Deliverable 1 — Data Collection, Cleaning, and Exploration (Organ Donation)

Course: MSCS634 - Advanced Big Data and Data Mining
Author: Samrat Baral

This notebook is designed to provide a comprehensive experience in data mining by guiding you through key stages of the data mining process for **Deliverable 1**. You will explore **data preprocessing** and **EDA**, generating insights that will guide **feature engineering, regression, classification, clustering, and association rule mining** in subsequent deliverables.

0. Setup

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (8, 5)
plt.rcParams['axes.grid'] = True

DATA_DIR = '../data'
CSV_CANDIDATES = ['organ_donation.csv', 'OrganDonation.csv', 'organ_dona
DATA_PATH = None
for name in CSV_CANDIDATES:
    candidate = os.path.join(DATA_DIR, name)
    if os.path.exists(candidate):
        DATA_PATH = candidate
        break

SAVE_FIGS = True
FIG_DIR = '../figures'
os.makedirs(FIG_DIR, exist_ok=True)

def savefig(name: str):
    if SAVE_FIGS:
        path = os.path.join(FIG_DIR, name)
        plt.savefig(path, bbox_inches='tight', dpi=150)
        print(f"Saved figure → {path}")
```

1. Dataset Selection, Description, and Justification

Chosen Dataset: Organ Donation and Transplantation Dataset
Source: Kaggle — *Organ Donation and Transplantation (HHS/OPTN derived)*

Granularity: One row represents a **single transplant case** (donor-recipient-organ with metadata).
Candidate Attributes (≥10):

- `Donor_ID`, `Recipient_ID`, `Organ_Type`, `Blood_Type`, `Donor_Age`, `Recipient_Age`, `Donor_Gender`, `Recipient_Gender`, `Transplant_Date`, `Outcome`, `Wait_Time_Days`

Why appropriate (Justification):

- Meets size/attributes criteria (≥500 rows, ≥10 features) in typical releases.
- Mix of **numeric & categorical** features enables full pipeline: preprocessing, feature engineering, **classification** (e.g., `Outcome`), **regression** (`Wait_Time_Days`), **clustering** (recipient profiles), and **association rules** (e.g., `Organ_Type` × `Blood_Type` patterns).
- **Healthcare relevance** with ethical public data supports meaningful insights and policy-relevant analysis.

2. Load Data

```
    if DATA_PATH is not None:
        print(f"Loading CSV from: {DATA_PATH}")
        df = pd.read_csv(DATA_PATH)
    else:
        print("No CSV found in ../data. Using a synthetic sample (50 rows) s
        rng = np.random.default_rng(42)
        n = 50
        organs = np.array(['Kidney','Liver','Heart','Lung'])
        bloods = np.array(['A','B','AB','O'])
        genders = np.array(['Male','Female'])
        outcomes = np.array(['Success','Failure'])
        df = pd.DataFrame({
            'Donor_ID': np.arange(1000, 1000+n),
            'Recipient_ID': np.arange(5000, 5000+n),
            'Organ_Type': rng.choice(organs, size=n),
            'Blood_Type': rng.choice(bloods, size=n),
            'Donor_Age': rng.integers(18, 70, size=n),
            'Recipient_Age': rng.integers(1, 80, size=n),
            'Donor_Gender': rng.choice(genders, size=n),
            'Recipient_Gender': rng.choice(genders, size=n),
            'Transplant_Date': pd.to_datetime('2018-01-01') + pd.to_timedelt
            'Outcome': rng.choice(outcomes, size=n, p=[0.8, 0.2]),
            'Wait_Time_Days': rng.integers(1, 1200, size=n).astype(float)
        })
        # introduce some missingness & duplicates deliberately for cleaning
        df.loc[rng.integers(0, n, size=3), 'Wait_Time_Days'] = np.nan
        df.loc[rng.integers(0, n, size=2), 'Blood_Type'] = None
        df = pd.concat([df, df.iloc[[0]]], ignore_index=True)  # one duplica

    print("Shape:", df.shape)
    display(df.head())
    display(df.sample(5, random_state=42))
    df.info()
```

```
No CSV found in ../data. Using a synthetic sample (50 rows) so the notebo
Shape: (51, 11)
```

	Donor_ID	Recipient_ID	Organ_Type	Blood_Type	Donor_Age	Recipient_Ag
0	1000	5000	Kidney	O	61	6
1	1001	5001	Lung	A	28	2
2	1002	5002	Heart	B	59	7
3	1003	5003	Liver	B	18	2
4	1004	5004	Liver	B	59	3

	Donor_ID	Recipient_ID	Organ_Type	Blood_Type	Donor_Age	Recipient_A
43	1043	5043	Liver	None	39	
40	1040	5040	Kidney	B	69	
46	1046	5046	Liver	A	19	
12	1012	5012	Heart	B	46	
24	1024	5024	Lung	AB	62	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Donor_ID          51 non-null     int64
 1   Recipient_ID      51 non-null     int64
 2   Organ_Type        51 non-null     object
 3   Blood_Type        49 non-null     object
 4   Donor_Age         51 non-null     int64
 5   Recipient_Age     51 non-null     int64
 6   Donor_Gender      51 non-null     object
 7   Recipient_Gender  51 non-null     object
 8   Transplant_Date   51 non-null     datetime64[ns]
 9   Outcome           51 non-null     object
 10  Wait_Time_Days    48 non-null     float64
dtypes: datetime64[ns](1), float64(1), int64(4), object(5)
memory usage: 4.5+ KB
```

3. Initial Inspection & Data Quality

```
# Numerical and categorical summaries
display(df.describe(include='number').T)
display(df.describe(include='object').T)

# Missingness overview
missing_counts = df.isna().sum().sort_values(ascending=False)
```

```
missing_pct = (missing_counts / len(df) * 100).round(2)
missing = pd.DataFrame({'missing_count': missing_counts, 'missing_pct':
print("Missingness (top 20):")
display(missing.head(20))

# Duplicate rows
dup_count = df.duplicated().sum()
print(f"Duplicate rows: {dup_count}")
```

	count	mean	std	min	25%	50%	75%
Donor_ID	51.0	1024.019608	14.833058	1000.0	1011.5	1024.0	1036.50
Recipient_ID	51.0	5024.019608	14.833058	5000.0	5011.5	5024.0	5036.50
Donor_Age	51.0	43.431373	14.721759	18.0	31.0	46.0	57.50
Recipient_Age	51.0	39.843137	21.853487	2.0	24.0	37.0	56.50
Wait_Time_Days	48.0	642.416667	355.820528	8.0	347.5	715.5	944.75

	count	unique	top	freq
Organ_Type	51	4	Lung	16
Blood_Type	49	4	B	15
Donor_Gender	51	2	Male	27
Recipient_Gender	51	2	Male	31
Outcome	51	2	Success	40

```
Missingness (top 20):
```

	missing_count	missing_pct
Wait_Time_Days	3	5.88
Blood_Type	2	3.92
Donor_ID	0	0.00
Organ_Type	0	0.00
Recipient_ID	0	0.00
Recipient_Age	0	0.00
Donor_Age	0	0.00
Donor_Gender	0	0.00
Recipient_Gender	0	0.00
Transplant_Date	0	0.00
Outcome	0	0.00

```
Duplicate rows: 1
```

4. Data Cleaning (with explanations)

Planned actions & rationale:

- **Type fixes:** Parse `Transplant_Date` as datetime; ensure numeric columns are numeric.
- **Duplicates:** Remove exact duplicates to avoid bias and leakage.
- **Missing values:** Median-impute numeric (robust to outliers), mode-impute categorical; log decisions.
- **Categorical normalization:** Trim whitespace, unify cases to reduce spurious levels.

```
df_clean = df.copy()

# 4.1 Type fixes
if 'Transplant_Date' in df_clean.columns:
    df_clean['Transplant_Date'] = pd.to_datetime(df_clean['Transplant_Da

# Example: ensure numeric
if 'Wait_Time_Days' in df_clean.columns:
    df_clean['Wait_Time_Days'] = pd.to_numeric(df_clean['Wait_Time_Days'

# 4.2 Normalize categoricals
for col in df_clean.select_dtypes(include='object').columns:
    df_clean[col] = df_clean[col].astype(str).str.strip()

# 4.3 Remove exact duplicates
before = len(df_clean)
df_clean = df_clean.drop_duplicates()
print(f"Removed {before - len(df_clean)} duplicate rows")

# 4.4 Missing handling
num_cols = df_clean.select_dtypes(include=['number']).columns.tolist()
cat_cols = df_clean.select_dtypes(exclude=['number','datetime64[ns]']).c

impute_log = []

for col in num_cols:
    if df_clean[col].isna().any():
        med = df_clean[col].median()
        df_clean[col] = df_clean[col].fillna(med)
        impute_log.append(f"Numeric imputed median for {col} → {med}")

for col in cat_cols:
    if df_clean[col].isna().any():
        mode = df_clean[col].mode(dropna=True)
        if len(mode):
            df_clean[col] = df_clean[col].fillna(mode.iloc[0])
            impute_log.append(f"Categorical imputed mode for {col} → {mo

print("\nImputation decisions:")
for line in impute_log:
    print(" -", line)

print("\nMissing after imputation (top 10):")
display(df_clean.isna().sum().sort_values(ascending=False).head(10))

print("New shape:", df_clean.shape)
```

```
Removed 1 duplicate rows

Imputation decisions:
 - Numeric imputed median for Wait_Time_Days → 715.0

Missing after imputation (top 10):
```

	0
Donor_ID	0
Recipient_ID	0
Organ_Type	0
Blood_Type	0
Donor_Age	0
Recipient_Age	0
Donor_Gender	0
Recipient_Gender	0
Transplant_Date	0
Outcome	0

dtype: int64

```
New shape: (50, 11)
```

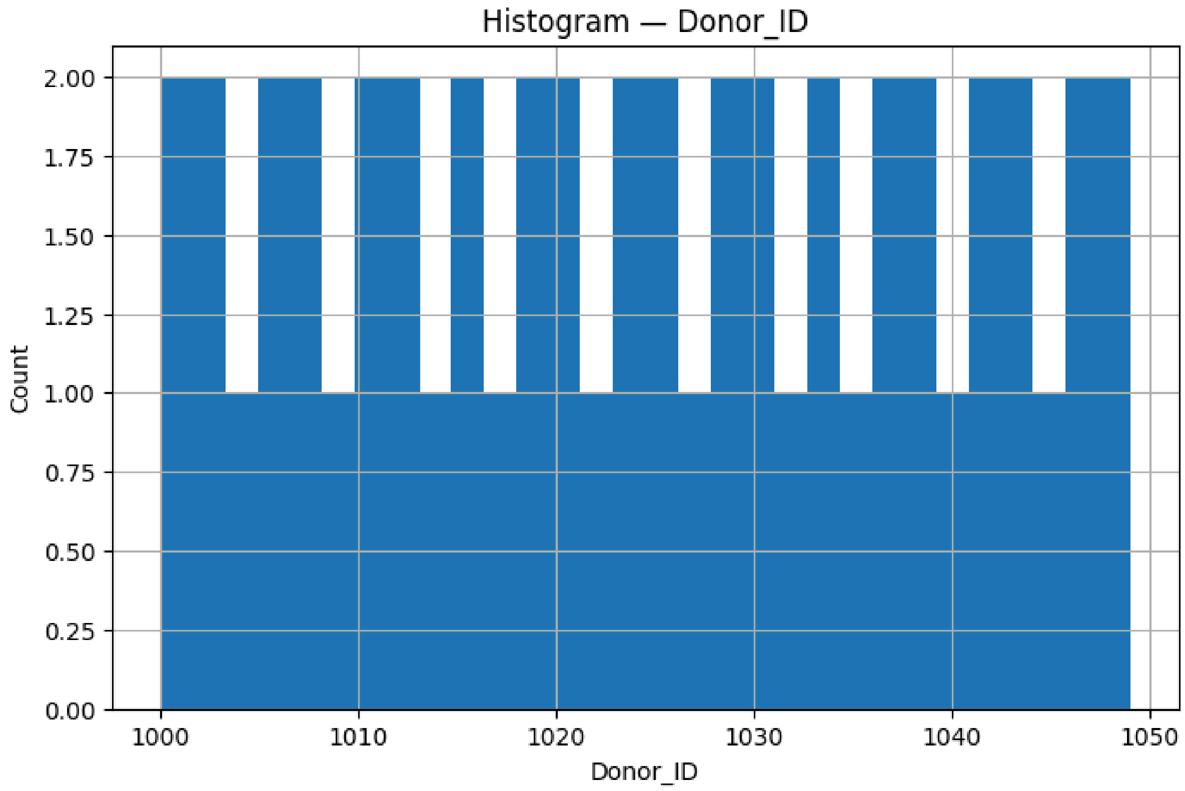
5. Exploratory Data Analysis (EDA)

5.1 Numeric Distributions

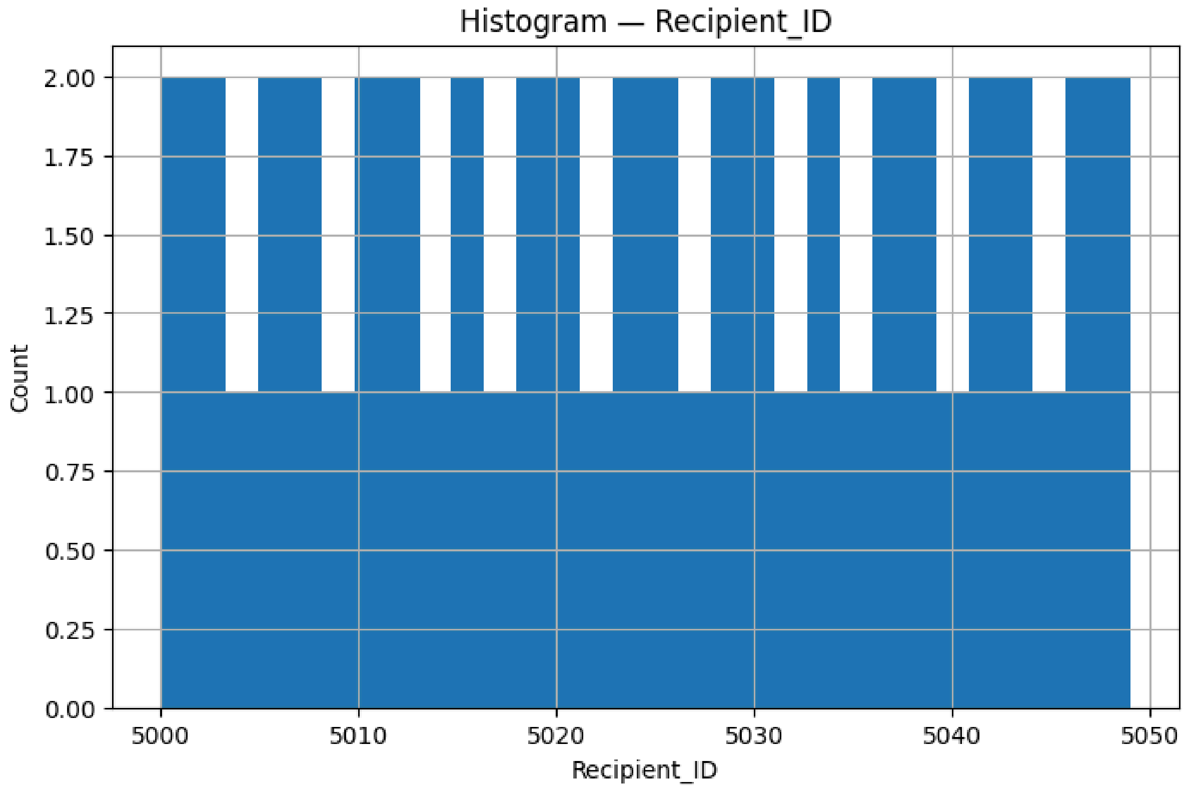
```
numeric_cols = df_clean.select_dtypes(include='number').columns.tolist()
for col in numeric_cols[:12]:
    plt.figure()
```

```
        plt.hist(df_clean[col].dropna(), bins=30)
        plt.title(f"Histogram — {col}")
        plt.xlabel(col); plt.ylabel("Count")
        savefig(f"hist_{col}.png")
        plt.show()
```

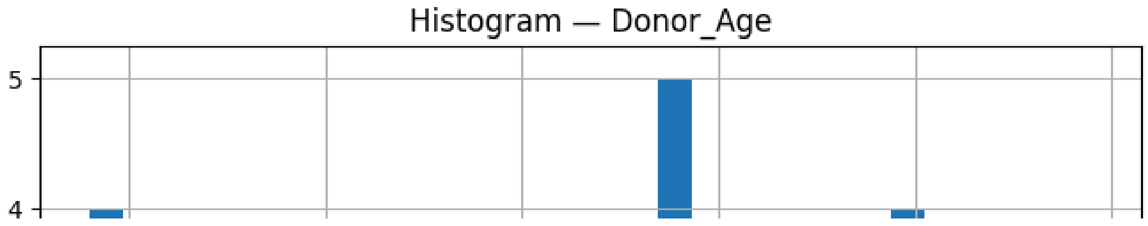
```
Saved figure → ../figures/hist_Donor_ID.png
```

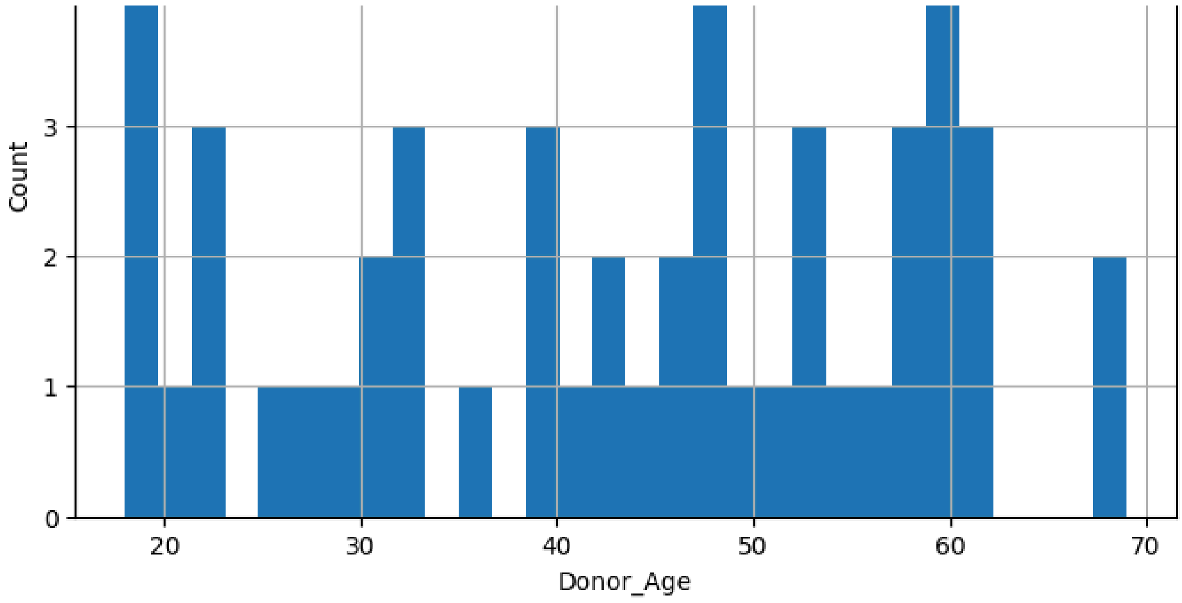


```
Saved figure → ../figures/hist_Recipient_ID.png
```

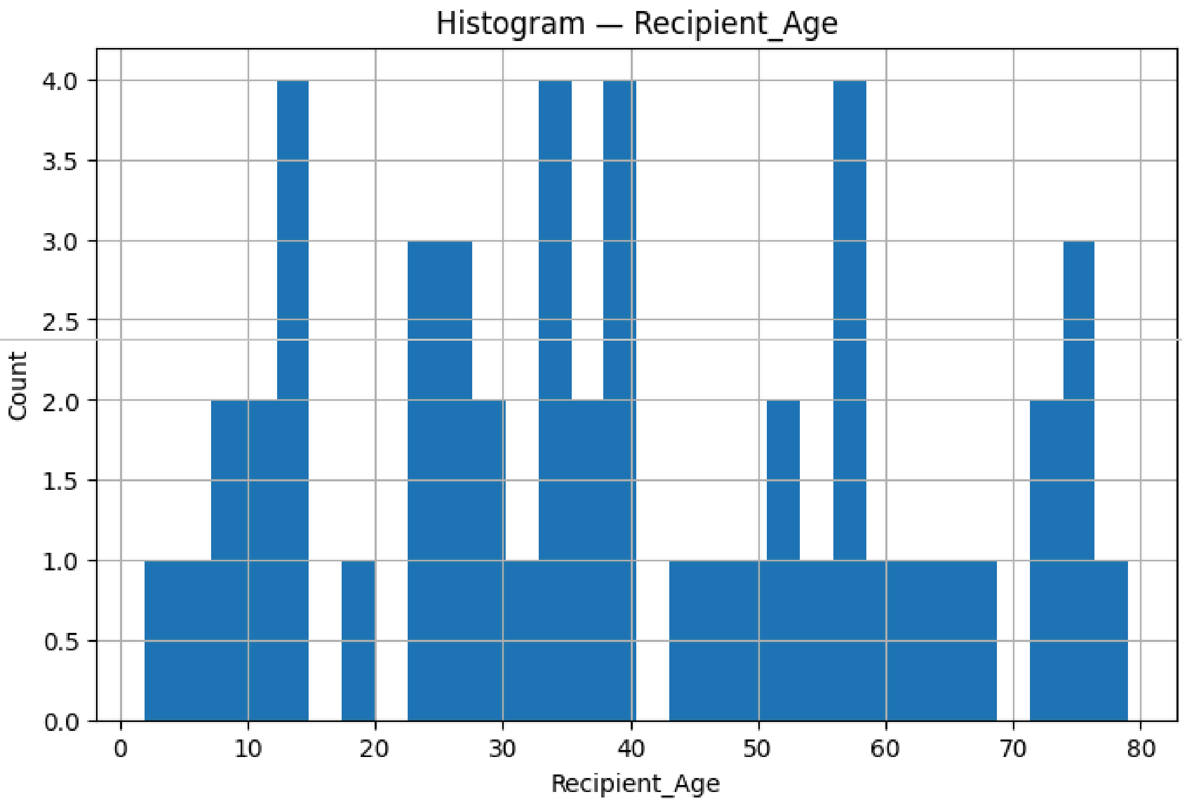


```
Saved figure → ../figures/hist_Donor_Age.png
```

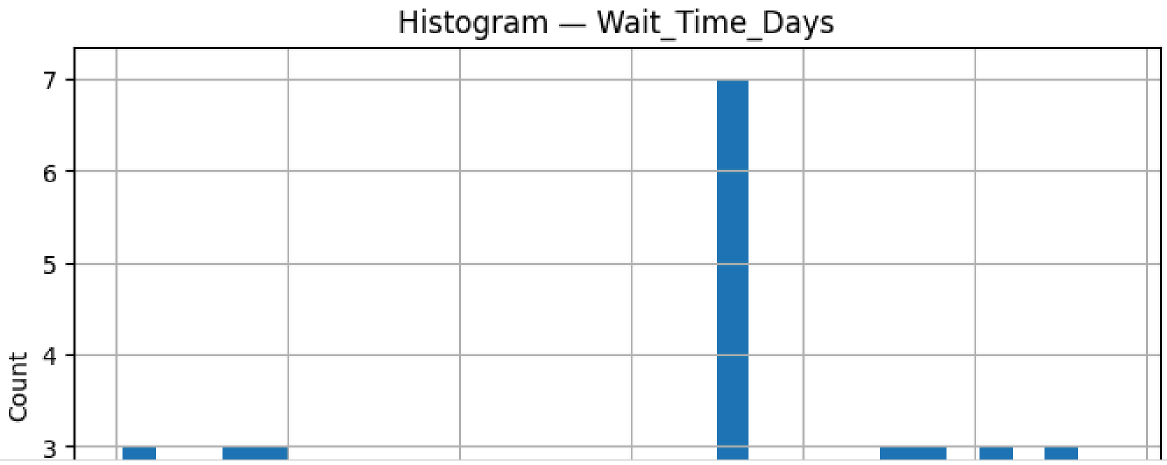


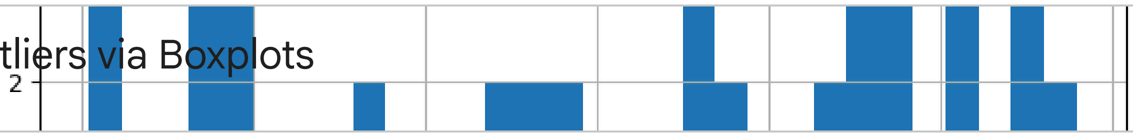


```
Saved figure → ../figures/hist_Recipient_Age.png
```



```
Saved figure → ../figures/hist_Wait_Time_Days.png
```





5.2 Outliers via Boxplots

```
for col in numeric_cols[:12]:
    plt.figure()
    plt.boxplot(df_clean[col].dropna(), vert=True, whis=1.5)
    plt.title(f"Boxplot — {col}")
    savefig(f"box_{col}.png")
    plt.show()
```