DESCRIPTION

were are currently updating our project and you will be regularly updated and for more infomation notes,pdf,code,full png ans svg map you can follow us on following and DM on social media platforms.

`instagram` https://www.instagram.com/ahanger.asif?igsh=MXZiczBreWgwd25iMA==

`facebook` https://www.facebook.com/ahangerasifahanger.asif

In [18]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy as stats
```

In [19]:
```
df = sns.load_dataset('titanic')
df.head()
```

Out[19]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

In [20]:
```
sex = df["sex"]

sex.head()
```

Out[20]:
```
0      male
1    female
2    female
3    female
4      male
Name: sex, dtype: object
```

In [21]:
```
fare = df["fare"]
fare = fare.mode()

# fare
float(fare.iloc[0])
```

Out[21]: 8.05

In [22]:
```
df['deck'].value_counts()
```

Out[22]:
```
deck
C    59
B    47
D    33
E    32
A    15
F    13
G     4
Name: count, dtype: int64
```

In [23]:
```
# filtered = df.drop(df["deck"]=="NaN")
filtered = df.dropna(subset=["deck"])

filtered.head(2)
```

Out[23]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	

In [24]:
```
# dirt = df[df["deck"] == "NaN"]
dirt =  df[df["deck"] != "NaN"]
dirt.head()
```

Out[24]:

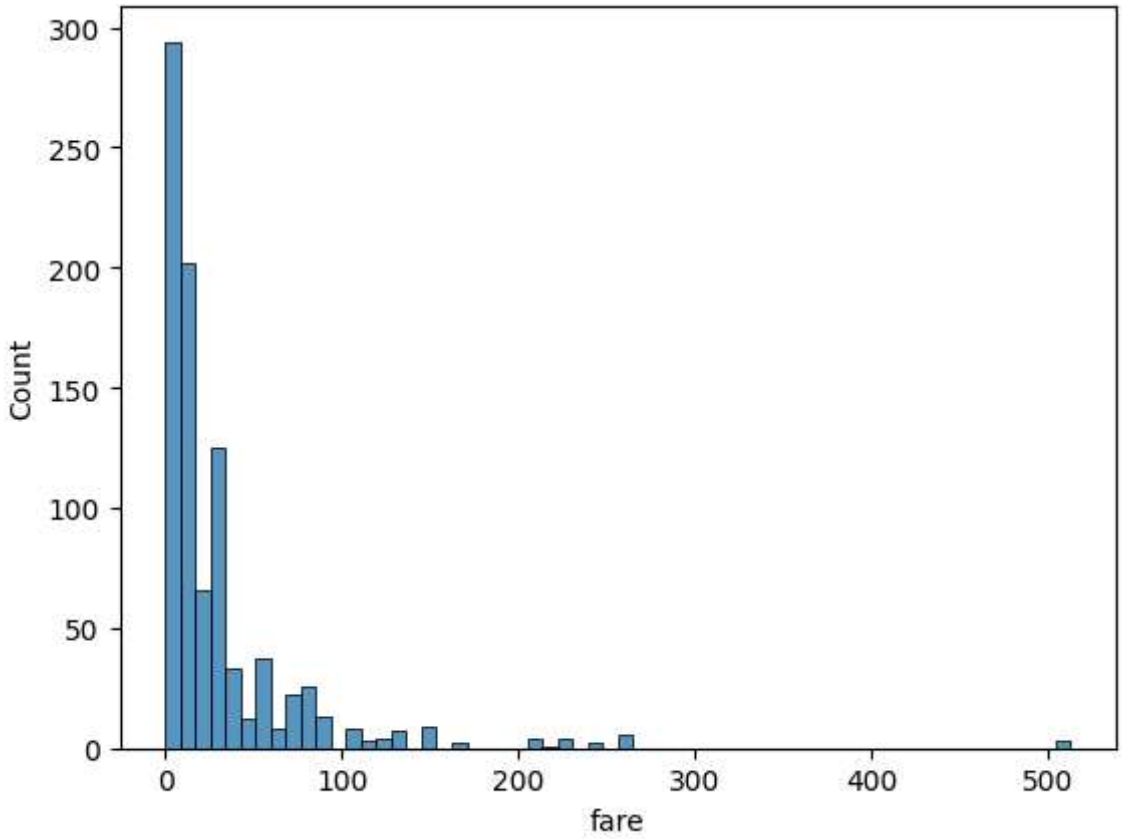
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

In [25]:
```
df["deck"].isna().sum()
```

Out[25]:
```
np.int64(688)
```

In [26]:
```
sns.histplot(df,x="fare")
```

Out[26]:
```
<Axes: xlabel='fare', ylabel='Count'>
```



In [27]:
```
df.describe()
```

Out[27]:

	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

In [28]:
```
# Lets find range of a data
# df = np.array([10, 20, 30, 40, 50])
fare = df["fare"]

max_fare = fare.max() # maximum value


np.ptp(fare)  # range (max - min)
```

Out[28]:
```
np.float64(512.3292)
```

In [31]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Generating a dataset with potential outliers
np.random.seed(0)
data = np.random.normal(100, 20, 200)
data = np.append(data, [300, 5])  # Adding potential outliers
```

In [33]:
```
Q1 = np.percentile(data,25)

Q3 = np.percentile(data,75)

IQR = Q3 - Q1
```

In [34]:
```
# Defining the outlier criteria (1.5 times the IQR)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

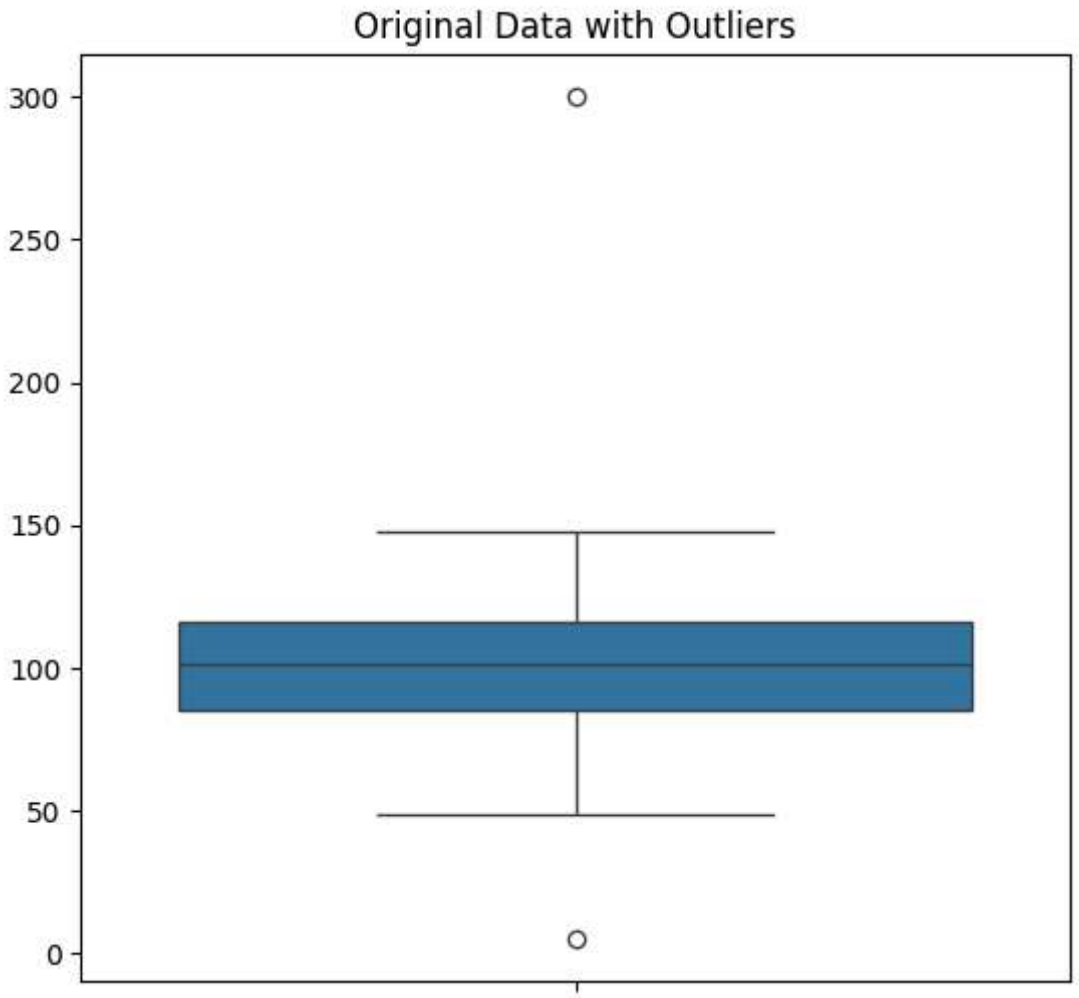
In [35]:
```
outliers = (data < lower_bound) | (data > upper_bound)

outliers
```

Out[35]:
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False,  True,  True])
```

