Akram Naoufel Tabet

Student ID – w1979246

1. **Task (1) – Domain Understanding: Classification**

Attribute Name	**Retain or Drop**
Test ID	**Drop**
Systemic Illness	**Drop**
Sore Throat	Retain
Rectal Pain	Retain
Penile Oedema	Retain
Oral Lesions	Retain
Solitary Lesion	Retain
Swollen Tonsils	Retain
HIV Infection	Retain
Red blood cells	Retain
White blood cells	Retain
Home ownership	**Drop**
Age	Retain
Month of Birth	**Drop**
Health Insurance	**Drop**
Sexually Transmitted Infection	Retain
MPOX	Retain

The decision to retain or drop specific attributes during data preprocessing depends on various factors, including their relevance to the machine learning task, redundancy, and potential noise they might introduce.

1. **Test ID**: Dropped
    - Reason: It's an identifier for the test, which doesn't contribute to the prediction task. Dropping it helps reduce unnecessary information.
2. **Systemic Illness**: Dropped

- Reason: It is because it's encoded in another form (`Encoded Systemic Illness`). If the encoded version is retained, having both might introduce redundancy.

3. **Home ownership, Month of Birth, Health Insurance**: Dropped
   - Reason: These attributes were dropped, because they either don't provide valuable information for the prediction task.

2. Task (2) – Data Understanding: Producing Your Experimental Designing

1. Basic Statistical Description:

*Task (2) – Data Understanding *

```
retained_columns=list(datasave.columns)
datasave.describe()
```

	Encoded Systemic Illness	Rectal Pain	Sore Throat	Penile Oedema	Solitary Lesion	Swollen Tonsils	HIV Infection	Red blood cells count	White blood cells count	Sexually Transmitted Infection
count	24998.000000	24997.000000	25000.000000	24994.000000	25000.000000	24993.000000	24995.000000	2.500000e+04	25000.000000	24996.000000
mean	1.497640	0.493819	0.502160	0.504441	0.501080	0.501260	0.503301	5.004591e+06	7749.114440	0.497880
std	1.116872	0.499972	0.500005	0.499990	0.500009	0.500008	0.499999	5.204760e+05	1885.213591	0.500006
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.100135e+06	4500.000000	0.000000
25%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.555418e+06	6116.000000	0.000000
50%	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.002304e+06	7747.000000	0.000000
75%	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.458530e+06	9379.000000	1.000000
max	3.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.899806e+06	11000.000000	1.000000

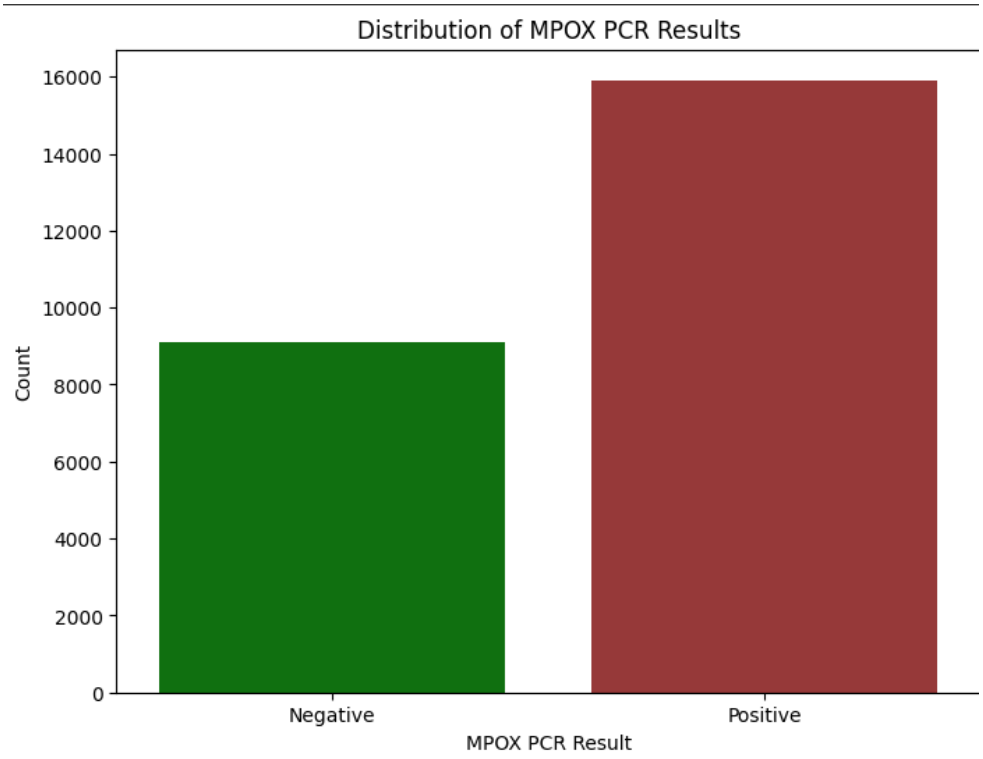
2. **Measurement Scale Type**:

```
datasave.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 13 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Encoded Systemic Illness        24998 non-null  float64
 1   Rectal Pain                     24997 non-null  float64
 2   Sore Throat                     25000 non-null  int64
 3   Penile Oedema                   24994 non-null  float64
 4   Oral Lesions                    24996 non-null  object
 5   Solitary Lesion                 25000 non-null  int64
 6   Swollen Tonsils                 24993 non-null  float64
 7   HIV Infection                   24995 non-null  float64
 8   Red blood cells count           25000 non-null  int64
 9   White blood cells count         25000 non-null  int64
 10  Age                             24964 non-null  object
 11  Sexually Transmitted Infection  24996 non-null  float64
 12  MPOX PCR Result                 25000 non-null  object
dtypes: float64(6), int64(4), object(3)
memory usage: 2.5+ MB
```

```
[7] measurement_scale = datasave[retained_columns].dtypes
    print(measurement_scale)

    Encoded Systemic Illness          float64
    Rectal Pain                       float64
    Sore Throat                         int64
    Penile Oedema                     float64
    Oral Lesions                       object
    Solitary Lesion                     int64
    Swollen Tonsils                   float64
    HIV Infection                     float64
    Red blood cells count               int64
    White blood cells count             int64
    Age                                object
    Sexually Transmitted Infection    float64
    MPOX PCR Result                    object
    dtype: object
```

