Phase 1

1. Dataset Loading and Initial Exploration

- Import libraries
- Load dataset from given URL
- Display first few rows (head)

```
import pandas as pd

# Load dataset from URL
url = "https://raw.githubusercontent.com/salemprakash/EDA/main/Data/SuicideChina.csv"
df = pd.read_csv(url)

# Preview data
df.head()
```

	rownames	Person_ID	Hospitalised	Died	Urban	Year	Month	Sex	Age	Education	Occupation	method
0	1	1	yes	no	no	2010	12	female	39	Secondary	household	Other poison
1	2	2	no	yes	no	2009	3	male	83	primary	farming	Hanging
2	3	3	no	yes	no	2010	2	male	60	primary	farming	Hanging
3	4	4	no	yes	no	2011	1	male	73	primary	farming	Hanging
4	5	5	yes	no	no	2009	8	male	51	Secondary	farming	Pesticide

Next steps: Generate code with df New interactive sheet

2. Data Summary and Metadata

- Data types of each column
- Summary statistics (numerical + categorical)

```
# Dataset shape
print("Dataset Shape:", df.shape)

# Columns
print("\nColumn Names:\n", df.columns.tolist())

# Data Types
print("\nData Types:\n", df.dtypes)
```

```
Dataset Shape: (2571, 12)

Column Names:
 ['rownames', 'Person_ID', 'Hospitalised', 'Died', 'Urban', 'Year', 'Month', 'Sex', 'Age', 'Education', 'Occupation', 'method']

Data Types:
 rownames         int64
Person_ID        int64
Hospitalised    object
Died            object
Urban           object
Year             int64
Month            int64
Sex             object
Age              int64
Education       object
Occupation      object
method          object
dtype: object
```

```
# Numeric summary
df.describe()

# Categorical summary
df.describe(include='object')
```

	Hospitalised	Died	Urban	Sex	Education	Occupation	method
count	2571	2571	2571	2571	2571	2571	2571
unique	2	2	3	2	5	10	9
top	yes	no	no	female	Secondary	farming	Pesticide
freq	1553	1315	2213	1328	1280	2032	1768

3. Data Cleaning and Handling

- Missing values check & handling
- Duplicate records check

```
# Missing values
print(df.isnull().sum())

# Check for duplicates
print("Duplicate Rows:", df.duplicated().sum())
```

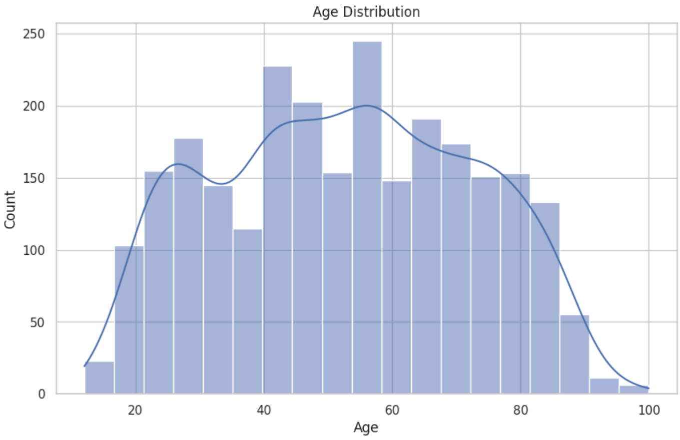
```
rownames        0
Person_ID       0
Hospitalised    0
Died            0
Urban           0
Year            0
Month           0
Sex             0
Age             0
Education       0
Occupation      0
method          0
dtype: int64
Duplicate Rows: 0
```

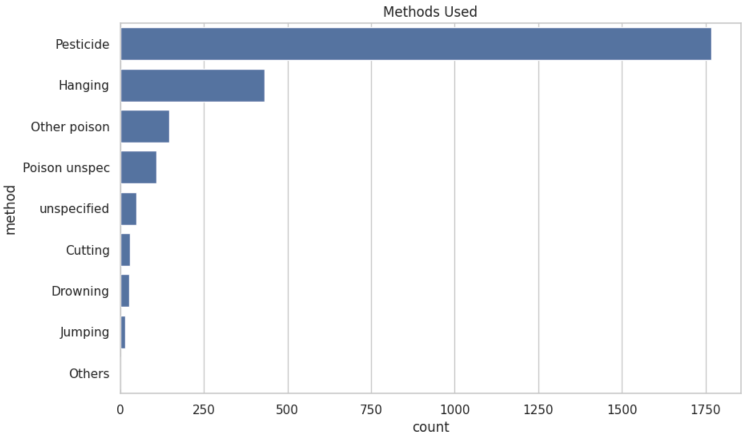
```
# Remove duplicates
df = df.drop_duplicates()

# Example: Convert 'Year' and 'Month' to string
df['Year'] = df['Year'].astype(str)
df['Month'] = df['Month'].astype(str)
```

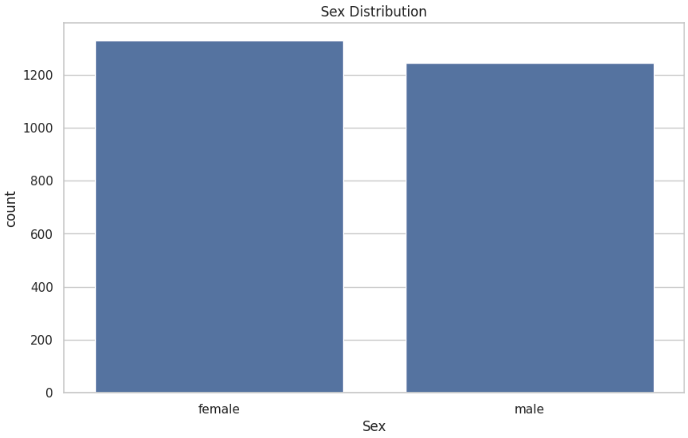
4. Univariate Analysis

- Numerical variables: histograms, density plots
- Categorical variables: value counts, bar plots, pie charts
- Insights on distribution, outliers, skewness

```
#Univariate Analysis
import seaborn as sns
import matplotlib.pyplot as plt

# Age Distribution
sns.histplot(df['Age'], kde=True)
plt.title("Age Distribution")
plt.show()

# Method Count
sns.countplot(data=df, y='method', order=df['method'].value_counts().index)
plt.title("Methods Used")
plt.show()

# Sex Distribution
sns.countplot(data=df, x='Sex')
plt.title("Sex Distribution")
plt.show()
```



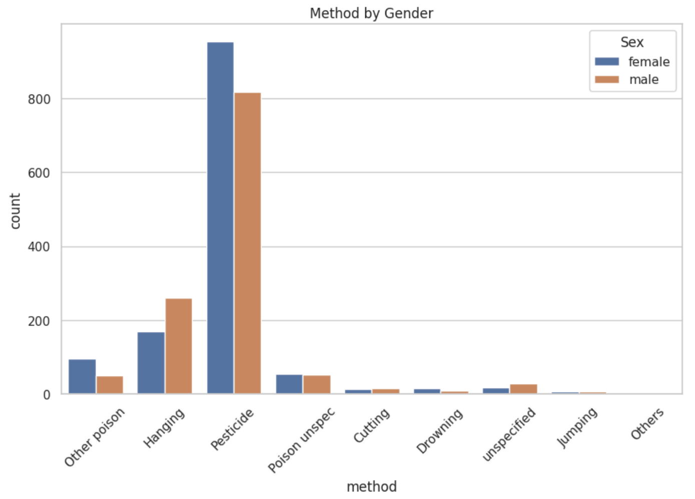


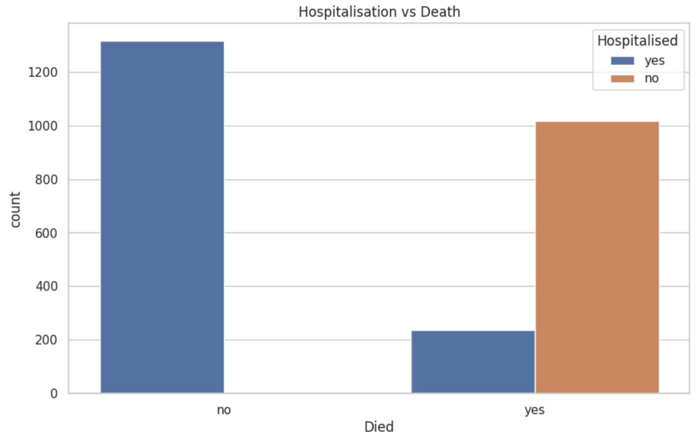


5. Bivariate Analysis

- Numerical vs Numerical
- Numerical vs Categorical

```
# Bivariate Analysis
# Method vs Sex
sns.countplot(data=df, x='method', hue='Sex')
plt.xticks(rotation=45)
plt.title("Method by Gender")
plt.show()

# Hospitalised vs Died
sns.countplot(data=df, hue='Hospitalised', x='Died')
plt.title("Hospitalisation vs Death")
plt.show()
```

