주제 : 데이터 분석으로 심부전증을 예방할 수 있을까?

데이터 소개

```
- Heart Failure Prediction 데이터셋을 사용합니다.

- csv 파일을 사용합니다.
heart_failure_clinical_records_dataset.csv

- 각 파일의 컬럼은 아래와 같습니다.
age: 환자의 나이
anaemia: 환자의 빈혈증 여부 (0: 정상, 1: 빈혈)
creatinine_phosphokinase: 크레아틴키나제 검사 결과
diabetes: 당뇨병 여부 (0: 정상, 1: 당뇨)
ejection_fraction: 박출계수 (%)
high_blood_pressure: 고혈압 여부 (0: 정상, 1: 고혈압)
platelets: 혈소판 수 (kiloplatelets/mL)
serum_creatinine: 혈중 크레아틴 레벨 (mg/dL)
serum_sodium: 혈중 나트륨 레벨 (mEq/L)
sex: 성별 (0: 여성, 1: 남성)
smoking: 흡연 여부 (0: 비흡연, 1: 흡연)
time: 관찰 기간 (일)
DEATH_EVENT: 사망 여부 (0: 생존, 1: 사망)
```

- 데이터 출처: https://www.kaggle.com/andrewmvd/heart-failure-clinical-data

최종 목표

```
- 의료 데이터와 그 분석에 대한 이해
- 데이터 시각화를 통한 인사이트 습득 방법의 이해
- Scikit-learn 기반의 모델 학습 방법 습득
- Classification 모델의 학습과 평가 방법 이해
```

Step 0. 의료 데이터셋에 대하여

의료 데이터의 수집

의료 데이터 분석의 현재

Accuracy, Precision, 그리고 Recall

Step 1. 데이터셋 준비하기

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

```
  import os
```

1_1. 데이터 다운로드 및 압축 해제하기

```
# Linux 명령어로 Kaggle API를 이용하여 데이터셋 다운로드하기 (!kaggle ~)
# Linux 명령어로 압축 해제하기
#!kaggle datasets download -d andrewmvd/heart-failure-clinical-data
#!unzip '*.zip'

#!ls
```

1_2. Pandas 라이브러리로 csv파일 읽어들이기

```
# pd.read_csv()로 csv파일 읽어들이기
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
```

```
  df
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine
0	75.0	0	582	0	20	1	265000.00	1.9
1	55.0	0	7861	0	38	0	263358.03	1.1
2	65.0	0	146	0	20	0	162000.00	1.3
3	50.0	1	111	0	20	0	210000.00	1.9
4	65.0	1	160	1	20	0	327000.00	2.7
...	...	...	...	...	...	...	...	...
294	62.0	0	61	1	38	1	155000.00	1.1

Step 2. EDA 및 데이터 기초 통계 분석

2_1. 데이터프레임의 각 컬럼 분석하기

```
# DataFrame에서 제공하는 메소드를 이용하여 컬럼 분석하기 (head(), info(), describe())
df.head(5)
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	se
0	75.0	0	582	0	20	1	265000.00	1.9	
1	55.0	0	7861	0	38	0	263358.03	1.1	
2	65.0	0	146	0	20	0	162000.00	1.3	
3	50.0	1	111	0	20	0	210000.00	1.9	
4	65.0	1	160	1	20	0	327000.00	2.7	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
```

```
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

df.describe()

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	seru
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	

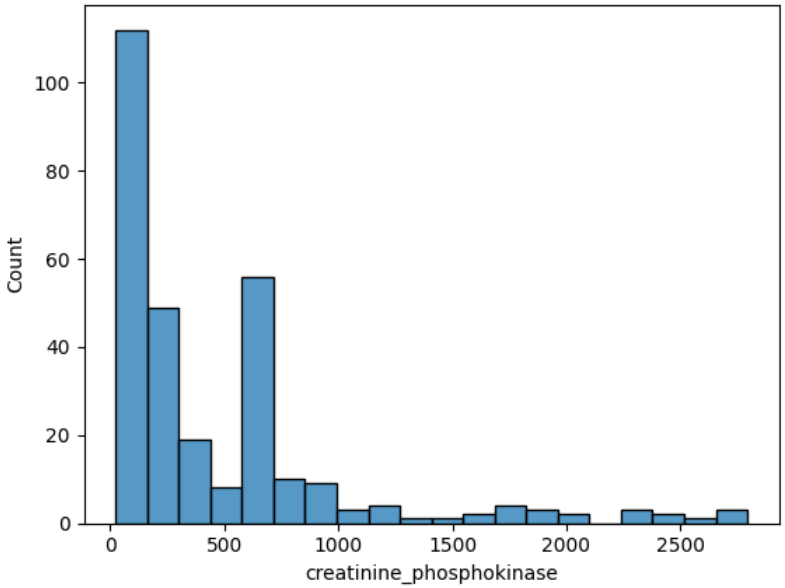
2_2. 수치형 데이터의 히스토그램 그리기

```
# seaborn의 histplot, jointplot, pairplot을 이용해 히스토그램 그리기
sns.histplot(x='age', data=df, hue='DEATH_EVENT', kde=True)
```

<Axes: xlabel='age', ylabel='Count'>

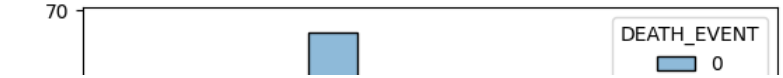

```
sns.histplot(data=df.loc[df['creatinine_phosphokinase'] < 3000, 'creatinine_phosphokinase'])
```

<Axes: xlabel='creatinine_phosphokinase', ylabel='Count'>



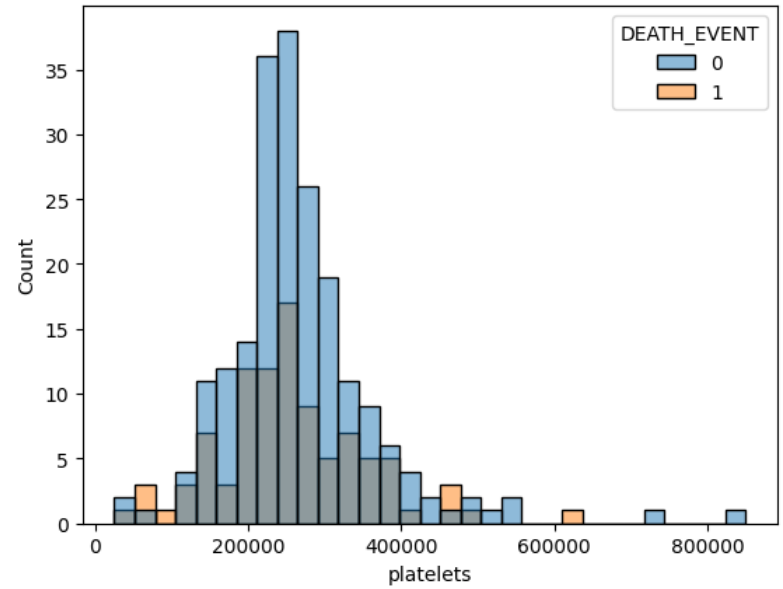

```
sns.histplot(x='ejection_fraction', data=df, bins=13, hue='DEATH_EVENT', kde=True)
```

```
<Axes: xlabel='ejection_fraction', ylabel='Count'>
```




```
sns.histplot(x='platelets', data=df, hue='DEATH_EVENT')
```

```
<Axes: xlabel='platelets', ylabel='Count'>
```

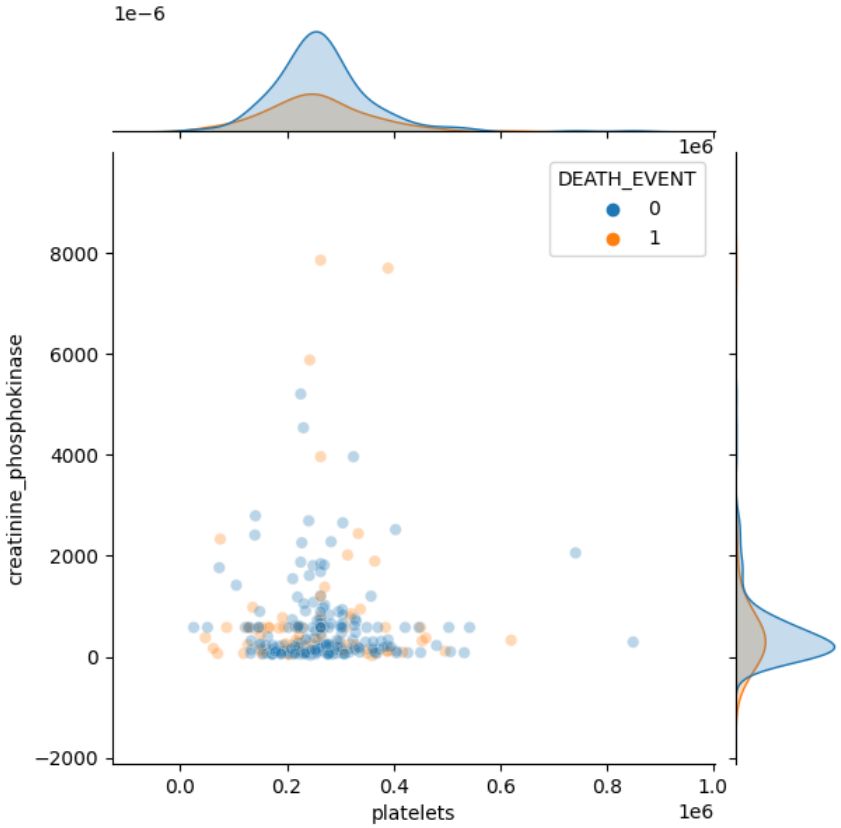



```
sns.histplot(x='time', data=df, hue='DEATH_EVENT', kde=True)
```

```
<Axes: xlabel='time', ylabel='Count'>
```

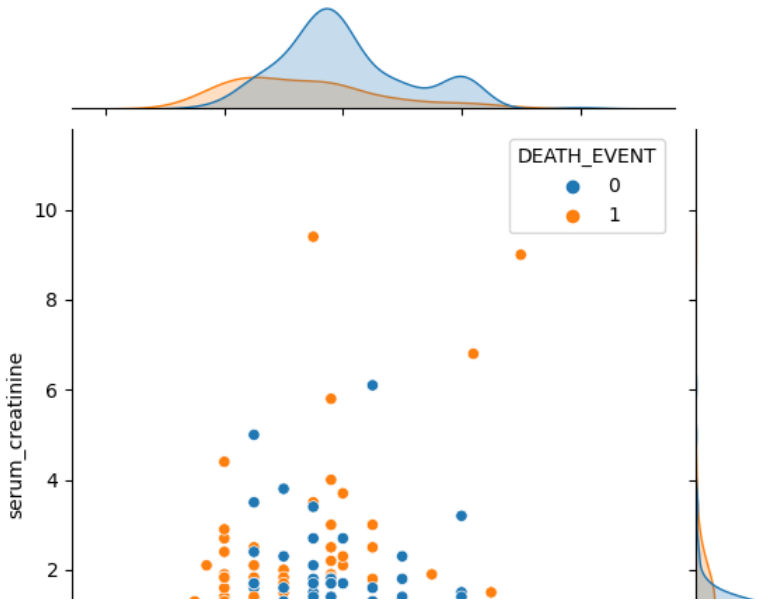

```
sns.jointplot(x='platelets', y='creatinine_phosphokinase', hue='DEATH_EVENT', data=df, alpha=0.3)
```

```
<seaborn.axisgrid.JointGrid at 0x78fab344c760>
```




```
sns.jointplot(x='ejection_fraction', y='serum_creatinine', data=df, hue='DEATH_EVENT')
```

```
<seaborn.axisgrid.JointGrid at 0x78fab32e9a80>
```



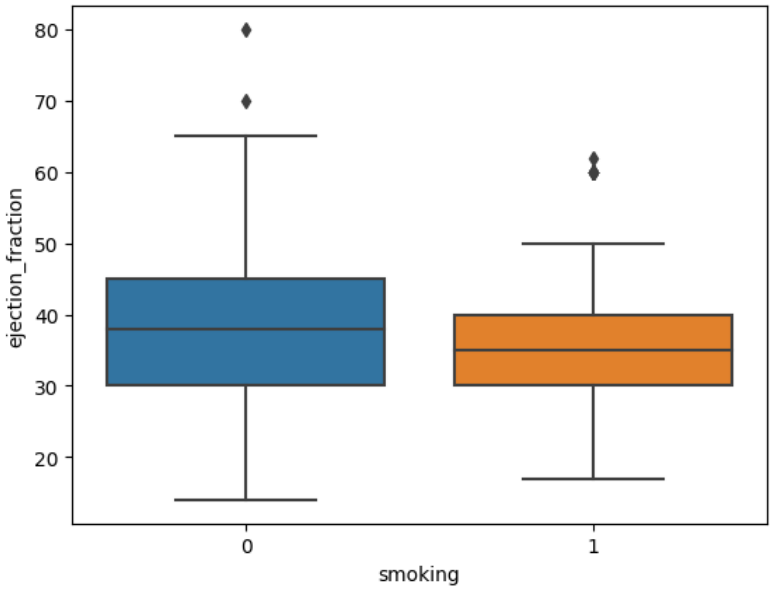
2_3. Boxplot 계열을 이용하여 범주별 통계 확인하기

```
# seaborn의 Boxplot 계열(boxplot(), violinplot(), swarmplot())을 사용
# Hint) hue 키워드를 사용하여 범주 세분화 가능
sns.boxplot(x='DEATH_EVENT', y='ejection_fraction', data=df)
```

```
<Axes: xlabel='DEATH_EVENT', ylabel='ejection_fraction'>
```


```
sns.boxplot(x='smoking', y='ejection_fraction', data=df)
```

```
<Axes: xlabel='smoking', ylabel='ejection_fraction'>
```

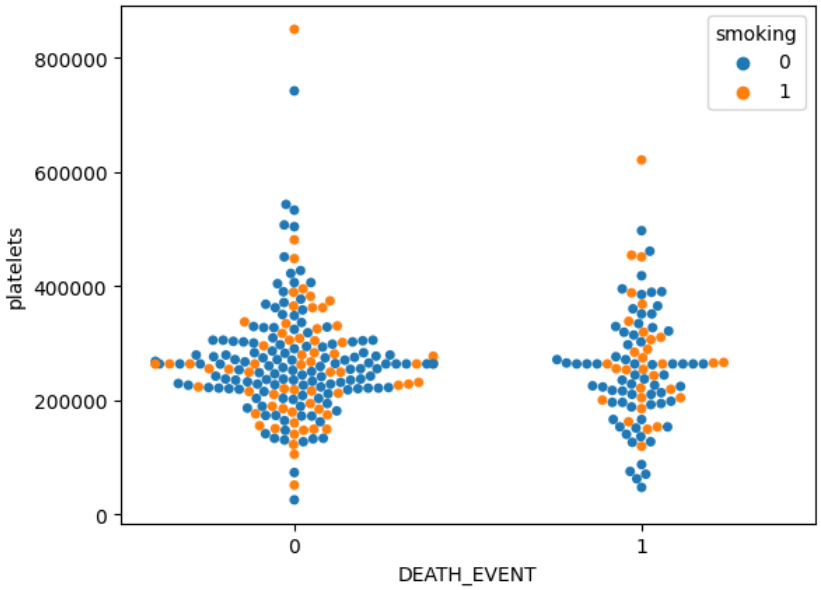



```
sns.violinplot(x='DEATH_EVENT', y='ejection_fraction', data=df)
```

```
<Axes: xlabel='DEATH_EVENT', ylabel='ejection_fraction'>
```


```
sns.swarmplot(x='DEATH_EVENT', y='platelets', hue='smoking', data=df)
```

```
<Axes: xlabel='DEATH_EVENT', ylabel='platelets'>
```



Step 3. 모델 학습을 위한 데이터 전처리

3_1. StandardScaler를 이용하여 데이터 전처리하기

```
from sklearn.preprocessing import StandardScaler
```

```
df.columns
```

```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

```
# 수치형 입력 데이터, 범주형 입력 데이터, 출력 데이터로 구분하기
X_num = df[['age', 'creatinine_phosphokinase','ejection_fraction', 'platelets','serum_creatinine', 'serum_sodium']]
X_cat = df[['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking']]
y = df['DEATH_EVENT']

# 수치형 입력 데이터를 전처리하고 입력 데이터 통합하기
scaler = StandardScaler()
scaler.fit(X_num)
X_scaled = scaler.transform(X_num)
X_scaled = pd.DataFrame(data=X_scaled, index=X_num.index, columns=X_num.columns)
X = pd.concat([X_scaled, X_cat], axis=1)

X.head()
```

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium	anaemia	diabetes	h
0	1.192945	0.000166	-1.530560	1.681648e-02	0.490057	-1.504036	0	0	
1	-0.491279	7.514640	-0.007077	7.535660e-09	-0.284552	-0.141976	0	0	
2	0.350833	-0.449939	-1.530560	-1.038073e+00	-0.090900	-1.731046	0	0	
3	-0.912335	-0.486071	-1.530560	-5.464741e-01	0.490057	0.085034	1	0	
4	0.350833	-0.435486	-1.530560	6.517986e-01	1.264666	-4.682176	1	1	

3_2. 학습데이터와 테스트데이터 분리하기

```
from sklearn.model_selection import train_test_split

# train_test_split() 함수로 학습 데이터와 테스트 데이터 분리하기
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Step 4. Classification 모델 학습하기

4_1. Logistic Regression 모델 생성/학습하기

```
from sklearn.linear_model import LogisticRegression
```

```
# LogisticRegression 모델 생성/학습
model_lr = LogisticRegression(max_iter=1000)
model_lr.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

4_2. 모델 학습 결과 평가하기

```
from sklearn.metrics import classification_report

# Predict를 수행하고 classification_report() 결과 출력하기
pred = model_lr.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.92      0.84        64
           1       0.64      0.35      0.45        26

    accuracy                           0.76        90
   macro avg       0.71      0.63      0.65        90
weighted avg       0.74      0.76      0.73        90
```

4_3. XGBoost 모델 생성/학습하기

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.3)
```

```
import xgboost

from xgboost import XGBClassifier

# XGBClassifier 모델 생성/학습
model_xgb = XGBClassifier()
model_xgb.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
```

4_4 모델 학습 결과 평가하기

```
# Predict를 수행하고 classification_report()결과 출력하기
pred = model_xgb.predict(X_test)
print(classification_report(y_test, pred))
```

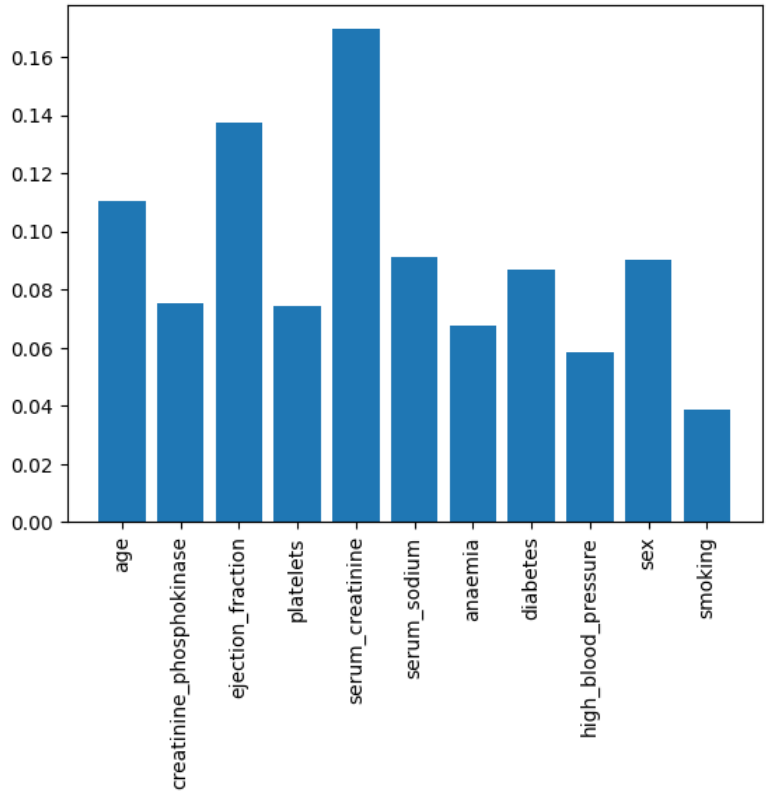
```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        64
           1       0.59      0.50      0.54        26

    accuracy                           0.76        90
   macro avg       0.70      0.68      0.69        90
weighted avg       0.75      0.76      0.75        90
```

4_5 특징의 중요도 확인하기

```
# XGBClassifier 모델의 feature_importances_를 이용하여 중요도 plot
plt.bar(X.columns, model_xgb.feature_importances_)
plt.xticks(rotation=90)
plt.show()
```


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
age
creatinine_phosphokinase
ejection_fraction
platelets
serum_creatinine
serum_sodium
anaemia
diabetes
high_blood_pressure
sex
smoking