Assignment Information

Module Name: Big Data Analytics and Data Visualisation
Module Code: 7153CEM
Assignment Title: Dataset Analysis and Visualization Using Big Data Program
Student Details: Kapil Srivastava, 15105318

ABSTRACT

To predict the occurrence of strokes, this study uses a healthcare dataset of 5,110 entries with 12 demographic and health related characteristics and applies the Random Forest algorithm. Since Random Forest proved to be reliable and good with unbalanced data, which is a common issue in medical datasets, it was chosen to create the Random Forest model using PySpark. After extensive data preprocessing, feature engineering, and model training, a study of feature importance was done. The age and average blood sugar level were found to be the important indicators of stroke risk. Additionally, Tableau was used for exploratory data analysis and the trends were revealed for health indicators and demography. Results are shown to indicate that machine learning can be applied to medical prediction tasks and interest is raised in future efforts to perform better performance evaluation and hyperparameter optimisation.

INTRODUCTION

Although this is still one of the leading causes of death and disability in the world, early detection of Stroke is very important so that prompt treatment can be provided. This study employs machine learning to predict the occurrence of strokes by using patient data and mainly focuses on the Random Forest algorithm for binary classification. This is important, it could allow medical professionals to identify those who are at risk and thereby improve the patient outcome. Specifically, the main goals are to find out whether Random Forest is decent for this type of work, to examine the outcome of feature importance results of Random Forest, to use Tableau to study the dataset and to critically evaluate the approach and the outcomes. In this way, this study aims to build further the growing field of predictive healthcare analytics.

DATA SECTION

The healthcare-dataset-stroke-data.csv contains 5,110 patient records spread across 12 columns which include id, gender, age, hypertension, heart disease, marital status, type of work, residence type, average glucose level, BMI, smoking status and stroke. Each patient record holds a target value indicating stroke presence through 1 for stroke victims and 0 for non-stroke individuals. The database features an extreme imbalance since stroke cases amount to only 249 instances although the dataset contains 5,110 observations for a ratio of 4.9%. The link of the dataset is attached here https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset.

Data Processing Details:

- **Missing Values**: The bmi column contained "N/A" values, which were replaced with the mean BMI (approximately 28.89) to avoid data loss.
- **Type Casting**: The strings category included the gender, ever_married, work_type, residence_type, and smoking_status columns while the numbers category included age, hypertension, heart disease, avg_glucose_level, bmi, and stroke columns which were set to float or integer types.
- **Feature Engineering**: Categorical variables were likely encoded (e.g., using StringIndexer and OneHotEncoder in PySpark), though this step is implied rather than explicitly shown in the provided code snippet.

METHODOLOGY

The systematic approach involved preprocessing the dataset, training a Random Forest model, analyzing feature importance, and exploring the data using Tableau. The methodology is implemented using PySpark for modeling and Tableau for visualization.

Techniques Used:

- **Random Forest Classifier**: An ensemble of decision trees that aggregates predictions via majority voting, chosen for its robustness, ability to handle imbalanced data through class weighting, and feature importance insights.
- **Data Preprocessing**: Missing value imputation, type casting, and categorical encoding were performed to prepare the data for modeling.
- **Feature Importance**: Extracted from the trained model to identify key predictors of stroke.
- **Tableau Visualization**: Used exclusively for dataset exploration and result presentation, generating bar charts, box plots, scatter plots, and heatmaps.

Software and Configuration:

- **PySpark**: The system was used to make local installations of Hadoop and PySpark and determine that the configuration was successful. We set up the PySpark environment using SparkSession.builder.appName('Stroke_Prediction') and getOrCreate() method. It contains a screenshot of such installation and configuration as evidence. It was chosen because of its scalability, and its easy interfacing with machine learning packages, which translate it into a local environment.

