As an analyst for a popular hospital chain with patients in almost every state in the United States, I am investigating the extent to which readmission is a problem for our chain of hospitals. I am researching this for business as external organizations are penalizing hospitals for excessive readmission, and these penalties have increased every year since its inception. As mentioned in the data dictionary, 78% of hospitals were fined in 2015 with fewer than 1 in 5 hospitals utilizing technology specific to readmission reduction. The stated purpose of my analysis from the stakeholders is to predict readmission based on other conditions and factors of the patient. My real-world organizational question is, “What factors related to the patient are causing readmissions?”. The goal is to understand what factors related to the patient are causing the readmissions so we can effectively analyze ways to stem readmissions and serve our patients more effectively.

I am choosing to use Gradient Boost for my analysis. Gradient Boosting can handle many different variables, as we see in our data, from continuous to ordinal and binary categorical variables. This dataset includes many variables that add complexity, and the analysis results are important for the business to stem readmission and prevent penalties, so using the most effective tool is important. Out of the choices of AdaBoost, Random Forest, and Gradient Boost, the most accurate ML model to use is Gradient Boosting. This performance and accuracy will come at a process time cost. This cost is worth it as we desire accuracy from the results to increase efficiency for our patients and reduce costs (Tuychiev, 2023).

Gradient Boost is a boosting algorithm that serves as an ensemble method (GeeksforGeeks, 2023a). This ensemble method is applied to decision trees ass the other 2 choices (Simic, 2025). This algorithm works by using weak learners, which are underpowered/simple regressions. Our first weak learner will make mistakes so that the next weak learner can learn from their mistakes to become stronger and reduce errors. This process will continue with each new weak learner learning from the previous weak learners. The weak learners learn via the gradient to decrease the loss function to have a more educated model (RitVikMath, 2021). For further clarification, the first weak learner is trained by predicting the mean of the target variable for all observations, while the rest of the weak learners build from this, focusing on correcting the errors of the prior weak learners, while incorporating more variables (GeeksForGeeks, 2023b).

There were different packages and libraries I decided to use for this project. Some of these have already been used in our previous work and were an easy addition to my list. These included pandas for our basic needs like creating dataframes and saving csv files, matplotlib.pyplot and seaborn for plotting our visuals, and variance\_inflation\_factor and add\_constraint for dealing with multicollinearity and adding a constant in our models. I have included sklearn.model\_selection for splitting the data, hyperparameter tuning, cross validation, sklearn.metrics to get all more metrics that are required for the accuracy, precision, recall, F1, and Area Under Curve scores, and sklearn.preprocessing for encoding my ordinal, nominal, and binary categorical variables (Tuychiev, 2023). The last important library I used was xgboost for my gradient boosting as this algorithm uses advanced optimization and regularization to reduce the overfitting that we need to be aware of and prepare for with gradient boosting (GeeksforGeeks, 2025).

One data preprocessing technique I used was removing unnecessary redundant rows. The first removal was with columns that held no value like ‘CaseOrder’, ‘Customer\_id’, ‘Interaction’, and ‘UID’. The second part I noticed TotalCharge and Additional\_Charges that were not directly tied factors of customers, so I removed those. Lastly, I noticed redundant location data, so I removed every location datapoint, but area. I felt like this would give a great idea of 3 common key variables tied to location that can show differences but won’t create a large number of columns by keeping a value like city or state.

After the removal of these variables, I had my complete dataset for the initial analysis. The dataset was diverse with continuous variables, binary categorical variables, nominal variables, and ordinal variables. The continuous variables were age, income, doc\_visits, full\_meals\_eaten, initial\_days, children, and vitD\_support. The binary categorical features were highblood, stroke, overweight, arthritis, diabetes, hyperlipidemia, backpain, anxiety, allergic\_rhinitis, reflux\_esophagitis, asthma, and soft drink. The nominal categorical features are gender, job, marital, area, and services. The ordinal categorical features are complication\_risks, initial\_admin, and item1-item8.

There were multiple steps I took to prepare the data. The first was obviously creating the dataframe from the csv data, dropping unnecessary columns, and cleaning whitespace and commas from the job names for consistency. I removed redundant columns for a reduction in multicollinearity and columns not directly tied to patients. Encoding the variables to numeric values was next for XGBoost to work accurately. Moving forward it was splitting the data for testing and validation prior to running my gradient boosting (XGBoost).

**Initial Model**

**A graph with numbers and a bar

AI-generated content may be incorrect.**

**Hyperparamter Tuning K-Fold Validation Results**

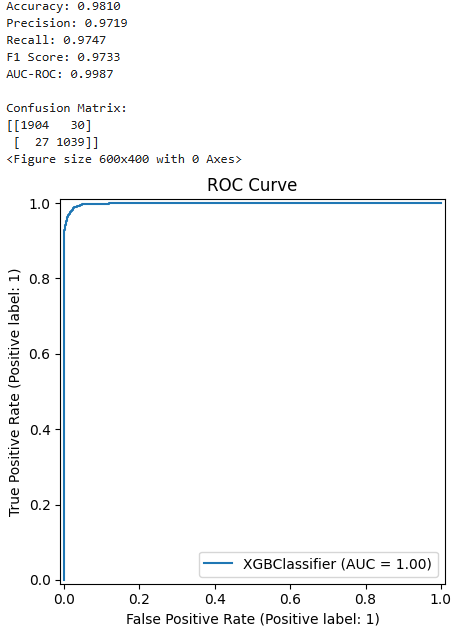
**A white background with black text

AI-generated content may be incorrect.**

**Cross Validation**

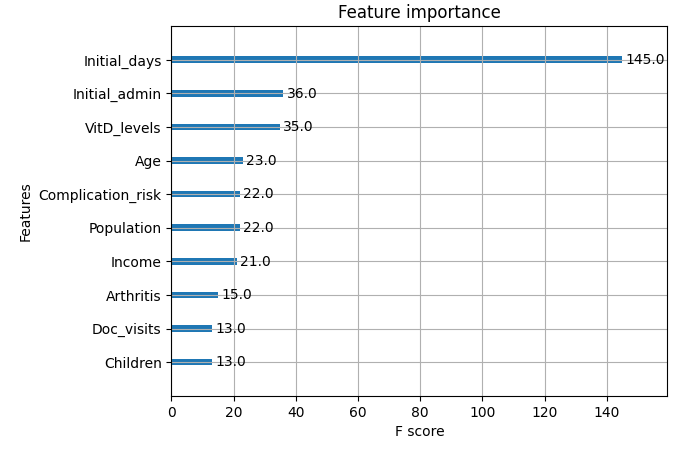
****

**Best Model**

****

I have shared screenshots of my initial model metrics, hyperparameter tuning using k-fold validation results, cross-validations results, and the optimized model metrics as seen above. The hyperparameters I chose for RandomSearchCV was learning\_rate, n\_estimators, max\_depth, subsample, colsample\_bytree, gamma, min\_child\_weight, and scale\_pos\_weight. Learning\_rate was used to find the best boosting for each step to reduce overfitting. N\_estimators is the number oftrees used in the model. Max\_depth is the maximum number of trees. Subsample is the fraction of the samples and the colsample\_bytree is the fraction of columns used for each tree. Gamma is the minimum loss reduction.Min\_child\_weight\_weight is the minimum total of instance weight in a child. Scale\_pos\_weight is used for handling class imbalance and adjusts weight of positive class. These parameters were all chosen to help improve accuracy, while also balancing computational costs, and preventing overfitting and underfitting (*XGBoost Parameters — Xgboost 2.1.3 Documentation*, n.d.). I also used verbose=3 for debugging (*XGBoost Parameters — xgboost 2.1.3 documentation*. (n.d.), n\_iter 50 to limit the number of iterations for computational cost while still getting a significant sample size, cv for the 5 folds, scoring on accuracy for interpretation, n\_jobs=-1 for the iteration process to use the maximum number of jobs to be run, and random\_state to accept a random integer at random. I had tried GridSearch prior to RandomSearch and the computational cost was too significant, so all these choices were made with computational limitations first from my machine in mind and accuracy second (Hassan, 2023). Luckily, this paid off as the model had significantly high accuracy, while limited computational cost.

The metrics for the optimized model in comparison to the initial model shows a boost in performance. Accuracy increased from 98.10% to 99.79% demonstrating the performance of predictions that the model gets correct increased. The precision increased from 97.64% to 99.61%, demonstrating a strong proportion of positive predictions that are correct. Recall increased from 97% to 99.81%, which demonstrates the proportion of accurate actual positives. The F1 Score increased from 97.32% to 99.71% demonstrating that the average of precision and recall is strong and balanced. The Area Under Curve – Receiver Operating (AUC-ROC) increased from 99.89% to 99.99%, demonstrating high classifier performance. These statistics demonstrate a strong optimization, for an already strong model.



