Heart_failure case study

May 15, 2024

0.0.1 Importing Libraries

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0.0.2 Loading Data

[4]:
```
df=pd.read_csv('heart_failure_clinical_records.csv')
```

0.0.3 Exploring dataset

[6]:
```
df.shape
```

[6]:
```
(5000, 13)
```

[7]:
```
df.head()
```

[7]:
```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  55.0        0                       748         0                 45
1  65.0        0                        56         0                 25
2  45.0        0                       582         1                 38
3  60.0        1                       754         1                 40
4  95.0        1                       582         0                 30

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    0  263358.03               1.3           137    1
1                    0  305000.00               5.0           130    1
2                    0  319000.00               0.9           140    0
3                    1  328000.00               1.2           126    1
4                    0  461000.00               2.0           132    1

   smoking  time  DEATH_EVENT
0        1    88            0
1        0   207            0
2        0   244            0
```

```
3        0    90            0
4        0    50            1
```

[8]:
```
df.tail()
```

[8]:
```
       age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
4995  45.0        0                       582         1                 55
4996  60.0        1                       582         0                 30
4997  95.0        1                       112         0                 40
4998  65.0        1                       160         1                 20
4999  40.0        0                       244         0                 45

      high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
4995                    0   543000.0               1.0           132    0
4996                    1   127000.0               0.9           145    0
4997                    1   196000.0               1.0           138    0
4998                    0   327000.0               2.7           116    0
4999                    1   275000.0               0.9           140    0

      smoking  time  DEATH_EVENT
4995        0   250            0
4996        0    95            0
4997        0    24            1
4998        0     8            1
4999        0   174            0
```

[9]:
```
df.columns
```

[9]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

[10]:
```
df.info
```

[10]:
```
<bound method DataFrame.info of        age  anaemia  creatinine_phosphokinase
diabetes  ejection_fraction  \
0     55.0        0                       748         0                 45
1     65.0        0                        56         0                 25
2     45.0        0                       582         1                 38
3     60.0        1                       754         1                 40
4     95.0        1                       582         0                 30
…     …       …                         …       …                 …
4995  45.0        0                       582         1                 55
4996  60.0        1                       582         0                 30
4997  95.0        1                       112         0                 40
```

```
4998  65.0        1                       160         1                 20
4999  40.0        0                       244         0                 45

      high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                       0  263358.03               1.3           137    1
1                       0  305000.00               5.0           130    1
2                       0  319000.00               0.9           140    0
3                       1  328000.00               1.2           126    1
4                       0  461000.00               2.0           132    1
…                     …        …                 …           …  …
4995                    0  543000.00               1.0           132    0
4996                    1  127000.00               0.9           145    0
4997                    1  196000.00               1.0           138    0
4998                    0  327000.00               2.7           116    0
4999                    1  275000.00               0.9           140    0

      smoking  time  DEATH_EVENT
0           1    88            0
1           0   207            0
2           0   244            0
3           0    90            0
4           0    50            1
…       …   …          …
4995        0   250            0
4996        0    95            0
4997        0    24            1
4998        0     8            1
4999        0   174            0

[5000 rows x 13 columns]>
```

0.0.4 Duplicated Values

[12]:
```
df.duplicated().sum()
```

[12]:
```
3680
```

0.0.5 Missing Values

[14]:
```
df.isna().sum()
```

[14]:
```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
```

```
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

0.0.6 Data Distribution

[16]:
```
df.dtypes
```

[16]:
```
age                         float64
anaemia                       int64
creatinine_phosphokinase      int64
diabetes                      int64
ejection_fraction             int64
high_blood_pressure           int64
platelets                   float64
serum_creatinine            float64
serum_sodium                  int64
sex                           int64
smoking                       int64
time                          int64
DEATH_EVENT                   int64
dtype: object
```

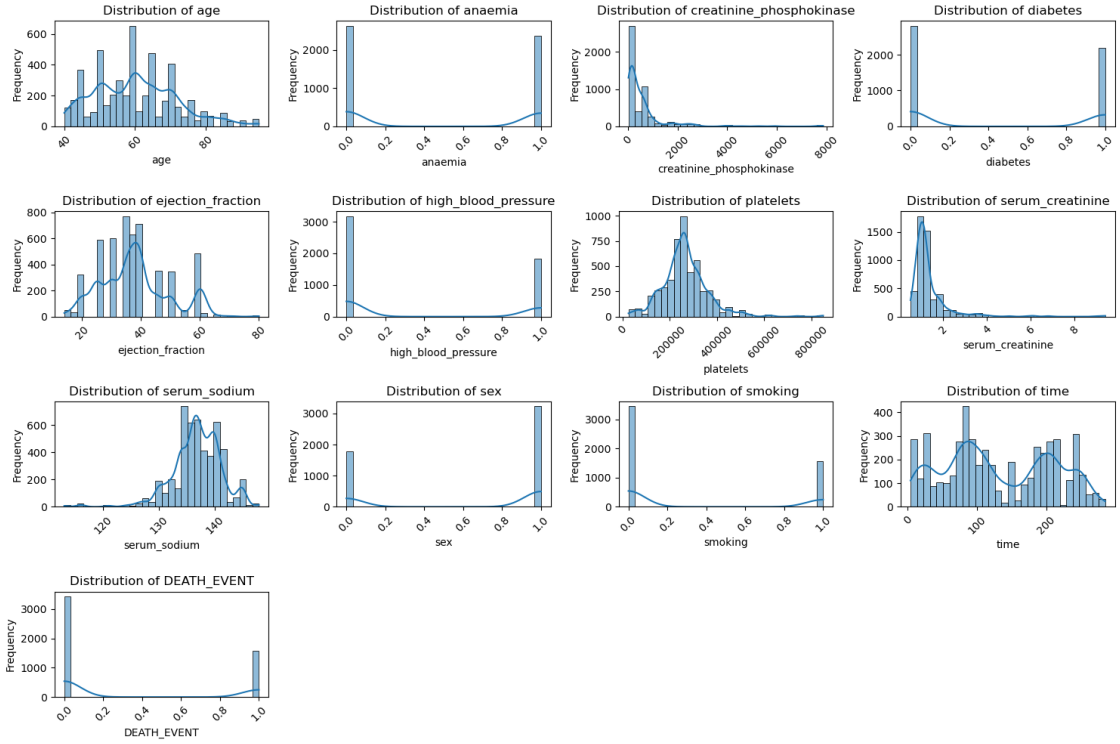
0.0.7 Visualizing distribution

[18]:
```
plt.figure(figsize=(15,10))
for i, column in enumerate(df):
    plt.subplot(4,4,i+1)
    sns.histplot(df[column],bins=30,kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
```