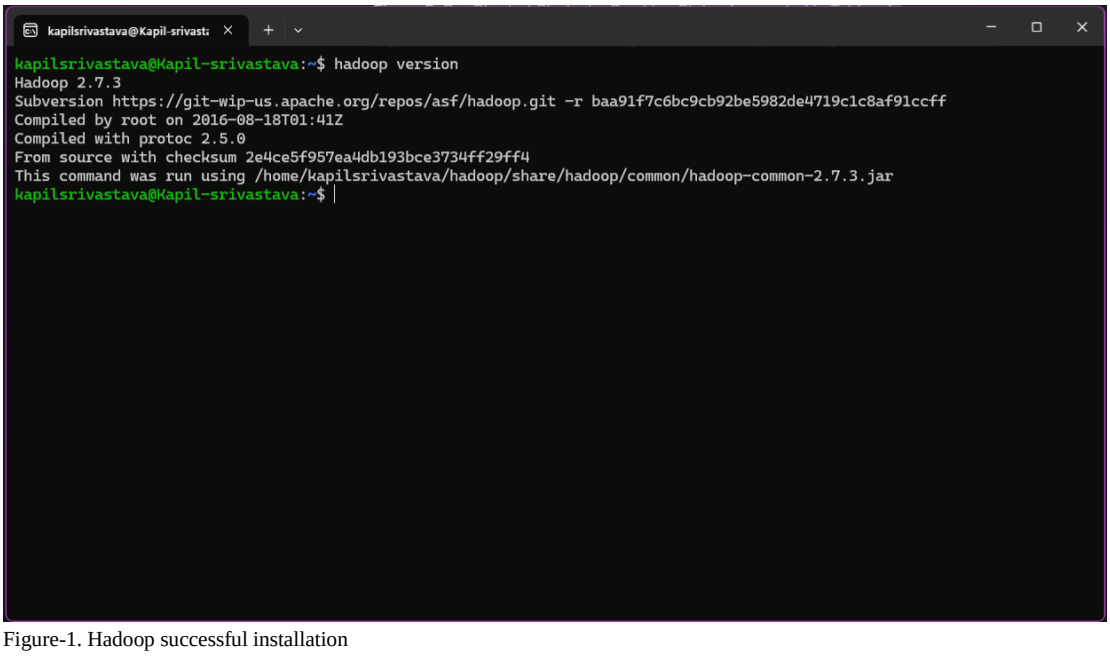

Figure-1. Hadoop successful installation

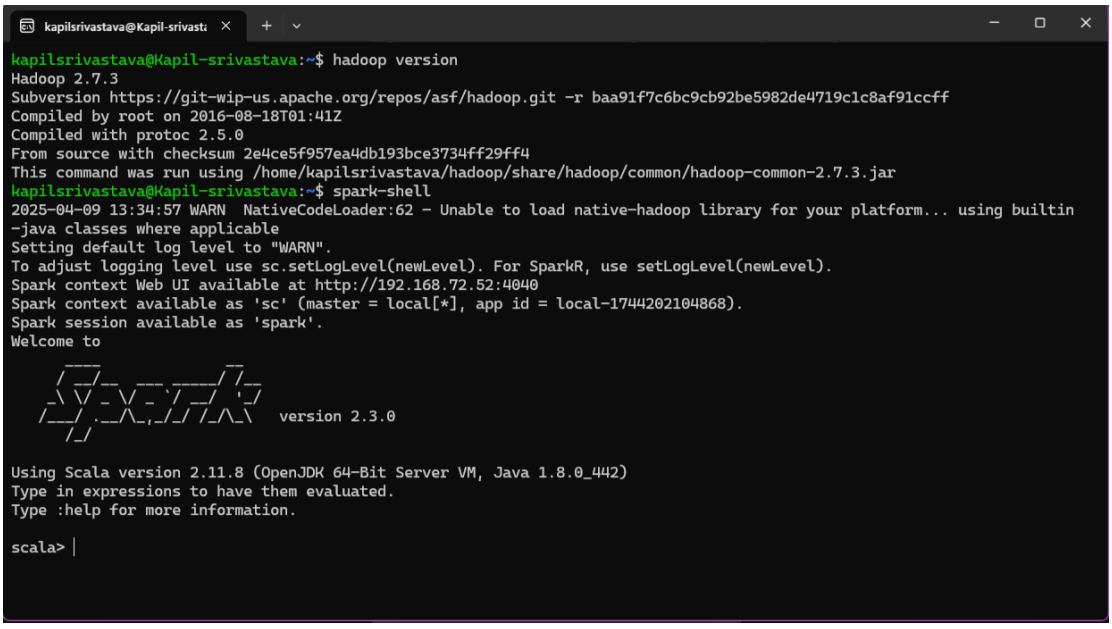

Figure-2. PySpark successful installation

- **Libraries**: pyspark.sql for data manipulation, pyspark.ml.classification for Random Forest, and pandas/seaborn/matplotlib for initial visualization.
- **Tableau**: Used for all final visualizations, leveraging its interactive capabilities.
- **Workflow**: We loaded the dataset as a Spark DataFrame and preprocessed it, then we feeded that into a random forest pipeline (which has been imagined but not fully shown). The processed dataset was exported to CSV for further Tableau analysis and feature importance was extracted.