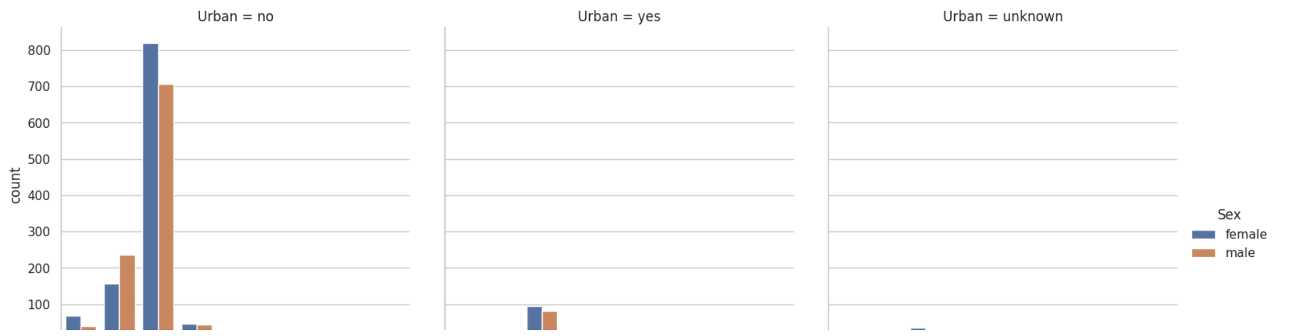




6. Multivariate Analysis

- Pairplots / heatmaps
- Interaction effects among 3 or more variables
- Key insights

```
#Multivariate Analysis

# Age Group
df['AgeGroup'] = pd.cut(df['Age'], bins=[0, 18, 35, 50, 65, 80, 100],
                        labels=['<18','18-35','36-50','51-65','66-80','80+'])

# Multivariate: Method vs Sex vs Urban
sns.catplot(data=df, x='method', hue='Sex', col='Urban', kind='count', height=5, aspect=1)
plt.xticks(rotation=90)
plt.show()
```

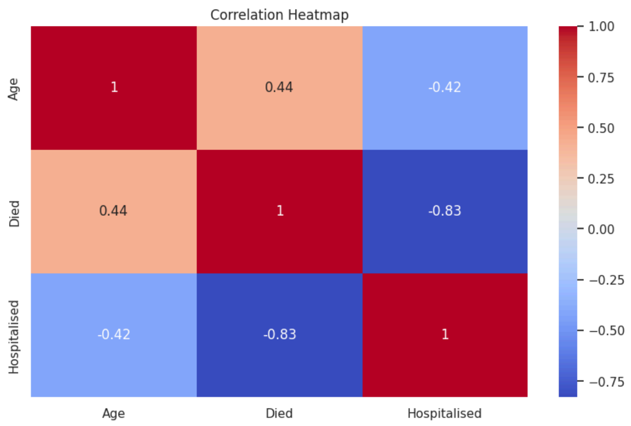


```
#Value count for additional columns
print("Sex:\n", df['Sex'].value_counts())
print("\nEducation:\n", df['Education'].value_counts())
print("\nOccupation:\n", df['Occupation'].value_counts())
```

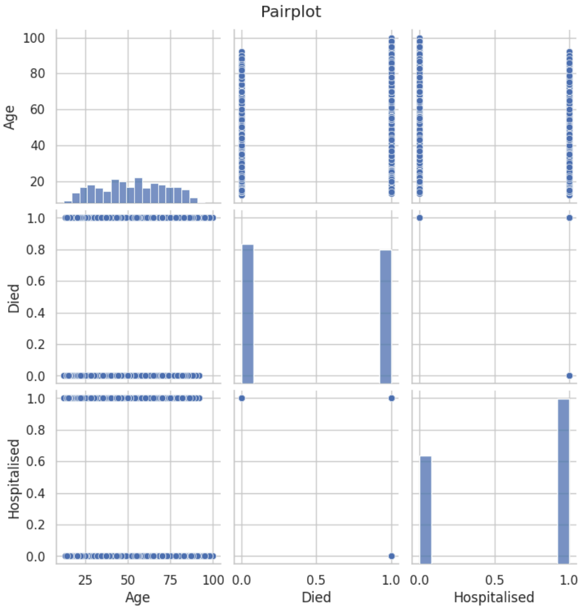
```
Sex:
 Sex
female    1328
male      1243
Name: count, dtype: int64

Education:
 Education
Secondary    1280
primary       659
iliterate     533
unknown        80
Tertiary       19
Name: count, dtype: int64

Occupation:
 Occupation
farming              2032
household             248
others/unknown        156
professional           37
student                35
unemployed             30
business/service       21
worker                  6
```

```
others              3
retiree             3
Name: count, dtype: int64
```

```
#Correlation Heatmap
df_corr = df.copy()
df_corr['Died'] = df_corr['Died'].map({'yes': 1, 'no': 0})
df_corr['Hospitalised'] = df_corr['Hospitalised'].map({'yes': 1, 'no': 0})

sns.heatmap(df_corr[['Age', 'Died', 'Hospitalised']].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```



```
#Pairplot
# This will not add much with limited numerics but valid for full dataset
sns.pairplot(df_corr[['Age', 'Died', 'Hospitalised']])
plt.suptitle("Pairplot", y=1.02)
plt.show()
```



Phase 2

1. Setup & Load

```
# Setup: imports, options, load data
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis
from scipy import stats
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.neighbors import LocalOutlierFactor
from scipy.cluster.hierarchy import dendrogram, linkage
from mpl_toolkits.mplot3d import Axes3D
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (10,6)
sns.set(style='whitegrid')

# Load dataset (change url if needed)
data_url = "https://raw.githubusercontent.com/salemprakash/EDA/main/Data/SuicideChina.csv"
df = pd.read_csv(data_url)
print("Data loaded. Shape:", df.shape)
display(df.head())
display(df.info())
```

```
Data loaded. Shape: (2571, 12)
```

	rownames	Person_ID	Hospitalised	Died	Urban	Year	Month	Sex	Age	Education	Occupation	method
0	1	1	yes	no	no	2010	12	female	39	Secondary	household	Other poison
1	2	2	no	yes	no	2009	3	male	83	primary	farming	Hanging
2	3	3	no	yes	no	2010	2	male	60	primary	farming	Hanging
3	4	4	no	yes	no	2011	1	male	73	primary	farming	Hanging
4	5	5	yes	no	no	2009	8	male	51	Secondary	farming	Pesticide

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2571 entries, 0 to 2570
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   rownames      2571 non-null   int64 
 1   Person_ID     2571 non-null   int64 
 2   Hospitalised  2571 non-null   object
 3   Died          2571 non-null   object
 4   Urban         2571 non-null   object
 5   Year          2571 non-null   int64 
 6   Month         2571 non-null   int64 
 7   Sex           2571 non-null   object
 8   Age           2571 non-null   int64 
 9   Education     2571 non-null   object
 10  Occupation    2571 non-null   object
 11  method        2571 non-null   object
dtypes: int64(5), object(7)
memory usage: 241.2+ KB
None
```

2. Quick overview + cleaning decisions

```
# Quick summary and cleaning plan
print("Columns and dtypes:")
display(df.dtypes)

# Missing values
missing = df.isnull().sum().sort_values(ascending=False)
display(missing[missing>0])

# Duplicates
print("Duplicate rows:", df.duplicated().sum())

# Basic cleaning decisions (DO NOT AUTO-DROP anything without checking)
# - If columns with >50% missing, consider dropping or documenting
# - If rows have missing targets, you may drop them for analyses that require target
pct_missing = (df.isnull().mean()*100).round(2).sort_values(ascending=False)
display(pct_missing)

# Example: if you want to drop columns with >60% missing (uncomment to apply)
# cols_to_drop = pct_missing[pct_missing > 60].index.tolist()
# df.drop(columns=cols_to_drop, inplace=True)
# print("Dropped columns:", cols_to_drop)
```

```
Columns and dtypes:
```

	0
rownames	int64
Person_ID	int64
Hospitalised	object
Died	object
Urban	object
Year	int64
Month	int64
Sex	object
Age	int64
Education	object
Occupation	object
method	object

dtype: object

0

dtype: int64

```
Duplicate rows: 0
```

	0
rownames	0.0
Person_ID	0.0
Hospitalised	0.0
Died	0.0
Urban	0.0
Year	0.0
Month	0.0
Sex	0.0
Age	0.0
Education	0.0
Occupation	0.0
method	0.0

dtype: float64

3. 1D Analysis — stats + plots

- Numerical summary: skewness, kurtosis, quantiles, IQR, outlier counts

```
# Identify numeric and categorical columns
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = df.select_dtypes(include=['object','category']).columns.tolist()
print("Numeric columns:", numeric_cols)
print("Categorical columns:", cat_cols)

# Descriptive statistics
desc = df[numeric_cols].describe().T
desc['skew'] = df[numeric_cols].skew().round(4)
desc['kurtosis'] = df[numeric_cols].apply(kurtosis).round(4)
q = df[numeric_cols].quantile([0.01,0.05,0.25,0.5,0.75,0.95,0.99]).T
desc = desc.join(q)
# IQR and outlier counts using 1.5*IQR rule
desc['IQR'] = desc[0.75] - desc[0.25]
def count_outliers(col):
    s = df[col].dropna()
    q1 = s.quantile(0.25); q3 = s.quantile(0.75); iqr = q3-q1
    low = q1 - 1.5*iqr; high = q3 + 1.5*iqr
    return ((s < low) | (s > high)).sum()
desc['outlier_count_1.5IQR'] = [count_outliers(c) for c in desc.index]
display(desc)
```

```
# Show columns with highest skew (absolute)
print("Top skewed numeric features:")
display(desc['skew'].abs().sort_values(ascending=False).head(10))
```

```
Numeric columns: ['rownames', 'Person_ID', 'Year', 'Month', 'Age']
Categorical columns: ['Hospitalised', 'Died', 'Urban', 'Sex', 'Education', 'Occupation', 'method']
```

	count	mean	std	min	25%	50%	75%	max	skew	kurtosis	0.01	0.05	0.25	0.5	0.75	0.95	0.99	IQR	outlier_count_1.5IQR
rownames	2571.0	1286.000000	742.328095	1.0	643.5	1286.0	1928.5	2571.0	0.0000	-1.2000	26.7	129.5	643.5	1286.0	1928.5	2442.5	2545.3	1285.0	0
Person_ID	2571.0	1286.000000	742.328095	1.0	643.5	1286.0	1928.5	2571.0	0.0000	-1.2000	26.7	129.5	643.5	1286.0	1928.5	2442.5	2545.3	1285.0	0
Year	2571.0	2010.045508	0.791412	2009.0	2009.0	2010.0	2011.0	2011.0	-0.0809	-1.3988	2009.0	2009.0	2009.0	2010.0	2011.0	2011.0	2011.0	2.0	0
Month	2571.0	6.298327	3.202515	1.0	4.0	6.0	9.0	12.0	0.0171	-1.0336	1.0	1.0	4.0	6.0	9.0	12.0	12.0	5.0	0
Age	2571.0	52.630883	19.783878	12.0	37.0	53.0	69.0	100.0	0.0143	-1.0036	17.0	22.0	37.0	53.0	69.0	84.0	89.3	32.0	0

```
Top skewed numeric features:
```

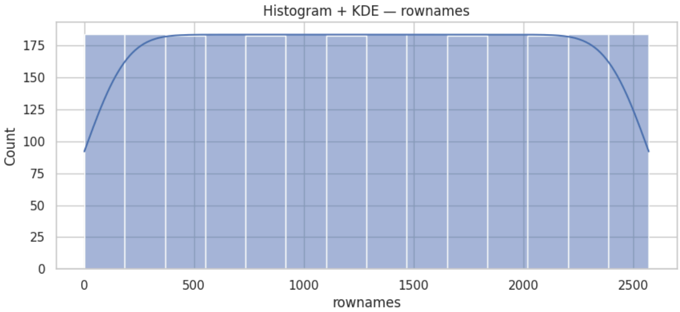
	skew
Year	0.0809
Month	0.0171
Age	0.0143
rownames	0.0000
Person_ID	0.0000

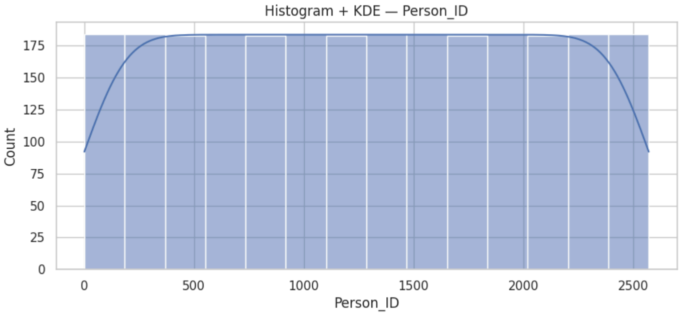
dtype: float64

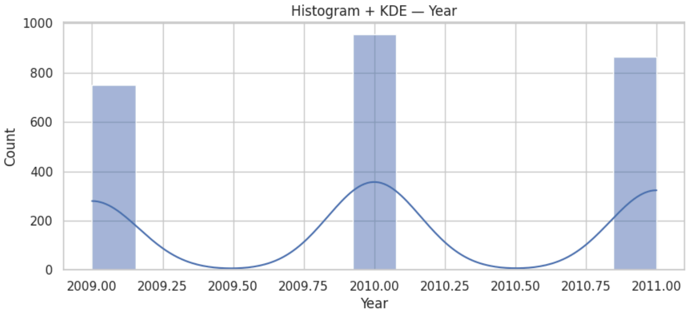
Next steps: Generate code with desc New interactive sheet

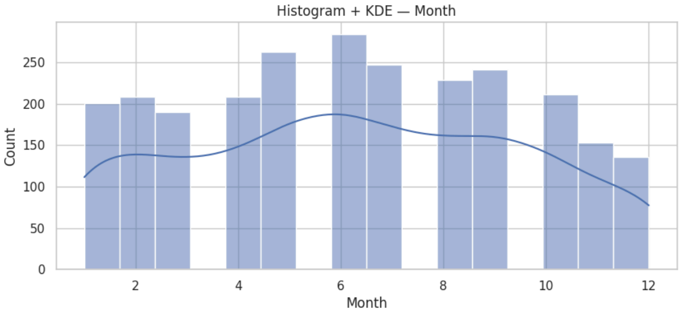
- 1D plots: histograms, kde, box, violin, frequency tables for cats

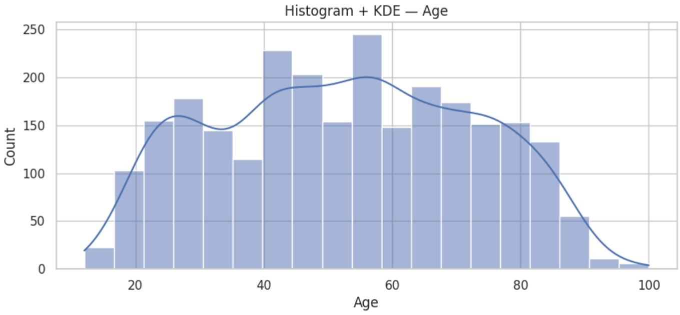
```
# Use sample for heavy columns
sample_frac = 1.0 if len(df) <= 5000 else 0.2

# Histograms + KDE
for c in numeric_cols:
    plt.figure(figsize=(10,4))
    sns.histplot(df[c].dropna().sample(frac=sample_frac, random_state=1), kde=True)
    plt.title(f"Histogram + KDE — {c}")
    plt.show()

# Box + Violin
for c in numeric_cols:
    fig, ax = plt.subplots(1,2, figsize=(12,4))
    sns.boxplot(x=df[c].dropna().sample(frac=sample_frac, random_state=1), ax=ax[0])
    ax[0].set_title(f"Boxplot — {c}")
    sns.violinplot(x=df[c].dropna().sample(frac=sample_frac, random_state=1), ax=ax[1])
    ax[1].set_title(f"Violin — {c}")
    plt.show()

# Categorical frequency & barplot
for c in cat_cols:
    counts = df[c].value_counts(dropna=False)
    print(f"Value counts for {c} (top 10):")
    display(counts.head(10))
    plt.figure(figsize=(10,4))
    sns.countplot(y=c, data=df, order=counts.index[:20])
    plt.title(f"Top categories in {c}")
    plt.show()
```

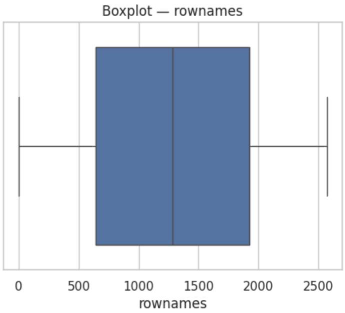

Histogram + KDE — rownames

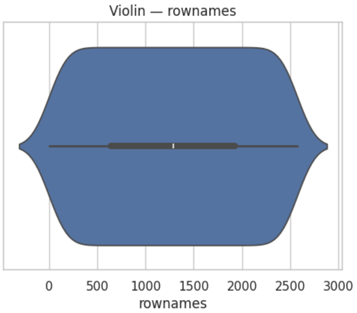

Histogram + KDE — Person_ID

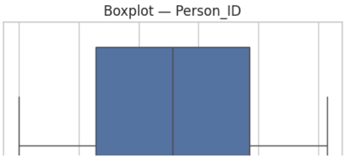

Histogram + KDE — Year

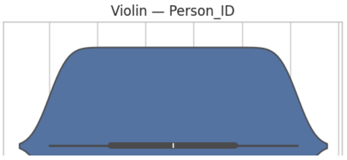

Histogram + KDE — Month

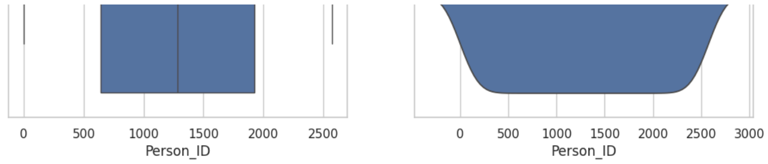

Histogram + KDE — Age

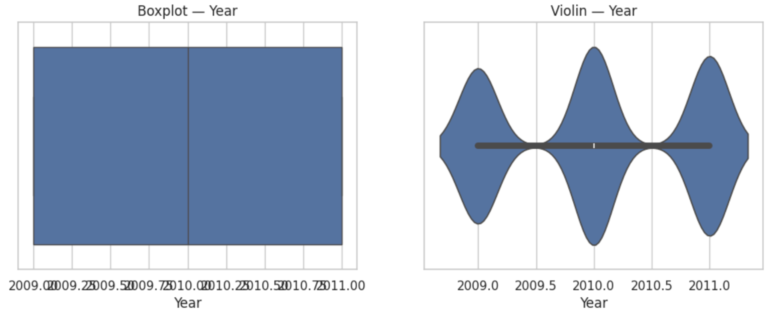

Boxplot — rownames

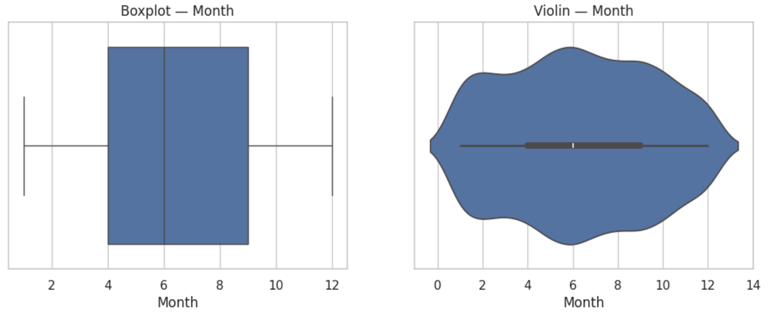

Violin — rownames

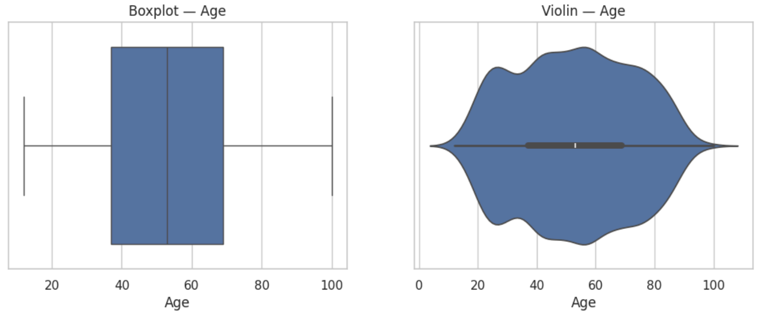

Boxplot — Person_ID


Violin — Person_ID





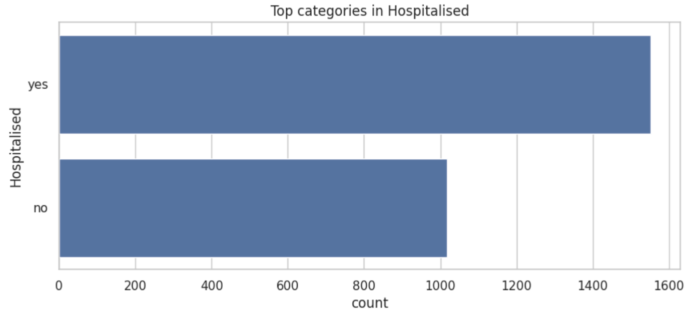




```
Value counts for Hospitalised (top 10):
```

Hospitalised	count
yes	1553
no	1018

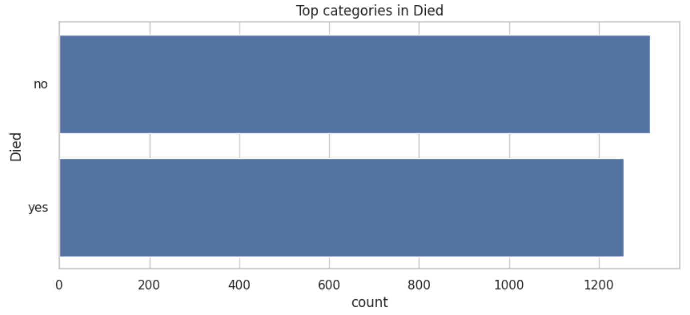
dtype: int64



```
Value counts for Died (top 10):
```

Died	count
no	1315
yes	1256

dtype: int64

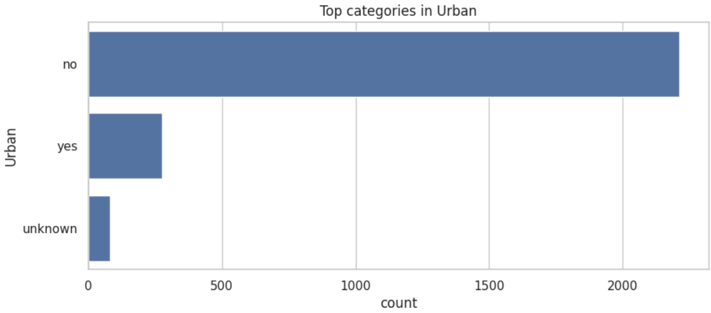


```
Value counts for Urban (top 10):
```

count

Urban	
no	2213
yes	277
unknown	81

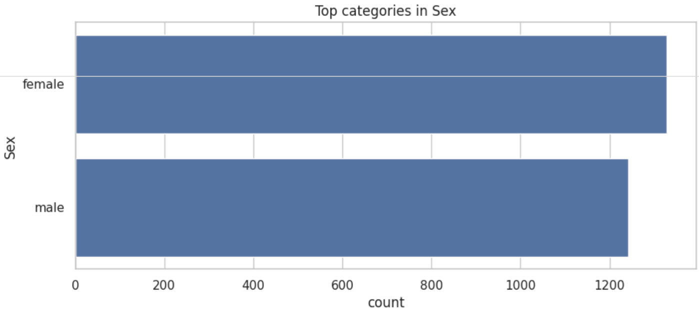
dtype: int64



```
Value counts for Sex (top 10):
```

	count
Sex	
female	1328
male	1243

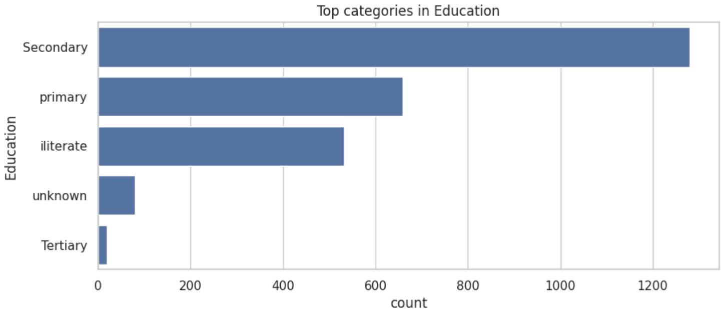
dtype: int64



```
Value counts for Education (top 10):
```

	count
Education	
Secondary	1280
primary	659
iliterate	533
unknown	80
Tertiary	19

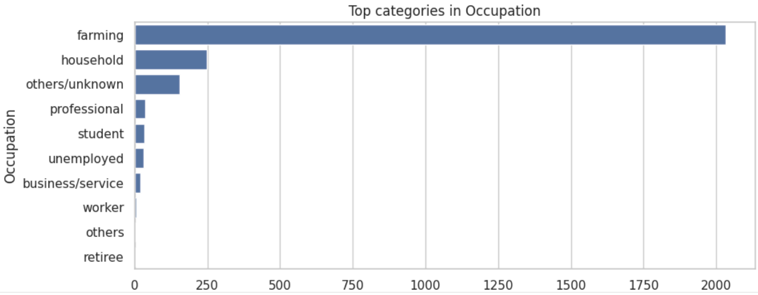
dtype: int64



```
Value counts for Occupation (top 10):
```

	count
Occupation	
farming	2032
household	248
others/unknown	156
professional	37
student	35
unemployed	30
business/service	21
worker	6
others	3
retiree	3

dtype: int64



count

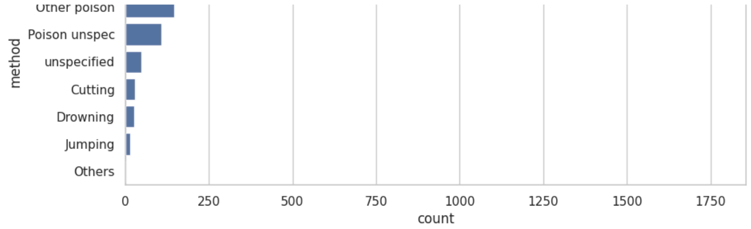
4. 2D Analysis — detailed comparisons

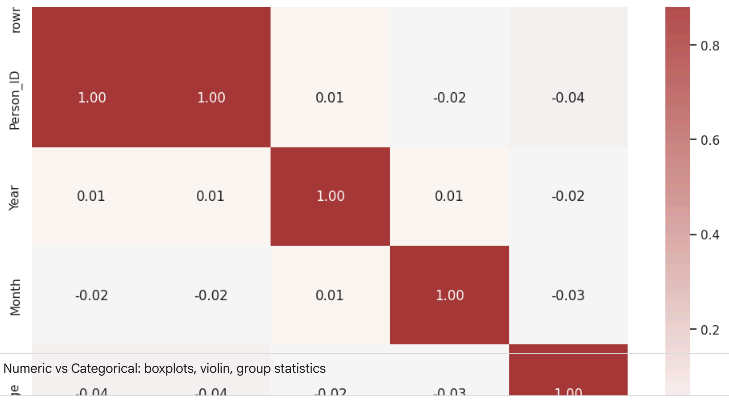
```
Value counts for method (top 10):
```

- Numeric vs Numeric: correlation matrix, pairplot (sample), regression for top pairs

```
count
method
```

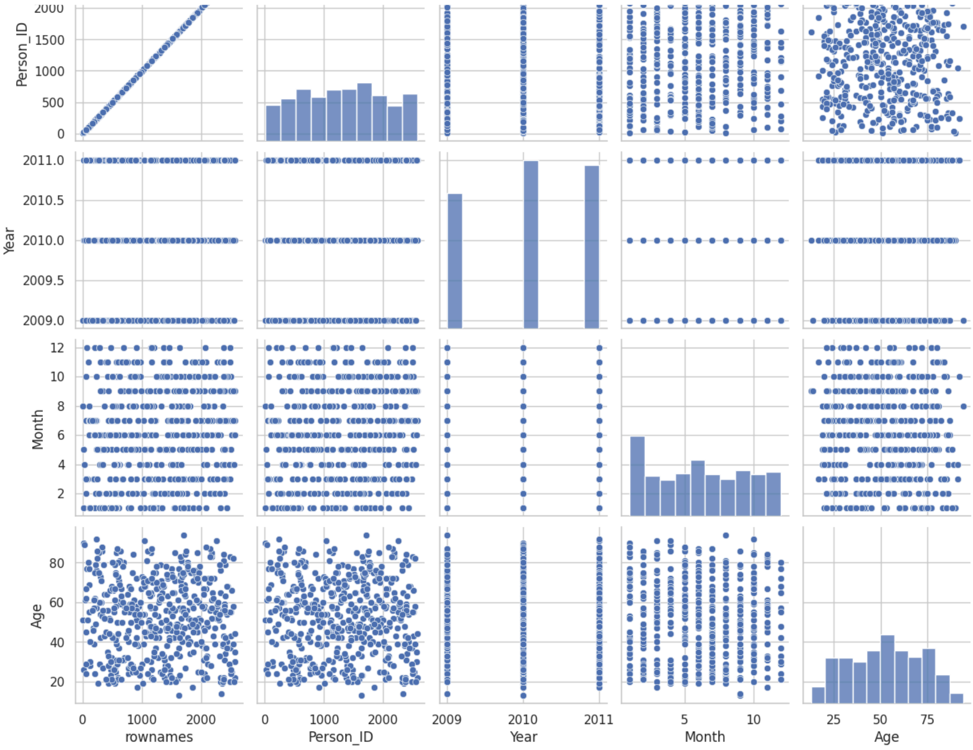
```
# Correlation matrix
corr = df[numeric_cols].corr()
plt.figure(figsize=(12,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="vlag", center=0)
plt.title("Numeric correlation matrix")
plt.show()

# Pairplot on sample (safe-guard large data)
pair_sample = df[numeric_cols].dropna().sample(n=min(500, len(df)), random_state=1)
sns.pairplot(pair_sample)
plt.suptitle("Pairplot (sample up to 500 rows)", y=1.02)
plt.show()

# Scatter + regression for top 3 absolute correlated pairs
corr_triu = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
top_pairs = corr_triu.abs().stack().sort_values(ascending=False).head(3)
print("Top correlated pairs:\n", top_pairs)
for (a,b), val in top_pairs.items():
    plt.figure(figsize=(8,4))
    sns.regplot(x=df[a], y=df[b], scatter_kws={'s':10}, line_kws={'color':'red'})
    plt.title(f"{a} vs {b} — corr={corr.loc[a,b]:.3f}")
    plt.show()
```



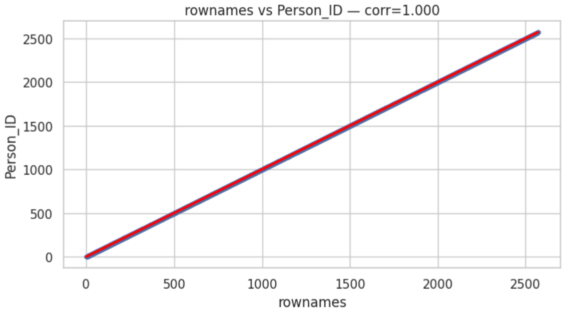


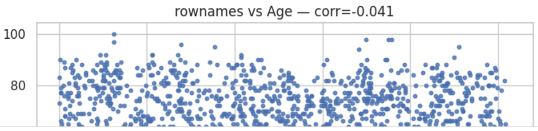
- Numeric vs Categorical: boxplots, violin, group statistics

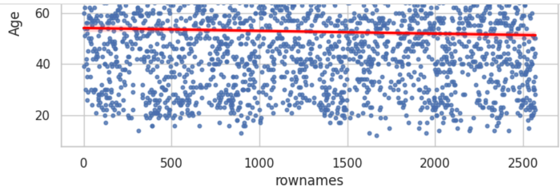
```
# For each categorical column, visualize distribution of numeric cols
for cat in cat_cols:
    # pick top 3 numeric columns by variance
    top_nums = df[numeric_cols].var().sort_values(ascending=False).index[:3].tolist()
    for num in top_nums:
        plt.figure(figsize=(12,5))
        sns.boxplot(x=cat, y=num, data=df)
        plt.xticks(rotation=45)
        plt.title(f"{num} by {cat} — Boxplot")
        plt.show()

        plt.figure(figsize=(12,5))
        sns.violinplot(x=cat, y=num, data=df)
        plt.xticks(rotation=45)
        plt.title(f"{num} by {cat} — Violin")
        plt.show()

# Group statistics
for cat in cat_cols:
    for num in numeric_cols[:3]:
        print(f"Group stats — {num} by {cat}")
        display(df.groupby(cat)[num].agg(['count','mean','median','std']).sort_values('count', ascending=False).head(20))
```

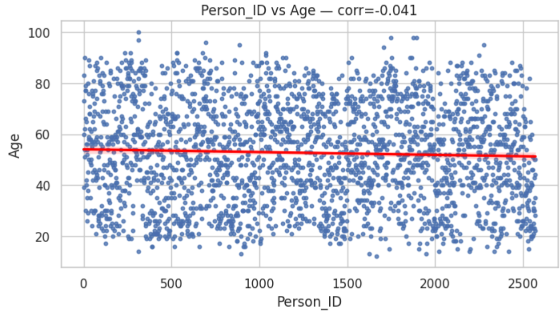


```
Top correlated pairs:
 rownames   Person_ID    1.000000
            Age          0.040988
Person_ID  Age          0.040988
dtype: float64
```

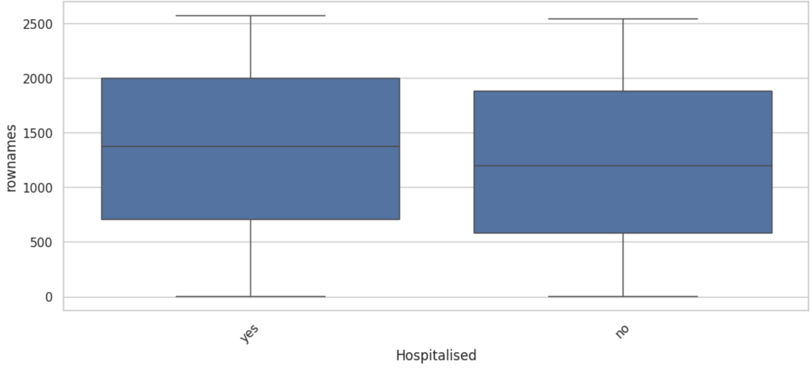




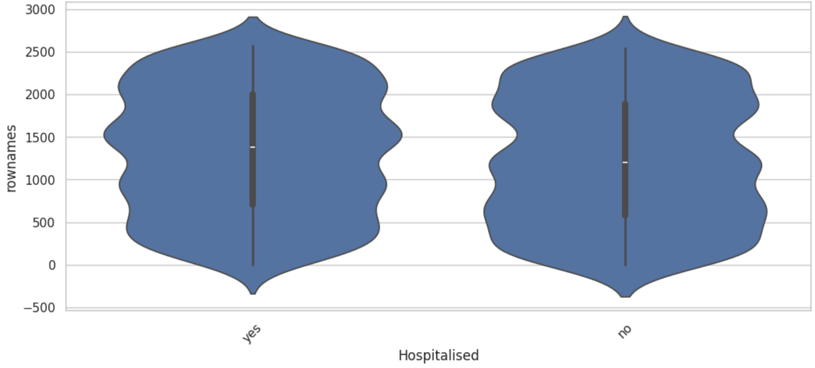

Age
60
40
20
0
500
1000
1500
2000
2500
rownames


Person_ID vs Age — corr=-0.041
100
80
60
40
20
Age
0
500
1000
1500
2000
2500
Person_ID

