PREDICTING H1N1 FLU VACCINATION STATUS USING MACHINE LEARNING

1. Business Understanding

1.1 Overview

In this project, the aim was to use data from the National Flu Survey (NHFS 2009) to predict whether respondents received the H1N1 vaccine. Understanding past vaccination trends is crucial for interpreting patterns in more recent pandemics, such as COVID-19. Key factors influencing vaccination status include Doctor recommendations for the H1N1 vaccine,health insurance, opinions on the vaccine's effectiveness and perceptions of the risk posed by H1N1. I employed six machine learning models for prediction:

1.Decision Tree Classifier

2.Logistic Regression

3.Random Forest

4.K-Nearest Neighbors Classifier

5.Gradient Boosting Classifier

6.XGBoost Classifier

Among these, the Gradient Boosting Classifier achieved the highest accuracy and precision.

1.2 Business Problem.

Vaccination stands as one of the most effective public health interventions ever implemented, leading to the elimination and control of diseases that were once widespread globally. Despite substantial medical evidence and the strong consensus among healthcare professionals supporting vaccination, skepticism has increased in many countries in recent years. This troubling trend has resulted in decreased immunization coverage, with several outbreaks of infectious diseases linked to undervaccinated communities. The growing issue of vaccine hesitancy has become so pervasive that it is now the subject of numerous studies aiming to understand the sources and correlations of attitudes toward vaccination.

This study aims to predict the likelihood of individuals receiving the H1N1 flu vaccine. We believe the predictive models and analyses from this study will provide public health professionals and policymakers with a clear understanding of the factors associated with low vaccination rates. This, in turn, will enable them to systematically address the barriers preventing people from getting vaccinated.

The methodologies employed in these models can serve as a reference for future work and can be compared with other models for performance evaluation. To accurately classify those who received the H1N1 flu shot from those who did not, we require models with high accuracy and high precision, which corresponds to a low false positive rate (those mistakenly identified as vaccinated when they were not). This will be further evaluated using the ROC curve, accuracy score, precision score, and confusion matrix.

Target Audience: Public health officials of the American Public Health Association (APHA)

OBJECTIVES:

1. Predicting who is vaccinated or not accurately.(Deliverable: Model)
2. Analyse the factors that influence people to get H1N1 vaccine or not. (Deliverable: Analysis)

Context:

- False negative: Saying people did not get the vaccine when they actually did.
- Outcome: Not a big problem
- False positive: Saying people got the vaccine when they actually did not.
- Outcome: Big problem

Evaluation: We will focus on accuracy, f1, and precision scores for our model iterations in order to minimize False Positives, because in our business context false positives are a much more costly mistake than false negatives.

- **Accuracy**
- **Precision**
- Recall
- **F1-Score**

2. Data Understanding

2.1 Importing the necessary libraries and exploring the data

In [ ]:
```
# Common libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Libraries for model training
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsSca
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import category_encoders as ce
from sklearn.model_selection import train_test_split, GridSearchCV, cross

# Libraries for algorithm

from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClas
from sklearn.neighbors import KNeighborsClassifier
import xgboost     # extreme gradient boosting


# Libraries for testing
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_scor
    ConfusionMatrixDisplay,confusion_matrix
)

# Removing warnings
import warnings
warnings.filterwarnings('ignore')

# Storing plots

%matplotlib inline

# To visualize the 100 many the columns in data
pd.options.display.max_columns=100
```

2.2 Load Dataset

This data comes from a NHFS National Flu Survey from 2009, which inquires about whether or not people received the seasonal flu and/or the H1N1 flu vaccination, as well as their demographic, behavioral, and health factors.

In [ ]:
```
# Reading in the data and previewing the dataset
Data1 = pd.read_csv('DATA/H1N1_Flu_Vaccines.csv')
Data1.head(5)
```

Out[ ]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavior
0	0	1.0	0.0	0.0	
1	1	3.0	2.0	0.0	
2	2	1.0	1.0	0.0	
3	3	1.0	1.0	0.0	
4	4	2.0	1.0	0.0	

In [ ]:
```
#dataset tail
Data1.tail(3)
```

Out[ ]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	beh
26704	26704	2.0	2.0	0.0	
26705	26705	1.0	1.0	0.0	
26706	26706	0.0	0.0	0.0	

2.3 Checking the Dataset

In [ ]:
```
#Determining the no. of records in our dataset
Data1.shape
```

Out[ ]:
```
(26707, 38)
```

In [ ]:
```
# Exploring the percentage breakdown of the two classes in one possible t
Data1['seasonal_vaccine'].value_counts(normalize=True)
```

Out[ ]:
```
seasonal_vaccine
0    0.534392
1    0.465608
Name: proportion, dtype: float64
```

In [ ]:
```
# Exploring the percentage breakdown of the two classes in one possible t
Data1['h1n1_vaccine'].value_counts(normalize=True)    # class imbalance pr
```

Out[ ]:
```
h1n1_vaccine
0    0.787546
1    0.212454
Name: proportion, dtype: float64
```

In [ ]:
```
# checking dataset information
Data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26707 entries, 0 to 26706
Data columns (total 38 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   respondent_id                26707 non-null  int64  
 1   h1n1_concern                 26615 non-null  float64
 2   h1n1_knowledge               26591 non-null  float64
 3   behavioral_antiviral_meds    26636 non-null  float64
 4   behavioral_avoidance         26499 non-null  float64
 5   behavioral_face_mask         26688 non-null  float64
 6   behavioral_wash_hands        26665 non-null  float64
 7   behavioral_large_gatherings  26620 non-null  float64
 8   behavioral_outside_home      26625 non-null  float64
 9   behavioral_touch_face        26579 non-null  float64
 10  doctor_recc_h1n1             24547 non-null  float64
 11  doctor_recc_seasonal         24547 non-null  float64
 12  chronic_med_condition        25736 non-null  float64
 13  child_under_6_months         25887 non-null  float64
 14  health_worker                25903 non-null  float64
 15  health_insurance             14433 non-null  float64
 16  opinion_h1n1_vacc_effective  26316 non-null  float64
 17  opinion_h1n1_risk            26319 non-null  float64
 18  opinion_h1n1_sick_from_vacc  26312 non-null  float64
 19  opinion_seas_vacc_effective  26245 non-null  float64
 20  opinion_seas_risk            26193 non-null  float64
 21  opinion_seas_sick_from_vacc  26170 non-null  float64
 22  age_group                    26707 non-null  object 
 23  education                    25300 non-null  object 
 24  race                         26707 non-null  object 
 25  sex                          26707 non-null  object 
 26  income_poverty               22284 non-null  object 
 27  marital_status               25299 non-null  object 
 28  rent_or_own                  24665 non-null  object 
 29  employment_status            25244 non-null  object 
 30  hhs_geo_region               26707 non-null  object 
 31  census_msa                   26707 non-null  object 
 32  household_adults             26458 non-null  float64
 33  household_children           26458 non-null  float64
 34  employment_industry          13377 non-null  object 
 35  employment_occupation        13237 non-null  object 
 36  h1n1_vaccine                 26707 non-null  int64  
 37  seasonal_vaccine             26707 non-null  int64  
dtypes: float64(23), int64(3), object(12)
memory usage: 7.7+ MB
```

In [ ]:
```
Data1.dtypes
```

Out[ ]:
```
respondent_id                    int64
h1n1_concern                   float64
h1n1_knowledge                 float64
behavioral_antiviral_meds      float64
behavioral_avoidance           float64
behavioral_face_mask           float64
behavioral_wash_hands          float64
behavioral_large_gatherings    float64
behavioral_outside_home        float64
behavioral_touch_face          float64
doctor_recc_h1n1               float64
doctor_recc_seasonal           float64
chronic_med_condition          float64
child_under_6_months           float64
health_worker                  float64
health_insurance               float64
opinion_h1n1_vacc_effective    float64
opinion_h1n1_risk              float64
opinion_h1n1_sick_from_vacc    float64
opinion_seas_vacc_effective    float64
opinion_seas_risk              float64
opinion_seas_sick_from_vacc    float64
age_group                       object
education                       object
race                            object
sex                             object
income_poverty                  object
marital_status                  object
rent_or_own                     object
employment_status               object
hhs_geo_region                  object
census_msa                      object
household_adults               float64
household_children             float64
employment_industry             object
employment_occupation           object
h1n1_vaccine                     int64
seasonal_vaccine                 int64
dtype: object
```

In [ ]:
```
# Getting number of null values.
Data1.isna().sum()
```

Out[ ]:
```
respondent_id                      0
h1n1_concern                      92
h1n1_knowledge                   116
behavioral_antiviral_meds         71
behavioral_avoidance             208
behavioral_face_mask              19
behavioral_wash_hands             42
behavioral_large_gatherings       87
behavioral_outside_home           82
behavioral_touch_face            128
doctor_recc_h1n1                2160
doctor_recc_seasonal            2160
chronic_med_condition            971
child_under_6_months             820
health_worker                    804
health_insurance               12274
opinion_h1n1_vacc_effective      391
opinion_h1n1_risk                388
opinion_h1n1_sick_from_vacc      395
opinion_seas_vacc_effective      462
opinion_seas_risk                514
opinion_seas_sick_from_vacc      537
age_group                          0
education                       1407
race                               0
sex                                0
income_poverty                  4423
marital_status                  1408
rent_or_own                     2042
employment_status               1463
hhs_geo_region                     0
census_msa                         0
household_adults                 249
household_children               249
employment_industry            13330
employment_occupation          13470
h1n1_vaccine                       0
seasonal_vaccine                   0
dtype: int64
```

In [ ]:
```
Data1.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# Explore numerical columns
Data1.describe()
```

Out[ ]:

	respondent_id	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	beha
count	26707.000000	26615.000000	26591.000000	26636.000000	
mean	13353.000000	1.618486	1.262532	0.048844	
std	7709.791156	0.910311	0.618149	0.215545	
min	0.000000	0.000000	0.000000	0.000000	
25%	6676.500000	1.000000	1.000000	0.000000	
50%	13353.000000	2.000000	1.000000	0.000000	
75%	20029.500000	2.000000	2.000000	0.000000	
max	26706.000000	3.000000	2.000000	1.000000	

In [ ]:

```
# Explore object columns
Data1[[c for c in Data1.columns if Data1[c].dtype =='object']].describe()
```

Out[ ]:

	age_group	education	race	sex	income_poverty	marital_status	rent_o
count	26707	25300	26707	26707	22284	25299	
unique	5	4	4	2	3	2	
top	65+ Years	College Graduate	White	Female	<= $75,000, Above Poverty	Married	
freq	6843	10097	21222	15858	12777	13555	

Observations

Upon initial exploration of the data, we've made several key observations:

1. There are 26,000 respondents to this survey.
2. There are 36 features.
3. There are lots of missing value so we need to impute them.
4. There no duplicates in dataset.

Further preprocessing is required to understand the relationships between different features.

I decided to choose the H1N1 vaccination rate as our target variable, because so many of the features are related to H1N1 vaccination.

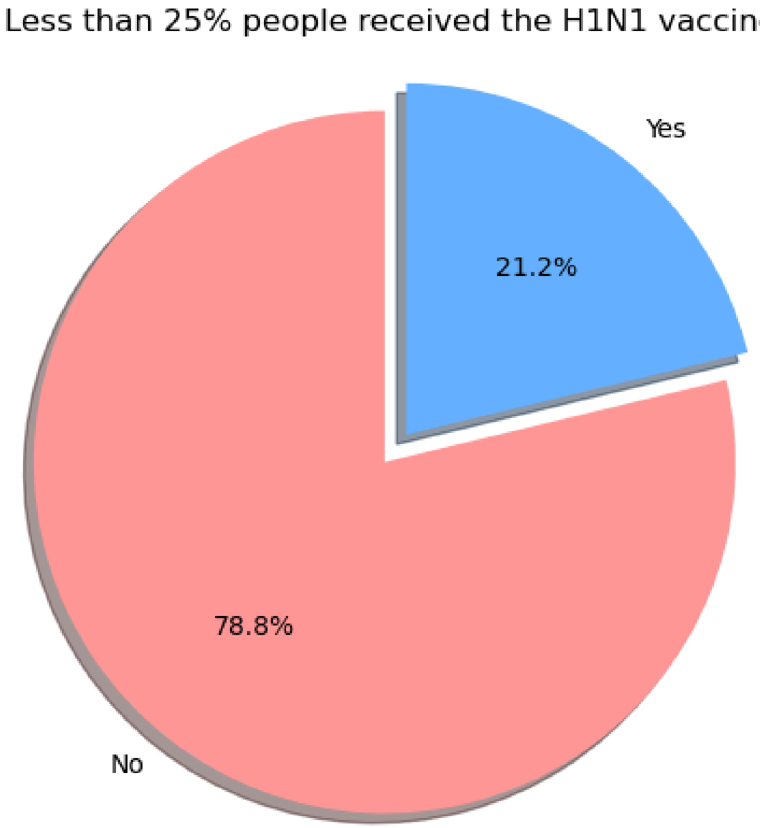
3. EXPLORATORY DATA ANALYSIS(EDA)

How many people got the H1N1 vaccine - from the given data set

In [ ]:

```
## for the H1N1 vaccine
fig1, ax1 = plt.subplots()
```

```
labels = ['No','Yes']
explode = (0,0.1)
colors = ['#ff9999','#66b3ff']
ax1.pie(Data1['h1n1_vaccine'].value_counts(), explode=explode, labels=lab
        shadow=True, startangle=90)
plt.title("Less than 25% people received the H1N1 vaccine")
ax1.axis('equal')
plt.tight_layout()
plt.savefig("H1n1pie")
plt.show()
```



Observation - There is a class imbalance in the adoption of the H1N1 vaccine. (less than 25% of the people chose to receive the H1N1 vaccine). This might help in future during the analysis.This class imbalance problem is what we want to deal with in this project.

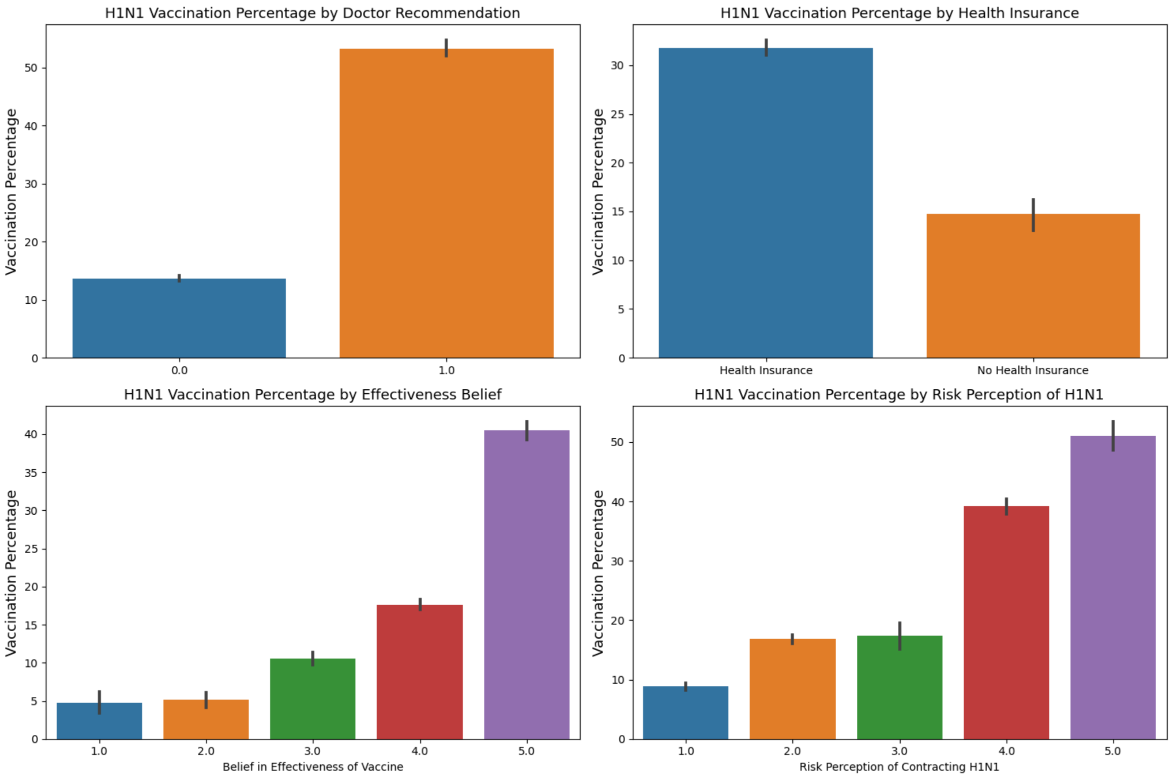
The following visualizations represent the top four most influential features in determining vaccination status for H1N1.

In [ ]:
```
# Making a copy of main dataframe to use for visualizations
Data2 = Data1.copy()
```

In [ ]:
```
# Creating dictionary for mapping in order to create better names for x a
ins_dict = {1: 'Health Insurance', 0: 'No Health Insurance'}
# Creating the column that will be used to create clear x axis tick marks
Data2['health_ins_words'] = Data2['health_insurance'].replace(ins_dict)
```

In [ ]:
```
def plot_bar(df, x_col, y_col, ax, xlabel, ylabel, title):
    """
    This function plots a bar chart on a given axis.
    """
    sns.barplot(x=df[x_col].dropna(), y=df[y_col]*100, ax=ax)
```

```
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel, fontsize=13)
    ax.set_title(title, fontsize=13)

def plot_vaccination_graphs(Data2):
    """
    This function takes a dataframe and plots four bar charts in a 2x2 la
    showing the relationship between H1N1 vaccination and various factors
    """
    # Making a copy of the main dataframe to use for visualizations
    Data2 = Data1.copy()

    # Creating dictionary for mapping in order to create better names for
    ins_dict = {1: 'Health Insurance', 0: 'No Health Insurance'}
    # Creating the column that will be used to create clear x axis tick m
    Data2['health_ins_words'] = Data2['health_insurance'].replace(ins_dic

    # Setting up the 2x2 subplot layout
    fig, axs = plt.subplots(2, 2, figsize=(15, 10))

    # Plotting the individual bar charts
    plot_bar(Data2, 'doctor_recc_h1n1', 'h1n1_vaccine', axs[0, 0], '', 'V
    plot_bar(Data2, 'health_ins_words', 'h1n1_vaccine', axs[0, 1], '', 'V
    plot_bar(Data2, 'opinion_h1n1_vacc_effective', 'h1n1_vaccine', axs[1,
    plot_bar(Data2, 'opinion_h1n1_risk', 'h1n1_vaccine', axs[1, 1], 'Risk

    # Adjust layout
    plt.tight_layout()
    plt.show()

# Call the function to plot the graphs
plot_vaccination_graphs(Data2)
```



Observations:-

1. The plot "H1N1 Vaccination Percentage by Doctor Recommendation" shows a higher vaccination percentage among individuals who received a doctor's recommendation for the H1N1 vaccine compared to those who did not

2. The plot "H1N1 Vaccination Percentage by Health Insurance" might reveals a higher vaccination percentage among individuals with health insurance compared to those without it. This observation indicates that having health insurance could positively impact an individual's likelihood of getting vaccinated, possibly due to better access to healthcare services.

3. The plot "H1N1 Vaccination Percentage by Effectiveness Belief" probably indicates that individuals who believe in the effectiveness of the H1N1 vaccine have a higher vaccination percentage.

4. The plot "H1N1 Vaccination Percentage by Risk Perception of H1N1" is expected to show that individuals who perceive a higher risk of contracting H1N1 have a higher vaccination percentage.

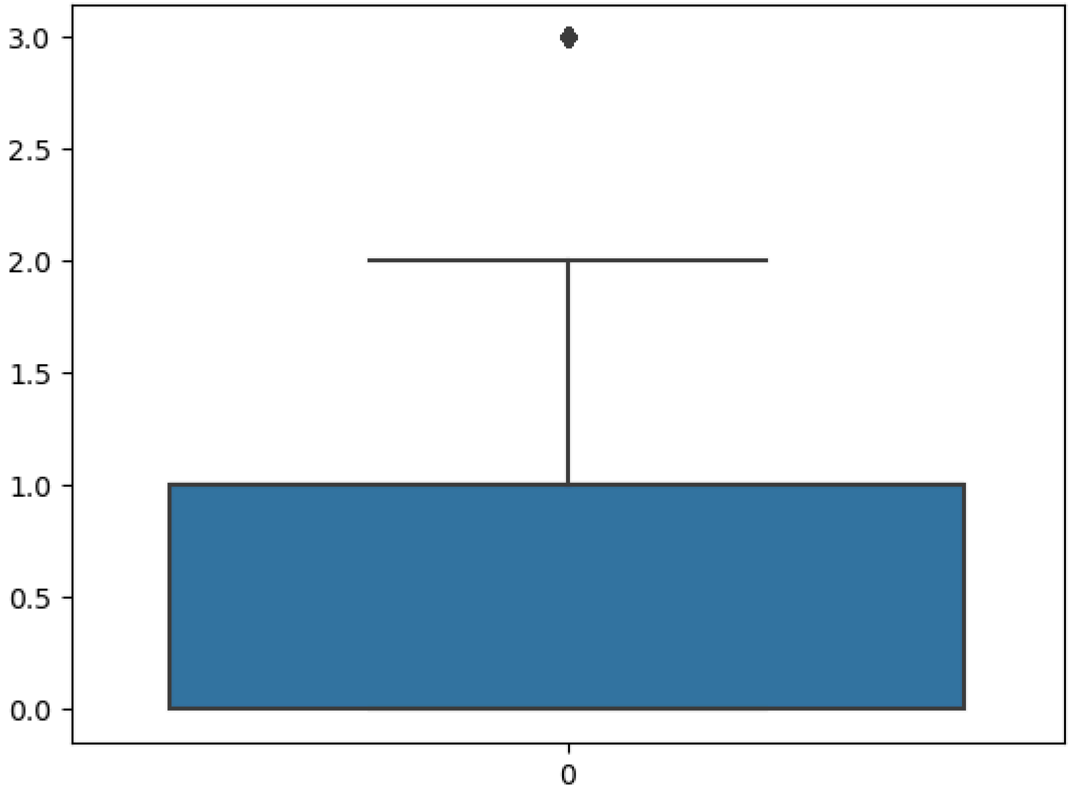
In [ ]:
```
# Function to find the outliers

def findoutliers(column):
    outliers=[]
    Q1=column.quantile(.25)
    Q3=column.quantile(.75)
    IQR=Q3-Q1
    lower_limit=Q1-(1.5*IQR)
    upper_limit=Q3+(1.5*IQR)
    for out1 in column:
        if out1>upper_limit or out1 <lower_limit:
            outliers.append(out1)

    return np.array(outliers)
```

In [ ]:
```
print(len(findoutliers(Data2.household_adults)))
print(len(findoutliers(Data2.household_children)))
```

```
1125
1747
```

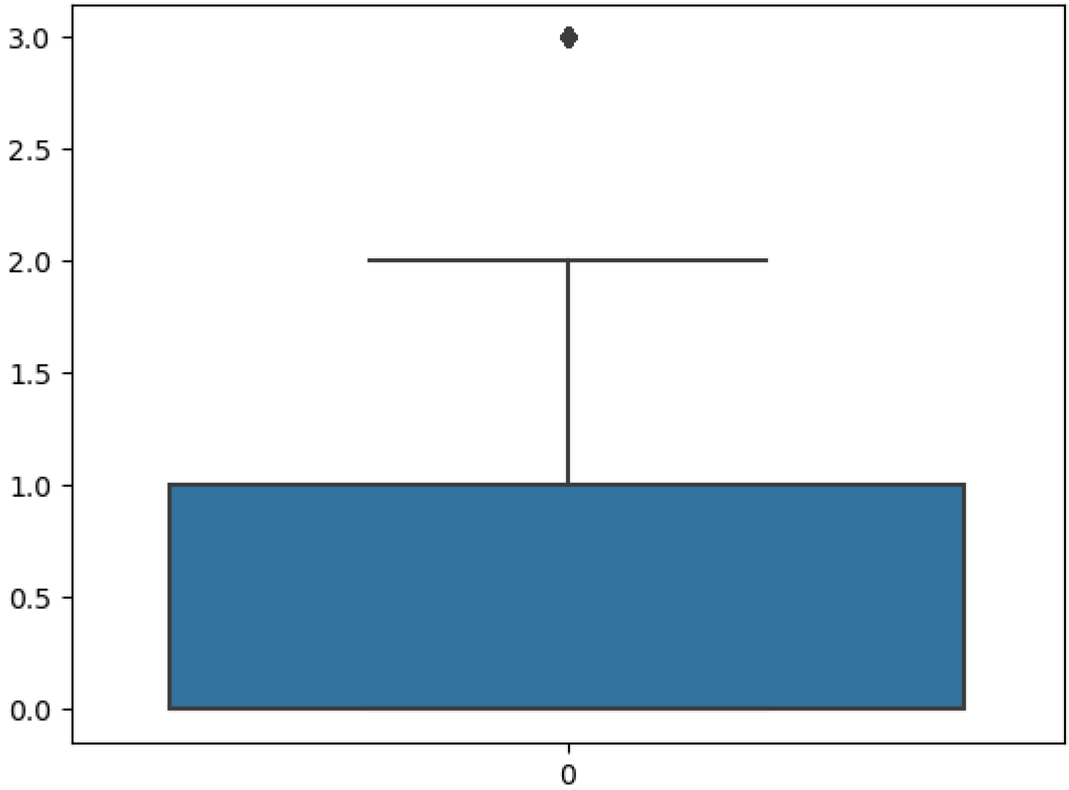
In [ ]:
```
# Visualising the outliers
sns.boxplot(Data2.household_adults)
```

Out[ ]: <Axes: >



In [ ]:
```
sns.boxplot(Data2.household_children)
```

Out[ ]: <Axes: >



There are outliers in the dataset but we are not removing them as some algorithms are not sensitive to outliers

4. Data Preparation (Data Cleaning)

There were a few changes I made to the data set.

1. First, I dropped the "respondent_id" and "seasonal_vaccine" columns because they were not relevant for our purposes.
2. I also added columns due to categorical columns that we transformed with OneHotEncoder.
3. I also filled null values with Iterative Imputer, which was a better alternative to simple imputer for our dataset.
4. I replaced category names with frequency counts with CountEncoder for the columns which had more than 10 unique categories.
5. I used pipelines to make preprocessing and modelling more efficient, and also to prevent data leakage.
6. I also decided to split training and testing data twice so that we could have a holdout set to test our final model's generalizability at the end.

In [ ]:
```
# Define our X and y
X = Data1.drop(columns = ['respondent_id', 'h1n1_vaccine', 'seasonal_vacc
y = Data1['h1n1_vaccine']
```

I chose 80%, 20% for train and validation. Also set the random seed to 42 and stratify=y.

Using stratify=y ensures that the holdout set has a similar distribution of classes as the original dataset, which is particularly important when dealing with imbalanced datasets to ensure that the model's evaluation is fair and representative.

4.1 Train-Test Split

I decided to split training and testing data twice so that we could have a holdout set to test our final model's generalizability at the end.

In [ ]:
```
# Train - Holdout Set Split
X_train, X_hold, y_train, y_hold = train_test_split(X, y, test_size=0.2,
```

In [ ]:
```
X_train.head()
```

Out[ ]:

	h1n1_concern	h1n1_knowledge	behavioral_antiviral_meds	behavioral_avoidanc
20417	1.0	2.0	0.0	1.
13969	2.0	2.0	0.0	1.
24930	2.0	2.0	0.0	1.
15420	2.0	1.0	0.0	0.
10998	2.0	1.0	0.0	0.

We used the stratify argument for y (our target) in both splits to help deal with the class imbalance problem.

In [ ]:

```
# Regular Train Test Split
X_tr, X_te, y_tr, y_te = train_test_split(X_train, y_train, test_size=0.2
```

In [ ]:

```
# Set up lists for each columns datatypes
num_cols = []
ohe_cols = []
freq_cols = []

for c in X.columns:
    if X[c].dtype in ['float64', 'int64']:
        num_cols.append(c)
    elif X[c].nunique() < 10:
        ohe_cols.append(c)
    else:
        freq_cols.append(c)
```

In [ ]:

```
# We wanted to see each column category
print(f'Numerical Columns:', num_cols)
print('\n')
print(f'Object Columns (with less than 10 unique values):', ohe_cols)
print('\n')
print(f'Object Columns (with more than 10 unique values):', freq_cols)
```

```
Numerical Columns: ['h1n1_concern', 'h1n1_knowledge', 'behavioral_antivira
l_meds', 'behavioral_avoidance', 'behavioral_face_mask', 'behavioral_wash_
hands', 'behavioral_large_gatherings', 'behavioral_outside_home', 'behavio
ral_touch_face', 'doctor_recc_h1n1', 'doctor_recc_seasonal', 'chronic_med_
condition', 'child_under_6_months', 'health_worker', 'health_insurance',
'opinion_h1n1_vacc_effective', 'opinion_h1n1_risk', 'opinion_h1n1_sick_fro
m_vacc', 'opinion_seas_vacc_effective', 'opinion_seas_risk', 'opinion_seas
_sick_from_vacc', 'household_adults', 'household_children']


Object Columns (with less than 10 unique values): ['age_group', 'educatio
n', 'race', 'sex', 'income_poverty', 'marital_status', 'rent_or_own', 'emp
loyment_status', 'census_msa']


Object Columns (with more than 10 unique values): ['hhs_geo_region', 'empl
oyment_industry', 'employment_occupation']
```

In [ ]:

```python
# Preprocessing with Pipelines
num_transformer = Pipeline(steps=[
    ('num_imputer', IterativeImputer(max_iter=100, random_state=42)),    #
    ('minmaxscaler', MinMaxScaler())                                      #
])

ohe_transformer = Pipeline(steps=[
    ('ohe_imputer', SimpleImputer(strategy='constant', fill_value='Unknow
    ('ohe_encoder', OneHotEncoder(handle_unknown='ignore'))
])

freq_transformer = Pipeline(steps=[
    ('freq_encoder', ce.count.CountEncoder(normalize=True, min_group_size
    ('freq_imputer', IterativeImputer(max_iter=100, random_state=42))
])
```

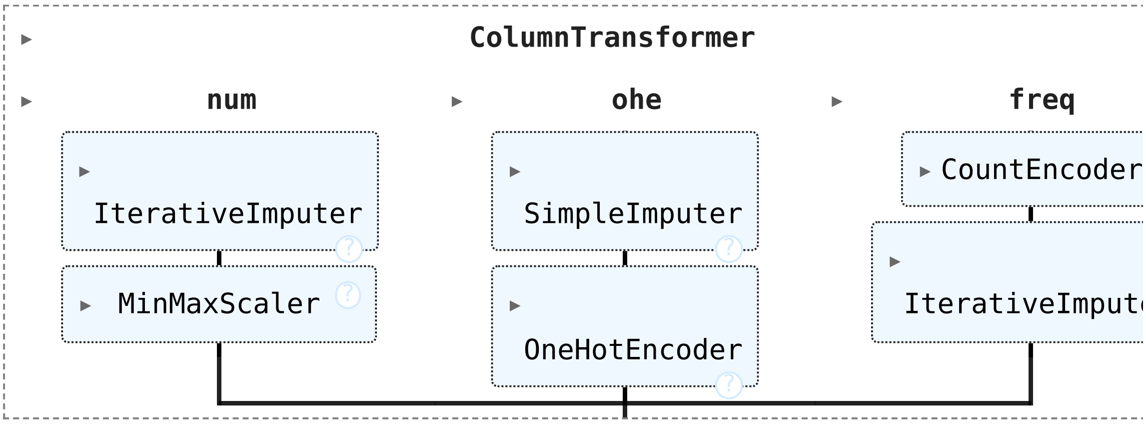
In [ ]:

```python
# Preprocessor defined using ColumnTransformer by packaging the all compo
preprocessor = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_cols),
        ('ohe', ohe_transformer, ohe_cols),
        ('freq', freq_transformer, freq_cols)
    ])
```

In [ ]:

```python
# Fitting preprocessor to see the components as a whole to the training s
preprocessor.fit(X_tr)
```

Out[ ]:



In [ ]:
```python
# Let'see what this looks like after the preprocessor transformation
X_tr_transformed = preprocessor.transform(X_tr)
X_tr_transformed.shape
```

Out[ ]:
```
(17092, 59)
```

The number of features increase from 36 to 59 while the no of rows reduce from 26707 to 17092.

In [ ]:
```python
# Visualize it with Pandas dataframe
pd.DataFrame(X_tr_transformed).head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9
0	1.000000	0.5	0.008339	0.000000	1.000000	0.884731	1.0	1.000000	1.0	1.00000
1	0.666667	1.0	0.008339	0.966636	0.038403	0.884731	0.0	0.020289	1.0	1.00000
2	0.000000	0.5	0.008339	0.966636	0.038403	0.884731	0.0	0.020289	1.0	1.00000
3	0.666667	1.0	0.008339	0.966636	0.038403	0.884731	0.0	0.020289	1.0	0.13727
4	0.666667	1.0	0.008339	0.966636	0.038403	0.884731	1.0	1.000000	1.0	0.13727

Modeling

I wanted to use a variety of different models so as to find the most accurate model.

Because there are many different hyperparameters for each model and I did not know the optimal combinations, we used GridSearrchCV to find the best combinations for each model. I specified class weight to be balanced in order to address the class imbalance issue for our models, whenever possible. I also analyzed the accuracy score, precision score, f1 score, and roc-auc curve for each model.

I also compared the different roc-auc curves of each model, to choose the final model. Additionally, I looked closely at the confusion matrix to see whether or not we were minimizing false positives. Gradient Boosting Classifier gave us the best accuracy and precision scores, so we chose it to be our final model.

We will use this function to evaluate the performance of our models:

In [ ]:
```python
def evaluate(estimator, X_tr, X_te, y_tr, y_te, roc_auc='skip'):
    """
    Evaluation function to show a few scores for both the train and test
    Also shows a confusion matrix for the test set.

    Parameters:
    estimator : a fit sklearn-style model or pipeline
    X_tr : array or pandas DataFrame
        Training input variables
    X_te : array or pandas DataFrame
        Testing input variables
    y_tr : array or pandas Series
        Training output variable
```

```
    y_te : array or pandas Series
        Testing output variable
    roc_auc : str
        'skip': default, skips calculating roc_auc
        'dec': use decision_function to calculate roc_auc
        'proba': use predict_proba to calculate roc_auc
    """

    # Grab predictions
    tr_preds = estimator.predict(X_tr)
    te_preds = estimator.predict(X_te)

    # Output needed for roc_auc_score
    if roc_auc == 'skip': # skips calculating the roc_auc_score
        train_out = False
        test_out = False
    elif roc_auc == 'dec':
        train_out = estimator.decision_function(X_tr)
        test_out = estimator.decision_function(X_te)
    elif roc_auc == 'proba':
        train_out = estimator.predict_proba(X_tr)[:, 1] # proba for the 1
        test_out = estimator.predict_proba(X_te)[:, 1]
    else:
        raise ValueError("The value for roc_auc should be 'skip', 'dec' o

    # Print training scores
    print("Training Scores:")
    print(f"Train Accuracy: {accuracy_score(y_tr, tr_preds)}")
    print(f"Train Precision: {precision_score(y_tr, tr_preds)}")
    print(f"Train Recall: {recall_score(y_tr, tr_preds)}")
    print(f"Train F1-Score: {f1_score(y_tr, tr_preds)}")
    if isinstance(train_out, np.ndarray): # checking for roc_auc
        print(f"Train ROC-AUC: {roc_auc_score(y_tr, train_out)}")

    print("*" * 10)

    # Print testing scores
    print("Testing Scores:")
    print(f"Test Accuracy: {accuracy_score(y_te, te_preds)}")
    print(f"Test Precision: {precision_score(y_te, te_preds)}")
    print(f"Test Recall: {recall_score(y_te, te_preds)}")
    print(f"Test F1-Score: {f1_score(y_te, te_preds)}")
    if isinstance(test_out, np.ndarray): # checking for roc_auc
        print(f"Test ROC-AUC: {roc_auc_score(y_te, test_out)}")

    # Plot confusion matrix for test set
    ConfusionMatrixDisplay.from_estimator(estimator, X_te, y_te, cmap="pl
    plt.show()
```

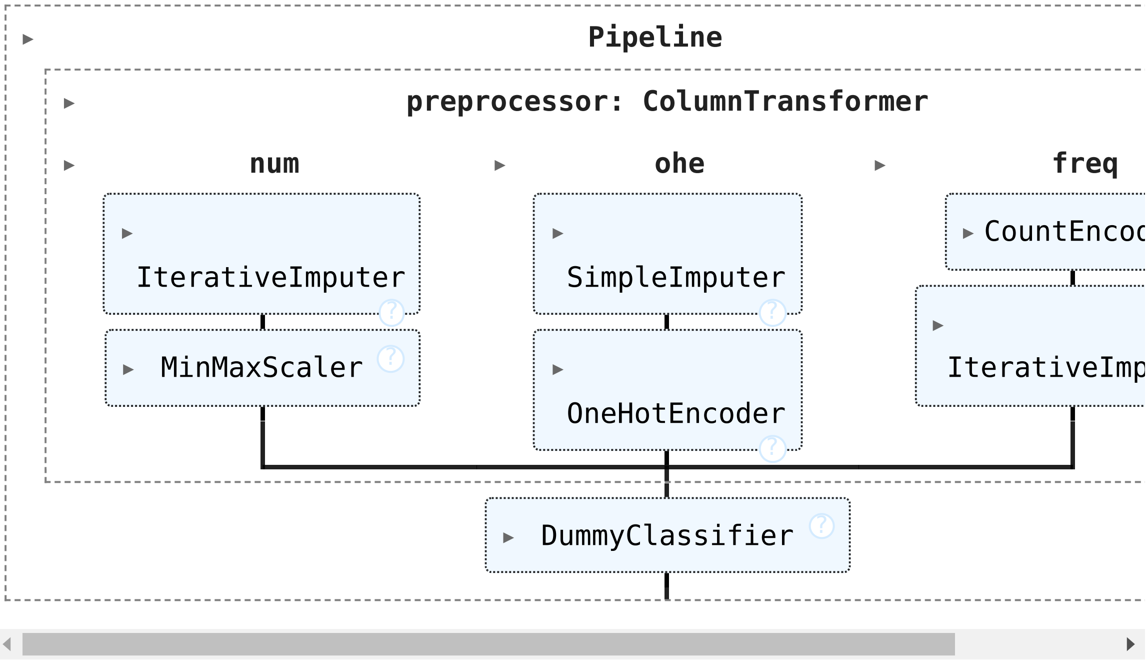
Baseline Understanding

- To be able to truly understand and then improve our model’s performance, we first need to establish a baseline for the data that we have
- Let's use DummyClassifier to make prediction based on the most frequent class in the target variable, which is 0 in our case.

In [ ]:
```
# Setting the up the dummy model to go through the pipeline
dummy_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DummyClassifier(strategy="most_frequent"))
])
```

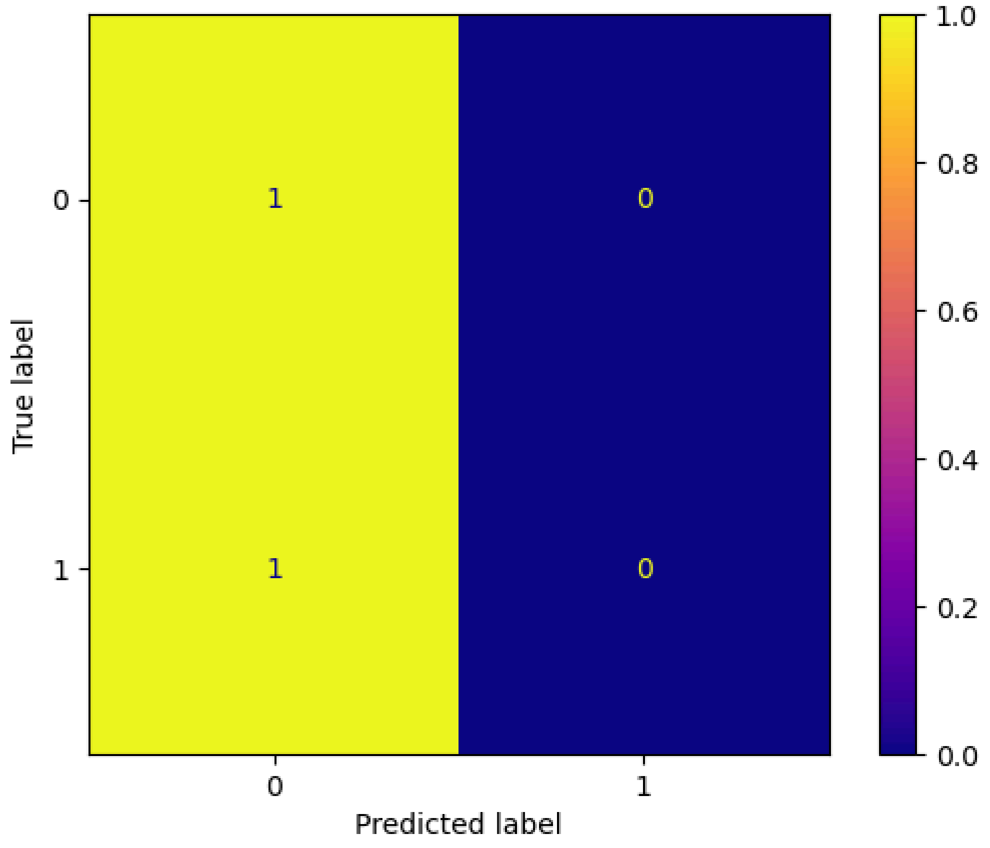
In [ ]:
```
# Fitting the dummy model
dummy_model.fit(X_tr, y_tr)
```

Out[ ]:

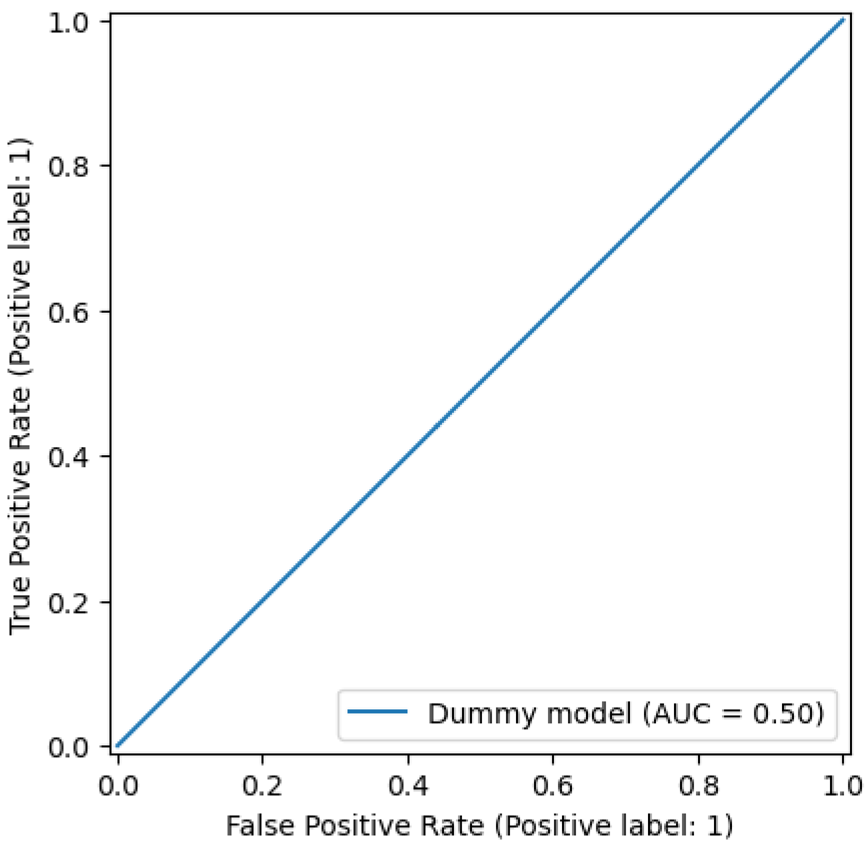


In [ ]:
```
# Evaluate dummy model using the evaluate function created above
evaluate(dummy_model, X_tr, X_te, y_tr, y_te, roc_auc='skip')
```

```
Training Scores:
Train Accuracy: 0.7875614322490054
Train Precision: 0.0
Train Recall: 0.0
Train F1-Score: 0.0
**********
Testing Scores:
Test Accuracy: 0.7875029253451907
Test Precision: 0.0
Test Recall: 0.0
Test F1-Score: 0.0
```



In [ ]:
```
# Plotting the ROC-AUC curve for the dummy model using from_estimator met
RocCurveDisplay.from_estimator(dummy_model, X_te, y_te, name='Dummy model
plt.show()
```



So, the mean of the accuracy score is a little over 78% if we always guess the majority class, which is 0 in this case.

Modeling Iterations

Now we will start to iterate over multiple models!

MODEL 1: Decision Tree Classifier

In [ ]:
```
# Setting up the DecisionTreeClassifier to go though the pipeline
dtc = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier())
])
```

In [ ]:
```
cross_validate(dtc, X_tr, y_tr, return_train_score=True)
```

Out[ ]:
```
{'fit_time': array([7.42701888, 5.89880443, 5.00161982, 8.07086277, 3.17
024493]),
 'score_time': array([0.18780661, 0.08220649, 0.12744069, 0.08681822, 0.
0756011 ]),
 'test_score': array([0.77127815, 0.7739105 , 0.77589233, 0.77940316, 0.
78730252]),
 'train_score': array([1., 1., 1., 1., 1.])}
```

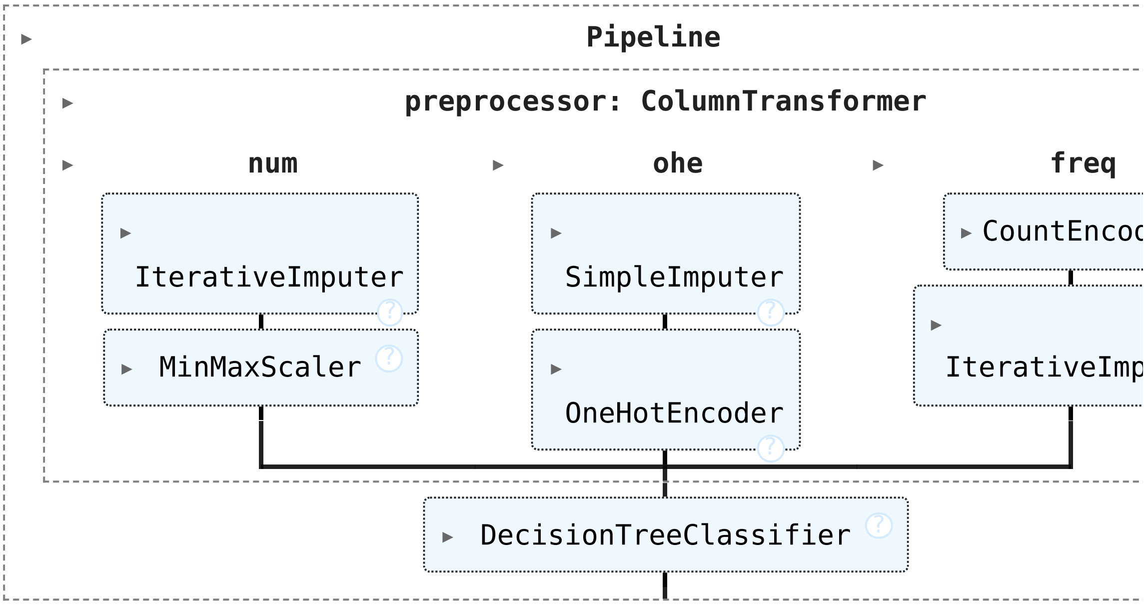
We see that, we have overfitting problem with DecisionTreeClassifier()!

In [ ]:
```
# Let's do GridSearchCV
param_grid = {
    "classifier__max_depth": [1, 2, 5],
    "classifier__min_samples_split": [2, 10],
    "classifier__class_weight": ['balanced', None]    # we have class-imba
}

# Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(dtc,
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1')
# Fit GridSearchCV
output_dtc = grid.fit(X_tr, y_tr)

print(output_dtc.best_params_)

best_model_dtc=output_dtc.best_estimator_
```

```
{'classifier__class_weight': 'balanced', 'classifier__max_depth': 5, 'clas
sifier__min_samples_split': 10}
```

In [ ]:
```
#Fit the model using the best parameters
best_model_dtc.fit(X_tr, y_tr)
```

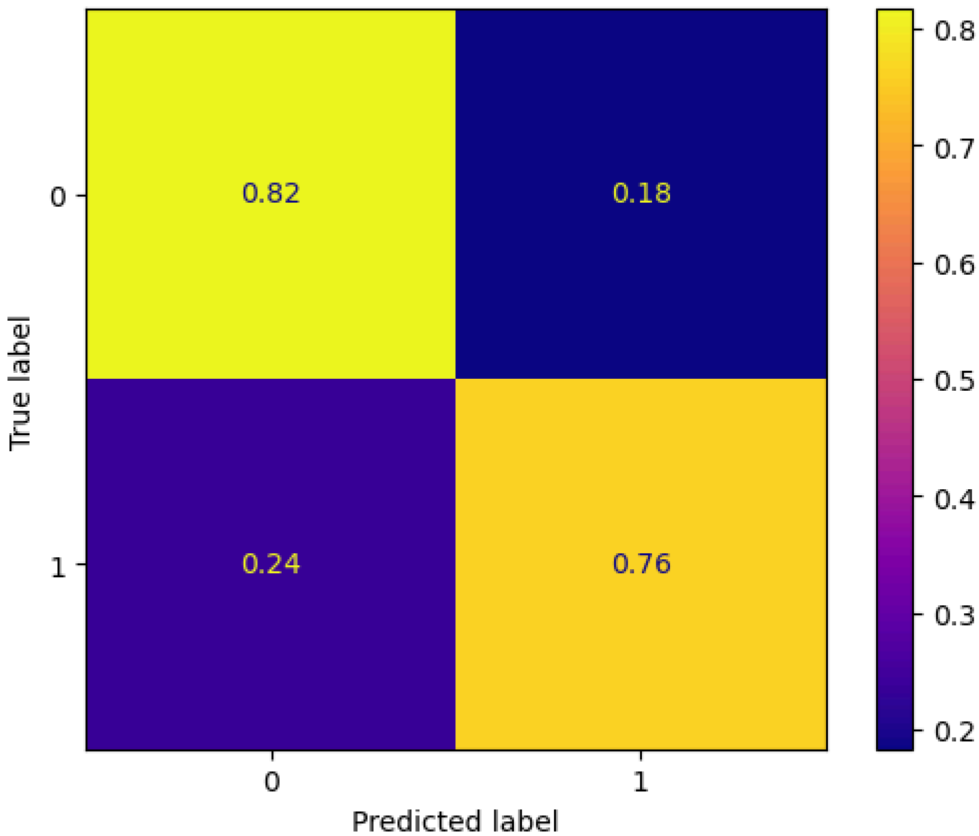
Out[ ]:



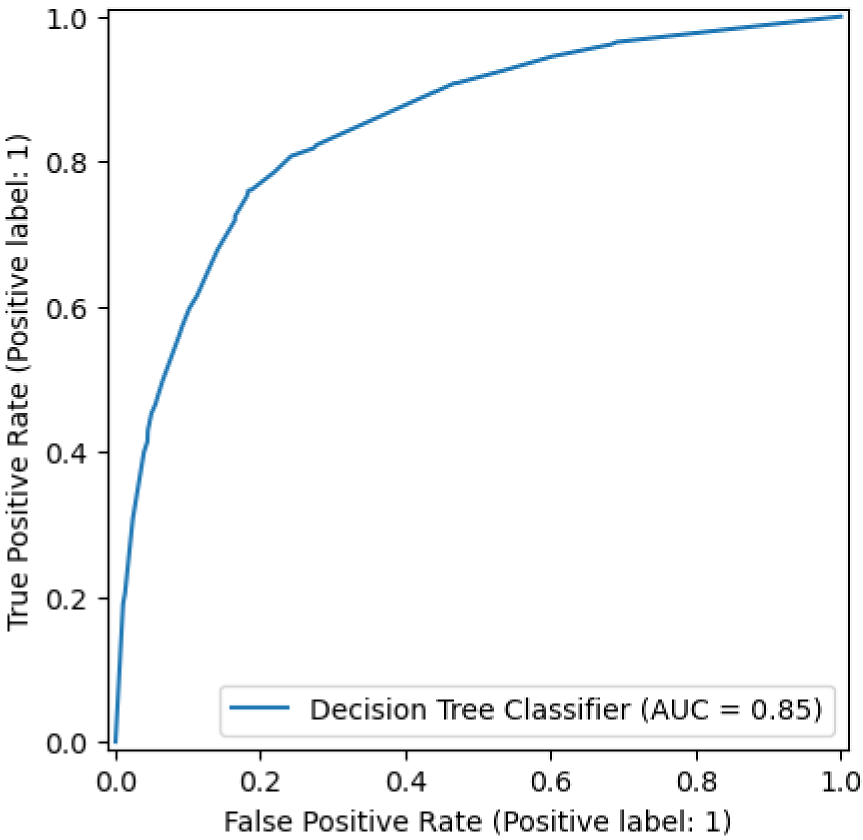
In [ ]:

```
# Evaluating the decision tree model for various metrics
evaluate(best_model_dtc, X_tr, X_te, y_tr, y_te, roc_auc='skip')
```

```
Training Scores:
Train Accuracy: 0.794640767610578
Train Precision: 0.5115303983228512
Train Recall: 0.7391903057009088
Train F1-Score: 0.6046406848389276
**********
Testing Scores:
Test Accuracy: 0.8045869412590686
Test Precision: 0.5279265493496557
Test Recall: 0.7599118942731278
Test F1-Score: 0.6230248306997742
```



In [ ]:
```
# Plotting the ROC-AUC curve for the dummy model using from_estimator met
RocCurveDisplay.from_estimator(best_model_dtc, X_te, y_te, name='Decision
plt.show()
```



This decision tree model is not overfitting, but we have a low precision score, as well as a low f1 score. However, the AUC for this model is 0.85, which is fairly high, meaning that it does an adequate job of maximizing true positives and minimizing the false positives. This model is not overfitting.

MODEL 2: Logistic Regression

In [ ]:
```
# Define your pipeline with preprocessing and classifier
logreg = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=500, random_state=42))
])
```

In [ ]:
```
cross_validate(logreg, X_tr, y_tr, return_train_score=True)
```

Out[ ]:
```
{'fit_time': array([4.93650317, 3.53602052, 2.53369737, 2.54796076, 2.62
224889]),
 'score_time': array([0.07737327, 0.0631144 , 0.07117891, 0.0715673 , 0.
0787549 ]),
 'test_score': array([0.83533197, 0.83503949, 0.82562902, 0.82709187, 0.
83762434]),
 'train_score': array([0.83346742, 0.83346742, 0.83516162, 0.83618546,
0.83282141])}
```

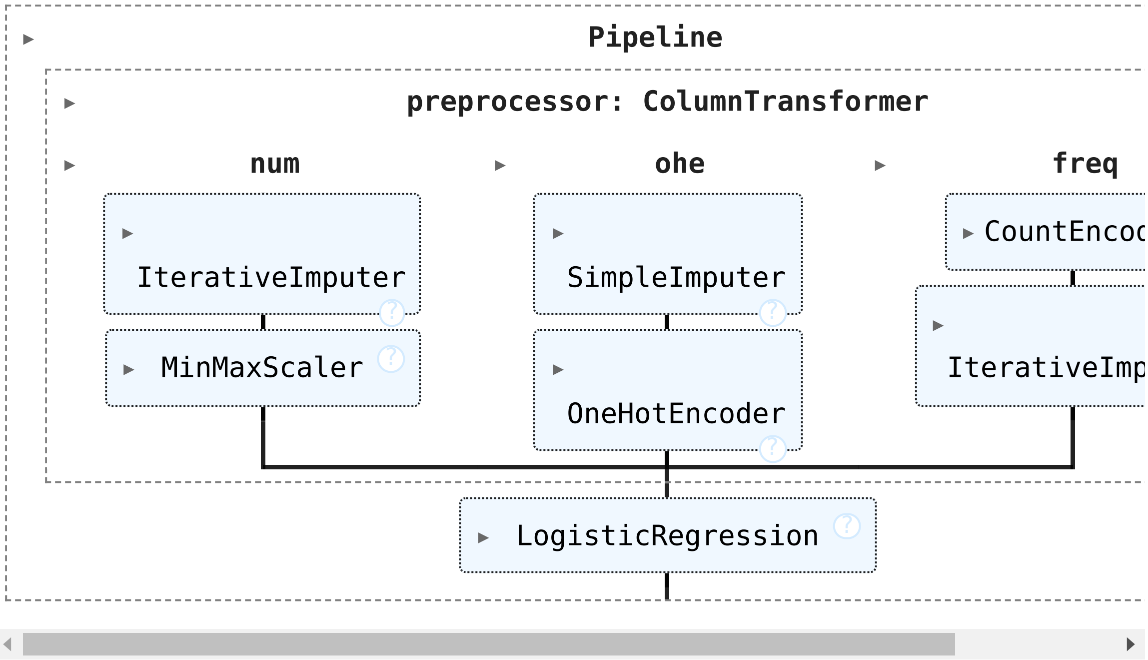
In [ ]:
```
# Define your grid for GridSearchCV
param_grid = {
```

```
    "classifier__solver": ['lbfgs', 'liblinear', 'newton-cg', 'saga'],
    "classifier__C" : [1, 0.1, 0.01], # regularization parameter
    "classifier__class_weight": ['balanced', None]
}
# Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(logreg,
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1')

# Fit GridSearchCV
output_logreg = grid.fit(X_tr, y_tr)

#print the best parameters for the model
print(output_logreg.best_params_)

# Retrieve the best estimator from GridSearchCV
best_model_logreg=output_logreg.best_estimator_
```

```
{'classifier__C': 1, 'classifier__class_weight': 'balanced', 'classifier__solver': 'saga'}
```

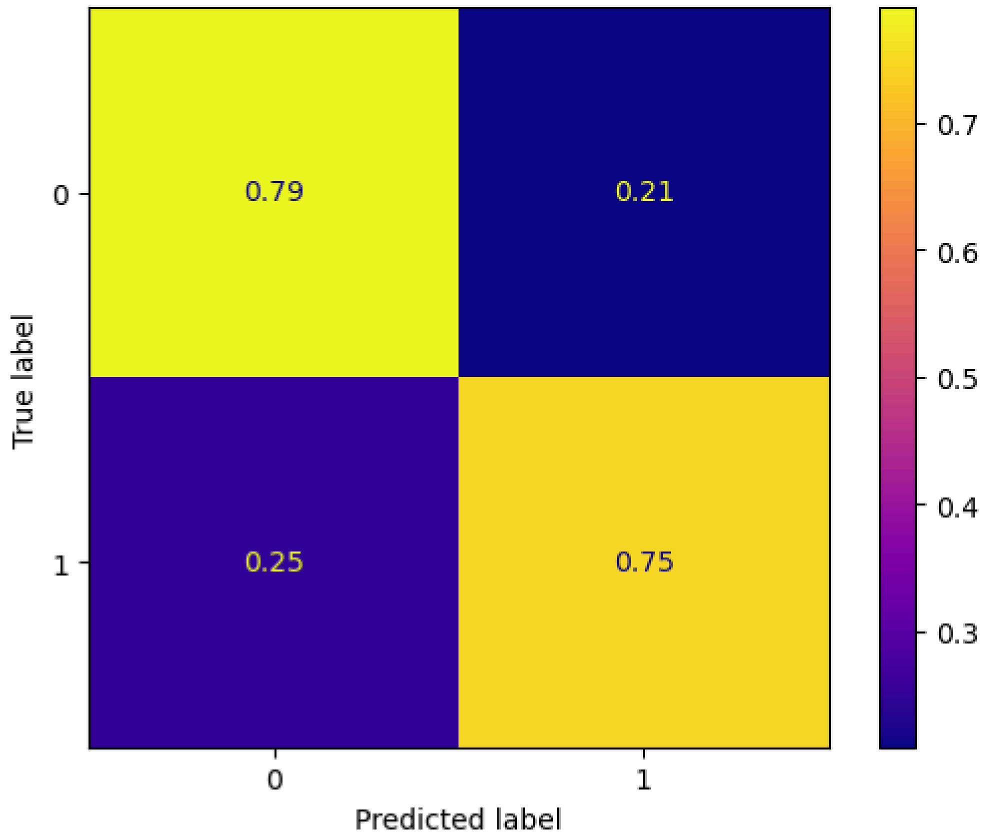
In [ ]:
```
#Fit the best model to the training data
best_model_logreg.fit(X_tr, y_tr)
```

Out[ ]:

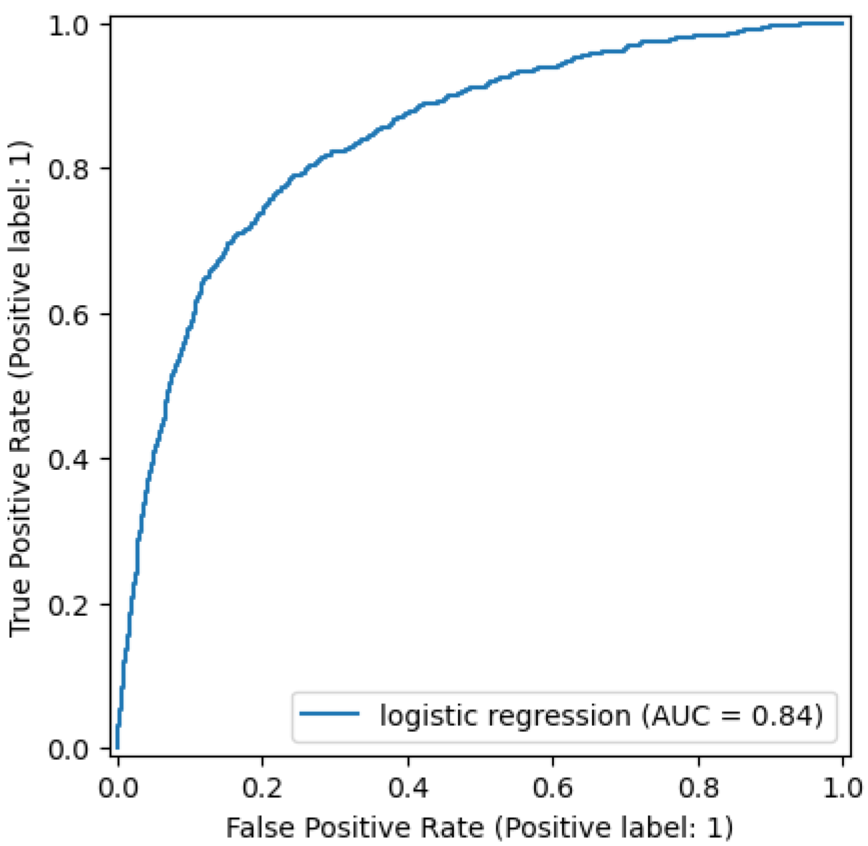


In [ ]:
```
# Evaluating the logistic regression for various metrics
evaluate(best_model_logreg, X_tr, X_te, y_tr, y_te)
```

```
Training Scores:
Train Accuracy: 0.7753334893517435
Train Precision: 0.48130924700411376
Train Recall: 0.7411181492701735
Train F1-Score: 0.5836044242029929
**********
Testing Scores:
Test Accuracy: 0.7825883454247601
Test Precision: 0.4924187725631769
Test Recall: 0.751101321585903
Test F1-Score: 0.5948539031836023
```



In [ ]:
```
RocCurveDisplay.from_estimator(best_model_logreg, X_te, y_te, name='logis
plt.show()
```



This logistic regression model has low precision and fl scores, and has an AUC almost equal to the decision tree AUC above. This model is not overfitting.

MODEL 3: Random Forest

In [ ]:
```
# Setting up the RandomForestClassifier to go through the pipeline
rfc = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

In [ ]:
```
cross_validate(rfc, X_tr, y_tr, return_train_score=True)
```

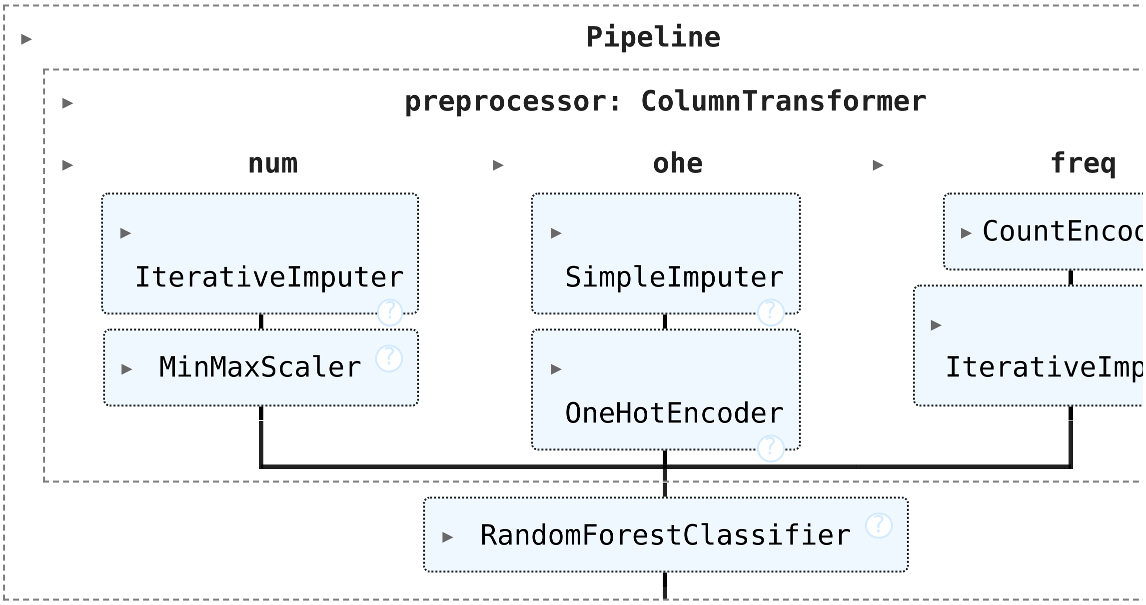
Out[ ]:
```
{'fit_time': array([5.17897797, 3.82426238, 3.97011614, 3.95122027, 4.16
435409]),
 'score_time': array([0.12792277, 0.12009501, 0.12692761, 0.12853265, 0.
12861967]),
 'test_score': array([0.85171103, 0.84907868, 0.84669397, 0.84698654, 0.
85225278]),
 'train_score': array([1.        , 1.        , 1.        , 1.        ,
0.99985374])}
```

In [ ]:
```
# Let's do GridSearchCV
param_grid = {
    "classifier__n_estimators": [100, 200],
    "classifier__max_depth" : [2, 5],
    "classifier__min_samples_leaf": [1, 2],
    "classifier__class_weight" :['balanced', 'balanced_subsample']
    # class weight should be balanced or balanced_subsample - we have imb
}

#Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(rfc,
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1')

# Fit GridSearchCV
output_rfc = grid.fit(X_tr, y_tr)

#Print the best parameters for the model
print(output_rfc.best_params_)

# Retrieve the best estimator from GridSearchCV
best_model_rfc=output_rfc.best_estimator_
```

```
{'classifier__class_weight': 'balanced_subsample', 'classifier__max_dept
h': 5, 'classifier__min_samples_leaf': 1, 'classifier__n_estimators': 100}
```

In [ ]:
```
#Fit the best parameters to the training data
output_rfc.best_estimator_.fit(X_tr, y_tr)
```

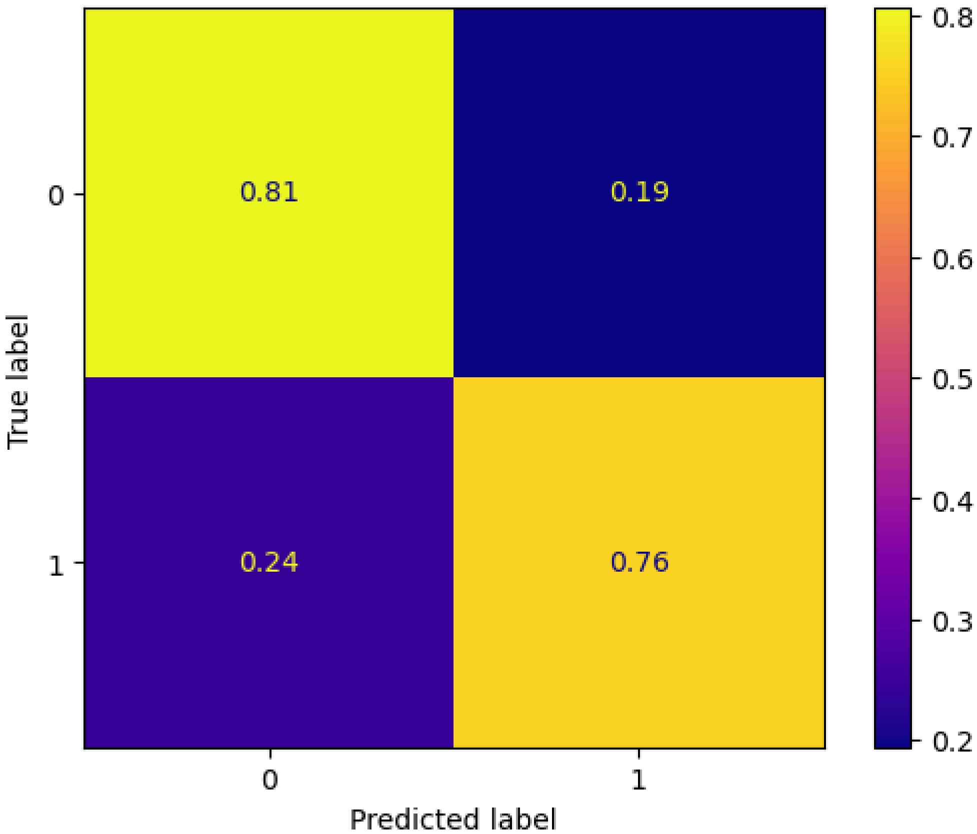
Out[ ]:



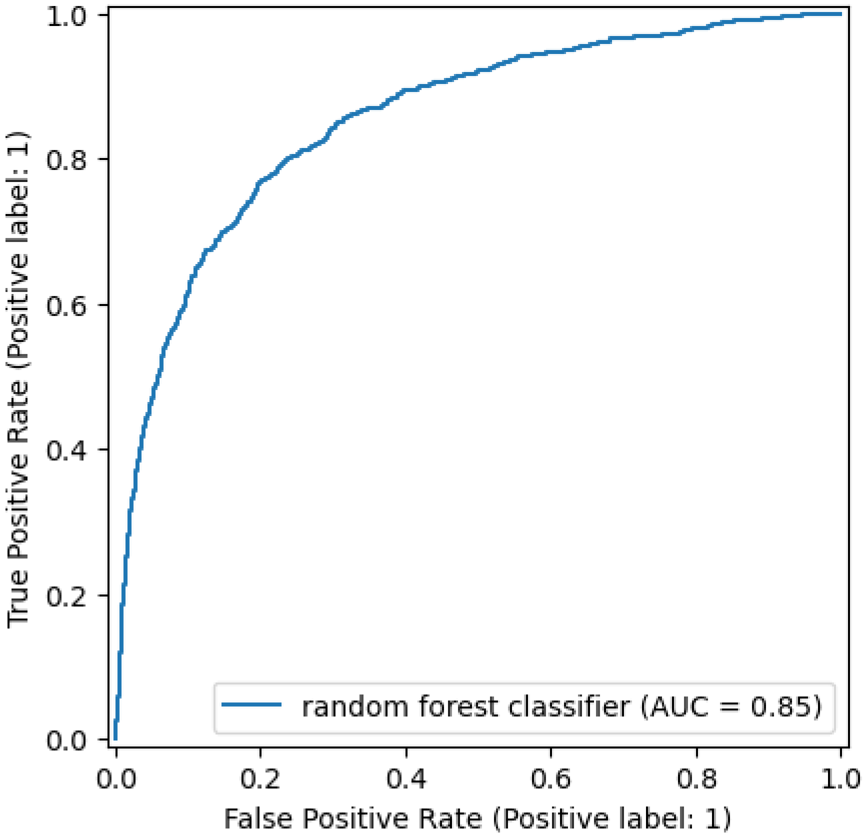
In [ ]:

```
# Evaluating various metrics of the random forest classifier
evaluate(best_model_rfc, X_tr, X_te, y_tr, y_te)
```

```
Training Scores:
Train Accuracy: 0.7930610812075825
Train Precision: 0.5088612368024132
Train Recall: 0.7433213990636188
Train F1-Score: 0.604141018466704
**********
Testing Scores:
Test Accuracy: 0.7954598642639832
Test Precision: 0.5126865671641792
Test Recall: 0.7566079295154186
Test F1-Score: 0.6112099644128114
```



In [ ]:
```
RocCurveDisplay.from_estimator(best_model_rfc, X_te, y_te, name='random f
plt.show()
```



This random forest classifier model also has low precision and fl scores. It has an AUC score of 0.85, which is slightly better than the decision tree model above. This model is not overfitting to a great extent.

MODEL 4: kNN Classifier

In [ ]:
```
kNN = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier())
])
```

In [ ]:
```
cross_validate(kNN, X_tr, y_tr, return_train_score=True)
```

Out[ ]:
```
{'fit_time': array([4.47753859, 2.20786595, 2.81130219, 2.70128775, 2.59
447789]),
 'score_time': array([0.18522739, 0.15040016, 0.15812731, 0.16451311, 0.
19717407]),
 'test_score': array([0.78911963, 0.79584674, 0.79256875, 0.79286132, 0.
79724985]),
 'train_score': array([0.83924523, 0.84056169, 0.84320608, 0.84050022,
0.84064648])}
```

In [ ]:
```
# Define your grid for RandomizedSearchCV
param_grid = {
    "classifier__n_neighbors": [5, 10],
    "classifier__weights" : ['uniform', 'distance'],
    "classifier__p": [1, 2, 3],
}
```

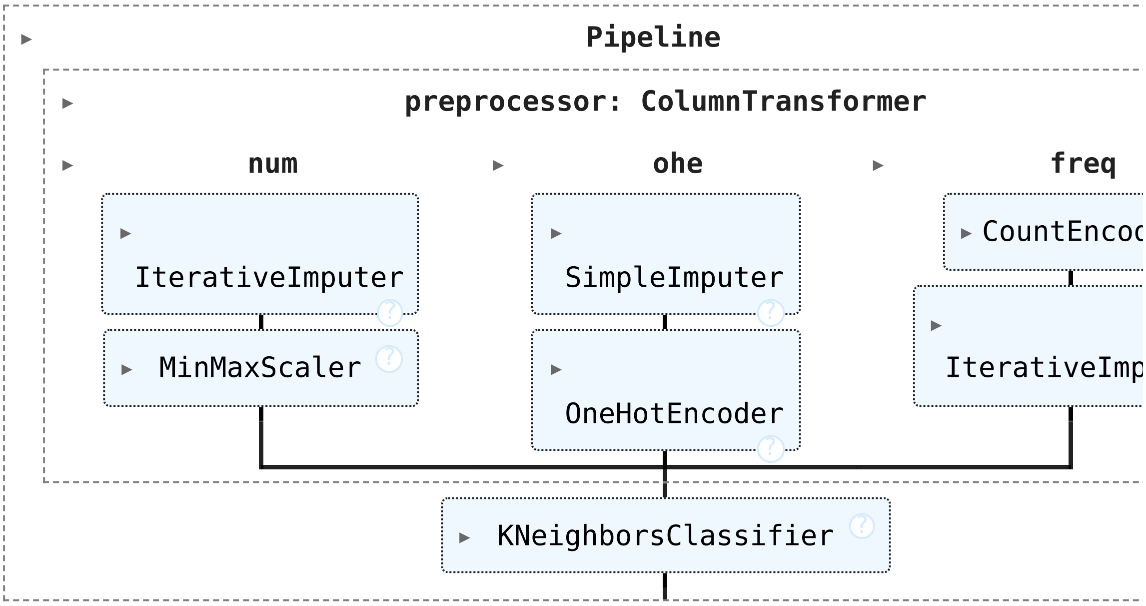
```
# Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(kNN,
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1')
#fit the GridSearchCV
output_kNN = grid.fit(X_tr, y_tr)


#Print the best parameters for the model
print(output_kNN.best_params_)

# Retrieve the best estimator from GridSearchCV
best_model_KNN =output_kNN.best_estimator_
```

```
{'classifier__n_neighbors': 5, 'classifier__p': 1, 'classifier__weights':
'distance'}
```

In [ ]:

```
output_kNN.best_estimator_.fit(X_tr, y_tr)
```

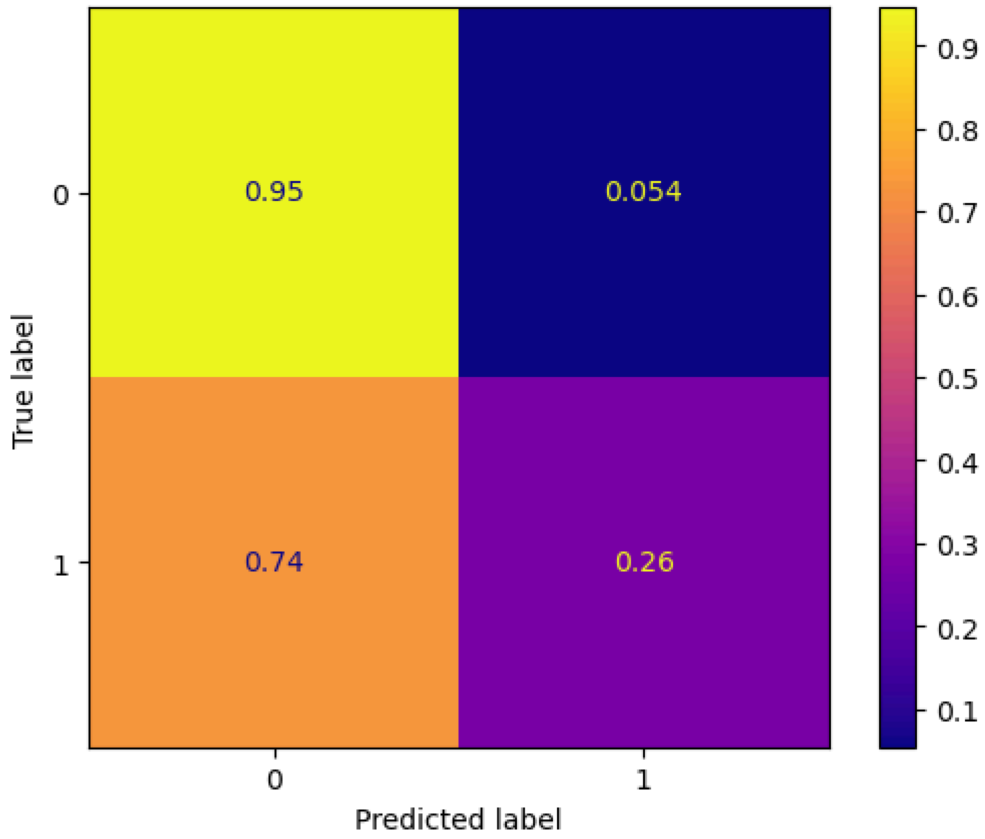
Out[ ]:



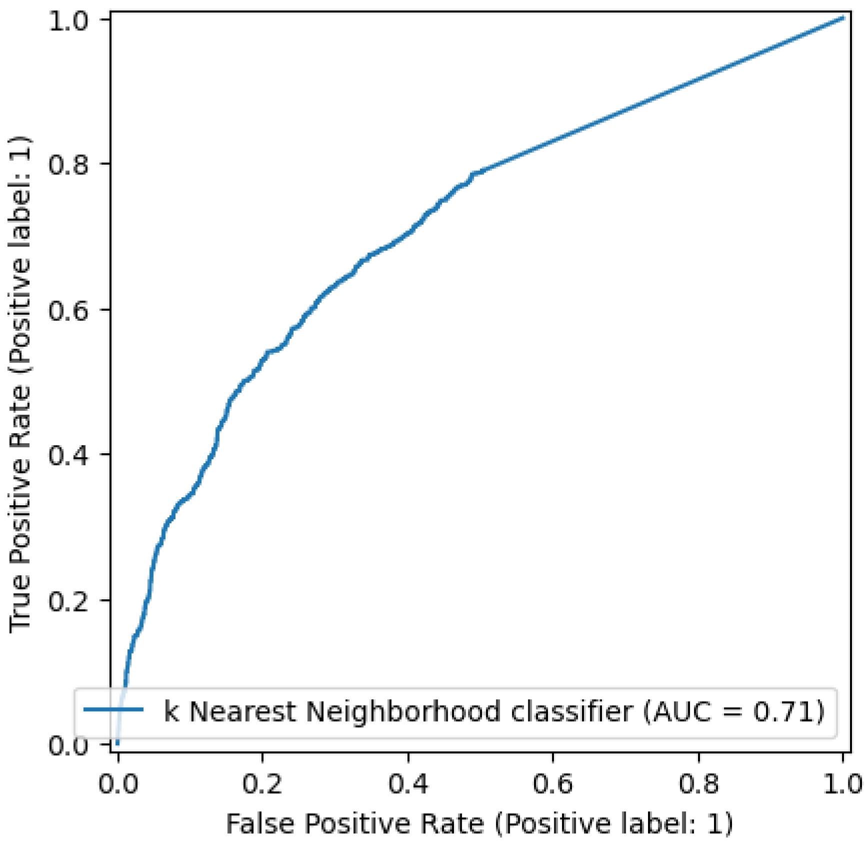
In [ ]:

```
# evaluating the KNN model for various metrics
evaluate(best_model_KNN, X_tr, X_te, y_tr, y_te)
```

```
Training Scores:
Train Accuracy: 1.0
Train Precision: 1.0
Train Recall: 1.0
Train F1-Score: 1.0
**********
Testing Scores:
Test Accuracy: 0.8013105546454482
Test Precision: 0.5700712589073634
Test Recall: 0.2643171806167401
Test F1-Score: 0.3611738148984199
```



In [ ]:
```
# Plotting the roc-auc curve for the KNN model
RocCurveDisplay.from_estimator(best_model_KNN, X_te, y_te, name='k Neares
plt.show()
```



This KNN model is definitely overfitting; the training data has perfect scores for all metrics, whereas the testing data scores are much lower. The AUC score is also much lower than on previous models.

MODEL 5: Gradient Boosting Classifier

In [ ]:
```
gbc = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(random_state=42))
])
```

In [ ]:
```
cross_validate(gbc, X_tr, y_tr, return_train_score=True)
```

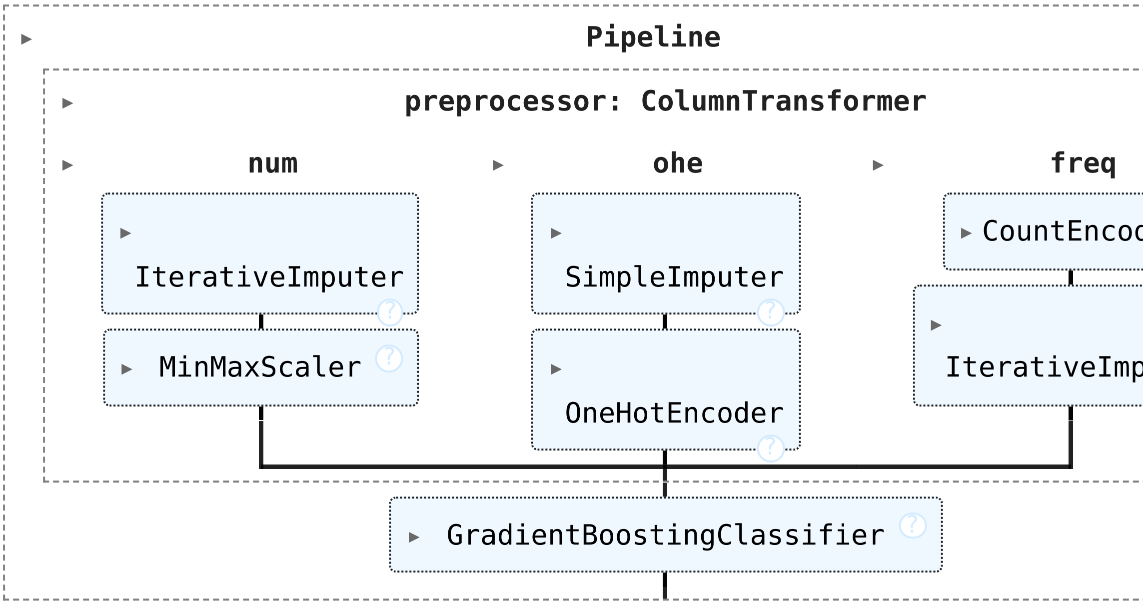
Out[ ]:
```
{'fit_time': array([6.09436631, 5.15299273, 5.54063416, 5.6029582 , 5.42
427301]),
 'score_time': array([0.0781045 , 0.07004285, 0.08713555, 0.07730174, 0.
07750106]),
 'test_score': array([0.85288096, 0.85434338, 0.85225278, 0.85137507, 0.
85868929]),
 'train_score': array([0.86338038, 0.86506253, 0.86192775, 0.86302472,
0.86295159])}
```

In [ ]:
```
# Let's do GridSearchCV
param_grid  = {
    "classifier__n_estimators": [100, 200],
    "classifier__max_depth" : [1, 2, 5],
    "classifier__learning_rate": [1, 0.1, 0.01],
}

# Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(gbc,
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1') # 2*3*3*5 for CV

# Fit GridSearchCV
output_gbc = grid.fit(X_tr, y_tr)

# Print the best parameters found by GridSearchCV
print("Best parameters found: ", output_gbc.best_params_)

# Retrieve the best estimator from GridSearchCV
best_model_gbc = output_gbc.best_estimator_
```

```
Best parameters found:  {'classifier__learning_rate': 1, 'classifier__max_
depth': 2, 'classifier__n_estimators': 100}
```

In [ ]:
```
# Fit the best estimator on the training data
best_model_gbc.fit(X_tr, y_tr)
```

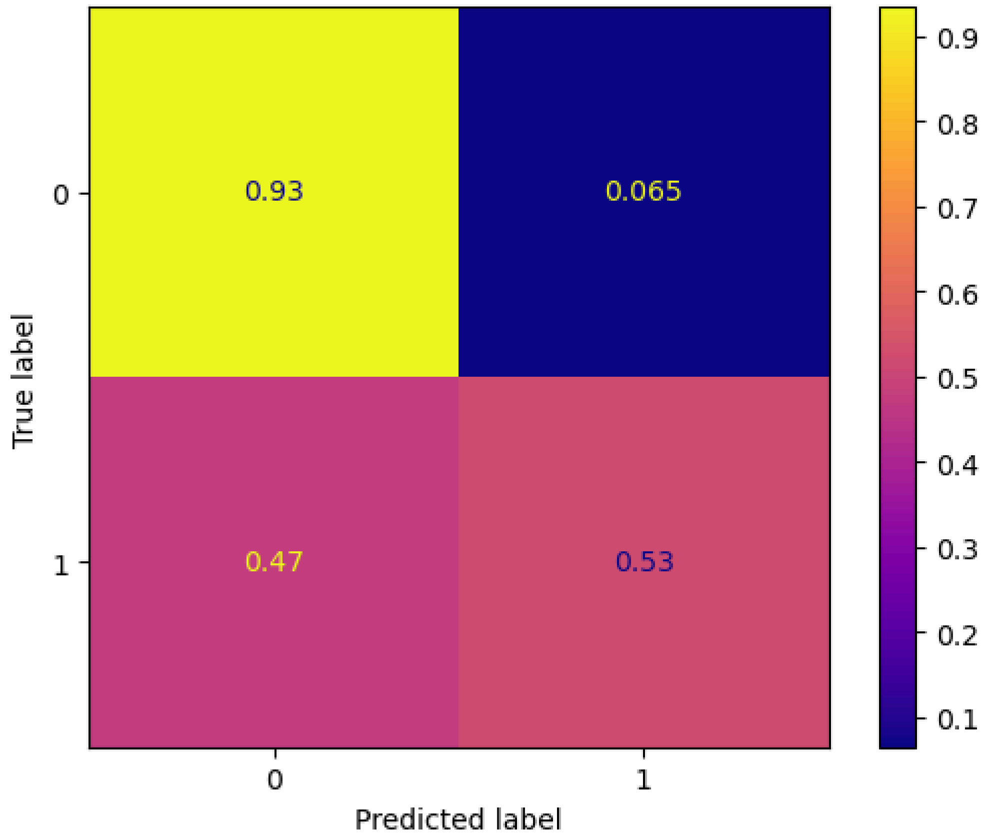
Out[ ]:



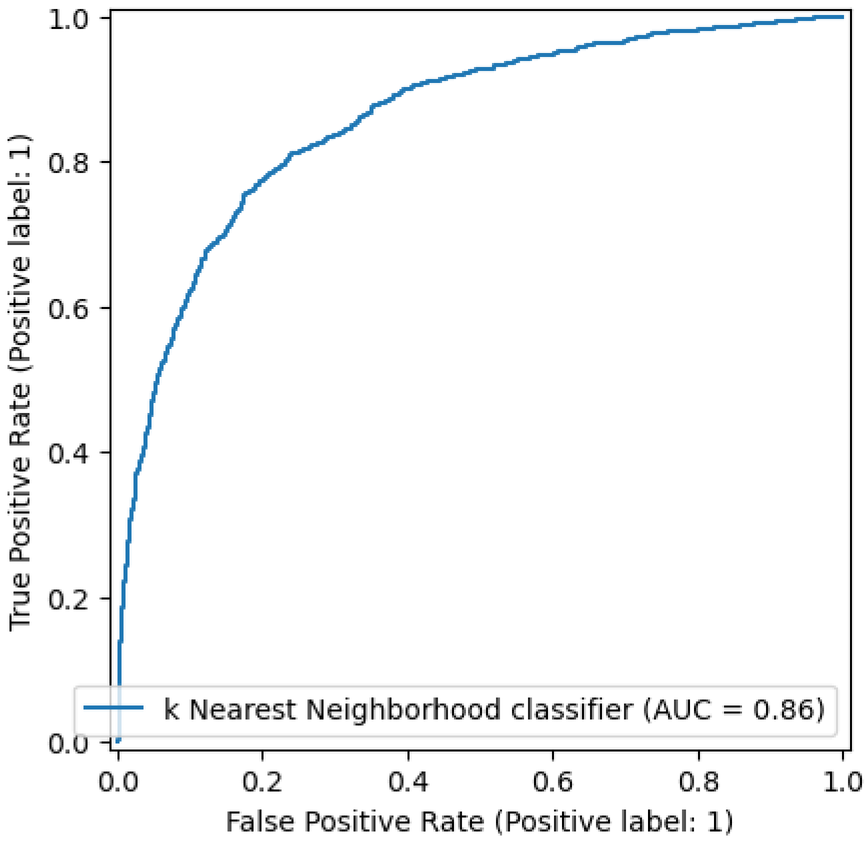
In [ ]:

```python
# Evaluate the best model using the evaluate function
evaluate(best_model_gbc, X_tr, X_te, y_tr, y_te)
```

```
Training Scores:
Train Accuracy: 0.8706412356658085
Train Precision: 0.772239263803681
Train Recall: 0.5546681354998623
Train F1-Score: 0.6456162846610034
**********
Testing Scores:
Test Accuracy: 0.847648022466651
Test Precision: 0.6843615494978479
Test Recall: 0.525330396475771
Test F1-Score: 0.594392523364486
```



In [ ]:
```
# Plotting the roc-auc curve for the KNN model
RocCurveDisplay.from_estimator(best_model_gbc, X_te, y_te, name='k Neares
plt.show()
```



MODEL 6: XG Boosting Classifier

In [ ]:
```
xgb = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', xgboost.XGBClassifier(random_state=42))
])
```

In [ ]:
```
cross_validate(xgb, X_tr, y_tr, return_train_score=True)
```

Out[ ]:
```
{'fit_time': array([5.63627338, 2.99075842, 3.72523427, 3.32572556, 3.25
237846]),
 'score_time': array([0.09734845, 0.06626916, 0.08217311, 0.08071756, 0.
07704997]),
 'test_score': array([0.84235156, 0.84586136, 0.84435342, 0.84669397, 0.
85342305]),
 'train_score': array([0.95370438, 0.95260733, 0.95443908, 0.9510019 ,
0.95575545])}
```

In [ ]:
```
# Let's do GridSearchCV
param_grid = {
    "classifier__n_estimators": [100, 200],
    "classifier__max_depth" : [1, 2],
    "classifier__learning_rate": [1, 0.1],
}


# Setup GridSearchCV with multiple scoring metrics
grid = GridSearchCV(xgb,
```

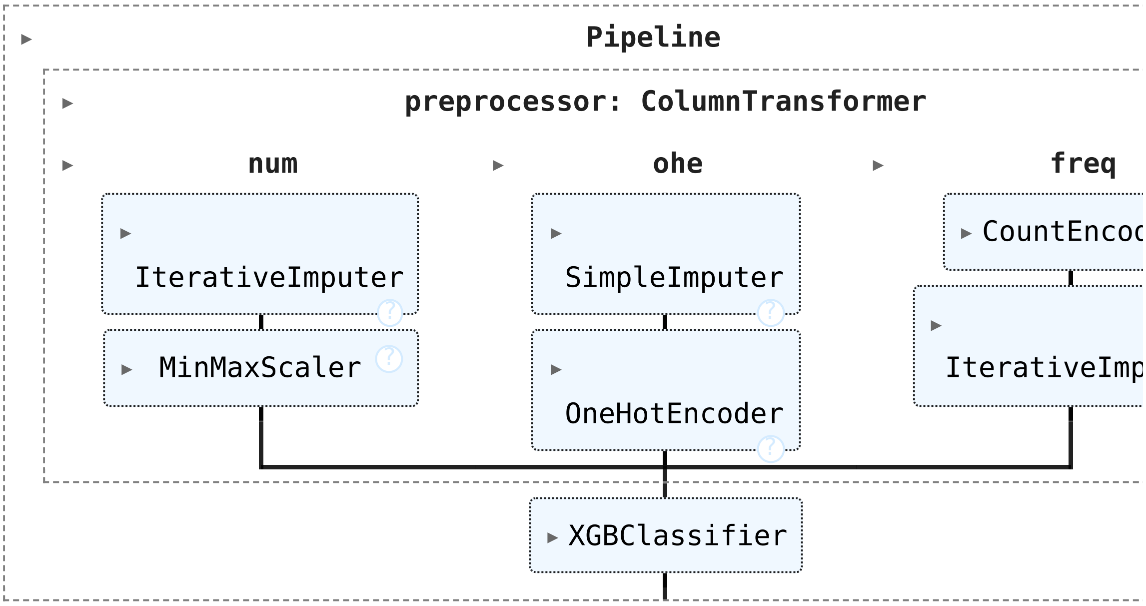
```
                    param_grid,
                    scoring=['f1','precision'],
                    refit = 'f1')

# Fit GridSearchCV
output_xgb = grid.fit(X_tr, y_tr)

#Print the best parameters of the model
print(output_xgb.best_params_)

# Retrieve the best estimator from GridSearchCV
best_model_xgb=output_xgb.best_estimator_
```

```
{'classifier__learning_rate': 1, 'classifier__max_depth': 2, 'classifier__
n_estimators': 100}
```

In [ ]:

```
c#Fit the best model to our training data
best_model_xgb.fit(X_tr ,y_tr)
```

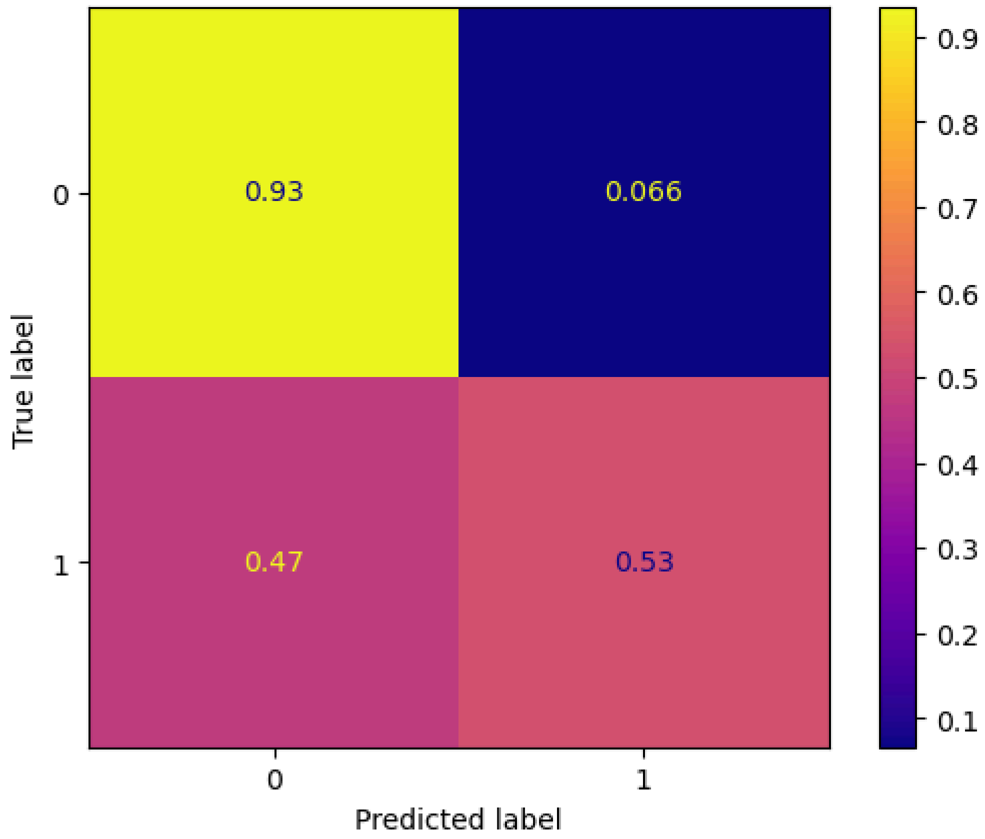
Out[ ]:



In [ ]:

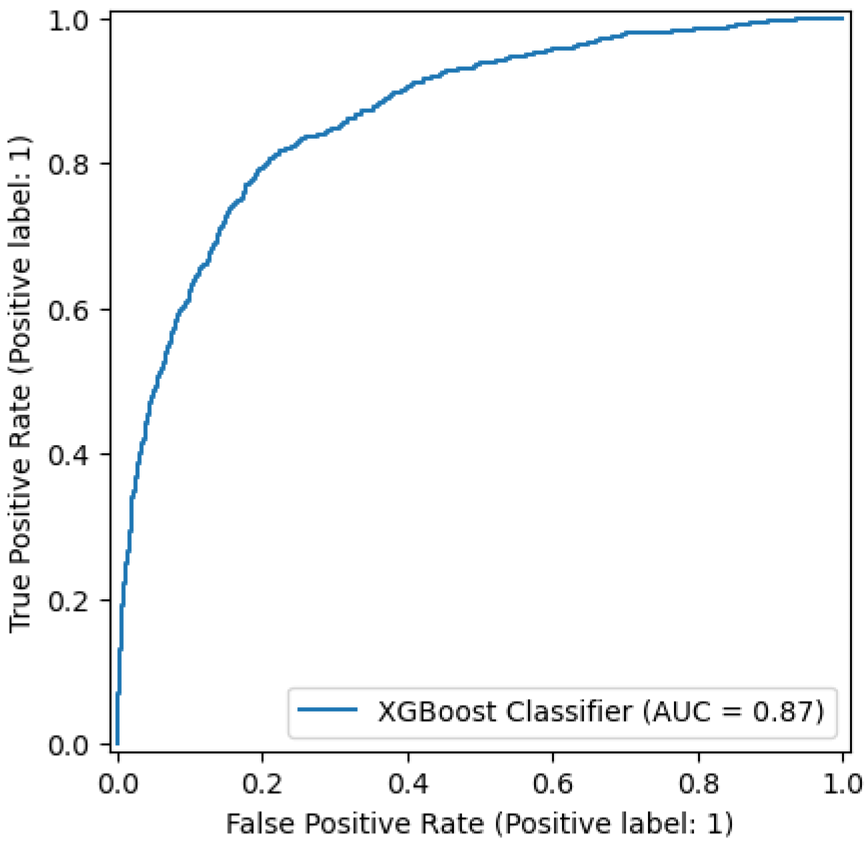
```
# Evaluate the best model using the evaluate function
evaluate(best_model_xgb, X_tr, X_te, y_tr, y_te)
```

```
Training Scores:
Train Accuracy: 0.8670138076293002
Train Precision: 0.754688672168042
Train Recall: 0.554117323051501
Train F1-Score: 0.6390344608543751
**********
Testing Scores:
Test Accuracy: 0.8481160776971682
Test Precision: 0.6842105263157895
Test Recall: 0.5297356828193832
Test F1-Score: 0.5971446306641838
```



In [ ]:

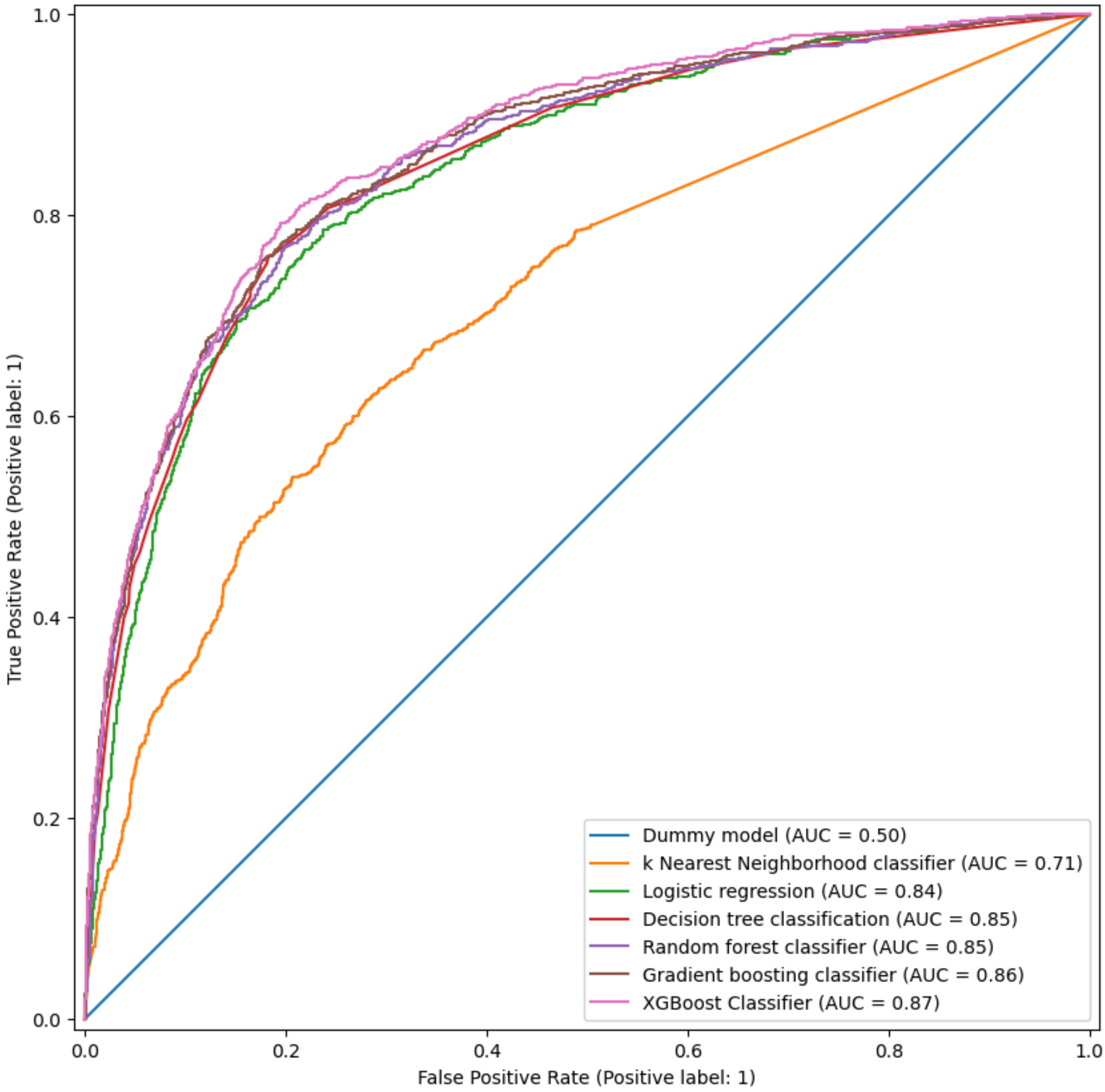
```
# Plotting the roc-auc curve for the XGB model
RocCurveDisplay.from_estimator(best_model_xgb, X_te, y_te, name='XGBoost
plt.show()
```



This model gave us similar scores to the gradient boosting model, but the gradient boosting model has the best AUC score and precision score. So, we will choose the gradient boosting classifier as the final model.

Comparison of Model ROC Curves

In [ ]:
```
# Function to plot ROC curves for multiple models
def plot_roc_curves(models, X_test, y_test):
    fig, ax = plt.subplots(figsize=(10, 10))

    for model in models:
        RocCurveDisplay.from_estimator(model['estimator'], X_test, y_test

    plt.show()

# Define your models
models = [
    {'estimator': dummy_model, 'name': 'Dummy model'},
    {'estimator': best_model_KNN, 'name': 'k Nearest Neighborhood classif
    {'estimator': best_model_logreg, 'name': 'Logistic regression'},
    {'estimator': best_model_dtc, 'name': 'Decision tree classification'}
    {'estimator': best_model_rfc, 'name': 'Random forest classifier'},
    {'estimator': best_model_gbc, 'name': 'Gradient boosting classifier'}
    {'estimator': best_model_xgb, 'name': 'XGBoost Classifier'}
]

# Plot ROC curves
plot_roc_curves(models, X_te, y_te)
```

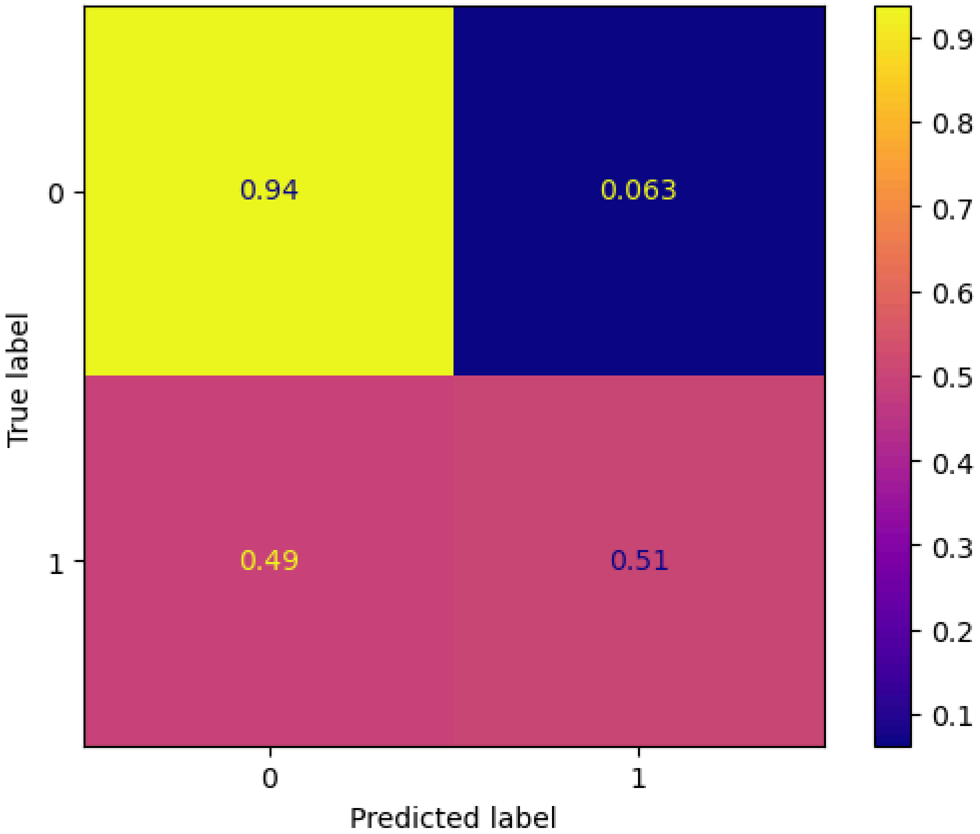


'Final' Model: Gradient Boosting Classifier

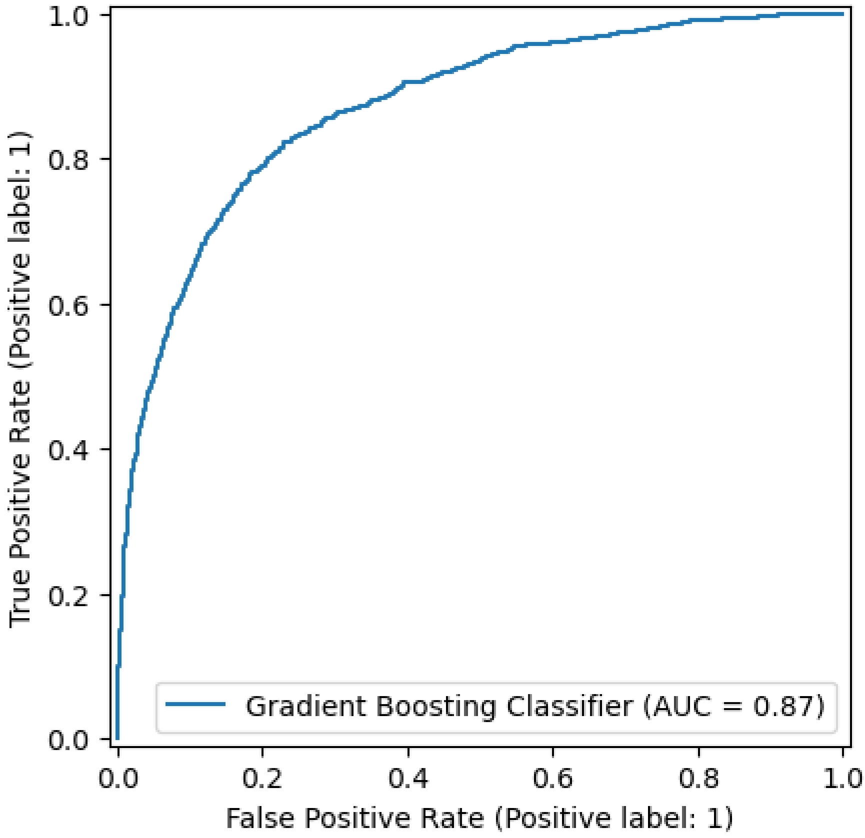
From the ROC curves comparison above, and confusion matrix of the method, we decided to choose Gradient Boosting Classifier as our final model.

In [ ]:
```
final_model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', GradientBoostingClassifier(learning_rate =0.1, n_estim
                                              max_depth=5,
                                              random_state=42))
])
output_final_model = final_model.fit(X_tr, y_tr)
evaluate(final_model, X_tr, X_hold, y_tr, y_hold)
```

```
Training Scores:
Train Accuracy: 0.9002457289960215
Train Precision: 0.8543046357615894
Train Recall: 0.6394932525475076
Train F1-Score: 0.731453772247598
**********
Testing Scores:
Test Accuracy: 0.845376263571696
Test Precision: 0.6841477949940405
Test Recall: 0.505726872246696
Test F1-Score: 0.5815602836879432
```



In [ ]:
```
# Plotting the roc-auc curve for the XGB model
RocCurveDisplay.from_estimator(final_model, X_te, y_te, name='Gradient Bo
plt.show()
```

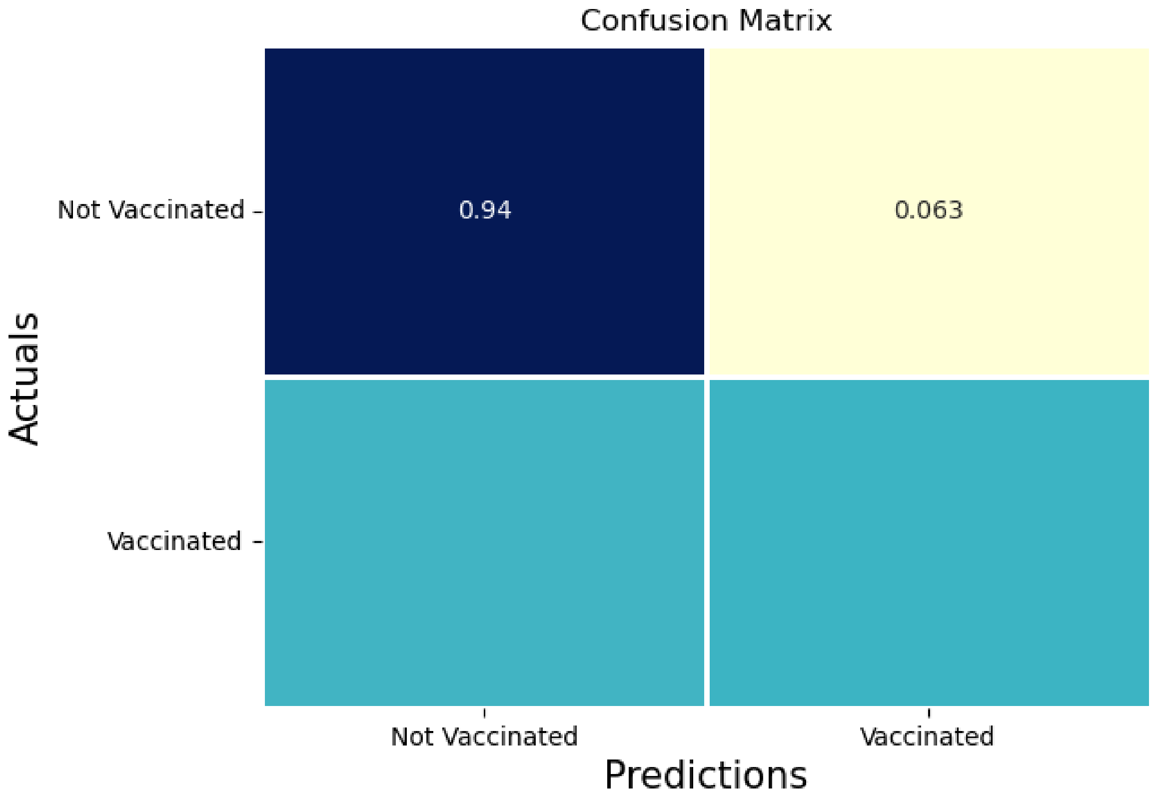


In [ ]:

```
# we can calculate predictions for confusion matrix
hold_preds_final_model = final_model.predict(X_hold)
cm=confusion_matrix(y_hold, hold_preds_final_model, normalize='true')
```

In [ ]:

```
# Plot heatmap for final model's confusion matrix for better visualizatic
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax=ax, cbar=False, linewidths=1, cmap="YlGnBu

ax.set_title('Confusion Matrix')
ax.set_xlabel('Predictions', fontsize = 15)
ax.set_ylabel('Actuals', fontsize = 15)
ax.xaxis.set_ticklabels(['Not Vaccinated', 'Vaccinated'])
ax.yaxis.set_ticklabels(['Not Vaccinated', 'Vaccinated'], rotation = 0, f
```



Evaluation

Our baseline model had an accuracy score of 78%, but a score of zero for precision, recall, and fl scores. When we compare all of our following models to this baseline, all have much better precision, recall, and f1 scores, and many have higher accuracy scores. The decision tree model is not overfitting, but it has a low precision score, as well as a low f1 score. However, it has an AUC score of 0.84, which is fairly high, meaning that it does an adequate job of maximizing true positives and minimizing the false positives. The decision tree model is not overfitting. This logistic regression model has low precision and fl scores, and has an AUC equal to the decision tree AUC above. This model is not overfitting. The random forest classifier model also has low precision and fl scores. It has an AUC score of 0.85, which is slightly better than the decision tree model above. This model is not overfitting to a great extent. The KNN model is definitely overfitting; the training data has perfect scores for all metrics, whereas the testing data scores are much lower. The AUC score is also much lower than on previous models. The Gradient Boosting model has the highest overall scores of all the models done so far, and also does the best job of minimizing the false positives. This is our candidate for the final model. The XG Boost model gave us similar scores to the gradient boosting model, but the gradient boosting model has the best AUC score and precision score. So, we will choose the gradient boosting classifier as the final model.

Our final model did not overfit to the training set, we got similar AUC, precision and accuracy scores for the holdout set. Because the model does a good job of minimizing the false positive rate on the hold out data, we are fairly confident that it will generalize well to unseen data and will accurately help public health offials determine the people who didn't get the vaccine. We are going to look into feature importances to understand the relationship between the features and vaccination behavior.