The results of my analysis have answered my business question of what factors related to patients are leading to readmissions. As seen from my graph of feature importance we can see the top 10 variables causing the readmits. Initial days, initial admin, vitamin D levels, complication risk, arthritis, age, and doctor visits are all not surprising at all, as a person who has spent a significant number of days in the initial admission, someone who was admitted for emergency, has low vitamin D, visit the doctor often, and those who have complication risks, are older, and have health issues would logically be more inclined to be readmitted. The income and children I believe could be connected and deserve deeper analysis to understand, if those who have higher income and less children are more likely to visit as they can afford to, or if those who have less income and more children are more likely to visit as they are more concerned for their family’s help and their income negatively impacted choices in their first admission. The population is also another variable where we can infer a few different things, like if a highly populated hospital has less doctor to patient ratio where they are unable to give sufficient time to the patients or even a lower population area is understaffed and causing for insufficient time for patients.

My personal recommendation is to now analyze these variables impacts on readmissions. Gain more details of how they are impacting readmissions, so we can continue to move forward in our goal of stemming penalties and providing the best service to our patients. While we have all the variables that are impacting readmissions, we do not fully understand how and why they are impacting readmissions. Gathering more details and completing further analysis can bring us the results we need to make the best data-driven decisions moving forward.

I will say there is a limitation to my analysis. I could have used a more significant hyperparameter method like GridSearch vs the RandomSearch I used. This decision was made for computational costs and time. It does appear however that the model is still very powerful with its metrics. After using cross validation and seeing a mean cross-validation accuracy of 98.71%, I felt very confident in my model. To further add to this confidence when testing the best model to make predictions we only seen a minimal drop in our metrics as accuracy was at 98.10%, Precision was 97.19%, recall was 97.74%, F1 score was 97.33%, and AUC-ROC was 99.87%. All these metrics showing high performance is a demonstration that the results we have obtained are reliable in moving forward on our understanding.

References:

GeeksforGeeks. (2023a, March 31). *Gradient boosting in ML*. GeeksforGeeks. <https://www.geeksforgeeks.org/ml-gradient-boosting/>

GeeksforGeeks. (2023b, May 29). *GradientBoosting vs AdaBoost vs XGBoost vs CatBoost vs LightGBM.* GeeksforGeeks. <https://www.geeksforgeeks.org/gradientboosting-vs-adaboost-vs-xgboost-vs-catboost-vs-lightgbm/>

GeeksforGeeks. (2025, February 12). *ML | XGBOOST (eXtreme Gradient Boosting).* GeeksforGeeks. <https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/>

Hassan, M. H. (2023, November 25). *Tuning model hyperparameters with random search - M. Hammad Hassan - Medium*. Medium. <https://medium.com/@hammad.ai/tuning-model-hyperparameters-with-random-search-f4c1cc88f528>

Malik, S. C. (2024, November 21). *Is xgboost immune to multicolinearity? - Sebastian Cabrera Malik - Medium*. Medium. <https://medium.com/@sebastian.cabrera-malik/is-xgboost-inmune-to-multicolinearity-4dd9978605b7>

*python xgboost continue training on existing model.* (n.d.). Stack Overflow. <https://stackoverflow.com/questions/47000253/python-xgboost-continue-training-on-existing-model>

RitVikMath. (2021, September 29). *Gradient boosting: data science’s silver bullet* [Video]. YouTube. <https://www.youtube.com/watch?v=en2bmeB4QUo>

Simic, M. (2025, February 13). *Gradient Boosting Trees vs. Random Forests.* Baeldung. <https://www.baeldung.com/cs/gradient-boosting-trees-vs-random-forests>

Singh, V. (2024, October 18). *Variance Inflation Factor (VIF): Addressing Multicollinearity in Regression Analysis*. DataCamp. <https://www.datacamp.com/tutorial/variance-inflation-factor>

*Tuning XGBoost Hyperparameters with RandomizedSearchCV*. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/69786993/tuning-xgboost-hyperparameters-with-randomizedsearchcv>

*Tuning XGBoost Hyperparameters with RandomizedSearchCV.* (n.d.-b). Stack Overflow. <http://stackoverflow.com/questions/69786993/tuning-xgboost-hyperparameters-with-randomizedsearchcv>

Tuychiev, B. (2023, December 27). *A Guide to The Gradient Boosting Algorithm.* DataCamp. <https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm>

*XGBoost Parameters — xgboost 2.1.3 documentation*. (n.d.). <https://xgboost.readthedocs.io/en/stable/parameter.html>

*XGBoost Robust to Correlated Input Features (multi-collinearity) | XGBoosting*. (n.d.). <https://xgboosting.com/xgboost-robust-to-correlated-input-features-multi-collinearity/>