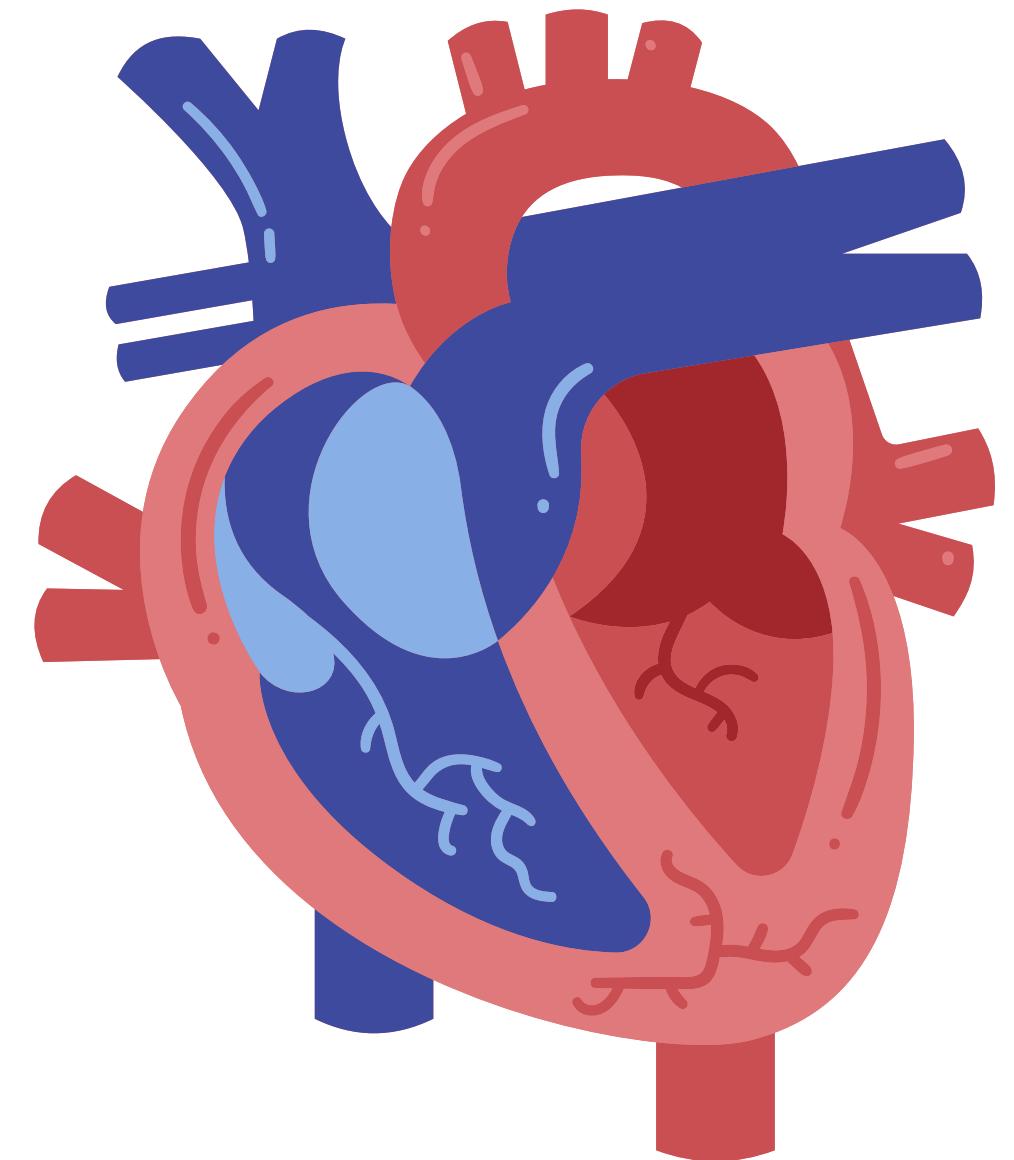


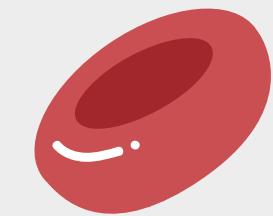
HEART FAILURE CLINICAL RECORDS

Alessia Tranquilli, Marco Quartu,
Mohamed Abouhoula, Virginia Aiello

Obiettivo

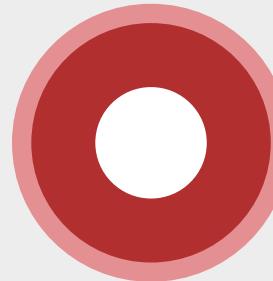
Analizzare i dati clinici dei pazienti con insufficienza cardiaca per identificare pattern nascosti, gruppi omogenei di pazienti e fattori predittivi di mortalità.





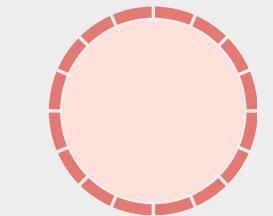
Analisi esplorativa

Comprensione delle distribuzioni cliniche e profilo dei pazienti



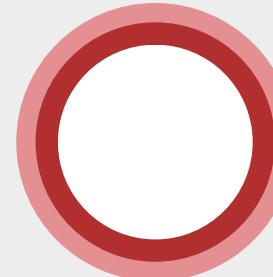
Clustering non supervisionato

Raggruppamento pazienti con caratteristiche simili, creando profili clinici distinti



Classificazione supervisionata

Previsione dell'esito clinico usando modelli di machine learning



Interpretazione clinico-statistica

Analisi dei cluster ricavati e comprensione delle features che influenzano maggiormente la mortalità

Dataset



N° istanze

299 istanze (pazienti)

Feature di tipo:

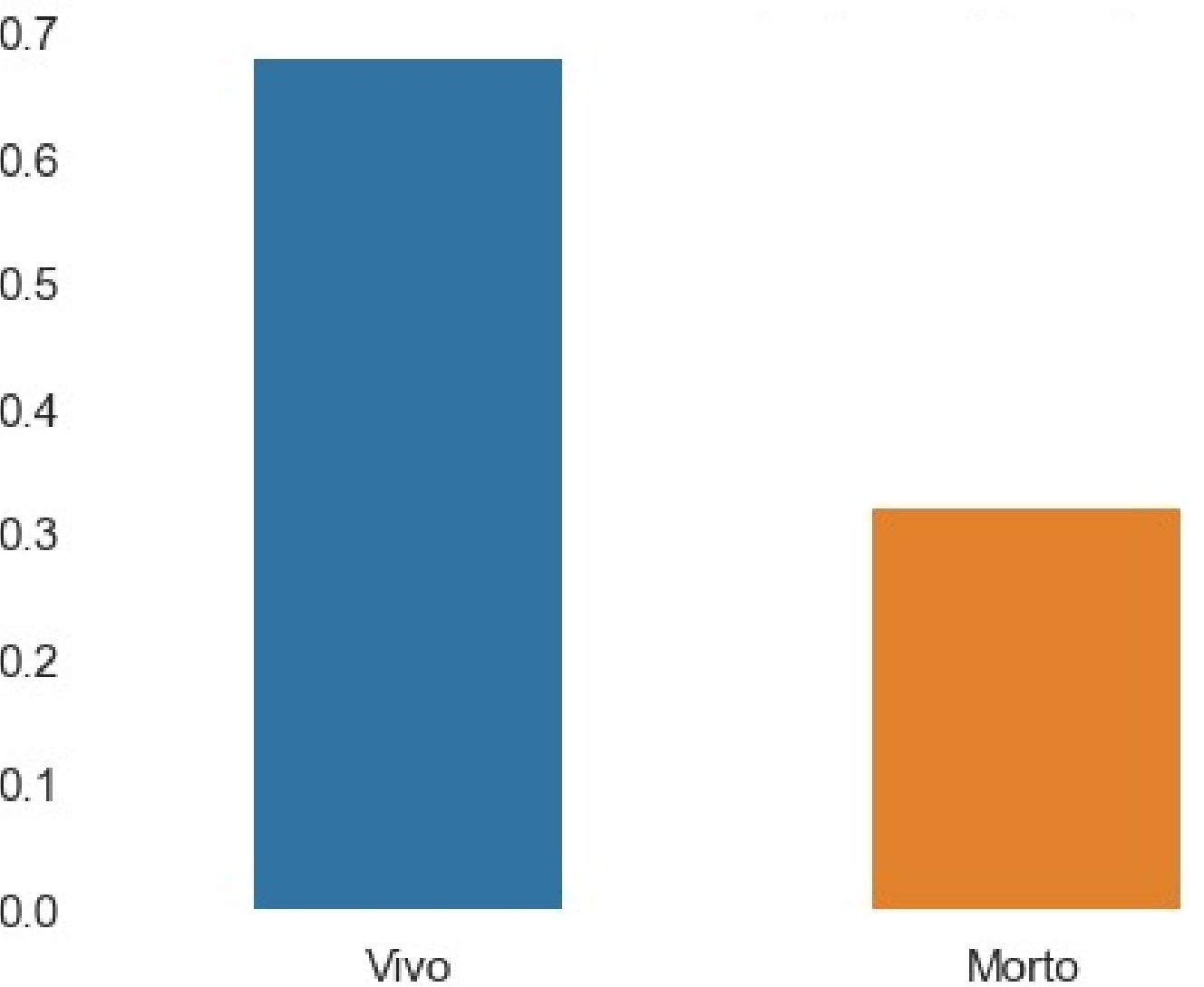
Continuous
Integer
Binary



Features del dataset

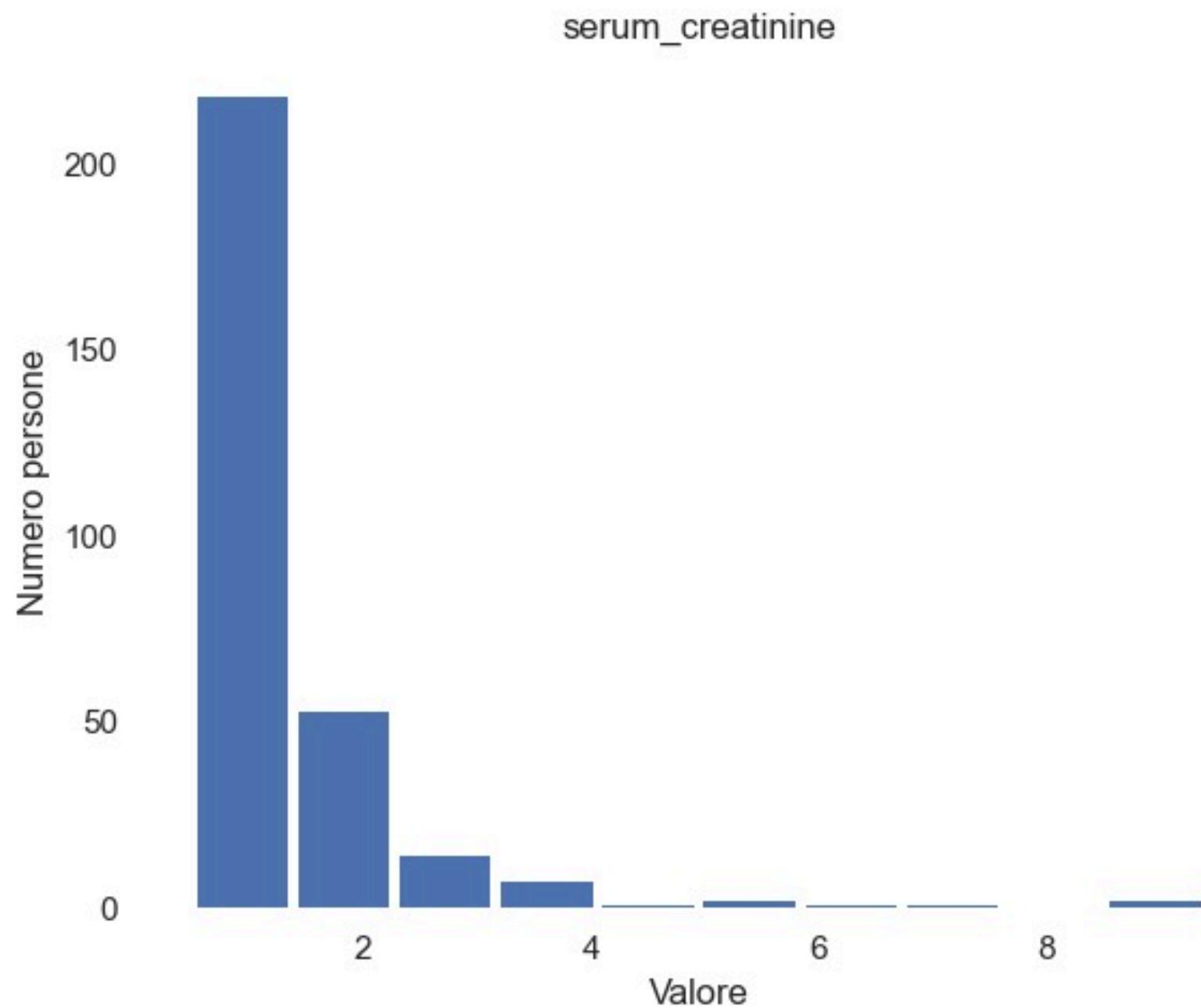
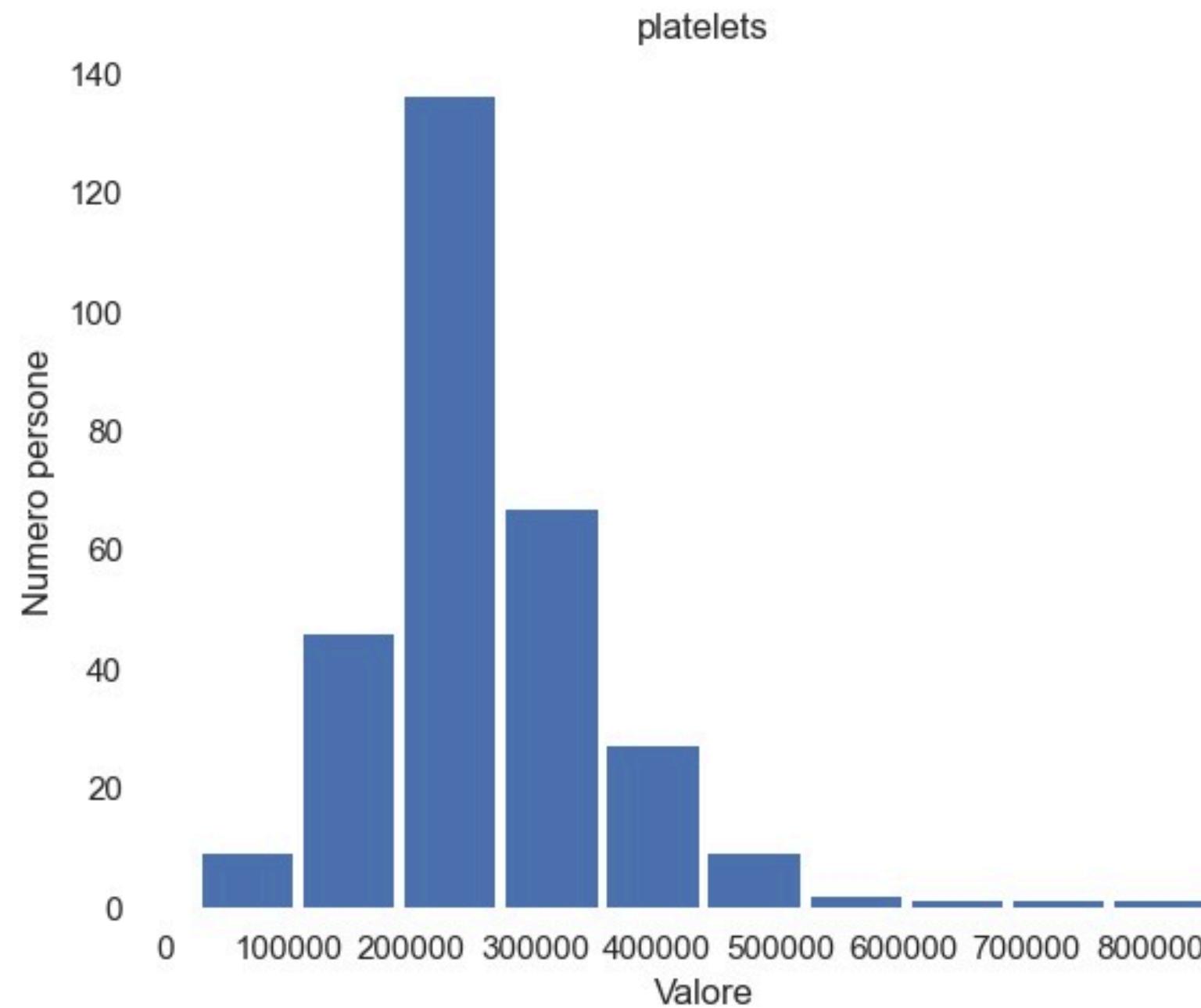
- Age
- Anaemia
- Creatinine_phosphokinase
- Diabetes
- Ejection-fraction
- High_blood_pressure
- Platelets
- Serum_creatinine
- Serum_sodium
- Sex
- Smoking
- Time (follow up)
- Death_event
(Target)

Distribuzione del target



Istogramma

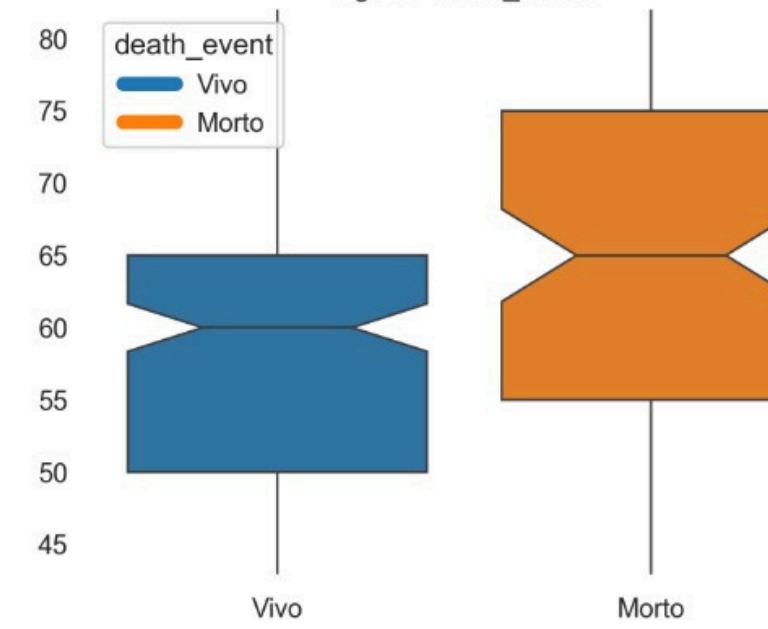
Features continue



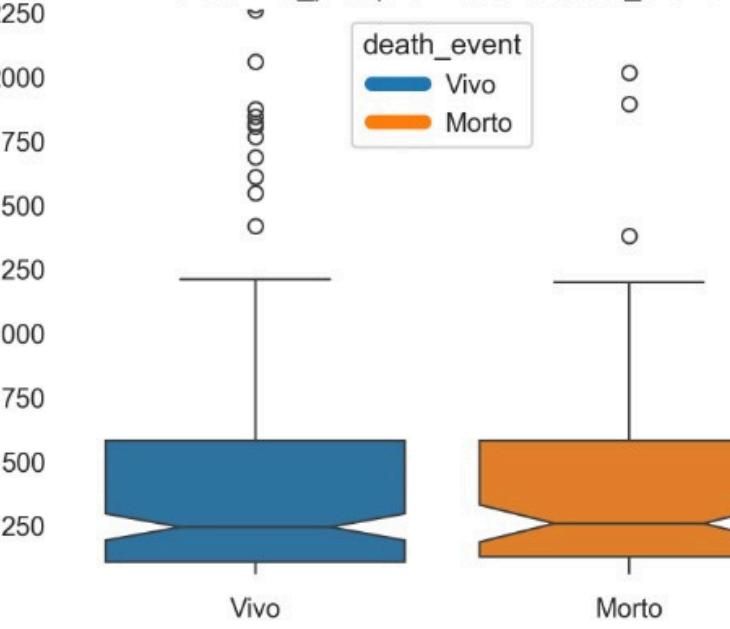
Boxplot

Features interessantes

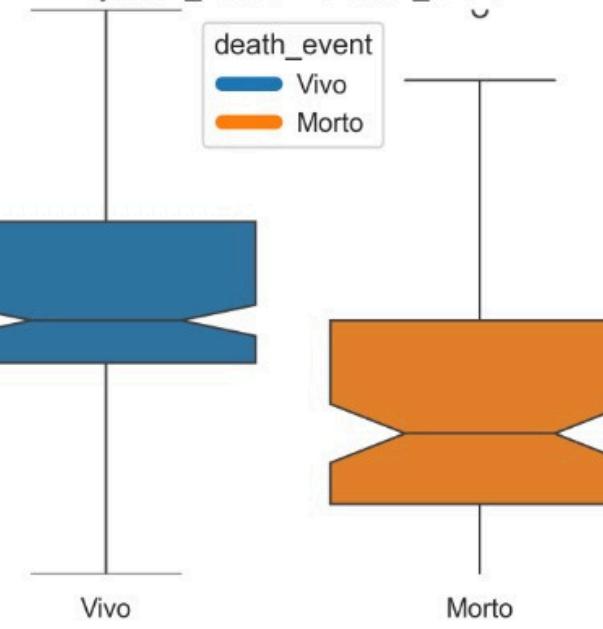
age vs death_event



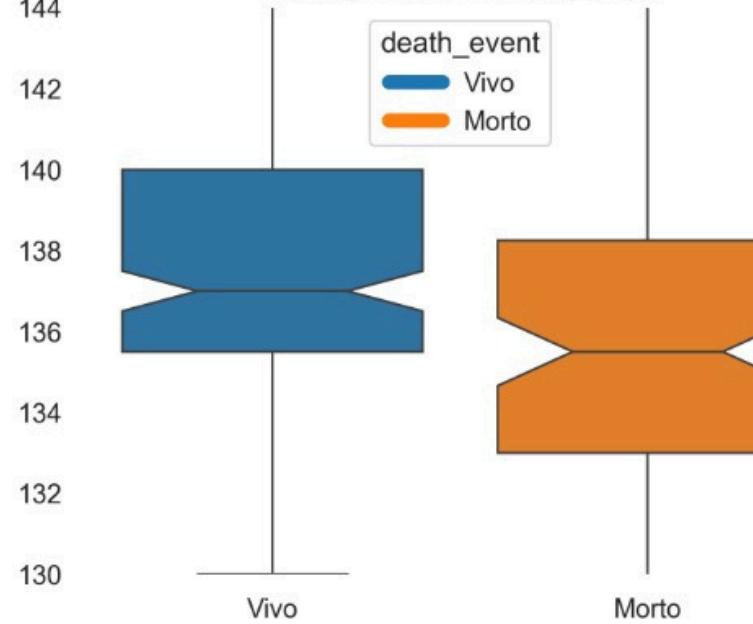
creatinine_phosphokinase vs death_event



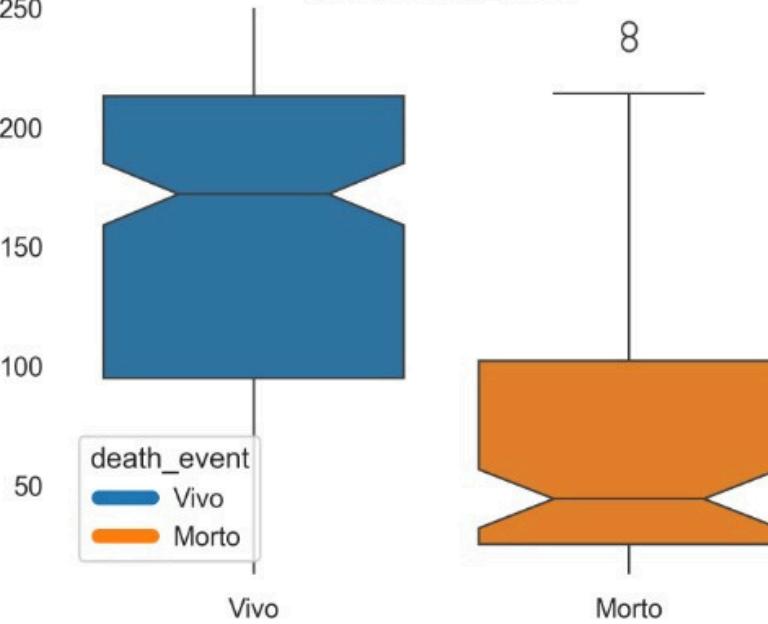
ejection_fraction vs death_event



serum_sodium vs death_event

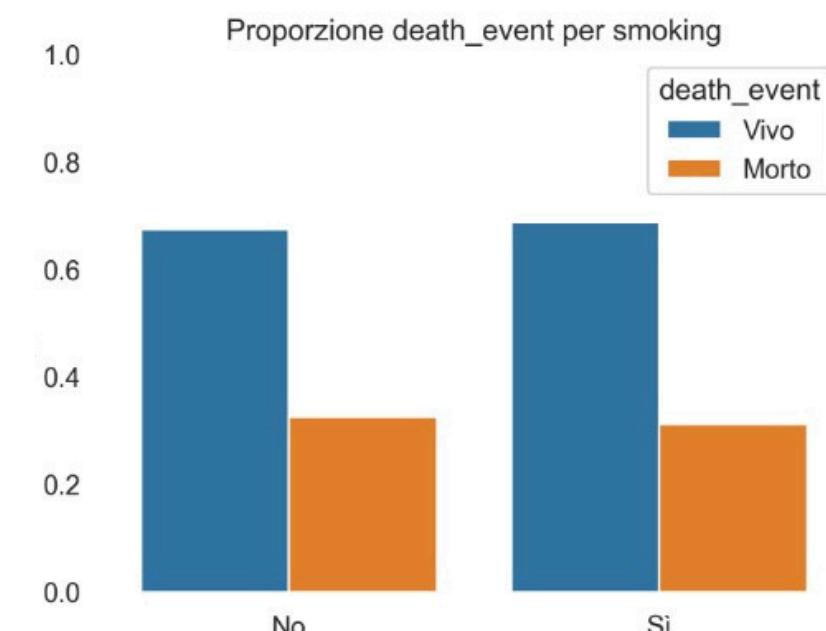
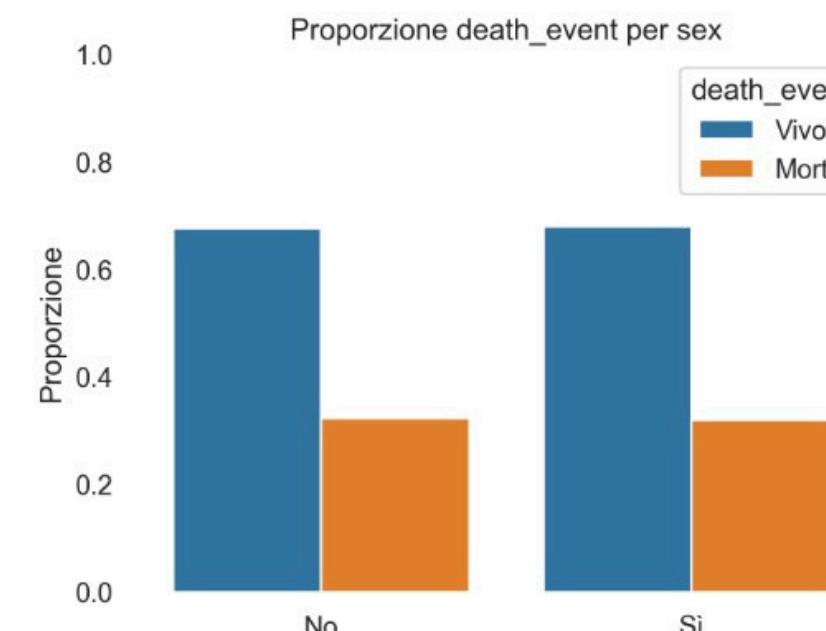
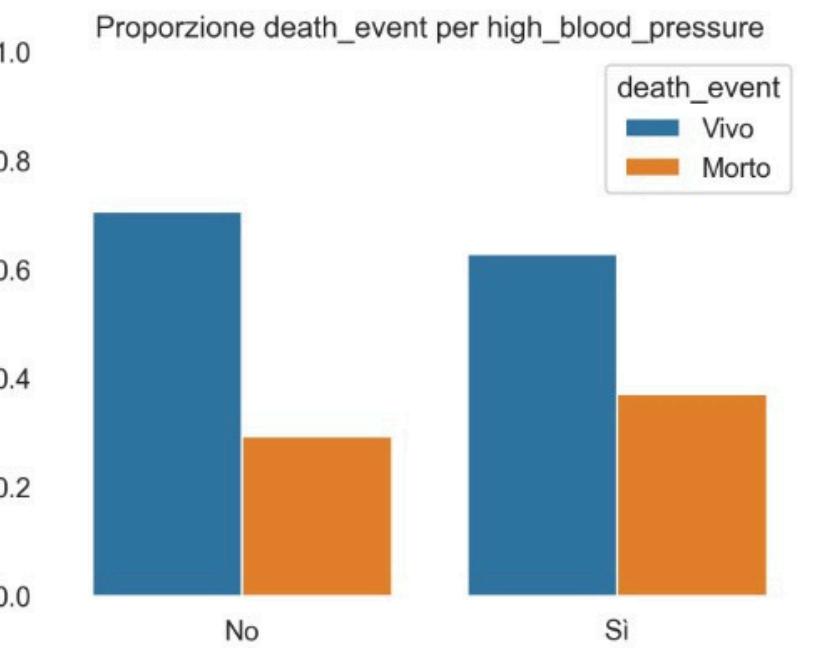
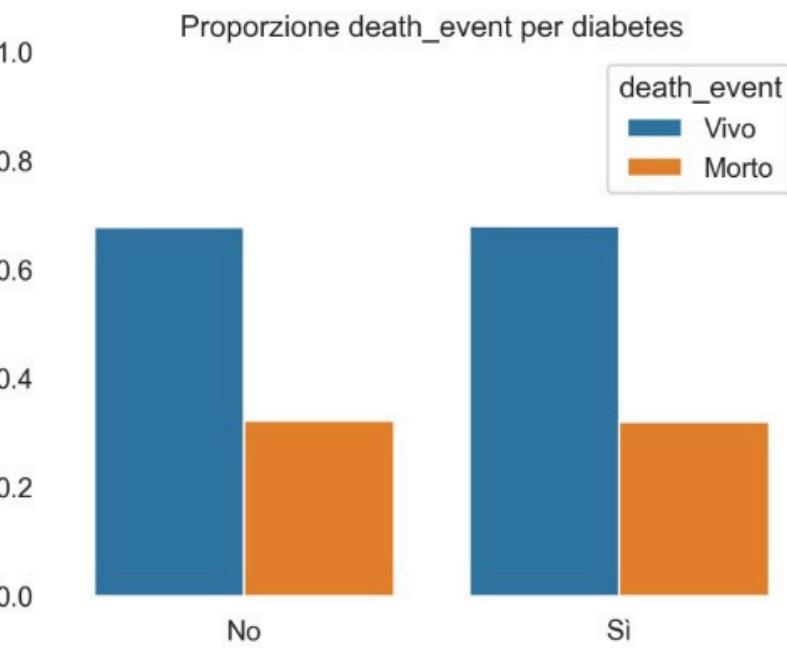
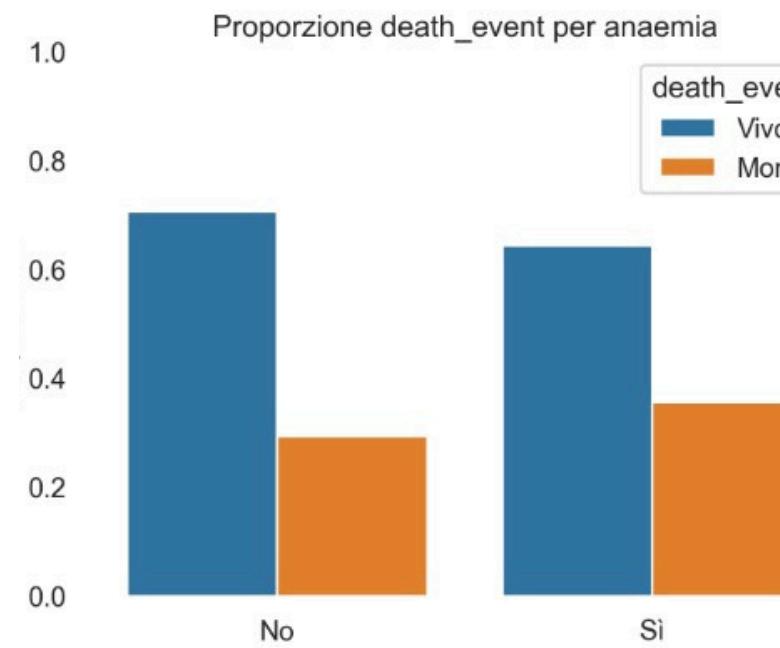


time vs death_event

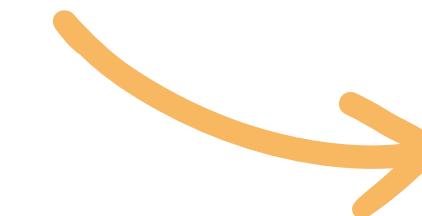
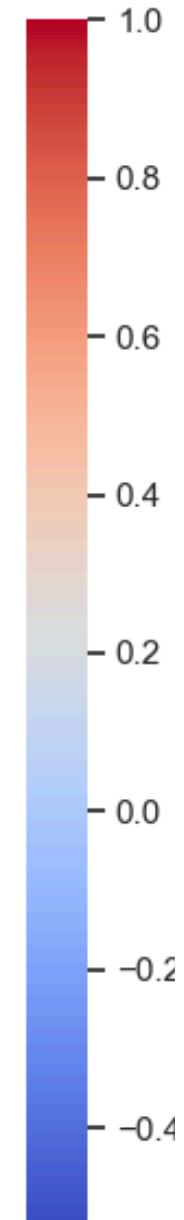
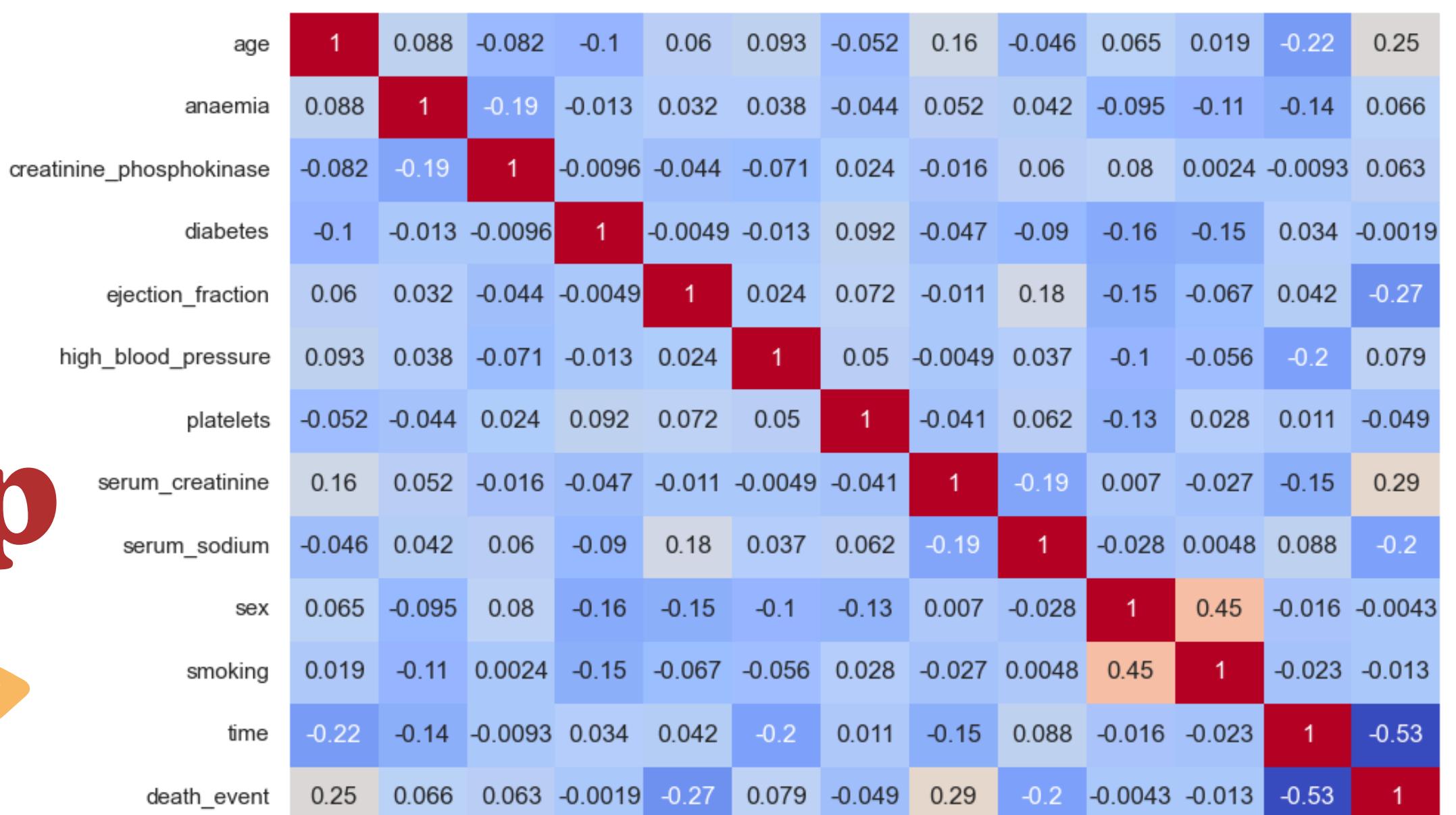


Countplot

Features binarie



Heatmap

Rimozione features e Gower

```
# Separiamo feature e target
x_clean = df_clean.drop("death_event", axis=1)
y_clean = df_clean["death_event"]

# Rimuovi le colonne che non devono essere considerate per il clustering
cols_to_remove = ['sex', 'smoking', 'time']
x_clustering = x_clean.drop(cols_to_remove, axis=1)

# Identifica colonne numeriche e categoriche
num_cols = x_clustering.select_dtypes(include=['int64', 'float64']).columns
cat_cols = x_clustering.select_dtypes(include=['object', 'category']).columns

# Converte solo le numeriche in float
x_clustering[num_cols] = x_clustering[num_cols].astype(float)

# Calcolo matrice distanza Gower
gower_dist = gower.gower_matrix(x_clustering)
```

Gower

Calcola la distanza tra coppie di istanze in un dataset con variabili miste (numeriche, categoriche e ordinarie), normalizzando e combinando i diversi tipi di dati per ottenere un valore compreso tra 0 e 1.

Distanze numeriche

$$d_{numeric}(x_i, x_j) = \frac{|x_i - x_j|}{\text{range}(x)}$$

Distanze binarie

$$d_k(x_{ik}, x_{jk}) = \begin{cases} 0 & \text{se } x_{ik} = x_{jk} \\ 1 & \text{se } x_{ik} \neq x_{jk} \end{cases}$$

Distanza di Gower normalizzata

$$\text{Distance} = \frac{\sum_i d_i}{\text{numero di variabili disponibili}}$$

Agglomerative clustering

Perchè?

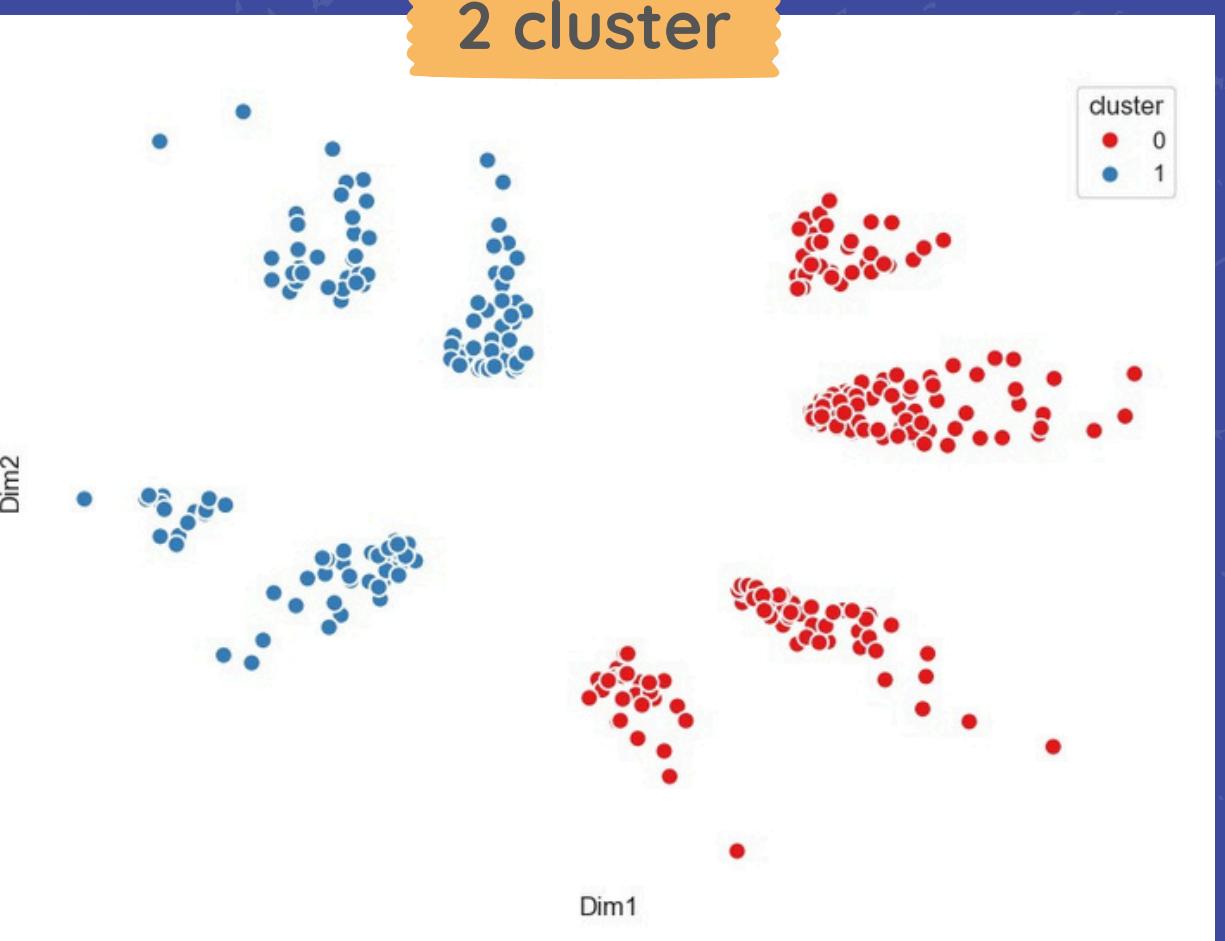
- Funziona bene con dataset piccoli
- Non presuppone che i cluster siano sferici
- Non è sensibile agli outlier
- Non devi scegliere a priori il numero di cluster

Agglomerative Clustering con Gower

- metric = 'precomputed' → matrice delle distanze precalcolata e fornita all'algoritmo
- linkage = 'complete' → distanza tra i cluster è definita dalla massima distanza tra i punti dei due cluster uniti
- Inizialmente ogni punto del dataset è un cluster separato
- Viene utilizzata la distanza di Gower per calcolare la similarità tra i punti
- I due cluster più simili (distanza minore) vengono uniti in un nuovo cluster
- Continua fino a una divisione corretta dei cluster

```
cluster_model_2 = AgglomerativeClustering(  
    n_clusters=2,  
    metric='precomputed',  
    linkage='complete'  
)  
  
clusters_2 = cluster_model_2.fit_predict(gower_dist)  
  
df_clean['cluster_2'] = clusters_2
```

2 cluster



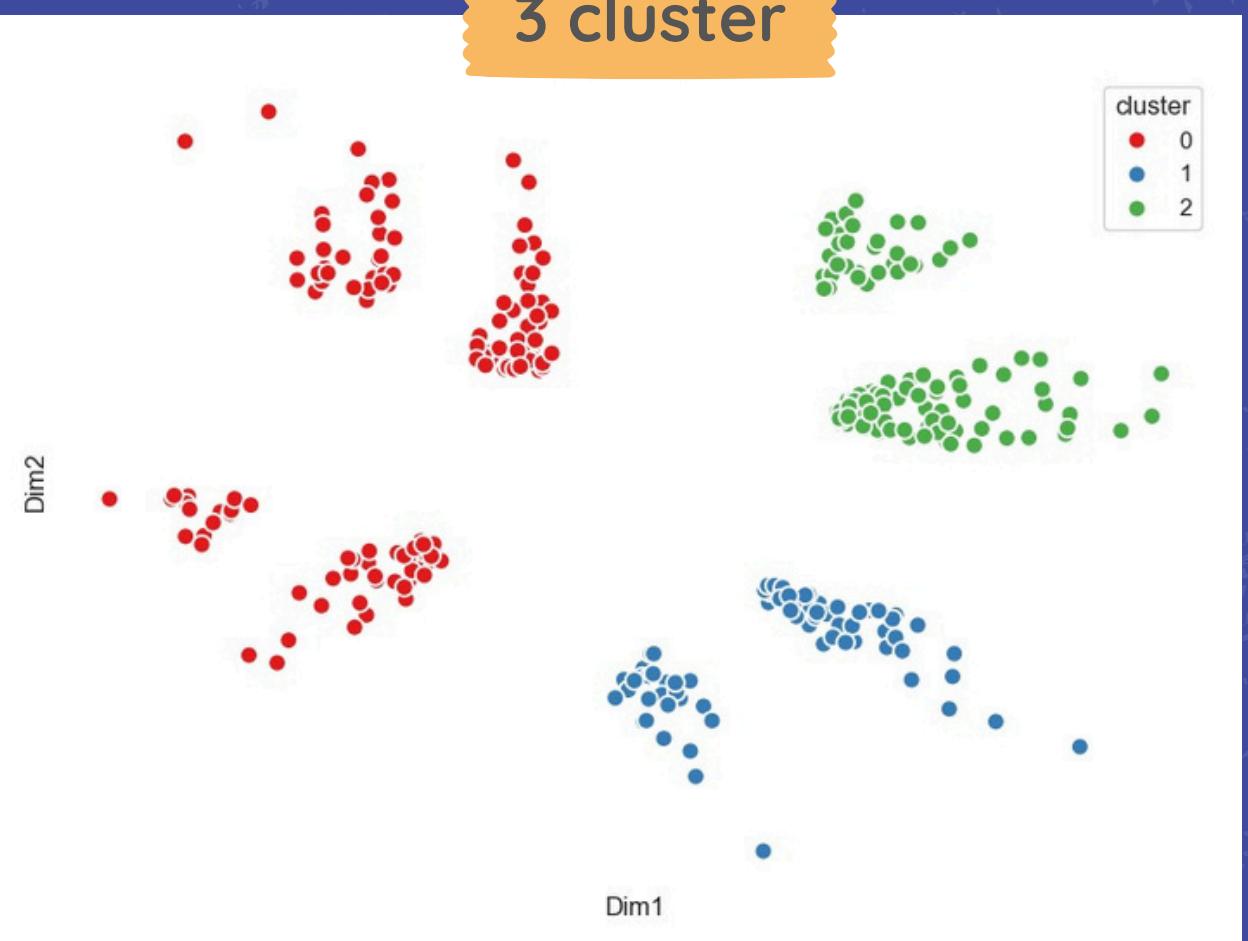
CLUSTER 0

Totale pazienti: 170
Maschi: 117 (68.8%)
Femmine: 53 (31.2%)
Vivi: 120 (70.6%)
Morti: 50 (29.4%)
Diabetici: 72 (42.4%)
Anemici: 0 (0.0%)

CLUSTER 1

Totale pazienti: 129
Maschi: 77 (59.7%)
Femmine: 52 (40.3%)
Vivi: 83 (64.3%)
Morti: 46 (35.7%)
Diabetici: 53 (41.1%)
Anemici: 129 (100.0%)

3 cluster



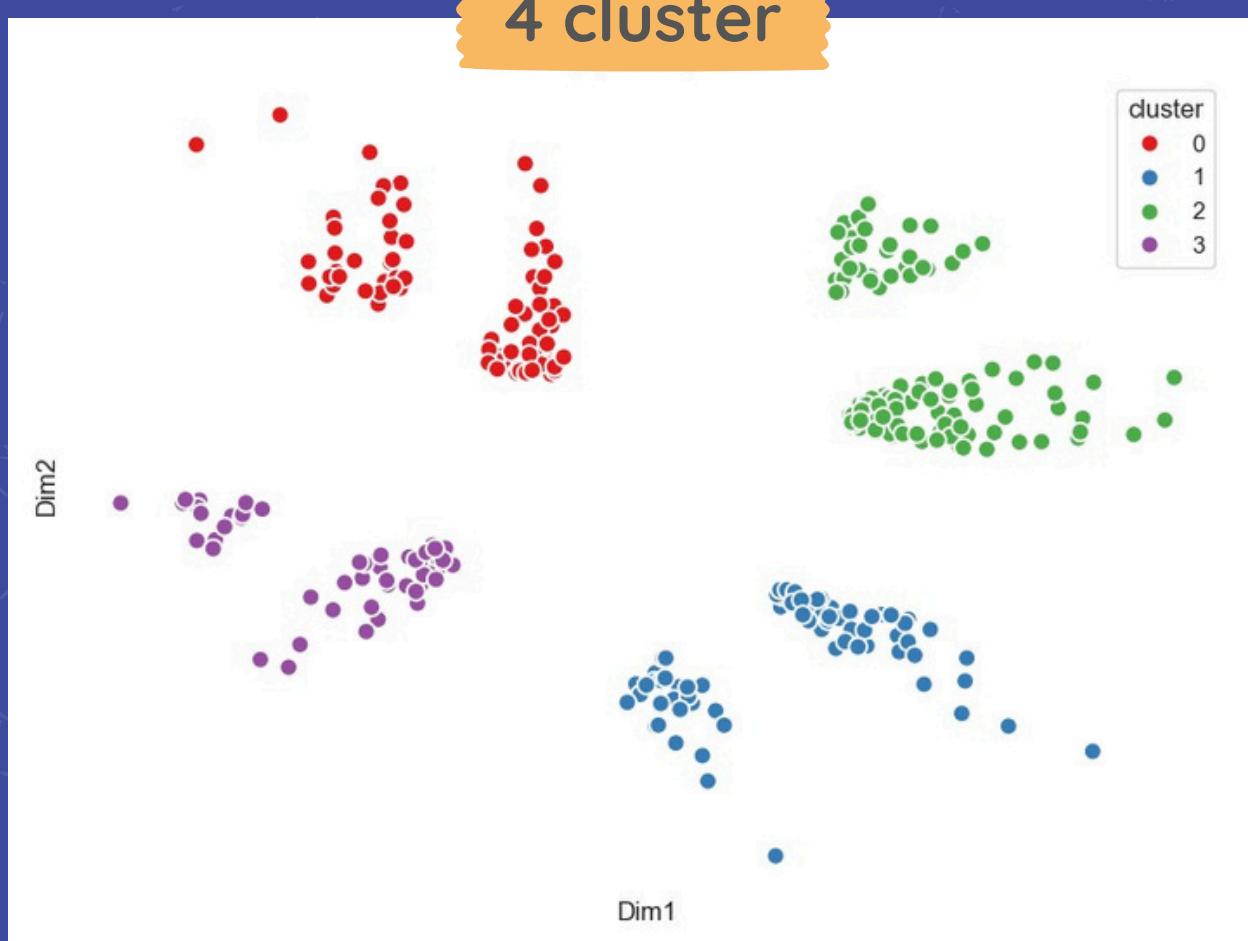
CLUSTER 0

Totale pazienti: 129
Maschi: 77 (59.7%)
Femmine: 52 (40.3%)
Vivi: 83 (64.3%)
Morti: 46 (35.7%)
Diabetici: 53 (41.1%)
Anemici: 129 (100.0%)

CLUSTER 1

Totale pazienti: 72
Maschi: 45 (62.5%)
Femmine: 27 (37.5%)
Vivi: 50 (69.4%)
Morti: 22 (30.6%)
Diabetici: 72 (100.0%)
Anemici: 0 (0.0%)

4 cluster



CLUSTER 0

Totale pazienti: 76
Maschi: 52 (68.4%)
Femmine: 24 (31.6%)
Vivi: 48 (63.2%)
Morti: 28 (36.8%)
Diabetici: 0 (0.0%)
Anemici: 76 (100.0%)

CLUSTER 1

Totale pazienti: 72
Maschi: 45 (62.5%)
Femmine: 27 (37.5%)
Vivi: 50 (69.4%)
Morti: 22 (30.6%)
Diabetici: 72 (100.0%)
Anemici: 0 (0.0%)

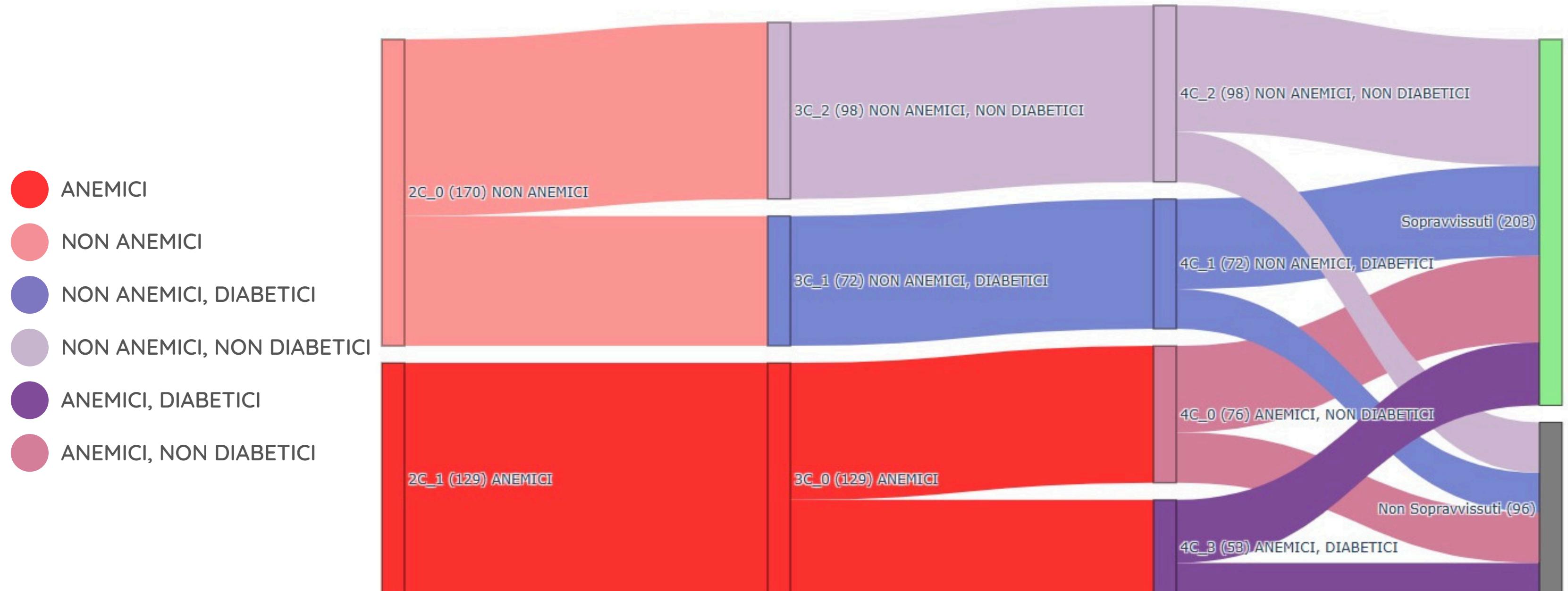
CLUSTER 2

Totale pazienti: 98
Maschi: 72 (73.5%)
Femmine: 26 (26.5%)
Vivi: 70 (71.4%)
Morti: 28 (28.6%)
Diabetici: 0 (0.0%)
Anemici: 0 (0.0%)

CLUSTER 3

Totale pazienti: 53
Maschi: 25 (47.2%)
Femmine: 28 (52.8%)
Vivi: 35 (66.0%)
Morti: 18 (34.0%)
Diabetici: 53 (100.0%)
Anemici: 53 (100.0%)

Sankey plot diagram

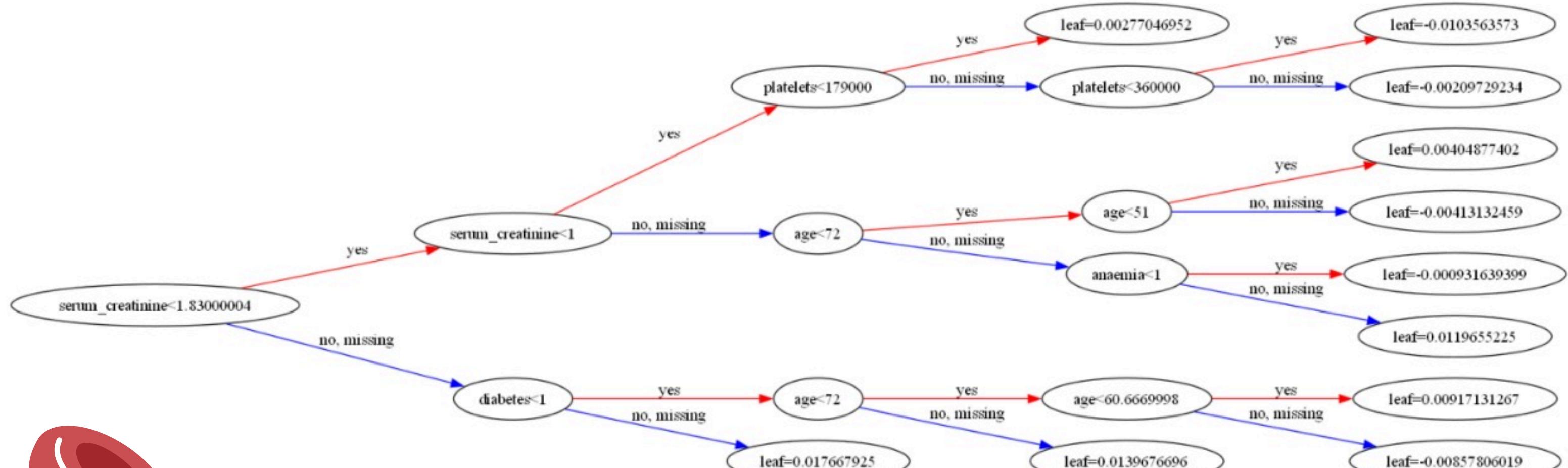


XGBoost funzionamento

- Basato sull'idea di boosting → migliora passo dopo passo
- Lavora sui residui (errori) → Se il modello (albero) sbaglia una previsione, XGBoost “segna” quel punto come importante
- Regolarizzazione integrata per evitare overfitting → limita la profondità degli alberi e penalizza quelli troppo complessi

Albero XGBoost

N° 32, primo ad avere anemia e diabete



XGBoost



- Separa i dati
- Fa train/test split
- Individua parametri ottimali con RandomizedSearchCV
- Prende il miglior modello
- Utilizza ROC-AUC
- Trova una soglia di decisione “conservatrice”, riducendo il numero di falsi negativi

```
# Feature e target
X = df.drop(columns=['death_event', 'time', 'sex', 'smoking'])
y = df['death_event']

# Train/test split 75% - 25%
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=32, stratify=y
)

# RandomizedSearchCV per parametri XGBoost
param_grid = {
    'n_estimators': [200, 300, 400],
    'max_depth': [3, 4, 5, 6],
    'learning_rate': [0.01, 0.03, 0.05, 0.1],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0]
}

xgb = XGBClassifier(eval_metric='logloss', use_label_encoder=False)
search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid,
    n_iter=20,
    scoring='roc_auc',
    cv=3,
    verbose=1,
    random_state=32,
    n_jobs=-1
)

search.fit(X_train, y_train)
best_model = search.best_estimator_
```

XGBoost

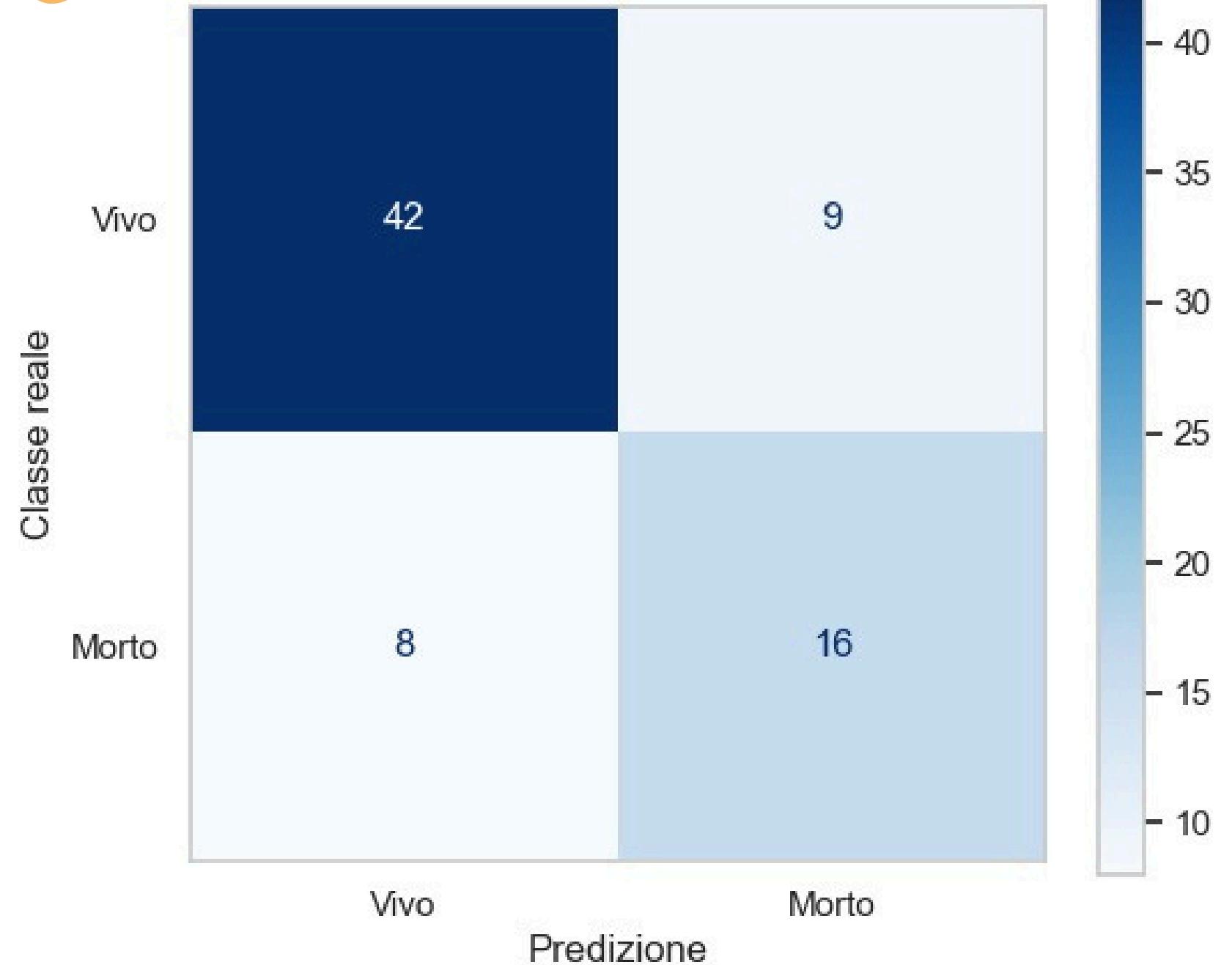
accuracy con il nostro dataset (299 istanze):

- training → 224 istanze
- test → 75 istanze → 58 corretti e 17 sbagliati

accuracy = 0.773



Matrice di Confusione – XGBoost



Probabilità di morte con un nuovo paziente

```
nuovo_paziente = pd.DataFrame({  
    'age': [75],  
    'creatinine_phosphokinase': [600],  
    'anaemia': [0],  
    'diabetes': [0],  
    'ejection_fraction': [35],  
    'high_blood_pressure': [1],  
    'platelets': [250000],  
    'serum_creatinine': [2.0],  
    'serum_sodium': [135]  
})
```

Probabilità di morte: 0.51

```
prob_morte = model.predict_proba(nuovo_paziente)[:, 1][0]  
predizione_binaria = model.predict(nuovo_paziente)[0]  
  
print(f"Probabilità di morte: {prob_morte:.2f}")  
print(f"Predizione (0= sopravvive, 1= muore): {predizione_binaria}")
```

'anaemia': [1],
'diabetes': [0],

Probabilità di morte: 0.60

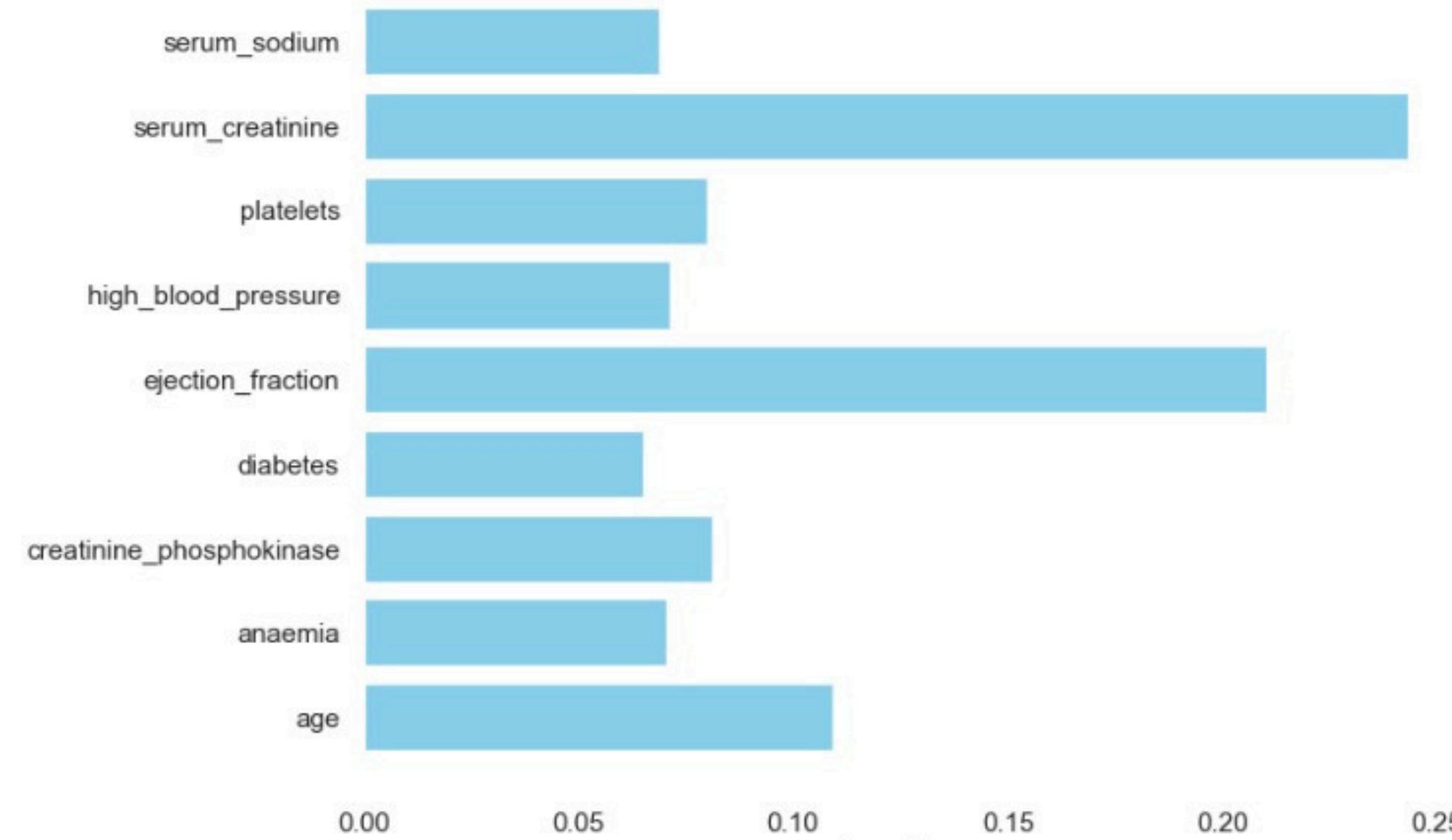
'anaemia': [0],
'diabetes': [1],

Probabilità di morte: 0.64

'anaemia': [1],
'diabetes': [1],

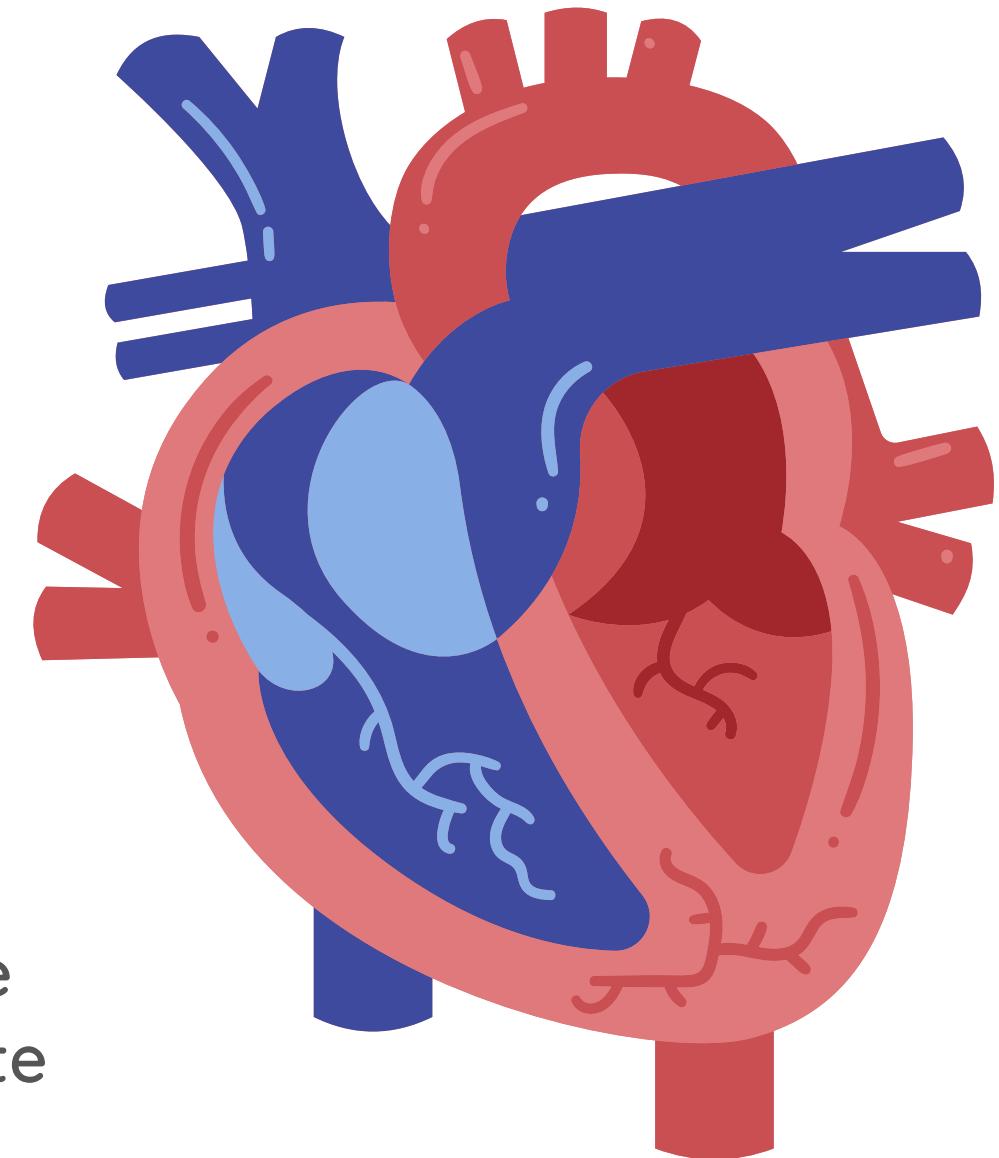
Probabilità di morte: 0.67

Feature Importance



Conclusioni

- Clustering: identifica gruppi differenziando per anemia e diabete, utili per descrivere profili clinici ma non significativamente predittivi della mortalità.
- Classificazione (XGBoost): evidenzia EF e serum_creatinine come principali fattori associati al rischio di death event.
- Sintesi: La segmentazione identifica i contesti clinici, mentre la classificazione individua i fattori che influiscono realmente sul rischio di mortalità.



Bibliografia

- Heart Failure Clinical Records Dataset, UCI Machine Learning Repository (2020), DOI: 10.24432/C5Z89R
- Chicco, D. & Jurman, G. (2020). Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone. *BMC Medical Informatics and Decision Making*, 20, 16, DOI: 10.1186/s12911-020-1023-5.
- Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 27(4), 857–871. DOI:10.2307/2528823
- Day, W. H. E. & Edelsbrunner, H. (1984). Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1, 7–24. DOI: 10.1007/BF01890115
- Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD*. DOI: 10.1145/2939672.2939785



GRAZIE PER L'ATTENZIONE

Alessia Tranquilli, Marco Quartu,
Mohamed Abouhoula, Virginia Aiello