P3_Cleaning_Visualizations

January 31, 2026

Practical 3: Data Cleaning and Visualization

1 Data Cleaning and Visualization in Python

Why this matters - Real-world data is messy: missing values, wrong types, duplicates, outliers. - Cleaning is 70-80% of a data analyst's work. - Visualization helps explore patterns, detect issues, and communicate insights.

Goals 1. Load and explore a real dataset. 2. Clean common data problems systematically. 3. Create insightful visualizations.

1.0.1 1. Import the necessary libraries

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

#To suppress all warnings
import warnings
warnings.filterwarnings('ignore')
```

1.0.2 2. Loading and Exploration

[3]:
```
# Load Titanic dataset
df = sns.load_dataset('titanic')

# have a quick look
print(df.shape)
df.head()
```

```
(891, 15)
```

[3]:
```
   survived  pclass     sex   age  sibsp  parch     fare embarked  class  \
0         0       3    male  22.0      1      0   7.2500        S  Third
1         1       1  female  38.0      1      0  71.2833        C  First
2         1       3  female  26.0      0      0   7.9250        S  Third
3         1       1  female  35.0      1      0  53.1000        S  First
4         0       3    male  35.0      0      0   8.0500        S  Third
```

```
     who  adult_male deck  embark_town alive  alone
0    man        True  NaN  Southampton    no  False
1  woman       False    C    Cherbourg   yes  False
2  woman       False  NaN  Southampton   yes   True
3  woman       False    C  Southampton   yes  False
4    man        True  NaN  Southampton    no   True
```

[5]:
```python
# Basic info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

[7]:
```python
# Statistical summary (numeric only)
df.describe()
```

[7]:

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[9]:
```python
# Categorical summary
df.describe(include=['object', 'category'])
```

[9]:

	sex	embarked	class	who	deck	embark_town	alive
count	891	889	891	891	203	889	891
unique	2	3	3	3	7	3	2
top	male	S	Third	man	C	Southampton	no
freq	577	644	491	537	59	644	549

[11]:
```python
# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Missing values per column:
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

[13]:
```python
# check duplicates
print(f"\nNumber of duplicate rows: {df.duplicated().sum()}")
```

```
Number of duplicate rows: 107
```

1.1 Data Cleaning

Common issues in Titanic: - Missing values: age (177), embarked (2), deck (688), embtown (not useful) - Data types: some could be categorical - Outliers - Redundant/unuseful columns

1.1.1 Handling Missing Values

[15]:
```python
# 1. Drop columns with too many missing values (>70%)
df_clean = df.drop(columns=['deck'])  # 688 missing out of 891
```

[17]:
```
# 2. Fill 'age' with median (robust to outliers)
median_age = df_clean['age'].median()
df_clean['age'].fillna(median_age, inplace=True)
```

[19]:
```
# 3. Fill 'embarked' with mode (most frequent)
mode_embarked = df_clean['embarked'].mode()[0]
df_clean['embarked'].fillna(mode_embarked, inplace=True)
```

[21]:
```
# 4. Fill 'embark_town' with mode
mode_embark_town = df_clean['embark_town'].mode()[0]
df_clean['embark_town'].fillna(mode_embark_town, inplace=True)
```

[23]:
```
print("Missing values after cleaning:")
print(df_clean.isnull().sum())
```

```
Missing values after cleaning:
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

1.1.2 Data Types and Categories

[26]:
```
# Convert to categorical for memory and plotting
categorical_cols = ['sex', 'embarked', 'class', 'who', 'adult_male', 'alone']
for col in categorical_cols:
    df_clean[col] = df_clean[col].astype('category')
```

[28]:
```
# Verify
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
```

```
 1   pclass       891 non-null    int64
 2   sex          891 non-null    category
 3   age          891 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     891 non-null    category
 8   class        891 non-null    category
 9   who          891 non-null    category
 10  adult_male   891 non-null    category
 11  embark_town  891 non-null    object
 12  alive        891 non-null    object
 13  alone        891 non-null    category
dtypes: category(6), float64(2), int64(4), object(2)
memory usage: 61.7+ KB
```

1.1.3 Duplicates and Redundant Columns

[31]:
```
# Remove exact duplicates (none in this dataset, but good practice)
df_clean.drop_duplicates(inplace=True)
```

[33]:
```
# Drop redundant columns
df_clean.drop(columns=['alive', 'pclass', 'embark_town'], inplace=True,␣
↪errors='ignore')
```

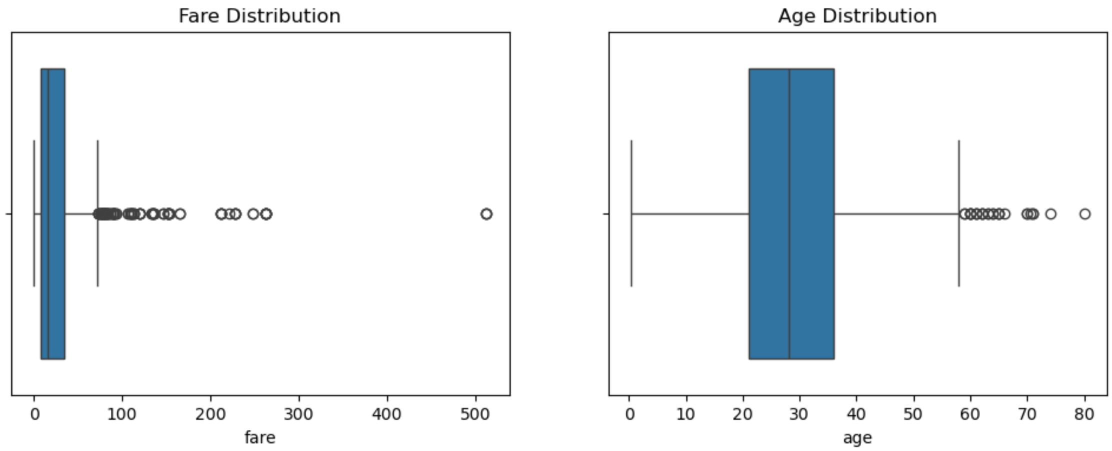
[35]:
```
df_clean.head()
```

[35]:
```
   survived     sex   age  sibsp  parch     fare embarked  class    who  \
0         0    male  22.0      1      0   7.2500        S  Third    man
1         1  female  38.0      1      0  71.2833        C  First  woman
2         1  female  26.0      0      0   7.9250        S  Third  woman
3         1  female  35.0      1      0  53.1000        S  First  woman
4         0    male  35.0      0      0   8.0500        S  Third    man

  adult_male  alone
0       True  False
1      False  False
2      False   True
3      False  False
4       True   True
```

1.1.4 Outlier Detection

[38]:
```
# Boxplot for fare and age
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
sns.boxplot(x=df_clean['fare'])
```

```
plt.title("Fare Distribution")

plt.subplot(1, 2, 2)
sns.boxplot(x=df_clean['age'])
plt.title("Age Distribution")

plt.show()
```



[40]:
```
# Optional: Cap extreme fares using IQR method
Q1 = df_clean['fare'].quantile(0.25)
Q3 = df_clean['fare'].quantile(0.75)
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR

print(f"Fares above {upper_bound:.2f} are potential outliers.")
df_clean['fare_capped'] = df_clean['fare'].clip(upper=upper_bound)
```

```
Fares above 73.42 are potential outliers.
```

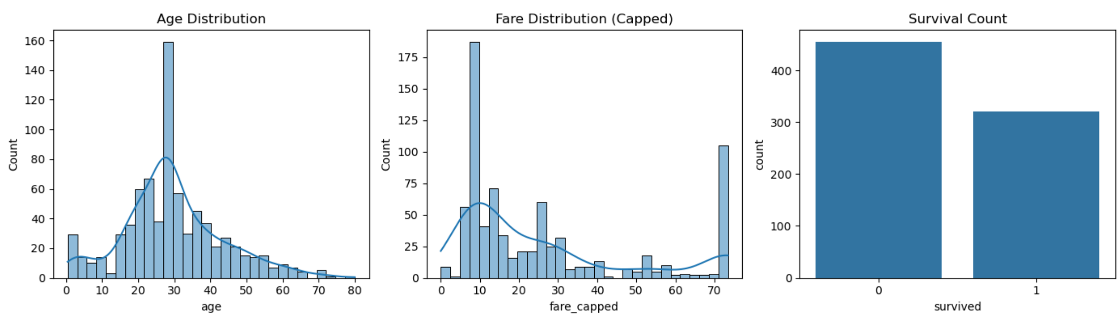
1.2 Visualizations

We will create: - Distribution plots - Categorical counts - Relationships (scatter, box) - Correlation heatmap

1.2.1 Univariate Plots

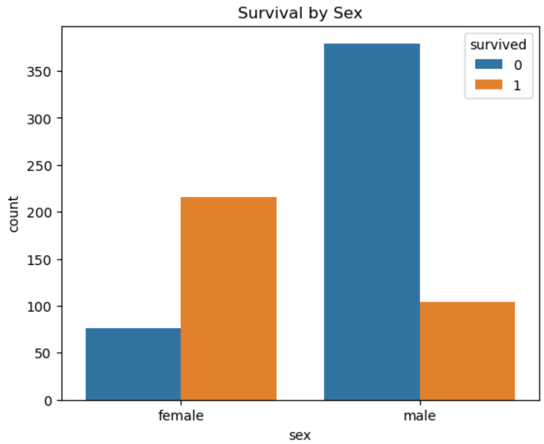
[44]:
```
plt.figure(figsize=(14, 4))

# Age distribution
plt.subplot(1, 3, 1)
sns.histplot(df_clean['age'], kde=True, bins=30)
plt.title('Age Distribution')
```

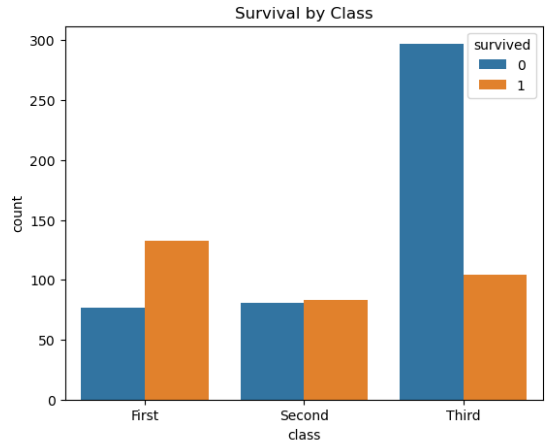
```
# Fare distribution (capped)
plt.subplot(1, 3, 2)
sns.histplot(df_clean['fare_capped'], kde=True, bins=30)
plt.title('Fare Distribution (Capped)')

# Survival rate
plt.subplot(1, 3, 3)
sns.countplot(x='survived', data=df_clean)
plt.title('Survival Count')

plt.tight_layout()
plt.show()
```



1.2.2 Categorical Relationships

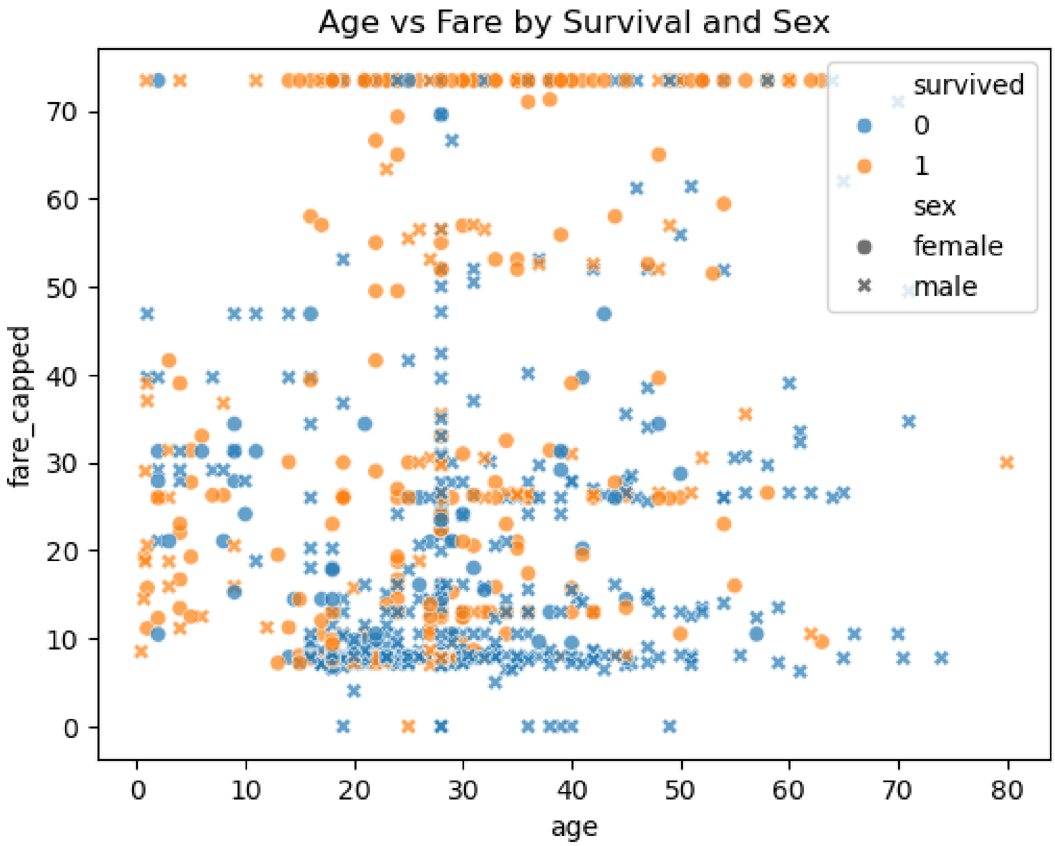
[47]:
```
plt.figure(figsize=(14, 5))

# Survival by sex
plt.subplot(1, 2, 1)
sns.countplot(x='sex', hue='survived', data=df_clean)
plt.title('Survival by Sex')

# Survival by class
plt.subplot(1, 2, 2)
sns.countplot(x='class', hue='survived', data=df_clean)
plt.title('Survival by Class')

plt.show()
```



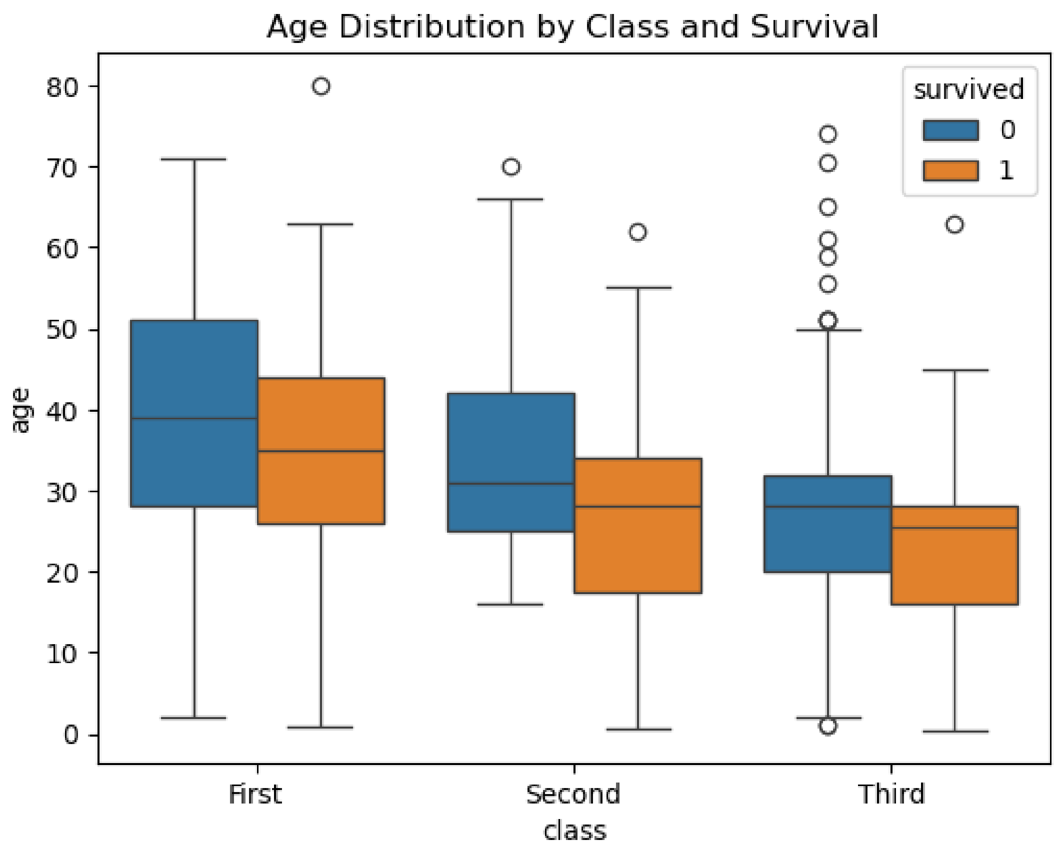


1.2.3 Numerical Relationships

[50]:
```
# Age vs Fare, colored by survival
sns.scatterplot(x='age', y='fare_capped', hue='survived', style='sex',
                alpha=0.7, data=df_clean)
plt.title('Age vs Fare by Survival and Sex')
plt.show()
```

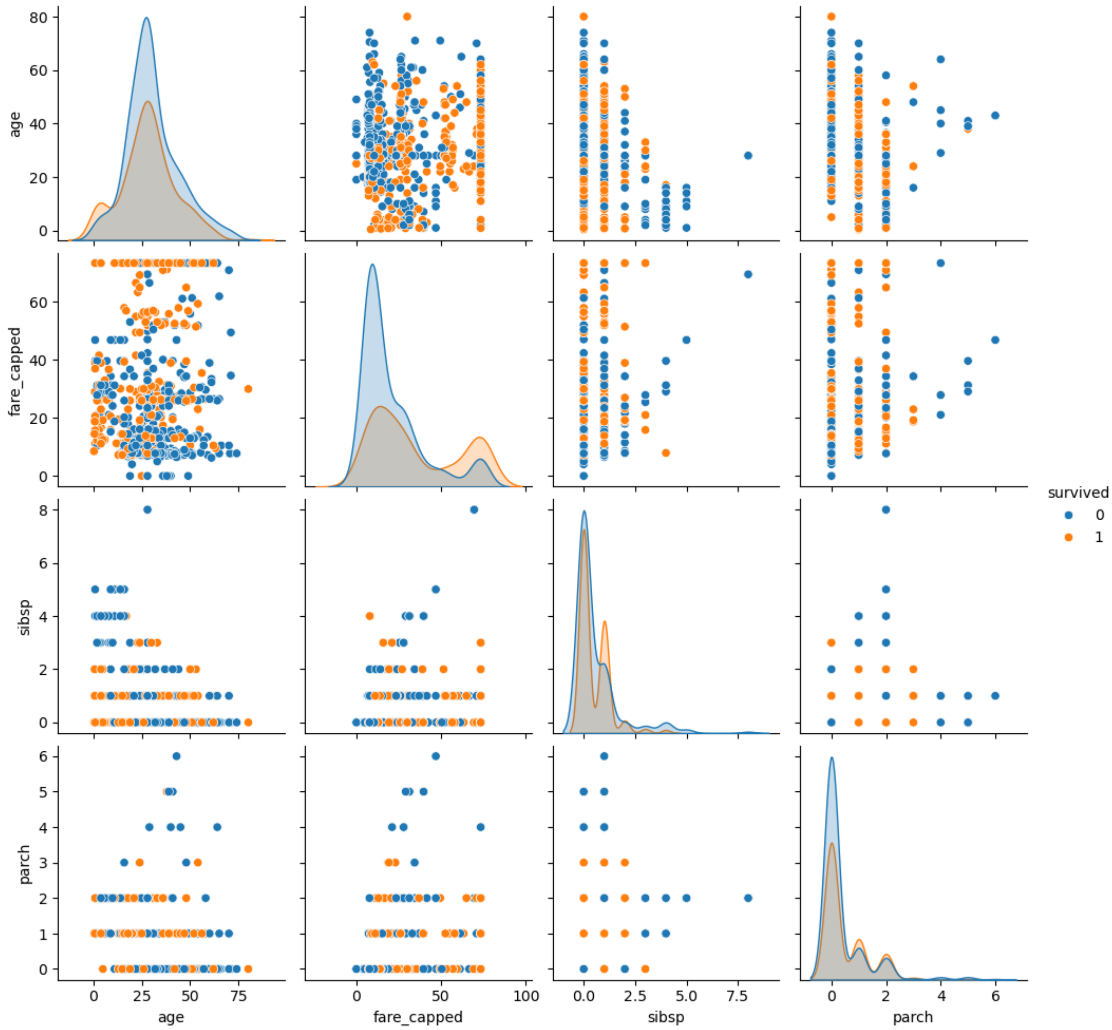


[52]:
```
# Boxplot: Age by Class
sns.boxplot(x='class', y='age', hue='survived', data=df_clean)
plt.title('Age Distribution by Class and Survival')
plt.show()
```

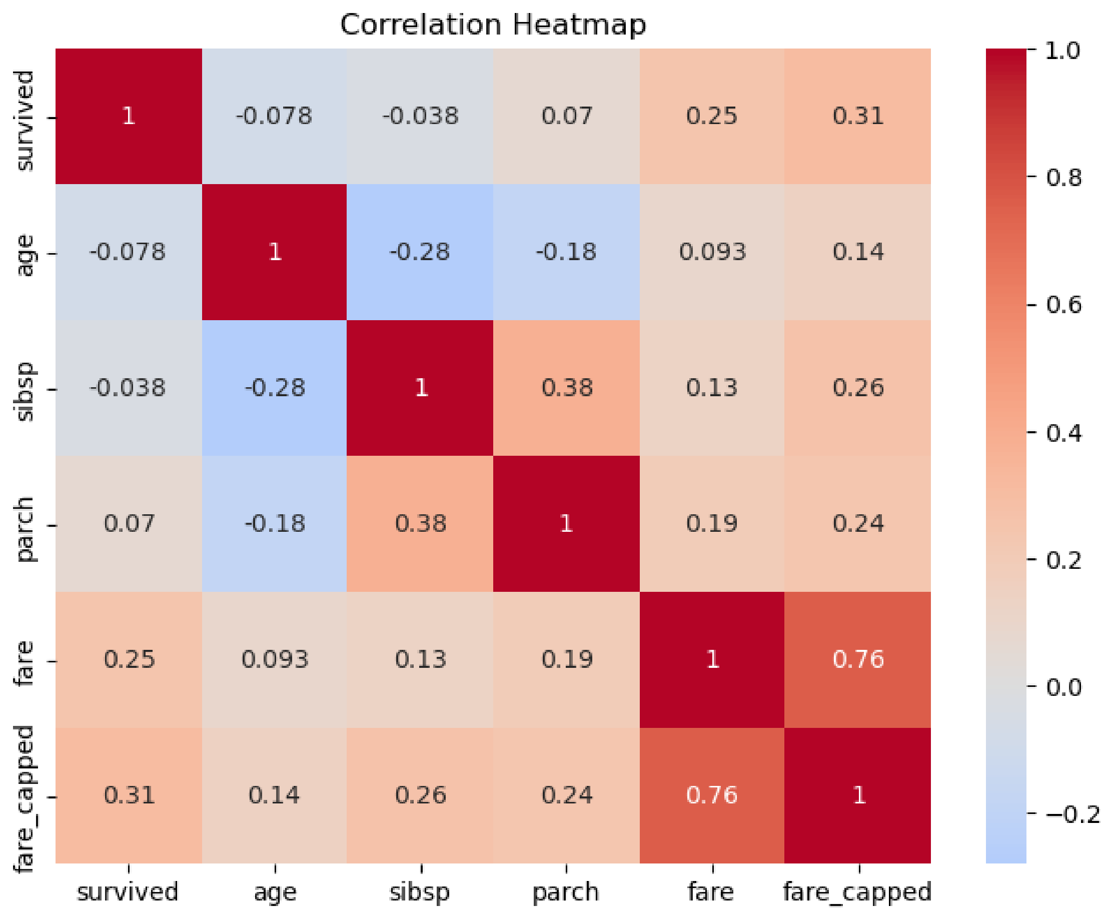


1.2.4 Pairplot and Heatmap

[55]:
```python
# Pairplot for key variables
sns.pairplot(df_clean, vars=['age', 'fare_capped', 'sibsp', 'parch'],
             hue='survived', diag_kind='kde')
plt.show()
```



[56]:
```
# Correlation heatmap
plt.figure(figsize=(8, 6))
numeric_cols = df_clean.select_dtypes(include=np.number).columns
sns.heatmap(df_clean[numeric_cols].corr(), annot=True, cmap='coolwarm',␣
  ↪center=0)
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
survived
age
sibsp
parch
fare
fare_capped
1
-0.078
-0.038
0.07
0.25
0.31
-0.078
1
-0.28
-0.18
0.093
0.14
-0.038
-0.28
1
0.38
0.13
0.26
0.07
-0.18
0.38
1
0.19
0.24
0.25
0.093
0.13
0.19
1
0.76
0.31
0.14
0.26
0.24
0.76
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

[ ]: