Assessment: Predicting Length of Stay for Hospitalised Covid-19 Patients Using Random Forest Machine Learning

B273025

2024-11-16

# 1 Introduction

The Covid-19 Pandemic placed unprecedented strain on health systems, making resource allocation essential to prevent services from being overwhelmed. Predicting hospital demand, including bed occupancy, staffing needs and equipment requirements, became critical for informed decision-making(1).

Hospital demand depends on two factors:

1. The number of individuals requiring hospitalisation, estimated using epidemic curves.
2. The length of hospital stay (LOS), derived from patient observations.

This report develops a model to predict LOS for hospitalised Covid-19 patients based on individual factors. Accurate LOS predictions support:

1. Optimised resource allocation
2. Improved patient care and logistic planning
3. Enhanced pandemic preparedness

#Load necessary libraries:  
library(sparklyr)  
library(dplyr)  
library(purrr)  
library(ggplot2)  
library(broom)  
library(gt)  
library(knitr)  
library(corrr)  
library(kableExtra)

#Connect to spark and read in data:  
# connects to local machine as if it were a cluster   
sc = spark\_connect(master = 'local')  
# load the dataset into spark (sdf = spark dataframe)   
sdf\_covid\_hosp <- spark\_read\_csv(sc, 'host\_train.csv')

# 2 Exploratory Data Analysis

# Define a function to summarise the spark dataframe (sdf) efficiently  
# note this function is applicable in a big data setting where there are a manageable number of columns (as is the case with this dataset)  
  
summarise\_big\_data <- function(sdf, cardinality\_limit = 15) {  
 # Retrieve schema to extract column names  
 schema <- sdf\_schema(sdf)  
   
 # Iterate over schema to compute metrics for each column  
 metrics <- purrr::map\_dfr(schema, ~ {  
 col\_name <- .x$name # Extract column name  
 # Compute cardinality and missing percentage in Spark  
 summary\_spark <- sdf %>%  
 summarise(cardinality = approx\_count\_distinct(!!sym(col\_name)), # Approximate distinct count for efficiency  
 nans\_pct = (sum(ifelse(is.na(!!sym(col\_name)), 1, 0)) / n()) \* 100) %>% # Percent missing  
 collect() # Collect results into R  
   
 # Extract cardinality and missing percentage from the collected summary  
 cardinality <- summary\_spark %>% pull(cardinality)  
 nans\_pct <- summary\_spark %>% pull(nans\_pct)  
   
 # If cardinality is low, extract sample categories in Spark  
 categories\_str <- if (cardinality < cardinality\_limit) {  
 sdf %>%  
 select(!!sym(col\_name)) %>%  
 distinct() %>%  
 head(cardinality\_limit) %>%  
 collect() %>%  
 pull(!!sym(col\_name)) %>%  
 paste(collapse = ", ")  
 } else {  
 NA} # Skip for high-cardinality columns  
   
 # Combine results into a data frame row  
 tibble(  
 feature = col\_name,  
 cardinality = cardinality,  
 nans\_pct = nans\_pct,  
 categories = categories\_str)})  
 return(metrics)}  
  
# Apply the function to the Spark DataFrame  
ldf\_summary\_table <- summarise\_big\_data(sdf\_covid\_hosp)  
  
# Present summary in a table   
ldf\_summary\_table <- ldf\_summary\_table %>%   
 select(feature, categories, cardinality, nans\_pct) %>% # order columns for table  
 gt(rowname\_col = "feature") %>% # use feature as row names  
 tab\_header(title = "Summary of Covid-19 Hospital Admissions Data",  
 subtitle= "") %>%   
 cols\_label(feature = "Feature",  
 categories= "Unique Categories",  
 cardinality="No. of Unique Categories",  
 nans\_pct= "% Missing Data")%>% # rename columns to make reader-friendly  
 tab\_row\_group(label = "Categorical Features",  
 rows = c("Department", "Ward\_Type", "Ward\_Facility", "Bed\_Grade","Type\_of\_Admission","Illness\_Severity", "Age", "Stay\_Days")) %>%  
 tab\_row\_group(label = "Numerical Features",  
 rows = c("Hospital", "Hospital\_type", "Hospital\_city", "Hospital\_region", "Available\_Extra\_Rooms\_in\_Hospital", "City\_Code\_Patient", "Patient\_Visitors", "Admission\_Deposit" )) %>%   
 tab\_row\_group(label = "Unique Identifiers",  
 rows = c("case\_id", "patientid")) %>%   
 cols\_align(align = "center", columns = c(nans\_pct,cardinality)) %>%  
 fmt\_number(columns = nans\_pct, decimals = 2) %>% # format percentage column  
 text\_transform(locations = cells\_body(columns = nans\_pct, rows = nans\_pct > 0),  
 fn = function(x) { paste0(x, "!!")}) %>% # Add a warning symbol for non-zero values in the % missing data  
 opt\_stylize(style = 6, color = "blue")  
  
ldf\_summary\_table

Table : Summary of Covid-19 Hospital Admissions Data

|  | Unique Categories | No. of Unique Categories | % Missing Data |
| --- | --- | --- | --- |
| Unique Identifiers | | | |
| case\_id | NA | 342348 | 0.00 |
| patientid | NA | 90280 | 0.00 |
| Numerical Features | | | |
| Hospital | NA | 32 | 0.00 |
| Hospital\_type | 6, 5, 1, 3, 2, 4, 0 | 7 | 0.00 |
| Hospital\_city | 13, 6, 9, 5, 10, 3, 1, 11, 7, 2, 4 | 11 | 0.00 |
| Hospital\_region | 1, 2, 0 | 3 | 0.00 |
| Available\_Extra\_Rooms\_in\_Hospital | NA | 18 | 0.00 |
| City\_Code\_Patient | NA | 35 | 1.42!! |
| Patient\_Visitors | NA | 28 | 0.00 |
| Admission\_Deposit | NA | 6771 | 0.00 |
| Categorical Features | | | |
| Department | TB & Chest disease, anesthesia, gynecology, radiotherapy, surgery | 5 | 0.00 |
| Ward\_Type | P, Q, R, S, T, U | 6 | 0.00 |
| Ward\_Facility | A, B, C, D, E, F | 6 | 0.00 |
| Bed\_Grade | 4, 1, NA, 3, 2 | 4 | 0.04!! |
| Type\_of\_Admission | Emergency, Trauma, Urgent | 3 | 0.00 |
| Illness\_Severity | Extreme, Minor, Moderate | 3 | 0.00 |
| Age | 0-10, 11-20, 21-30, 31-40, 41-50, 51-60, 61-70, 71-80, 81-90, 91-100 | 10 | 0.00 |
| Stay\_Days | 0-10, 11-20, 21-30, 31-40, 41-50, 51-60, 61-70, 71-80, 81-90, 91-100, More than 100 Days | 10 | 0.00 |

The dataset used for this model is publicly available and includes information on hospital characteristics, patient demographics, and admission details for Covid-19 inpatients. The target variable, Stay\_Days, is categorical, and therefore requires a classification model. The dataset contains 17 additional variables, comprising both numerical and categorical variables. High-cardinality identifiers (case\_id and patient\_id) were excluded from modelling as they lack predictive value.

The dataset is largely complete, with missing values in Bed\_Grade (n=113) and City\_Code\_Patient (n=4532), both under 2%. These rows were removed to prevent bias and avoid introducing noise from potentially erroneous data.

# clean data by removing missing values in Bed\_Grade and City\_Code\_Patient  
sdf\_covid\_hosp <- sdf\_covid\_hosp %>%   
 filter(!is.na(Bed\_Grade) & !is.na(City\_Code\_Patient))

# Distribution of Stay\_Days (target variable)  
dist\_stay\_days <- sdf\_covid\_hosp %>%  
 group\_by(Stay\_Days) %>%   
 summarise(count = n()) %>%   
 collect() %>%  
 ggplot(aes(x = Stay\_Days, y = count)) +  
 geom\_col(fill = "blue") +  
 theme\_minimal()+  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))+ # Rotate x-axis labels  
 labs(title = "Distribution of Days Spent in Hospital with Covid-19 Infection", x = "Days spent in Hospital", y = "Number of Patients")  
  
dist\_stay\_days

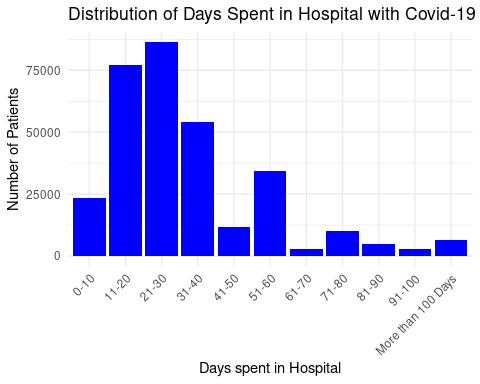


Figure 1: Distribution of Days Spent in Hospital due to Covid-19 infection

Figure 1 shows the distribution of Stay\_Days (LOS). Most patients stayed 11–30 days, with the highest frequency in the 21–30 day range. The distribution is right-skewed, highlighting an imbalance in the data, with stays beyond 40 days being under-represented. This limits the model’s ability to accurately predict these categories accurately.

To address this, stays over 40 days were combined into a single “above 40 days” category, improving accuracy from 27.4% to 33.2%. However, category imbalance likely still impacts performance, as discussed in section 4.

# Explore relationships between features  
# most important feature in model\_v1 was Patient\_Visitors  
# Compute boxplot statistics for Patient\_Visitors by Stay\_Days in Spark  
ldf\_visitors\_boxplot\_stats <- sdf\_covid\_hosp %>%  
 group\_by(Stay\_Days) %>%  
 summarise(  
 q1 = percentile\_approx(Patient\_Visitors, 0.25), # calculate approximate percentiles on spark dataframe   
 median = percentile\_approx(Patient\_Visitors, 0.5),  
 q3 = percentile\_approx(Patient\_Visitors, 0.75),  
 min = min(Patient\_Visitors),  
 max = max(Patient\_Visitors)  
 ) %>%  
 collect() # collect to R dataframe   
  
# Visualise the boxplot using pre-computed stats  
corr\_visitors <- ggplot(ldf\_visitors\_boxplot\_stats, aes(x = as.factor(Stay\_Days))) +  
 geom\_boxplot(  
 aes(ymin = min, lower = q1, middle = median, upper = q3, ymax = max),  
 stat = "identity",  
 fill = "lightblue") +  
 labs(title = "Number of Visitors by Stay Days",  
 x = "Stay Days",  
 y = "Number of Visitors") +  
 theme\_minimal()+  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1)) # Rotate x-axis labels  
  
corr\_visitors

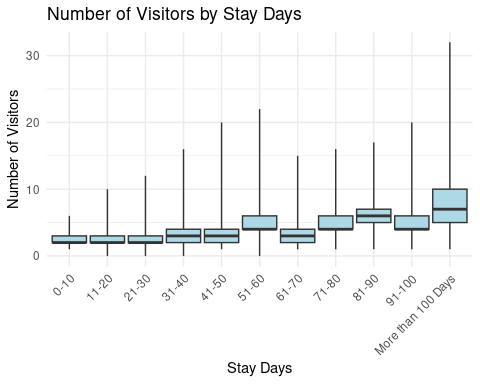


Figure 2: Relationship between the number of visitors and the length of stay in hospital

The preliminary model identified Patient\_Visitors as the most important feature for predicting LOS. Figure 2 shows a positive relationship between Patient\_Visitors and LOS, indicating it is a dependent variable and unsuitable for inclusion in the model.

Additionally this feature highlights the importance if designing the model for real-world use by incorporating only variables known at admission. Including retrospective variables like Patient\_Visitors undermines the model’s applicability for real-time decision-making.

# 3 Fit a Model to the Dataset

The outcome variable, Stay\_Days, is categorical, requiring a classification model. Literature suggests Random Forest performs best for predicting hospital length of stay due to Covid-19 (2,3). This supervised algorithm uses an ensemble of decision trees created through bootstrap aggregating (bagging), where subsets of data are sampled with replacement to train individual trees. Predictions are then averaged (for regression) or aggregated by majority vote (for classification), reducing overfitting and improving accuracy.

Random Forest suits this dataset as it handles both numerical and categorical features without requiring one-hot encoding or extensive preprocessing. It is robust to high-dimensional data, as it selects random subsets of features for splitting, making it less prone to overfitting compared to regression models. Additionally, it provides feature importance metrics, offering valuable insights into the key predictors of LOS.

