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

```r
#Load necessary libraries:
library(sparklyr)
library(dplyr)
library(purrr)
library(ggplot2)
library(gt)
library(knitr)

#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

```r
# 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 1: 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.

```r
# 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))
```

```r
# Distribution of Stay_Days (target variable)
dist_stay_days <- sdf_covid_hosp %>%
  group_by(Stay_Days) %>%
  summarise(count = n()) %>% # summarise in spark
  collect() %>% # collect back to R
  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
```

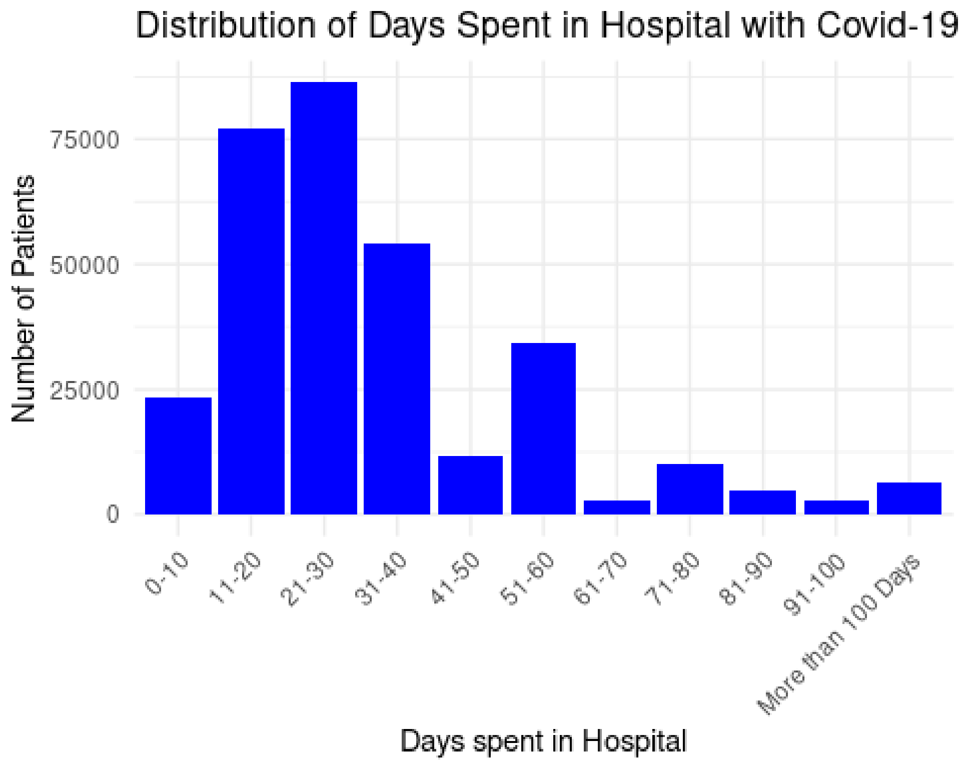

Figure 1: Distribution of Days Spent in Hospital due to Covid-19 infection

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.

```r
# 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
```

