Building Machine Learning Models for Diabetes Health Indicators: Report by Elham Ghiasi

Introduction:

In this project, we will work with the Diabetes Health Indicators dataset and build machine learning models to classify the presence of diabetes. We will perform exploratory data analysis (EDA) using Python. The dataset includes various health indicators such as BMI, physical activity, smoking habits, alcohol consumption, general health rating, and diabetes diagnosis status, which can be used to predict whether a person has diabetes.
The dataset includes 253,680 survey responses collected through the CDC's 2015 Behavioral Risk Factor Surveillance System (BRFSS).

The target variable, Diabetes_binary, is a binary classification with two values: 0 indicates no diabetes, while 1 represents either prediabetes or diabetes. There are 21 feature variables in total.

In this project, we will use Jupyter Notebook and Python, and we will analyze our results using graphs, charts, and visualizations.

Observation:

1. Exploratory Data Analysis

1.a.

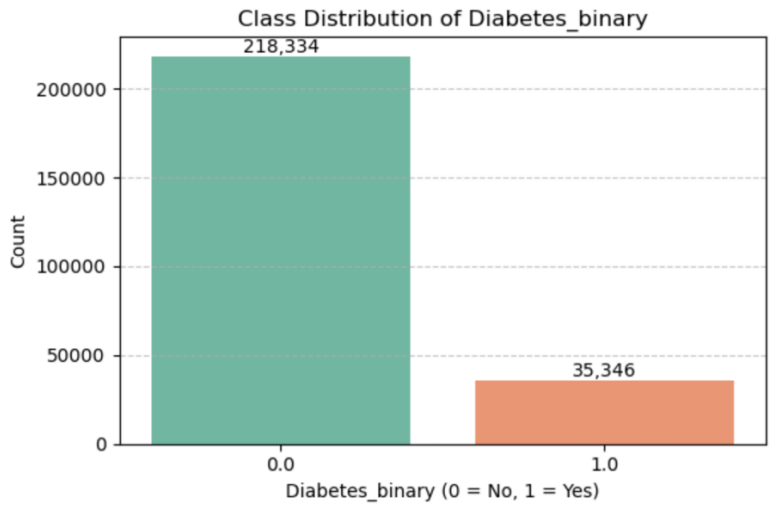
First, we need to load the dataset, explore its dimensions, and handle any missing values. The dataset contains 253,680 rows and 22 columns. Each row represents an individual's survey response, and each column corresponds to a health indicator, including the target variable Diabetes_binary. No missing values were found in the dataset. All entries are clean and numeric.

1.b.

Calculating the balance of the target variable (Diabetes_binary):
This dataset is quite imbalanced — about 86% (86.07) of the entries are labeled as "No diabetes" and only around 14% (13.93) as "Diabetes or prediabetes." That means most of the data belongs to the majority class, which could make it harder for models to learn and correctly predict the minority class. It's something we need to keep in mind when choosing and evaluating our models.

1.C.

Visualizing the class distribution using plots (bar plots…):
The bar plot makes it even clearer that the dataset is imbalanced, there are way more samples in the "No diabetes" group than in the "Diabetes/prediabetes" group. This kind of skew can make it tough for the model to learn patterns from the smaller class, which might hurt its accuracy when predicting those cases.
This is the bar Plot:



1.d.

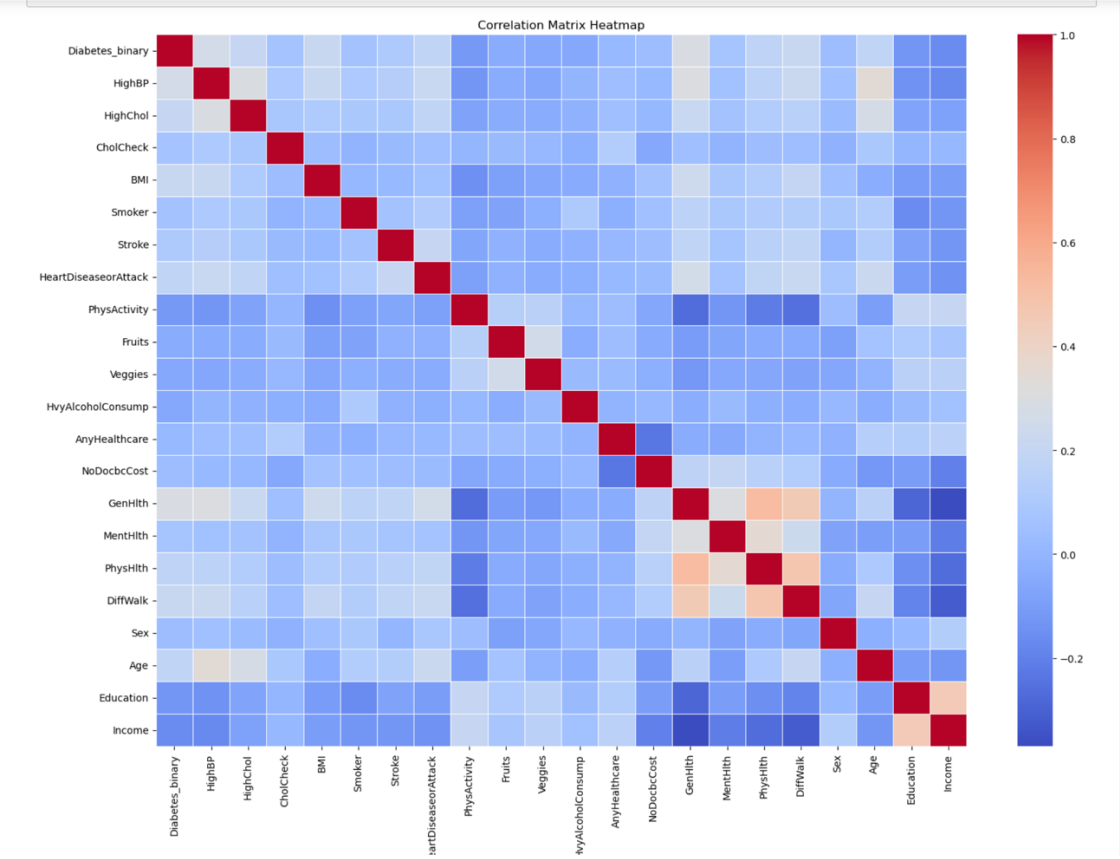
Exploring descriptive statistics and create a table:
The table that we created gives a quick summary of each feature, things like the average, standard deviation, min, and max values. It helps us get a feel for how the data is spread out. Some features like BMI, Age, and Mental/Physical Health have wide ranges, so if we don’t scale them properly, it could mess with how the model learns.
below you can see the table:

	count	mean	std	min	25%	50%	75%	max
Diabetes_binary	253680.0	0.14	0.35	0.0	0.0	0.0	0.0	1.0
HighBP	253680.0	0.43	0.49	0.0	0.0	0.0	1.0	1.0
HighChol	253680.0	0.42	0.49	0.0	0.0	0.0	1.0	1.0
CholCheck	253680.0	0.96	0.19	0.0	1.0	1.0	1.0	1.0
BMI	253680.0	28.38	6.61	12.0	24.0	27.0	31.0	98.0
Smoker	253680.0	0.44	0.50	0.0	0.0	0.0	1.0	1.0
Stroke	253680.0	0.04	0.20	0.0	0.0	0.0	0.0	1.0
HeartDiseaseorAttack	253680.0	0.09	0.29	0.0	0.0	0.0	0.0	1.0
PhysActivity	253680.0	0.76	0.43	0.0	1.0	1.0	1.0	1.0
Fruits	253680.0	0.63	0.48	0.0	0.0	1.0	1.0	1.0
Veggies	253680.0	0.81	0.39	0.0	1.0	1.0	1.0	1.0
HvyAlcoholConsump	253680.0	0.06	0.23	0.0	0.0	0.0	0.0	1.0
AnyHealthcare	253680.0	0.95	0.22	0.0	1.0	1.0	1.0	1.0
NoDocbcCost	253680.0	0.08	0.28	0.0	0.0	0.0	0.0	1.0
GenHlth	253680.0	2.51	1.07	1.0	2.0	2.0	3.0	5.0
MentHlth	253680.0	3.18	7.41	0.0	0.0	0.0	2.0	30.0
PhysHlth	253680.0	4.24	8.72	0.0	0.0	0.0	3.0	30.0
DiffWalk	253680.0	0.17	0.37	0.0	0.0	0.0	0.0	1.0
Sex	253680.0	0.44	0.50	0.0	0.0	0.0	1.0	1.0
Age	253680.0	8.03	3.05	1.0	6.0	8.0	10.0	13.0
Education	253680.0	5.05	0.99	1.0	4.0	5.0	6.0	6.0
Income	253680.0	6.05	2.07	1.0	5.0	7.0	8.0	8.0

1.e.

Calculate the correlation matrix and generate heatmaps to visualize correlations:
The heatmap shows how different features relate to each other. Most of them aren't strongly correlated, but there are a few pairs, like Mental Health and Physical Health, or Income and Education, that seem to have a stronger connection. There's no sign of serious multicollinearity, but if needed, we can dig deeper with something like a VIF check when we build the model.
To double-check for multicollinearity, I unstacked the correlation matrix and filtered out any pairs with a correlation higher than 0.8 (but not exactly 1). Nothing popped up as highly correlated, so there's no big red flag for multicollinearity right now. That said, just to be safe, we can always run a VIF check later during model building to make sure everything's stable.

Screenshot below:



2. Logistic Regression

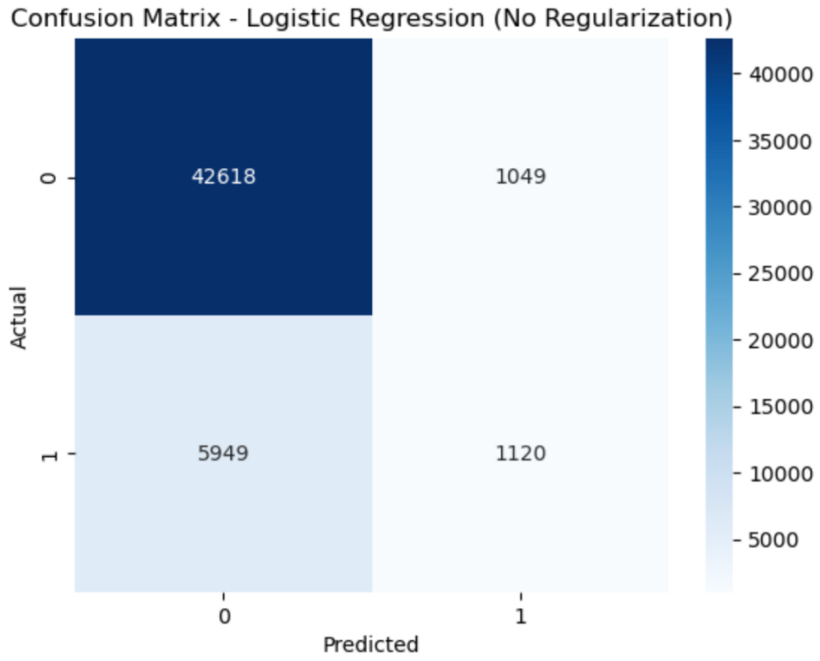
2.a. Logistic regression (no regularization)

Before beginning to modeling, the first step is to separate the features from the target variable. Then, we split the data into training and test sets so we can train the model on one part and test how well it does on the other. It's also a good idea to scale the features.

Results:
Accuracy: 0.8620703248186692
Precision: 0.5163669893960351
Recall: 0.1584382515207243
F1 Score: 0.2424767265641914

The model achieved an accuracy of 86.2%, which looks good at first glance. However, when we dig into the performance for Class 1 (positive diabetes cases), the precision is 51.6%, recall is just 15.8%, and the F1-score is 24.2%. This tells us that the model isn't doing a great job identifying those with diabetes. The low recall and F1-score suggest it's likely affected by class imbalance and possibly the lack of regularization.

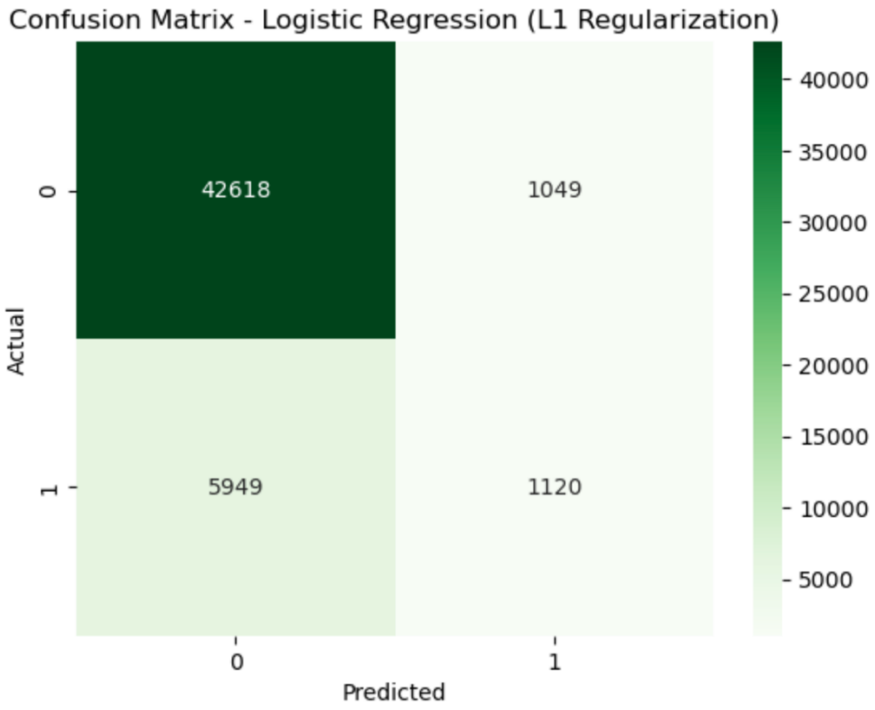
Confusion matrix:



2b: Logistic Regression with L1 (Lasso)

The model's accuracy is 86.2%, with a precision of 51.6%, recall of 15.8%, and F1-score of 24.2% for Class 1. When we applied L1 regularization, there was no noticeable change in performance—the confusion matrix and metrics stayed the same as the unregularized model. This suggests that L1 didn't have much effect, possibly because the model was already sparse or many of the features didn't contribute much to begin with.

Results:
Accuracy: 0.8620703248186692
Precision: 0.5163669893960351
Recall: 0.1584382515207243
F1 Score: 0.2424767265641914

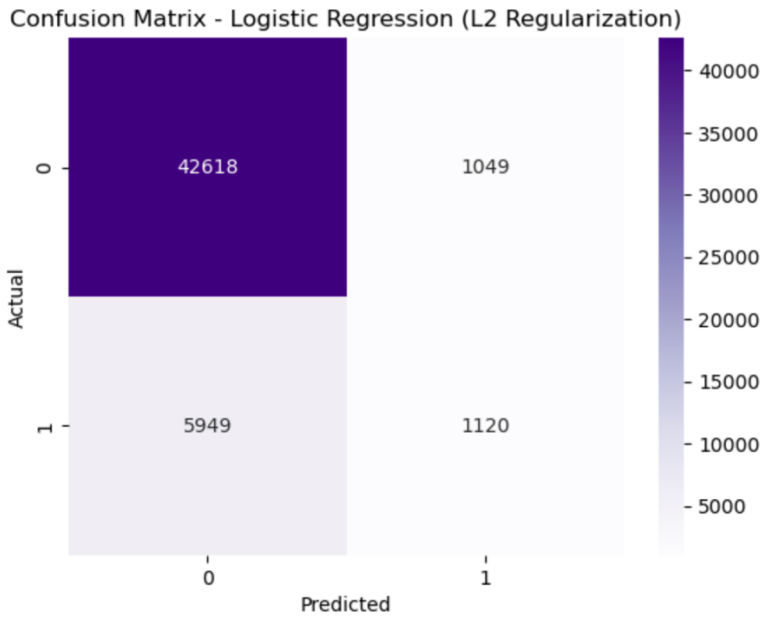


2c: Logistic Regression with L2 (Ridge)

(Logistic Regression with L2 Regularization)

The model with L2 regularization gave an accuracy of 86.21%, with precision at 51.8%, recall at 15.8%, and an F1-score of 24.2% for Class 1. These results are nearly identical to what we saw with both the L1 and non-regularized models. So, it seems that adding regularization didn't make much of a difference here likely because of the class imbalance and how the data is structured.

Results:
Accuracy: 0.8621294544307789
Precision: 0.517097966728281
Recall: 0.15829678879615222
F1 Score: 0.24239142207299902



2d: Comparison & discussion

All three models, Logistic Regression without regularization, with L1, and with L2—performed almost identically, showing 86.21% accuracy and similar precision, recall, and F1-scores for Class 1. Regularization didn't lead to any meaningful improvement, which might be because the dataset already has well-structured features or because Class 1 makes up only about 14% of the data. Regularization tends to help more when there's overfitting or noisy data, which doesn't seem to be the case here. To boost performance for the minority class, especially recall, it's worth exploring methods like class weighting, resampling, or using SMOTE.

📊 **2d – Performance Comparison Table**

Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1 Score (Class 1)
Logistic Regression (None)	86.21%	51.6%	15.8%	24.2%
L1 (Lasso) Regularization	86.21%	51.6%	15.8%	24.2%
L2 (Ridge) Regularization	86.21%	51.8%	15.8%	24.2%

3. Support Vector Machine- SVM (linear, poly, RBF):

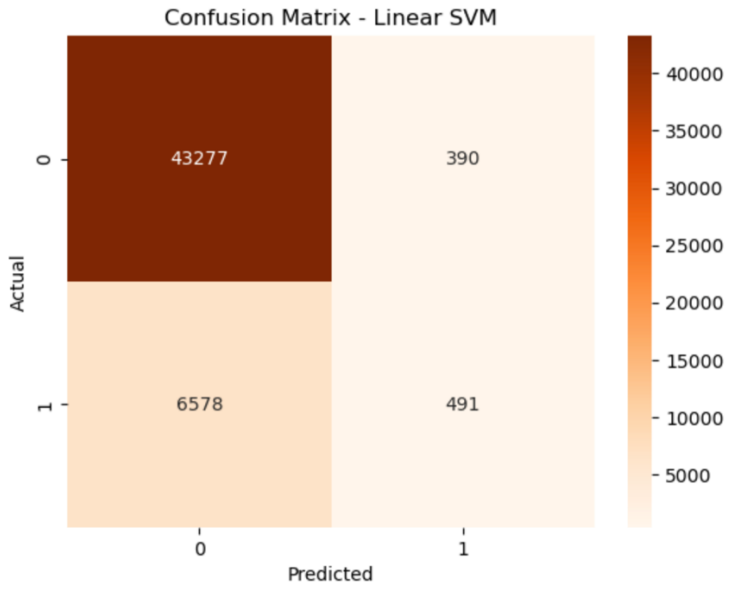
For this part of the assignment, we'll explore SVM models with different kernels. Since SVC with certain kernels can be slow, especially on large datasets, we'll use LinearSVC for part

3a as a faster alternative for the linear kernel. For part 3b, we'll go with SVC (kernel='poly') to test the polynomial kernel, and for part 3c, we'll use SVC(kernel='rbf') to evaluate the RBF kernel. Finally, in part 3d, we'll compare the results from all three and discuss how class imbalance affects their performance.

3a: SVM with linear kernel

The Linear SVM model reached an accuracy of 86.26%, which is close to what we saw with logistic regression. However, when it comes to identifying positive diabetes cases (Class 1), it struggled even more—recall dropped to just 6.95%, with an F1-score of 12.35%. Most of the actual positive cases were misclassified as negatives, which really shows how much class imbalance is affecting the model's ability to catch those minority class instances.

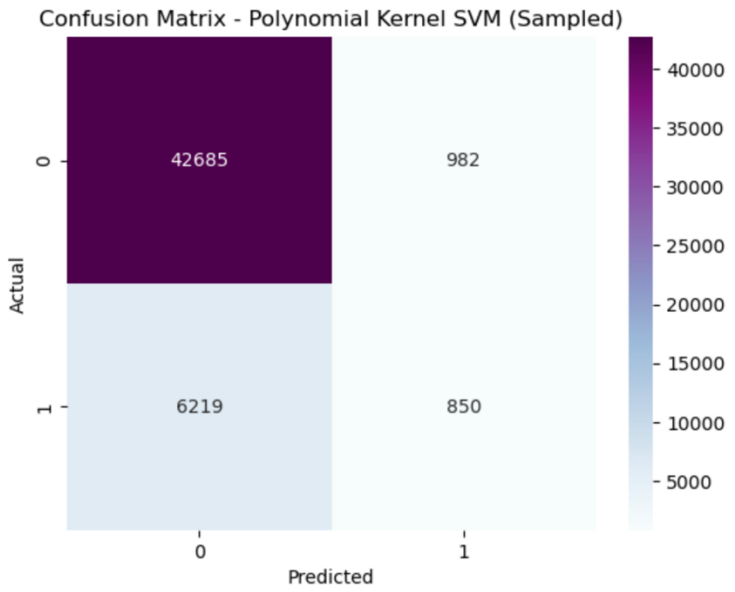
Results:
Accuracy: 0.8626616209397666
Precision: 0.5573212258796821
Recall: 0.06945819776488896
F1 Score: 0.12352201257861635



3b: SVM with Polynomial Kernel

polynomial kernel can be slow on large datasets. We started with a low degree to manage performance. It was too slow; we sampled the data.
Results of Sample 10,000 training examples:

The SVM with a polynomial kernel gave an accuracy of 85.81%, with a precision of 46.4%, recall of 12.0%, and an F1-score of 19.1% for Class 1. Compared to the linear SVM, it showed a slight improvement in recall and F1-score for the minority class, but overall performance is still weak. To manage the longer training time, we used a smaller subset of the data, which helped with speed, but the model continued to struggle with class imbalance, misclassifying most positive cases.
Results:
Accuracy: 0.8580692210659098
Precision: 0.46397379912663755
Recall: 0.12024331588626397
F1 Score: 0.19098977642961465

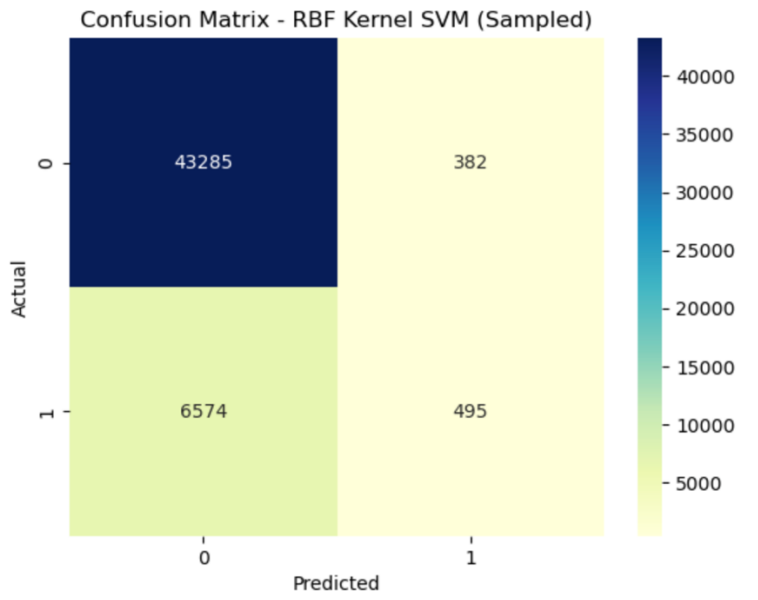


3c: SVM with RBF kernel (RBF Kernel SVM (Sampled Data)

The RBF kernel SVM reached an accuracy of 86.29%, with a precision of 56.4%, recall of just 7.0%, and an F1-score of 12.5% for Class 1. Even though the RBF kernel is more flexible and powerful in capturing non-linear patterns, its performance ended up being pretty like both the linear and polynomial kernels. Once again, the very low recall highlights how much class imbalance is hurting the model—most of the positive cases were still misclassified.

Results:

Accuracy: 0.8628981393882056
Precision: 0.56442417331813
Recall: 0.07002404866317725
F1 Score: 0.12459098917694438



3d: Comparison:

Among the three SVM models, the linear kernel was the fastest to train but had the weakest recall for detecting positive diabetes cases. The polynomial kernel performed slightly better in terms of recall and F1-score, though it took longer to train and still didn't give strong results. The RBF kernel had the best precision out of the three but didn't do much better at identifying positives. Across the board, class imbalance continued to be a major challenge, as all models consistently struggled to catch the minority class.

📊 3d – Kernel Comparison Table (Summary)

Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1 Score (Class 1)
Linear SVM	86.26%	55.7%	6.95%	12.4%
Polynomial SVM	85.81%	46.4%	12.0%	19.1%
RBF SVM	86.29%	56.4%	7.0%	12.5%

Question 3 Conclusion:

Because polynomial and RBF kernels can take a lot of time, we used only 10,000 samples from the training data for them. we evaluated all models using accuracy, precision, recall, and F1 score.

Among the three models, the **linear SVM** was the fastest to run, but it faced challenge the most with recall, it missed a lot of actual diabetes cases.

The **polynomial kernel** showed a bit better performance in terms of recall and F1 score, but it was slower to train.

All three models showed similar challenges in handling the class imbalance. Even though the flexible kernels (like polynomial and RBF) are more powerful in theory, they didn't make a big difference here, probably because detecting or identifying rare cases like diabetes needs more than just a powerful model, like balancing the classes or changing the threshold. So, while SVMs worked decently, the low recall reminds us we might need to explore more techniques to improve detection of positive cases.

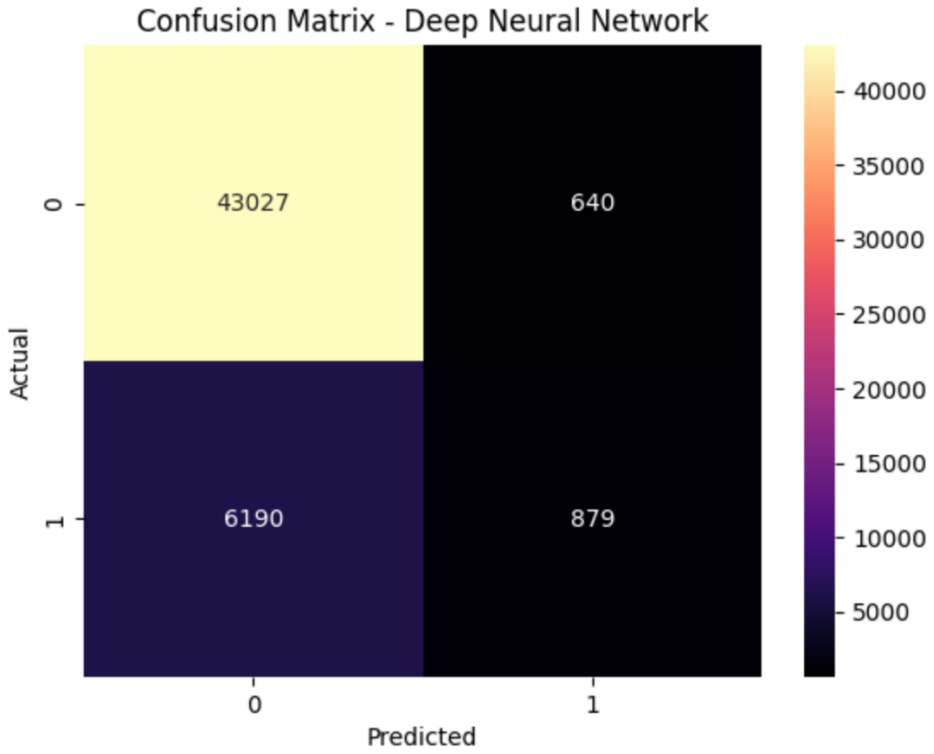
Using more data (20,000 instead of 10,000) gives the model a **little better chance to learn patterns**, in some minority class (people with diabetes). This is helpful because it reduces the risk of overfitting, especially for more complex kernels like polynomial and RBF. But Training time and memory usage will **increase**, especially with polynomial and RBF kernels, which are computationally heavier, and it can make our model non efficient regarding of The improvement in performance is **not dramatic**.

4. Neural Networks

4a: Define the Simple NN (Model A) and then Evaluate Model A on Test Set
Build Model B (Deeper NN + Dropout) and then Evaluate Model B on Test Set

We designed two neural network models: a simple feedforward network with one or two hidden layers, and a deeper version with more layers and dropout for regularization. For the simpler model (Model A), we got an accuracy of 86.5%, with a precision of 56.6%, recall of 13.8%, and an F1-score of 22.2% for the positive class. While these numbers aren't drastically better, the recall and F1-score were slightly higher than in previous models, suggesting that even a basic neural network can pick up on some of the non-linear patterns in the data. Still, the overall performance is clearly held back by the class imbalance issue.

Results:
Accuracy: 0.8653815830968149
Precision: 0.5786701777485188
Recall: 0.12434573489885416
F1 Score: 0.20470423847228691



4b: Model B (Deep NN + Dropout)

Model B, the deeper neural network with dropout, performed similarly to the simpler model, with an accuracy of 86.53%. It showed a slight boost in precision (57.9%) and a modest F1-score of 20.5%, but recall stayed almost the same at 12.4%. The use of dropout likely helped reduce overfitting, but just like with the previous models, the persistent class imbalance continued to limit the model's ability to effectively identify positive cases.

📊 Comparison Table for 4b:

Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1 Score (Class 1)
Simple NN (1 layer)	86.51%	56.6%	13.8%	22.2%
Deep NN (3+ layers)	86.53%	57.9%	12.4%	20.5%

5.Test with Different Cross-Validation Folds

5a: In this part we Implemented k-fold cross-validation on Logistic regression and SVM models with different values of k (e.g., 5, 10) and observations are:
Results:
Logistic Regression - Stratified 5-Fold F1 Scores: [0.24637053 0.23535887 0.24409449 0.23860885 0.24282415]
Mean F1 (Stratified): 0.24145137820671447

Logistic Regression - Non-Stratified 5-Fold F1 Scores: [0.25337282 0.23204787 0.2386376 0.2459695 0.23429579]
Mean F1 (Non-Stratified): 0.24086471553919467

5a and 5b: Logistic Regression

Fold Type || F1 Scores || Mean F1
Stratified K=5 || 0.246, 0.235, 0.244, 0.239, 0.243 || 0.2415
Non-Stratified || 0.253, 0.232, 0.239, 0.246, 0.234 || 0.2409
.
Stratification ensures each fold has a similar class distribution. Without stratification, class imbalance may be unevenly spread, affecting performance slightly. In this case, both strategies gave very similar mean F1, but stratified folds are more reliable for imbalanced datasets like this one.

Fold Type	F1 Scores	Mean F1
Stratified K=5	0.246, 0.235, 0.244, 0.239, 0.243	**0.2415**
Non-Stratified	0.253, 0.232, 0.239, 0.246, 0.234	**0.2409**

5a and 5b: Linear SVM

k-fold cross-validation for SVM using LinearSVC (fast and efficient for large sets).

Results:
Linear SVM - Stratified 5-Fold
F1 Scores: [0.12416404 0.11766198 0.12526834 0.12044213 0.12612613]
Mean F1 (Stratified): 0.12273252213629629

Linear SVM - Non-Stratified 5-Fold
F1 Scores: [0.12461696 0.11969504 0.119 0.12427086 0.12474645]
Mean F1 (non-stratified): 0.12246586212120585

For the Linear SVM, we tested both stratified and non-stratified 5-fold cross-validation. The F1 scores were very close in both cases—stratified folds gave a mean F1 of 0.1227, while non-stratified folds resulted in 0.1225. Although the overall performance was nearly identical, the stratified version showed slightly more consistent scores across the folds. This makes sense, especially given the class imbalance in our diabetes dataset, where maintaining the class distribution in each fold helps ensure more reliable evaluation.

Fold Type	F1 Scores	Mean F1
Stratified K=5	0.124, 0.118, 0.125, 0.120, 0.126	**0.1227**
Non-Stratified	0.125, 0.120, 0.119, 0.124, 0.125	**0.1225**

So, In this part, we tested how Logistic Regression and Linear SVM models perform using 5-fold cross-validation. We tried two versions of it: one where the data was split randomly (non-stratified), and one where we made sure each split had about the same number of diabetic and non-diabetic cases (stratified). This is important because our dataset is imbalanced, there are many more people without diabetes than with it, so random splits could end up with some folds having almost no diabetic cases at all, which would make the model look better than it actually is.
By comparing the F1 scores from both versions, we noticed that the stratified method gives more stable and consistent results. It better reflects how the model will actually perform on real-world data where we want to detect the less common class (people with diabetes). The non-stratified results, while similar, can be less reliable when dealing with this kind of class imbalance. Overall, this shows that stratified cross-validation is a better choice when working with datasets where one class is much smaller than the other.

Conclusion:

In this project, we explored different machine learning models,logistic regression, SVMs with different kernels, and neural networks, to predict diabetes from health indicators. While most models achieved high accuracy, they consistently struggled with recall due to class imbalance. SVMs with more complex kernels and deeper neural networks showed slight improvements, but not enough to significantly make performance better on the minority class. We also compared cross-validation strategies and found that stratified folds provided more reliable results, especially in imbalanced data. Overall, the analysis highlights the importance of not just choosing powerful models, but also addressing class imbalance with smarter sampling(regarding to usage) or evaluation methods to better identify at risk individuals for better analysis.