UNIWERSYTET RADOMSKI
im. Kazimierza Pułaskiego

WYDZIAŁ
TRANSPORTU, ELEKTROTECHNIKI
I INFORMATYKI

Kierunek: Informatyka

Analiza Zgonów z Powodu Niewydolności Serca

Sztuczna Inteligencja

Wykonał(a):

Miłosz Milosavljević

nr albumu 112209

112209@student.uthrad.pl

Prowadzący:

dr Jacek Wołoszyn

Radom 2024

Analiza Eksploracyjna Danych (EDA) Dla Zbioru Danych O Niewydolności Serca

1. Cel

Celem mojej analizy eksploracyjnej danych było zrozumienie zależności oraz struktury zestawu danych dotyczącego niewydolności serca. Dzięki analizie udało mi się wykryć powiązane ze sobą właściwości oraz pozwoliło mi wykryć wzorce pozwalające na wykrycie potencjalnego zgonu z powodu niewydolności serca.

2. Analiza Danych

2.1. Wczytanie I Analiza Danych

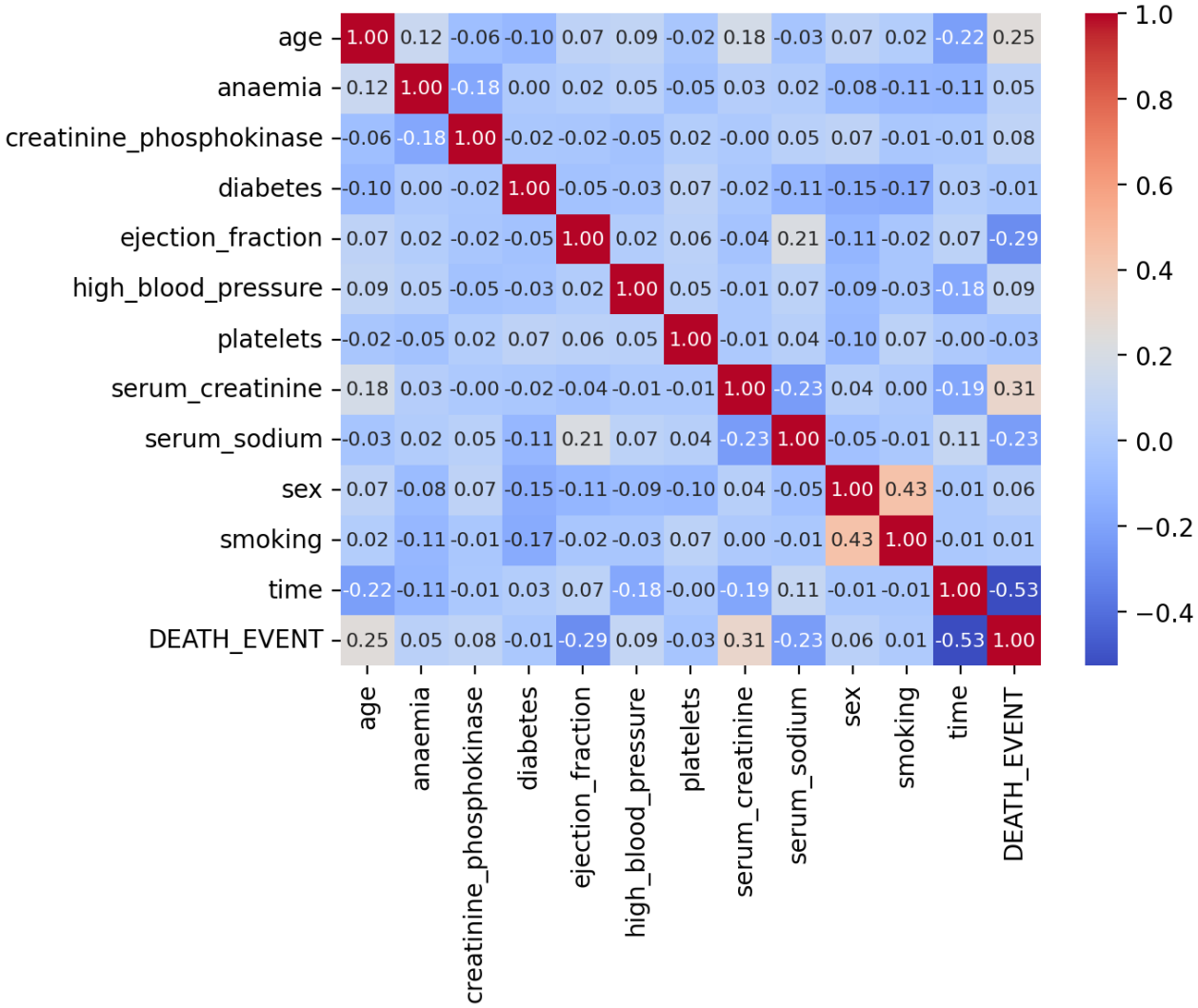
Pracę nad projektem rozpocząłem od wczytania danych z pliku CSV przy użyciu biblioteki Pandas. Następnie wyświetliłem pierwsze 10 wierszy danych, aby zobaczyć przybliżoną strukturę oraz zakres wartości. Utworzyłem tabele z wartościami średnimi, mediany, odchylania standardowego, wartości minimalnej oraz maksymalnej dla każdej cechy zestawu danych.

2.2. Analiza Korelacji

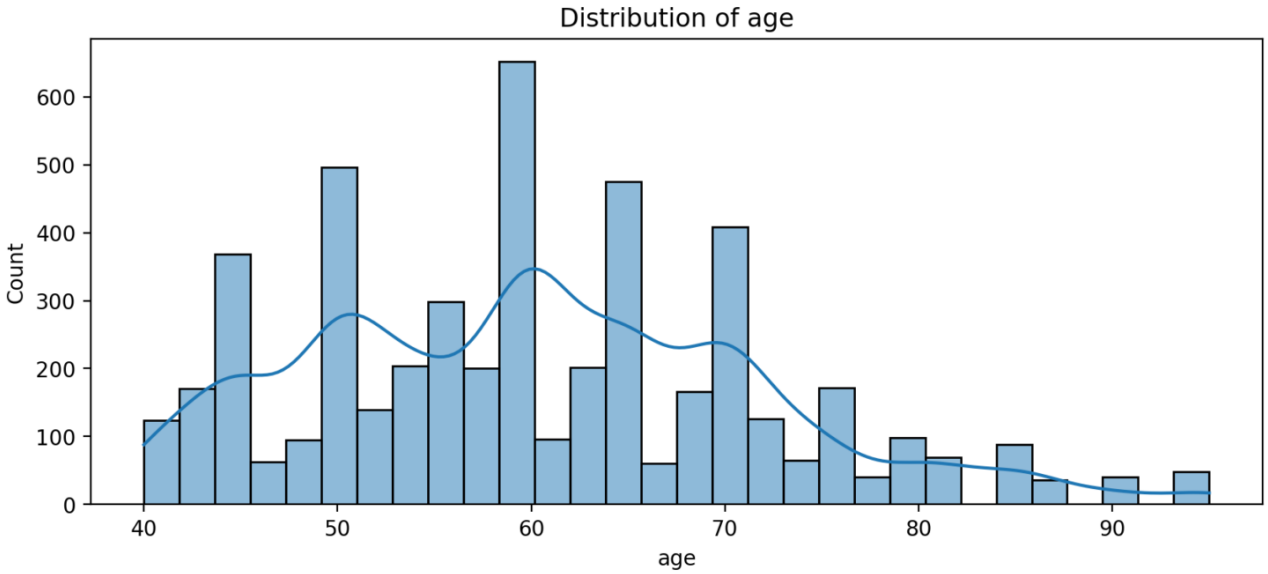
Do analizy korelacji wykorzystałem macierz korelacji, pokazującą jak cechy zmieniają się względem siebie. Wykorzystałem również mapę termiczną (Rysunek 1), która pokazuje w kolorze, jak cechy zmieniają się względem siebie.

2.3. Wizualizacja Rozkładu Własności

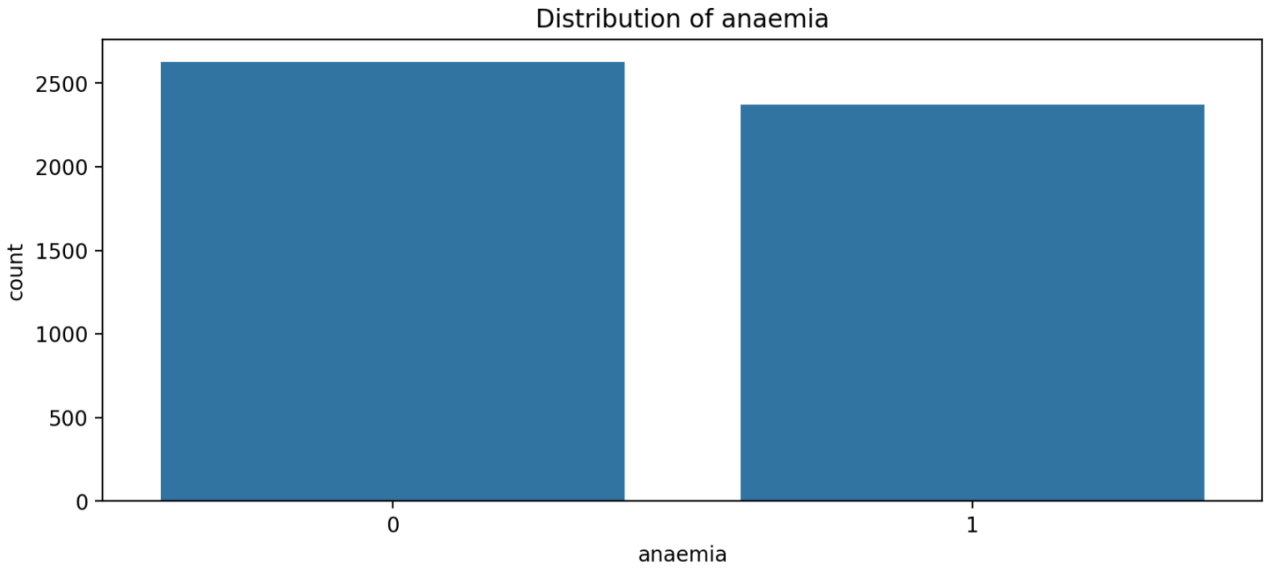
Stworzyłem histogramy odpowiadające każdej z cech zestawu danych, takich jak wiek (Rysunek 2), anemia (Rysunek 3), fosfokinaza kreatyniny (miligram/litr; Rysunek 4), cukrzyca (Rysunek 5), frakcja wyrzutowa — procent krwi opuszczającej serce przy każdym skurczu, nadciśnienie (Rysunek 6), nadciśnienie (Rysunek 7), płytki krwii (kilopłytek/mililitr; Rysunek 8), kreatynina w surowicy (miligram/decylitr; Rysunek 9), sód w surowicy (miliekwiwalenty/litr; Rysunek 10), płeć (Rysunek 11), osoba paląca (Rysunek 12), czas pomiędzy wizytami (dni; Rysunek 13). Stworzone wykresy pozwoliły mi na zapoznanie się z rozłożeniem wartości w zbiorze danych, dzięki czemu mogłem zaobserwować potencjalne anomalie, symetryczność danych lub skośność.



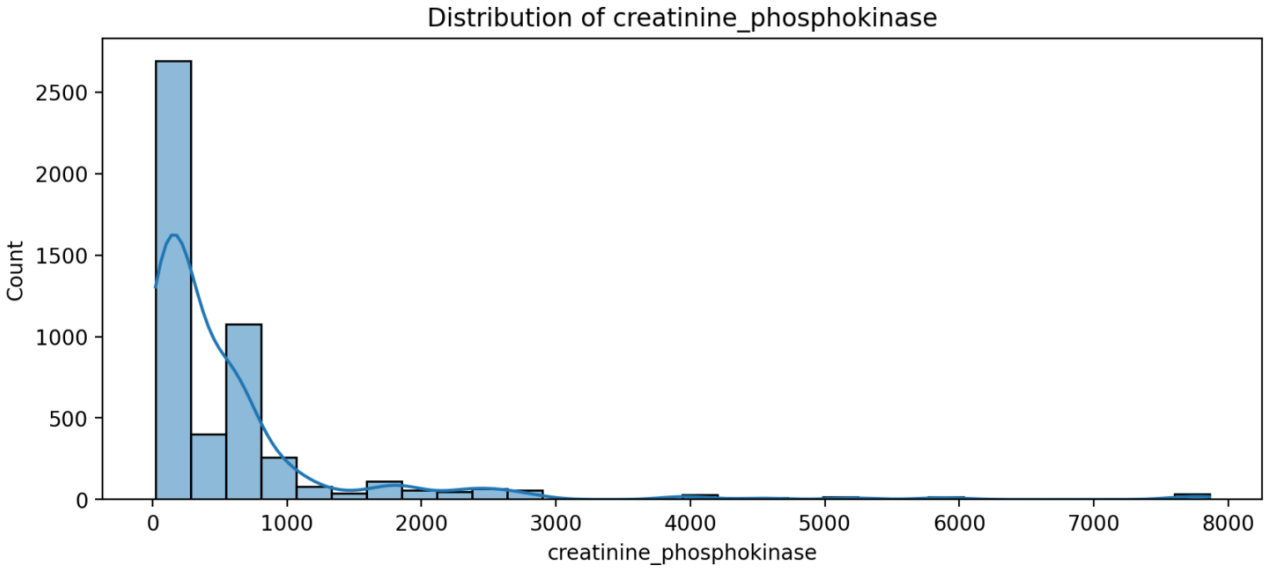
Rysunek 1. Mapa ciepła



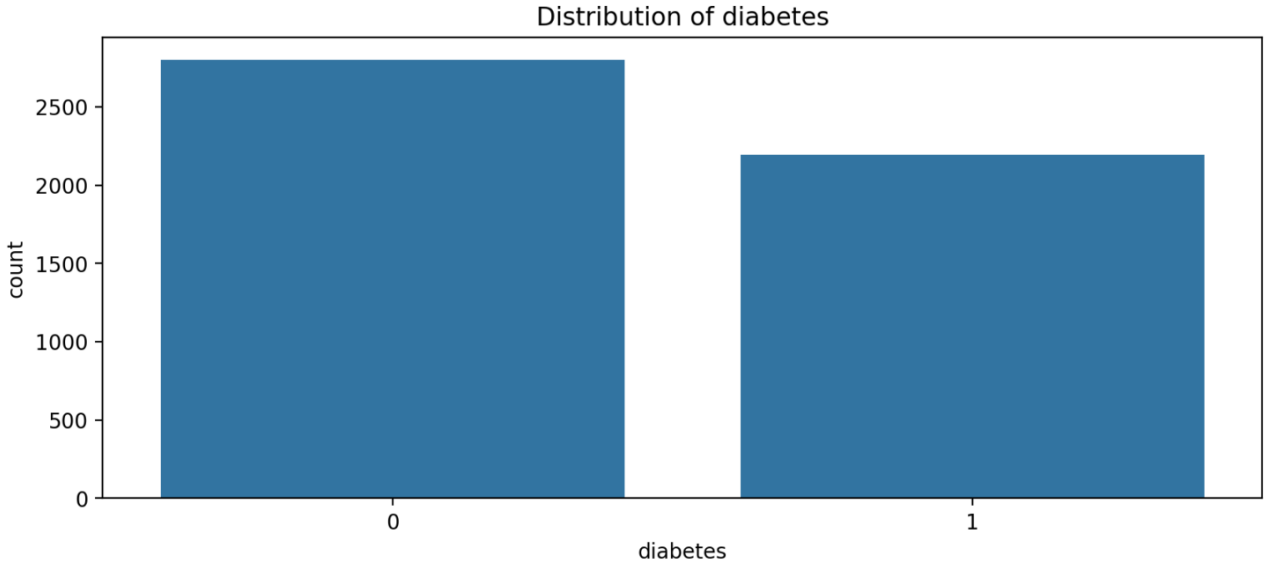
Rysunek 2. Rozkład wieku (age)



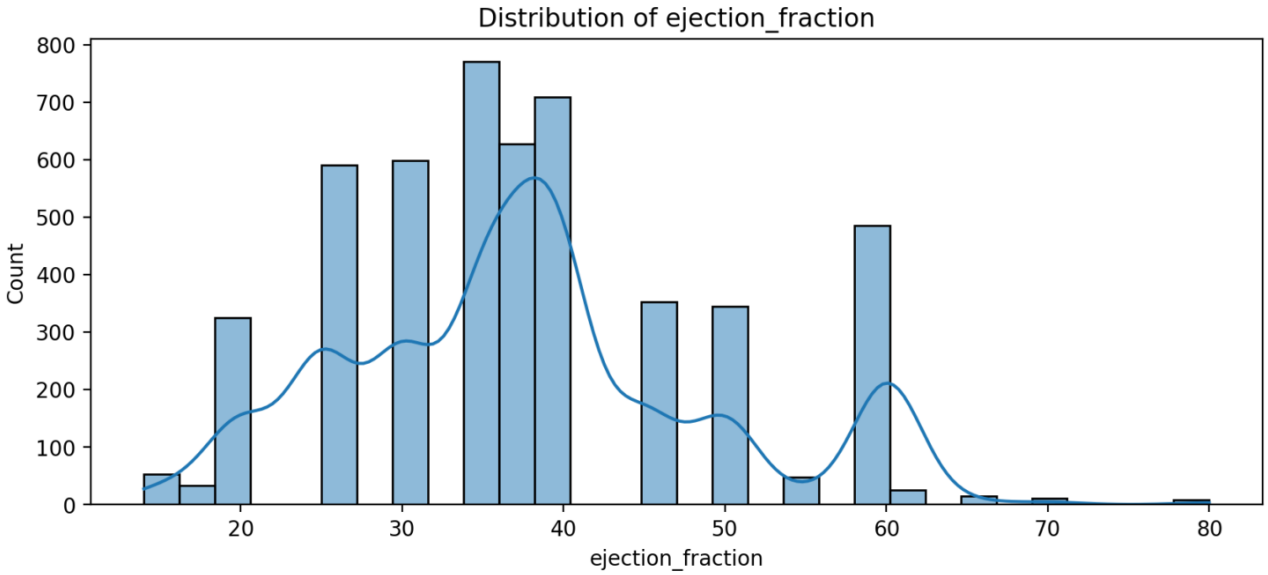
Rysunek 3. Rozkład anemii (anaemia)



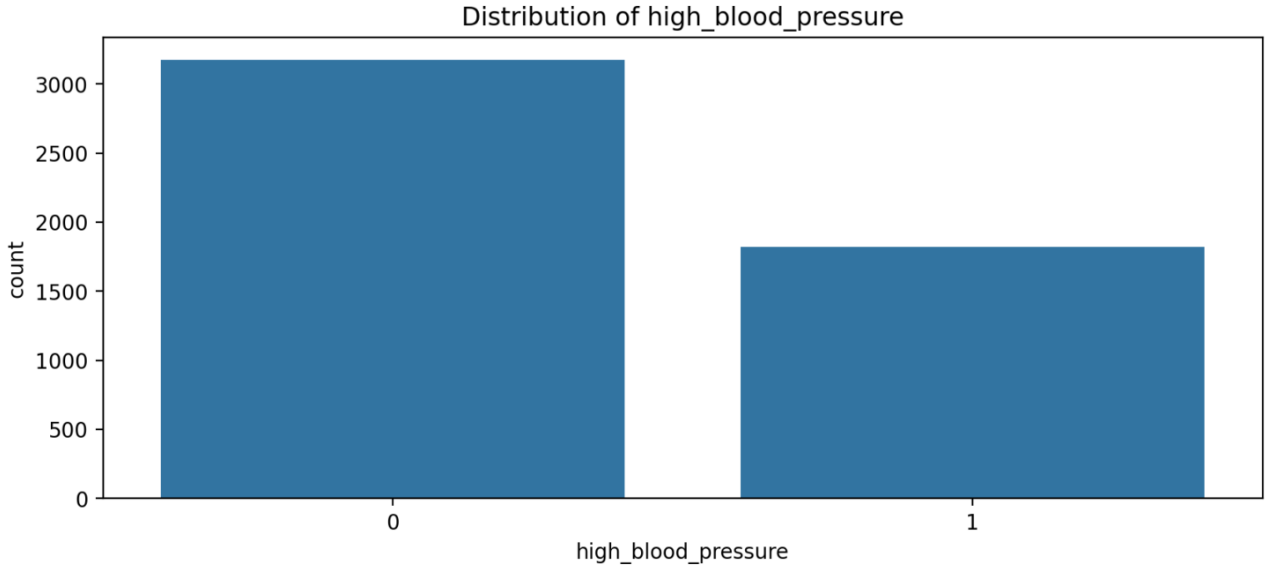
Rysunek 4. Rozkład fosfokinaza kreatyniny w mg/l (creatinine_phosphokinase)



Rysunek 5. Rozkład cukrzycy (diabetes)

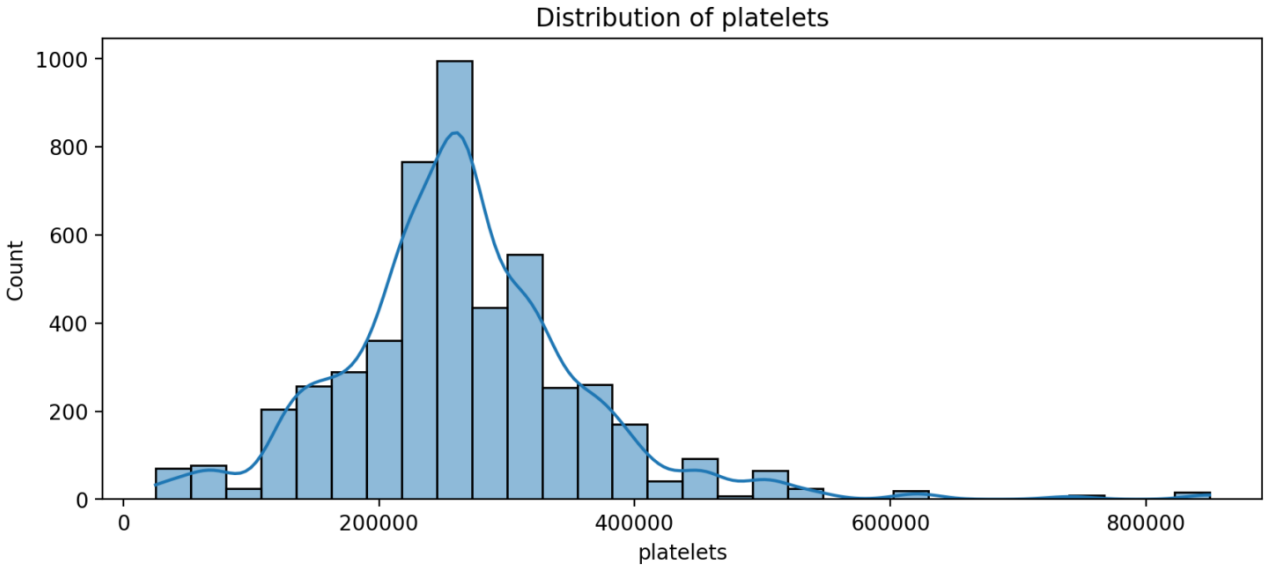


Rysunek 6. Rozkład frakcji wyrzutu w procentach (ejection_fraction)

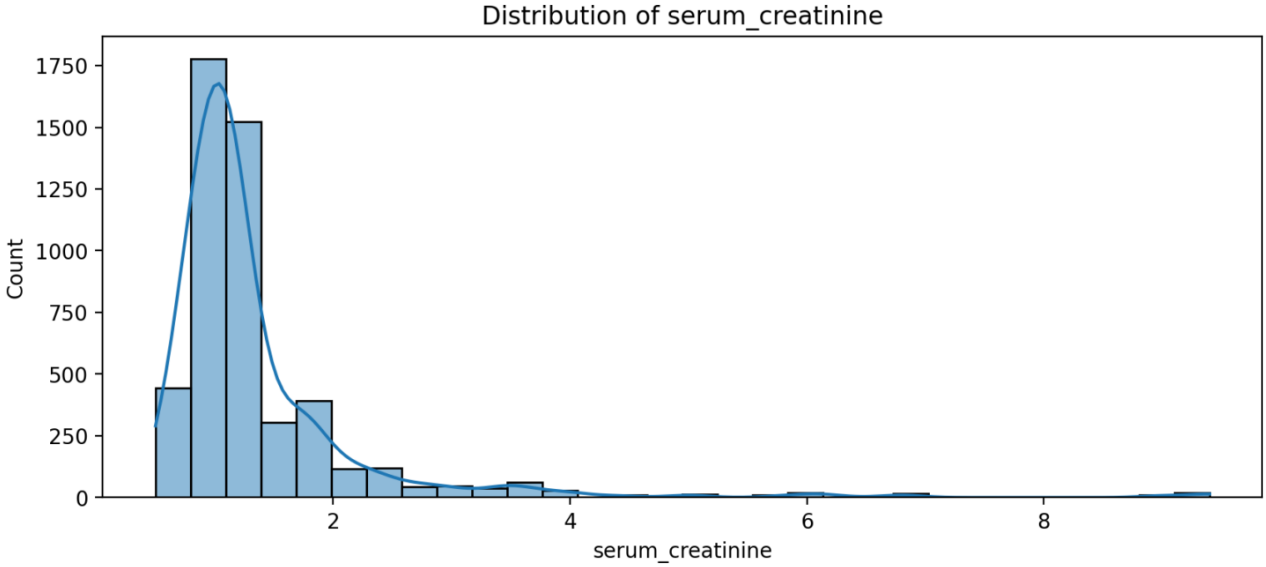


Rysunek 7. Rozkład nadciśnienia (high_blood_pressure)

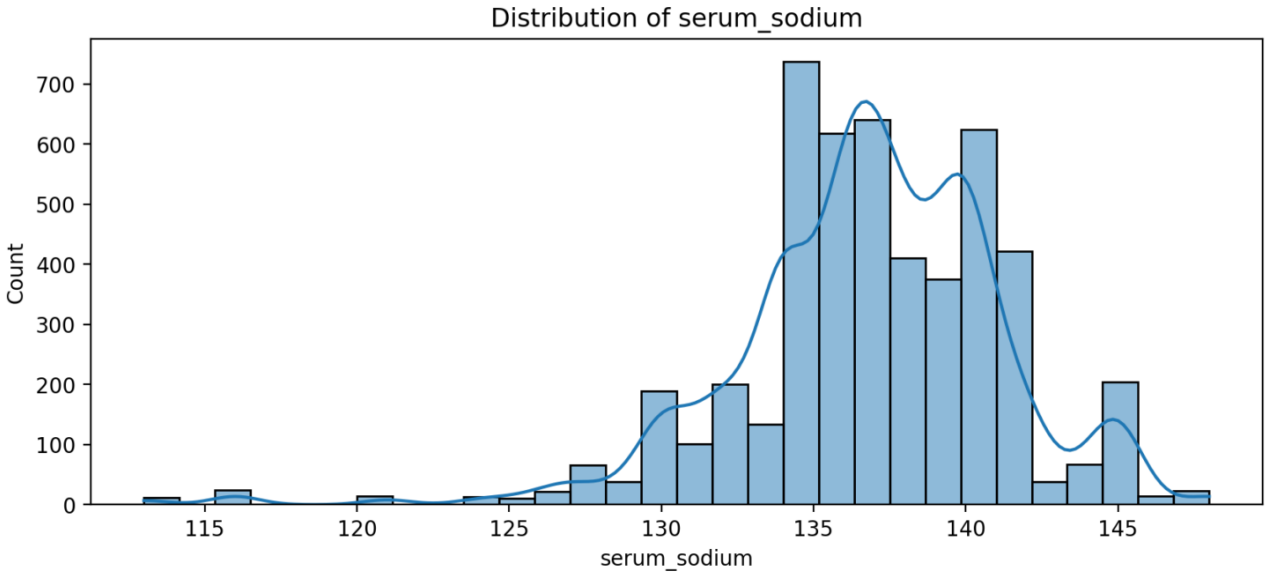
0 – brak nadciśnienia 1 – nadciśnienie



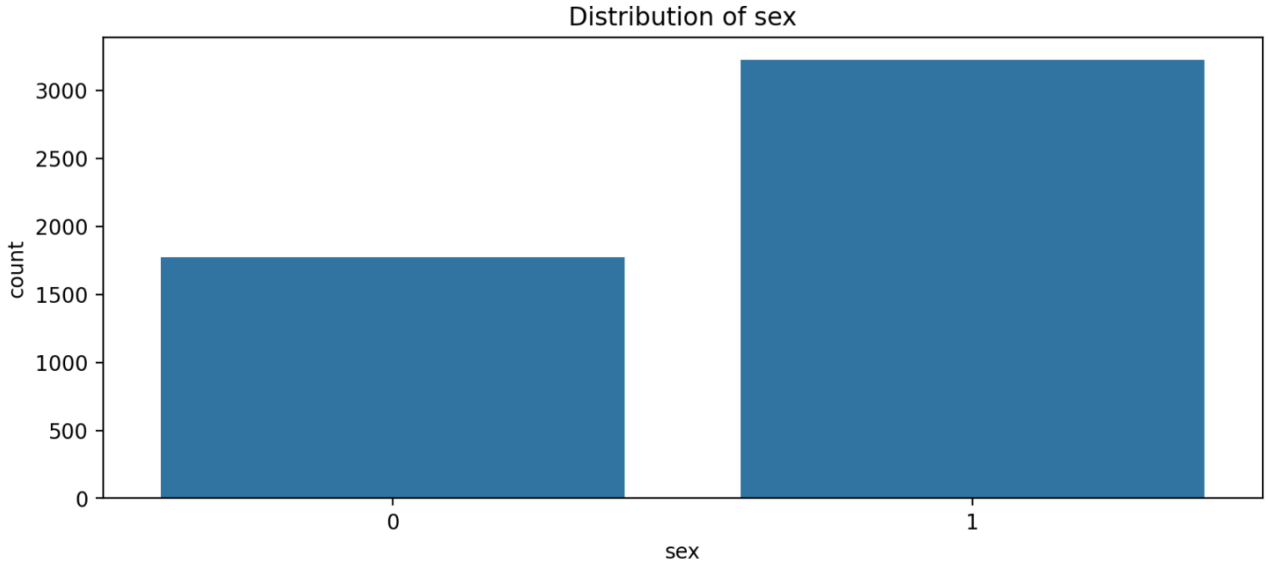
Rysunek 8. Rozkład płytek krwii w kilopłytek/ml (platelets)



Rysunek 9. Rozkład kreatyny w soczewicy mg/dl (serum_creatinine)

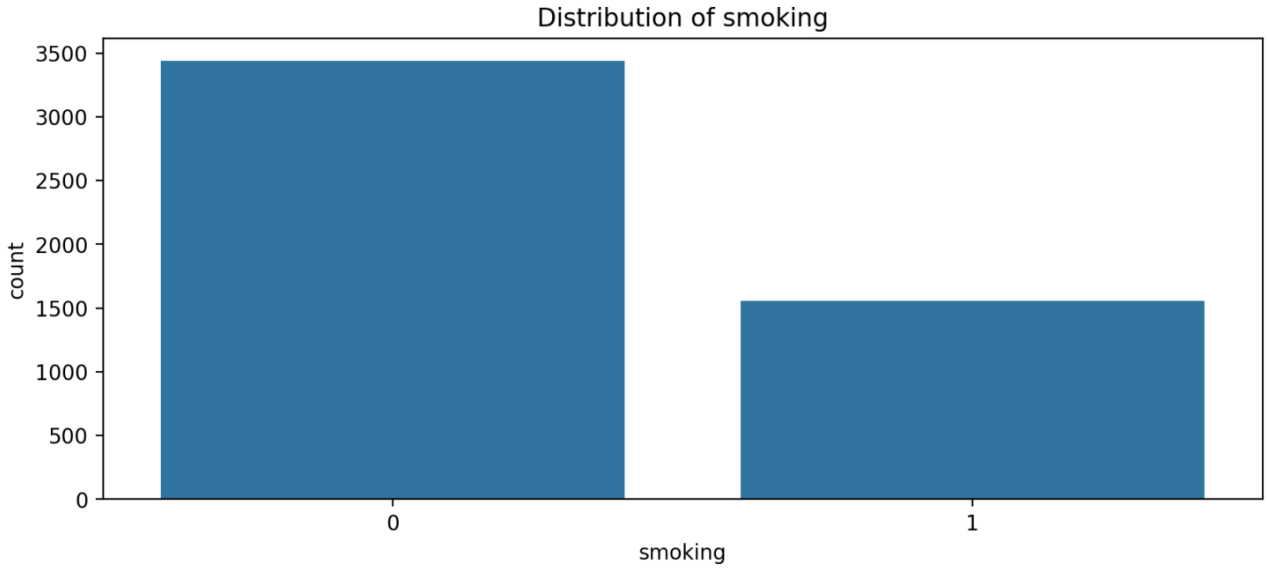


Rysunek 10. Rozkład sodu w soczewicy mEq/l (serum_sodium)



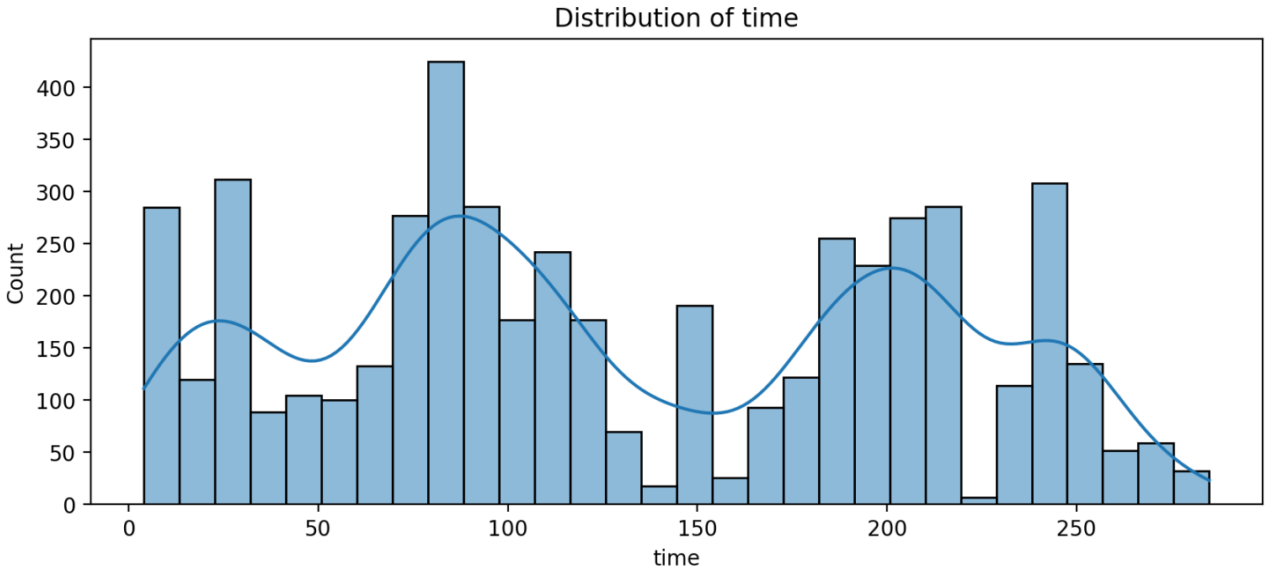
Rysunek 11. Rozkład płcii (sex)

0 – kobieta 1 – mężczyzna



Rysunek 12. Rozkład palaczy (smoking)

0 – niepalący 1 – palący



Rysunek 13. Rozkład czasu pomiędzy wizytami w dniach (time)

Z mapy ciepła (Rysunek 1) można odczytać, że największa korelacja ze śmiercią pacjenta (DEATH_EVENT) wystepuje dla czasu pomiędzy wizytami (time), wiekiem (age), frakcją wyrzutu (ejection fraction), kreatynina w soczewicy (serum creatinine) oraz sodą w soczewicy (serum sodium). Czas pomiędzy wizytami nie będzie brany pod uwagę, ponieważ w przypadku śmierci pacjenta czas ten był krótszy, ponieważ uwzględniał czas śmierci, co nie będzie pokazane w danych przy próbie przewidzenia śmierci przed jej wystąpieniem, to samo dotyczy wieku.

2.4. Kod Programu

main.py

```python
import pandas as pd
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sb

DATA = "./data/heart_failure_clinical_records.csv"

def loadCSV(path: str):
    df = pd.read_csv(path)
    showPlots(df)
    return df

def showPlots(df: pd.DataFrame):
    st.markdown("# Heart failure")
    st.markdown("## Datasets")

    # Og chart with no deaths
    df_no_death = df.copy(deep=True)
    df_no_death = df_no_death.drop(index=df_no_death[df_no_death["DEATH_EVENT"] == 0].index)
    df_no_death = df_no_death.drop("DEATH_EVENT", axis=1)
    df_no_death.sort_values(by="creatinine_phosphokinase")
    st.markdown("### No deaths")
    st.dataframe(df_no_death)
    st.markdown("#### Median")
    no_death_median = df_no_death.median()

    # No death median table creatinine
    st.table(no_death_median)

    # Chart with only deaths
    df_death = df.copy(deep=True)
    df_death = df_death.drop(index=df_death[df_death["DEATH_EVENT"] == 1].index)
    df_death = df_death.drop("DEATH_EVENT", axis=1)
    df_death.sort_values(by="creatinine_phosphokinase")
    st.markdown("### Only deaths")
    st.dataframe(df_death)
    st.markdown("#### Median")
    death_median = df_death.median()

    # Death median table creatinine
    st.table(death_median)

    st.markdown("### Corelation table")
    corr = df.corr()
    st.dataframe(corr)

    # Corelation heatmap
    sb.heatmap(corr, annot=True, fmt=".2f", annot_kws={"size": 8},  cmap='coolwarm')
    st.pyplot(plt)
    plt.close()

    # Pairplot
    relevant_columns = ['ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT']
    data_relevant = df[relevant_columns]
    plt.figure(figsize=(10, 6))
    sb.pairplot(data_relevant, hue="DEATH_EVENT")
    st.pyplot(plt)
    plt.close()

    # Histograms of each column
    for column in df.columns:
        if column == 'DEATH_EVENT':
            continue
        plt.figure(figsize=(10, 4))
        plt.title(f'Distribution of {column}')
        if column in ['sex', 'anaemia', 'diabetes', 'high_blood_pressure', 'smoking']:
            sb.countplot(data=df, x=column)
            st.pyplot(plt)
            plt.close()
            continue
        sb.histplot(df[column], kde=True, bins=30)
        st.pyplot(plt)
        plt.close()

if __name__ == "__main__":
    df = loadCSV(DATA)
```

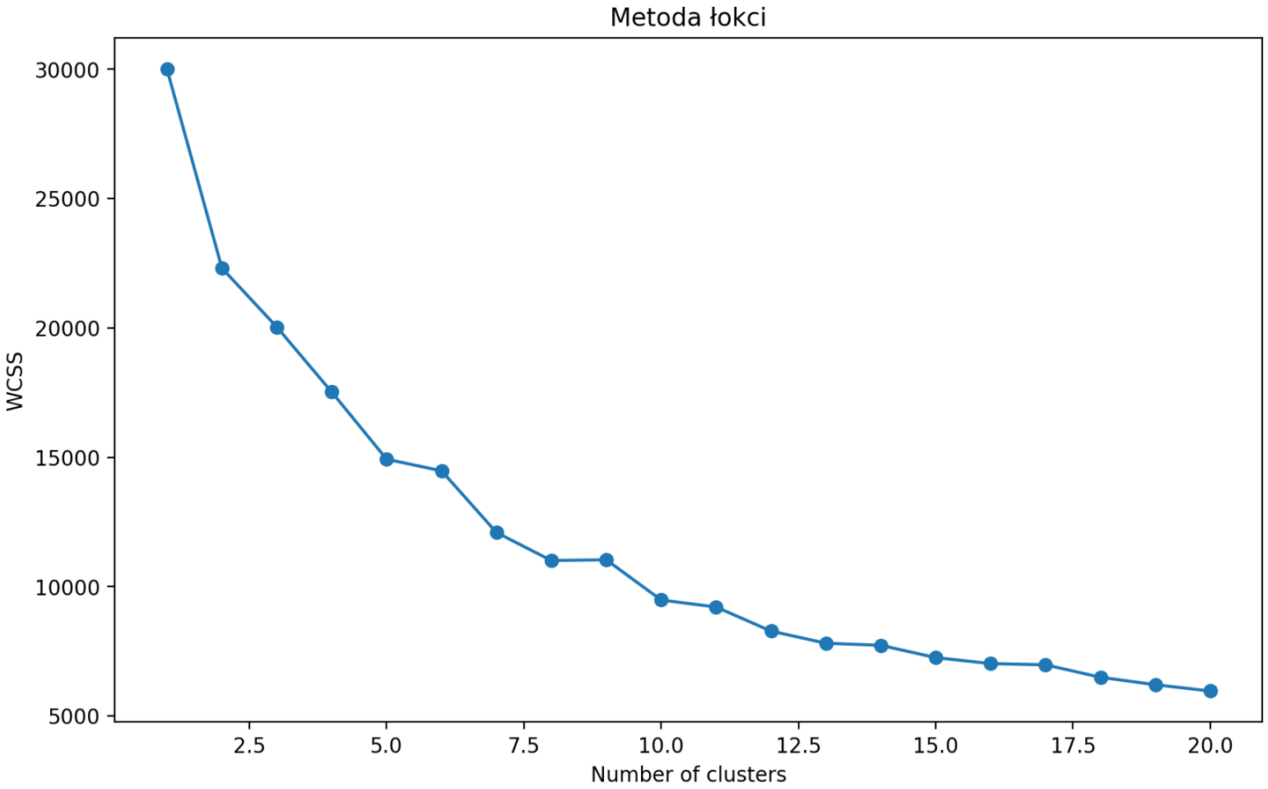
Rysunek 14. Kod programu pokazujący wykresy

3. Modele Predykcyjne I Klasteryzacja Danych

Po przeanalizowaniu zależności pomiędzy wartościami, następnymi krokami jest stworzenie modeli predykcyjnych oraz klasteryzacja danych doprowadzających do niewydolności serca.

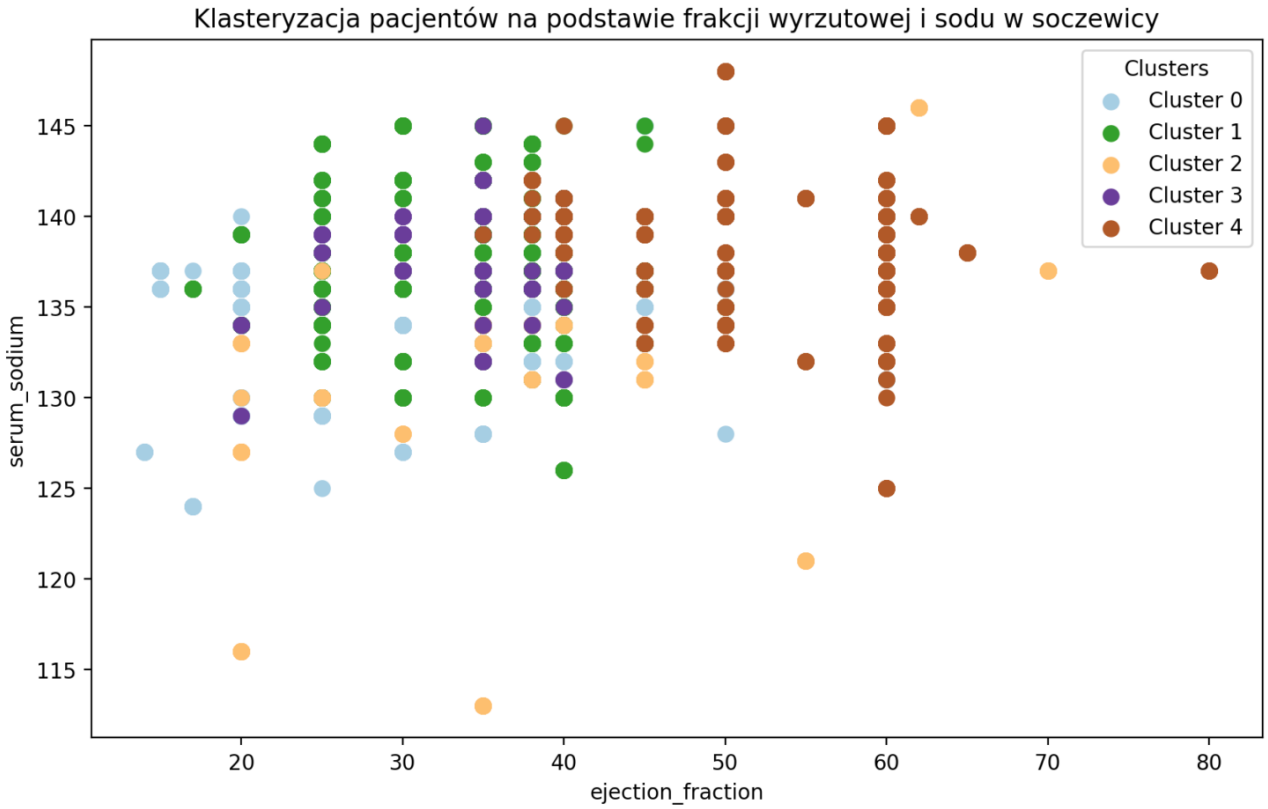
3.1. Budowa Klastrów

Do poprawnego stworzenia klastrów danych najpierw należy wyznaczyć, jak dużą ilość klastrów będzie optymalnym wyborem. W tym celu najczęściej wykorzystuje się metodę łokci polegającą na kolejnym tworzeniu klastrów z wybranego zestawu danych przy jednoczesnym zapisywaniu wyników wewnątrz-klastrowych sum kwadratów (ang. WCSS). Tak wyliczone wartości pokazujemy na wykresie WCSS/ilość klastrów, po czym wybieramy wizualnie punkt, w którym wykres zaczyna się stabilizować, czyli gdzie jest tak zwany łokieć.



Rysunek 15. Wykres dla metody łokcia

Metoda łokci pomogła nam znaleźć punkt stabilizacji wykresu, jakim jest ilość 5 klastrów, co wykorzystamy do znalezienia klastrów danych.



Rysunek 16. Wykres klasteryzacji pacjentów na podstawie frakcji wyrzutowej i sodu w soczewicy

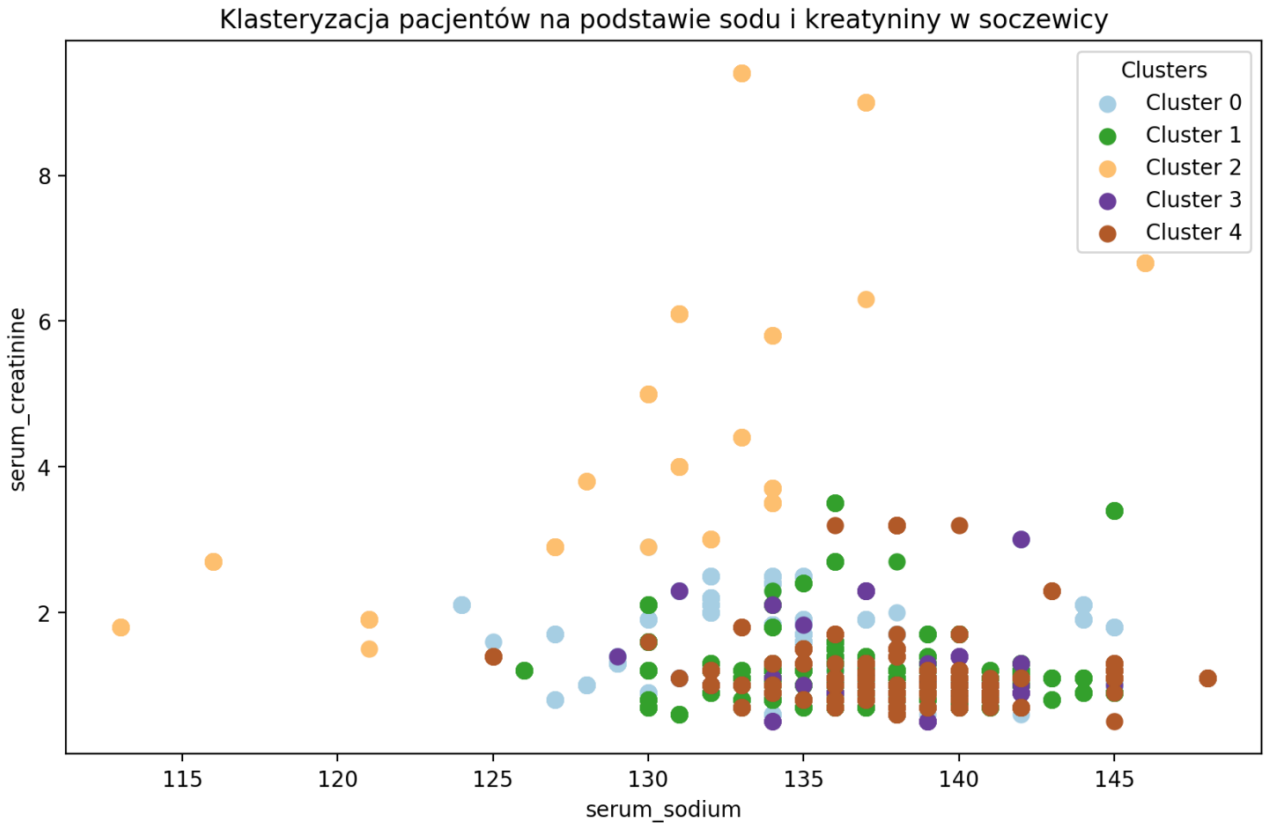
Klastr 0 (jasno-niebieski): Pacjenci o niskiej frakcji wyrzutowej i niskiej ilości sodu w soczewicy

Klastr 1 (zielony): Pacjenci o niskiej frakcji wyrzutowej i umiarkowanej ilości sodu w soczewicy

Klastr 2 (beżowy): Pacjenci o zróżnicowanej frakcji wyrzutowej i zróżnicowanej ilości sodu w soczewicy

Klastr 3 (fioletowy): Pacjenci o niskiej frakcji wyrzutowej i umiarkowanej ilości sodu w soczewicy

Klastr 4 (brązowy): Pacjenci o wysokiej frakcji wyrzutowej i wysokiej ilości sodu w soczewicy



Rysunek 17. Wykres klasteryzacji pacjentów na podstawie sodu i kreatyniny w soczewicy

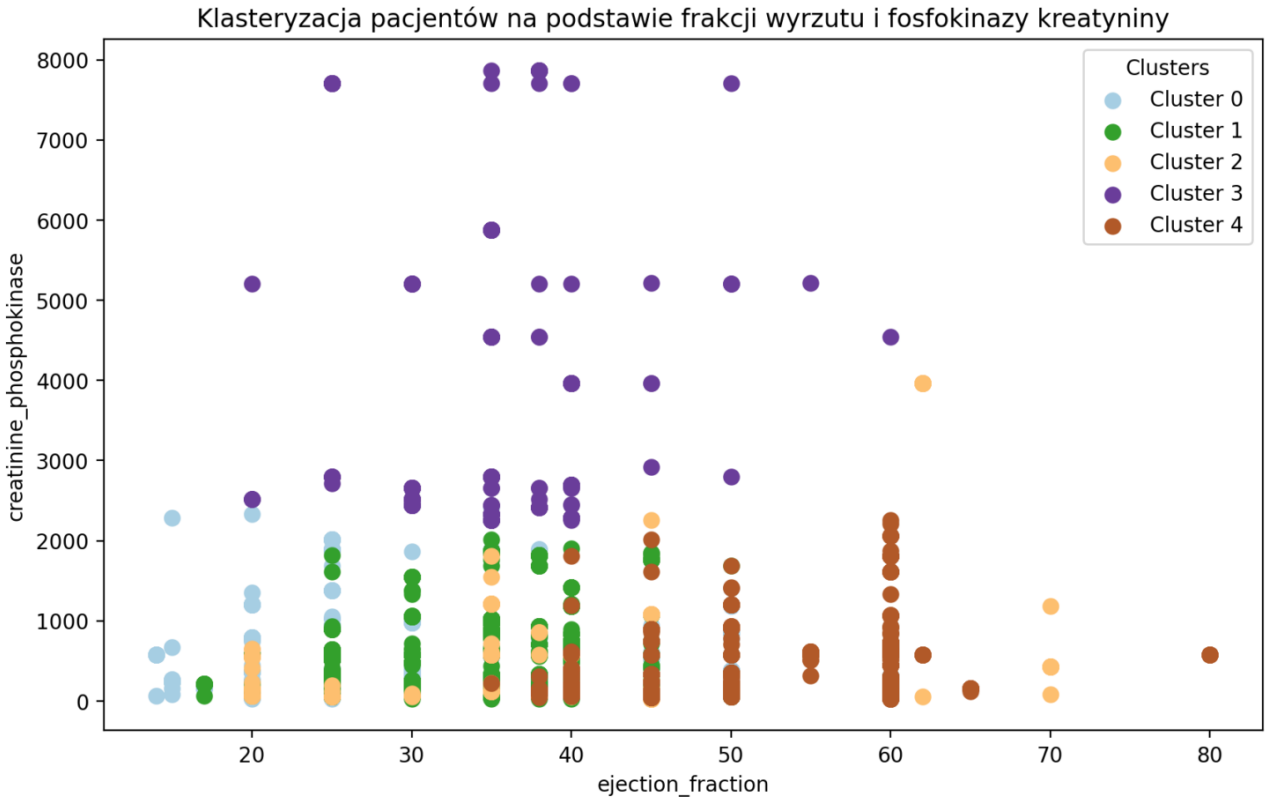
Klastr 0 (jasno-niebieski): Pacjenci o niskiej ilości sodu w soczewicy i umiarkowanej ilości kreatyniny w soczewicy

Klastr 1 (zielony): Pacjenci o zróżnicowanej ilości sodu w soczewicy i zróżnicowanej ilości kreatyniny w soczewicy

Klastr 2 (beżowy): Pacjenci o niskiej ilości sodu w soczewicy i wysokiej ilości kreatyniny w soczewicy

Klastr 3 (fioletowy): Pacjenci o umiarkowanej ilości sodu w soczewicy i umiarkowanej ilości kreatyniny w soczewicy

Klastr 4 (brązowy): Pacjenci o wysokiej ilości sodu w soczewicy i niskiej ilości kreatyniny w soczewicy



Rysunek 18. Wykres klasteryzacji pacjentów na podstawie frakcji wyrzutu i fosfokinazy kreatyniny

Klastr 0 (jasno-niebieski): Pacjenci o niskiej frakcji wyrzutu i niskiej ilości fosfokinazy kreatyniny

Klastr 1 (zielony): Pacjenci o niskiej frakcji wyrzutu i umiarkowanej ilości fosfokinazy kreatyniny

Klastr 2 (beżowy): Pacjenci o zróżnicowanej frakcji wyrzutu i zróżnicowanej ilości fosfokinazy kreatyniny

Klastr 3 (fioletowy): Pacjenci o umiarkowanej frakcji wyrzutu i wysokiej ilości fosfokinazy kreatyniny

Klastr 4 (brązowy): Pacjenci o wysokiej frakcji wyrzutu i niskiej ilości fosfokinazy kreatyniny

3.1.1. Interpretacja

Dzięki klasteryzacji danych można zauważyć korelację pomiędzy fosfokinazą kreatyniny i frakcją wyrzutu układającą się w łuk. Sód i kreatynina w soczewicy oraz frakcja wyrzutu i sód w soczewicy nie są ze sobą powiązane, co sugeruje wpływ innych czynników na te własności.

3.1.2. Kod Programu

main.py

```python
import pandas as pd
import streamlit as st
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sb
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

DATA = "./data/heart_failure_clinical_records.csv"

def loadCSV(path: str):
    df = pd.read_csv(path)
    showPlots(df)
    return df

def showPlots(df: pd.DataFrame):
    st.markdown("# Heart failure")
    st.markdown("## Datasets")

    # Og chart with no deaths
    df_no_death = df.copy(deep=True)
    df_no_death = df_no_death.drop(index=df_no_death[df_no_death["DEATH_EVENT"] == 0].index)
    df_no_death = df_no_death.drop("DEATH_EVENT", axis=1)
    df_no_death.sort_values(by='creatinine_phosphokinase")
    st.markdown("### No deaths")
    st.dataframe(df_no_death)
    st.markdown("#### Median")
    no_death_median = df_no_death.median()

    # No death median table creatinine
    st.table(no_death_median)

    # Chart with only deaths
    df_death = df.copy(deep=True)
    df_death = df_death.drop(index=df_death[df_death["DEATH_EVENT"] == 1].index)
    df_death = df_death.drop("DEATH_EVENT", axis=1)
    df_death.sort_values(by="creatinine_phosphokinase")
    st.markdown("### Only deaths")
    st.dataframe(df_death)
    st.markdown("#### Median")
    death_median = df_death.median()

    # Death median table creatinine
    st.table(death_median)

    st.markdown("### Corelation table")
    corr = df.corr()
    st.dataframe(corr)

    # Corelation heatmap
    sb.heatmap(corr, annot=True, fmt=".2f", annot_kws={"size": 8},  cmap='coolwarm')
    st.pyplot(plt)
    plt.close()

    # Pairplot
    relevant_columns = ['ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT']
    data_relevant = df[relevant_columns]
    plt.figure(figsize=(10, 6))
    sb.pairplot(data_relevant, hue="DEATH_EVENT")
    st.pyplot(plt)
    plt.close()

    # Histograms of each column
    for column in df.columns:
        if column != 'DEATH_EVENT':
            plt.figure(figsize=(10, 4))
            sb.histplot(df[column], kde=True, bins=30)
            plt.title(f'Distribution of {column}')
            st.pyplot(plt)
            plt.close()

    # Clustering data
    ## Elbow method
    wcss = []
    # data_scaled = StandardScaler().fit_transform(data_relevant.drop("DEATH_EVENT", axis=1))
    data_scaled = StandardScaler().fit_transform(data_relevant)
    for k in range(1, 21):
        kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(data_scaled)

        wcss.append(kmeanModel.inertia_)

    st.markdown("# Clustering")
    st.markdown("## Metoda łokci")
    plt.figure(figsize=(10, 6))

    # Najlepszy wynik daje 5 klastrow
    plt.plot(range(1, 21), wcss, marker='o', linestyle='-')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.title('Metoda łokci')
    st.pyplot(plt)
    plt.close()

    st.markdown("## Scatters")
    kmeanModel = KMeans(n_clusters=5)
    cdf = df.copy(deep=True)
    cdf['cluster'] = kmeanModel.fit_predict(data_scaled)
    st.write("Cluster mean:")
    st.table(cdf.groupby("cluster", sort=True).mean())

    """
    above aberage serum_creatinine
    below average serum_sodium
    below average time
    below average platelets
    """

    plt.figure(figsize=(10, 6))
    colors = mpl.colormaps['Paired'].resampled(5)
    for cluster_id in range(5):
        cluster_data = df[cdf['cluster'] == cluster_id]
        plt.scatter(cluster_data['ejection_fraction'], cluster_data['serum_sodium'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)

    # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'

    plt.xlabel(relevant_columns[0])
    plt.ylabel(relevant_columns[1])
    plt.title('Klasteryzacja pacjentów na podstawie frakcji wyrzutowej i sodu w soczewicy')
    plt.legend(title="Clusters")
    st.pyplot(plt)
    plt.close()


    plt.figure(figsize=(10, 6))
    colors = mpl.colormaps['Paired'].resampled(5)
    for cluster_id in range(5):
        cluster_data = df[cdf['cluster'] == cluster_id]
        plt.scatter(cluster_data['serum_sodium'], cluster_data['serum_creatinine'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)

    # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'

    plt.xlabel(relevant_columns[1])
    plt.ylabel(relevant_columns[2])
    plt.title('Klasteryzacja pacjentów na podstawie sodu i kreatyniny w soczewicy')
    plt.legend(title="Clusters")
    st.pyplot(plt)
    plt.close()

    plt.figure(figsize=(10, 6))
    colors = mpl.colormaps['Paired'].resampled(5)
    for cluster_id in range(5):
        cluster_data = df[cdf['cluster'] == cluster_id]
        plt.scatter(cluster_data['ejection_fraction'], cluster_data['creatinine_phosphokinase'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)

    # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'

    plt.xlabel(relevant_columns[0])
    plt.ylabel(relevant_columns[4])
    plt.title('Klasteryzacja pacjentów na podstawie frakcji wyrzutu i fosfokinazy kreatyniny')
    plt.legend(title="Clusters")
    st.pyplot(plt)
    plt.close()

if __name__ == "__main__":
    df = loadCSV(DATA)
```

Rysunek 19. Kod programu tworzący klastry danych

3.2. Budowanie Modeli Predykcyjnych

Pierwszym krokiem do stworzenia modeli predykcyjnych jest podzielenie danych na dane przeznaczone do trenowania oraz walidacji. Ze względu na brak zmiennych kategorycznych, zestaw danych wymaga jedynie przeskalowania go za pomocą skalera min-max. Skaler min-max liniowo

przemienia wartości tak, aby wartość minimalna i maksymalna odpowiadały podanemu zakresowi, utrzymując procentowe odległości pomiędzy wartościami. Modele predykcyjne wybrane zostały na podstawie mapy wyboru na stronie scikit-learn https://scikit-learn.org/stable/tutorial /machine_learning_map/, co pozwoliło mi uplasować problem predykcji niewydolności serca jako problem klasyfikacyjny.

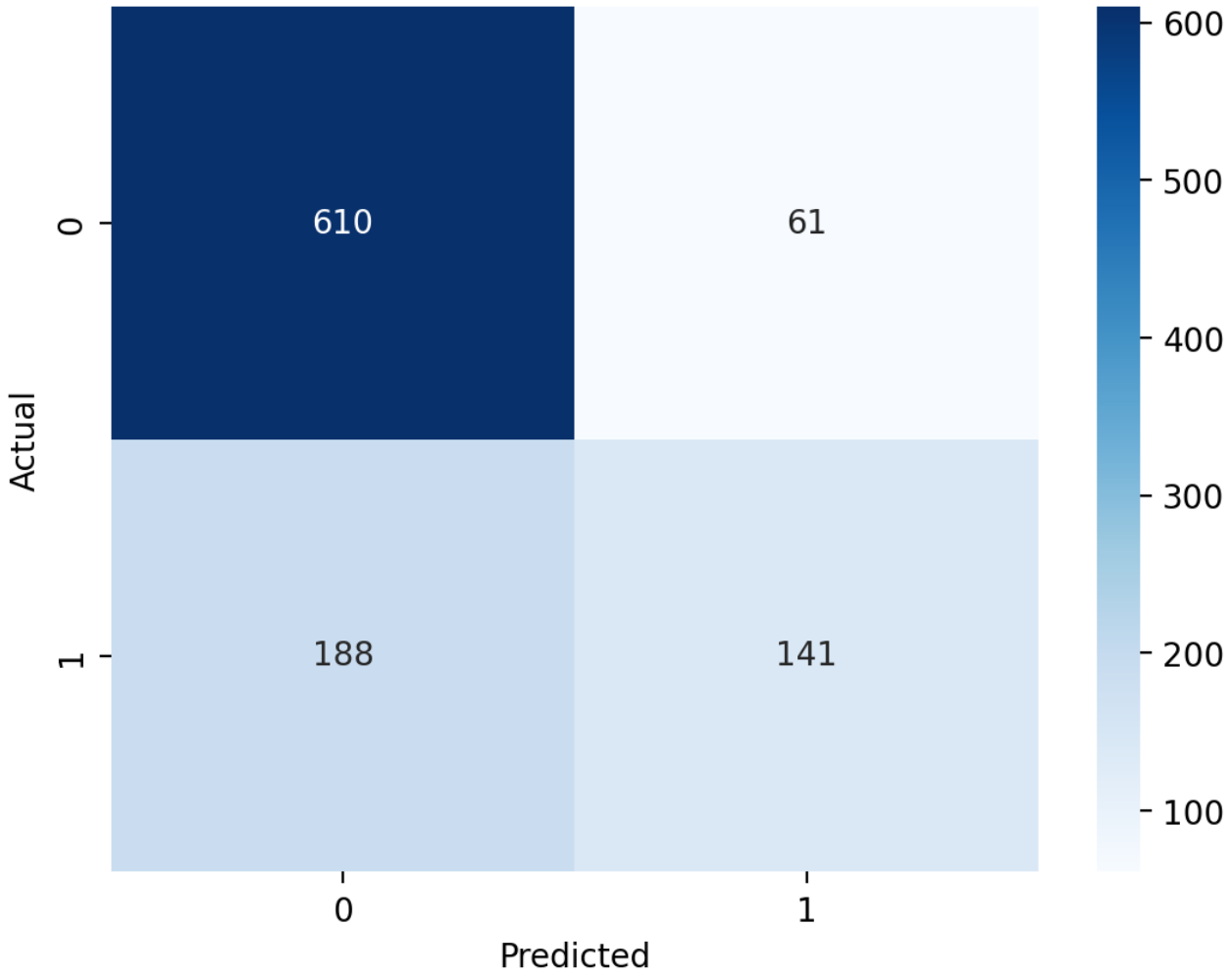
3.2.1. Cel

Moim celem w stworzeniu modeli predykcyjnych jest ocena na podstawie danych, czy u danej osoby może w przyszłości występować niewydolność serca, co pozwoli na dalszą diagnozę i ewentualne przeciwdziałanie.

3.2.2. Regresja Liniowa

Jednym z pierwszych wybranych przeze mnie modeli jest model regresji liniowej, która jest jednym z najmniej skomplikowanych modeli. W moim badaniu regresja była czwartym co do precyzji modelem, szczycąc się poniższymi wynikami:

- średniej ważonej precyzji 0.753,
- średnia czułość ważona 0.766,
- średnia ważona miary F1 0.748.



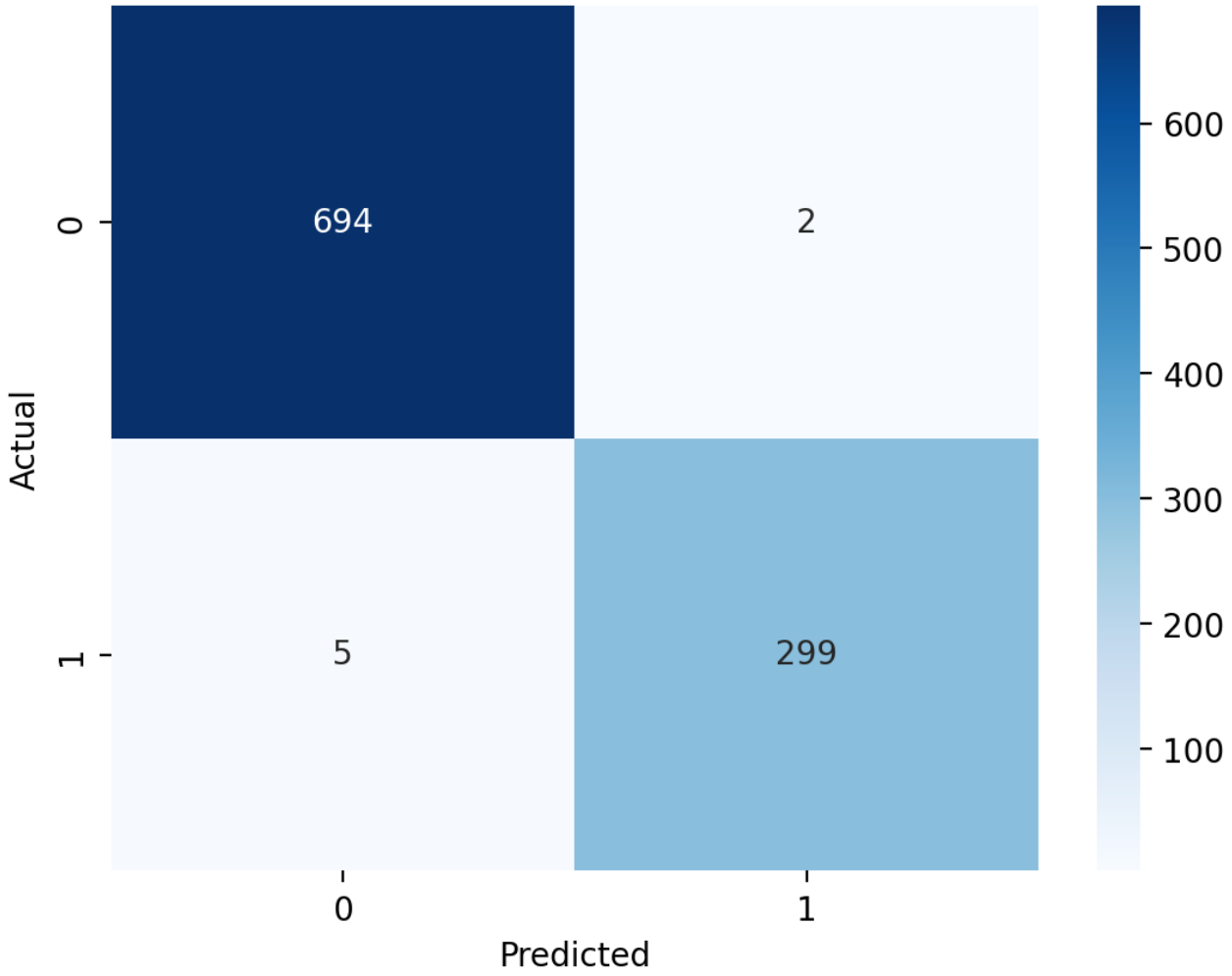
Rysunek 20. Macierz pomyłek regresji liniowej

Powyższy model uzyskał dość dobre wyniki pomimo swojej prostotę i byłby wystarczająco dobrym wyborem, jeśli zależy nam na małym wykorzystaniu zasobów.

3.2.3. Losowy Las Decyzyjny

Kolejnym modelem jest losowy las decyzyjny, który jest znacznie bardziej złożonym modelem, co pozytywnie przekłada się na wyniki precyzji w testach. Losowy Las poradził sobie najlepiej, a jego statystyki wynoszą:

- średnia ważona precyzji 0.9851,
- średnia ważona czułości 0.985,
- średnia ważona miary F1 0.9849.



Rysunek 21. Macierz pomyłek losowego lasu decyzyjnego

Model lasu najlepiej oszacował szansę wystąpienia niewydolności serca, jednak obarczone jest to kosztem większych zasobów potrzebnych do kalkulacji.

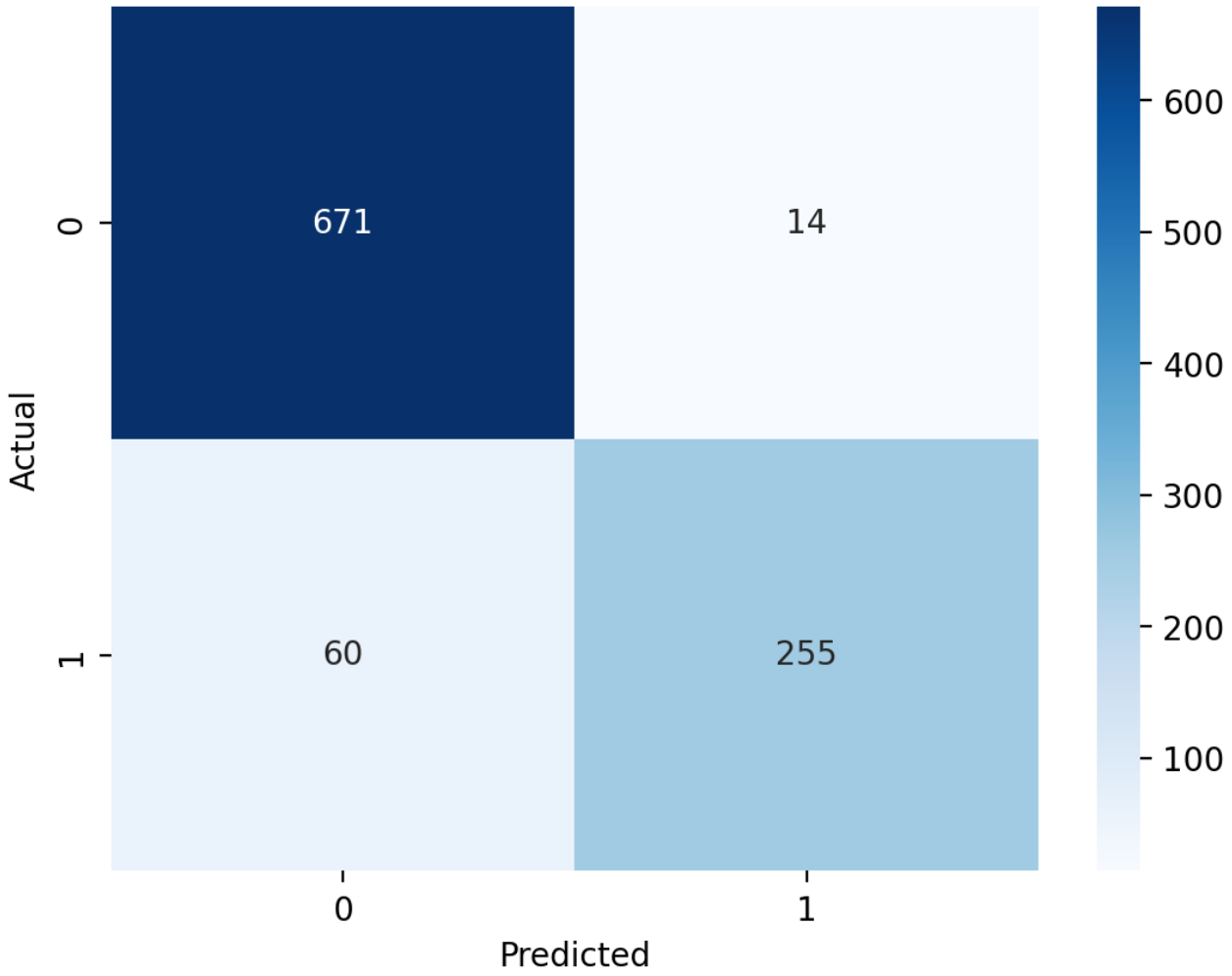
3.2.4. Wzmocnienie Gradientowe

Następnym modelem jest wzmocnienie gradientowe, który poradził sobie tylko gorzej niż losowy las decyzyjny z wynikami:

- średnia ważona precyzji 0.9395.

- średnia ważona czułości 0.937.

- średnia ważona miary F1 0.9355.



Rysunek 22. Macierz pomyłek wzmocnienia gradientowego

Wzmocnienie gradientowe pomimo bardziej złożonej architektury osiąga zbliżone wyniki do lasu decyzyjnego, co nie jest równomierne do większego zapotrzebowania na zasoby.

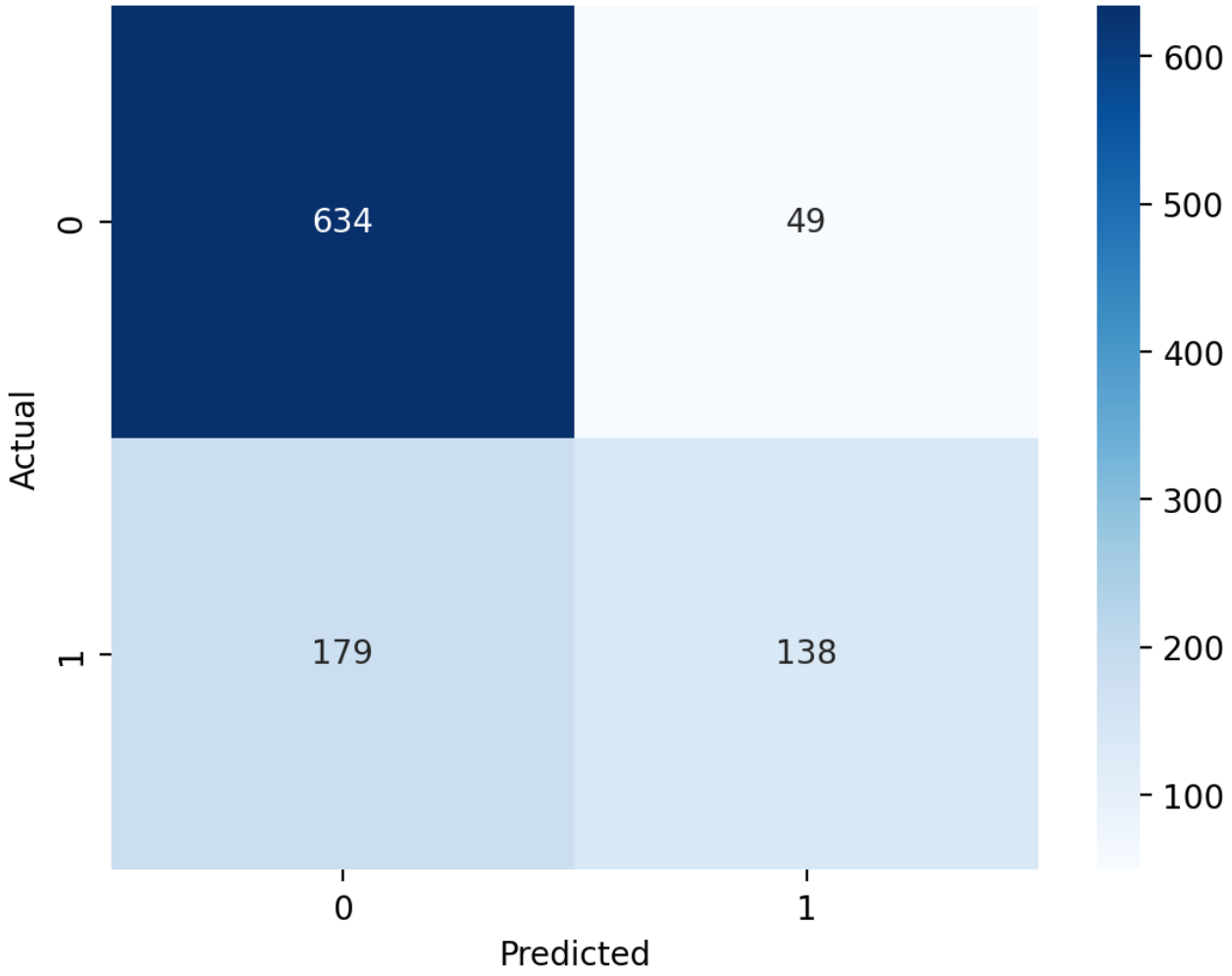
3.3. Liniowy Wektor Nośny (SVM)

Kolejnym przetestowanym modelem jest wektor nośny (SVM), okazał się podobnie precyzyjny jak regresja liniowa, ale gorszy od wzmocnienia gradientowego z wynikami:

- średnia ważona precyzji 0.7476,

- średnia ważona czułości 0.754,

- średnia ważona miary F1 0.734.



Rysunek 23. Macierz pomyłek liniowego wektora nośnego

Liniowy wektor nośny pomimo większej złożoności nie przejawia lepszych wyników od prostszego modelu regresji liniowej, co oznacza, że nie jest on dobrym wyborem.

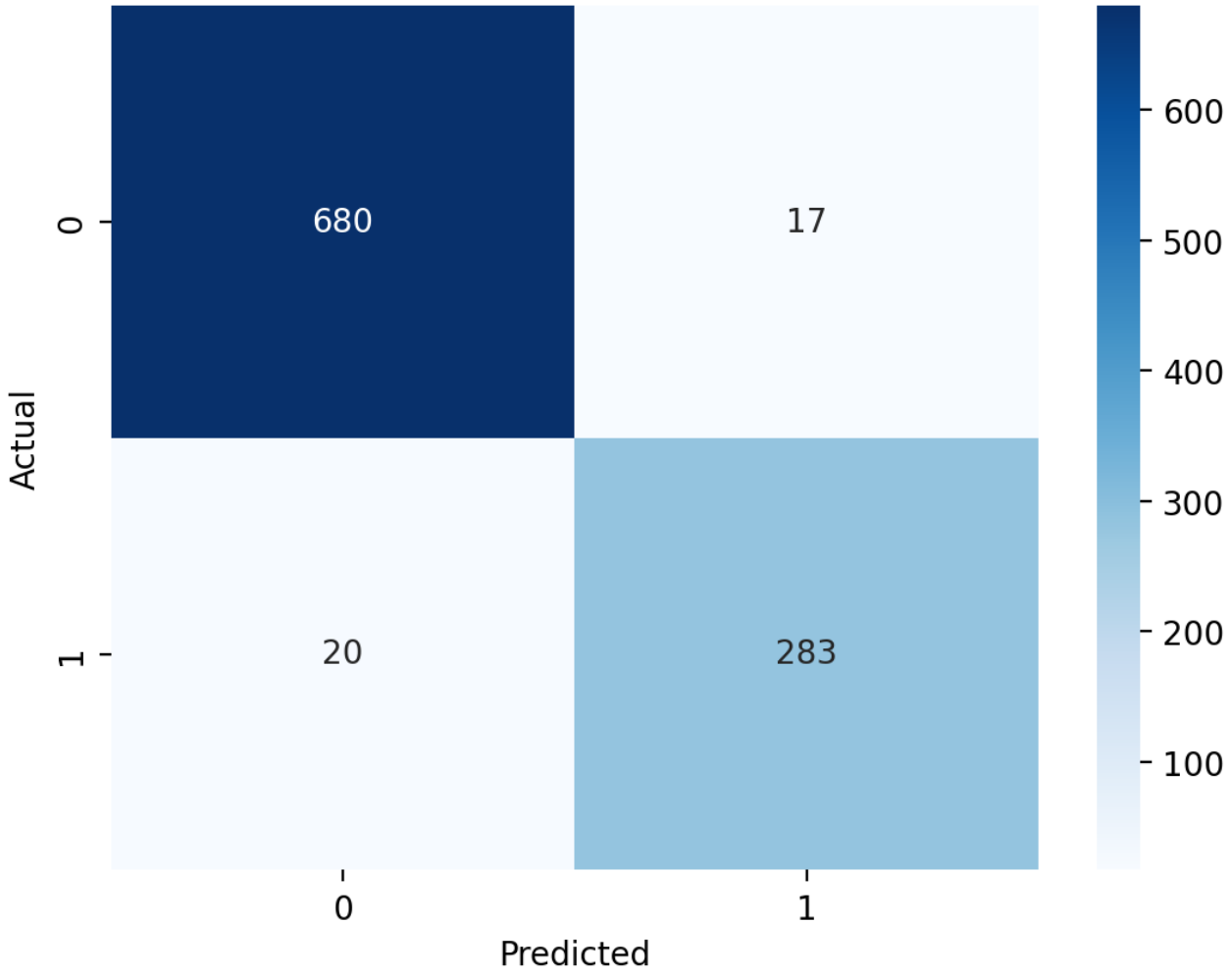
3.3.1. K — Najbliżsi Sąsiedzi (KNN)

Ostatnim modelem, jaki został przetestowany jest k-najbliżsi sąsiedzi, zajął on drugie miejsce pod względem precyzji, a jego wyniki plasują się następująco:

- średnia ważona precyzji 0.9569,

- średnia ważona czułości 0.957,

- średnia ważona miary F1 0.9569.



Rysunek 24. Macierz pomyłek K-Najbliższych Sąsiadów

Najbliżsi sąsiedzi jest drugim najlepszym modelem predykcyjnym. Posiada podobnie skomplikowaną architekturę jak model predykcyjnym losowego lasu decyzyjnego, jednak ze względu na większe zapotrzebowanie zasobów i porównywalne wyniki precyzji, jest mniej optymalnym wyborem niż losowy las decyzyjny

3.3.2. Wnioski

Najlepszym modelem do predykcji ocenienia, czy osoba może posiadać ryzyko niewydolność serca, okazuje się Losowy Las Decyzyjny. W każdej mierze był lepszym od innych modeli, radząc sobie w ponad 90% przypadków, co czyni go optymalnym wyborem do klasyfikacji ryzyka pacjenta na śmierć w wyniku niewydolności serca.

3.3.3. Kod Programu

main.py

```python
import pandas as pd
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

DATA = "./data/heart_failure_clinical_records.csv"

def loadCSV(path: str):
    df = pd.read_csv(path)
    return df

def create_model(df, model, name):
    x = df.drop("DEATH_EVENT", axis=1).drop("time", axis=1)
    y = df["DEATH_EVENT"]

    # Data scaling from 0 to 1
    min_max_scaler = MinMaxScaler()

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

    x_train = min_max_scaler.fit_transform(x_train)
    # x_train = standard_scaler.fit_transform(x_train)
    x_test = min_max_scaler.fit_transform(x_test)
    # x_test = standard_scaler.fit_transform(x_test)

    model.fit(x_train, y_train)

    # Predict
    y_pred = model.predict(x_test)
    st.markdown(f'# {name}')
    showMetrics(y_test, y_pred)

def showMetrics(y_test, y_pred):
    # Metrics
    ## Accuracy
    st.write("Accuracy:", accuracy_score(y_test, y_pred))

    ## Confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)
    st.header('Confusion Matrix')
    fig, ax = plt.subplots()
    sb.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', ax=ax)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    st.pyplot(fig)

    ## Classification report
    class_report = classification_report(y_test, y_pred, output_dict=True)
    st.header('Classification Report')
    class_report_df = pd.DataFrame(class_report).transpose()
    st.dataframe(class_report_df)

if __name__ == "__main__":
    df = loadCSV(DATA)

    # Models
    logistic_reg = LogisticRegression()
    create_model(df.copy(deep=True), logistic_reg, "Logistic Regression")

    random_forest = RandomForestClassifier()
    create_model(df.copy(deep=True), random_forest, "Random Forest Classifier")

    gradient = GradientBoostingClassifier()
    create_model(df.copy(deep=True), gradient, "Gradient Boosting")

    svc = LinearSVC(dual='auto')
    create_model(df.copy(deep=True), svc, "Linear Support Vector Machines (SVC)")

    k_neighbors = KNeighborsClassifier()
    create_model(df.copy(deep=True), k_neighbors, "K-Nearest Neighbors (KNN)")
```

Rysunek 25. Kod programu tworzący modele predykcyjne

4. Podsumowanie

Cel projektu, czyli analiza właściwości oraz struktury danych odnośnie do pacjentów z niewydolnością serca, został wypełniony. Udało mi się zidentyfikować najważniejsze cechy mające wpływ na to zjawisko – kreatynina oraz sód w soczewicy i frakcja wyrzutu. Stworzyłem histogramy pokazujące rozkład wartości względem ilości pacjentów, jak i mapę cieplną pokazującą zależności oraz ich siłę względem śmierci pacjentów, na podstawie której udało mi się odnaleźć najważniejsze cechy.

Zobrazowałem klastry danych dla wybranych własności, co pokazało mi korelację pomiędzy fosfokinazą kreatyniny a frakcją wyrzutową. Ilość klastrów dobrałem dzięki użyciu metody łokci.

Wyćwiczyłem modele predykcyjne takie jak losowy las decyzyjny, wzmocnie gradient, k-najbliższsi sąsiedzi, liniowy wektor nośny, regresja liniowa. Przeskalowałem dane przy pomocy skalatora min-max. Następnie podzieliłem dane na zestaw uczący i walidacyjny w proporcjach 80%/20%, czyli 4000/1000 próbek danych. Po ocenie działania modelów poprzez wykorzystanie precyzji, czułości oraz miary F1, losowy las okazał się najlepszym modelem predykcyjnym do oceny czy pacjent umrze na niewydolność serca.

4.1. Wnioski

4.1.1. Korelacje

Analiza wykazała znaczącą korelację pomiędzy śmiercią pacjenta a ilością kreatyniny oraz w soczewicy. Udało się również wykazać negatywną korelację sodu w soczewicy i frakcji wyrzutu.

4.1.2. Klasteryzacja

Klastry danych wykazały zależność między frakcją wyrzutową a fosfokinazą kreatyniny.

4.1.3. Modele Predykcyjne

Modele losowego lasu decyzyjnego oraz KNN mogą być używane do przewidywania śmierci pacjentów na podstawie podanych cech. Modele wykazują bardzo dobrą precyzję.