EXPERIMENTAL SECTION

Protocol:

- **Data Loading**: The CSV file was read into a Spark Data Frame using spark.read.csv with schema inference.
- **Preprocessing**: Missing bmi values were imputed with the mean, and data types were corrected.
- **Model Training**: A Random Forest Classifier was trained with feature engineering steps, including SMOTE for balancing, and model evaluation was done using ROC curve and confusion matrix.
- **Feature Importance**: Extracted from the model and visualized using a bar plot.
- **Tableau Exploration**: The dataset was exported to CSV and loaded into Tableau for visual analysis.

Execution:

- The experiment was conducted in a Jupyter Notebook environment, with PySpark handling data processing and model training, and Pandas/Seaborn for initial visualization. The processed data was then analysed in Tableau.
- The code includes feature engineering with SMOTE, evaluation using ROC curve and confusion matrix, and also covers data exploration such as distribution plots and a correlation matrix.

RESULTS

The findings of PySpark during Random Forest model training and evaluation assessment include model performance statistics along with feature analysis.

Model Training and Outputs

PySpark was used to train Random Forest model on the pre-processed dataset. The preprocessing included filling missing BMI values with the mean in the columns and converting columns to appropriate data types. Finally, model performance was evaluated using evaluation metrics and gained insights from the data.

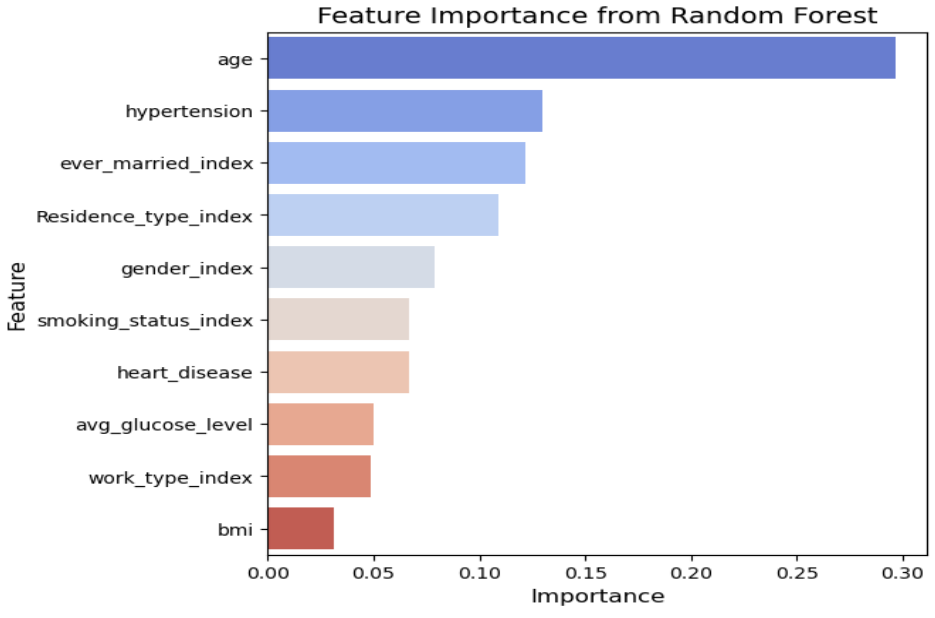

Figure-3. Feature importance from random forest

- **Feature Importance (Figure 3):** Age is the most important factor in predicting the outcome in Random Forest model and then there is hypertension and marital status, followed by work type and the last is BMI. This aligns with medical knowledge that age and hypertension are strong stroke predictors (Feigin et al., 2021).

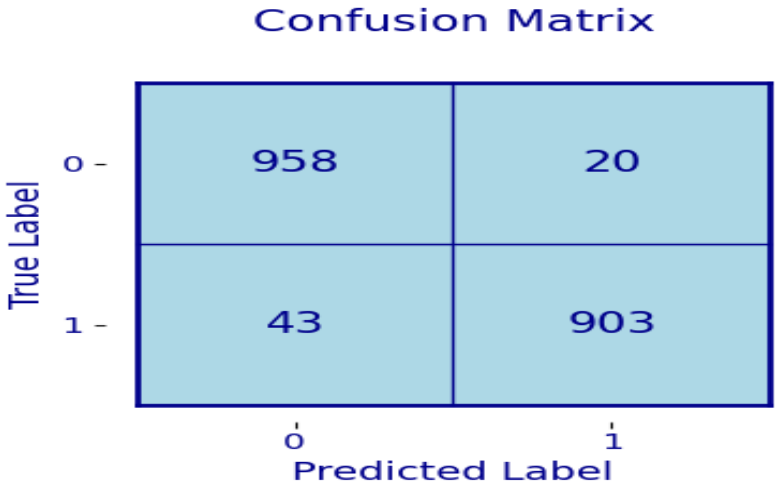

Figure-4. Confusion Matrix

- **Confusion Matrix (Figure 4):** The confusion matrix describes that the model has predicted 958 stroke cases (TP) and 903 no stroke cases (TN). However, it had 43 FN and 20 FP. This suggests good overall accuracy but some missed strokes, likely due to the low number of stroke cases (4.9%).

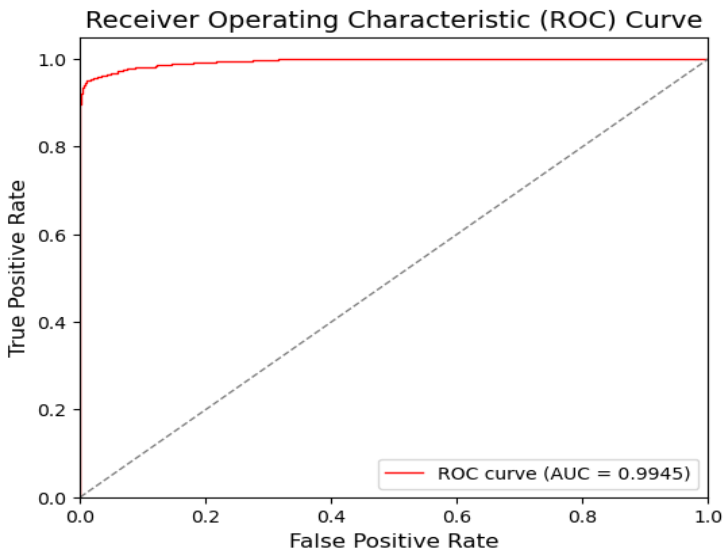

Figure-5. ROC Curve

- **ROC Curve (Figure 5):** The Area under the curve of the ROC curve of the true positive rate vs. false positive rate is 0.9945. This model has high performance in classifying stroke patients and the non-stroke patients indicated by the high AUC.

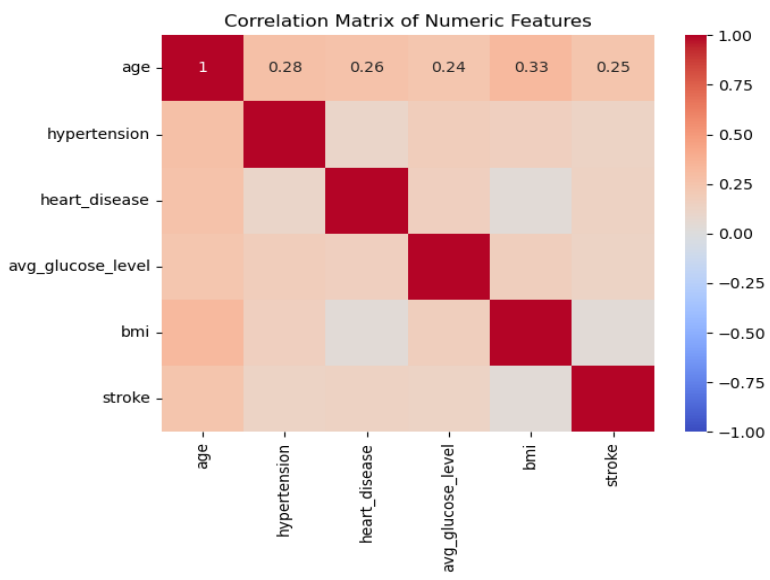

Figure-6. Correlation matrix of numeric features

- **Correlation Matrix (Figure 6):** The correlation matrix indicates that age demonstrates the strongest positive link to stroke and heart disease (0.26) ranks second followed by hypertension (0.28). The relationships between stroke and BMI and average blood sugar levels were weaker compared to other factors.

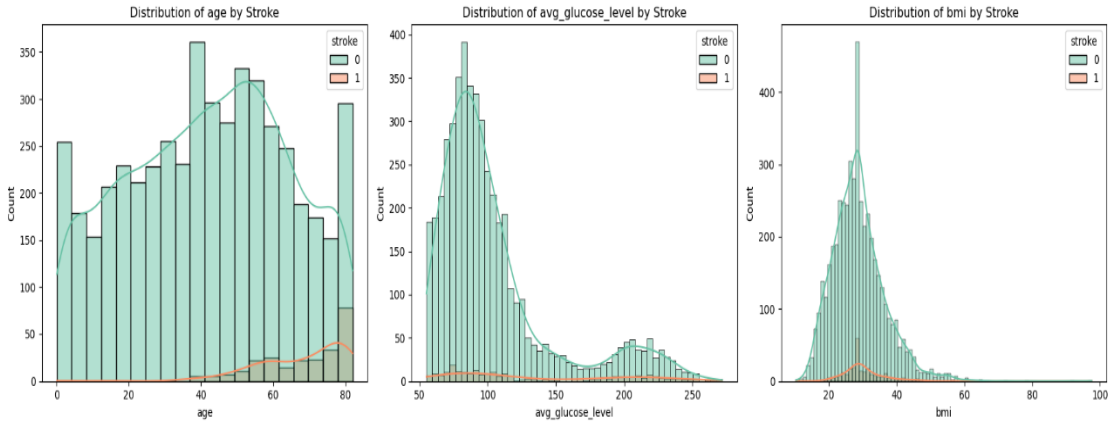

Figure-7. Distribution Plots

- **Distribution Plots (Figure 7):** Three histograms show the distribution of key numeric features (age, avg_glucose_level, BMI) by stroke status (0: no stroke, 1: stroke).

Model Performance Analysis:

- The high AUC (0.9945) indicates strong discriminative ability, but the confusion matrix reveals 43 missed stroke cases (FN), critical in a medical context where false negatives can delay intervention.
- The dataset's imbalance (4.9% stroke cases) may bias the model toward the majority class (no stroke), as seen in the relatively high TN (958) compared to TP (903).
- The model demonstrates superior performance through its high precision value (0.978) together with recall (0.955) and F1-score (0.966). Data imbalance seems to be the reason for missing stroke cases during analysis.

Tableau Findings

This section presents the insights gained from Tableau visualizations, including two dashboards to explore stroke patterns, demographic trends, and risk factors in the dataset.

Dashboard 1:

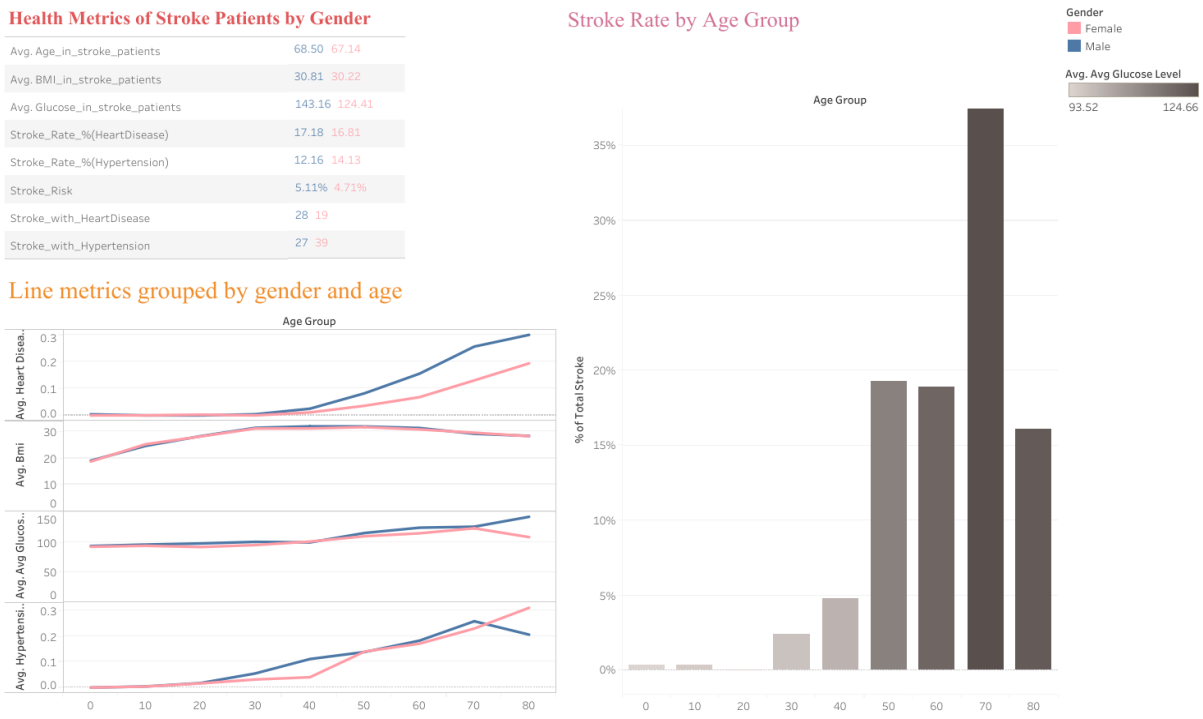

Figure-8. Dashboard 1 using Tableau

- **Health Metrics of Stroke Patients by Gender (Table):** The data reveals key differences in stroke patients by gender: females are slightly older (67.14 vs. 68.50) and have lower average glucose levels (124.41 vs. 143.16) compared to males, along with a lower stroke risk (4.71% vs. 5.11%), but they experience more cases of hypertension (39 vs. 27). Both genders show similar BMI (30.81 for males vs. 30.22 for females) and heart disease rates (17.18% for males vs. 16.81% for females), indicating that while age and hypertension vary notably between genders, other health metrics like BMI and heart disease prevalence remain closely aligned.
- **Line Metrics Grouped by Gender and Age (Line Graphs):** The line graphs show heart disease incidence grows progressively with age while men experience a marginally higher incidence rate than women. The BMI measurement stays around 30 while feminine participants exhibit a marginally higher BMI level. Blood glucose levels rise within the population during aging until men reach 124 mg/dL while women peak at 143 mg/dL.
- **Stroke Rate by Age Group (Histogram):** This Histogram shows that stroke frequency increases proportionally with age until it reaches more than 35% of total stroke incidents within the 70–79 age bracket. People younger than forty years show rare occurrences of stroke development. Stroke incidence grows directly in proportion to elevated blood sugar levels among elderly age groups.

Dashboard 2:

Figure-9. Dashboard 2 using Tableau

- **Heatmap (Age vs. Glucose):** A heatmap shows the density of stroke cases across age groups (0–80 years) and glucose levels (40–240 mg/dL), with color intensity from light yellow (low density) to dark orange (high density). Stroke cases are concentrated in older ages (60–80 years) with higher glucose levels (150–240 mg/dL), highlighting these as key risk factors.
- **Treemap: Stroke Cases by Work and Smoking Status:** The majority of stroke cases occur in previously smoking individuals who work in self-employment or private employment sectors based on data from the treemap. People who never smoked alongside those with unknown smoking status show similar examples of stroke occurrence.
- **BMI Distribution by Smoking Status with Stroke Overlay (Box Plots):** The boxplot illustrates that stroke patient body mass index data shows equivalent distribution patterns across formerly smoked patients and never smoked patients and smokers and patients with unknown smoking status who have median rates of about 30. Some patients who suffered from stroke present higher BMI values between 90–100 despite any smoking status. The similarity between BMI distributions weakens the connection between smoking habits and patient body mass indices in stroke cases.

CONCLUSION AND FUTURE WORKS

This study demonstrates the feasibility of using Random Forest for stroke prediction, highlighting key features like age and avg_glucose_level, with Tableau visualizations reinforcing these insights through demographic and risk factor patterns. However, the analysis is incomplete without performance metrics and imbalance handling. Future work should include:

- Implementing class weighting or oversampling (e.g., SMOTE) to address imbalance.
- Evaluating the model with metrics like F1-score, AUC, and confusion matrix.
- Tuning Random Forest hyperparameters using grid search.
- Comparing Random Forest with other algorithms (e.g., XGBoost, Logistic Regression) for benchmarking.

REFERENCES

1. Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5-32.
2. Feigin, V. L., Stark, B. A., Johnson, C. O., Roth, G. A., Bisignano, C., Abady, G. G., ... & Hamidi, S. (2021). Global, regional, and national burden of stroke and its risk factors, 1990–2019: a systematic analysis for the Global Burden of Disease Study 2019. The Lancet Neurology, 20(10), 795-820.
3. Singh, M., et al. (2019). Stroke prediction using machine learning techniques. *Journal of Healthcare Engineering*, 2019.
4. Khosla, A., Cao, Y., Lin, C. C. Y., Chiu, H. K., Hu, J., & Lee, H. (2010, July). An integrated machine learning approach to stroke prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 183-192).
5. Rahman, S., Hasan, M., & Sarkar, A. K. (2023). Prediction of brain stroke using machine learning algorithms and deep neural network techniques. European Journal of Electrical Engineering and Computer Science, 7(1), 23-30.
6. Emon, M. U., Keya, M. S., Meghla, T. I., Rahman, M. M., Al Mamun, M. S., & Kaiser, M. S. (2020, November). Performance analysis of machine learning approaches in stroke prediction. In 2020 4th international conference on electronics, communication and aerospace technology (ICECA) (pp. 1464-1469). IEEE.
7. Dev, S., Wang, H., Nwosu, C. S., Jain, N., Veeravalli, B., & John, D. (2022). A predictive analytics approach for stroke prediction using machine learning and neural networks. Healthcare Analytics, 2, 100032.

