In [ ]:
```
# Name of the Experiment : Pandas Buit in function; Numpy Buit in fuction- Array slicing, Ravel,Reshape,ndim and
# EX NO : 01
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 30/07/2024
```

In [1]:
```
import pandas as pd
import numpy as np

# --- Part 1: Pandas --- #

# Create a sample DataFrame with multiple columns and rows
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [25, 30, 35, 40, 45],
    'Salary': [50000, 60000, 70000, 80000, 90000],
    'Department': ['HR', 'IT', 'Finance', 'Marketing', 'Sales']
}

df = pd.DataFrame(data)

# Display the DataFrame
print("Original DataFrame:")
print(df)
print("-" * 50)

# Access specific rows and columns using `iloc` (index-based slicing)
print("Sliced DataFrame using iloc (index-based slicing):")
print(df.iloc[1:4])  # Slicing rows 1 to 3 (0-indexed)
print("-" * 50)

# Access specific rows and columns using `loc` (label-based slicing)
print("Sliced DataFrame using loc (label-based slicing):")
print(df.loc[1:3])  # Slicing rows 1 to 3 (inclusive)
print("-" * 50)

# Access specific column(s)
print("Accessing 'Age' column:")
print(df['Age'])
print("-" * 50)

# Select multiple columns
print("Accessing multiple columns 'Name' and 'Salary':")
print(df[['Name', 'Salary']])
print("-" * 50)

# Filter DataFrame based on condition
print("Filtering DataFrame where Age > 30:")
print(df[df['Age'] > 30])
print("-" * 50)

# Add a new column with calculated values
df['Salary_in_K'] = df['Salary'] / 1000
print("DataFrame with new 'Salary_in_K' column:")
print(df)
print("-" * 50)

# Number of dimensions (ndim) of the DataFrame
print("Number of dimensions of DataFrame:")
print(df.ndim)  # Should return 2 (since it's a DataFrame)
print("-" * 50)

# --- Part 2: NumPy --- #

# Create a NumPy 2D array
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Display the original NumPy array
print("Original NumPy array:")
print(arr)
print("-" * 50)

# Reshaping the array to 1D using reshape()
reshaped_arr = arr.reshape(-1)
print("Reshaped NumPy array (1D):")
print(reshaped_arr)
print("-" * 50)

# Flatten the array using ravel()
raveled_arr = arr.ravel()
print("Raveled NumPy array:")
```

```python
print(raveled_arr)
print("-" * 50)

# Number of dimensions (ndim) of the NumPy array
print("Number of dimensions of NumPy array:")
print(arr.ndim)  # Should return 2 (since it's a 2D array)
print("-" * 50)

# Array slicing (slicing rows and columns)
print("Array slicing - Select rows 1 and 2, columns 0 and 1:")
print(arr[1:3, 0:2])  # Slicing rows 1 to 2, and columns 0 to 1
print("-" * 50)

# Transpose the array
transposed_arr = arr.T
print("Transposed NumPy array:")
print(transposed_arr)
print("-" * 50)

# Operations on NumPy array (addition, multiplication)
print("Element-wise addition (arr + 10):")
print(arr + 10)
print("-" * 50)

print("Element-wise multiplication (arr * 2):")
print(arr * 2)
print("-" * 50)

# --- Part 3: Combining Pandas and NumPy --- #

# Convert DataFrame column to NumPy array
numpy_salary = df['Salary'].to_numpy()
print("Converted 'Salary' column from DataFrame to NumPy array:")
print(numpy_salary)
print("-" * 50)

# Perform a NumPy operation on the 'Salary' column of the DataFrame
new_salaries = numpy_salary * 1.1  # Increase salary by 10%
df['Updated_Salary'] = new_salaries
print("DataFrame with updated salaries (10% increase):")
print(df)
print("-" * 50)

# Create a NumPy array from multiple columns of the DataFrame
salary_dept_arr = df[['Salary', 'Department']].to_numpy()
print("NumPy array from 'Salary' and 'Department' columns of DataFrame:")
print(salary_dept_arr)
print("-" * 50)

# --- End of Program ---
```

```
Original DataFrame:
      Name  Age  Salary Department
0    Alice   25   50000         HR
1      Bob   30   60000         IT
2  Charlie   35   70000    Finance
3    David   40   80000  Marketing
4      Eva   45   90000      Sales
--------------------------------------------------
Sliced DataFrame using iloc (index-based slicing):
      Name  Age  Salary Department
1      Bob   30   60000         IT
2  Charlie   35   70000    Finance
3    David   40   80000  Marketing
--------------------------------------------------
Sliced DataFrame using loc (label-based slicing):
      Name  Age  Salary Department
1      Bob   30   60000         IT
2  Charlie   35   70000    Finance
3    David   40   80000  Marketing
--------------------------------------------------
Accessing 'Age' column:
0    25
1    30
2    35
3    40
4    45
Name: Age, dtype: int64
--------------------------------------------------
Accessing multiple columns 'Name' and 'Salary':
      Name  Salary
0    Alice   50000
1      Bob   60000
2  Charlie   70000
```

```
3    David   80000
4      Eva   90000
--------------------------------------------------
Filtering DataFrame where Age > 30:
      Name  Age  Salary Department
2  Charlie   35   70000    Finance
3    David   40   80000  Marketing
4      Eva   45   90000      Sales
--------------------------------------------------
DataFrame with new 'Salary_in_K' column:
      Name  Age  Salary Department  Salary_in_K
0    Alice   25   50000         HR         50.0
1      Bob   30   60000         IT         60.0
2  Charlie   35   70000    Finance         70.0
3    David   40   80000  Marketing         80.0
4      Eva   45   90000      Sales         90.0
--------------------------------------------------
Number of dimensions of DataFrame:
2
--------------------------------------------------
Original NumPy array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
--------------------------------------------------
Reshaped NumPy array (1D):
[1 2 3 4 5 6 7 8 9]
--------------------------------------------------
Raveled NumPy array:
[1 2 3 4 5 6 7 8 9]
--------------------------------------------------
Number of dimensions of NumPy array:
2
--------------------------------------------------
Array slicing - Select rows 1 and 2, columns 0 and 1:
[[4 5]
 [7 8]]
--------------------------------------------------
Transposed NumPy array:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
--------------------------------------------------
Element-wise addition (arr + 10):
[[11 12 13]
 [14 15 16]
 [17 18 19]]
--------------------------------------------------
Element-wise multiplication (arr * 2):
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
--------------------------------------------------
Converted 'Salary' column from DataFrame to NumPy array:
[50000 60000 70000 80000 90000]
--------------------------------------------------
DataFrame with updated salaries (10% increase):
      Name  Age  Salary Department  Salary_in_K  Updated_Salary
0    Alice   25   50000         HR         50.0         55000.0
1      Bob   30   60000         IT         60.0         66000.0
2  Charlie   35   70000    Finance         70.0         77000.0
3    David   40   80000  Marketing         80.0         88000.0
4      Eva   45   90000      Sales         90.0         99000.0
--------------------------------------------------
NumPy array from 'Salary' and 'Department' columns of DataFrame:
[[50000 'HR']
 [60000 'IT']
 [70000 'Finance']
 [80000 'Marketing']
 [90000 'Sales']]
--------------------------------------------------
```

In [3]:
```python
import pandas as pd

# Create a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 27, 22, 32],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}

df = pd.DataFrame(data)
```

```python
# Display first few rows
print("Head of DataFrame:")
print(df.head())

# Display last few rows
print("\nTail of DataFrame:")
print(df.tail())

# Summary statistics
print("\nSummary Statistics:")
print(df.describe())

# Information about DataFrame
print("\nDataFrame Info:")
df.info()
```

```
Head of DataFrame:
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston

Tail of DataFrame:
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston

Summary Statistics:
             Age
count   4.000000
mean   26.250000
std     4.349329
min    22.000000
25%    23.500000
50%    25.500000
75%    28.250000
max    32.000000

DataFrame Info:
<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: 228.0+ bytes
```

In [5]:

```python
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Slice array (from index 2 to 7, with a step of 2)
sliced_arr = arr[2:8:2]
print("Sliced Array:", sliced_arr)
```

```
Sliced Array: [3 5 7]
```

In [7]:

```python
# Create a 2D array
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])

# Flatten the array
flat_arr = arr_2d.ravel()
print("Flattened Array:", flat_arr)
```

```
Flattened Array: [1 2 3 4 5 6]
```

In [9]:

```python
# Reshape 1D array into a 3x3 matrix
reshaped_arr = arr.reshape(3, 3)
print("Reshaped Array (3x3):\n", reshaped_arr)
```

```
Reshaped Array (3x3):
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [11]:

```python
# Check the number of dimensions
print("Number of Dimensions:", arr.ndim)
print("Number of Dimensions (2D array):", arr_2d.ndim)
```

```
Number of Dimensions: 1
Number of Dimensions (2D array): 2
```

In [13]:
```
import numpy as np

# Create an array
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Array slicing
sliced_arr = arr[2:8:2]
print("Sliced Array:", sliced_arr)

# Ravel (flatten the array)
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
flat_arr = arr_2d.ravel()
print("Flattened Array:", flat_arr)

# Reshape
reshaped_arr = arr.reshape(3, 3)
print("Reshaped Array (3x3):\n", reshaped_arr)

# Number of dimensions
print("Number of Dimensions (original array):", arr.ndim)
print("Number of Dimensions (2D array):", arr_2d.ndim)
```

```
Sliced Array: [3 5 7]
Flattened Array: [1 2 3 4 5 6]
Reshaped Array (3x3):
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Number of Dimensions (original array): 1
Number of Dimensions (2D array): 2
```

In [15]:
```
# Name of the Experiment : Outlier detection
# EX NO : 02
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 13/08/2024
```

In [17]:
```
#sample calculation for low range(lr) , upper range (ur),percentile
import numpy as np
array=np.random.randint(1,100,16) # randomly generate 16 numbers between 1 to 100
array
```

Out[17]:
```
array([76, 47, 16, 24, 21, 95,  8, 19, 25,  8, 65, 21, 85, 64, 45, 19])
```

In [21]:
```
array.mean()
```

Out[21]:
```
39.875
```

In [23]:
```
np.percentile(array,25)
```

Out[23]:
```
19.0
```

In [25]:
```
np.percentile(array,50)
```

Out[25]:
```
24.5
```

In [27]:
```
np.percentile(array,75)
```

Out[27]:
```
64.25
```

In [29]:
```
np.percentile(array,100)
```

Out[29]:
```
95.0
```

In [33]:
```
def outDetection(array):
   sorted(array)
   Q1,Q3=np.percentile(array,[25,75])
   IQR=Q3-Q1
   lr=Q1-(1.5*IQR)
   ur=Q3+(1.5*IQR)
   return lr,ur
```

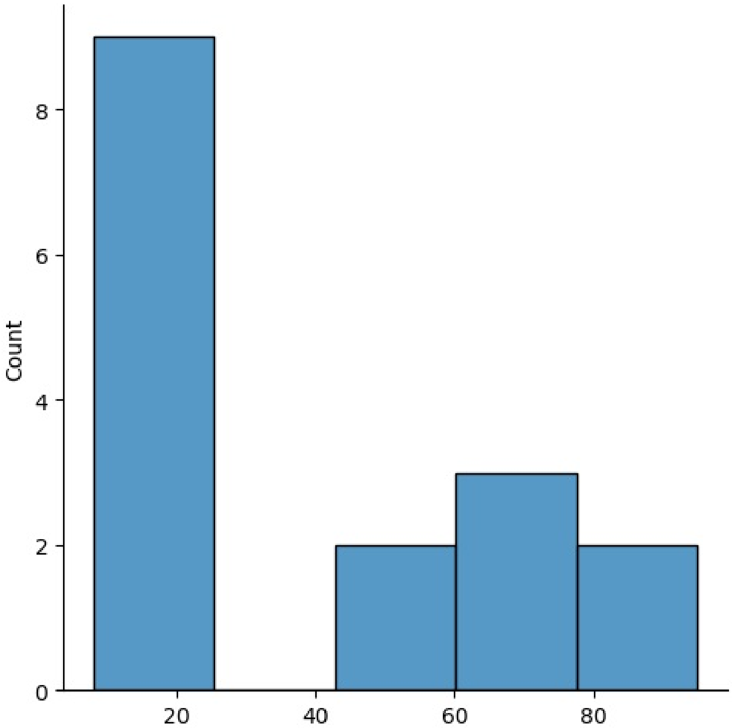
In [35]:
```
lr,ur=outDetection(array)
```

In [37]:
```
lr,ur
```

Out[37]:
```
(-48.875, 132.125)
```

In [39]:
```
import seaborn as sns
%matplotlib inline
sns.displot(array)
```

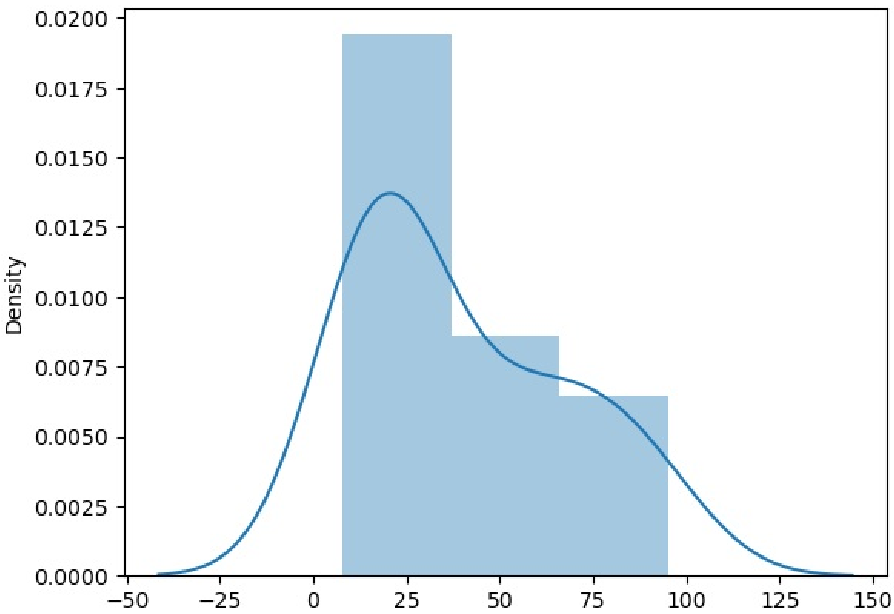
Out[39]: <seaborn.axisgrid.FacetGrid at 0x2737269b470>



In [41]:
```
sns.distplot(array)
```

```
C:\Users\chand\AppData\Local\Temp\ipykernel_8112\1133588802.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(array)
```

Out[41]: <Axes: ylabel='Density'>



In [43]:
```
new_array=array[(array>lr) & (array<ur)]
new_array
```

Out[43]: array([76, 47, 16, 24, 21, 95, 8, 19, 25, 8, 65, 21, 85, 64, 45, 19])

In [45]:
```
lr1,ur1=outDetection(new_array)
lr1,ur1
```

Out[45]: (-48.875, 132.125)

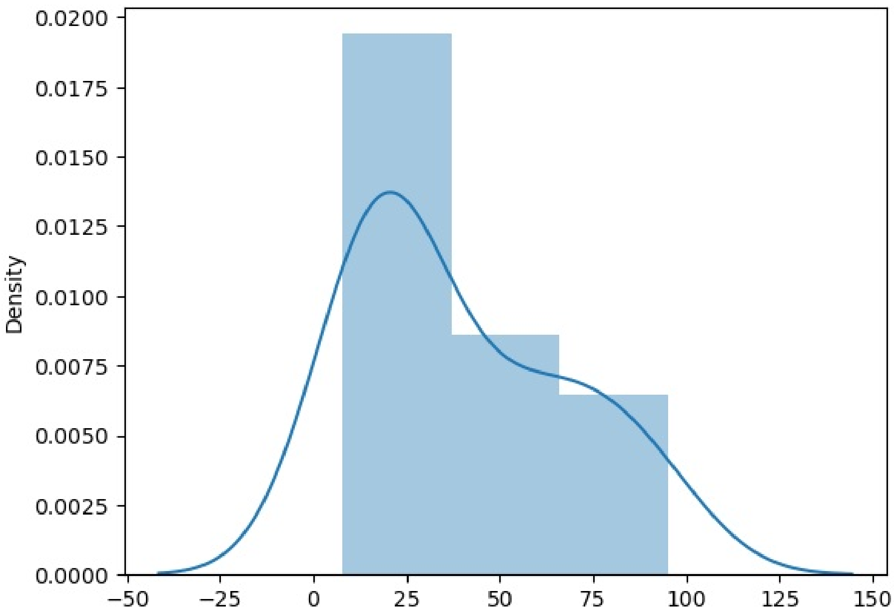
In [47]:
```
final_array=new_array[(new_array>lr1) & (new_array<ur1)]
final_array
```

Out[47]: array([76, 47, 16, 24, 21, 95, 8, 19, 25, 8, 65, 21, 85, 64, 45, 19])

In [49]:
```
sns.distplot(final_array)
```

```
C:\Users\chand\AppData\Local\Temp\ipykernel_8112\209491988.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(final_array)
```

Out[49]: <Axes: ylabel='Density'>



In [25]:
```
# Name of the Experiment : Missing and inappropriate data
# EX NO : 03
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 20/08/2024
```

In [42]:
```
import numpy as np
import pandas as pd

# Upload Hotel.csv and convert it into DataFrame
df = pd.read_csv("Hotel_Dataset.csv")
print("Original DataFrame:")
print(df)

# From the dataframe, identify duplicate rows (i.e., row 9)
print("\nChecking for duplicates:")
print(df.duplicated())

# The info() method prints information about the DataFrame, including the number of columns, column data types,
print("\nDataFrame Information:")
df.info()

# Remove duplicate rows
df.drop_duplicates(inplace=True)
print("\nDataFrame after removing duplicates:")
print(df)

# Reset index after dropping duplicate rows
print("\nResetting index after removing duplicates:")
df.reset_index(drop=True, inplace=True)
print(df)

# Use axis=1 to drop 'Age_Group.1' column from the DataFrame (if it exists)
df.drop(['Age_Group.1'], axis=1, inplace=True, errors='ignore')
print("\nDataFrame after dropping 'Age_Group.1' column:")
```

```
print(df)

# Correcting negative values in CustomerID, Bill, and EstimatedSalary using loc to avoid chained assignment
df.loc[df.CustomerID < 0, 'CustomerID'] = np.nan
df.loc[df.Bill < 0, 'Bill'] = np.nan
df.loc[df.EstimatedSalary < 0, 'EstimatedSalary'] = np.nan
print("\nDataFrame after replacing negative values with NaN:")
print(df)

# Replacing invalid 'NoOfPax' values (<1 or >20) with NaN using loc
df.loc[(df['NoOfPax'] < 1) | (df['NoOfPax'] > 20), 'NoOfPax'] = np.nan
print("\nDataFrame after replacing invalid 'NoOfPax' values with NaN:")
print(df)

# Show unique values of 'Age_Group', 'Hotel' and 'FoodPreference'
print("\nUnique values in 'Age_Group' column:")
print(df.Age_Group.unique())

print("\nUnique values in 'Hotel' column:")
print(df.Hotel.unique())

print("\nUnique values in 'FoodPreference' column:")
print(df.FoodPreference.unique())

# Replace incorrect or inconsistent values in 'Hotel' column using loc
df.loc[df.Hotel == 'Ibys', 'Hotel'] = 'Ibis'
print("\nDataFrame after replacing 'Ibys' with 'Ibis' in 'Hotel' column:")
print(df)

# Replace values in 'FoodPreference' column using loc
df.loc[df.FoodPreference.isin(['Vegetarian', 'veg']), 'FoodPreference'] = 'Veg'
df.loc[df.FoodPreference == 'non-Veg', 'FoodPreference'] = 'Non-Veg'
print("\nDataFrame after replacing inconsistent values in 'FoodPreference' column:")
print(df)

# Fill missing values in numerical columns with mean (for continuous) and median (for discrete) using loc
df.loc[:, 'EstimatedSalary'] = df['EstimatedSalary'].fillna(round(df['EstimatedSalary'].mean()))
df.loc[:, 'NoOfPax'] = df['NoOfPax'].fillna(round(df['NoOfPax'].median()))
df.loc[:, 'Rating(1-5)'] = df['Rating(1-5)'].fillna(round(df['Rating(1-5)'].median()))
df.loc[:, 'Bill'] = df['Bill'].fillna(round(df['Bill'].mean()))

# Fill missing values in categorical columns (if needed) with the mode
df.loc[:, 'Age_Group'] = df['Age_Group'].fillna(df['Age_Group'].mode()[0])
df.loc[:, 'Hotel'] = df['Hotel'].fillna(df['Hotel'].mode()[0])
df.loc[:, 'FoodPreference'] = df['FoodPreference'].fillna(df['FoodPreference'].mode()[0])

# Display final cleaned DataFrame
print("\nFinal cleaned DataFrame:")
print(df)

# Save the cleaned DataFrame to a new CSV file
df.to_csv("Cleaned_Hotel_Dataset.csv", index=False)
```

```
Original DataFrame:
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  \
0            1     20-25            4       Ibis            veg  1300
1            2     30-35            5  LemonTree        Non-Veg  2000
2            3     25-30            6     RedFox            Veg  1322
3            4     20-25           -1  LemonTree            Veg  1234
4            5       35+            3       Ibis     Vegetarian   989
5            6       35+            3       Ibys        Non-Veg  1909
6            7       35+            4     RedFox     Vegetarian  1000
7            8     20-25            7  LemonTree            Veg  2999
8            9     25-30            2       Ibis        Non-Veg  3456
9            9     25-30            2       Ibis        Non-Veg  3456
10          10     30-35            5     RedFox        non-Veg -6755

    NoOfPax  EstimatedSalary Age_Group.1
0         2            40000       20-25
1         3            59000       30-35
2         2            30000       25-30
3         2           120000       20-25
4         2            45000         35+
5         2           122220         35+
6        -1            21122         35+
7       -10           345673       20-25
8         3           -99999       25-30
9         3           -99999       25-30
10        4            87777       30-35

Checking for duplicates:
0     False
1     False
2     False
```

```
3     False
4     False
5     False
6     False
7     False
8     False
9      True
10    False
dtype: bool

DataFrame Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11 entries, 0 to 10
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CustomerID       11 non-null     int64 
 1   Age_Group        11 non-null     object
 2   Rating(1-5)      11 non-null     int64 
 3   Hotel            11 non-null     object
 4   FoodPreference   11 non-null     object
 5   Bill             11 non-null     int64 
 6   NoOfPax          11 non-null     int64 
 7   EstimatedSalary  11 non-null     int64 
 8   Age_Group.1      11 non-null     object
dtypes: int64(5), object(4)
memory usage: 924.0+ bytes

DataFrame after removing duplicates:
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  \
0            1     20-25            4       Ibis            veg  1300   
1            2     30-35            5  LemonTree        Non-Veg  2000   
2            3     25-30            6     RedFox            Veg  1322   
3            4     20-25           -1  LemonTree            Veg  1234   
4            5       35+            3       Ibis     Vegetarian   989   
5            6       35+            3       Ibys        Non-Veg  1909   
6            7       35+            4     RedFox     Vegetarian  1000   
7            8     20-25            7  LemonTree            Veg  2999   
8            9     25-30            2       Ibis        Non-Veg  3456   
10          10     30-35            5     RedFox        non-Veg -6755   

    NoOfPax  EstimatedSalary Age_Group.1  
0         2            40000       20-25  
1         3            59000       30-35  
2         2            30000       25-30  
3         2           120000       20-25  
4         2            45000         35+  
5         2           122220         35+  
6        -1            21122         35+  
7       -10           345673       20-25  
8         3           -99999       25-30  
10        4            87777       30-35  

Resetting index after removing duplicates:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  \
0           1     20-25            4       Ibis            veg  1300        2   
1           2     30-35            5  LemonTree        Non-Veg  2000        3   
2           3     25-30            6     RedFox            Veg  1322        2   
3           4     20-25           -1  LemonTree            Veg  1234        2   
4           5       35+            3       Ibis     Vegetarian   989        2   
5           6       35+            3       Ibys        Non-Veg  1909        2   
6           7       35+            4     RedFox     Vegetarian  1000       -1   
7           8     20-25            7  LemonTree            Veg  2999      -10   
8           9     25-30            2       Ibis        Non-Veg  3456        3   
9          10     30-35            5     RedFox        non-Veg -6755        4   

   EstimatedSalary Age_Group.1  
0            40000       20-25  
1            59000       30-35  
2            30000       25-30  
3           120000       20-25  
4            45000         35+  
5           122220         35+  
6            21122         35+  
7           345673       20-25  
8           -99999       25-30  
9            87777       30-35  

DataFrame after dropping 'Age_Group.1' column:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference  Bill  NoOfPax  \
0           1     20-25            4       Ibis            veg  1300        2   
1           2     30-35            5  LemonTree        Non-Veg  2000        3   
2           3     25-30            6     RedFox            Veg  1322        2   
```

```
3          4     20-25           -1  LemonTree            Veg  1234        2
4          5       35+            3       Ibis     Vegetarian   989        2
5          6       35+            3       Ibys        Non-Veg  1909        2
6          7       35+            4     RedFox     Vegetarian  1000       -1
7          8     20-25            7  LemonTree            Veg  2999      -10
8          9     25-30            2       Ibis        Non-Veg  3456        3
9         10     30-35            5     RedFox        non-Veg -6755        4

   EstimatedSalary
0            40000
1            59000
2            30000
3           120000
4            45000
5           122220
6            21122
7           345673
8           -99999
9            87777