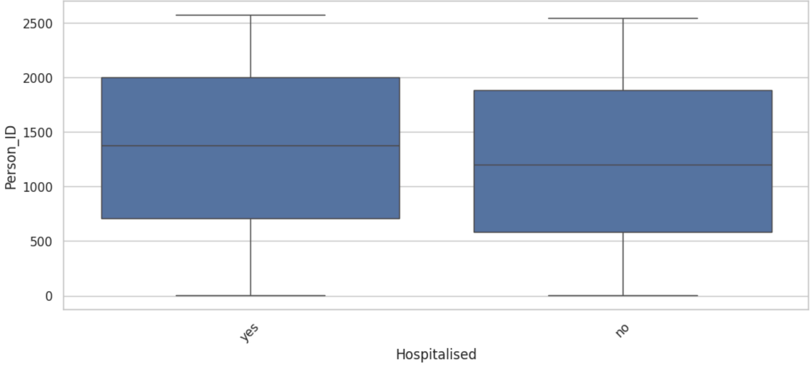
rownames by Hospitalised — Boxplot



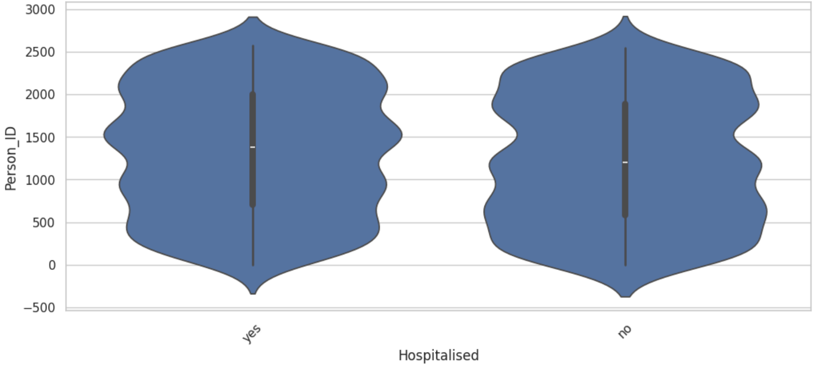
rownames by Hospitalised — Violin



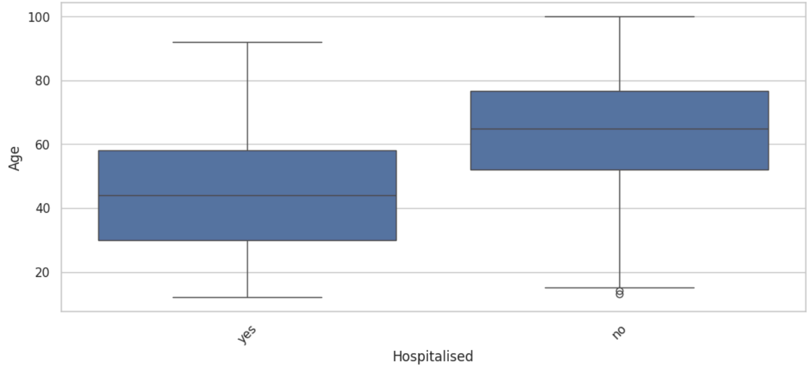
Person_ID by Hospitalised — Boxplot



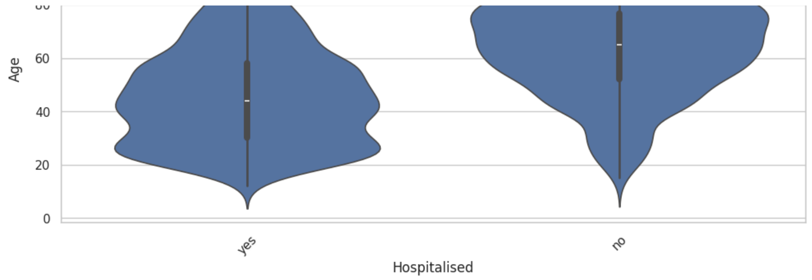
Person_ID by Hospitalised — Violin



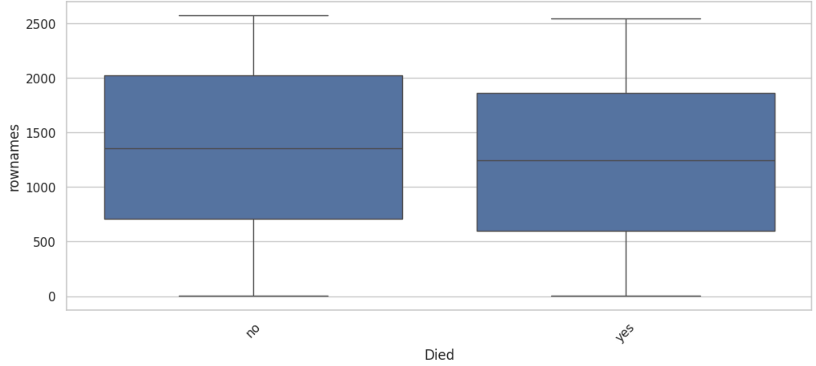
Age by Hospitalised — Boxplot



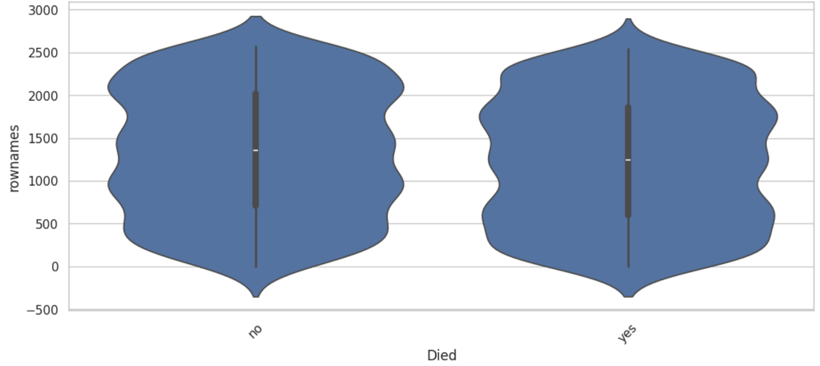
Age by Hospitalised — Violin



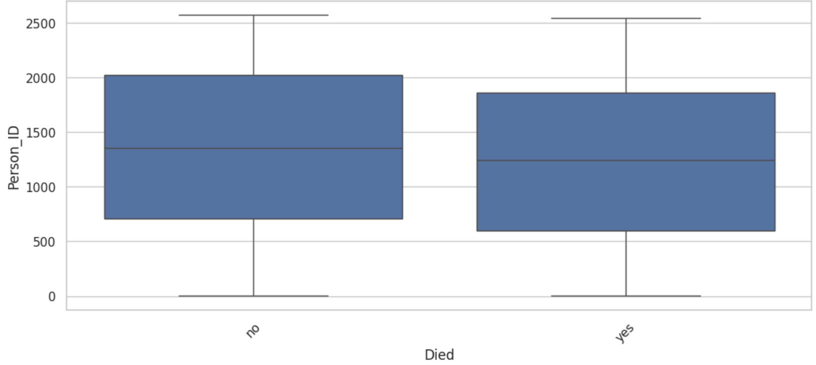
rownames by Died — Boxplot



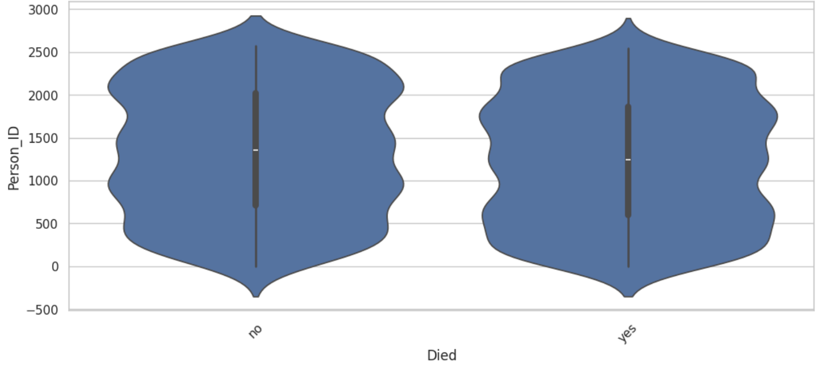
rownames by Died — Violin



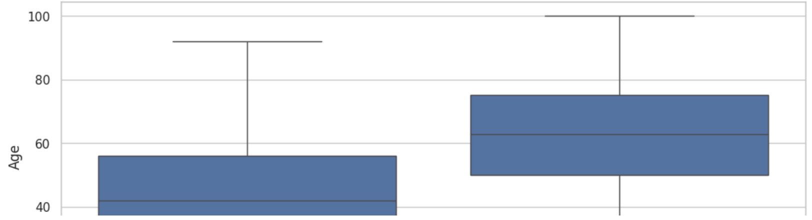
Person_ID by Died — Boxplot

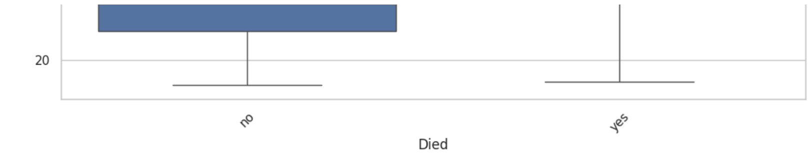


Person_ID by Died — Violin

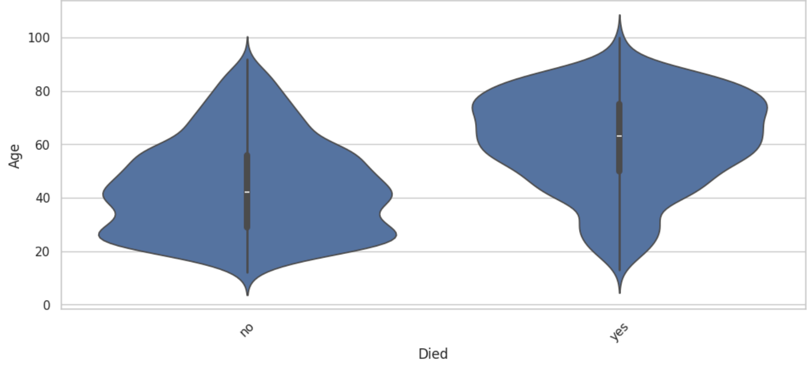


Age by Died — Boxplot

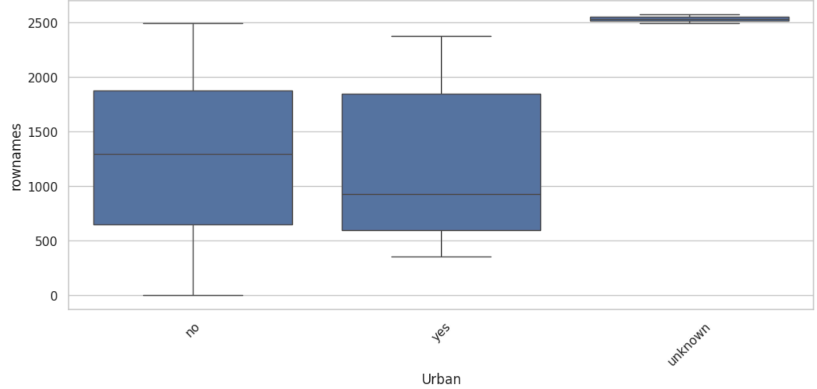




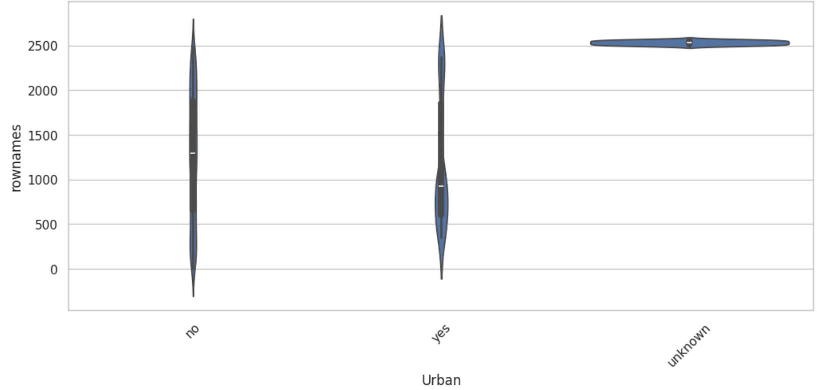
Age by Died — Violin



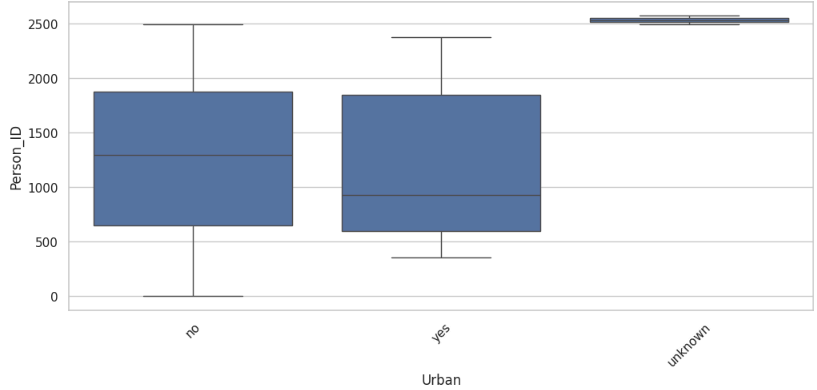
rownames by Urban — Boxplot



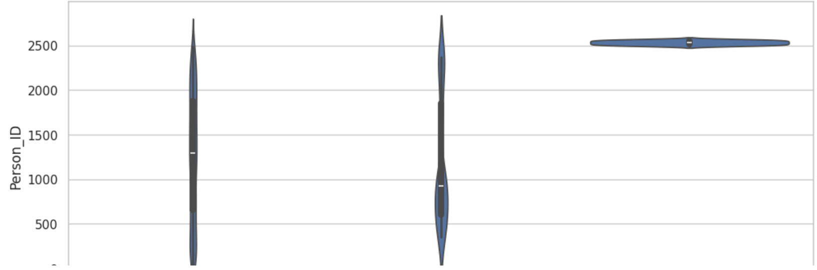
rownames by Urban — Violin

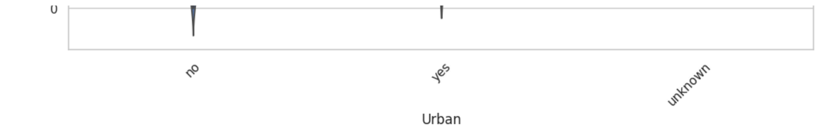


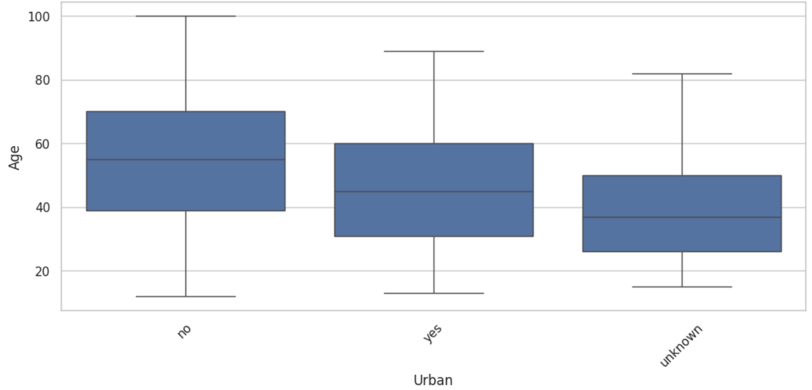
Person_ID by Urban — Boxplot

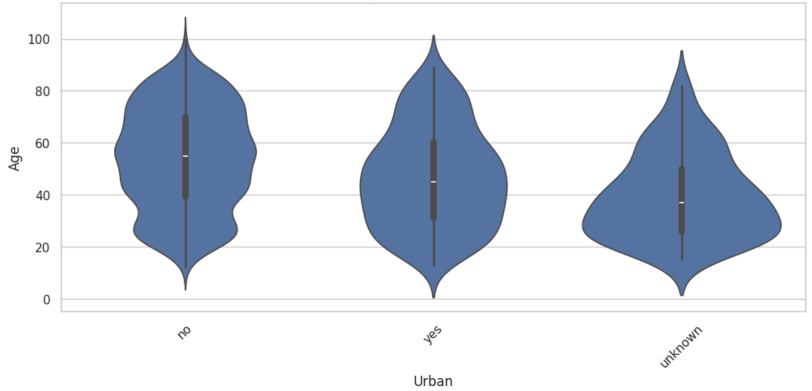


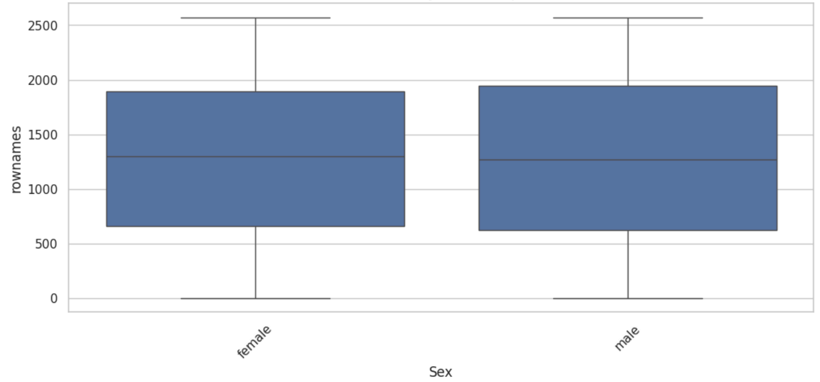
Person_ID by Urban — Violin

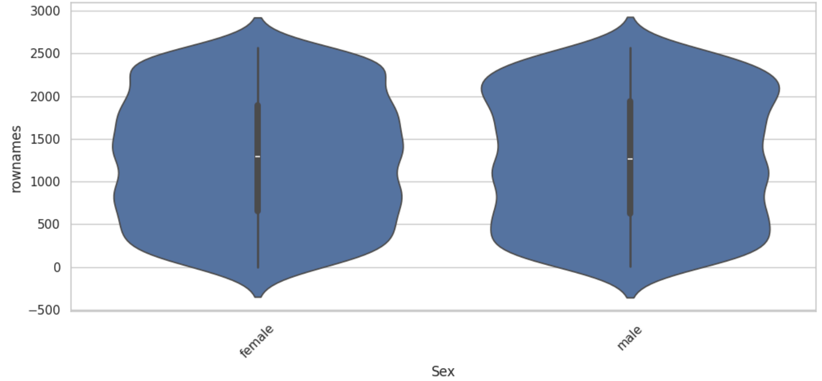


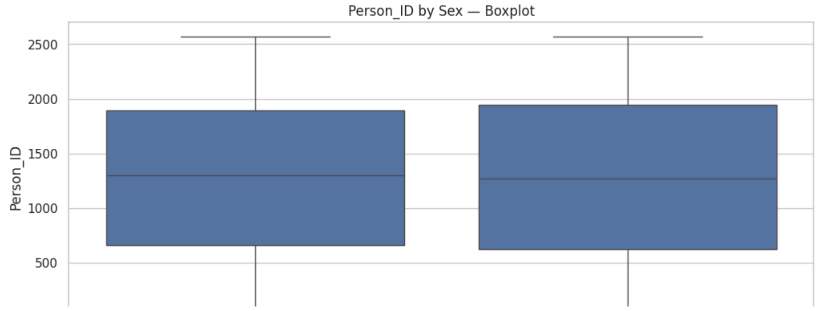



Age by Urban — Boxplot


Age by Urban — Violin


rownames by Sex — Boxplot


rownames by Sex — Violin


Person_ID by Sex — Boxplot