3. Distribution of the class variable



3. **Task (3) – Data Preparation: Cleaning and Transforming your data.**

a)

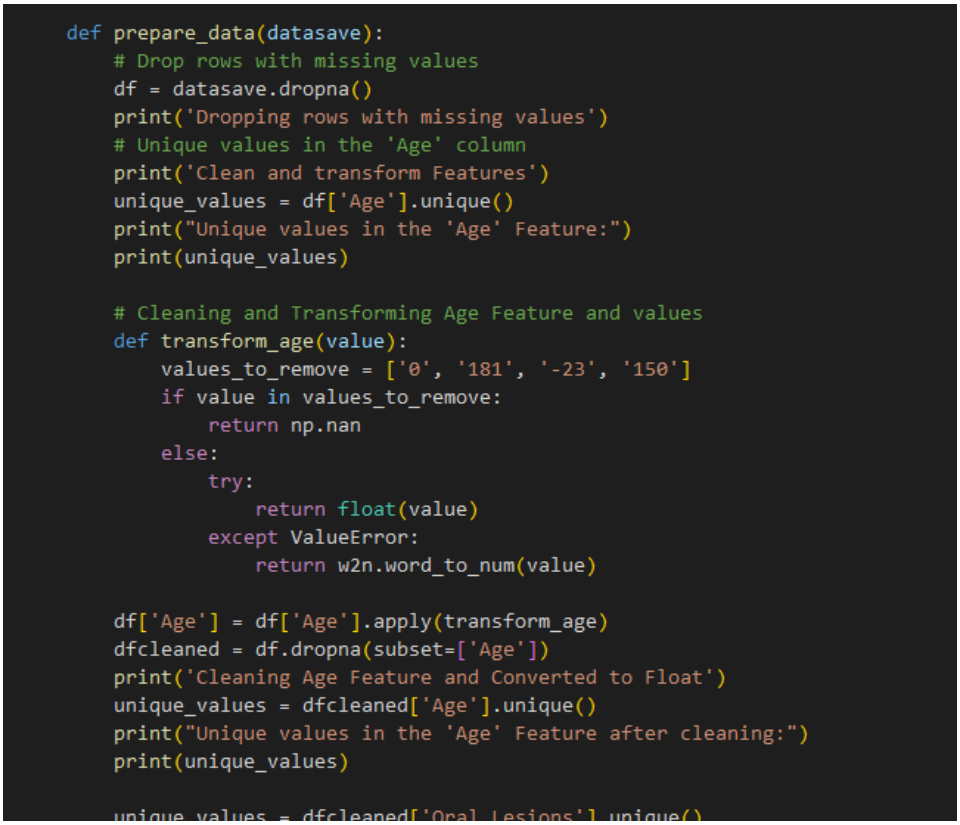
Dataset or Variable Issue?	**Name of variable**	**Issue description**
Dataset Issue	Null Values	The issue with the dataset is that it has missing values.
Variable Issue	Age	The issue with this variable is that it contains values in negative values, unusual data and values in different formats.
Variable issue	White Blood cell and Red Blood Cell	For example, if you look at the WBC count in the statistical description, the values are huge, and they differ from the rest of the values.
Variable Issue	Oral Lesions	The issue with this variable is that it contains some values in 'YES' and 'NO' and the rest in 0 and 1.

b)

Dataset or Variable Issue?	Name of variable	The Issue	Solution	Justification
Dataset Issue	Null Values	The issue with the dataset is that it contains missing values.	Remove them(drop rows or instances)	We have lots of instances of 25000 and very few missing values(less than 70), so no difference is made when imputing them with mean or dropping them.
Variable issue	Age	Negative values, unusual data and values in different formats.	Remove the null value, remove the unusual data, we changed all the formats to float.	Because the unusual values are outliers, as age is numerical, we make everything float

Variable issue	White Blood cell	Huge values and they differ from the rest of the values.	Min-Max Scaling(Normaliz-ation)	Because if we normalize the values, they are all in the same scale type(range 0 to 1).
Variable Issue	Oral Lesions	Values are in ‘YES’ and ‘NO’ and the rest in 0 and 1.	Transform the values into 0 and 1.	Because if we normalize the values, they are all in the same scale type.
Variable Issue	Red Blood Cell	Huge values and they differ from the rest of the values.	Min-Max Scaling(Normaliz-ation)	Because if we normalize the values, they are all in the same scale type.
Variable Issue	Encoded Systemic Illness	categorical data ranging from 0 to 3	Min-Max Scaling(Normaliz-ation)	It is not necessary to scale it since it's continuous data**(i didn’t normalize it)**

C) For the suggested solution for the preparation, i have created a function containing the whole process of the preprocessing starting from dropping the instances that contains missing values to split data into train and test then normalizing it as well. The first capture shows the code of function and the second one shows the outputs.



```
def prepare_data(datasave):
    # Drop rows with missing values
    df = datasave.dropna()
    print('Dropping rows with missing values')
    # Unique values in the 'Age' column
    print('Clean and transform Features')
    unique_values = df['Age'].unique()
    print("Unique values in the 'Age' Feature:")
    print(unique_values)

    # Cleaning and Transforming Age Feature and values
    def transform_age(value):
        values_to_remove = ['0', '181', '-23', '150']
        if value in values_to_remove:
            return np.nan
        else:
            try:
                return float(value)
            except ValueError:
                return w2n.word_to_num(value)

    df['Age'] = df['Age'].apply(transform_age)
    dfcleaned = df.dropna(subset=['Age'])
    print('Cleaning Age Feature and Converted to Float')
    unique_values = dfcleaned['Age'].unique()
    print("Unique values in the 'Age' Feature after cleaning:")
    print(unique_values)

    unique_values = dfcleaned['Oral Lesions'].unique()
```

```
Dropping rows with missing values
Clean and transform Features
Unique values in the 'Age' Feature:
['37' '24' '34' '40' '36' '30' '23' '41' '32' '46' '27' '47' '53' '31'
 '25' '26' '52' '51' '56' '39' '35' '50' '33' '28' '45' '38' '57' '55'
 '43' '60' '61' '42' '59' '44' '48' '49' '58' '54' '150' '29' '0' 'Twenty'
 '181' '-23']
Cleaning Age Feature and Converted to Float
Unique values in the 'Age' Feature after cleaning:
[37. 24. 34. 40. 36. 30. 23. 41. 32. 46. 27. 47. 53. 31. 25. 26. 52. 51.
 56. 39. 35. 50. 33. 28. 45. 38. 57. 55. 43. 60. 61. 42. 59. 44. 48. 49.
 58. 54. 29. 20.]
Unique values in the 'Oral Lesions' Feature:
['1' '0' 'YES' 'No']
Cleaning and transforming Oral Lesions Feature and values
Unique values in the 'Oral Lesions' Feature after cleaning:
[1. 0.]
Transforming target variable into 0 and 1
Splitting into features (X) and target variable (y)
Train-Test split
X_train before normalizing
       Encoded Systemic Illness  Rectal Pain  Sore Throat  Penile Oedema  \
20491                       2.0          0.0            0            1.0
3287                        1.0          1.0            1            1.0
16673                       0.0          0.0            1            0.0
5706                        2.0          0.0            0            1.0
18428                       3.0          1.0            0            1.0
...                         ...          ...          ...            ...
3392                        1.0          0.0            0            0.0
3094                        0.0          0.0            0            1.0
23151                       3.0          0.0            0            1.0
9444                        0.0          1.0            0            0.0
8217                        1.0          1.0            0            0.0
```

This function streamlines the preprocessing steps and considers the separation of training and testing data before normalization. Here's a detailed summary:

1. Data Cleaning and Transformation:
    - Handling Missing Values: Dropping rows with missing values (e.g., using `dropna()`).
    - Cleaning Age Feature: Transforming the 'Age' column to float by handling specific values and converting words to numbers.
    - Handling Categorical Data: Converting categorical data like 'Oral Lesions' to numeric format.
    - Transforming Target Variable: Converting the 'MPOX PCR Result' column to binary (e.g., 1 for 'Positive', 0 for 'Negative').
2. Data Splitting:
    - Using `train_test_split` to split the data into training and testing sets (e.g., 75% training, 25% testing).
3. Normalization:
    - Utilizing `MinMaxScaler` to normalize selected columns ('Red blood cells count', 'White blood cells count', 'Age') to the range [0, 1].
    - Applying normalization separately to training and testing sets to prevent data leakage.

```
[18697 rows x 12 columns]
Normalize count columns using MinMaxScaler
Fit on the training data and transform both training and testing data
Done preprocessing Data
Normalized X_train:
       Encoded Systemic Illness  Rectal Pain  Sore Throat  Penile Oedema  \
20491                       2.0          0.0            0            1.0
3287                        1.0          1.0            1            1.0
16673                       0.0          0.0            1            0.0
5706                        2.0          0.0            0            1.0
18428                       3.0          1.0            0            1.0

       Oral Lesions  Solitary Lesion  Swollen Tonsils  HIV Infection  \
20491           0.0               0              1.0            0.0
3287            1.0               1              1.0            1.0
16673           0.0               1              1.0            1.0
5706            0.0               1              1.0            0.0
18428           0.0               0              1.0            1.0
```

4. **Task (4) – Modeling: Create Predictive Classification Models.**

   NB. For the Hyperparameters i have just mentioned the possible ones and what values it might take for every algorithm, i did not mention the detailed explanation for every hyperparameters like C means for regularization etc.. I have mentioned them in a machine learning project way .

Algorithm Name	**Type of Algorithm**	**Learnable Parameters**	**Possible Hyper - Parameters**	**Python package source code to call the algorithm**
LR	Supervised learning Logistic Regression	['coef_', 'intercept_'] = coefficients and intercept	{'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000], 'fit_intercept': [True, False], 'solver': ['liblinear', 'saga'], 'max_iter': [100, 200, 300, 400, 500]}	from sklearn.linear_model import LogisticRegression
DT	Supervised learning DT	['tree_'] =split points and threshold for each features	{'criterion': ['gini', 'entropy'], 'splitter': ['best', 'random'], 'max_depth': [None, 10, 20, 30, 40, 50], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}	from sklearn.tree import DecisionTreeClassifier
KNN	Supervised learning KNN	None during training.	{'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance'], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'p': [1, 2]}	from sklearn.neighbors import KNeighborsClassifier

SVM (RBF)	Supervised learning SVM (RBF)	['dual_coef_', 'support_', 'n_support_'] = support vectors and coefficients	{'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto'], 'kernel': ['rbf']}	from sklearn.svm import SVC
NB	Supervised learning NB	['class_count_', 'feature_count_'] And probabilities	Smoothing parameters,etc..	from sklearn.naive_bayes import GaussianNB from sklearn.naive_bayes import BernoulliNB

b) The choice of a 75-25 training-test split ratio is commonly used and strikes a balance between having sufficient data for model training and a substantial dataset for evaluation. This ratio is supported by widely accepted practices in machine learning literature (Hastie, Tibshirani, & Friedman, 2009). The code line ensuring consistency in the test dataset across models is exemplified by utilizing `train_test_split` with the `stratify` parameter, maintaining the MPOX Negatives to MPOX Positives ratio between training and test set. To ensure as well that all models were tested on the same test dataset,we should use the random state=42 in all the algorithms.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)
```

X_train with the features used for training including the shape:

X_train_normalized

	Encoded Systemic Illness	Rectal Pain	Sore Throat	Penile Oedema	Oral Lesions	Solitary Lesion	Swollen Tonsils	HIV Infection	Red blood cells count	White blood cells count	Age	Sexually Transmitted Infection
20491	2.0	0.0	0	1.0	0.0	0	1.0	0.0	0.666039	0.822434	0.634146	0.0
3287	1.0	1.0	1	1.0	1.0	1	1.0	1.0	0.366862	0.487613	0.975610	0.0
16673	0.0	0.0	1	0.0	0.0	1	1.0	1.0	0.047739	0.935990	0.268293	0.0
5706	2.0	0.0	0	1.0	0.0	1	1.0	0.0	0.320062	0.733036	0.951220	0.0
18428	3.0	1.0	0	1.0	0.0	0	1.0	1.0	0.213070	0.786275	0.341463	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
3392	1.0	0.0	0	0.0	0.0	0	0.0	1.0	0.040699	0.212648	0.317073	1.0
3094	0.0	0.0	0	1.0	1.0	0	1.0	1.0	0.025216	0.040006	0.780488	0.0
23151	3.0	0.0	0	1.0	0.0	0	0.0	1.0	0.144319	0.155870	0.268293	0.0
9444	0.0	1.0	0	0.0	1.0	0	0.0	1.0	0.249160	0.092476	0.146341	0.0
8217	1.0	1.0	0	0.0	1.0	1	1.0	0.0	0.083528	0.877827	0.975610	0.0

18697 rows × 12 columns

For the building models part,I have created another function called `run_classification_algorithm` that takes an algorithm name, training, and testing data, and returns a dictionary containing model information and evaluation metrics. The function covers Logistic Regression, K-Nearest Neighbors, Decision Tree, Support Vector Machine with RBF Kernel, Gaussian Naive Bayes, and Bernoulli Naive Bayes.

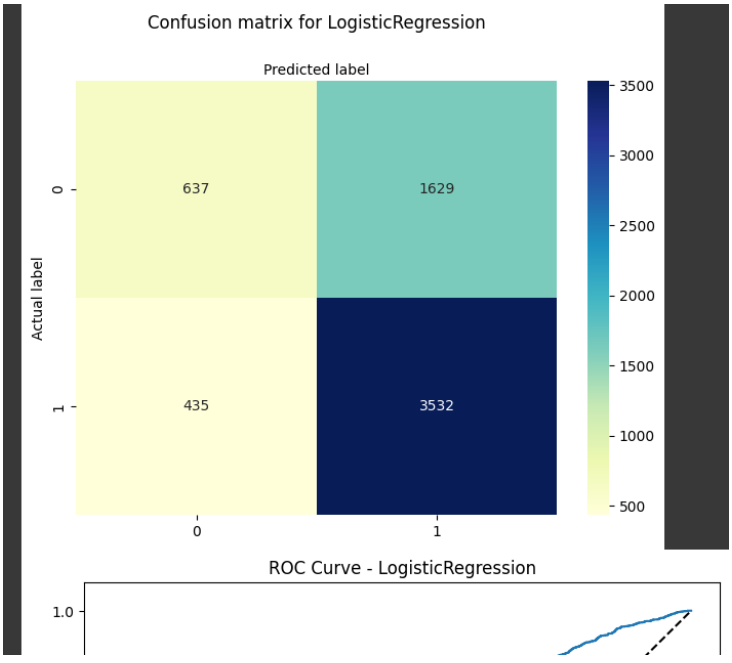
- **Modeling Section:** It includes the chosen algorithm, learnable parameters, hyperparameters, and the source code to call the algorithm.
- **Evaluation Section:** It comprises key evaluation metrics such as accuracy, precision, recall, F1 score, confusion matrix, AUC-ROC, false positive rate (fpr), true positive rate (tpr), and the classification report.

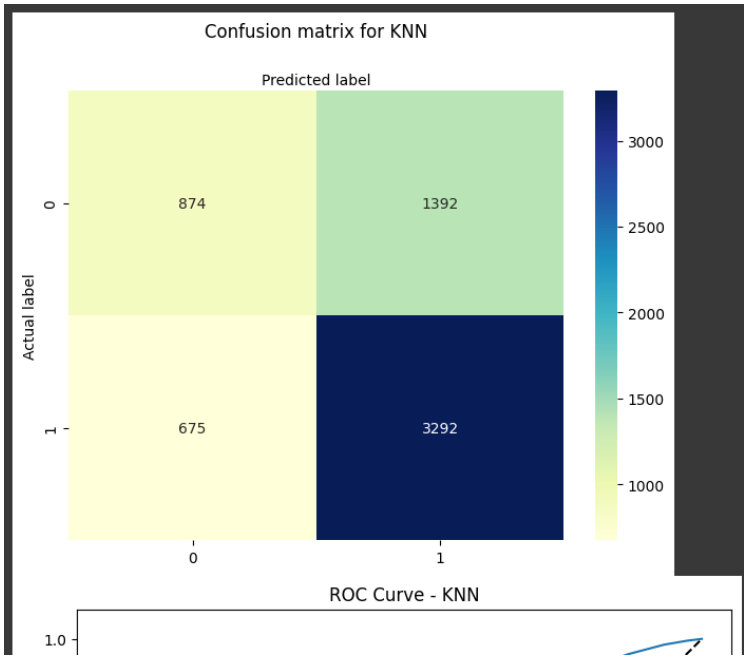
The function is then applied to multiple algorithms in a loop, and the results are stored in a dictionary called `resultsAll`, where each algorithm's results are indexed with an integer key.

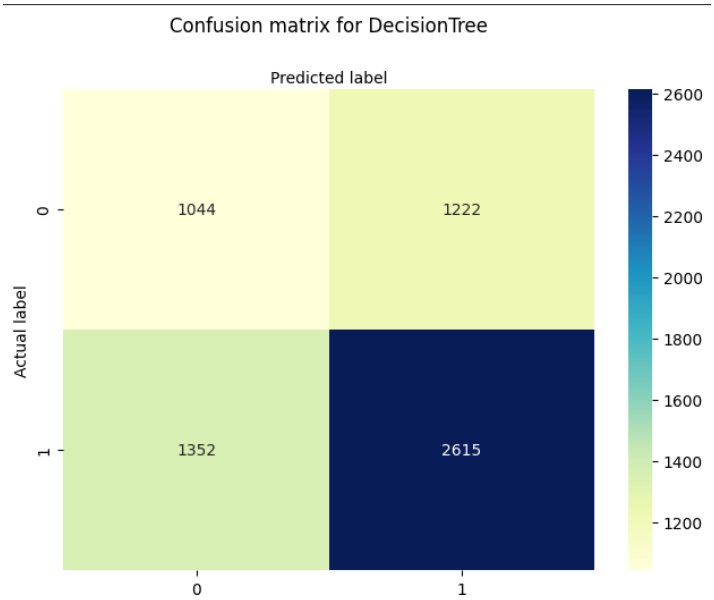
```
    # Results dictionary
    results = {
       'Modeling':{
        'Algorithm': algorithm,
        'LearnableParameters': learnable_parameters,
        'HyperParameters': hyperparameters,
        'SourceCode': source_code},
        'Evaluation':{
        'Accuracy': accuracy,
        'Precision':precision,
        'Recall':recall,
        'f1':f1,
        'confusion_matrix':cnf_matrix,
        'AUC-ROC': auc_roc,
        'fpr':fpr,
        'tpr':tpr,
        'classification_rep':classification_rep}
    }

    return results

resultsAll={}
i=0
algorithms_to_run = ['LogisticRegression', 'KNN', 'DecisionTree', 'SVM', 'NaiveBayesGaussian', 'NaiveBayesBernoulli']
for algorithm in algorithms_to_run:
    result= run_classification_algorithm(algorithm, X_train_normalized, X_test_normalized, y_train, y_test)
    resultsAll[i]=result
    i+=1
```

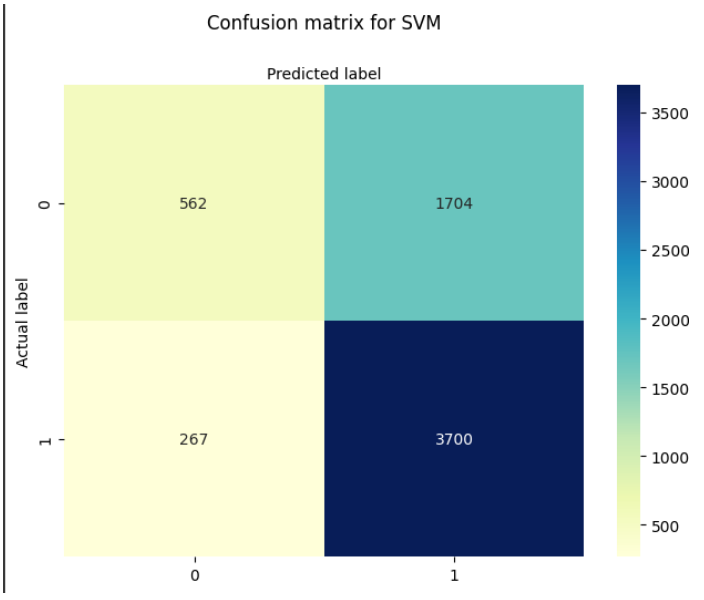
5. Task (5) – Evaluation: How good are your model

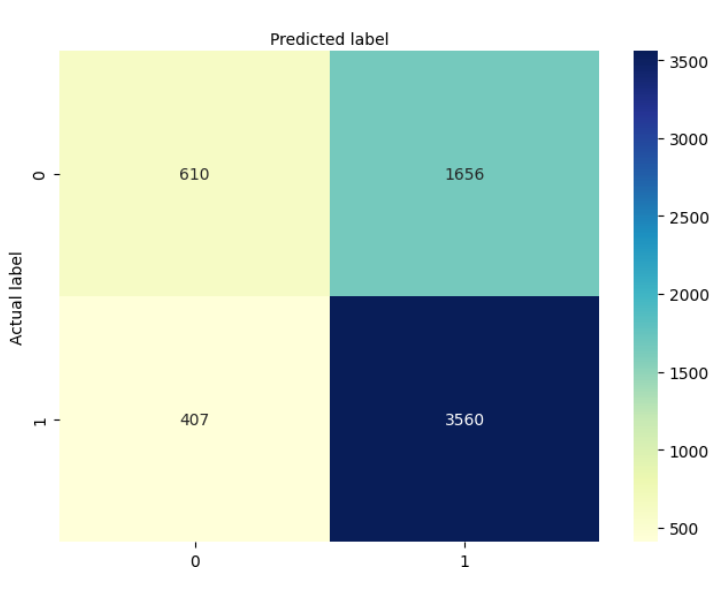
a) The test confusion matrix for each trained model,I have created as well the Roc-curve to see how the model is able to distinguish between the positive and negative values. I got all the evaluation metrics from the dictionary that I created.

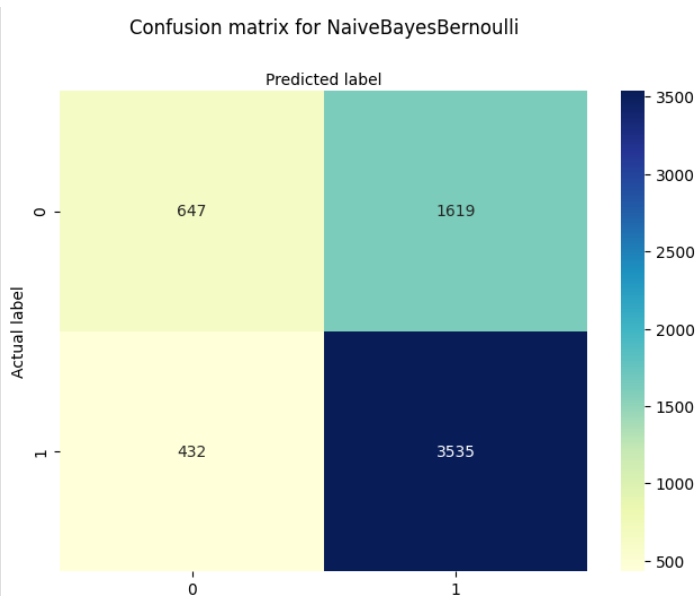












b) The success criteria in our project might be related to the effectiveness of the models in accurately predicting cases of MPOX positivity,that means the primary goal is to detect individuals who are sick with MPOX, minimizing false negatives (i.e., individuals who are sick but predicted as not sick) becomes crucial. In this case, the following metrics become particularly important:

1. **High Recall:** We want to maximize the number of true positives relative to all actual positive cases. This helps in capturing as many individuals with MPOX as possible.
2. **F1 Score:** While high recall is essential, a good balance with precision is achieved through the F1 score, which considers both false positives and false negatives.

3. **AUC-ROC Score:** This metric is valuable for assessing the model's ability to discriminate between positive and negative cases. A high AUC-ROC indicates a good overall performance.

In summary, our focus should be on metrics that prioritize the identification of individuals with MPOX, and these metrics will guide the evaluation of our models.

Metric Name	"USE" or "DO NOT USE"	Justification in relation to the success criteria	Model Name	Metric Score
Accuracy	**Do not use**	Accuracy may not be the primary focus in an imbalanced dataset(as in our case Class variable) is imbalanced i.e high positive and low negative value in Mpox.	**LR**	**0.668**
			DT	**0.587**
			KNN	**0.668**
			SVM (RBF)	**0.683**
			NB	**0.669(Gau) 0.670(Ber)**
Recall	**Use**	The most critical metric for our project. It ensures that the model captures as many true positive cases (individuals with MPOX) as possible, minimizing the risk of missing actual cases.	**LR**	**0.890**
			DT	**0.659**