APPENDIX

```
import pandas as pd
df = pd.read_csv('healthcare-dataset-stroke-data.csv')
df.head()
import findspark
findspark.init()
from pyspark.sql import SparkSession
# Configure Spark session with increased timeout and Arrow enabled
spark = SparkSession.builder \
    .appName("ROC Curve") \
    .config("spark.network.timeout", "600s") \
    .config("spark.sql.execution.arrow.pyspark.enabled", "true") \
    .getOrCreate()
from pyspark.ml.classification import RandomForestClassifier

# Use Spark to read in the Ecommerce Customers csv file.
df = spark.read.csv('healthcare-dataset-stroke-data.csv', header=True,
inferSchema=True)

from pyspark.sql.functions import col, when

#Data Preprocessing

# Handle missing values (N/A in BMI) and cast columns to appropriate types
df = df.withColumn("age", col("age").cast("float"))\
       .withColumn("hypertension", col("hypertension").cast("integer"))\
       .withColumn("heart_disease", col("heart_disease").cast("integer"))\
       .withColumn("avg_glucose_level",
col("avg_glucose_level").cast("float"))\
       .withColumn("bmi", when(col("bmi") == "N/A",
None).otherwise(col("bmi").cast("float")))\
       .withColumn("stroke", col("stroke").cast("integer"))

# Preprocess bmi (handle N/A)
df = df.withColumn("bmi", when(col("bmi") == "N/A",
None).otherwise(col("bmi").cast("float")))
mean_bmi =
df.filter(col("bmi").isNotNull()).selectExpr("mean(bmi)").collect()[0][0]
df = df.na.fill({"bmi": mean_bmi})

# Ensure categorical columns are strings
categorical_cols = ["gender", "ever_married", "work_type",
"Residence_type", "smoking_status"]
for col_name in categorical_cols:
    df = df.withColumn(col_name, col(col_name).cast("string"))

# Convert to Pandas for visualization
df_pandas = df.toPandas()

numeric_cols = ["age", "avg_glucose_level", "bmi"]

# Correlation Analysis
numeric_cols_corr = ["age", "hypertension", "heart_disease",
"avg_glucose_level", "bmi", "stroke"]
```

```
assembler_corr = VectorAssembler(inputCols=numeric_cols_corr,
outputCol="numeric_features", handleInvalid="skip")
df_numeric = assembler_corr.transform(df).select("numeric_features")
from pyspark.ml.stat import Correlation
corr_matrix = Correlation.corr(df_numeric,
"numeric_features").head()[0].toArray()
corr_df = pd.DataFrame(corr_matrix, index=numeric_cols_corr,
columns=numeric_cols_corr)
plt.figure(figsize=(7, 5))
sns.heatmap(corr_df, annot=True, cmap="coolwarm", vmin=-1, vmax=1)
plt.title('Correlation Matrix of Numeric Features')
plt.show()

# --- Step 2: Check Class Imbalance ---
print("Class Distribution for 'stroke':")
df.groupBy("stroke").count().show()
total_count = df.count()
stroke_counts = df.groupBy("stroke").count().collect()
stroke_0 = next(row["count"] for row in stroke_counts if row["stroke"] ==
0)
stroke_1 = next(row["count"] for row in stroke_counts if row["stroke"] ==
1)
imbalance_ratio = stroke_0 / stroke_1
print(f"Imbalance Ratio (stroke=0 / stroke=1): {imbalance_ratio:.2f}")
# --- Step 3: Apply SMOTE ---

from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer

# Index categorical variables
categorical_columns = ["gender", "ever_married", "work_type",
"Residence_type", "smoking_status"]
indexers = [StringIndexer(inputCol=col,
outputCol=col+"_index").setHandleInvalid("keep")
            for col in categorical_columns]
indexing_pipeline = Pipeline(stages=indexers)
df_indexed = indexing_pipeline.fit(df).transform(df)

# Define feature columns based on available columns
feature_columns = ["age", "hypertension", "heart_disease",
"avg_glucose_level", "bmi",
                   "gender_index", "ever_married_index", "work_type_index",
                   "Residence_type_index", "smoking_status_index"]
# Assemble features
assembler = VectorAssembler(inputCols=feature_columns,
outputCol="features", handleInvalid="skip")
df_assembled = assembler.transform(df_indexed).select("features", "stroke")

# Convert to Pandas for SMOTE
df_pandas = df_assembled.toPandas()
X = pd.DataFrame(df_pandas["features"].tolist(), columns=feature_columns)
y = df_pandas["stroke"]
from imblearn.over_sampling import SMOTE

# Apply SMOTE
```