However, Random Forest has limitations, including computational expense with large datasets and does not inherently provide insights into feature relationships. Despite these drawbacks, its ability to model complex, non-linear relationships and handle both categorical and numerical data efficiently makes it the best choice for this task.

## 3.1 Designing a Model

I sampled 10% of the dataset to reduce computational demand and combined stays over 40 days into a single category, ‘More than 40,’ to address the imbalance in the right-skewed Stay\_Days distribution. The dataset was split into 80% for training and 20% for testing, with the training set used to build the model and the test set for evaluation.

# Sample 10% of the data (as full dataset takes long time)  
sdf\_covid\_hosp\_sampled <- sdf\_covid\_hosp %>%  
 sdf\_sample(fraction = 0.1, replacement = FALSE, seed = 123)  
  
# combine stay\_days bucket for above 40 to improve usefulness of the model  
sdf\_covid\_hosp\_sampled <- sdf\_covid\_hosp\_sampled %>%  
 mutate(stay\_days\_mutated = case\_when(  
 Stay\_Days %in% c("0-10", "11-20", "21-30", "31-40") ~ Stay\_Days,  
 Stay\_Days %in% c("41-50", "51-60", "61-70", "71-80", "81-90", "91-100", "More than 100 Days") ~ "More than 40"))  
  
# Prepare data for model:   
# Apply StringIndexer to convert all categorical features into integers for random forest model and remove unnecessary columns:  
sdf\_covid\_hosp\_stay\_days\_map <- sdf\_covid\_hosp\_sampled %>%  
 ft\_string\_indexer(input\_col = "stay\_days\_mutated", output\_col = "stay\_days\_mutated\_index") %>%  
 ft\_string\_indexer(input\_col = "Department", output\_col = "Department\_index") %>%  
 ft\_string\_indexer(input\_col = "Ward\_Facility", output\_col = "Ward\_Facility\_index") %>%  
 ft\_string\_indexer(input\_col = "Ward\_Type", output\_col = "Ward\_Type\_index") %>%  
 ft\_string\_indexer(input\_col = "Bed\_Grade", output\_col = "Bed\_Grade\_index") %>%  
 ft\_string\_indexer(input\_col = "Type\_of\_Admission", output\_col = "Type\_of\_Admission\_index") %>%  
 ft\_string\_indexer(input\_col = "Illness\_Severity", output\_col = "Illness\_Severity\_index") %>%   
 ft\_string\_indexer(input\_col = "Age", output\_col = "Age\_index") %>%   
 select(-patientid, -case\_id, -Stay\_Days, -Department, -Ward\_Facility,-Ward\_Type, -Bed\_Grade, -Type\_of\_Admission, -Illness\_Severity, -Age, -Patient\_Visitors, -Department\_index,-Hospital\_region)   
# remove patientid and case\_ID from data set as they are not predictive features  
# remove columns with categorical data which have been encoded as integers \_index  
# remove Patient\_Visitors as this is a dependent variable, which is not useful in prediction of LOS  
# remove features with importance of <0.01 (Department\_Index, Hospital\_region) to simplify model   
# keep in stay\_days\_mutated to enable mapping of index to categories for interpretation of results later in report   
  
sdf\_covid\_hosp\_model <- sdf\_covid\_hosp\_stay\_days\_map %>%   
 select(-stay\_days\_mutated) # remove stay\_days\_mutated for model, as it is already encoded in stay\_days\_mutated\_index  
  
# # Persist transformed data to reduce memory and computation costs  
# sdf\_covid\_hosp\_model <- sdf\_covid\_hosp\_model %>%  
# sdf\_persist(storage.level = "MEMORY\_AND\_DISK")  
  
# Split into training and test sets for evaluation   
data\_splits <- sdf\_covid\_hosp\_model %>%  
 sdf\_random\_split(training = 0.8, testing = 0.2, seed = 123)  
  
train\_data <- data\_splits$training  
test\_data <- data\_splits$testing  
  
  
# Train a Random Forest classifier  
rf\_model <- train\_data %>%   
 ml\_random\_forest(stay\_days\_mutated\_index ~.,type = "classification", # categorical outcome   
 num\_trees = 100, # balance performance and computational cost  
 max\_depth = 10,  
 min\_instances\_per\_node = 5)  
  
# use ml\_predict() to create a Spark data frame that contains the predictions from the testing data set  
pred <- ml\_predict(rf\_model, test\_data)

## 3.2 Hyperparametric Tuning

Hyperparameter tuning tests combinations of parameters (e.g., num\_trees, max\_depth, min\_instances\_per\_node) to find the optimal model configuration. Below is an example, evaluating 27 combinations of hyperparameters.

# # Not inlcuded code as computer and noteable did not have the storage to execute.   
  
# # Define the pipeline for Random Forest  
# pipeline <- sc %>%  
# ml\_pipeline() %>%  
# ft\_r\_formula(stay\_days\_mutated\_index ~ .) %>% # Converts data into a formula-like structure  
# ml\_random\_forest\_classifier()  
#   
# # Define the grid of hyperparameters to test  
# grid <- list(random\_forest\_classifier = list(  
# num\_trees = c(50, 100, 150), # Test 3 values for number of trees  
# max\_depth = c(5, 10, 15), # Test 3 values for maximum tree depth  
# min\_instances\_per\_node = c(1, 5, 10))) # Test 3 values for minimum instances per node  
#   
# # Define the evaluator (accuracy metric for classification)  
# evaluator <- ml\_multiclass\_classification\_evaluator(sc, metric\_name = "accuracy") # measures accuracy of each model during cross-validation   
#   
# # Set up cross-validation  
# cv <- ml\_cross\_validator(x = sc,  
# estimator = pipeline,  
# estimator\_param\_maps = grid,  
# evaluator = evaluator,  
# num\_folds = 5, # 5-fold cross-validation  
# parallelism = 4) # Use 4 threads for parallel execution  
#   
# # Fit the cross-validation model  
# cv\_model <- ml\_fit(x = cv, dataset = train\_data)  
#   
# # Extract validation metrics  
# validation\_metrics <- ml\_validation\_metrics(cv\_model)  
#   
# # Display validation metrics  
# print(validation\_metrics)  
#   
# # Select the best model from cross-validation  
# best\_model <- ml\_best\_model(cv\_model)  
#   
# # Evaluate the best model on the test dataset  
# test\_predictions <- ml\_predict(best\_model, test\_data)  
# test\_metrics <- ml\_evaluate(best\_model, test\_data)  
#   
# # Display test metrics  
# print(test\_metrics)

# 4 Evaluation of the Model

## 4.1 Main Findings

# extract importance of each feature and remove features with importance <0.01 from model  
  
ldf\_importance <- ml\_feature\_importances(rf\_model) # ldf\_ = local dataframe  
  
ldf\_importance <- ldf\_importance %>%   
 ggplot(aes(x = reorder(feature, importance), y = importance)) +  
 geom\_bar(stat = "identity", fill = "blue") +  
 coord\_flip() +  
 labs(title = "Feature Importance", x = "Feature", y = "Importance")+  
 theme\_minimal()  
  
ldf\_importance

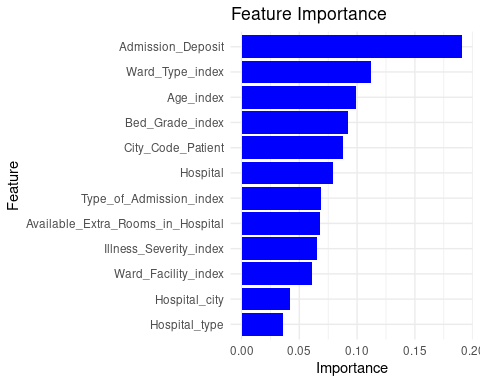


Figure 3: Ranked Importance of each Feature

Admission Deposit, Ward Type, Age, and City Code Patient were the most influential predictors of LOS. Hospital Department and Hospital Region (<0.01% importance) were excluded from the final model for simplicity and efficiency.

### 4.1.1 Model Performance Metrics:

1. Classification Model Metrics

# Define function to calculate metrics for prediction dataset   
calculate\_metric <- function(metric\_name)   
 {pred %>%  
 ml\_multiclass\_classification\_evaluator(  
 label\_col = "label",  
 prediction\_col = "prediction",  
 metric\_name = metric\_name)}  
  
# Create a dataframe and calculate metrics  
metrics\_df <- data.frame(  
 Metric = c("Accuracy", "Precision", "Recall", "F1-Score"),  
 Value = c(  
 calculate\_metric("accuracy"),  
 calculate\_metric("weightedPrecision"),  
 calculate\_metric("weightedRecall"),  
 calculate\_metric("f1")))  
  
# Make table with gt  
metrics\_table <- metrics\_df %>%  
 gt() %>%  
 tab\_header(title = "Model Evaluation Metrics",  
 subtitle="") %>%  
 fmt\_percent(columns = "Value", decimals = 2) %>% # Limit decimals to 4 places for clarity  
 tab\_style(style = cell\_text(weight = "bold"), locations = cells\_column\_labels(everything())) %>% # Bold headers  
 opt\_stylize(style = 6, color = "blue")  
  
metrics\_table

Table : Model Evaluation Metrics

| Metric | Value |
| --- | --- |
| Accuracy | 38.56% |
| Precision | 35.54% |
| Recall | 38.56% |
| F1-Score | 33.37% |

The model correctly predicts the LOS for 38.11% of test cases, indicating low accuracy and reliability. A precision of 36.77% shows many false positives, while a recall of 38.11% indicates the model misses a large portion of true cases. The F1-Score of 33.20% reflects poor balance between precision and recall. These metrics suggests the model struggles to capture patterns to accurately predict LOS. This may result from dataset imbalances, insufficient predictive features or randomness in LOS.

1. Confusion Matrix: Highlights true positives, false positives, true negatives and false negatives for each class.

# Retrieve the mapping of indices to categories  
index\_mapping <- sdf\_covid\_hosp\_stay\_days\_map %>%  
 select(stay\_days\_mutated, stay\_days\_mutated\_index) %>%  
 distinct() %>%  
 arrange(stay\_days\_mutated\_index) %>%  
 collect()  
  
# Rename the columns in the mapping table for clarity  
index\_mapping <- index\_mapping %>%  
 rename(truth\_index = stay\_days\_mutated\_index,  
 truth\_category = stay\_days\_mutated)  
  
# Aggregate counts for confusion matrix in Spark  
ldf\_confusion\_matrix <- pred %>%  
 group\_by(truth = label, prediction = prediction) %>%  
 summarise(count = n(), .groups = "drop") %>% # Compute the counts for each actual-predicted pair  
 collect()  
  
# Join for truth (actual labels)  
ldf\_confusion\_matrix <- ldf\_confusion\_matrix %>%  
 left\_join(index\_mapping, by = c("truth" = "truth\_index"))  
  
# Rename mapping columns again for predictions  
index\_mapping <- index\_mapping %>%  
 rename(prediction\_index = truth\_index, prediction\_category = truth\_category)  
  
# Join for predictions  
ldf\_confusion\_matrix <- ldf\_confusion\_matrix %>%  
 left\_join(index\_mapping, by = c("prediction" = "prediction\_index"))  
  
# Select only the readable categories and counts  
ldf\_confusion\_matrix\_readable <- ldf\_confusion\_matrix %>%  
 select(truth\_category,   
 prediction\_category,   
 count)  
  
# Create a heatmap to visualise confusion matrix   
confusion\_matrix <- ggplot(ldf\_confusion\_matrix\_readable, aes(x = prediction\_category, y = truth\_category, fill = count)) +  
 geom\_tile(color = "white") +  
 scale\_fill\_gradient(low = "#DDEEFF", high = "blue") +  
 labs(  
 title = "Confusion Matrix",  
 x = "Predicted Category",  
 y = "Actual Category"  
 ) +  
 theme\_minimal()  
  
confusion\_matrix

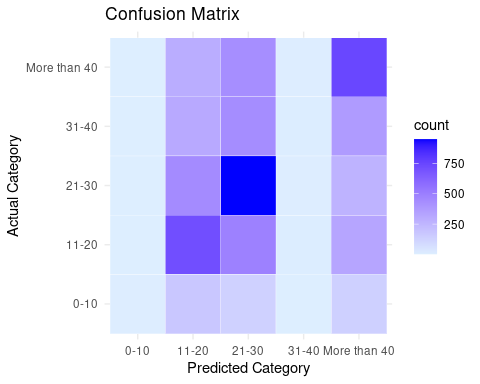


Figure 4: Confusion Matrix

The confusion matrix reveals that the model performs best at predicting the 21-30 and More\_than\_40 categories, which have the most data, indicating the model is bias towards majority classes. Categories such as 0-10 and 31-40, with fewer samples (n=31 and n=40), are rarely predicted correctly. This imbalance limits the model’s ability to learn patterns for under-represented categories, a common issue with Random Forest models when faced with imbalanced datasets.

## 4.2 Strengths and Weaknesses of the Approach:

Strengths:

1. **Robustness to Overfitting:** Random Forest good choice for complex datasets consisting of many features with non-linear relationships.
2. **Interpretability:** Importance metrics highlight key predictors of LOS which can guide decision-making and further analysis.
3. **Handling of Mixed Data:** The model effectively processes both numerical and categorical features without extensive preprocessing.

Weaknesses:

1. **Poor Predictive Power:** Low accuracy and F1-Score suggest the model is not effectively capturing patterns in the data.
2. **Bias Toward Majority Classes:** Under-representation of minority categories skews predictions.
3. **Limited Real-World Applicability:** Predictions lack reliability for real-world decisions.
4. **Computational Cost:** Training and evaluating Random Forest models are resource-intensive for large datasets.

## 4.3 Broader Implications of Analysis

Future Random Forest models must address class imbalance to improve predictions for under-represented categories. Techniques such as oversampling minority classes, under sampling majority classes, or making the weight of each class a feature in the model can be used to ensure categories are equally represented.

This dataset may lack critical features which determine LOS limiting accuracy. Prospective studies such as ISARIC4C CO-CIN gathered more parameters on admission characteristics and hospital resource utilisation which allowed development of robust models which informed public health policy in the UK.(4,5) Additionally, this analysis highlights the importance of selecting features known at admission for real-time applicability.

This model’s low accuracy and reliability make it unsuitable for actionable decision-making. A simpler approach, such as setting a threshold and grouping outcomes (e.g., <30 days vs. >30 days), may enhance performance.

Insights from this analysis can guide predictive tools for future pandemics, aiding healthcare system preparedness.

# 5 References

1. Rees EM, Nightingale ES, Jafari Y, Waterlow NR, Clifford S, Pearson CA, et al. COVID-19 length of hospital stay: a systematic review and data synthesis. *BMC Medicine* [Internet]. 2020 Sep 3 [cited 2024 Dec 11];18(1):270. Available from: <https://doi.org/10.1186/s12916-020-01726-3>
2. Alabbad DA, Almuhaideb AM, Alsunaidi SJ, Alqudaihi KS, Alamoudi FA, Alhobaishi MK, et al. Machine learning model for predicting the length of stay in the intensive care unit for COVID-19 patients in the eastern province of Saudi Arabia. *Informatics in Medicine Unlocked* [Internet]. 2022 [cited 2024 Dec 11];30:100937. Available from: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9010025/>
3. Samy SS, Karthick S, Ghosal M, Singh S, Sudarsan JS, Nithiyanantham S. Adoption of machine learning algorithm for predicting the length of stay of patients (construction workers) during COVID pandemic. *International Journal of Information Technology* [Internet]. 2023 Jun 1 [cited 2024 Dec 11];15(5):2613–21. Available from: <https://doi.org/10.1007/s41870-023-01296-6>
4. Keogh RH, Diaz-Ordaz K, Jewell NP, Semple MG, de Wreede LC, Putter H, et al. Estimating distribution of length of stay in a multi-state model conditional on the pathway, with an application to patients hospitalised with Covid-19*. Lifetime Data Anal* [Internet]. 2023 Apr 1 [cited 2024 Dec 12];29(2):288–317. Available from: <https://doi.org/10.1007/s10985-022-09586-0>
5. Leclerc QJ, Fuller NM, Keogh RH, Diaz-Ordaz K, Sekula R, Semple MG, et al. Importance of patient bed pathways and length of stay differences in predicting COVID-19 hospital bed occupancy in England. *BMC Health Serv Res* [Internet]. 2021 Jun 9 [cited 2024 Dec 12];21(1):566. Available from: <https://doi.org/10.1186/s12913-021-06509-x>

Word Count = 1044 words (excluding titles, code chunks & references) Figures/Tables = 6

spark\_disconnect\_all(sc)

## [1] 1