



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

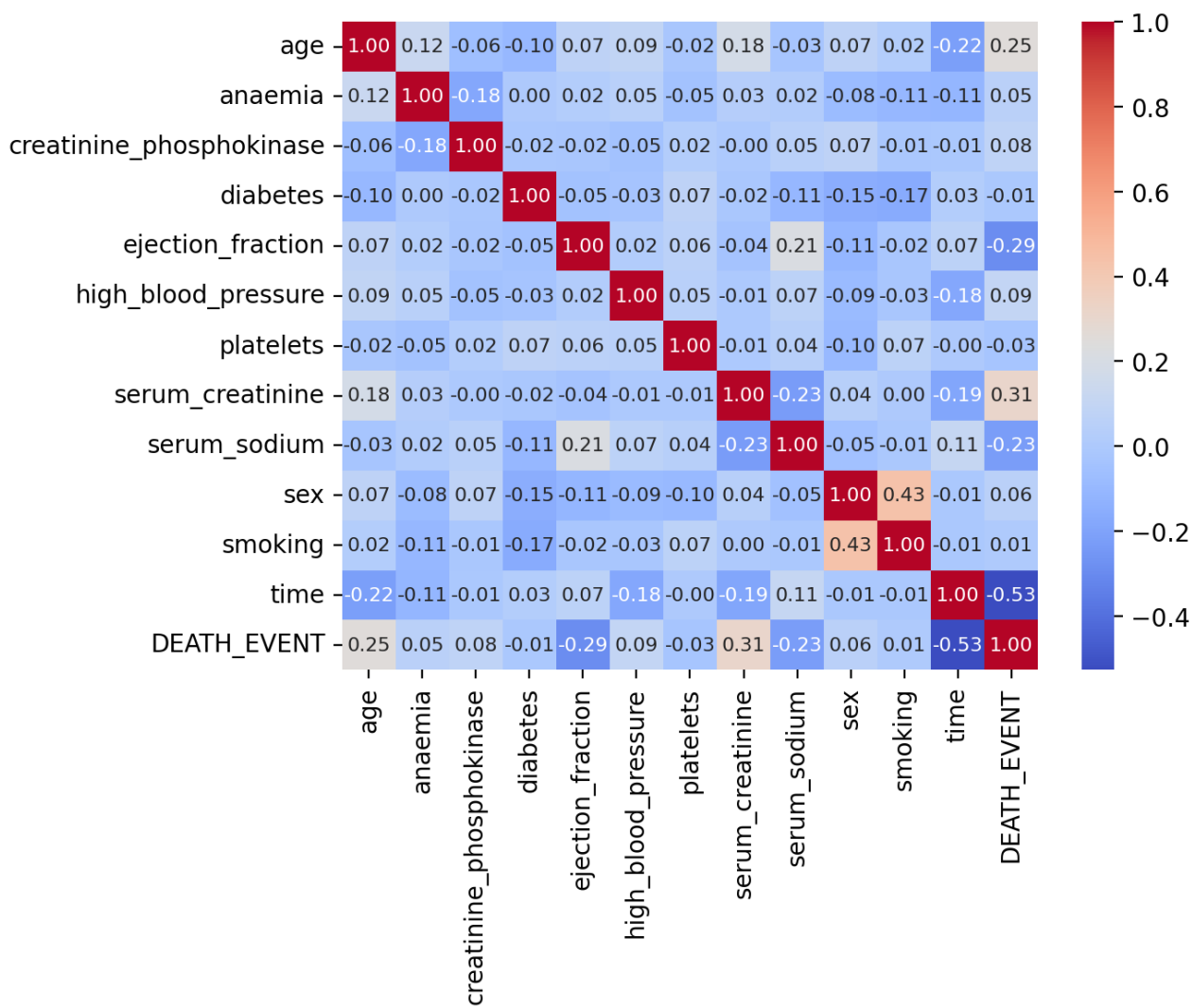
Pracę nad projektem rozpocząłem od wczytania danych z pliku CSV przy użyciu biblioteki Pandas. Następnie wyświetliłem pierwsze 10 wierszy danych, aby zobaczyć przybliżoną strukturę oraz zakres wartości. Utworzyłem tabele z wartościami średnimi, mediany, odchylenia standardowego, wartości minimalnej oraz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

2.2. Analiza Korelacji

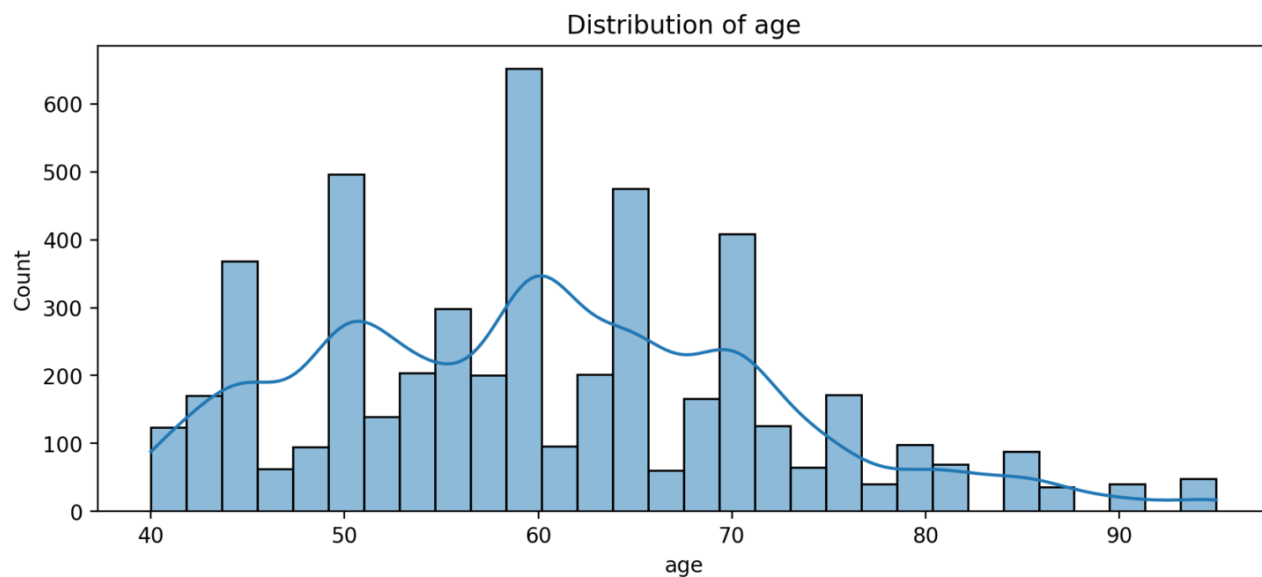
Do analizy korelacji wykorzystałem macierz korelacji, pokazującą jak cechy zmieniają się względem siebie. Wykorzystałem również mapę termiczną (Rysunek 1), która pokazuje w kolorze, jak cechy zmieniają się względem siebie.

2.3. Wizualizacja Rozkładu Własności

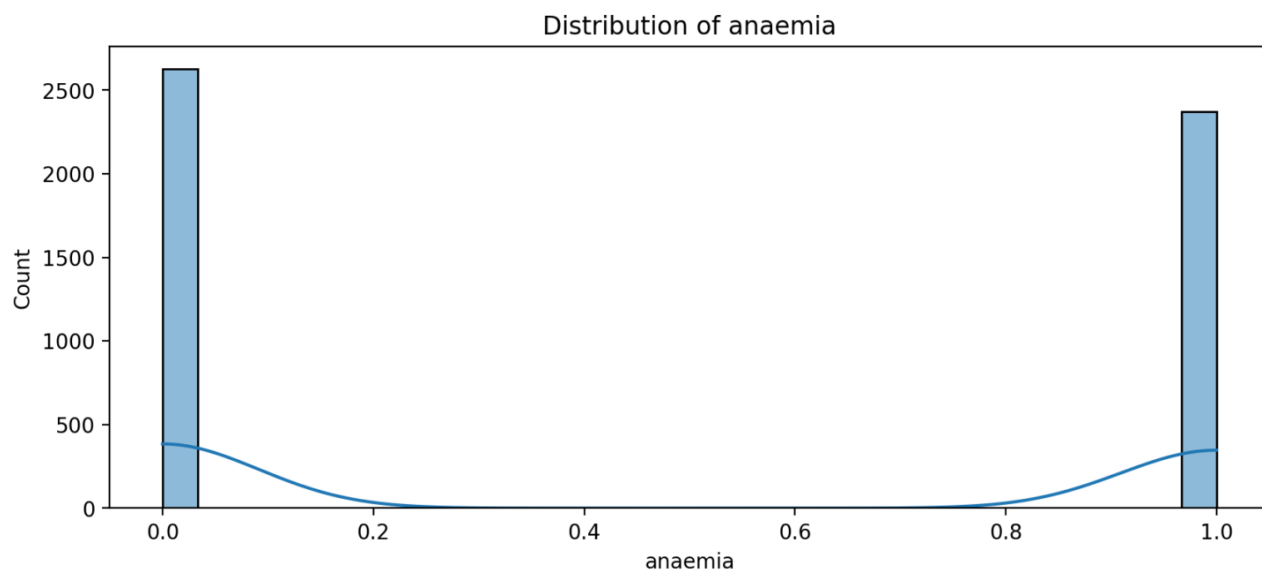
Stworzyłem histogramy odpowiadające każdej z cech zestawu danych, takich jak wiek (Rysunek 2), anemia (Rysunek 3), fosfokinaza kreatyniny (miligram/litr; Rysunek 4), cukrzyca (Rysunek 5), frakcja wyrzutowa — procent krwi opuszczającej serce przy każdym skurczu, nadciśnienie (Rysunek 6), nadciśnienie (Rysunek 7), płytki krwi (kiloplatek/mililitr; Rysunek 8), kreatynina w surowicy (miligram/decylitr; Rysunek 9), sód w surowicy (miliiekwiwalenty/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.



