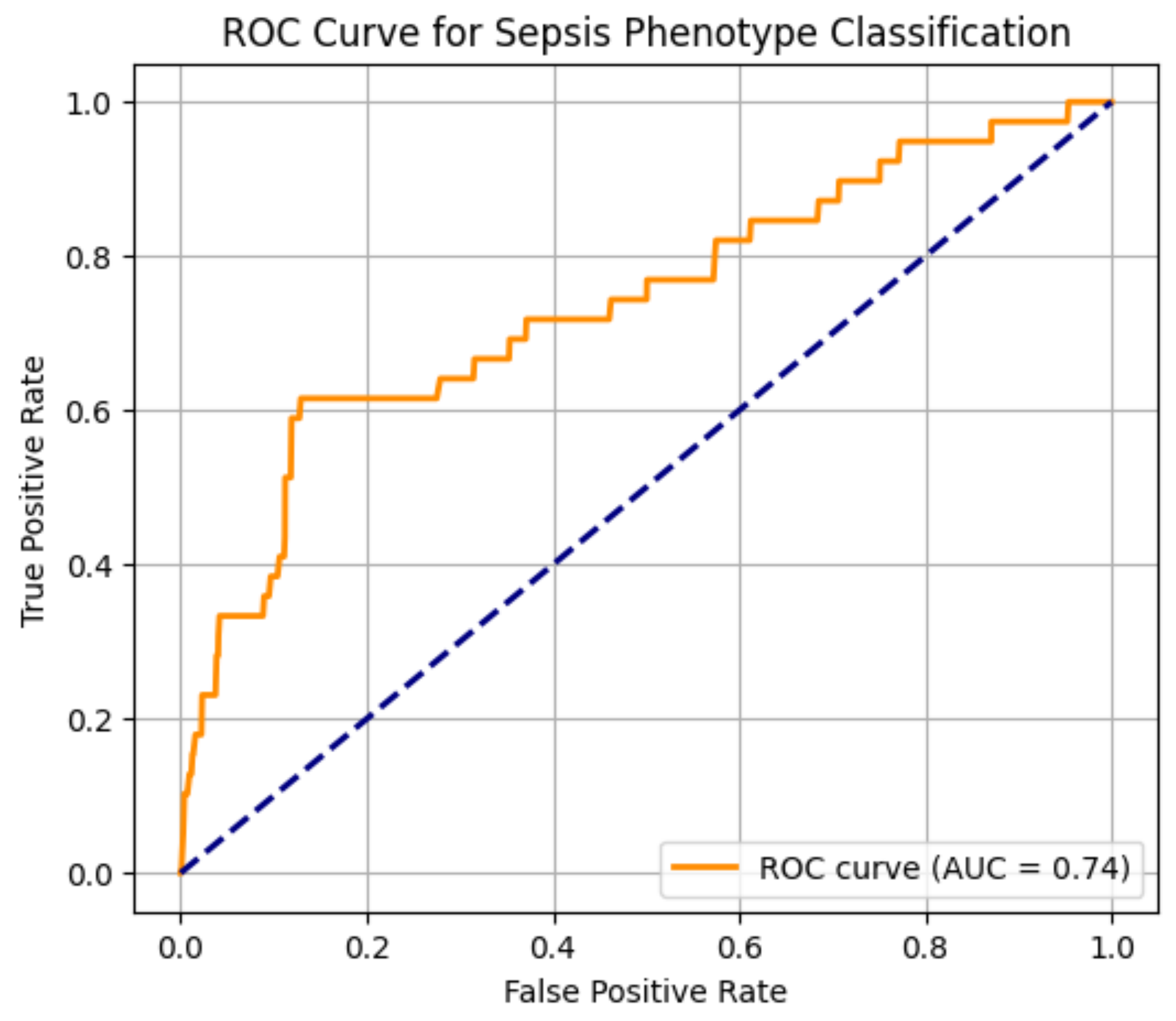


Additionally, one can look at **feature importance** to interpret the model. With logistic regression, we could examine the largest coefficients to see which dimensions of the ClinicalBERT embedding contribute most to the prediction. However, since individual embedding dimensions are not easily human-interpretable, a better approach is using SHAP values or other explanation techniques to identify influential words or sections in the note.

