Covid_predictions

February 6, 2024

1 COVID Predictions

In this short project, I aim to build a classification model that indicates whether a patient is at high risk due to covid or not. I will use a dataset from Kaggle, provided under a creative commons licence.

Note about this Jupyter Notebook

- In tackling this problem, I will employ a workflow inspired from the workflow in Appendix A of the "Hands-On Machine Learningwith Sci-kit Learn, Keras and Tensorflo", Third Edition by Aurelien Geron : https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/

1.1 Workflow

This end-to-end Machine Learning Project employs the following workflow. To quickly navigate to the desired section, please click on the title of the section

0. Frame the problem

1. Inspect and Clean the Data

- Get the Data
- Data Profiling
- Bringing Missing Values to a Consistent Format
- Examining Missing Values
- Missing Values Correlation
- Dealing with Missing Values
- Clean Data Visualization

2. Split the Data

- Undersampling the Majority Class
- Train/Test

3. Explore the Data

- Looking for Correlations
- Rxperimenting with Transformation

4. Pre-process Data for Machine Learning Algorithms

- Built Pre-processing Pipeline
- Evaluation Metric

- Create New Variables for Evaluation

5. Select and train models

- Dummy Classifier
- SGD Classifier
- RandomForest Classifier
- Logistic Regression
- Nearest Neighbours Classifier
- Multi Layer Perceptron
- AdaBoost

6. Voting Classifier

- Soft Voting

7. Generalization Errors of Best Model

- Get Final Predictions
- MAE
- Confusion Matrix
- Accuracy

8. Launch

2 0. Frame the problem

The goal is to develop a predictive model for COVID-19 high risk patients based on historical data. I will attempt to find a modelling approach that (i) gives a specified cut-off for high-risk patients and (ii) returns a probability model together with a confidence interval that a patient is at high risk. The decision will be based only on the variables provided in this dataset, such that, if new data becomes available or different data is fed the model can be externally validated. An evaluation function consisisting of several metrics will be used to internally validate the model. After cleaning the data, I will make training/test set splits for evaluation. After testing many models on the validation set, the best one (the model that gives best performance in terms of its generalization error) will be selected . Finally, some conclusions and interpretation of results will be offered with this notebook.

The data can be found at Kaggle : https://www.kaggle.com/datasets/meirnizri/covid19-dataset/

According to Kaggle : "The raw dataset consists of 21 different features and 1,048,576 unique patients. In the Boolean features, 1 means"yes" and 2 means "no". values as 97 and 99 are missing data."

Info about the dataset, including a dictionary is given on Kaggle: https://www.kaggle.com/datasets/meirnizri/covid19-dataset/

For ease of access, we reproduce it here :

"The dataset was provided by the Mexican government (link). This dataset contains an enormous number of anonymized patient-related information including pre-conditions. The raw dataset consists of 21 unique features and 1,048,576 unique patients. In the Boolean features, 1 means"yes" and 2 means "no". values as 97 and 99 are missing data.

- sex: 1 for female and 2 for male.
- age: of the patient.
- classification: covid test findings. Values 1-3 mean that the patient was diagnosed with covid in different -degrees. 4 or higher means that the patient is not a carrier of covid or that the test is inconclusive.
- patient type: type of care the patient received in the unit. 1 for returned home and 2 for hospitalization.
- pneumonia: whether the patient already have air sacs inflammation or not.
- pregnancy: whether the patient is pregnant or not.
- diabetes: whether the patient has diabetes or not.
- copd: Indicates whether the patient has Chronic obstructive pulmonary disease or not.
- asthma: whether the patient has asthma or not.
- inmsupr: whether the patient is immunosuppressed or not.
- hypertension: whether the patient has hypertension or not.
- cardiovascular: whether the patient has heart or blood vessels related disease.
- renal chronic: whether the patient has chronic renal disease or not.
- other disease: whether the patient has other disease or not.
- obesity: whether the patient is obese or not.
- tobacco: whether the patient is a tobacco user.
- usmr: Indicates whether the patient treated medical units of the first, second or third level.
- medical unit: type of institution of the National Health System that provided the care.
- intubed: whether the patient was connected to the ventilator.
- icu: Indicates whether the patient had been admitted to an Intensive Care Unit.
- date died: If the patient died indicate the date of death, and 9999-99-99 otherwise."

Before proceeding further, let's import the necessary libraries for data manipulation and visualization, together with an automatic EDA tool : ydata_profiling

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from ydata_profiling import ProfileReport
%matplotlib inline
```

3 1. Inspect and Clean the Data

3.1 Get the Data

I will use the data from Kaggle (it can be found in the link at the top of this notebook). But first, I will write a function that download the data directly from there.

[2]:
```
import os
from kaggle.api.kaggle_api_extended import KaggleApi

def download_kaggle_dataset(dataset):
    api = KaggleApi()
    api.authenticate()
```

```
    ## Create a directory to store the data
    os.makedirs('kaggle_covid_data', exist_ok=True)

    ## Download the dataset
    api.dataset_download_files(dataset, path='kaggle_covid_data', unzip=True)

## Use the function defined above to download the dataset
## It is sufficient to use the dataset identifier found in the dataset URL
download_kaggle_dataset('meirnizri/covid19-dataset/')
```

We upload the data into a dataframe

[3]:
```
covid_df = pd.read_csv('kaggle_covid_data/Covid Data.csv')
```

Let's create a copy of the data, to preserve the original dataframe in case it is needed later for comparison.

[4]:
```
covid = covid_df.copy()
```

Let us have a quick look at the data

[5]:
```
##evaluating the first 10 rows of the dataframe
covid.head(10)
```

[5]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	\
0	2	1	1	1	03/05/2020	97	1	
1	2	1	2	1	03/06/2020	97	1	
2	2	1	2	2	09/06/2020	1	2	
3	2	1	1	1	12/06/2020	97	2	
4	2	1	2	1	21/06/2020	97	2	
5	2	1	1	2	9999-99-99	2	1	
6	2	1	1	1	9999-99-99	97	2	
7	2	1	1	1	9999-99-99	97	1	
8	2	1	1	2	9999-99-99	2	2	
9	2	1	1	2	9999-99-99	2	2	

	AGE	PREGNANT	DIABETES	…	ASTHMA	INMSUPR	HIPERTENSION	OTHER_DISEASE	\
0	65	2	2	…	2	2	1	2	
1	72	97	2	…	2	2	1	2	
2	55	97	1	…	2	2	2	2	
3	53	2	2	…	2	2	2	2	
4	68	97	1	…	2	2	1	2	
5	40	2	2	…	2	2	2	2	
6	64	2	2	…	2	2	2	2	
7	64	2	1	…	2	1	1	2	
8	37	2	1	…	2	2	1	2	
9	25	2	2	…	2	2	2	2	

	CARDIOVASCULAR	OBESITY	RENAL_CHRONIC	TOBACCO	CLASIFFICATION_FINAL	ICU
0	2	2	2	2	3	97
1	2	1	1	2	5	97
2	2	2	2	2	3	2
3	2	2	2	2	7	97
4	2	2	2	2	3	97
5	2	2	2	2	3	2
6	2	2	2	2	3	97
7	2	2	1	2	3	97
8	2	1	2	2	3	2
9	2	2	2	2	3	2

[10 rows x 21 columns]

[6]:
```
##looking at the last 10 rows of the dataframe
covid.tail(10)
```

[6]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	\
1048565	1	13	1	1	9999-99-99	97	
1048566	2	13	2	1	9999-99-99	97	
1048567	1	13	1	2	9999-99-99	2	
1048568	1	13	2	1	9999-99-99	97	
1048569	1	13	1	2	9999-99-99	2	
1048570	2	13	2	1	9999-99-99	97	
1048571	1	13	2	2	9999-99-99	2	
1048572	2	13	2	1	9999-99-99	97	
1048573	2	13	2	1	9999-99-99	97	
1048574	2	13	2	1	9999-99-99	97	

	PNEUMONIA	AGE	PREGNANT	DIABETES	…	ASTHMA	INMSUPR	\
1048565	2	39	2	2	…	2	2	
1048566	2	24	97	2	…	1	2	
1048567	2	23	2	1	…	2	2	
1048568	2	47	97	1	…	2	2	
1048569	2	56	2	1	…	2	2	
1048570	2	40	97	2	…	2	2	
1048571	2	51	97	2	…	2	2	
1048572	2	55	97	2	…	2	2	
1048573	2	28	97	2	…	2	2	
1048574	2	52	97	2	…	2	2	

	HIPERTENSION	OTHER_DISEASE	CARDIOVASCULAR	OBESITY	RENAL_CHRONIC	\
1048565	2	2	2	2	2	
1048566	2	2	2	2	2	
1048567	2	2	2	1	2	
1048568	2	2	2	2	2	
1048569	2	2	2	2	2	

```
1048570            2              2              2        2              2
1048571            1              2              2        2              2
1048572            2              2              2        2              2
1048573            2              2              2        2              2
1048574            2              2              2        2              2

         TOBACCO  CLASIFFICATION_FINAL  ICU
1048565        2                     7   97
1048566        2                     7   97
1048567        2                     7    2
1048568        2                     7   97
1048569        2                     7    2
1048570        2                     7   97
1048571        2                     7    2
1048572        2                     7   97
1048573        2                     7   97
1048574        2                     7   97

[10 rows x 21 columns]
```

[7]:
```
##shape of the dataframe
covid.shape
```

[7]:
```
(1048575, 21)
```

Note: There are 1048575 observations (rows) and 21 columns

Are there any null values? Let's further investigate this in order to find a way to deal with them. This is because Machine Learning models to not handle well missing (NaN) values.

[8]:
```
##this shows the data types of the columns
covid.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   USMER                 1048575 non-null  int64
 1   MEDICAL_UNIT          1048575 non-null  int64
 2   SEX                   1048575 non-null  int64
 3   PATIENT_TYPE          1048575 non-null  int64
 4   DATE_DIED             1048575 non-null  object
 5   INTUBED               1048575 non-null  int64
 6   PNEUMONIA             1048575 non-null  int64
 7   AGE                   1048575 non-null  int64
 8   PREGNANT              1048575 non-null  int64
 9   DIABETES              1048575 non-null  int64
 10  COPD                  1048575 non-null  int64
```

```
 11  ASTHMA                1048575 non-null  int64
 12  INMSUPR               1048575 non-null  int64
 13  HIPERTENSION          1048575 non-null  int64
 14  OTHER_DISEASE         1048575 non-null  int64
 15  CARDIOVASCULAR        1048575 non-null  int64
 16  OBESITY               1048575 non-null  int64
 17  RENAL_CHRONIC         1048575 non-null  int64
 18  TOBACCO               1048575 non-null  int64
 19  CLASIFFICATION_FINAL  1048575 non-null  int64
 20  ICU                   1048575 non-null  int64
dtypes: int64(20), object(1)
memory usage: 168.0+ MB
```

Observation: There are object, integer and float data.

[9]:
```
covid_isnull = covid.isnull().sum()
covid_isnull
```

[9]:
```
USMER                 0
MEDICAL_UNIT          0
SEX                   0
PATIENT_TYPE          0
DATE_DIED             0
INTUBED               0
PNEUMONIA             0
AGE                   0
PREGNANT              0
DIABETES              0
COPD                  0
ASTHMA                0
INMSUPR               0
HIPERTENSION          0
OTHER_DISEASE         0
CARDIOVASCULAR        0
OBESITY               0
RENAL_CHRONIC         0
TOBACCO               0
CLASIFFICATION_FINAL  0
ICU                   0
dtype: int64
```

3.2 Data Profiling

There are no null values. We can run a profile report to see any interesting trends in the data

[10]:
```
html_report = ProfileReport(covid)
html_report.to_file(output_file='covid_data.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[11]:
```
html_report
```

```
<IPython.core.display.HTML object>
```

[11]:

Observations on Profile Report:

- We have 5.3% duplicate rows
- Several categorical columns are highly imbalanced
- There are no null values, but missing values are marked as 98, 99
- We have "1" for positive and "2" for negative (with other numbers signifying no data)
- The Age of the patient closely resembles a normal distribution
- Pregnancy status is largely missing
- There is a relatively small number of missing observations regarding comorbidity : diabetes, copd, asthma etc
- Classification Final (according to the dictionary) shows covid status; for our purposes, I must create new, well-defined categories based on the data in this column that will serve as targets and inform of the severity of covid
- Apparently, there are high correlations between columns indicating diseases
- There are 55672 duplicate rows (approx 5.3%) : these provide no value and must be eliminated from the dataframe

In the following, I will start cleaning the data. After the data is clean, I can create a test set and a validation set right away and begin experimenting on the training set. The succesfull transformations will be integrated into a pipeline and applied to the data - this is to ensure the data is ready to be fed to any ML model. For exploration, I will create a copy of the data. Once the final pipeline is made, it will be applied on any covid data.

The plan for the cleaning is:

- replace "2" with "0" and any other numbers in categorical numbers with NaN : please note that encoding "no" as 0 instead of "2" is a personal choice meant to increase interpretability of the data
- clarify male/female categories : I will create explicit categories showing the sex of the patient
- drop duplicates

3.3 Bring Missing Values to a Consistent Format

[12]:
```
covid.columns
```

[12]:
```
Index(['USMER', 'MEDICAL_UNIT', 'SEX', 'PATIENT_TYPE', 'DATE_DIED', 'INTUBED',
       'PNEUMONIA', 'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA', 'INMSUPR',
       'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY',
       'RENAL_CHRONIC', 'TOBACCO', 'CLASIFFICATION_FINAL', 'ICU'],
```

```
      dtype='object')
```

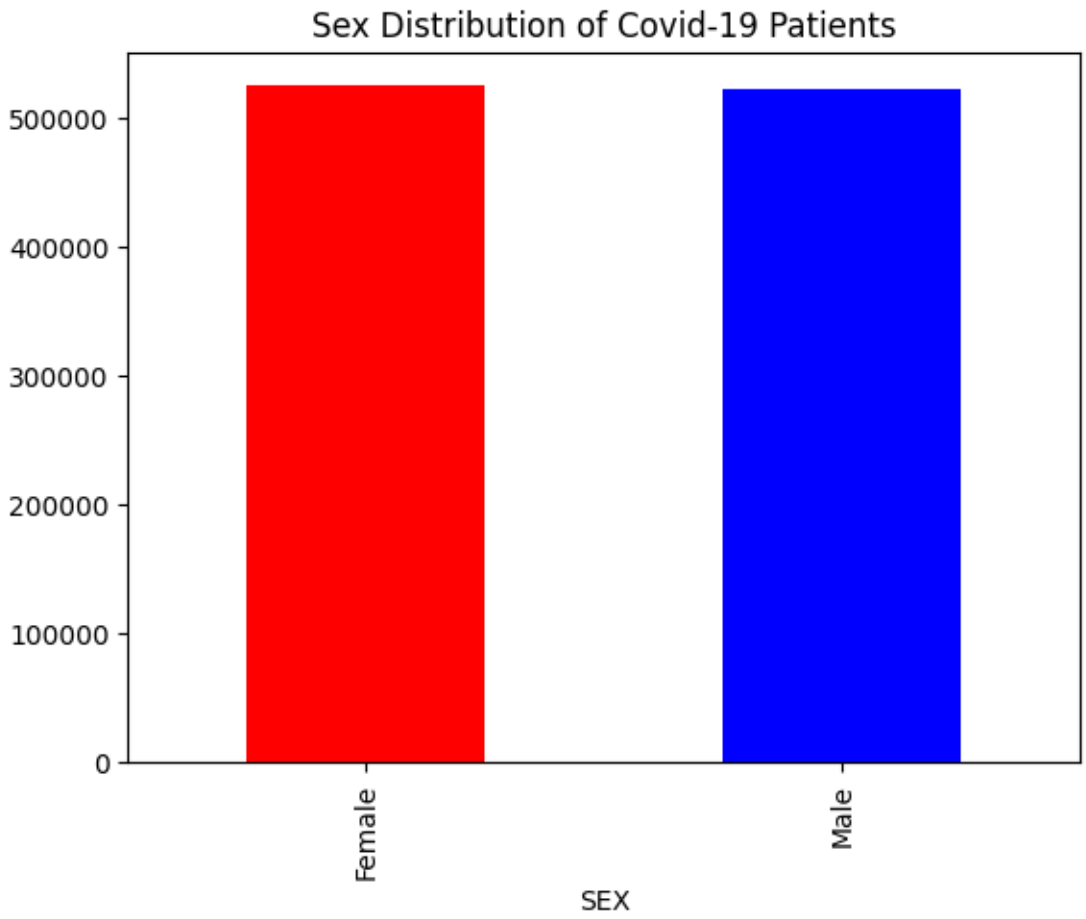
As a first step, let us replace the “2” with “0”, so the columns can become one-hot encoded

[13]:
```
##replace "2" with 0 in categorical columns that show the presence of a symptom
covid['USMER'].replace(2,0, inplace=True)
covid['INTUBED'].replace(2,0, inplace=True)
covid['PNEUMONIA'].replace(2,0, inplace=True)
covid['DIABETES'].replace(2,0, inplace=True)
covid['COPD'].replace(2,0, inplace=True)
covid['ASTHMA'].replace(2,0, inplace=True)
covid['INMSUPR'].replace(2,0, inplace=True)
covid['HIPERTENSION'].replace(2,0, inplace=True)
covid['OTHER_DISEASE'].replace(2,0, inplace=True)
covid['CARDIOVASCULAR'].replace(2,0, inplace=True)
covid['OBESITY'].replace(2,0, inplace=True)
covid['RENAL_CHRONIC'].replace(2,0, inplace=True)
covid['TOBACCO'].replace(2,0, inplace=True)
covid['ICU'].replace(2,0, inplace=True)
covid['PREGNANT'].replace(2,0, inplace=True)
```

Let’s replace the sex with Male(2) or Female(1)

[14]:
```
covid["SEX"] = covid['SEX'].astype('string')
covid['SEX'].replace({"1": 'Female', "2": 'Male'}, inplace=True)
```

[15]:
```
covid["SEX"].value_counts().plot(kind='bar', color=['red', 'blue'])
plt.title("Sex Distribution of Covid-19 Patients")
plt.show()
```



Observe:

- we have succesfully replaced the two distributions with “Male” and “Female”
- the distributions are roughly equal, so there is no sex that is less affected

Let’s also make it clear if the patient was sent home(1) or hospitalized(2)

[16]:
```
covid["PATIENT_TYPE"] = covid['PATIENT_TYPE'].astype('string')
covid['PATIENT_TYPE'].replace({"1": 'Home', "2": 'Hospital'}, inplace=True)
```

To understand how severe a particular covid case was, it is very important to understand if the patient died or not. Let us create two columns that one-hot encode whether the patient passed away

[17]:
```
covid["Lower_Risk"] = covid["DATE_DIED"].apply(lambda x: 1 if x == "9999-99-99"␣
 ↪else 0)
covid["Higher_Risk"] = covid["DATE_DIED"].apply(lambda x: 1 if x !=␣
 ↪"9999-99-99" else 0)
```

Observe: Based on whether the patients died or survived, I chose to encode the patients as

"Higher_Risk" or "Lower_Risk". The ratinale is that we can infer the patients who were at high_risk based on the information on whether they survived or not.

Finally, let us get rid of missing data. Instead of 97 or 98 in our categorical columns, let us fill the values with NaN. Then, based on the number of NaN and the underlying distributions, let's make decisions on how to deal with them.

[18]:
```
covid.replace({97: np.nan, 98: np.nan, 99: np.nan}, inplace=True)
```

Now, let us see how many NaN (null) values we have for each row and decide the best strategy to deal with them

[19]:
```
covid.isnull().sum()
```

[19]:
```
USMER                       0
MEDICAL_UNIT                0
SEX                         0
PATIENT_TYPE                0
DATE_DIED                   0
INTUBED                855869
PNEUMONIA               16003
AGE                       345
PREGNANT                527265
DIABETES                  3338
COPD                      3003
ASTHMA                    2979
INMSUPR                   3404
HIPERTENSION              3104
OTHER_DISEASE             5045
CARDIOVASCULAR            3076
OBESITY                   3032
RENAL_CHRONIC             3006
TOBACCO                   3220
CLASIFFICATION_FINAL         0
ICU                     856032
Lower_Risk                   0
Higher_Risk                  0
dtype: int64
```

Observe:

- The "INTUBED" and "ICU" and "PREGNANT" have over 80% missing values; they do follow the distributions observed in the EDA
- For the remaining columns we have at most 0.5% null values.
- there are multiple options for dealling with missing data

3.4 Examining Missing Data

The first step in dealing with missing data, is to *understand if there is a pattern.* Data may be missing at random (for example, due to imputation errors) or it might be connected to the other

observations. For example, a missing entry for "INTUBED" may also be accompanied by a missing entry in "ICU". The data for "PREGNANT" should necessarily be correlated with the category "FEMALE". In any case, any particular imputation method should be informed by the data. The strategy to do this is as follows:

1. Look at the distributions of the features with missing and understand if there is any patterns that emerge
2. Use a *missingness* correlation. This indicates whether missing data in one column predicts the existence of missing data in another column

Information on the missingno library that provides a small toolset for understanding and visualizing data can be found at : https://github.com/ResidentMario/missingno

[20]:
```
##select columns with missing data

##seelct numerical features
num_features = covid.select_dtypes(include=[np.number])
##seelct feature with no missing data
num_features_no_missing = num_features.dropna(axis=1)
##select features with missing data
num_features_missing = num_features.drop(num_features_no_missing.columns,␣
 ↪axis=1)
```

[21]:
```
##these are the columns with no missing data
num_features_no_missing.columns
```

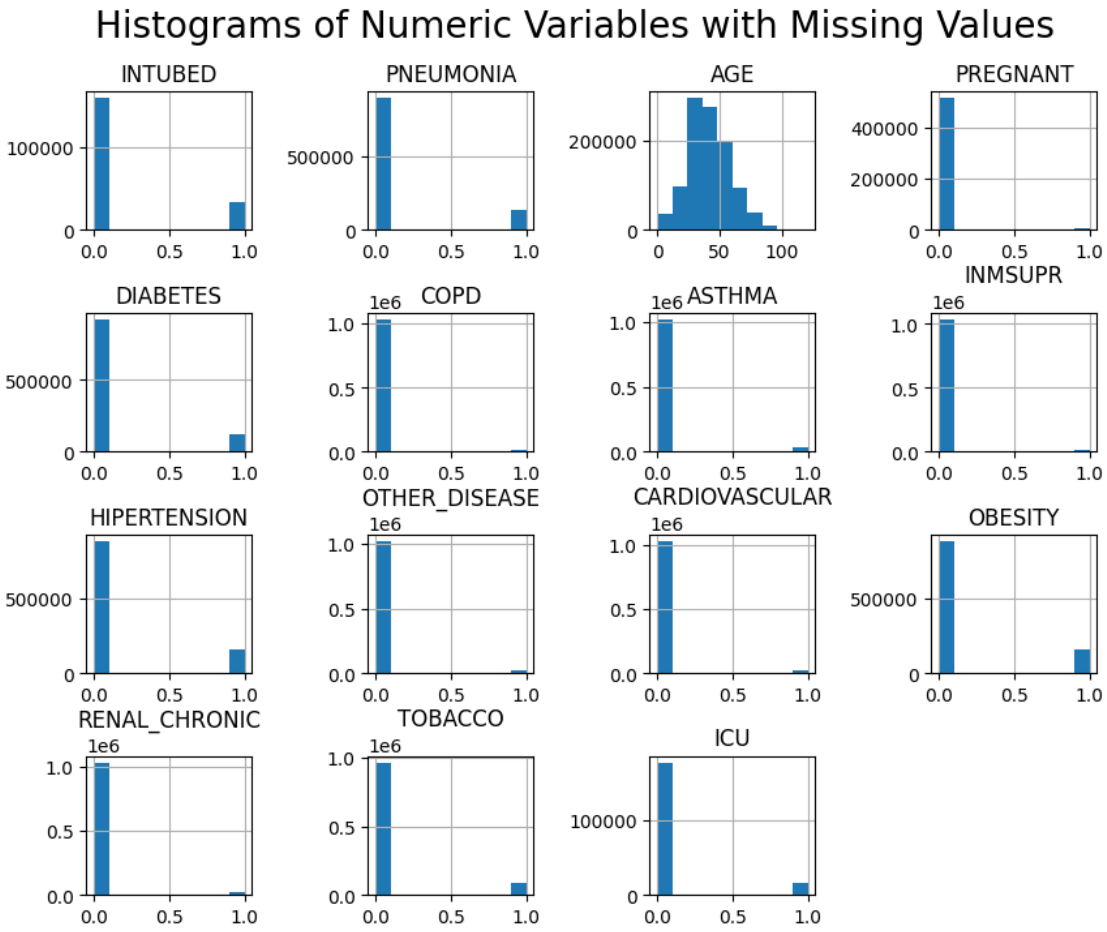
[21]:
```
Index(['USMER', 'MEDICAL_UNIT', 'CLASIFFICATION_FINAL', 'Lower_Risk',
       'Higher_Risk'],
      dtype='object')
```

[22]:
```
##these are the columns with missing data
num_features_missing.columns
```

[22]:
```
Index(['INTUBED', 'PNEUMONIA', 'AGE', 'PREGNANT', 'DIABETES', 'COPD', 'ASTHMA',
       'INMSUPR', 'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY',
       'RENAL_CHRONIC', 'TOBACCO', 'ICU'],
      dtype='object')
```

Let's not plot the columns with missing data, and the columns with no missing data

[23]:
```
##histograms of the numerical features with missing data
num_features_missing.hist(figsize=(10,8))
plt.subplots_adjust(hspace=0.6, wspace=0.7)
plt.suptitle("Histograms of Numeric Variables with Missing Values",␣
 ↪fontsize=20, y=0.96)
plt.rcParams.update({'font.size': 10})
plt.show()
```

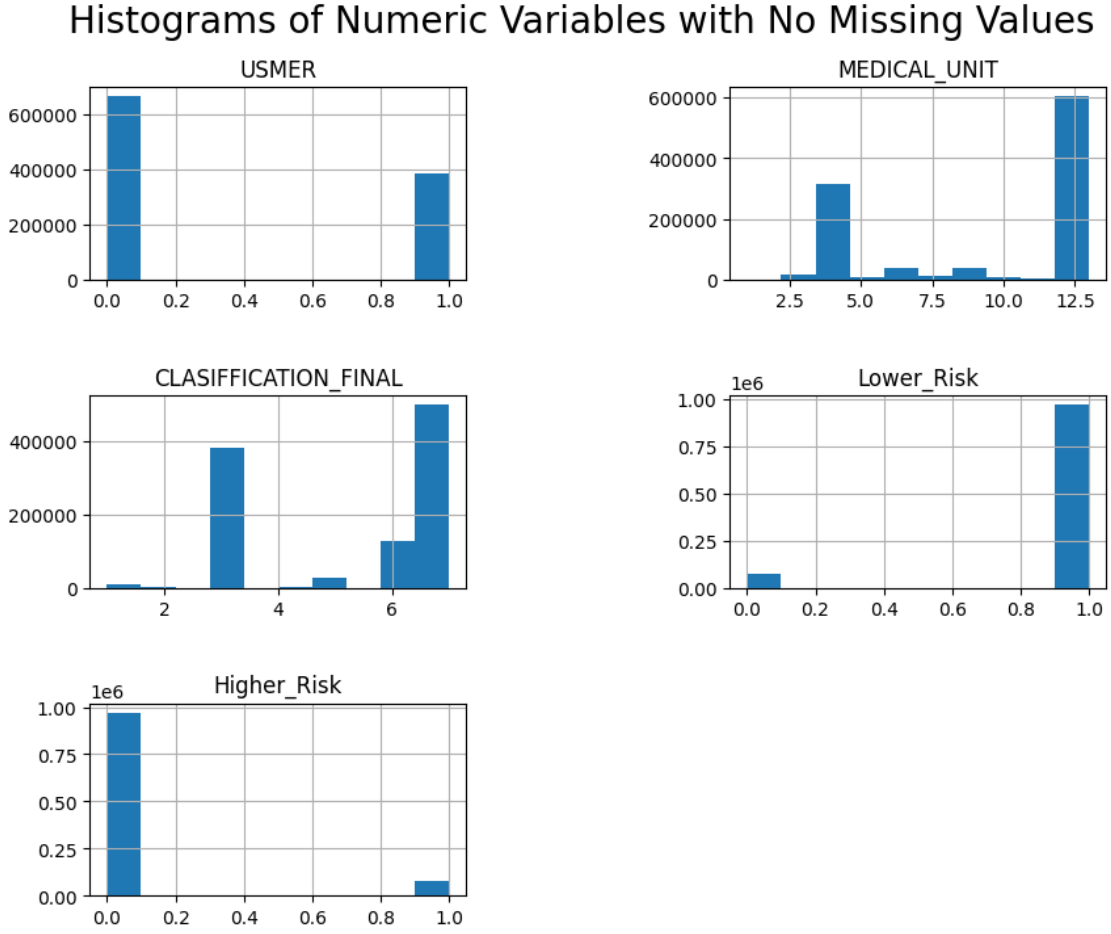


Observe:

- All distributions are clearly categorical and one-hot encoded
- The categorical distributions are highly skewed, so replacing the data based on most frequent would likely introduce bias

Let's now look at the columns with no missing values and understand how these distributions are shaped.

[24]:
```
##histograms of the numerical features with no missing data
num_features_no_missing.hist(figsize=(10,8))
plt.subplots_adjust(hspace=0.6, wspace=0.7)
plt.suptitle("Histograms of Numeric Variables with No Missing Values",␣
 ↪fontsize=20, y=0.96)
plt.rcParams.update({'font.size': 10})
plt.show()
```



Let us check the "PREGNANT" data and it's relation to "SEX". This is easy to check and clean as only "FEMALE" can have positive "PREGNANT"

[25]:
```
covid_pregnant_sex = covid[["PREGNANT", "SEX"]]
```

[26]:
```
covid_pregnant_sex[covid_pregnant_sex["SEX"] =='Male'].value_counts()
```

[26]:
```
Series([], Name: count, dtype: int64)
```

[27]:
```
covid_pregnant_sex[covid_pregnant_sex["SEX"] =='Female'].value_counts()
```

[27]:
```
PREGNANT  SEX
0.0       Female    513179
1.0       Female      8131
Name: count, dtype: int64
```

There are no "PREGNANT" values associated with "Male", but just in case we can try to clean this

[28]:
```
covid.loc[:, 'PREGNANT'] = covid["PREGNANT"].apply(lambda x: 0 if x == 'Male'␣
↪else x)
```

[29]:
```
covid['PREGNANT'].value_counts()
```

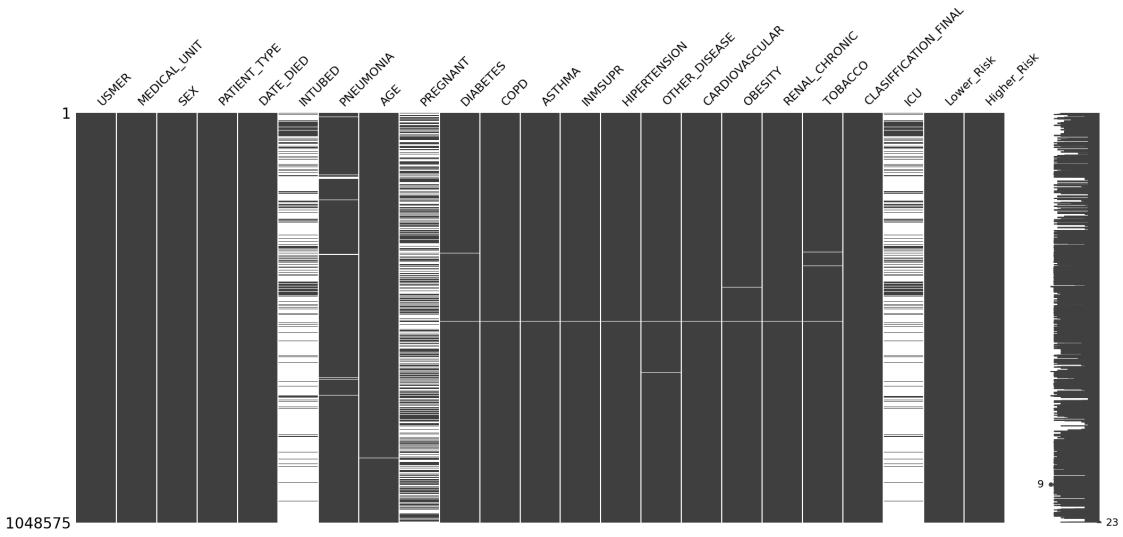
[29]:
```
PREGNANT
0.0    513179
1.0      8131
Name: count, dtype: int64
```

3.5 Missing Values Correlations

[30]:
```
import missingno as msno
```

[31]:
```
##plot the missing data points scaterred
plt.figure(figsize=(10,5))
msno.matrix(covid)
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```

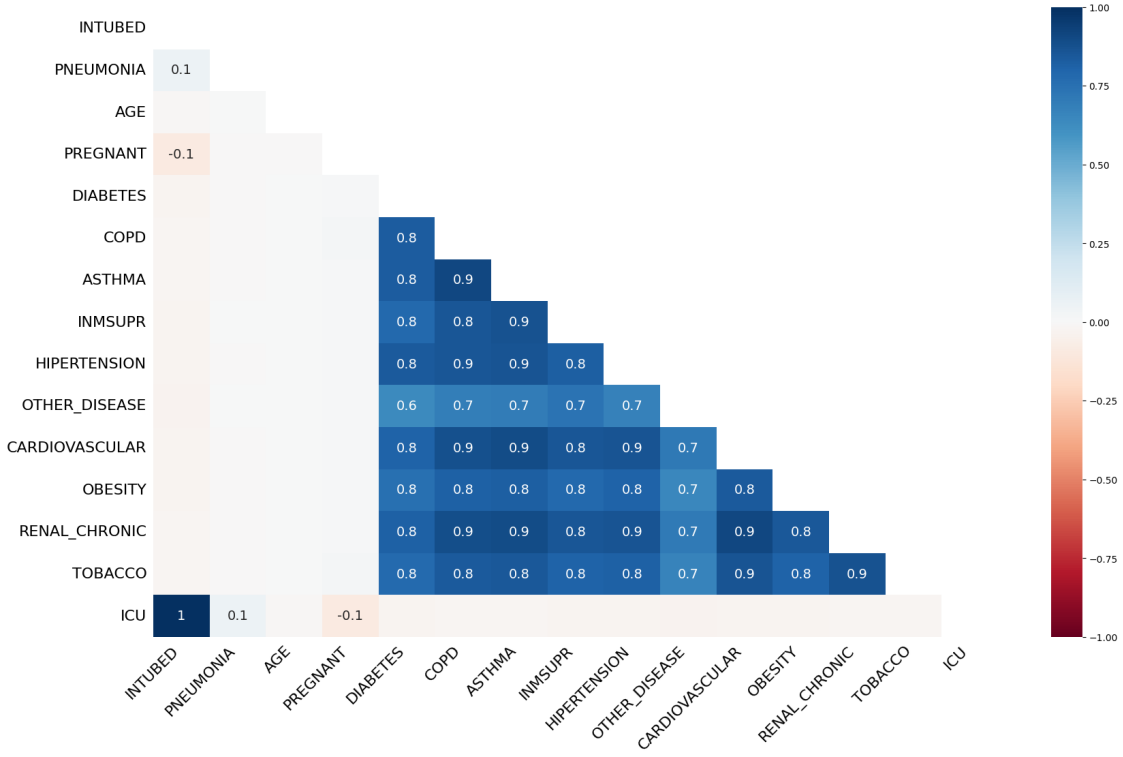


Observe:

- There is clearly a missing pattern in "INTUBED" and "ICU" : the missing values seem perfectly correlated
- The columns dealing with diseases seem to be correlated in terms of missing values

[32]:
```
## plot the missingness correlation heatmap
msno.heatmap(covid)
```

[32]: <Axes: >



Observation :

- Missing entries for “INTUBED” and “ICU| are perfectly correlated; this is because patients must be admitted to an Intensive Care Unit (ICU) in order to be”INTUBED”. Hence, data for these two columns should be treated together
- There is a very high correlation of data missing for all diseases: if there is data missing for one disease, it is very likely missing for all other diseases.
- It seems that “PREGNANT” missing values are not correlated with anything else, this data is probably simply not available

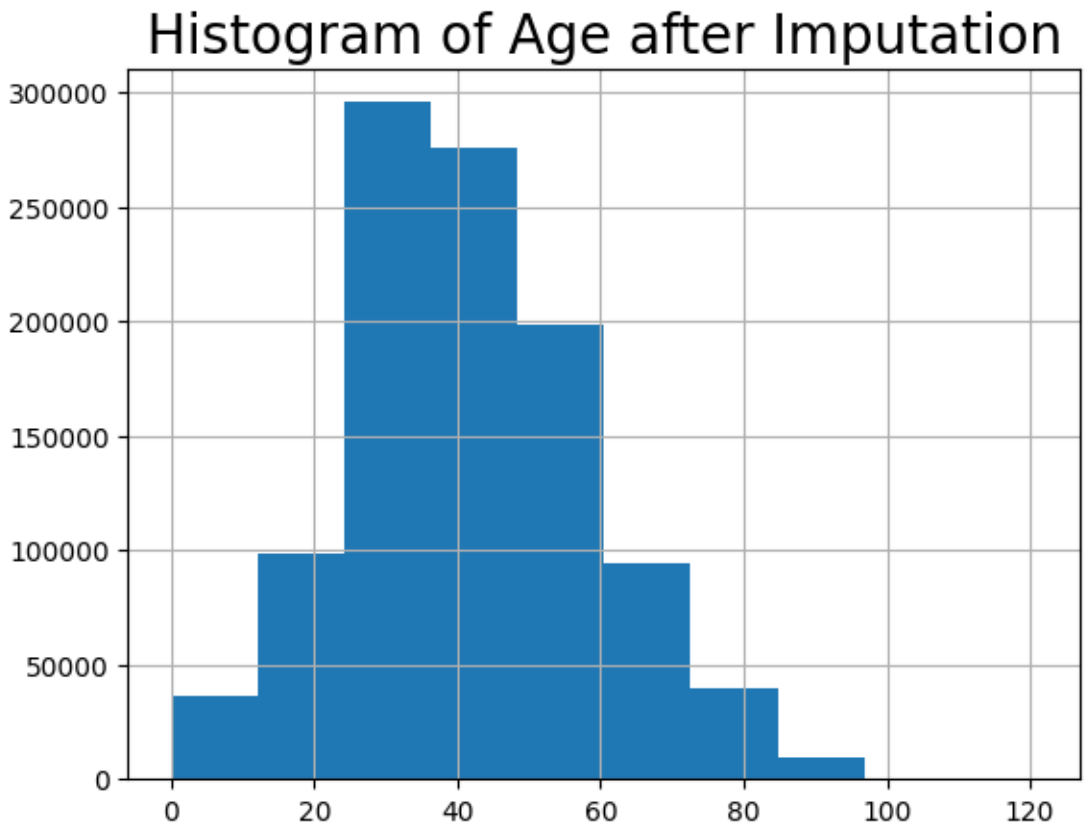
3.6 Dealing with Missing Values

Dealing with missing values can be done in various ways: by dropping rows with missing values, imputing the missing values or inserting some placeholder instead of the missing value. Dropping means that all associated rows will be lost, while imputing does introduce some bias depending on the model used. One smart way to deal with this is to look at the number of missing values and the corresponding distributions and then deal with them individually

The “AGE” column only has 345 missing values, these can be easily be replaced with the median of the column without significantly affecting the distribution. Given the number of missing values in the “AGE” column, this must be a case of missing at random errors

[33]:
```
##for age we can simply impute the median
covid['AGE'].fillna(covid['AGE'].median(), inplace=True)
```

[34]:
```
covid['AGE'].hist()
plt.title("Histogram of Age after Imputation", fontsize=20)
plt.show()
```



Now, the rest of the columns mostly have a number of missing data that is clearly not at random. This is particularly true of the columns associated to diseases : where one disease is missing, data regarding disease is missing too. This columns should be somehow "marked" as missing, so any algorithm learns differently from them. Let us fill them with some value that indicates *missingness*

[35]:
```
disease_features = [ 'PNEUMONIA', 'DIABETES', 'COPD', 'ASTHMA',
       'INMSUPR', 'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR', 'OBESITY',
       'RENAL_CHRONIC', 'TOBACCO']
```

[36]:
```
for col in disease_features:
    covid[col].fillna(0.5, inplace=True)
```

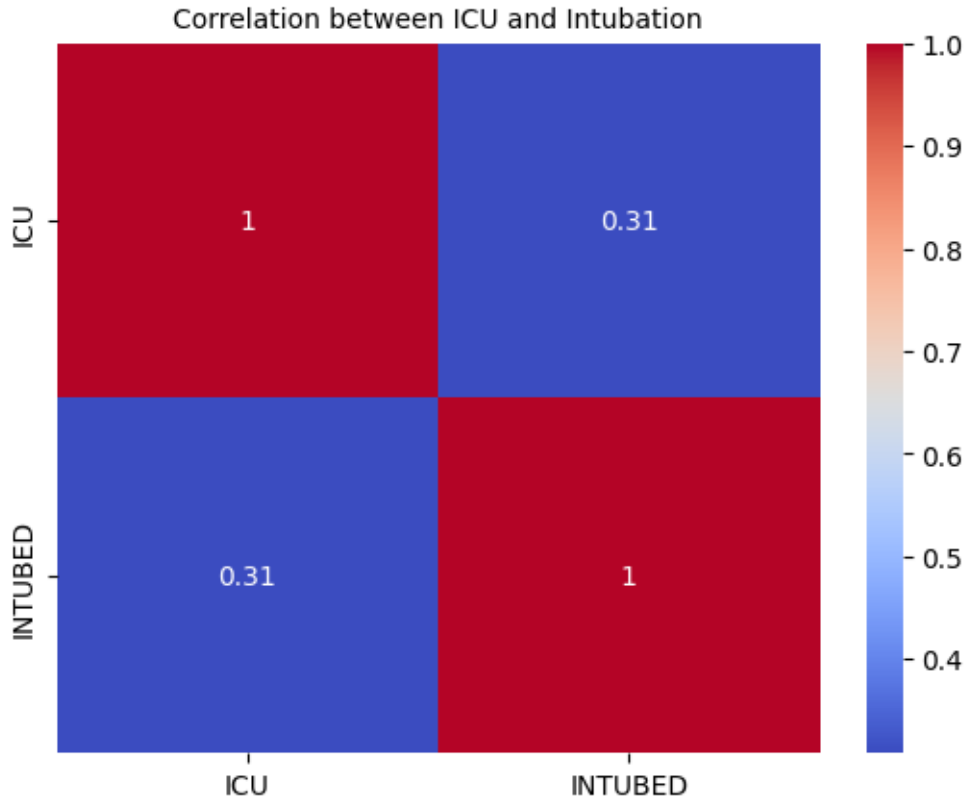
Let's do the same with pregnancy, we introduce 0.5 (between 0 and 1) to indicate a new category of uncertainity. Hence, by 0.5 the value "unknown" is encoded

[37]:
```
covid["PREGNANT"].fillna(0.5, inplace=True)
```

Let's now examine "ICU" and "INTUBED" more closey. If the existing values of these categories are also perfectly correlated there is no need to keep both of them into the final dataset.

[38]:
```
corr_icu_intubed = covid[['ICU', 'INTUBED']].corr()
```

[39]:
```
sns.heatmap(corr_icu_intubed, annot=True, cmap='coolwarm')
plt.title("Correlation between ICU and Intubation", fontsize=10)
plt.show()
```



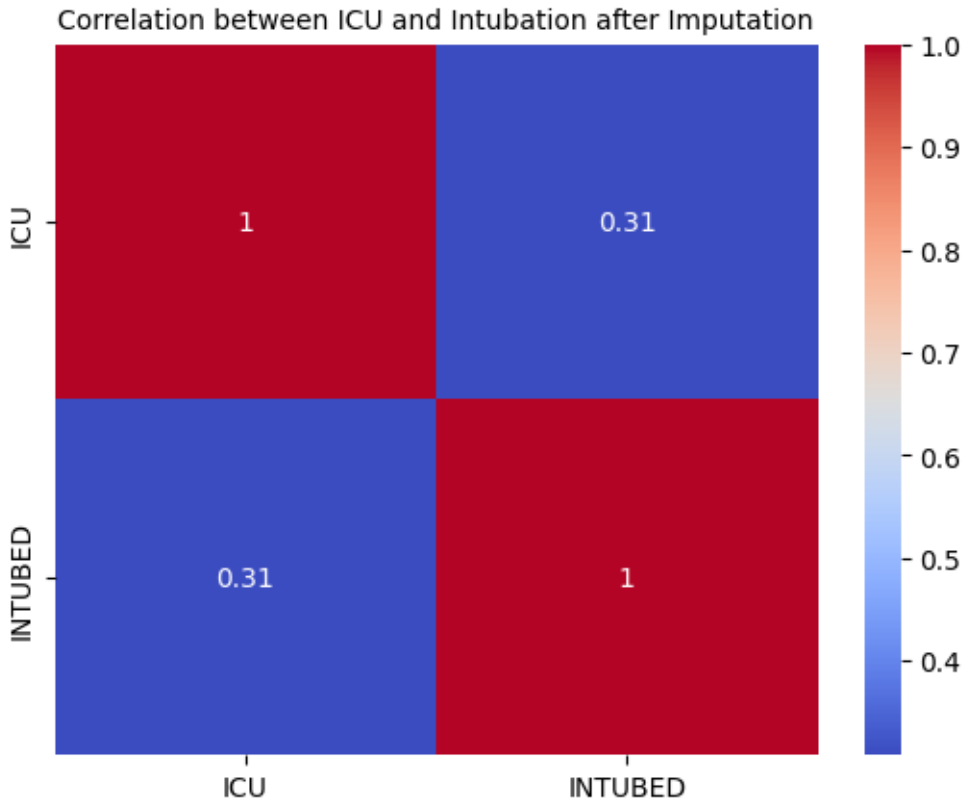
It turns out the values are moderately correlated, hence it would make sense to deal with the "missingness" of each column while keeping the two values separated.

[40]:
```
##this fills the missing values in the INTUBED column with 0.5
covid["INTUBED"].fillna(0.5, inplace=True)
```

[41]:
```
##this fills the missing values in the ICU column with 0.5
covid["ICU"].fillna(0.5, inplace=True)
```

[42]:
```
sns.heatmap(corr_icu_intubed, annot=True, cmap='coolwarm')
```

```
plt.title("Correlation between ICU and Intubation after Imputation",␣
  ↪fontsize=10)
plt.show()
```



In sum, In all binary columns I introuduced 0.5 to indicate missing information. Although this strategy is simple, it can prove useful when one-hot encoding the categorical columns such that *missingness* of various types of data also becomes a feature

It is important to create labels based on the "CLASSIFICATION_FINAL" : based on the dictionary <=3 indicates COVID diagnosis, while >=4 means that the patient may not be a carrier of covid

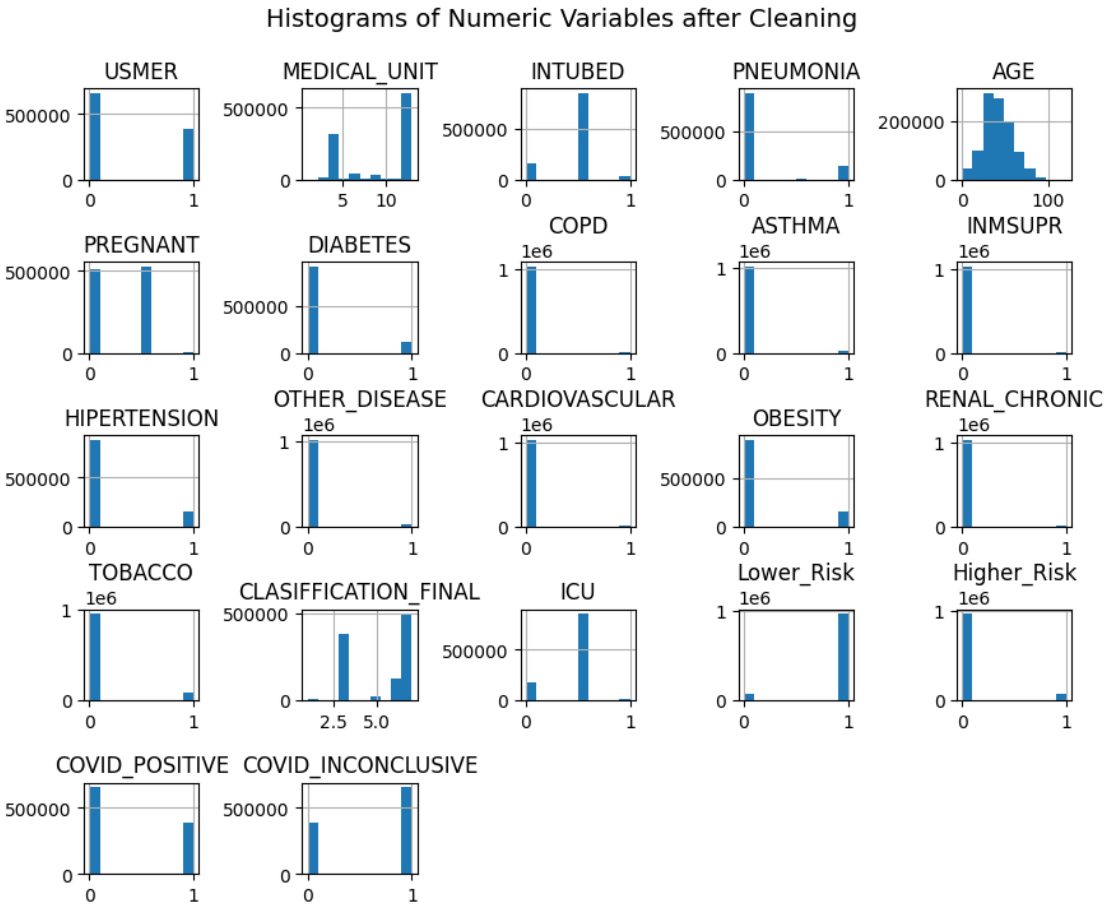
[43]:
```
covid['COVID_POSITIVE'] = covid['CLASIFFICATION_FINAL'].apply(lambda x: 1 if␣
  ↪x<=3 else 0)
covid['COVID_INCONCLUSIVE'] = covid['CLASIFFICATION_FINAL'].apply(lambda x: 1␣
  ↪if x>=4 else 0)
```

3.7 Clean Data Visualization

Let us visualize the clean data and see if any interesting patterns are immediately spotted.

[44]:
```
covid.hist(figsize=(10,8))
plt.subplots_adjust(hspace=0.9, wspace=0.9)
```

```
plt.suptitle("Histograms of Numeric Variables after Cleaning", fontsize=14, y=0.
↪96)
plt.rcParams.update({'font.size': 10})
```



Observe:

- We have clean data, and the unknown data is simply encoded as 0.5
- Most distributions are certainly skewed: there are many patients without diseases than with any diseases
- The majority of the patients in this data set survived covid, so only a small proportion died
- Less than half of the admited patients were diagnosed with Covid

Let's also look at some descriptive statistics

[45]:
```
covid.describe()
```

[45]:

	USMER	MEDICAL_UNIT	INTUBED	PNEUMONIA	AGE	\
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	
mean	3.678058e-01	8.980565e+00	4.402074e-01	1.411816e-01	4.177507e+01	
std	4.822084e-01	3.723278e+00	2.058388e-01	3.426865e-01	1.687679e+01	

min	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	4.000000e+00	5.000000e-01	0.000000e+00	3.000000e+01
50%	0.000000e+00	1.200000e+01	5.000000e-01	0.000000e+00	4.000000e+01
75%	1.000000e+00	1.200000e+01	5.000000e-01	0.000000e+00	5.300000e+01
max	1.000000e+00	1.300000e+01	1.000000e+00	1.000000e+00	1.210000e+02

	PREGNANT	DIABETES	COPD	ASTHMA	INMSUPR	\
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	
mean	2.591741e-01	1.207906e-01	1.579620e-02	3.152993e-02	1.513673e-02	
std	2.574744e-01	3.246606e-01	1.217815e-01	1.727008e-01	1.187268e-01	
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
50%	5.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
75%	5.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
max	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	

	…	CARDIOVASCULAR	OBESITY	RENAL_CHRONIC	TOBACCO	\
count	…	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	
mean	…	2.127363e-02	1.538583e-01	1.946165e-02	8.200272e-02	
std	…	1.417311e-01	3.598099e-01	1.355220e-01	2.729664e-01	
min	…	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	…	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
50%	…	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
75%	…	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
max	…	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	

	CLASIFFICATION_FINAL	ICU	Lower_Risk	Higher_Risk	\
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	
mean	5.305653e+00	4.242653e-01	9.266223e-01	7.337768e-02	
std	1.881165e+00	2.004250e-01	2.607556e-01	2.607556e-01	
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	3.000000e+00	5.000000e-01	1.000000e+00	0.000000e+00	
50%	6.000000e+00	5.000000e-01	1.000000e+00	0.000000e+00	
75%	7.000000e+00	5.000000e-01	1.000000e+00	0.000000e+00	
max	7.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	

	COVID_POSITIVE	COVID_INCONCLUSIVE
count	1.048575e+06	1.048575e+06
mean	3.738207e-01	6.261793e-01
std	4.838171e-01	4.838171e-01
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	1.000000e+00
75%	1.000000e+00	1.000000e+00
max	1.000000e+00	1.000000e+00

[8 rows x 22 columns]

Observe:

- 75% of patients are under the age of 53

Let's also look at the categorical distributions

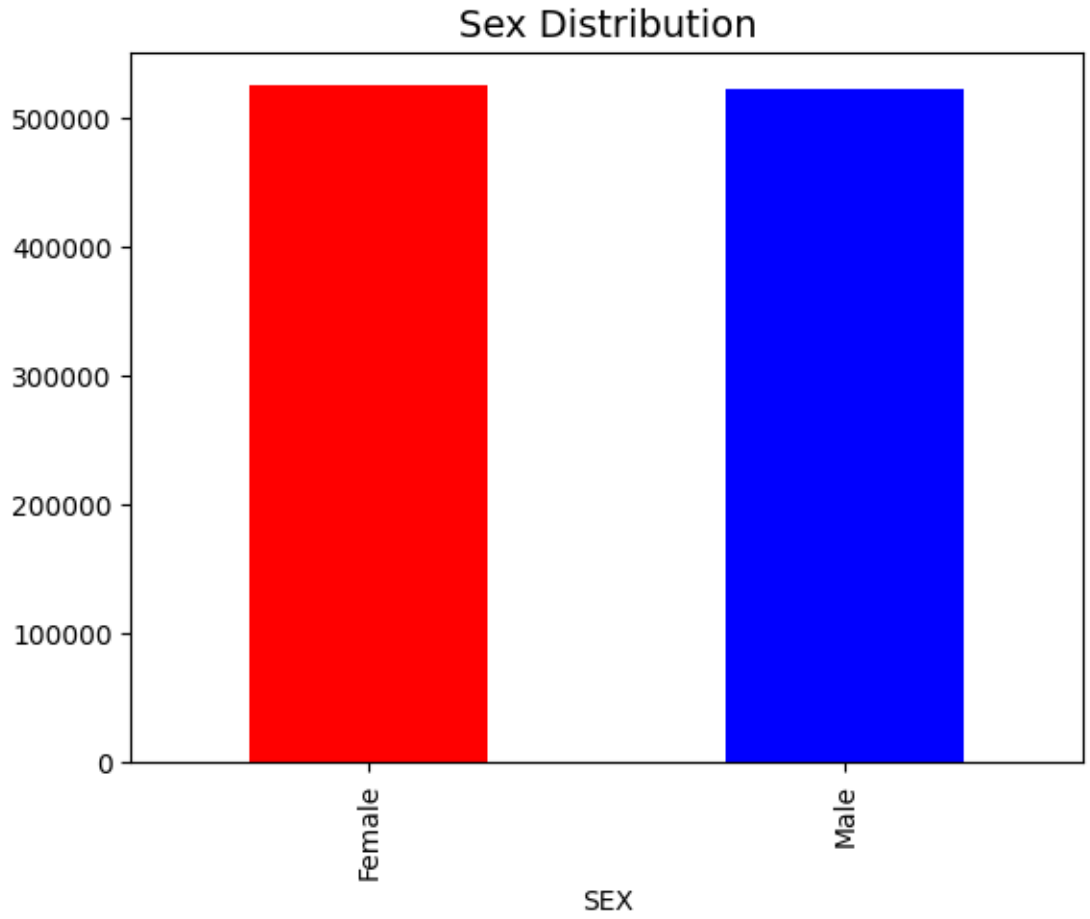
[46]:
```
cat_covid = covid.select_dtypes(include=['string'])
```

[47]:
```
cat_covid.columns
```

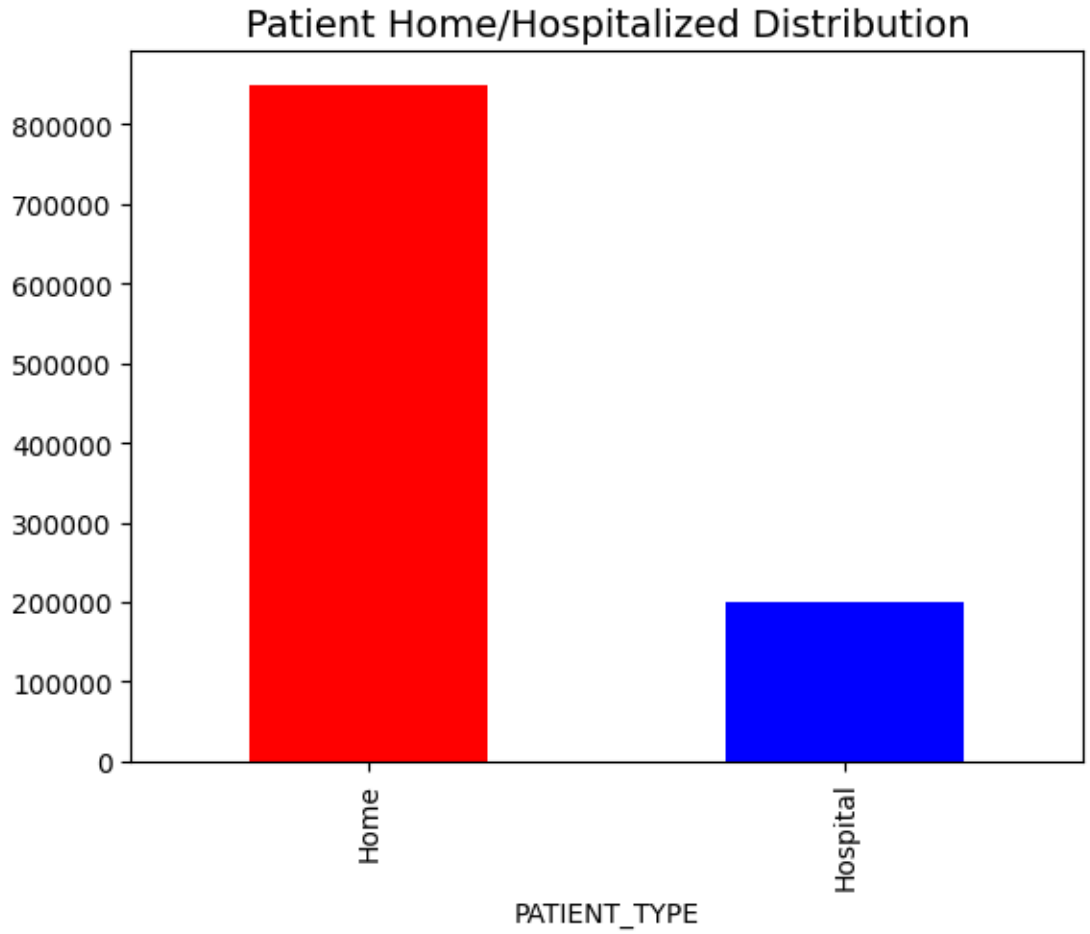
[47]:
```
Index(['SEX', 'PATIENT_TYPE'], dtype='object')
```

Observe: There are only two "String" columns, "SEX" and "PATIENT_TYPE"

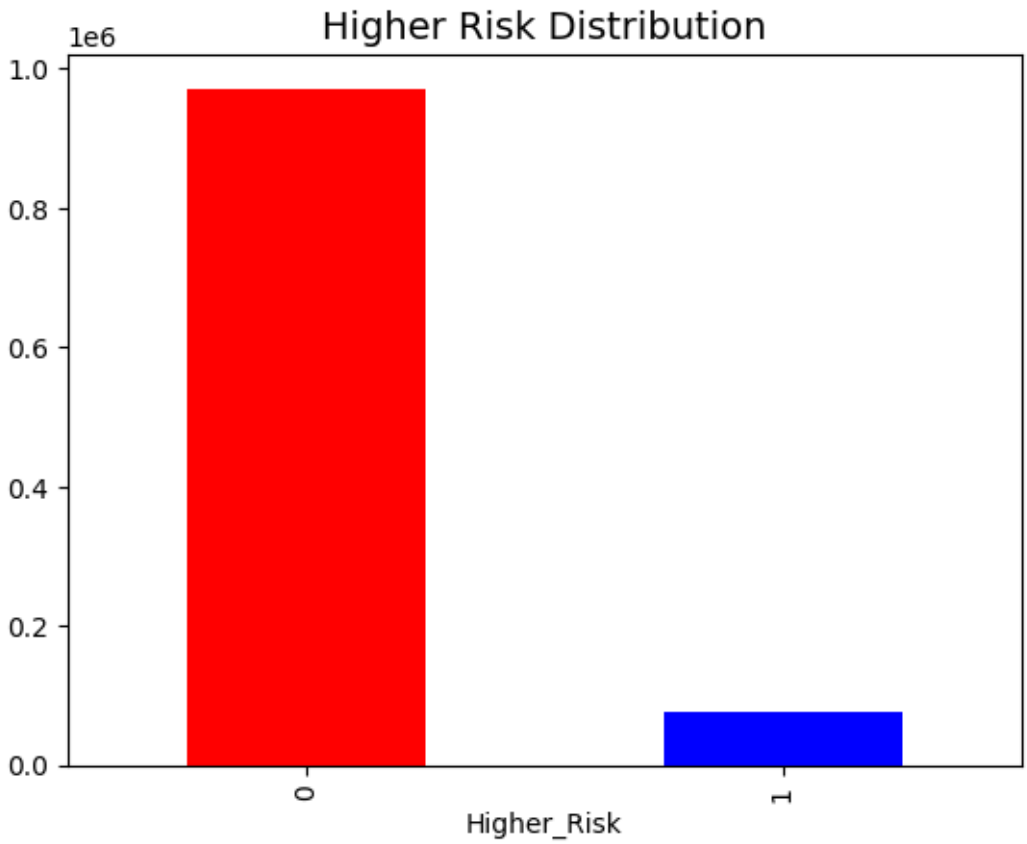
[48]:
```
covid["SEX"].value_counts().plot(kind='bar', color=['red', 'blue'])
plt.title("Sex Distribution", fontsize=14)
plt.show()
```



[49]:
```
covid["PATIENT_TYPE"].value_counts().plot(kind='bar', color=['red', 'blue'])
plt.title("Patient Home/Hospitalized Distribution", fontsize=14)
plt.show()
```



[50]:
```
covid["Higher_Risk"].value_counts().plot(kind='bar', color=['red', 'blue'])
plt.title("Higher Risk Distribution", fontsize=14)
plt.show()
```



[51]:
```
covid["Higher_Risk"].value_counts()[1]
```

[51]: 76942

[52]:
```
print("The percentage of patients at higher risk is", covid["Higher_Risk"].
↪value_counts()[1]/covid["Higher_Risk"].value_counts()[0]*100)
```

```
The percentage of patients at higher risk is 7.918833551351179
```

Note:

1. The majority of the patients survived COVID-19. Hence, when building a model that detects higher risk (the labels of higher risk are associated with patients that passed away) it is important to provide a high level of confidence in predictions of patients at high risk. What this means, is that the model should provide as few false negatives (FN) as possible. This is because a patient which is at high risk (positive) must receive urgent care to improve chances of survival. If this type of patient is classified as negative (not at high risk), then this could result in urgent care being delayed which could lead to loss of life, which is what this classification model tries to prevent. Hence, it is important that the model achieves high confidence in positive predictions, which is even more important that patients which are not at high risk being misclassified.

2. The classes “Higher_Risk”, “Lower_Risk” are imbalanced, in the sense that there is a small percentage of the total patients at high risk (close to 8%). This means that model accuracy is crucial to be above a 92%, anything less than that might as well classify patients at random
3. In selecting the model, it is very important to be able to interpret the predictions. As such, I will aim to have a high as possible recall for the model (TPR). The true positive rate gives the ratio of correctly idetified positives $TPR = TP/(TP + FN)$ where TP stands for true positives (correctly classified cases) and FN stands for false negatives (positives that are incorrectly) classified. The recall (TPR) is even more important than accuracy for this problem, as recall can be clearly interpreted as the probability that a given prediction of a patient at high risk is correct.

4 2. Split the Data

4.1 Undersampling

Before I do anything else, I will create a training set, a test set and a validation set. This is to avoid contamining the data by further exploration on the training set. However, one consideration must be kept in mind, which is the very high imbalanced classes to be predicted - as the ratio of imbalanced classes is close to 8:100, we are dealing with highly imbalanced classes. The downside of this approach is that the ML models may learn from a a non-representative collection of samples. Choosing the training set is indeed a very important step. The techniques we can use are:

1. Oversampling the minority class
2. Undersampling the majority class

For the following, I will choose to undersample the majority class (deleting examples from the majority class). The reason is that given enough datapoints in the majority class, it is entirely possible to sample a sufficient number of relevant examples from the training set. There is, however, a downside to undersampling : deleting valuable instances that can help our model learn.

The code for undersampling was adapted from the tutorial found at : https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

To help preserve the shape of the distributions in the undersamples datasets, I will use Near Miss, a nearest neighbours strategy : https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.NearMiss.html

[53]:
```
## I will use the imblearn library to balance the data
from imblearn.under_sampling import RandomUnderSampler
```

[54]:
```
##instantiating the RandomUnderSampler to delete examples from the majority␣
 ↪class
undersampler0 = RandomUnderSampler(sampling_strategy='majority', random_state=0)
undersampler1 = RandomUnderSampler(sampling_strategy='majority', random_state=1)
undersampler2 = RandomUnderSampler(sampling_strategy='majority', random_state=2)
undersampler3 = RandomUnderSampler(sampling_strategy='majority', random_state=3)
undersampler4 = RandomUnderSampler(sampling_strategy='majority', random_state=4)
```

Above, I created 5 different undersampler models. Each does the same things : deletes examples

from the majority class such that the classes for higher risk patients are balanced. Since, each undersampling gives different data, it is important to that the models that predict covid perform consistently on the dataset.

[55]:
```
##create the balanced data set
balanced0 = undersampler0.fit_resample(covid, covid['Higher_Risk'])
```

[56]:
```
balanced0 = pd.DataFrame(balanced0[0], columns=covid.columns)
```

Now that the original covid data was undersamples and the results stored in balanced0 as a dataframe, I will look at the distributions in this new dataset - this is important to convince ourselves that the method worked

[57]:
```
balanced0
```

[57]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	\
310388	1	4	Female	Home	9999-99-99	0.5	
756480	0	12	Female	Home	9999-99-99	0.5	
627261	0	12	Female	Home	9999-99-99	0.5	
312464	0	4	Male	Home	9999-99-99	0.5	
967580	0	12	Female	Home	9999-99-99	0.5	
…	…	…	…	…	…	…	
1047635	1	13	Male	Hospital	18/07/2020	1.0	
1047636	1	13	Male	Home	26/07/2020	0.5	
1047637	1	13	Female	Home	27/07/2020	0.5	
1047638	1	13	Male	Hospital	29/07/2020	1.0	
1047639	1	13	Male	Hospital	29/07/2020	0.0	

	PNEUMONIA	AGE	PREGNANT	DIABETES	…	CARDIOVASCULAR	OBESITY	\
310388	0.0	37.0	0.0	0.0	…	0.0	0.0	
756480	0.0	25.0	0.0	0.0	…	0.0	0.0	
627261	0.0	46.0	0.0	0.0	…	0.0	0.0	
312464	0.0	38.0	0.5	0.0	…	0.0	0.0	
967580	0.0	49.0	0.0	0.0	…	0.0	0.0	
…	…	…	…	…	…	…	…	
1047635	1.0	73.0	0.5	0.0	…	0.0	1.0	
1047636	1.0	82.0	0.5	0.0	…	0.0	0.0	
1047637	1.0	82.0	0.0	0.0	…	0.0	0.0	
1047638	1.0	80.0	0.5	0.0	…	0.0	1.0	
1047639	1.0	61.0	0.5	1.0	…	0.0	0.0	

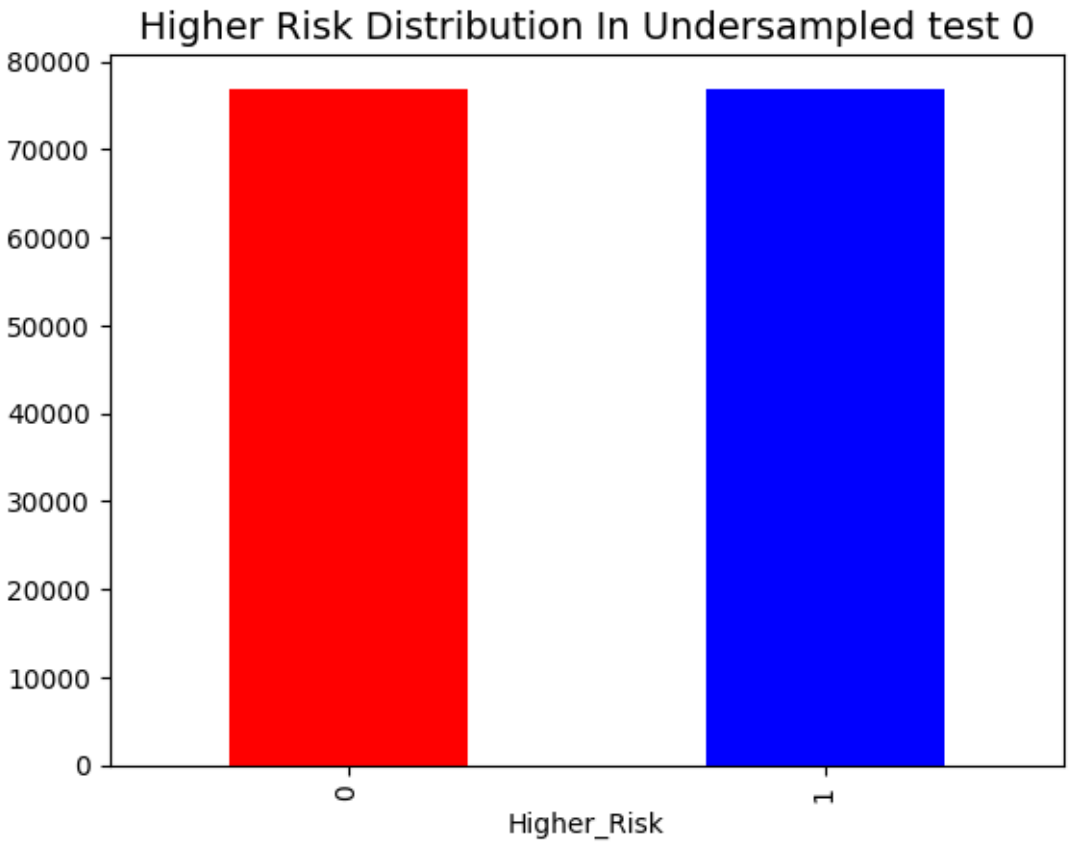
	RENAL_CHRONIC	TOBACCO	CLASIFFICATION_FINAL	ICU	Lower_Risk	\
310388	0.0	0.0	7	0.5	1	
756480	0.0	0.0	7	0.5	1	
627261	0.0	0.0	3	0.5	1	
312464	0.0	1.0	7	0.5	1	
967580	0.0	0.0	7	0.5	1	
…	…	…	…	…	…	

```
1047635           0.0      0.0                    3  0.0           0
1047636           0.0      0.0                    3  0.5           0
1047637           0.0      0.0                    3  0.5           0
1047638           0.0      0.0                    3  0.0           0
1047639           1.0      1.0                    7  0.0           0

         Higher_Risk  COVID_POSITIVE  COVID_INCONCLUSIVE
310388             0               0                   1
756480             0               0                   1
627261             0               1                   0
312464             0               0                   1
967580             0               0                   1
…                …               …                   …
1047635            1               1                   0
1047636            1               1                   0
1047637            1               1                   0
1047638            1               1                   0
1047639            1               0                   1

[153884 rows x 25 columns]
```

[58]:
```
print ("The number of examples in the balanced dataset is", balanced0.shape[0])
print ("The number of examples in the original dataset is", covid.shape[0])
```

```
The number of examples in the balanced dataset is 153884
The number of examples in the original dataset is 1048575
```

[59]:
```
balanced0["Higher_Risk"].value_counts().plot(kind='bar', color=['red', 'blue'])
plt.title("Higher Risk Distribution In Undersampled test 0", fontsize=14)
plt.show()
```



Quite clearly, there are now equal numbers of patients at higher/lower risk in the dataset. In a simillar fashion as the above, I will create all the other undersamples states and store them for later use

[60]:
```
## create all other balanced datasets using different random states
##each particular random state will create a different balanced dataset
##the random state is used to ensure reproducibility
balanced1 = undersampler1.fit_resample(covid, covid['Higher_Risk'])
balanced1 = pd.DataFrame(balanced1[0], columns=covid.columns)

balanced2 = undersampler2.fit_resample(covid, covid['Higher_Risk'])
balanced2 = pd.DataFrame(balanced2[0], columns=covid.columns)

balanced3 = undersampler3.fit_resample(covid, covid['Higher_Risk'])
balanced3 = pd.DataFrame(balanced3[0], columns=covid.columns)

balanced4 = undersampler4.fit_resample(covid, covid['Higher_Risk'])
balanced4 = pd.DataFrame(balanced4[0], columns=covid.columns)
```

4.2 Train/Test

Now we can create training and test sets from the new balanced samples. To do this, I will use Scikit-Learn as in the following.

[61]:
```
from sklearn.model_selection import train_test_split
```

[62]:
```
## each split  ensures the proportions 80-20 into a training set and the test␣
 ↪set
## random_state is a seed for the random number generator, ensures␣
 ↪reproducibility
## we do a stratified split to ensure enough number of samples from each class␣
 ↪in the training set
train0, test0 = train_test_split(balanced0, test_size=0.2, stratify␣
 ↪=balanced0['Higher_Risk'], random_state=1923)
train1, test1 = train_test_split(balanced1, test_size=0.2, stratify␣
 ↪=balanced1['Higher_Risk'], random_state=1923)
train2, test2 = train_test_split(balanced2, test_size=0.2, stratify␣
 ↪=balanced2['Higher_Risk'], random_state=1923)
train3, test3 = train_test_split(balanced3, test_size=0.2, stratify␣
 ↪=balanced3['Higher_Risk'], random_state=1923)
train4, test4 = train_test_split(balanced4, test_size=0.2, stratify␣
 ↪=balanced4['Higher_Risk'], random_state=1923)
```

Before proceeding further, let us look at the split resulting from Balanced0 and understand if there is sufficient data.

[63]:
```
train0.shape,  test0.shape
```

[63]:
```
((123107, 25), (30777, 25))
```

Observe: even if sample sizes are a small fraction of the entire covid dataset, there are 123000 data points reserved for training and about 30000 for testing. These dataset are not small, and machine learning models can be trained and tested on any particular dataset.

[64]:
```
train0.head()
```

[64]:

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	\
193064	0	4	Female	Home	9999-99-99	0.5	
20620	0	4	Male	Hospital	10/04/2020	0.0	
586380	1	12	Male	Home	9999-99-99	0.5	
351666	1	6	Male	Hospital	9999-99-99	0.0	
17764	1	3	Male	Home	9999-99-99	0.5	

	PNEUMONIA	AGE	PREGNANT	DIABETES	…	CARDIOVASCULAR	OBESITY	\
193064	0.0	4.0	0.0	0.0	…	0.0	0.0	
20620	1.0	55.0	0.5	1.0	…	0.0	0.0	
586380	0.0	27.0	0.5	0.0	…	0.0	0.0	
351666	1.0	52.0	0.5	0.0	…	0.0	0.0	

```
17764         0.0  46.0       0.5       0.0  …              0.0       0.0

        RENAL_CHRONIC  TOBACCO  CLASIFFICATION_FINAL  ICU  Lower_Risk  \
193064            0.0      0.0                     6  0.5           1
20620             1.0      0.0                     6  0.0           0
586380            0.0      0.0                     3  0.5           1
351666            0.0      0.0                     3  0.0           1
17764             0.0      0.0                     7  0.5           1

        Higher_Risk  COVID_POSITIVE  COVID_INCONCLUSIVE
193064            0               0                   1
20620             1               0                   1
586380            0               1                   0
351666            0               1                   0
17764             0               0                   1

[5 rows x 25 columns]
```

Now, the above subsamples can be used to train the machine learning models. They were carefully selected to ensure that the models do not overgeneralize for a disproportionate majority class. However, it would be useful to create a test/training based on the original dataset that will be used for hyper-parameter tuning. The reason is that a model that can perform as well as possible on hugely imbalanced classes should be able to perform very well on a subsample that contains balanced classes. Let us do the split as before.

[65]:
```
##split the original data (no undersampling) into train and test sets
train, test = train_test_split(covid, test_size=0.2, stratify␣
 ↪=covid['Higher_Risk'], random_state=1923)
```

Let us do all exploration on a copy of the covid data. The models will be tested on the training sets that are randomly sampled, and the ones that are consistently performant will be selected.

[66]:
```
covid_eda = covid.copy()
```

5 3. Explore the Data

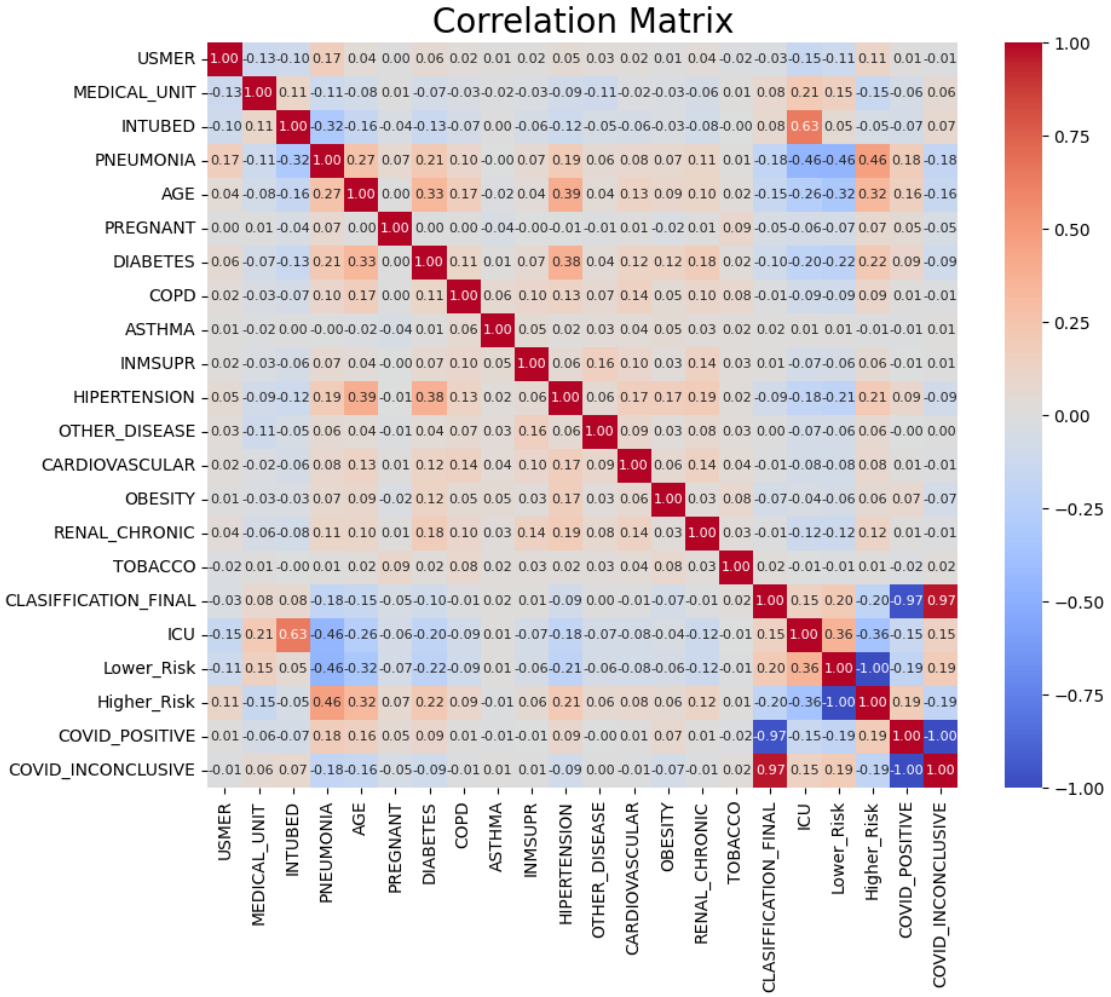
5.1 Looking for Correlations

[67]:
```
covid_eda.head()
```

[67]:
```
    USMER  MEDICAL_UNIT     SEX PATIENT_TYPE   DATE_DIED  INTUBED  PNEUMONIA  \
0       0             1  Female         Home  03/05/2020      0.5        1.0
1       0             1    Male         Home  03/06/2020      0.5        1.0
2       0             1    Male     Hospital  09/06/2020      1.0        0.0
3       0             1  Female         Home  12/06/2020      0.5        0.0
4       0             1    Male         Home  21/06/2020      0.5        0.0

    AGE  PREGNANT  DIABETES  …  CARDIOVASCULAR  OBESITY  RENAL_CHRONIC  \
```

```
0  65.0       0.0       0.0  …            0.0      0.0           0.0
1  72.0       0.5       0.0  …            0.0      1.0           1.0
2  55.0       0.5       1.0  …            0.0      0.0           0.0
3  53.0       0.0       0.0  …            0.0      0.0           0.0
4  68.0       0.5       1.0  …            0.0      0.0           0.0

   TOBACCO  CLASIFFICATION_FINAL  ICU  Lower_Risk  Higher_Risk  \
0      0.0                     3  0.5           0            1
1      0.0                     5  0.5           0            1
2      0.0                     3  0.0           0            1
3      0.0                     7  0.5           0            1
4      0.0                     3  0.5           0            1

   COVID_POSITIVE  COVID_INCONCLUSIVE
0               1                   0
1               0                   1
2               1                   0
3               0                   1
4               1                   0

[5 rows x 25 columns]
```

[68]:
```
covid_num = covid_eda.select_dtypes(include=[np.number])
corr_cov = covid_num.corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr_cov, annot=True, cmap='coolwarm',  fmt=".2f",␣
 ↪annot_kws={"size": 8})
plt.title("Correlation Matrix", fontsize=20)
plt.show()
```



Observe:

- Classification_Final and COVID_POSITIVE are almost identical 0.97% correlation; this suggests that there might be some outliers in CLASIFICATION_FINAL
- High correlation between ICU and INTUBED: This is expected also due to imputed values for uncertainity (missing data in one column perfectly predicts missing data in another columns)
- There is a high correlation between Higher_Risk and PNEUMONIA (46%) and Higher_risk and AGE (32) : this suggests older patients with pneumonia are at risk of death
- AGE and HIPERTENSION are also correlated
- The correlation between COVID_POSITIVE and Higher_risk is only 19%, which suggest that other factors might be at play
- PNEUMONIA is also a risk factor for being admitted into intensive care (46%) correlation
- AGE increases the probability of “DIABETES” and “HIPERTENSION”
- There is a 36% correlation of being admitted at ICU (Intensive Care Unit) and surviving covid : this suggests that receiving emergency care can definitely increase the chances of survival for patients with COVID-19

[69]:
```
##let's look at factors that are highly correlated with COVID_POSITIVE
corr_cov["COVID_POSITIVE"].sort_values(ascending=False)
```

[69]:
```
COVID_POSITIVE          1.000000
Higher_Risk             0.192564
PNEUMONIA               0.176604
AGE                     0.155362
DIABETES                0.094026
HIPERTENSION            0.087060
OBESITY                 0.071992
PREGNANT                0.050521
RENAL_CHRONIC           0.014522
USMER                   0.012075
CARDIOVASCULAR          0.011998
COPD                    0.009642
OTHER_DISEASE          -0.004058
INMSUPR                -0.007235
ASTHMA                 -0.014753
TOBACCO                -0.019348
MEDICAL_UNIT           -0.058648
INTUBED                -0.074522
ICU                    -0.145065
Lower_Risk             -0.192564
CLASIFFICATION_FINAL   -0.966963
COVID_INCONCLUSIVE     -1.000000
Name: COVID_POSITIVE, dtype: float64
```

Note: Age, as well as various diseases are more correlated with a patient being COVID positive

[70]:
```
##let's look at factors that are highly correlated with death
corr_cov["Higher_Risk"].sort_values(ascending=False)
```

[70]:
```
Higher_Risk             1.000000
PNEUMONIA               0.463481
AGE                     0.320374
DIABETES                0.217830
HIPERTENSION            0.205348
COVID_POSITIVE          0.192564
RENAL_CHRONIC           0.123091
USMER                   0.112671
COPD                    0.094935
CARDIOVASCULAR          0.081664
PREGNANT                0.070329
OTHER_DISEASE           0.062667
OBESITY                 0.059091
INMSUPR                 0.056399
TOBACCO                 0.008696
ASTHMA                 -0.012579
```

```
INTUBED                -0.053472
MEDICAL_UNIT           -0.149030
COVID_INCONCLUSIVE     -0.192564
CLASIFFICATION_FINAL   -0.196085
ICU                    -0.364282
Lower_Risk             -1.000000
Name: Higher_Risk, dtype: float64
```

Note: Pneumonia, Age and Diabetes are the most correlated features with death, whereas being COVID positive is the fifth correlated feature in this list

5.2 Experimenting with Transformations

It is clear that most columns are categorical, contains positives, negatives and missing values. To deal with this data, we can use KBinsDiscretizer that uses a Kmeans algorithm to bucketize the columns and visualize how close it is to our predictions.

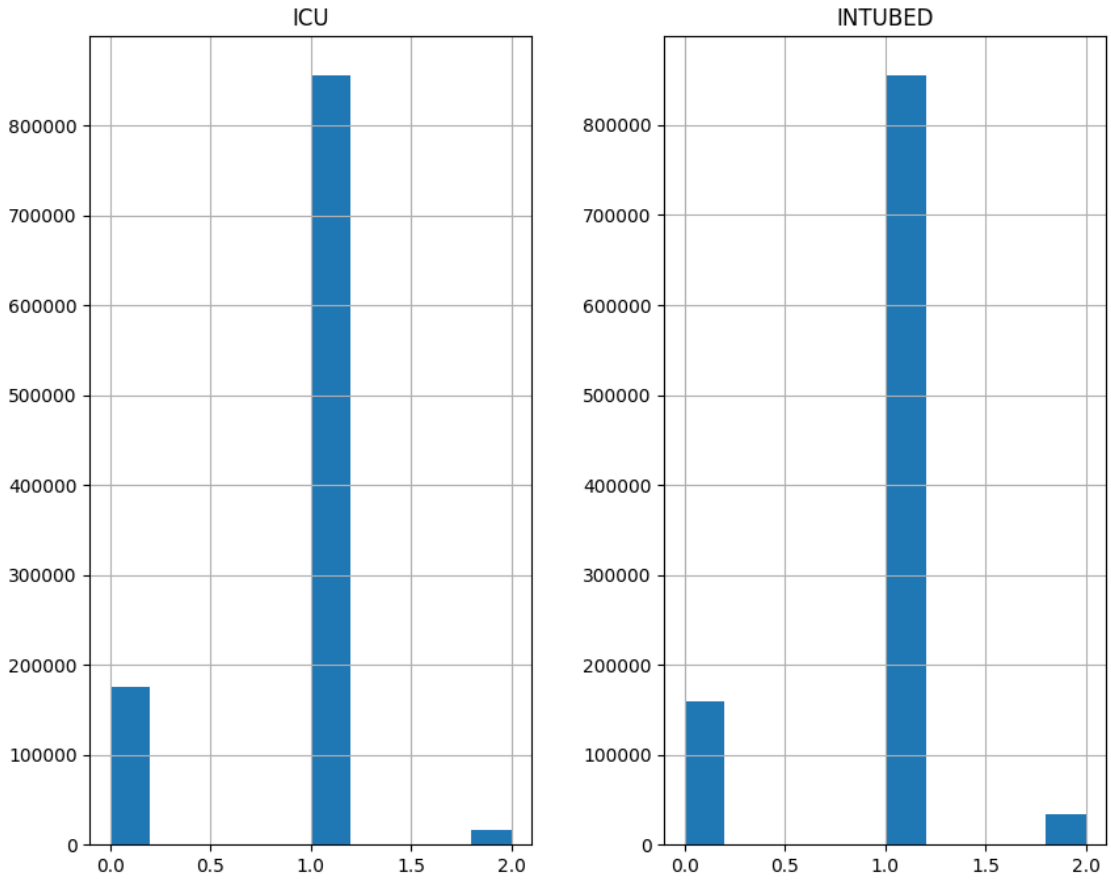
[71]:
```
##Let's deal with the categories that are already encoded
cat_num = covid_eda.select_dtypes(include=[np.number])
cat_num.drop(['MEDICAL_UNIT', 'AGE', 'CLASIFFICATION_FINAL'], axis=1,␣
 ↪inplace=True)
```

[72]:
```
[cat_num.columns]
```

[72]:
```
[Index(['USMER', 'INTUBED', 'PNEUMONIA', 'PREGNANT', 'DIABETES', 'COPD',
        'ASTHMA', 'INMSUPR', 'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR',
        'OBESITY', 'RENAL_CHRONIC', 'TOBACCO', 'ICU', 'Lower_Risk',
        'Higher_Risk', 'COVID_POSITIVE', 'COVID_INCONCLUSIVE'],
       dtype='object')]
```

[73]:
```
from sklearn.preprocessing import KBinsDiscretizer
dataframe_urgent = covid[['ICU', 'INTUBED']].copy() ##create a new dataframe␣
 ↪with only with numerical columns
##create an instance of the KBinsDiscretizer class with 3 bins and the kmeans␣
 ↪strategy;
##the encode argument is set to ordinal to label each bin with an integer
##the k-means strategies ensures that values in each bin have the same nearest␣
 ↪center of a 1D k-means cluster
discretizer_3bins = KBinsDiscretizer(n_bins=3, encode='ordinal',␣
 ↪strategy='kmeans')
##let's now transform the data and plot it to examine if the results is very␣
 ↪different from the manual approach
dataframe_buketized = discretizer_3bins.fit_transform(dataframe_urgent)
dataframe_buketized = pd.DataFrame(dataframe_buketized, columns=['ICU',␣
 ↪'INTUBED'])
##let us plot the histograms of the bucketized data
dataframe_buketized.hist(figsize=(10,8))
```

[73]:
```
array([[<Axes: title={'center': 'ICU'}>,
        <Axes: title={'center': 'INTUBED'}>]], dtype=object)
```



The Discretizer has simply replaced the values of the original distributions with ordinal numbers. Hence, using K-means with 3 clusters, it found very similar categories (the edges of the discretizer are given in the cell below)

[74]:
```
##this shows the edges of the bins
for i, edges in enumerate(discretizer_3bins.bin_edges_):
    print(f"{edges}")
```

```
[0.   0.25 0.75 1.  ]
[0.   0.25 0.75 1.  ]
```

An alternative treatment, is to transform these column types to “object” and then one-hot encode based on the original values while keeping in mind that “0.5” indicates missing values (remember that I used “0.5” to encode missing values for categorical one-hot encoded columns)

[75]:
```
for col in cat_num.columns:
    covid_eda[col] = covid_eda[col].astype('string')
```

[76]:
```
covid_eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 25 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   USMER                 1048575 non-null  string
 1   MEDICAL_UNIT          1048575 non-null  int64
 2   SEX                   1048575 non-null  string
 3   PATIENT_TYPE          1048575 non-null  string
 4   DATE_DIED             1048575 non-null  object
 5   INTUBED               1048575 non-null  string
 6   PNEUMONIA             1048575 non-null  string
 7   AGE                   1048575 non-null  float64
 8   PREGNANT              1048575 non-null  string
 9   DIABETES              1048575 non-null  string
 10  COPD                  1048575 non-null  string
 11  ASTHMA                1048575 non-null  string
 12  INMSUPR               1048575 non-null  string
 13  HIPERTENSION          1048575 non-null  string
 14  OTHER_DISEASE         1048575 non-null  string
 15  CARDIOVASCULAR        1048575 non-null  string
 16  OBESITY               1048575 non-null  string
 17  RENAL_CHRONIC         1048575 non-null  string
 18  TOBACCO               1048575 non-null  string
 19  CLASIFFICATION_FINAL  1048575 non-null  int64
 20  ICU                   1048575 non-null  string
 21  Lower_Risk            1048575 non-null  string
 22  Higher_Risk           1048575 non-null  string
 23  COVID_POSITIVE        1048575 non-null  string
 24  COVID_INCONCLUSIVE    1048575 non-null  string
dtypes: float64(1), int64(2), object(1), string(21)
memory usage: 200.0+ MB
```

[77]:
```
covid_eda.drop(['DATE_DIED'], axis=1, inplace=True)
```

[78]:
```
##let's create a dataframe containing all categories (string columns)
covid_eda_all_cat = covid_eda.select_dtypes(include=['string'])
```

[79]:
```
covid_eda_all_cat
```

[79]:

	USMER	SEX	PATIENT_TYPE	INTUBED	PNEUMONIA	PREGNANT	DIABETES	COPD	\
0	0	Female	Home	0.5	1.0	0.0	0.0	0.0	
1	0	Male	Home	0.5	1.0	0.5	0.0	0.0	
2	0	Male	Hospital	1.0	0.0	0.5	1.0	0.0	
3	0	Female	Home	0.5	0.0	0.0	0.0	0.0	
4	0	Male	Home	0.5	0.0	0.5	1.0	0.0	

```
…        …      …            …      …        …       …       …  …
1048570     0    Male          Home     0.5       0.0      0.5      0.0  0.0
1048571     1    Male      Hospital     0.0       0.0      0.5      0.0  0.0
1048572     0    Male          Home     0.5       0.0      0.5      0.0  0.0
1048573     0    Male          Home     0.5       0.0      0.5      0.0  0.0
1048574     0    Male          Home     0.5       0.0      0.5      0.0  0.0

         ASTHMA  INMSUPR  … OTHER_DISEASE  CARDIOVASCULAR  OBESITY  \
0           0.0      0.0  …           0.0             0.0      0.0
1           0.0      0.0  …           0.0             0.0      1.0
2           0.0      0.0  …           0.0             0.0      0.0
3           0.0      0.0  …           0.0             0.0      0.0
4           0.0      0.0  …           0.0             0.0      0.0
…          …      …   …             …               …      …
1048570     0.0      0.0  …           0.0             0.0      0.0
1048571     0.0      0.0  …           0.0             0.0      0.0
1048572     0.0      0.0  …           0.0             0.0      0.0
1048573     0.0      0.0  …           0.0             0.0      0.0
1048574     0.0      0.0  …           0.0             0.0      0.0

         RENAL_CHRONIC  TOBACCO   ICU  Lower_Risk  Higher_Risk  COVID_POSITIVE  \
0                  0.0      0.0  0.5           0            1               1
1                  1.0      0.0  0.5           0            1               0
2                  0.0      0.0  0.0           0            1               1
3                  0.0      0.0  0.5           0            1               0
4                  0.0      0.0  0.5           0            1               1
…                 …      …   …          …           …             …
1048570            0.0      0.0  0.5           1            0               0
1048571            0.0      0.0  0.0           1            0               0
1048572            0.0      0.0  0.5           1            0               0
1048573            0.0      0.0  0.5           1            0               0
1048574            0.0      0.0  0.5           1            0               0

         COVID_INCONCLUSIVE
0                         0
1                         1
2                         0
3                         1
4                         0
…                       …
1048570                   1
1048571                   1
1048572                   1
1048573                   1
1048574                   1

[1048575 rows x 21 columns]
```

Now, let us one-hot encode these features. To do this, I will built a pipeline via the tools provided with Scikit-Learn.

[80]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
```

[81]:
```
##pipeline that one-hot encodes the categorical features
one_hot = make_pipeline(OneHotEncoder(sparse=False, handle_unknown='ignore'))
```

[82]:
```
##let us apply the pipeline to the categorical features
cat_num_onehot = one_hot.fit_transform(covid_eda_all_cat)
```

```
/opt/homebrew/lib/python3.11/site-
packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[83]:
```
##this returns a numpy array with one-hot encoded features
cat_num_onehot
```

[83]:
```
array([[1., 0., 1., …, 1., 1., 0.],
       [1., 0., 0., …, 0., 0., 1.],
       [1., 0., 0., …, 1., 1., 0.],
       …,
       [1., 0., 0., …, 0., 0., 1.],
       [1., 0., 0., …, 0., 0., 1.],
       [1., 0., 0., …, 0., 0., 1.]])
```

[84]:
```
##we can put the one-hot encoded features in a dataframe
cat_num_onehot = pd.DataFrame(cat_num_onehot, columns = one_hot.
 ↪get_feature_names_out(), index=covid_eda_all_cat.index)
```

[85]:
```
##we have easilty one-hot encoded the categorical features and put them in a␣
 ↪dataframe while preserving the index
cat_num_onehot
```

[85]:

	USMER_0	USMER_1	SEX_Female	SEX_Male	PATIENT_TYPE_Home	\
0	1.0	0.0	1.0	0.0	1.0	
1	1.0	0.0	0.0	1.0	1.0	
2	1.0	0.0	0.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	1.0	
4	1.0	0.0	0.0	1.0	1.0	
…	…	…	…	…	…	
1048570	1.0	0.0	0.0	1.0	1.0	
1048571	0.0	1.0	0.0	1.0	0.0	
1048572	1.0	0.0	0.0	1.0	1.0	
1048573	1.0	0.0	0.0	1.0	1.0	

```
1048574      1.0      0.0         0.0       1.0                1.0

         PATIENT_TYPE_Hospital  INTUBED_0.0  INTUBED_0.5  INTUBED_1.0  \
0                          0.0          0.0          1.0          0.0
1                          0.0          0.0          1.0          0.0
2                          1.0          0.0          0.0          1.0
3                          0.0          0.0          1.0          0.0
4                          0.0          0.0          1.0          0.0
…                          …            …            …            …
1048570                    0.0          0.0          1.0          0.0
1048571                    1.0          1.0          0.0          0.0
1048572                    0.0          0.0          1.0          0.0
1048573                    0.0          0.0          1.0          0.0
1048574                    0.0          0.0          1.0          0.0

         PNEUMONIA_0.0  …  ICU_0.5  ICU_1.0  Lower_Risk_0  Lower_Risk_1  \
0                  0.0  …      1.0      0.0           1.0           0.0
1                  0.0  …      1.0      0.0           1.0           0.0
2                  1.0  …      0.0      0.0           1.0           0.0
3                  1.0  …      1.0      0.0           1.0           0.0
4                  1.0  …      1.0      0.0           1.0           0.0
…                  …    …      …        …             …             …
1048570            1.0  …      1.0      0.0           0.0           1.0
1048571            1.0  …      0.0      0.0           0.0           1.0
1048572            1.0  …      1.0      0.0           0.0           1.0
1048573            1.0  …      1.0      0.0           0.0           1.0
1048574            1.0  …      1.0      0.0           0.0           1.0

         Higher_Risk_0  Higher_Risk_1  COVID_POSITIVE_0  COVID_POSITIVE_1  \
0                  0.0            1.0               0.0               1.0
1                  0.0            1.0               1.0               0.0
2                  0.0            1.0               0.0               1.0
3                  0.0            1.0               1.0               0.0
4                  0.0            1.0               0.0               1.0
…                  …              …                 …                 …
1048570            1.0            0.0               1.0               0.0
1048571            1.0            0.0               1.0               0.0
1048572            1.0            0.0               1.0               0.0
1048573            1.0            0.0               1.0               0.0
1048574            1.0            0.0               1.0               0.0

         COVID_INCONCLUSIVE_0  COVID_INCONCLUSIVE_1
0                         1.0                   0.0
1                         0.0                   1.0
2                         1.0                   0.0
3                         0.0                   1.0
4                         1.0                   0.0
```

```
...                    ...                    ...
1048570                0.0                    1.0
1048571                0.0                    1.0
1048572                0.0                    1.0
1048573                0.0                    1.0
1048574                0.0                    1.0

[1048575 rows x 56 columns]
```

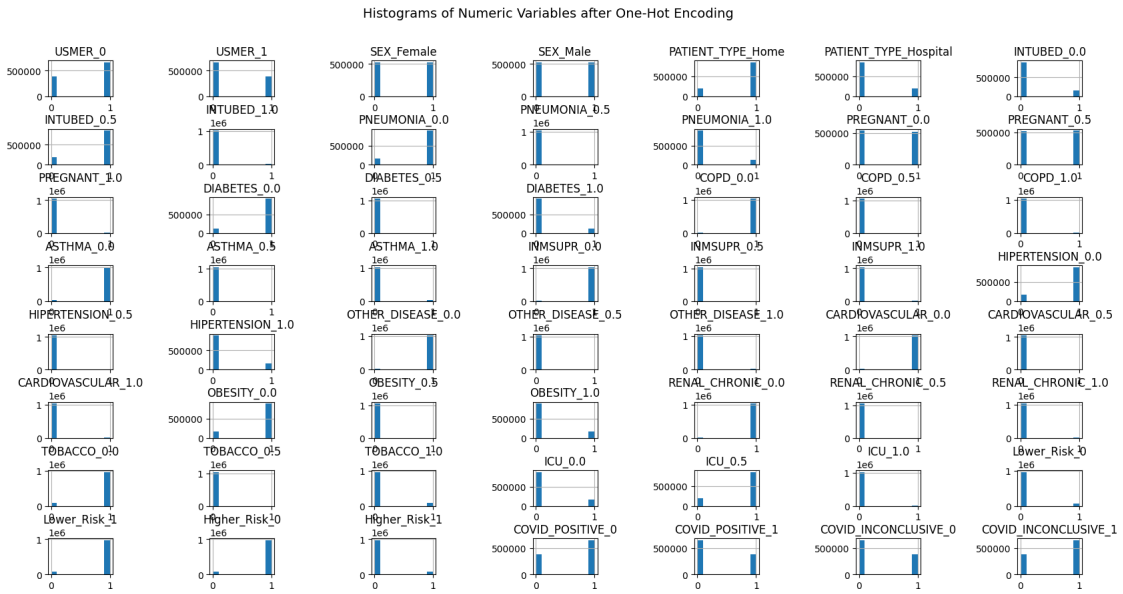
[86]:
```
cat_num_onehot.shape
```

[86]:
```
(1048575, 56)
```

Observe:

-The number of categories greatly increased due to having one-hot encoded features

[87]:
```
##let us visualize the new features
cat_num_onehot.hist(figsize=(20,10))
plt.subplots_adjust(hspace=0.9, wspace=1.5)
plt.suptitle("Histograms of Numeric Variables after One-Hot Encoding",␣
 ↪fontsize=14, y=0.96)
plt.rcParams.update({'font.size': 8})
```



Observe:

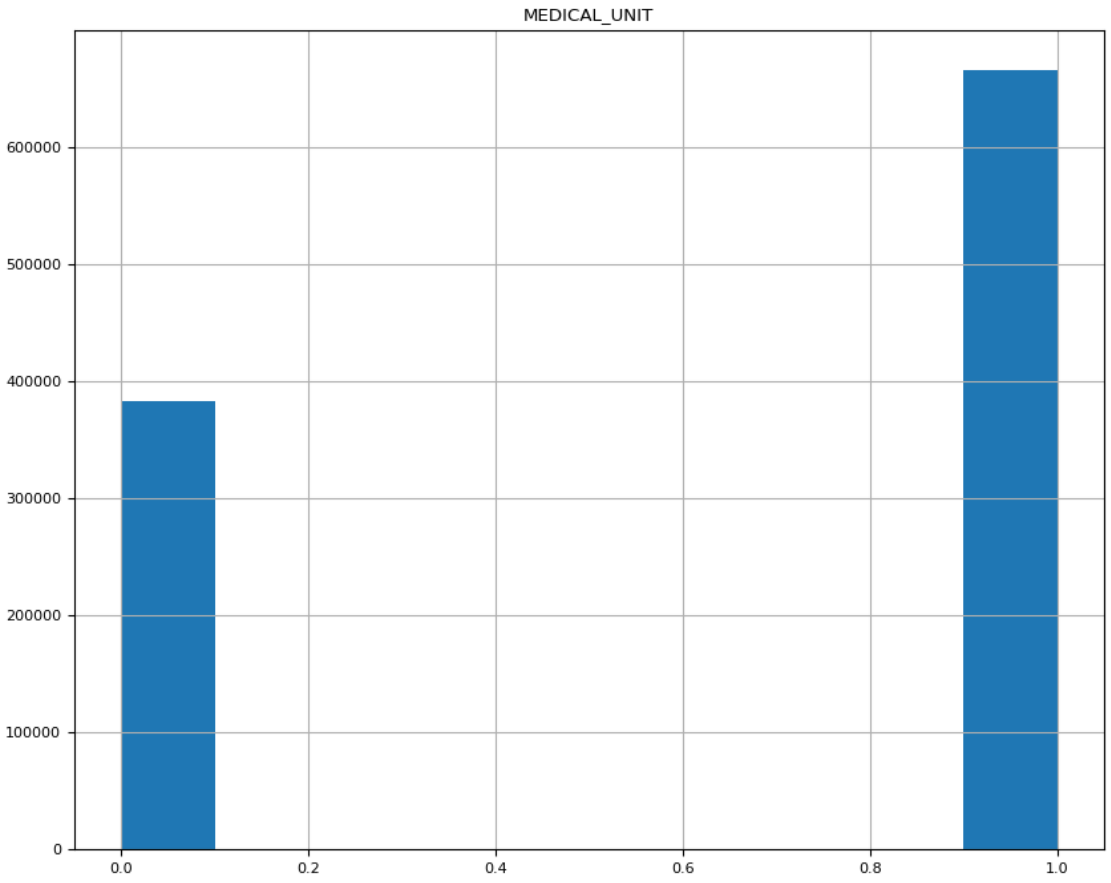
- all the new columns have the feature and the characteristic they one-hot encode: 0.0 is a negative, 1.0 is a positive and 0.5 is a missing value

- for the “SEX” and “PATIENT_TYPE” as well as for the COVID results we have one-hot encoded categories
- all these can be fed to the machine learning model

One category that we did not treat is the medical unit. We can try to find clusters with KMEans and Bucketize

[88]:
```
dataframe_medunit = covid_eda[['MEDICAL_UNIT']].copy()
##let's create a discretizer with 2 bins and the kmeans strategy;
discretizer_2bins = KBinsDiscretizer(n_bins=2, encode='ordinal',␣
 ↪strategy='kmeans')
##let's now transform the data and plot it to examine if the results is very␣
 ↪different from the manual approach
dataframe_buketized_med = discretizer_2bins.fit_transform(dataframe_medunit)
dataframe_buketized_med = pd.DataFrame(dataframe_buketized_med,␣
 ↪columns=['MEDICAL_UNIT'])
##let us plot the histograms of the bucketized data
dataframe_buketized_med.hist(figsize=(10,8))
```

[88]:
```
array([[<Axes: title={'center': 'MEDICAL_UNIT'}>]], dtype=object)
```



[89]:
```
##let's look at the edges of the bins
for i, edges in enumerate(discretizer_2bins.bin_edges_):
    print(f"{edges}")
```

```
[ 1.          7.96059277 13.        ]
```

It seems that the Discretizer has grouped the instances between 1 and 8, and 8 and 13. This seems consistent with the previous distribution. As there is not information on what the numbers associated with the medical units mean, everything is grouped under two clusters, showing "Type I" Medical Unit and "Type II" Medical Unit

6 4. Pre-process data for Machine Learning Algorithms

- The previous transformations can all be put into pipelines
- I will build a pipeline that contains the one-hot encoding transformations, as well as scalling
- I will then apply this pipeline to data : training, test and validation
- With the help of an evaluation metric, I will then test various ML models from Scikit-Learn

6.1 Built a Pre-Processing Pipeline

[90]:
```
##column transformers to organize the pipelines
from sklearn.compose import ColumnTransformer
##make_pipeline to create a pipeline based on Function transformers
from sklearn.pipeline import make_pipeline
##StandardScaler to scale the numerical features
from sklearn.preprocessing import StandardScaler
##and function transformer to apply the discretizer
from sklearn.preprocessing import FunctionTransformer
```

[91]:
```
##displays transformations
from sklearn import set_config

set_config(display='diagram')
```

[92]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class CustomTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        ## Apply your transformation
        return X.astype(str)

    def get_feature_names_out(self, input_features=None):
        ## Implement this method to return the feature names
        return input_features
```

[93]:
```
##this creates a pipeline that only does one-hot encoding
cat_pipeline_default = make_pipeline(
                CustomTransformer(),
                OneHotEncoder(handle_unknown='ignore')
                )

##this creates a pipeline that only does one-hot encoding
cat_pipeline = make_pipeline(
                OneHotEncoder(handle_unknown='ignore')
                )
```

[94]:
```
##Now I am creating two pipelines for discretizing, and one-hot-encoding.

bucket_pipeline_1 = make_pipeline(
    KBinsDiscretizer(n_bins=2, encode='ordinal', strategy='kmeans',␣
 ↪random_state=1923),
     OneHotEncoder(handle_unknown='ignore'))
```

[95]:
```
## now for standard scaling
num_pipeline = make_pipeline(
    StandardScaler()
)
```

Now let us define the pre-processing pipeline, that allows the transformations to be imputed

[96]:
```
preprocessing = ColumnTransformer([

        ('bucket_pipeline_1', bucket_pipeline_1, ['MEDICAL_UNIT']),
        ('categorical', cat_pipeline_default, ['USMER', 'INTUBED', 'PNEUMONIA',␣
 ↪'PREGNANT', 'DIABETES', 'COPD',
        'ASTHMA', 'INMSUPR', 'HIPERTENSION', 'OTHER_DISEASE', 'CARDIOVASCULAR',
        'OBESITY', 'RENAL_CHRONIC', 'TOBACCO', 'ICU',
        'COVID_POSITIVE', 'COVID_INCONCLUSIVE']),
        ('numerical', num_pipeline, ['AGE']),
        ('categorical simple', cat_pipeline, ['SEX', 'PATIENT_TYPE']),
         ('drop_age', 'drop', ['CLASIFFICATION_FINAL', 'DATE_DIED',␣
 ↪'Higher_Risk', 'Lower_Risk']),
    ], remainder='passthrough')
```

Let us test it on a copy of the data

[97]:
```
covid_prep = covid.copy()
```

[98]:
```
covid_pip_prep = preprocessing.fit_transform(covid_prep)
```

[99]:
```
##after pre-rpocessing, we have a numpy array. Let us put it in a dataframe
covid_pip_prep =pd.DataFrame(covid_pip_prep, columns=preprocessing.
 ↪get_feature_names_out(), index=covid_prep.index)
```

[100]:
```
covid_pip_prep
```

[100]:

```
         bucket_pipeline_1__MEDICAL_UNIT_0.0  \
0                                        1.0
1                                        1.0
2                                        1.0
3                                        1.0
4                                        1.0
…                                        …
1048570                                  0.0
1048571                                  0.0
1048572                                  0.0
1048573                                  0.0
1048574                                  0.0

         bucket_pipeline_1__MEDICAL_UNIT_1.0  categorical__USMER_0  \
0                                        0.0                   1.0
1                                        0.0                   1.0
2                                        0.0                   1.0
3                                        0.0                   1.0
4                                        0.0                   1.0
…                                        …                     …
1048570                                  1.0                   1.0
1048571                                  1.0                   0.0
1048572                                  1.0                   1.0
1048573                                  1.0                   1.0
1048574                                  1.0                   1.0

         categorical__USMER_1  categorical__INTUBED_0.0  \
0                         0.0                       0.0
1                         0.0                       0.0
2                         0.0                       0.0
3                         0.0                       0.0
4                         0.0                       0.0
…                         …                         …
1048570                   0.0                       0.0
1048571                   1.0                       1.0
1048572                   0.0                       0.0
1048573                   0.0                       0.0
1048574                   0.0                       0.0

         categorical__INTUBED_0.5  categorical__INTUBED_1.0  \
0                             1.0                       0.0
1                             1.0                       0.0
2                             0.0                       1.0
3                             1.0                       0.0
4                             1.0                       0.0
```

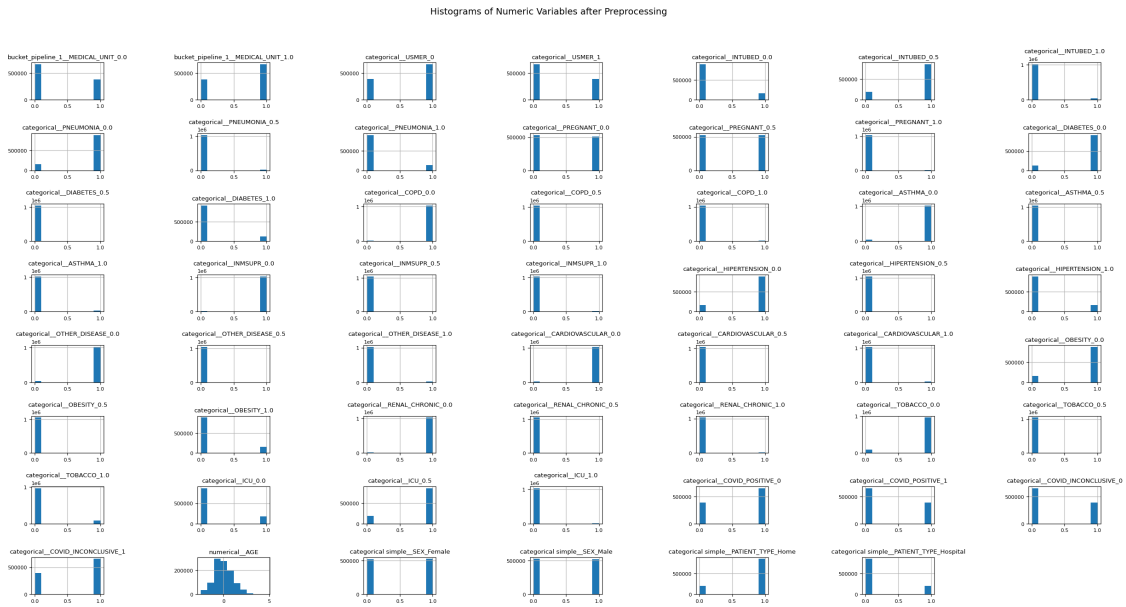
```
…                              …                              …
1048570                      1.0                            0.0
1048571                      0.0                            0.0
1048572                      1.0                            0.0
1048573                      1.0                            0.0
1048574                      1.0                            0.0

         categorical__PNEUMONIA_0.0  categorical__PNEUMONIA_0.5  \
0                               0.0                         0.0
1                               0.0                         0.0
2                               1.0                         0.0
3                               1.0                         0.0
4                               1.0                         0.0
…                               …                           …
1048570                         1.0                         0.0
1048571                         1.0                         0.0
1048572                         1.0                         0.0
1048573                         1.0                         0.0
1048574                         1.0                         0.0

         categorical__PNEUMONIA_1.0  …  categorical__ICU_1.0  \
0                               1.0  …                   0.0
1                               1.0  …                   0.0
2                               0.0  …                   0.0
3                               0.0  …                   0.0
4                               0.0  …                   0.0
…                               …  …                   …
1048570                         0.0  …                   0.0
1048571                         0.0  …                   0.0
1048572                         0.0  …                   0.0
1048573                         0.0  …                   0.0
1048574                         0.0  …                   0.0

         categorical__COVID_POSITIVE_0  categorical__COVID_POSITIVE_1  \
0                                  0.0                            1.0
1                                  1.0                            0.0
2                                  0.0                            1.0
3                                  1.0                            0.0
4                                  0.0                            1.0
…                                  …                              …
1048570                            1.0                            0.0
1048571                            1.0                            0.0
1048572                            1.0                            0.0
1048573                            1.0                            0.0
1048574                            1.0                            0.0

         categorical__COVID_INCONCLUSIVE_0  categorical__COVID_INCONCLUSIVE_1  \
```

```
0                                    1.0                                    0.0
1                                    0.0                                    1.0
2                                    1.0                                    0.0
3                                    0.0                                    1.0
4                                    1.0                                    0.0
…                                    …                                      …
1048570                              0.0                                    1.0
1048571                              0.0                                    1.0
1048572                              0.0                                    1.0
1048573                              0.0                                    1.0
1048574                              0.0                                    1.0

         numerical__AGE  categorical simple__SEX_Female  \
0              1.376147                             1.0
1              1.790918                             0.0
2              0.783617                             0.0
3              0.665111                             1.0
4              1.553906                             0.0
…                 …                                 …
1048570       -0.105178                             0.0
1048571        0.546605                             0.0
1048572        0.783617                             0.0
1048573       -0.816214                             0.0
1048574        0.605858                             0.0

         categorical simple__SEX_Male  categorical simple__PATIENT_TYPE_Home  \
0                                 0.0                                    1.0
1                                 1.0                                    1.0
2                                 1.0                                    0.0
3                                 0.0                                    1.0
4                                 1.0                                    1.0
…                                 …                                      …
1048570                           1.0                                    1.0
1048571                           1.0                                    0.0
1048572                           1.0                                    1.0
1048573                           1.0                                    1.0
1048574                           1.0                                    1.0

         categorical simple__PATIENT_TYPE_Hospital
0                                              0.0
1                                              0.0
2                                              1.0
3                                              0.0
4                                              0.0
…                                              …
1048570                                        0.0
1048571                                        1.0
```

```
1048572                                    0.0
1048573                                    0.0
1048574                                    0.0

[1048575 rows x 55 columns]
```

[101]:
```
##let's look at the histograms
plt.figure(figsize=(20,10))
covid_pip_prep.hist(figsize=(30,15))
plt.subplots_adjust(hspace=0.9, wspace=1.3)
plt.suptitle("Histograms of Numeric Variables after Preprocessing",␣
 ↪fontsize=14, y=0.96)
plt.rcParams.update({'font.size': 10})
plt.show()
```

```
<Figure size 2000x1000 with 0 Axes>
```



Great! Before proceeding any further, let us define an evaluation metric

6.2 Evaluation Metric

[102]:
```
##let us import the metrics used for the evaluation function. I will make use␣
 ↪of Scikit-Learn for this
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score,␣
 ↪f1_score, precision_score, recall_score, accuracy_score
```

```
## Evaluation function that defines the metrics to be used for evaluating the␣
 ↪models
##this function assumes that the model has already been fitted to the training␣
 ↪data
def evaluation_metric(model, xTrain, yTrain, xTest, yTest):

    ##predict the labels on the training data
    yTrain_pred = model.predict(xTrain)

    ##predict the labels on the test data
    yTest_pred = model.predict(xTest)


    ##now I can calculate cross validation scores for the prediction on the␣
 ↪test set and training set

    ##cross validation scores for the training set. cv=5 means 5-fold cross␣
 ↪validation ( there are 5 folds used for cross validation)
    scores_train = cross_val_score(model, xTrain, yTrain, cv=5)
    mean_cv_train = scores_train.mean()
    print("-"*100)
    print("The mean cross validation score on the training set is:",␣
 ↪mean_cv_train)


    ##cross validation scores for the test set
    scores_test = cross_val_score(model, xTest, yTest, cv=5)
    mean_cv_test = scores_test.mean()
    print("-"*100)
    print("The mean cross validation score on the test set is:", mean_cv_test)


    ##overfitting/underfitting measure
    print("-"*100)
    print("The difference between the mean cross validation scores on the␣
 ↪training and test data:", mean_cv_train - mean_cv_test)


    ##fpr is the false positive rate, tpr is the true positive rate
    ##The roc_curve method calculates the aforemntioned rates for different␣
 ↪thresholds and plot the results based on the test labels and the predicted␣
 ↪labels
    ## We first must calculate the probabilities of the positive class
    yTest_prob = model.predict_proba(xTest)[:,1]
    fpr, tpr, thresholds = roc_curve(yTest, yTest_prob)
```

```
   ## the roc_curve method requires the second argument to be either␣
↪probability estimates of the positive class, confidence values, or␣
↪non-thresholded measure of decisions
   ## the roc_curve below used the probability estimates of the positive class␣
↪as given by the predict_proba method (yTest_prob)
   plt.figure(figsize=(8, 6))
   plt.plot(fpr, tpr, linewidth=2, label="ROC curve")
   plt.plot([0, 1], [0, 1], "k--", label="Random classifier's ROC curve")
   plt.xlabel("False positive rate")
   plt.ylabel("True positive rate")
   plt.title("ROC curve")
   plt.legend()
   plt.show()


   ##the AUC score is the area under the ROC curve
   auc_score = roc_auc_score(yTest, yTest_prob)
   print("-"*100)
   print("The AUC score is:", auc_score)

   ##accuracy score of predictions
   acc_score = accuracy_score(yTest, yTest_pred)
   print("-"*100)
   print("The accuracy score is:", acc_score)


   ##true negative rate
   print("-"*100)
   tn, fp, fn, tp = confusion_matrix(yTest, yTest_pred).ravel()
   tnr = tn / (tn+fp)
   print("The true negative rate (TNR) is:", tnr)

   ##true positive rate
   print("-"*100)
   tn, fp, fn, tp = confusion_matrix(yTest, yTest_pred).ravel()
   tpr = tp / (tp+fn)
   print("The true positive rate (TPR) is:", tpr)


   ##confusion matrix
   print("-"*100)
   cm = confusion_matrix (yTest_pred, yTest)
   ##now let us put it into a dataframe
   cm_df = pd.DataFrame(cm, columns=["Predicted negative", "Predicted␣
↪positive"], index=["Actual negative", "Actual positive"])
   ##plot the confusion matrix via seaborn
   sns.heatmap(cm_df, annot=True, fmt="d")
```

```
    plt.title("Confusion matrix")
    plt.show()
```

Notes on the Evaluation Metric:

1. The cross-validation scores on the training set and on the test set are calculated to see if the model overfits or underfits. If there is overfit, the model *learns more* on the training set than on the test set, which means that the function it creates is less reliable on new data. Hence, overfit can be detected by a higher cross validation score on the training set than on the test set. If the opposite is true (there is a lower score on the training set) there is a degree of underfit.
2. The AUC is the area under the ROC curve, a graph showing the dependence of the true positive rate on the true negative rate for all possible threholds. A classifier that distinguishes perfectly has an AUC of 1, while a classifier that is truly random has an AUC of 0.5 (interpreted as 50% probability of being any of the classes). For this project, AUC must be as high as possible
3. The TPR (true positive rate) is calculated from the values of the confusion matrix as $TP/(TP + FN)$ where TP are correct prediction of positive cases (True Positives) and FN are misclassifed cases of positive (high risk) patients (False Negatives). It is crucial for this project to achieve a high TPR as this is the probability of a patient classified as high risk to be really at high risk.
4. Accuracy represents the percentage of correct predictions. Accuracy should be as high as possible and consistent for different training data.

6.3 Create New Variables for the Data

Let us transform the training test data, such that they are processed and ready for the machine learning models. As a modelling approach, I am interested in predicting the 1's (positive High_risk) cases in the High_Risk column. The High_Risk column is already one-hot encoded, and there are five different undersamples sets where each undersampled set contains most patients that died (high_risk) and a balanced number of patients that survived (lower_risk). Hence, I create 5 different training and test sets such as that each ML model can be tested thoroughly.

[103]:
```
##training data for the first subsample
xTrain0 = preprocessing.fit_transform(train0)
y0 = train0["Higher_Risk"]
X0= pd.DataFrame(xTrain0, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train0.index)
xTest0 = preprocessing.fit_transform(test0)
yy0 = test0["Higher_Risk"]
XX0 = pd.DataFrame(xTest0, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test0.index)
```

As in the above, let us create new training variables for the sampled data and subsequently test them.

[104]:
```
xTrain1 = preprocessing.fit_transform(train1)
y1 = train1["Higher_Risk"]
```

```
X1 = pd.DataFrame(xTrain1, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train1.index)
xTest1 = preprocessing.fit_transform(test1)
yy1 = test1["Higher_Risk"]
XX1 = pd.DataFrame(xTest1, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test1.index)
```

[105]:
```
xTrain2 = preprocessing.fit_transform(train2)
y2 = train2["Higher_Risk"]
X2 = pd.DataFrame(xTrain2, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train2.index)
xTest2 = preprocessing.fit_transform(test2)
yy2 = test2["Higher_Risk"]
XX2 = pd.DataFrame(xTest2, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test2.index)
```

[106]:
```
xTrain3 = preprocessing.fit_transform(train3)
y3 = train3["Higher_Risk"]
X3 = pd.DataFrame(xTrain3, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train3.index)
xTest3 = preprocessing.fit_transform(test3)
yy3 = test1["Higher_Risk"]
XX3 = pd.DataFrame(xTest3, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test3.index)
```

[107]:
```
xTrain4 = preprocessing.fit_transform(train4)
y4 = train4["Higher_Risk"]
X4 = pd.DataFrame(xTrain4, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train4.index)
xTest4 = preprocessing.fit_transform(test4)
yy4 = test4["Higher_Risk"]
XX4 = pd.DataFrame(xTest4, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test4.index)
```

Finally, let us pre-process the training and test data, now sampled on the entirety of the dataset. I will use this when I do the final test for the best performing model.

[108]:
```
xTrain = preprocessing.fit_transform(train)
y = train["Higher_Risk"]
X = pd.DataFrame(xTrain, columns=preprocessing.get_feature_names_out(),␣
 ↪index=train.index)
xTest = preprocessing.fit_transform(test)
yy = test["Higher_Risk"]
XX = pd.DataFrame(xTest, columns=preprocessing.get_feature_names_out(),␣
 ↪index=test.index)
```

7 5. Train Machine Learning Models

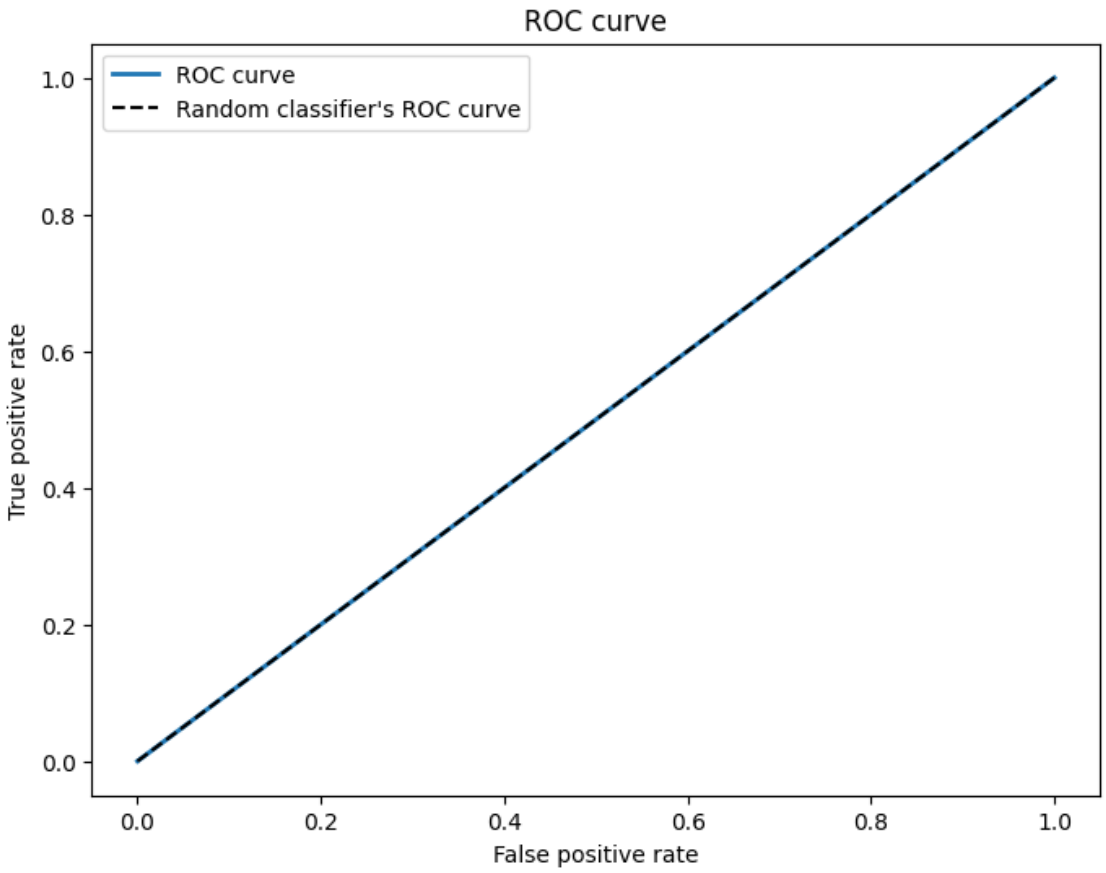
The aim of this section is to test as many Machine Learning models as possible, to understand which performs better in terms of accuracy, recall and AUC (all these metrics are self-consistent, but recall is very important for confidence in the predictions)

7.1 Dummy Classifier

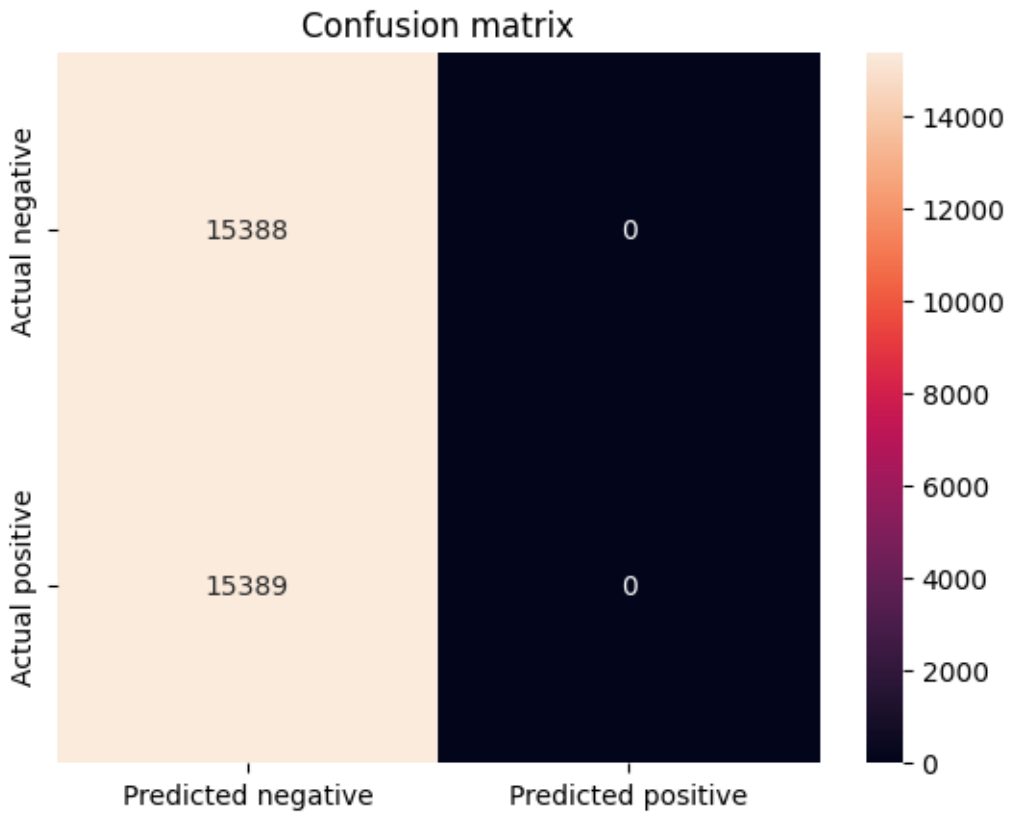
Let us first see the Dummy Classifier, which classifies completely at random. This is the worst possible classifier, but looking at it might offer insights into how well other models are performing.

[109]:
```
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier().fit(X0, y0)
evaluation_metric(dummy, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.5000040615734536
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.4999512591389115
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 5.280243454208744e-05
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.5
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.49998375410208923
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 1.0
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.0
--------------------------------------------------------------------------------
--------------------
```



Observe:

1. The accuracy is close to 50%, which coincides with the balance of predicted classes (50-50): an accuracy well-above 0.5 should be aimed at, as an accuracy close to 50% is simply guessing at random
2. The true positive rate (recall) is 0 - worst possible prediction
3. the AUC score is 0.5 signifying no ability to distinguish between classes

Note: All subsequent models should have performance metrics as far as possible (higher) than the random classifier. Anything performing simillar to the Dummy Classifier is clearly a bad model

7.2 SGD Classifier

[110]:
```
from sklearn.linear_model import SGDClassifier
```

In the following, I will use Bayesian optimization to find the best parameters for the model.

[111]:
```
##imports bayes search cv
from skopt import BayesSearchCV
```

[112]:
```
##let's define the hyperparameter space
search_space = {
    'alpha': (0.0001, 0.01, 'log-uniform'),
```

```
    'max_iter': (100, 1000),
    'penalty': ['l1', 'l2', 'elasticnet'],
    'l1_ratio': (0, 1.0),
    'learning_rate': ['constant', 'optimal', 'invscaling', 'adaptive'],
    'eta0': (0.01, 1.0, 'log-uniform'),
    'power_t': (0.1, 0.9),
    'early_stopping': [True, False],
}
```

[113]:
```
## now, let us start searching
opt = BayesSearchCV(
    SGDClassifier(),
    search_space,
        n_iter=32,
    random_state=1923
)

opt.fit(X, y)

print("val. score: %s" % opt.best_score_)
print("test score: %s" % opt.score(X, y))
print("Best parameters: ", opt.best_params_)
```

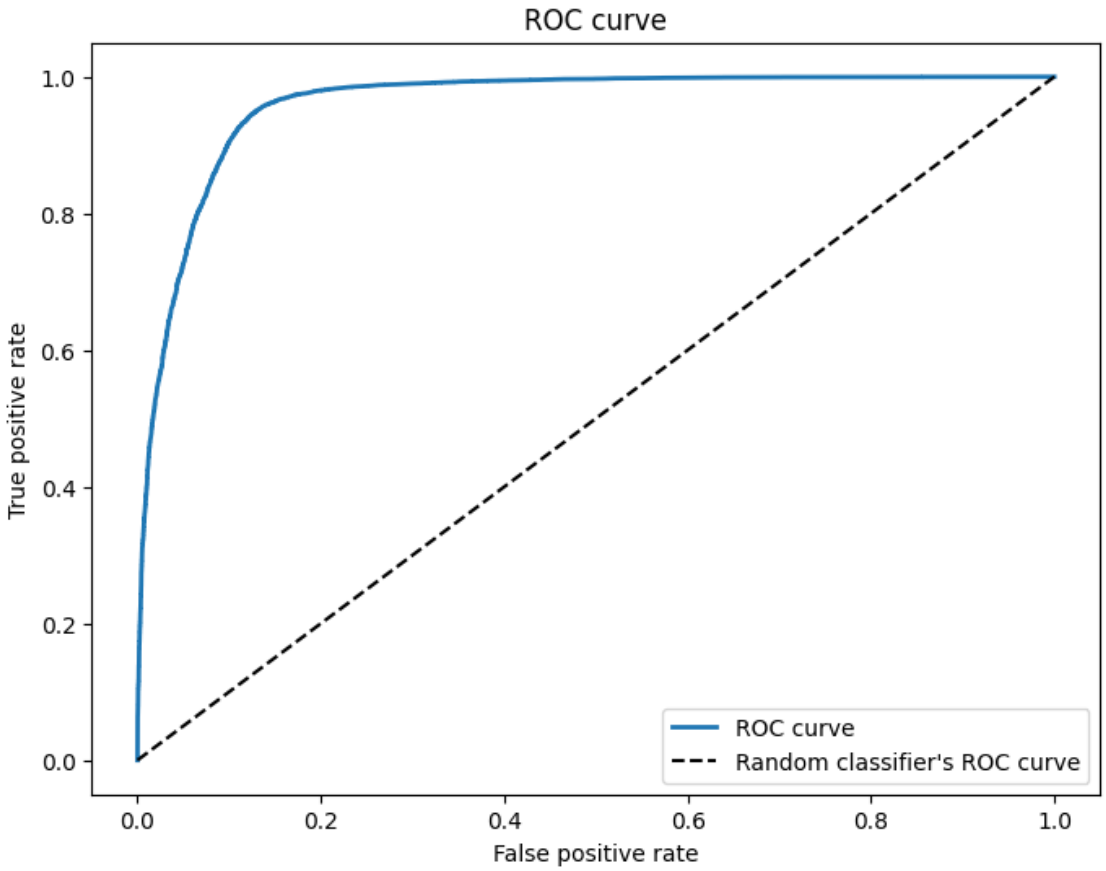
```
val. score: 0.9456941563550533
test score: 0.9438869418019693
Best parameters:  OrderedDict([('alpha', 0.00013946498668087882),
('early_stopping', False), ('eta0', 0.014763068316927273), ('l1_ratio',
0.6912218141776066), ('learning_rate', 'constant'), ('max_iter', 211),
('penalty', 'l2'), ('power_t', 0.5006359285631012)])
```

[114]:
```
##instantiating the model with the best parameters
sgd_clf = SGDClassifier(alpha= 0.00013946498668087882, eta0=0.
↪014763068316927273, l1_ratio=0.6912218141776066, learning_rate='constant', ␣
↪loss="log_loss", max_iter=211, penalty="l2", power_t=0.5, random_state=1923)
```

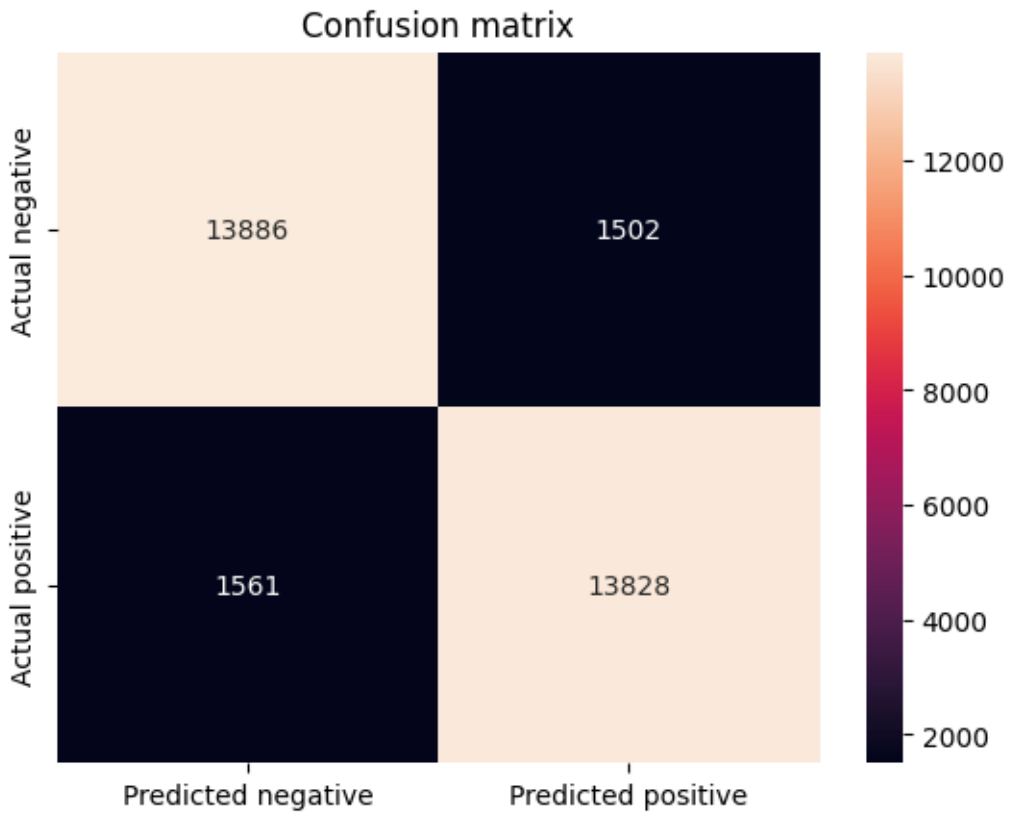
[115]:
```
##fit and evaluate model on the first subsample
sgd_clf.fit(X0, y0)
evaluation_metric(sgd_clf, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.8978775604750753
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9085677924992703
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
```

```
data: -0.010690232024195057
```

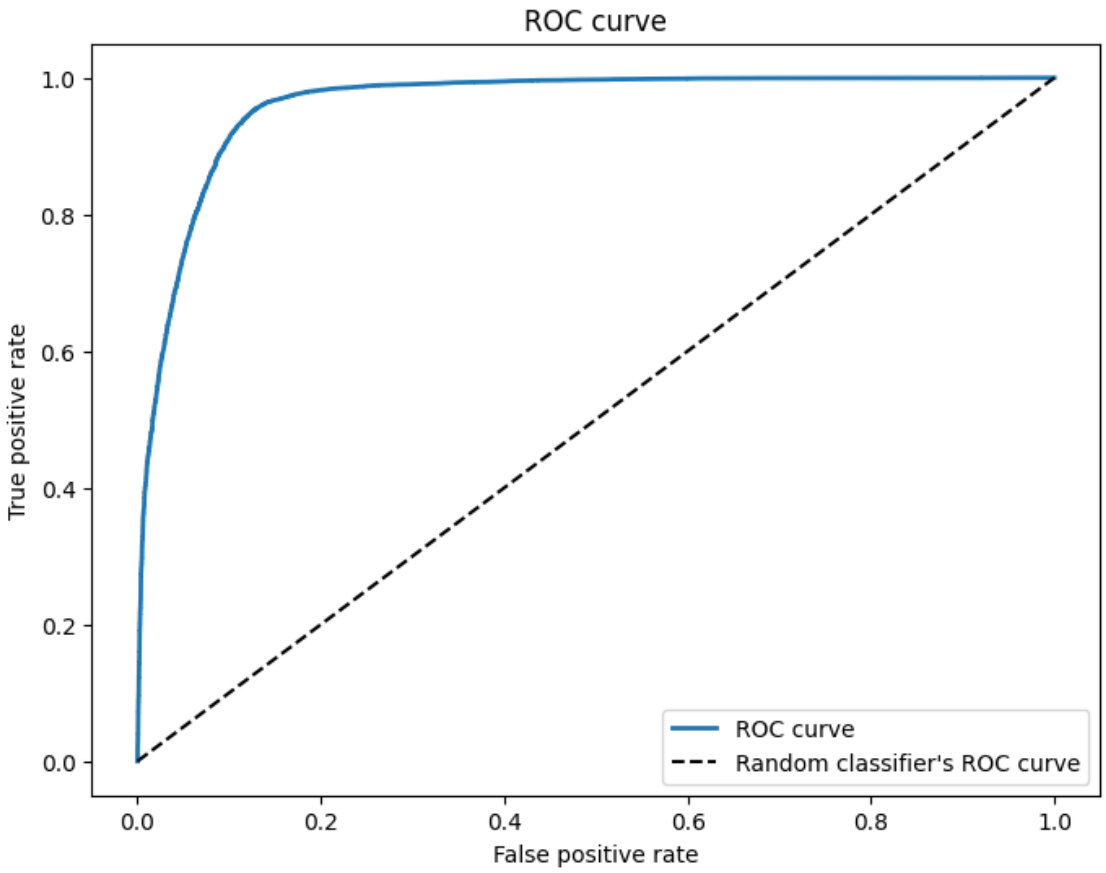



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.960738793908254
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9004776293985769
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.9023914738757474
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.8985639092858535
--------------------------------------------------------------------------------
--------------------
```

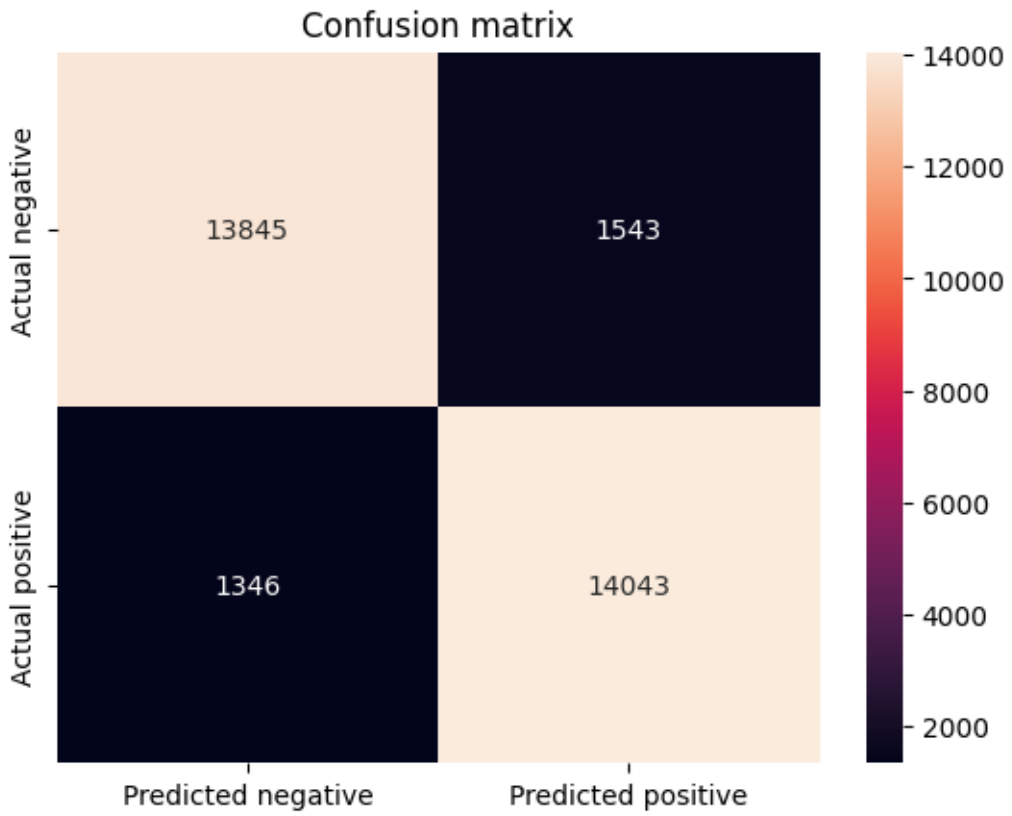


[116]:
```
##fit and evaluate model on the second subsample
sgd_clf.fit(X1, y1)
evaluation_metric(sgd_clf, X1, y1, XX1, yy1)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.8959524099589068
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9075282777754026
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.011575867816495777
```

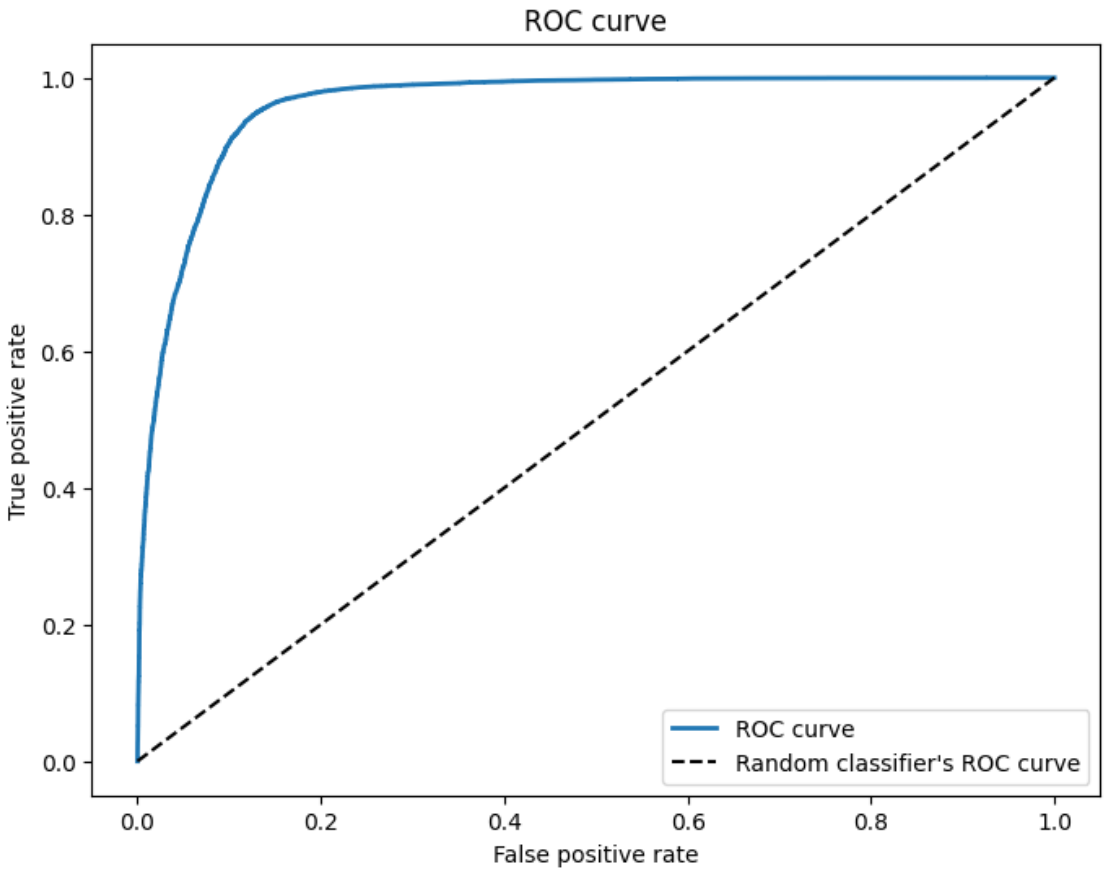



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9622342484224592
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9061312018715274
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8997270600467897
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9125349275456495
--------------------------------------------------------------------------------
--------------------
```

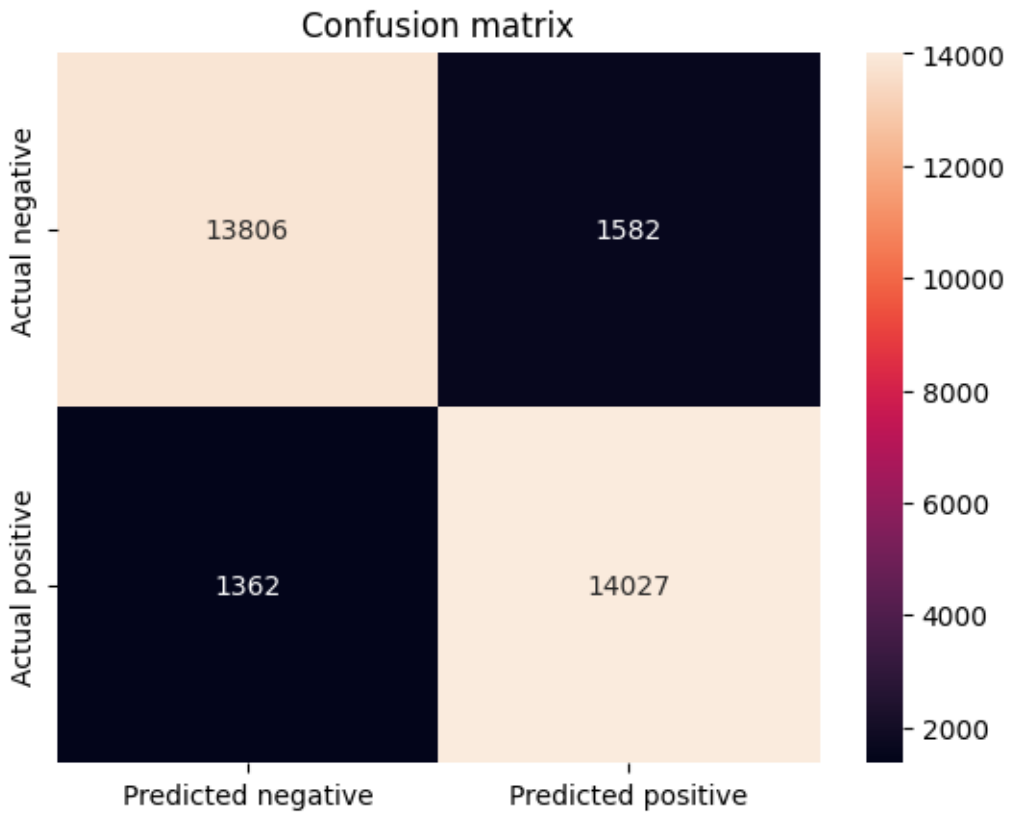


[117]:
```
##fit and evaluate model on the third subsample
sgd_clf.fit(X2, y2)
evaluation_metric(sgd_clf, X2, y2, XX2, yy2)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9010861348812351
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9077881498583539
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.006702014977118753
```

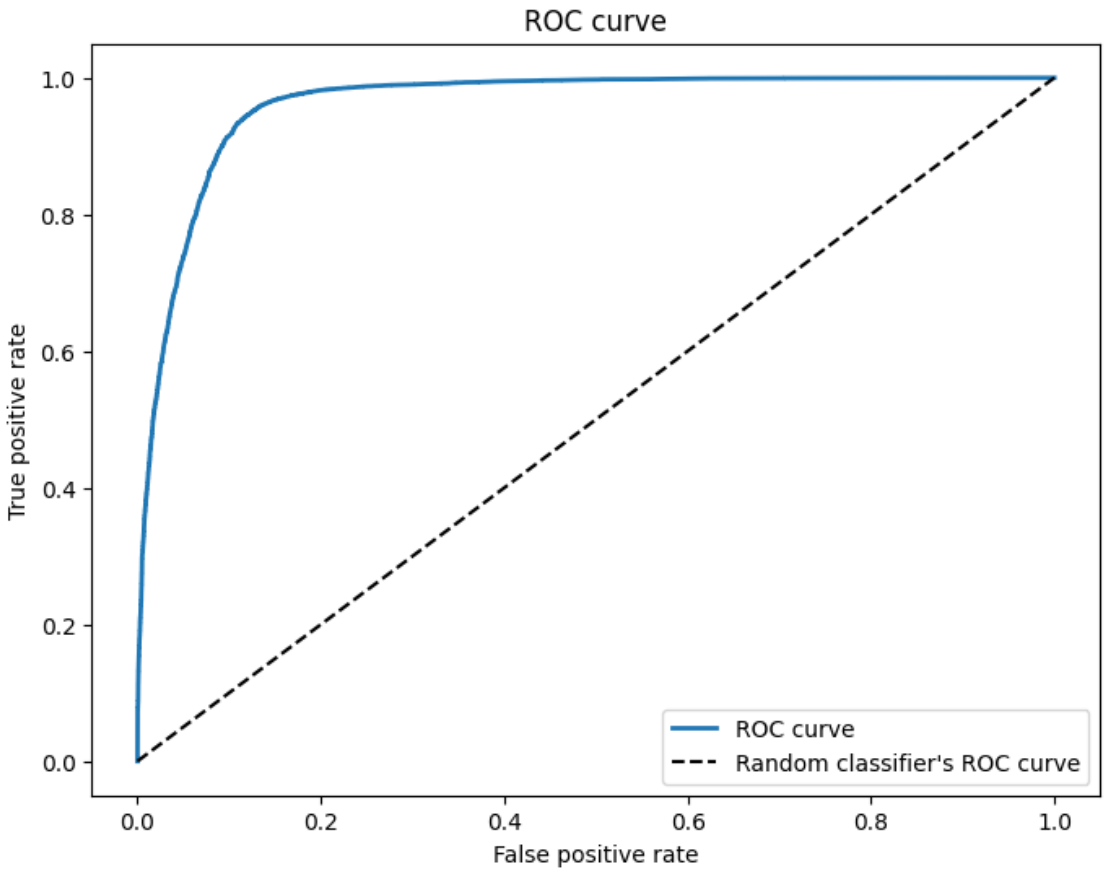



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9604284131699876
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9043441531013419
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8971926176241227
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9114952238611995
--------------------------------------------------------------------------------
--------------------
```

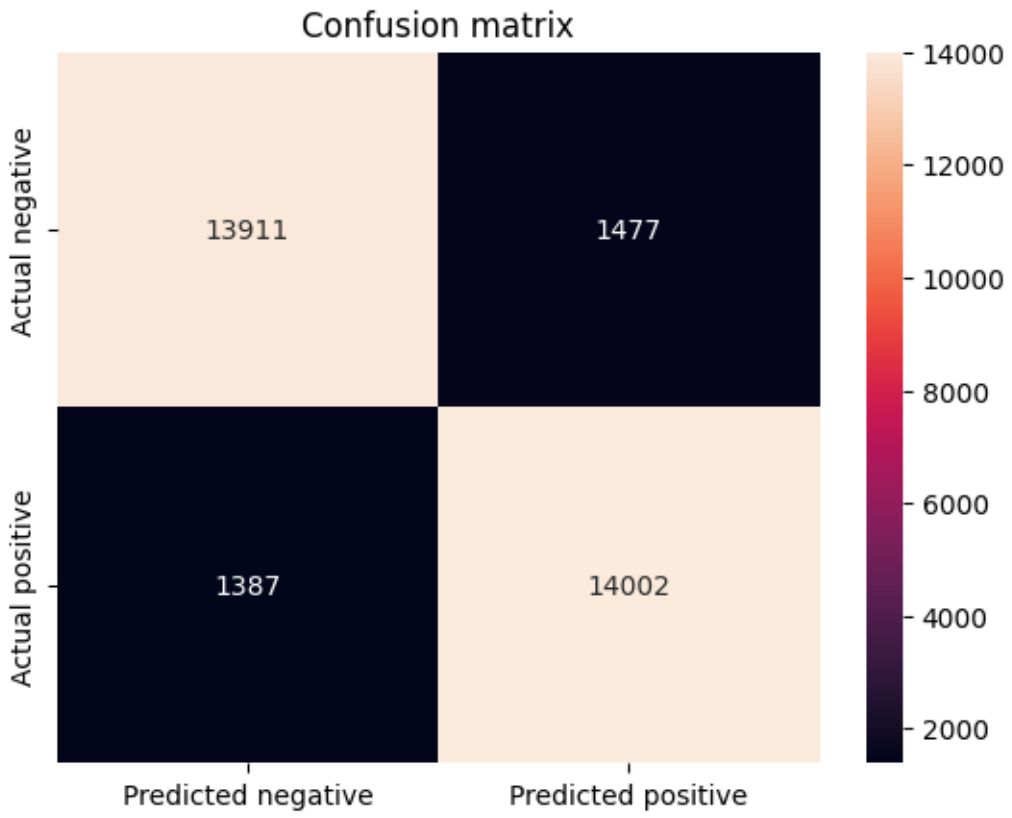


[118]:
```
##fit and evaluate model on the third subsample
sgd_clf.fit(X3, y3)
evaluation_metric(sgd_clf, X3, y3, XX3, yy3)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9050094924392098
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9107450795958213
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.005735587156611488
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9620622193704167
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9069434967670663
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.9040161164543801
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9098706868542465
--------------------------------------------------------------------------------
--------------------
```



Observations:

- Simillar accuracy scores when the model was trained and fitted on the training and test sets indicate no overfit/underfit
- 90% accuracy shows that the model performs significantly better than the dummy classifier
- The true positive rate (recall) of 90% reflects the confidence in positive predictions - 90% chance of a patient detected as high_risk to be high_risk
- The AUC score of the model is very high (96%)

7.3 RandomForest Classifer

Let's try a RandomForest Classifier and evaluate its performance

[119]:
```
from sklearn.ensemble import RandomForestClassifier
```

[120]:
```
##Hyperparameter search with BayesSearchCV

opt_rf = BayesSearchCV(
        RandomForestClassifier(),
        {
        'n_estimators': [100, 200, 300, 400, 500],
        'max_depth': [None, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
```

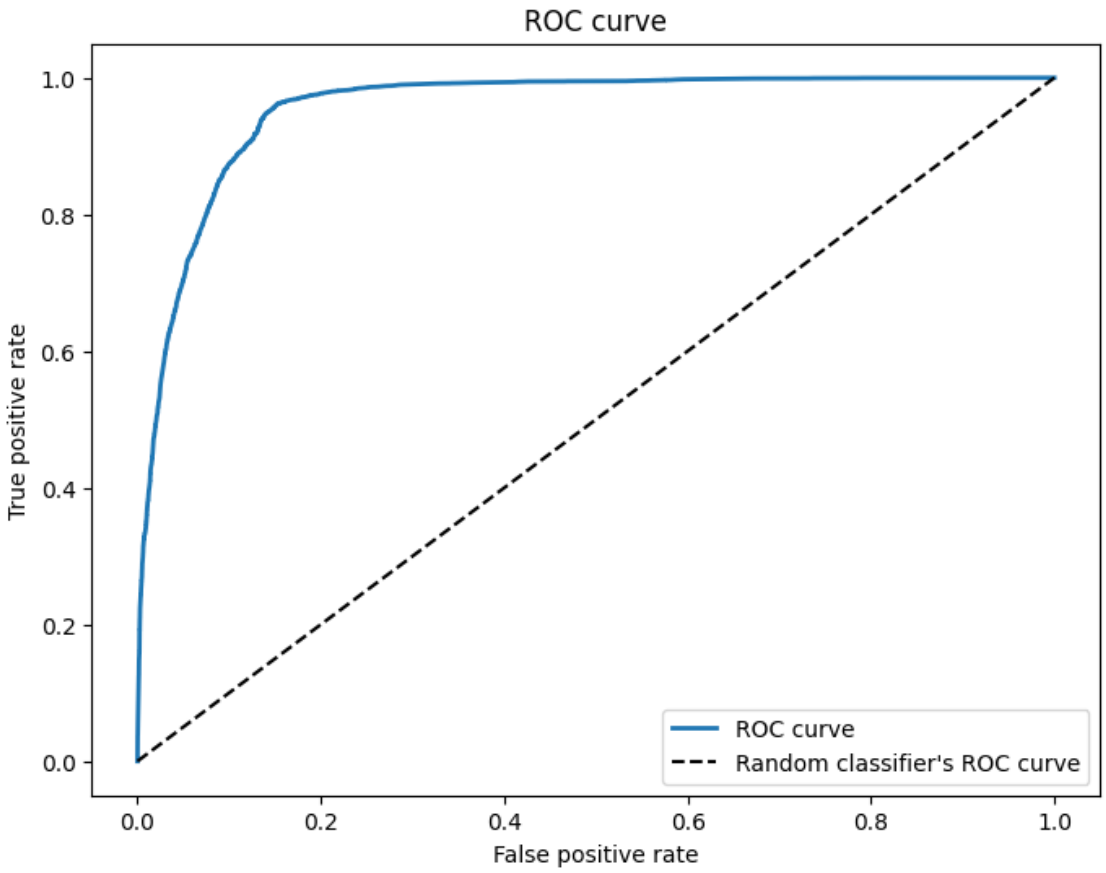
```
        'max_features': ['sqrt', 'log2'],
        },
        n_iter=3,
        random_state=1923)

##let's fit each model to the training data
opt_rf.fit(X0, y0)

##let's print the best score and the best parameters
print("val. score: %s" % opt_rf.best_score_)
print("Best parameters: ", opt_rf.best_params_)
```

```
val. score: 0.9027675339150374
Best parameters:  OrderedDict([('max_depth', 5), ('max_features', 'log2'),
('n_estimators', 100)])
```

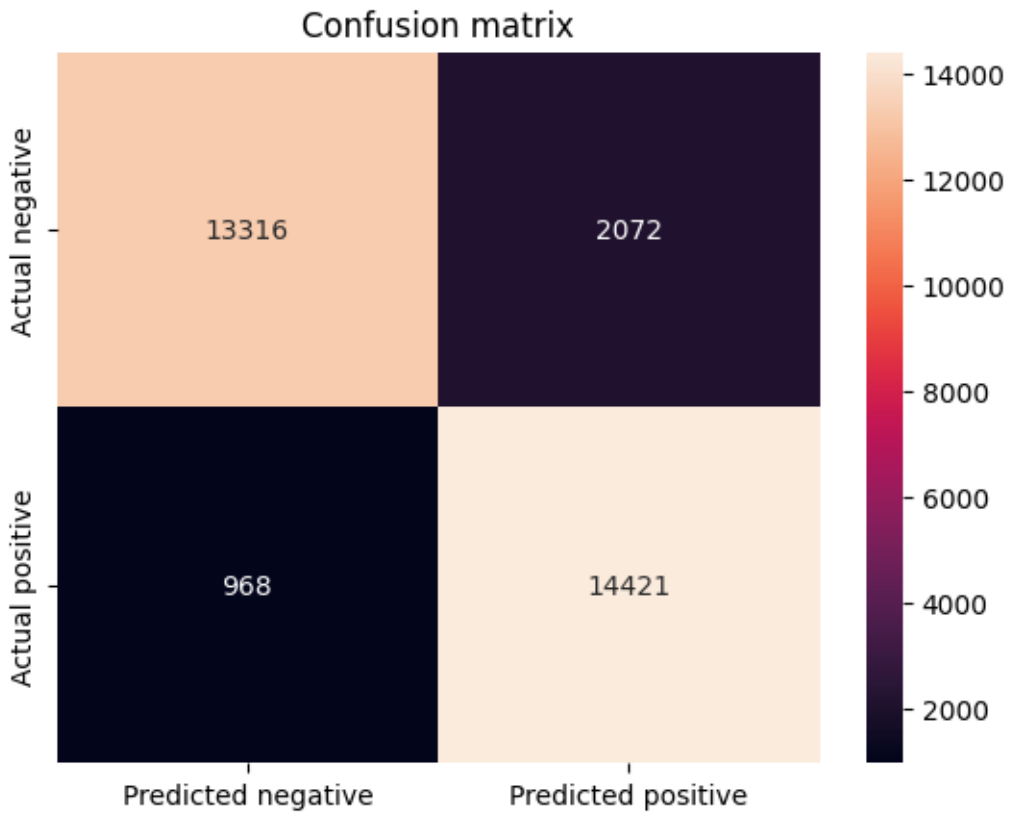
[121]:
```
##Let's instantiate the random forest classifier with the best parameters found␣
 ↪by the hyperparameter search
rf_clf = RandomForestClassifier(max_depth= 5, max_features='log2',␣
 ↪n_estimators= 100, random_state=1923)
```

[122]:
```
##Let's fit the model to the training data and the evaluate it on the␣
 ↪validation set
rf_clf.fit(X0, y0)
evaluation_metric(rf_clf, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9012404027511793
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9012245917015965
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 1.581104958281454e-05
```

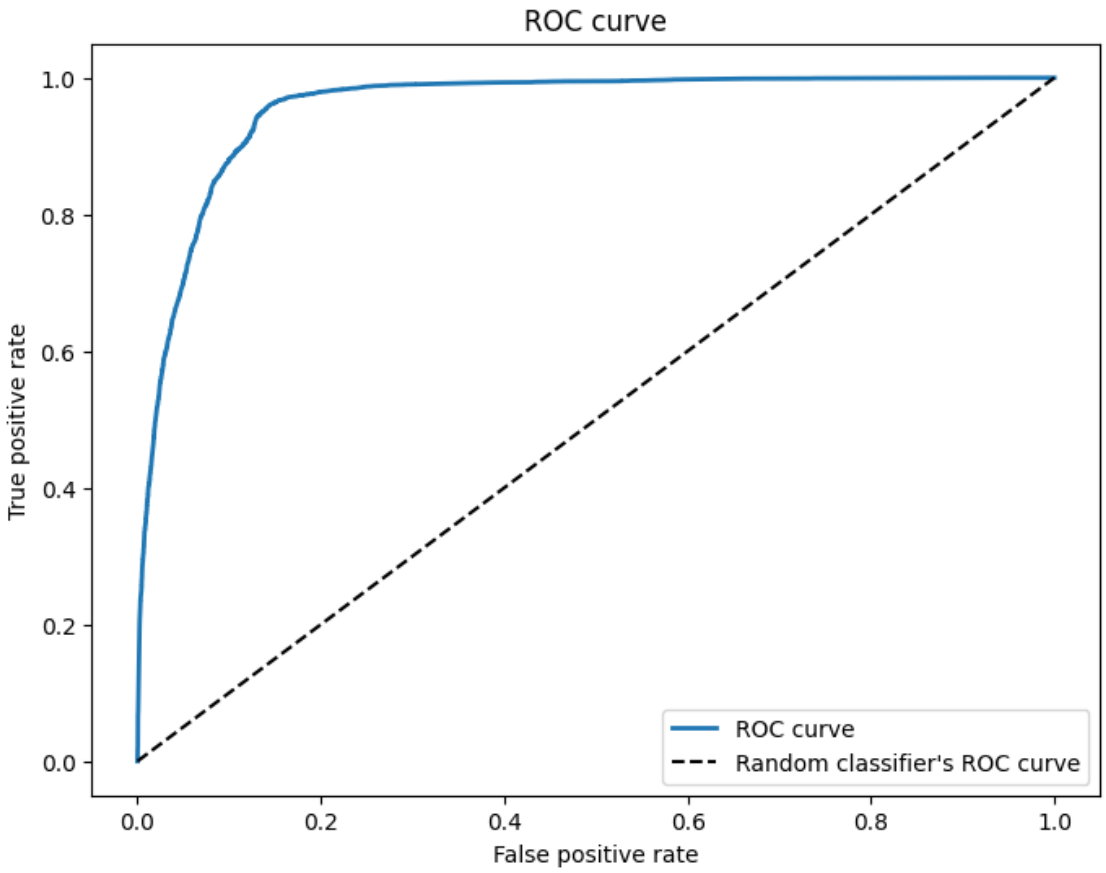



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9560676672575922
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9012249407024726
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8653496230829217
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9370979270907791
--------------------------------------------------------------------------------
--------------------
```

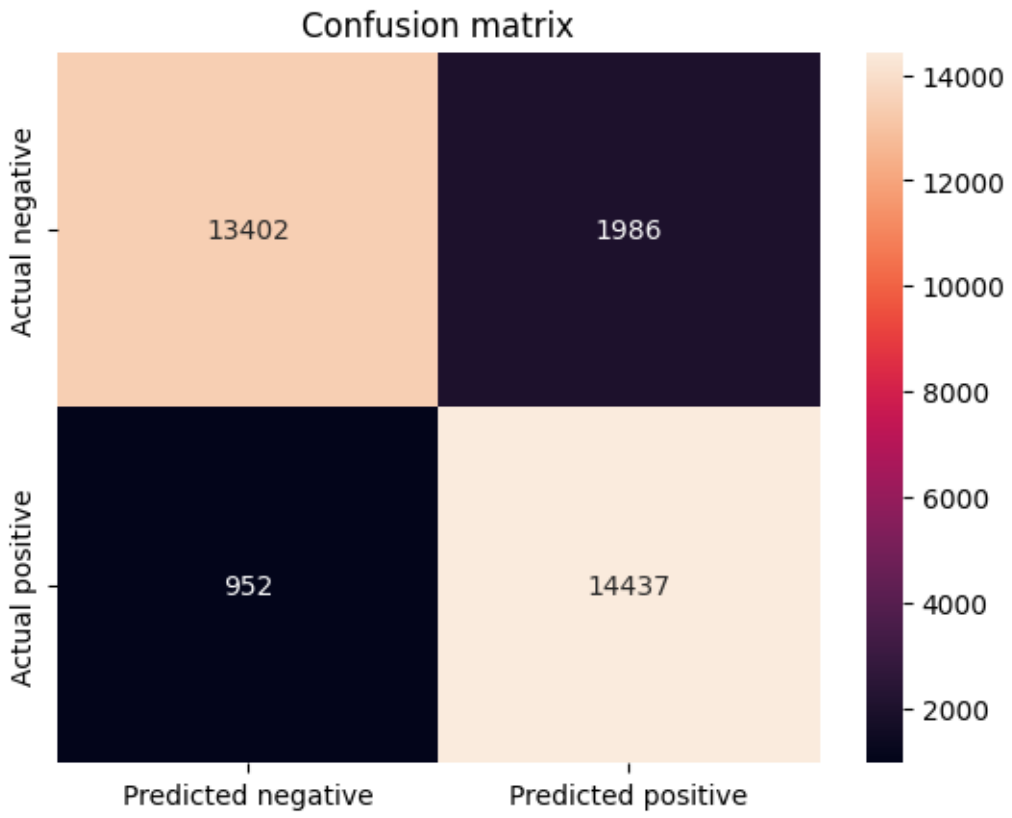


[123]:
```
##Let's fit the model to the training data and the evaluate it on the␣
 ↪validation set
rf_clf.fit(X1, y1)
evaluation_metric(rf_clf, X1, y1, XX1, yy1)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.901297257191503
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9056112269722657
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.004313969780762705
```

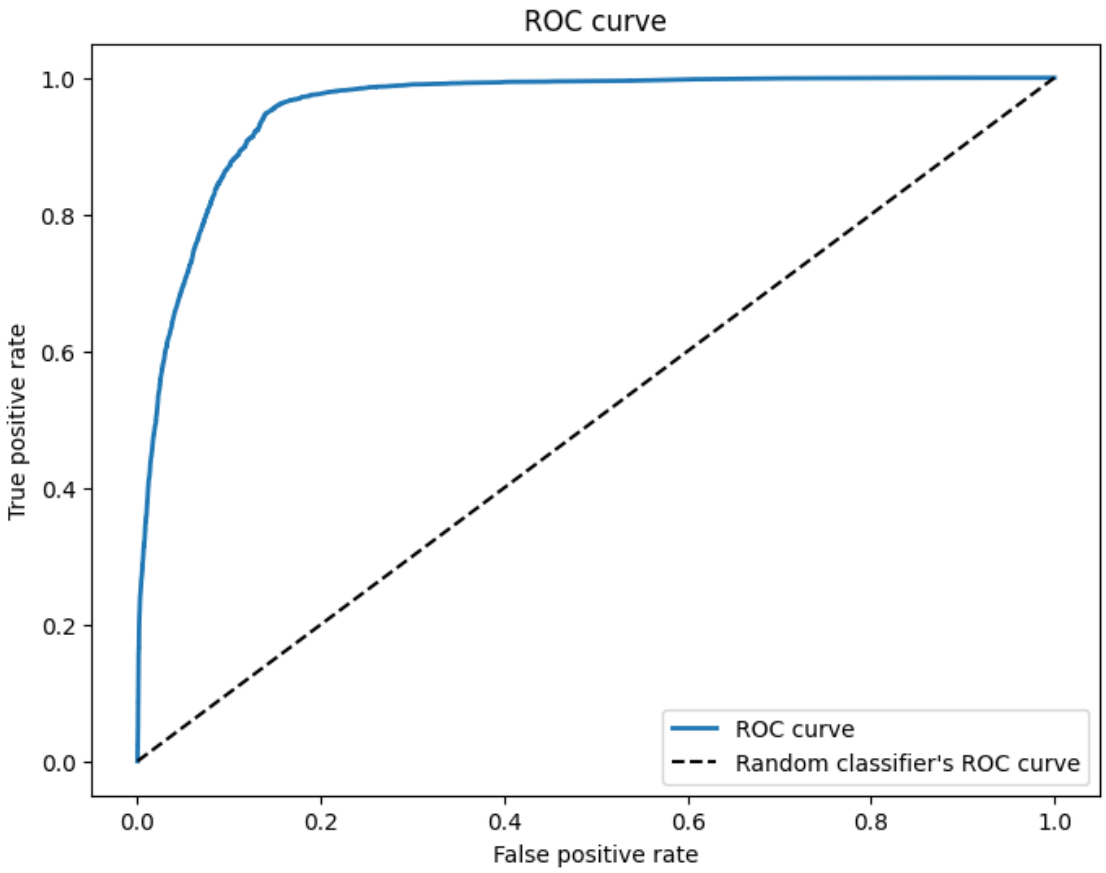



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9572540437880586
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9045391038762712
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8709383935534183
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.938137630775229
--------------------------------------------------------------------------------
--------------------
```

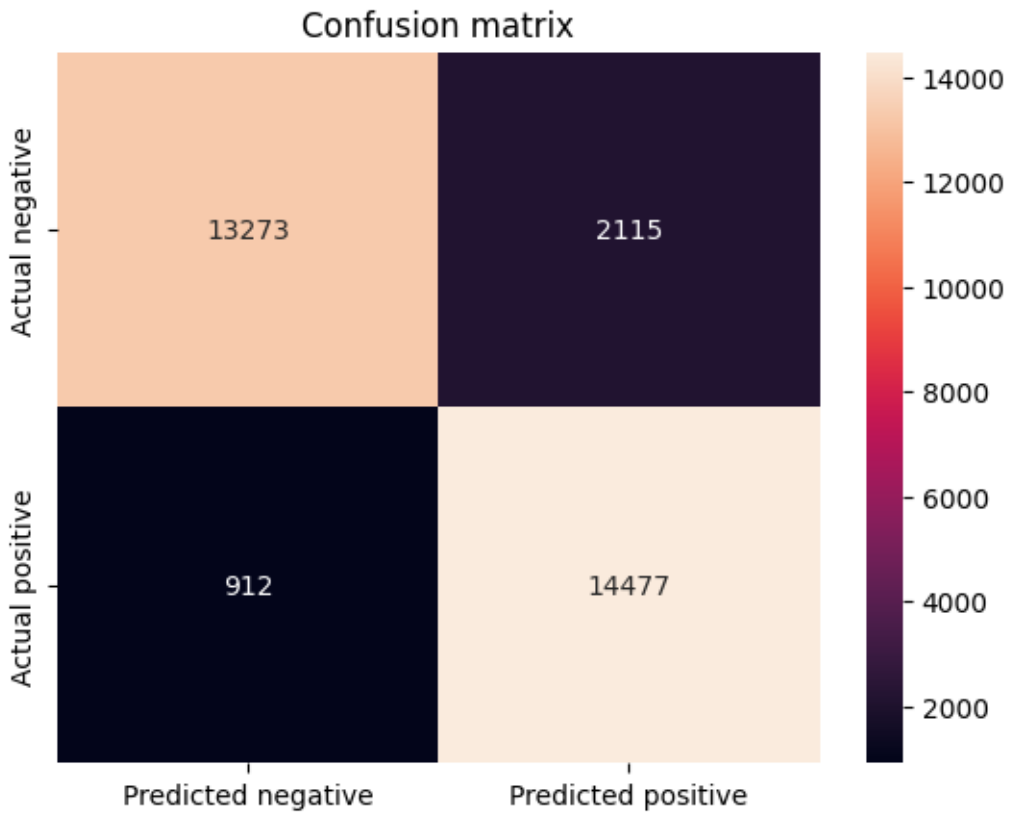


[124]:
```
rf_clf.fit(X2, y2)
evaluation_metric(rf_clf, X2, y2, XX2, yy2)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9016221979139256
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9007050692290193
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0009171286849063431
```

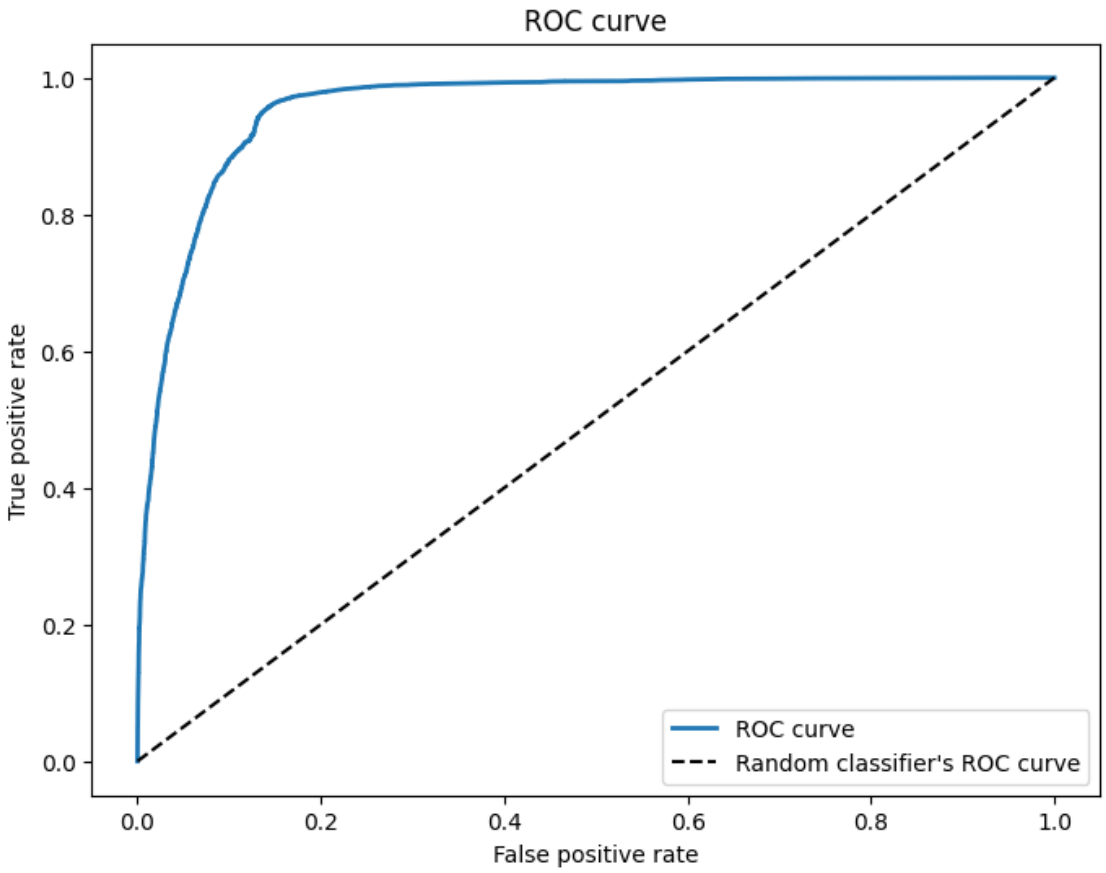



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9560303307773557
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9016473340481529
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8625552378476735
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9407368899863539
--------------------------------------------------------------------------------
--------------------
```

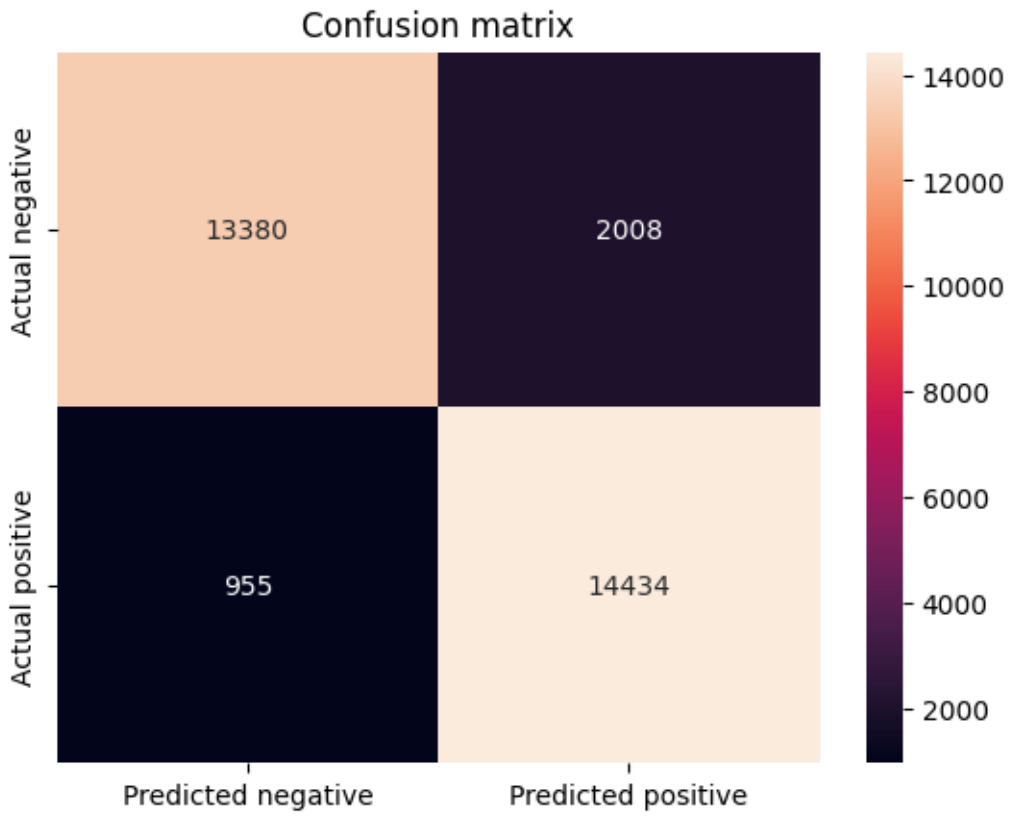


[125]:
```
rf_clf.fit(X3, y3)
evaluation_metric(rf_clf, X3, y3, XX3, yy3)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9022476676890344
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9038242890374233
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.0015766213483888736
```

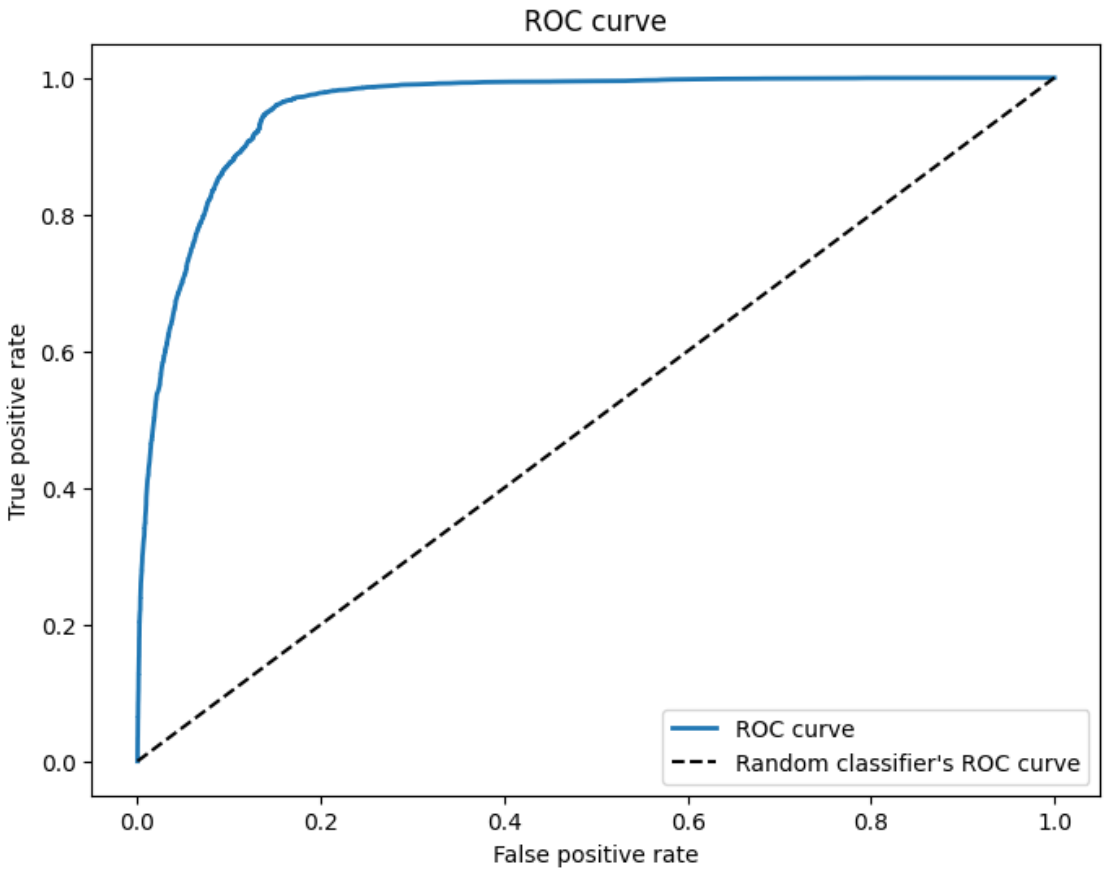



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9568149880637279
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9037268089807323
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8695087080842214
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9379426863343947
--------------------------------------------------------------------------------
--------------------
```

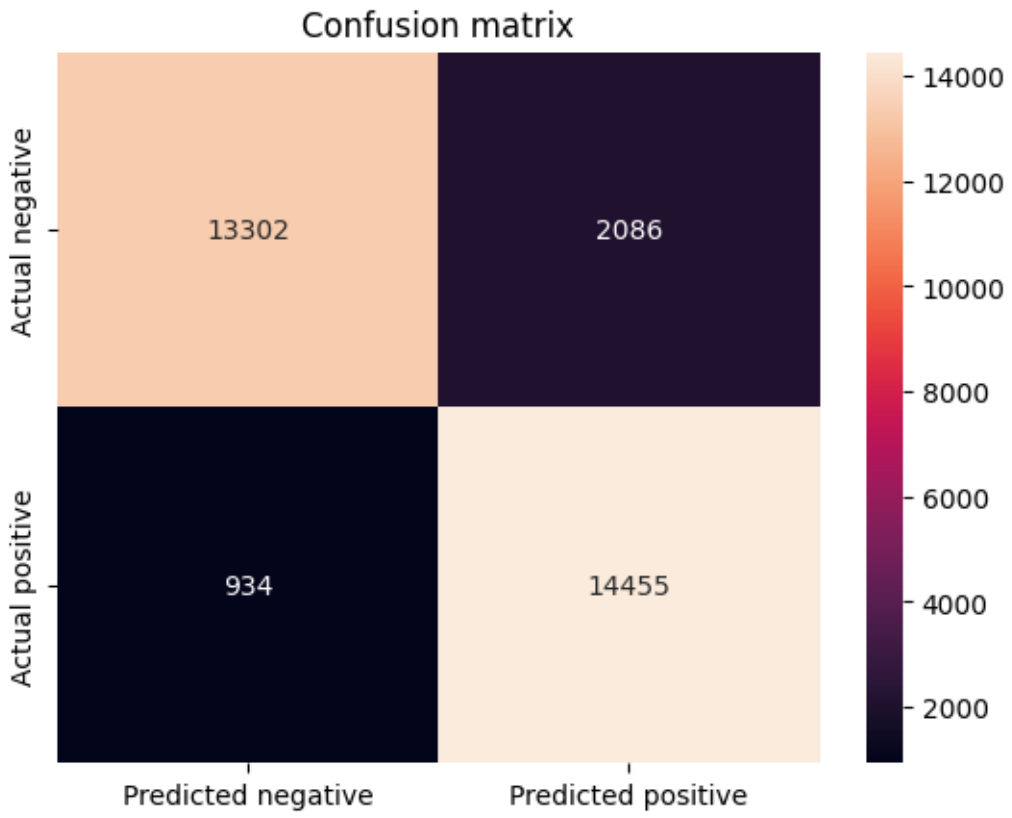


[126]:
```
rf_clf.fit(X4, y4)
evaluation_metric(rf_clf, X4, y4, XX4, yy4)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9027594015305332
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9017772995536047
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0009821019769284955
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9571605980715044
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9018747766189037
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8644398232388875
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9393072974202352
--------------------------------------------------------------------------------
--------------------
```



Observe :

- There is clealry no overfit or underfit, judging by the mean cross validation scores on the training and validation set
- The model has very good accuracy and AUC, simillar to the SGD classifier
- The TPR close to 94% indicates very high confidence in detecting positives
- Accuracy is simillar to SGD classifier at around 90%

7.4 Logistic Regression

[127]:
```
from sklearn.linear_model import LogisticRegression
```

[128]:
```
opt_log = BayesSearchCV(
        LogisticRegression(),
        { 'C' : (0.0001, 10.0, 'log-uniform'),
        'penalty' : ['l1', 'l2'],
        'solver' : ['liblinear', 'saga'],
        'max_iter' : [1000, 2000, 3000, 4000, 5000]},
        n_iter=3,
        random_state=1923)
```

```
opt_log.fit(X, y)
print("val. score: %s" % opt_log.best_score_)
print("Best parameters: ", opt_log.best_params_)
```

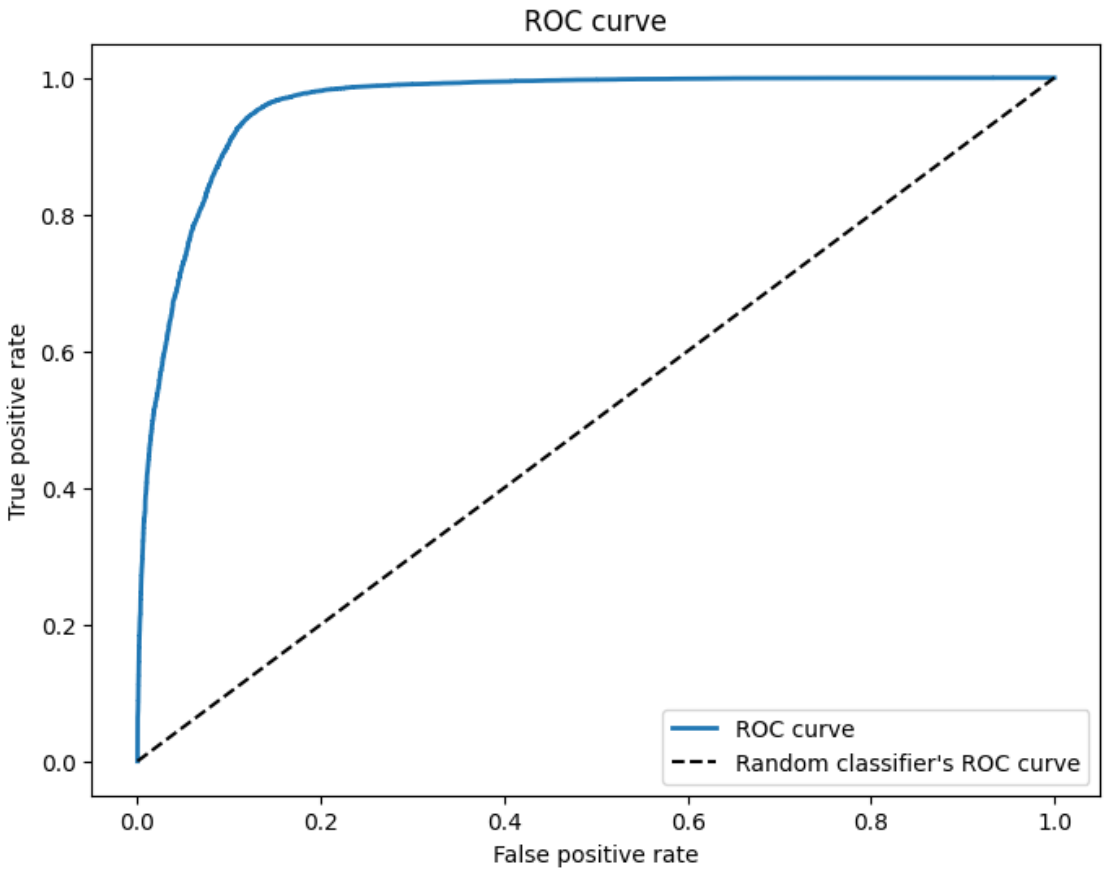
```
val. score: 0.9485396848103378
Best parameters:  OrderedDict([('C', 0.04067517192324702), ('max_iter', 3000),
('penalty', 'l1'), ('solver', 'liblinear')])
```

Let us try these hyper-parameters to instantiate a models

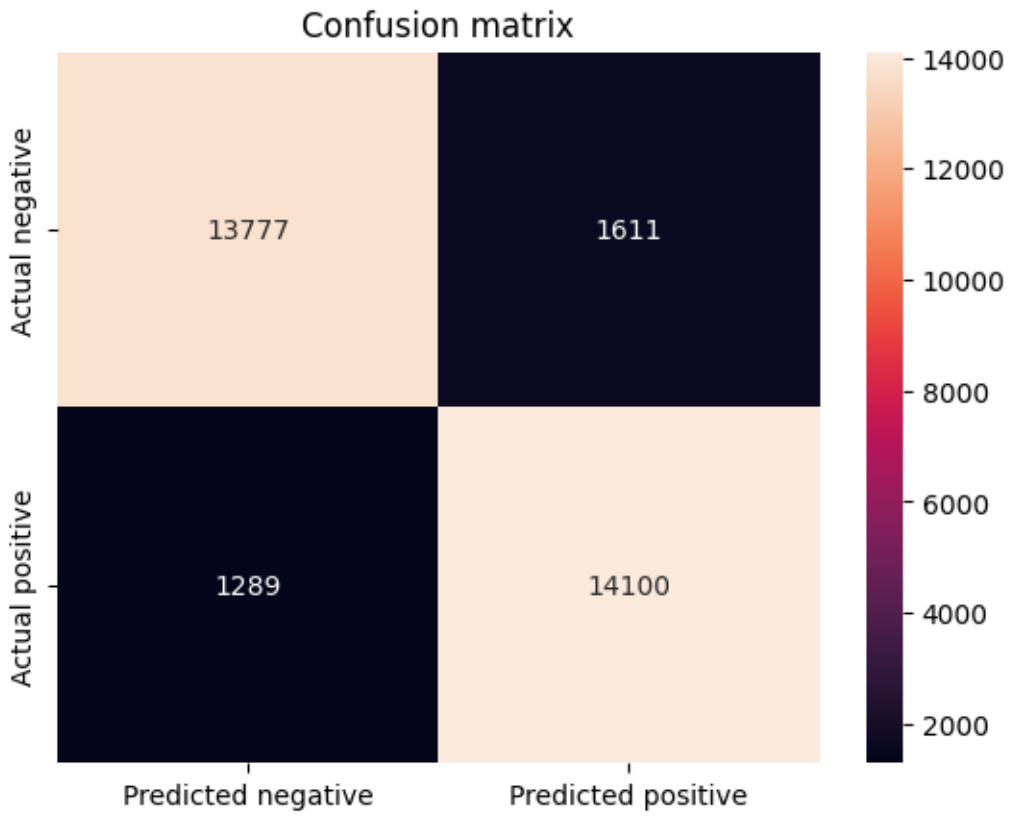
[129]:
```
log_clf = LogisticRegression(C= 0.04, max_iter= 3000, penalty= 'l1', solver=␣
 ↪'liblinear', random_state=1923)
```

[130]:
```
##now train and evaluate the model
log_clf.fit(X0, y0)
evaluation_metric(log_clf, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9054725019154899
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.90577365956034
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.0003011576448500586
```

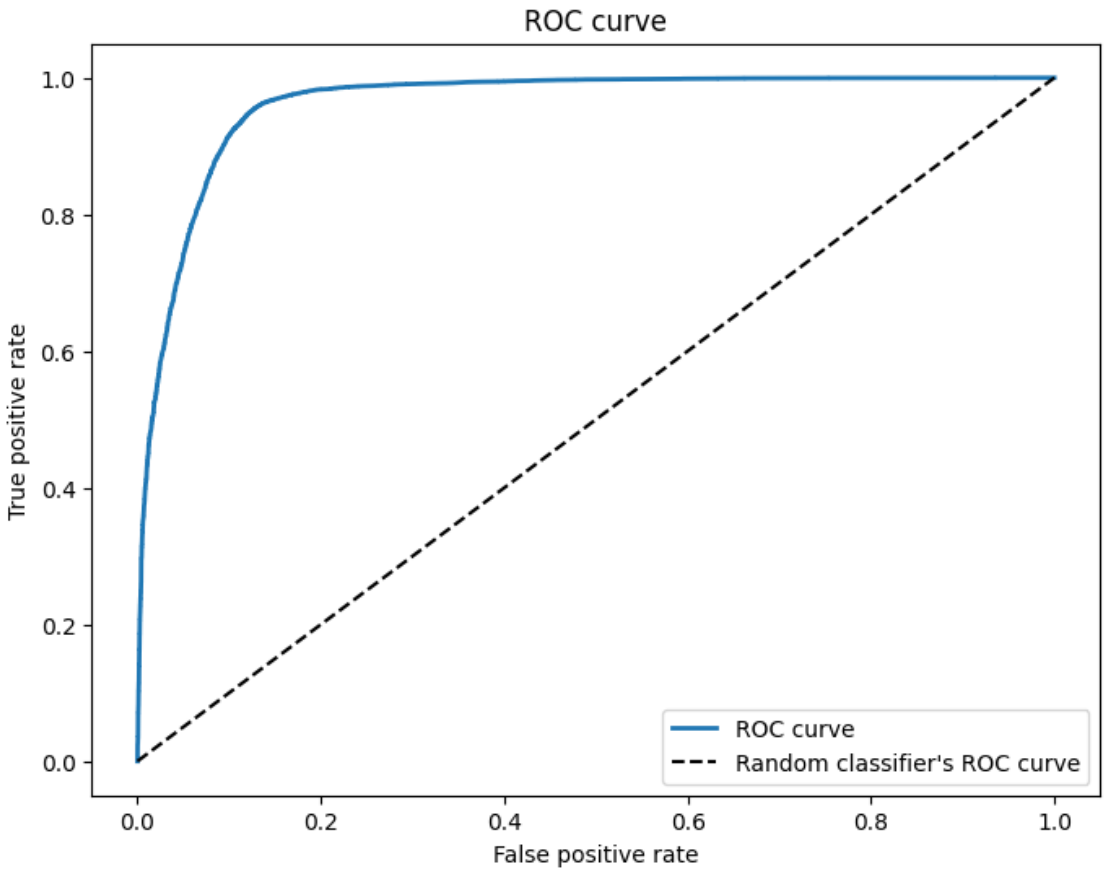



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9610088716020848
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9057737921174903
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8953080322329088
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9162388719215023
--------------------------------------------------------------------------------
--------------------
```

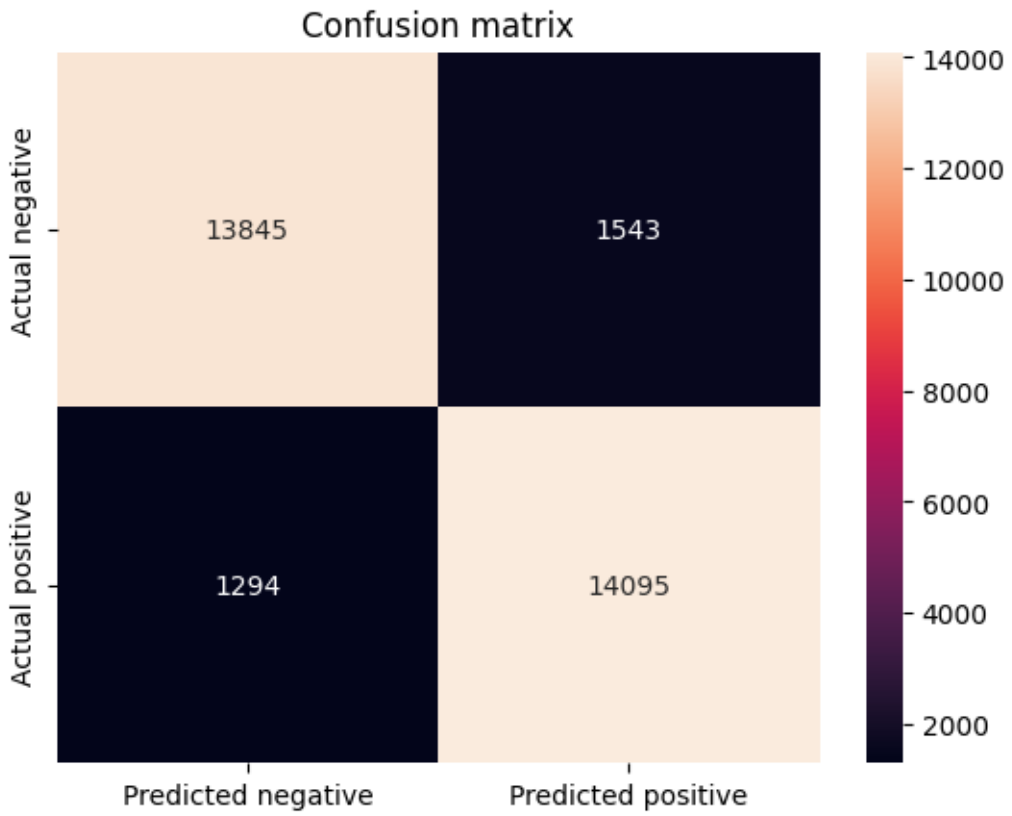


[131]:
```
log_clf.fit(X1, y1)
evaluation_metric(log_clf, X1, y1, XX1, yy1)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9046764585920706
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9077232148276941
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.00304675623562356
```

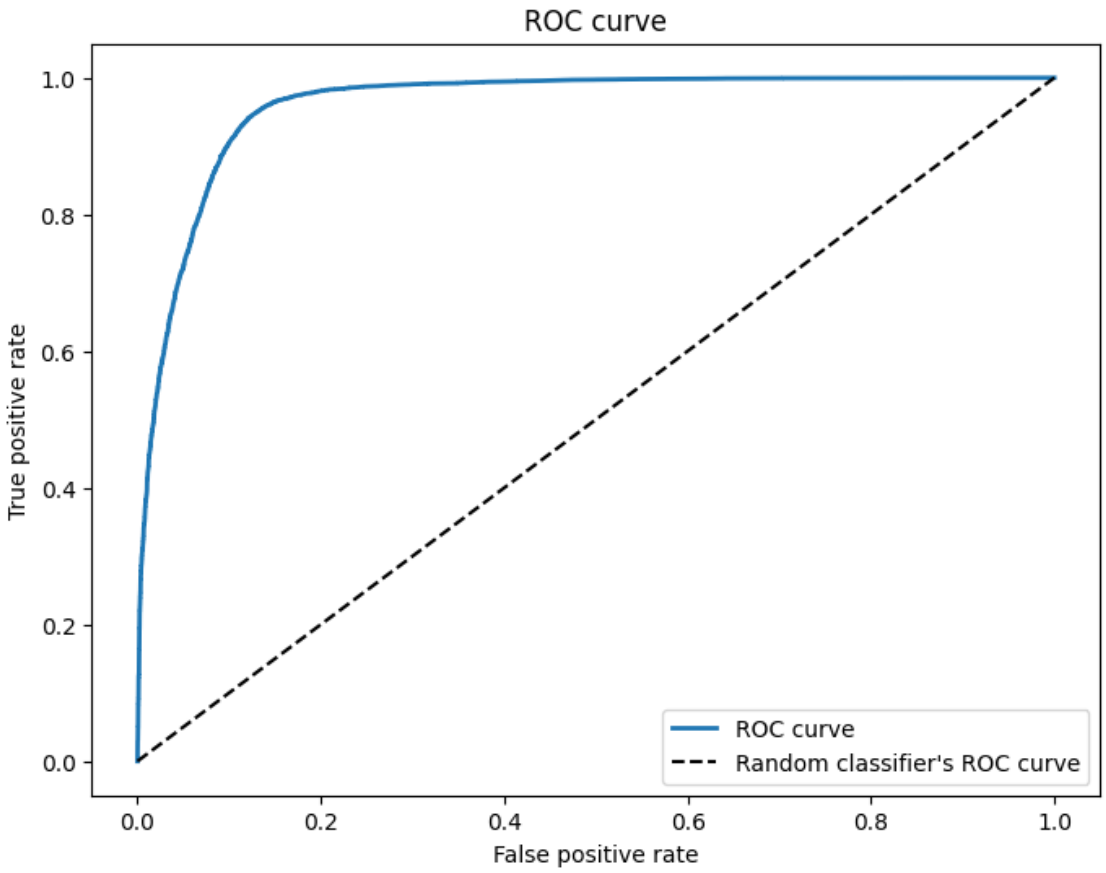



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9626471371502636
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9078207752542483
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8997270600467897
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9159139645201118
--------------------------------------------------------------------------------
--------------------
```

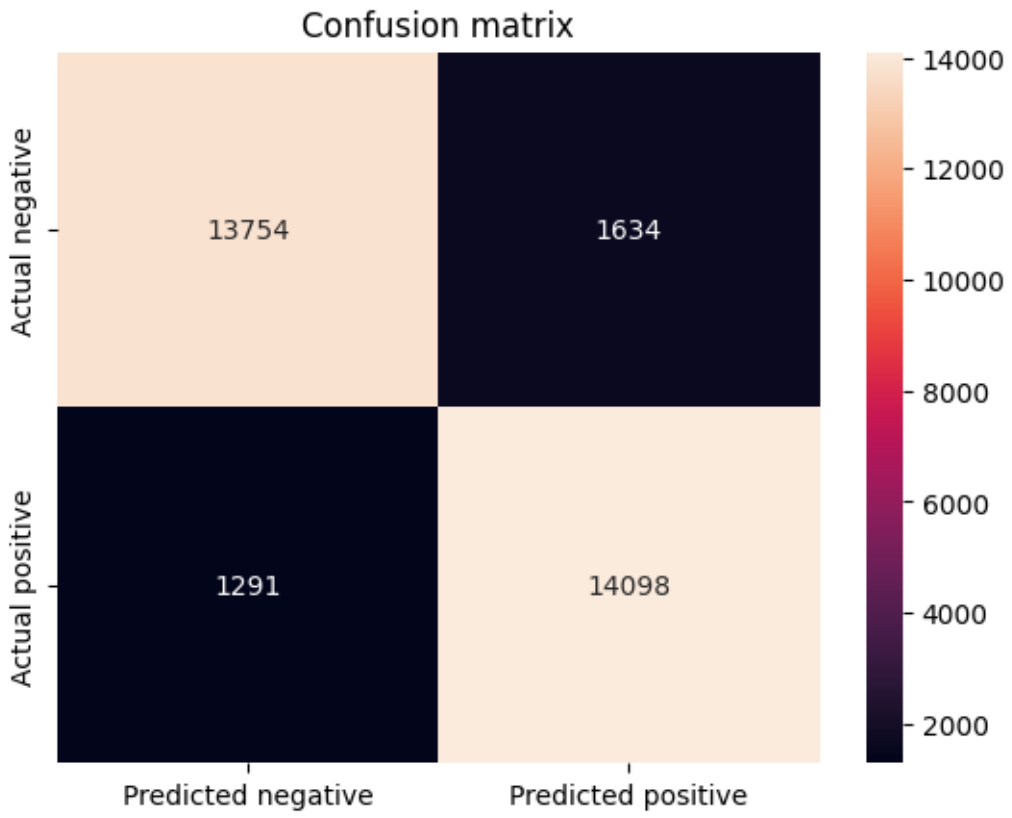


[132]:
```
log_clf.fit(X2, y2)
evaluation_metric(log_clf, X2, y2, XX2, yy2)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9053344275530915
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9044741882989207
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0008602392541707804
```

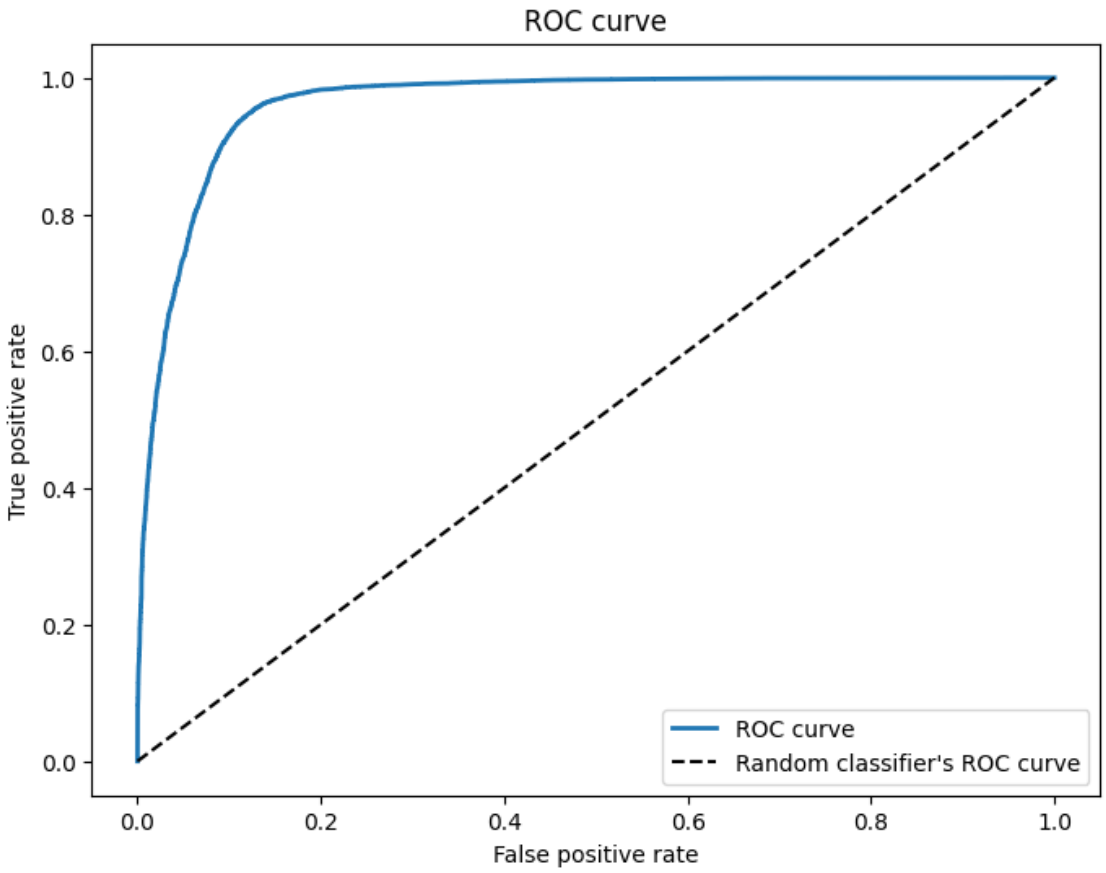



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9606886156889008
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9049614972219514
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8938133610605666
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9161089089609461
--------------------------------------------------------------------------------
--------------------
```

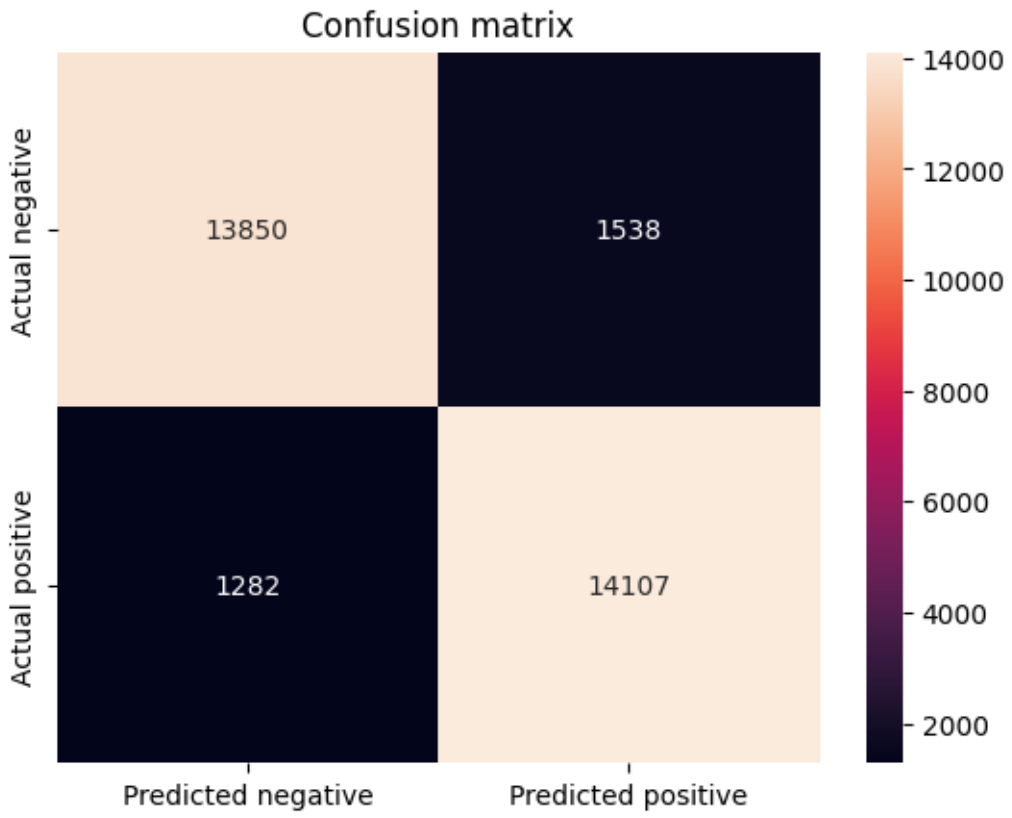


[133]:
```
log_clf.fit(X3, y3)
evaluation_metric(log_clf, X3, y3, XX3, yy3)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9062198248326607
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9084057610705465
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.002185936237885766
```

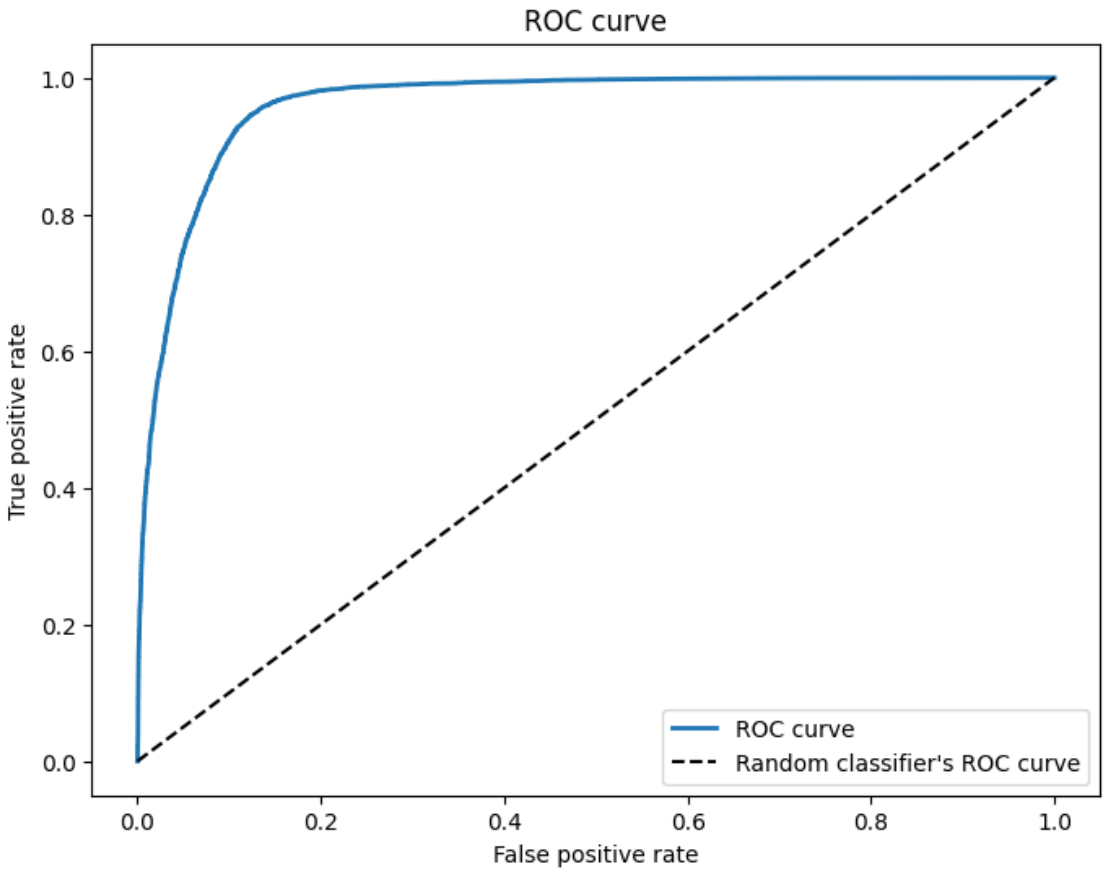



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9622099648246989
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9083731357832148
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.9000519885625162
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9166937422834492
--------------------------------------------------------------------------------
--------------------
```

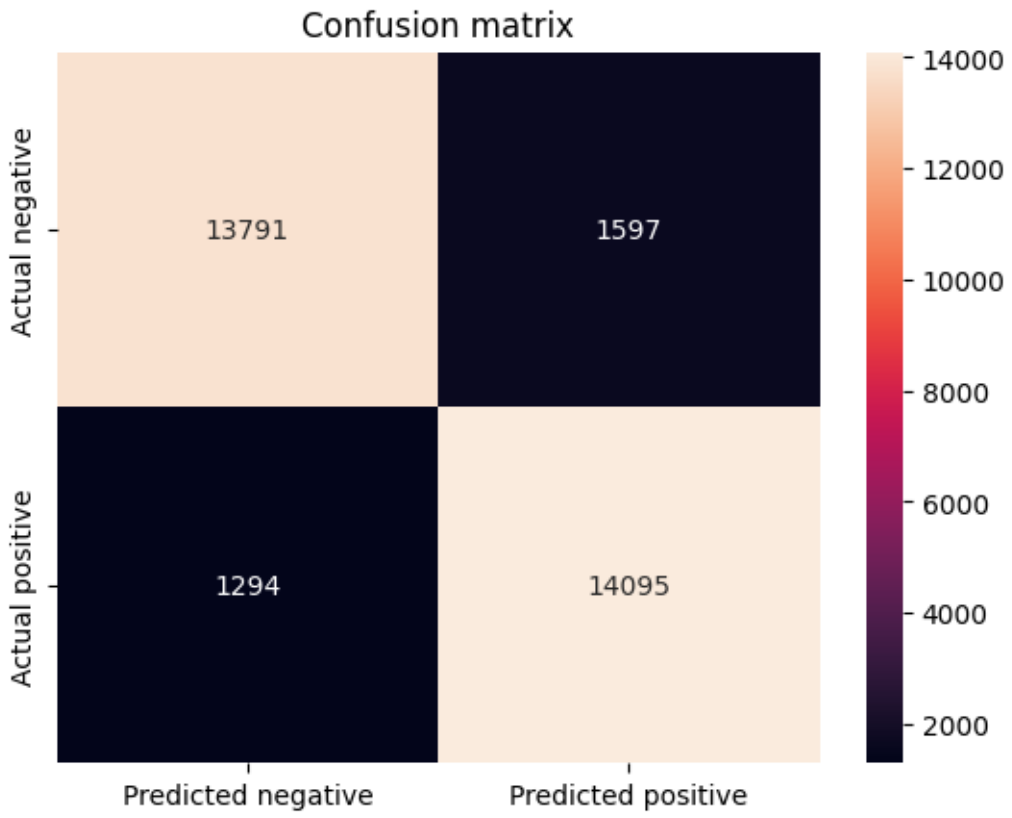


[134]:
```
log_clf.fit(X4, y4)
evaluation_metric(log_clf, X4, y4, XX4, yy4)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9062035729303053
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9059363349553895
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0002672379749157905
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.96191008213426
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9060662182798843
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.896217832076943
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9159139645201118
--------------------------------------------------------------------------------
--------------------
```



Observe:

- Accuracy is simillar to previous models at around 90%
- The true positive rate is slightly lower than that of the RandomForest model at close to 92%
- AUC is over 96%

7.5 Nearest Neighbours Classifier

[135]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[136]:
```
#opt_knn = BayesSearchCV(
        #KNeighborsClassifier(),
        #{'n_neighbors': [10, 20, 30, 50],
        # 'weights': ['uniform', 'distance'],
         #'metric': ['euclidean', 'manhattan'],
         #'algorithm':  ['auto', 'ball_tree', 'kd_tree', 'brute']},
       # n_iter=3,
        #random_state=1923)

#opt_knn.fit(X, y)
#print("val. score: %s" % opt_knn.best_score_)
```

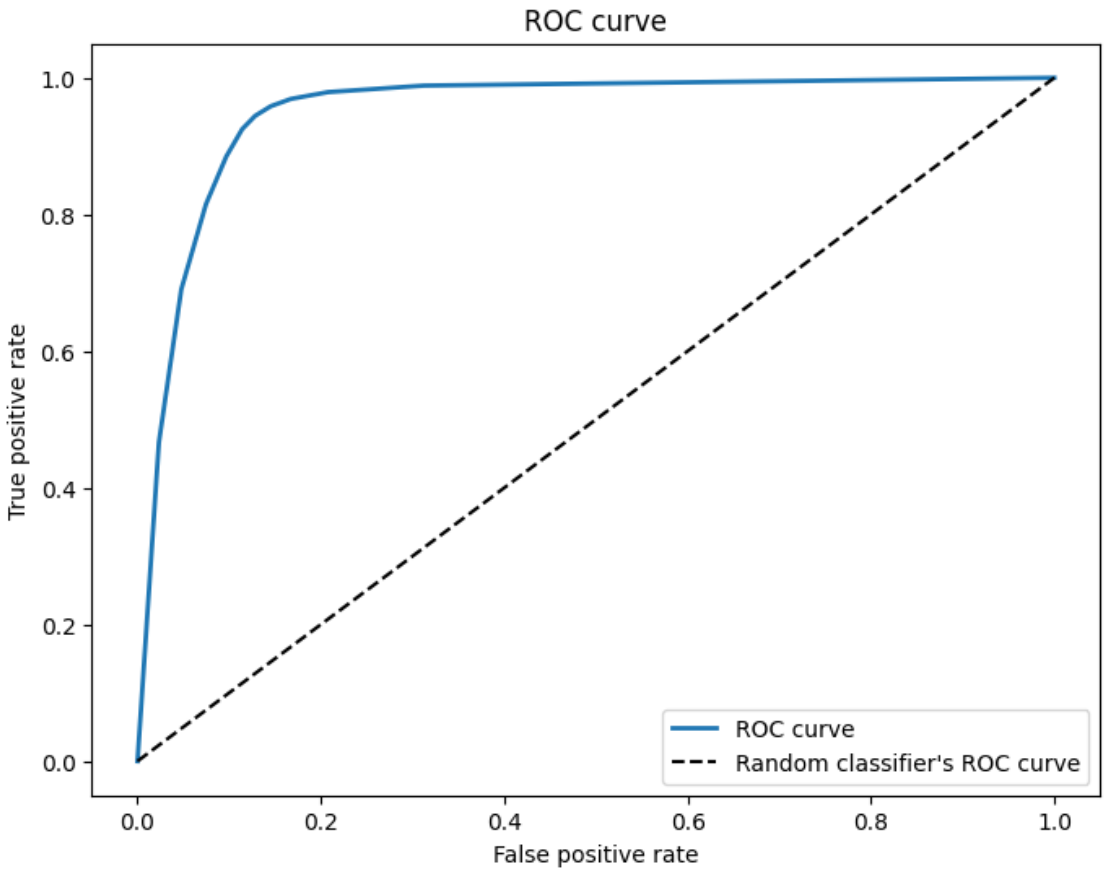
```
#print("Best parameters: ", opt_knn.best_params_)
```

Note: It took about 83 minutes to find the best hyperparameters on an M2 Macbook. According to Bayes Search Cross Validation the best algorithm is 'auto', the distance measure is 'euclidean', it has 10 nearest neighbours and the weights are uniform. Let us instantiate a model with these metrics and see how well it performs given our evaluation function.

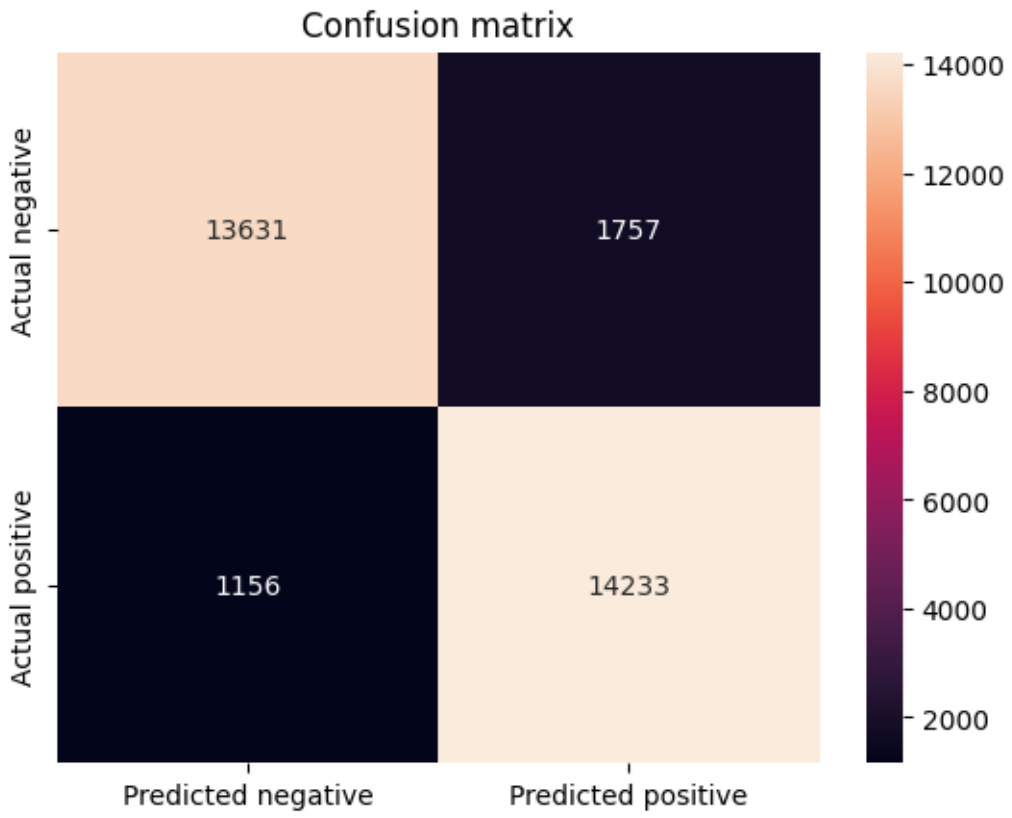
[137]:
```
##instantiating the model with the best parameters found by the BayesSearchCV␣
↪hyperparameter search
knn_clf = KNeighborsClassifier(algorithm= 'auto', metric= 'euclidean',␣
↪n_neighbors= 10, weights= 'uniform')
```

[138]:
```
##let us fit and evaluate the model
knn_clf.fit(X0, y0)
evaluation_metric(knn_clf, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9060167237258188
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9034990702076369
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.002517653518181895
```

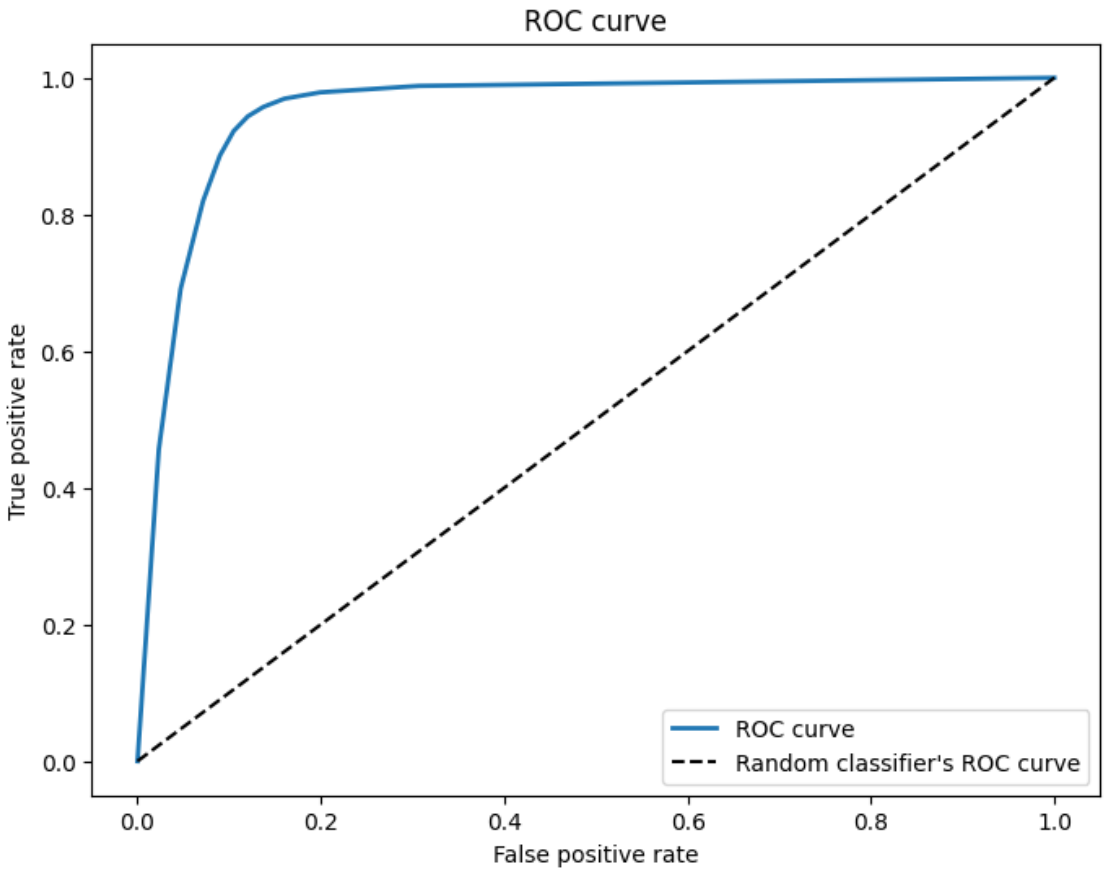



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9509492249543816
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9053513987718101
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8858201195736938
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9248814087984925
--------------------------------------------------------------------------------
--------------------
```

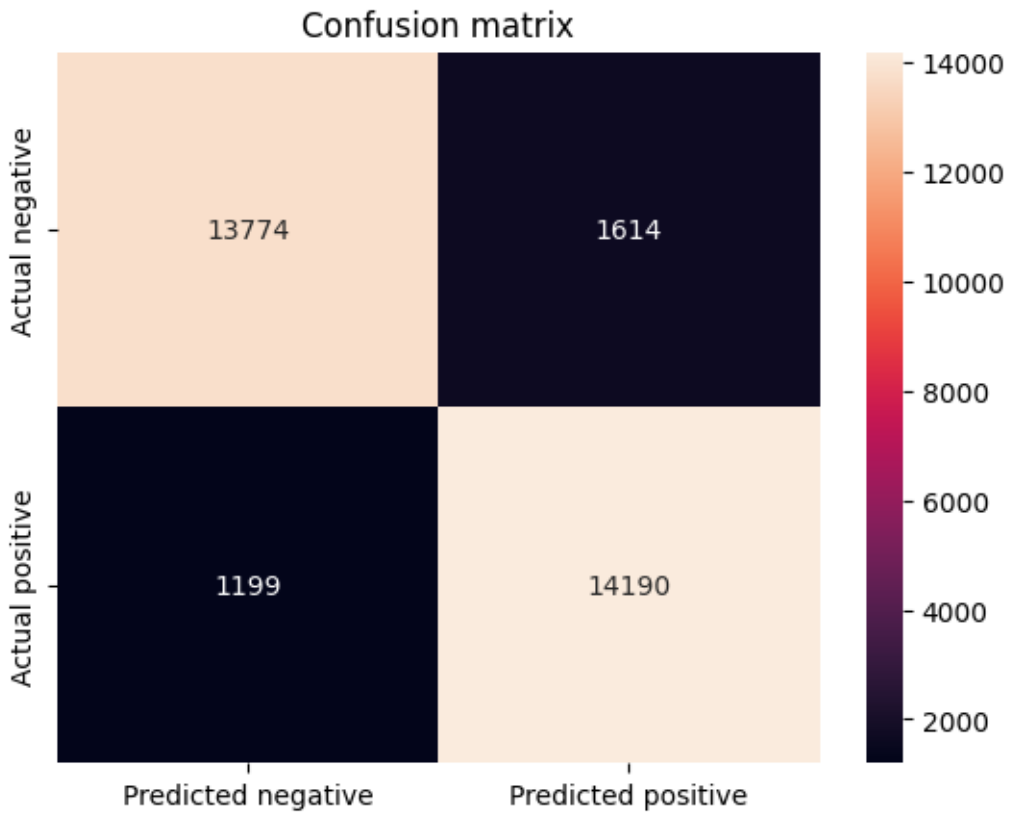


[139]:
```
knn_clf.fit(X1, y1)
evaluation_metric(knn_clf, X1, y1, XX1, yy1)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9057243145208977
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9060010377358989
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.00027672321500116226
```

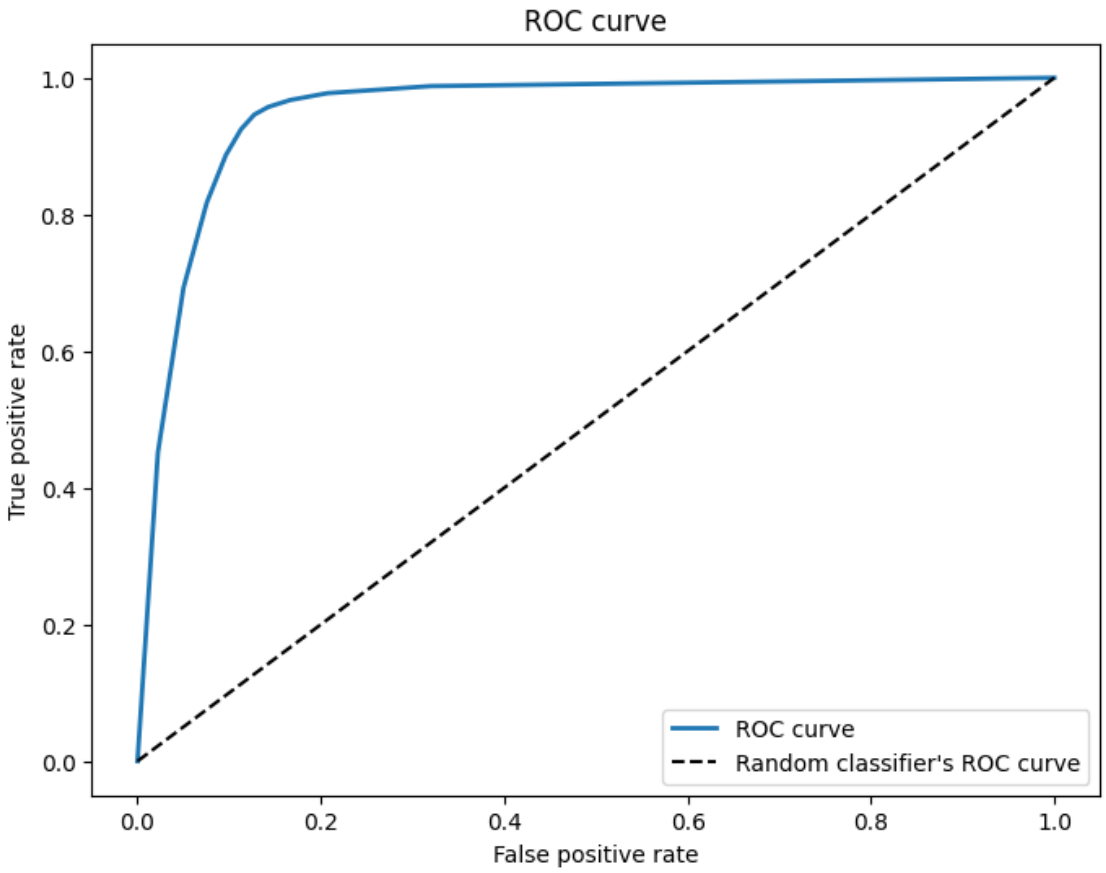



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9521557382270307
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9086005783539656
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8951130751234728
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9220872051465332
--------------------------------------------------------------------------------
--------------------
```

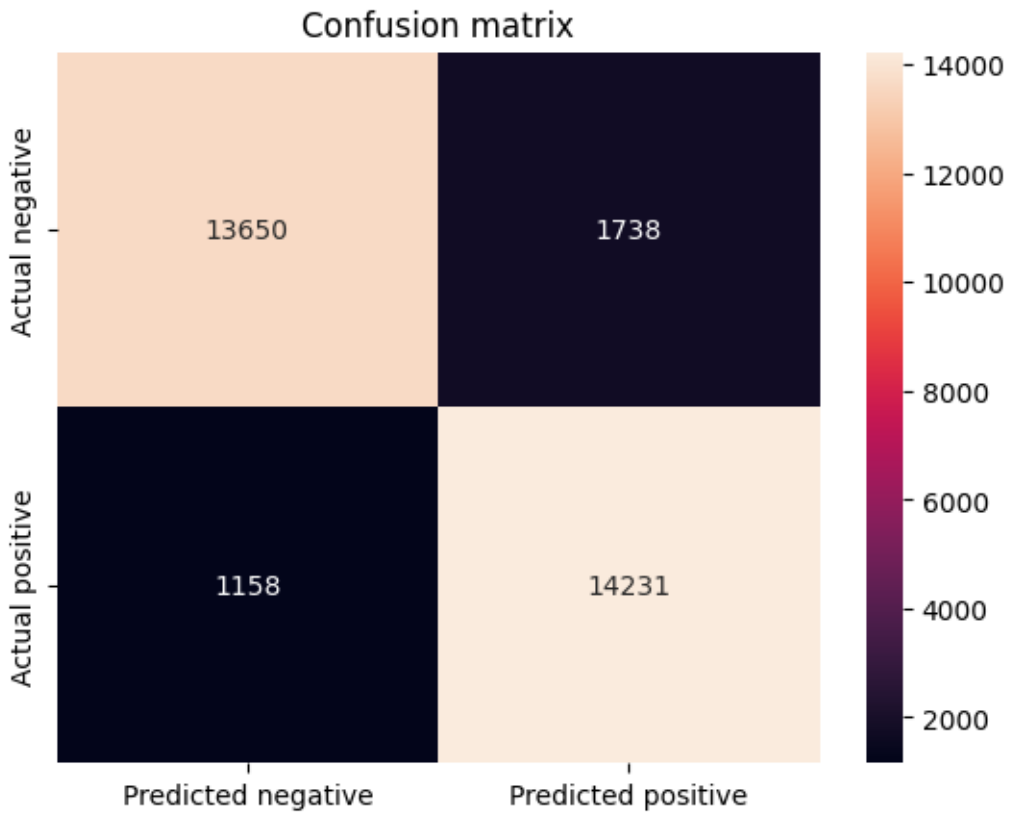


[140]:
```
##for 2
knn_clf.fit(X2, y2)
evaluation_metric(knn_clf, X2, y2, XX2, yy2)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9062766670661597
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9039867532959727
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0022899137701870576
```

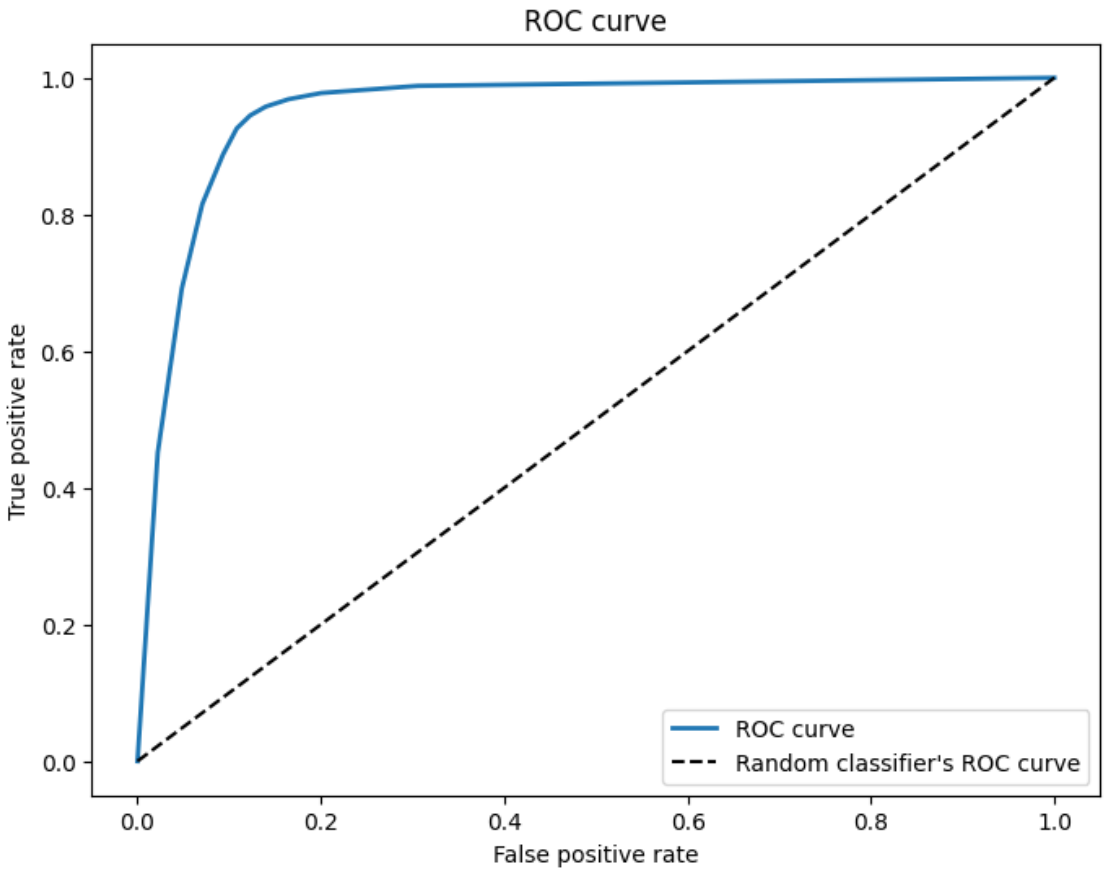



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9501646753511226
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9059037593007766
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8870548479334547
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9247514458379362
--------------------------------------------------------------------------------
--------------------
```

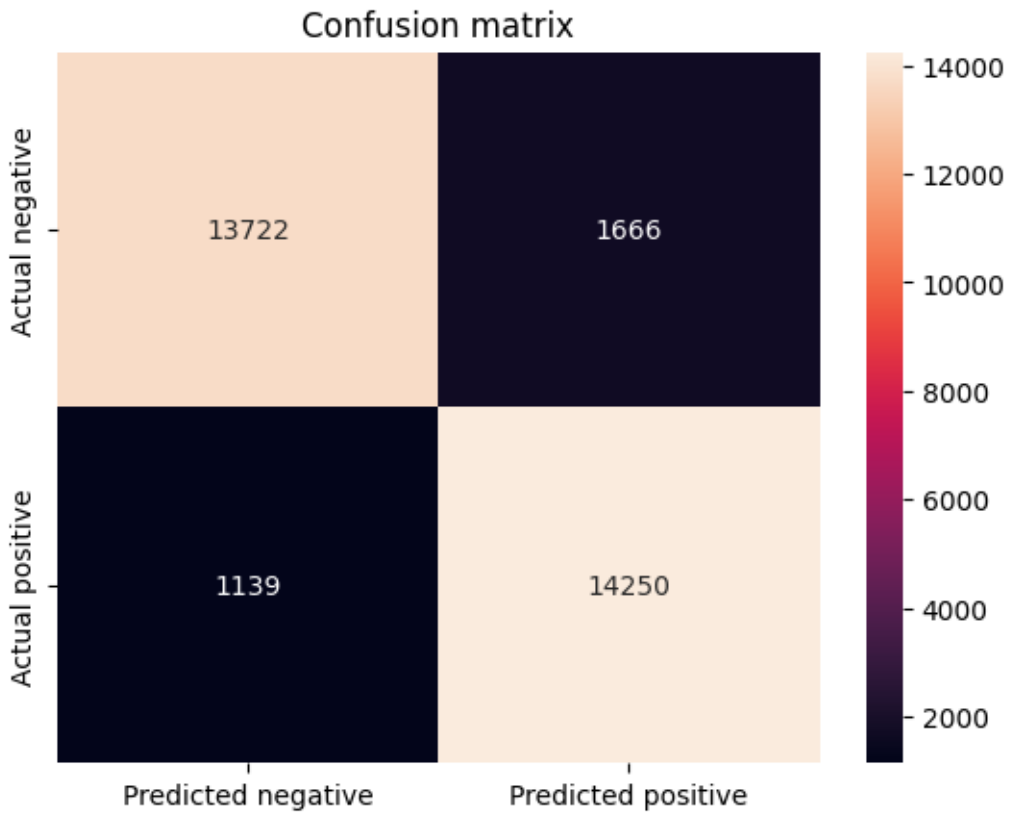


[141]:
```
##for 3
knn_clf.fit(X3, y3)
evaluation_metric(knn_clf, X3, y3, XX3, yy3)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9071458269595976
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9069435985788402
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.00020222838075734106
```

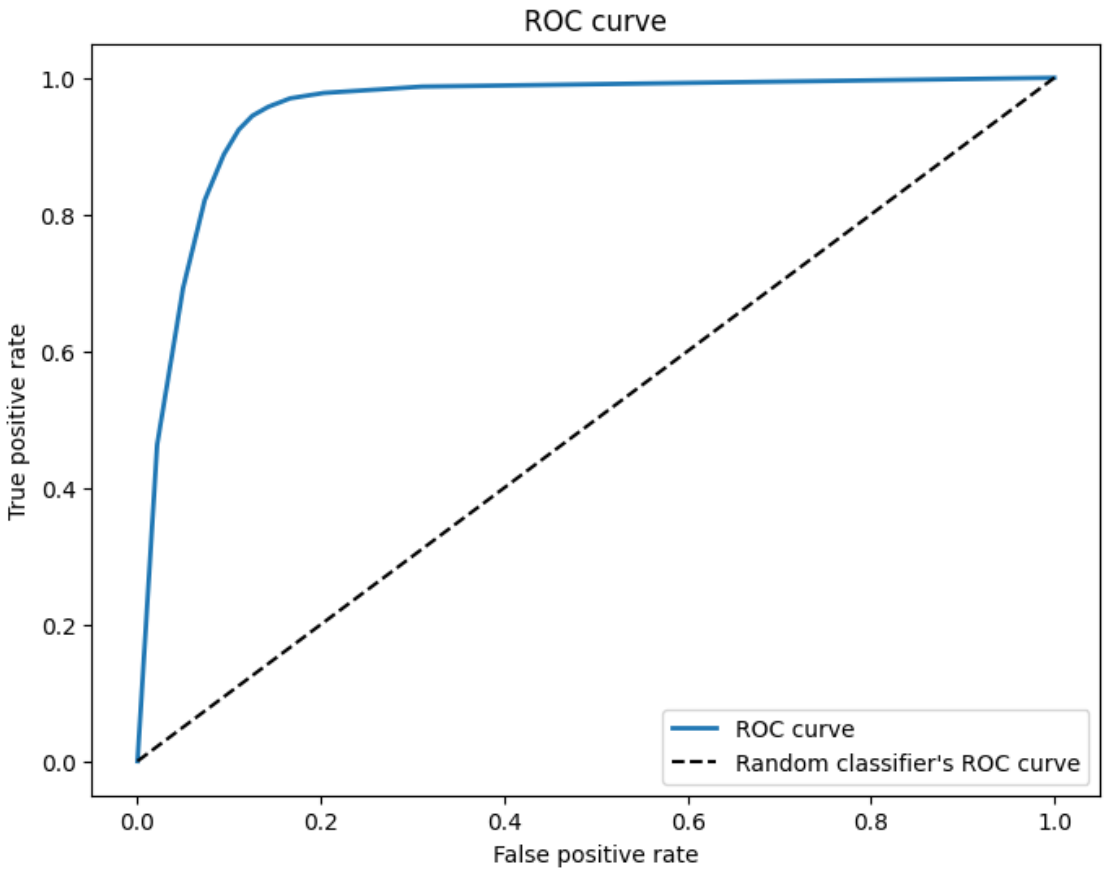



```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9518469748468971
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.908860512720538
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8917338185599168
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9259860939632205
--------------------------------------------------------------------------------
--------------------
```

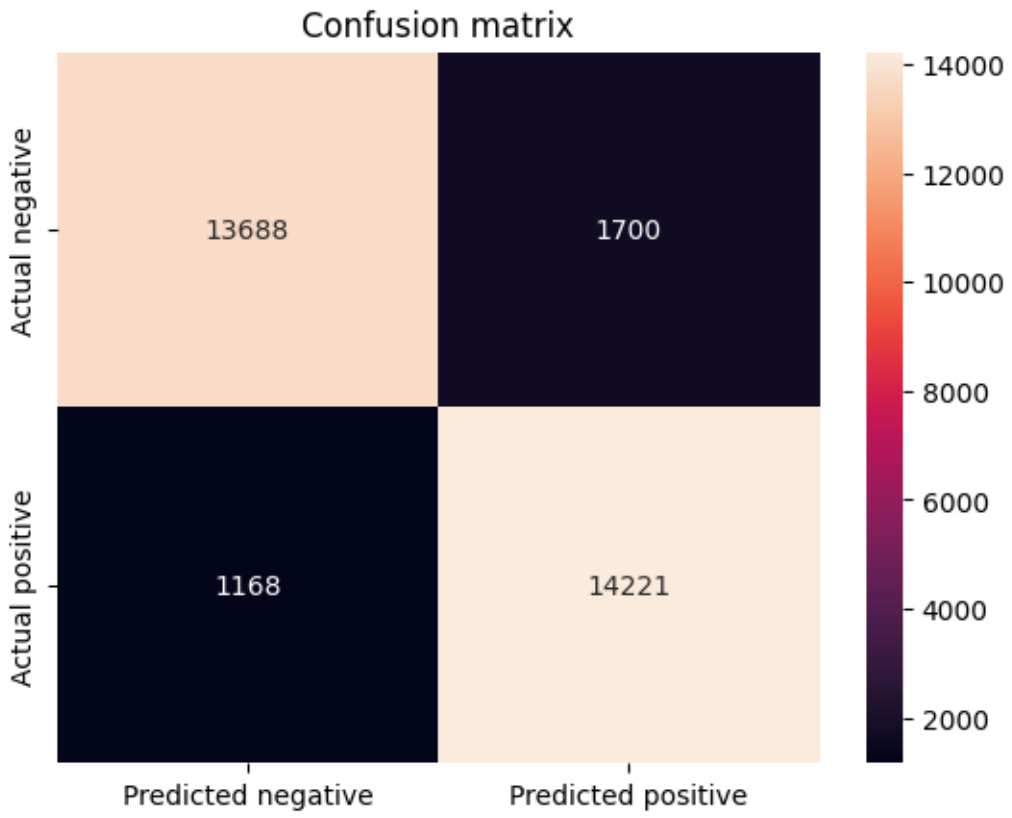


[142]:
```
##for 4
knn_clf.fit(X4, y4)
evaluation_metric(knn_clf, X4, y4, XX4, yy4)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9061791658793676
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9032067992287184
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0029723666506492608
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9512854644198693
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9068135295837801
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8895243046529764
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.924101631035155
--------------------------------------------------------------------------------
--------------------
```



Observe:

- Practically, the same performance as for logistic regression

7.6 MultiLayer Perceptron

Let us train an MLP model to see if it performs better than the previous ones

[143]:
```
from sklearn.neural_network import MLPClassifier
```

[144]:
```
##Hyperparameter search with BayesSearchCV

opt_mlp = BayesSearchCV(
        MLPClassifier(),
        {
        'activation': ['relu', 'tanh', 'logistic'],
         'solver': ['adam', 'sgd'],
        'alpha': [0.001, 0.01, 0.1],
        'learning_rate': ['constant',  'adaptive']},
        n_iter=3,
        random_state=1923)
```

```
##let's fit each mlp model to the training data
opt_mlp.fit(X, y)

##let's print the best score and the best parameters
print("val. score: %s" % opt_mlp.best_score_)
print("Best parameters: ", opt_mlp.best_params_)
```

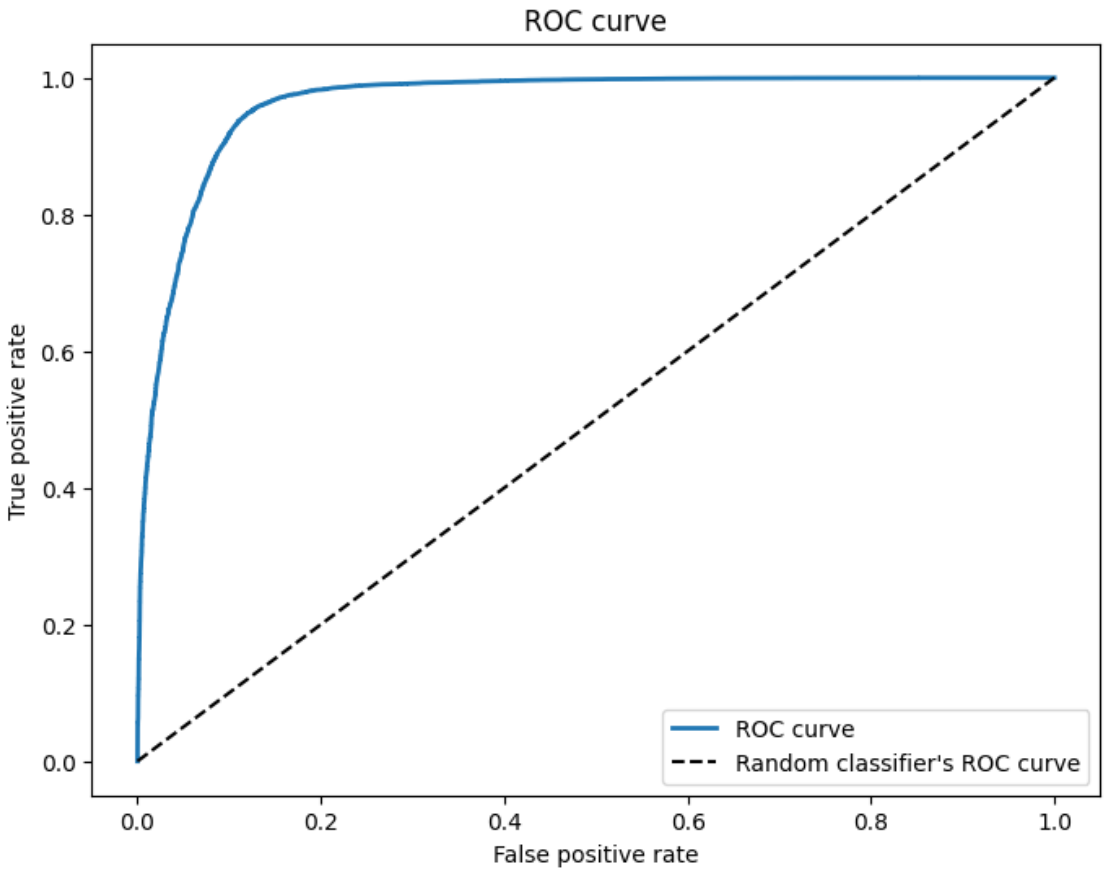
```
val. score: 0.950047683761295
Best parameters:  OrderedDict([('activation', 'relu'), ('alpha', 0.01),
('learning_rate', 'constant'), ('solver', 'adam')])
```

Let's use these hyperparams to instantiate, train and evaluate the model

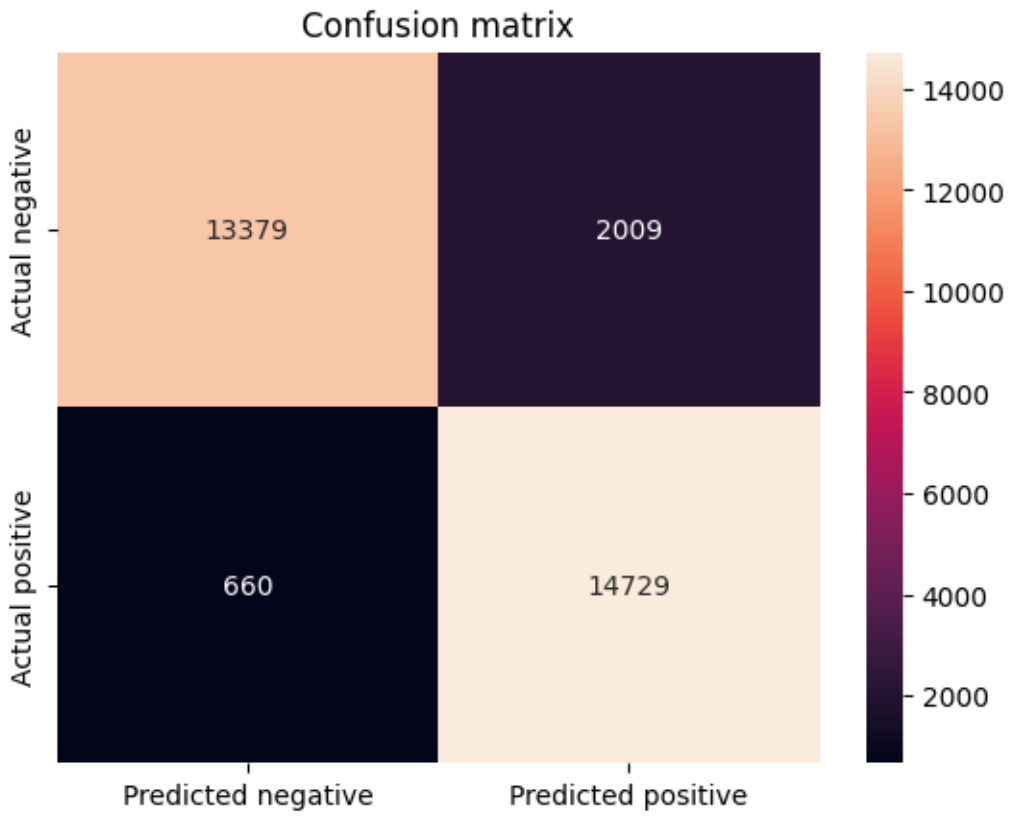
[145]:
```
##instantiating the model with the best parameters found by the hyperparameter␣
 ↪search
mlp_clf = MLPClassifier(activation= 'relu', alpha= 0.01, learning_rate=␣
 ↪'constant', solver= 'adam', random_state=1923)
```

[146]:
```
##fit and evaluate the model
mlp_clf.fit(X0, y0)
evaluation_metric(mlp_clf, X0, y0, XX0, yy0)
```

```
------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9137092287068052

/opt/homebrew/lib/python3.11/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:686:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(

------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.910907306325808
------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0028019223809971905
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9638422085642686
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9132793969522696
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8694437223810761
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9571122230164403
--------------------------------------------------------------------------------
--------------------
```



Observe:

- The TPR is the highest at 95.71%
- This also applies to the 96% AUC score
- Accuracy is slightly higher than other models at 91%

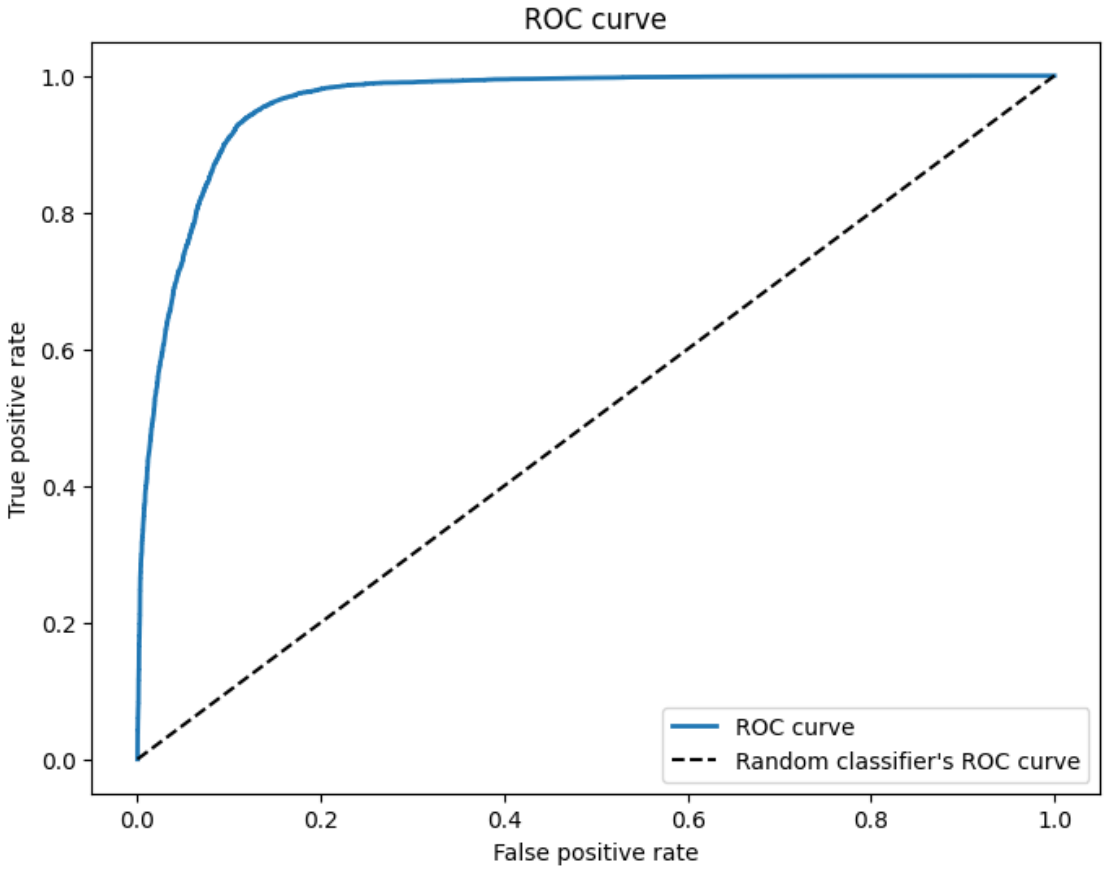
7.7 AdaBoost Classifier

As it was noticed that the RandomForest Classifier showed promising results, let us try to to fit an AdaBoost Classifier. For this example, I won't be looking at tuning the hyperparameters with BayesSearchCV or with GridSearchCV, as the model simply boosts a number of previous selected estimators by making use of the residuals of the previous model. In this case, the AdaBoost model uses 300 RandomForest Classifiers to gradually refine its predictions, paying attention to misclassified instances with each iteration.

[147]:
```
from sklearn.ensemble import AdaBoostClassifier
##let us boost the random forest classifier model with adaboost
##this creates an instance of the adaboost that uses the previous random forest␣
 ↪classifier as the base estimator
ada_clf = AdaBoostClassifier(estimator=RandomForestClassifier(max_depth= 5,␣
 ↪max_features='log2', n_estimators= 100, random_state=42), n_estimators=300,␣
 ↪learning_rate=0.01,  random_state=42)
```

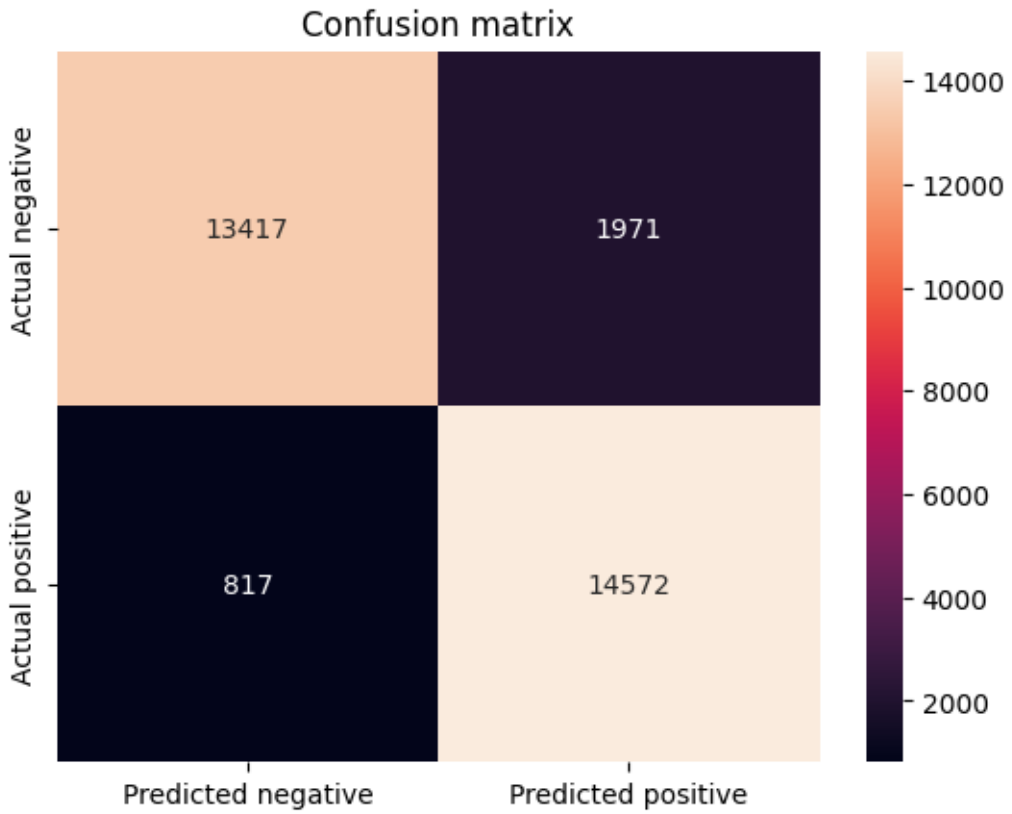
[148]:
```
##fit and evaluate the model
ada_clf.fit(X0, y0)
evaluation_metric(ada_clf, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9095340008084415
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the test set is: 0.9100299655478015
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: -0.0004959647393599775
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9616987234086686
--------------------------------------------------------------------------------
--------------------
```

```
The accuracy score is: 0.9094128732495045
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8719131791005978
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9469101306127754
--------------------------------------------------------------------------------
--------------------
```



Observe:

- Despite taking relatively longer to run, the scores offered by the AdaBoost Model are inferior to the MultiLayered Perceptron model
- The true positive rate os quite high at 94.69% and accuracy is comparable with most other models
- Given that this takes a long time to run, it is unlikely it will offer a signfificant performance boost on the other subsamples

8 6. Voting Classifier

In this section, I will attempt an alternative way of ensemble learning, which is the voting classifier. The aim is to combine several "weaker" learners to see if I can obtain a significantly stronger learner. To maximise the performance of the voting classifiers, I will select the top 3 performing models: MLP, RandomForest and NearestNeighbours. There are two main options forvoting classifiers : (i) a hard voting one (majority wins) and a soft voting classifier (combines probabilities). As hard voting has no predict_proba method, the evaluation metric I chose will not be able to evaluate the recall, which is crucial for this model. Hence, I will use a soft voting approach

8.1 Soft Voting

[149]:
```
from sklearn.ensemble import VotingClassifier
```

[150]:
```
##create instance of voting classifier
##creates an instance of the voting classifier with soft voting
voting_soft = VotingClassifier(estimators=[
                                           ('rf', rf_clf),
                                          ('mlp', mlp_clf),
                                           ('nn', knn_clf),
                                           ],
                                           voting='soft')
```

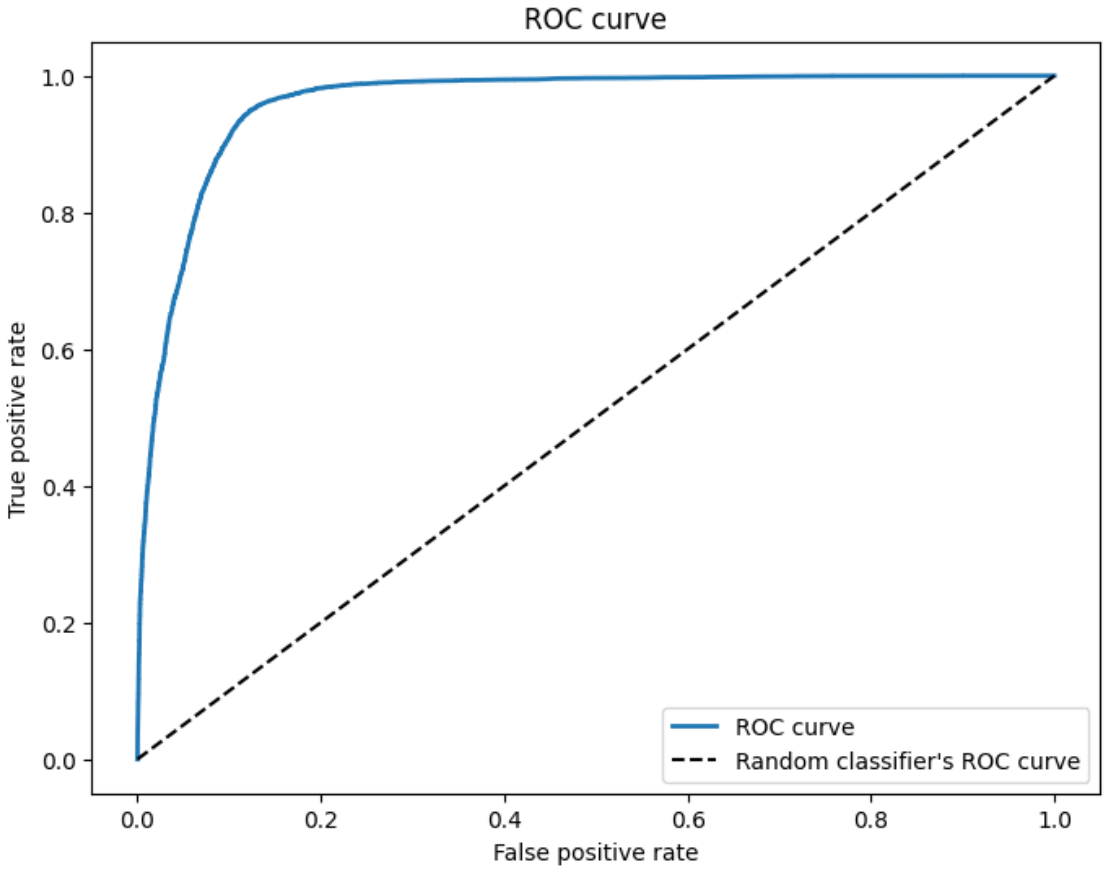
[151]:
```
voting_soft.fit(X0, y0)
```

[151]:
```
VotingClassifier(estimators=[('rf',
                              RandomForestClassifier(max_depth=5,
                                                     max_features='log2',
                                                     random_state=1923)),
                             ('mlp',
                              MLPClassifier(alpha=0.01, random_state=1923)),
                             ('nn',
                              KNeighborsClassifier(metric='euclidean',
                                                   n_neighbors=10))],
                 voting='soft')
```

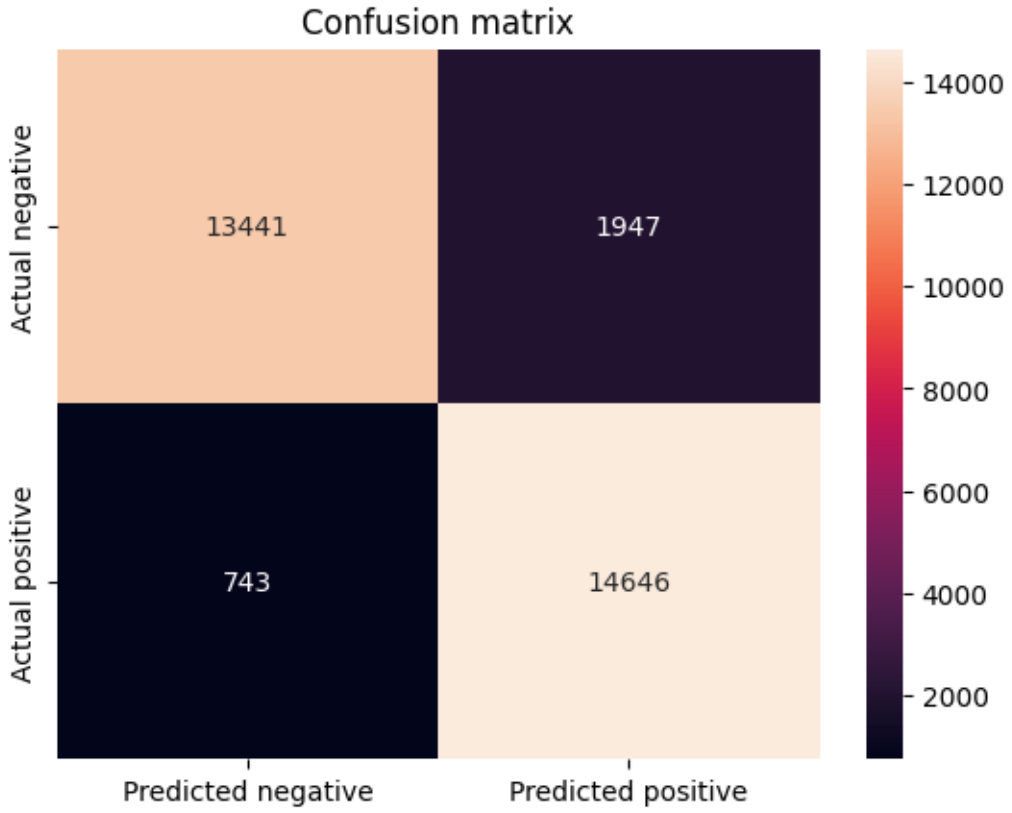
[152]:
```
evaluation_metric(voting_soft, X0, y0, XX0, yy0)
```

```
--------------------------------------------------------------------------------
--------------------
The mean cross validation score on the training set is: 0.9124176651741976

/opt/homebrew/lib/python3.11/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:686:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(

--------------------------------------------------------------------------------
```

```
--------------------
The mean cross validation score on the test set is: 0.9113621682451759
--------------------------------------------------------------------------------
--------------------
The difference between the mean cross validation scores on the training and test
data: 0.0010554969290217375
```




```
--------------------------------------------------------------------------------
--------------------
The AUC score is: 0.9610314343814664
--------------------------------------------------------------------------------
--------------------
The accuracy score is: 0.9125970692400169
--------------------------------------------------------------------------------
--------------------
The true negative rate (TNR) is: 0.8734728359760853
--------------------------------------------------------------------------------
--------------------
The true positive rate (TPR) is: 0.9517187601533563
--------------------------------------------------------------------------------
```



Observe:

- The soft voting classifier is actually inferior to the MLP, which alone is a better classifier
- This indicates that the MLP is the best classifier

9 7. Generalization Errors of Best Model

As the MLP neural network is clearly the superior model, it is worth understand if it can be made even better. So, let us test it onto the real test dataset, that contains imbalanced data and see how well it does the job at detecting anomalies.

9.1 Get Final Predictions

[153]:
```
final_predictions = mlp_clf.predict(XX)
```

[154]:
```
##these are the final predictions on the test set
final_predictions
```

[154]:
```
array([0, 0, 0, …, 0, 1, 1])
```

[155]:
```
yy
```

[155]:
```
839287    0
366635    0
9199      0
716029    0
538602    0
         ..
320942    0
347670    0
481257    0
390328    0
652881    0
Name: Higher_Risk, Length: 209715, dtype: int64
```

9.2 MAE

[156]:
```
from scipy import stats
```

[157]:
```
##let's set the confidence level to 95%
confidence = 0.95
```

[158]:
```
##this is simply a vector of sqaured errors
sqr_errors = (final_predictions - yy)**2
```

[159]:
```
##this is a vector containing the absolute errors
errors = np.abs(final_predictions - yy)
```

Let's calculate the confidence interval for the mean error

[160]:
```
(stats.t.interval(confidence, len(errors)-1, loc=errors.mean(), scale=stats.
 ↪sem(errors)))
```

[160]:
```
(0.1432049605752543, 0.14621639698142977)
```

And the root mean squared error:

[161]:
```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(yy, final_predictions)
```

[162]:
```
mae
```

[162]:
```
0.14471067877834204
```

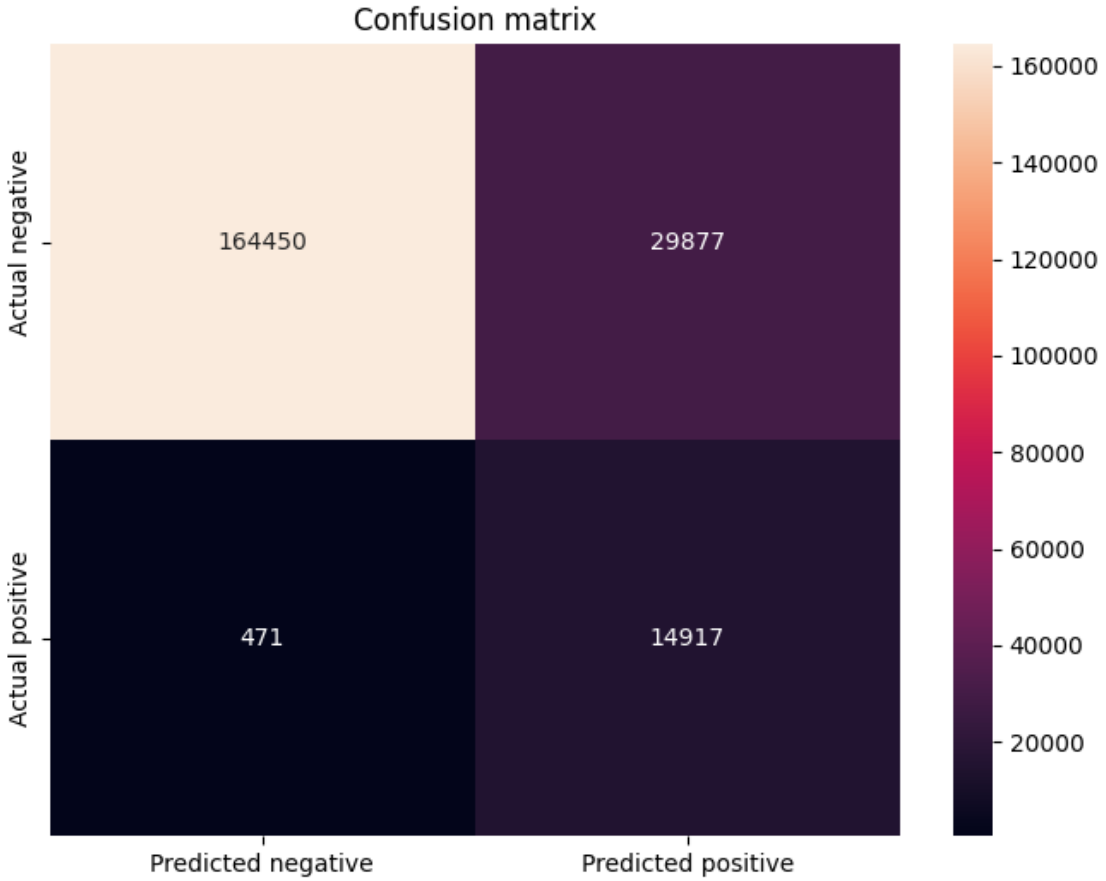
The resulting mean absolute error is 0.1447. Given that we predict 1's (Higher_Risk) and 0's (Lower_Risks) the scale is 0 to 1, so the MAE is 14.47% of the predictions. This MAE means that estimated probabilities are 14.47% away from the actual probabilities. To undersand how precise this MAE is, I used the 95% confidence interval from scipy stats, which shows that the mean squared error can be anywhere from 0.1432 to 0.1462. This shows that the precision is about 2% for the

mean absolute error. However, this precision only refers to the MAE itself, the actual probabilities for the patient being at high risk should be considered within a range of 14.47% probability.

9.3 Confusion Matrix

Let's also look at the confusion matrix for these final predictions with the MLP model

[163]:
```
conf_mat = confusion_matrix (yy, final_predictions)
cm_df = pd.DataFrame(conf_mat, columns=["Predicted negative", "Predicted␣
 ↪positive"], index=["Actual negative", "Actual positive"])
##plot the confusion matrix via seaborn

plt.figure(figsize=(8, 6))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title("Confusion matrix")
plt.show()
```



[164]:
```
yy.value_counts()
```

[164]:
```
Higher_Risk
0    194327
1     15388
Name: count, dtype: int64
```

[168]:
```
len(yy)
```

[168]:
```
209715
```

[170]:
```
print("Imbalance in the test set", yy.value_counts()[1]/yy.
↪value_counts()[0]*100)
```

```
Imbalance in the test set 7.918611412721856
```

As the test data on which we test the MLP is taken from the covid original data, it has 209715 observations and the classes are imbalanced with with just over 7.9% of all cases being high risk. From the confusion matrix, the MLP model did quite well. Let's summarise the main points:

- There are 14917 TP (true positives). This is the number of correct classified higher_risk covid patients. The false negatives (FN) represent the higher risk patients that were misclassified : 471. From the values in the confusion matrix, we can get the true positive rate with the usual formula : $TPR = TP/(TP + FN)$. If we insert the values from this confusion matrix the TPR turns out to be 96.93% which gives a pretty confidence in these predictions. The TPR can be interpreted as the probability of a positive prediction to be true. In this case, the model probability for each positive prediction to be correct is 96.93%.
- However, the model misclassifes many negative cases : 29877. This means that we get many more patients classified as high risk than they actually are
- The true negative rate TNR can be calculated similarly to understand how confident we should be in the values of the majority class (lower risk). TNR for this model is 84.62%, however, this is not a big problem given that the model gives high confidence in higher risk predictions (96%). Hence, the models correctly classifies almost all patients at high risk appropiately, the TNR simply suggests that there are many patients that are lower risk that will most likely be classified as high risk (about 30000). Since getting more treatment than necessary is not harmful, the False Positives do not present a major problem.

This model's strength is that is is very confidence in making predictions of the positive class, which is the minority class. Given that the ratio between minority class and majority class is close to 8:100, the model does a great job in offering high confidence predictions for the minority class (covid high risk patients). This is very important, as the goal of the project was to find such a model, that predicts the minority class very well, to avoid misclassifying patients that are at high risk.

9.4 Accuracy

Let's also have a look at the accuracy of the model, which shows how many predictions (percetange wise) are correct. This metric is complementary to the TPR which is the most important for this case (to predict well the minority class). However, accuracy can be useful too in understanding how the model performs. Since the test set contains highly imbalanced classes, we should get accuracy over 92% for the model to performs well by this metric. The reason for this is that around 92%

of the data are 0's, so if a model would predict 0 all the time, it would be right 92% of the times. However, a model that predicts 0 all the time, is clearly not a good model.

[181]:
```
##calculate accuracy directly from Scikit-Learn's accuracy_score method
accuracy_score(yy, final_predictions)
```

[181]:
```
0.855289321221658
```

The accuracy is clearly not very good, but as said before, the more important metric is TPR which is over 96%. To test the calculation above, it is simply to calculate the accuracy directly from the confusion matrix. It is the sum of true values (main diagonal) over all values.

[179]:
```
true_predictions = conf_mat[0][0] + conf_mat[1][1]
print ("The number of true predictions is", true_predictions)
```

```
The number of true predictions is 179367
```

[180]:
```
accuracy_hand = true_predictions/len(yy)
print ("The accuracy calculated by hand is", accuracy_hand)
```

```
The accuracy calculated by hand is 0.855289321221658
```

Clearly this calculation seems to be correct. Hence, we have a model that predicts correctly 85% of the time, but when it predicts a high risk covid patient, there is over 96% probability that it's prediction are correct. Tthe confidence in the negative predictions are lower, with a true negative rate of 84.6%

10 Launch

Now that we have our best model, let us save it for further use. I will use joblib to do it.

[111]:
```
import joblib
```

[183]:
```
joblib.dump(mlp_clf, "MultiLayer_Perceptron_Classifier_Covid .pkl")
```

[183]:
```
['MultiLayer_Perceptron_Classifier_Covid .pkl']
```

For further use, let us also save the preprocessing pipeline

[109]:
```
preprocessing
```

[109]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('bucket_pipeline_1',
                                 Pipeline(steps=[('kbinsdiscretizer',
KBinsDiscretizer(encode='ordinal',
                                                                  n_bins=2,
random_state=1923,
strategy='kmeans')),
                                                 ('onehotencoder',
OneHotEncoder(handle_unknown='ignore'))]),
```

```
                                    ['MEDICAL_UNIT']),
                                   ('categorical',
                                    Pipeline(steps=[('customtransformer',
                                                     CustomTransformer()),
                                                    ('oneh…
                                     'CARDIOVASCULAR', 'OBESITY', 'RENAL_CHRONIC',
                                     'TOBACCO', 'ICU', 'COVID_POSITIVE',
                                     'COVID_INCONCLUSIVE']),
                                   ('numerical',
                                    Pipeline(steps=[('standardscaler',
                                                     StandardScaler())]),
                                    ['AGE']),
                                   ('categorical simple',
                                    Pipeline(steps=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore'))]),
                                    ['SEX', 'PATIENT_TYPE']),
                                   ('drop_age', 'drop',
                                    ['CLASIFFICATION_FINAL', 'DATE_DIED',
                                     'Higher_Risk', 'Lower_Risk'])])
```

[112]:
```
joblib.dump(preprocessing, "preprocessing.pkl")
```

[112]:
```
['preprocessing.pkl']
```

[ ]:
```

```