Heart Failure 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:

```
5]: patients.shape
```

```
5]: (299, 13)
```

```
6]: patients.describe()
```

6]:

	age	anaemia	creatinine_phosphokinase	diabetes	eject
count	299.000000	299.000000	299.000000	299.000000	
mean	60.833893	0.431438	581.839465	0.418060	
std	11.894809	0.496107	970.287881	0.494067	
min	40.000000	0.000000	23.000000	0.000000	
25%	51.000000	0.000000	116.500000	0.000000	
50%	60.000000	0.000000	250.000000	0.000000	
75%	70.000000	1.000000	582.000000	1.000000	
max	95.000000	1.000000	7861.000000	1.000000	

```
9]: df_0 = patients[patients['DEATH_EVENT'] == 0]
```

```
.1]: df_1 = patients[patients['DEATH_EVENT'] == 1]
```

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 files
Answer:

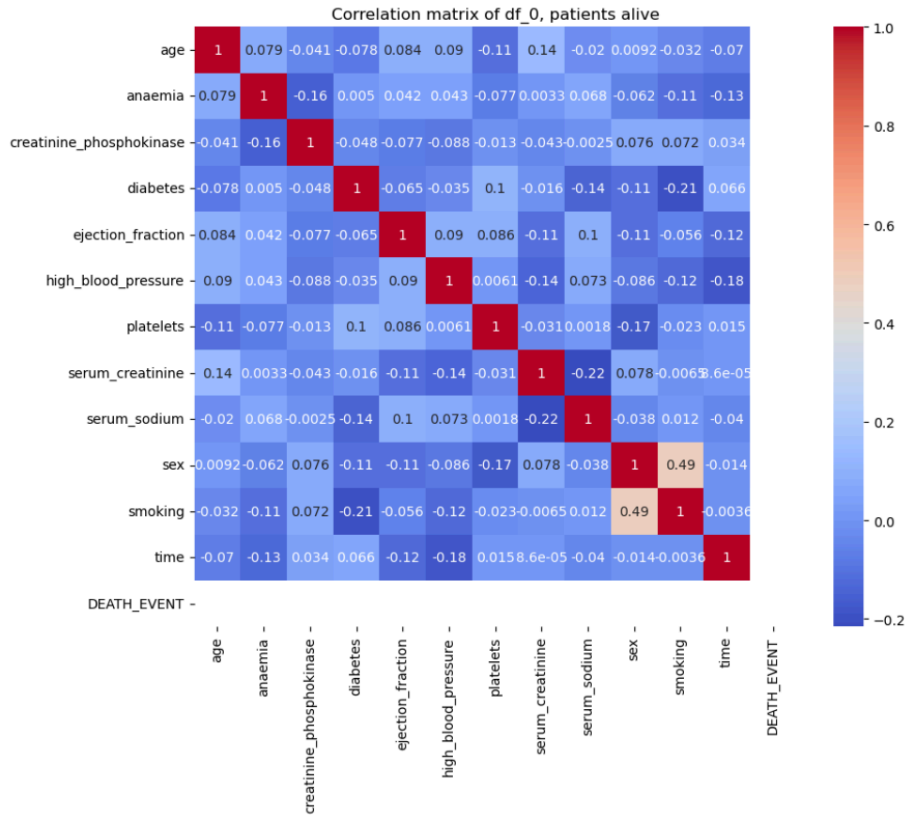
Heart Failure Part.

```
#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>
```