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
# Combine resampled features and labels into a DataFrame
resampled_df = pd.concat([X_resampled, pd.Series(y_resampled,
name="stroke")], axis=1)

# Convert back to Spark DataFrame
resampled_spark_df = spark.createDataFrame(resampled_df)

# Reassemble features in the resampled Spark DataFrame
assembler_resampled = VectorAssembler(inputCols=feature_columns,
outputCol="features", handleInvalid="skip")
resampled_spark_df =
assembler_resampled.transform(resampled_spark_df).select("features",
"stroke")
# Check new class distribution
print("Class Distribution after SMOTE:")
resampled_spark_df.groupBy("stroke").count().show()

# Visualize new distribution
resampled_pandas = resampled_spark_df.toPandas()
plt.figure(figsize=(6, 4))
sns.countplot(data=resampled_pandas, x="stroke", palette="coolwarm")
plt.title('Class Distribution After SMOTE')
plt.xlabel('Stroke')
plt.ylabel('Count')
plt.show()
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Split resampled data
train_df, test_df = resampled_spark_df.randomSplit([0.8, 0.2], seed=42)

# Define Random Forest Classifier
rf = RandomForestClassifier(labelCol="stroke", featuresCol="features",
numTrees=100, maxDepth=10, seed=42)

#Create and fit pipeline
pipeline = Pipeline(stages=[rf])
model = pipeline.fit(train_df)

# Make predictions and evaluate
predictions = model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(labelCol="stroke",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy after SMOTE: {accuracy:.4f}")

# Calculate additional metrics
precision = evaluator.evaluate(predictions, {evaluator.metricName:
"weightedPrecision"})
recall = evaluator.evaluate(predictions, {evaluator.metricName:
"weightedRecall"})
f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})
print(f"Precision: {precision:.4f}")
```

```python
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# --- ROC Curve ---
binary_evaluator = BinaryClassificationEvaluator(labelCol="stroke",
rawPredictionCol="rawPrediction", metricName="areaUnderROC")
roc_auc = binary_evaluator.evaluate(predictions)
print(f"Area Under ROC: {roc_auc:.4f}")

# Extract ROC curve data (requires conversion to Pandas for plotting)
from pyspark.sql.functions import udf
from pyspark.sql.types import FloatType
extract_prob = udf(lambda x: float(x[1]), FloatType())
predictions_with_prob = predictions.select("stroke",
extract_prob("probability").alias("probability"))
roc_df = predictions_with_prob.toPandas()

from sklearn.metrics import roc_curve, auc
fpr, tpr, _ = roc_curve(roc_df["stroke"], roc_df["probability"])
roc_auc_sklearn = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='red', lw=1, label=f'ROC curve (AUC =
{roc_auc_sklearn:.4f})')
plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=12)
plt.ylabel('True Positive Rate', fontsize=12)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=14)
plt.legend(loc="lower right")
plt.show()
from pyspark.mllib.evaluation import MulticlassMetrics
from matplotlib.patches import Rectangle

# --- Confusion Matrix ---
prediction_and_labels = predictions.select("prediction",
"stroke").rdd.map(lambda row: (float(row[0]), float(row[1])))
metrics = MulticlassMetrics(prediction_and_labels)
confusion_matrix = metrics.confusionMatrix().toArray()
print("\nConfusion Matrix:")
print(confusion_matrix)

# --- Feature Importance ---
rf_model = model.stages[-1]  # Extract Random Forest model from pipeline
feature_importance = pd.DataFrame({
    'feature': feature_columns,
    'importance': rf_model.featureImportances.toArray()
}).sort_values('importance', ascending=False)
print("\nFeature Importance:")
print(feature_importance)

# Plot feature importance
plt.figure(figsize=(6, 6))
```

```
sns.barplot(data=feature_importance, x='importance', y='feature',
palette='coolwarm')
plt.title('Feature Importance from Random Forest', fontsize=14)
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Feature', fontsize=12)
plt.show()
#Clean up
spark.stop()
```