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

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

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

Basic statistical summary measures for numerical attributes

```
df.describe()
```

| | Encoded Systemic Illness | Rectal Pain | Sore Throat | Penile Oedema | Solitary Lesion | Swollen Tonsils | HIV Infection | RBC Count | WBC Count | Home ownership | Birth Month | Health Insurance | Sexually Transmitted Infection |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 24998.000000 | 24997.000000 | 25000.000000 | 24994.000000 | 25000.000000 | 24993.000000 | 24995.000000 | 2.500000e+04 | 25000.000000 | 25000.000000 | 25000.000000 | 25000.000000 | 24996.000000 |
| mean | 1.497640 | 0.493819 | 0.502160 | 0.504441 | 0.501080 | 0.501260 | 0.503301 | 5.004591e+06 | 7749.114440 | 0.495600 | 6.517040 | 0.498480 | 0.497880 |
| std | 1.116872 | 0.499972 | 0.500005 | 0.499990 | 0.500009 | 0.500008 | 0.499999 | 5.204760e+05 | 1885.213591 | 0.499991 | 3.441038 | 0.500008 | 0.500006 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 4.100135e+06 | 4500.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 |
| 25% | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 4.555418e+06 | 6116.000000 | 0.000000 | 4.000000 | 0.000000 | 0.000000 |
| 50% | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 5.002304e+06 | 7747.000000 | 0.000000 | 7.000000 | 0.000000 | 0.000000 |
| 75% | 3.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 5.458530e+06 | 9379.000000 | 1.000000 | 10.000000 | 1.000000 | 1.000000 |
| max | 3.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 5.899806e+06 | 11000.000000 | 1.000000 | 12.000000 | 1.000000 | 1.000000 |

Summary of the dataframe structure, including the column names, data types, non-null counts, and memory usage

```
df.info()
```

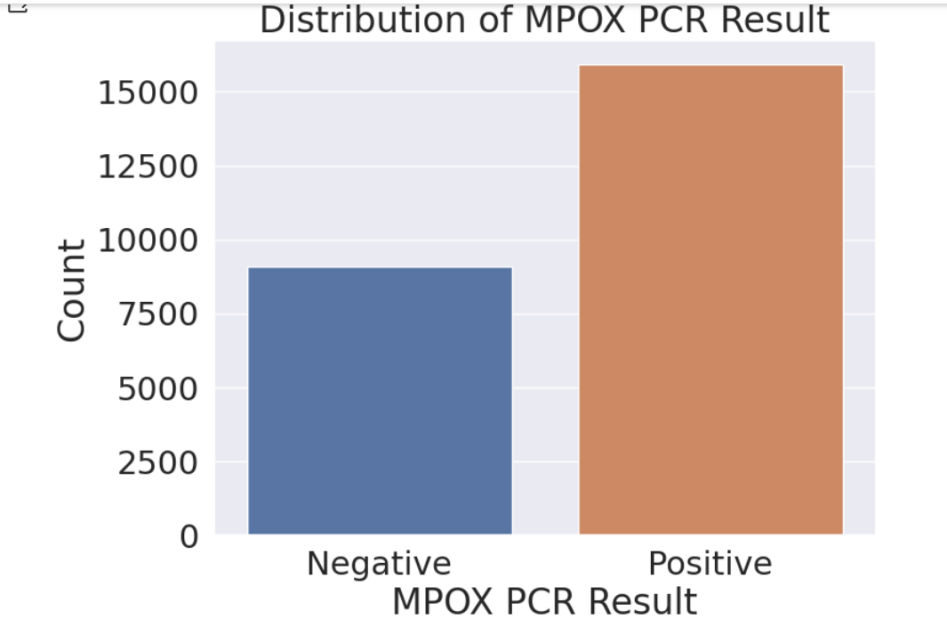
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Encoded Systemic Illness        24998 non-null  float64
 1   Rectal 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   RBC Count                       25000 non-null  int64
 9   WBC Count                       25000 non-null  int64
 10  Home ownership                  25000 non-null  int64
 11  Age                             24964 non-null  object
 12  Birth Month                     25000 non-null  int64
 13  Health Insurance                25000 non-null  int64
 14  Sexually Transmitted Infection  24996 non-null  float64
 15  MPOX PCR Result                 25000 non-null  object
dtypes: float64(6), int64(7), object(3)
memory usage: 3.1+ MB
```

Dataset dimensions: 25 000 rows 16 columns

```
df.shape
```

```
(25000, 16)
```

Plot distribution of class variable:




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

3.1. Issues found in retained dataset and the possible variables.

| Dataset or Variable Issue? | Name of variable | Issue description |
|---|---|---|
| Variable Issue | Encoded Systemic Illness | The issue with this variable is missing values |
| Variable Issue | Rectal Pain | The issue with this variable is missing values |
| Variable Issue | Penile Oedema | The issue with this variable is missing values |
| Variable Issue | Oral Lesions | The issue with this variable is missing values |
| Variable Issue | Swollen Tonsils | The issue with this variable is missing values |

| | | |
|---|---|---|
| Variable Issue | HIV Infection | The issue with this variable is missing values |
| Variable Issue | Age | The issue with this variable is missing values |
| Variable Issue | Sexually Transmitted Infection | The issue with this variable is missing values |
| Variable Issue | Oral Lesions | The issue with this variable is that it incorporates both numerical (0, 1) and categorical (Yes, No) values |
| Variable Issue | Age | The issue with this variable is that it contains textual data ("Twenty") instead of numeric data |
| Variable Issue | MPOX PCR Result | The issue with this variable is that the variable values are categorical consist of "positive" and "negative" |
| Variable Issue | Age | The issue with this variable is that the variable contains outliers (0, -23, 150, 181) |
| Whole Dataset | Whole dataset | The issue with this dataset is that data type of each column is different (float, int and object) which causes the model cannot understand the data correctly |

### 3.2. Proposed solutions and justifications for the found issues in 3.1 section

| **Dataset or Variable Issue?** | **Name of variable** | **Issue description** | **Solution** | **Justification** |
|---|---|---|---|---|
| Variable Issue | Encoded Systemic Illness | The issue is missing values | Remove rows with missing values | Missing values contains only 2 rows (0.01 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Rectal Pain | The issue is missing values | Remove rows with missing values | Missing values contains only 3 rows (0.01 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Penile Oedema | The issue is missing values | Remove rows with missing values | Missing values contains only 6 rows (0.02 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Oral Lesions | The issue is missing values | Remove rows with missing values | Missing values contains only 4 rows (0.02 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Swollen Tonsils | The issue is missing values | Remove rows with missing values | Missing values contains only 7 rows (0.03 %) of the whole dataset. Removing them almost doesn't affect the data loss |

| | | | | |
|---|---|---|---|---|
| Variable Issue | HIV Infection | The issue is missing values | Remove rows with missing values | Missing values contains only 5 rows (0.02 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Age | The issue is missing values | Remove rows with missing values | Missing values contains only 36 rows (0.14 %) of the whole dataset. Removing them almost doesn't effect the data loss |
| Variable Issue | Sexually Transmitted Infection | The issue is missing values | Remove rows with missing values | Missing values contains only 4 rows (0.02 %) of the whole dataset. Removing them almost doesn't affect the data loss |
| Variable Issue | Oral Lesions | The issue with this variable is that it incorporates both numerical (0, 1) and categorical (Yes, No) values | Mapping categorical values to numeric. Uniform the values by converting text into numeric. | Consistency is needed for variable values. Some models like Logistic regression cannot interpret text or models usually work well with numeric values |
| Variable Issue | Age | The issue with this variable is that it contains textual data ("Twenty") instead of numeric data | Data cleaning: Conversion to numeric | Age field must contain only integers. The model cannot interpret text. Values must be in the same data type. |
| Variable Issue | MPOX PCR Result | The issue with this variable is that the variable values are categorical consist of "positive" and "negative" | Label encoding | ML models generally require numeric inputs, encoding categorical labels. Label encoding is the most appropriate in case when the problem is a binary classification task, the variable is target, and no need to create extra columns |
| Variable Issue | Age | The issue with this variable is that the variable contains outliers (0, -23, 150, 181) | Remove outliers | Outliers can distort model performance. In Age feature there are 0 or even minus values which is unreal values for human age. The Outliers in Age column contains only a small portion of the whole data. They were simply excluded by finding boundaries |

| | | | | |
|---|---|---|---|---|
| Whole Dataset | Whole dataset | The issue with this dataset is that data type of each column is different (float, int and object) which causes the model cannot understand the data correctly | Data type conversion | ML models generally do not perform well when the data is in inconsistent and/or object format. Some algorithms like Naïve base or Logistic Regression work efficiently well when the data is numeric. The float can offer advantage over integers due to its capability to represent a broader range, including fractions when applying for feature engineering like Scaling, normalization or machine learning techniques for instance, Logistic regression, coefficient work well with float format. |
| Whole Dataset | Whole Dataset | The class distribution in your dataset is imbalanced, with approximately 64% of the data belonging to the positive class and 36% belonging to the negative class. | Appling suitable evaluation metrics | I applied different techniques - SMOTE, Random Over Sampling and Random Under Sampling but none of these techniques improved the model performance. Instead after applying these techniques some models performed lower score than without them. That's why solution to overcome this issue is to use different evaluation metrics (like precision, recall, F1-score). |

### 3.3. **Evidence of implementing suggested solutions**

The issue with missing values: **Before** **After** removing rows with missing values.

| | Missing values | % What is the percentage of the full column |
|---|---|---|
| Age | 36 | 0.14 |
| Swollen Tonsils | 7 | 0.03 |
| Penile Oedema | 6 | 0.02 |
| HIV Infection | 5 | 0.02 |
| Oral Lesions | 4 | 0.02 |
| Sexually Transmitted Infection | 4 | 0.02 |
| Rectal Pain | 3 | 0.01 |
| Encoded Systemic Illness | 2 | 0.01 |

```
Encoded Systemic Illness          0
Rectal Pain                       0
Sore Throat                       0
Penile Oedema                     0
Oral Lesions                      0
Solitary Lesion                   0
Swollen Tonsils                   0
HIV Infection                     0
RBC Count                         0
WBC Count                         0
Home ownership                    0
Age                               0
Birth Month                       0
Health Insurance                  0
Sexually Transmitted Infection    0
MPOX PCR Result                   0
dtype: int64
```

The issue with ‘Oral Lesions’ variable is that it incorporates both numerical (0, 1) and categorical (Yes, No) values and ‘Yes’ and ‘No’ values are converted into 1 and 0.

**Before**

```
Records with 'Yes' or 'No' in the 'Oral Lesions' column:
332    YES
412    YES
489    YES
532    YES
591     No
597     No
630     No
655     No
707     No
732     No
760     No
Name: Oral Lesions, dtype: object
```

**After**

```
Records with 'Yes' or 'No' in the 'Oral Lesions' column:
332    1
412    1
489    1
532    1
591    0
597    0
630    0
655    0
707    0
732    0
760    0
Name: Oral Lesions, dtype: object
Unique values in the 'Oral Lesions' column:
['1' '0' 1 0]
```

The issue with ‘Age’ variable is that it contains textual data ("Twenty") instead of numeric data and the text is converted into numeric data:

**Before**

```
Records with non-numeric values in the 'Age' column:
323    Twenty
Name: Age, dtype: object
```

**After**

```
Records with non-numeric values in the 'Age' column:
323    20
Name: Age, dtype: object
```

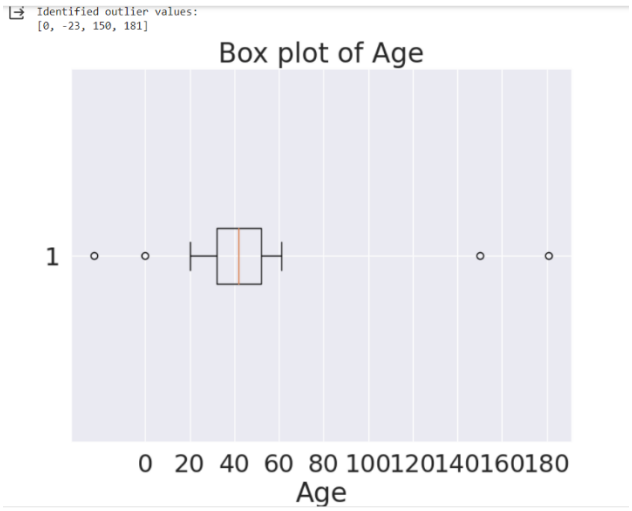
The issue with ‘Age’ variable is that the variable contains outliers (0, -23, 150, 181). The outliers are removed from the variable:

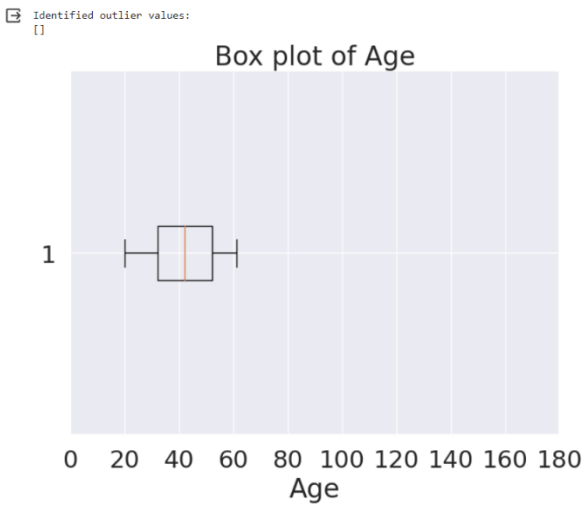
**Before**: Age column unique values:

```
['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' 20 '181'
 '-23']
```

**Before**:

**After**:




The issue with ‘MPOX PCR Result’ variable is that the variable values are categorical consist of "positive" and "negative" and they are encoded into 0 and 1 numeric data:

**Before**: MPOX PCR Result unique values:

```
['Negative' 'Positive']
```

**After**:

```
0        0
1        1
2        1
3        1
4        1
        ..
24995    1
24996    1
24997    1
24998    0
24999    1
Name: MPOX PCR Result, Length: 24934, dtype: int64
```

The issue with the whole dataset is that the data type of each column is different (float, int and object) which causes the model cannot understand the data correctly. The format of each independent variable is converted into float64, and the data has been brought into the unique format. Similarly, the target variable is converted into integer format.

**Before**:

```
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 16 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   RBC Count                       25000 non-null  int64
 9   WBC Count                       25000 non-null  int64
 10  Home ownership                  25000 non-null  int64
 11  Age                             24964 non-null  object
 12  Birth Month                     25000 non-null  int64
 13  Health Insurance                25000 non-null  int64
 14  Sexually Transmitted Infection  24996 non-null  float64
 15  MPOX PCR Result                 25000 non-null  object
dtypes: float64(6), int64(7), object(3)
memory usage: 3.1+ MB
```

**After**

```
Encoded Systemic Illness          float64
Rectal Pain                       float64
Sore Throat                       float64
Penile Oedema                     float64
Oral Lesions                      float64
Solitary Lesion                   float64
Swollen Tonsils                   float64
HIV Infection                     float64
RBC Count                         float64
WBC Count                         float64
Home ownership                    float64
Age                               float64
Birth Month                       float64
Health Insurance                  float64
Sexually Transmitted Infection    float64
MPOX PCR Result                     int64
dtype: object
```

## 4. Task (4) – Modelling: Create Predictive Classification Models

### 4.1. Machine Learning Algorithms Summary

| Algorithm Name | Type of Algorithm | Learnable Parameters | Possible Hyperparameters | Python package source code to call the algorithm |
|---|---|---|---|---|
| **Logistic Regression** | parametric | Coefficients, Intercepts | 'C': [0.01, 0.1, 1, 10, 100],<br>'penalty': ['l1', 'l2', 'none'],<br>'solver' : ['lbfgs', 'liblinear', 'newton-cg', 'sag'],<br>max_iter': [100, 200, 300],<br>'multi_class' : ['auto', 'ovr'] | from sklearn.linear_model import LogisticRegression<br>logreg = LogisticRegression() |
| **Naïve Bayes (GaussianNB)** | parametric | Prior probabilities and Likelihoods | For GaussianNB - 'var_smoothing': [1e-9, 1e-8, 1e-7] | from sklearn.naive_bayes import GaussianNB<br>nb = GaussianNB() |
| **Decision Tree** | non-parametric | Tree structure (root node, branch, and leaf nodes) | 'max_depth': [None, 5, 10, 15, 20],<br>'min_samples_split': [2, 5, 10],<br>'min_samples_leaf': [1, 2, 4]<br>'criterion': ['gini', 'entropy'] | from sklearn.tree import DecisionTreeClassifier<br>dec_tree = DecisionTreeClassifier(criterion="entropy, max_depth=5) |
| **Support Vector Machine (SVM) with** | non-parametric | Support vectors (soft margin | 'C': [0.1, 1, 10],<br>'gamma': [0.1, 0.01, 0.001],<br>'kernel': ['rbf'] | from sklearn.svm import SVC<br>svclassifier = SVC(kernel='rbf', gamma="auto", probability=True, class_weight='balanced') |

| | | | | | |
|---|---|---|---|---|---|
| **RBF kernel** | | classifier, soft margin) | | | |
| **KNN** | non-parametric | - | 'n_neighbors': [3, 5, 7, 9], | from sklearn.neighbors import KNeighborsClassifier<br>knn = KNeighborsClassifier(n_neighbors = 29) | |
| | | | 'weights': ['uniform', 'distance'], | | |
| | | | 'p': [1, 2] | | |

4.2. Training-test split ration and its justification

The list of all feature names used for building the classification models and the corresponding data shape function output.

```
List of Feature Names:
Encoded Systemic Illness
Rectal Pain
Sore Throat
Penile Oedema
Oral Lesions
Solitary Lesion
Swollen Tonsils
HIV Infection
RBC Count
WBC Count
Home ownership
Age
Birth Month
Health Insurance
Sexually Transmitted Infection

Data Shape:
(24930, 15)
```

**Training-test split ratio**. For my predictive classification models, I opted for a consistent 70 % training data and 30 % testing data split. The idea behind my choice is to aim to allocate enough data for training while still retaining a substantial percentage for the assessing the model performance.

While the split ratios – 80:20, 70:30, 67:33 or 75:25 are the most common practice in machine learning, there isn't a definitive best size. Scholars like Vrigazova, (2021) suggests that experimenting with different test set structures, such as 50/50, 60/40, 70/30, 80/20, can optimize model performance. Through various experiments, a 70/30 split was recommended as it consistently produced reliable results. In my experimentation, I found the 70/30 split to be the most suitable for this dataset, as it yielded optimal model performance across multiple algorithms.

*Note: I used scaled data (X_scaled) for Logistic Regression and Naïve Bayes algorithms based on scaled techniques due to their sensitivity to variations in magnitude, units, and range.*

```
[64] X_train, X_test, y_train, y_test = train_test_split(X_scaled,
                                                         y,
                                                         test_size=0.3,
                                                         stratify=y,
                                                         random_state=42,
                                                         shuffle=True)
```

To ensure that all models are tested on the same test dataset, the following specific parameters of ‘train_test_split’ function from the sklearn.model_selection module were added:

```
Training Set Class Distribution:
1    0.636411
0    0.363589
Name: MPOX PCR Result, dtype: float64

Test Set Class Distribution:
1    0.636315
0    0.363685
Name: MPOX PCR Result, dtype: float64
X_train shape: (17451, 15)
X_test shape: (7479, 15)
y_train shape: (17451,)
y_test shape: (7479,)
```

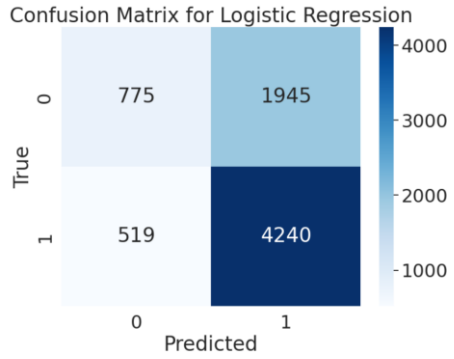
The **random_state** parameter with value 42 ensures that the data split is reproducible.

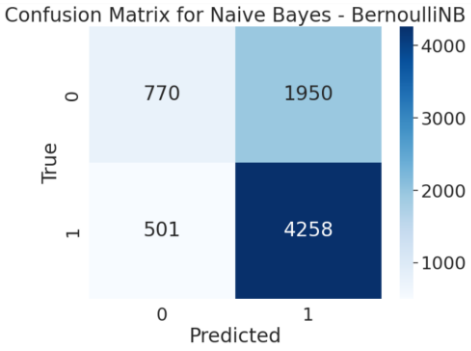
The **shuffle=True** means shuffling the data before splitting, preventing any underlying patters in the dataset from affecting the split.

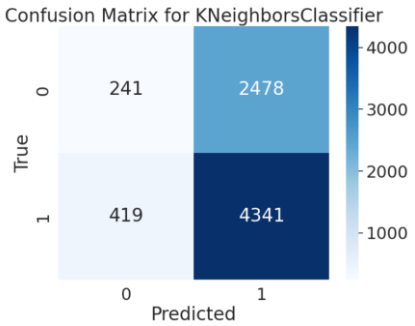
The **stratify=y** helps that the class label ratio (MPOX Negatives : MPOX Positives) are preserved in both the training and test sets. This means that the ratio of different classes in the target variable 'y' is maintained across the splits, which is crucial to avoid an imbalance that might bias the model's learning.

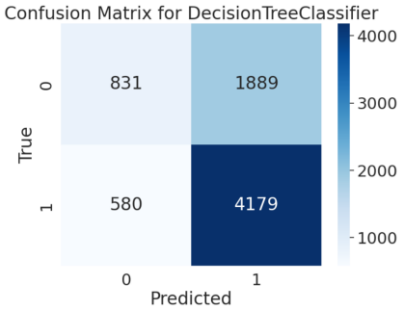
## 5. Task (5) – Evaluation: How good are your models.

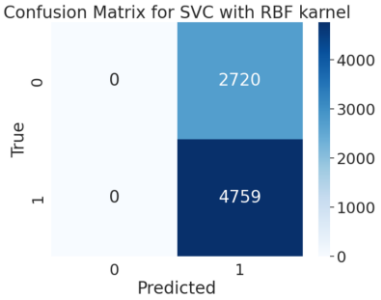
### 5.1. Screenshots of confusion matrix for each trained model




### 5.2. Evaluation metrics and their interpretation

| Metric Name | “USE” or “DO NOT USE” | Justification in relation to the success criteria | Model Name | Metric Score |
|---|---|---|---|---|
| **Accuracy** | DO NOT USE | As the data is imbalanced (positive labels are almost double higher than negative labels) this matric is not suitable. Accuracy can be misleading in imbalanced datasets. | LR | 0.67 |
| | | | DT | 0.67 |
| | | | KNN | 0.61 |

| | | | | |
|---|---|---|---|---|
| | | Additionally, the accuracy can be used when both positive and negative labels are important but according to the success criteria, maximizing positive cases is important, which means this metric does not align with success criteria | SVM (RBF) | 0.63 |
| | | | NB | 0.67 |
| **Recall** | USED | Important metric as it focuses on the model's ability to identify positive cases (MPOX positive subjects), aligning with the success criteria. | LR | 0.89 |
| | | | DT | 0.88 |
| | | | KNN | 0.91 |
| | | | SVM (RBF) | 0.69 |
| | | | NB | 0.89 |
| **Precision** | USED | This metric is important metrics as it evaluates the accuracy of positive predictions. It ensures that a high percentage of positive predictions are accurate, crucial for recommending actions like PCR tests. | LR | 0.69 |
| | | | DT | 0.69 |
| | | | KNN | 0.64 |
| | | | SVM (RBF) | 0.72 |
| | | | NB | 0.69 |
| **F-Measure** | USED | F-measure combines precision and recall, providing an overall measure of the model's performance on positive predictions. | LR | 0.77 |
| | | | DT | 0.77 |
| | | | KNN | 0.74 |
| | | | SVM (RBF) | 0.70 |
| | | | NB | 0.78 |
| **AUC-ROC** | DO NOT USE | AUC-ROC is useful when the data is balanced well and positive and negative classes are equally cared. According to our business requirement, it is more important to maximise positives classified labels, even it means classifying some negatives as positive | LR | 0.66 |
| | | | DT | 0.67 |
| | | | KNN | 0.49 |
| | | | SVM (RBF) | 0.65 |
| | | | NB | 0.67 |

### 5.3. The best classification model or models suggestion

Naïve Bayes model meets the healthcare professional success criteria by:

**High Recall**: Considering the success criteria provided by the healthcare professionals, which is about maximizing MPOX positive subjects, meeting the goal of recommending PCR test and self-isolation for as many true positive cases as possible. This model along with LR demonstrates a slightly higher score in recall. However, in overall score, LR's f-measure score is slightly lower than NB and this was one reason why NB model has been chosen as the best model.

**Reasonable Precision**: when prediciting both positive and negative cases truly, NB and SVM demonstrate slightly better scores than the rest of algorithms. In other words, the both models are performing well at predicating positive cases, which are a significant portion belongs to the MPOX positive class and minimizing unnecessary PCR tests and self-isolation recommendations for false positives. However, SVM model does not have a balance in other metrics – in recall and f-measure. In terms of Recall the SVM is the worst model, which means that this model is not a cost-effective screening tool, aligning with the healthcare professionals' need for an inexpensive screening method. For that reason SVM is excluded from the best models.

**Balanced F-Measure**: F-measure is the hamonic mean of precision and recall, which shows balance of between the model's identifying positive cases. Considering this, the NB is demonstrating a reasonable balance with 0.78 score.

This provide a balance between identifying true positive cases (high recall) and maintaining correctness within the predicted positive cases (reasonable precision), aligning well with the healthcare professionals' objectives.

### 5.4. Models Hyper-Parameters Tuning

**Recall**: There was a notable rise in recall scores, from 0.89 to 0.93 after tuning. This means that the model is correctly improved towards the maximizing positive classes as the success criteria. This is even more important for the healthcare professionals' requirement of minimizing false negatives, ensuring that more individuals who may have contracted the virus are correctly identified.

**Precision**: There was a slight decrease in precision, from 0.69 to 0.67 after tuning. This means that false positive predictions among the predicted positive cases might be increased slightly, but as the dataset is imbalanced and the main aim is to increase recall even it might happen to decreasing of precision, I think the trade-off between precision and recall is acceptable.

**F-Measure**: There was not a change in this metric after tuning (0.78).

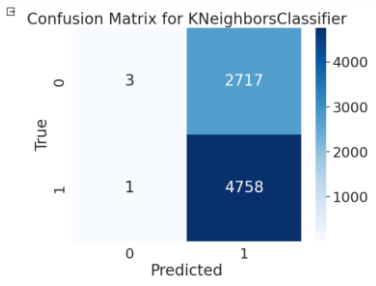
Based on GridSearch approach the best hyperparameters for the NB model has been identified as {'var_smoothing': 1e-09}

```
Best Hyperparameters: {'var_smoothing': 1e-09}
Estimated Best Hyperparameters: {'var_smoothing': 1e-09}
Test Confusion Matrix:
 [[ 537 2183]
 [ 326 4433]]
Accuracy: 0.6645273432277042
Precision: 0.6700423216444982
Recall: 0.9314982139104854
F1-score: 0.7794285714285715
AUC-ROC: 0.6540917441256814
```

| | A | B | D | E | F |
|---|---|---|---|---|---|
| 1 | Metric Name | "USE" or "DO NOT USE" | Model Name | Metric Score | Metric Score After Tunir |
| 6 | Accuracy | DO NOT | NB | 0.67 | 0.66 |
| 11 | Recall | USED | NB | 0.89 | 0.93 |
| 16 | Precision | USED | NB | 0.69 | 0.67 |
| 21 | F-Measure | USED | NB | 0.78 | 0.78 |
| 26 | AUC-ROC | DO NOT | NB | 0.67 | 0.65 |
| 27 | | | | | |

### 5.5. Ensemble Learning Using KNN and Naive Bayes

```
Test Confusion Matrix:
 [[   3 2717]
 [   1 4758]]
Accuracy: 0.6365824308062575
Precision: 0.6365217391304347
Recall: 0.9997898718218113
F1-score: 0.7778322707209415
```




We combined KNN for its precision and NB for its balanced performance to create an ensemble. While KNN delivered high recall and NB showed balanced recall and precision, the ensemble remarkably improved recall to 0.9998 by leveraging KNN's strengths, reducing missed positive cases. Despite a slight decrease in precision, the

ensemble significantly minimized the risk of failing to detect MPOX positive cases, crucial for effective screening.

5.6. Criticism of the best-performing model, and limitations and ethical issues

Answer to Research Question:

Our ensemble model, combining KNN and Naive Bayes, demonstrates promise in predicting potential MPOX cases without lab tests. By integrating KNN's precision and NB's balanced performance, we improved recall significantly to identify more MPOX positive cases. However, the model's precision slightly decreased, a trade-off for enhancing recall to minimize false negatives.

Criticism and Limitations:

Critically, the model's performance might be limited by its reliance on features in the dataset, potentially overlooking crucial factors not included. Ethically, using this model for screening raises concerns about false positives leading to unnecessary stress and PCR tests for individuals incorrectly identified as MPOX positive.

Algorithm Selection:

The ensemble overtook other models by leveraging KNN's precision and NB's balanced performance, ensuring higher recall, critical for effective screening. This ensemble approacxccxv , ,

h harnesses the strengths of both algorithms, compensating for each other's weaknesses to maximize overall predictive capability./.,

Reference:

Vrigazova, B. (2021), “The Proporti nmnmxvcon for Splitting Data into Training and Test Set for the Bootstrap in Classification Problems”, Business Systems Research, 12(1):228-242. DOI: https://doi.org/10.2478/bsrj-2021-0015