# Comprehensive Pandas Cheat Sheet

This cheat sheet covers the essential and advanced features of Pandas, a powerful library for data manipulation and analysis in Python.

## 1. Introduction

### 1.1 What is Pandas?

Pandas is a powerful, open-source library used for data manipulation and analysis in Python. It provides data structures like Series and DataFrame which make data manipulation easy and intuitive.

### 1.2 Installing Pandas

You can install Pandas using pip:

```
pip install pandas
```

Or, if you are using Anaconda:

```
conda install pandas
```

### 1.3 Importing Pandas

To use Pandas, you need to import it in your script:

In [1]:
```
import pandas as pd
# Now you can use Pandas functions with the pd prefix
```

## 2. Data Structures

### 2.1 Series ¶

A Series is a one-dimensional labeled array capable of holding any data type.

#### 2.1.1 Creating Series

In [2]:
```
# Creating a Series
import pandas as pd
import numpy as np

data = np.array([1, 2, 3, 4, 5])
series = pd.Series(data)
print(series)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

#### 2.1.2 Accessing Series Elements

In [3]:
```
# Accessing elements in a Series
print(series[0])  # Output: 1
print(series[:3])  # Output:
# 0    1
# 1    2
# 2    3
# dtype: int64
```

```
1
0    1
1    2
2    3
dtype: int64
```

#### 2.1.3 Series Methods

In [4]:
```
# Series methods
print(series.max())  # Output: 5
print(series.min())  # Output: 1
print(series.mean())  # Output: 3.0
```

```
5
1
3.0
```

### 2.2 DataFrame

A DataFrame is a two-dimensional labeled data structure with columns of potentially different types.

#### 2.2.1 Creating DataFrame

In [5]:
```
# Creating a DataFrame
data = {
    'Name': ['John', 'Anna', 'Peter', 'Linda'],
    'Age': [28, 24, 35, 32],
    'City': ['New York', 'Paris', 'Berlin', 'London']
}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age      City
0   John   28  New York
1   Anna   24     Paris
2  Peter   35    Berlin
3  Linda   32    London
```

#### 2.2.2 Accessing DataFrame Elements

In [6]:
```
# Accessing DataFrame elements
print(df['Name'])  # Accessing a single column
print(df[['Name', 'City']])  # Accessing multiple columns
print(df.loc[1])  # Accessing a row by index
print(df.iloc[2])  # Accessing a row by position
```

```
0     John
1     Anna
2    Peter
3    Linda
Name: Name, dtype: object
    Name      City
0   John  New York
1   Anna     Paris
2  Peter    Berlin
3  Linda    London
Name     Anna
Age        24
City    Paris
Name: 1, dtype: object
Name     Peter
Age         35
City    Berlin
Name: 2, dtype: object
```

#### 2.2.3 DataFrame Methods

In [7]:
```
# DataFrame methods
print(df.describe())  # Summary statistics
print(df.info())  # DataFrame information
print(df.head(2))  # First two rows
```

```
             Age
count   4.000000
mean   29.750000
std     4.787136
min    24.000000
25%    27.000000
50%    30.000000
75%    32.750000
max    35.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Name    4 non-null      object
 1   Age     4 non-null      int64 
 2   City    4 non-null      object
dtypes: int64(1), object(2)
memory usage: 224.0+ bytes
None
   Name  Age      City
0  John   28  New York
1  Anna   24     Paris
```

# 3. Data Manipulation

## 3.1 Reading and Writing Data

Pandas provides functions to read data from various file formats and write data to these formats.

### 3.1.1 Reading Data from CSV

In [8]:
```
# Reading data from CSV
df_csv = pd.read_csv('file.csv')
print(df_csv.head())
```

```
    Name  Age      City
0   John   28  New York
1   Anna   24     Paris
2  Peter   35    Berlin
3  Linda   32    London
```

#### 3.1.2 Writing Data to CSV, Excel

In [9]:
```
# Writing data to CSV
df.to_csv('output.csv', index=False)

# Writing data to Excel
df.to_excel('output.xlsx', index=False)
```

## 3.2 Handling Missing Data

Handling missing data is crucial for data cleaning and preparation.

#### 3.2.1 Detecting Missing Data

In [10]:
```
# Detecting missing data
data = {'A': [1, 2, np.nan], 'B': [5, np.nan, np.nan], 'C': [10, 20
df = pd.DataFrame(data)
print(df)
print(df.isnull())  # Detect missing values
print(df.isnull().sum())  # Count missing values
```

```
     A    B   C
0  1.0  5.0  10
1  2.0  NaN  20
2  NaN  NaN  30
       A      B      C
0  False  False  False
1  False   True  False
2   True   True  False
A    1
B    2
C    0
dtype: int64
```

#### 3.2.2 Filling Missing Data

In [11]:
```
# Filling missing data
print(df.fillna(0))  # Fill with a specified value
print(df.fillna(method='ffill'))  # Forward fill
print(df.fillna(method='bfill'))  # Backward fill
```

```
     A    B   C
0  1.0  5.0  10
1  2.0  0.0  20
2  0.0  0.0  30
     A    B   C
0  1.0  5.0  10
1  2.0  5.0  20
2  2.0  5.0  30
     A    B   C
0  1.0  5.0  10
1  2.0  NaN  20
2  NaN  NaN  30
```

#### 3.2.3 Dropping Missing Data

In [12]:
```
# Dropping missing data
print(df.dropna())  # Drop rows with any missing value
print(df.dropna(axis=1))  # Drop columns with any missing value
```

```
     A    B   C
0  1.0  5.0  10
    C
0  10
1  20
2  30
```

### 3.3 Data Transformation

Transforming data is often necessary to prepare it for analysis.

#### 3.3.1 Sorting

In [13]:
```
# Sorting data
data = {'Name': ['John', 'Anna', 'Peter', 'Linda'], 'Age': [28, 24,
df = pd.DataFrame(data)
print(df.sort_values(by='Age'))  # Sort by age
print(df.sort_values(by='Name'))  # Sort by name
```

```
    Name  Age
1   Anna   24
0   John   28
3  Linda   32
2  Peter   35
    Name  Age
1   Anna   24
0   John   28
3  Linda   32
2  Peter   35
```

#### 3.3.2 Ranking

In [14]:
```
# Ranking data
df['Rank'] = df['Age'].rank()
print(df)
```

```
    Name  Age  Rank
0   John   28   2.0
1   Anna   24   1.0
2  Peter   35   4.0
3  Linda   32   3.0
```

#### 3.3.3 Applying Functions

In [15]:
```
# Applying functions
df['AgePlusTen'] = df['Age'].apply(lambda x: x + 10)
print(df)

# Using applymap to apply function to each element
df[['Age', 'AgePlusTen']] = df[['Age', 'AgePlusTen']].applymap(lamb
print(df)
```

```
    Name  Age  Rank  AgePlusTen
0   John   28   2.0          38
1   Anna   24   1.0          34
2  Peter   35   4.0          45
3  Linda   32   3.0          42
    Name  Age  Rank  AgePlusTen
0   John   56   2.0          76
1   Anna   48   1.0          68
2  Peter   70   4.0          90
3  Linda   64   3.0          84
```

# 4. Data Aggregation and Grouping

## 4.1 GroupBy Operations

GroupBy operations allow you to split data into groups based on some criteria, apply a function to each group, and then combine the results.

In [16]:
```
# GroupBy operations
data = {
    'Team': ['A', 'B', 'A', 'B'],
    'Player': ['John', 'Anna', 'Peter', 'Linda'],
    'Points': [10, 15, 10, 25]
}
df = pd.DataFrame(data)
grouped = df.groupby('Team')
print(grouped.sum())  # Sum of points by team
```

```
      Points
Team
A         20
B         40
```

## 4.2 Pivot Tables

In [17]:
```
# Pivot tables
pivot_table = df.pivot_table(values='Points', index='Team', columns
print(pivot_table)
```

```
Player  Anna  John  Linda  Peter
Team
A        NaN  10.0    NaN   10.0
B       15.0   NaN   25.0    NaN
```

## 4.3 Cross Tabulation

In [18]:
```
# Cross tabulation
cross_tab = pd.crosstab(df['Team'], df['Player'], values=df['Points
print(cross_tab)
```

```
Player  Anna  John  Linda  Peter
Team
A        NaN  10.0    NaN   10.0
B       15.0   NaN   25.0    NaN
```

# 5. Data Merging and Concatenation

## 5.1 Concatenation

Concatenation combines data from multiple DataFrames into one.

In [19]:
```
# Concatenation
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']})
df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']})

frames = [df1, df2, df3]
result = pd.concat(frames)
print(result)
```

```
     A    B    C    D
0   A0   B0   C0   D0
1   A1   B1   C1   D1
2   A2   B2   C2   D2
3   A3   B3   C3   D3
0   A4   B4   C4   D4
1   A5   B5   C5   D5
2   A6   B6   C6   D6
3   A7   B7   C7   D7
0   A8   B8   C8   D8
1   A9   B9   C9   D9
2  A10  B10  C10  D10
3  A11  B11  C11  D11
```

## 5.2 Merging

Merging combines DataFrames based on keys or indexes.

In [20]:
```
# Merging
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})
right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
result = pd.merge(left, right, on='key')
print(result)
```

```
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

### 5.3 Joining

Joining combines DataFrames on their indexes.

In [21]:
```
# Joining
left = left.set_index('key')
right = right.set_index('key')
result = left.join(right)
print(result)
```

```
      A   B   C   D
key
K0   A0  B0  C0  D0
K1   A1  B1  C1  D1
K2   A2  B2  C2  D2
K3   A3  B3  C3  D3
```

# 6. Time Series Analysis

### 6.1 Date and Time Functions

Pandas provides rich functionality for working with dates and times.

In [22]:
```
# Date and time functions
rng = pd.date_range('2024-01-01', periods=10, freq='D')
print(rng)

ts = pd.Series(np.random.randn(len(rng)), index=rng)
print(ts)
```

```
DatetimeIndex(['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-
04',
               '2024-01-05', '2024-01-06', '2024-01-07', '2024-01-
08',
               '2024-01-09', '2024-01-10'],
              dtype='datetime64[ns]', freq='D')
2024-01-01   -0.939045
2024-01-02    1.104689
2024-01-03    1.043026
2024-01-04   -0.384542
2024-01-05    0.299754
2024-01-06    0.356170
2024-01-07    2.105780
2024-01-08   -0.874885
2024-01-09    0.194465
2024-01-10   -0.669901
Freq: D, dtype: float64
```

## 6.2 Resampling

Resampling changes the frequency of your time series data.

In [23]:
```
# Resampling
ts_resampled = ts.resample('5D').mean()
print(ts_resampled)
```

```
2024-01-01    0.224776
2024-01-06    0.222326
Freq: 5D, dtype: float64
```

## 6.3 Time Zone Handling

Pandas makes it easy to convert date and time data to different time zones.

In [24]:
```
# Time zone handling
ts_utc = ts.tz_localize('UTC')
print(ts_utc)

ts_est = ts_utc.tz_convert('US/Eastern')
print(ts_est)
```

```
2024-01-01 00:00:00+00:00   -0.939045
2024-01-02 00:00:00+00:00    1.104689
2024-01-03 00:00:00+00:00    1.043026
2024-01-04 00:00:00+00:00   -0.384542
2024-01-05 00:00:00+00:00    0.299754
2024-01-06 00:00:00+00:00    0.356170
2024-01-07 00:00:00+00:00    2.105780
2024-01-08 00:00:00+00:00   -0.874885
2024-01-09 00:00:00+00:00    0.194465
2024-01-10 00:00:00+00:00   -0.669901
Freq: D, dtype: float64
2023-12-31 19:00:00-05:00   -0.939045
2024-01-01 19:00:00-05:00    1.104689
2024-01-02 19:00:00-05:00    1.043026
2024-01-03 19:00:00-05:00   -0.384542
2024-01-04 19:00:00-05:00    0.299754
2024-01-05 19:00:00-05:00    0.356170
2024-01-06 19:00:00-05:00    2.105780
2024-01-07 19:00:00-05:00   -0.874885
2024-01-08 19:00:00-05:00    0.194465
2024-01-09 19:00:00-05:00   -0.669901
Freq: D, dtype: float64
```

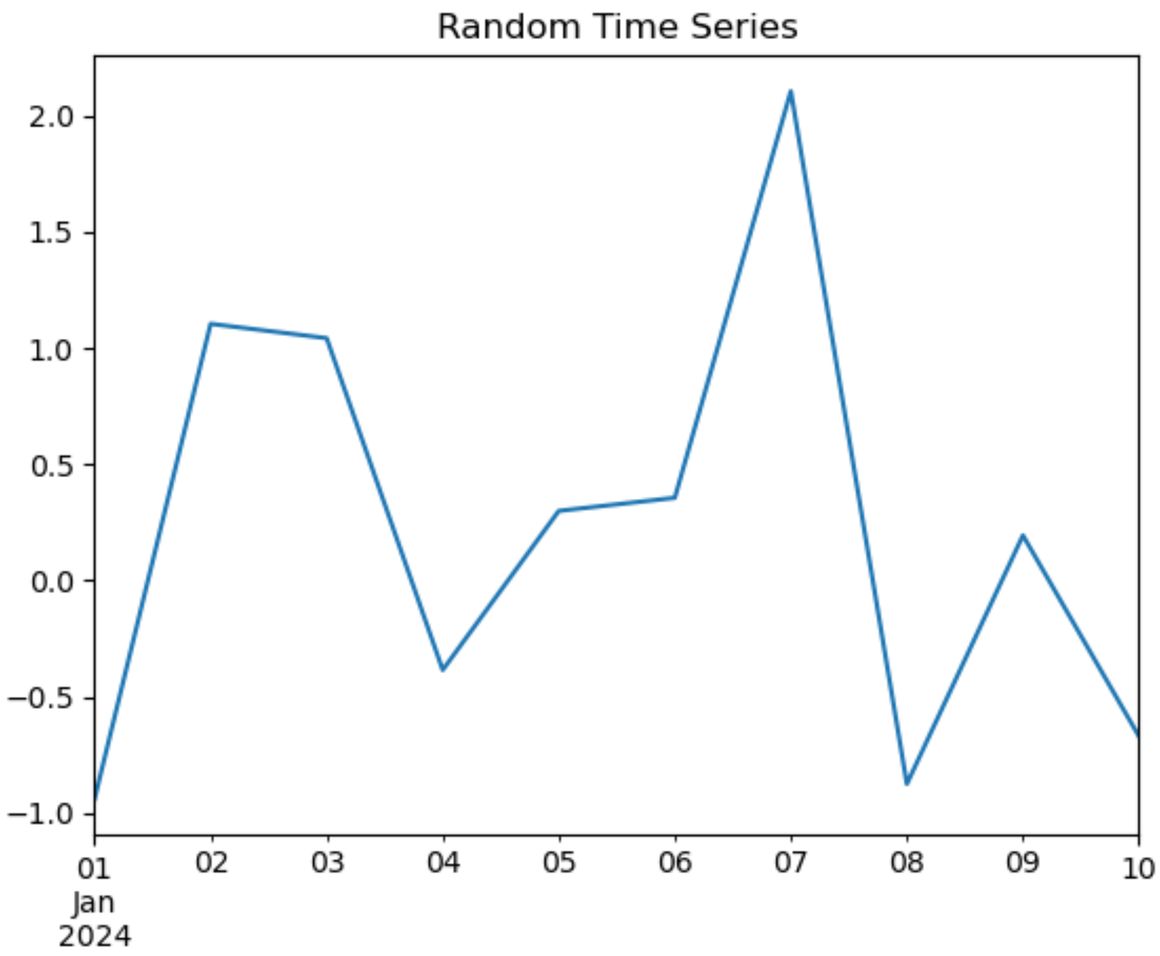
# 7. Visualization

## 7.1 Plotting with Pandas

Pandas integrates with Matplotlib to provide easy plotting functionality.

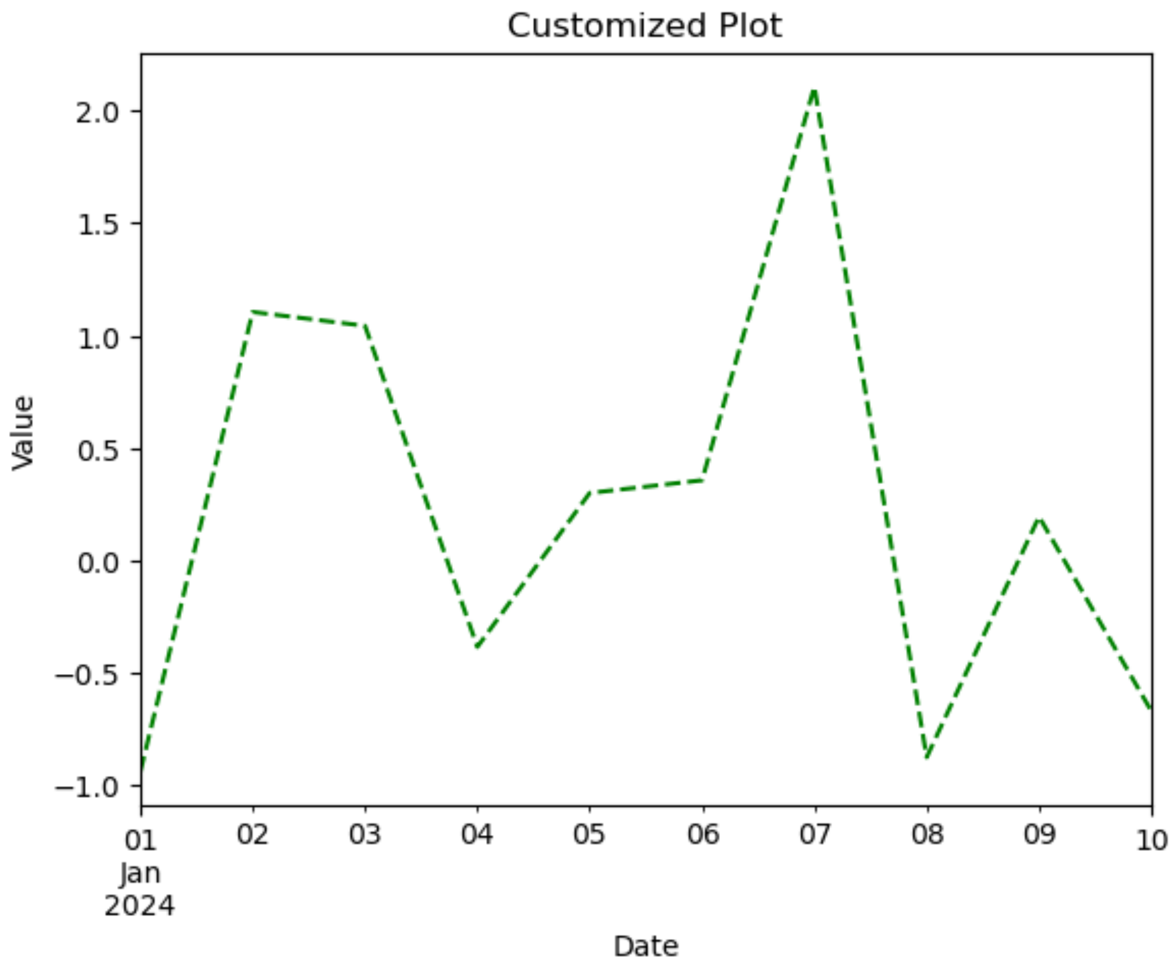
In [25]:
```
# Plotting with Pandas
import matplotlib.pyplot as plt

ts.plot(title='Random Time Series')
plt.show()
```




## 7.2 Customizing Plots

In [26]:
```
# Customizing plots
ts.plot(title='Customized Plot', color='green', style='--')
plt.xlabel('Date')
plt.ylabel('Value')
plt.show()
```




# 8. Advanced Topics

## 8.1 MultiIndex

MultiIndex allows you to work with hierarchical indexing in Pandas.

In [27]:
```
# MultiIndex
arrays = [[1, 1, 2, 2], ['red', 'blue', 'red', 'blue']]
index = pd.MultiIndex.from_arrays(arrays, names=('number', 'color')
df = pd.DataFrame({'value': [1, 2, 3, 4]}, index=index)
print(df)
```

```
              value
number color
1      red        1
       blue       2
2      red        3
       blue       4
```

### 8.2 Sparse Data

In [28]:
```
# Sparse data
s = pd.Series([0, 0, 1, 0, 2], dtype='Sparse[int]')
print(s)

df = pd.DataFrame({'A': [0, 1, 0], 'B': [1, 0, 0]}, dtype='Sparse[i
print(df)
```

```
0    0
1    0
2    1
3    0
4    2
dtype: Sparse[int64, 0]
   A  B
0  0  1
1  1  0
2  0  0
```

## 9. Working with Built-in Datasets

Pandas can work with datasets from various sources. In this section, we'll demonstrate how to use built-in datasets from popular libraries.

### 9.1 Loading Built-in Datasets

We'll use the `seaborn` library to load a built-in dataset and perform various Pandas operations on it.

In [29]:
```
import seaborn as sns
import pandas as pd

# Load the 'titanic' dataset
titanic = sns.load_dataset('titanic')
print(titanic.head())
```

```
   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
```

## 9.2 Data Overview

In [30]:
```
# Display basic information about the dataset
print(titanic.info())
# Display summary statistics
print(titanic.describe(include='all'))
# Check for missing values
print(titanic.isnull().sum())
```

```
<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
None
          survived      pclass   sex         age       sibsp      
parch  \
count   891.000000  891.000000   891  714.000000  891.000000  891.
000000
unique         NaN         NaN     2         NaN         NaN      
NaN
top            NaN         NaN  male         NaN         NaN      
NaN
freq           NaN         NaN   577         NaN         NaN      
NaN
mean      0.383838    2.308642   NaN   29.699118    0.523008    0.
381594
std       0.486592    0.836071   NaN   14.526497    1.102743    0.
806057
min       0.000000    1.000000   NaN    0.420000    0.000000    0.
000000
25%       0.000000    2.000000   NaN   20.125000    0.000000    0.
000000
50%       0.000000    3.000000   NaN   28.000000    0.000000    0.
000000
75%       1.000000    3.000000   NaN   38.000000    1.000000    0.
000000
max       1.000000    3.000000   NaN   80.000000    8.000000    6.
000000

              fare embarked  class  who adult_male deck  embark_to
wn alive  \
count   891.000000      889    891  891        891  203           8
89   891
unique         NaN        3      3    3          2    7            
3     2
top            NaN        S  Third  man       True    C  Southampt
on    no
freq           NaN      644    491  537        537   59           6
44   549
mean     32.204208      NaN    NaN  NaN        NaN  NaN           N
aN   NaN
std      49.693429      NaN    NaN  NaN        NaN  NaN           N
aN   NaN
min       0.000000      NaN    NaN  NaN        NaN  NaN           N
aN   NaN
25%       7.910400      NaN    NaN  NaN        NaN  NaN           N
aN   NaN
50%      14.454200      NaN    NaN  NaN        NaN  NaN           N
aN   NaN
75%      31.000000      NaN    NaN  NaN        NaN  NaN           N
```

```
aN   NaN
max     512.329200      NaN    NaN  NaN         NaN  NaN            N
aN   NaN

       alone
count    891
unique     2
top     True
freq     537
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
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
```

## 9.3 Handling Missing Data

In [31]:
```
# Fill missing values in 'age' with the mean age
titanic['age'].fillna(titanic['age'].mean(), inplace=True)
# Fill missing values in 'embarked' with the most frequent value
titanic['embarked'].fillna(titanic['embarked'].mode()[0], inplace=T
# Drop rows with missing 'deck' values
titanic.drop(columns=['deck'], inplace=True)
# Verify missing values are handled
print(titanic.isnull().sum())
```

```
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    2
alive          0
alone          0
dtype: int64
```

## 9.4 Data Transformation

In [32]:
```
# Convert 'sex' to numerical values
titanic['sex'] = titanic['sex'].map({'male': 0, 'female': 1})
# Convert 'embarked' to numerical values
titanic = pd.get_dummies(titanic, columns=['embarked'], drop_first=
print(titanic.head())
```

```
   survived  pclass  sex   age  sibsp  parch     fare  class    wh
o  \
0         0       3    0  22.0      1      0   7.2500  Third    ma
n
1         1       1    1  38.0      1      0  71.2833  First  woma
n
2         1       3    1  26.0      0      0   7.9250  Third  woma
n
3         1       1    1  35.0      1      0  53.1000  First  woma
n
4         0       3    0  35.0      0      0   8.0500  Third    ma
n

   adult_male  embark_town alive  alone  embarked_Q  embarked_S
0        True  Southampton    no  False           0           1
1       False    Cherbourg   yes  False           0           0
2       False  Southampton   yes   True           0           1
3       False  Southampton   yes  False           0           1
4        True  Southampton    no   True           0           1
```

## 9.5 Data Aggregation and Grouping

In [33]:
```
# Group by 'pclass' and calculate the mean age and fare
grouped = titanic.groupby('pclass').agg({'age': 'mean', 'fare': 'me
print(grouped)
```

```
              age       fare
pclass
1       37.048118  84.154687
2       29.866958  20.662183
3       26.403259  13.675550
```

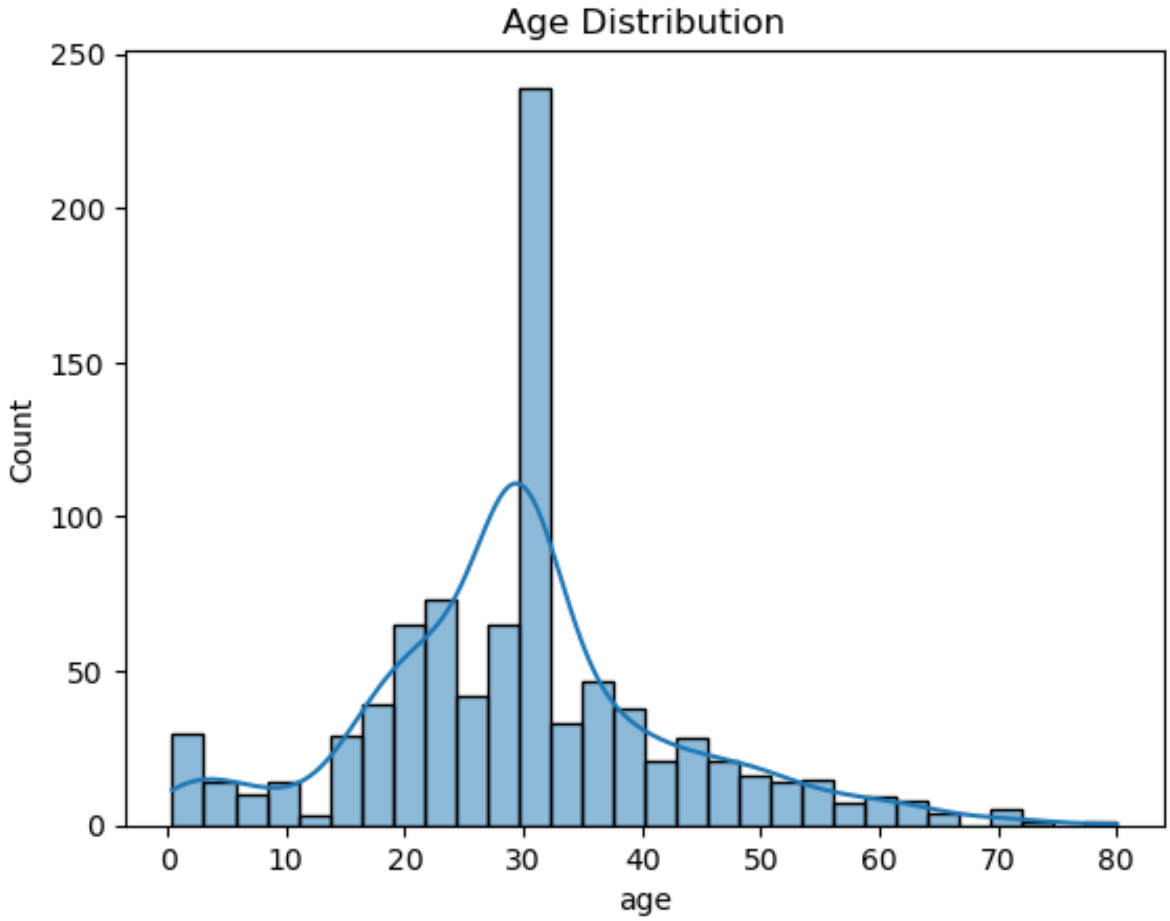
## 9.6 Data Visualization

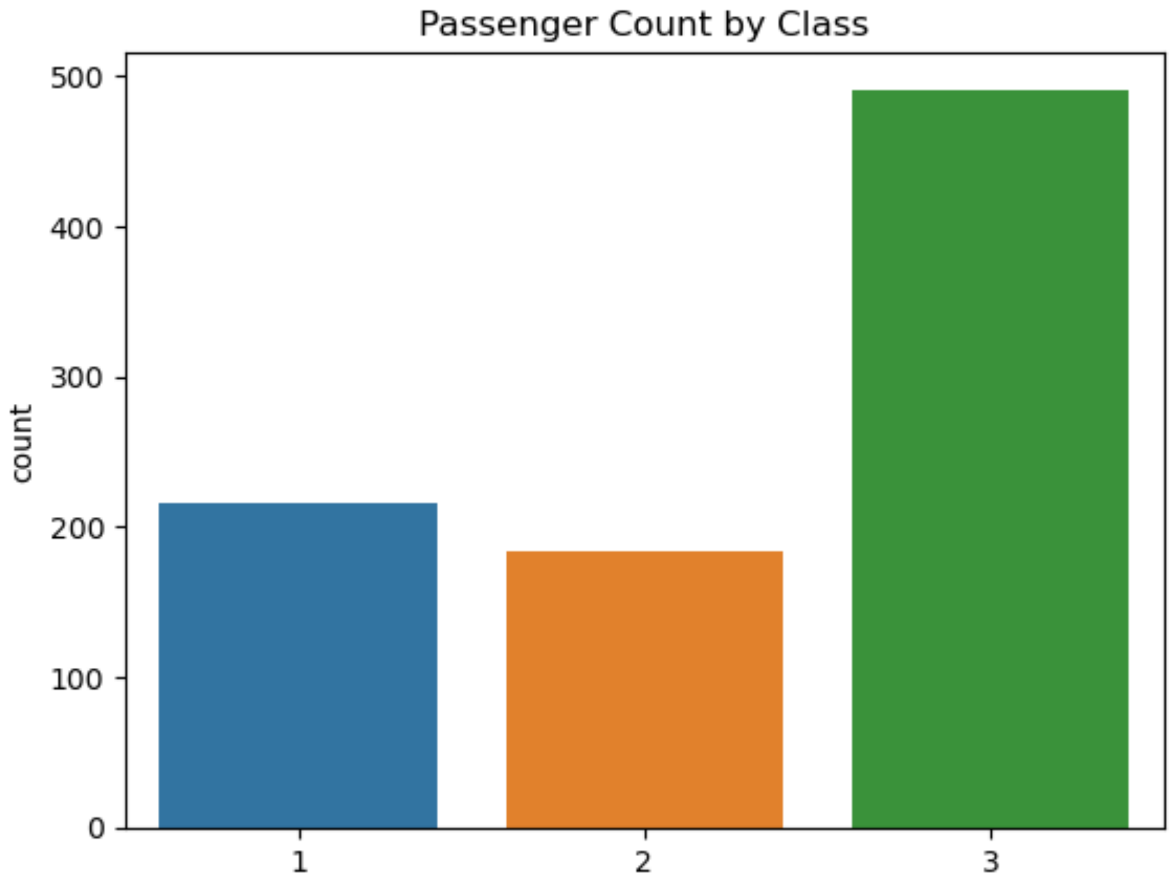
In [34]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of ages
sns.histplot(titanic['age'], kde=True)
plt.title('Age Distribution')
plt.show()

# Plot the count of passengers by class
sns.countplot(x='pclass', data=titanic)
```

```
plt.title('Passenger Count by Class')
plt.show()
```



pclass