# jayipylyn

December 28, 2024

### 0.0.1 What is Exploratory Data Analysis (EDA)?

Exploratory Data Analysis (EDA) is a process of examining and analyzing data to uncover patterns, detect anomalies, test hypotheses, and summarize its main characteristics, often using visual and statistical techniques. It is typically the first step in the data analysis pipeline and helps in understanding the data structure, relationships, and underlying patterns.

Let's wait till 8:35

### 0.0.2 Why is EDA Important?

EDA is critical for several reasons:
1. **Data Understanding**: It provides a deep understanding of the dataset, such as its shape, size, types of variables, and missing values.
2. **Data Cleaning**: Identifies and resolves issues like missing, duplicate, or erroneous data, ensuring the quality of the dataset. 3. **Hypothesis Formation**: Helps formulate questions or hypotheses for further analysis or modeling. 4. **Feature Selection and Engineering**: Reveals which features (columns) are relevant, enabling efficient and effective feature engineering.
5. **Guiding Model Selection**: Gives insights into which types of models may perform best (e.g., regression vs. classification models).
6. **Preventing Biases**: Detects skewness, outliers, or imbalances that could lead to biased or misleading results.

### 0.0.3 Why is EDA Done in Every Data Analysis or Research Project?

EDA is essential in every project because:
1. **Ensures Data Integrity**: Without EDA, undetected issues can lead to incorrect conclusions or poorly performing models.
2. **Foundation for Decision-Making**: The insights gained through EDA drive informed decisions for preprocessing and model building.
3. **Improves Efficiency**: Helps prioritize resources by identifying irrelevant or redundant features early in the pipeline.
4. **Identifies Limitations**: Reveals limitations of the data, such as insufficient samples or biases, guiding more realistic analyses.

### 0.0.4 What is Done in EDA?

1. **Data Overview**: Checking dataset size, column types, and data summary (mean, median, etc.).

2. **Handling Missing Values**: Identifying missing values and deciding on imputation or removal strategies.
3. **Detecting Outliers**: Using methods like box plots or statistical thresholds to detect anomalies.
4. **Univariate Analysis**: Analyzing individual variables using histograms, box plots, and descriptive statistics.
5. **Bivariate and Multivariate Analysis**: Exploring relationships between variables through scatter plots, correlation matrices, or pair plots.
6. **Data Visualization**: Visual tools (e.g., bar charts, line plots, heatmaps) to better understand data distributions and relationships.
7. **Identifying Patterns and Trends**: Observing patterns over time or across categories.
8. **Hypothesis Testing**: Testing initial hypotheses to validate assumptions.

---

# 1 Download/Upload Data

[1]:
```
!git clone https://github.com/ciol-researchlab/CIOL-Winter-ML-Bootcamp.git
```

```
Cloning into 'CIOL-Winter-ML-Bootcamp'…
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 23 (delta 3), reused 23 (delta 3), pack-reused 0 (from 0)
Receiving objects: 100% (23/23), 373.58 KiB | 1.22 MiB/s, done.
Resolving deltas: 100% (3/3), done.
```

# 2 2. Setting up the enviroment

- Pandas

[2]:
```
# Tabular Data Analysis
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Utility
import time
```

```
import warnings
warnings.filterwarnings('ignore')
```

# 3 3. Basic Python

[4]:
```
# Python Data Types

# Integers and Floats
x = 10          # Integer
y = 3.14        # Float

# Strings
name = "Python"  # String

# Booleans
is_valid = True  # Boolean (True or False)

# Lists (ordered, mutable collections)
numbers = [1, 2, 3, 4, "A"]

# Tuples (ordered, immutable collections)
coordinates = (10, 20)

# Sets (unordered, unique elements)
unique_numbers = {1, 2, 3}

# Dictionaries (key-value pairs)
person = {"name": "Alice", "age": 25}

# None (represents "no value")
empty_value = None
```

[5]:
```
# Check Data Types
print(type(unique_numbers))
```

```
<class 'set'>
```

[6]:
```
# Conditional Logic

age = 20

if age < 18:
    print("You are a minor.")
elif 18 <= age <= 60:
    print("You are an adult.")
else:
```

```
    print("You are a senior citizen.")
```

```
You are an adult.
```

[7]:
```
x = 15

if x > 0:
    if x % 2 == 0:
        print("Positive even number")
    else:
        print("Positive odd number")
else:
    print("Not a positive number")
```

```
Positive odd number
```

[9]:
```
# Loop

# Iterating over a list
fruits = ["apple", "banana", "cherry"]
for i in fruits:
    print(i)
```

```
apple
banana
cherry
```

[12]:
```
# Range-based loop
for i in range(2,5,2):  # 0 to 4
    print(i)
```

```
2
4
```

[13]:
```
# Loop until a condition is met
count = 0
while count < 5:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

[14]:
```
# Break example
for i in range(10):
```

```
    if i == 5:
        break
    print(i)  # Stops when i == 5
```

```
0
1
2
3
4
```

[15]:
```
# Continue example
for i in range(10):
    if i % 2 == 0:
        continue  # Skip even numbers
    print(i)  # Only prints odd numbers
```

```
1
3
5
7
9
```

[16]:
```
person = {"name": "Alice", "age": 25, "city": "New York"}

# Accessing values
print(person["name"])  # Output: Alice
```

```
Alice
```

[18]:
```
# Using get() to avoid KeyError
print(person.get("fathers name", "Key not found"))  # Output: 25
```

```
Key not found
```

[19]:
```
person["job"] = "Engineer"  # Add new key-value pair
person["age"] = 30          # Update existing value
print(person)
```

```
{'name': 'Alice', 'age': 30, 'city': 'New York', 'job': 'Engineer'}
```

---

# 4 4. Load the dataset

[20]:
```
df = pd.read_csv("/content/CIOL-Winter-ML-Bootcamp/datasets/session1/main/
 ↪spaceship-titanic/train.csv")
```

[22]:
```
df.head(3)
```

[22]:
```
  PassengerId HomePlanet CryoSleep  Cabin  Destination   Age    VIP  \
0     0001_01     Europa     False  B/0/P  TRAPPIST-1e  39.0  False
1     0002_01      Earth     False  F/0/S  TRAPPIST-1e  24.0  False
2     0003_01     Europa     False  A/0/S  TRAPPIST-1e  58.0   True

   RoomService  FoodCourt  ShoppingMall     Spa  VRDeck             Name  \
0          0.0        0.0           0.0     0.0     0.0  Maham Ofracculy
1        109.0        9.0          25.0   549.0    44.0     Juanna Vines
2         43.0     3576.0           0.0  6715.0    49.0    Altark Susent

   Transported
0        False
1         True
2        False
```

[23]:
```
df["HomePlanet"]
```

[23]:
```
0       Europa
1        Earth
2       Europa
3       Europa
4        Earth
         …
8688    Europa
8689     Earth
8690     Earth
8691    Europa
8692    Europa
Name: HomePlanet, Length: 8693, dtype: object
```

[24]:
```
df[['PassengerId','RoomService','FoodCourt']]
```

[24]:
```
     PassengerId  RoomService  FoodCourt
0        0001_01          0.0        0.0
1        0002_01        109.0        9.0
2        0003_01         43.0     3576.0
3        0003_02          0.0     1283.0
4        0004_01        303.0       70.0
…          …          …        …
8688     9276_01          0.0     6819.0
8689     9278_01          0.0        0.0
8690     9279_01          0.0        0.0
8691     9280_01          0.0     1049.0
8692     9280_02        126.0     4688.0

[8693 rows x 3 columns]
```

# 5 5. Exploratory Data Analysis (EDA)

## 5.1 5.1. Data Overview

[25]:
```
#Check Dataset Size: The number of rows and columns in the dataset.

print(f"Dataset Size: {df.shape}")  # Rows and Columns
```

```
Dataset Size: (8693, 14)
```

[26]:
```
# Check Column Types: Information about columns and their data types.

df.dtypes
```

[26]:
```
PassengerId      object
HomePlanet       object
CryoSleep        object
Cabin            object
Destination      object
Age             float64
VIP              object
RoomService     float64
FoodCourt       float64
ShoppingMall    float64
Spa             float64
VRDeck          float64
Name             object
Transported        bool
dtype: object
```

[27]:
```
# Basic Summary Statistics

df.describe()
```

[27]:
```
               Age   RoomService     FoodCourt  ShoppingMall           Spa  \
count  8514.000000   8512.000000   8510.000000   8485.000000   8510.000000
mean     28.827930    224.687617    458.077203    173.729169    311.138778
std      14.489021    666.717663   1611.489240    604.696458   1136.705535
min       0.000000      0.000000      0.000000      0.000000      0.000000
25%      19.000000      0.000000      0.000000      0.000000      0.000000
50%      27.000000      0.000000      0.000000      0.000000      0.000000
75%      38.000000     47.000000     76.000000     27.000000     59.000000
max      79.000000  14327.000000  29813.000000  23492.000000  22408.000000

             VRDeck
count   8505.000000
mean     304.854791
std     1145.717189
```

```
min        0.000000
25%        0.000000
50%        0.000000
75%       46.000000
max    24133.000000
```

[28]:
```python
df.iloc[:, :-1].describe().T.sort_values(by='std' , ascending = False)\
                     .style.background_gradient(cmap='GnBu')\
                     .bar(subset=["max"], color='#BB0000')\
                     .bar(subset=["mean",], color='green')
```

[28]: <pandas.io.formats.style.Styler at 0x7bc9db771ff0>

[29]:
```python
# Unique Values: Number of unique values for each column.

df.nunique()
```

[29]:
```
PassengerId     8693
HomePlanet         3
CryoSleep          2
Cabin           6560
Destination        3
Age               80
VIP                2
RoomService     1273
FoodCourt       1507
ShoppingMall    1115
Spa             1327
VRDeck          1306
Name            8473
Transported        2
dtype: int64
```

[30]:
```python
# Data Types and Memory Usage: Detailed information about the DataFrame.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   8693 non-null   object
 1   HomePlanet    8492 non-null   object
 2   CryoSleep     8476 non-null   object
 3   Cabin         8494 non-null   object
 4   Destination   8511 non-null   object
```

```
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object
 13  Transported   8693 non-null   bool
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

[31]:
```
# Select numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns.tolist()
print("Numerical Columns:", numerical_columns)
```

```
Numerical Columns: ['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa',
'VRDeck']
```

[32]:
```
# Select categorical columns
categorical_columns = df.select_dtypes(include=['object', 'category']).columns.
 ↪tolist()
print("Categorical Columns:", categorical_columns)
```

```
Categorical Columns: ['PassengerId', 'HomePlanet', 'CryoSleep', 'Cabin',
'Destination', 'VIP', 'Name']
```

[33]:
```
categorical_columns.remove('PassengerId')
```

[36]:
```
# Filtering

df[df['VIP']==True]
```

[36]:
```
     PassengerId HomePlanet CryoSleep    Cabin  Destination   Age   VIP  \
2        0003_01     Europa     False    A/0/S  TRAPPIST-1e  58.0  True
108      0112_01     Europa     False    B/1/S  55 Cancri e  48.0  True
120      0128_01       Mars     False    D/3/S  TRAPPIST-1e  61.0  True
214      0224_01       Mars     False   F/42/S  TRAPPIST-1e  32.0  True
291      0321_01        NaN     False   F/61/S  TRAPPIST-1e  59.0  True
…          …          …       …       …        …   …   …
8579     9158_01     Europa      True  B/298/P  55 Cancri e  30.0  True
8614     9194_02     Europa     False  E/603/S  TRAPPIST-1e  32.0  True
8621     9197_02     Europa     False  C/308/P          NaN  41.0  True
8652     9230_01     Europa     False  C/342/S  TRAPPIST-1e  36.0  True
8688     9276_01     Europa     False   A/98/P  55 Cancri e  41.0  True

      RoomService  FoodCourt  ShoppingMall     Spa  VRDeck               Name  \
2            43.0     3576.0           0.0  6715.0    49.0      Altark Susent
```

```
108          0.0     2537.0          87.0     17.0    13.0        Moth Cowtale
120       2353.0      334.0           9.0    316.0     2.0          Grohs Fles
214        181.0        0.0           5.0   1634.0     0.0        Blues Queen
291       1018.0        0.0         209.0      0.0     0.0        Quites Bache
…            …         …          …      …      …                 …
8579         0.0        0.0           0.0      0.0     0.0     Magnon Maglible
8614      1003.0      909.0           0.0      0.0    15.0       Tachba Subwor
8621         0.0     7964.0           0.0   3238.0  5839.0      Aludram Platch
8652         0.0     5600.0         715.0   2868.0   971.0                 NaN
8688         0.0     6819.0           0.0   1643.0    74.0  Gravior Noxnuther

      Transported
2           False
108          True
120         False
214         False
291         False
…            …
8579         True
8614        False
8621        False
8652         True
8688        False

[199 rows x 14 columns]
```

## 5.2 5.2. Handling Missing Values

[37]:
```
df.isnull()
```

[37]:
```
      PassengerId  HomePlanet  CryoSleep  Cabin  Destination    Age    VIP  \
0           False       False      False  False        False  False  False
1           False       False      False  False        False  False  False
2           False       False      False  False        False  False  False
3           False       False      False  False        False  False  False
4           False       False      False  False        False  False  False
…            …          …         …    …          …    …    …
8688        False       False      False  False        False  False  False
8689        False       False      False  False        False  False  False
8690        False       False      False  False        False  False  False
8691        False       False      False  False        False  False  False
8692        False       False      False  False        False  False  False

      RoomService  FoodCourt  ShoppingMall    Spa  VRDeck   Name  Transported
0           False      False         False  False   False  False        False
1           False      False         False  False   False  False        False
2           False      False         False  False   False  False        False
```

```
3          False     False        False  False   False  False       False
4          False     False        False  False   False  False       False
…          …      …         …   …      …    …           …
8688       False     False        False  False   False  False       False
8689       False     False        False  False   False  False       False
8690       False     False        False  False   False  False       False
8691       False     False        False  False   False  False       False
8692       False     False        False  False   False  False       False

[8693 rows x 14 columns]
```

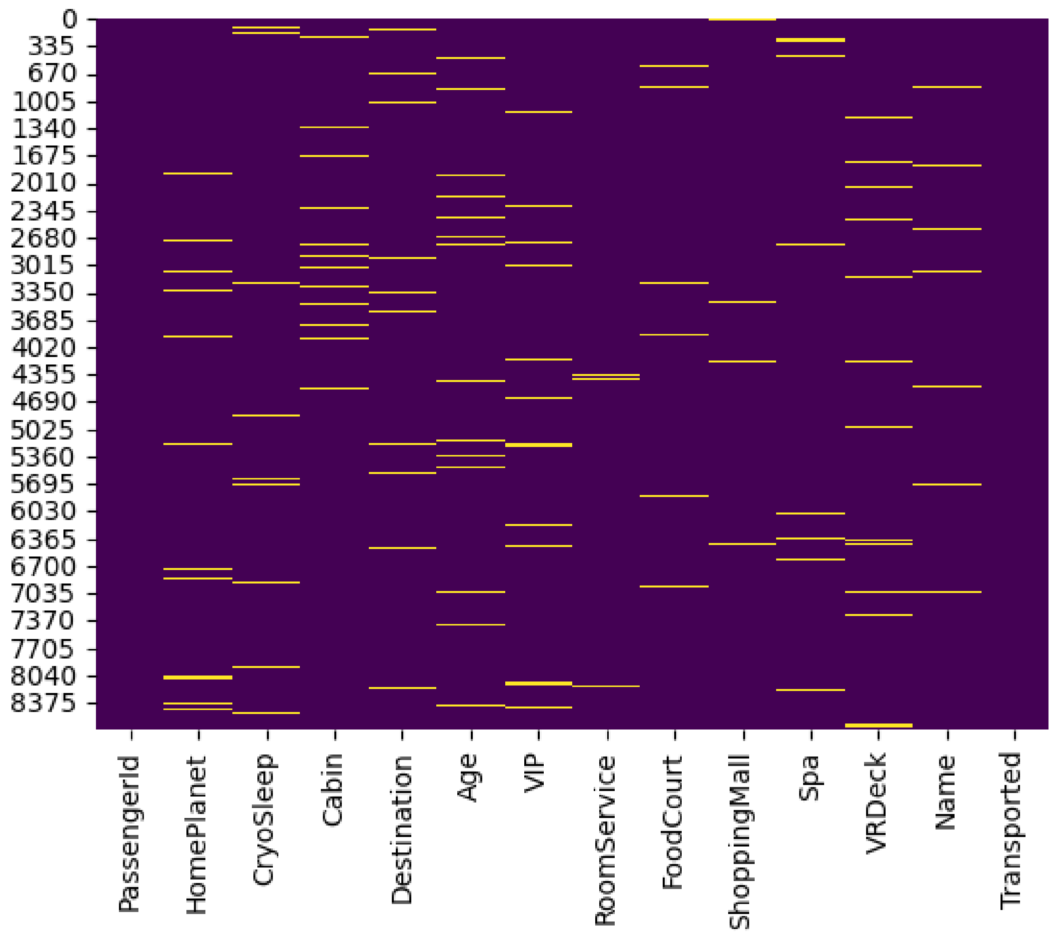
[38]:

```python
# Check for missing values in each column
print(df.isnull().sum())

# Check for missing values as a percentage of the total
print(df.isnull().mean() * 100)
```

```
PassengerId       0
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
Transported       0
dtype: int64
PassengerId     0.000000
HomePlanet      2.312205
CryoSleep       2.496261
Cabin           2.289198
Destination     2.093639
Age             2.059128
VIP             2.335212
RoomService     2.082135
FoodCourt       2.105142
ShoppingMall    2.392730
Spa             2.105142
VRDeck          2.162660
Name            2.300702
Transported     0.000000
dtype: float64
```

[39]:
```
# Visualize missing values using a heatmap (requires seaborn)
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.show()
```




**Insights:** - We can understand that there is no specific missing data pattern. All missing data seems random.

How to handle them?

[40]:
```
# Remove rows with missing values:

df_cleaned = df.dropna()
df_cleaned.shape
```

[40]:
```
(6606, 14)
```

[41]:
```
# Remove columns with missing values:

df_cleaned = df.dropna(axis=1)
df_cleaned.shape
```

[41]: (8693, 2)

[42]:
```
# Dummy df
dfx = pd.DataFrame()
```

[43]:
```
# Fill with a constant value:

dfx['HomePlanet'] = df['HomePlanet'].fillna(0)  # Replace missing with 0
print(df['HomePlanet'].isnull().sum())
print(dfx['HomePlanet'].isnull().sum())
```

```
201
0
```

[45]:
```
df['RoomService'].max()
```

[45]: 14327.0

[ ]:
```
#Fill with the mean, median, or mode:

# Mean
dfx['RoomService'] = df['RoomService'].fillna(df['RoomService'].mean())

# Median
dfx['RoomService'] = df['RoomService'].fillna(df['RoomService'].median())

# Mode (most frequent value)
dfx['RoomService'] = df['RoomService'].fillna(df['HomePlanet'].mode()[0])
```

Use packages scikit-learn

# 6 5.3. Detecting and Fixing Outliers

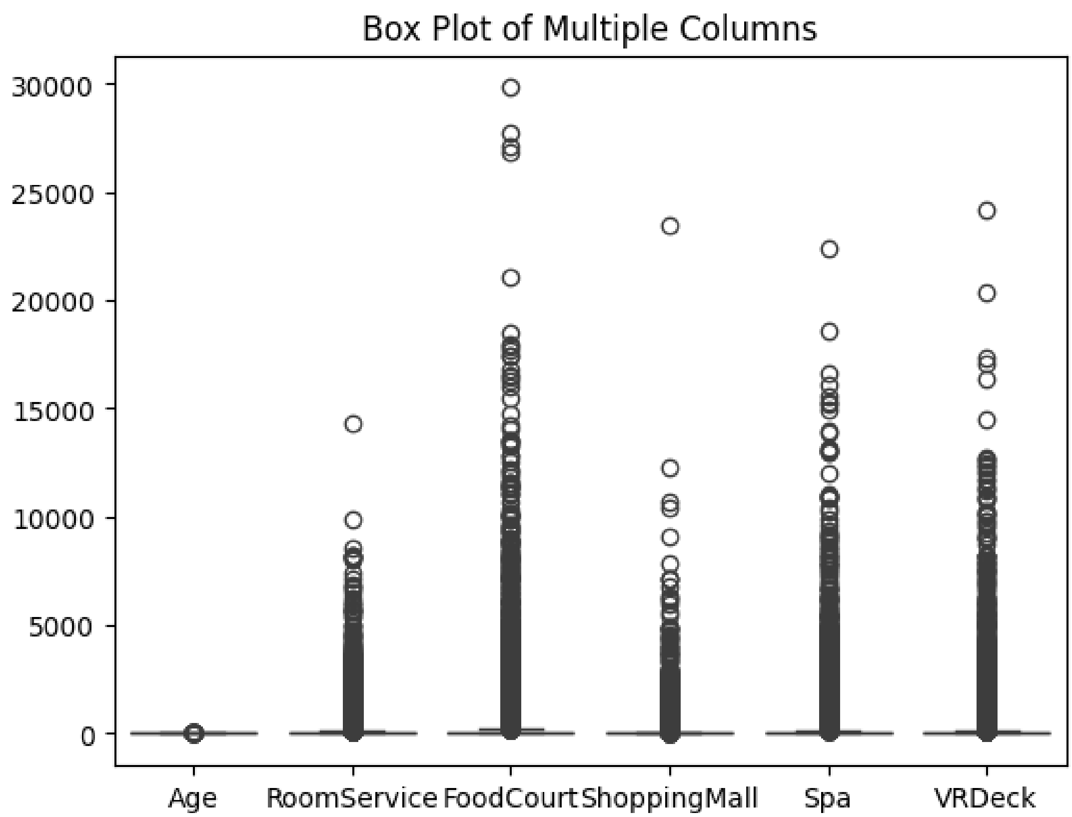
[46]:
```
df['RoomService']
```

[46]:
```
0         0.0
1       109.0
2        43.0
3         0.0
4       303.0
        …
8688      0.0
```

```
8689      0.0
8690      0.0
8691      0.0
8692    126.0
Name: RoomService, Length: 8693, dtype: float64
```

[47]:

```
sns.boxplot(data=df[numerical_columns])
plt.title('Box Plot of Multiple Columns')
plt.show()
```




[48]:

```
# Violin plot for multiple columns
sns.violinplot(data=df[numerical_columns])
plt.title('Violin Plot of Multiple Columns')
plt.show()
```


**Insights:** - There are a good number of outliers in these numeral variables.

#### 6.0.1 Fixing Outliers

[51]:
```
df['RoomService'].quantile(0.90)
```

[51]: 753.0

[52]:
```
# IQR Method (Interquartile Range)

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['RoomService'].quantile(0.25)
Q3 = df['RoomService'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
```

```
outliers = df[(df['RoomService'] < lower_bound) | (df['RoomService'] >
 ↪upper_bound)]
outliers['RoomService']
```

[52]:
```
4        303.0
13       719.0
16      1286.0
20       412.0
27       980.0
         …
8646     676.0
8661     699.0
8675    1030.0
8682     240.0
8692     126.0
Name: RoomService, Length: 1861, dtype: float64
```

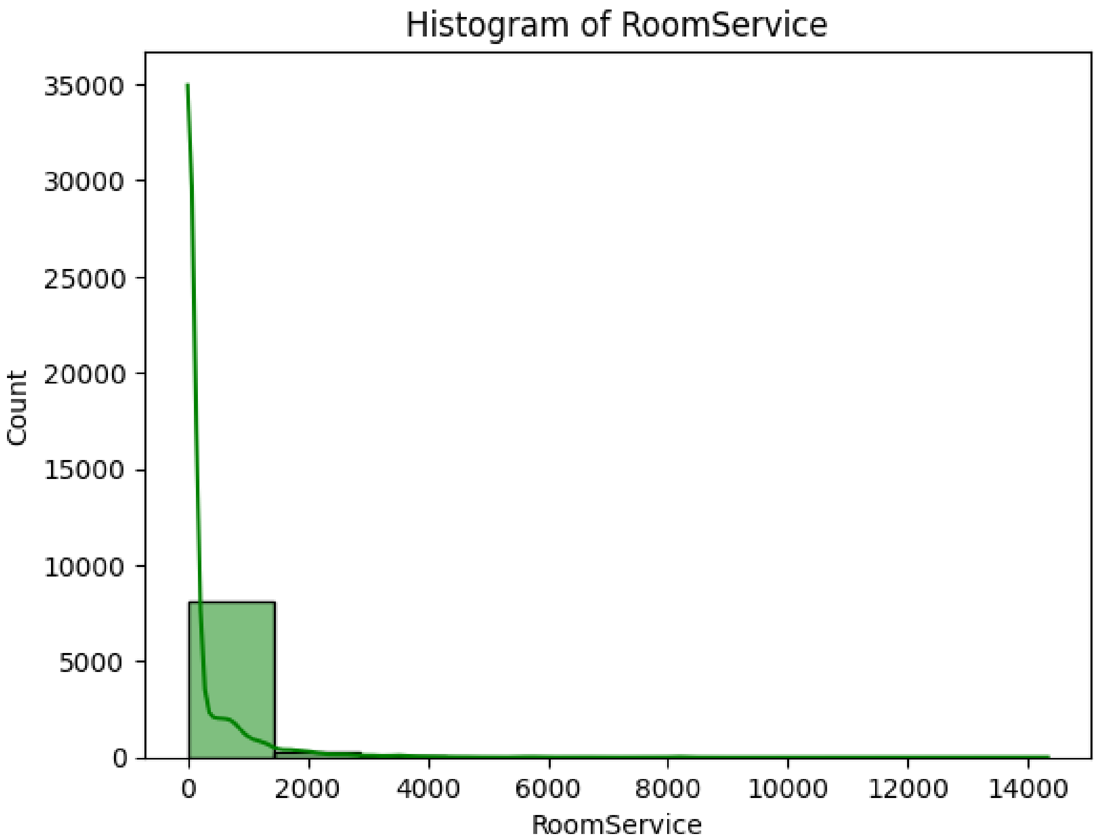
[53]:
```
# Remove rows with outliers
df_cleaned = df[(df['RoomService'] >= lower_bound) & (df['RoomService'] <=
 ↪upper_bound)]
df_cleaned.shape
```

[53]: (6651, 14)

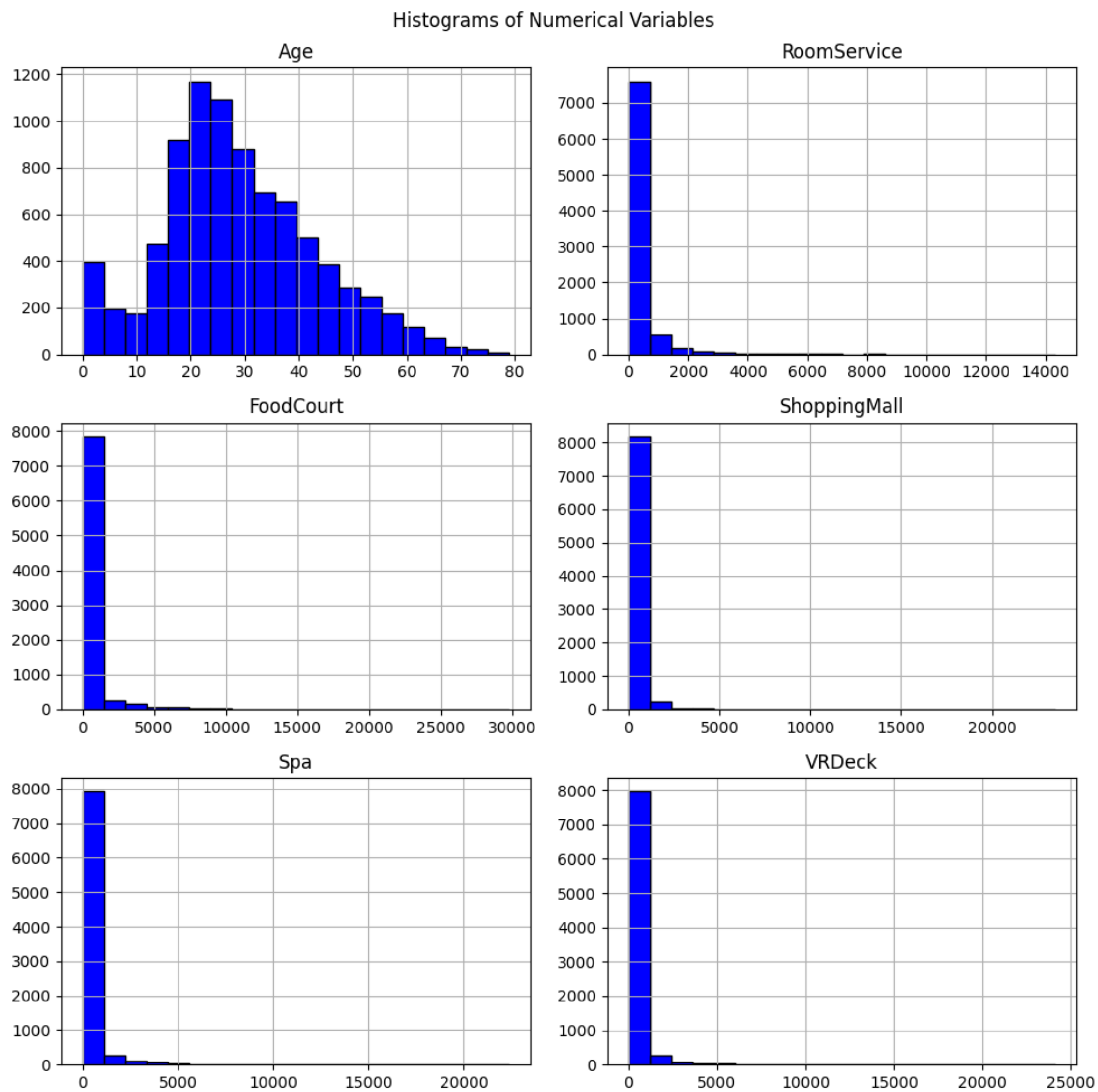
## 6.1 5.4. Univariate Analysis

### 6.1.1 Numerical Variables

[57]:
```
# Plot histogram for individual numerical columns
sns.histplot(df['RoomService'], kde=True, bins=10, color="green")
plt.title(f'Histogram of RoomService')
plt.show()
```




[58]:
```
# Plot multiple histograms together
df[numerical_columns].hist(bins=20, figsize=(10, 10), layout=(3, 2),␣
 ↪color="blue", edgecolor="black")
plt.suptitle('Histograms of Numerical Variables')
plt.tight_layout()
plt.show()
```




[59]:
```
from scipy import stats

def normality_test(df, column, alpha=0.05):

    data = df[column]
    # Perform the Shapiro-Wilk test for normality
    stat, p_value = stats.shapiro(data)

    # Check if p-value is less than alpha (significance level)
    if p_value > alpha:
        return f"Data of {column} column is normally distributed (p-value =␣
↪{p_value:.4f})"
    else:
```

```
        return f"Data of {column} column is NOT normally distributed (p-value =␣
↪{p_value:.4f})"
```

[63]:
```
normality_test(df, numerical_columns[0])
```

[63]:
```
'Data of Age column is NOT normally distributed (p-value = nan)'
```

### 6.1.2 Categorical Variables

[65]:
```
print(df[categorical_columns[2]].value_counts())
```

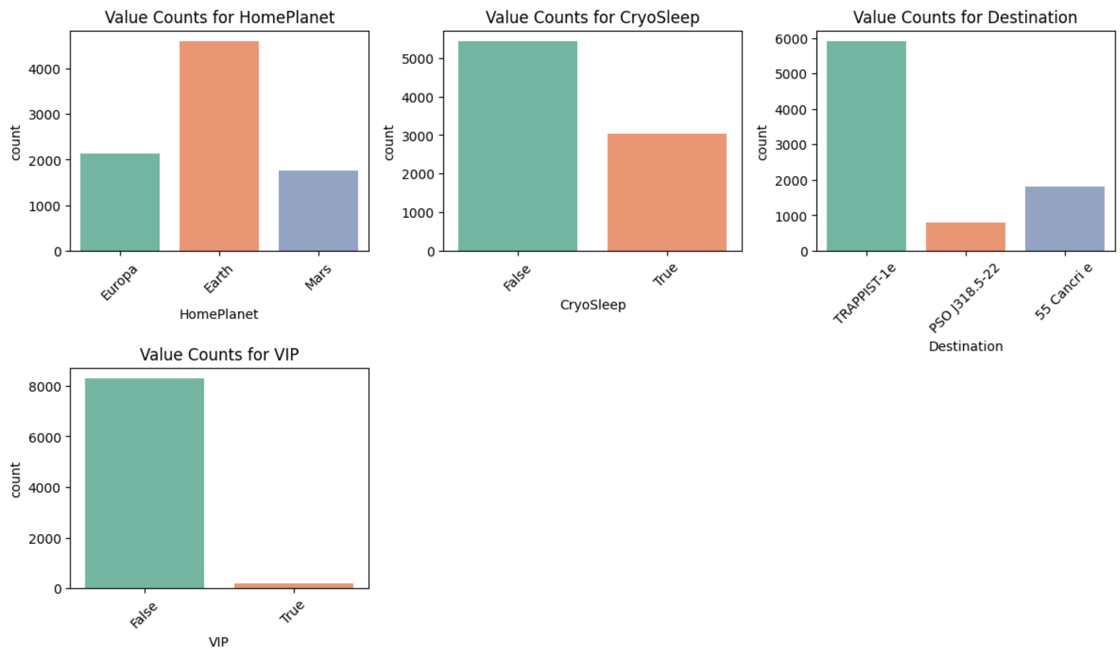
```
Cabin
G/734/S     8
G/109/P     7
B/201/P     7
G/1368/P    7
G/981/S     7
           ..
G/556/P     1
E/231/S     1
G/545/S     1
G/543/S     1
F/947/P     1
Name: count, Length: 6560, dtype: int64
```

[66]:
```
few_category_columns=['HomePlanet','CryoSleep','Destination','VIP']
```

[69]:
```
# Set up the plotting area (adjust size as needed)
plt.figure(figsize=(12, 10))

# Plot each categorical column's value counts
for i, col in enumerate(few_category_columns):
    plt.subplot(3, 3, i+1)  # Adjust grid size (3x3 here)
    sns.countplot(x=col, data=df, palette='Set2')
    plt.title(f'Value Counts for {col}')
    plt.xticks(rotation=45)

# Tight layout for better spacing
plt.tight_layout()
plt.show()
```




## 6.2 5. 5. Bivariate and Multivariate Analysis

### 6.2.1 GroupBy

[71]:
```
# Group by 'HomePlanet' and calculate the mean of 'RoomService'
df.groupby('HomePlanet')['RoomService'].max()
```

[71]:
```
HomePlanet
Earth      6256.0
Europa    14327.0
Mars       9920.0
Name: RoomService, dtype: float64
```

[72]:
```
# Group by 'CryoSleep' and calculate the sum of 'FoodCourt'
df.groupby('CryoSleep')['FoodCourt'].sum()
```

[72]:
```
CryoSleep
False    3799600.0
True           0.0
Name: FoodCourt, dtype: float64
```

[73]:
```
# Group by 'Destination' and 'VIP' and calculate the mean of 'Spa'
df.groupby(['Destination', 'VIP'])['Spa'].mean()
```

[73]:
```
Destination    VIP
55 Cancri e    False    460.360725
```

```
                True      1069.000000
PSO J318.5-22   False       81.632432
                True      1423.722222
TRAPPIST-1e     False      281.909452
                True       465.300885
Name: Spa, dtype: float64
```

[75]:
```
# combined Complex One

# Group by 'HomePlanet', 'CryoSleep', and 'VIP'
df.groupby(['HomePlanet', 'CryoSleep', 'VIP']).agg({
    'RoomService': ['mean', 'sum', 'count'],   # Mean, sum, and count of␣
 ↪'RoomService'
    'FoodCourt': ['mean', 'sum']       # Mean, sum, and count of 'FoodCourt'
}).reset_index()
```

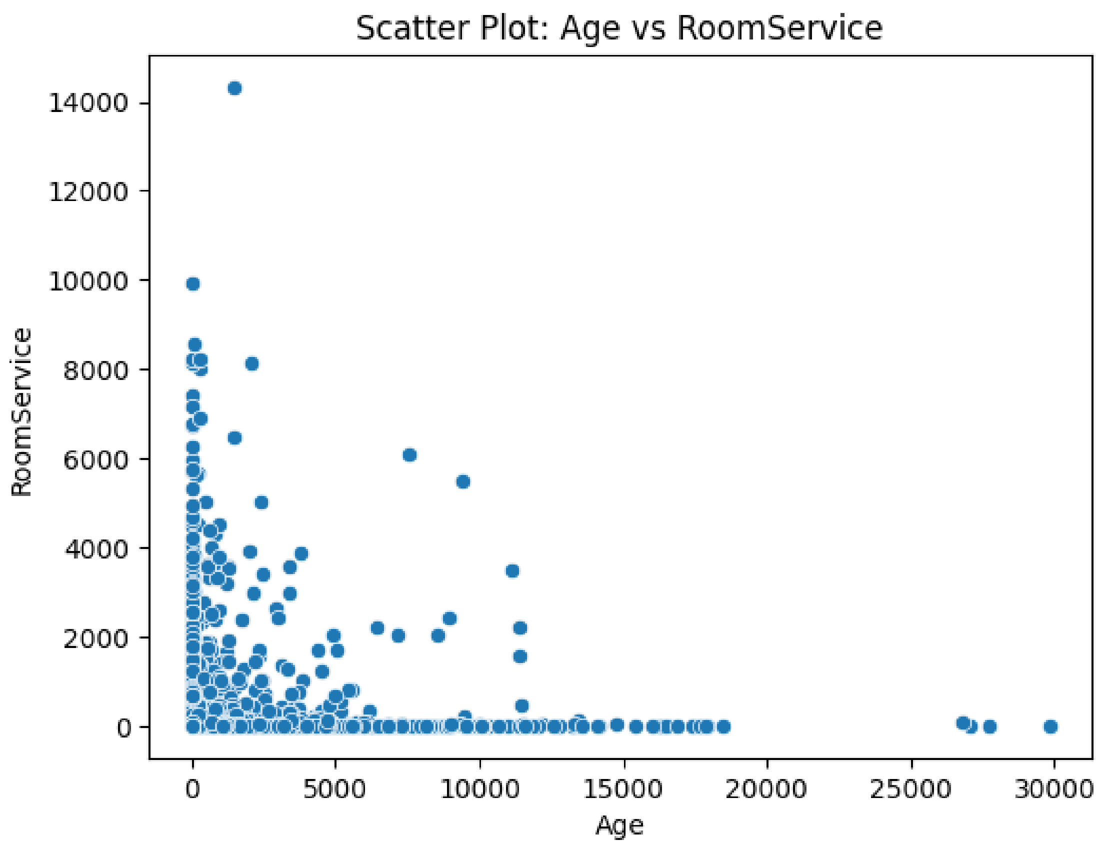
[75]:
```
  HomePlanet CryoSleep    VIP RoomService                         FoodCourt  \
                                     mean       sum count              mean
0      Earth     False  False  197.552676  586929.0  2971    197.056998
1      Earth      True  False    0.000000       0.0  1310      0.000000
2     Europa     False  False  242.744324  245900.0  1013   2627.767258
3     Europa     False   True  263.009434   27879.0   106   3105.271028
4     Europa      True  False    0.000000       0.0   856      0.000000
5     Europa      True   True    0.000000       0.0    20      0.000000
6       Mars     False  False  911.488842  857711.0   941     81.968220
7       Mars     False   True  844.360656   51506.0    61    163.516667
8       Mars      True  False    0.000000       0.0   637      0.000000


         sum
0   584274.0
1        0.0
2  2664556.0
3   332264.0
4        0.0
5        0.0
6    77378.0
7     9811.0
8        0.0
```

### 6.2.2 Scatterplots

[77]:
```
# Plot scatter plot for two variables (e.g., 'Age' and 'RoomService')
sns.scatterplot(x='FoodCourt', y='RoomService', data=df)
plt.title('Scatter Plot: Age vs RoomService')
plt.xlabel('Age')
plt.ylabel('RoomService')
```

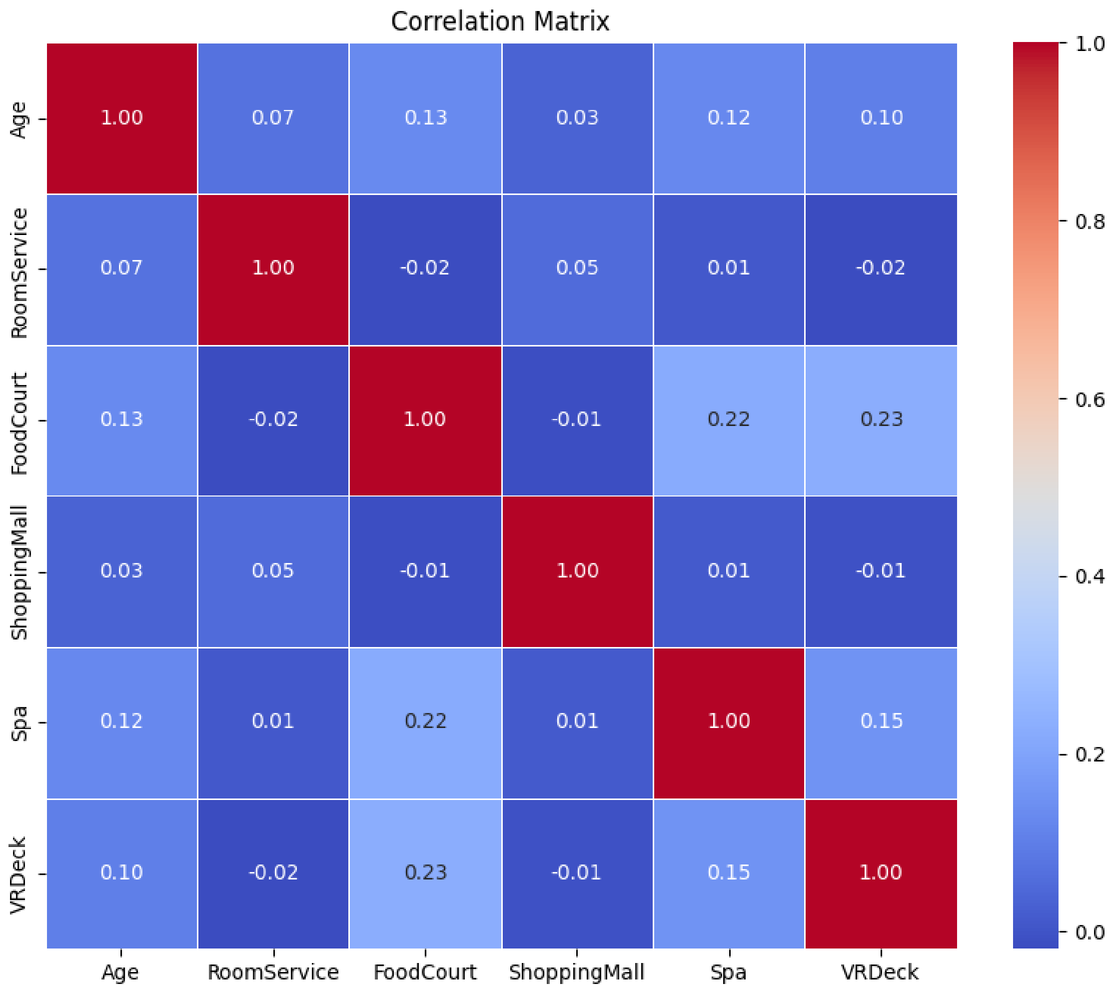
```
plt.show()
```



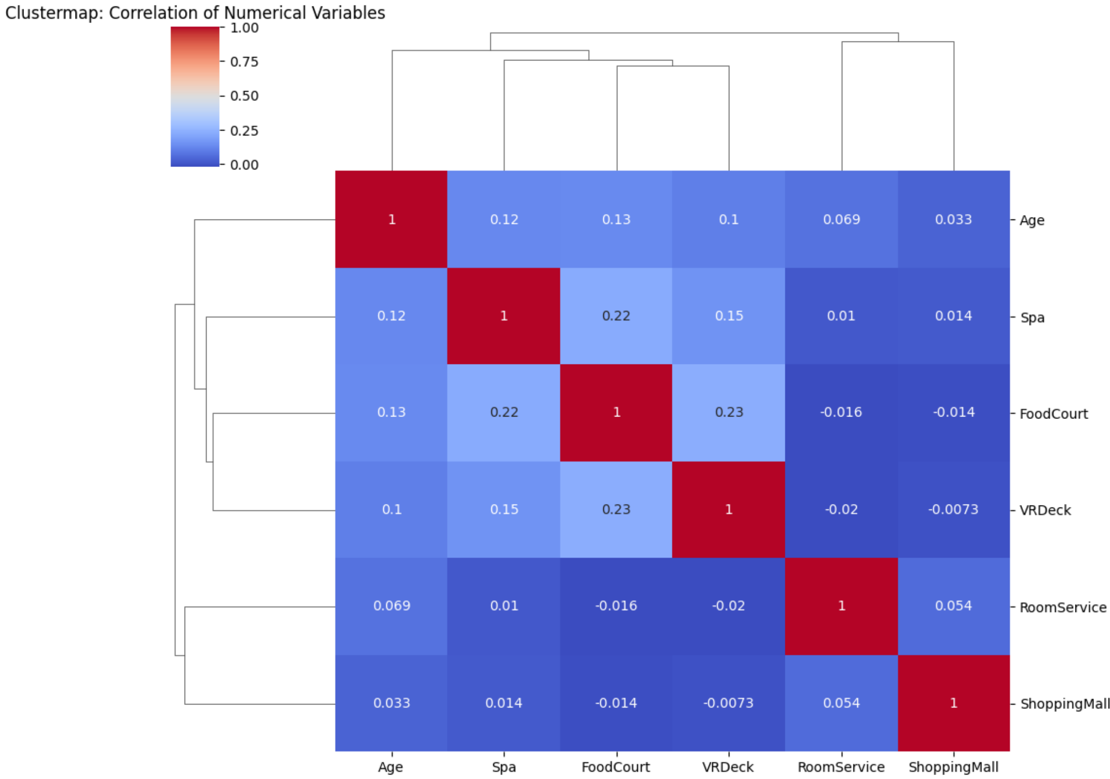

### 6.2.3 Corelation and Heatmap

[78]:
```
# Calculate the correlation matrix
corr_matrix = df[numerical_columns].corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

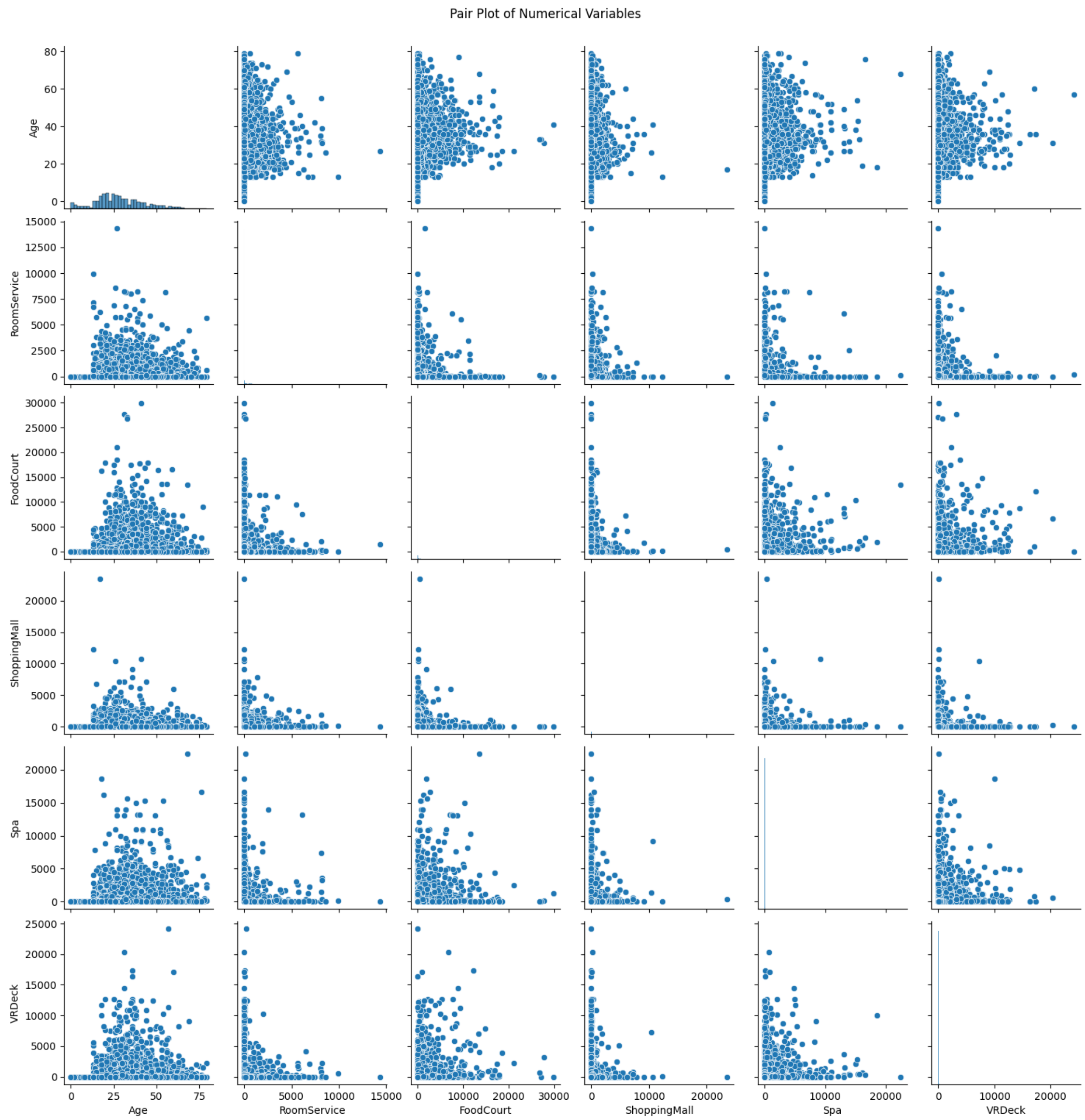



[79]:
```
# Create a clustermap for correlation matrix
sns.clustermap(corr_matrix, annot=True, cmap='coolwarm', figsize=(10, 8))
plt.title('Clustermap: Correlation of Numerical Variables')
plt.show()
```

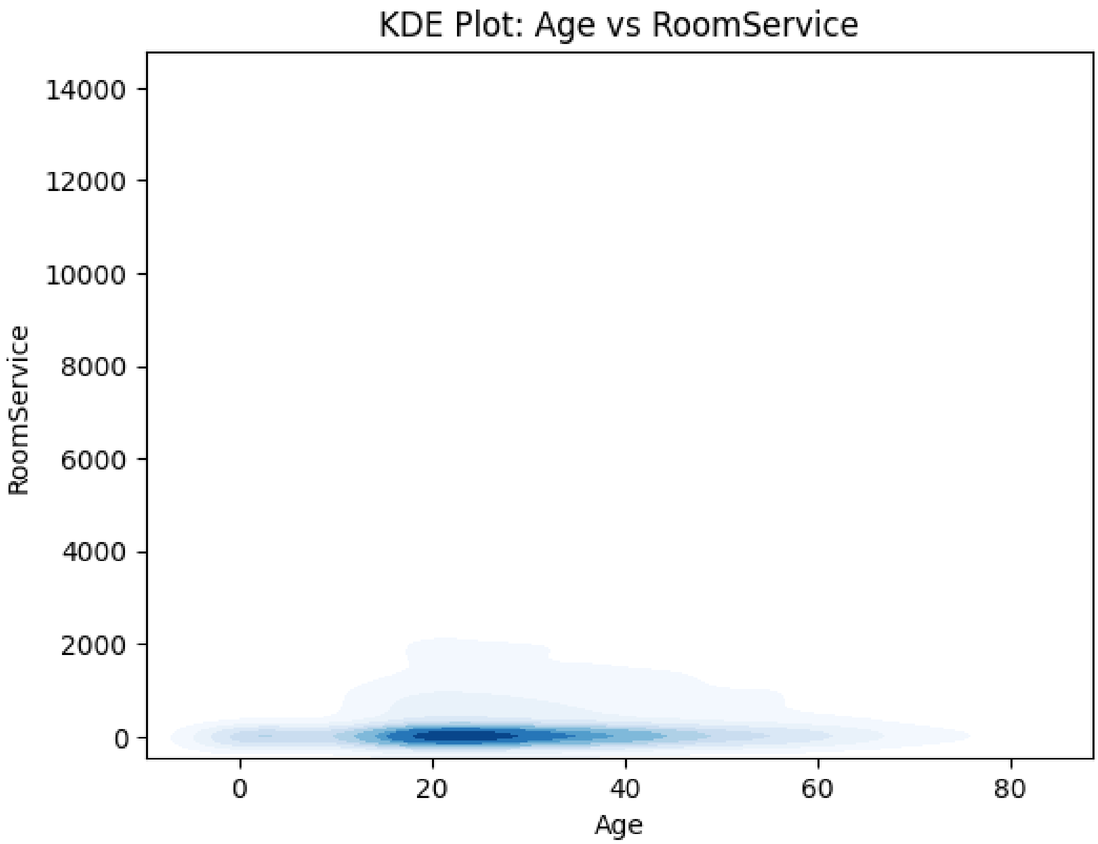



### 6.2.4 Pairplots

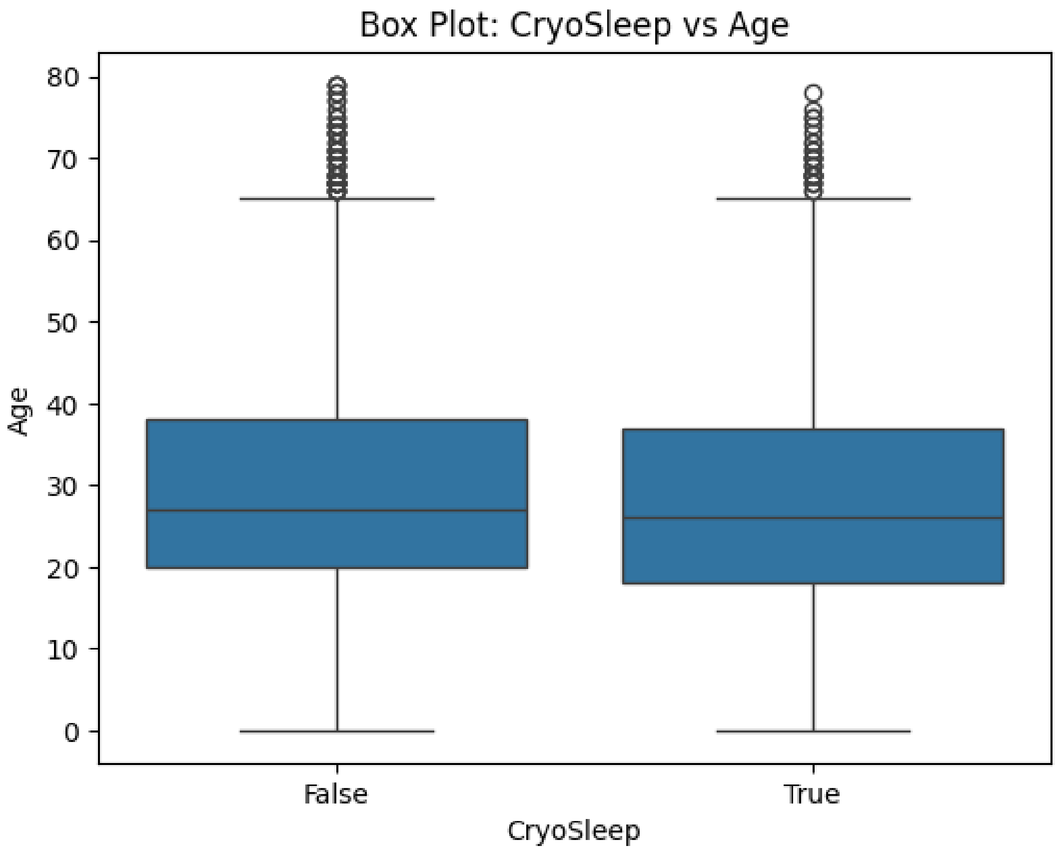
[80]:
```
# Plot pair plot for numerical columns
sns.pairplot(df[numerical_columns])
plt.suptitle('Pair Plot of Numerical Variables', y=1.02)
plt.show()
```




[81]:
```python
# KDE plot for two variables
sns.kdeplot(x='Age', y='RoomService', data=df, cmap='Blues', shade=True,␣
 ↪fill=True)
plt.title('KDE Plot: Age vs RoomService')
plt.show()
```

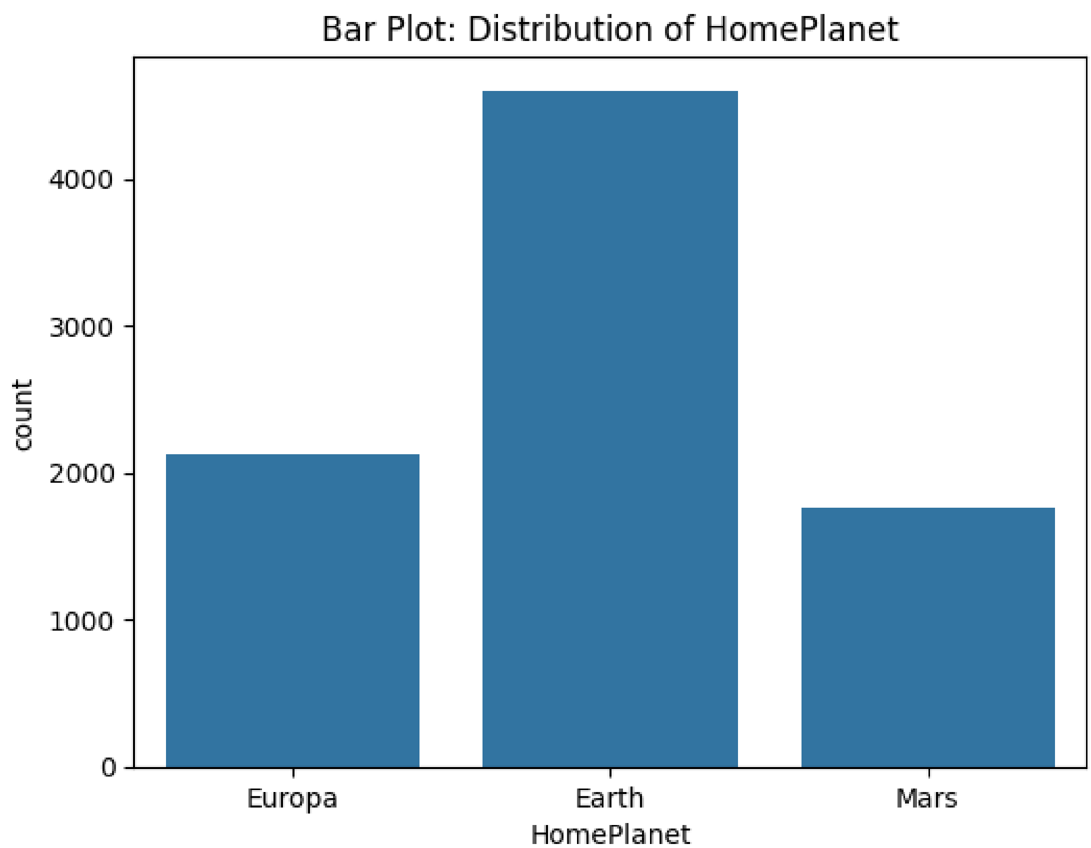



[82]:
```
# Box plot for numerical variable ('Age') across categories ('CryoSleep')
sns.boxplot(x='CryoSleep', y='Age', data=df)
plt.title('Box Plot: CryoSleep vs Age')
plt.show()
```

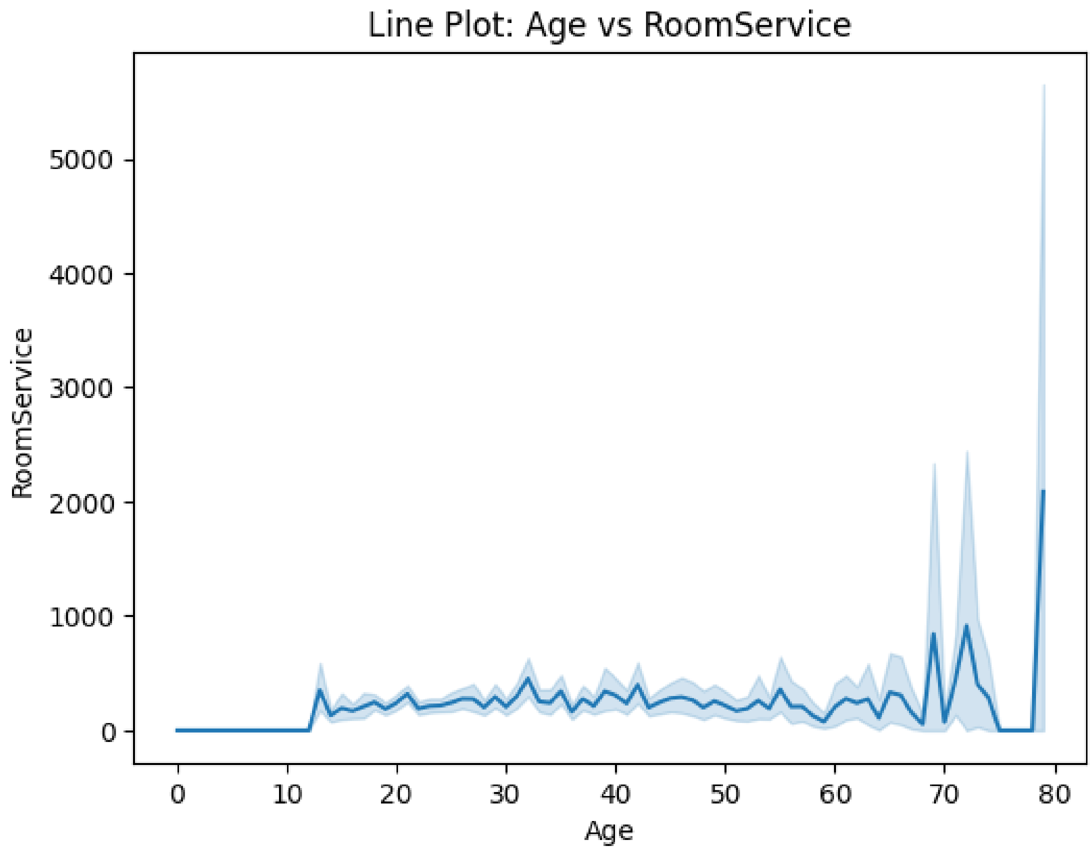



### 6.3 5.6. Data Visualization

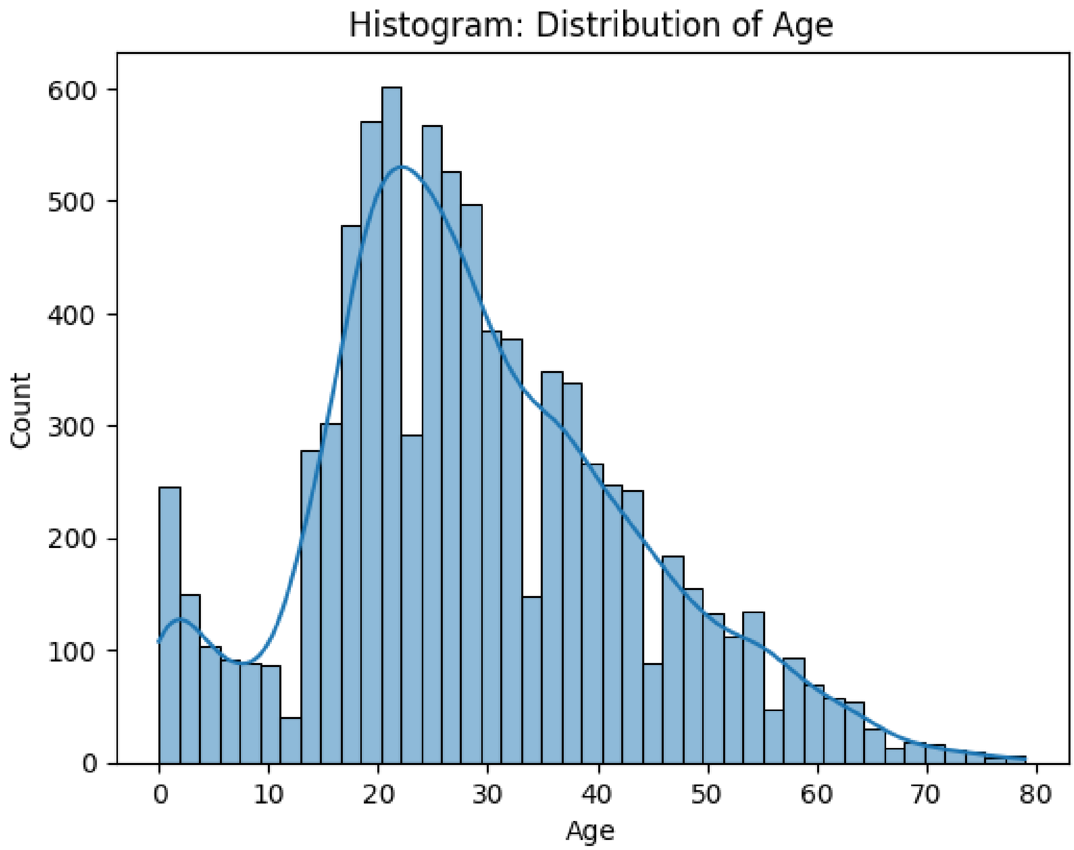
[83]:
```
# Bar plot for categorical data (e.g., distribution of 'HomePlanet')
sns.countplot(x='HomePlanet', data=df)
plt.title('Bar Plot: Distribution of HomePlanet')
plt.show()
```



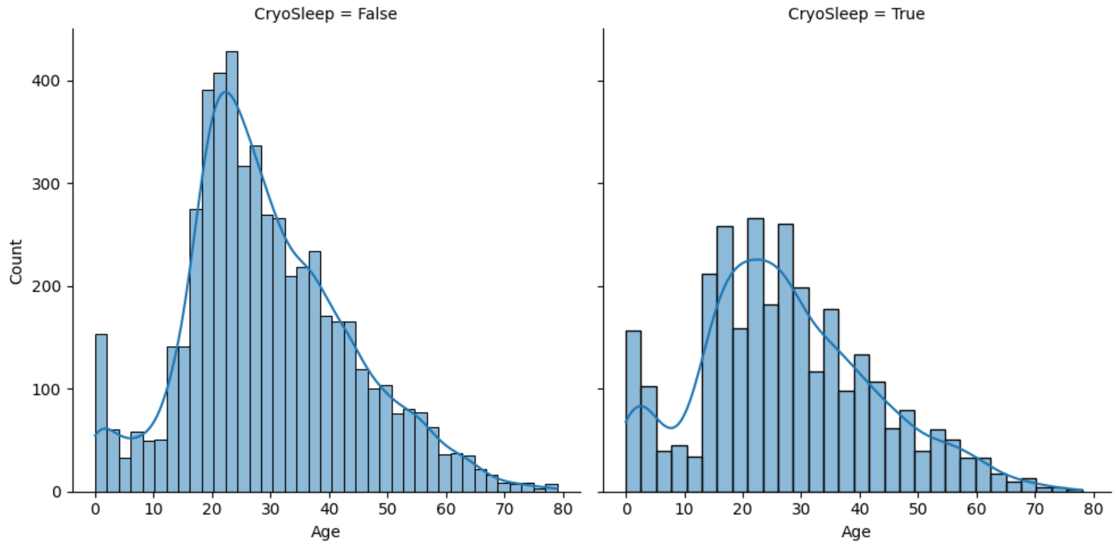

[84]:
```
# Line plot for numerical data (e.g., 'Age' over 'RoomService')
sns.lineplot(x='Age', y='RoomService', data=df)
plt.title('Line Plot: Age vs RoomService')
plt.show()
```



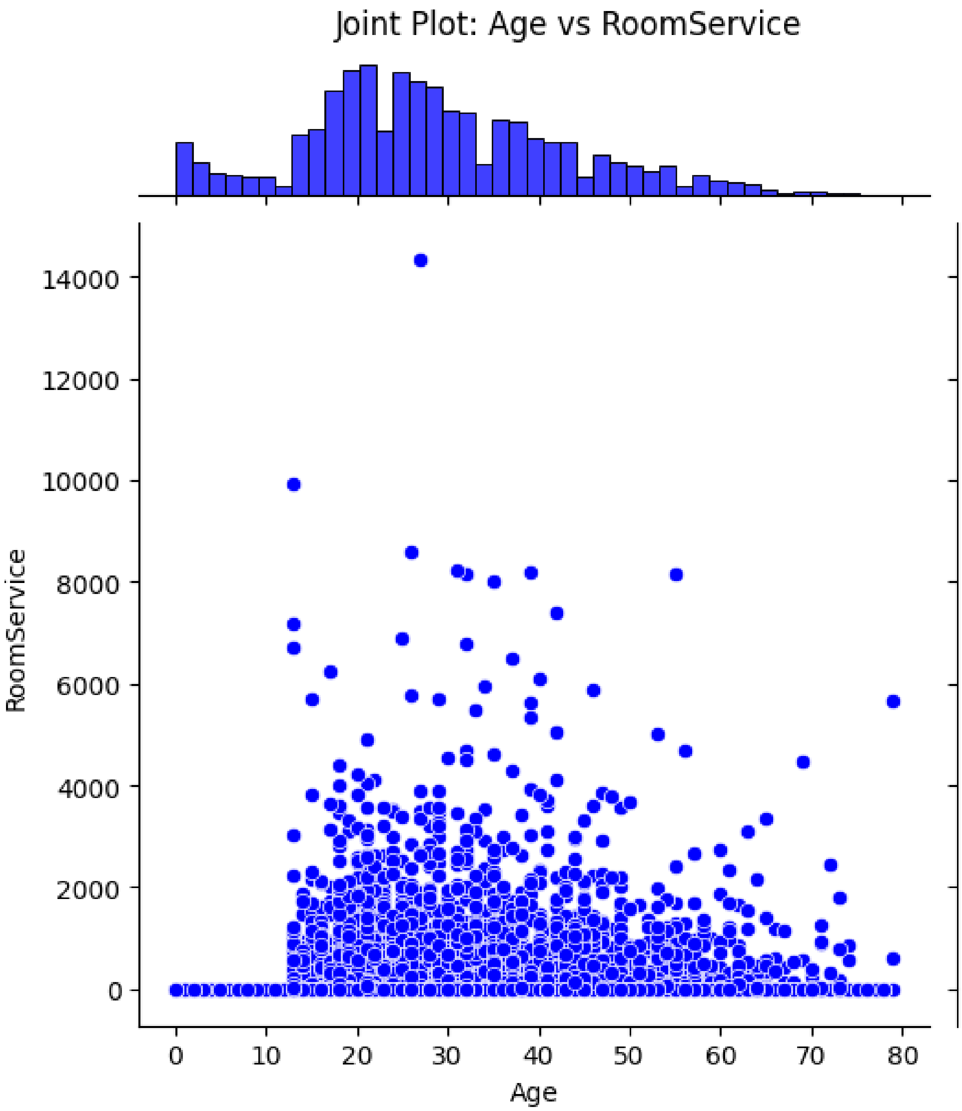

[85]:
```
# Histogram for numerical variable (e.g., 'Age')
sns.histplot(df['Age'], kde=True)
plt.title('Histogram: Distribution of Age')
plt.show()
```



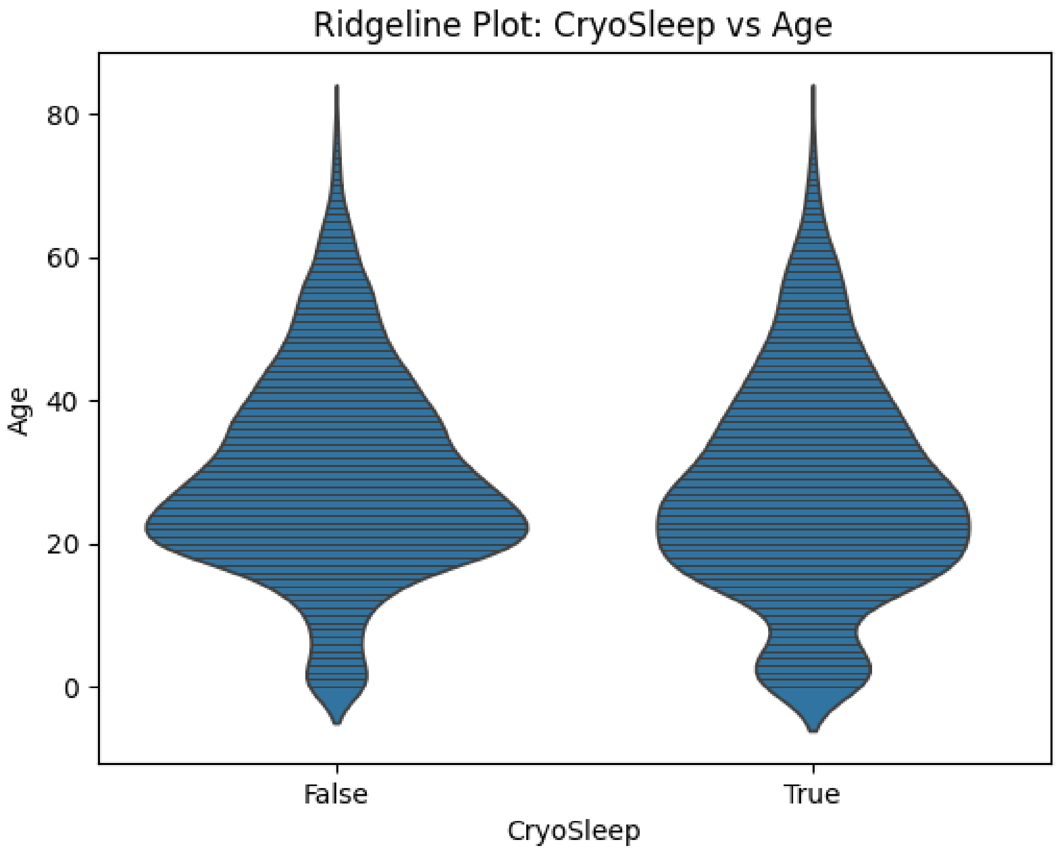

[86]:
```
# FacetGrid for visualizing the distribution of 'Age' across different␣
↪'CryoSleep' categories
g = sns.FacetGrid(df, col='CryoSleep', height=5)
g.map(sns.histplot, 'Age', kde=True)
g.set_titles('CryoSleep = {col_name}')
plt.show()
```



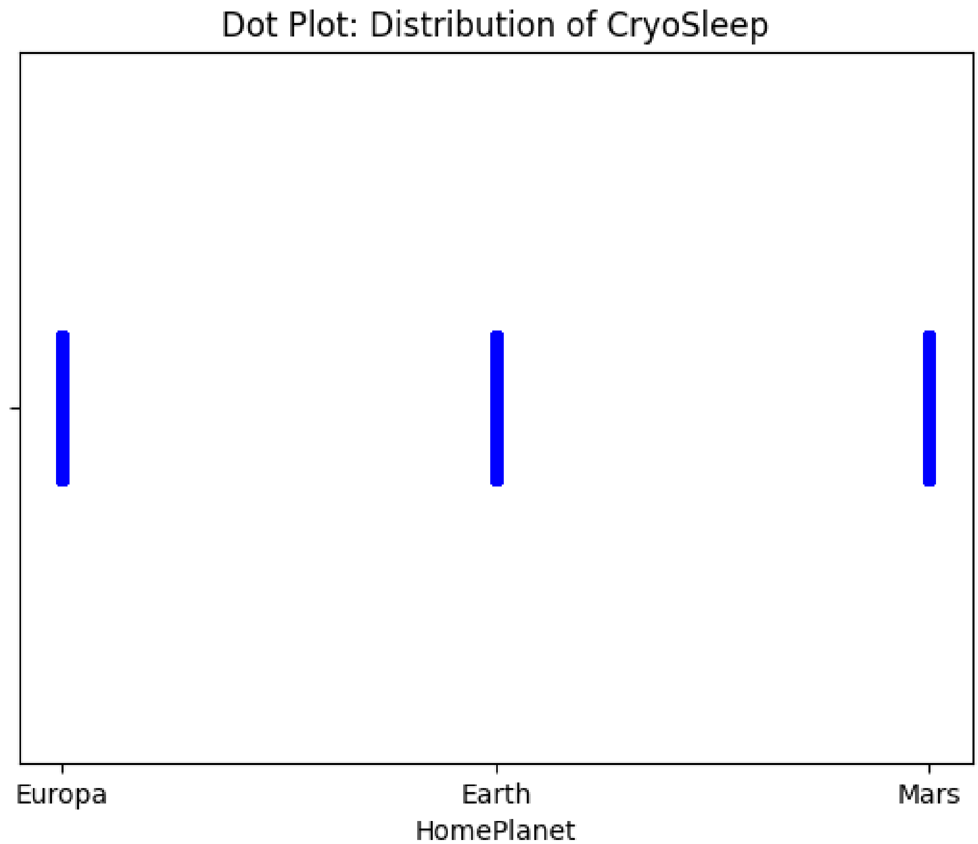

[87]:
```
# Joint plot for two numerical variables ('Age' vs 'RoomService')
sns.jointplot(x='Age', y='RoomService', data=df, kind='scatter', color='blue')
plt.suptitle('Joint Plot: Age vs RoomService', y=1.02)
plt.show()
```



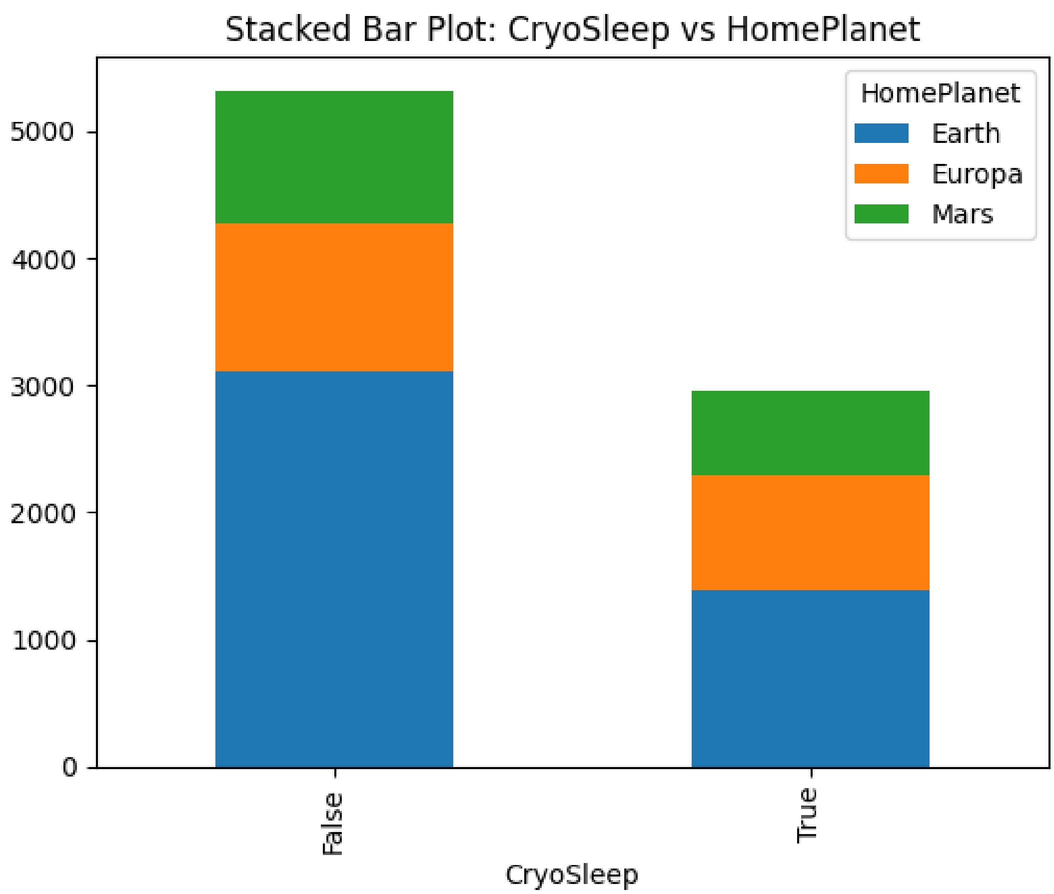

[88]:
```
# Ridgeline plot for distribution of 'Age' across different 'CryoSleep'␣
↪categories
sns.violinplot(x='CryoSleep', y='Age', data=df, inner="stick")
plt.title('Ridgeline Plot: CryoSleep vs Age')
plt.show()
```




[90]:
```
# Dot plot for categorical data ('CryoSleep')
sns.stripplot(x='HomePlanet', data=df, jitter=True, size=5, color='blue')
plt.title('Dot Plot: Distribution of CryoSleep')
plt.show()
```

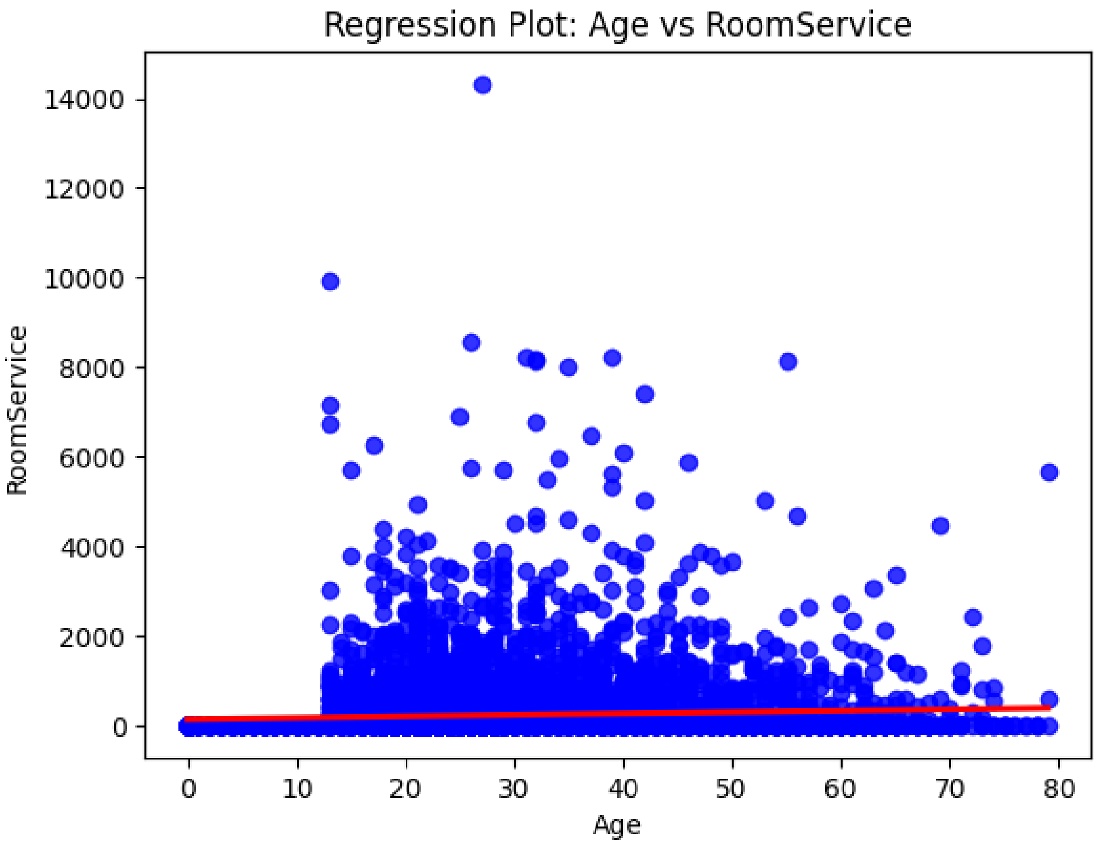



[91]:
```
# Stacked bar plot for categorical data
df.groupby(['CryoSleep', 'HomePlanet']).size().unstack().plot(kind='bar',␣
    ↪stacked=True)
plt.title('Stacked Bar Plot: CryoSleep vs HomePlanet')
plt.show()
```



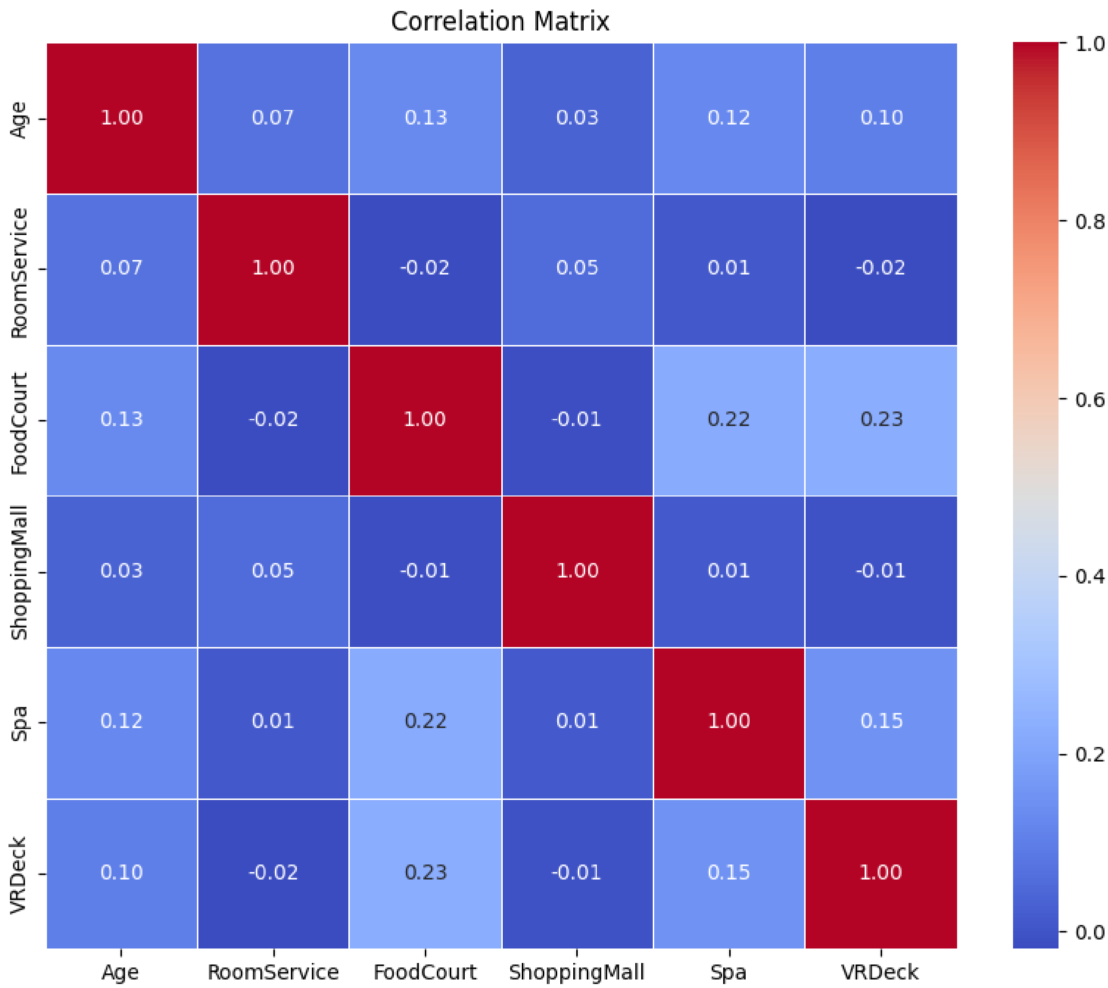

### 6.4 5.7. Identifying Patterns and Trends:

[92]:
```
# Regression plot for two variables ('Age' vs 'RoomService')
sns.regplot(x='Age', y='RoomService', data=df, scatter_kws={'color': 'blue'},␣
↪line_kws={'color': 'red'})
plt.title('Regression Plot: Age vs RoomService')
plt.show()
```




[93]:
```
# Calculate the correlation matrix
corr_matrix = df[numerical_columns].corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



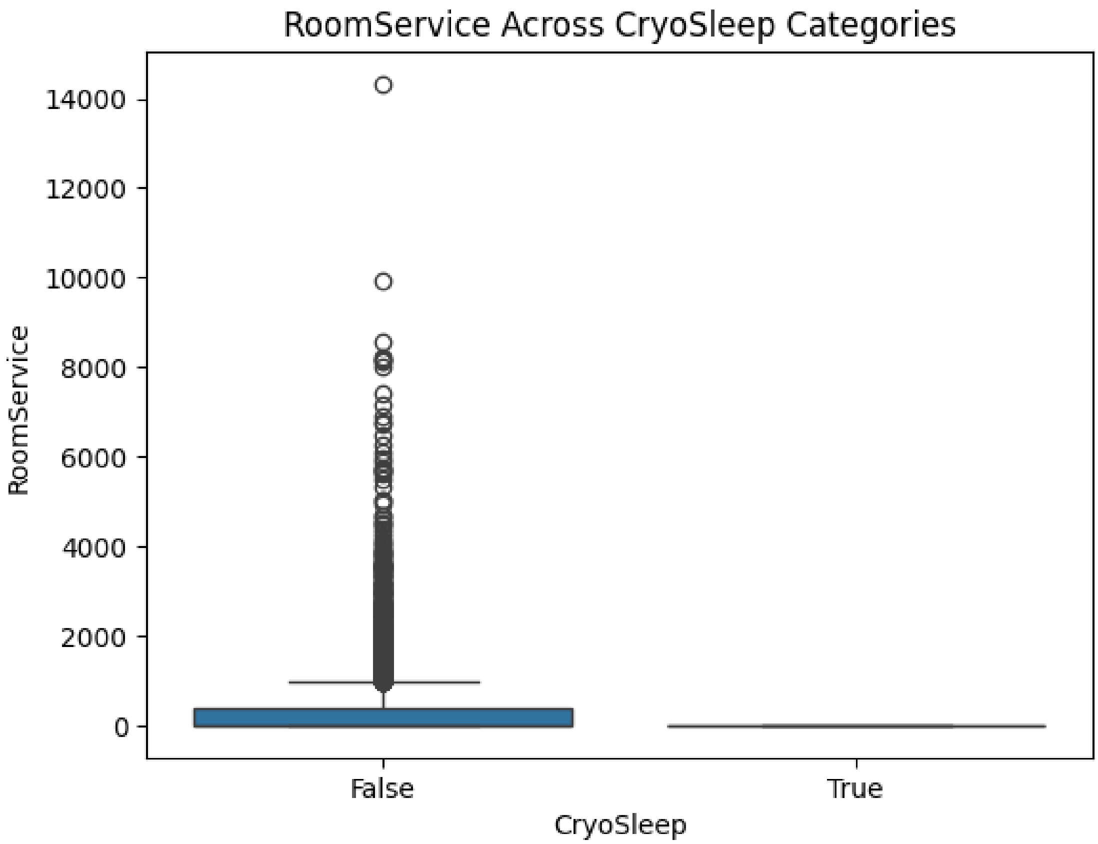

### 6.4.1 Aggregating Data to Identify Patterns

[94]:
```python
# Grouping by 'CryoSleep' and calculating the mean of 'RoomService'
aggregated_data = df.groupby('CryoSleep')['RoomService'].mean().reset_index()
aggregated_data
```

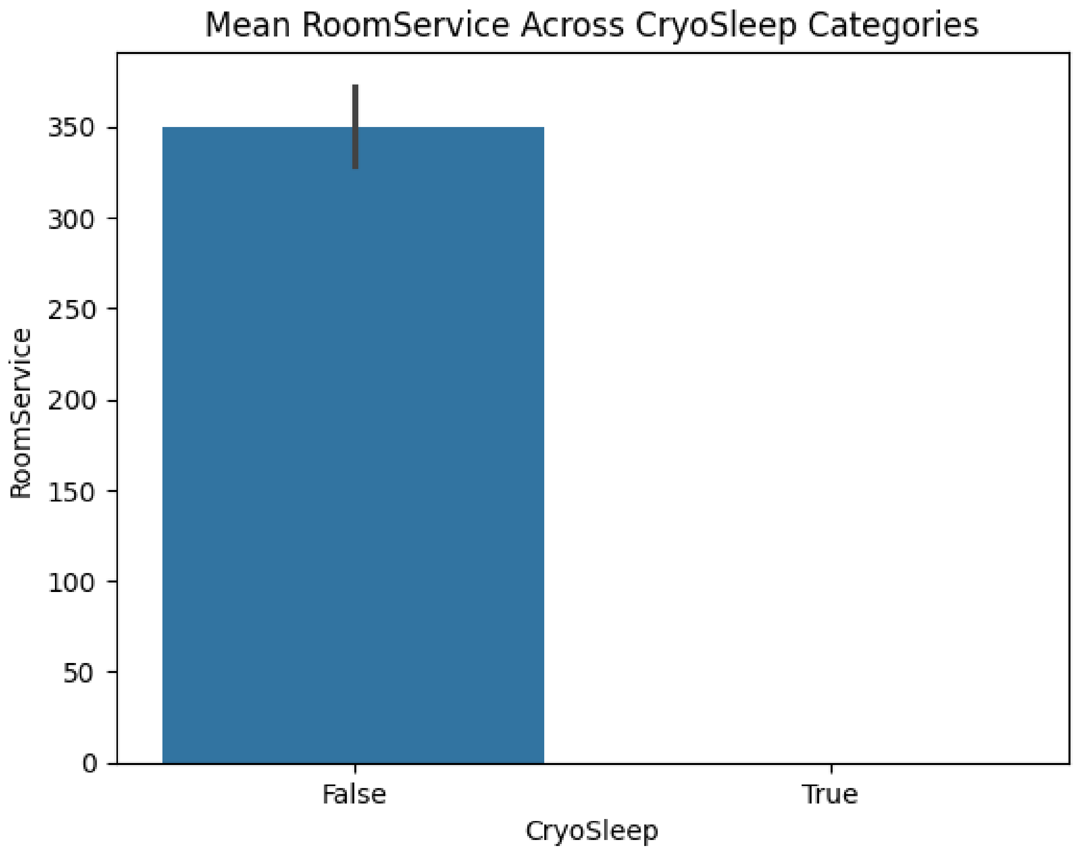
[94]:

| | CryoSleep | RoomService |
|---|---|---|
| 0 | False | 350.146772 |
| 1 | True | 0.000000 |

[95]:
```python
# Box plot for 'RoomService' across 'CryoSleep'
sns.boxplot(x='CryoSleep', y='RoomService', data=df)
plt.title('RoomService Across CryoSleep Categories')
plt.show()
```




[96]:
```python
# Bar plot for 'RoomService' mean across 'CryoSleep'
sns.barplot(x='CryoSleep', y='RoomService', data=df, estimator='mean')
plt.title('Mean RoomService Across CryoSleep Categories')
plt.show()
```



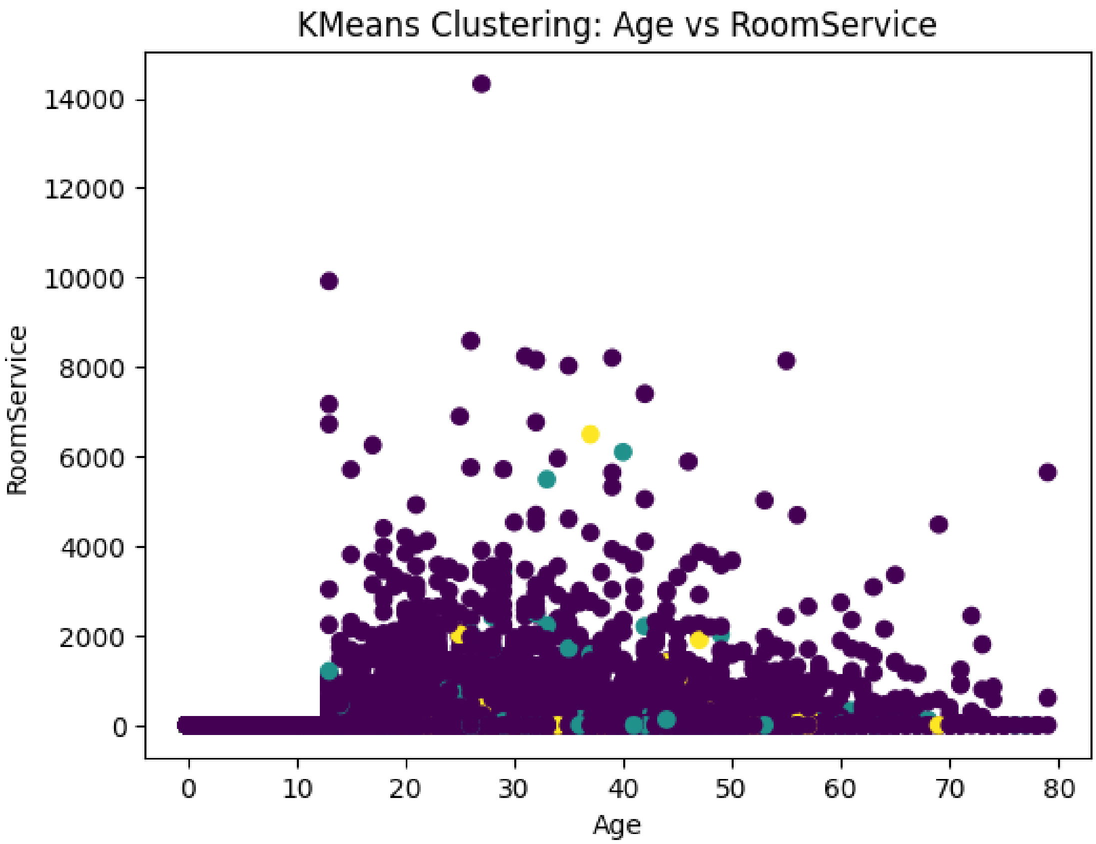

[97]:
```
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans

# Selecting numerical columns for clustering
X = df[['Age', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']]

# Impute missing values by the most frequent value (mode)
imputer = SimpleImputer(strategy='most_frequent')
X_imputed = imputer.fit_transform(X)

# Applying KMeans clustering
kmeans = KMeans(n_clusters=3)
df['Cluster'] = kmeans.fit_predict(X_imputed)

# Visualizing clusters with a scatter plot
plt.scatter(df['Age'], df['RoomService'], c=df['Cluster'], cmap='viridis')
plt.title('KMeans Clustering: Age vs RoomService')
plt.xlabel('Age')
plt.ylabel('RoomService')
plt.show()
```




## 6.5 5.8. Hypothesis Testing

### 6.5.1 Numerical Example: One-Sample t-Test

We'll use **Age** as the variable and test whether the average age is significantly different from 30.

#### Hypotheses

- **H** : The mean age is 30. ( = 30)
- **H** : The mean age is not 30. ( 30)

f `p-value <= 0.05`, we reject the null hypothesis and conclude that the mean age is significantly different from 30.

[98]:
```
# Hypothetical Data: Age column
age_data = df['Age'].dropna()   # Drop any NaN values

# Perform a one-sample t-test
from scipy import stats

t_stat, p_value = stats.ttest_1samp(age_data, 30)
```

```
print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")
```

```
T-statistic: -7.464163520677807
P-value: 9.209233395572929e-14
```

### 6.5.2 Numerical Example: Two-Sample t-Test

We'll compare the **RoomService** spending between two groups: people who have **CryoSleep** and people who don't.

**Hypotheses** - **H** : The mean RoomService spending is the same for both CryoSleep and non-CryoSleep passengers. - **H** : The mean RoomService spending is different for CryoSleep and non-CryoSleep passengers.

If `p-value <= 0.05`, we reject the null hypothesis and conclude that there is a significant difference in RoomService spending between the two groups.

[100]:
```
# Data: RoomService spending and CryoSleep status
cryo_sleep_yes = df[df['VIP'] == True]['RoomService'].dropna()
cryo_sleep_no = df[df['VIP'] == False]['RoomService'].dropna()

# Perform a two-sample t-test
t_stat, p_value = stats.ttest_ind(cryo_sleep_yes, cryo_sleep_no)

print(f"T-statistic: {t_stat}")
print(f"P-value: {p_value}")
```

```
T-statistic: 5.368367669553093
P-value: 8.159697243123737e-08
```

### 6.5.3 Categorical Example: Chi-Square Test of Independence

We'll test whether there is an association between **Destination** and **CryoSleep** status.

**Hypotheses** - **H** : There is no association between Destination and CryoSleep. - **H** : There is an association between Destination and CryoSleep.

If `p-value <= 0.05`, we reject the null hypothesis and conclude that there is an association between **Destination** and **CryoSleep** status.

[102]:
```
from scipy.stats import chi2_contingency

# Data: Destination and CryoSleep status
contingency_table = pd.crosstab(df['VIP'], df['CryoSleep'])

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Square Statistic: {chi2_stat}")
```

```
print(f"P-value: {p_value}")
```

```
Chi-Square Statistic: 53.75457042560369
P-value: 2.2716514281207392e-13
```

### 6.5.4 Categorical Example: Chi-Square Test for VIP and CryoSleep

We'll test whether there is a relationship between **VIP status** and **CryoSleep** (assumed to be a column in the dataset).

**Hypotheses** - **H** : There is no association between VIP status and CryoSleep. - **H** : There is an association between VIP status and CryoSleep.

If `p-value <= 0.05`, we reject the null hypothesis and conclude that there is an association between **VIP status** and **CryoSleep**.

[103]:
```
# Data: VIP status and Survival status (assuming Survival is a column in the␣
↪dataset)
contingency_table = pd.crosstab(df['VIP'], df['CryoSleep'])

# Perform the Chi-Square test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Square Statistic: {chi2_stat}")
print(f"P-value: {p_value}")
```

```
Chi-Square Statistic: 53.75457042560369
P-value: 2.2716514281207392e-13
```

# 7 7. Working with Dates

[104]:
```
date_df = pd.read_csv("/content/CIOL-Winter-ML-Bootcamp/datasets/session1/main/
↪date/data_date.csv")
date_df.head()
```

[104]:

| | Date | Country | Status | AQI Value |
|---|---|---|---|---|
| 0 | 2022-07-21 | Albania | Good | 14 |
| 1 | 2022-07-21 | Algeria | Moderate | 65 |
| 2 | 2022-07-21 | Andorra | Moderate | 55 |
| 3 | 2022-07-21 | Angola | Unhealthy for Sensitive Groups | 113 |
| 4 | 2022-07-21 | Argentina | Moderate | 63 |

[105]:
```
# Convert 'Date' column to datetime
date_df['Date'] = pd.to_datetime(date_df['Date'])
```

[107]:
```
date_df.dtypes
```

[107]:
```
Date          datetime64[ns]
Country               object
Status                object
AQI Value              int64
dtype: object
```

[108]:
```
# Extracting year, month, and day from the 'Date' column
date_df['Year'] = date_df['Date'].dt.year
date_df['Month'] = date_df['Date'].dt.month
date_df['Day'] = date_df['Date'].dt.day

# Display the updated dataframe
date_df.head()
```

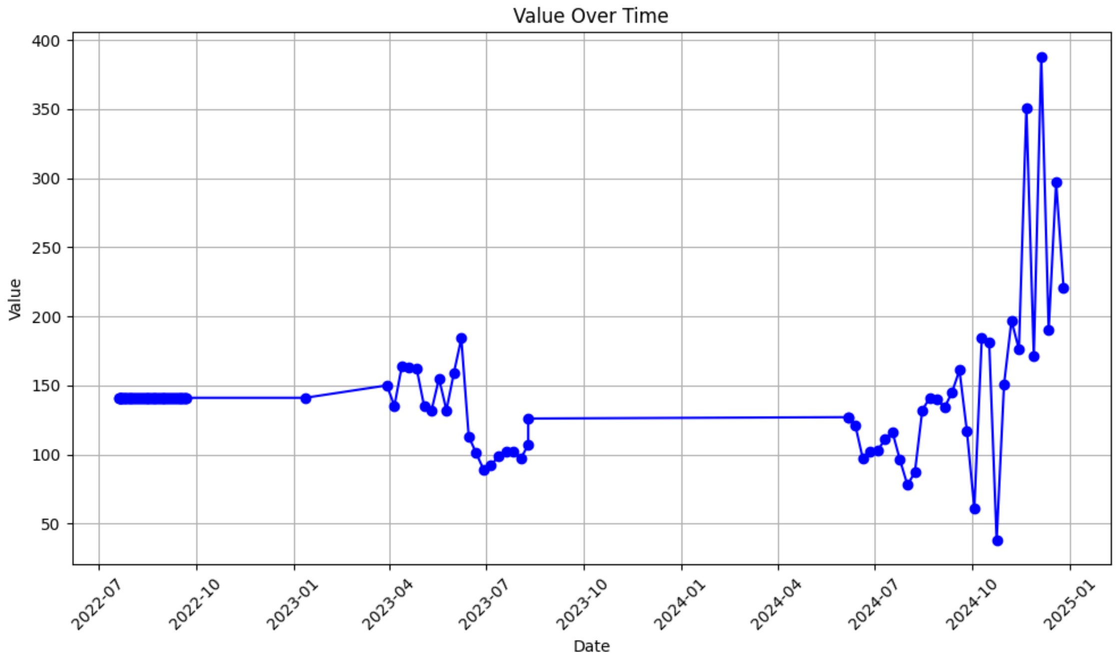
[108]:
```
        Date    Country                          Status  AQI Value  Year  \
0 2022-07-21    Albania                            Good         14  2022
1 2022-07-21    Algeria                        Moderate         65  2022
2 2022-07-21    Andorra                        Moderate         55  2022
3 2022-07-21     Angola  Unhealthy for Sensitive Groups        113  2022
4 2022-07-21  Argentina                        Moderate         63  2022

   Month  Day
0      7   21
1      7   21
2      7   21
3      7   21
4      7   21
```

[109]:
```
bangladesh_data = date_df[date_df['Country']=='Bangladesh']

# Plot the 'Value' column over time
plt.figure(figsize=(10, 6))
plt.plot(bangladesh_data['Date'], bangladesh_data['AQI Value'], marker='o',␣
 ↪linestyle='-', color='b')
plt.title('Value Over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```



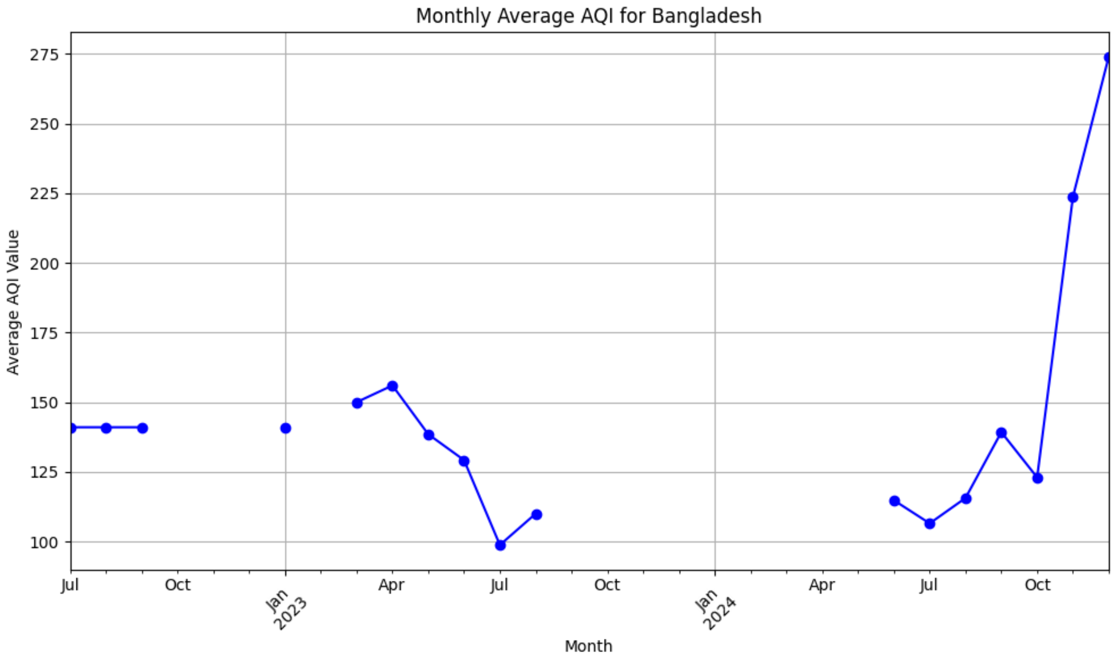

[110]:
```
## Resampling and Aggregating by Date

# Set 'Date' as the index for easier resampling
bangladesh_data.set_index('Date', inplace=True)

# Resample by month and calculate the mean AQI for each month
monthly_aqi = bangladesh_data.resample('M')['AQI Value'].mean()

# Plot the monthly average AQI for Bangladesh
plt.figure(figsize=(10, 6))
monthly_aqi.plot(marker='o', linestyle='-', color='b')
plt.title("Monthly Average AQI for Bangladesh")
plt.xlabel("Month")
plt.ylabel("Average AQI Value")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```

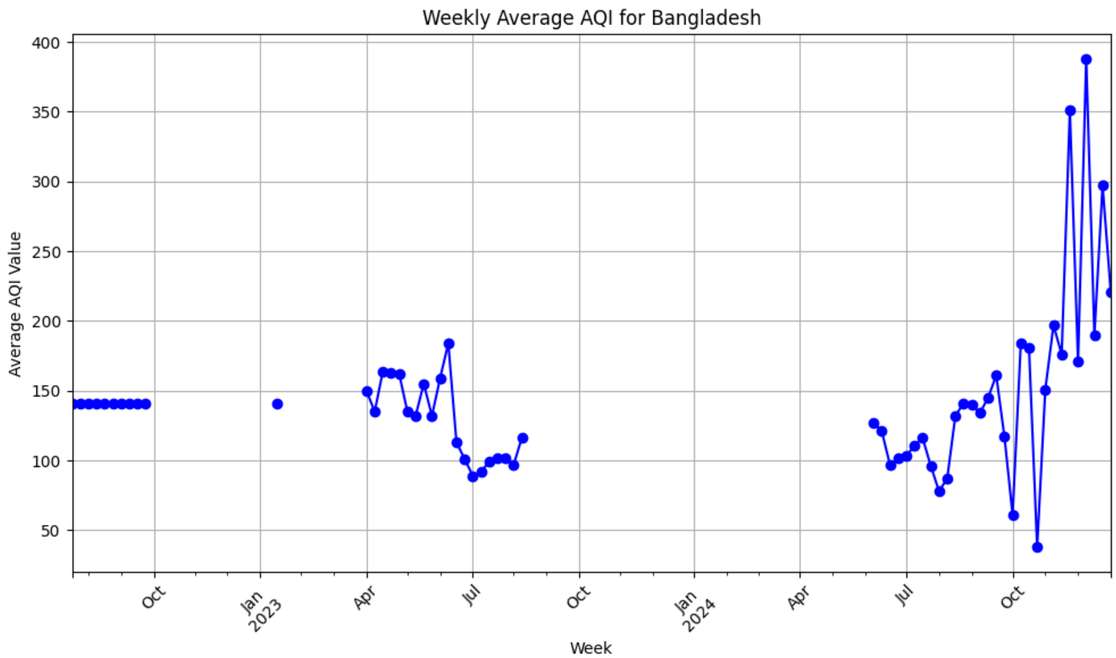



[111]:

```python
# Resample by week and calculate the mean AQI for each week
weekly_aqi = bangladesh_data.resample('W')['AQI Value'].mean()

# Plot the weekly average AQI for Bangladesh
plt.figure(figsize=(10, 6))
weekly_aqi.plot(marker='o', linestyle='-', color='b')
plt.title("Weekly Average AQI for Bangladesh")
plt.xlabel("Week")
plt.ylabel("Average AQI Value")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```




# 8 Final report

500 words

---

Thank you!!

---

If you use it, cite:

*Azmine Toushik Wasi. (2024). CIOL Presnts Winer ML BootCamp. https://github.com/ciol-researchlab/CIOL-Winter-ML-Bootcamp*

```
@misc{wasi2024CIOL-WMLB,       title={CIOL Presnts Winer ML
BootCamp},       author={Azmine Toushik Wasi},       year={2024},
url={https://github.com/ciol-researchlab/CIOL-Winter-ML-Bootcamp}, }
```