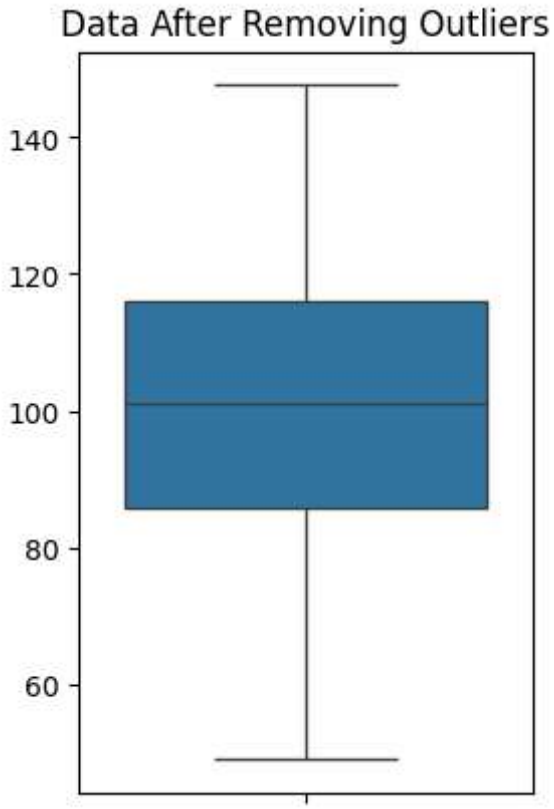
In [36]:
```
# Removing outliers
data_no_outliers = data[~outliers]
```

In [37]:
```
# Plotting the original data with outliers
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.boxplot(data=data)
plt.title('Original Data with Outliers')
```

Out[37]:
```
Text(0.5, 1.0, 'Original Data with Outliers')
```



In [38]:
```
# Plotting the data after removing outliers
plt.subplot(1, 2, 2)
sns.boxplot(data=data_no_outliers)
plt.title('Data After Removing Outliers')

plt.show()
```



In [39]:
```
# import library

import numpy as np

# create a dataset

df = np.array([2,3,4,5,6])
mean = np.mean(df)
var = np.var(df, ddof=1)
var
```

Out[39]: np.float64(2.5)

In [40]:
```
# import library
import numpy as np

df = np.array([2,3,4,5,6])

#calculate standard deviation
std = np.std(df, ddof=1)
std
print("Standard Deviation: ", std)
# calculate the standard error
se = std/np.sqrt(len(df))
print("Standard Error: ", se)
```

```
Standard Deviation:  1.5811388300841898
Standard Error:  0.7071067811865476
```

In [41]:
```
# load the data from seaborn
df = sns.load_dataset('tips')
df.head()
```

Out[41]:

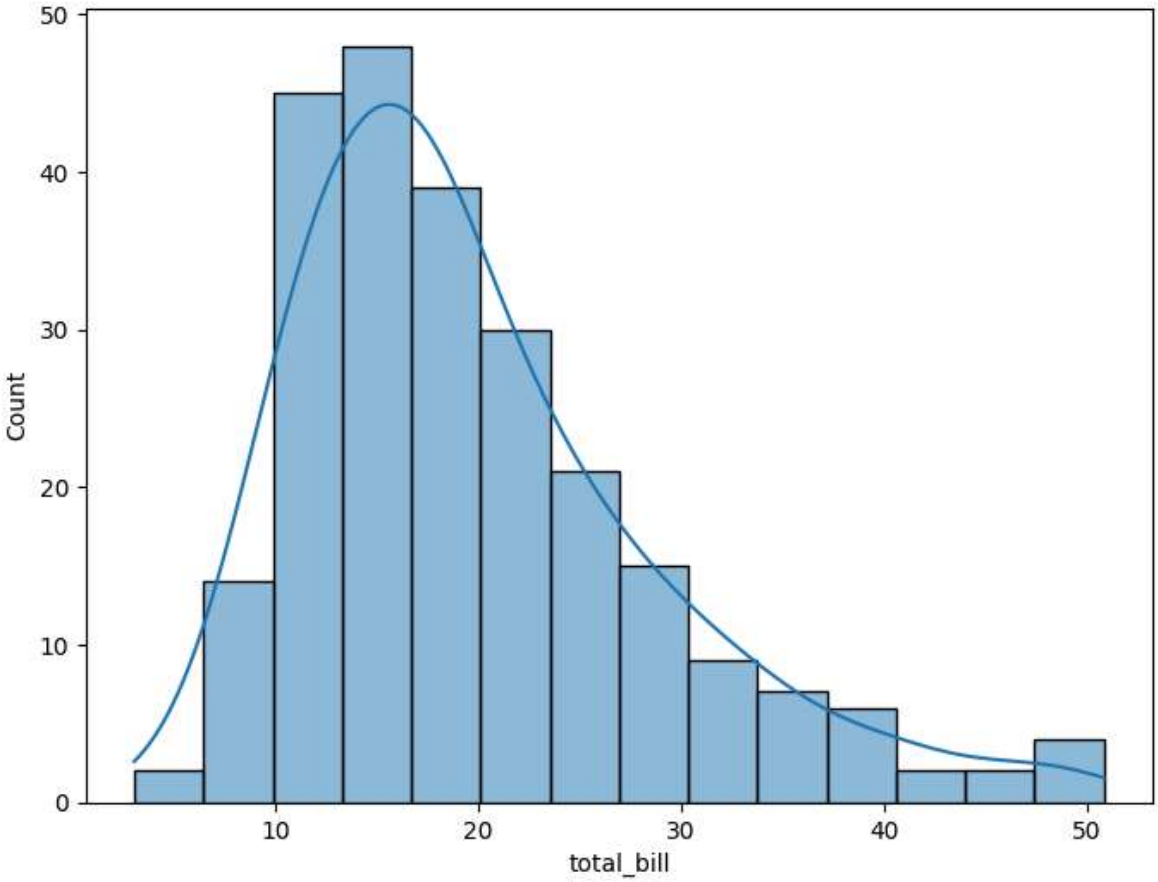
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [42]:

```
# draw a histplot curve on it
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='total_bill', kde=True)
```

Out[42]:

```
<Axes: xlabel='total_bill', ylabel='Count'>
```



In [43]:

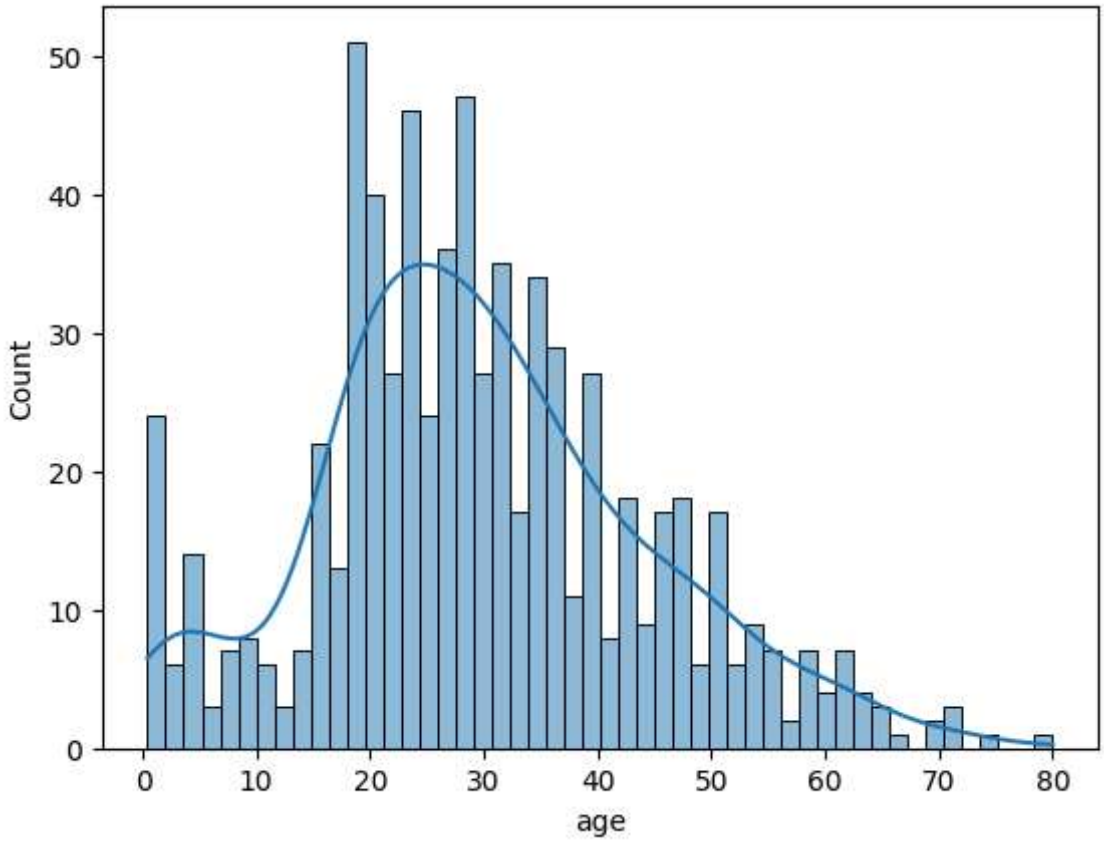
```
# load titanic data
df_01 = sns.load_dataset('titanic')
df_01.head()
```

Out[43]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

In [44]:
```
sns.histplot(data= df_01, x = 'age', kde=True, bins=50)
```

Out[44]:
```
<Axes: xlabel='age', ylabel='Count'>
```



In [45]:
```
df_01
```

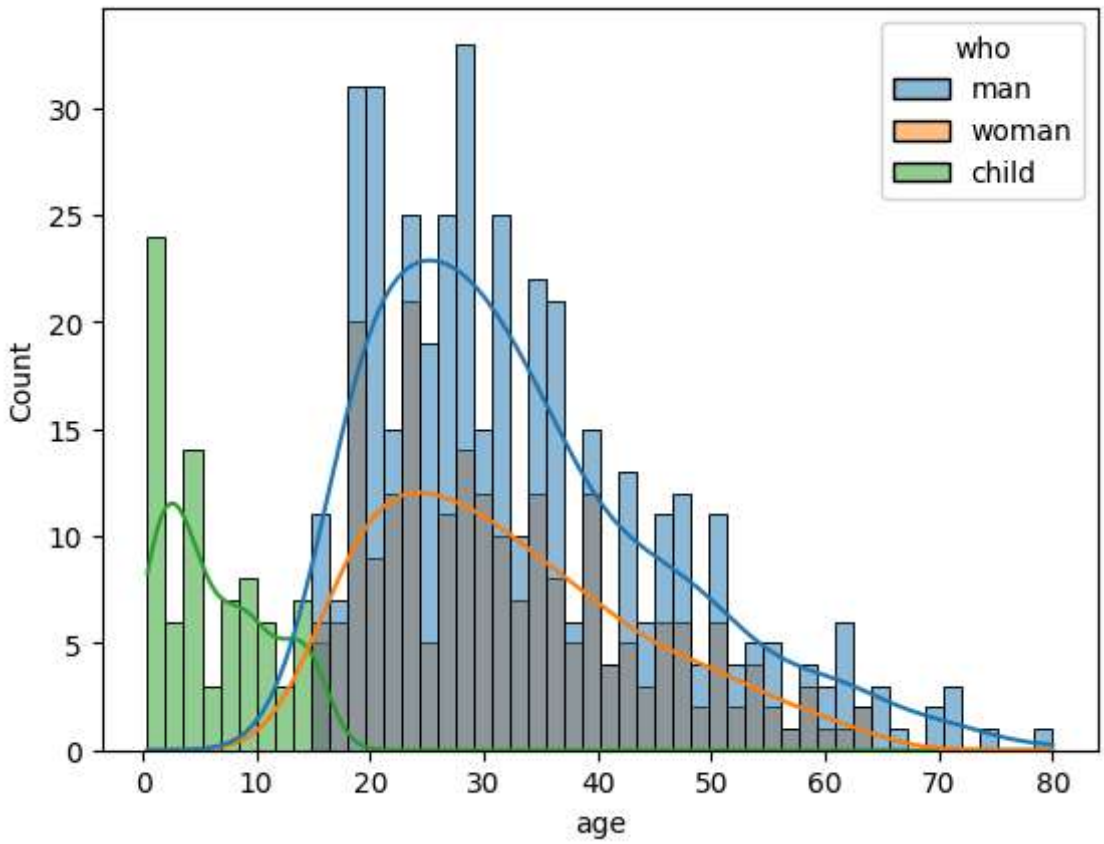
Out[45]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	0	3	male	35.0	0	0	8.0500	S	Third	man
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man
887	1	1	female	19.0	0	0	30.0000	S	First	woman
888	0	3	female	NaN	1	2	23.4500	S	Third	woman
889	1	1	male	26.0	0	0	30.0000	C	First	man
890	0	3	male	32.0	0	0	7.7500	Q	Third	man