Precision Recall Curve



ROC Curve and AUC



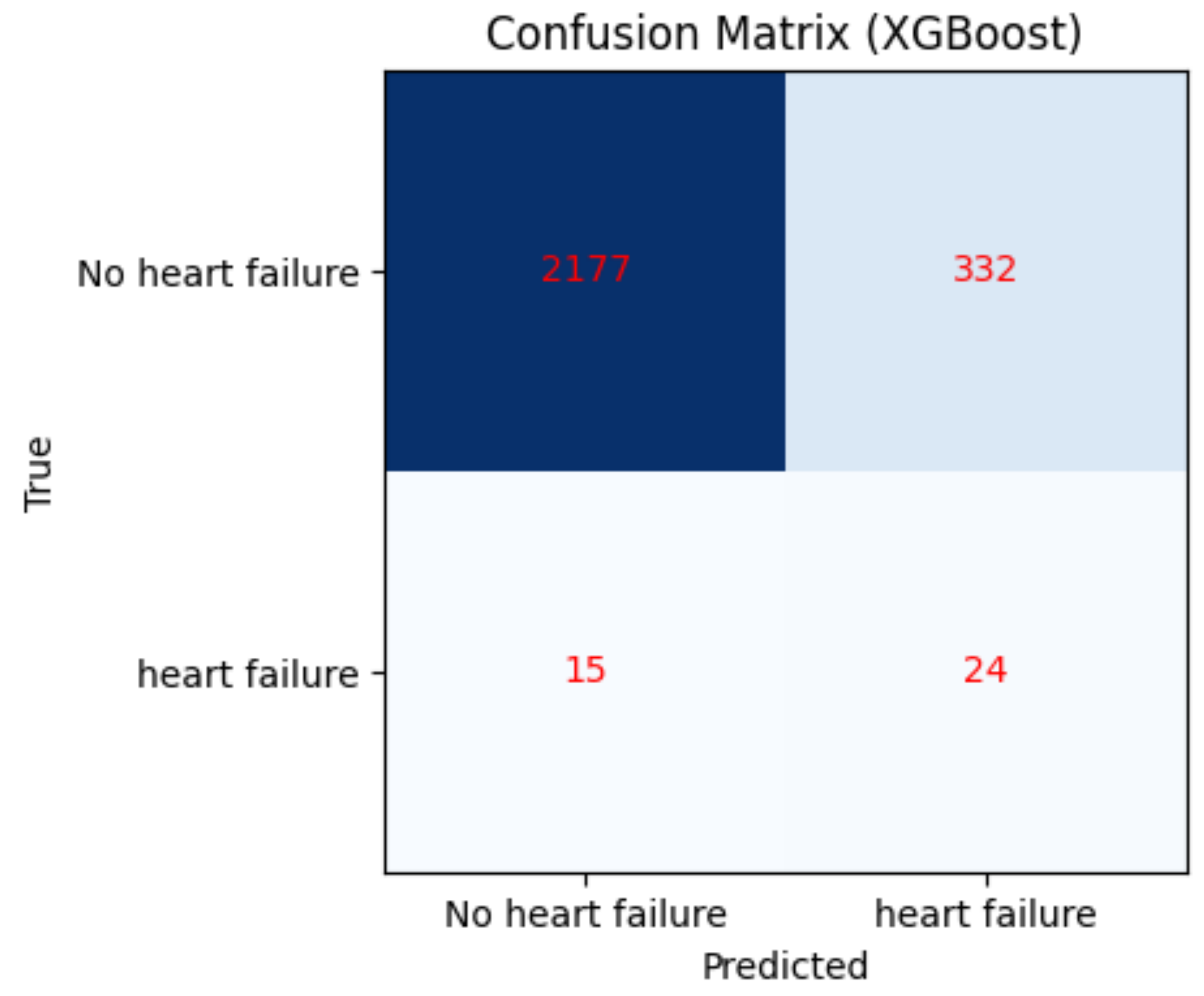
ROC Curve and precision-recall curve: For a phenotype that has a more balanced prevalence (or simply to evaluate ranking performance), an ROC curve is useful. Below is the ROC curve for the sepsis phenotype classifier. The ROC curve plots the True Positive Rate (recall) against the False Positive Rate at various classification thresholds. The **AUC (Area Under Curve)** summarizing this curve is around 0.74, indicating moderate discriminative ability.

ROC curve for sepsis phenotyping using ClinicalBERT embeddings. AUC ≈ 0.74 indicates moderate classification performance. In this case, the model is better at identifying sepsis than it was for heart failure, likely because sepsis was more commonly represented in the data (making the classes less skewed). We see a decent true positive rate can be achieved at the expense of some false positives. In general, an AUC closer to 1.0 would be ideal, whereas 0.5 would mean no better than chance. An AUC of 0.74 suggests the model is capturing some real signal from the notes for sepsis prediction, though there's room for improvement.

Performance Comparison

XGBoost Classification Report

	precision	recall	f1-score	support
No Heart failure	0.99	0.87	0.93	2509
Heart failure	0.07	0.62	0.12	39
accuracy			0.86	2548
macro avg	0.53	0.74	0.52	2548
weighted avg	0.98	0.86	0.91	2548



It's important to compare our ClinicalBERT-based approach with baseline methods to gauge its value. Traditional machine learning approaches for phenotyping might use simpler text representations or general NLP models:

- A basic baseline could be a **logistic regression on bag-of-words features** (or TF-IDF features) from the notes.
- Another baseline is to use a general **BERT-base model** (pretrained on Wikipedia/BookCorpus, i.e., not clinical text) either as a feature extractor or fine-tuned, to see if the clinical domain pretraining makes a difference.
- We could also compare against classical rule-based phenotyping or simpler models like Naive Bayes on the text.

We *did* compare two classifiers using the ClinicalBERT embeddings: logistic regression vs. XGBoost (a boosted tree model). Interestingly, the overall performance of XGBoost was quite similar to logistic regression in our heart failure task. XGBoost achieved the same 86% overall accuracy and likewise struggled with the positive class (precision ~7%, recall ~62%). This suggests that the limiting factor was not the choice of classifier but the inherent difficulty of the task (extreme class imbalance and possibly limited signal in a short note for that phenotype).

The takeaway is that **ClinicalBERT embeddings** provided a strong feature representation – any reasonably good classifier can leverage them to a point, but to further improve, we might need to either incorporate more data (e.g. multiple notes, structured data) or fine-tune the language model itself on the task.

Conclusion

In this tutorial, we demonstrated a full workflow for ICU phenotyping using ClinicalBERT:

1. We started with the MIMIC-IV ICU dataset and identified phenotypes (like sepsis, heart failure) using diagnosis codes.
2. We preprocessed the clinical note text, cleaning and preparing it for modeling.
3. We leveraged ClinicalBERT, a domain-specific BERT model, to transform unstructured notes into meaningful vector embeddings.
4. On top of these embeddings, we trained a simple classifier (logistic regression) to predict the presence or absence of phenotypes.
5. We evaluated the model using appropriate metrics and visualizations, seeing reasonable accuracy but also challenges with imbalanced data.

Insights: ClinicalBERT was effective in featurizing clinical notes such that even a basic classifier could pick up some signal for complex conditions. The model performed fairly well for more common phenotypes (as evidenced by a decent AUC for sepsis), but it struggled with very rare conditions (low precision for heart failure). This highlights the limitation of limited positive examples and the need for careful model tuning or data augmentation. We also saw that domain-specific language understanding is crucial – using a general BERT or bag-of-words likely would not have achieved the same level of performance.

In summary, ClinicalBERT-based phenotyping shows better results than using a general-language model or simpler text features, especially for capturing nuances in clinical language[2]. However, performance can still vary by phenotype: common phenotypes (with more training examples and clearer mentions in text) will be predicted more accurately than rare ones. Our results highlight that while ClinicalBERT provides a powerful foundation, proper handling of class imbalance and possibly task-specific fine-tuning are important to reach high recall on the rarer conditions.

Limitations and Next Steps: One limitation was that we did not fine-tune the ClinicalBERT model on our specific task; we used it as a fixed feature extractor. Fine-tuning could potentially improve the embeddings for the classification task. In the future, we could attach a feed-forward layer on top of BERT and train the whole model on the phenotyping labels. Another step is to try more advanced classifiers or ensemble methods to squeeze out better recall on the hard cases. We could also incorporate multiple notes per patient or other data (like vitals or lab results) to provide more context. Finally, exploring techniques like data balancing, threshold tuning (to optimize precision/recall trade-off), and more interpretability (e.g., highlighting which parts of the note led the model to predict a phenotype) would be valuable for a clinical application.

Overall, this project illustrates how **ClinicalBERT can be applied to clinical free-text for phenotype prediction**, and how to go from raw ICU notes to an evaluated classification model. With further refinement, such models could assist clinicians in automatically flagging patients with certain conditions from their notes, aiding in cohort building for research or clinical decision support. The combination of domain-specific NLP models and critical care data is a promising avenue to improve automated understanding of electronic health records.

[1]medium.com. [2]ai.jmir.org

THANK YOU