			KNN	**0.829**
			SVM (RBF)	**0.932**
			NB	**0.897(Gau) 0.891(Ber)**
Precision	**Do not use**	Captures less true positives compared to recall(it takes a lower priority).	**LR**	**0.684**
			DT	**0.681**
			KNN	**0.702**
			SVM (RBF)	**0.684**
			NB	**0.682(Gau) 0.685(Ber)**
F-Measure	**Use**	Useful in imbalanced class variable,when both false positives and	**LR**	**0.773**
			DT	**0.670**
			KNN	**0.761**
			SVM (RBF)	**0.789**

		false negatives need to be minimized.	**NB**	**0.775(Gau) 0.775(Ber)**
AUC-ROC	**Use**	A high AUC-ROC indicates the model's ability to distinguish between individuals with and without MPOX.	**LR**	**0.675**
			DT	**0.559**
			KNN	**0.648**
			SVM (RBF)	**0.678**
			NB	**0.657(Gau) 0.663(Ber)**

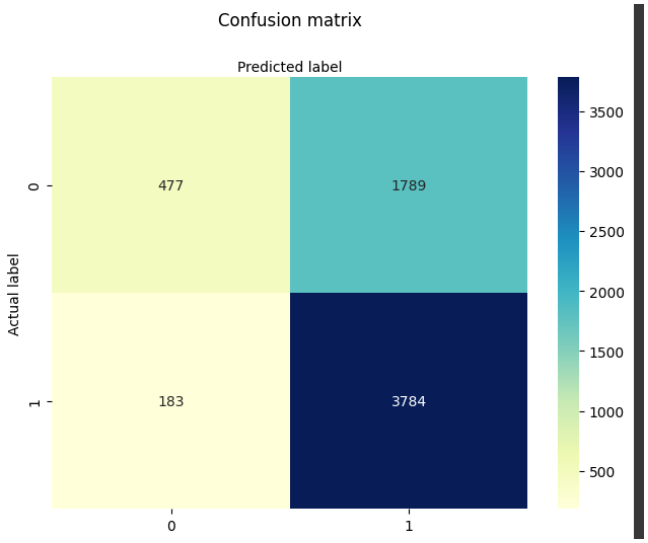
c) In comparison to other models evaluated, the Support Vector Machine (SVM) with an RBF kernel demonstrates superior performance in key metrics. With an accuracy of 0.6838, precision of 0.6847, recall of 0.9327, and an F1 score of 0.7897, the SVM model effectively balances the trade-off between correctly identifying individuals with MPOX (high recall) and minimizing false negatives. Additionally, the AUC-ROC score of 0.6782 indicates strong discriminatory power. Considering these metrics and the specific healthcare needs to detect individuals with MPOX, the SVM model emerges as the preferred choice among the evaluated models. Even if it has more value in terms of false positives which are healthy and predicted as sick we can send them to further treatment and the result will show as healthy as well.

d) The SVM model with RBF kernel was fine-tuned using GridSearchCV with 10-fold cross-validation.

The optimal hyperparameters were identified as {'C': 0.1, 'gamma': 1}. After tuning, the model exhibited improved recall (0.9539), F1 score (0.7933), and maintained high accuracy (0.6836). The area under the ROC curve (AUC-ROC) also increased to 0.680. This suggests that hyperparameter tuning enhanced the model's generalization and performance, particularly in correctly identifying positive cases.It also produced a confusion matrix of [[477, 1789], [183, 3784]]. Notably, there was a reduction in the number of false negatives (183) compared to the previous model, indicating an improvement in correctly identifying individuals with MPOX. This reduction aligns with the goal of the project, emphasizing the positive impact of hyperparameter tuning on the model's ability to detect true positive cases.

```
# Print evaluation metrics
print('best hyperparameters')
print(grid_search.best_params_)
print(f"Recall: {recall:.4f}")
print(auc_roc)
print(f"F1 Score: {f1:.4f}")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
```

```
best hyperparameters
{'C': 0.1, 'gamma': 1}
Recall: 0.9539
None
F1 Score: 0.7933
Accuracy: 0.6836
Precision: 0.6790
```



e) **Ensemble Learning with SVM and Bernoulli Naive Bayes**

In the ensemble learning approach, we combined the SVM with RBF Kernel and Bernoulli Naive Bayes classifiers, each exhibiting promising results individually. The choice was based on their respective performance metrics and alignment with the success criteria of our project, aimed at accurately detecting individuals with mpox.

```
ensemble_learners = [('SVM', svm_rbf), ('BernoulliNB', bnb_model)]
voting_classifier_model = train_evaluate_voting_classifier(ensemble_learners, X_train_normalized, y_train, X_test_normalized, y_test)
```

For the code,I have implemented another function for training and evaluating the ensemble classifier.

Comparison and Observations:

- The Voting Classifier, combining SVM and Bernoulli Naive Bayes, shows competitive performance.
- While the Voting Classifier achieved slightly higher accuracy and precision, SVM demonstrated better recall and F1 score.
- SVM with RBF Kernel remains the preferred choice due to its marginally superior overall performance and reduced false negatives, aligning with our project's success criteria.

f) The selected Support Vector Machine (SVM) with an RBF kernel emerged as the best-performing model for MPOX detection. SVM demonstrated superior recall (93.27%) and an overall F1 score of 78.97%, aligning with our project's crucial success criteria of minimizing false negatives and improving true positives. The algorithm's ability to capture patterns in complex datasets, like those in healthcare, contributed to its success. However, the model has limitations. It might struggle with large datasets (that's why now they are using the recent technologies like xgboost and deep neural networks images based to detect mpox) , and the fine-tuning process is computationally intensive(it took me 3h). Ethically, using this model for MPOX screening raises concerns related to privacy and potential biases, especially if it's deployed without rigorous validation across diverse demographic groups. Additionally, the model is only as robust as the data it's trained on, emphasizing the importance of continual evaluation and updates to maintain accuracy and fairness in real-world applications.

```
Results for Ensemble (Voting Classifier):
Accuracy: 0.6859
Precision: 0.6910
Recall: 0.9161
F1 Score: 0.7878
AUC-ROC: 0.6824
Confusion Matrix:
[[ 641 1625]
 [ 333 3634]]
```