891 rows × 15 columns

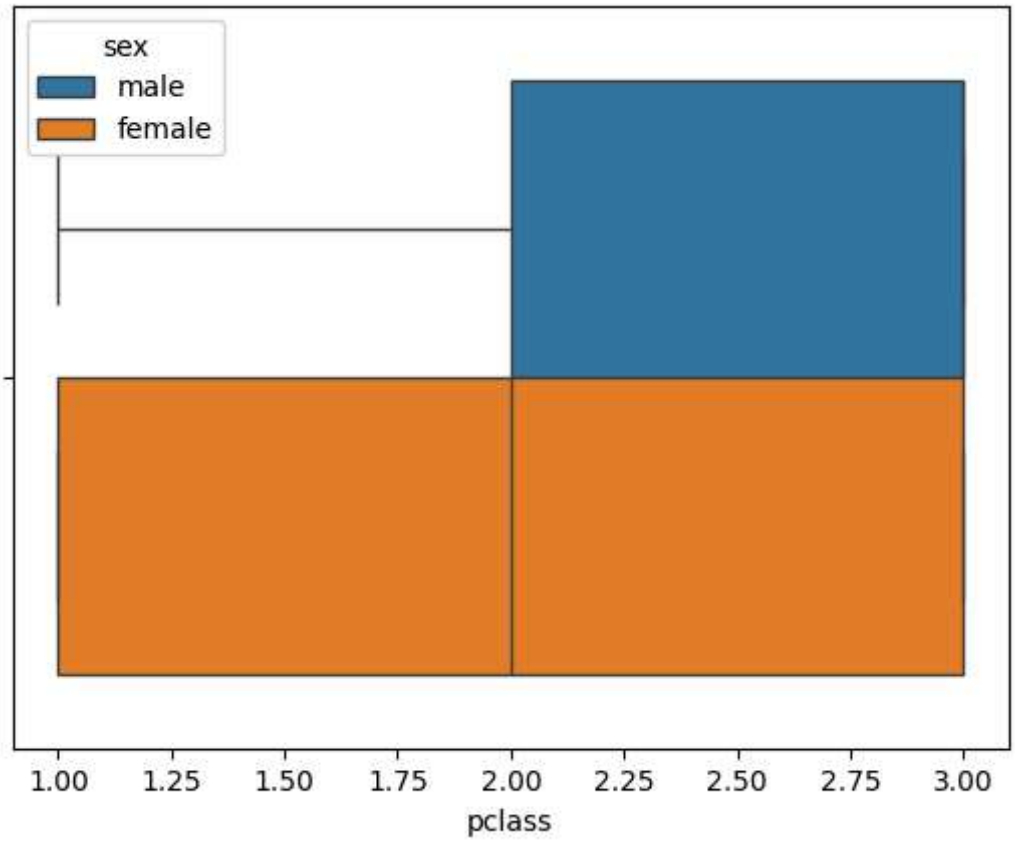
In [46]:
```
sns.histplot(data= df_01, x = 'age', kde=True, bins=50, hue='who')
```

Out[46]:
```
<Axes: xlabel='age', ylabel='Count'>
```



In [47]:
```
sns.boxplot(data=df_01,x="pclass",hue="sex")
```

Out[47]:
```
<Axes: xlabel='pclass'>
```

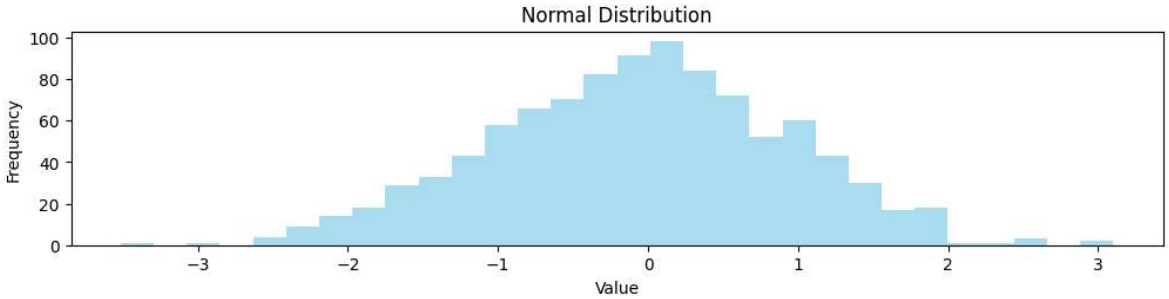


In [48]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [49]:
```
# Generating example datasets with different distributions
np.random.seed(0)  # For reproducibility
np.random.rand(3)  # above line preserves the random values so that it wount cha


# Normal distribution data
normal_data = np.random.normal(loc=0, scale=1, size=1000)

# Uniform distribution data
uniform_data = np.random.uniform(low=-2, high=2, size=1000)

# Exponential distribution data
exponential_data = np.random.exponential(scale=1, size=1000)
```

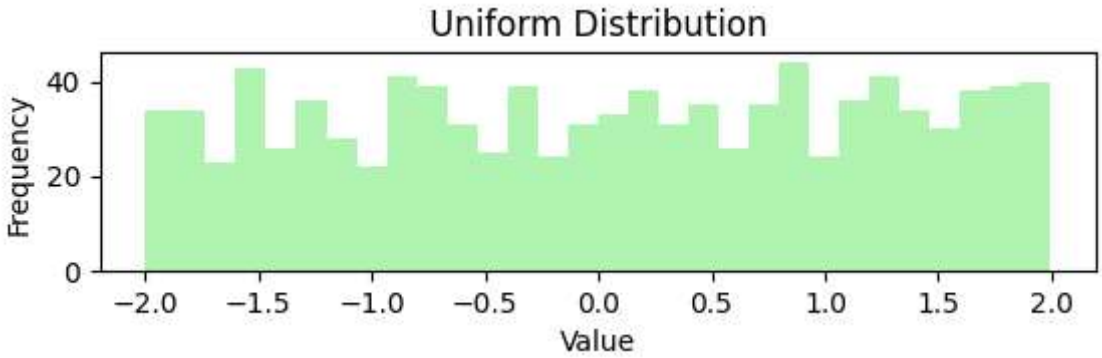
In [50]:
```
# Plotting the distributions
plt.figure(figsize=(12, 8))

# Normal Distribution Plot
plt.subplot(3, 1, 1)
plt.hist(normal_data, bins=30, color='skyblue', alpha=0.7)
plt.title('Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

Out[50]:
```
Text(0, 0.5, 'Frequency')
```



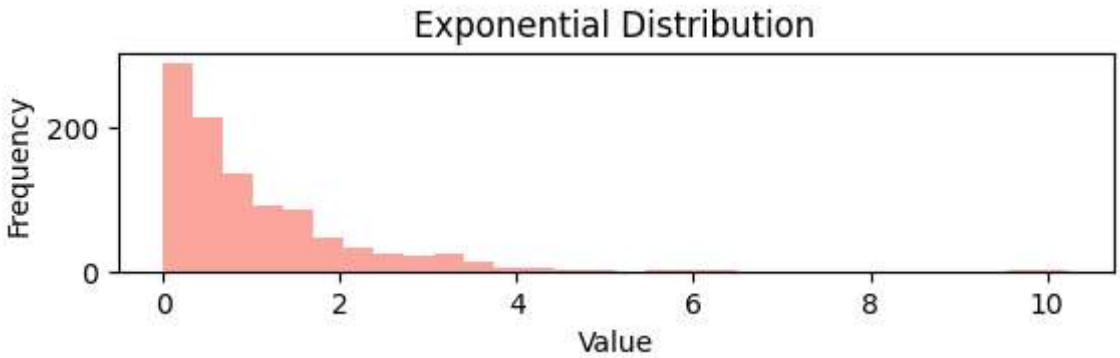
In [51]:
```
# Uniform Distribution Plot
plt.subplot(3, 1, 2)
plt.hist(uniform_data, bins=30, color='lightgreen', alpha=0.7)
plt.title('Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

Out[51]: Text(0, 0.5, 'Frequency')



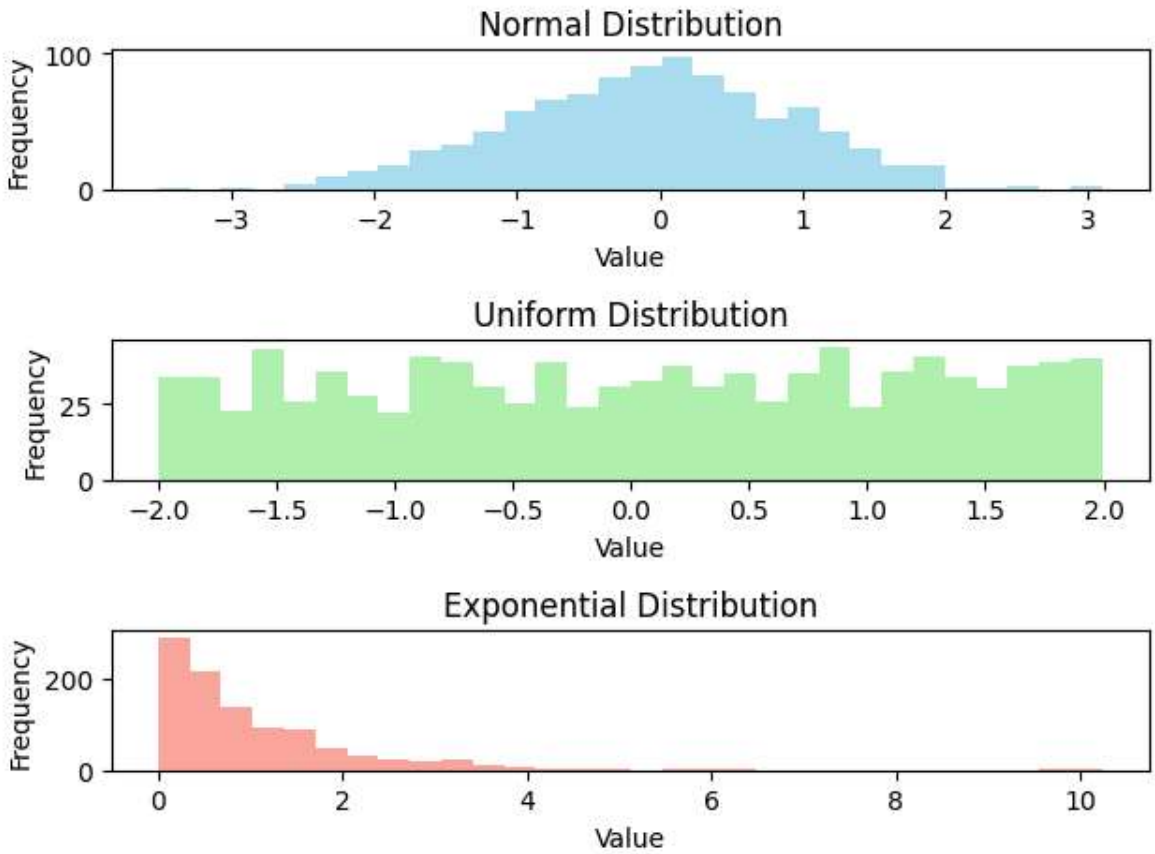
In [52]:
```
# Exponential Distribution Plot
plt.subplot(3, 1, 3)
plt.hist(exponential_data, bins=30, color='salmon', alpha=0.7)
plt.title('Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
```

Out[52]: Text(0, 0.5, 'Frequency')



In [53]:
```
# Normal Distribution Plot
plt.subplot(3, 1, 1)
plt.hist(normal_data, bins=30, color='skyblue', alpha=0.7)
plt.title('Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Uniform Distribution Plot
```

```
plt.subplot(3, 1, 2)
plt.hist(uniform_data, bins=30, color='lightgreen', alpha=0.7)
plt.title('Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Exponential Distribution Plot
plt.subplot(3, 1, 3)
plt.hist(exponential_data, bins=30, color='salmon', alpha=0.7)
plt.title('Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')

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



In [71]:
```
df[["age"]].mean()
```

Out[71]:
```
age    29.699118
dtype: float64
```

In [72]:
```
df[["age"]].median()
```

Out[72]:
```
age    28.0
dtype: float64
```

In [73]:
```
df[["age"]].mode()
```

Out[73]:

	age
0	24.0

In [74]:
```
df[["age"]].mode()
```

Out[74]:

	age
0	24.0

In [61]:
```
df_01.head()
```

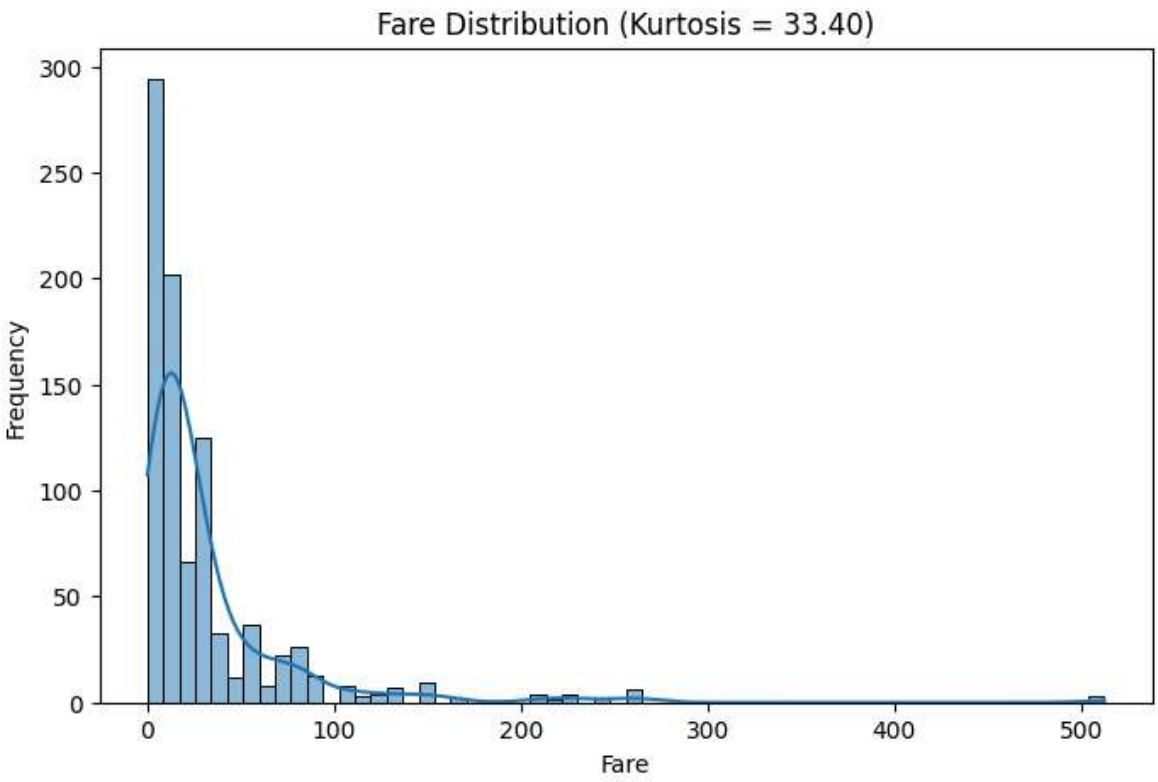
Out[61]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

Kurtosis

In [62]:
```
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
```

In [65]:
```
kurt_fare = df_01['fare'].kurtosis()

plt.figure(figsize=(8,5))
sns.histplot(df_01['fare'], kde=True,bins=60)
plt.title(f"Fare Distribution (Kurtosis = {kurt_fare:.2f})")
plt.xlabel("Fare")
plt.ylabel("Frequency")
plt.show()
```



📌 Shapiro–Wilk Test — Definition

Definition: The Shapiro–Wilk test checks whether a numerical dataset is drawn from a normal (Gaussian) distribution.

Null Hypothesis (H_0): Data is normally distributed

Alternative Hypothesis (H_1): Data is NOT normally distributed

Decision Rule

If p-value > 0.05 → Fail to reject H_0 → Data is normal

If p-value ≤ 0.05 → Reject H_0 → Data is not normal

📌 Best for small to medium datasets (n < 5000)

In [66]:
```
import numpy as np
import pandas as pd
from scipy.stats import shapiro
import seaborn as sns
import matplotlib.pyplot as plt
```

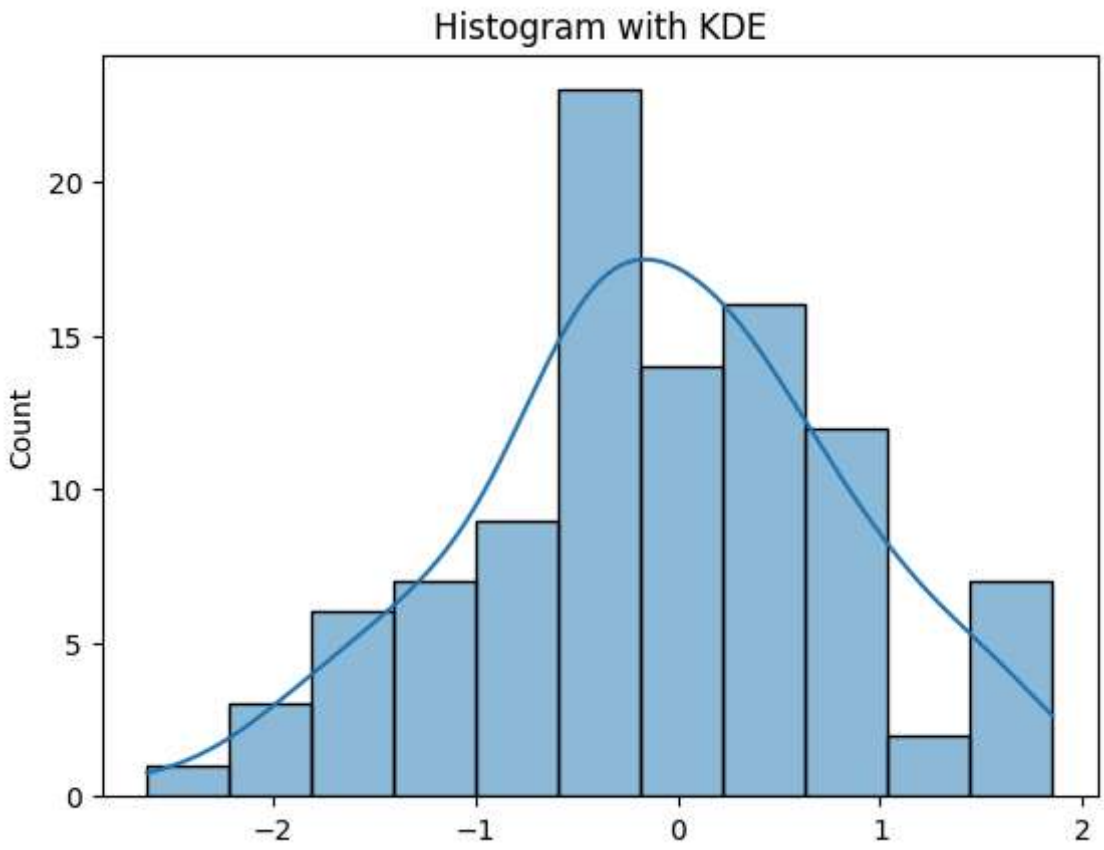
In [67]:
```
# Generate sample data
np.random.seed(42)
data = np.random.normal(loc=0, scale=1, size=100)

# Shapiro-Wilk test
stat, p_value = shapiro(data)

print("Test Statistic:", stat)
print("P-value:", p_value)
```

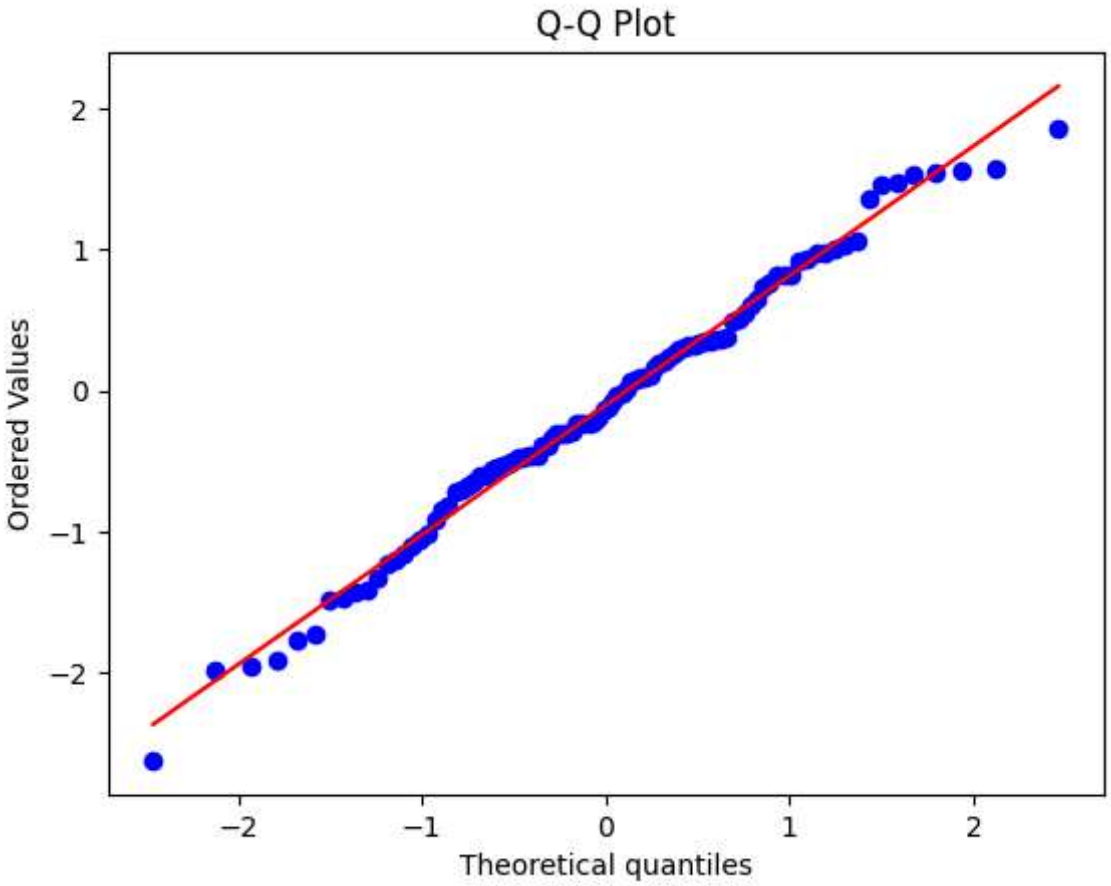
```
if p_value > 0.05:
    print("Data is normally distributed")
else:
    print("Data is NOT normally distributed")
```

```
Test Statistic: 0.9898833815158516
P-value: 0.6551676754214818
Data is normally distributed
```

In [68]:

```
sns.histplot(data, kde=True)
plt.title("Histogram with KDE")
plt.show()

import scipy.stats as stats
stats.probplot(data, plot=plt)
plt.title("Q-Q Plot")
plt.show()
```





In [69]:
```
# Load dataset
df = sns.load_dataset("titanic")

# Drop missing values
age_data = df['age'].dropna()

# Shapiro-Wilk test
stat, p_value = shapiro(age_data)

print("Statistic:", stat)
print("P-value:", p_value)

if p_value > 0.05:
    print("Age is normally distributed")
else:
    print("Age is NOT normally distributed")
```

```
Statistic: 0.9814577414504954
P-value: 7.337348958673592e-08
Age is NOT normally distributed
```

Categorical Data

│ ├── Nominal (No order)

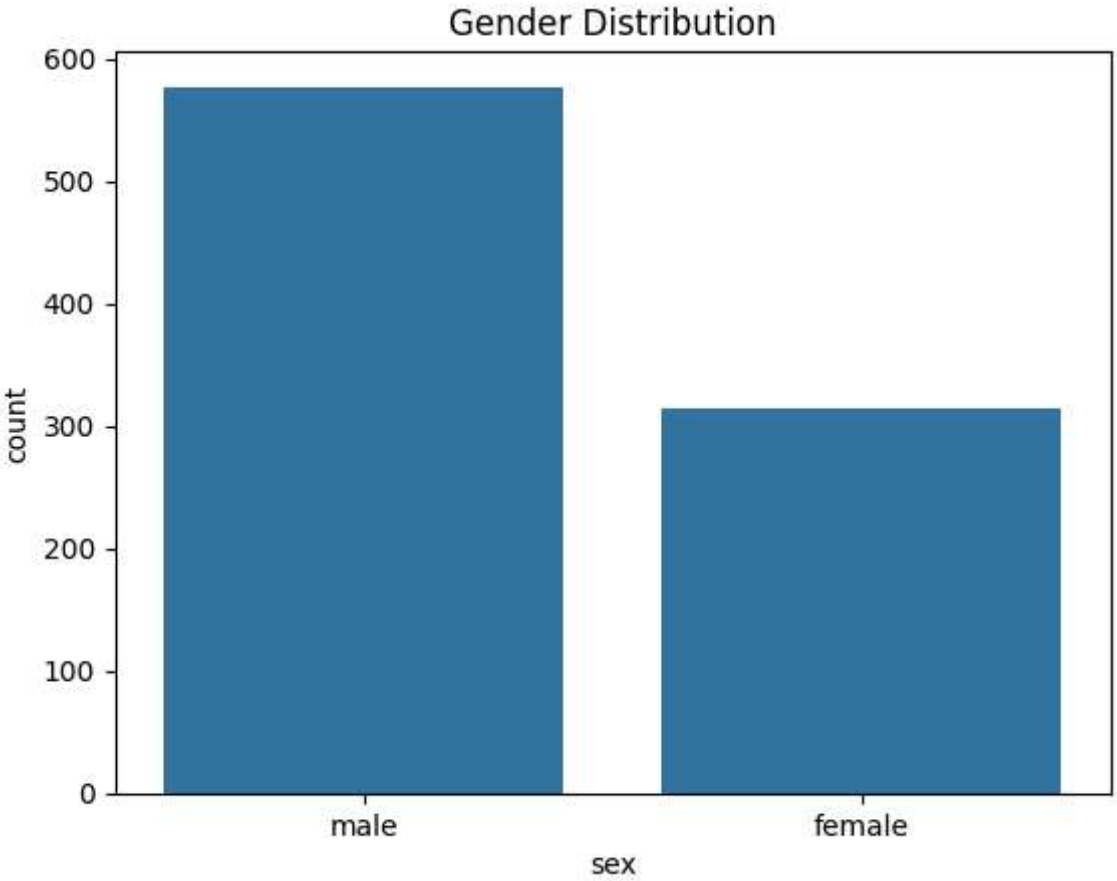
│ ├── Gender │ ├── Country │ └── Color

│ └── Ordinal (Order matters)

```
    ├── Low < Medium < High
    ├── Education level
    └── Rating (1-5)
```

Categorical Encoding │ ├── Label Encoding (Ordinal only) ├── One-Hot Encoding (Nominal) ├── Target Encoding └── Frequency Encoding

In [70]:
```
import seaborn as sns
import matplotlib.pyplot as plt

df = sns.load_dataset("titanic")

# Count plot
sns.countplot(x='sex', data=df)
plt.title("Gender Distribution")
plt.show()

# Percentage
df['sex'].value_counts(normalize=True) * 100
```



Out[70]:
```
sex
male      64.758698
female    35.241302
Name: proportion, dtype: float64
```

STATICAL ANALYSIS

chI SQUARE TEST

In [16]:
```
import numpy as np
```

```
import pandas as pd

import seaborn as sns

from scipy import stats
```

In [5]:
```
df = sns.load_dataset('titanic')

df.head()
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

null hypotheis

sex vs sirvived

In [18]:
```
# contigency table

contegency_table = pd.crosstab(df['sex'], df['survived'])
contegency_table
```

Out[18]:

survived	0	1
sex		
female	81	233
male	468	109

In [20]:
```
#chi square test

# perform chi-square test
chi2, p, dof, expected = stats.chi2_contingency(contegency_table)

# print the results
print('Chi-square statistic:', chi2)
print('p-value:', p)
print('Degrees of freedom:', dof)
print(f"Expected:\n {expected}")
```

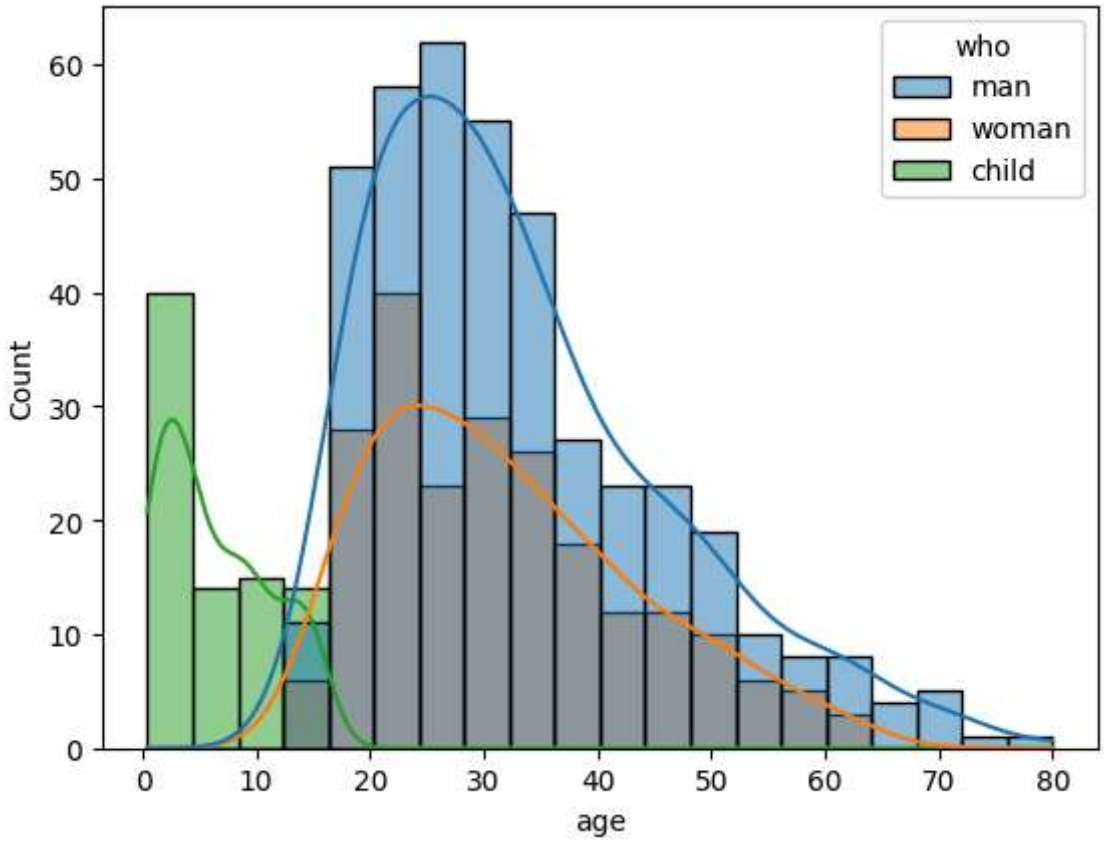
```
Chi-square statistic: 260.71702016732104
p-value: 1.1973570627755645e-58
Degrees of freedom: 1
Expected:
 [[193.47474747 120.52525253]
 [355.52525253 221.47474747]]
```

graphical Method

normal distribution

In [23]:
```
# graphical Method
sns.histplot(data=df, x='age',hue="who", kde=True)
```

Out[23]: <Axes: xlabel='age', ylabel='Count'>



shapiro wilk Test on Normal Distribution Gaussian Distribution

Null Hypothesis: The data is normally distributed.

Alternative Hypothesis: Data is not normally distributed.

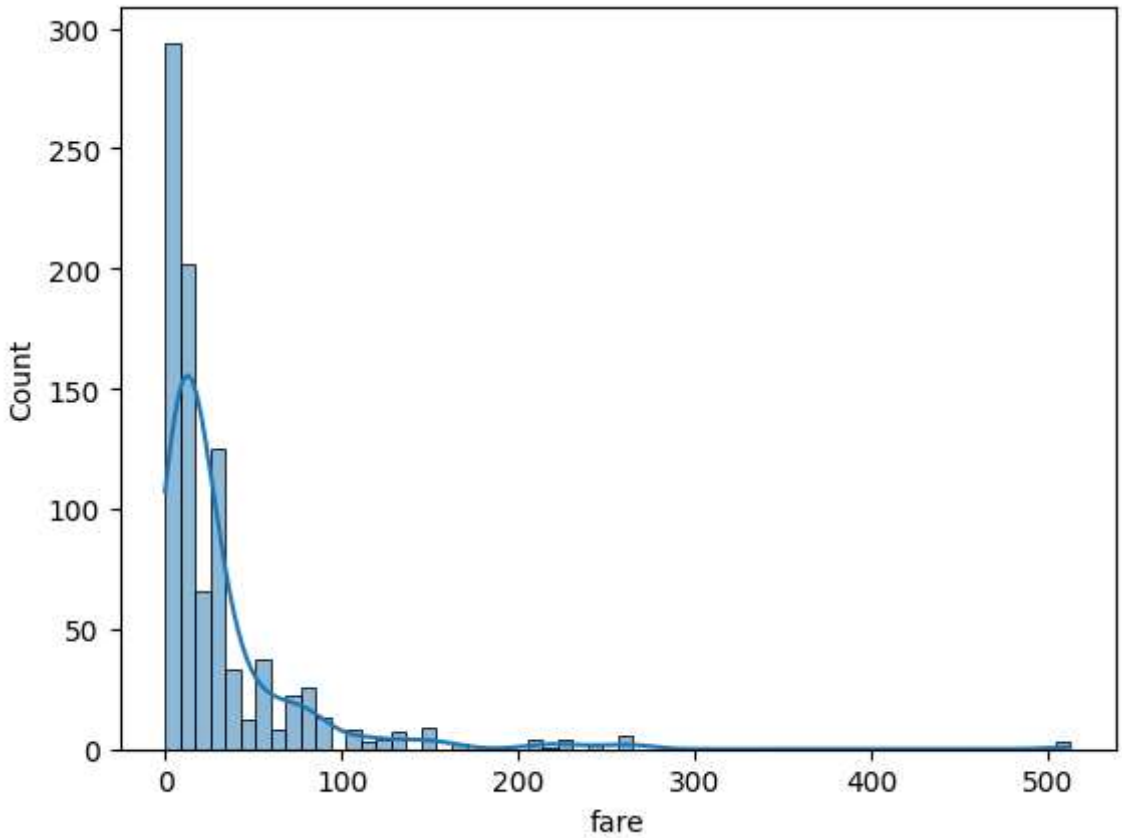
In [21]:
```
# shapiro Wilk Test
statistic, p = stats.shapiro(df['age'])

# print the results using if else
if p > 0.05:
```

```
    print(f'p-value: {p}, Sample looks Gaussian/Normally Distributed (fail to re
else:
    print(f'p-value: {p}, Sample does not look Gaussian/Normally Distributed (re
```

```
p-value: nan, Sample does not look Gaussian/Normally Distributed (reject H0)
```

In [14]:
```
sns.histplot(data=df, x='fare', kde=True)
```

Out[14]: <Axes: xlabel='fare', ylabel='Count'>



In [22]:
```
statistic, p = stats.shapiro(df['fare'])
# print the results using if else
if p > 0.05:
    print(f'p-value: {p}, Sample looks Gaussian/Normally Distributed (fail to re
else:
    print(f'p-value: {p}, Sample does not look Gaussian/Normally Distributed (re
```

```
p-value: 1.0840444395829658e-43, Sample does not look Gaussian/Normally Distribut
ed (reject H0)
```

In [24]:
```
import scipy.stats as stats

# sample data
x = [4,5,6,7,8,9,10]

# known population mean
mu = 4

# perform one sample t-test
t_statistic, p = stats.ttest_1samp(x, mu)

# print results
print('t-statistic:', t_statistic)
print('p-value:', p)
```

```
# print the results using if else
if p > 0.05:
    print(f'p-value: {p}, Sample mean is equal to population mean (fail to rejec
else:
    print(f'p-value: {p}, Sample mean is not equal to population mean (reject H0
```

```
t-statistic: 3.6742346141747673
p-value: 0.010401720935464018
p-value: 0.010401720935464018, Sample mean is not equal to population mean (rejec
t H0)
```

T-test:

1. **When to Use**:
    - The population standard deviation is unknown.
    - The sample size is small (typically, n < 30).
    - Suitable for cases where the data is approximately normally distributed, especially in small samples.
2. **Characteristics**:
    - Based on the T-distribution, which accounts for the additional uncertainty due to the estimation of the population standard deviation from the sample.
    - T-distribution becomes closer to the normal distribution as the sample size increases.

- **Large Samples**: With large sample sizes, the T-test and Z-test will give similar results. This is because the T-distribution approaches the normal distribution as the sample size increases.
- **Small Samples**: When the sample size is small and the population standard deviation is unknown, the T-test is generally the appropriate choice due to its ability to account for the uncertainty in the standard deviation estimate.
- **Unknown Population Standard Deviation**: Even with large samples, if the population standard deviation is unknown and cannot be reliably estimated, a T-test is usually preferred.

In [25]:
```
#  Bartlett's Test


import scipy.stats as stats

# Sample data
group1 = [2.3, 3.4, 4.5, 6.3, 6.4]
group2 = [1.2, 2.2, 3.2, 2.2, 6.3]

# Perform Bartlett's Test
t_stat, p_val = stats.bartlett(group1, group2)

print("t-statistic:", t_stat)
print("p-value:", p_val)

# print the results using if else conditions
if p_val > 0.05:
```

```
    print(f'p-value: {p_val}, Variances are equal (fail to reject H0)')
else:
    print(f'p-value: {p_val}, Variances are not equal (reject H0)')
```

```
t-statistic: 0.029640946492450995
p-value: 0.8633074493096582
p-value: 0.8633074493096582, Variances are equal (fail to reject H0)
```

if sample < 30 t-test

else z-test

In [26]:

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.multivariate.manova import MANOVA

# Example dataset
data = {
    'Method': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],
    'MathScore': [20, 22, 21, 19, 18, 20, 22, 23, 21],
    'ScienceScore': [30, 28, 29, 33, 32, 31, 29, 27, 28]
}

df = pd.DataFrame(data)
print(df.head(10))
# Perform MANOVA
maov = MANOVA.from_formula('MathScore + ScienceScore ~ Method', data=df)
print(maov.mv_test())
```

```
  Method  MathScore  ScienceScore
0      A         20            30
1      A         22            28
2      A         21            29
3      B         19            33
4      B         18            32
5      B         20            31
6      C         22            29
7      C         23            27
8      C         21            28
                    Multivariate linear model
================================================================

----------------------------------------------------------------
       Intercept          Value   Num DF Den DF  F Value  Pr > F
----------------------------------------------------------------
          Wilks' lambda    0.0005 2.0000 5.0000 4711.5000 0.0000
         Pillai's trace    0.9995 2.0000 5.0000 4711.5000 0.0000
 Hotelling-Lawley trace 1884.6000 2.0000 5.0000 4711.5000 0.0000
    Roy's greatest root 1884.6000 2.0000 5.0000 4711.5000 0.0000
----------------------------------------------------------------

----------------------------------------------------------------
             Method         Value  Num DF  Den DF F Value Pr > F
----------------------------------------------------------------
              Wilks' lambda 0.1802 4.0000 10.0000  3.3896 0.0534
             Pillai's trace 0.8468 4.0000 12.0000  2.2031 0.1301
     Hotelling-Lawley trace 4.4000 4.0000  5.1429  5.4000 0.0444
        Roy's greatest root 4.3656 2.0000  6.0000 13.0969 0.0065
================================================================
```