DataFrame after replacing negative values with NaN:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0         1.0     20-25            4       Ibis            veg  1300.0
1         2.0     30-35            5  LemonTree        Non-Veg  2000.0
2         3.0     25-30            6     RedFox            Veg  1322.0
3         4.0     20-25           -1  LemonTree            Veg  1234.0
4         5.0       35+            3       Ibis     Vegetarian   989.0
5         6.0       35+            3       Ibys        Non-Veg  1909.0
6         7.0       35+            4     RedFox     Vegetarian  1000.0
7         8.0     20-25            7  LemonTree            Veg  2999.0
8         9.0     25-30            2       Ibis        Non-Veg  3456.0
9        10.0     30-35            5     RedFox        non-Veg     NaN

   NoOfPax  EstimatedSalary
0        2          40000.0
1        3          59000.0
2        2          30000.0
3        2         120000.0
4        2          45000.0
5        2         122220.0
6       -1          21122.0
7      -10         345673.0
8        3              NaN
9        4          87777.0

DataFrame after replacing invalid 'NoOfPax' values with NaN:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0         1.0     20-25            4       Ibis            veg  1300.0
1         2.0     30-35            5  LemonTree        Non-Veg  2000.0
2         3.0     25-30            6     RedFox            Veg  1322.0
3         4.0     20-25           -1  LemonTree            Veg  1234.0
4         5.0       35+            3       Ibis     Vegetarian   989.0
5         6.0       35+            3       Ibys        Non-Veg  1909.0
6         7.0       35+            4     RedFox     Vegetarian  1000.0
7         8.0     20-25            7  LemonTree            Veg  2999.0
8         9.0     25-30            2       Ibis        Non-Veg  3456.0
9        10.0     30-35            5     RedFox        non-Veg     NaN

   NoOfPax  EstimatedSalary
0      2.0          40000.0
1      3.0          59000.0
2      2.0          30000.0
3      2.0         120000.0
4      2.0          45000.0
5      2.0         122220.0
6      NaN          21122.0
7      NaN         345673.0
8      3.0              NaN
9      4.0          87777.0

Unique values in 'Age_Group' column:
['20-25' '30-35' '25-30' '35+']

Unique values in 'Hotel' column:
['Ibis' 'LemonTree' 'RedFox' 'Ibys']

Unique values in 'FoodPreference' column:
['veg' 'Non-Veg' 'Veg' 'Vegetarian' 'non-Veg']

DataFrame after replacing 'Ibys' with 'Ibis' in 'Hotel' column:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0         1.0     20-25            4       Ibis            veg  1300.0
1         2.0     30-35            5  LemonTree        Non-Veg  2000.0
```

```
2         3.0     25-30            6     RedFox            Veg  1322.0
3         4.0     20-25           -1  LemonTree            Veg  1234.0
4         5.0       35+            3       Ibis     Vegetarian   989.0
5         6.0       35+            3       Ibis        Non-Veg  1909.0
6         7.0       35+            4     RedFox     Vegetarian  1000.0
7         8.0     20-25            7  LemonTree            Veg  2999.0
8         9.0     25-30            2       Ibis        Non-Veg  3456.0
9        10.0     30-35            5     RedFox        non-Veg     NaN

   NoOfPax  EstimatedSalary
0      2.0          40000.0
1      3.0          59000.0
2      2.0          30000.0
3      2.0         120000.0
4      2.0          45000.0
5      2.0         122220.0
6      NaN          21122.0
7      NaN         345673.0
8      3.0              NaN
9      4.0          87777.0

DataFrame after replacing inconsistent values in 'FoodPreference' column:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0         1.0     20-25            4       Ibis            Veg  1300.0
1         2.0     30-35            5  LemonTree        Non-Veg  2000.0
2         3.0     25-30            6     RedFox            Veg  1322.0
3         4.0     20-25           -1  LemonTree            Veg  1234.0
4         5.0       35+            3       Ibis            Veg   989.0
5         6.0       35+            3       Ibis        Non-Veg  1909.0
6         7.0       35+            4     RedFox            Veg  1000.0
7         8.0     20-25            7  LemonTree            Veg  2999.0
8         9.0     25-30            2       Ibis        Non-Veg  3456.0
9        10.0     30-35            5     RedFox        Non-Veg     NaN

   NoOfPax  EstimatedSalary
0      2.0          40000.0
1      3.0          59000.0
2      2.0          30000.0
3      2.0         120000.0
4      2.0          45000.0
5      2.0         122220.0
6      NaN          21122.0
7      NaN         345673.0
8      3.0              NaN
9      4.0          87777.0

Final cleaned DataFrame:
   CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0         1.0     20-25            4       Ibis            Veg  1300.0
1         2.0     30-35            5  LemonTree        Non-Veg  2000.0
2         3.0     25-30            6     RedFox            Veg  1322.0
3         4.0     20-25           -1  LemonTree            Veg  1234.0
4         5.0       35+            3       Ibis            Veg   989.0
5         6.0       35+            3       Ibis        Non-Veg  1909.0
6         7.0       35+            4     RedFox            Veg  1000.0
7         8.0     20-25            7  LemonTree            Veg  2999.0
8         9.0     25-30            2       Ibis        Non-Veg  3456.0
9        10.0     30-35            5     RedFox        Non-Veg  1801.0

   NoOfPax  EstimatedSalary
0      2.0          40000.0
1      3.0          59000.0
2      2.0          30000.0
3      2.0         120000.0
4      2.0          45000.0
5      2.0         122220.0
6      2.0          21122.0
7      2.0         345673.0
8      3.0          96755.0
9      4.0          87777.0
```

In [38]:

```python
# Name of the Experiment : Data Preprocessing
# EX NO : 04
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 27/08/2024
```

In [56]:

```python
import numpy as np
import pandas as pd

# Create a sample dataset
data = {
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'France', 'Franc
```

```
    'Age': [44, 27, 30, 38, 40, 35, 38, 48, 50, 37],
    'Salary': [72000, 48000, 54000, 61000, 63778, 58000, 52000, 79000, 83000, 67000],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display the original dataset
print("Original DataFrame:")
print(df)

# Handling missing values (if any)
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])  # Fill missing 'Country' with mode
df['Age'] = df['Age'].fillna(df['Age'].median())  # Fill missing 'Age' with median
df['Salary'] = df['Salary'].fillna(round(df['Salary'].mean()))  # Fill missing 'Salary' with mean

# One-hot encode the 'Country' column
df_encoded = pd.concat([pd.get_dummies(df['Country']), df[['Age', 'Salary', 'Purchased']]], axis=1)

# Handle the downcasting warning for 'Purchased' column
# Option 1: Setting the option to suppress downcasting warning
pd.set_option('future.no_silent_downcasting', True)

# Replace the 'Purchased' column ('Yes'/'No' to 1/0)
df_encoded['Purchased'] = df_encoded['Purchased'].replace(['No', 'Yes'], [0, 1])

# Display the processed DataFrame
print("\nProcessed DataFrame:")
print(df_encoded)

# Additional Operations (to showcase more code)
# Calculate summary statistics
summary_stats = df_encoded.describe()

# Group by countries and calculate mean of 'Age' and 'Salary'
country_grouped = df_encoded.groupby(['France', 'Germany', 'Spain']).agg({'Age': 'mean', 'Salary': 'mean'})

# Handle missing values (replacing 'Purchased' with the mode)
df_encoded['Purchased'] = df_encoded['Purchased'].fillna(df_encoded['Purchased'].mode()[0])

# Display the summary and grouped data
print("\nSummary Statistics:")
print(summary_stats)

print("\nCountry Grouped by Average Age and Salary:")
print(country_grouped)

# Resetting the option to avoid future warnings
pd.reset_option('future.no_silent_downcasting')
```

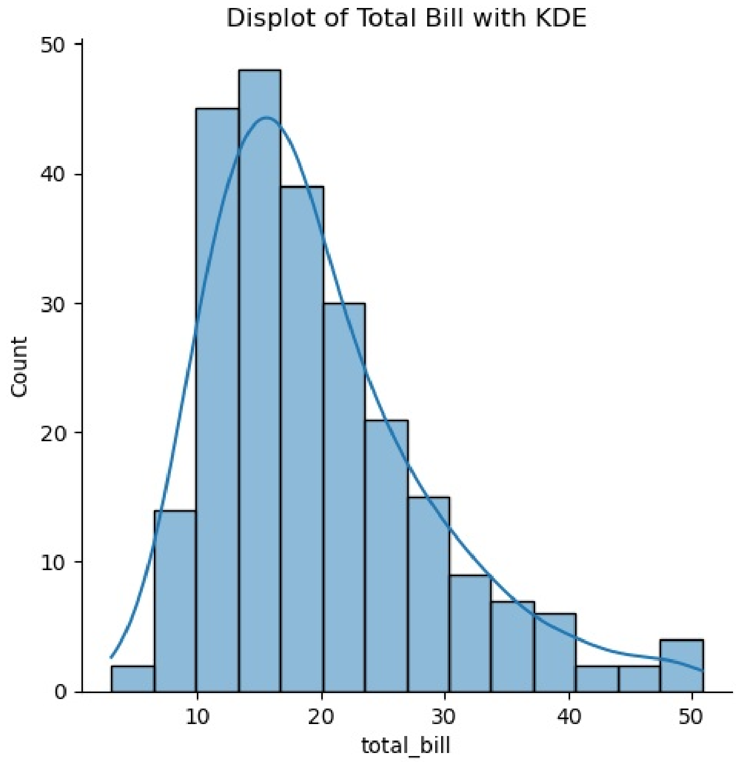
```
Original DataFrame:
   Country  Age  Salary Purchased
0   France   44   72000        No
1    Spain   27   48000       Yes
2  Germany   30   54000        No
3    Spain   38   61000        No
4  Germany   40   63778       Yes
5   France   35   58000       Yes
6    Spain   38   52000        No
7   France   48   79000       Yes
8   France   50   83000        No
9   France   37   67000       Yes

Processed DataFrame:
   France  Germany  Spain  Age  Salary  Purchased
0    True    False  False   44   72000          0
1   False    False   True   27   48000          1
2   False     True  False   30   54000          0
3   False    False   True   38   61000          0
4   False     True  False   40   63778          1
5    True    False  False   35   58000          1
6   False    False   True   38   52000          0
7    True    False  False   48   79000          1
8    True    False  False   50   83000          0
9    True    False  False   37   67000          1

Summary Statistics:
             Age        Salary
count  10.000000     10.000000
mean   38.700000  63777.800000
std     7.257946  11564.099406
min    27.000000  48000.000000
25%    35.500000  55000.000000
50%    38.000000  62389.000000
75%    43.000000  70750.000000
max    50.000000  83000.000000

Country Grouped by Average Age and Salary:
                            Age        Salary
France Germany Spain
False  False   True   34.333333  53666.666667
       True    False  35.000000  58889.000000
True   False   False  42.800000  71800.000000
```

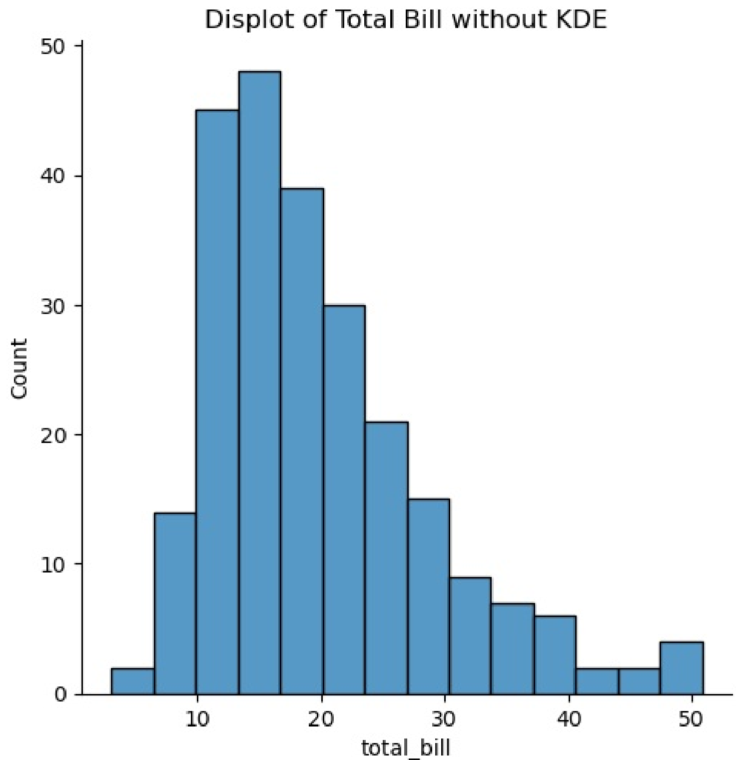
In [52]:
```python
# Name of the Experiment : EDA-Quantitative and Qualitative plots - Experiments 1
# EX NO : 05
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 03/09/2024
```

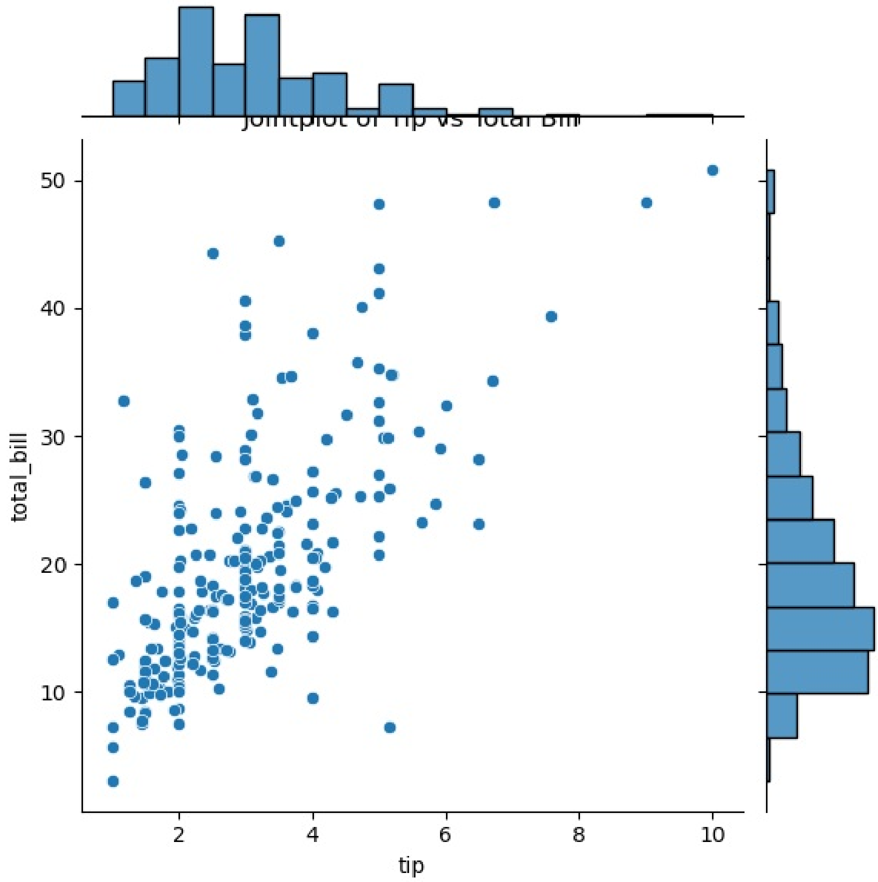
In [61]:
```python
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the 'tips' dataset from seaborn
tips = sns.load_dataset('tips')

# Display the first few rows of the dataset
print(tips.head())

# Visualization 1: Displot with KDE for the 'total_bill' column
sns.displot(tips.total_bill, kde=True)
plt.title("Displot of Total Bill with KDE")
plt.show()

# Visualization 2: Displot without KDE for the 'total_bill' column
sns.displot(tips.total_bill, kde=False)
plt.title("Displot of Total Bill without KDE")
plt.show()

# Visualization 3: Jointplot for 'tip' vs 'total_bill'
sns.jointplot(x=tips.tip, y=tips.total_bill)
plt.title("Jointplot of Tip vs Total Bill")
plt.show()

# Visualization 4: Jointplot with regression line for 'tip' vs 'total_bill'
sns.jointplot(x=tips.tip, y=tips.total_bill, kind="reg")
plt.title("Jointplot with Regression of Tip vs Total Bill")
plt.show()

# Visualization 5: Jointplot with hexbin for 'tip' vs 'total_bill'
sns.jointplot(x=tips.tip, y=tips.total_bill, kind="hex")
```

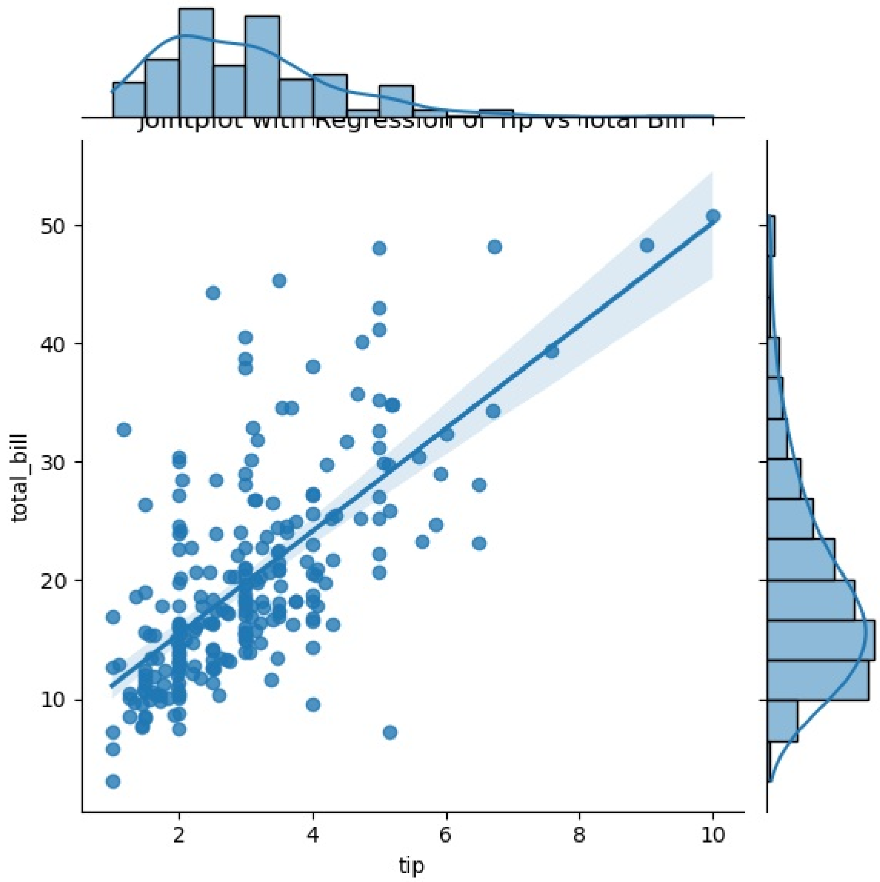
```python
plt.title("Hexbin Jointplot of Tip vs Total Bill")
plt.show()

# Visualization 6: Pairplot of all numerical columns
sns.pairplot(tips)
plt.title("Pairplot of Numerical Columns")
plt.show()

# Visualization 7: Pairplot with hue based on 'time'
sns.pairplot(tips, hue='time')
plt.title("Pairplot with Hue on Time")
plt.show()

# Visualization 8: Pairplot with hue based on 'day'
sns.pairplot(tips, hue='day')
plt.title("Pairplot with Hue on Day")
plt.show()

# Visualization 9: Heatmap of correlation matrix for numerical columns
sns.heatmap(tips.corr(numeric_only=True), annot=True)
plt.title("Heatmap of Correlation Matrix")
plt.show()

# Visualization 10: Boxplot for 'total_bill'
sns.boxplot(tips.total_bill)
plt.title("Boxplot of Total Bill")
plt.show()

# Visualization 11: Boxplot for 'tip'
sns.boxplot(tips.tip)
plt.title("Boxplot of Tip")
plt.show()

# Visualization 12: Countplot of 'day'
sns.countplot(tips.day)
plt.title("Countplot of Day")
plt.show()

# Visualization 13: Countplot of 'sex'
sns.countplot(tips.sex)
plt.title("Countplot of Sex")
plt.show()

# Visualization 14: Pie chart of 'sex' value counts
tips.sex.value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart of Sex Distribution")
plt.ylabel('')  # Hide the 'sex' label
plt.show()

# Visualization 15: Bar chart of 'sex' value counts
tips.sex.value_counts().plot(kind='bar')
plt.title("Bar Chart of Sex Distribution")
plt.show()

# Visualization 16: Countplot for 'day' based on 'time'=='Dinner'
sns.countplot(tips[tips.time=='Dinner']['day'])
plt.title("Countplot of Day for Dinner Time")
plt.show()
```

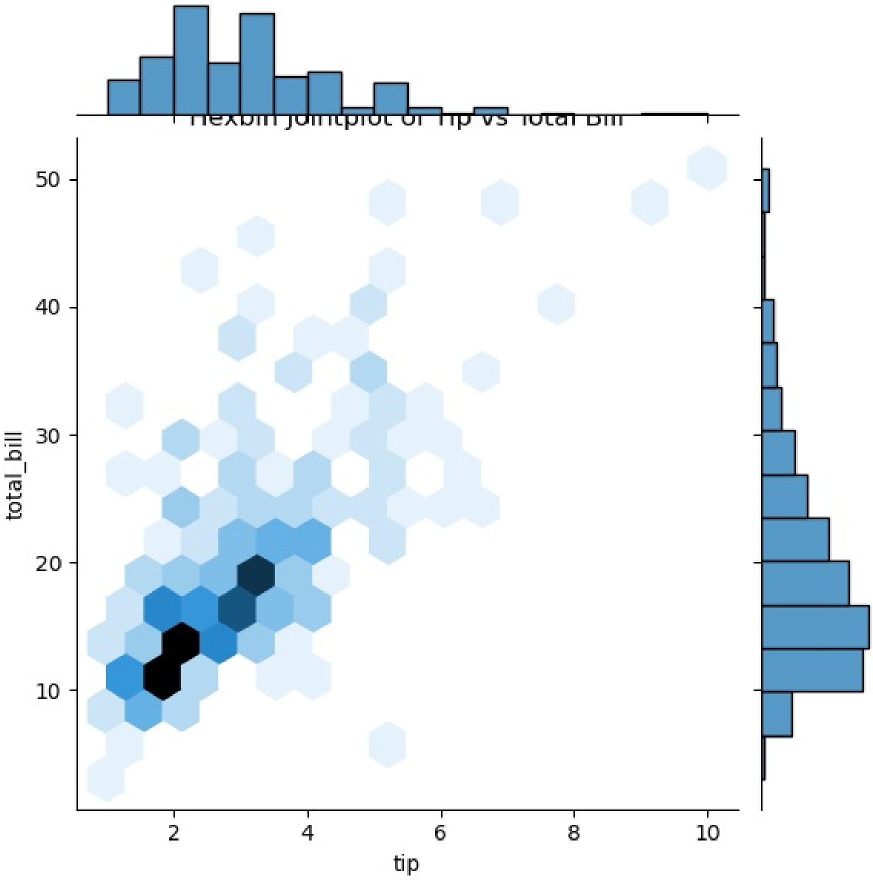
```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female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


Displot of Total Bill with KDE
Count
total_bill
0
10
20
30
40
50

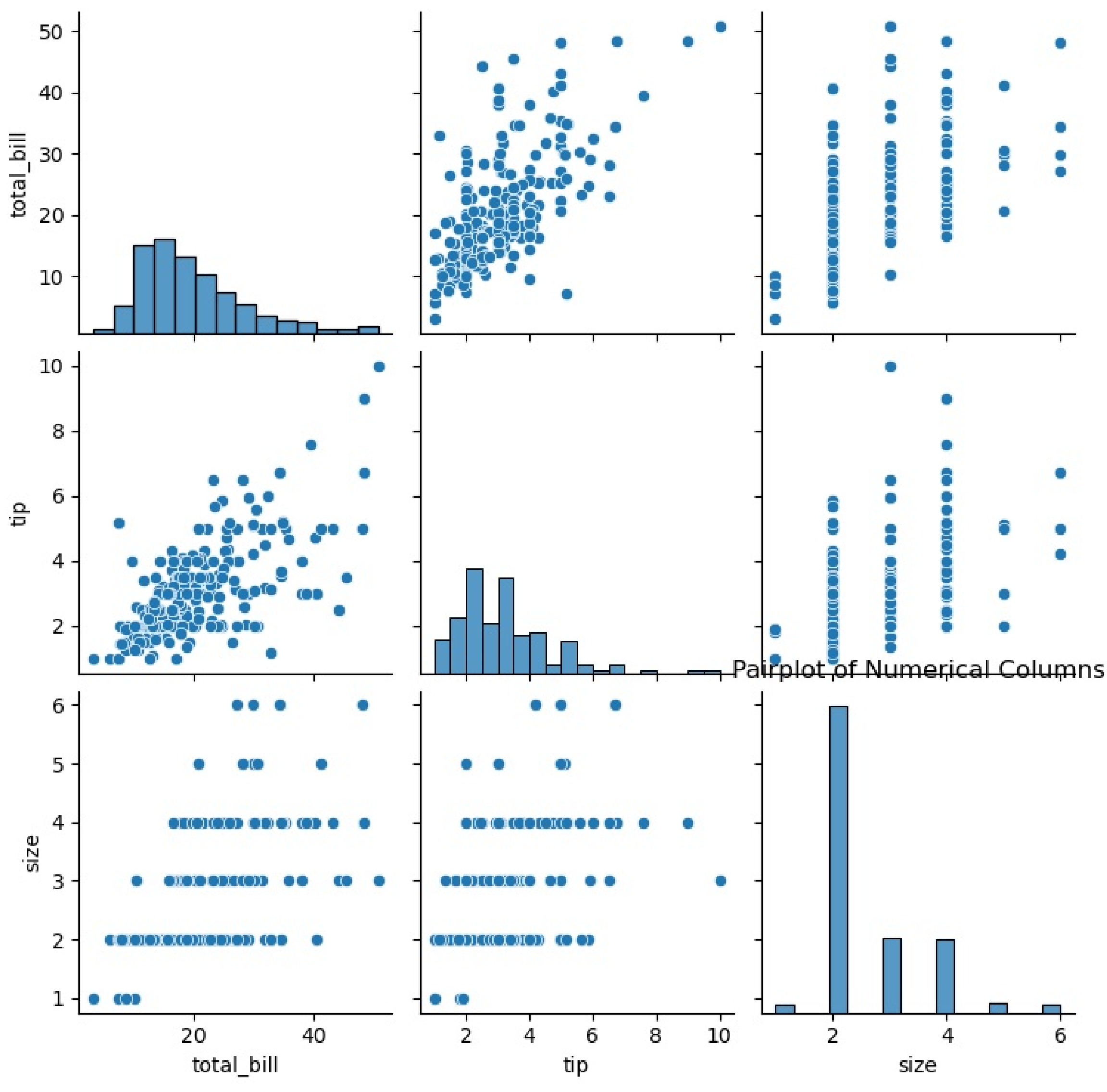

Displot of Total Bill without KDE
Count
total_bill
0
10
20
30
40
50


Jointplot of Tip vs Total Bill
total_bill
tip
50
40
30
20
10
2
4
6
8
10

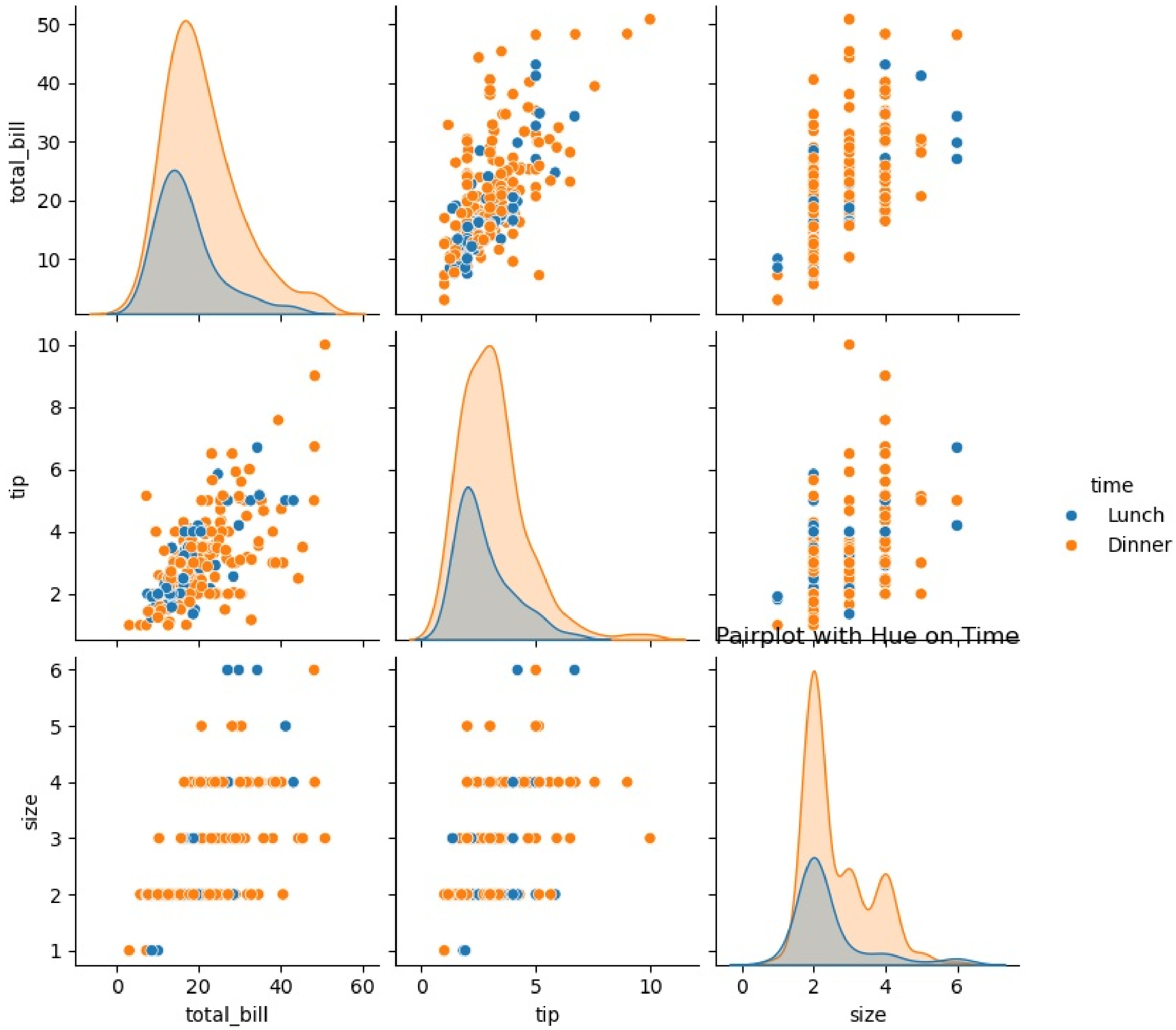

Jointplot with Regression of Tip vs Total Bill
total_bill
tip
50
40
30
20
10
2
4
6
8
10

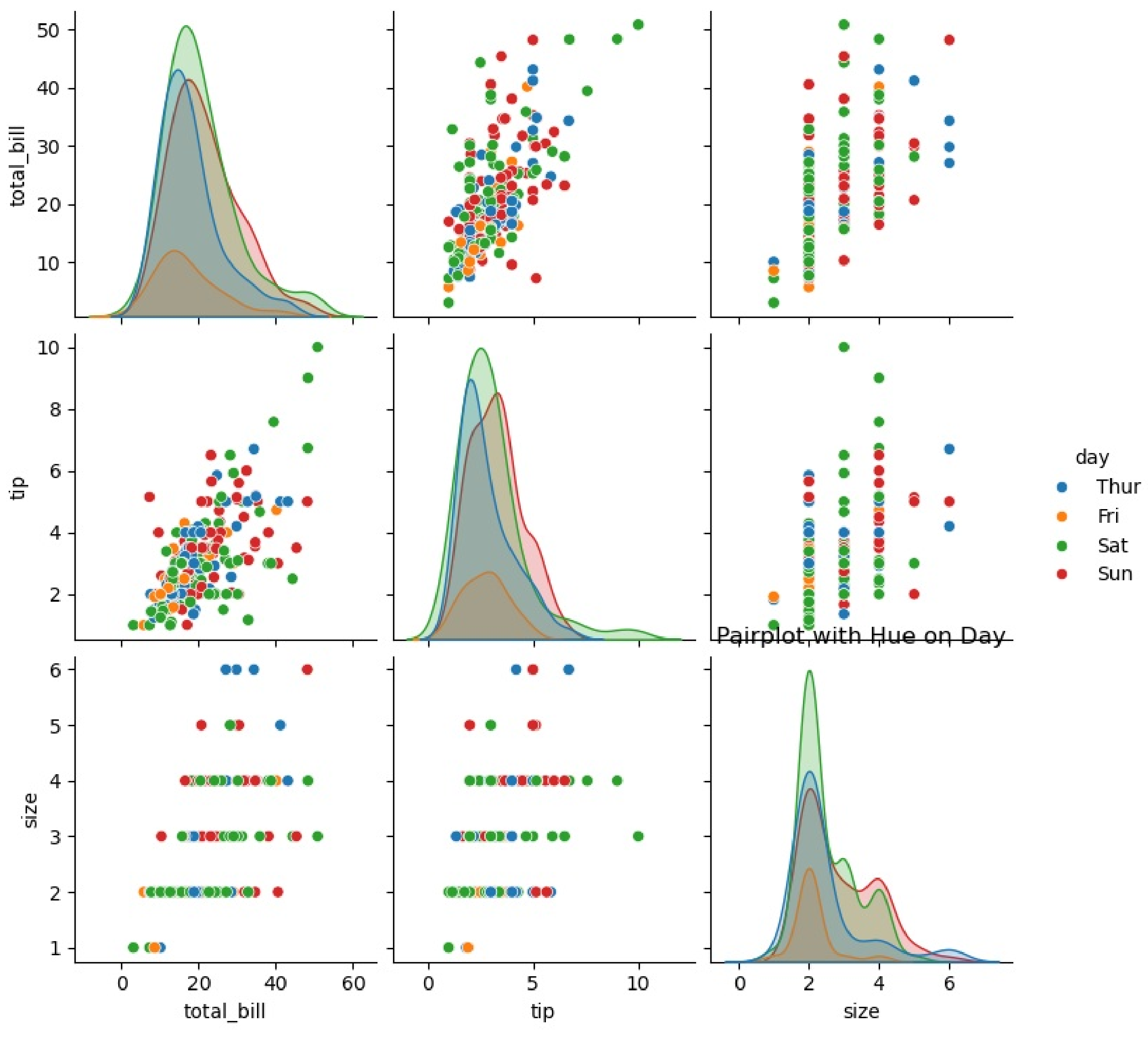


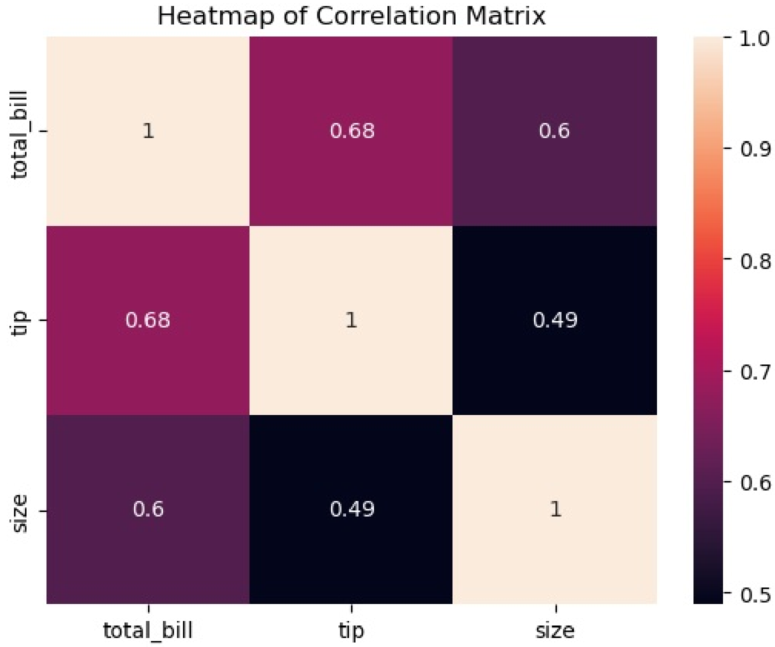
Hexbin Jointplot of Tip vs Total Bill
total_bill
tip
50
40
30
20
10
2
4
6
8
10

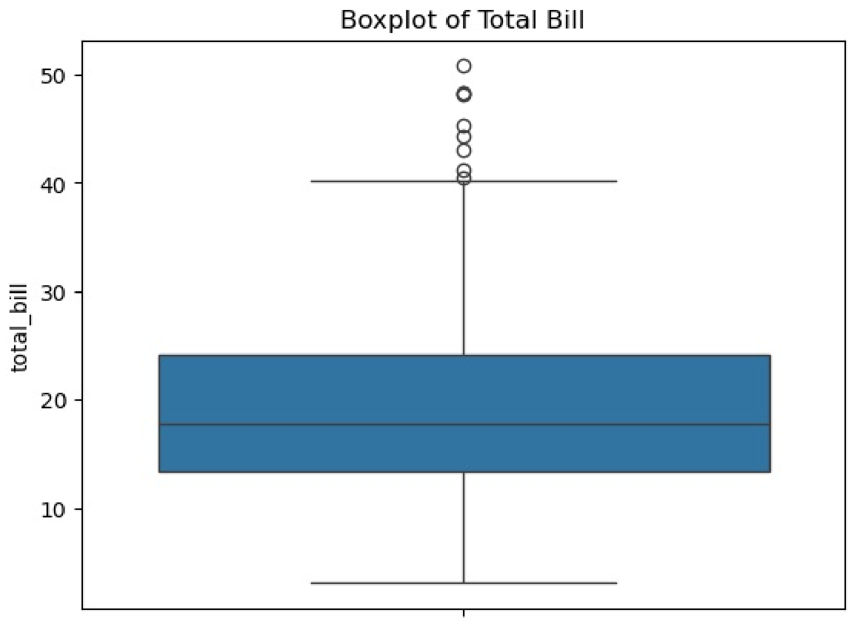


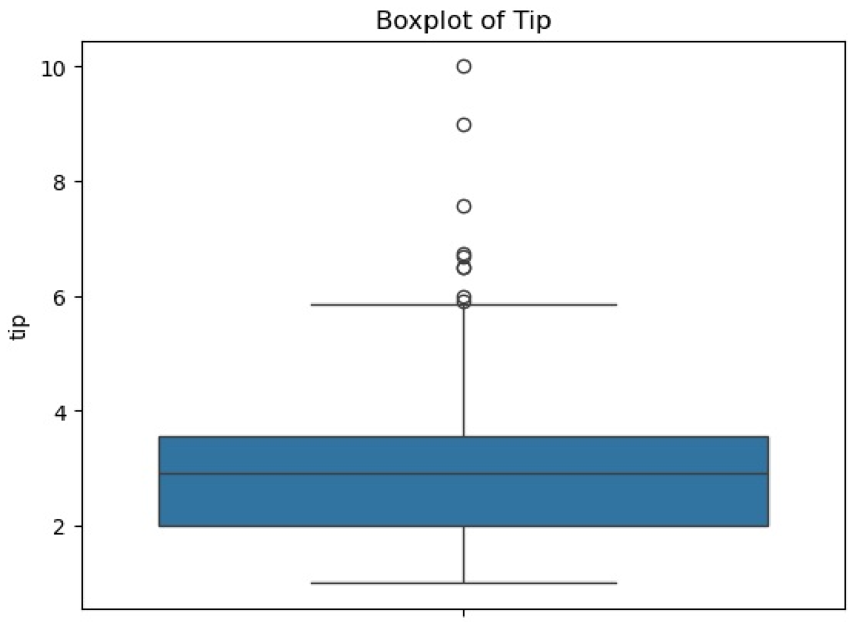
total_bill
tip
size
Pairplot of Numerical Columns

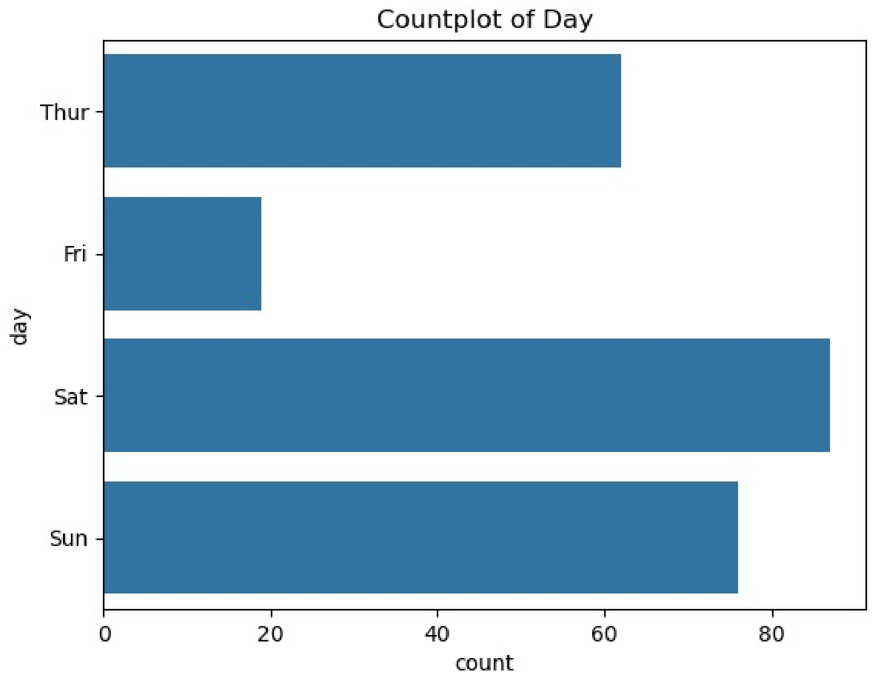

Pairplot with Hue on Time
time
Lunch
Dinner
total_bill
tip
size
total_bill
tip
size

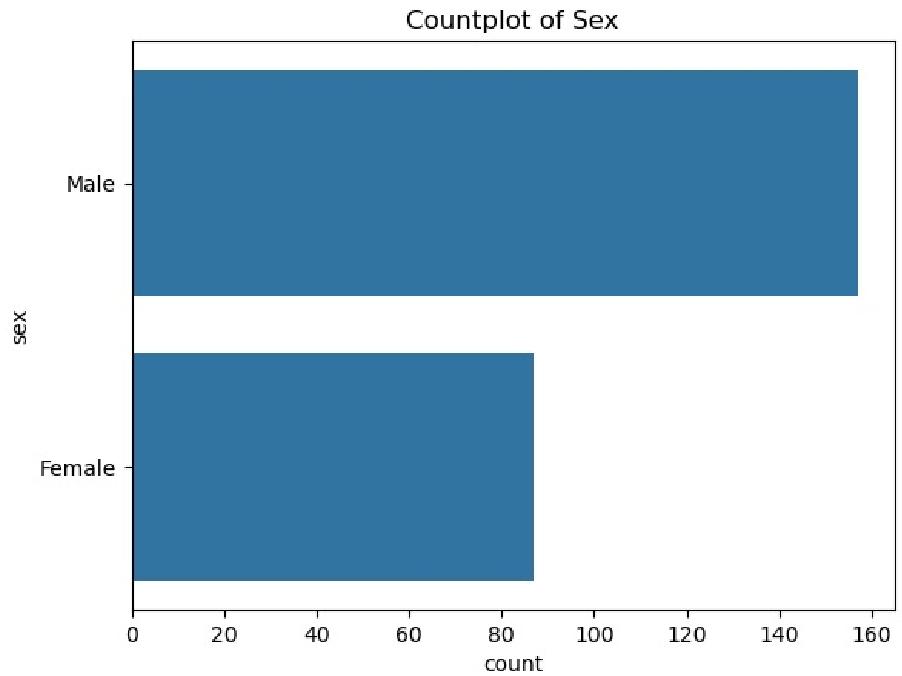

Pairplot with Hue on Day
total_bill
tip
size
day
Thur
Fri
Sat
Sun

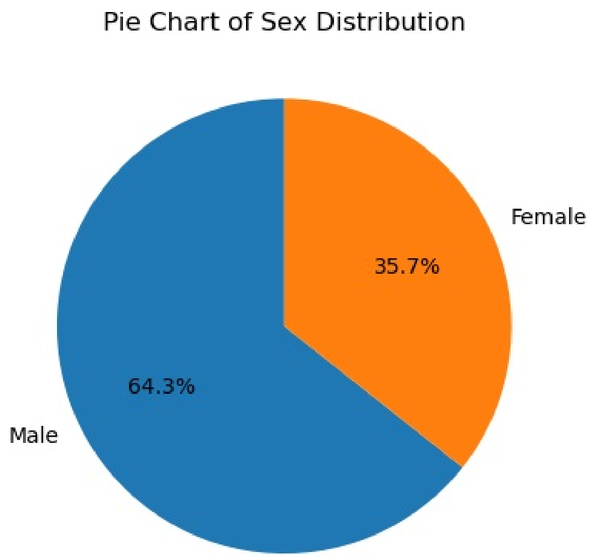

Heatmap of Correlation Matrix
total_bill
tip
size
1
0.68
0.6
0.68
1
0.49
0.6
0.49
1
1.0
0.9
0.8
0.7
0.6
0.5

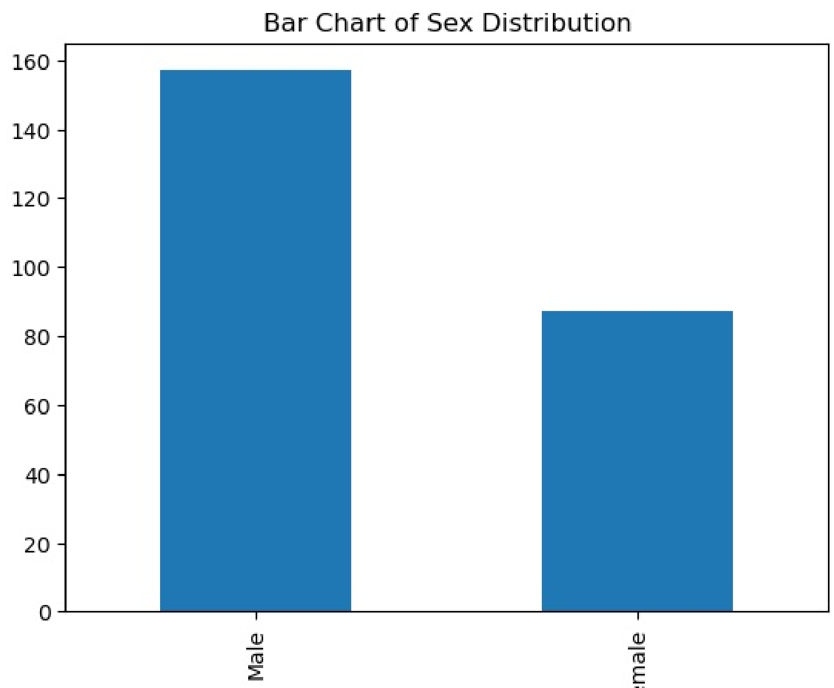

Boxplot of Total Bill
total_bill
50
40
30
20
10

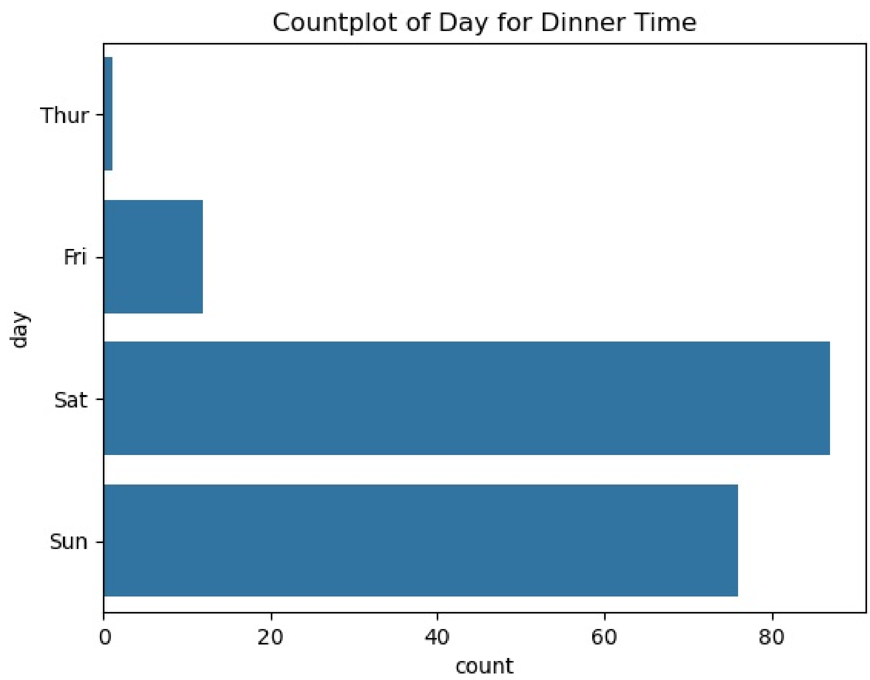

Boxplot of Tip
tip
10
8
6
4
2


Countplot of Day
Thur
Fri
Sat
Sun
day
0
20
40
60
80
count


Countplot of Sex
Male
Female
sex
0
20
40
60
80
100
120
140
160
count

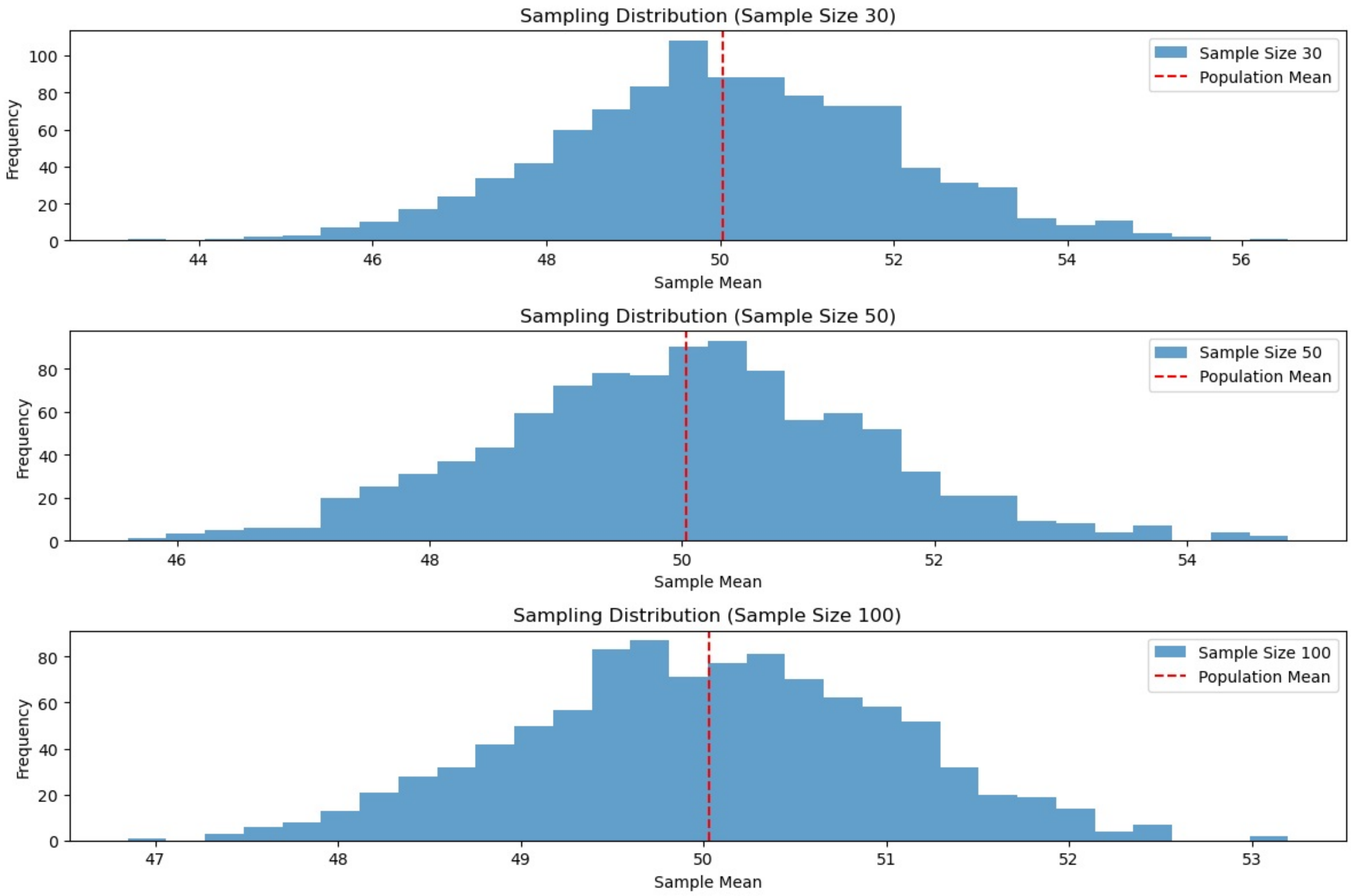

Pie Chart of Sex Distribution
Female
35.7%
64.3%
Male


Bar Chart of Sex Distribution
160
140
120
100
80
60
40
20
0
Male
Female
sex



In [54]:
```
# Name of the Experiment : Random Sampling and Sampling Distribution
# EX NO : 06
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date :10/09/2024
```

In [63]:
```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Generate a population (e.g., normal distribution)
population_mean = 50
population_std = 10
population_size = 100000
population = np.random.normal(population_mean, population_std, population_size)

# Step 2: Random sampling
sample_sizes = [30, 50, 100]  # Different sample sizes to consider
num_samples = 1000  # Number of samples for each sample size
sample_means = {}

# Loop through each sample size
for size in sample_sizes:
    sample_means[size] = []
    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means[size].append(np.mean(sample))

# Step 3: Plotting sampling distributions
plt.figure(figsize=(12, 8))

# Loop through sample sizes and plot each distribution
for i, size in enumerate(sample_sizes):
    plt.subplot(len(sample_sizes), 1, i + 1)
    plt.hist(sample_means[size], bins=30, alpha=0.7, label=f'Sample Size {size}')
    plt.axvline(np.mean(population), color='red', linestyle='dashed', linewidth=1.5,
                label='Population Mean')
    plt.title(f'Sampling Distribution (Sample Size {size})')
    plt.xlabel('Sample Mean')
    plt.ylabel('Frequency')
    plt.legend()

# Adjust layout for better readability and show the plot
plt.tight_layout()
plt.show()
```



In [65]:
```
# Name of the Experiment : Z-Test
# EX NO : 07
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 10/09/2024
```

In [67]:
```
import numpy as np
import scipy.stats as stats

# Define the sample data (hypothetical weights in grams)
sample_data = np.array([152, 148, 151, 149, 147, 153, 150, 148, 152,
                        149, 151, 150, 149, 152, 151, 148, 150, 152, 149,
                        150, 148, 153, 151, 150, 149, 152, 148, 151, 150, 153])

# Population mean under the null hypothesis
population_mean = 150

# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)  # Using sample standard deviation

# Number of observations
n = len(sample_data)

# Calculate the Z-statistic
z_statistic = (sample_mean - population_mean) / (sample_std / np.sqrt(n))

# Calculate the p-value (two-tailed test)
p_value = 2 * (1 - stats.norm.cdf(np.abs(z_statistic)))

# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-Statistic: {z_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average weight is significantly different from 150 grams.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average weight from 150 gra
```

```
Sample Mean: 150.20
Z-Statistic: 0.6406
P-Value: 0.5218
Fail to reject the null hypothesis: There is no significant difference in average weight from 150 grams.
```

In [13]:
```
# Name of the Experiment : T-Test
```

```
# EX NO : 08
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 08/10/2024
```

In [69]:
```
import numpy as np
import scipy.stats as stats

# Set a random seed for reproducibility
np.random.seed(42)

# Generate hypothetical sample data (IQ scores)
sample_size = 25
sample_data = np.random.normal(loc=102, scale=15, size=sample_size)  # Mean IQ of 102, SD of 15

# Population mean under the null hypothesis
population_mean = 100

# Calculate sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)  # Using sample standard deviation

# Number of observations
n = len(sample_data)

# Calculate the T-statistic and p-value using a one-sample t-test
t_statistic, p_value = stats.ttest_1samp(sample_data, population_mean)

# Print results
print(f"Sample Mean: {sample_mean:.2f}")
print(f"T-Statistic: {t_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average IQ score is significantly different from 100.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.
```

```
Sample Mean: 99.55
T-Statistic: -0.1577
P-Value: 0.8760
Fail to reject the null hypothesis: There is no significant difference in average IQ score from 100.
```

In [15]:
```
# Name of the Experiment : Annova TEST
# EX NO : 09
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 08/10/2024
```

In [71]:
```
import numpy as np
import scipy.stats as stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd

# Set a random seed for reproducibility
np.random.seed(42)

# Generate hypothetical growth data for three treatments (A, B, C)
n_plants = 25
# Growth data (in cm) for Treatment A, B, and C
growth_A = np.random.normal(loc=10, scale=2, size=n_plants)
growth_B = np.random.normal(loc=12, scale=3, size=n_plants)
growth_C = np.random.normal(loc=15, scale=2.5, size=n_plants)

# Combine all data into one array
all_data = np.concatenate([growth_A, growth_B, growth_C])

# Treatment labels for each group
treatment_labels = ['A'] * n_plants + ['B'] * n_plants + ['C'] * n_plants

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(growth_A, growth_B, growth_C)

# Print results
print("Treatment A Mean Growth:", np.mean(growth_A))
print("Treatment B Mean Growth:", np.mean(growth_B))
print("Treatment C Mean Growth:", np.mean(growth_C))
print()
print(f"F-Statistic: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Decision based on the significance level
```

```
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in mean growth rates among the three t
else:
    print("Fail to reject the null hypothesis: There is no significant difference in mean growth rates among the

# Additional: Post-hoc analysis (Tukey's HSD) if ANOVA is significant
if p_value < alpha:
    tukey_results = pairwise_tukeyhsd(all_data, treatment_labels, alpha=0.05)
    print("\nTukey's HSD Post-hoc Test:")
    print(tukey_results)
```

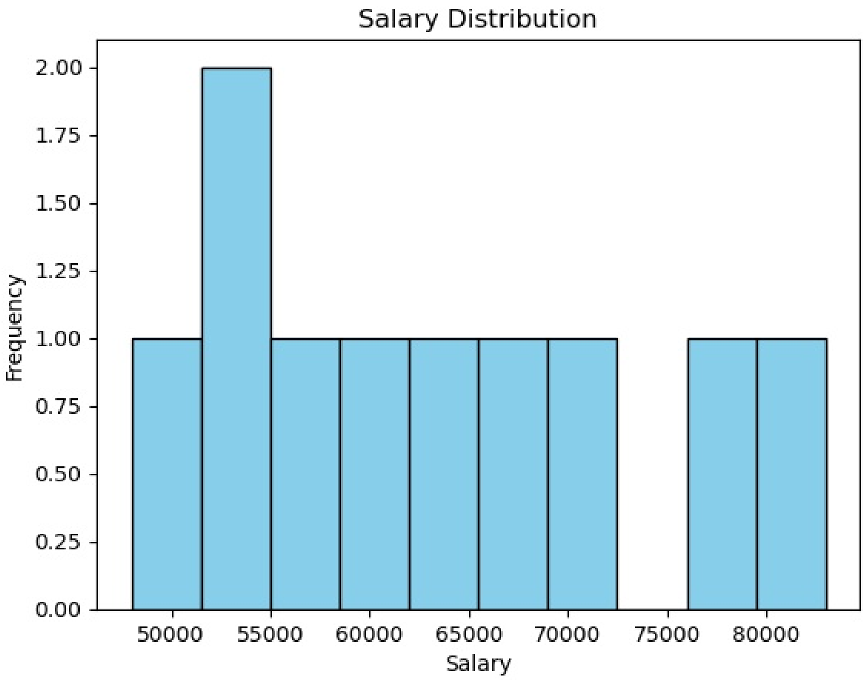
```
Treatment A Mean Growth: 9.672983882683818
Treatment B Mean Growth: 11.137680744437432
Treatment C Mean Growth: 15.265234904828972

F-Statistic: 36.1214
P-Value: 0.0000
Reject the null hypothesis: There is a significant difference in mean growth rates among the three treatments.

Tukey's HSD Post-hoc Test:
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
     A      B   1.4647 0.0877 -0.1683 3.0977  False
     A      C   5.5923    0.0  3.9593 7.2252   True
     B      C   4.1276    0.0  2.4946 5.7605   True
---------------------------------------------------
```

In [17]:
```
# Name of the Experiment : Feature Scaling
# EX NO : 10
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 22/10/2024
```

In [77]:
```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import matplotlib.pyplot as plt

# Sample dataset
data = {
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'France', 'Franc
    'Age': [44, 27, 30, 38, 40, 35, 38, 48, 50, 37],
    'Salary': [72000, 48000, 54000, 61000, 65000, 58000, 52000, 79000, 83000, 67000],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
}

# Create DataFrame
df = pd.DataFrame(data)

# Display the first few rows of the dataset
print("Original Data:")
print(df)

# Handle missing values
# Fill missing 'Country' with the mode (most frequent value)
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])

# Separate features and labels
features = df.iloc[:, :-1].values
labels = df.iloc[:, -1].values

# Use SimpleImputer to handle missing values for 'Age' and 'Salary'
age_imputer = SimpleImputer(strategy="mean")
salary_imputer = SimpleImputer(strategy="mean")

# Impute missing values
features[:, 1] = age_imputer.fit_transform(features[:, [1]]).flatten()
features[:, 2] = salary_imputer.fit_transform(features[:, [2]]).flatten()

# OneHotEncoder for 'Country' column
oh = OneHotEncoder(sparse_output=False)
country_encoded = oh.fit_transform(features[:, [0]])

# Combine the encoded 'Country' values with the rest of the features
final_features = np.concatenate((country_encoded, features[:, 1:]), axis=1)

# Standardize the features using StandardScaler
scaler = StandardScaler()
standardized_features = scaler.fit_transform(final_features)
```

```python
# Normalize the features using MinMaxScaler
mms = MinMaxScaler(feature_range=(0, 1))
normalized_features = mms.fit_transform(final_features)

# Display the final processed features
print("\nProcessed Features (Standardized):")
print(standardized_features)

print("\nProcessed Features (Normalized):")
print(normalized_features)

# Plotting the processed data (just an example with a histogram for 'Salary')
plt.hist(df['Salary'], bins=10, color='skyblue', edgecolor='black')
plt.title('Salary Distribution')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.show()

# Perform One-Way ANOVA to compare the mean 'Salary' across countries
from scipy import stats

f_stat, p_value = stats.f_oneway(df[df['Country'] == 'France']['Salary'],
                                  df[df['Country'] == 'Spain']['Salary'],
                                  df[df['Country'] == 'Germany']['Salary'])

print("\nANOVA Results:")
print(f"F-Statistic: {f_stat:.4f}, P-Value: {p_value:.4f}")

# Decision based on significance level
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in mean Salary across countries.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in mean Salary across countri

# Perform Tukey's HSD test if ANOVA is significant
if p_value < alpha:
    tukey_results = pairwise_tukeyhsd(df['Salary'], df['Country'], alpha=0.05)
    print("\nTukey's HSD Post-hoc Test Results:")
    print(tukey_results)
```

```
Original Data:
   Country  Age  Salary Purchased
0   France   44   72000        No
1    Spain   27   48000       Yes
2  Germany   30   54000        No
3    Spain   38   61000        No
4  Germany   40   65000       Yes
5   France   35   58000       Yes
6    Spain   38   52000        No
7   France   48   79000       Yes
8   France   50   83000        No
9   France   37   67000       Yes

Processed Features (Standardized):
[[ 1.         -0.5        -0.65465367  0.76973439  0.7379204 ]
 [-1.         -0.5         1.52752523 -1.69922498 -1.44851041]
 [-1.          2.         -0.65465367 -1.26352627 -0.90190271]
 [-1.         -0.5         1.52752523 -0.10166303 -0.26419372]
 [-1.          2.         -0.65465367  0.18880278  0.10021141]
 [ 1.         -0.5        -0.65465367 -0.53736175 -0.53749758]
 [-1.         -0.5         1.52752523 -0.10166303 -1.08410528]
 [ 1.         -0.5        -0.65465367  1.35066601  1.37562939]
 [ 1.         -0.5        -0.65465367  1.64113182  1.74003452]
 [ 1.         -0.5        -0.65465367 -0.24689594  0.28241398]]

Processed Features (Normalized):
[[1.         0.         0.         0.73913043 0.68571429]
 [0.         0.         1.         0.         0.        ]
 [0.         1.         0.         0.13043478 0.17142857]
 [0.         0.         1.         0.47826087 0.37142857]
 [0.         1.         0.         0.56521739 0.48571429]
 [1.         0.         0.         0.34782609 0.28571429]
 [0.         0.         1.         0.47826087 0.11428571]
 [1.         0.         0.         0.91304348 0.88571429]
 [1.         0.         0.         1.         1.        ]
 [1.         0.         0.         0.43478261 0.54285714]]
```



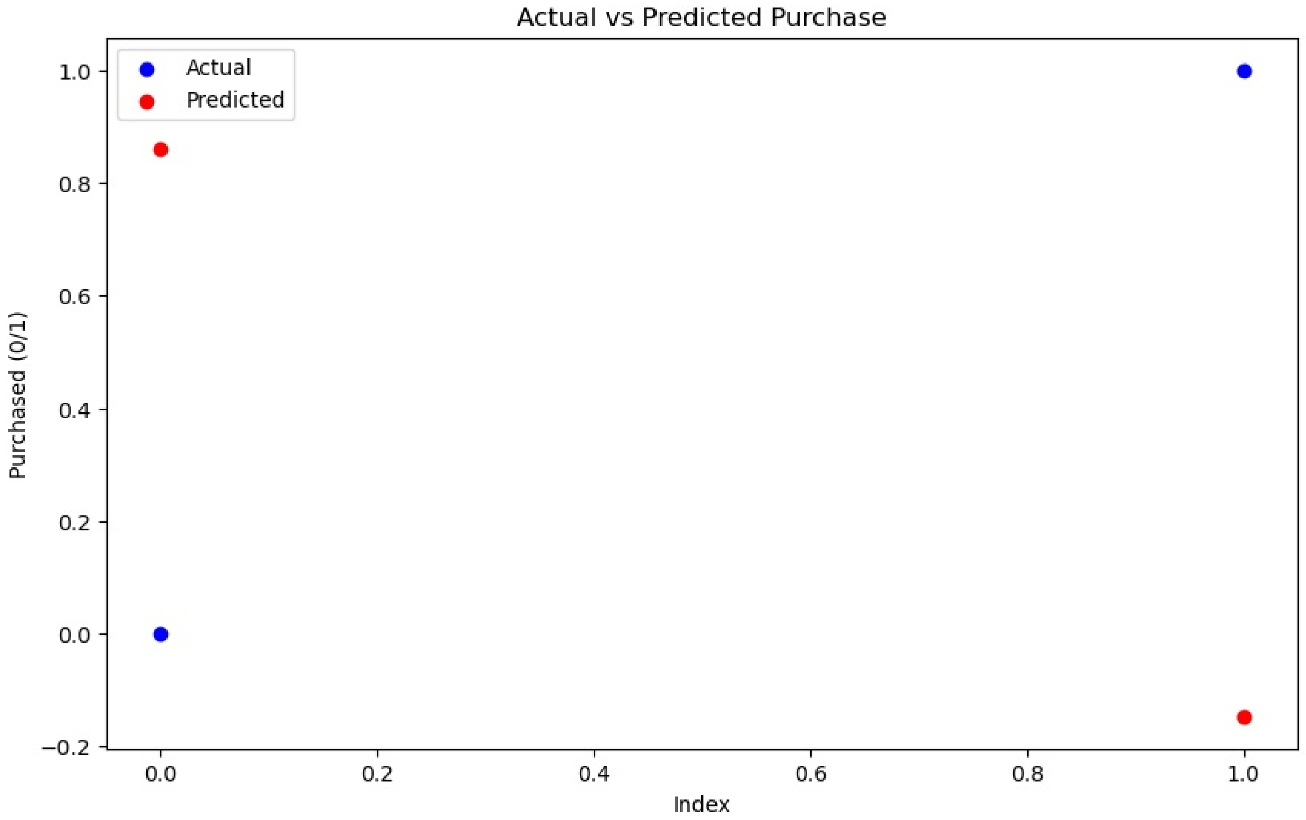

```
ANOVA Results:
F-Statistic: 4.3100, P-Value: 0.0602
Fail to reject the null hypothesis: There is no significant difference in mean Salary across countries.
```

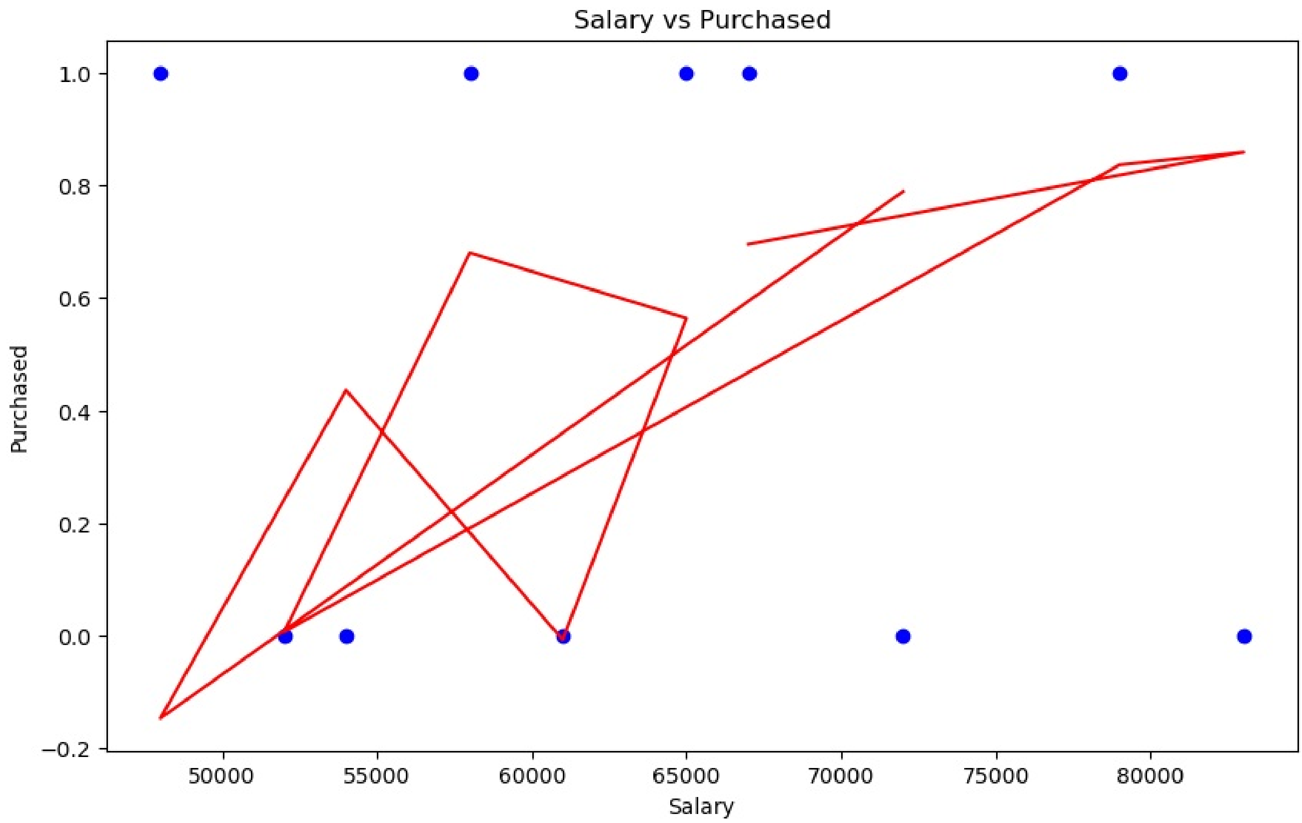
In [19]:
```
# Name of the Experiment : Linear Regression
# EX NO : 11
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 29/10/2024
```

In [81]:
```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# Sample dataset
data = {
    'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'France', 'Franc
    'Age': [44, 27, 30, 38, 40, 35, 38, 48, 50, 37],
    'Salary': [72000, 48000, 54000, 61000, 65000, 58000, 52000, 79000, 83000, 67000],
    'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']
}

# Create DataFrame
df = pd.DataFrame(data)

# Handle missing values (if any)
df['Country'] = df['Country'].fillna(df['Country'].mode()[0])  # Filling missing countries
df['Age'] = df['Age'].fillna(df['Age'].mean())  # Filling missing age
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())  # Filling missing salary

# Encoding 'Country' (categorical to numerical using OneHotEncoding)
from sklearn.preprocessing import OneHotEncoder

# Encode the 'Country' feature
encoder = OneHotEncoder(sparse_output=False)  # Updated parameter
country_encoded = encoder.fit_transform(df[['Country']])

# Combine the encoded 'Country' with 'Age' and 'Salary'
X = np.concatenate((country_encoded, df[['Age', 'Salary']].values), axis=1)

# Convert 'Purchased' to numeric (0 for 'No', 1 for 'Yes')
df['Purchased'] = df['Purchased'].map({'No': 0, 'Yes': 1})

y = df['Purchased'].values
```

```
# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling (standardization)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
rmse = np.sqrt(mse)  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared score

# Output evaluation metrics
print(f"Mean Squared Error: {mse:.4f}")
print(f"Root Mean Squared Error: {rmse:.4f}")
print(f"R-squared: {r2:.4f}")

# Visualize the comparison of predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', label='Actual')
plt.scatter(range(len(y_test)), y_pred, color='red', label='Predicted')
plt.title('Actual vs Predicted Purchase')
plt.xlabel('Index')
plt.ylabel('Purchased (0/1)')
plt.legend()
plt.show()

# Optional: Visualizing the regression line for Salary vs Purchased
plt.figure(figsize=(10, 6))
plt.scatter(df['Salary'], df['Purchased'], color='blue')
plt.plot(df['Salary'], model.predict(scaler.transform(np.concatenate((encoder.transform(df[['Country']]), df[['/
plt.title('Salary vs Purchased')
plt.xlabel('Salary')
plt.ylabel('Purchased')
plt.show()
```

```
Mean Squared Error: 1.0261
Root Mean Squared Error: 1.0130
R-squared: -3.1044
```





In [21]:
```
# Name of the Experiment : Logistic Regression
# EX NO : 12
# Student Register Number : 230701059
# Student Name : M N CHANDNI
# Date : 05/11/2024
```

In [101…
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Corrected dataset with equal-length lists
data = {
    "User ID": [15624510, 15810944, 15668575, 15603246, 15804002, 15683016, 15707098, 15686536, 15621310, 15682
                15746732, 15680352, 15820022, 15636760, 15717341, 15755018, 15691863, 15706071, 15654296, 15755
    "Gender": ['Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female',
               'Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male'],
    "Age": [19, 35, 26, 27, 19, 30, 35, 38, 28, 25, 35, 31, 35, 32, 34, 36, 46, 51, 50, 36],
    "EstimatedSalary": [19000, 20000, 43000, 57000, 76000, 85000, 150000, 60000, 62000, 55000,
                        90000, 50000, 58000, 45000, 80000, 33000, 41000, 23000, 20000, 33000],
    "Purchased": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame to ensure it's correct
print("Dataset:\n", df.head())

# Preprocessing the data
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})  # Encoding 'Gender'

# Features and labels
features = df[['Gender', 'Age', 'EstimatedSalary']].values
labels = df['Purchased'].values

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=0)

# Standardize the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```python
# Train the Logistic Regression model
model = LogisticRegression()
model.fit(x_train, y_train)

# Evaluate the model
train_score = model.score(x_train, y_train)
test_score = model.score(x_test, y_test)

print(f"\nTraining Accuracy: {train_score:.4f}")
print(f"Testing Accuracy: {test_score:.4f}")

# Classification report
y_pred = model.predict(x_test)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Predicting on the entire dataset (for the sake of example)
y_pred_full = model.predict(features)
print("\nFull dataset predictions:\n", y_pred_full)
```

```
Dataset:
     User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0

Training Accuracy: 0.9375
Testing Accuracy: 1.0000

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         1

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4


Full dataset predictions:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```