

# Hearth Disease recover planner

Francesco Buffalmano 279132, Mattia Lupinetti 278712

June 2024

## 1 Abstract

In questo lavoro, presentiamo un Pianificatore di Recupero dalle Malattie Cardiache basato su tecniche avanzate di machine learning. Il sistema è progettato per assistere i professionisti della salute nel monitoraggio e nella pianificazione del recupero dei pazienti affetti da malattie cardiache. Il nucleo del nostro software è un modello di machine learning supervisionato di classificazione il quale predirrà se un paziente è a rischio di malattie cardiache. Il Pianificatore di Recupero dalle Malattie Cardiache rappresenta uno strumento per migliorare i risultati clinici e la qualità della vita dei pazienti.

## 2 Introduction

Le malattie cardiache rappresentano una delle principali cause di mortalità a livello globale, richiedendo interventi medici tempestivi e piani di recupero personalizzati per migliorare la prognosi dei pazienti. La complessità nella gestione delle malattie cardiache deriva dalla necessità di monitorare un vasto numero di variabili cliniche e di rispondere in modo adeguato alle specifiche esigenze di ciascun paziente.

La malattia cardiaca (malattia del cuore) è un insieme di patologie legate alle malattie cardiovascolari, manifestate da una violazione del normale funzionamento del cuore. Può essere causata da danni all'epicardio, pericardio, mio-cardio, endocardio, apparato valvolare del cuore e ai vasi sanguigni del cuore. La malattia cardiaca può durare a lungo in forma latente, senza manifestarsi clinicamente. Insieme a vari tumori, queste malattie rappresentano oggi la principale causa di morte prematura nei paesi sviluppati. Il funzionamento ininterrotto del sistema circolatorio, composto dal cuore come pompa muscolare e da una rete di vasi sanguigni è una condizione necessaria per il normale funzionamento del corpo. Secondo il National Heart, Lung and Blood Institute di Framingham (USA), i fattori più importanti nello sviluppo delle malattie cardiovascolari negli esseri umani sono l'obesità, lo stile di vita sedentario e il fumo. In questo contesto, l'uso delle tecnologie di machine learning offre nuove opportunità per sviluppare strumenti avanzati che supportano i professionisti della salute nella pianificazione del recupero e nel miglioramento dei risultati

clinici. Questo progetto introduce un Pianificatore di Recupero dalle Malattie Cardiache basato su tecniche avanzate di machine learning, con l'obiettivo di fornire previsioni accurate e assistenza personalizzata ai pazienti. Il nucleo del sistema è costituito da un modello di KNeighborsClassifier, noto per la sua robustezza e la capacità di gestire dataset complessi con numerose variabili. La scelta di KNeighborsClassifier è motivata dalla sua efficienza nell'elaborare dati eterogenei e dalla sua resistenza al rischio di overfitting, garantendo così previsioni affidabili. In sintesi, questo strumento non solo supporta i professionisti della salute nella pianificazione del recupero dei pazienti, ma contribuisce anche a migliorare la qualità della vita dei pazienti attraverso una gestione più efficace e personalizzata del loro percorso di recupero.

## 3 Materials and Methods

### 3.1 Librerie

Nel seguente frammento di codice vengono mostrate le librerie utilizzate. In particolare:

- pickle - libreria che consente di salvare e caricare un modello preaddestrato
- pandas, numpy - librerie che consentono di manipolare i dati (in particolare la prima ci consente di creare strutture dati come DataSet e Series, la seconda ci permette di manipolare array ed effettuare operazioni matematico complesse).
- sklearn (scikit-learn) - libreria che presenta vari modelli di machine learning, permette la scelta delle features migliori, ne calcola l'importanza.
- Imbalanced-learn (imblearn) - libreria Python progettata per affrontare il problema della classificazione in dataset con classi sbilanciate
- xgboost, lightgbm - altri modelli di machine learning.
- Tkinter - una libreria standard di Python utilizzata per creare interfacce grafiche utente (GUI). È un'interfaccia per il toolkit GUI Tk, che è ampiamente utilizzato e ben supportato.

### 3.2 Dataset

Per il nostro Pianificatore di Recupero dalle Malattie Cardiache, abbiamo utilizzato il dataset "Heart Disease" disponibile su Kaggle. Questo dataset comprende una serie di variabili cliniche rilevanti per la diagnosi e la prognosi delle malattie cardiache, come età, sesso, livelli di colesterolo, pressione sanguigna e risultati di vari test diagnostici. Il nostro obiettivo era creare un modello di machine learning che potesse prevedere con precisione il rischio di malattie cardiache e supportare la pianificazione del recupero dei pazienti. Il dataset utilizzato è disponibile su Kaggle all'indirizzo: Kaggle. Questo dataset contiene

informazioni dettagliate su vari aspetti clinici dei pazienti, inclusi i seguenti attributi:

- HeartDisease - malattia cardiaca, tratto di interesse.
- BMI - un valore che consente di valutare il grado di corrispondenza tra la massa corporea di una persona e la sua altezza, e quindi giudicare indirettamente se la massa è insufficiente, normale o eccessiva. È importante per determinare le indicazioni per il trattamento.
- Smoking - il fumo è un fattore di rischio importante per le malattie cardiovascolari. Quando si inhala il fumo di una sigaretta, la reazione del sistema cardiovascolare segue immediatamente: entro un minuto, la frequenza cardiaca inizia a salire, aumentando del 30% entro dieci minuti dal fumo. Questa cattiva abitudine aumenta anche la pressione sanguigna, i livelli di fibrinogeno e piastrine, aumentando il rischio di coaguli sanguigni.
- AlcoholDrinking - l'alcol causa non solo disturbi temporanei nel funzionamento del cuore, ma anche disturbi permanenti. Il dolore al cuore dopo l'assunzione di alcol non è l'unico problema di salute associato al consumo di alcol.
- Stroke - l'ictus ischemico si verifica 4 volte più spesso dell'emorragico. Una delle principali cause di questa sofferenza è la malattia cardiaca, che compromette il suo funzionamento, con conseguente disturbo del flusso sanguigno nelle arterie e riduzione dell'apporto di sangue al cervello. Un'altra causa di ictus nelle malattie cardiache è il tromboembolismo, quando si formano coaguli nelle cavità del cuore (più frequentemente con l'insufficienza cardiaca) - coaguli di sangue.
- PhysicalHealth - quanti giorni al mese hai avvertito cattiva salute fisica.
- MentalHealth - quanti giorni al mese hai avvertito cattiva salute mentale.
- DiffWalking - difficoltà a salire le scale.
- Sex - genere di una persona.
- AgeCategory - categoria di età dei soggetti.
- Diabetic - diabete.
- PhysicalActivity - adulti che hanno dichiarato di aver praticato attività fisica o esercizio fisico durante gli ultimi 30 giorni al di fuori del loro lavoro regolare.
- GenHealth - stato di benessere generale.
- SleepTime - numero di ore di sonno.
- Asthma - asma.

- KidneyDisease - malattia renale.
- Skin Cancer - cancro della pelle.

Le variabili numeriche sono BMI, PhysicalHealth, MentalHealth, SleepTime. Il resto è categorico.

### 3.3 Dataset sbilanciato

Per garantire che il modello di machine learning potesse apprendere in modo efficace da tutte le classi presenti nel dataset, abbiamo applicato un preprocessamento dettagliato, incluso il bilanciamento delle classi tramite l'algoritmo di RandomOverSampler. Questo passaggio è stato essenziale per affrontare la natura sbilanciata del dataset, dove alcune classi erano significativamente più rappresentate rispetto ad altre.

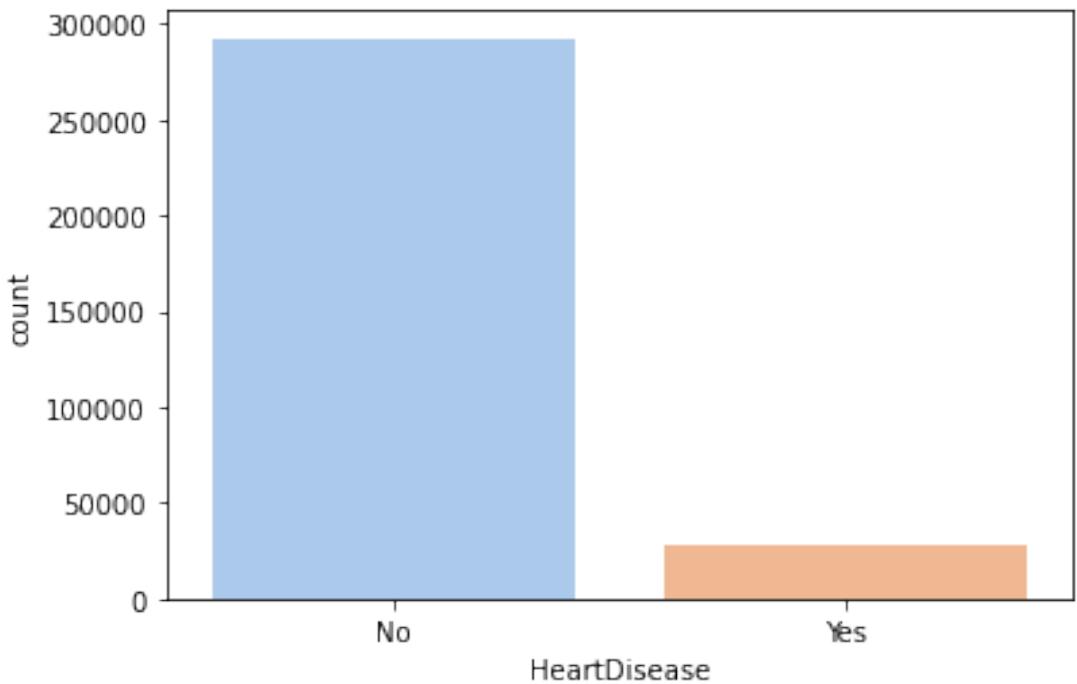


Figure 1: Siamo di fronte ad un campione sbilanciato, dove la maggioranza delle persone sono sane (circa il 90%).

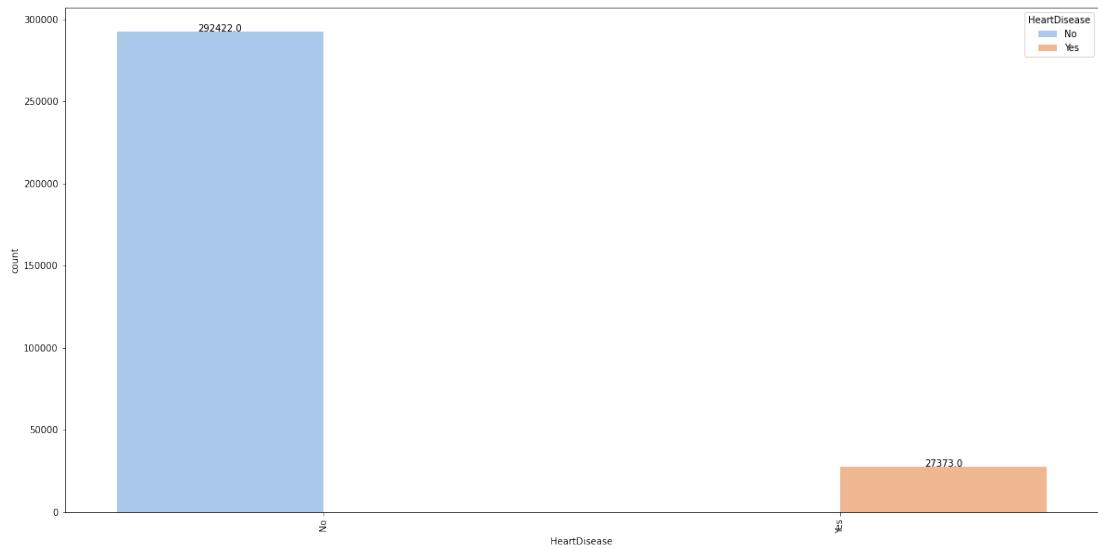


Figure 2: HeartDisease

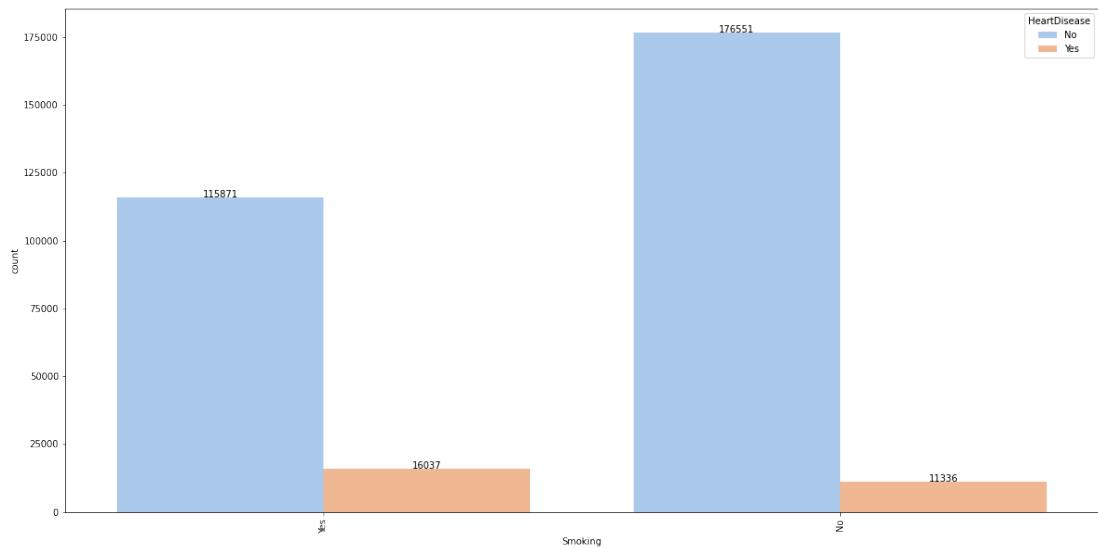


Figure 3: Smoking

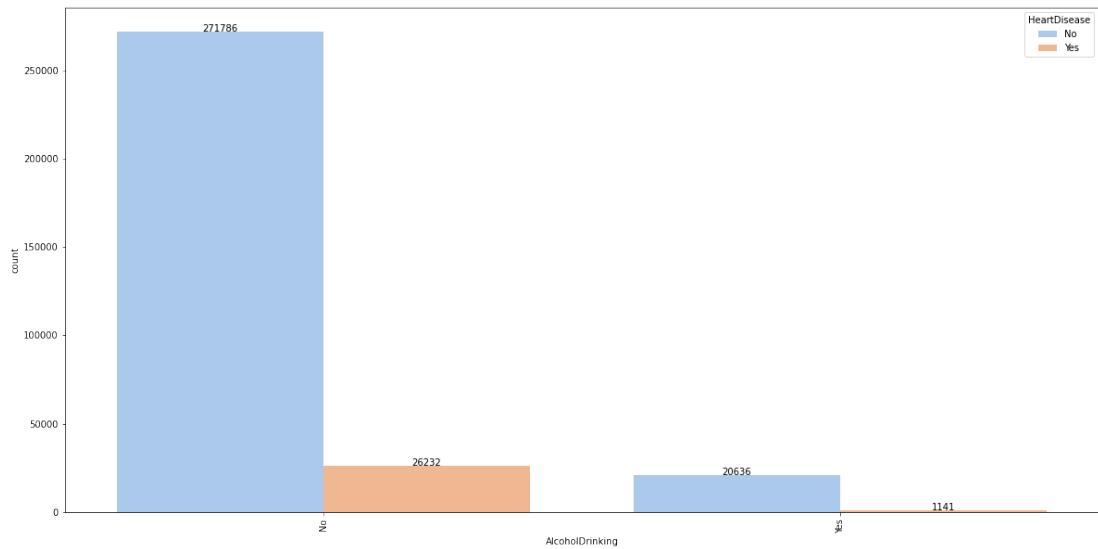


Figure 4: AlcholDrinking

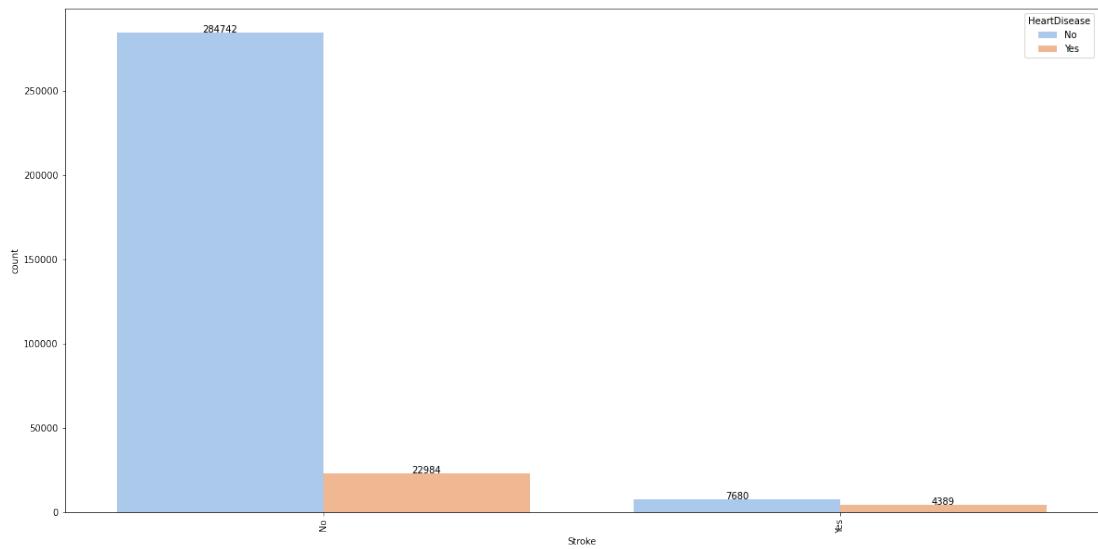


Figure 5: DiffWalking

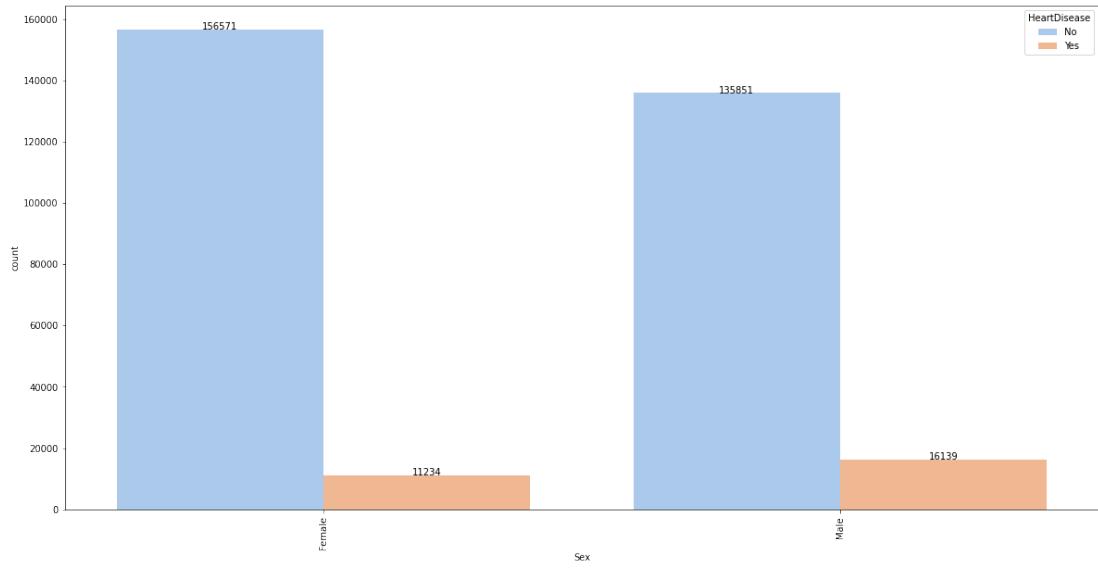


Figure 6: Sex

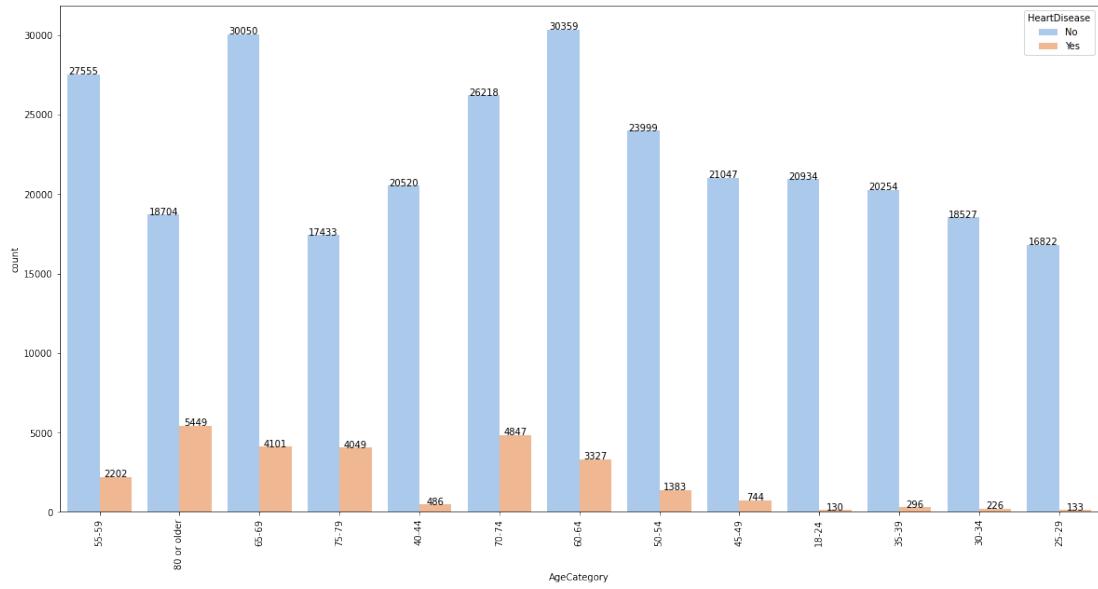


Figure 7: AgeCategory

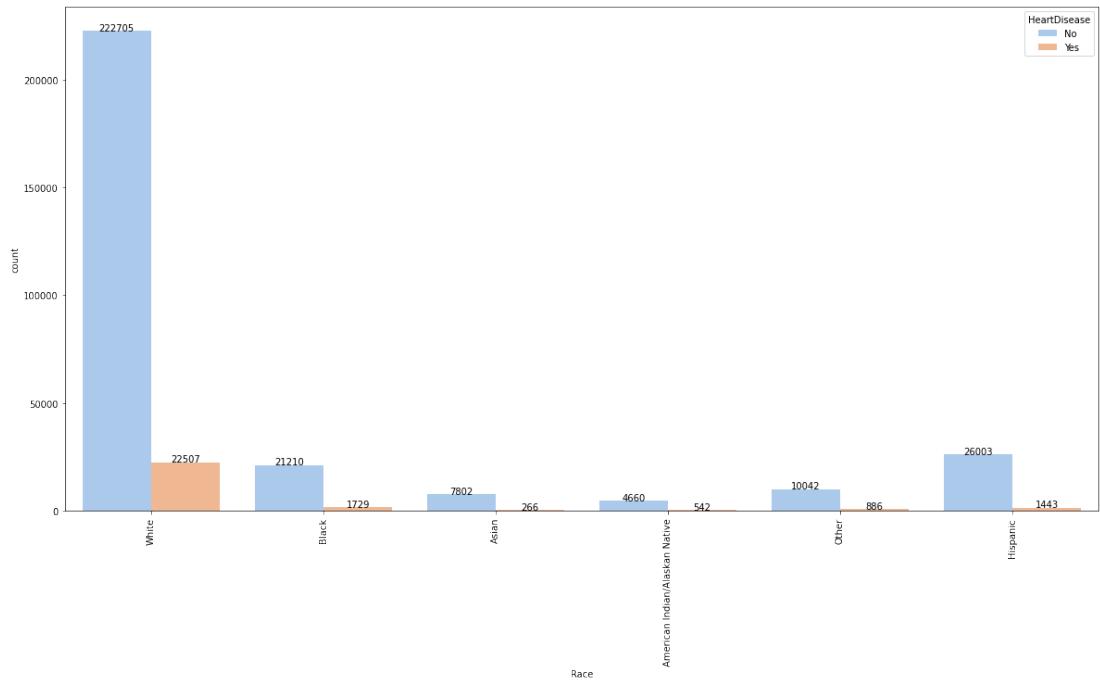


Figure 8: Race

### 3.4 Componenti fondamentali

Un componente fondamentale del nostro piano di recupero è il calcolo di uno stato finale del paziente da raggiungere attraverso un preciso planning in cui i suoi parametri, secondo il modello, lo rendono il paziente fuori pericolo. Per ottenere tale stato modifichiamo le feutures che la matrice di correlazione del nostro dataset ritiene più impattanti per il valore del nostro goal: HearthDisease.

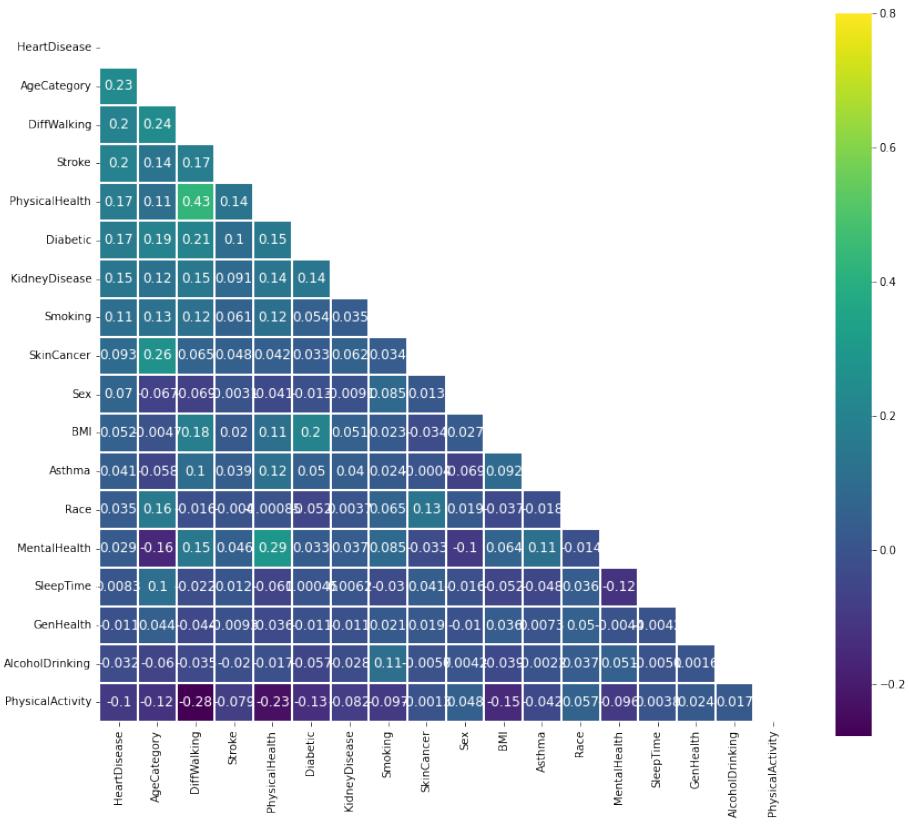


Figure 9: Matrice di correlazione del dataset originale

Per calcolare uno o più goal abbiamo creato la classe recoverCalculator:

```

import pandas as pd
import numpy as np
from modelLoader import load_model
from domainManager import is_normal_weight

class Recover_Calculator(object):

    def __init__(self, patient: dict, modelFileName: str):
        self.feature_indices = {
            "BMI": 0,
            "Smoking": 1,
            "AlcoholDrinking": 2,
            'Stroke': 3,
            "PhysicalHealth": 4,
            "MentalHealth": 5,
            "DiffWalking": 6,
            'Sex': 7,
            'AgeCategory': 8,
            'Race': 9,
            'Diabetic': 10,
            "PhysicalActivity": 11,
            'GenHealth': 12,
            "SleepTime": 13,
            "Asthma": 14,
            'KidneyDisease': 15,
            'SkinCancer': 16
        }

        self.patient = np.array([patient[feature] for feature in self.feature_indices])
        self.model = load_model(modelFileName)
        if self.has_heartDisease(self.patient):
            self.new_status, self.new_values = self.calculate_goals(self.patient)
        else:
            self.new_status = None

    def make_patient_test(self, patient_array):
        return pd.DataFrame([patient_array], columns=[feature for feature in self.feature_indices])

    def has_heartDisease(self, patient: np):
        return bool(self.model.predict(self.make_patient_test(patient)))

    def calculate_goals(self, patient):
        """
        The method modifies the boolean features and the numerical features
        in non-increasing order of their correlation with the target,
        and checks if the change in lifestyle or disease
        state removes the patient from danger.
        If so, adds the patient's data to the list of solutions.

        Args:
            patient (ndarray): initial patient status

        Returns:
            list : patient version
        """

```

```

healthy_patients = []
new_vals = {}
health_trends = {"PhysicalHealth", "MentalHealth"}
range_bin = {"DiffWalking", "Smoking", "PhysicalActivity", "Asthma"}
test = patient.copy()

for feature in range_bin:
    index = self.feature_indices[feature]
    if feature == "PhysicalActivity":
        if test[index] == 0:
            test[index] = 1
    else:
        test[index] = 0
    if self.has_heartDisease(test):
        new_vals[feature] = test[index]
    elif test not in healthy_patients:
        healthy_patients.append(test)

for feature in health_trends:
    index = self.feature_indices[feature]
    while test[index] > 0 and self.has_heartDisease(test):
        test[index] -= 1
    if self.has_heartDisease(test):
        new_vals[feature] = test[index]
    elif test not in healthy_patients:
        healthy_patients.append(test)

index = self.feature_indices["SleepTime"]
while test[index] <= 7 and self.has_heartDisease(test):
    test[index] += 1
if self.has_heartDisease(test):
    new_vals[feature] = test[index]
elif test not in healthy_patients:
    healthy_patients.append(test)

index = self.feature_indices["BMI"]
while not is_normal_weight(test[index]) and self.has_heartDisease(test):
    test[index] -= 1
if self.has_heartDisease(test):
    new_vals[feature] = test[index]
elif test not in healthy_patients:
    healthy_patients.append(test)

return healthy_patients, new_vals

def get_new_status(self):
    return self.new_status

```

---

Nel metodo calculate\_goals() modifichiamo i parametri del paziente legati allo stile di vita o a malattie da cui il paziente può guarire (ci siamo quindi astenuti dal modoficare features come "SkinCancer", "KidneyDisease" o "Diabetic").

In seguito al calcolo dei goal c'è bisogno di creare un planning problem la cui soluzione sarà trovata da un algoritmo di ricerca, nel nostro caso un algoritmo di ricerca MPP (Maximal Pattern Search):

```

from domainManager import setFlags, get_bmi, get_overweight, is_sickly, is_old, is_depressed, is_gt_28, is_obese_st, is_obese_nd, is_obese_rd, is_overweight,
from stripsProblem import Strips, STRIPS_domain, Planning_problem
from stripsForwardPlanner import Forward_STRIPS
from searchMPP import SearcherMPP
import numpy as np

class RehabilitationPlanner(object):
    def __init__(self, patient: dict, g_patient: dict, height: float, weight: int) -> None:
        self.height = height
        self.weight = weight
        self.feature_indices = {
            "BMI": 0,
            "Smoking": 1,
            "AlcoholDrinking": 2,
            'Stroke': 3,
            "PhysicalHealth": 4,
            "MentalHealth": 5,
            "DiffWalking": 6,
            'Sex': 7,
            'AgeCategory': 8,
            'Race': 9,
            'Diabetic': 10,
            "Physicalactivity": 11,
            'GenHealth': 12,
            "SleepTime": 13,
            "Asthma": 14,
            'KidneyDisease': 15,
            'SkinCancer': 16
        }

    self.patient = np.array([patient[feature] for feature in self.feature_indices])
    self.g_patient = np.array([g_patient[feature] for feature in self.feature_indices])
    self.STRIPS_domain_feature = {'BMI_obese_I': boolean_domain, 'BMI_obese_II': boolean_domain, 'BMI_obese_III': boolean_domain,
                                   'BMI': BMI_value_domain, 'bmi_gt_28': boolean_domain, 'is_overweight': boolean_domain,
                                   'is_depressed': boolean_domain, 'is_sickly': boolean_domain, 'is_sleepless': boolean_domain,
                                   'is_old': boolean_domain, 'bariatric_surgery': boolean_domain,
                                   'Smoking': boolean_domain, 'AlcoholDrinking': boolean_domain,
                                   'Stroke': boolean_domain, 'PhysicalHealth': physical_health_domain,
                                   'MentalHealth': mental_health_domain, 'DiffWalking': boolean_domain,
                                   'Sex': boolean_domain, 'AgeCategory': ageCategory_domain, 'Race': race_domain,
                                   'Diabetic': boolean_domain, 'PhysicalActivity': boolean_domain,
                                   'SleepTime': sleep_time_domain, 'GenHealth': genHealth_domain,
                                   'Asthma': boolean_domain, 'KidneyDisease': boolean_domain,
                                   'SkinCancer': boolean_domain}
    self.bmi = get_bmi(height, weight)
    self.initial_patient = self.initialize_patient(patient, g_patient, BMI_flags, bin_features)
    self.goal_patient = self.initialize_goal_patient(g_patient, BMI_flags, bin_features)

    self.actions = [
        Strips('bariatric_surgery', {'BMI_obese_III': True, 'bariatric_surgery': False},
               {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI'])},
               'BMI_obese_III': is_obese_rd(self.bmi),
               'BMI_obese_II': is_obese_nd(self.bmi),
               'BMI_obese_I': is_obese_st(self.bmi),
               'bariatric_surgery': True}),
    ]

```

```

    Strips('bariatric_surgery_with_comorbidity', {'is_overweight': False, 'BMI_obese_II': True, 'Diabetic': True, 'bariatric_surgery': False},
           {'Diabetic': True, 'bariatric_surgery': True,
            'BMI_obese_II': False, 'BMI_obese_I': True,
            'BMI': self.initialize_BMI_effects(self.goal_patient['BMI'])}),

    Strips('pharmacological_treatment_Saxenda_liraglutide_post_bariatric_surgery_and_hypocaloric_diet_1',
           {'BMI_obese_II': True, 'bariatric_surgery': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_II': is_obese_nd(self.bmi),
            'BMI_obese_I': is_obese_st(self.bmi)}),

    Strips('pharmacological_treatment_Saxenda_liraglutide_obese_II', {'BMI_obese_II': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_II': is_obese_nd(self.bmi),
            'BMI_obese_I': is_obese_st(self.bmi),
            'bmi_gt_28': is_gt_28(self.bmi),
            'is_overweight': is_overweight(self.bmi)}),

    Strips('pharmacological_treatment_Saxenda_liraglutide_post_bariatric_surgery_and_hypocaloric_diet_2',
           {'BMI_obese_I': True, 'bariatric_surgery': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_I': is_obese_st(self.bmi),
            'bmi_gt_28': is_gt_28(self.bmi),
            'is_overweight': is_overweight(self.bmi)}),

    Strips('stop_AlcoholDrinking', {'AlcoholDrinking': True}, {'AlcoholDrinking': False}),
    Strips('stop_smoking', {'Smoking': True}, {'Smoking': False}),

```

```

    Strips('pharmacological_treatment_Saxenda_liraglutide', {'BMI_obese_I': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_I': is_obese_st(self.bmi),
            'bmi_gt_28': is_gt_28(self.bmi),
            'is_overweight': is_overweight(self.bmi)}),

    Strips('pharmacological_treatment_Saxenda_liraglutide_with_comorbidity',
           {'bmi_gt_28': True, 'Diabetic': True, 'is_overweight': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'bmi_gt_28': is_gt_28(self.bmi),
            'is_overweight': is_overweight(self.bmi)}),

    Strips('rehabilitation', {'DiffWalking': True}, {'DiffWalking': False}),
    Strips('physical_activity', {'DiffWalking': False, 'PhysicalActivity': False},
           {'PhysicalActivity': True,
            'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'is_overweight': is_overweight(self.bmi)}),

    Strips('mental_therapy', {'is_depressed': True, 'MentalHealth': self.initial_patient['MentalHealth']},
           {'is_depressed': False, 'MentalHealth': self.goal_patient['MentalHealth']}),

    Strips('physical_therapy', {'is_sickly': True},
           {'is_sickly': False, 'PhysicalHealth': self.goal_patient['PhysicalHealth']}),

    Strips('Asthma_therapy', {'Asthma': True}, {'Asthma': False}),
    Strips('increase_hours_of_sleep', {'is_sleepless': True},
           {'is_sleepless': is_sleepless(self.goal_patient['SleepTime']), 'SleepTime': self.goal_patient['SleepTime']}),

```

```

    Strips('pharmacological_treatment_liraglutide_overweight_with_comorbidity',
           {'is_overweight': True, 'is_old': True, 'Diabetic': True},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']), 'is_overweight': False}),

    Strips('improve_lifestyle1',
           {'BMI_obese_II': True, 'DiffWalking': False, 'is_old': False, 'physical_activity': False, 'GenHealth': 'Good'},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_II': is_obese_nd(self.bmi),
            'BMI_obese_I': is_obese_st(self.bmi),
            'physical_activity': True}), # obese II in good health

    Strips('improve_lifestyle2',
           {'BMI_obese_II': True, 'DiffWalking': False, 'is_old': False, 'physical_activity': False, 'GenHealth': 'Very Good'},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_II': is_obese_nd(self.initialize_BMI_effects(self.bmi)),
            'BMI_obese_I': is_obese_st(self.initialize_BMI_effects(self.bmi)),
            'physical_activity': True}), # obese II in very good health

    Strips('improve_lifestyle3',
           {'BMI_obese_II': True, 'DiffWalking': False, 'is_old': False, 'physical_activity': False, 'GenHealth': 'Excellent'},
           {'BMI_obese_II': is_obese_nd(self.initialize_BMI_effects(self.goal_patient['BMI'])),
            'BMI_obese_I': is_obese_st(self.bmi),
            'physical_activity': True, 'BMI': self.bmi}),

    Strips('improve_hypocaloric_diet', {'is_overweight': True, 'Diabetic': False},
           {'BMI': self.initialize_BMI_effects(self.goal_patient['BMI']),
            'BMI_obese_II': is_obese_nd(self.bmi),
            'BMI_obese_I': is_obese_st(self.bmi),
            'physical_activity': True, 'bmi_gt_28': is_gt_28(self.bmi)})


    self.rehabilitation_planning = Planning_problem(STrips_domain(self.STrips_domain_feature, self.actions), self.initial_patient, self.goal_patient)

def initialize_patient(self, patient: np, g_patient: np, BMI_flags: list, bin_features: list) -> dict:
    """Initialize the patient by converting features represented by 1 and 0 into boolean values and adding flags useful for planning

    Args:
        patient (np): original patient values
        g_patient (np): Values that the patient needs to reach to avoid the risk of heart diseases
        BMI_flags (list): Flags representing the BMI levels to initialize
        bin_features (list): patient's binary features

    Returns:
        dict: returns a dict that represents a patient in a planning problem
    """

    initial_patient = {}

    for k in patient.keys():
        if k in bin_features:
            initial_patient[k] = bool(patient[k])
        else:
            initial_patient[k] = patient[k]
    for i in range(0, len(BMI_flags)):
        initial_patient[BMI_flags[i]] = setFlags(initial_patient['BMI'], BMI_flags[i])

```

```

    return initial_patient

def initialize_goal_patient(self, g_patient: dict, BMI_flags: list, bin_features: list) -> dict:
    """Initialize the goal of patient by converting features represented by 1 and 0 into boolean values and adding flags useful for planning

    Args:
        g_patient (np): Values that the patient needs to reach to avoid the risk of heart diseases
        BMI_flags (list): Flags representing the BMI levels to initialize
        bin_features (list): patient's binary features

    Returns:
        dict: returns a dict that represents the goal of the patient in a planning problem
    """
    goal_patient = dict()
    for k in g_patient.keys():
        if k in bin_features:
            |   goal_patient[k] = bool(g_patient[k])
        else:
            |   goal_patient[k] = g_patient[k]

    for i in range(0, len(BMI_flags)):
        goal_patient[BMI_flags[i]] = setFlags(goal_patient['BMI'], BMI_flags[i])

    goal_patient['is_depressed'] = is_depressed(goal_patient['MentalHealth'], g_patient['MentalHealth'])
    goal_patient['is_sickly'] = is_sickly(goal_patient['PhysicalHealth'], g_patient['PhysicalHealth'])
    goal_patient['is_sleepless'] = is_sleepless(goal_patient['SleepTime'])
    goal_patient['is_old'] = is_old(goal_patient['AgeCategory'])

    return goal_patient

```

---

```

def bmi_post_bariatric_surgery(self) -> float:
    """Following an intervention, a patient loses between 50% and 70% of the excess weight; we return the BMI case in the worst scenario

    Returns:
        int: new BMI value
    """
    self.bmi = get_bmi(self.height, (self.weight - get_overweight(self.bmi, self.height) / 2))
    return self.bmi

```

---

```

def initialize_BMI_effects(self, BMI_goal) -> float:
    """Initialize BMI values in the effects of each action in the STRIPS problem for a specific patient

    Args:
        BMI_goal (float): BMI value to reach

    Returns:
        float: Depending on the BMI level, an intermediate or final goal to achieve is returned
    """
    if is_obese_rd(self.bmi) and is_obese_rd(BMI_goal):
        self.bmi = BMI_goal
        return BMI_goal
    elif is_obese_rd(self.bmi):
        return self.bmi_post_bariatric_surgery()

```

```

    elif is_obese_nd(self.bmi):
        if self.initial_patient['Diabetic']:
            return self.bmi_post_bariatric_surgery()
        else:
            self.bmi = max_obese_I
        return max_obese_I
    if is_obese_st(self.bmi) and is_obese_st(BMI_goal):
        self.bmi = BMI_goal
        return BMI_goal
    elif is_obese_st(self.bmi):
        self.bmi = max_overweight
        return max_overweight
    if is_overweight(self.bmi) and is_overweight(BMI_goal):
        self.bmi = BMI_goal
        return BMI_goal
    elif is_overweight(self.bmi):
        self.bmi = max_normal_weight
        return max_normal_weight
    self.bmi = BMI_goal

    return BMI_goal


def get_plan(self):
    """returns the plans to reach the goals

    Returns:
    | list: list of plans
    """
    searcher = SearcherMPP(Forward_STRIIPS(self.rehabilitation_planning)) # A* with MPP
    searcher.search()
    return searcher.solution, searcher.actions, searcher.states

```

Il BMI dei paziente viene classificato, il livello del BMI momentaneo induce a uno specifico trattamento. In base al BMI può essere distinta in: -obesità di I grado (BMI 30-35) -obesità di II grado (BMI 35-40) -obesità di III grado (severa) (BMI $\geq$ 40) I livelli estremi di BMI sono trattati da farmaci. FARMACI OBESITÀ:

- Orlistat: – pazienti obesi con un BMI superiore a 30 kg/m<sup>2</sup> – pazienti con BMI superiore a 28 kg/m<sup>2</sup> ma con concomitanti fattori di rischio, in associazione ad una dieta moderatamente ipocalorica.
- Liraglutide; - è stata autorizzata anche nel trattamento dell'obesità, ma alla dose di 3 mg/die in pazienti adulti con un BMI 30 kg/m<sup>2</sup> o tra 27 e 30 kg/m<sup>2</sup> con comorbidità correlate al peso (iperglycemia o diabete tipo 2, ipertensione arteriosa, dislipidemia, apnee notturne).

FONTE: Dott. Massimiliano Andrioli

## 4 Experiments and Results

### 4.1 Valutazione del Modello

Il modello è stato valutato utilizzando una serie di metriche chiave:

- Accuratezza: La proporzione di previsioni corrette su tutte le previsioni effettuate, è una misura complessiva della correttezza del modello, ma può non essere adeguata in presenza di classi sbilanciate
- Precisione: La proporzione di vere positività su tutte le previsioni positive, è utile quando è importante ridurre al minimo i falsi positivi, ad esempio in problemi dove gli errori positivi hanno costi elevati.
- Richiamo (Recall): La proporzione di vere positività su tutte le effettive positività, è utile quando è importante ridurre al minimo i falsi negativi, ad esempio in problemi dove perdere un caso positivo può avere conseguenze gravi.
- F1-score: La media armonica della precisione e del richiamo.

Quelli che seguono sono le percentuali di accuratezza, precisione, recall e F1 su i seguenti modelli:

#### 4.1.1 KNeighborsClassifier

- model: KNeighborsClassifier()
- Accuracy score: 0.9025328330206379
- Precision score: 0.3203463203463203
- Recall score: 0.07740585774058577
- F1-score: 0.12468407750631845

#### 4.1.2 LogisticRegression

- model: LogisticRegression()
- Accuracy score: 0.9115697310819262
- Precision score: 0.5423728813559322
- Recall score: 0.08926080892608089
- F1-score: 0.1532934131736527

#### 4.1.3 XGBClassifier

- model: XGBClassifier
- Accuracy score: 0.9121013133208256
- Precision score: 0.5580448065173116
- Recall score: 0.09553695955369595
- F1-score: 0.16314379279547486

#### 4.1.4 ExtraTreesClassifier

- model: ExtraTreesClassifier()
- Accuracy score: 0.8931519699812382
- Precision score: 0.30462633451957294
- Recall score: 0.1492329149232915
- F1-score: 0.20032763866136205

## 4.2 Conclusioni Valutazione del modello

Possiamo notare facilmente che l'Accuracy score è molto elevata per tutti i modelli ma essendo questa un'applicazione medica poniamo l'accento sulle metriche riguardanti le previsioni positive: nelle prestazioni dei modelli sul dataset originale la precision e la recall non superano mai il 56% e il 16% quindi nel caso migliore avremo una precision di pochi punti superiore al 50% ma con un altissimo tasso di falsi negativi che è esattamente il caso che più vorremmo evitare in un'applicazione di carattere medico. Constatato ciò abbiamo deciso di applicare sul dataset le seguenti classi della libreria `imblearn`:

### 4.2.1 RandomUnderSampler

#### Descrizione:

RandomUnderSampler è una tecnica di bilanciamento dei dati che agisce riducendo casualmente il numero di campioni dalla classe maggioritaria nel dataset. Questo approccio è utile quando si desidera ridurre l'eccesso di campioni della classe dominante per equilibrare il dataset.

#### Funzionamento:

1. Seleziona casualmente un sottoinsieme dei campioni dalla classe maggioritaria.
2. Mantiene solo i campioni selezionati nella nuova versione del dataset.
3. È semplice da implementare ma potrebbe portare a una perdita di informazioni se i campioni importanti vengono eliminati.

#### 4.2.2 RandomOverSampler

##### Descrizione:

RandomOverSampler è un'altra tecnica di bilanciamento che opera aumentando casualmente il numero di campioni della classe minoritaria nel dataset. Questo metodo è efficace per migliorare la rappresentanza delle classi meno frequenti.

##### Funzionamento:

1. Seleziona casualmente campioni dalla classe minoritaria.
2. Replica questi campioni per aumentare il numero di istanze della classe minoritaria.
3. Può portare a un aumento del rischio di overfitting se non viene controllato adeguatamente.

#### 4.2.3 SMOTE (Synthetic Minority Over-sampling Technique)

##### Descrizione:

SMOTE è una tecnica di sovraccampionamento progettata per affrontare il problema della classe minoritaria creando nuovi campioni sintetici anziché replicare quelli esistenti. Utilizza l'interpolazione tra i campioni della classe minoritaria per generare nuovi esempi.

##### Funzionamento:

1. Identifica i campioni della classe minoritaria vicini nel feature space.
2. Crea nuovi campioni sintetici tra i campioni esistenti della classe minoritaria.
3. Migliora la rappresentazione della classe minoritaria senza duplicare i campioni esistenti, riducendo il rischio di overfitting.

#### 4.2.4 SMOTE-ENN (SMOTE + Edited Nearest Neighbors)

##### Descrizione:

SMOTE-ENN è una combinazione di SMOTE e un'operazione di editing dei campioni della classe maggioritaria. Questo metodo non solo sovraccampaona la classe minoritaria ma elimina anche i campioni della classe maggioritaria che sono vicini ai campioni della classe minoritaria.

##### Funzionamento:

1. Utilizza SMOTE per generare campioni sintetici della classe minoritaria.
2. Applica Edited Nearest Neighbors (ENN) per eliminare i campioni della classe maggioritaria che sono classificati in modo errato vicino ai campioni della classe minoritaria.
3. Aiuta a ridurre il rumore introdotto da campioni sintetici non rappresentativi.

#### **4.2.5 ADASYN (Adaptive Synthetic Sampling)**

##### **Descrizione:**

ADASYN è una tecnica di sovraccampionamento che genera campioni sintetici per la classe minoritaria con una distribuzione proporzionale alla densità locale dei campioni vicini.

##### **Funzionamento:**

1. Calcola la densità locale dei campioni minoritari.
2. Genera campioni sintetici per la classe minoritaria con una densità proporzionale alla densità locale.
3. Crea un bilanciamento più fine rispetto a SMOTE in aree del feature space dove la densità dei campioni è variabile.

Lo script python utilizzato per bilanciare i dataset è `data_resampler.py` e salvano i nuovi dataset in `./data/`

#### 4.2.6 data\_resampler.py

```
import numpy as np
from sklearn.model_selection import train_test_split
import numpy as np
from imblearn.over_sampling import RandomOverSampler,SMOTE, ADASYN
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN
import pandas as pd
import pickle
from sklearn.preprocessing import OrdinalEncoder
from domainManager import features

categorical_features=['HeartDisease']
categorical_features.append(features)
resampling_methods = [
    RandomUnderSampler(sampling_strategy='majority', random_state=42),
    SMOTE(sampling_strategy='minority', random_state=42),
    ADASYN(sampling_strategy='minority', random_state=42),
    SMOTEENN(sampling_strategy='minority', random_state=42),
    RandomOverSampler(sampling_strategy='minority', random_state=42)
]

def convert_data(data,categorical_f,target,split=False):
    enc = OrdinalEncoder()
    enc.fit(data[categorical_f])
    data[categorical_f] = enc.transform(data[categorical_f])
    if split:
        y = data[target]
        data.drop(target, axis=1,inplace=True)
        X_train, X_test, y_train, y_test=train_test_split(data,y,test_size=0.1,random_state=42)
        return X_train, X_test, y_train, y_test
    return data
```

```

def dataResampler(X,y, methods,target):
    i = 0
    data = dict()
    for m in methods:
        X,y = m.fit_resample(X, y)
        X[target] = y
        data.update({i:X})
        df = pd.DataFrame(X)
        df.to_csv('./data/heart_2020_cleaned_' + str(m)+'.csv')
        i = i + 1
    return data

target = "HeartDisease"

data = pd.read_csv("./data/heart_2020_cleaned.csv")

data = convert_data(data,categorical_features,target, split=False)
y = data[target]
data.drop(target, axis=1,inplace=True)
data = dataResampler(data,y,resampling_methods,target)

```

#### 4.2.7 Test in seguito al bilanciamento

Infine, abbiamo confrontato le prestazioni del nostro modello con i test sui modelli presenti nel notebook da cui abbiamo prelevato il dataset: in seguito al bilanciamento dei dati, come riportato nel file `prestazione_modelli.txt`, abbiamo ottenuto un notevole miglioramento delle prestazioni. Poniamo l'accento sulle metriche quali il richiamo (Recall) e la precision come metriche principali per valutare le prestazioni dei modelli. Questo perché, nel contesto delle malattie cardiache, è fondamentale porre la massima attenzione nell'identificare il maggior numero di casi positivi.

```

from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score,recall_score,accuracy_score
from sklearn.metrics import f1_score
from sklearn.tree import DecisionTreeClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from modelLoader import save_model
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder
from domainManager import features

```

```

models = [
    GradientBoostingClassifier(n_estimators=100, random_state=42),
    DecisionTreeClassifier(),
    ExtraTreesClassifier(),
    LGBMClassifier(),
    RandomForestClassifier(),
    KNeighborsClassifier(),
    LogisticRegression(max_iter=40000000),
    XGBClassifier()
]
resampled_data = [
    "heart_2020_cleaned_RandomOverSampler(random_state=42, sampling_strategy='minority').csv",
    #"heart_2020_cleaned_RandomUnderSampler(random_state=42, sampling_strategy='majority').csv",
    "heart_2020_cleaned_SMOTE(random_state=42, sampling_strategy='minority').csv",
    "heart_2020_cleaned_ADASYN(random_state=42, sampling_strategy='minority').csv",
    "heart_2020_cleaned_RandomOverSampler(random_state=42, sampling_strategy='minority').csv",
    "heart_2020_cleaned_SMOTEENN(random_state=42, sampling_strategy='minority').csv"
]

def printResult(model, X_train, X_test, y_train, y_test, method, save=False):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"viene utilizzato {method}")
    print(f"accuracy {accuracy_score(y_test, y_pred)}")
    print(f"model: {str(model)}")
    print(f'Accuracy_score: {accuracy_score(y_test, y_pred)}')
    print(f'Precision_score: {precision_score(y_test, y_pred)}')
    print(f'Recall_score: {recall_score(y_test, y_pred)}')
    print(f'F1-score: {f1_score(y_test, y_pred)}')
    print('*'*30, '\n')
    if save:
        index_par_mod = str(model).find("(")
        index_par_method = str(method).find("(")
        end_str = "_" + str(method)[index_par_method:] + ".pkl"
        save_model("./model_resampled/" + str(model)[index_par_mod:end_str], model)

def convert_data(data, categorical_f, target, split=False):
    enc = OrdinalEncoder()
    enc.fit(data[categorical_f])
    data[categorical_f] = enc.transform(data[categorical_f])

    if split:
        y = data[target]
        data.drop(target, axis=1, inplace=True)
        X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.1, random_state=42)
        return X_train, X_test, y_train, y_test
    return data

```

```

target = "HeartDisease"
data = pd.read_csv("./data/heart_2020_cleaned_RandomUnderSampler(random_state=42, sampling_strategy='majority').csv")
y = data[target]
data.drop(target,axis=1,inplace=True)
X_train, X_test, y_train, y_test = train_test_split(data,y,test_size=0.1,random_state=42)

for mod in models:
    printResult(mod,X_train, X_test, y_train, y_test,method="RandomUnderSampler",save=True)
X_test[target] = y_test
df = pd.DataFrame(X_test)
df.to_csv("./data/test_set_"+"heart_2020_cleaned_RandomUnderSampler"+".csv")

# tutti i modelli addestrati con dataset bilanciati con lo stesso algoritmo vengono testati con lo stesso test set
for d in resampled_data:
    data = pd.read_csv("./data/" + d)
    y = data[target]
    data.drop(target,axis=1,inplace=True)
    X_train, X_test, y_train, y_test = train_test_split(data,y,test_size=0.1,random_state=42)

    index_parenthesis_resampled_data = str(d).find("(")
    X_test[target] = y_test
    df = pd.DataFrame(X_test)
    df.to_csv("./data/test_set_" + str(d)[index_parenthesis_resampled_data:] + ".csv")
    X_test.drop(target,axis=1,inplace=True)

for mod in models:
    printResult(mod,X_train, X_test, y_train, y_test,method=d,save=True)

```

I risultati di Precision, Recall e F1 sono riportati di seguito:

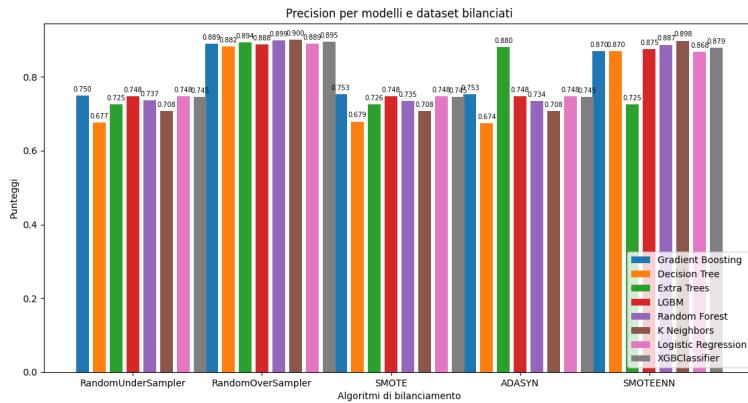


Figure 10: Istogramma precision

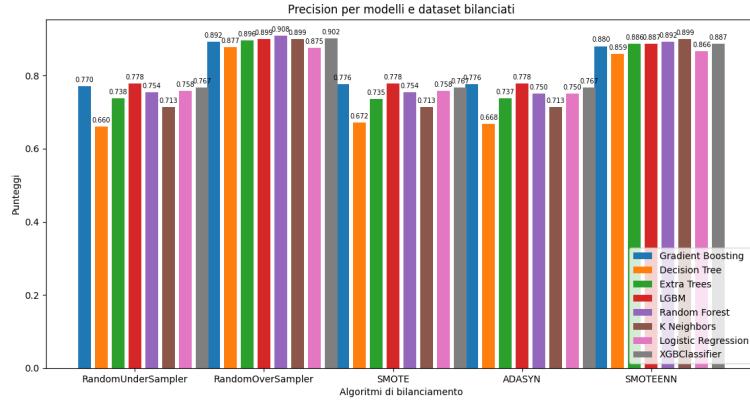


Figure 11: Istogramma F1

I punteggi migliori sono stati raggiunti da modelli che hanno effettuato la fase di training su dataset bilanciati con le classi SMOTEEN e RandomOverSampler, specialmente i modelli allenati con quest ultimo raggiungono percentuali molto simili sia di precision-scoree di recall-score, ne conseguono valori di F1-score molto simili. Il modello che infine abbiamo scelto KNeighborsClassifier addestrato sul dataset bilanciato attraverso la classe RandomOverSampler che ha un'ottima precision-score (90%) e un altrettanto ottima recall-score (89%) questi punteggi permettono una F1-score che si attesta sul 90%

## 5 Conclusion

Per avviare il programma eseguendo lo script main.py.

### 5.1 Interfaccia

Per interfacciarsi con il programma di planning abbiamo creato una semplice interfaccia grafica che permette di creare il profilo del paziente e avere un responso testuale in pochi semplici passi:

### 5.1.1 Creazione scheda paziente

Heart Disease Rehab Planner

X

### Patient Personal Data

Name	<input type="text" value="test"/>
Surname	<input type="text" value="test"/>
Height	<input type="text" value="1.80"/>
Weight	<input type="text" value="100"/>
Ethnicity	<input type="text" value="white"/>
Age	<input type="text" value="60-64"/>
Gender	<input type="text" value="female"/>
PhysicalActivity	<input checked="" type="checkbox"/> Physical Activity

\*Enter height in meters

### 5.1.2 Abitudini paziente

Heart Disease Rehab Planner

## Patient Habits

Are you a smoker?

Do you drink alcohol?

How many hours do you sleep on average?

How many days this month did you have poor physical health?

How many days this month did you have poor mental health?

How do you currently rate your health?

**Next**

### 5.1.3 Patologie paziente

The screenshot shows a window titled "Patient Pathologies" from the "Heart Disease Rehab Planner" application. The window lists several medical conditions with checkboxes. The condition "Stroke" has a checked checkbox. The condition "Diabetic" has a dropdown menu set to "Yes". A "Next" button is visible at the bottom right.

Pathology	Status
Stroke	<input checked="" type="checkbox"/>
Asthma	<input type="checkbox"/>
DiffWalking	<input type="checkbox"/>
KidneyDisease	<input type="checkbox"/>
SkinCancer	<input type="checkbox"/>
Diabetic	Yes

**Next**

#### 5.1.4 Responso

