Data:
There is strong imbalance between positive and negative class, 20% to 80%.
Some exploratory visualization were ran to reveal patterns, and hidden connections; explained in the EDA part of the code.

Features:
- A heat map revealed there's isn't multicollinearity in the data, meaning applying PCA is not appropriate to reduce dimensionality here, because there's no linear correlations, and the features are already almost orthogonal.
- Moreover, there's no significant quadratic relationship between any of the features and the target variable, as revealed by adding polynomial features and after examining the correlations (2nd degree of each feature, and their interactions), meaning isn't helpful to use those either as strong predictors of the target variable.
- A histogram of all numerical variables showed they are all roughly Gaussian, expect for 'x45' which took the same range of values for all the cases (an interval of very small neighborhood of zero). Such variables are safe to drop because they carry the same value regardless of anything else. An examination of its correlation with the other features and the target variable also confirmed it didn't carry much explanatory weight.

There was no distinct outliers.

Initial models tested on a training dataset:
Naïve Bayes models, Random Forests, Boosted Random Forests (XGBoost), Support Vector Machine Classifier. In addition to Stochastic Gradient Descent, and K-Nearest Neighbors.
I also tried a neural network with Keras, because I wanted to give it a shot, it was fast enough. It scored badly on AUC.
Note:
I didn't think that the last two would outperform SVC or boosted Random Forests, and they didn't. Although initially, for default hyper-parameters, KNN did better than XGBoost, after grid searching on KNN, it turned out best parameters are the default ones (5 neighbors and uniform weights), and gave 91.11% accuracy, and 0.791 AUC score. While SGD was 85.22% accurate at best, with 0.741 AUC score.
Meanwhile, XGBoost, increased accuracy by approx. 7% after tuning.

Why I thought XGBoost and SVM would do best?
We have very imbalanced classes in the data. The problem with that, positive class won't be picked up in predictions. Best techniques to remedy that are using SVM and Boosted Random Forests. Other techniques include up-sampling the minority class, or down-sampling the majority class. I'm not in favor of either, in the first, we are faking data, and in the second we are omitting otherwise useful information.
- *SVM* is a penalizing learning algorithm that increase the cost of classification mistakes on the minority class. During training, we can use the argument class_weight='balanced' to penalize mistakes on the minority class by an amount proportional to how under-represented it is.
- *XGBoost*: Because Decision Trees often perform well on imbalanced datasets, as their hierarchical structure allows them to learn signals from both classes. Boosted ensemble of Decision Trees always outperforms single one. Moreover, in boosted trees, one can select the number of features to be used in the learning, which is great instead of manually searching for the best group of features to use.

In our case though, all features did the best.

Comparison between the SVC and XGBoost:
SVC is very slow, and thus costly. It is unwieldy to GridSearch on it to fine-tune it. But it is the only algorithm I know that aims to classify observations by mapping them to higher dimensions, which is neat, and in this case, amazingly good. No wonder it was the state of the art in ML not long ago.
XGBoost is usually my trusted tool, it is awesome for imbalanced data, it is very versatile, customizable, and truly learn from the data, thus robust enough to noise, and one can select number of features to use to reduce complexity. Boosting tackles overfitting by introducing bias, making it almost always the best model. Plus, it is faster than SVM. Fine-tuning XGBoost model went a long way enhancing performance.

A word on the performance metric:
Because of the imbalance of the dataset, accuracy would be a misleading metric for performance, and the neural network model confirmed that.
AUC is the right metric to use in this case, because it shows how good the model was in correctly identifying the actual positives.

Last thought
I highly appreciate your feedback, specially if you correct me on my logic, teach me a better way of doing things, or teach me something new.
Thank you for your time reviewing my submission. I hope it was worthwhile.