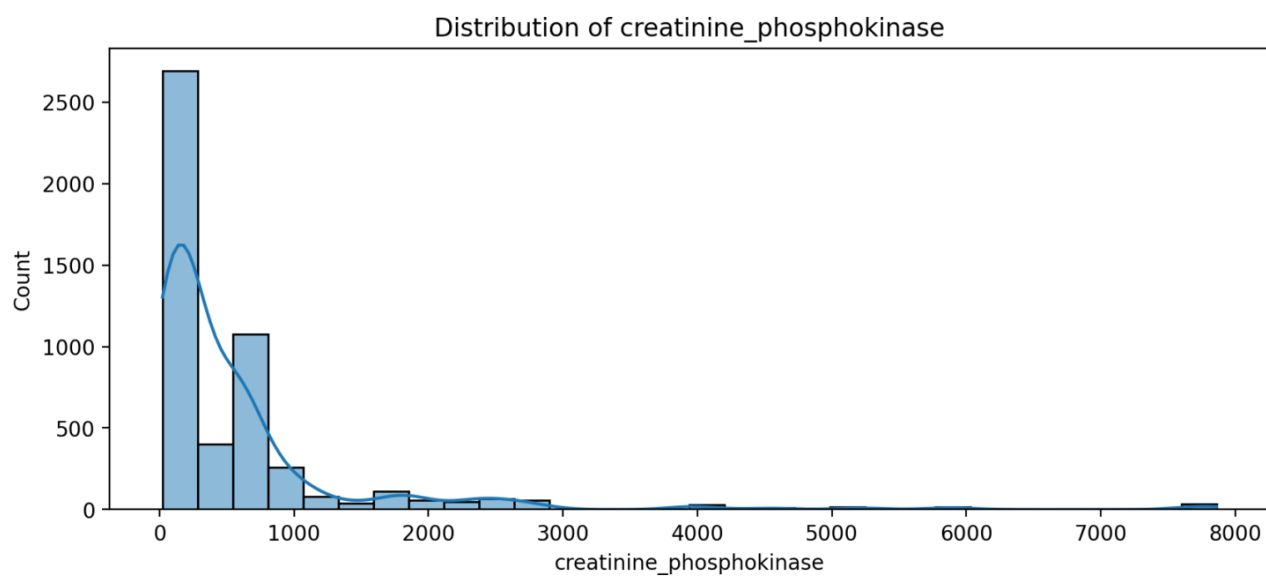
Rysunek 1. Mapa ciepła



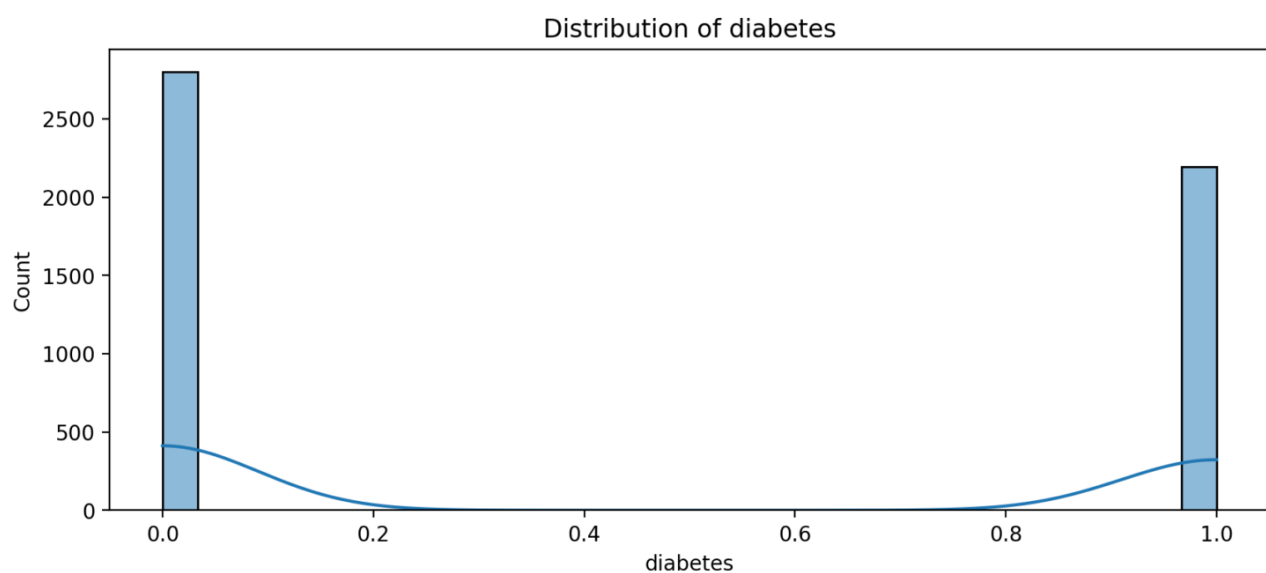
Rysunek 2. Rozkład wieku (age)



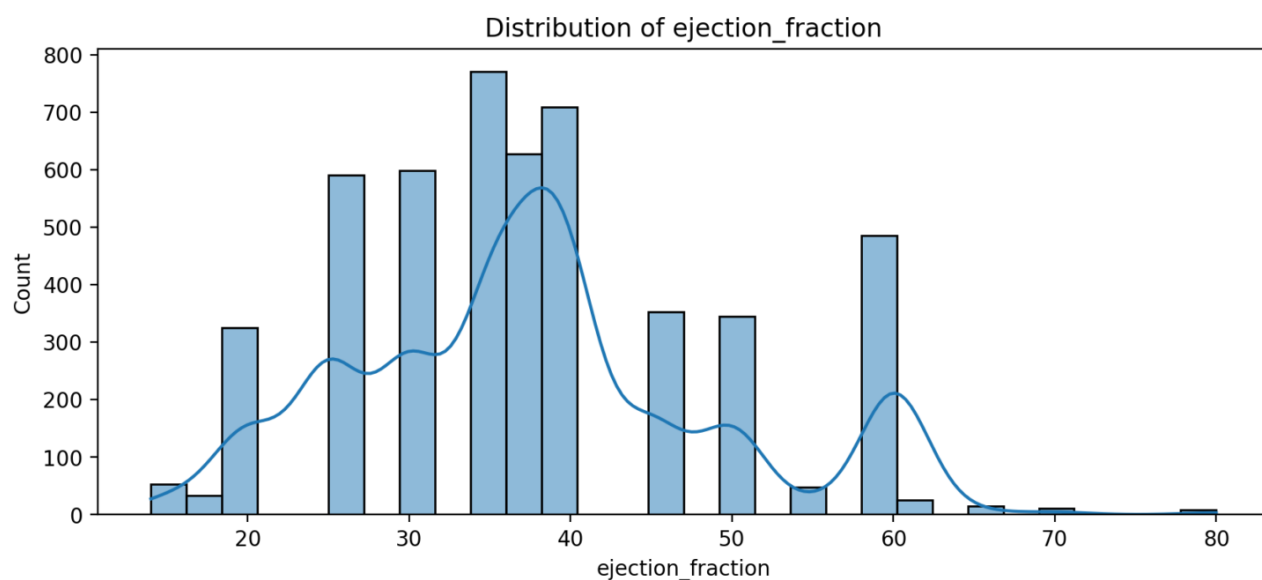
Rysunek 3. Rozkład anemii (anaemia)



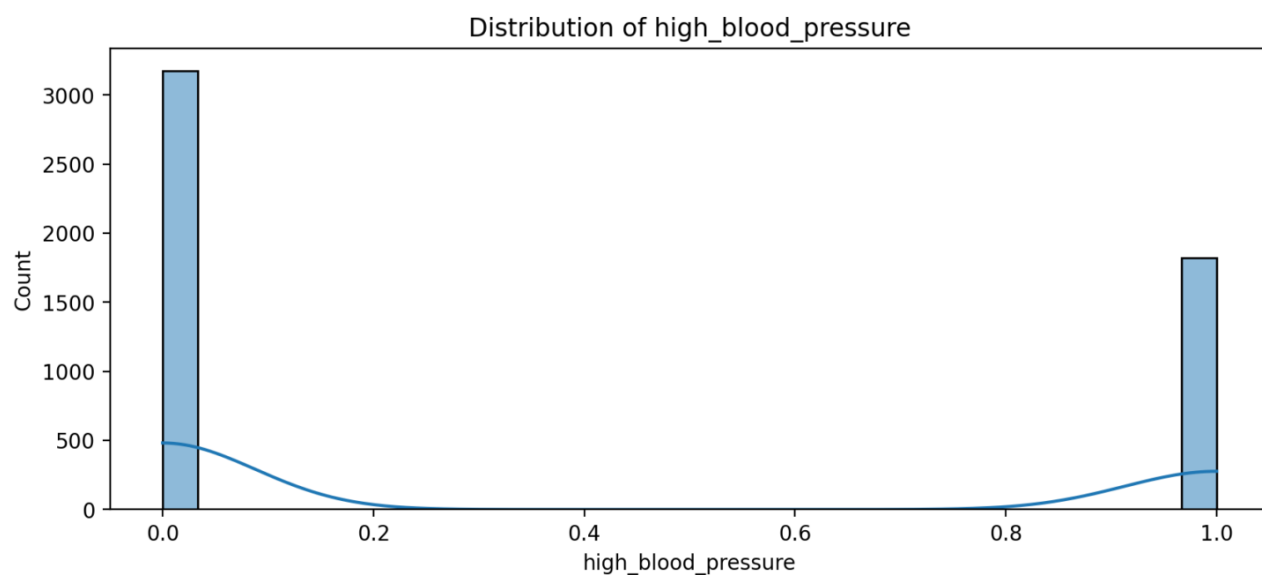
Rysunek 4. Rozkład fosfokinaza kreatyniny w mg/l (creatinine_phosphokinase)



Rysunek 5. Rozkład cukrzycy (diabetes)

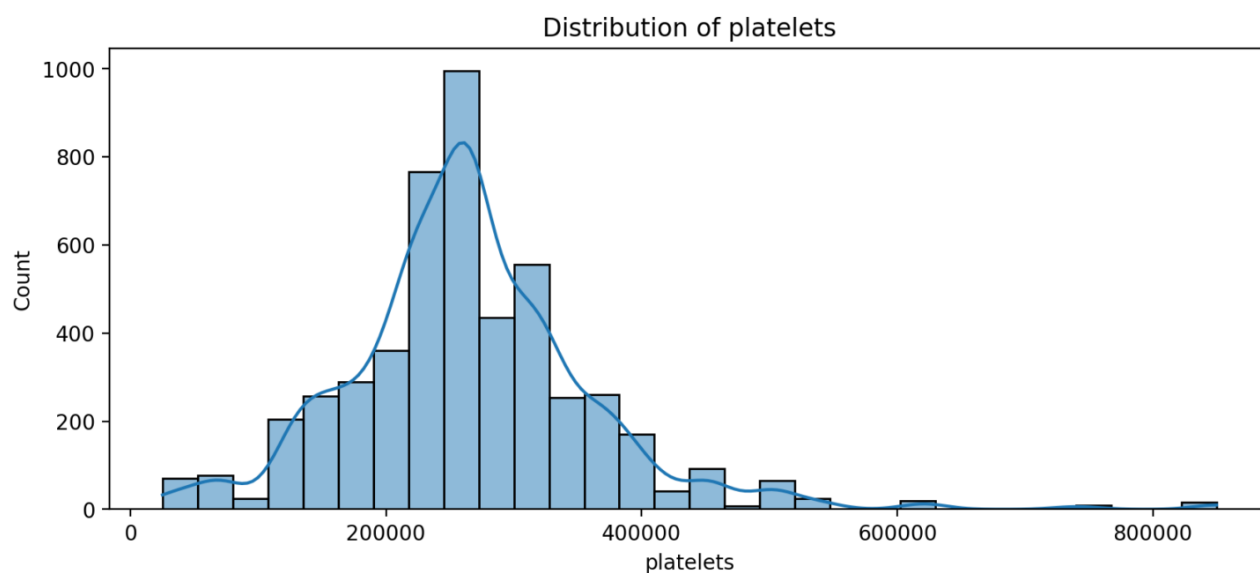


Rysunek 6. Rozkład frakcji wyrzutu w procentach (ejection_fraction)

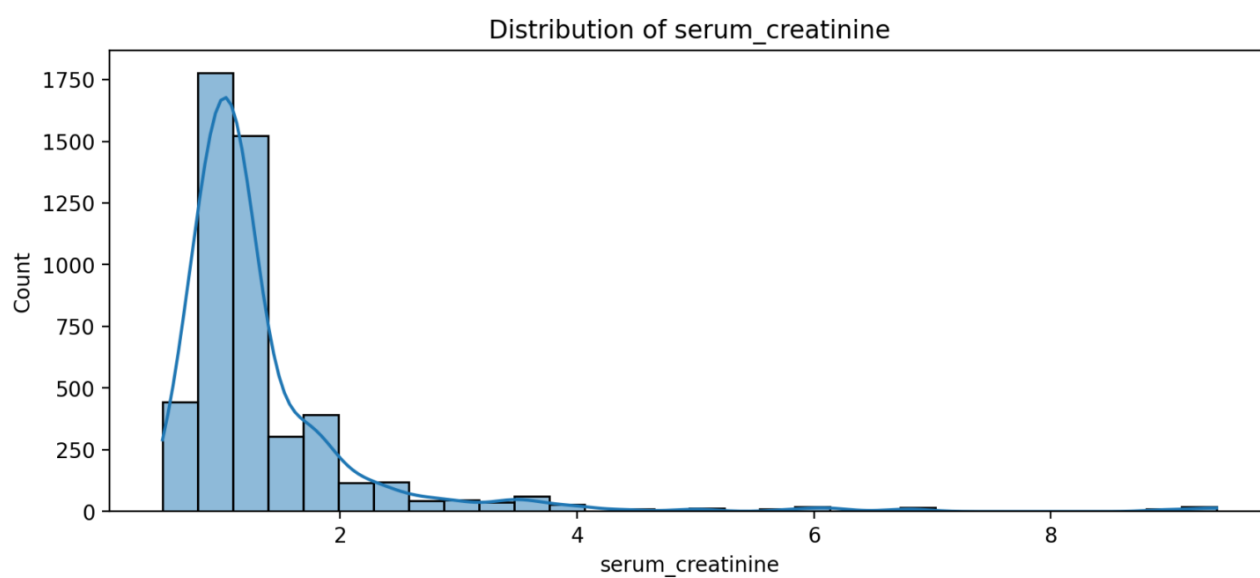


Rysunek 7. Rozkład nadciśnienia (high_blood_pressure)

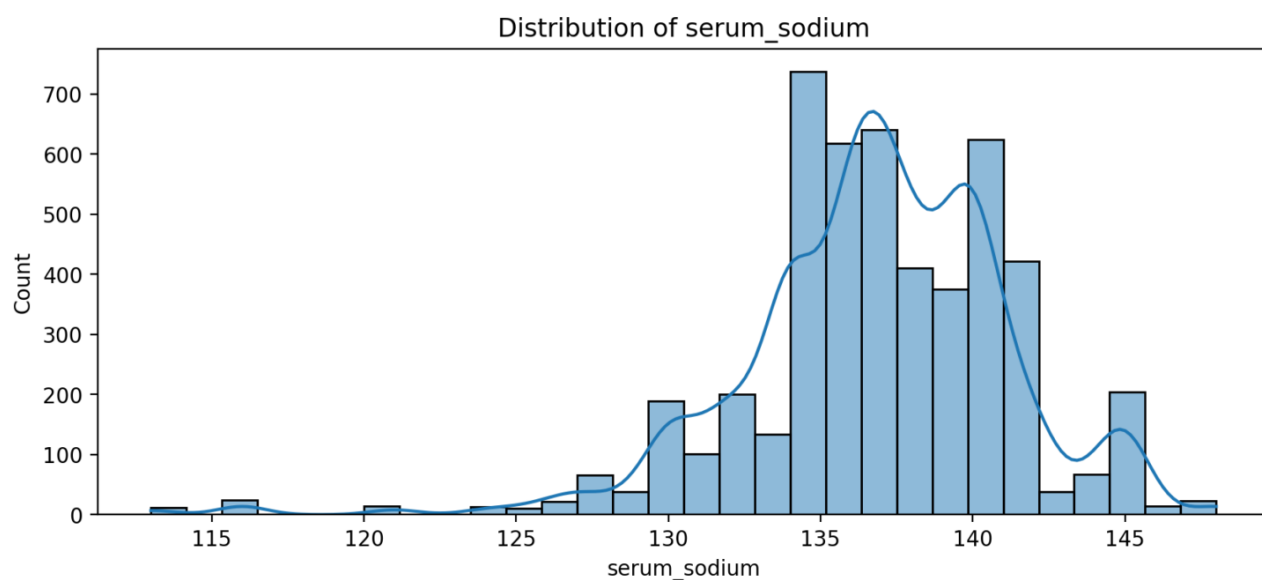
0 – brak nadciśnienia 1 – nadciśnienie



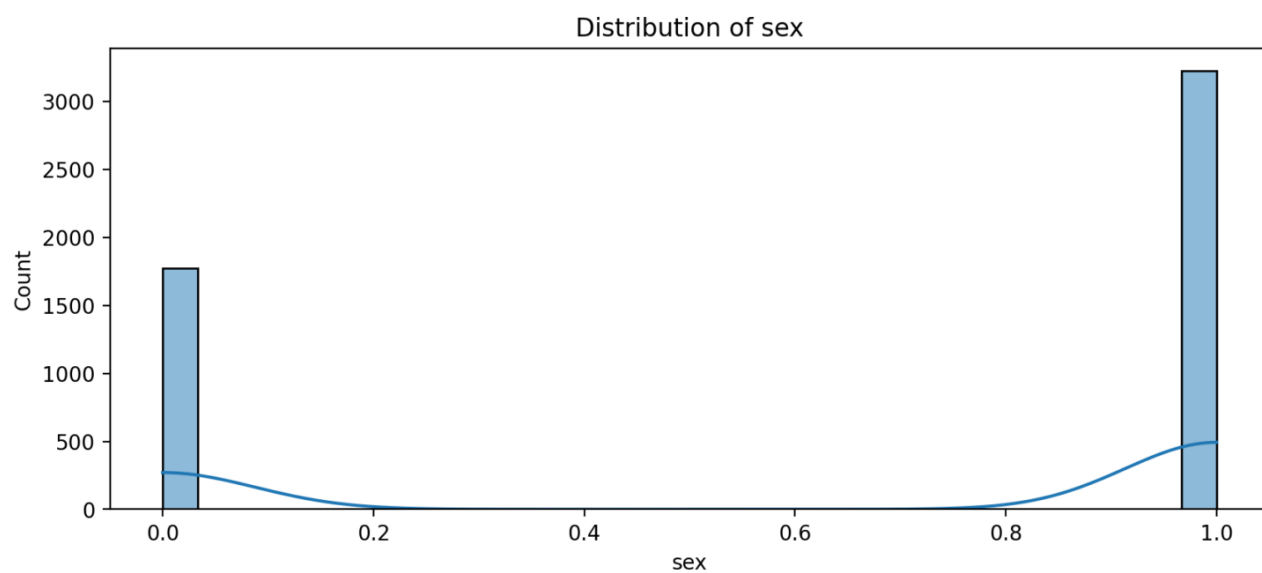
Rysunek 8. Rozkład płytek krwi w kilopłytek/ml (platelets)



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

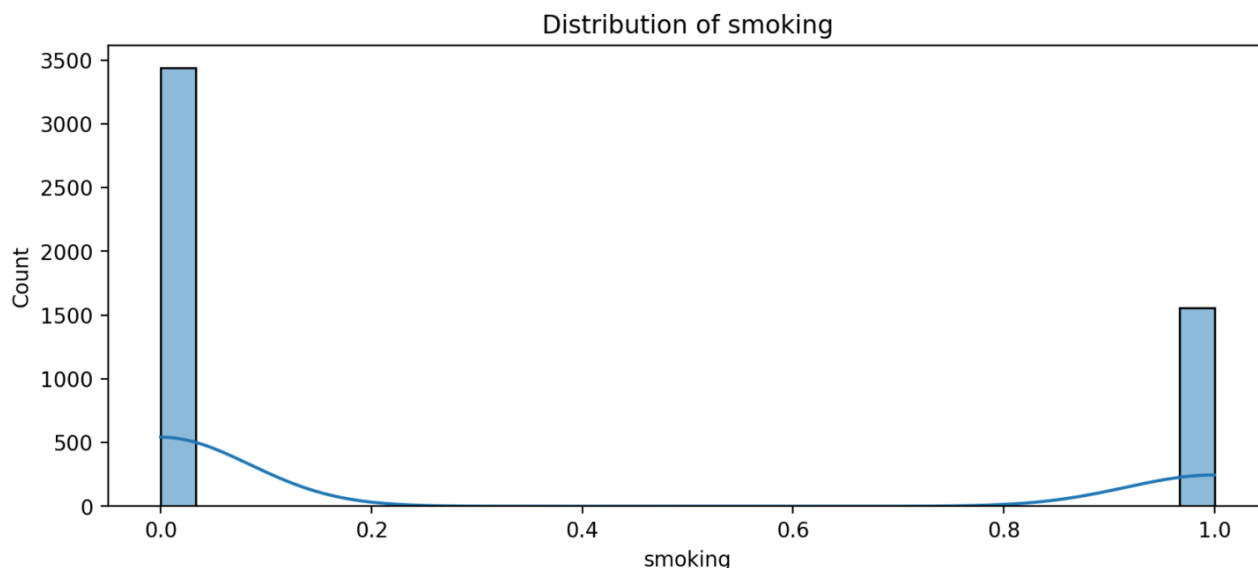


Rysunek 10. Rozkład sodu w soczewicy mEq/l (serum_sodium)



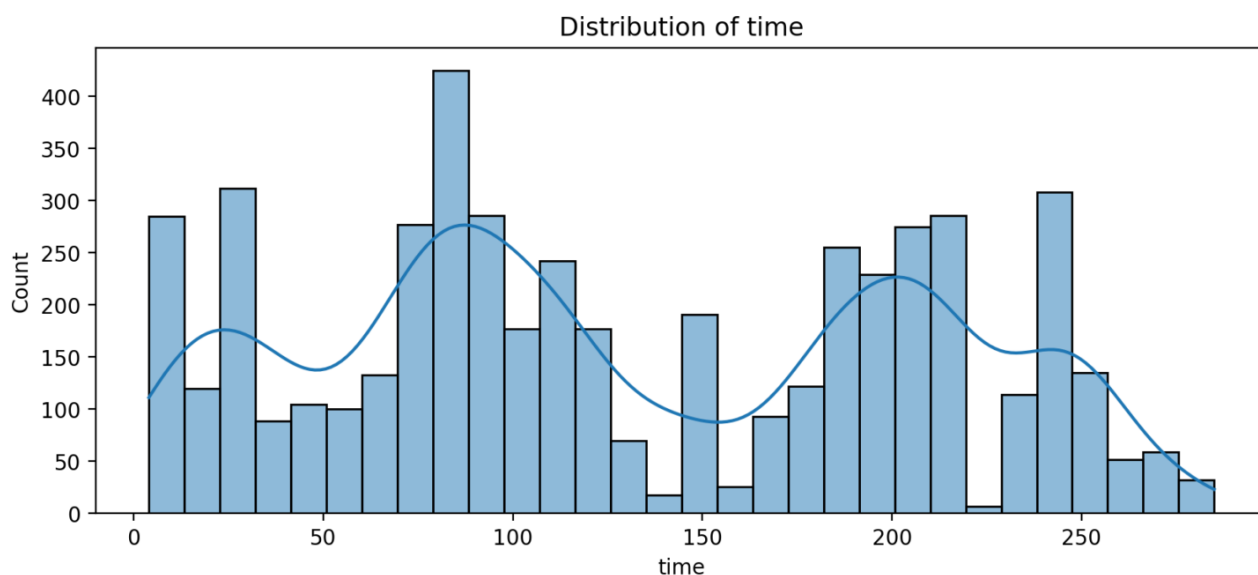
Rysunek 11. Rozkład płci (sex)

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) występuje 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
1 import pandas as pd
2 import streamlit as st
3 import matplotlib.pyplot as plt
4 import seaborn as sb
5
6 DATA = "../data/heart_failure_clinical_records.csv"
7
8 def loadCSV(path: str):
9     df = pd.read_csv(path)
10    showPlots(df)
11    return df
12
13 def showPlots(df: pd.DataFrame):
14     st.markdown("# Heart failure")
15     st.markdown("## Datasets")
16
17     # Og chart with no deaths
18     df_no_death = df.copy(deep=True)
19     df_no_death = df_no_death.drop(index=df_no_death[df_no_death["DEATH_EVENT"] == 0].index)
20     df_no_death = df_no_death.drop("DEATH_EVENT", axis=1)
21     df_no_death.sort_values(by="creatinine_phosphokinase")
22     st.markdown("### No deaths")
23     st.dataframe(df_no_death)
24     st.markdown("#### Median")
25     no_death_median = df_no_death.median()
26
27     # No death median table creatinine
28     st.table(no_death_median)
29
30     # Chart with only deaths
31     df_death = df.copy(deep=True)
32     df_death = df_death.drop(index=df_death[df_death["DEATH_EVENT"] == 1].index)
33     df_death = df_death.drop("DEATH_EVENT", axis=1)
34     df_death.sort_values(by="creatinine_phosphokinase")
35     st.markdown("### Only deaths")
36     st.dataframe(df_death)
37     st.markdown("#### Median")
38     death_median = df_death.median()
39
40     # Death median table creatinine
41     st.table(death_median)
42
43     st.markdown("### Corelation table")
44     corr = df.corr()
45     st.dataframe(corr)
46
47     # Corelation heatmap
48     sb.heatmap(corr, annot=True, fmt=".2f", annot_kws={"size": 8}, cmap='coolwarm')
49     st.pyplot(plt)
50
51     # Pairplot
52     relevant_columns = ['ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT']
53     data_relevant = df[relevant_columns]
54     sb.pairplot(data_relevant, hue="DEATH_EVENT")
55     st.pyplot(plt)
56
57     # Histograms of each column
58     for column in df.columns:
59         if column != 'DEATH_EVENT':
60             plt.figure(figsize=(10, 4))
61             sb.histplot(df[column], kde=True, bins=30)
62             plt.title(f'Distribution of {column}')
63             st.pyplot(plt)
64
65 if __name__ == "__main__":
66     df = loadCSV(DATA)
```

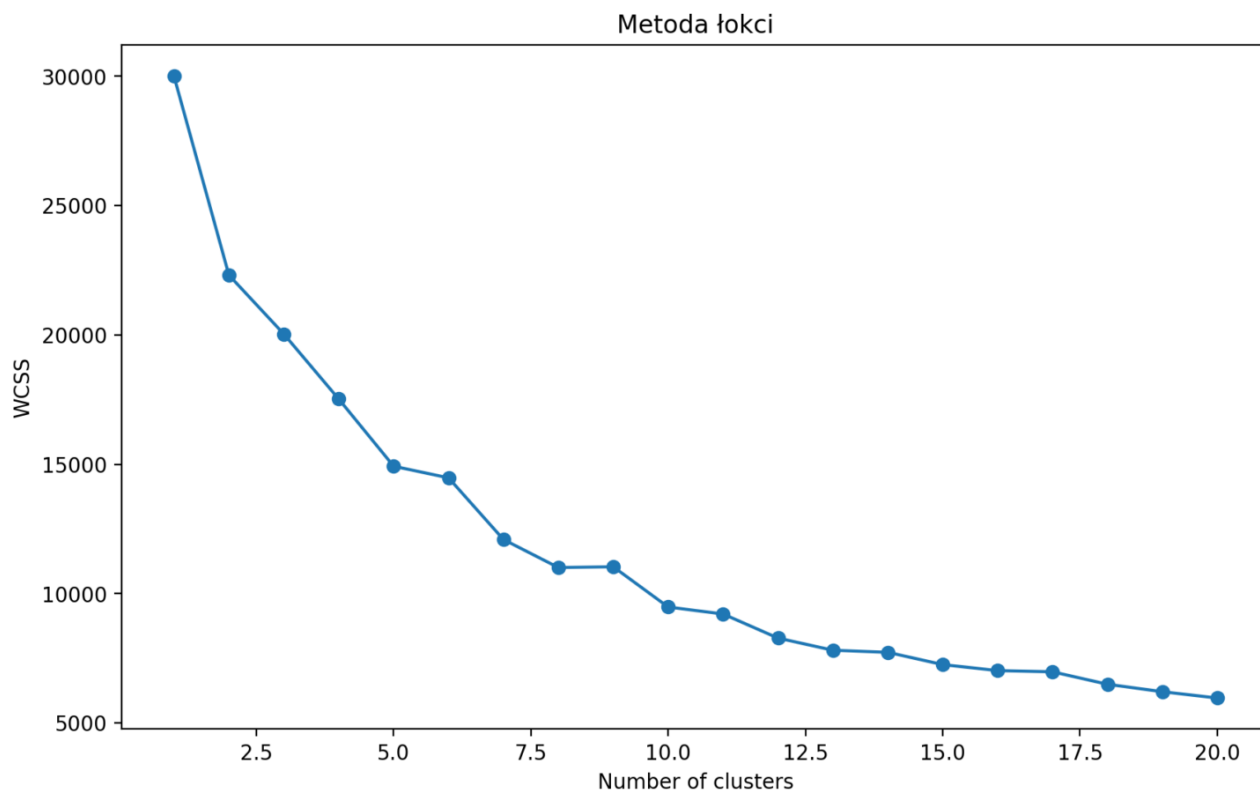
Rysunek 14. Kod programu pokazujący wykresy

3. Modele Predykcyjne I Klasteryzacja Danych

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

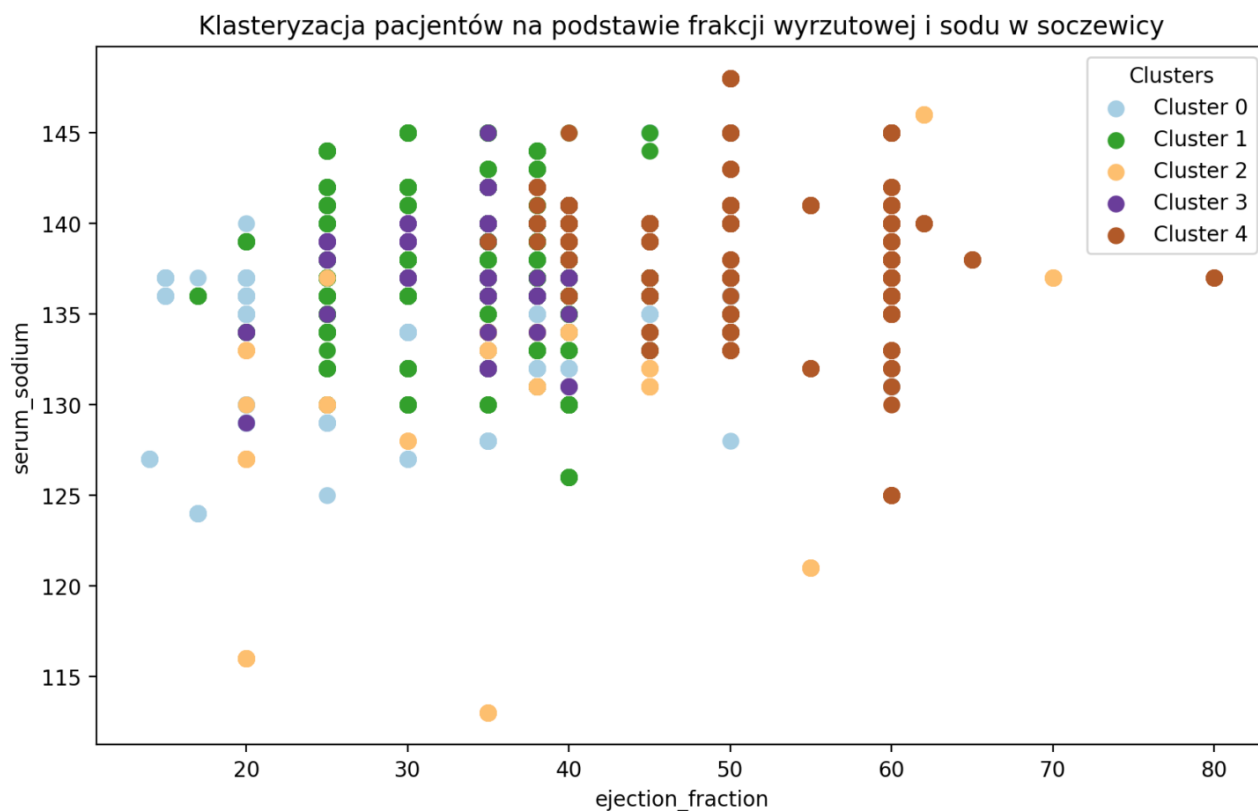
3.1. Budowa Klastrów

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

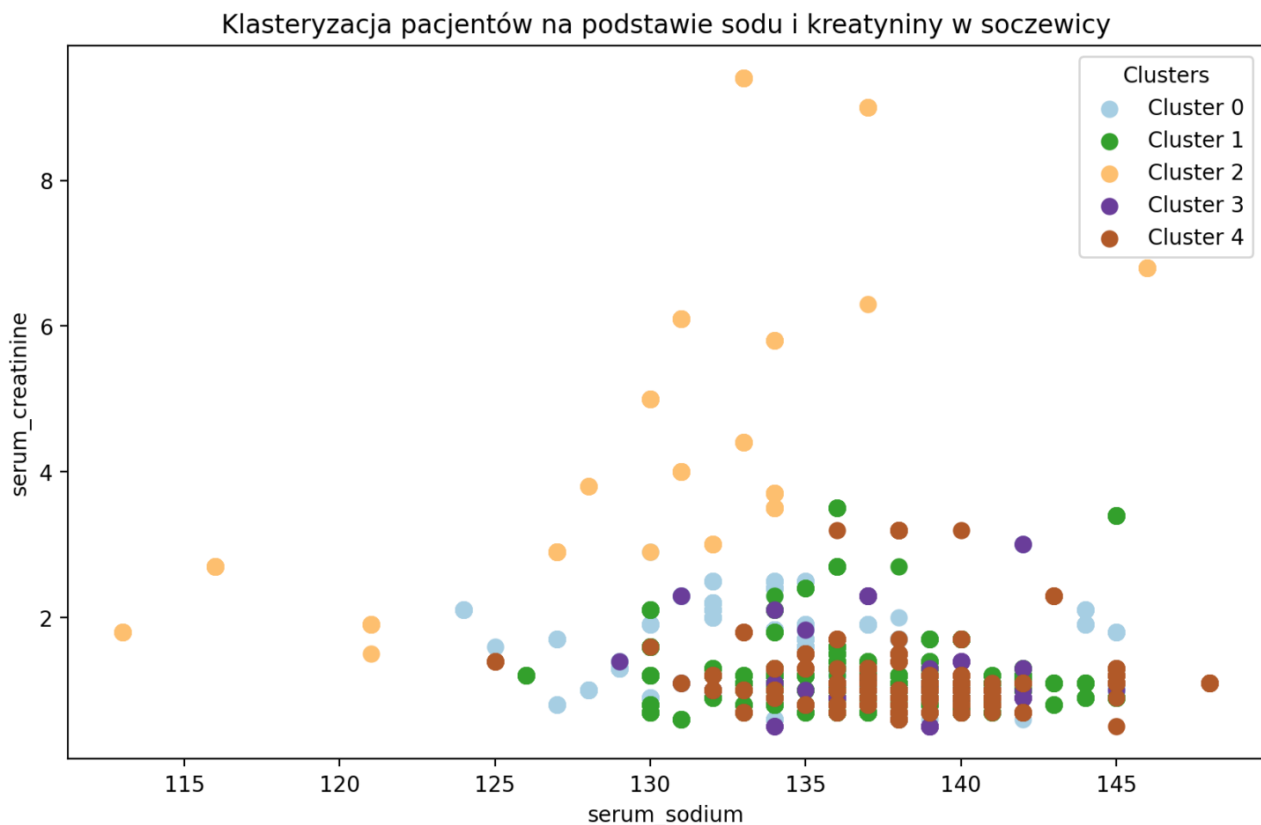
Klastr 0 (jasno-niebieski): Pacjenci o niskiej frakcji wyrzutowej i niskiej ilości sodu w soczewicy

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

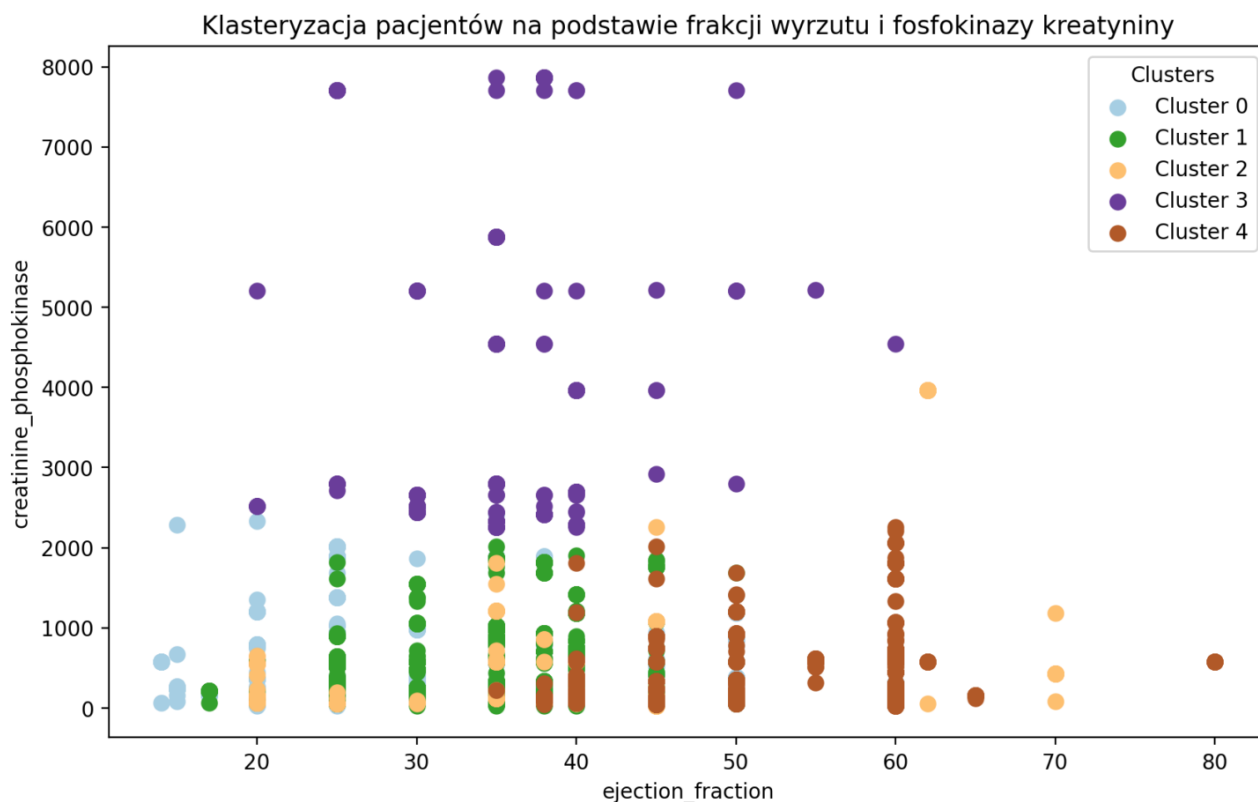
Klastr 0 (jasno-niebieski): Pacjenci o niskiej ilości sodu w soczewicy i umiarkowanej ilości kreatyniny w soczewicy

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

```
1 import pandas as pd
2 import streamlit as st
3 import matplotlib.pyplot as plt
4 import matplotlib as mpl
5 import seaborn as sb
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.cluster import KMeans
8
9 DATA = "../data/heart_failure_clinical_records.csv"
10
11 def loadCSV(path: str):
12     df = pd.read_csv(path)
13     showlots(df)
14     return df
15
16 def showlots(df: pd.DataFrame):
17     st.markdown("# Heart Failure")
18     st.markdown("# Datasets")
19
20 # Og chart with no deaths
21 df_no_death = df.copy(deep=True)
22 df_no_death = df_no_death.drop(index=df_no_death[df_no_death["DEATH_EVENT"] == 0].index)
23 df_no_death = df_no_death.drop("DEATH_EVENT", axis=1)
24 df_no_death.sort_values(by="creatinine_phosphokinase")
25 st.markdown("## No deaths")
26 st.dataframe(df_no_death)
27 st.markdown("### Median")
28 no_death_median = df_no_death.median()
29
30 # No death median table creatinine
31 st.table(no_death_median)
32
33 # Chart with only deaths
34 df_death = df.copy(deep=True)
35 df_death = df_death.drop(index=df_death[df_death["DEATH_EVENT"] == 1].index)
36 df_death = df_death.drop("DEATH_EVENT", axis=1)
37 df_death.sort_values(by="creatinine_phosphokinase")
38 st.markdown("## Only deaths")
39 st.dataframe(df_death)
40 st.markdown("### Median")
41 death_median = df_death.median()
42
43 # Death median table creatinine
44 st.table(death_median)
45
46 st.markdown("## Correlation table")
47 corr = df.corr()
48 st.dataframe(corr)
49
50 # Pairplot
51 sb.pairplot(corr, annot=True, fmt=".2f", annot_kws={"size": 8}, cmap="coolwarm")
52 st.pyplot(plt)
53 plt.close()
54
55 # Pairplot
56 relevant_columns = ["ejection_fraction", "serum_sodium", "serum_creatinine", "time", "creatinine_phosphokinase", "DEATH_EVENT"]
57 data_relevant = df[relevant_columns]
58 plt.figure(figsize=(8, 6))
59 sb.pairplot(data_relevant, hue="DEATH_EVENT")
60 st.pyplot(plt)
61 plt.close()
62
63 # Histograms of each column
64 for column in df.columns:
65     if column == "DEATH_EVENT":
66         plt.figure(figsize=(10, 4))
67         sb.histplot(df[column], kde=True, bins=30)
68         plt.title(f"Distribution of {column}")
69         st.pyplot(plt)
70         plt.close()
71
72 # Clustering data
73 # Elbow method
74 wcss = []
75 # data_scaled = StandardScaler().fit_transform(data_relevant.drop("DEATH_EVENT", axis=1))
76 data_scaled = StandardScaler().fit_transform(data_relevant)
77 for k in range(1, 21):
78     kmeanModel = KMeans(n_clusters=k)
79     kmeanModel.fit(data_scaled)
80     wcss.append(kmeanModel.inertia_)
81
82 st.markdown("# Clustering")
83 st.markdown("## Metoda lokci")
84 plt.figure(figsize=(10, 6))
85
86 # Najlepszy wynik daje 5 klastrow
87 plt.plot(range(1, 21), wcss, marker='o', linestyle='--')
88 plt.xlabel('Number of clusters')
89 plt.ylabel('WCSS')
90 plt.title('Metoda lokci')
91 st.pyplot(plt)
92 plt.close()
93
94 st.markdown("## Scatter")
95 kmeanModel = KMeans(n_clusters=5)
96 cdf = df.copy(deep=True)
97 cdf['cluster'] = kmeanModel.fit_predict(data_scaled)
98 st.write("Cluster mean:")
99 st.table(cdf.groupby('cluster', sort=True).mean())
100
101 ==
102
103 above average serum_creatinine
104 below average serum_sodium
105 below average time
106 below average platelets
107 ==
108
109 plt.figure(figsize=(10, 6))
110 colors = mpl.colormaps['paired'].resampled(5)
111 for cluster_id in range(5):
112     cluster_data = df[cdf['cluster'] == cluster_id]
113     plt.scatter(cluster_data['ejection_fraction'], cluster_data['serum_sodium'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)
114
115 # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'
116 plt.xlabel(relevant_columns[0])
117 plt.ylabel(relevant_columns[1])
118 plt.title('Klasyfikacja pacjentów na podstawie frakcji wyrzutowej i sodu w surowicy')
119 plt.legend(title="Clusters")
120 st.pyplot(plt)
121 plt.close()
122
123
124 plt.figure(figsize=(10, 6))
125 colors = mpl.colormaps['paired'].resampled(5)
126 for cluster_id in range(5):
127     cluster_data = df[cdf['cluster'] == cluster_id]
128     plt.scatter(cluster_data['ejection_fraction'], cluster_data['serum_creatinine'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)
129
130 # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'
131 plt.xlabel(relevant_columns[1])
132 plt.ylabel(relevant_columns[2])
133 plt.title('Klasyfikacja pacjentów na podstawie sodu i kreatyniny w surowicy')
134 plt.legend(title="Clusters")
135 st.pyplot(plt)
136 plt.close()
137
138 plt.figure(figsize=(10, 6))
139 colors = mpl.colormaps['paired'].resampled(5)
140 for cluster_id in range(5):
141     cluster_data = df[cdf['cluster'] == cluster_id]
142     plt.scatter(cluster_data['ejection_fraction'], cluster_data['creatinine_phosphokinase'], color=colors(cluster_id/5), label=f'Cluster {cluster_id}', s=50)
143
144 # 'ejection_fraction', 'serum_sodium', 'serum_creatinine', 'time', 'creatinine_phosphokinase', 'DEATH_EVENT'
145 plt.xlabel(relevant_columns[0])
146 plt.ylabel(relevant_columns[4])
147 plt.title('Klasyfikacja pacjentów na podstawie frakcji wyrzutu i fosfokinazy kreatyniny')
148 plt.legend(title="Clusters")
149 st.pyplot(plt)
150 plt.close()
151
152 if __name__ == "__main__":
153     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.

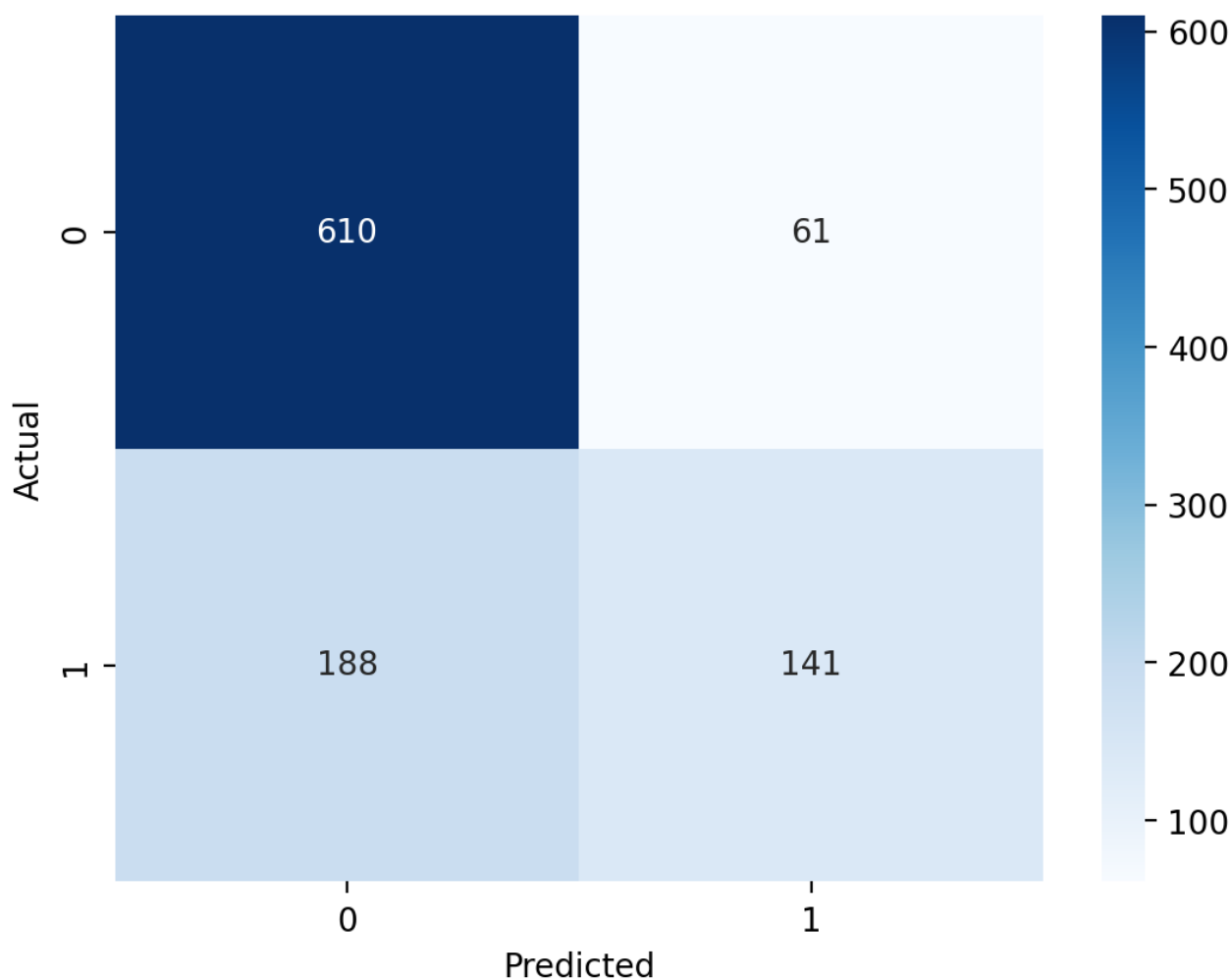
3.2.1. Cel

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, szcząc się poniższymi wynikami:

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



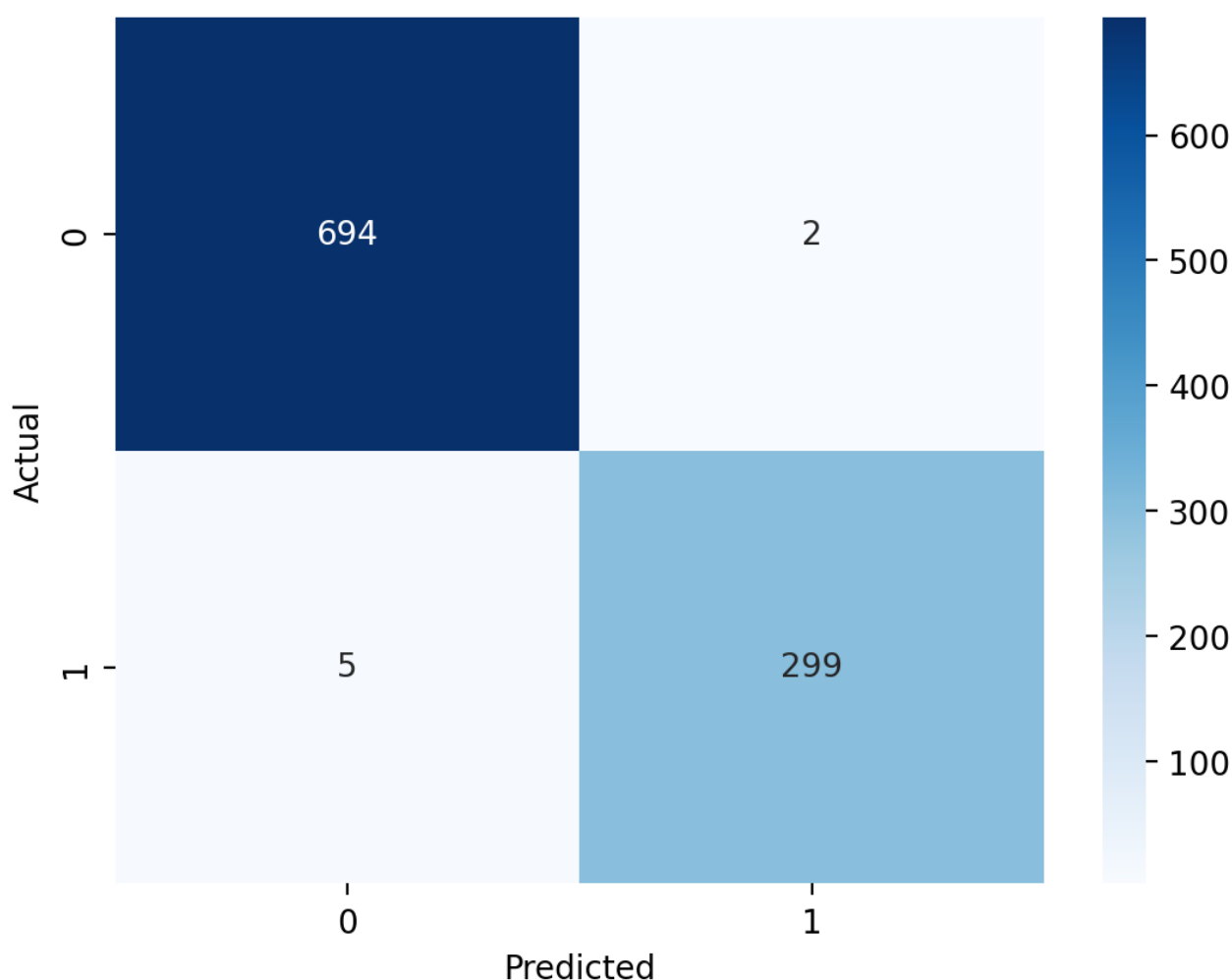
Rysunek 20. Macierz pomyłek regresji liniowej

Powyższy model uzyskał dość dobre wyniki pomimo swojej prostoty 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

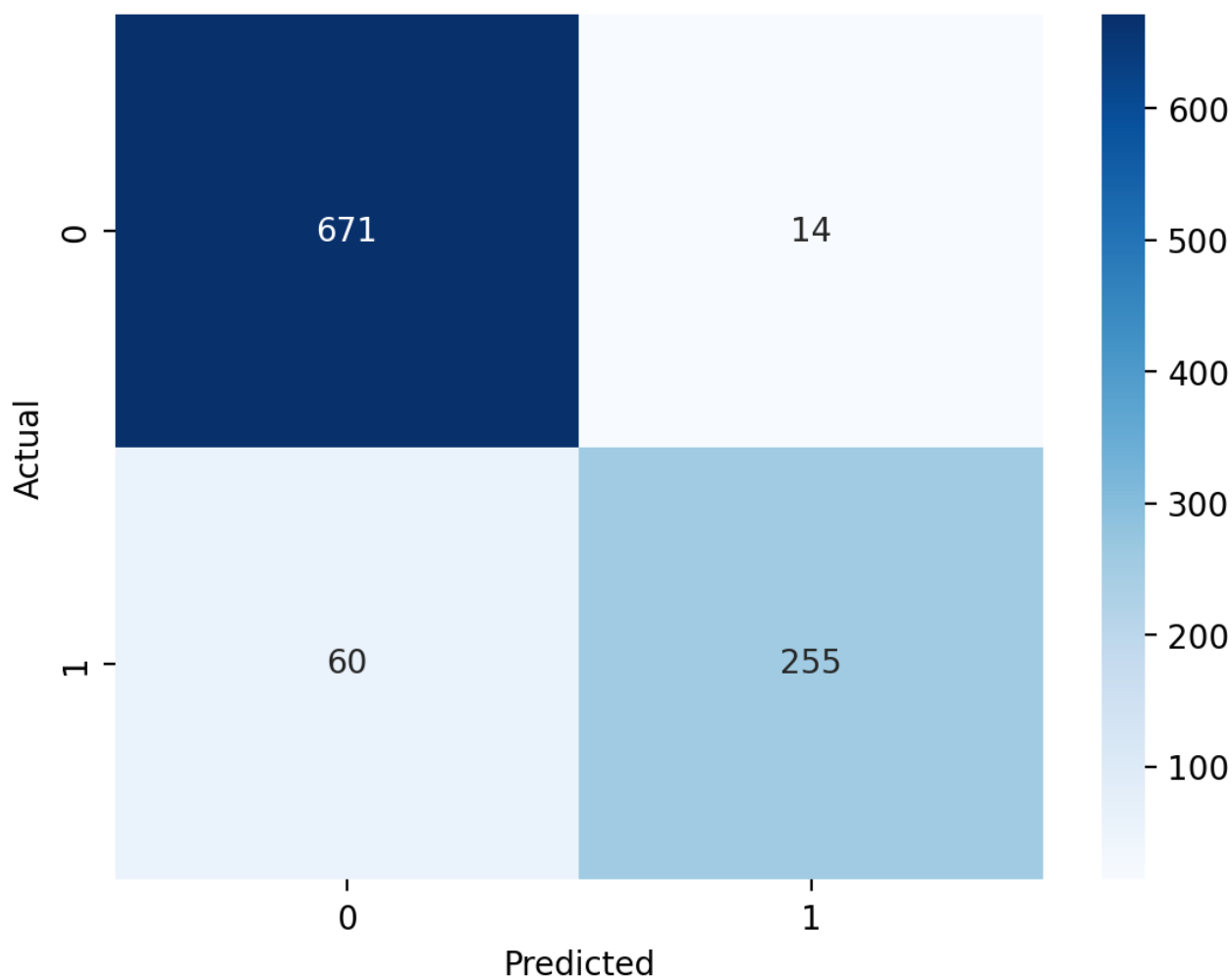
Model lasu najlepiej oszacował szansę wystąpienia niewydolności serca, jednak obarczone jest to kosztem większych zasobów potrzebnych do kalkulacji.

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.



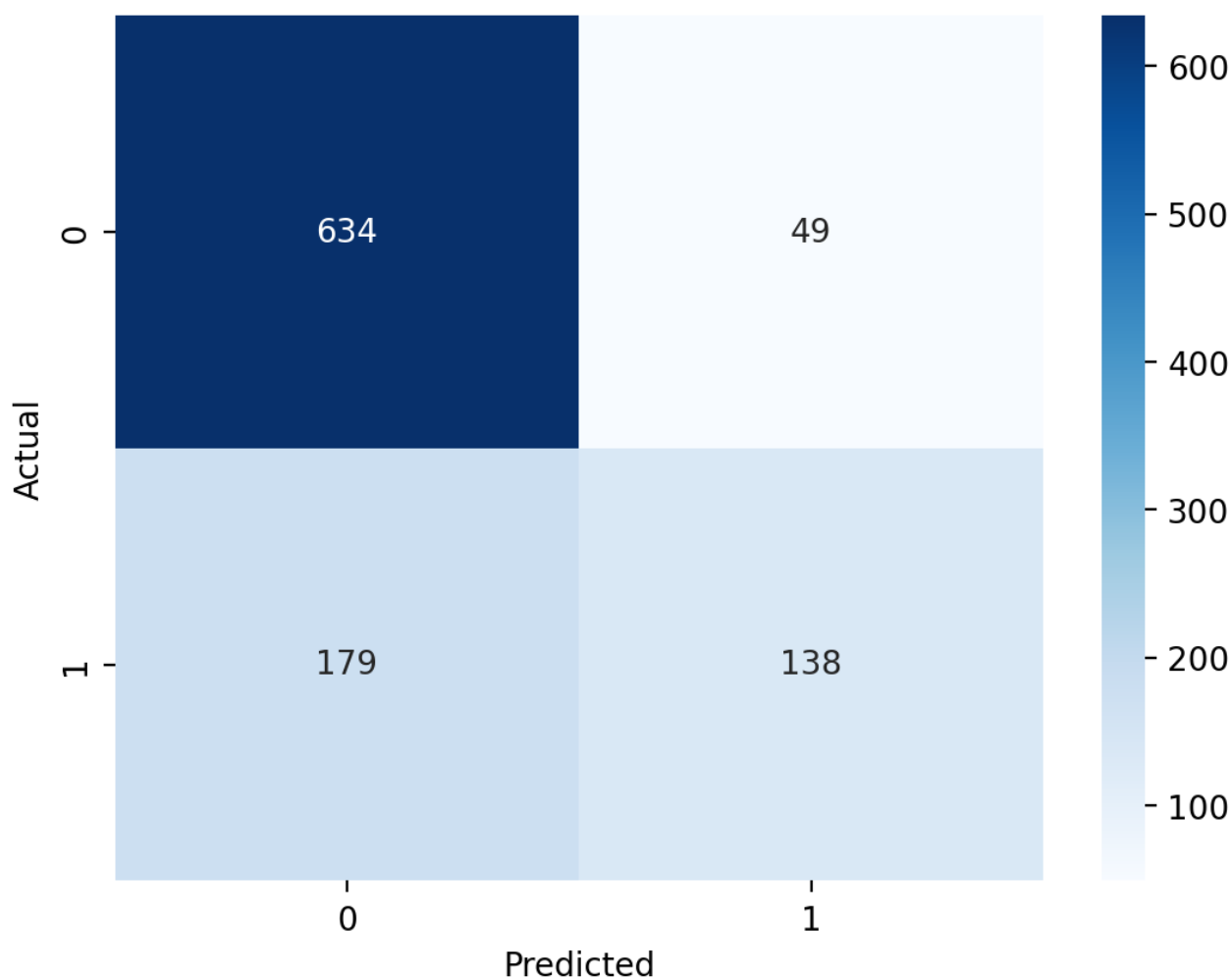
Rysunek 22. Macierz pomyłek wzmocnienia gradientowego

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.



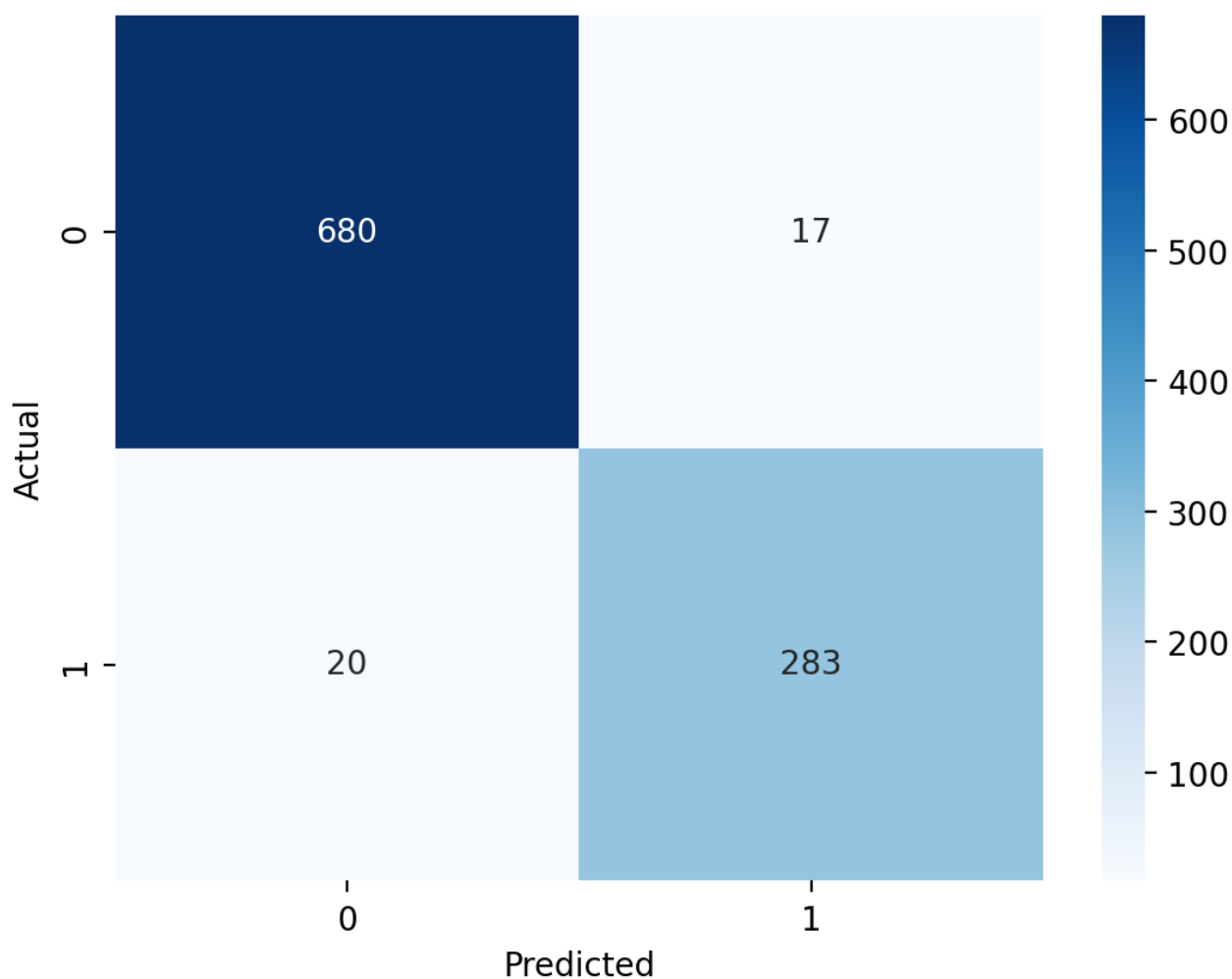
Rysunek 23. Macierz pomyłek liniowego wektora nośnego

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
1 import pandas as pd
2 import streamlit as st
3 import matplotlib.pyplot as plt
4 import seaborn as sb
5 from sklearn.preprocessing import MinMaxScaler
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
8 from sklearn.neighbors import KNeighborsClassifier
9 from sklearn.svm import LinearSVC
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
12
13 DATA = "../data/heart_failure_clinical_records.csv"
14
15 def loadCSV(path: str):
16     df = pd.read_csv(path)
17     return df
18
19 def create_model(df, model, name):
20     x = df.drop("DEATH_EVENT", axis=1).drop("time", axis=1)
21     y = df["DEATH_EVENT"]
22
23     # Data scaling from 0 to 1
24     min_max_scaler = MinMaxScaler()
25
26     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
27
28     x_train = min_max_scaler.fit_transform(x_train)
29     # x_train = standard_scaler.fit_transform(x_train)
30     x_test = min_max_scaler.fit_transform(x_test)
31     # x_test = standard_scaler.fit_transform(x_test)
32
33     model.fit(x_train, y_train)
34
35     # Predict
36     y_pred = model.predict(x_test)
37     st.markdown(f'#{name}')
38     showMetrics(y_test, y_pred)
39
40 def showMetrics(y_test, y_pred):
41     # Metrics
42     ## Accuracy
43     st.write("Accuracy:", accuracy_score(y_test, y_pred))
44
45     ## Confusion matrix
46     conf_matrix = confusion_matrix(y_test, y_pred)
47     st.header('Confusion Matrix')
48     fig, ax = plt.subplots()
49     sb.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', ax=ax)
50     ax.set_xlabel('Predicted')
51     ax.set_ylabel('Actual')
52     st.pyplot(fig)
53
54     ## Classification report
55     class_report = classification_report(y_test, y_pred, output_dict=True)
56     st.header('Classification Report')
57     class_report_df = pd.DataFrame(class_report).transpose()
58     st.dataframe(class_report_df)
59
60 if __name__ == "__main__":
61     df = loadCSV(DATA)
62
63     # Models
64     logistic_reg = LogisticRegression()
65     create_model(df.copy(deep=True), logistic_reg, "Logistic Regression")
66
67     random_forest = RandomForestClassifier()
68     create_model(df.copy(deep=True), random_forest, "Random Forest Classifier")
69
70     gradient = GradientBoostingClassifier()
71     create_model(df.copy(deep=True), gradient, "Gradient Boosting")
72
73     svc = LinearSVC(dual='auto')
74     create_model(df.copy(deep=True), svc, "Linear Support Vector Machines (SVC)")
75
76     k_neighbors = KNeighborsClassifier()
77     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 klastrowi dobrałem dzięki użyciu metody łokci.

Wyćwiczyłem modele predykcyjne takie jak losowy las decyzyjny, wzmocnie gradient, k-najbliższy 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 modeli 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.