Handling Missing and Inappropriate Data in a Dataset Aim: Demonstrate an experiment to handle missing data and inappropriate data in a Data set using Python Pandas Library for Data Preprocessing. Dataset Given: Hotel.csv CustomerID Age_Group Rating(1-5) Hotel FoodPreference Bill NoOfPax EstimatedSalary Age_Group 1 20-25 4 Ibis veg 1300 2 40000 20-25 2 30-35 5 LemonTree Non-Veg 2000 3 59000 30-35 3 25-30 6 RedFox Veg 1322 2 30000 25-30 4 20-25 -1 LemonTree Veg 1234 2 120000 20-25 5 35+ 3 Ibis Vegetarian 989 2 45000 35+ 6 35+ 3 Ibys Non-Veg 1909 2 122220 35+ 7 35+ 4 RedFox Vegetarian 1000 -1 21122 35+ 8 20-25 7 LemonTree Veg 2999 -10 345673 20-25 9 25-30 2 Ibis Non-Veg 3456 3 -99999 25-30 9 25-30 2 Ibis Non-Veg 3456 3 -99999 25-30 10 30-35 5 RedFox non-Veg - 6755 4 87777 30-35 About Dataset: No.of Columns =9 (called as series CustomerID, Age_Group, Rating(1-5),Hotel, FoodPreference, Bill, NoOfPax, EstimatedSalary) CutomerID: Numerical Continuous data Age: Categorical Data Rating (1-5): Numerical Discrete Data Hotel: Categorical Data Food: Categorical Data Bill: Numerical Continuous data NoOfPax: Numerical Discrete EstimatedSalary: Numerical Continuous data

In [2]:

```
import pandas as pd
import numpy as np

data = {
    'CustomerID': [1,2,3,4,5,6,7,8,9,9,10],
    'Age_Group': ['20-25','30-35','25-30','20-25','35+','35+','35+','20-25','25-30','25-30','30-35'],
    'Rating(1-5)': [4,5,6,-1,3,3,4,7,2,2,5],
    'Hotel': ['Ibis','LemonTree','RedFox','LemonTree','Ibis','Ibys','RedFox','LemonTree','Ibis','Ibis','RedFox'],
    'FoodPreference': ['veg','Non-Veg','Veg','Veg','Vegetarian','Non-Veg','Vegetarian','Veg','Non-Veg','Non-Veg','non-Veg'],
    'Bill': [1300,2000,1322,1234,989,1909,1000,2999,3456,3456,'-'],
    'NoOfPax': [2,3,2,2,2,2,-1,-10,3,3,4],
    'EstimatedSalary': [40000,59000,30000,120000,45000,122220,21122,345673,-99999,-99999,87777]
}

df = pd.DataFrame(data)
df['Bill'] = df['Bill'].astype(str).replace('-', np.nan)
df['Bill'] = df['Bill'].astype(float)
df['Bill'] = df['Bill'].fillna(df['Bill'].mean())
df.loc[(df['Rating(1-5)'] > 5) | (df['Rating(1-5)'] < 1), 'Rating(1-5)'] = df['Rating(1-5)'].median()
df.loc[df['NoOfPax'] <= 0, 'NoOfPax'] = df['NoOfPax'].median()
df.loc[df['EstimatedSalary'] <= 0, 'EstimatedSalary'] = df['EstimatedSalary'].median()
df['Hotel'] = df['Hotel'].replace({'Ibys': 'Ibis'})
df['FoodPreference'] = df['FoodPreference'].replace({'Vegetarian': 'Veg', 'non-Veg': 'Non-Veg', 'veg': 'Veg'})

df.drop_duplicates(inplace=True)
```

```
print(df)
print(df.info())
```

```
    CustomerID Age_Group  Rating(1-5)      Hotel FoodPreference    Bill  \
0            1     20-25            4       Ibis            Veg  1300.0
1            2     30-35            5  LemonTree        Non-Veg  2000.0
2            3     25-30            4     RedFox            Veg  1322.0
3            4     20-25            4  LemonTree            Veg  1234.0
4            5       35+            3       Ibis            Veg   989.0
5            6       35+            3       Ibis        Non-Veg  1909.0
6            7       35+            4     RedFox            Veg  1000.0
7            8     20-25            4  LemonTree            Veg  2999.0
8            9     25-30            2       Ibis        Non-Veg  3456.0
10          10     30-35            5     RedFox        Non-Veg  1966.5

    NoOfPax  EstimatedSalary
0         2            40000
1         3            59000
2         2            30000
3         2           120000
4         2            45000
5         2           122220
6         2            21122
7         2           345673
8         3            45000
10        4            87777
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, 0 to 10
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerID       10 non-null     int64
 1   Age_Group        10 non-null     object
 2   Rating(1-5)      10 non-null     int64
 3   Hotel            10 non-null     object
 4   FoodPreference   10 non-null     object
 5   Bill             10 non-null     float64
 6   NoOfPax          10 non-null     int64
 7   EstimatedSalary  10 non-null     int64
dtypes: float64(1), int64(4), object(3)
memory usage: 720.0+ bytes
None
```

In [ ]:

```
Experiment to understand the data preprocessing in Data science Description:
Understand the importance of Data preprocessing in data science
```

In [3]:

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler
data = {
'CustomerID': [1, 2, 3, 4, 5],
'Age': [22, 25, 47, 52, 46],
'EstimatedSalary': [20000, 35000, 80000, 110000, 150000]

}
df = pd.DataFrame(data)
print("Original Data:\n", df)
scaler_minmax = MinMaxScaler()
df_minmax = pd.DataFrame(scaler_minmax.fit_transform(df[['Age',
'EstimatedSalary']]),
columns=['Age', 'EstimatedSalary'])
print("\nAfter Min-Max Scaling:\n", df_minmax)
scaler_std = StandardScaler()
df_std = pd.DataFrame(scaler_std.fit_transform(df[['Age',
'EstimatedSalary']]),
columns=['Age', 'EstimatedSalary'])
print("\nAfter Standardization:\n", df_std)
```

```
Original Data:
    CustomerID  Age  EstimatedSalary
0            1   22            20000
1            2   25            35000
2            3   47            80000
3            4   52           110000
4            5   46           150000

After Min-Max Scaling:
         Age  EstimatedSalary
0  0.000000         0.000000
1  0.100000         0.115385
2  0.833333         0.461538
3  1.000000         0.692308
4  0.800000         1.000000

After Standardization:
         Age  EstimatedSalary
0 -1.325688        -1.234537
1 -1.083184        -0.920671
2  0.695178         0.020924
3  1.099351         0.648655
4  0.614343         1.485629
```

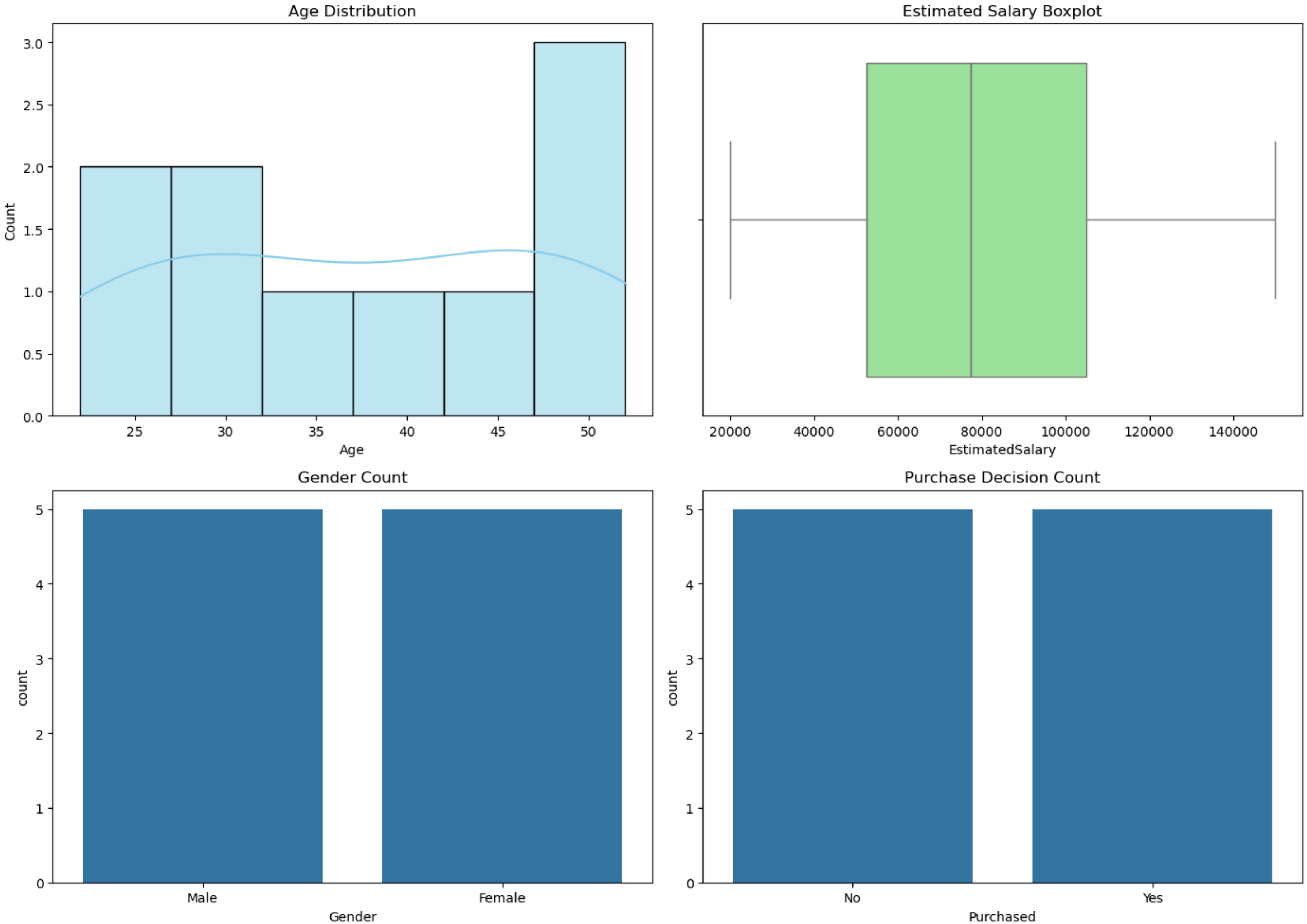
In [ ]:
```
Experiment to understand EDA-Quantitative and Qualitative analysis. Description:
Understand the importance of EDA-Quantitative and Qualitative analysis.
```

In [6]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create sample dataset dictionary
data = {
    'CustomerID': [1,2,3,4,5,6,7,8,9,10],
    'Age': [22,25,47,52,46,30,28,35,40,50],
    'Gender': ['Male','Female','Female','Male','Male','Female','Female','Male','Male','Female'],
    'EstimatedSalary': [20000,35000,80000,110000,150000,60000,75000,90000,120000,50000],
    'Purchased': ['No','Yes','Yes','Yes','No','No','Yes','No','Yes','No']
}
```

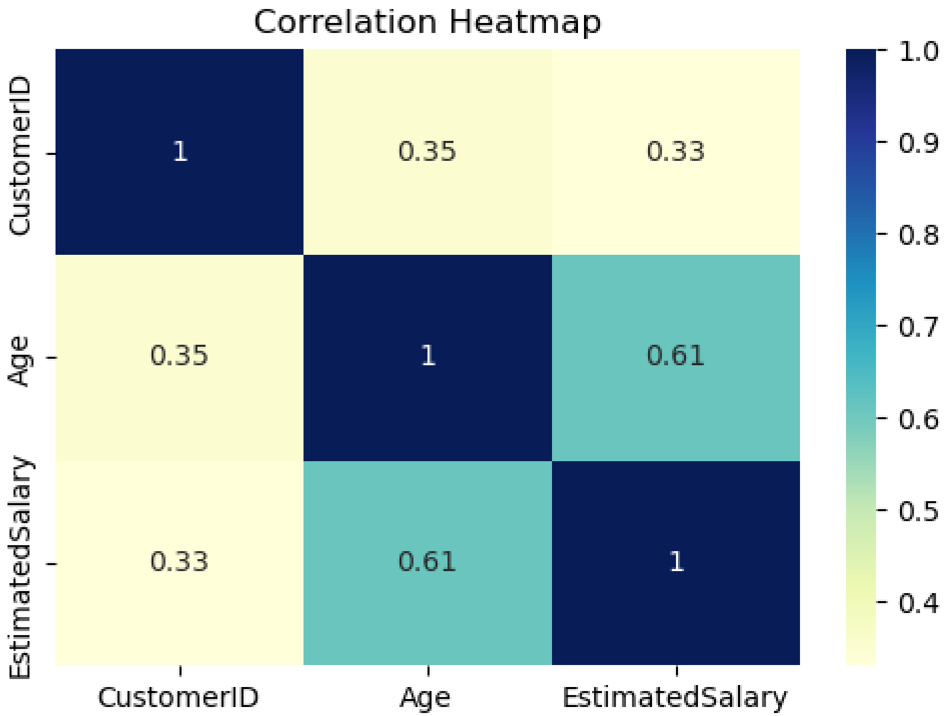
```python
# Convert dictionary to pandas DataFrame
df = pd.DataFrame(data)

# Display summary statistics for numerical columns
print("\nSummary Statistics:\n", df.describe())

# Display correlation matrix for numerical features
print("\nCorrelation Matrix:\n", df.corr(numeric_only=True))

# Show the distribution counts for Gender column
print("\nGender Distribution:\n", df['Gender'].value_counts())

# Show the distribution counts for Purchased column
print("\nPurchase Distribution:\n", df['Purchased'].value_counts())

# Create a figure with 4 subplots for visualization
plt.figure(figsize=(14,10))

# Plot histogram with KDE for Age distribution
plt.subplot(2,2,1)
sns.histplot(df['Age'], bins=6, kde=True, color='skyblue')
plt.title('Age Distribution')

# Plot boxplot for Estimated Salary distribution
plt.subplot(2,2,2)
sns.boxplot(x=df['EstimatedSalary'], color='lightgreen')
plt.title('Estimated Salary Boxplot')

# Plot countplot for Gender without palette to avoid warnings
plt.subplot(2,2,3)
sns.countplot(x='Gender', data=df)
plt.title('Gender Count')

# Plot countplot for Purchased without palette to avoid warnings
plt.subplot(2,2,4)
sns.countplot(x='Purchased', data=df)
plt.title('Purchase Decision Count')

# Adjust subplot spacing
plt.tight_layout()
plt.show()
```

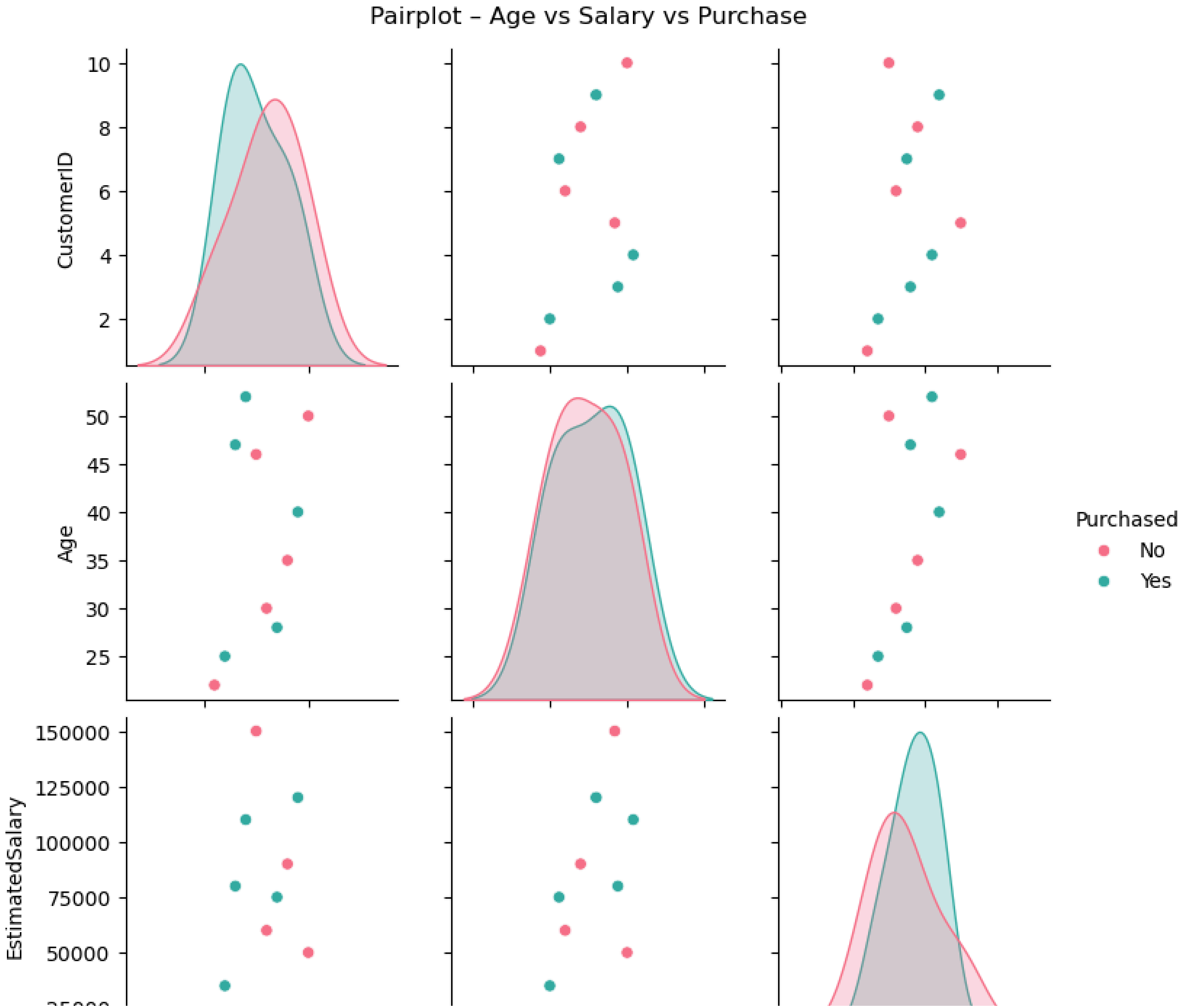
```python
# Plot correlation heatmap of numerical features
plt.figure(figsize=(6,4))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='YlGnBu')
plt.title('Correlation Heatmap')
plt.show()

# Plot pairplot showing pairwise relationships colored by 'Purchased' status
sns.pairplot(df, hue='Purchased', diag_kind='kde', palette='husl')
plt.suptitle('Pairplot - Age vs Salary vs Purchase', y=1.02)
plt.show()
```

```
Summary Statistics:
        CustomerID        Age  EstimatedSalary
count    10.00000  10.000000        10.000000
mean      5.50000  37.500000     79000.000000
std       3.02765  10.977249     40055.517029
min       1.00000  22.000000     20000.000000
25%       3.25000  28.500000     52500.000000
50%       5.50000  37.500000     77500.000000
75%       7.75000  46.750000    105000.000000
max      10.00000  52.000000    150000.000000

Correlation Matrix:
                 CustomerID       Age  EstimatedSalary
CustomerID         1.000000  0.349361         0.329831
Age                0.349361  1.000000         0.611529
EstimatedSalary    0.329831  0.611529         1.000000

Gender Distribution:
 Gender
Male      5
Female    5
Name: count, dtype: int64

Purchase Distribution:
 Purchased
No     5
Yes    5
Name: count, dtype: int64
```

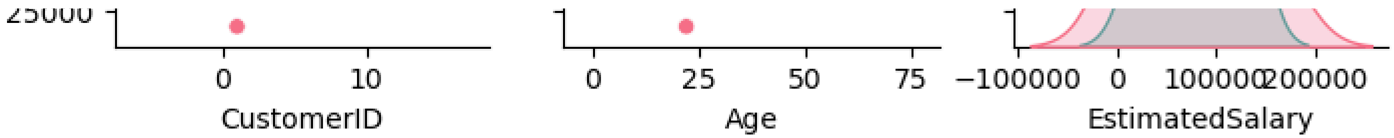


Age Distribution
Count
Age
Estimated Salary Boxplot
EstimatedSalary
Gender Count
count
Male
Female
Gender
Purchase Decision Count
count
No
Yes
Purchased


Correlation Heatmap
CustomerID
Age
EstimatedSalary
1
0.35
0.33
0.35
1
0.61
0.33
0.61
1
CustomerID
Age
EstimatedSalary
1.0
0.9
0.8
0.7
0.6
0.5
0.4



Pairplot – Age vs Salary vs Purchase
CustomerID
Age
EstimatedSalary
10
8
6
4
2
50
45
40
35
30
25
150000
125000
100000
75000
50000
Purchased
No
Yes



In [ ]:

```
Experiment to understand Linear Regression for a given data set. Description:
Understand the Linear regression for the dataset given.
```

In [8]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
data = {
    'Experience': [1,2,3,4,5,6,7,8,9,10],
    'Salary': [25000,28000,35000,40000,45000,52000,60000,67000,75000,82000]
}
df = pd.DataFrame(data)
print("Original Data:\n", df)
X = df[['Experience']]
y = df['Salary']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nPredicted vs Actual:")
result = pd.DataFrame({
    'Experience': X_test.values.flatten(),
    'Actual Salary': y_test,
    'Predicted Salary': y_pred
})
print(result)

print("\nR2 Score:", r2_score(y_test, y_pred))
```

```python
plt.scatter(X, y, color='blue')
plt.plot(X, model.predict(X), color='red')
plt.title('Linear Regression - Experience vs Salary')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

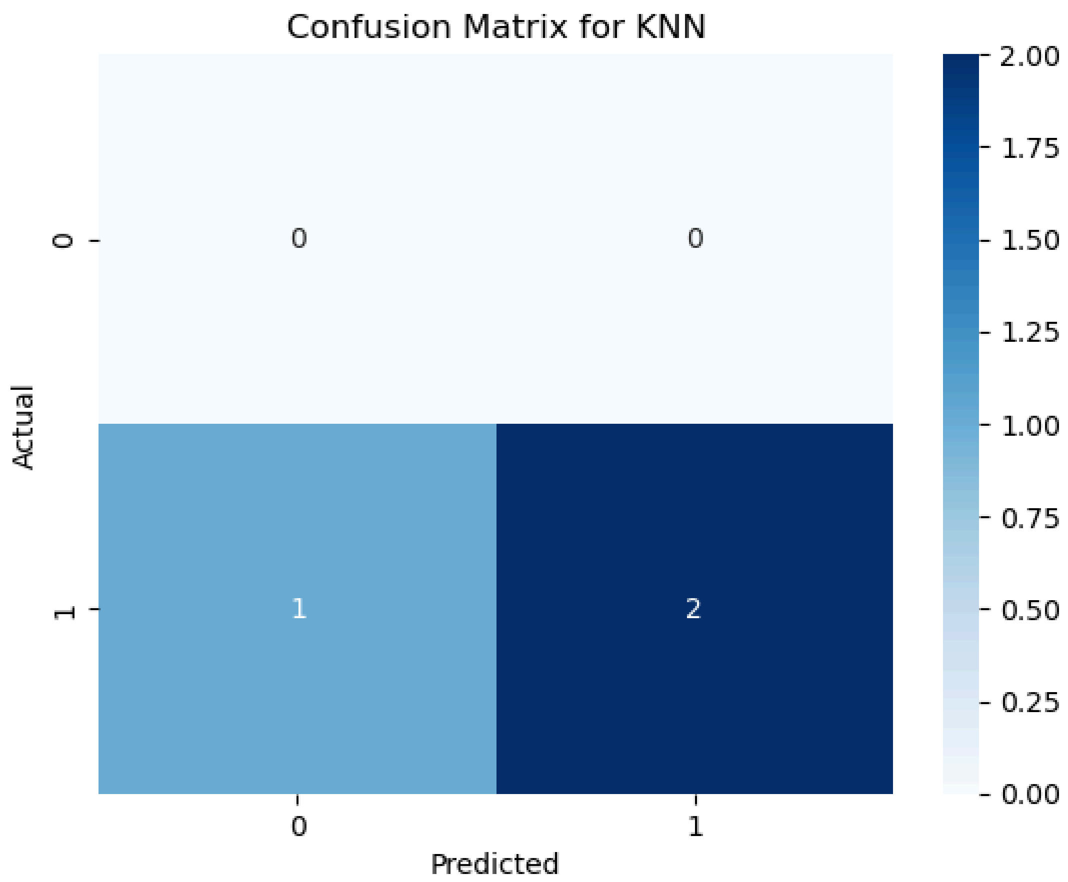
```
Original Data:
    Experience  Salary
0            1   25000
1            2   28000
2            3   35000
3            4   40000
4            5   45000
5            6   52000
6            7   60000
7            8   67000
8            9   75000
9           10   82000

Predicted vs Actual:
   Experience  Actual Salary  Predicted Salary
2           3          35000      34653.620352
8           9          75000      73107.632094

R2 Score: 0.9953737060596429
```

In [ ]:
```
Experiment to understand KNN algorithm for a given data set Description:
Understand the KNN algorithm for the dataset given.
```

In [10]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

data = {
```

```
    'Age': [22, 25, 47, 52, 46, 30, 28, 35, 40, 50],
    'Salary': [20000, 35000, 80000, 110000, 150000, 60000, 75000, 90000, 120000, 50000],
    'Purchased': [0, 0, 1, 1, 1, 0, 1, 0, 1, 0]
}

df = pd.DataFrame(data)
print("Dataset:\n", df)

X = df[['Age', 'Salary']]
y = df['Purchased']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print("\nActual vs Predicted:\n")
result = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(result)

acc = accuracy_score(y_test, y_pred)
print("\nAccuracy:", acc)

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.title('Confusion Matrix for KNN')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Dataset:
    Age  Salary  Purchased
0   22   20000          0
1   25   35000          0
2   47   80000          1
3   52  110000          1
4   46  150000          1
5   30   60000          0
6   28   75000          1
7   35   90000          0
8   40  120000          1
9   50   50000          0

Actual vs Predicted:

   Actual  Predicted
2       1          0
8       1          1
4       1          1

Accuracy: 0.6666666666666666
```

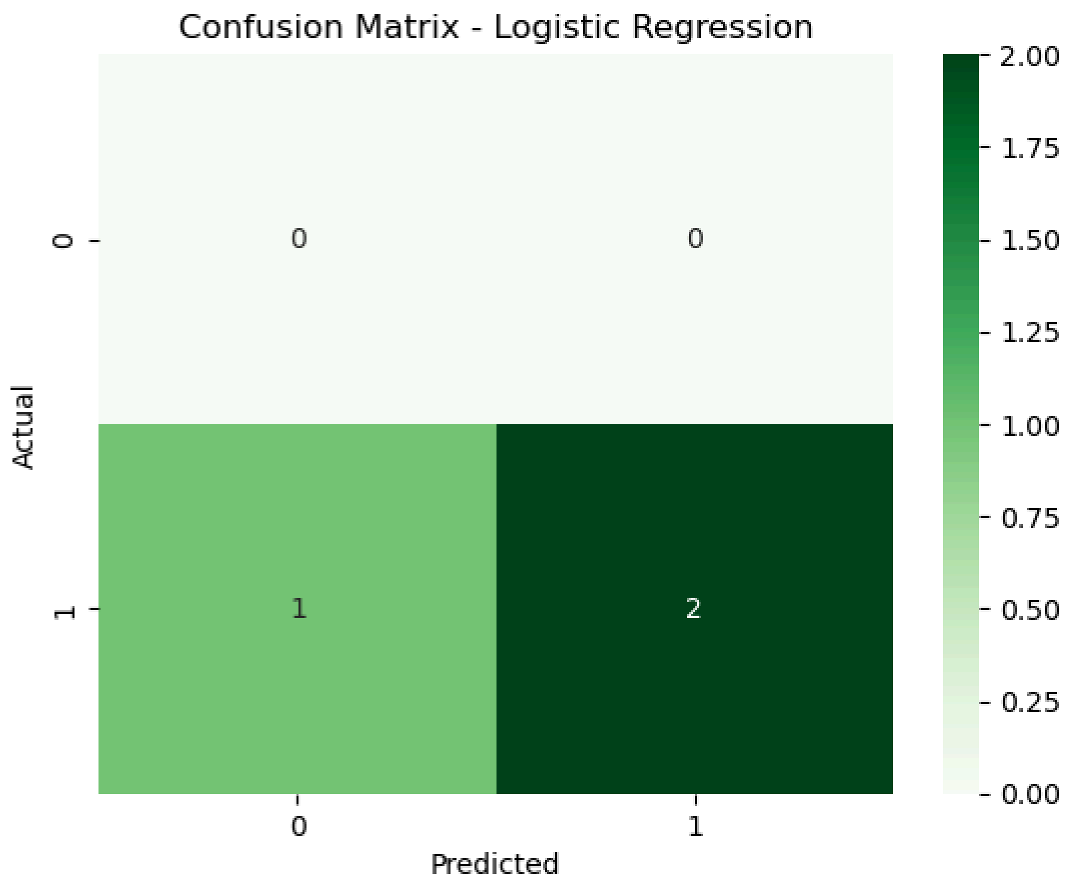


In [ ]:
```
Experiment to understand Logistic Regression for a given data set. Description:
Understand the Logistic Regression algorithm for the dataset given.
```

In [11]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

data = {
```

```python
    'Age': [22,25,47,52,46,30,28,35,40,50],
    'Salary': [20000,35000,80000,110000,150000,60000,75000,90000,120000,50000],
    'Purchased': [0,0,1,1,1,0,1,0,1,0]
}

df = pd.DataFrame(data)
print("Dataset:\n", df)

X = df[['Age', 'Salary']]
y = df['Purchased']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nActual vs Predicted:")
print(pd.DataFrame({'Actual': y_test, 'Predicted': y_pred}))

acc = accuracy_score(y_test, y_pred)
print("\nAccuracy:", acc)

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Greens', fmt='d')
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Dataset:
    Age  Salary  Purchased
0   22   20000          0
1   25   35000          0
2   47   80000          1
3   52  110000          1
4   46  150000          1
5   30   60000          0
6   28   75000          1
7   35   90000          0
8   40  120000          1
9   50   50000          0

Actual vs Predicted:
   Actual  Predicted
2       1          0
8       1          1
4       1          1

Accuracy: 0.6666666666666666
```



In [ ]:

```
No:20 Experiment to understand K-means clustering algorithm for a given data set.
Description: Understand the K-means clustering algorithm for the dataset given.
```

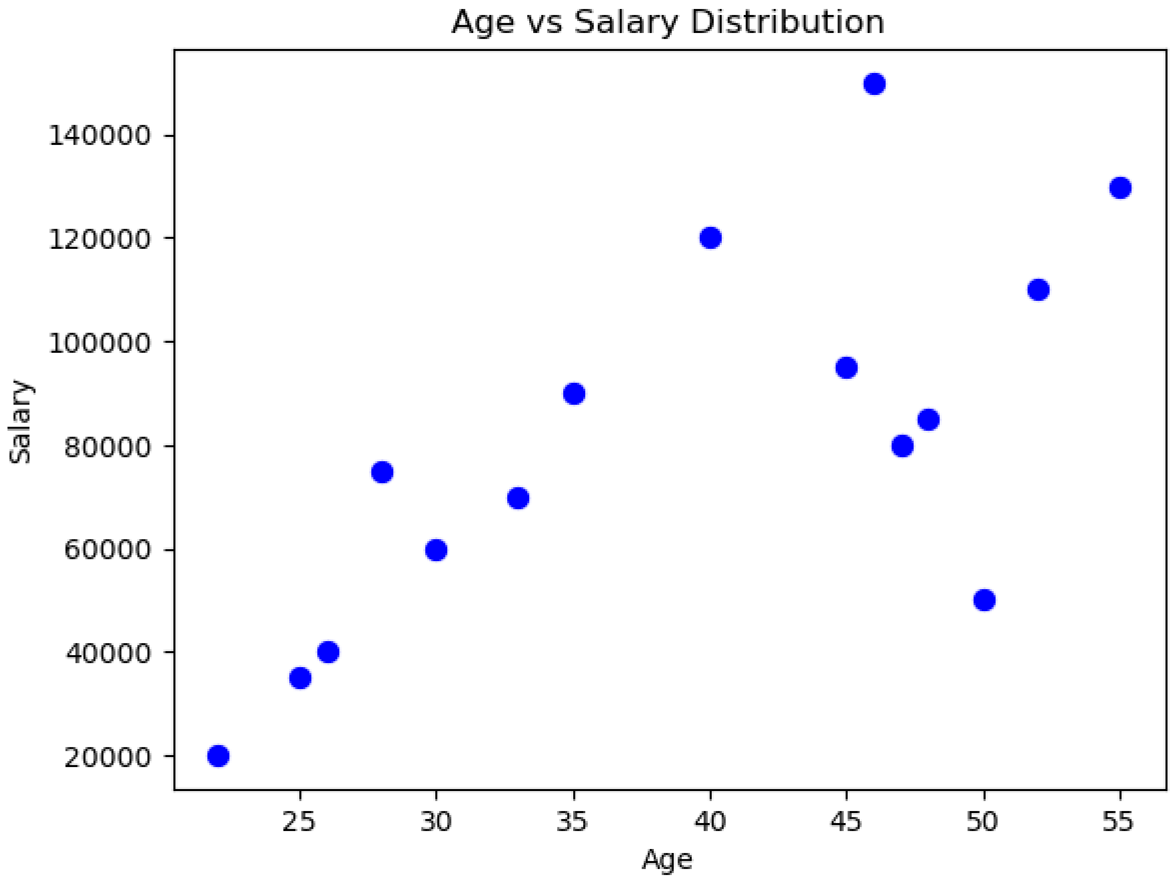
In [14]:

```
import os
os.environ["OMP_NUM_THREADS"] = "1"

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data = {
```

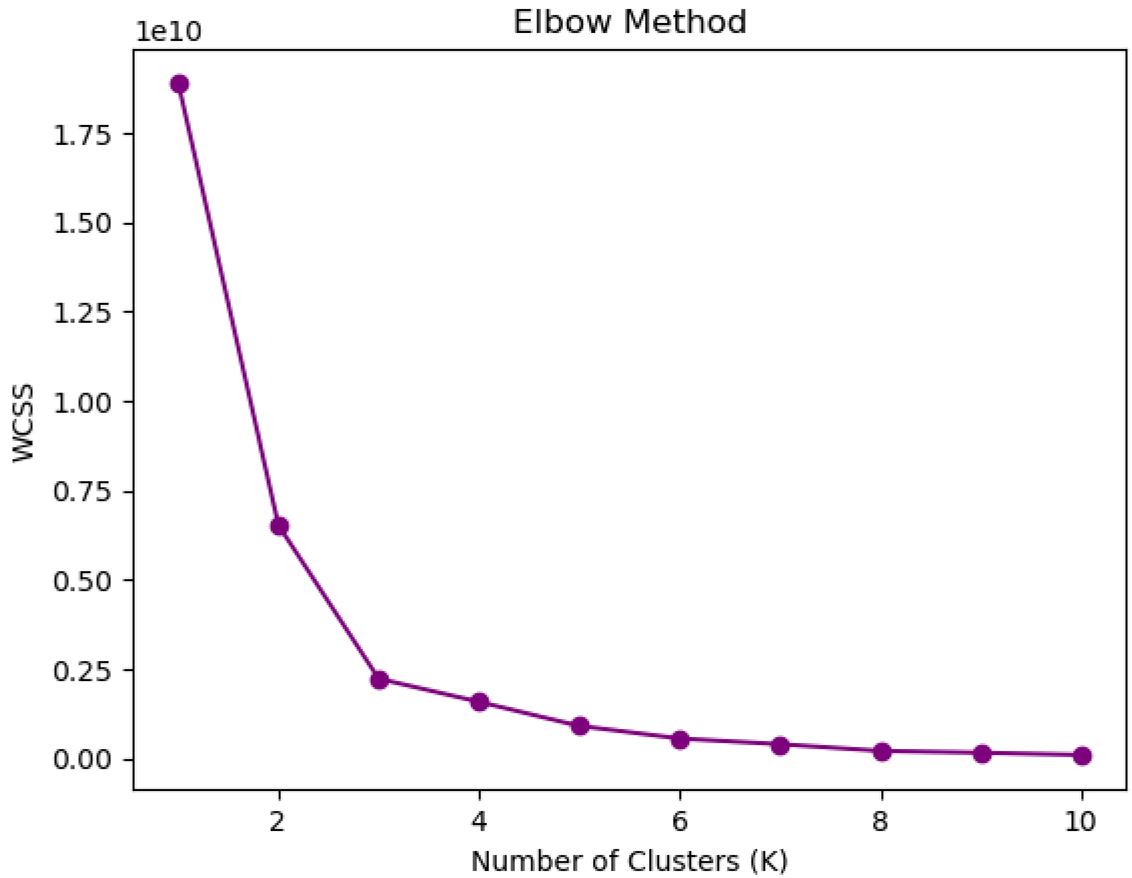
```python
    'Age': [22, 25, 47, 52, 46, 30, 28, 35, 40, 50, 48, 33, 26, 45, 55],
    'Salary': [20000, 35000, 80000, 110000, 150000, 60000, 75000, 90000, 120000,
               50000, 85000, 70000, 40000, 95000, 130000]
}

df = pd.DataFrame(data)
print("Dataset:\n", df)

sns.scatterplot(x='Age', y='Salary', data=df, s=80, color='blue')
plt.title("Age vs Salary Distribution")
plt.show()

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss, marker='o', color='purple')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS")
plt.show()

kmeans = KMeans(n_clusters=3, init='k-means++', random_state=0)
df['Cluster'] = kmeans.fit_predict(df)
print("\nClustered Data:\n", df)

plt.figure(figsize=(8,6))
sns.scatterplot(x='Age', y='Salary', hue='Cluster', data=df, palette='Set2', s=100)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='black', marker='X', label='Centroids')
plt.title("K-Means Clustering")
plt.xlabel("Age")
plt.ylabel("Salary")
plt.legend()
plt.show()

sns.pairplot(df, hue='Cluster', palette='husl', diag_kind='kde')
```

```python
plt.suptitle("Pairplot of Clusters", y=1.02)
plt.show()
```

```
Dataset:
     Age  Salary
0    22   20000
1    25   35000
2    47   80000
3    52  110000
4    46  150000
5    30   60000
6    28   75000
7    35   90000
8    40  120000
9    50   50000
10   48   85000
11   33   70000
12   26   40000
13   45   95000
14   55  130000
```


Age vs Salary Distribution
140000
120000
100000
80000
60000
40000
20000
Salary
25
30
35
40
45
50
55
Age

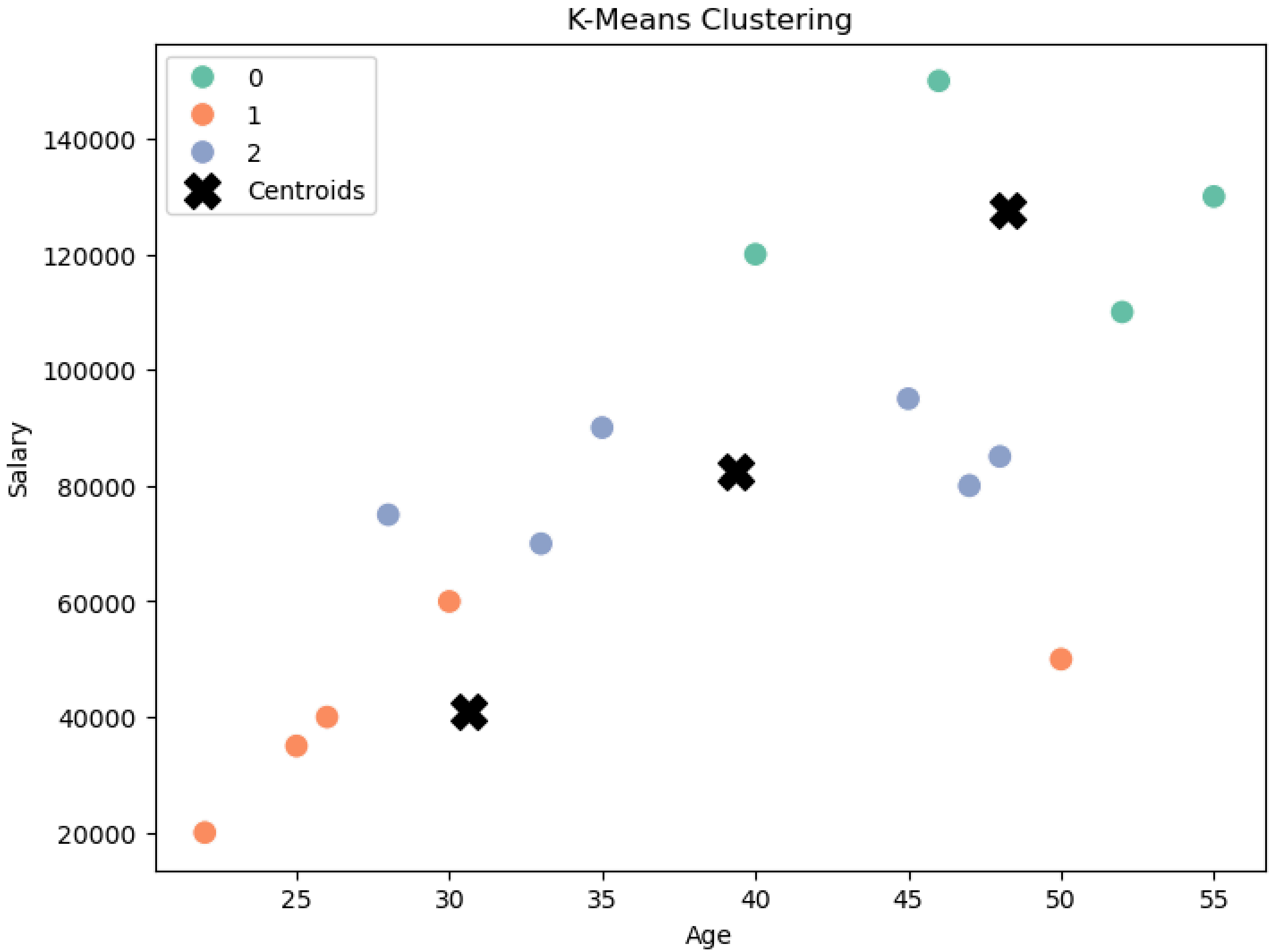
```
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```



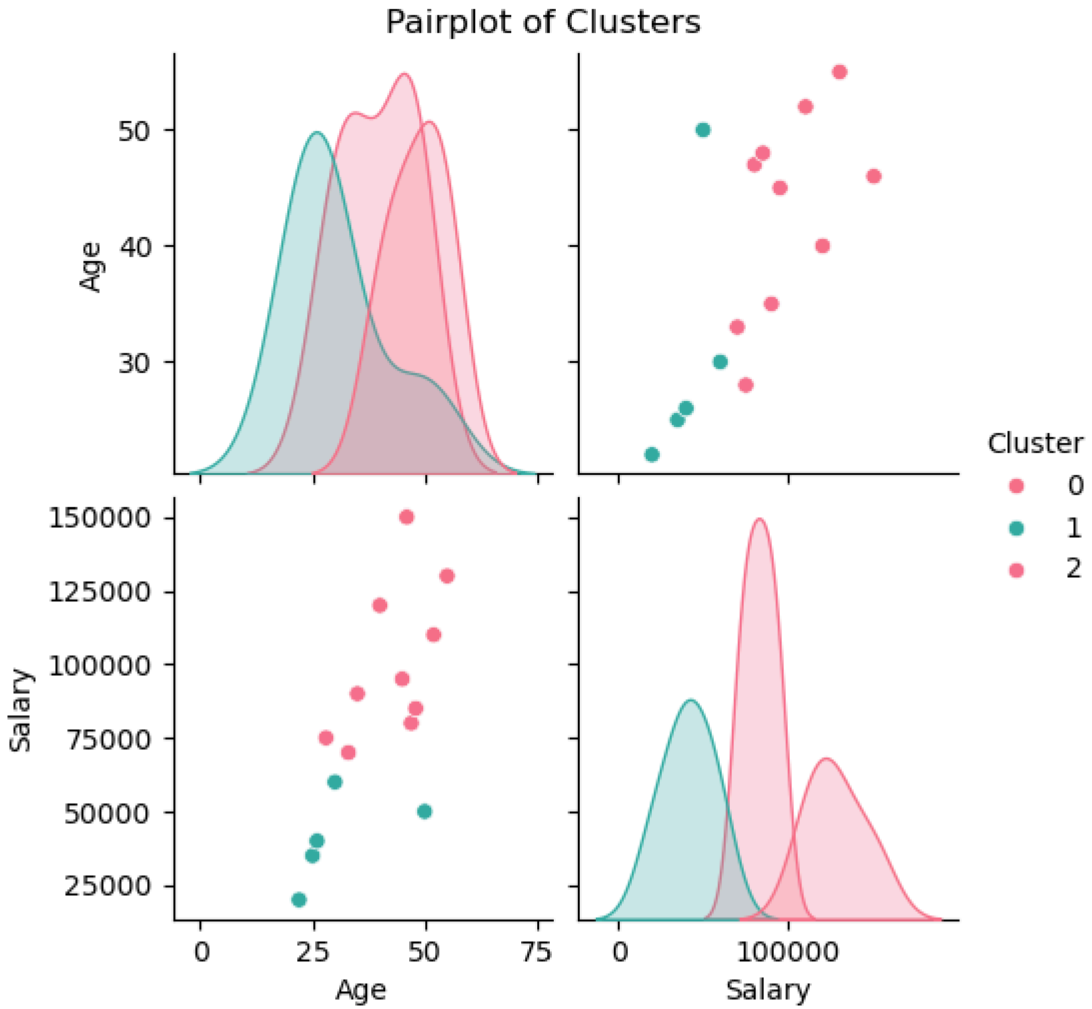

```
C:\Users\gurup\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1419: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```

Clustered Data:

	Age	Salary	Cluster
0	22	20000	1
1	25	35000	1
2	47	80000	2
3	52	110000	0
4	46	150000	0
5	30	60000	1
6	28	75000	2
7	35	90000	2
8	40	120000	0
9	50	50000	1
10	48	85000	2
11	33	70000	2
12	26	40000	1
13	45	95000	2
14	55	130000	0



K-Means Clustering
0
1
2
Centroids
Salary
140000
120000
100000
80000
60000
40000
20000
25
30
35
40
45
50
55
Age


Pairplot of Clusters
Age
50
40
30
Salary
150000
125000
100000
75000
50000
25000
0
25
50
75
Age
0
100000
Salary
Cluster
0
1
2

In [ ]: