In [ ]:
```
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Reading the file and checking it
df = pd.read_csv('C:\\Users\\jeeva\\Desktop\\Git Practise\\AIDS-Virus-Infection-Prediction-\\AIDS_Classifica
df.head(5)
```

Out[ ]:

| | time | trt | age | wtkg | hemo | homo | drugs | karnof | oprior | z30 | ... | str2 | strat | symptom | treat | offtrt | cd40 | cd420 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1073 | 1 | 37 | 79.46339 | 0 | 1 | 0 | 100 | 0 | 1 | ... | 1 | 2 | 0 | 1 | 0 | 322 | 469 |
| **1** | 324 | 0 | 33 | 73.02314 | 0 | 1 | 0 | 90 | 0 | 1 | ... | 1 | 3 | 1 | 1 | 1 | 168 | 575 |
| **2** | 495 | 1 | 43 | 69.47793 | 0 | 1 | 0 | 100 | 0 | 1 | ... | 1 | 1 | 0 | 0 | 0 | 377 | 333 |
| **3** | 1201 | 3 | 42 | 89.15934 | 0 | 1 | 0 | 100 | 1 | 1 | ... | 1 | 3 | 0 | 0 | 0 | 238 | 324 |
| **4** | 934 | 0 | 37 | 137.46581 | 0 | 1 | 0 | 100 | 0 | 0 | ... | 0 | 3 | 0 | 0 | 1 | 500 | 443 |

5 rows × 23 columns

# Exploratory Data analysis - EDA

## Explore, study and visualize data

In [ ]:
```
# Bottom five data values
df.tail()
```

Out[ ]:

| | time | trt | age | wtkg | hemo | homo | drugs | karnof | oprior | z30 | ... | str2 | strat | symptom | treat | offtrt | cd40 | cd42 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **49995** | 953 | 3 | 46 | 61.28204 | 0 | 0 | 0 | 90 | 0 | 1 | ... | 1 | 3 | 0 | 1 | 1 | 234 | 40 |
| **49996** | 1036 | 0 | 42 | 73.36768 | 0 | 1 | 0 | 100 | 0 | 1 | ... | 1 | 3 | 0 | 0 | 1 | 369 | 57 |
| **49997** | 1157 | 0 | 40 | 78.75824 | 0 | 1 | 0 | 100 | 0 | 1 | ... | 1 | 1 | 0 | 1 | 0 | 308 | 66 |
| **49998** | 596 | 0 | 31 | 52.20371 | 0 | 0 | 0 | 100 | 0 | 1 | ... | 1 | 1 | 0 | 1 | 1 | 349 | 44 |
| **49999** | 612 | 2 | 41 | 77.12100 | 0 | 1 | 0 | 90 | 0 | 1 | ... | 1 | 3 | 0 | 1 | 0 | 428 | 39 |

5 rows × 23 columns

In [ ]:
```
df.shape
```

Out[ ]:
```
(50000, 23)
```

The shape of the dataset is 50,000 rows and 23 columns of variables.

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 23 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   time      50000 non-null  int64
 1   trt       50000 non-null  int64
 2   age       50000 non-null  int64
 3   wtkg      50000 non-null  float64
 4   hemo      50000 non-null  int64
 5   homo      50000 non-null  int64
 6   drugs     50000 non-null  int64
 7   karnof    50000 non-null  int64
 8   oprior    50000 non-null  int64
 9   z30       50000 non-null  int64
 10  preanti   50000 non-null  int64
 11  race      50000 non-null  int64
 12  gender    50000 non-null  int64
 13  str2      50000 non-null  int64
 14  strat     50000 non-null  int64
 15  symptom   50000 non-null  int64
 16  treat     50000 non-null  int64
 17  offtrt    50000 non-null  int64
 18  cd40      50000 non-null  int64
 19  cd420     50000 non-null  int64
 20  cd80      50000 non-null  int64
 21  cd820     50000 non-null  int64
 22  infected  50000 non-null  int64
dtypes: float64(1), int64(22)
memory usage: 8.8 MB
```

We can find the dataset has no null values. we can understand the variable datatypes through this. No Object type variable is found.

In [ ]:
```
df.columns.to_list()
```

Out[ ]:
```
['time',
 'trt',
 'age',
 'wtkg',
 'hemo',
 'homo',
 'drugs',
 'karnof',
 'oprior',
 'z30',
 'preanti',
 'race',
 'gender',
 'str2',
 'strat',
 'symptom',
 'treat',
 'offtrt',
 'cd40',
 'cd420',
 'cd80',
 'cd820',
 'infected']
```

In [ ]:
```
df.describe().T
```

Out[ ]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **time** | 50000.0 | 877.369780 | 307.288688 | 66.00000 | 542.000000 | 1045.000000 | 1136.000000 | 1231.00000 |
| **trt** | 50000.0 | 1.384800 | 1.233272 | 0.00000 | 0.000000 | 1.000000 | 3.000000 | 3.00000 |
| **age** | 50000.0 | 34.164020 | 7.091152 | 12.00000 | 29.000000 | 34.000000 | 39.000000 | 68.00000 |
| **wtkg** | 50000.0 | 75.861991 | 12.028730 | 42.36162 | 68.253682 | 74.054115 | 81.142185 | 149.83087 |
| **hemo** | 50000.0 | 0.033480 | 0.179888 | 0.00000 | 0.000000 | 0.000000 | 0.000000 | 1.00000 |
| **homo** | 50000.0 | 0.653540 | 0.475847 | 0.00000 | 0.000000 | 1.000000 | 1.000000 | 1.00000 |
| **drugs** | 50000.0 | 0.132220 | 0.338733 | 0.00000 | 0.000000 | 0.000000 | 0.000000 | 1.00000 |
| **karnof** | 50000.0 | 96.831560 | 5.091788 | 76.00000 | 90.000000 | 100.000000 | 100.000000 | 100.00000 |
| **oprior** | 50000.0 | 0.042300 | 0.201275 | 0.00000 | 0.000000 | 0.000000 | 0.000000 | 1.00000 |
| **z30** | 50000.0 | 0.640880 | 0.479747 | 0.00000 | 0.000000 | 1.000000 | 1.000000 | 1.00000 |
| **preanti** | 50000.0 | 318.159560 | 402.932765 | 0.00000 | 0.000000 | 123.000000 | 503.000000 | 2828.00000 |
| **race** | 50000.0 | 0.293300 | 0.455279 | 0.00000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |
| **gender** | 50000.0 | 0.856700 | 0.350382 | 0.00000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |
| **str2** | 50000.0 | 0.575200 | 0.494318 | 0.00000 | 0.000000 | 1.000000 | 1.000000 | 1.00000 |
| **strat** | 50000.0 | 1.936420 | 0.895318 | 1.00000 | 1.000000 | 2.000000 | 3.000000 | 3.00000 |
| **symptom** | 50000.0 | 0.083460 | 0.276579 | 0.00000 | 0.000000 | 0.000000 | 0.000000 | 1.00000 |
| **treat** | 50000.0 | 0.734160 | 0.441784 | 0.00000 | 0.000000 | 1.000000 | 1.000000 | 1.00000 |
| **offtrt** | 50000.0 | 0.342220 | 0.474458 | 0.00000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |
| **cd40** | 50000.0 | 319.079540 | 102.525976 | 0.00000 | 236.000000 | 299.000000 | 396.000000 | 930.00000 |
| **cd420** | 50000.0 | 438.090100 | 144.806831 | 81.00000 | 327.000000 | 415.000000 | 531.000000 | 1119.00000 |
| **cd80** | 50000.0 | 1045.936440 | 488.617434 | 96.00000 | 713.000000 | 885.000000 | 1245.000000 | 4656.00000 |
| **cd820** | 50000.0 | 905.938440 | 339.707976 | 173.00000 | 649.000000 | 858.000000 | 1084.000000 | 3538.00000 |
| **infected** | 50000.0 | 0.310120 | 0.462547 | 0.00000 | 0.000000 | 0.000000 | 1.000000 | 1.00000 |

In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
df.corr().style.background_gradient(cmap='inferno')
```

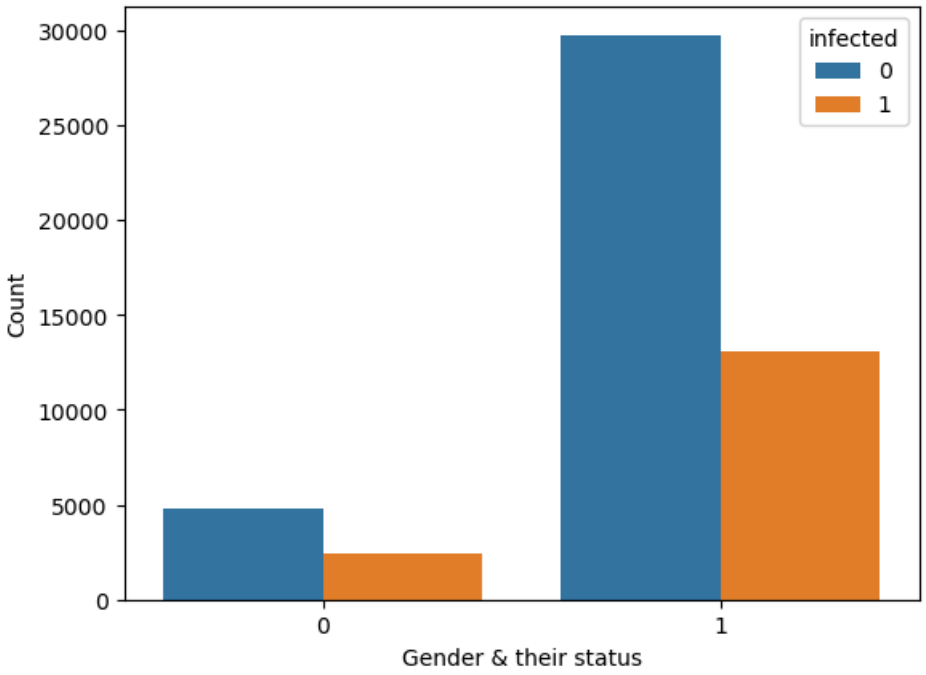
Out[ ]:

| | time | trt | age | wtkg | hemo | homo | drugs | karnof | oprior | z30 | preanti |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **time** | 1.000000 | 0.051746 | 0.006911 | 0.039390 | -0.019287 | 0.056241 | -0.012310 | 0.021562 | -0.021745 | -0.074956 | -0.056202 |
| **trt** | 0.051746 | 1.000000 | 0.040517 | 0.010247 | 0.002240 | 0.064000 | -0.005980 | -0.038132 | -0.004581 | -0.001674 | 0.001603 |
| **age** | 0.006911 | 0.040517 | 1.000000 | 0.001856 | -0.017836 | 0.108080 | 0.002495 | -0.046430 | 0.011142 | 0.118811 | 0.093419 |
| **wtkg** | 0.039390 | 0.010247 | 0.001856 | 1.000000 | -0.034036 | 0.089131 | -0.016851 | 0.008592 | -0.004932 | -0.106208 | -0.066063 |
| **hemo** | -0.019287 | 0.002240 | -0.017836 | -0.034036 | 1.000000 | -0.068466 | 0.001203 | -0.002599 | 0.013915 | 0.064002 | 0.043161 |
| **homo** | 0.056241 | 0.064000 | 0.108080 | 0.089131 | -0.068466 | 1.000000 | -0.051067 | -0.011770 | 0.008095 | -0.013145 | 0.014152 |
| **drugs** | -0.012310 | -0.005980 | 0.002495 | -0.016851 | 0.001203 | -0.051067 | 1.000000 | 0.010188 | -0.010457 | -0.033090 | -0.034399 |
| **karnof** | 0.021562 | -0.038132 | -0.046430 | 0.008592 | -0.002599 | -0.011770 | 0.010188 | 1.000000 | 0.000356 | -0.131726 | -0.090110 |
| **oprior** | -0.021745 | -0.004581 | 0.011142 | -0.004932 | 0.013915 | 0.008095 | -0.010457 | 0.000356 | 1.000000 | 0.058107 | 0.046862 |
| **z30** | -0.074956 | -0.001674 | 0.118811 | -0.106208 | 0.064002 | -0.013145 | -0.033090 | -0.131726 | 0.058107 | 1.000000 | 0.425415 |
| **preanti** | -0.056202 | 0.001603 | 0.093419 | -0.066063 | 0.043161 | 0.014152 | -0.034399 | -0.090110 | 0.046862 | 0.425415 | 1.000000 |
| **race** | -0.041361 | -0.067184 | -0.075247 | -0.045631 | 0.008795 | -0.178007 | 0.047854 | 0.034598 | -0.018406 | -0.092531 | -0.085702 |
| **gender** | 0.053726 | 0.039346 | 0.034213 | 0.067468 | -0.015268 | 0.192607 | -0.025554 | -0.002476 | 0.002859 | -0.024403 | -0.002565 |
| **str2** | -0.078147 | 0.000497 | 0.121492 | -0.104913 | 0.062104 | -0.010187 | -0.040928 | -0.136562 | 0.065423 | 0.605567 | 0.454841 |
| **strat** | -0.068901 | 0.003392 | 0.109482 | -0.092254 | 0.063511 | 0.010497 | -0.045483 | -0.130202 | 0.070862 | 0.573218 | 0.469166 |
| **symptom** | -0.019133 | -0.018105 | 0.001566 | 0.004827 | -0.002698 | 0.022002 | 0.000906 | 0.008108 | 0.017419 | 0.022249 | 0.021176 |
| **treat** | 0.068051 | 0.232762 | 0.041837 | 0.001226 | 0.008057 | 0.072673 | -0.009026 | -0.053328 | -0.011640 | 0.010907 | 0.023678 |
| **offtrt** | -0.101019 | -0.027697 | -0.049142 | -0.029979 | 0.011043 | -0.103003 | 0.037158 | -0.025224 | -0.008963 | -0.007565 | -0.011919 |
| **cd40** | 0.040720 | 0.013352 | -0.053918 | 0.047813 | -0.034345 | 0.000200 | 0.025400 | 0.045776 | -0.041128 | -0.217151 | -0.183870 |
| **cd420** | 0.089658 | 0.028357 | -0.065441 | 0.066893 | -0.034451 | -0.010264 | 0.027702 | 0.062692 | -0.042355 | -0.284487 | -0.234264 |
| **cd80** | 0.018400 | -0.004262 | 0.005863 | 0.019929 | -0.004844 | 0.025367 | -0.005788 | 0.023325 | 0.006786 | -0.018809 | -0.020495 |
| **cd820** | 0.026256 | 0.019198 | -0.001756 | -0.006143 | -0.004207 | -0.004070 | 0.006794 | 0.013964 | -0.010503 | 0.001278 | -0.000010 |
| **infected** | -0.102671 | -0.047112 | 0.028718 | -0.055527 | 0.026407 | -0.007341 | -0.025684 | -0.025860 | 0.043416 | 0.238531 | 0.172826 |

# Data visualization

In [ ]:

```
sns.countplot(df,x='gender',hue='infected')
plt.xlabel("Gender & their status")
plt.ylabel("Count")
plt.show()
```

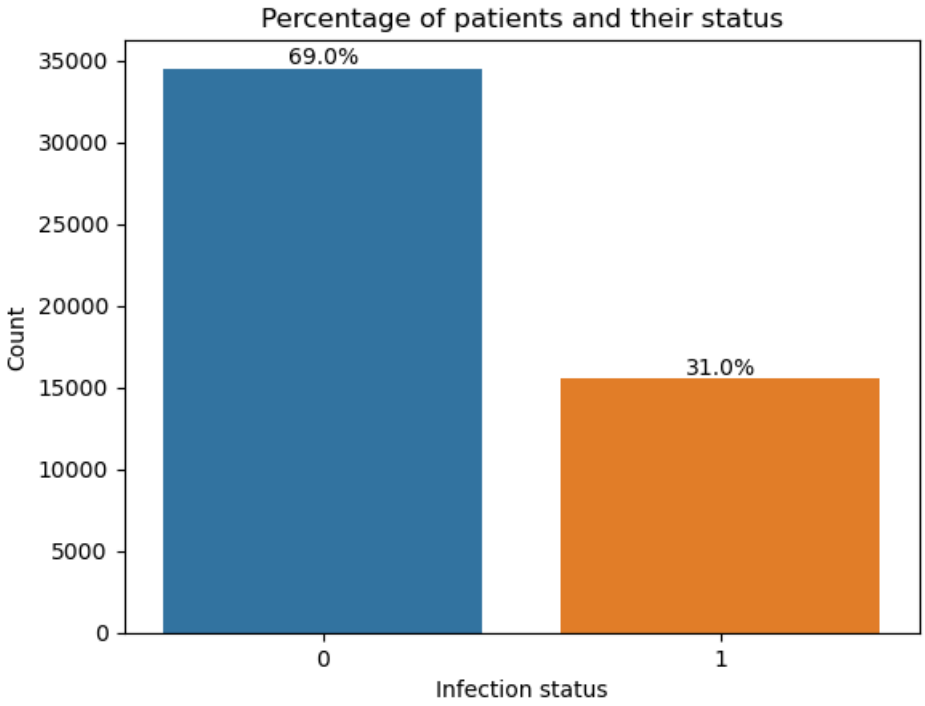


In [ ]:
```
# Calculate total number of people by gender
total_by_gender = df['gender'].value_counts()

# Calculate number of infected people by gender
infected_by_gender = df[df['infected'] == True]['gender'].value_counts()

# Calculate percentage of infected people by gender
percentage_infected_by_gender = (infected_by_gender / total_by_gender) * 100

print(percentage_infected_by_gender)
```

```
gender
1    30.594140
0    33.510119
Name: count, dtype: float64
```

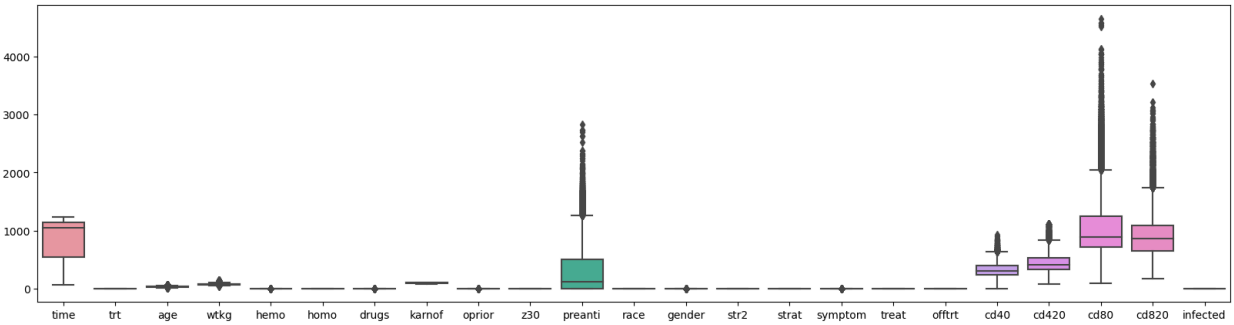
Gender based infection distribution is 30% - Male & 33% - Female

In [ ]:
```
ax =sns.countplot(df,x='infected')
total = df['infected'].count()
ax.bar_label(ax.containers[0], fmt=lambda x: f'{(x/total)*100:.1f}%')
plt.title('Percentage of patients and their status')
plt.xlabel("Infection status")
plt.ylabel("Count")
plt.show()
```




We could see that dataset is imbalanced. Infected column consists of categorical of 0 & 1. '0' is 69% registered and '1' is 31% registered.

In [ ]:
```
# Box plot
plt.figure(figsize=(20,5))
sns.boxplot(df)
plt.show()
```




# Model preparation

In [ ]:
```
# Before sampling
X = df.drop(['infected'], axis='columns')
y = df['infected']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=5)
```

In [ ]:
```
# Importing standard scler and Scaling the data

ss = StandardScaler()
X_train.iloc[:,:] = ss.fit_transform(X_train.iloc[:,:])
X_test.iloc[:,:] = ss.transform(X_test.iloc[:,:])
```

In [ ]:
```
lr = LogisticRegression()
log_reg = lr.fit(X_train,y_train)
```

In [ ]:
```
y_train_pred = log_reg.predict(X_train)
```

In [ ]:
```
log_reg.coef_
```

Out[ ]:
```
array([[-0.1586788 , -0.07051645, -0.02920935, -0.04288642,  0.00148883,
         0.01227915, -0.04402175,  0.02522251,  0.04151225,  0.23938347,
         0.03238654,  0.01737807, -0.02549107,  0.16427716,  0.22879375,
         0.03799094, -0.10034448, -0.08008363, -0.15012168, -0.2134787 ,
        -0.00447248, -0.02991158]])
```

In [ ]:
```
log_reg.intercept_
```

Out[ ]:
```
array([-0.9098153])
```

In [ ]:
```
# Model score

model_score = lr.score(X_train,y_train)
model_score
```

Out[ ]:
```
0.7071714285714286
```

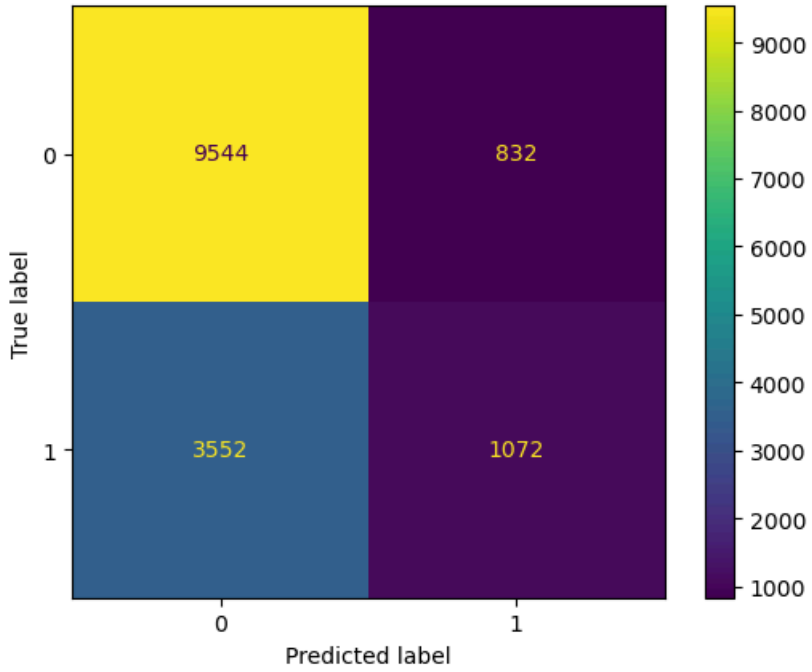
In [ ]:
```
# Y train probability
y_train_prob = lr.predict_proba(X_train)

# Y train predict
y_train_pred = lr.predict(X_train)

# Y test probability
y_test_prob = lr.predict_proba(X_test)

# Y test predict
y_test_pred = lr.predict(X_test)
```

# Confusion Matrix

In [ ]:
```
# Accuracy score for test data
metrics.accuracy_score(y_test,y_test_pred)
```

Out[ ]:
```
0.7077333333333333
```

In [ ]:
```
cm = confusion_matrix(y_test,y_test_pred)

dip = ConfusionMatrixDisplay(confusion_matrix=cm)
dip.plot()
plt.show()
```




- Due to the imbalance Dataset the target class has low True positive rate
- Further we can evaluate by Oversampling the data to Balance the dataset

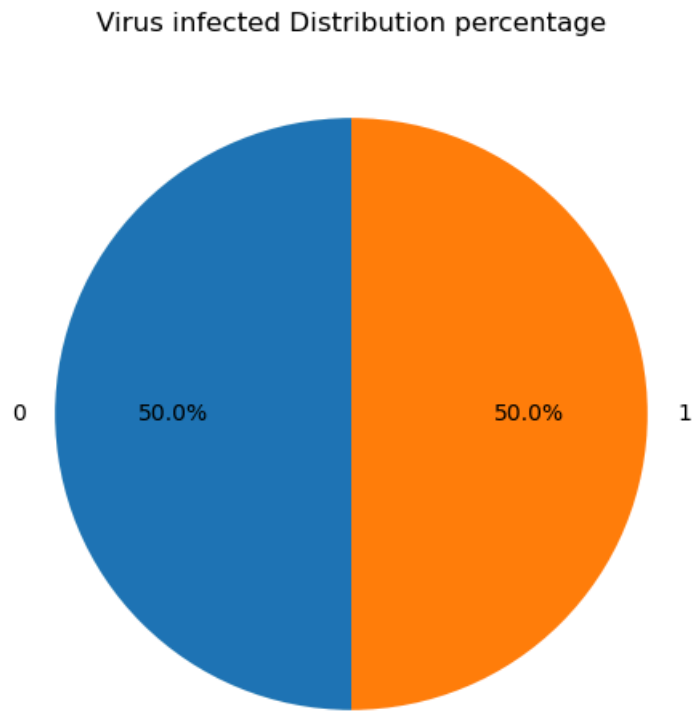
In [ ]:
```
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,classification_report, ConfusionMatrixDisplay
```