```
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



0.0.8 Visualizing skewness

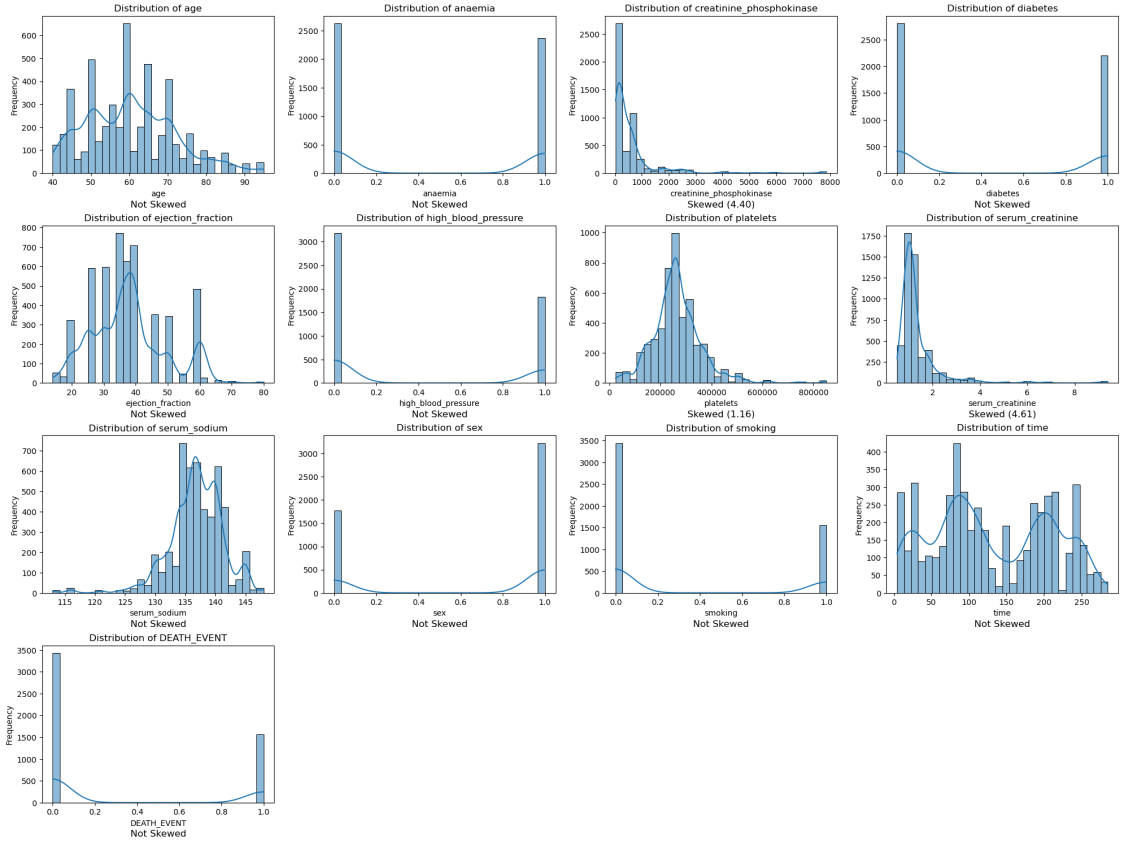
[20]:
```
from scipy import stats
```

[21]:
```
plt.figure(figsize=(20, 15))
for i, column in enumerate(df.columns):
    plt.subplot(4, 4, i+1)
    sns.histplot(df[column], bins=30, kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
```

```
    plt.ylabel('Frequency')

    skewness = stats.skew(df[column])
    if skewness < -1 or skewness > 1:
        plt.text(0.5, -0.2, f'Skewed ({skewness:.2f})',␣
↪horizontalalignment='center', verticalalignment='center', transform=plt.
↪gca().transAxes, fontsize=12)
    else:
        plt.text(0.5, -0.2, 'Not Skewed', horizontalalignment='center',␣
↪verticalalignment='center', transform=plt.gca().transAxes, fontsize=12)
plt.tight_layout()
plt.show()
```

```
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
```

```
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN
before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



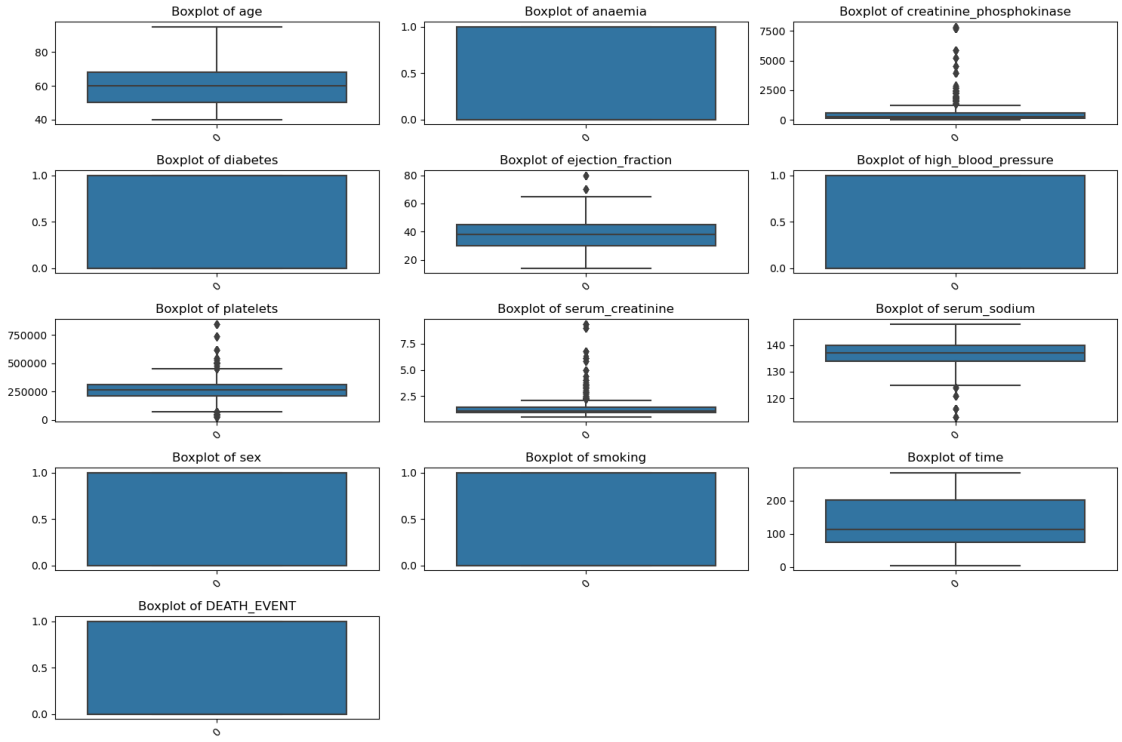
[22]:
```
for col in df:
    skewness = df[col].skew()
    print(f"Skewness of '{col}': {skewness}")
```

```
Skewness of 'age': 0.45081168033242247
Skewness of 'anaemia': 0.10256525432936629
Skewness of 'creatinine_phosphokinase': 4.403444256118772
Skewness of 'diabetes': 0.2442735058883124
Skewness of 'ejection_fraction': 0.4944047809955734
Skewness of 'high_blood_pressure': 0.5618939376858764
Skewness of 'platelets': 1.157117070165634
Skewness of 'serum_creatinine': 4.615903417419207
Skewness of 'serum_sodium': -0.997974578554996
Skewness of 'sex': -0.6089660368778839
Skewness of 'smoking': 0.8128015889407144
Skewness of 'time': 0.11498892481750181
Skewness of 'DEATH_EVENT': 0.8037658645232266
```

0.1 Outliers

0.1.1 Checking for outliers

[25]:
```
plt.figure(figsize=(15,10))
for i, column in enumerate(df):
    plt.subplot(5,3,i+1)
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Columns with outliers 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium'

0.1.2 Percentage of outliers

[29]:
```
def robust_zscore_outlier_removal(df, columns):
    for column in columns:
        median = df[column].median()
        mad = np.median(np.abs(df[column] - median))
        modified_z_scores = 0.6745 * (df[column] - median) / mad
        df = df[(modified_z_scores > -3.5) & (modified_z_scores < 3.5)]
```

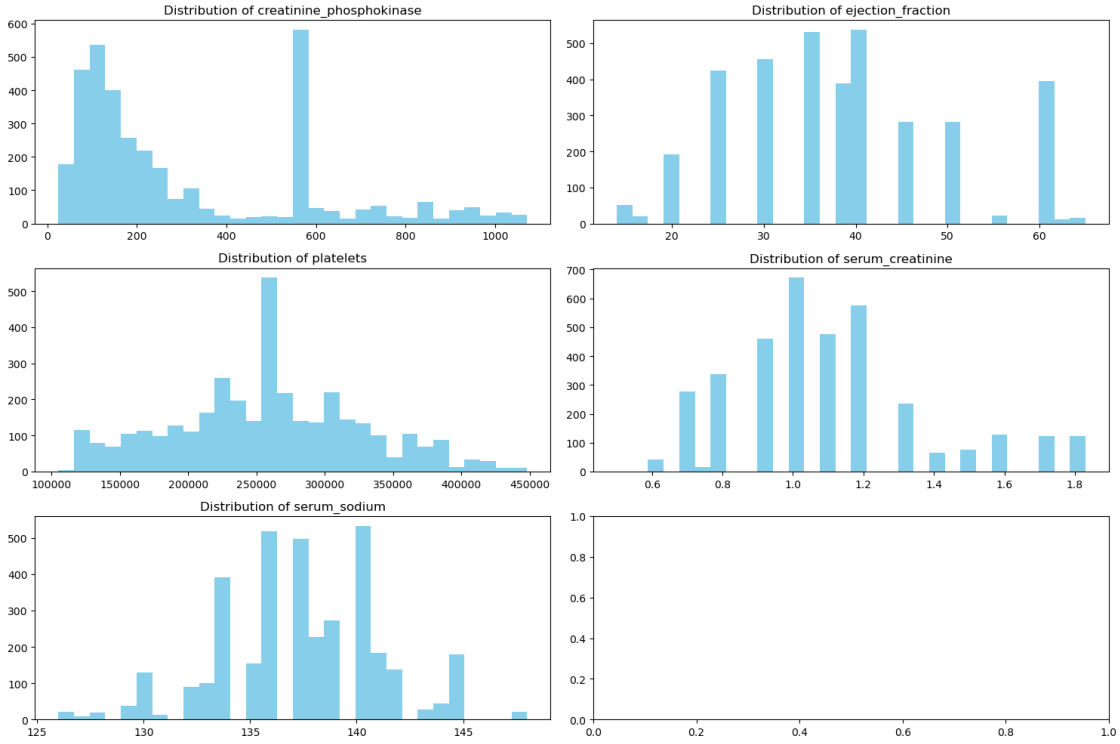
```
    return df

outlier_columns = ['creatinine_phosphokinase', 'ejection_fraction',␣
 ↪'platelets', 'serum_creatinine', 'serum_sodium']
df1 = robust_zscore_outlier_removal(df, outlier_columns)
def calculate_outlier_percentages(df, columns):
    outlier_percentages = {}

    for column in columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]


        outlier_percentage = len(outliers) / len(df) * 100
        outlier_percentages[column] = outlier_percentage
    return outlier_percentages
outlier_percentages_df1 = calculate_outlier_percentages(df1, outlier_columns)
print("Percentage of outliers in each column of df1 after robust Z-score␣
 ↪cleaning:")
for column, percentage in outlier_percentages_df1.items():
    print(f"{column}: {percentage:.2f}%")
```

```
Percentage of outliers in each column of df1 after robust Z-score cleaning:
creatinine_phosphokinase: 0.00%
ejection_fraction: 0.00%
platelets: 5.10%
serum_creatinine: 10.60%
serum_sodium: 2.13%
```

0.1.3 Handling outliers using quartile method

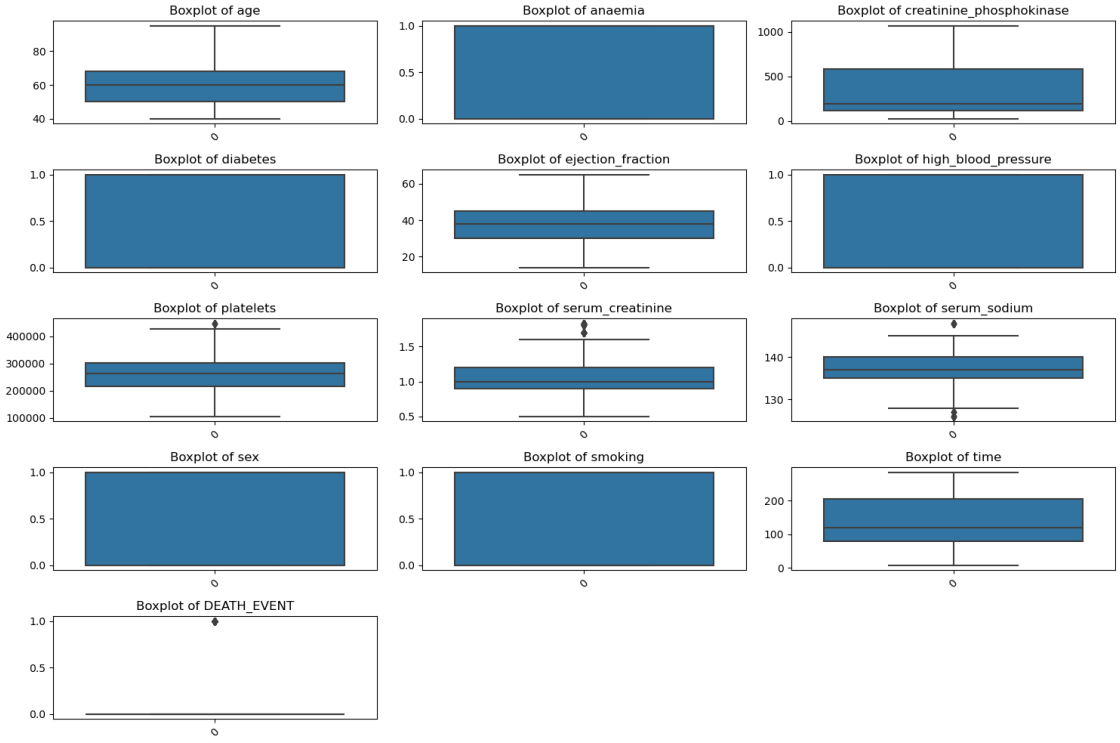
[31]:
```
def robust_zscore_outlier_removal(df, columns, thresholds):
    for column in columns:
        median = df[column].median()
        mad = np.median(np.abs(df[column] - median))
        modified_z_scores = 0.6745 * (df[column] - median) / mad
        threshold = thresholds[column]
        df = df[(modified_z_scores > -threshold) & (modified_z_scores <␣
 ↪threshold)]

    return df
outlier_columns = ['creatinine_phosphokinase', 'ejection_fraction',␣
 ↪'platelets', 'serum_creatinine', 'serum_sodium']
```

```
thresholds = {
    'creatinine_phosphokinase': 3.5,  # No outliers, keep standard
    'ejection_fraction': 3.5,         # No outliers, keep standard
    'platelets': 2.5,                 # More stringent due to persistent␣
 ↪outliers
    'serum_creatinine': 2.5,          # More stringent due to persistent␣
 ↪outliers
    'serum_sodium': 2.5               # More stringent due to persistent␣
 ↪outliers
}

df1 = robust_zscore_outlier_removal(df, outlier_columns, thresholds)
fig, axs = plt.subplots(3, 2, figsize=(15, 10))
axs = axs.flatten()
for i, column in enumerate(outlier_columns):
    axs[i].hist(df1[column], bins=30, color='skyblue')
    axs[i].set_title(f'Distribution of {column}')
plt.tight_layout()
plt.show()
outlier_percentages_df1 = calculate_outlier_percentages(df1, outlier_columns)

print("Percentage of outliers in each column of df1 after adjusted robust␣
 ↪Z-score cleaning:")
for column, percentage in outlier_percentages_df1.items():
    print(f"{column}: {percentage:.2f}%")
```



```
Percentage of outliers in each column of df1 after adjusted robust Z-score
cleaning:
creatinine_phosphokinase: 0.00%
ejection_fraction: 0.00%
platelets: 0.30%
serum_creatinine: 6.79%
serum_sodium: 1.44%
```

[40]:
```
plt.figure(figsize=(15,10))
for i, column in enumerate(df1):
    plt.subplot(5,3,i+1)
    sns.boxplot(df1[column])
    plt.title(f'Boxplot of {column}')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



0.1.4 Identifying the best model

[50]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

features = ['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',␣
 ↪'ejection_fraction', 'high_blood_pressure',
           'platelets', 'serum_creatinine', 'serum_sodium', 'sex', 'smoking',␣
 ↪'time']
target = 'DEATH_EVENT'
X = df1[features]
y = df1[target]

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# List of classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
    "Multinomial Naive Bayes": MultinomialNB(),
    "Gaussian Naive Bayes": GaussianNB()
}

# Training and evaluating each classifier
results = {}
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = {
        "Accuracy": accuracy,
        "Misclassification Rate": 1 - accuracy,
        "Classification Report": classification_report(y_test, y_pred),
        "Confusion Matrix": confusion_matrix(y_test, y_pred)
    }
```

```python
for name, result in results.items():
    print(f"{name}:")
    print(f"Accuracy: {result['Accuracy']}")
    print(f"Misclassification Rate: {result['Misclassification Rate']}")
    print("Classification Report:")
    print(result["Classification Report"])
    print("Confusion Matrix:")
    print(result["Confusion Matrix"])
    print()
```

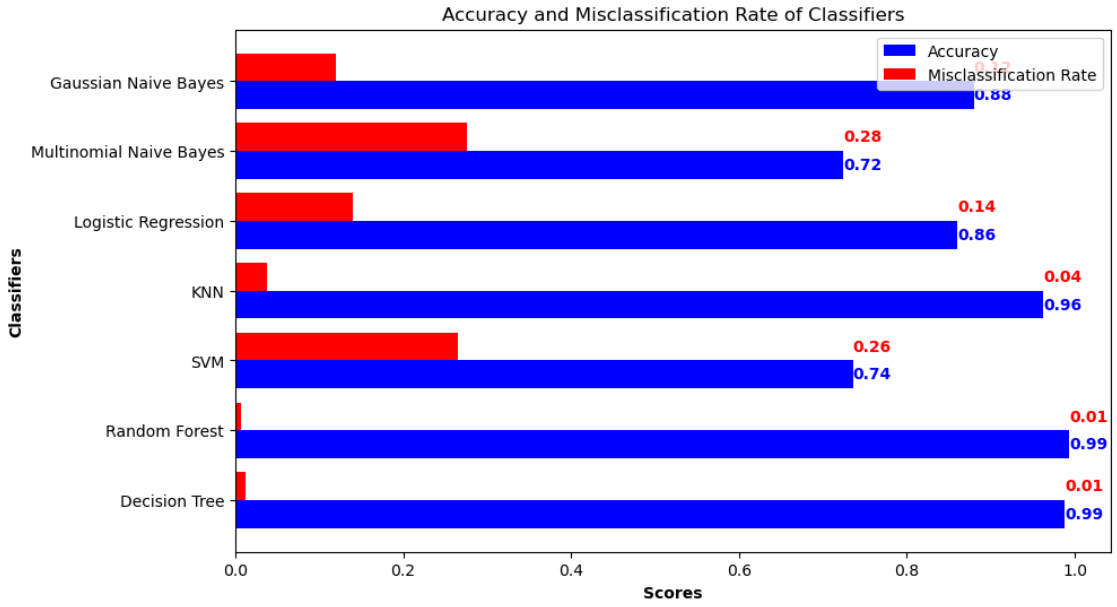
```
Decision Tree:
Accuracy: 0.9847645429362881
Misclassification Rate: 0.015235457063711877
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       531
           1       0.98      0.96      0.97       191

    accuracy                           0.98       722
   macro avg       0.98      0.98      0.98       722
weighted avg       0.98      0.98      0.98       722

Confusion Matrix:
[[528   3]
 [  8 183]]

Random Forest:
Accuracy: 0.9916897506925207
Misclassification Rate: 0.008310249307479256
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       531
           1       0.99      0.97      0.98       191

    accuracy                           0.99       722
   macro avg       0.99      0.99      0.99       722
weighted avg       0.99      0.99      0.99       722

Confusion Matrix:
[[530   1]
 [  5 186]]

SVM:
Accuracy: 0.7354570637119113
```

```
Misclassification Rate: 0.26454293628808867
Classification Report:
              precision    recall  f1-score   support

           0       0.74      1.00      0.85       531
           1       0.00      0.00      0.00       191

    accuracy                           0.74       722
   macro avg       0.37      0.50      0.42       722
weighted avg       0.54      0.74      0.62       722

Confusion Matrix:
[[531   0]
 [191   0]]

KNN:
Accuracy: 0.9626038781163435
Misclassification Rate: 0.037396121883656486
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       531
           1       0.95      0.91      0.93       191

    accuracy                           0.96       722
   macro avg       0.96      0.95      0.95       722
weighted avg       0.96      0.96      0.96       722

Confusion Matrix:
[[521  10]
 [ 17 174]]

Logistic Regression:
Accuracy: 0.8601108033240997
Misclassification Rate: 0.13988919667590027
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91       531
           1       0.83      0.60      0.69       191

    accuracy                           0.86       722
   macro avg       0.85      0.78      0.80       722
weighted avg       0.86      0.86      0.85       722

Confusion Matrix:
[[507  24]
 [ 77 114]]
```

```
Multinomial Naive Bayes:
Accuracy: 0.724376731301939
Misclassification Rate: 0.275623268698061
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.75      0.80       531
           1       0.48      0.66      0.56       191

    accuracy                           0.72       722
   macro avg       0.67      0.70      0.68       722
weighted avg       0.76      0.72      0.74       722

Confusion Matrix:
[[397 134]
 [ 65 126]]

Gaussian Naive Bayes:
Accuracy: 0.8795013850415513
Misclassification Rate: 0.12049861495844871
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92       531
           1       0.89      0.62      0.73       191

    accuracy                           0.88       722
   macro avg       0.88      0.80      0.83       722
weighted avg       0.88      0.88      0.87       722

Confusion Matrix:
[[516  15]
 [ 72 119]]


/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-
packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning:
```

```
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[54]:
```
fig, ax = plt.subplots(figsize=(10, 6))
y_pos = np.arange(len(classifiers_names))

# Plotting bars
bar1 = ax.barh(y_pos, accuracy_values, color='b', height=0.4, label='Accuracy')
bar2 = ax.barh(y_pos + 0.4, misclassification_rates, color='r', height=0.4,␣
 ↪label='Misclassification Rate')

# Adding labels
ax.set_xlabel('Scores', fontweight='bold')
ax.set_ylabel('Classifiers', fontweight='bold')
ax.set_yticks(y_pos + 0.2)
ax.set_yticklabels(classifiers_names)
ax.legend()

# Adding labels to bars
for bar, acc, mis in zip(bar1, accuracy_values, misclassification_rates):
    ax.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{acc:.2f}',␣
 ↪va='center', ha='left', fontsize=10, fontweight='bold', color='blue')
    ax.text(bar.get_width(), bar.get_y() + bar.get_height() / 2 + 0.4, f'{mis:.
 ↪2f}', va='center', ha='left', fontsize=10, fontweight='bold', color='red')

# Displaying the plot
plt.title('Accuracy and Misclassification Rate of Classifiers')
plt.show()
```



0.1.5 Using Random Forest

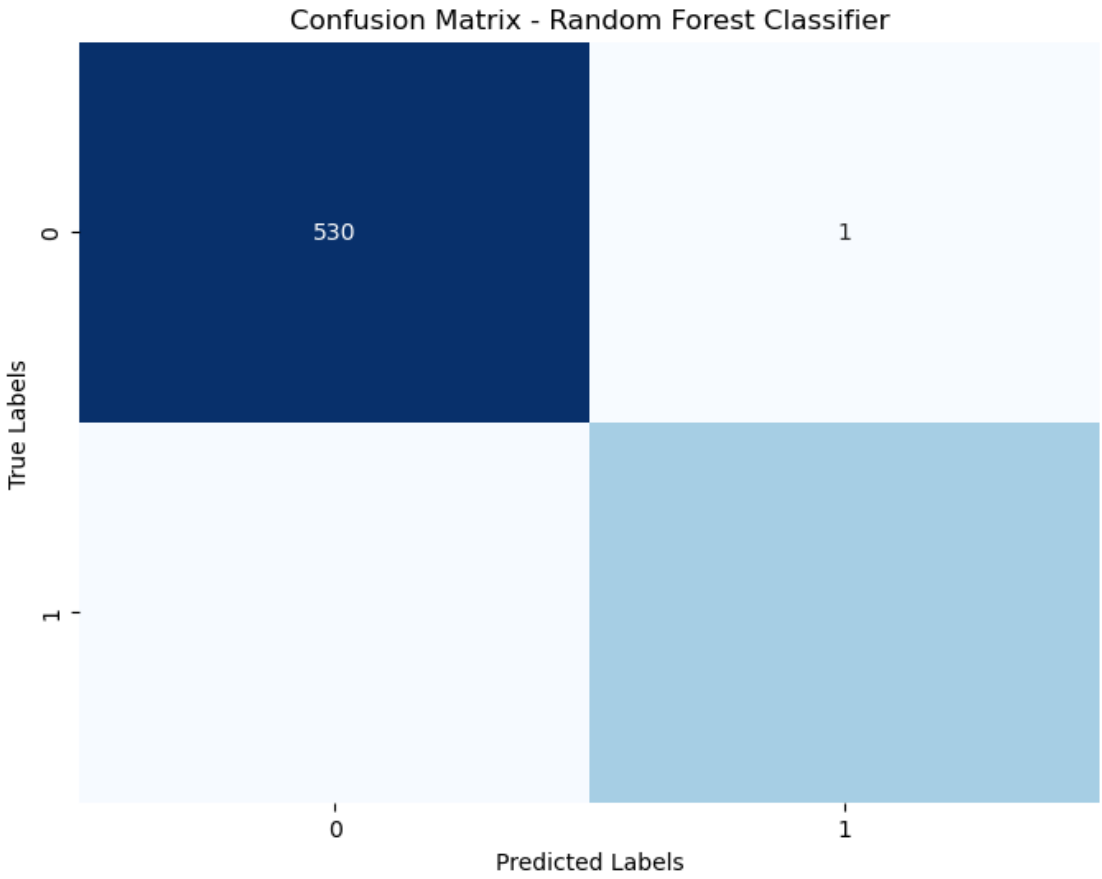
[65]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Features and target variable
features = ['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',␣
 ↪'ejection_fraction', 'high_blood_pressure',
            'platelets', 'serum_creatinine', 'serum_sodium', 'sex', 'smoking',␣
 ↪'time']
target = 'DEATH_EVENT'
X = df1[features]
y = df1[target]

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

# Predictions on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# Misclassification Rate
misclassification_rate_rf = 1 - accuracy_rf

print("Random Forest:")
print(f"Accuracy: {accuracy_rf}")
print(f"Misclassification Rate: {misclassification_rate_rf}")
```

```
Random Forest:
Accuracy: 0.9916897506925207
Misclassification Rate: 0.008310249307479256
```

0.1.6 Confusion Matrix

[68]:
```
from sklearn.metrics import confusion_matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', cbar=False)
```

```
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Random Forest Classifier')
plt.show()
```



[ ]: