EDA Assignment – 1 (Heart Failure Clinical Records)

Submitter Name: Aasif Majeed
Date: 14 Mar 2024

This notebook performs **Exploratory Data Analysis (EDA)** on the **Heart Failure Clinical Records** dataset described in the assignment PDF.
It answers each question using:

- clean data loading
- summary statistics
- clear plots (Matplotlib)
- short explanations in Markdown

Dataset source (Google Drive link provided in assignment):
`https://drive.google.com/file/d/1xxpwaYGWx0iw33FuMe_mAPpnzBhaQZeo/view`

---

0) Setup

We will use:

- `pandas` for data handling
- `numpy` for numerical ops
- `matplotlib` for visualization

1. Install `gdown` (`pip install gdown`) OR
2. Manually download the CSV from the Drive link and put it in the same folder as this notebook.

> If you run this notebook in a new environment and the download step fails, you can:

In [9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Better display
pd.set_option("display.max_columns", 50)
pd.set_option("display.width", 120)
```

---

1) Download / Load Dataset

Goal

Download the dataset CSV from Google Drive using its file-id.

Drive link:
`https://drive.google.com/file/d/1xxpwaYGWx0iw33FuMe_mAPpnzBhaQZeo/view`

The file-id is:
`1xxpwaYGWx0iw33FuMe_mAPpnzBhaQZeo`

In [10]:
```
# ---- Download (optional) ----
# If you already downloaded the dataset manually, set DATA_PATH to that file.
# Otherwise, this cell attempts to download via gdown.

DATA_PATH = "heart_failure_clinical_records.csv"  # output file name

file_id = "1xxpwaYGWx0iw33FuMe_mAPpnzBhaQZeo"
gdrive_url = f"https://drive.google.com/uc?id={file_id}"

try:
    import os
    if not os.path.exists(DATA_PATH):
        try:
            import gdown
        except ImportError:
            raise ImportError("gdown is not installed. Run: pip install gdown
        gdown.download(gdrive_url, DATA_PATH, quiet=False)
    print("Dataset available at:", DATA_PATH)
except Exception as e:
    print("Download step note:", e)
    print("If download fails, manually download the CSV from the Drive link ar
```

```
Dataset available at: heart_failure_clinical_records.csv
```

Load dataset into a DataFrame

We also normalize column names to lower-case for safer referencing (because some versions use `DEATH_EVENT` vs `death_event`, etc.).

In [11]:
```
df = pd.read_csv(DATA_PATH)

# Normalize column names
df.columns = [c.strip().lower().replace(" ", "_") for c in df.columns]
```

```
df.head()
```

Out[11]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_bl
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	

---

2) Quick Dataset Overview

- Shape (rows, columns)
- Data types and missing values
- Summary statistics

In [12]:
```
print("Shape:", df.shape)
df.info()
```

```
Shape: (299, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    int64  
 3   diabetes                  299 non-null    int64  
 4   ejection_fraction         299 non-null    int64  
 5   high_blood_pressure       299 non-null    int64  
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64  
 9   sex                       299 non-null    int64  
 10  smoking                   299 non-null    int64  
 11  time                      299 non-null    int64  
 12  death_event               299 non-null    int64  
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

In [5]:
```
# Missing values check
df.isna().sum()
```

Out[5]:
```
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
```

In [6]:
```
df.describe(include="all")
```

Out[6]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fr
count	299.000000	299.000000	299.000000	299.000000	299.(
mean	60.833893	0.431438	581.839465	0.418060	38.(
std	11.894809	0.496107	970.287881	0.494067	11.:
min	40.000000	0.000000	23.000000	0.000000	14.(
25%	51.000000	0.000000	116.500000	0.000000	30.(
50%	60.000000	0.000000	250.000000	0.000000	38.(
75%	70.000000	1.000000	582.000000	1.000000	45.(
max	95.000000	1.000000	7861.000000	1.000000	80.(

---

3) Helper functions (for consistent plots)

In [7]:
```
def plot_hist(series, bins=20, title="", xlabel="", ylabel="Count"):
    plt.figure(figsize=(7,4))
    plt.hist(series.dropna(), bins=bins)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.show()

def plot_bar(labels, values, title="", xlabel="", ylabel="Count", rotate=0):
    plt.figure(figsize=(7,4))
    plt.bar(labels, values)
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
```

```
    plt.xticks(rotation=rotate)
    plt.show()

def plot_box(data_list, labels, title="", ylabel=""):
    plt.figure(figsize=(7,4))
    plt.boxplot(data_list, labels=labels)
    plt.title(title)
    plt.ylabel(ylabel)
    plt.show()

def safe_col(name_candidates):
    """Return the first matching column from candidates (case-insensitive), el
    cols = set(df.columns)
    for c in name_candidates:
        if c in cols:
            return c
    raise KeyError(f"None of {name_candidates} found in columns: {sorted(df.cc
```

In [8]:
```
# Identify important columns (robust to naming variations)
age_col = safe_col(["age"])
sex_col = safe_col(["sex"])
death_col = safe_col(["death_event", "death"])
platelets_col = safe_col(["platelets"])
serum_creatinine_col = safe_col(["serum_creatinine"])
serum_sodium_col = safe_col(["serum_sodium"])
high_bp_col = safe_col(["high_blood_pressure", "high_blood_pressure_"])
smoking_col = safe_col(["smoking"])
diabetes_col = safe_col(["diabetes"])
ejection_col = safe_col(["ejection_fraction"])

print("Columns used:")
print(age_col, sex_col, death_col, platelets_col, serum_creatinine_col, serum_
```

```
Columns used:
age sex death_event platelets serum_creatinine serum_sodium high_blood_pressure
smoking diabetes ejection_fraction
```

Assignment Questions (EDA)

The PDF asks EDA questions about the Heart Failure Clinical Records dataset.

We answer them below with charts + brief interpretation.

Q1) What is the distribution of age among heart failure patients in the dataset?

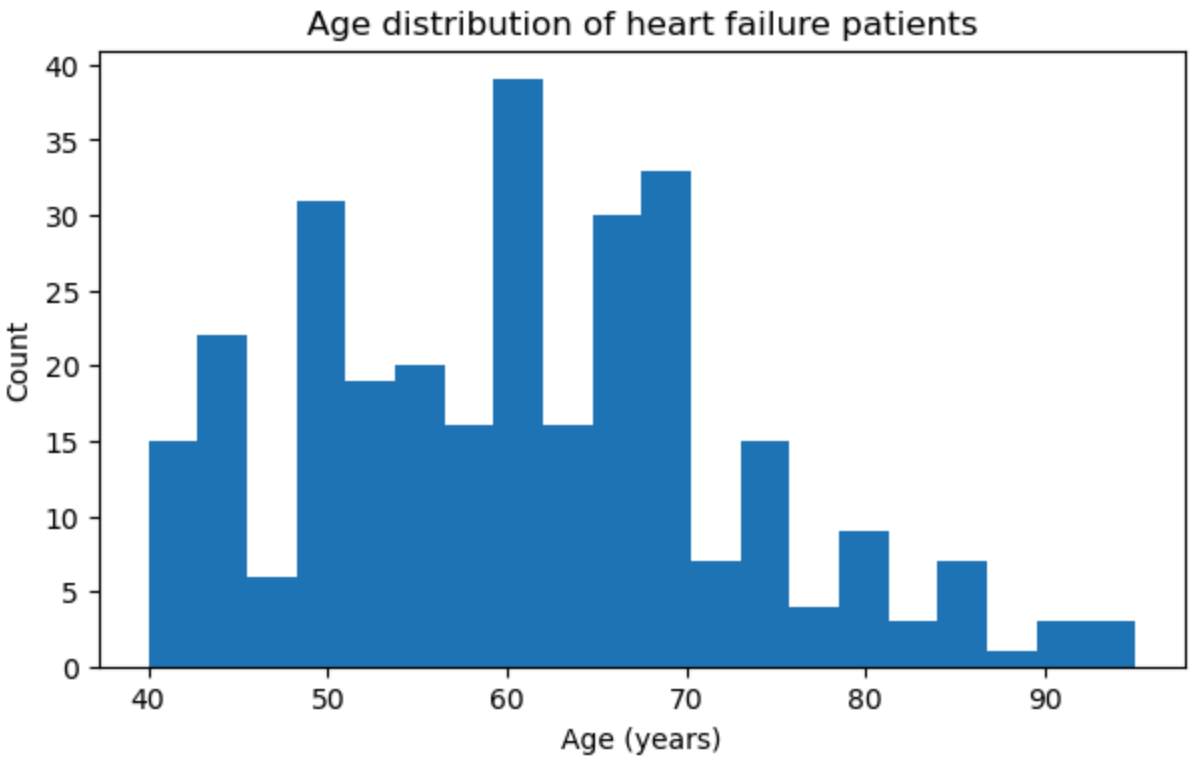
Explanation

We use:

- histogram to see frequency across age
- summary stats (mean/median/min/max) to understand central tendency and spread

In [13]:
```
age = df[age_col]
print(age.describe())

plot_hist(age, bins=20, title="Age distribution of heart failure patients", xl
```

```
count    299.000000
mean      60.833893
std       11.894809
min       40.000000
25%       51.000000
50%       60.000000
75%       70.000000
max       95.000000
Name: age, dtype: float64
```



Q2) How does the death rate vary with age?

Explanation

We create age groups (bins) and compute:

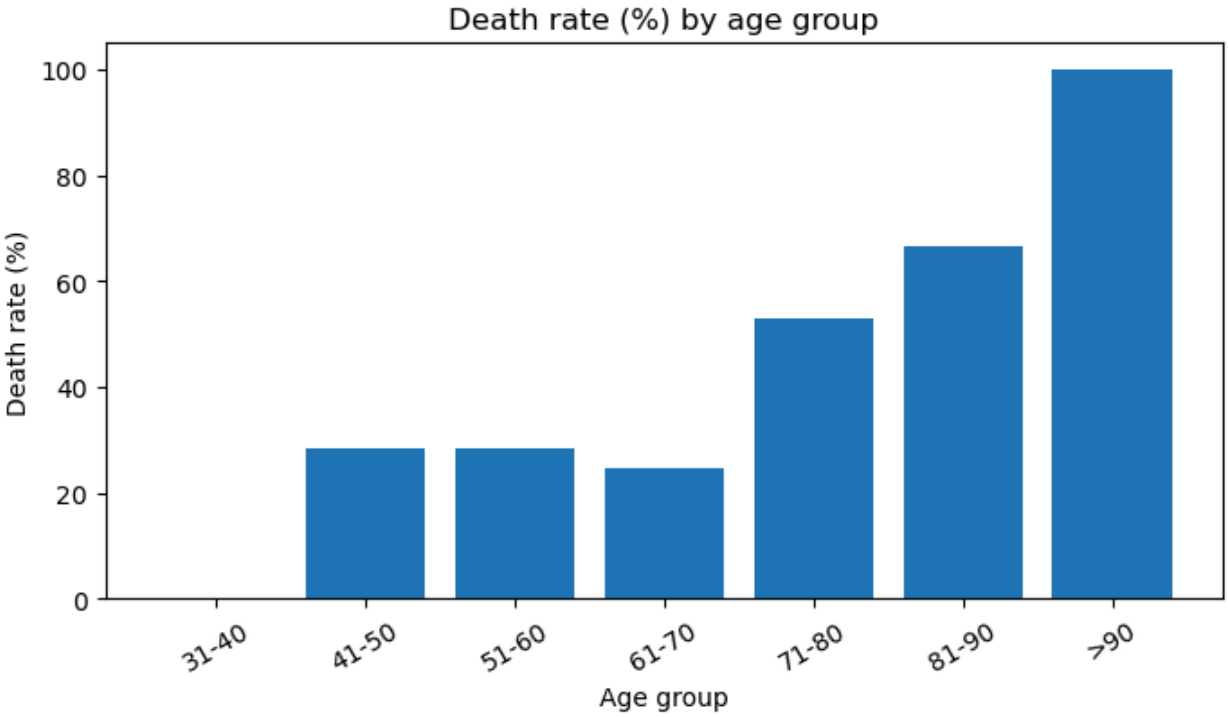
- death rate = mean(death_event) within each group Because `death_event` is binary (0/1), the mean is the percentage who died in that group.

In [14]:
```
# Create age bins (customizable)
age_bins = [0, 30, 40, 50, 60, 70, 80, 90, 200]
age_labels = ["<=30","31-40","41-50","51-60","61-70","71-80","81-90",">90"]

df["age_group"] = pd.cut(df[age_col], bins=age_bins, labels=age_labels, right=

death_rate_by_age = df.groupby("age_group")[death_col].mean() * 100
counts_by_age = df["age_group"].value_counts().reindex(age_labels)

print("Counts by age group:")
print(counts_by_age)
print("\nDeath rate (%) by age group:")
print(death_rate_by_age)

# Plot death rate
plt.figure(figsize=(8,4))
```

```
plt.bar(death_rate_by_age.index.astype(str), death_rate_by_age.values)
plt.title("Death rate (%) by age group")
plt.xlabel("Age group")
plt.ylabel("Death rate (%)")
plt.xticks(rotation=30)
plt.show()
```

```
Counts by age group:
age_group
<=30      0
31-40     7
41-50    67
51-60    88
61-70    85
71-80    34
81-90    15
>90       3
Name: count, dtype: int64

Death rate (%) by age group:
age_group
<=30            NaN
31-40      0.000000
41-50     28.358209
51-60     28.409091
61-70     24.705882
71-80     52.941176
81-90     66.666667
>90      100.000000
Name: death_event, dtype: float64
```

```
/var/folders/7_/1wxc_1j94cbd8y1bkg945yk00000gq/T/ipykernel_5190/810470378.py:7:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current be
havior or observed=True to adopt the future default and silence this warning.
  death_rate_by_age = df.groupby("age_group")[death_col].mean() * 100
```



Q3) What is the percentage of male and female patients in the dataset?

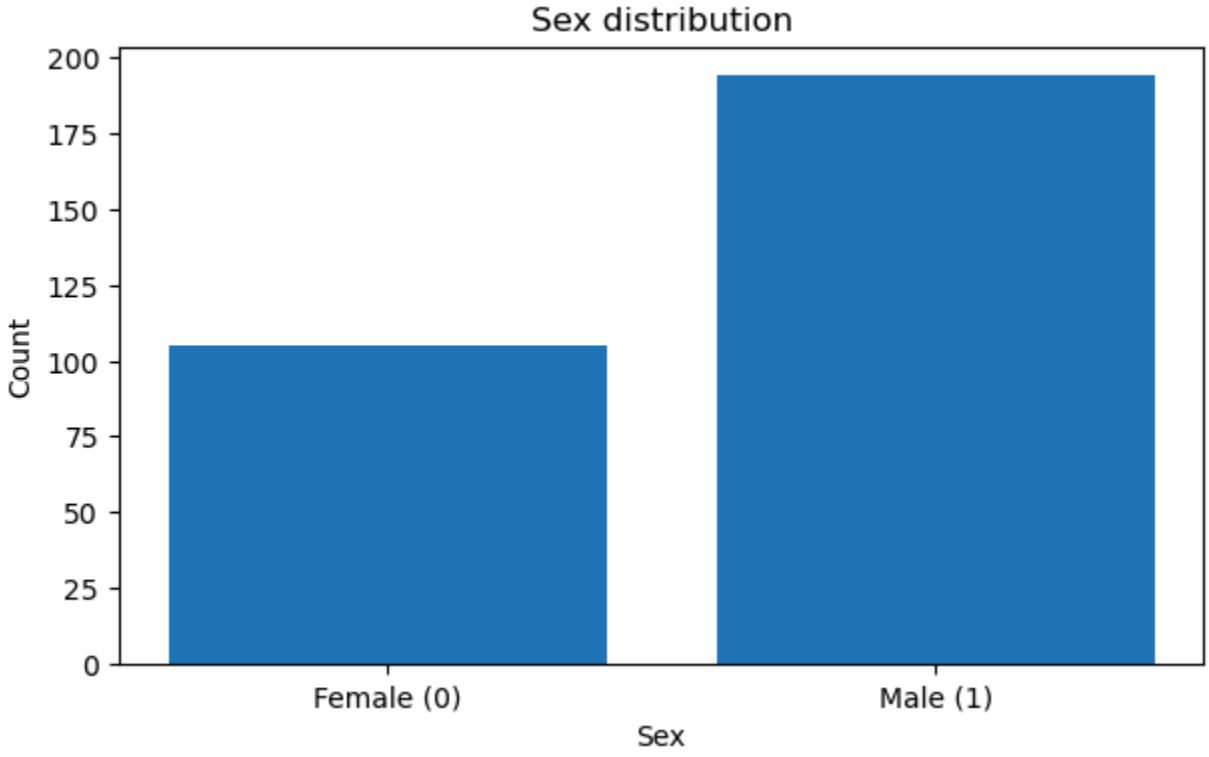
Explanation

`sex` is binary in this dataset:

- 0 = female
- 1 = male We compute counts and percentages.

In [15]:
```
sex_counts = df[sex_col].value_counts().sort_index()
sex_pct = sex_counts / sex_counts.sum() * 100

# Map for display
sex_map = {0: "Female (0)", 1: "Male (1)"}
labels = [sex_map.get(i, str(i)) for i in sex_counts.index]

print("Counts:")
print(sex_counts)
print("\nPercentages (%):")
print(sex_pct.round(2))

plot_bar(labels, sex_counts.values, title="Sex distribution", xlabel="Sex", yl
```

```
Counts:
sex
0    105
1    194
Name: count, dtype: int64

Percentages (%):
sex
0    35.12
1    64.88
Name: count, dtype: float64
```



Q4) How does the platelet count vary among different age groups?

Explanation

We use the same age groups and compare platelet distributions using:

- group-wise summary (mean/median)
- boxplot (shows distribution and outliers)

In [17]:
```
plate_summary = df.groupby("age_group")[platelets_col].agg(["count","mean","me
plate_summary
```

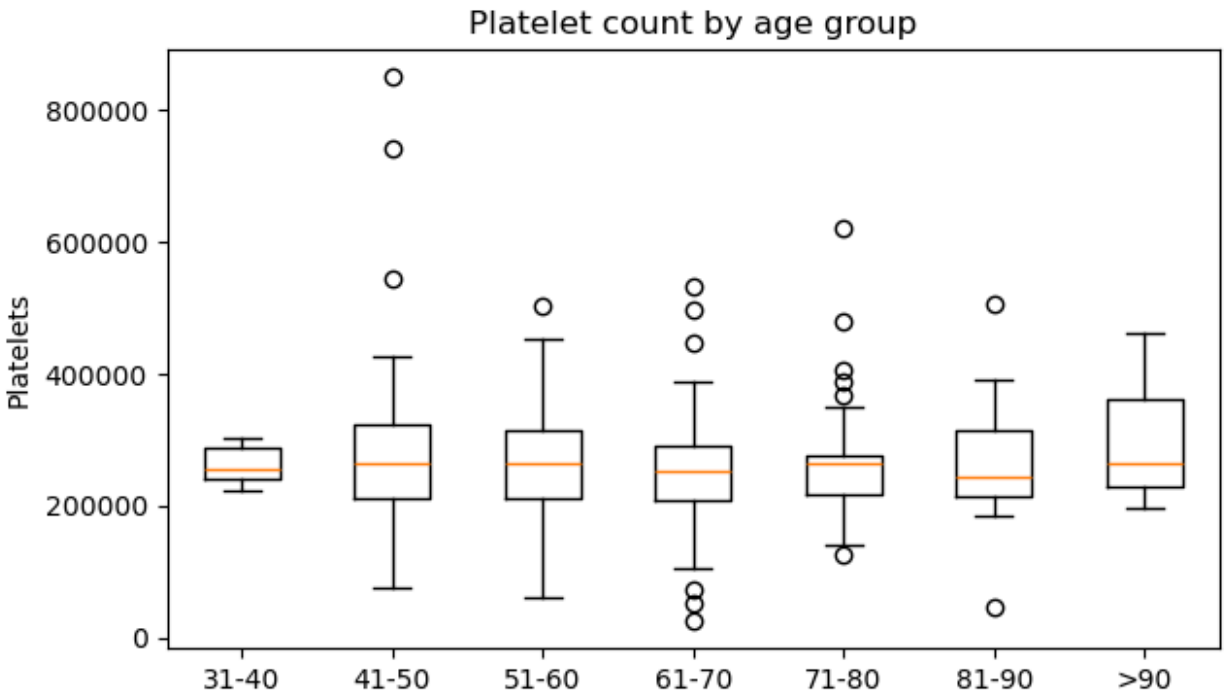
```
/var/folders/7_/1wxc_1j94cbd8y1bkg945yk00000gq/T/ipykernel_5190/1976945682.p
y:1: FutureWarning: The default of observed=False is deprecated and will be cha
nged to True in a future version of pandas. Pass observed=False to retain curre
nt behavior or observed=True to adopt the future default and silence this warni
ng.
  plate_summary = df.groupby("age_group")[platelets_col].agg(["count","mean","m
edian","std"]).round(2)
```

Out[17]:

age_group	count	mean	median	std
<=30	0	NaN	NaN	NaN
31-40	7	262428.57	255000.00	32567.15
41-50	67	277106.69	263358.03	128437.63
51-60	88	261369.39	263358.03	86280.94
61-70	85	251681.06	254000.00	81084.39
71-80	34	266052.65	263358.03	100481.84
81-90	15	265423.87	243000.00	104816.85
>90	3	306786.01	263358.03	137734.32

In [18]:

```
# Boxplot platelets by age group (drop empty groups)
data_list = []
labels = []
for g in age_labels:
    vals = df.loc[df["age_group"] == g, platelets_col].dropna()
    if len(vals) > 0:
        data_list.append(vals.values)
        labels.append(g)

plot_box(data_list, labels, title="Platelet count by age group", ylabel="Plate
```



Q5) Is there a correlation between creatinine and sodium levels in the blood?

Explanation

We analyze:

- Pearson correlation coefficient
- scatter plot

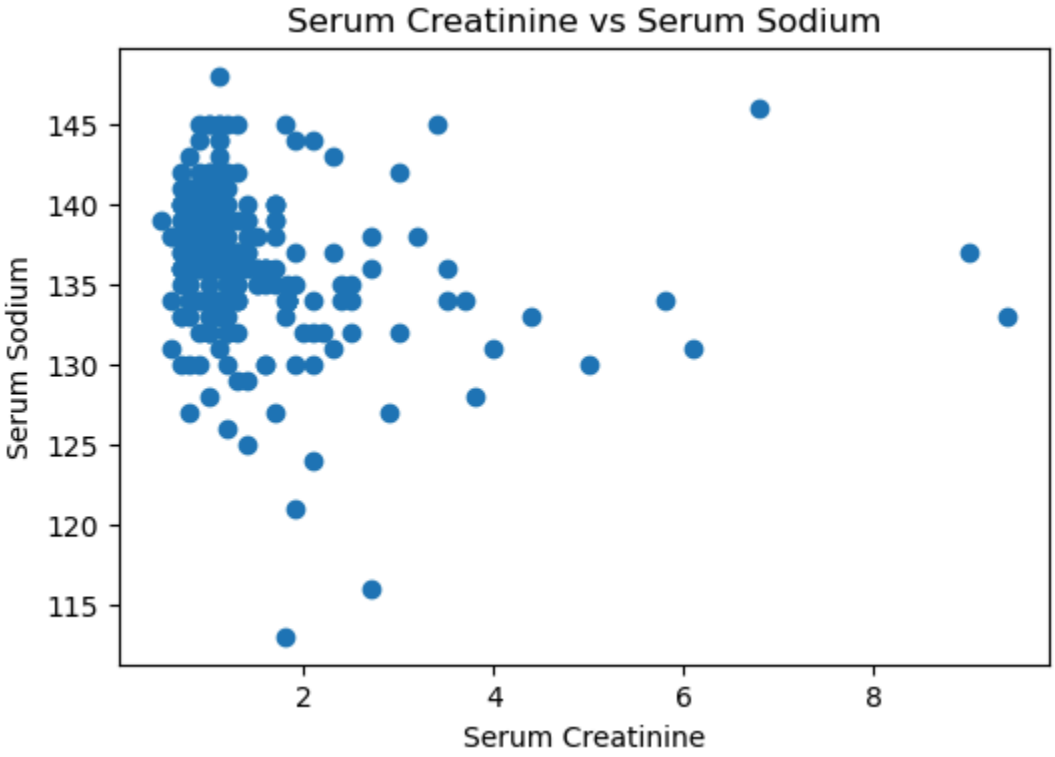
Interpretation:

- correlation close to 0 → weak/no linear relationship
- positive → increase together
- negative → one increases as other decreases

In [19]:
```
x = df[serum_creatinine_col]
y = df[serum_sodium_col]

corr = x.corr(y)  # Pearson
print("Pearson correlation (serum_creatinine vs serum_sodium):", round(corr, 4

plt.figure(figsize=(6,4))
plt.scatter(x, y)
plt.title("Serum Creatinine vs Serum Sodium")
plt.xlabel("Serum Creatinine")
plt.ylabel("Serum Sodium")
```

```
plt.show()
```

```
Pearson correlation (serum_creatinine vs serum_sodium): -0.1891
```



Q6) How does the prevalence of high blood pressure differ between male and female patients?

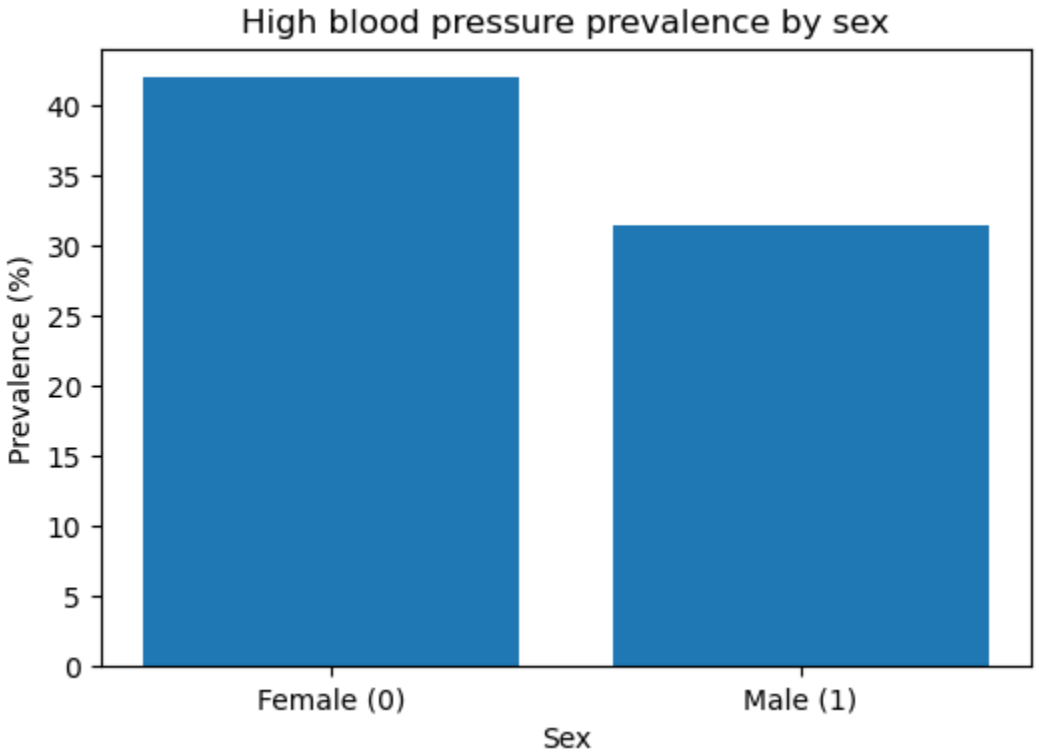
Explanation

We compute prevalence as:

- mean(high_blood_pressure) within each sex group × 100 (percentage)

In [20]:
```
hbp_by_sex = df.groupby(sex_col)[high_bp_col].mean() * 100
hbp_by_sex.index = [sex_map.get(i, str(i)) for i in hbp_by_sex.index]

print("High blood pressure prevalence (%) by sex:")
print(hbp_by_sex.round(2))

plt.figure(figsize=(6,4))
plt.bar(hbp_by_sex.index.astype(str), hbp_by_sex.values)
plt.title("High blood pressure prevalence by sex")
plt.xlabel("Sex")
plt.ylabel("Prevalence (%)")
plt.show()
```

```
High blood pressure prevalence (%) by sex:
Female (0)    41.90
Male (1)      31.44
Name: high_blood_pressure, dtype: float64
```



Q7) What is the relationship between smoking habits and the occurrence of heart failure?

Important Note (Dataset context)

This dataset contains **only patients diagnosed with heart failure** (all 299 are heart failure patients).
So we cannot measure *occurrence of heart failure vs non-heart-failure* from this dataset alone.

What we can analyze instead (meaningful inside this dataset):

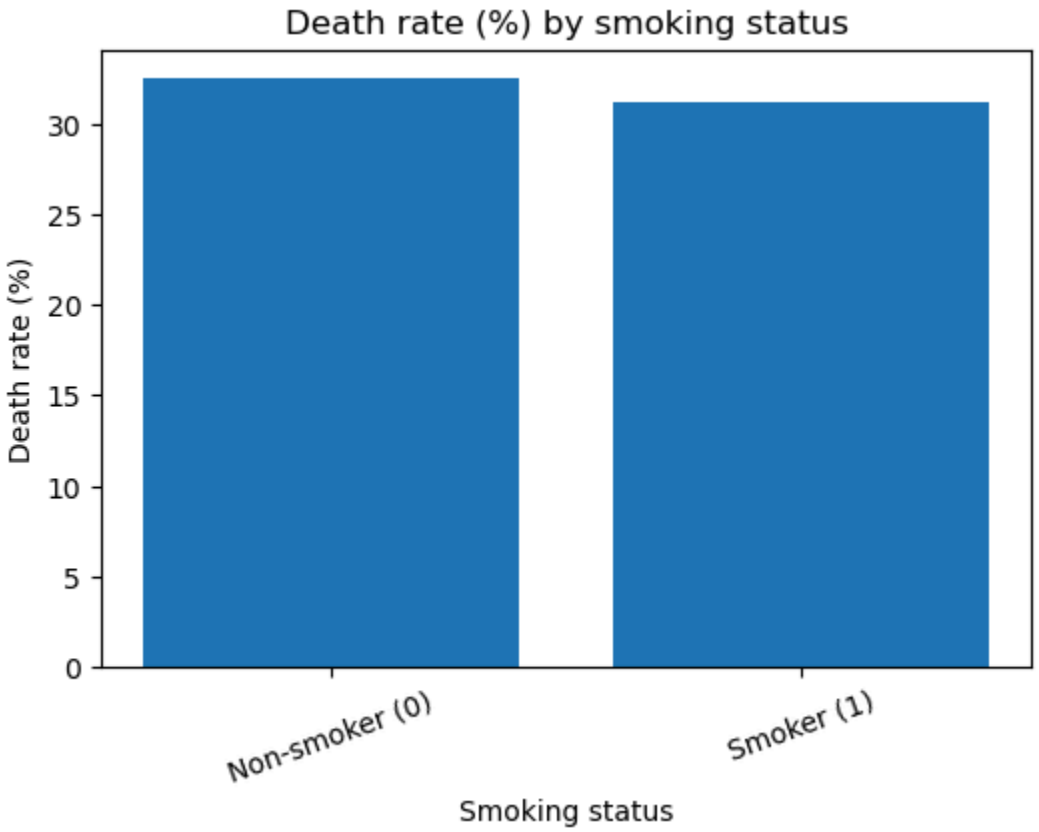
- Whether **smoking is associated with worse outcomes**, e.g. **death_event**
- Compare death rate between smokers and non-smokers

We'll compute death rate by smoking group.

In [21]:
```
smoke_counts = df[smoking_col].value_counts().sort_index()
smoke_rate_death = df.groupby(smoking_col)[death_col].mean() * 100
```

```
smoke_map = {0: "Non-smoker (0)", 1: "Smoker (1)"}
print("Counts by smoking:")
print(smoke_counts)
print("\nDeath rate (%) by smoking:")
print(smoke_rate_death.round(2))

labels = [smoke_map.get(i, str(i)) for i in smoke_rate_death.index]
plt.figure(figsize=(6,4))
plt.bar(labels, smoke_rate_death.values)
plt.title("Death rate (%) by smoking status")
plt.xlabel("Smoking status")
plt.ylabel("Death rate (%)")
plt.xticks(rotation=20)
plt.show()
```

```
Counts by smoking:
smoking
0    203
1     96
Name: count, dtype: int64

Death rate (%) by smoking:
smoking
0    32.51
1    31.25
Name: death_event, dtype: float64
```



Q8) Are there any noticeable patterns in the distribution of death events across different age groups?

Explanation

We visualize:

- counts of death_event=0 and death_event=1 per age group This shows which age groups have higher death counts and higher rates.

In [22]:
```
# Count death events by age group
death_counts_age = df.pivot_table(index="age_group", columns=death_col, values
death_counts_age.columns = ["Survived (0)", "Died (1)"] if set(death_counts_ag
death_counts_age = death_counts_age.reindex(age_labels)
death_counts_age
```

```
/var/folders/7_/1wxc_1j94cbd8y1bkg945yk00000gq/T/ipykernel_5190/4183060845.p
y:2: FutureWarning: The default value of observed=False is deprecated and will
change to observed=True in a future version of pandas. Specify observed=False t
o silence this warning and retain the current behavior
  death_counts_age = df.pivot_table(index="age_group", columns=death_col, value
s=age_col, aggfunc="count").fillna(0)
```

Out[22]:

age_group	Survived (0)	Died (1)
<=30	0	0
31-40	7	0
41-50	48	19
51-60	63	25
61-70	64	21
71-80	16	18
81-90	5	10
>90	0	3

In [ ]:
```
# Stacked bar plot
plt.figure(figsize=(9,4))
surv = death_counts_age["Survived (0)"].values
died = death_counts_age["Died (1)"].values

plt.bar(death_counts_age.index.astype(str), surv, label="Survived (0)")
plt.bar(death_counts_age.index.astype(str), died, bottom=surv, label="Died (1)
plt.title("Death event distribution by age group")
plt.xlabel("Age group")
```

```
plt.ylabel("Number of patients")
plt.xticks(rotation=30)
plt.legend()
plt.show()
```

Q9) Is there any significant difference in ejection fraction between patients with and without diabetes?

Explanation

We compare `ejection_fraction` between:

- diabetes = 0 (no diabetes)
- diabetes = 1 (has diabetes)

We show:

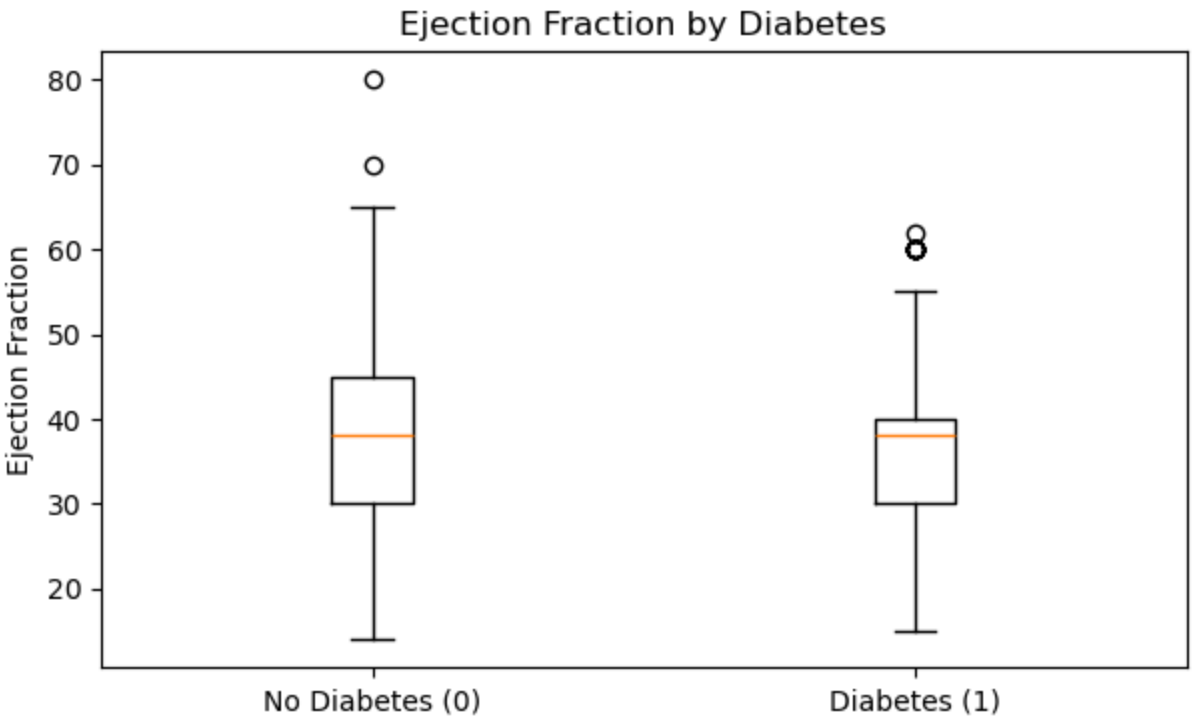
- group-wise statistics
- boxplot (Optional) quick t-test if needed (not mandatory for EDA, but useful).

In [23]:
```
ef_by_diabetes = df.groupby(diabetes_col)[ejection_col].agg(["count","mean","m
ef_by_diabetes.index = ["No Diabetes (0)" if i==0 else "Diabetes (1)" for i in
ef_by_diabetes
```

Out[23]:

	count	mean	median	std
No Diabetes (0)	174	38.13	38.0	12.25
Diabetes (1)	125	38.02	38.0	11.28

In [24]:
```
ef0 = df.loc[df[diabetes_col] == 0, ejection_col].dropna()
ef1 = df.loc[df[diabetes_col] == 1, ejection_col].dropna()

plot_box([ef0.values, ef1.values], ["No Diabetes (0)", "Diabetes (1)"], title=
```



Q10) How does the serum creatinine level vary between patients who survived and those who did not?

Explanation

We compare serum creatinine by `death_event`:

- death_event = 0 → survived
- death_event = 1 → died

We show:

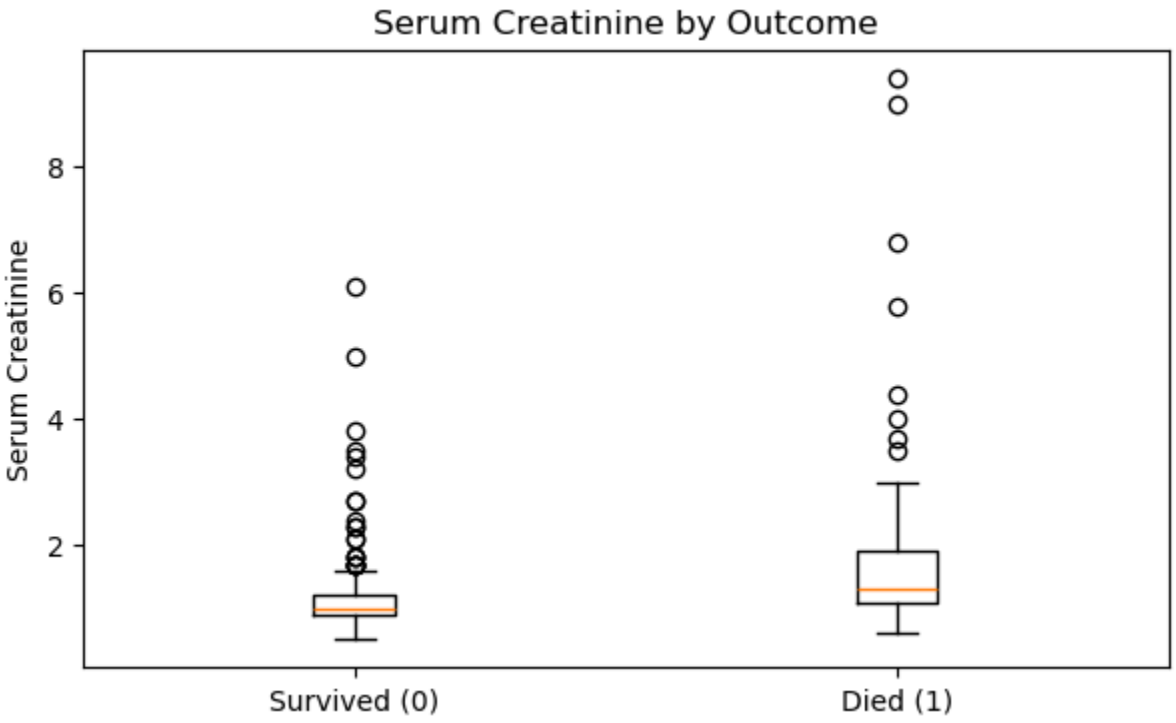
- group summary stats
- boxplot

In [25]:
```
scr_by_outcome = df.groupby(death_col)[serum_creatinine_col].agg(["count","mea
scr_by_outcome.index = ["Survived (0)" if i==0 else "Died (1)" for i in scr_by
scr_by_outcome
```

Out[25]:

	count	mean	median	std
Survived (0)	203	1.185	1.0	0.654
Died (1)	96	1.836	1.3	1.469

In [26]:
```
scr0 = df.loc[df[death_col] == 0, serum_creatinine_col].dropna()
scr1 = df.loc[df[death_col] == 1, serum_creatinine_col].dropna()

plot_box([scr0.values, scr1.values], ["Survived (0)", "Died (1)"], title="Seru
```



Final Summary (EDA Insights Checklist)

In this notebook, we:

- loaded the Heart Failure dataset (299 rows, 13 features typically)
- checked missing values and data types
- answered all EDA questions from the assignment:
  - age distribution
  - death rate vs age
  - sex percentage
  - platelets by age group
  - creatinine vs sodium correlation
  - high BP prevalence by sex
  - smoking vs death outcome (since all records are heart-failure patients)
  - death pattern across age groups
  - ejection fraction diabetes vs non-diabetes
  - serum creatinine survived vs died

If your instructor expects additional visuals (pairplot, full correlation heatmap, etc.), you can extend this notebook easily.