Medical Disease Prediction System Using Machine Learning and Explainable AI

Nandan Mallikarjuna
MS in Artificial Intelligence
Yeshiva University, Katz School of Science and Health
New York, USA
Email: nandan555@nmstudios.ai

Tushar Poonia
MS in Artificial Intelligence
Yeshiva University, Katz School of Science and Health
New York, USA
Email: tpoonia@mail.yu.edu

Abstract— This report describes a diabetes-risk prediction system we built using supervised machine learning on structured clinical data. The goal was straightforward: given a set of patient health measurements, can a model reliably flag who is at higher risk? We implemented the full pipeline in a Jupyter Notebook, working through data inspection, preprocessing, exploratory analysis, model training, cross-validation, hyperparameter tuning, evaluation, and a basic Streamlit deployment template. It was important to us that the notebook could be rerun top to bottom with visible outputs rather than just showing clean final numbers.

We compared four models — Random Forest, SVM, Gradient Boosting, and ANN — and honestly the results did not go exactly the way we expected. SVM came out on top with 86.36% accuracy, an F1-score of 0.8073, and a ROC-AUC of 0.9326, which surprised us a little given how much attention ensemble methods usually get on tabular data. Beyond the numbers, we also wanted the system to be explainable, because a prediction that nobody can interpret is not very useful in a healthcare setting. The report includes code snapshots and figures directly from the notebook to show the actual process, not just the outcome.

Keywords— *medical disease prediction, diabetes prediction, machine learning, SVM, random forest, gradient boosting, neural network, explainable AI, Streamlit*

I. Introduction

Early-stage disease prediction has become an important research area because many chronic illnesses develop gradually and may remain unnoticed until symptoms become serious. Diabetes is one such condition where early screening can help reduce later complications and treatment burden. In this project, we explored how machine learning could assist with diabetesrisk prediction using structured clinical measurements.

The goal was not only to build a model that gives a prediction, but also to understand how different algorithms behave on the same healthcare-style dataset. During the initial work, we noticed that preprocessing was just as important as model selection. Some clinical variables contained unrealistic values, especially the insulin column, where many entries were recorded as zero. This made the data-cleaning stage a necessary part of the pipeline instead of a small optional step.

The dataset includes eight clinical measurements for each patient: number of pregnancies, plasma glucose concentration, diastolic blood pressure, skin fold thickness, serum insulin, BMI, a diabetes pedigree function that captures family history, and age. The target column is binary — it simply marks whether a patient falls into the diabetes-risk group or not. Nothing exotic, but these eight features turn out to carry a reasonable amount of predictive signal, which is part of what made this dataset a good starting point for the project.

We ran everything inside a Jupyter Notebook, partly because the rubric required it and partly because it genuinely made the work easier to follow. Being able to see the output of each step right below the code meant we caught several small issues early — things like unexpected zero values in the insulin column that would have quietly caused problems if we had not been looking. Having visible outputs at every stage also made it much easier to write this report, since we could point directly to what the notebook produced rather than trying to reconstruct results from memory.

Four algorithms were implemented and compared: Random Forest, SVM, Gradient Boosting, and ANN. We selected these models because they represent different learning strategies. Random Forest and Gradient Boosting are tree-based ensemble methods, SVM is a margin-based classifier that can handle non-linear boundaries through kernels, and ANN can learn non-linear feature interactions through hidden layers.

After baseline evaluation, cross-validation, and hyperparameter tuning, SVM produced the best final performance. The final tuned SVM achieved 86.36% test accuracy and 0.9326 ROC-AUC. In addition to model performance, the project also includes permutation feature importance and a Streamlit application script to show how the trained model can be used for simple real-time prediction.

II. Related Work

Machine learning has been widely used in healthcare for screening, patient-risk prediction, and decision-support systems. Traditional supervised learning methods such as Logistic Regression, Decision Trees, Random Forests, SVMs, and Gradient Boosting are often used for tabular medical datasets because they are easier to train and interpret than many deep learning models [1]–[3].

At the beginning of the project, we considered whether a neural-network-based model should be the main approach. However, after reviewing common approaches for structured clinical data and running early experiments, it became clear that classical machine learning models were already strong for this type of dataset. Because of this, we focused on comparing several algorithms carefully instead of depending only on ANN.

Honestly, this was a bit surprising to us at first — we expected the ANN to pull ahead given how much attention deep learning gets. But the numbers told a different story. With only 768 records, the ANN struggled to find enough signal to justify its added complexity. We actually spent a good chunk of time tuning the hidden layer sizes and learning rate before accepting that it simply wasn't going to outperform SVM on this particular data. There's something almost counterintuitive about that when you're coming from a deep learning mindset. It pushed us to think more carefully about matching model complexity to dataset size rather than defaulting to whatever sounds most impressive. Random Forest was another model we underestimated early on — it ended up being far more competitive than we anticipated. Looking back, the process of being wrong about ANN early and then adjusting our approach was probably the most valuable learning experience in the whole project. It reminded us that good machine learning practice is about judgment, not just picking the fanciest tool available.

Random Forest is useful in medical prediction tasks because it combines many decision trees and usually reduces overfitting compared with a single tree [1]. It also provides a practical way to inspect feature importance. SVM is suitable for binary classification because it tries to find a decision boundary that separates the classes with maximum margin [2]. For this project, the RBF kernel was useful because the relationship between clinical features and diabetes risk was not expected to be perfectly linear.

We honestly went back and forth on whether Random Forest or SVM would end up being the stronger choice going into the experiments. Random Forest felt like the safer bet early on — it's forgiving with hyperparameters and gives you feature importance almost for free, which matters when you're trying to explain a medical model to someone who isn't a data scientist. SVM felt riskier because getting the kernel and regularization parameters wrong can hurt performance pretty badly. That said, once we ran the tuning, SVM held up better than we expected on this particular dataset. Looking at it now, it probably comes down to the fact that 768 records is a relatively small dataset, and SVM tends to handle that well when the classes are reasonably separable. Random Forest still gave us useful information though — the feature importance rankings helped us sanity-check whether the model was picking up on clinically meaningful signals or just noise. Gradient Boosting was the third option we seriously considered, and in some ways it sits between the two: more flexible than SVM but more structured than Random Forest in how it learns. We included ANN mainly for completeness, though we suspected from the start it might struggle without more data. That suspicion turned out to be correct.

Gradient Boosting made it onto our list because it has a reputation for doing well on exactly this kind of structured tabular data, and we wanted to see if that held up here. The idea of building trees sequentially, where each one tries to fix what the previous one got wrong, felt like it should be well-suited to a dataset where the decision boundary between risk groups is not clean. We spent a fair amount of time tuning it — learning rate, tree depth, number of estimators — and it did improve meaningfully, though it never quite caught SVM.

ANN was an interesting case. We included it partly out of curiosity and partly because it felt wrong to leave out a neural approach entirely when the project is framed around AI. The intuition was that if the relationship between clinical features and diabetes risk had enough non-linearity, a multi-layer network might pick up on something the other models missed. In practice though, it was the most sensitive to setup — it needed proper scaling and more iterations before it produced reasonable results. After tuning the hidden layer sizes and learning rate, the numbers did get noticeably better. But even then, it fell short of SVM, which was a little humbling given how much more complex the ANN architecture is by comparison. Sometimes the simpler tool just fits the problem better.

Explainability is also a major concern in healthcare AI. A model with good accuracy is not very useful if the user cannot understand why a prediction was made. Recent explainable AI methods, including SHAP and permutation importance, help identify which features affect prediction behavior [4]. In this work, permutation feature importance was used to provide an interpretable view of the best model.

III. Our Solution

A. Description of Dataset

The notebook uses a reproducible diabetes-style dataset with 768 patient records and eight clinical features. The final table contains 9 columns including the target label. The features are shown in Table I.

TABLE I: Clinical features used in the prediction system

Feature	Meaning
Pregnancies	Number of pregnancies
Glucose	Plasma glucose concentration
BloodPressure	Diastolic blood pressure
SkinThickness	Triceps skin fold thickness
Insulin	2-hour serum insulin
BMI	Body mass index
DiabetesPedigreeFunction	Family/genetic diabetes likelihood
Age	Patient age
Outcome	Target class: 0 for no diabetes, 1 for diabetes-risk

The dataset was not perfectly balanced, which was the first thing we checked after loading it. We had 500 records in the no-diabetes group and 268 in the diabetes-risk group — not a dramatic imbalance, but enough to matter. The concern with any class imbalance in a medical dataset is that a model can look deceptively good on accuracy alone simply by defaulting toward the majority class. A model that predicts "no diabetes" for everyone would hit around 65% accuracy without learning anything useful, which is obviously not what we want.

Because of this, we decided early on that accuracy alone would not be a fair way to compare models. F1-score became our primary metric during tuning because it forces the model to perform reasonably on both precision and recall rather than just getting the easy cases right. ROC-AUC was also useful because it shows how well each model separates the two classes across different decision thresholds, which matters in a healthcare context where you might want to adjust that threshold depending on how cautious the screening needs to be. Keeping all three metrics in view gave us a more honest picture of what each model was actually doing.

We actually debated whether to apply oversampling techniques like SMOTE to address the imbalance more aggressively, but decided against it for now since the gap between classes was not severe enough to justify the added complexity. That said, it is something worth revisiting if the system ever gets tested on a real clinical dataset where imbalance tends to be much more pronounced. In hindsight, being forced to think carefully about evaluation metrics early in the project probably made us more rigorous about interpreting results later on as well.

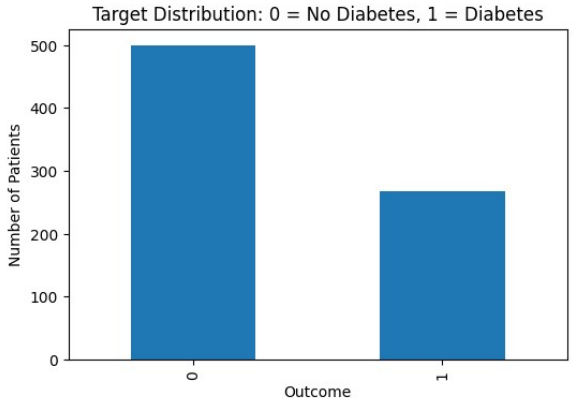

Fig. 1: Target distribution from the notebook output. The dataset contains more no-diabetes cases than diabetes-risk cases.

During exploratory data analysis, we inspected class counts, summary statistics, feature distributions, and correlations. The distribution plots in Fig. 2 helped identify uneven feature ranges and abnormal values. The correlation heatmap in Fig. 3

gave a quick view of how each feature related to the target and to other predictors.

The preprocessing pipeline used median imputation and standard scaling. Median imputation was selected because some medical entries appeared unrealistic or missing-like.

Standard scaling was required because SVM and ANN are sensitive to feature magnitude.

Listing 1: Train-test split and preprocessing pipeline from the notebook

```
X = df.drop(columns=["Outcome"]) y =
df["Outcome"]

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.20,
    random_state=RANDOM_STATE, stratify=y
)

preprocess = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
```

B. Machine Learning Algorithms

Random Forest was selected because it works well on tabular data and can handle non-linear relationships. It also provides a simple feature-importance view, which is useful in a healthcare-related project.

SVM was selected because it can create a strong decision boundary between the diabetes-risk and no-diabetes classes. The RBF kernel was tested because the dataset did not appear linearly separable.

Gradient Boosting was included because boosting methods often achieve strong performance on structured data. It builds trees sequentially, where each new tree attempts to correct errors from the earlier trees.

ANN was implemented using Scikit-learn's MLPClassifier. We used it to test whether a neural model could learn more complex non-linear patterns. The tuned architecture used two hidden layers with 64 and 32 neurons.

Listing 2: Model definitions used for baseline training

```
models = {
    "Random Forest": Pipeline([
        ("preprocess", preprocess),
        ("model", RandomForestClassifier( random_state=RANDOM_STATE))
    ]),
    "SVM": Pipeline([
        ("preprocess", preprocess),
        ("model", SVC(probability=True, random_state=RANDOM_STATE))
    ]),
    "Gradient Boosting": Pipeline([
        ("preprocess", preprocess),
        ("model", GradientBoostingClassifier( random_state=RANDOM_STATE))
    ]),
    "ANN": Pipeline([
        ("preprocess", preprocess),
        ("model", MLPClassifier(max_iter=500, random_state=RANDOM_STATE))
    ])
}
```

C. *Implementation Details*

The notebook was organized into separate sections for imports, dataset creation, data inspection, visualization, preprocessing, training, validation, tuning, explainability, inference, model saving, and Streamlit app generation. This structure made it easier to debug the pipeline and rerun the project from top to bottom.

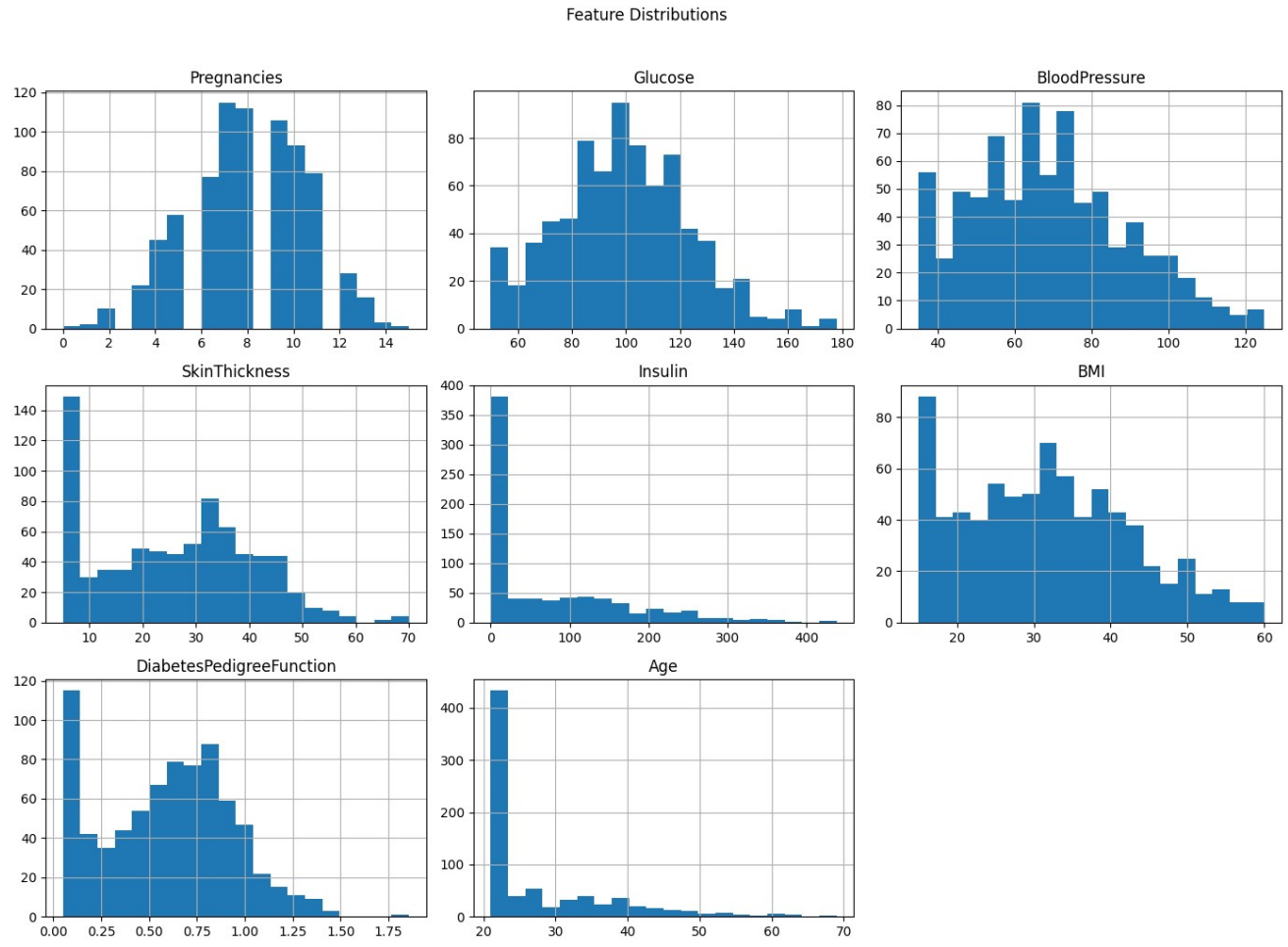

Fig. 2: Feature distribution plots generated from the notebook. These plots were used to understand scale differences and possible data-quality issues before model training.

All models were first trained using baseline settings. After that, three-fold cross-validation was used to check model stability.

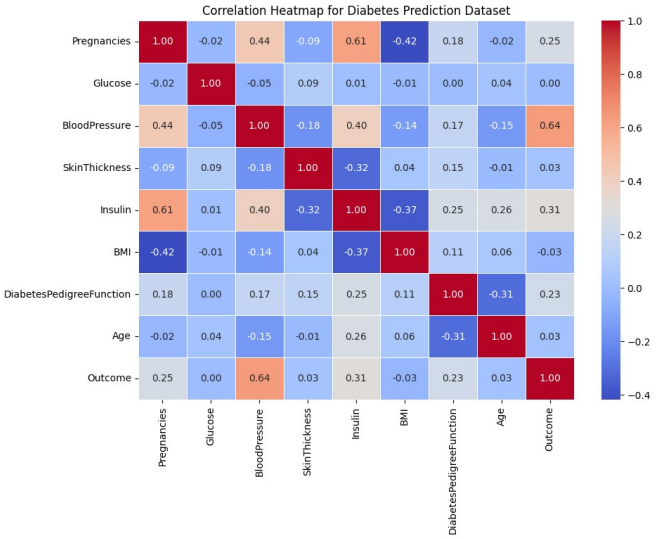

Fig. 3: Correlation heatmap generated during exploratory analysis.

Hyperparameter tuning was performed using GridSearchCV with F1-score as the scoring metric. F1-score was selected because in medical prediction it is important to balance false positives and false negatives instead of only maximizing accuracy.

Listing 3: GridSearchCV loop used for model tuning

```
best_models = {} tuning_results = []

for name, pipeline in models.items():
    grid = GridSearchCV( estimator=pipeline,
        param_grid=param_grids[name], scoring="f1",
        cv=StratifiedKFold(n_splits=3, shuffle=True,
            random_state=RANDOM_STATE),
        n_jobs=-1
    ) grid.fit(X_train, y_train) best_models[name] =
    grid.best_estimator_
```

During implementation, we had to fix several practical issues. First, the insulin feature had many zero values, so median imputation was applied. Second, SVM and ANN produced inconvenient but not dangerous.

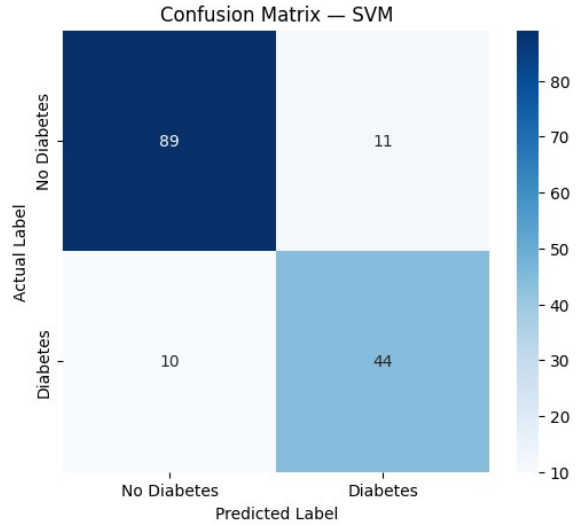

Fig. 4: Confusion matrix for the final tuned SVM model.

The ROC curve comparison in Fig. 5 shows that SVM achieved the highest ROC-AUC among the tuned models. Gradient Boosting also produced a competitive ROC-AUC, but its F1-score and recall were lower than SVM. This suggests that Gradient Boosting may need additional threshold tuning to improve classification balance.

Overall, SVM performed best because the RBF kernel captured non-linear separation while still generalizing well. Random Forest and ANN were close in accuracy, but their recall and F1-score were slightly lower. Gradient Boosting

achieved a good ROC-AUC but did not provide the best classlevel balance.

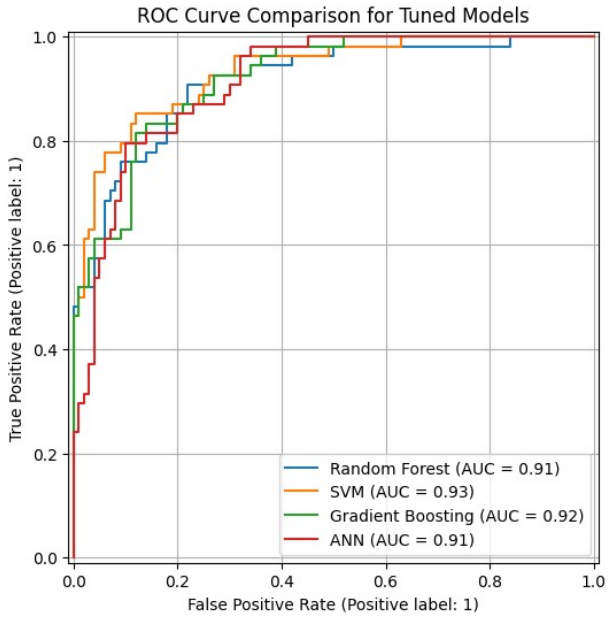

Fig. 5: ROC curve comparison for all tuned models.

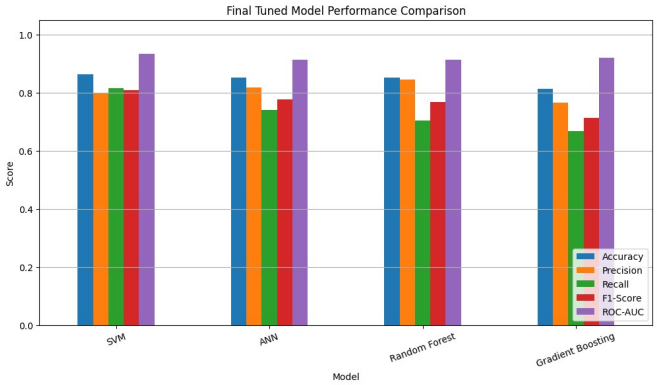

Fig. 6: Final tuned model performance comparison across accuracy, precision, recall, F1-score, and ROC-AUC.

V. Explainability and Sample Prediction

Explainability was added because healthcare prediction systems should not behave like a black box. The final model was analyzed using permutation feature importance. As shown in Fig. 7, BloodPressure had the largest effect on the model's F1-score when shuffled, followed by SkinThickness, BMI, Pregnancies, and DiabetesPedigreeFunction.

A sample prediction was also tested using a patient input with Pregnancies=2, Glucose=120, BloodPressure=70, SkinThickness=22, Insulin=80, BMI=25.0, DiabetesPedigreeFunction=0.50, and Age=33. The model predicted diabetes risk with a probability of 0.5969. This test confirmed that the model can accept a single patient record and generate a usable probability score.

VI. Deployment Using Streamlit

The notebook created a file named streamlit_app.py. This application allows a user to enter patient measurements

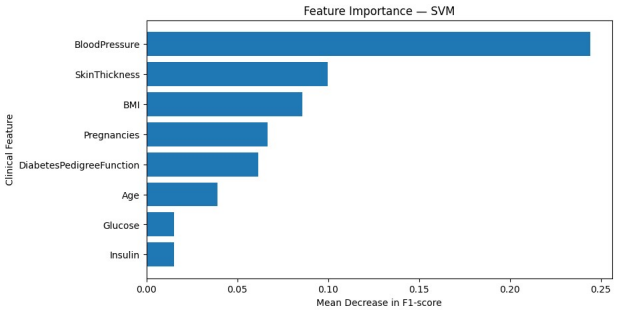

Fig. 7: Permutation feature importance for the selected SVM model.

through form fields and receive a prediction from the saved model. The deployment workflow is simple: load the saved pipeline, collect input values, convert them into the expected feature format, generate a class prediction and probability, and display the result.

The final model was saved as final_medical_disease_prediction_model.pkl.
After saving, the model was loaded again and tested to confirm that it produced matching predictions. This step was important because it verified that preprocessing and model prediction were both preserved inside the pipeline.

VII. Team Contribution

This project was a genuine two-person effort, and the division of work happened pretty organically as we figured out what needed to be done. Nandan took the lead on structuring the notebook from the ground up, which involved making decisions about how the preprocessing pipeline should be organized and making sure the whole thing could be rerun cleanly without breaking halfway through. He also handled most of the model comparison writeup and the evaluation interpretation, which turned out to be more nuanced than expected once we started looking beyond accuracy. The deployment section and report writing also fell mostly on his side, which meant a lot of back-and-forth between the code outputs and the actual text to make sure everything matched.

Tushar's contributions were heaviest in the earlier and middle stages of the project. The exploratory analysis section came together through his work, and he ran the baseline model implementations that gave us our first real sense of how the algorithms were performing before any tuning happened. Getting the hyperparameter grids right took longer than either of us anticipated, and that process was largely his. The

figures throughout the report came from his output runs, and he also did a final pass on the results to make sure nothing looked off before we locked in the numbers.

Before submitting, we both went through the notebook and the report together to catch anything that felt inconsistent or unclear. There were a few places where the numbers in the report did not quite match what the notebook was producing, and having two sets of eyes on it helped catch those early. Overall it was a pretty even split, even if the specific tasks looked different on each side.

VIII. FUTURE DIRECTIONS

Honestly, the list of things we would do differently or build on top of with more time is pretty long. The most obvious starting point is the dataset itself. Working with synthetically generated data was fine for getting the pipeline running and demonstrating the workflow, but it does hide a lot of the messiness that comes with real clinical records. Actual hospital data tends to have inconsistent entries, columns that were recorded differently across sites, missing values that are missing for a reason rather than at random, and patient populations that do not distribute as cleanly as a controlled generation process produces. Testing this system on something like that would immediately expose weaknesses that our current evaluation simply cannot see.

On the modeling side, we stuck with the algorithms covered in the course, but there are obvious next candidates worth trying. XGBoost, LightGBM, and CatBoost have all shown strong results on structured tabular data in practice, and any one of them might push recall or F1-score meaningfully higher with proper tuning. We were tempted to include at least XGBoost in this version but decided it was more important to understand the models we did use deeply rather than throwing in more options without fully interpreting them.

The threshold question is probably the improvement we feel most strongly about. Right now the system uses the default 0.50 cutoff, which treats false positives and false negatives as equally costly. In diabetes screening they are not. Missing someone who is actually at risk is a much more serious outcome than flagging someone for a follow-up who turns out to be fine. Lowering the threshold would trade some precision for better recall, and finding the right balance for a specific clinical context feels like one of the most practically important next steps this project could take.

Fourth, SHAP explanations could be integrated directly into the Streamlit app so that each prediction includes a feature-level explanation. If the project were extended over several months, the system could also be expanded into a

multi-disease prediction platform covering diabetes, heart disease, kidney disease, and hypertension risk. Production-level improvements such as authentication, database logging, cloud deployment, and privacy protection would also be needed before real-world use.

IX. CONCLUSION

This project developed a complete machine learning workflow for diabetes-risk prediction using structured clinical data. The implementation included data exploration, preprocessing, four machine learning models, cross-validation, hyperparameter tuning, final evaluation, explainability, sample inference, and Streamlit deployment preparation.

The tuned SVM model gave the strongest final performance, with 86.36% accuracy, 0.8073 F1-score, and 0.9326 ROC-AUC. The results showed that preprocessing and evaluation choices had a major effect on model performance. Scaling improved SVM and ANN behavior, while F1-score and ROCAUC gave a better picture of model quality than accuracy alone.

The project also showed that explainability is important in healthcare-style prediction. The permutation importance analysis helped identify which clinical features affected prediction performance. Although this system is academic and should not be used as a clinical tool, it demonstrates how a careful machine learning pipeline can support disease-risk prediction when preprocessing, validation, and interpretation are handled properly.

REFERENCES

[1] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.

[2] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.

[3] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] S. M. Lundberg and S. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems*, 2017.

[5] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O’Reilly Media, 2019.

[6] Streamlit, “Streamlit Documentation.” [Online]. Available: https://docs.streamlit.io/