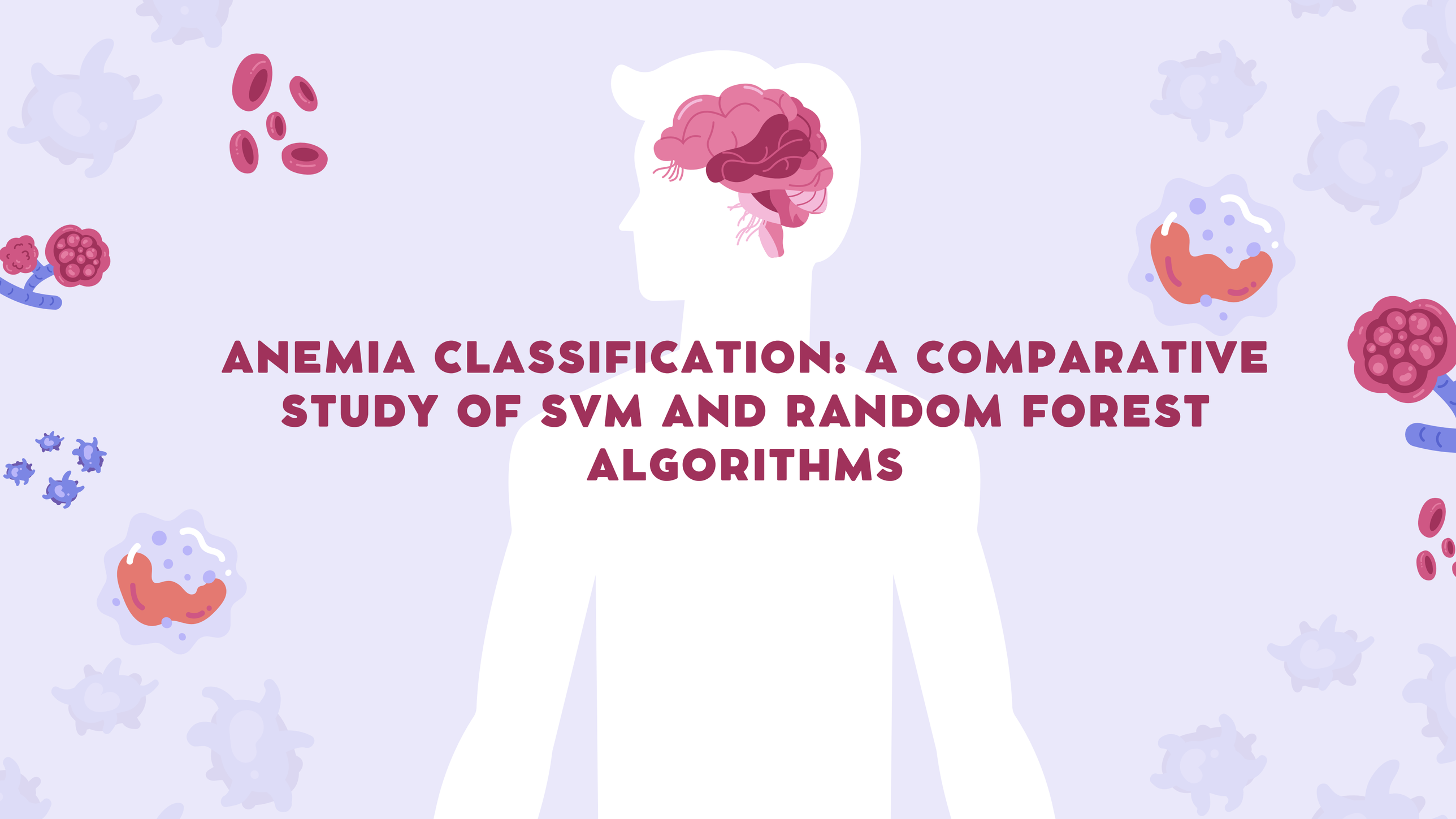


ANEMIA CLASSIFICATION: A COMPARATIVE STUDY OF SVM AND RANDOM FOREST ALGORITHMS

AGENDA

01
Introduction

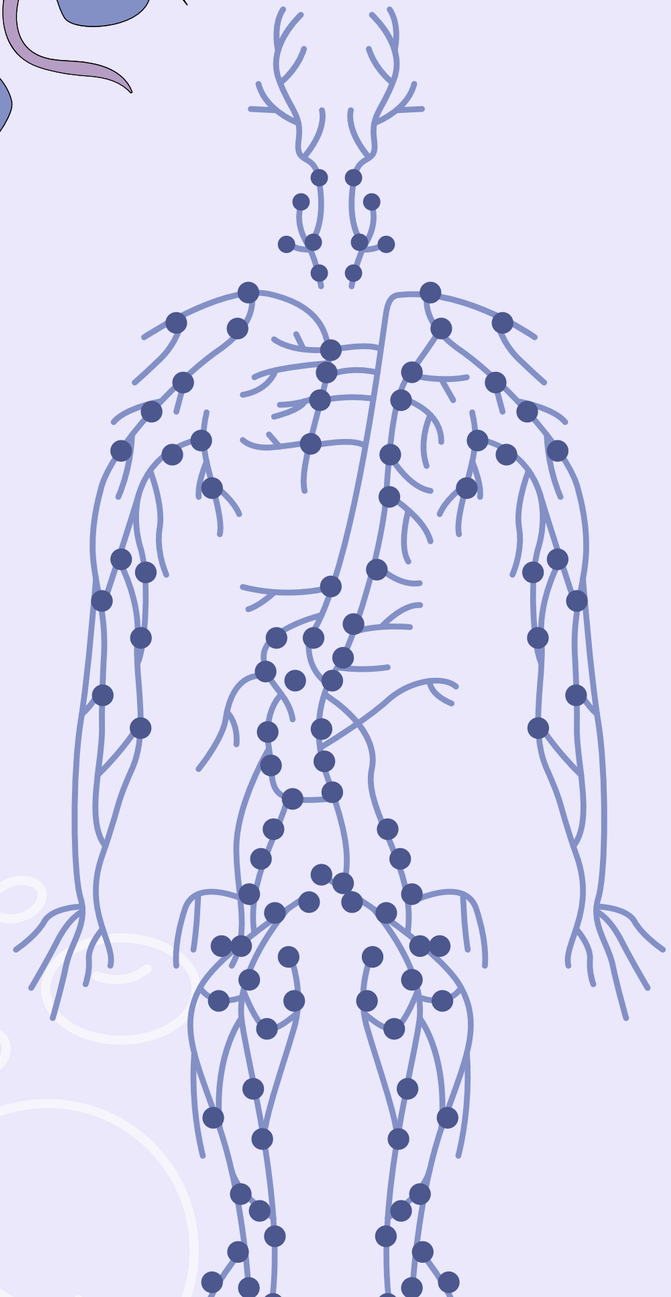
02
Dataset Description

03
Data Preprocessing

04
Data Visualization

05
Model Training

06
Result and Discussion



INTRODUCTION

Did you know that anemia affects millions of people worldwide? It is a condition characterized by a deficiency of red blood cells or hemoglobin, resulting in reduced oxygen-carrying capacity. Accurate and timely diagnosis of anemia is crucial for effective treatment and management.

In this project, we explore the power of machine learning algorithms, specifically Support Vector Machines (SVM) and Random Forest, to classify anemia. By leveraging the vast potential of these algorithms, we aim to develop a robust and reliable model that can assist healthcare professionals in accurately identifying anemia cases.

DATASET DESCRIPTION

Dataset

Dataset that we use consist of training data (113 data) and testing data (57 data)

