Data Science part

1.1 load the data into Pandas dataframe. Extract two dataframes with the above 4 features: df 0 for surviving patients (DEATH EVENT = 0) and df 1 for deceased patients (DEATH EVENT = 1)
Answer:

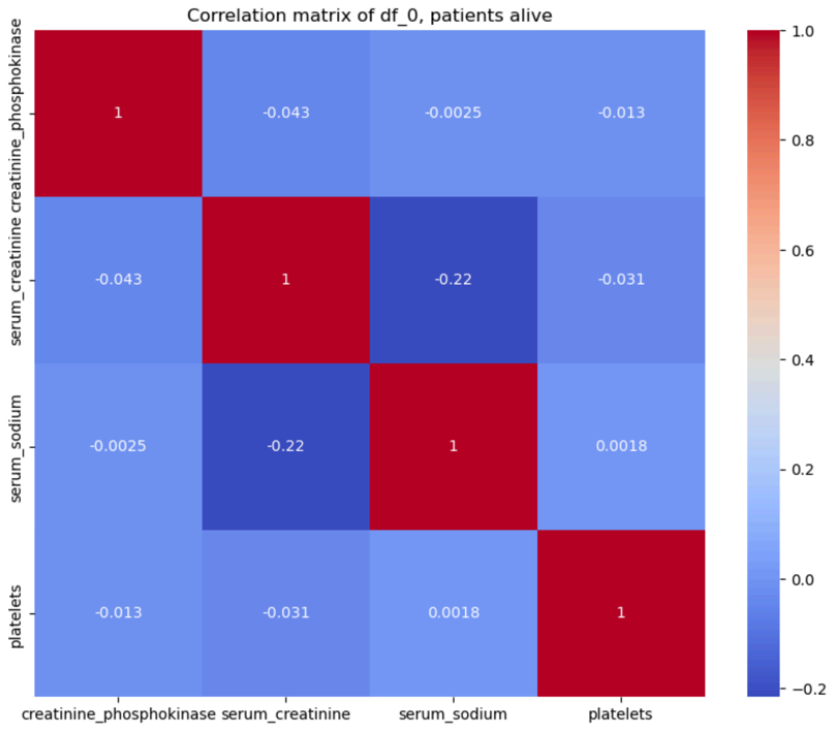
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
patients = pd.read_csv("heart_failure_clinical_records_dataset.csv")
```

```
df_0 = patients[patients['DEATH_EVENT'] == 0][['creatinine_phosphokinase', 'serum_creatinine', 'serum_sodium', 'platelets']]

df_1 = patients[patients['DEATH_EVENT'] == 1][['creatinine_phosphokinase', 'serum_creatinine', 'serum_sodium', 'platelets']]
```

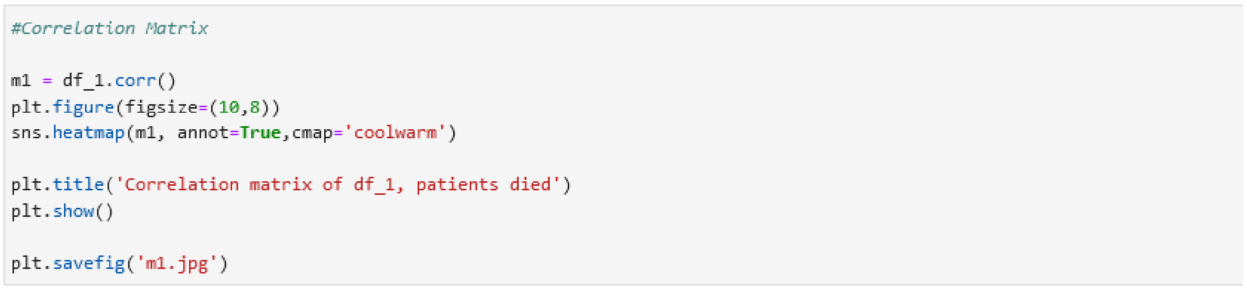
1.2 for each dataset, construct the visual representations of correponding correlation matrices M0 (from df 0) and M1 (from df 1) and save the plots into two separate les

```
#Correlation Matrix

m0 = df_0.corr()
plt.figure(figsize=(10,8))
sns.heatmap(m0, annot=True, cmap='coolwarm')

plt.title('Correlation matrix of df_0, patients alive')
plt.show()

plt.savefig('m0.jpg')
```

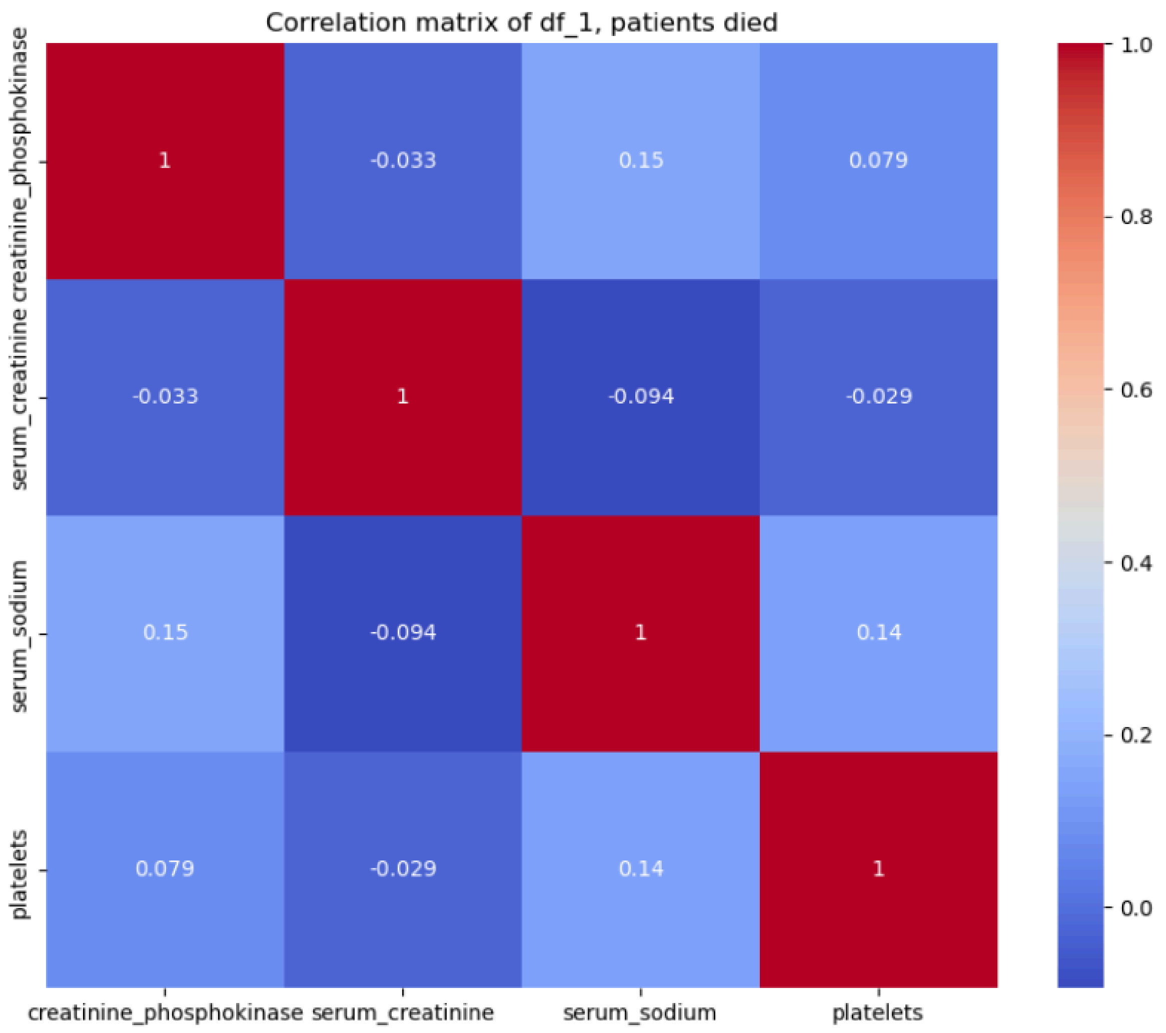


```
<Figure size 640x480 with 0 Axes>
```

Data Science part



```
#Correlation Matrix

m1 = df_1.corr()
plt.figure(figsize=(10,8))
sns.heatmap(m1, annot=True,cmap='coolwarm')

plt.title('Correlation matrix of df_1, patients died')
plt.show()

plt.savefig('m1.jpg')
```



3. examine your correlation matrix plots visually and answer the following:
(a) which features have the highest correlation for surviving Patients?
Answer: Serum_creatinine and serum_sodium
(b) which features have the lowest correlation for surviving Patients?
Answer: Serum_sodium and platelets
(c) which features have the highest correlation for deceased

Data Science part

Patients?
Answer: creatine_phosphokinase and serum_sodium
(d) which features have the lowest correlation for deceased
Patients?
Answer: platelets and sodium_creatinine
(e) are results the same for both cases?
Answer: No, different.

Question 2:

Data Science part

11]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

def fit_and_evaluate(X_train, X_test, Y_train, Y_test, model, transformation=None):
    if transformation == 'log-log':
        Y_train = np.log(Y_train)
        Y_test_log = np.log(Y_test)

    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    if transformation == 'log-log':
        Y_pred = np.exp(Y_pred)
        Y_test = np.exp(Y_test_log)

    residuals = Y_test - Y_pred
    SSE = np.sum(residuals**2)
    return SSE, Y_pred

def analyze_subset(subset, x_feature, y_feature):
    X = subset[[x_feature]].values
    Y = subset[[y_feature]].values

    # Ensuring no zero or negative values for log transformation
    valid_indices = (X > 0) & (Y > 0)
    X = X[valid_indices].reshape(-1, 1)  # Reshaping X to be 2D
    Y = Y[valid_indices]

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, random_state=42)

    # Simple Linear Regression
    lin_reg = LinearRegression()
    SSE, Y_pred = fit_and_evaluate(X_train, X_test, Y_train, Y_test, lin_reg)
    print("Linear Regression SSE:", SSE)

    # Polynomial Regression (Quadratic)
    poly_features_2 = PolynomialFeatures(degree=2, include_bias=False)
    X_train_poly2 = poly_features_2.fit_transform(X_train)
    X_test_poly2 = poly_features_2.transform(X_test)
    SSE, Y_pred = fit_and_evaluate(X_train_poly2, X_test_poly2, Y_train, Y_test, lin_reg)
    print("Quadratic Regression SSE:", SSE)

    # Polynomial Regression (Cubic)
    poly_features_3 = PolynomialFeatures(degree=3, include_bias=False)
    X_train_poly3 = poly_features_3.fit_transform(X_train)
    X_test_poly3 = poly_features_3.transform(X_test)
    SSE, Y_pred = fit_and_evaluate(X_train_poly3, X_test_poly3, Y_train, Y_test, lin_reg)
    print("Cubic Regression SSE:", SSE)

    # GLM: y = a*log(x) + b
    X_train_log = np.log(X_train)
    X_test_log = np.log(X_test)
    SSE, Y_pred = fit_and_evaluate(X_train_log, X_test_log, Y_train, Y_test, lin_reg)
    print("GLM (log-linear) SSE:", SSE)

    # GLM: log(y) = a*log(x) + b (log-log)
    SSE, Y_pred = fit_and_evaluate(X_train_log, X_test_log, Y_train, Y_test, lin_reg, transformation='log-log')
    print("GLM (log-log) SSE:", SSE)

# Assuming 'patients' is your DataFrame and is already loaded
# Extracting the subsets for surviving and deceased patients
surviving_patients = patients[patients['DEATH_EVENT'] == 0]
deceased_patients = patients[patients['DEATH_EVENT'] == 1]

# Applying the analysis to each subset
analyze_subset(surviving_patients, 'creatinine_phosphokinase', 'platelets')
analyze_subset(deceased_patients, 'creatinine_phosphokinase', 'platelets')
```

Data Science part

Question 3:
Answer:

Model	SSE (death event=0)	(death event=1)
y = ax + b	1062012216122.4734	387967154665.1771
y = ax2 + bx + c	1077552289006.5336	386831416885.73804
y = ax3 + bx2 + cx + d	1099720270641.245	393235090242.4724
y = a log x + b	1090577997860.4241	394514732600.9402
log y = a log x + b	1112836698424.6328	423971658753.32776

**Question 3.1: which model was the best (smallest SSE) for surviving pa-
tients? for deceased patients?**
Answer: Linear Model was the best because it had the lowest SSE

**Question 3.2: which model was the worst (largest SSE) for surving pa-
tients? for deceased patients?**
Answer: Logarithmic Model was the worst for surviving patients because it had the highest SSE.