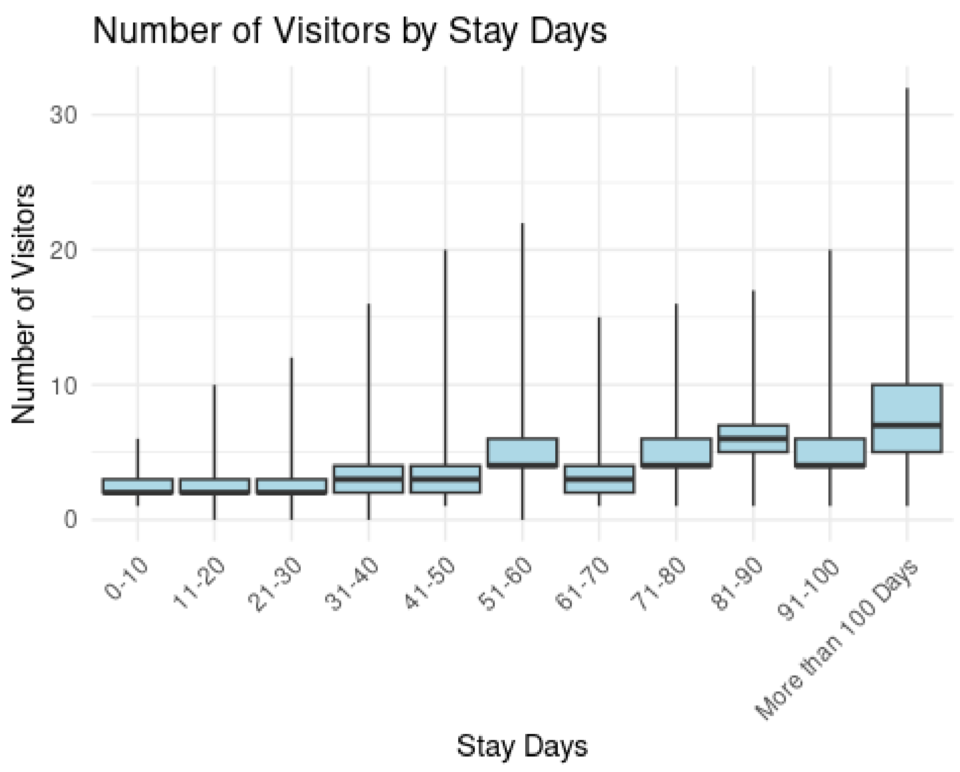

Figure 2: Relationship between the number of visitors and the length of stay in hospital

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.

This feature highlights the importance of 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 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

```r
# 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
```

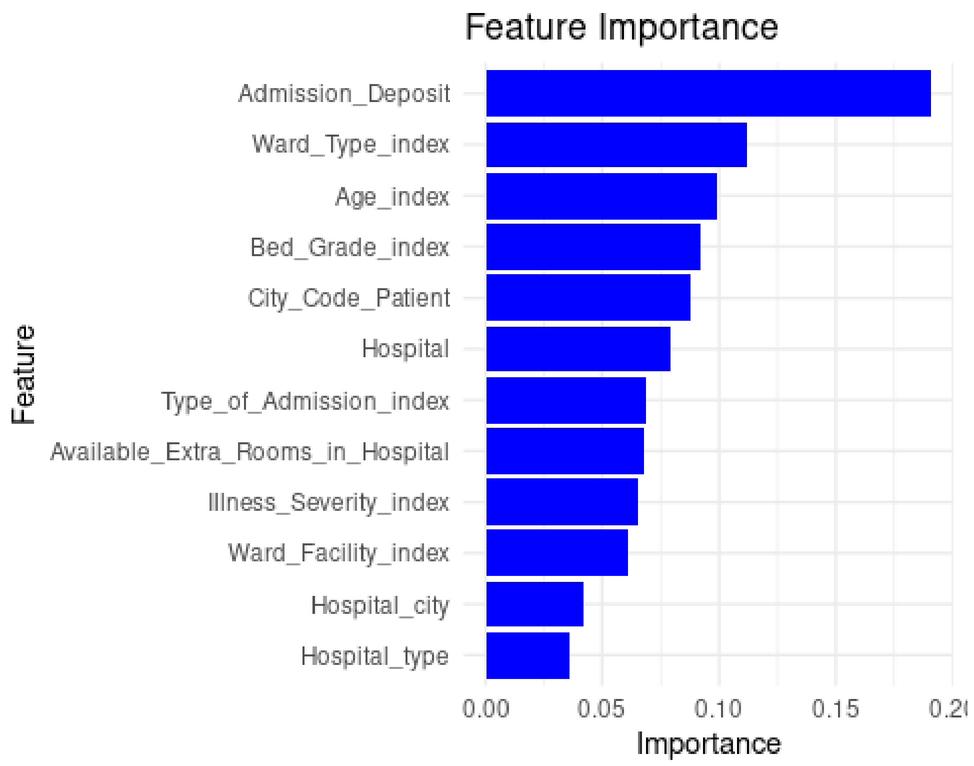

Figure 3: Ranked Importance of each Feature

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

```r
# 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 2
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 2: 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 many true cases. The F1-Score of 33.20% reflects a poor balance between precision and recall. These metrics suggest the model struggles to capture patterns to predict LOS accurately. This may result from dataset imbalances, insufficient predictive features or randomness in LOS.

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

```r
# Retrieve the mapping of indices to categories
ldf_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
ldf_index_mapping <- ldf_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(ldf_index_mapping, by = c("truth" = "truth_index"))

# Rename mapping columns again for predictions
ldf_index_mapping <- ldf_index_mapping %>%
  rename(prediction_index = truth_index, prediction_category =
truth_category)

# Join for predictions
ldf_confusion_matrix <- ldf_confusion_matrix %>%
  left_join(ldf_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
```

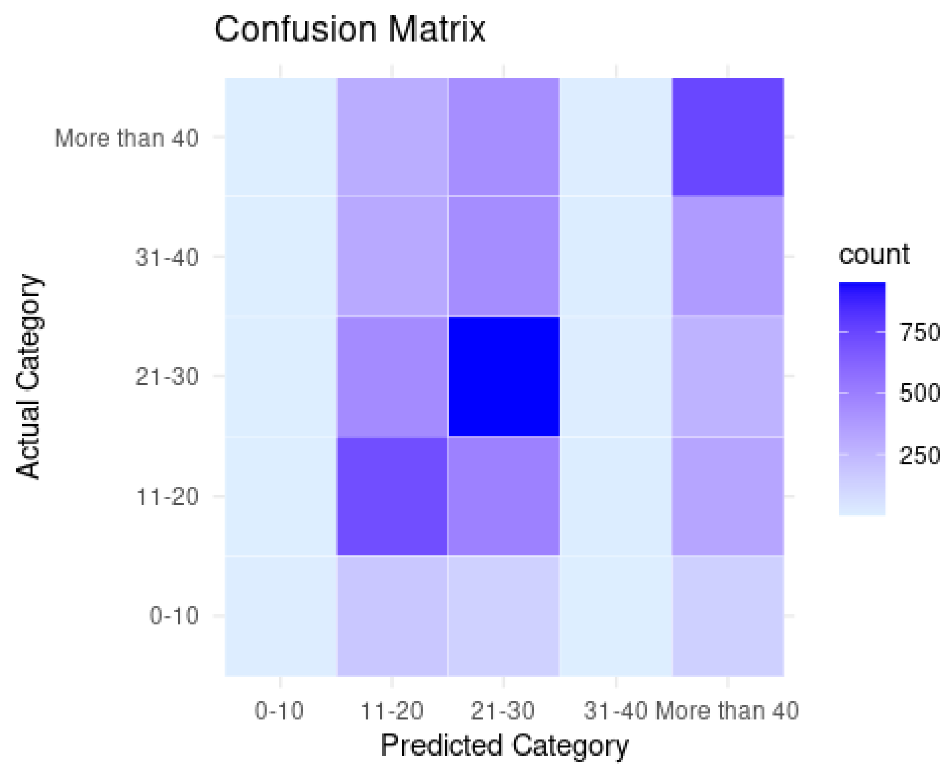

Figure 4: Confusion Matrix

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 biased 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's CO-CIN gathered more parameters on admission characteristics and hospital resource utilisation allowing robust models to be developed.(4,5) Additionally, such analysis highlights the importance of selecting features known at admission for real-time applicability.

Our 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
```