No.	RBC	HGB	Cellular	HCT	MCV	MCH	MCHC	CHCM	CH	RDW	HDW	Retic %	CHr	CHm	RBC %	RBC %	RBC %	RBC %	RBC	Group
E541	3,78	9	10,1	29,4	77,7	23,8	30,7	34,2	25,9	21,1	7,23	2,16	28,5	25,5	18,2	20,7	1,2	11,9	0,04	1
D272	5,94	10,3	12	37,5	63,1	17,4	27,5	32,1	20,1	15,9	3,59	1,38	23,1	20,3	2	9,3	0	38,8	0,06	1
F728	5,93	10,8	12	34,4	58	18,2	31,3	34,8	20	16	3,96	1,31	22,6	19,8	6,3	2,4	0	60,7	0,11	1
N1,848	5,29	11	11,6	37,6	71	20,7	29,2	30,8	21,7	16,9	3,07	1,42	25,9	22	0,7	15,3	0,1	15,4	0,08	1
O2320	5,79	11	11,9	37	63,9	19	29,7	32,2	20,4	16,6	3,36	1,06	24,5	20,7	1,4	8,2	0	36,8	0,05	1
A33	5,85	11,4	12,3	38,5	65,8	19,5	29,7	31,9	20,8	16,1	3,34	1,76	23,1	20,9	1,4	9,2	0	29	0,06	1
O2278	5,85	11,6	12,1	39,3	67,2	19,9	29,6	30,9	20,5	16,4	3,08	0,99	24,3	20,9	0,5	16,1	0	25,5	0,07	1
N1,940	5,96	11,7	12,7	40,2	67,5	19,7	29,1	31,6	21,1	17,3	3,19	0,88	24,8	21,3	1	10,7	0	25,7	0,1	1
O2035	6,16	11,7	12,9	40,2	65,3	19,1	29,2	32,1	20,7	17,6	3,35	0,74	25,6	21	1,6	8	0	31,9	0,02	1
K1,115	4,81	12,3		38,7	80,4	25,6	31,8			14,1		1,64	30,8	27,1	1,2	0,9	0,1	2,7	0,02	1
K1,119	5,15	13		40,3	78,3	25,2	32,1			14,5		1,58	28,2	25,8	0,4	2,2	0	4,3	0,01	1
K1,010	5,42	13,4		41,4	76,3	24,7	32,4			15,1		2,25	26,6	25,3	0,8	3,1	0,1	6,8	0,03	1
D292	6,83	13,8	15,1	44,3	64,9	20,2	31,2	34	21,8	16,5	3,15	0,87	25	21,8	2,3	1,2	0	33,2	0,05	1
K1,069	6,94	14,2		47,5	68,5	20,4	29,8			16		2,16	24,6	21,7	0,4	7,5	0	21,5	0,04	1
N1,844	6,8	14,2	15,2	47,2	69,3	20,9	30,2	32,3	22,1	16,6	3,1	0,53	27,2	22,4	0,7	7,1	0	21,1	0,03	1
K1,121	6	16,8		47,7	79,4	27,9	35,2			13,7		2,13	32,7	29,6	10,2	0,1	0,1	2,7	0,02	1
D314	5,12	11,2	13,4	37,9	74,1	21,9	29,5	35,3	26	14,9	3	1,71	28,9	25,6	3,4	0,3	0	9,3	0,04	1
A16	4,77	11,4	12,2	36,3	76,2	23,9	31,4	33,6	25,4	14,8	2,55	1,34	27,3	25,3	0,9	0,6	0	6,6	0,03	1
O2327	4,8	11,7	12,5	38,9	80,9	24,5	30,2	32,2	25,9	14,7	2,4	0,49	30	26,1	0,4	2,1	0,1	3,1	0,02	1
C226	4,91	12	12,7	38,1	77,6	24,5	31,6	33,3	25,6	15	2,6	1,04	27,7	25,7	0,8	1,2	0,1	5,5	0,02	1
D331	5,29	12,1	13,2	39,3	74,3	22,9	30,9	33,4	24,6	15,4	2,6	1,17	28,3	24,7	0,7	0,9	0	9,7	0,03	1
O2241	4,84	12,1	12,8	38,9	80,5	25,1	31,1	32,9	26,3	15	2,59	0,69	29,7	26,5	0,6	1,7	0,1	3,3	0,03	1
L1,339	4,83	12,2	12,7	38	78,7	25,2	32,1	33,5	26,2	14,9	2,6	1,09	28,5	26,2	1	1	0	4,4	0,01	1
F669	5,21	12,4	13,1	38,2	73,3	23,8	32,4	34,4	25	14,8	2,67	1,22	27,7	25	1,3	0,4	0	10	0,02	1
K1,319	5,21	12,4		38,9	74,7	23,8	31,9			15		1,91	28	25,6	1,2	0,7	0	8,4	0,03	1
D323	5	12,5	13,5	38,6	77,3	25	32,4	34,9	26,7	14,6	2,74	1,87	29,9	26,7	1,9	0,3	0,1	5,1	0,02	1
E566	5,17	12,6	13,2	38,5	74,5	24,3	32,6	34,3	25,4	15,1	2,69	1,4	27,8	25,3	1,5	0,3	0	8,8	0,02	1
L1,340	4,85	12,6	13,1	36,9	76,1	26	34,2	35,4	26,8	14,4	2,78	1,86	29,8	26,7	2,6	0,2	0	6,1	0,02	1
L1,404	5,04	12,8	13,3	39,4	78,2	25,4	32,5	33,8	26,3	14,5	2,35	1,54	29,1	26,4	0,5	0,6	0	4,5	0,02	1
O2268	5,46	12,9	13,5	43	78,7	23,7	30,1	31,4	24,5	15,4	2,64	0,62	28	24,9	0,4	7,7	0	5,1	0,03	1
B83	5,99	13,1	14	41,9	70	21,9	31,2	33,3	23,1	14,4	2,75	1,07	25,4	23,2	1,2	1,1	0	15,6	0,03	1
D662	5,25	13,3	14,2	41,6	79,2	25,3	32	34,1	26,9	14,3	2,4	1,51	29,2	26,8	0,8	0,5	0	3,5	0,03	1
L1,353	5,81	13,4	14,6	42,8	73,8	23,1	31,4	34	24,9	15,1	2,79	1,75	27,3	24,9	1,4	0,7	0	10	0,02	1
O2225	5,77	13,4	14,2	43,5	75,3	23,2	30,7	32,6	24,4	14,7	2,56	0,78	28,7	24,7	0,4	2,3	0	6,9	0,02	1
O2093	5,72	13,5	14,1	41,3	72,1	23,5	32,6	34,2	24,5	14,9	2,71	0,7	28,2	24,7	1,3	0,7	0	12,1	0,03	1
F670	5,54	13,6	14,3	42,3	76,3	24,6	32,2	33,8	25,6	14,6	2,41	1,4	28	25,7	0,7	0,4	0	6,1	0,04	1
E532	5,49	13,9	14,5	42,8	78,1	25,4	32,6	33,9	26,3	14,6	2,66	1,02	28,5	26,1	1,1	0,6	0	4,7	0,02	1
F694	5,81	14,1	14,9	43,6	75,2	24,2	32,3	34,1	25,5	14	2,54	1	28,5	25,6	1,1	0,4	0	6,4	0,01	1
K1,175	5,92	14,2		44,4	75	24	32			15		1,17	29,1	25,2	0,8	0,6	0	7,9	0,02	1
E420	6,25	14,8	15,9	47	75,2	23,7	31,6	33,8	25,2	14,8	2,74	1,7	27,7	25,1	1,2	0,9	0	7,6	0,01	1
K1,052	6,35	14,8		46,9	73,8	23,3	31,6			14,9		1,72	27,7	24,8	1,4	0,6	0	9,5	0,02	1
N1,914	6,33	15,3	16,2	47,3	74,8	24,1	32,3	34,2	25,4	13,9	2,51	0,78	28,7	25,6	0,9	0,3	0	6,9	0,01	1
E457	6,54	15,5	16,6	48,7	74,5	23,7	31,8	34	25,1	14,8	2,84	1,35	27,5	25	1,8	0,7	0	8,5	0,02	1
O2214	4,66	9	9,4	32,2	69,2	19,4	28	29,3	20,1	18,3	3,18	0,73	24,2	20,2	0,3	34,7	0	23,3	0,06	2
O2325	4,67	9	9,4	33,9	72,6	19,2	26,5	27,6	19,9	19,1	3,22	0,88	24,2	20,6	0,1	57,8	0,1	17,4	0,06	2
K1,127	4,12	9,1		30,8	74,9	22	29,4			16,6		1,94	24,4	23,2	0,4	16,6	0	10,4	0,06	2
N1,955	4,93	9,3	10,1	32	65	18,9	29,1	31,5	20,3	18,3	3,25	0,82	23,5	20,5	0,8	12,6	0	34,9	0,04	2
O1996	5,06	9,4	9,5	32,5	64,3	18,5	28,8	29,2	18,6	19,5	3,7	0,62	22,6	19	0,7	40	0	37,6	0,09	2

DATASET PRE-PROCESSING

01 Data Encoding

02 Handling Missing Values

03 Handling Outliers

04 Normalization

05 Transformation

06 Feature Selection

01 Data Encoding

Reformatting the data into appropriate format

```python
import locale

def replace_string_with_float(df, features):
    locale.setlocale(locale.LC_ALL, 'en_US.UTF-8')  # Set the desired locale

    for feature in features:
        df[feature] = df[feature].apply(lambda x: locale.atof(x) if isinstance(x, str) else x)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113 entries, 0 to 112
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   No. Sample     113 non-null    object
 1   RBC            113 non-null    object
 2   HGB            113 non-null    object
 3   Cellular HGB   101 non-null    object
 4   HCT            113 non-null    object
 5   MCV            113 non-null    float64
 6   MCH            113 non-null    object
 7   MCHC           113 non-null    object
 8   CHCM           101 non-null    object
 9   CH             101 non-null    object
 10  RDW            113 non-null    object
 11  HDW            101 non-null    object
 12  Retic %        113 non-null    float64
 13  CHr            113 non-null    object
 14  CHm            113 non-null    object
 15   RBC % Hyper   113 non-null    object
 16  RBC % Hypo     113 non-null    object
 17  RBC % Macro    113 non-null    object
 18  RBC % Micro    113 non-null    object
 19  RBC Fragment   113 non-null    float64
 20  Group          113 non-null    int64
dtypes: float64(3), int64(1), object(17)
memory usage: 18.7+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113 entries, 0 to 112
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   No. Sample     113 non-null    object
 1   RBC            113 non-null    float64
 2   HGB            113 non-null    float64
 3   Cellular HGB   101 non-null    float64
 4   HCT            113 non-null    float64
 5   MCV            113 non-null    float64
 6   MCH            113 non-null    float64
 7   MCHC           113 non-null    float64
 8   CHCM           101 non-null    float64
 9   CH             101 non-null    float64
 10  RDW            113 non-null    float64
 11  HDW            101 non-null    float64
 12  Retic %        113 non-null    float64
 13  CHr            113 non-null    float64
 14  CHm            113 non-null    float64
 15   RBC % Hyper   113 non-null    float64
 16  RBC % Hypo     113 non-null    float64
 17  RBC % Macro    113 non-null    float64
 18  RBC % Micro    113 non-null    float64
 19  RBC Fragment   113 non-null    float64
 20  Group          113 non-null    int64
dtypes: float64(19), int64(1), object(1)
memory usage: 18.7+ KB
```

02 Handling Missing Values

Filling the missing values with mean

```
def replace_null_with_mean(df, features):
    mean_values = df[features].mean()
    df_filled = df[features].fillna(mean_values)
    df[features] = df_filled
    return df
```

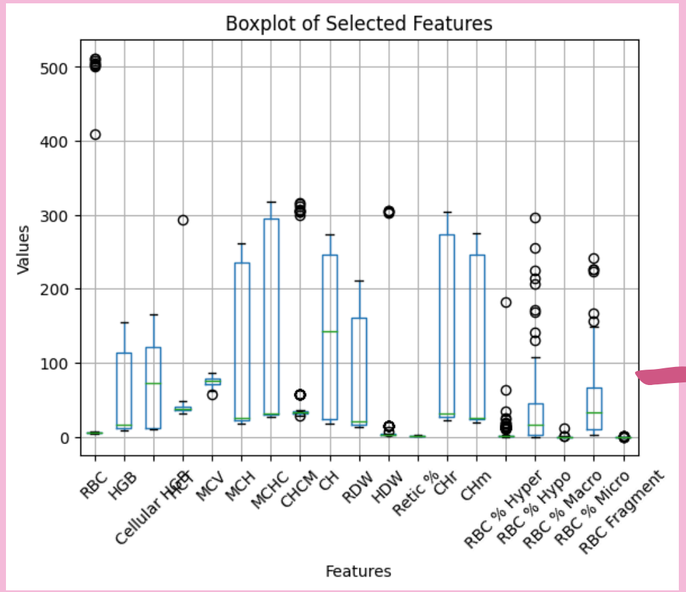
```
count_null_values(df_train)

No. Sample        0
RBC               0
HGB               0
Cellular HGB     12
HCT               0
MCV               0
MCH               0
MCHC              0
CHCM             12
CH               12
RDW               0
HDW              12
Retic %           0
CHr               0
CHm               0
 RBC % Hyper      0
RBC % Hypo        0
RBC % Macro       0
RBC % Micro       0
RBC Fragment      0
Unnamed: 20       0
dtype: int64
```

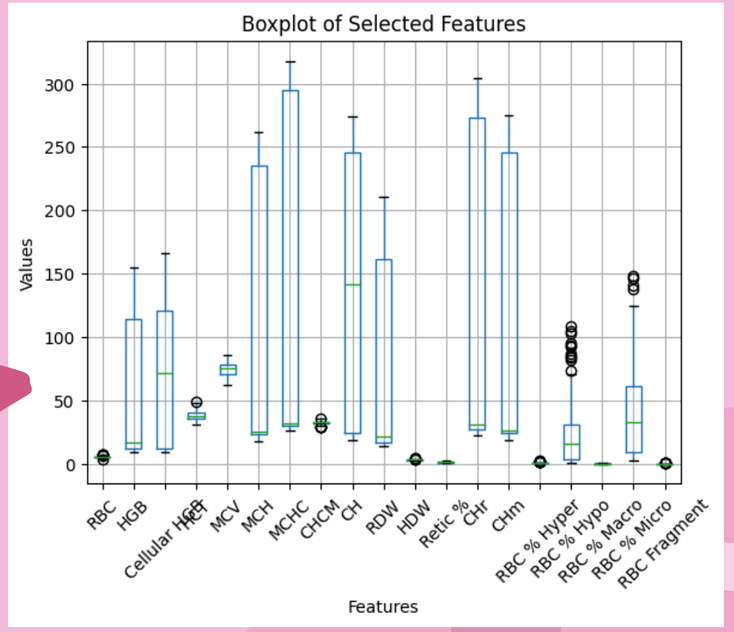
```
count_null_values(df_train)

No. Sample        0
RBC               0
HGB               0
Cellular HGB      0
HCT               0
MCV               0
MCH               0
MCHC              0
CHCM              0
CH                0
RDW               0
HDW               0
Retic %           0
CHr               0
CHm               0
 RBC % Hyper      0
RBC % Hypo        0
RBC % Macro       0
RBC % Micro       0
RBC Fragment      0
Unnamed: 20       0
Group             0
dtype: int64
```

03 Handling Outliers

Filling the outliers with median

```
def handle_outliers_with_median(data, features, threshold=1.5):
    for feature in features:
        # Convert the column to numeric values
        data[feature] = pd.to_numeric(data[feature], errors='coerce')

        # Calculate the IQR for the current column
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1

        # Determine the lower and upper bounds for outliers
        lower_bound = Q1 - (threshold * IQR)
        upper_bound = Q3 + (threshold * IQR)

        # Calculate the median of the feature
        median = np.median(data[feature])
```






```
        # Replace outliers with the median value
        data[feature] = np.where((data[feature] < lower_bound) | (data[feature] > upper_bound), median, data[feature])

    return data
```

04 Normalizations

A crucial preprocessing step in machine learning that helps bring features onto a similar scale. It ensures that no single feature dominates the learning process or introduces bias due to differences in their ranges or units.

```python
from sklearn.preprocessing import StandardScaler

def standardize_features(df, selected_features):
    # Convert the selected features to numeric type
    df[selected_features] = df[selected_features].apply(pd.to_numeric, errors='coerce')

    # Create a new DataFrame with the selected numeric features
    numeric_df = df[selected_features]

    # Perform standardization on the numeric features
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_df)

    # Create a DataFrame from the scaled data
    scaled_df = pd.DataFrame(scaled_data, columns=selected_features)

    # Replace the original numeric features with the scaled values
    df[selected_features] = scaled_df

    return df
```

05 Transformation

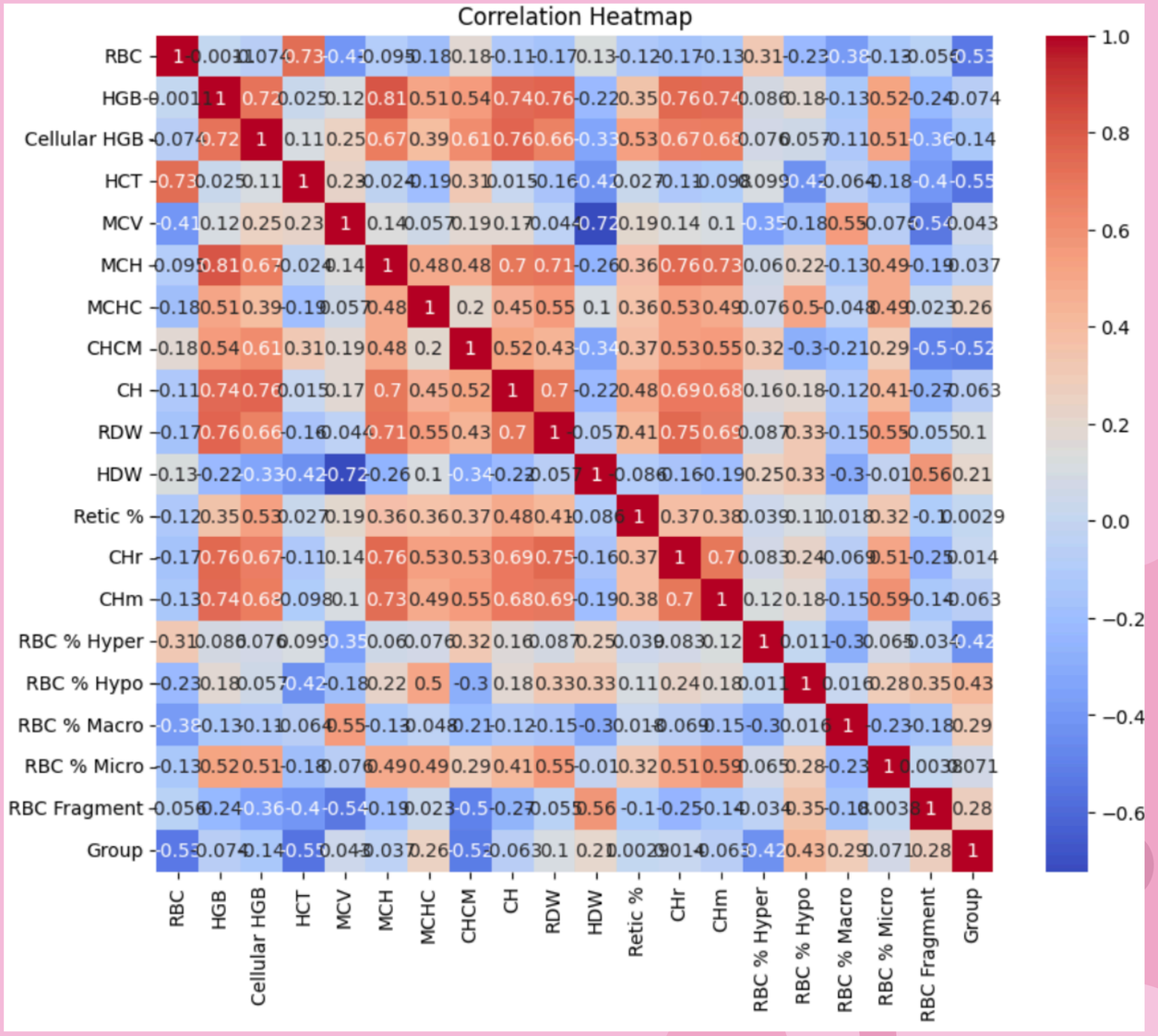
Technique used in data preprocessing to handle skewed distributions and reduce the impact of outliers. It involves applying the natural logarithm function to the numerical features in the dataset.

```
def log_transform(data, selected_features):
    transformed_data = data.copy()

    for feature in selected_features:
        transformed_data[feature] = np.log1p(data[feature])

    return transformed_data
```

06 Feature Selection

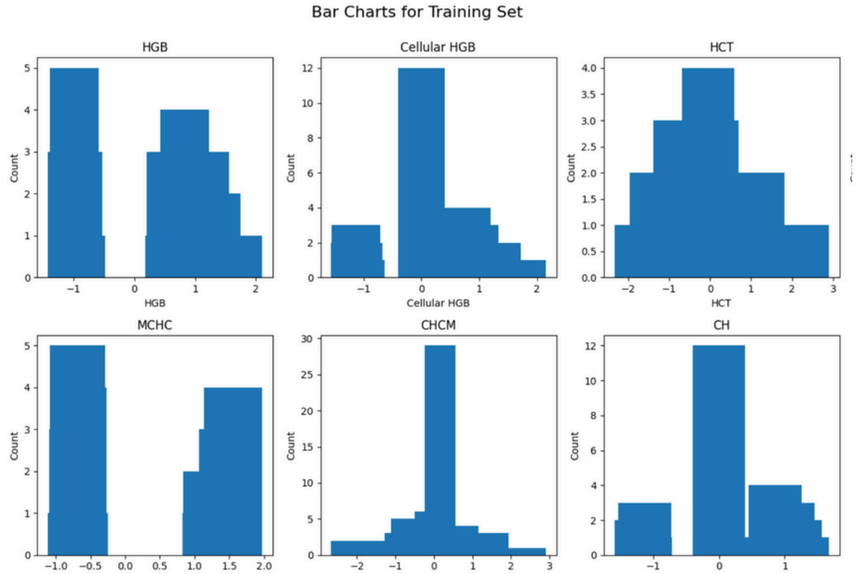
Due to there are several features that have high correlation we participate to have options which features/feature to drop.

```
df_train_norm1=df_train_norm.drop(['MCH', 'CH'], axis=1, inplace=True)
df_train_norm2=df_train_norm.drop('CH', axis=1, inplace=True)
df_train_norm3=df_train_norm.drop('MCH', axis=1, inplace=True)
```

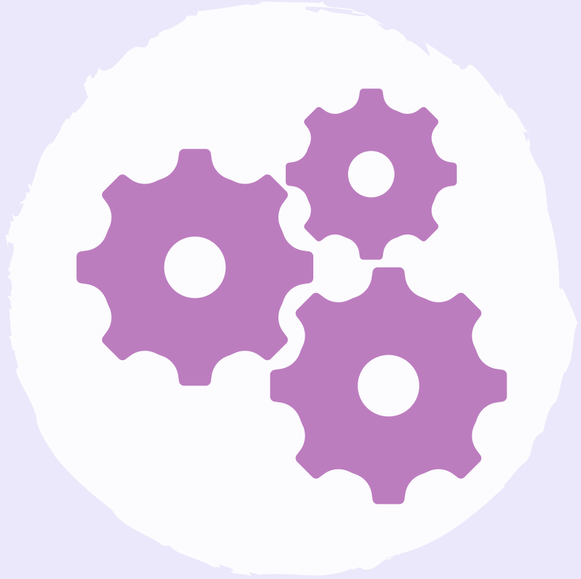


DATASET VISUALIZATION

when we plot the original data into a graph we can see that the data is not balance with a high peak in the middle or the data is split into two region in the right and left side.



MODEL TRAINING
RANDOM FOREST



First Model trained using Model Random Forest

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score

     # Separate the features and target variables in the training set
     X_train = df_train_model.drop('Group', axis=1)
     y_train = df_train_model['Group']

     # Separate the features and target variables in the test set
     X_test = df_test_model.drop('Group', axis=1)
     y_test = df_test_model['Group']

     # Create and train the Random Forest classifier
     model = RandomForestClassifier()
     model.fit(X_train, y_train)

     # Predict the target variable for the test set
     y_pred = model.predict(X_test)

     # Evaluate the model performance
     accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy)
```

```
Accuracy: 0.8214285714285714
Precision: 0.875
Sensitivity: 0.6363636363636364
F1-score: 0.7368421052631579
```

MODEL TRAINING

HYPERTUNED RANDOM FOREST

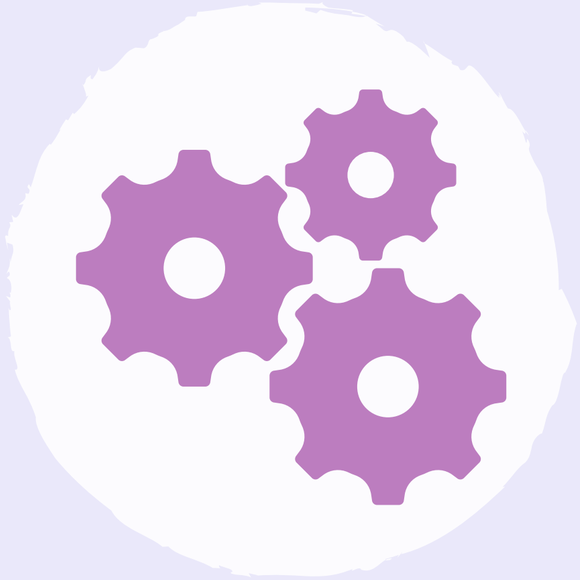
```
from sklearn.ensemble import RandomForestClassifier
                            SearchCV

                            bles in the training set
                           xis=1)

                            d their possible values
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2]
}

# Create the Random Forest classifier
model = RandomForestClassifier()

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", grid_search.best_params_)

# Use the best model to make predictions on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Calculate the performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

Run cell (⌘/Ctrl+Enter)
cell has not been executed in this session

executed by M. Rafi Syafrinaldi
Sunday, June 18, 2023 (2 days ago)
executed in 21.001s

```
Accuracy: 0.8392857142857143
Precision: 0.8823529411764706
Sensitivity: 0.6818181818181818
F1-score: 0.7692307692307693
```

MODEL TRAINING
SUPPORT VECTOR MACHINES



```python
from sklearn.svm import SVC

# Create an SVM model
model = SVC()

# Train the model on the training data
model.fit(df_train_model.drop('Group', axis=1), df_train_model['Group'])

# Make predictions on the test data
predictions = model.predict(df_test_model.drop('Group', axis=1))

# Evaluate the model performance
accuracy = accuracy_score(df_test_model['Group'], predictions)
precision = precision_score(df_test_model['Group'], predictions)
recall = recall_score(df_test_model['Group'], predictions)
f1 = f1_score(df_test_model['Group'], predictions)

# Print the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8214285714285714
Precision: 0.7727272727272727
Sensitivity: 0.7727272727272727
F1-score: 0.7727272727272727
```

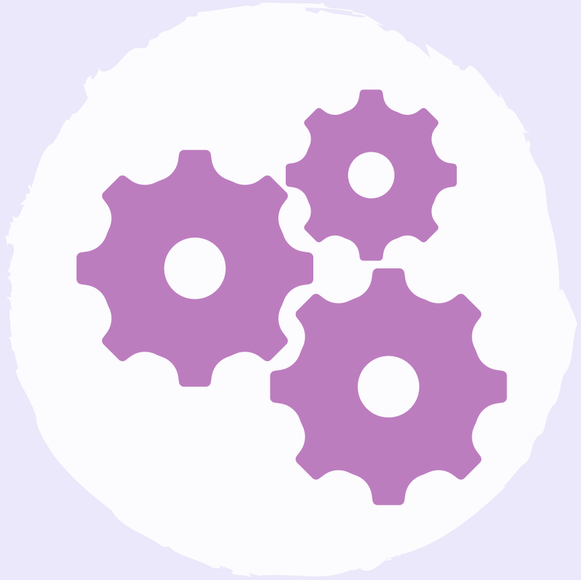
MODEL TRAINING

HYPERTUNED SVMS

```python
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define the hyperparameters and their possible values
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

# Create an SVM model
model = SVC()

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(df_train_model.drop('Group', axis=1), df_train_model['Group'])

# Print the best hyperparameters found
print("Best Hyperparameters:", grid_search.best_params_)

# Use the best model to make predictions on the test data
best_model = grid_search.best_estimator_
predictions = best_model.predict(df_test_model.drop('Group', axis=1))

# Calculate the performance metrics
accuracy = accuracy_score(df_test_model['Group'], predictions)
precision = precision_score(df_test_model['Group'], predictions)
recall = recall_score(df_test_model['Group'], predictions)
f1 = f1_score(df_test_model['Group'], predictions)

# Print the performance metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8571428571428571
Precision: 0.85
Sensitivity: 0.7727272727272727
F1-score: 0.8095238095238095
```

MODEL TRAINING
DECISION TREE



First Model trained using Model Random Forest

```
[ ]  model = DecisionTreeClassifier()
     model.fit(X_train, y_train)

     y_pred = model.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     precision = precision_score(y_test, y_pred)
     sensitivity = recall_score(y_test, y_pred)
     f1 = f1_score(y_test, y_pred)
```

```
Accuracy: 0.8571428571428571
Precision: 0.8888888888888888
Sensitivity: 0.7272727272727273
F1-score: 0.7999999999999999
```

MODEL TRAINING
HYPERTUNED DECISION TREE

```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
best_model = DecisionTreeClassifier(max_depth=5, min_samples_leaf=1, min_samples_split=10, criterion='entropy')
best_model.fit(X_train, y_train)

y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
sensitivity = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

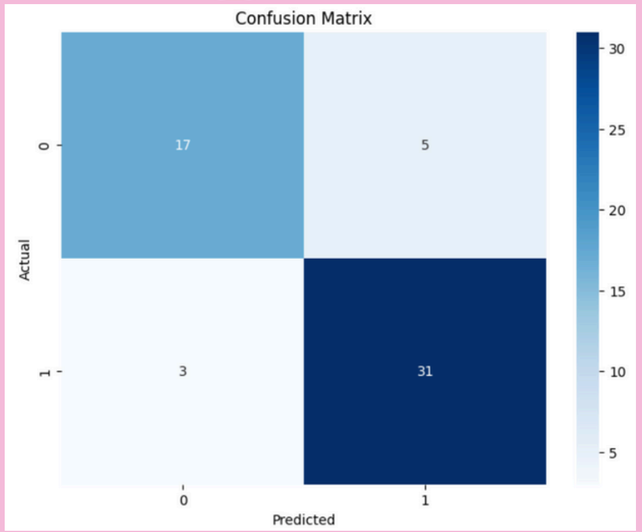
```
Accuracy: 0.8035714285714286
Precision: 0.761904761904761
Sensitivity: 0.7272727272727273
F1-score: 0.7441860465116279
```

EVALUATION

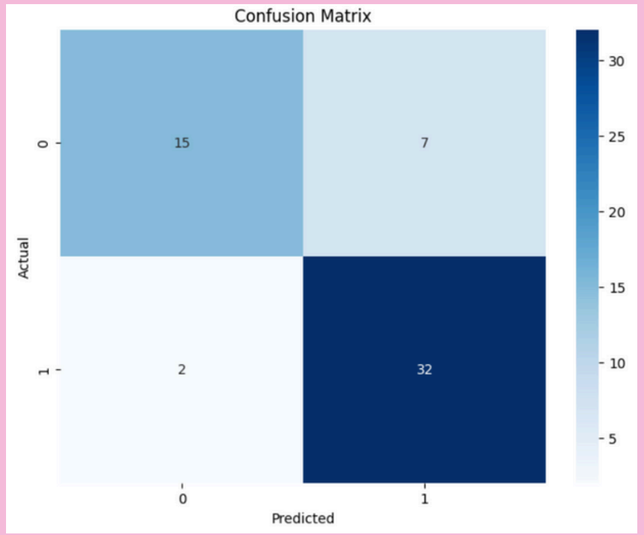
PERFORMANCE

All model perform well with an average correct guess at 46 and after hyperparameter tuning goes up to 47 correct guess.

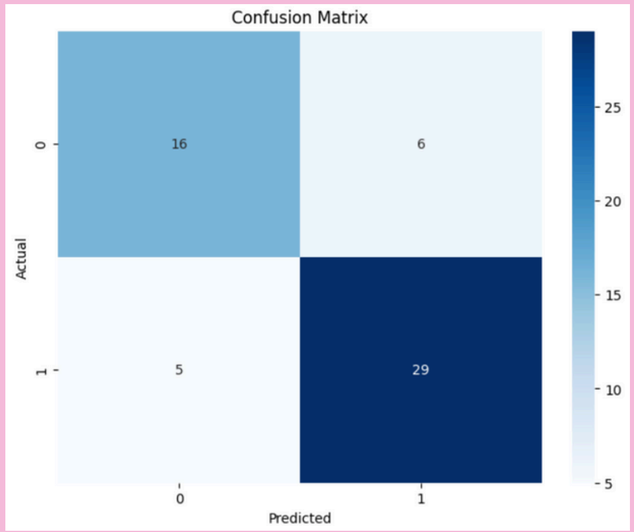
RESULT



SVM



Random Forest



Decision Tree

RESULT

DISCUSSION

From our testing the result that we get shows that SVM performance the best compare to other model with 48 correct guess and 8 wrong guess followed by Random Forest with 47 correct guess and 9 wrong guess and last decision tree with 45 correct guess and 11 wrong guess.

THANK YOU

Google Colab Link:
https://colab.research.google.com/drive/10NUQ49y5SwIUdgUUk-wvDgUEu1u4JhpZ#scrollTo=FKLHbdC_GvjX