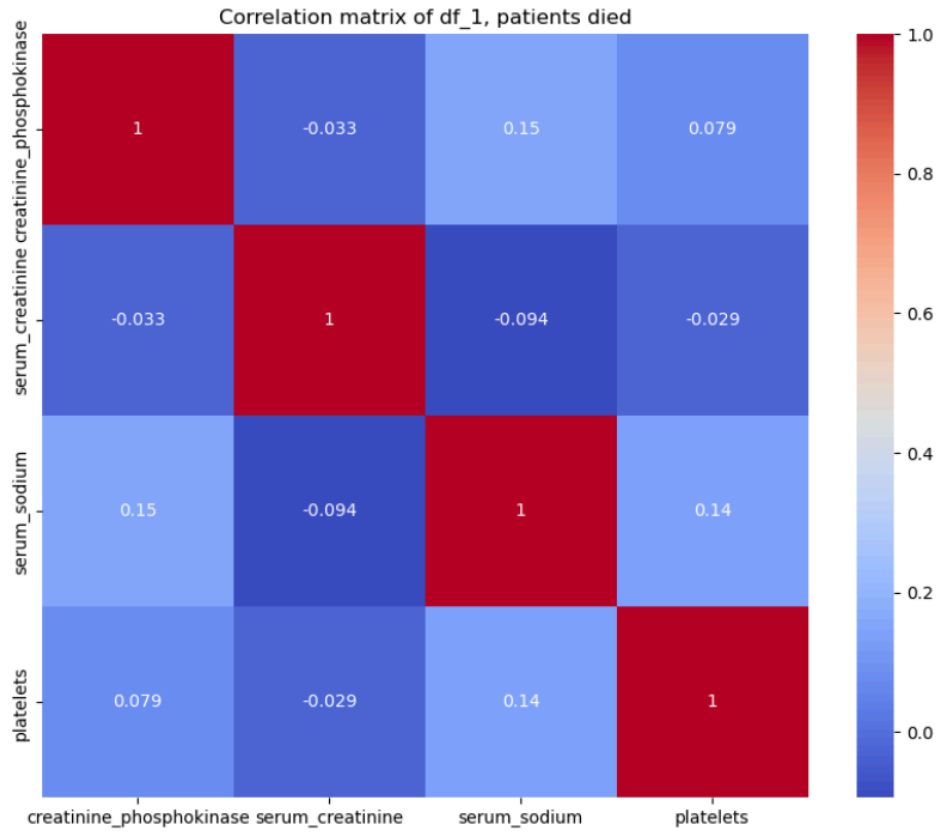
Heart Failure Part.

```
#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')
```



<Figure size 640x480 with 0 Axes>

1. 3. examine your correlation matrix plots visually and answer the following:

(a) which features have the highest correlation for surviving Patients?

Answer: Sex and smoking

(b) which features have the lowest correlation for surviving Patients?

Answer: Serum creatinine and time

(c) which features have the highest correlation for deceased Patients?

Answer: Sex and smoking

(d) which features have the lowest correlation for deceased Patients?

Answer: Serum creatinine and time

Heart Failure Part.

(e) are results the same for both cases?
Answer: Yes.

1. 4. for each class and for each feature f1, f2, f3, f4, compute its mean μ() and standard deviation σ(). Round the results to 2 decimal places and summarize them in a table as shown below:

```
(                   feature       μ(0)      σ(0)       μ(1)      σ(1)  \
 0  creatinine_phosphokinase     540.05    753.80     670.20   1316.58
 1          serum_creatinine       1.18      0.65       1.84      1.47
 2              serum_sodium     137.22      3.98     135.38      5.00
 3                 platelets  266657.49  97531.20  256381.04  98525.68
```

1. 5 examine your table. Are there any obvious patterns in the distribution of in each class

Mean: The mean value is higher in deceased patients (670.20) compared to surviving patients (540.05). This suggests that higher levels of creatinine phosphokinase may be associated with a higher risk of mortality in this dataset.
Standard Deviation: The standard deviation is significantly higher in deceased patients (1316.58) than in surviving patients (753.80), indicating more variability in creatinine phosphokinase levels among deceased patients.
Serum Creatinine (f2):

Mean: Deceased patients have a higher average serum creatinine (1.84) than surviving patients (1.18), which might indicate that higher serum creatinine levels could be linked to higher mortality.
Standard Deviation: Again, deceased patients show a higher standard deviation (1.47) compared to surviving patients (0.65), suggesting greater variability among the deceased.
Serum Sodium (f3):

Mean: The mean serum sodium is lower in deceased patients (135.38) than in surviving patients (137.22), possibly indicating that lower serum sodium levels might be associated with higher mortality.
Standard Deviation: The standard deviation is larger in the deceased group (5.00) than in the surviving group (3.98), indicating more variability among those who did not survive.
Platelets (f4):

Mean: The mean value is slightly higher in surviving patients (266657.49) than in deceased patients (256381.04), but this difference is not as pronounced as in the other features.
Standard Deviation: Variability (standard deviation) in platelet counts is roughly similar between the two groups, with no obvious pattern indicating a link to mortality.

Heart Failure Part.

2.1 split your dataset X into training Xtrain and Xtesting parts (50/50 split). Using ”pairplot” from seaborn package, plot pairwise relationships in Xtrain separately for class 0 and class 1. Save your results into 2 pdf files ”survived.pdf” and ”not-survived.pdf”

```
from sklearn.model_selection import train_test_split

df_0 = df_0[features]
df_1 = df_1[features]

X_train_0, X_test_0 = train_test_split(df_0, test_size=0.5, random_state=42)

X_train_0.shape, X_test_0.shape
```

```
((101, 4), (102, 4))
```

```
X_train_1, X_test_1 = train_test_split(df_1, test_size=0.5, random_state=42)

X_train_1.shape, X_test_1.shape
```

```
((48, 4), (48, 4))
```

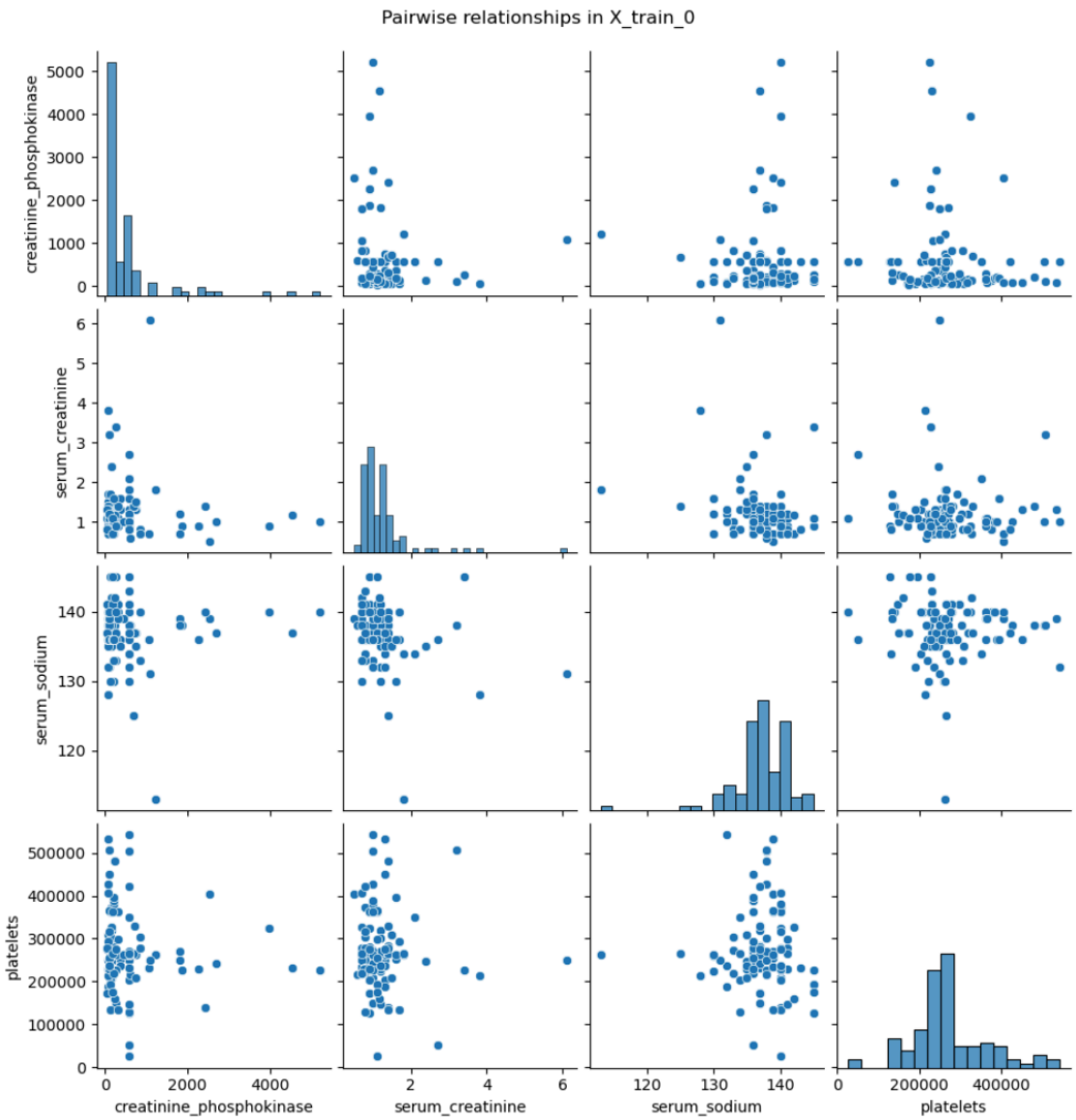
```
pairplot = sns.pairplot(X_train_0)

# Adjust the plot title and layout
plt.suptitle("Pairwise relationships in X_train_0", y=1.02)
plt.savefig("survived.pdf")
plt.show()
```

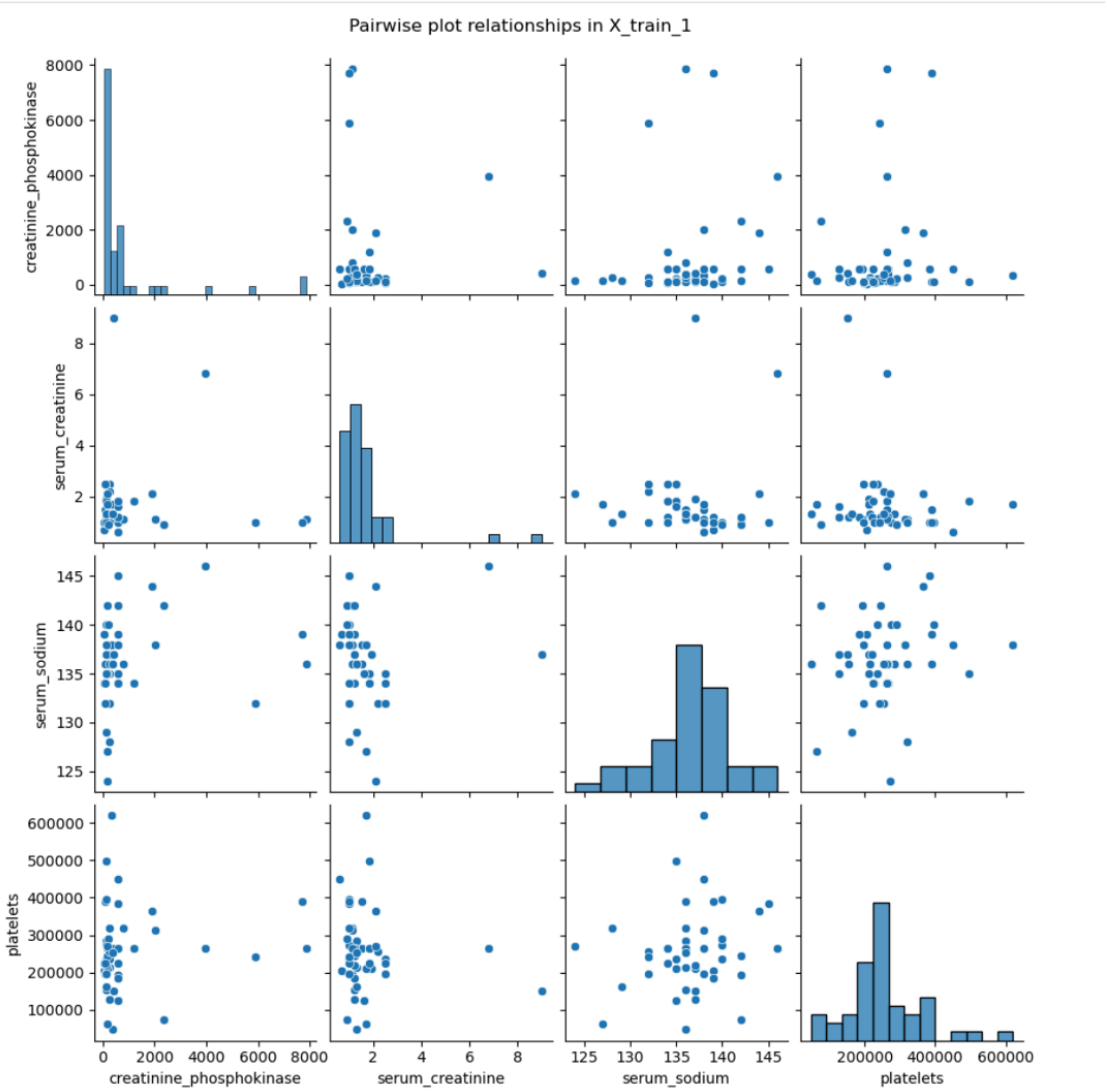
```
pairplot_dead = sns.pairplot(X_train_1, hue='category')

plt.suptitle("Pairwise plot relationships in X_train_1", y=1.02)
plt.savefig("not_survived.pdf")
plt.show()
```

Heart Failure Part.



Heart Failure Part.



2.2 visually examine your results. Come up with three simple comparisons that you think may be sufficient to predict a survival. For example, your classifier may look like this:
Answer: #Here's my classifier ~(((df['serum_sodium'] > 130) & (df['serum_sodium'] < 144)) & (df['creatinine_phosphokinase'] < 1000))

2.3 apply your simple classifier to Xtest and compute predicted class labels

Heart Failure Part.

```
]: #Here's my classifier ~(((df['serum_sodium'] > 130) & (df['serum_sodium'] < 144)) & (df['creatinine_phosphokinase'] < 1000))
X = patients.drop(columns=['DEATH_EVENT'])  # or any other column that's your target variable
y = patients['DEATH_EVENT']

# Split the data into training and testing sets (50/50 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

```
]: #apply classifier
predicted_labels = ~(((X_test['serum_sodium'] > 130) & (X_test['serum_sodium'] < 144)) & (X_test['creatinine_phosphokinase'] < 1000))
predicted_labels = predicted_labels.astype(int)
```

2.5 Summarize the findings in the table.

TP	FP	TN	FN	Accuracy	TPR	TNR
14	22	73	41	0.58	0.2545454545454545	0.7684210526315789

2.6 does your simple classifier gives you higher accuracy on identifying ”fake” bills or ”real” bills” Is your accuracy better than 50% (”coin” flipping)?
Answer: Yes.

Question 3.1 take k = 3, 5, 7. For each k, generate Xtrain and Xtest using 50/50 split as before. Train your k-NN classifier on Xtrain and compute its accuracy for Xtest

Heart Failure Part.

Answer:

```
: #Question 3

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Assuming 'label' is the column name for your target variable
X = patients.drop(columns=['DEATH_EVENT'])  # Replace 'label' with your actual target column name
y = patients['DEATH_EVENT']

# List of k values
k_values = [3, 5, 7]

# Dictionary to store accuracies for each k
accuracies = {}

for k in k_values:
    # Split the data into training and testing sets (50/50 split)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

    # Initialize the k-NN classifier with the current k
    knn = KNeighborsClassifier(n_neighbors=k)

    # Train the classifier
    knn.fit(X_train, y_train)

    # Predict the labels for the test set
    y_pred = knn.predict(X_test)

    # Compute the accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Store the accuracy in the dictionary
    accuracies[k] = accuracy

# Output the accuracies for each k
print(accuracies)
```

```
{3: 0.64, 5: 0.64, 7: 0.6133333333333333}
```

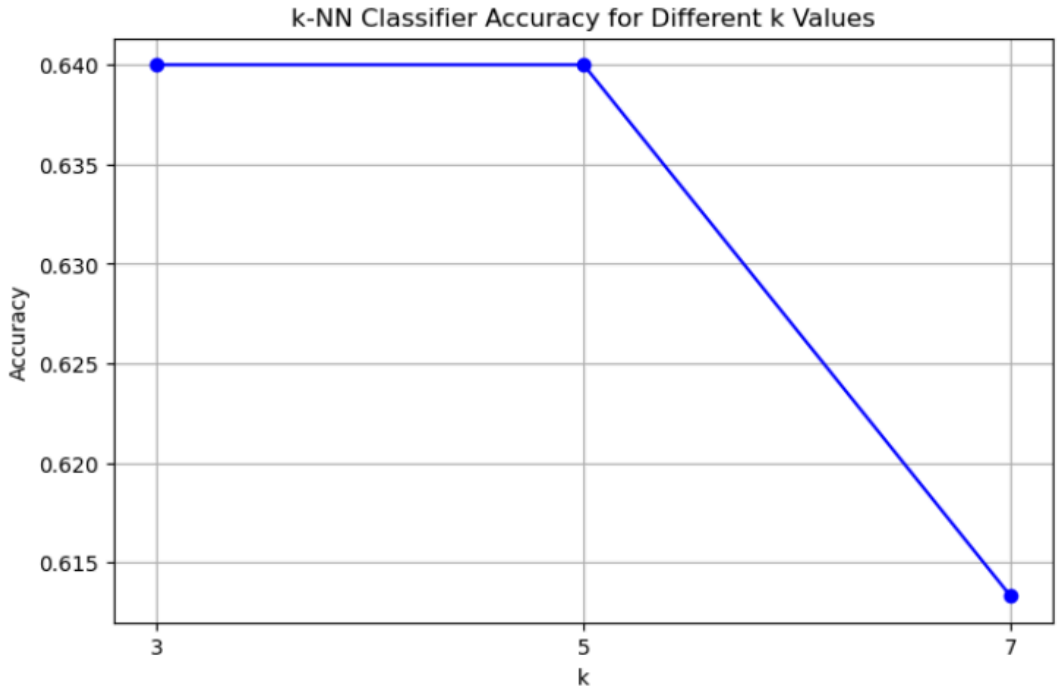
3.2 plot a graph showing the accuracy. On x axis you plot k and on y-axis you plot accuracy. What is the optimal value $k*$ of k?

Heart Failure Part.

```
]: k_values = list(accuracies.keys())
accuracy_values = list(accuracies.values())

# Plotting the accuracies
plt.figure(figsize=(8, 5))
plt.plot(k_values, accuracy_values, marker='o', linestyle='-', color='b')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('k-NN Classifier Accuracy for Different k Values')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Determine the optimal value of k (k*), where the accuracy is maximum
optimal_k = k_values[accuracy_values.index(max(accuracy_values))]
print("The optimal value of k (k*) is:", optimal_k)
```



Optimal value is 3

3.3 use the optimal value k∗ to compute performance measures and summarize them in the table

TP	FP	TN	FN	Accuracy	TPR	TNR
11	10	85	44	0.64	0.2	0.8947368421052632

Heart Failure Part.

3.4 is your k-NN classifier better than your simple classifier for any of the measures from the previous table?
Answer: Yes, accuracy is higher. Everything else, not so much.

Question 4.1 take your best value k∗. For each of the four features f1, . . . , f4, generate new Xtest and Xtrain and drop that feature from both Xtrain and Xtest. Train your classifier on the ”truncated” Xtrain and predict labels on Xtest using just 3 remaining features. You will repeat this for 4 cases: (1) just f1 is missing, (2) just f2 is missing, (3) just f3 missing and (4) just f4 is missing. Compute the accuracy for each of these scenarios.
Answer:

```
accuracies = {}

for feature in features:
    # Drop the current feature from the dataset
    X_modified = patients.drop(columns=[feature, 'DEATH_EVENT'])  # Replace 'label' with your target variable
    y = patients['DEATH_EVENT']

    # Split the modified dataset into training and testing sets (50/50 split)
    X_train_modified, X_test_modified, y_train, y_test = train_test_split(X_modified, y, test_size=0.5, random_state=42)

    # Initialize the k-NN classifier using the optimal k value
    knn = KNeighborsClassifier(n_neighbors=3)

    # Train the classifier on the truncated X_train
    knn.fit(X_train_modified, y_train)

    # Predict the labels for the truncated X_test
    y_pred = knn.predict(X_test_modified)

    # Compute the accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Store the accuracy in the dictionary
    accuracies[feature] = accuracy

# Output the accuracies for each scenario
for feature, acc in accuracies.items():
    print(f"Accuracy without the feature '{feature}': {acc:.4f}")
```

```
Accuracy without the feature 'creatinine_phosphokinase': 0.6533
Accuracy without the feature 'serum_creatinine': 0.6400
Accuracy without the feature 'serum_sodium': 0.6400
Accuracy without the feature 'platelets': 0.7333
```

4.2 did accuracy increase in any of the 4 cases compared with accuracy when all 4 features are used
Answer: It did substantially, accuracy when platelets was removed improved substantially, in line with my original analysis that platelets offered less than what it demanded. I want 10 watts of my computing power back.

4.3 which features, when removed, contributed the most to loss of accuracy?

Heart Failure Part.

Answer: Serum_creatinine and serum_sodium.

4.4 which features, when removed, contributed the least to loss of accuracy?
Answer: Platelets.

Question 5.1 Use 50/50 split to generate new Xtrain and Xtest. Train your logistic regression classifier on Xtrain and compute its accuracy for Xtest

```
|: from sklearn.linear_model import LogisticRegression

X = patients.drop(columns=['DEATH_EVENT'])
y = patients['DEATH_EVENT']

# Split the data into training and testing sets (50/50 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# Initialize the logistic regression classifier
logistic_regression = LogisticRegression(max_iter=1000)  # max_iter may need adjustment for convergence

# Train the classifier on the training data
logistic_regression.fit(X_train, y_train)

# Predict the target variable for the test data
y_pred = logistic_regression.predict(X_test)

# Compute the accuracy of the classifier on the test data
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of the logistic regression classifier on the test set: {accuracy:.4f}")
```

```
Accuracy of the logistic regression classifier on the test set: 0.7867
```

5.2 Performance measure in the table.
Answer:

TP	FP	TN	FN	Accuracy	TPR	TNR
11	10	85	44	0.7867	0.2	0.8947368421052632

5.3 is your logistic regression better than your simple classifier for any of the measures from the previous table?
Answer: Yes, accuracy is higher.

5.4 is your logistic regression better than your k-NN classifier (using the best k∗) for any of the measures from the previous

Heart Failure Part.

Table?
Answer: Yes, all measures.

6.1 For each of the four features f1, . . . , f4, generate new Xtrain and Xtest and drop that feature from both Xtrain and Xtest. Train your logistic regression classifier on the ”truncated” Xtrain and predict labels on ”truncated” Xtest using just 3 remaining features. You will repeat this for 4 cases: (1) just f1 is missing, (2) just f2 missing, (3) just f3 missing and (4) just f4 is missing. Compute the accuracy for each of these scenarios.

```
]: accuracies = {}

for feature in features:
    # Drop the current feature from the dataset
    X_modified = patients.drop(columns=[feature, 'DEATH_EVENT'])
    y = patients['DEATH_EVENT']

    # Split the modified dataset into training and testing sets (50/50 split)
    X_train_modified, X_test_modified, y_train, y_test = train_test_split(X_modified, y, test_size=0.5, random_state=42)

    # Initialize the logistic regression classifier
    logistic_regression = LogisticRegression(max_iter=1000)  # Adjust max_iter if needed for convergence

    # Train the classifier on the truncated X_train
    logistic_regression.fit(X_train_modified, y_train)

    # Predict the labels for the truncated X_test
    y_pred_modified = logistic_regression.predict(X_test_modified)

    # Compute the accuracy
    accuracy_modified = accuracy_score(y_test, y_pred_modified)

    # Store the accuracy in the dictionary
    accuracies[feature] = accuracy_modified

# Output the accuracies for each scenario
for feature, acc in accuracies.items():
    print(f"Accuracy without the feature '{feature}': {acc:.4f}")
```

```
Accuracy without the feature 'creatinine_phosphokinase': 0.7600
Accuracy without the feature 'serum_creatinine': 0.8133
Accuracy without the feature 'serum_sodium': 0.7733
Accuracy without the feature 'platelets': 0.8333
```

6.2 did accuracy increase in any of the 4 cases compared with accuracy when all 4 features are used?
Yes, all 4 cases.

6.3 which features, when removed, contributed the most to loss of accuracy?
Answer: Creatinine_phoskinase, led to the least gain of accuracy.

Heart Failure Part.

6.4 which features, when removed, contributed the least to loss of accuracy?
Answer: Platelets, led to the most gain of accuracy.

6.5 is the relative significance of features the same as you obtained using k-NN?
Answer: yes, the feature responsible of for the most loss in accuracy turned out to be adding the most accuracy in the analysis.