In [ ]:
```
# Now Oversampling data to match value counts of two classification

# separate majority and minority classes
majority = df[df.infected == 0]
minority = df[df.infected == 1]

# oversample the minority class
minority_oversampled = resample(minority,
                                replace=True,
                                n_samples=len(majority),
                                random_state=42)

# combine majority class with oversampled minority class
oversampled_data = pd.concat([majority, minority_oversampled])

# check the distribution of undersampled and oversampled datasets
oversampled_distribution = oversampled_data.infected.value_counts()

oversampled_distribution
```

Out[ ]:
```
infected
0    34494
1    34494
Name: count, dtype: int64
```

- By Oversampling we matched the Two classes with same level of data values

In [ ]:
```
# Count the occurrences of each category
category_counts = oversampled_data['infected'].value_counts()

# Extract labels and sizes for the pie plot
labels = category_counts.index.tolist()
sizes = category_counts.values.tolist()

# Plotting the pie chart
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Virus infected Distribution percentage')
plt.show()
```



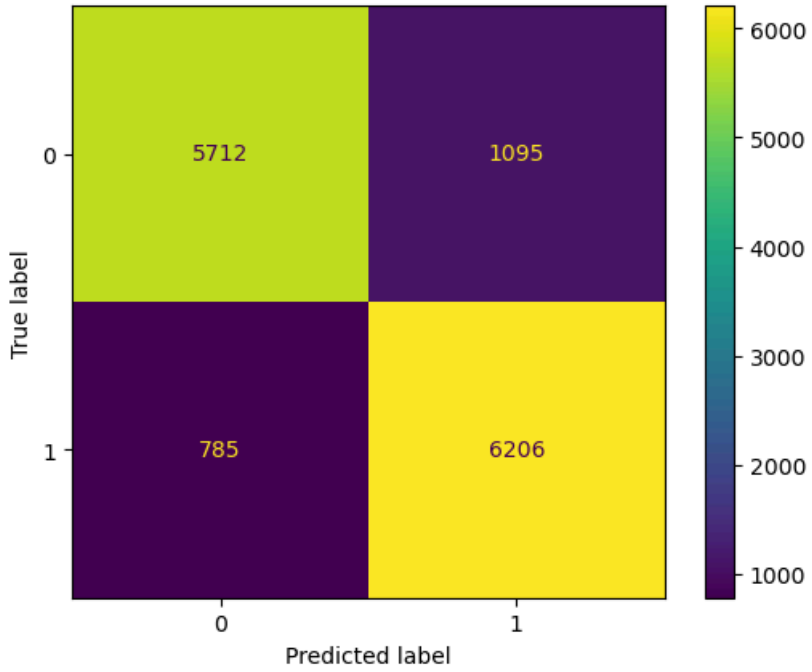

- Visual representation of two classes value counts

In [ ]:
```
# prepare the oversampled data
X_oversampled = oversampled_data.drop('infected', axis=1)
y_oversampled = oversampled_data[['infected']]


# splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_oversampled, y_oversampled, test_size=0.2, random_state=42)

# create and train the Random Forest model
rf_model_oversampled = RandomForestClassifier(random_state=42)
rf_model_oversampled.fit(X_train, y_train)

# predictions
y_pred = rf_model_oversampled.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      6807
           1       0.85      0.89      0.87      6991

    accuracy                           0.86     13798
   macro avg       0.86      0.86      0.86     13798
weighted avg       0.86      0.86      0.86     13798
```

- By oversampling the dataset we have achieved 16% of accuracy increase
- We used Random forest classifier

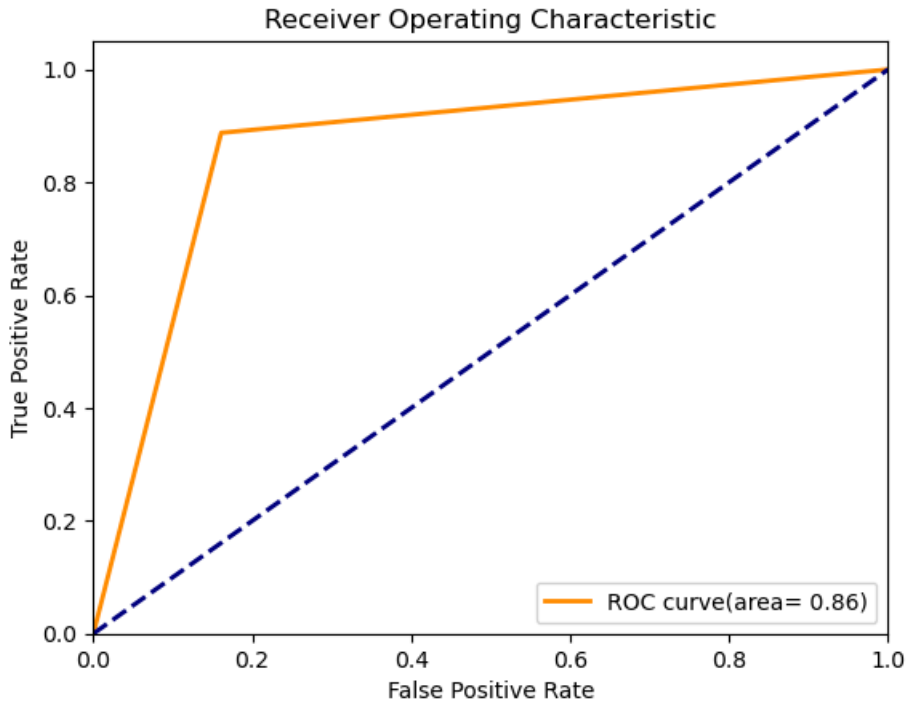
In [ ]:
```
cm = confusion_matrix(y_test,y_pred)

dip = ConfusionMatrixDisplay(confusion_matrix=cm)
dip.plot()
plt.show()
```




- This confusion matrix shows very good amount of True positives compared to True negative

In [ ]:
```
# Calculate AUC-ROC score:
from sklearn.metrics import confusion_matrix,roc_auc_score,roc_curve

auc_roc = roc_auc_score(y_test,y_pred)
print(f'AUC-ROC Score:{auc_roc}')

# step 6 : plot ROC curve
fpr,tpr,_ = roc_curve(y_test,y_pred)
# Calculates the FPR and TPR
# at various threshold settings for the given y_test(actual)
# and y_predict(predicted probabilities or scores).

plt.figure()
# creates a new figure for the plot

plt.plot(fpr,tpr,color='darkorange',lw=2,label='ROC curve(area= %0.2f)'%auc_roc)
# plots the roc curve using  the FPR (x-axis) and TPR(y-axis) values
# color
#lw = line width = 2
# label = auc-roc adds a label to the curve
# showing the AUC -ROC score.

plt.plot([0,1],[0,1],color='navy',lw=2,linestyle='--')
# plot a diagonal line from(0,0), to (1,1)representing
# the roc curve of a purely random classifier.


plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])

#sets the limits for the x-axis and y-axis representively.
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

```
AUC-ROC Score:0.8634244784533461
```




# Using Cross validation method to evaluate the parameters we could change to obtain best fit

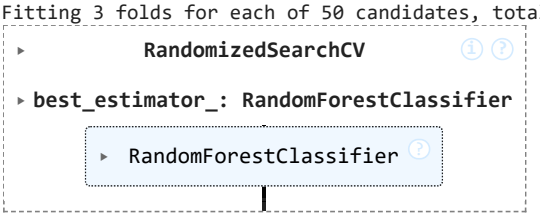
In [ ]:
```
n_estimators = [5,20,50,100] # number of trees in the random forest
max_features = ['auto', 'sqrt'] # number of features in consideration at every split
max_depth = [int(x) for x in np.linspace(10, 100, num = 10)] # maximum number of levels allowed in each deci
min_samples_split = [2, 6, 10] # minimum sample number to split a node
min_samples_leaf = [1, 3, 4] # minimum sample number that can be stored in a leaf node
bootstrap = [True, False] # method used to sample data points

random_grid = {'n_estimators': n_estimators,

'max_features': max_features,

'max_depth': max_depth,

'min_samples_split': min_samples_split,

'min_samples_leaf': min_samples_leaf,

'bootstrap': bootstrap}
```

In [ ]:
```
# Importing randomized CV
from sklearn.model_selection import RandomizedSearchCV

rf = RandomForestClassifier(random_state=2)
rf_random = RandomizedSearchCV(estimator = rf,param_distributions = random_grid,
               n_iter = 50, cv = 3, verbose=2, random_state=35, n_jobs = -1)
```

In [ ]:
```
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

Out[ ]:




In [ ]:
```
print ('Random grid: ', random_grid, '\n')
# print the best parameters
print ('Best Parameters: ', rf_random.best_params_, ' \n')
```

```
Random grid:  {'n_estimators': [5, 20, 50, 100], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], 'min_samples_split': [2, 6, 10], 'min_samples_leaf': [1, 3, 4], 'bootstrap': [True, False]}

Best Parameters:  {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 50, 'bootstrap': False}
```

# The best parameter is obtained now we need to impute this in RFclassifier to fit the model

In [ ]:
```
randmf = RandomForestClassifier(n_estimators = 100, min_samples_split = 2, min_samples_leaf= 1, max_features
randmf.fit( X_train, y_train)
```

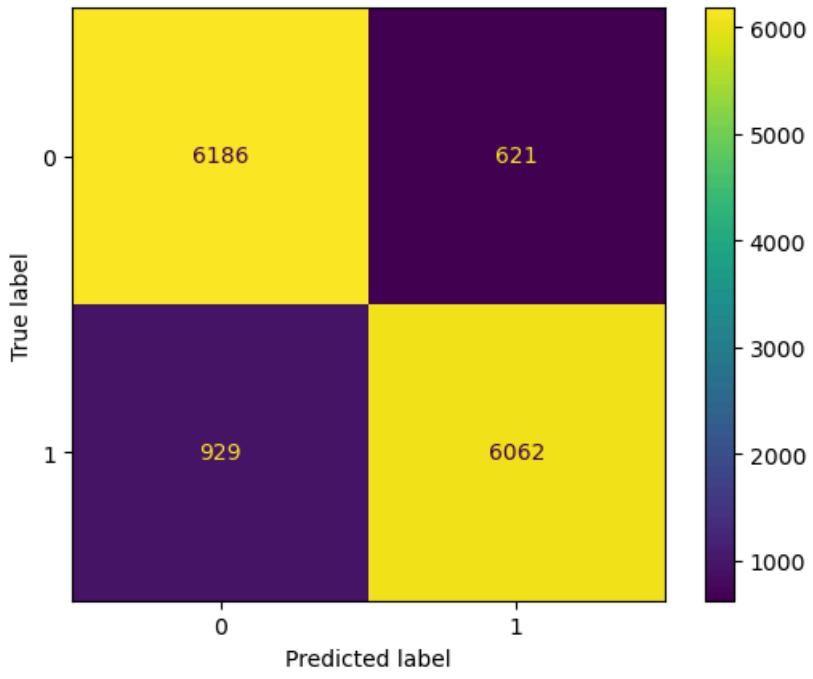
Out[ ]:
```
RandomForestClassifier(bootstrap=False, max_depth=50)
```

In [ ]:
```
# predictions
y_pred = randmf.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89      6807
           1       0.90      0.86      0.88      6991

    accuracy                           0.89     13798
   macro avg       0.89      0.89      0.89     13798
weighted avg       0.89      0.89      0.89     13798
```

# Increase in 3% of accuracy from already fitted model

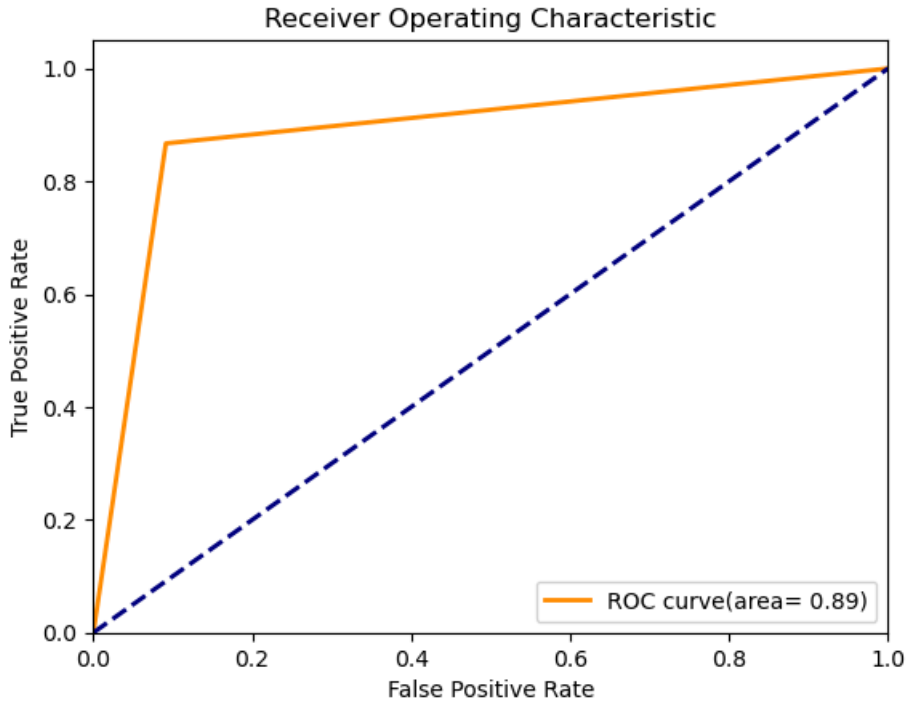
In [ ]:
```
cm = confusion_matrix(y_test,y_pred)

dip = ConfusionMatrixDisplay(confusion_matrix=cm)
dip.plot()
plt.show()
```




In [ ]:
```
auc_roc = roc_auc_score(y_test,y_pred)
print(f'AUC-ROC Score:{auc_roc}')

# step 6 : plot ROC curve
fpr,tpr,_ = roc_curve(y_test,y_pred)
```

```
# Calculates the FPR and TPR
# at various threshold settings for the given y_test(actual)
# and y_predict(predicted probabilities or scores).

plt.figure()
# creates a new figure for the plot

plt.plot(fpr,tpr,color='darkorange',lw=2,label='ROC curve(area= %0.2f)'%auc_roc)
# plots the roc curve using  the FPR (x-axis) and TPR(y-axis) values
# color
#lw = line width = 2
# label = auc-roc adds a label to the curve
# showing the AUC -ROC score.

plt.plot([0,1],[0,1],color='navy',lw=2,linestyle='--')
# plot a diagonal line from(0,0), to (1,1)representing
# the roc curve of a purely random classifier.


plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])

#sets the limits for the x-axis and y-axis representively.
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```

```
AUC-ROC Score:0.8879426226971037
```




- Key things to note
- Even the new CV model has increase in accuracy score
- The True positive count is reduced.
- Even increase in overall prediction is statisfies 89%.
- We need to predict the target variable with max accuracy
- So I will go with previous model or retry with different range of parameters.
- This could help the clients to reach good prediction acuuracy.
- Because infected patient predictability is the need of this dataset

In [ ]: