In [1]:

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df = pd.read_csv("C:/Users/sahil/Documents/ecommerce.csv")
print(df)
```

```
            Order ID Order Date Ship Date  Aging       Ship Mode  \
0          AU-2015-1    11/9/15  11/17/15    8.0     First Class
1          AU-2015-2    6/30/15    7/2/15    2.0     First Class
2          AU-2015-3    12/5/15  12/13/15    8.0     First Class
3          AU-2015-4     5/9/15   5/16/15    7.0     First Class
4          AU-2015-5     7/9/15   7/18/15    9.0     First Class
...              ...        ...       ...    ...             ...
51285  FA-2015-30771    1/21/15   1/27/15    6.0  Standard Class
51286  FA-2015-30772    6/22/15   6/24/15    2.0  Standard Class
51287  FA-2015-30773     1/1/15    1/7/15    6.0  Standard Class
51288  FA-2015-30774    12/7/15  12/14/15    7.0  Standard Class
51289  FA-2015-30775    12/1/15   12/6/15    5.0  Standard Class

         Product Category            Product     Sales Quantity Discount  ...  \
0      Auto & Accessories  Car Media Players  $140.00         2     0.05  ...
1      Auto & Accessories       Car Speakers  $211.00         3     0.03  ...
2      Auto & Accessories    Car Body Covers  $117.00         5     0.01  ...
3      Auto & Accessories    Car & Bike Care  $118.00         2     0.05  ...
4      Auto & Accessories               Tyre  $250.00         1     0.04  ...
...                   ...                ...       ...      ...      ...  ...
51285             Fashion        Sports Wear   $85.00         5     0.04  ...
51286             Fashion        Sports Wear   $85.00         1     0.03  ...
51287             Fashion        Sports Wear   $85.00         1     0.05  ...
51288             Fashion        Sports Wear   $85.00         3     0.04  ...
51289             Fashion        Sports Wear   $85.00         3     0.03  ...

      Shipping Cost Order Priority Customer ID    Customer Name      Segment  \
0             $4.60          Medium      LS-001     Lane Daniels     Consumer
1            $11.20          Medium      IZ-002    Alvarado Kriz  Home Office
2             $3.10        Critical      EN-003       Moon Weien     Consumer
3             $2.60            High      AN-004  Sanchez Bergman    Corporate
4            $16.00        Critical      ON-005     Rowe Jackson    Corporate
...             ...            ...         ...              ...          ...
51285         $1.70          Medium  IN-0040977       Welch Fein    Corporate
51286         $0.20          Medium  TT-0040978  Martinez Arnett    Corporate
51287         $0.10          Medium  ON-0040979     Mccoy Duston  Home Office
51288         $2.80          Medium  RN-0040980   Bentley Zypern     Consumer
51289         $2.80          Medium  RZ-0040981  Mcclure Schwarz  Home Office

             City       State        Country        Region Months
0        Brisbane  Queensland      Australia       Oceania    Nov
1          Berlin      Berlin        Germany       Central    Jun
2         Porirua  Wellington    New Zealand       Oceania    Dec
3           Kabul       Kabul    Afghanistan  Central Asia    May
4      Townsville  Queensland      Australia       Oceania    Jul
...           ...         ...            ...           ...    ...
51285    Pasadena       Texas  United States       Central    Jan
51286      Harare      Harare       Zimbabwe        Africa    Jun
51287  Townsville  Queensland      Australia       Oceania    Jan
51288     Houston       Texas  United States       Central    Dec
51289    Valinhos   São Paulo         Brazil         South    Dec

[51290 rows x 21 columns]
```

```
C:\Users\sahil\AppData\Local\Temp\ipykernel_18252\3304675805.py:1: DtypeWarning: Col
umns (8,9) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("C:/Users/sahil/Documents/ecommerce.csv")
```

In [3]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Order ID          51290 non-null  object 
 1   Order Date        51289 non-null  object 
 2   Ship Date         51290 non-null  object 
 3   Aging             51289 non-null  float64
 4   Ship Mode         51290 non-null  object 
 5   Product Category  51290 non-null  object 
 6   Product           51290 non-null  object 
 7   Sales             51290 non-null  object 
 8   Quantity          51289 non-null  object 
 9   Discount          51290 non-null  object 
 10  Profit            51290 non-null  object 
 11  Shipping Cost     51290 non-null  object 
 12  Order Priority    51288 non-null  object 
 13  Customer ID       51289 non-null  object 
 14  Customer Name     51290 non-null  object 
 15  Segment           51289 non-null  object 
 16  City              51290 non-null  object 
 17  State             51290 non-null  object 
 18  Country           51290 non-null  object 
 19  Region            51289 non-null  object 
 20  Months            51290 non-null  object 
dtypes: float64(1), object(20)
memory usage: 8.2+ MB
None
```

In [4]:

```
print(df.isnull().sum())
```

```
Order ID            0
Order Date          1
Ship Date           0
Aging               1
Ship Mode           0
Product Category    0
Product             0
Sales               0
Quantity            1
Discount            0
Profit              0
Shipping Cost       0
Order Priority      2
Customer ID         1
Customer Name       0
Segment             1
City                0
State               0
Country             0
Region              1
Months              0
dtype: int64
```

In [8]:

```
df[['Ship Date','Order Date']] = df[['Ship Date','Order Date']].apply(lambda col: p
```

In [9]:

```
df[['Sales', 'Shipping Cost', 'Profit']] = (df[['Sales', 'Shipping Cost', 'Profit']
                                            .astype(float))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 2
      1 df[['Sales', 'Shipping Cost', 'Profit']] = (df[['Sales', 'Shipping Cost', 'P
rofit']].replace({r'\$': ''}, regex=True)
----> 2                                             .astype(float))

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\generi
c.py:6643, in NDFrame.astype(self, dtype, copy, errors)
   6637     results = [
   6638         ser.astype(dtype, copy=copy, errors=errors) for _, ser in self.items
()
   6639     ]
   6641 else:
   6642     # else, only a single dtype is given
-> 6643     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   6644     res = self._constructor_from_mgr(new_data, axes=new_data.axes)
   6645     return res.__finalize__(self, method="astype")

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\interna
ls\managers.py:430, in BaseBlockManager.astype(self, dtype, copy, errors)
    427 elif using_copy_on_write():
    428     copy = False
--> 430 return self.apply(
    431     "astype",
    432     dtype=dtype,
    433     copy=copy,
    434     errors=errors,
    435     using_cow=using_copy_on_write(),
    436 )

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\interna
ls\managers.py:363, in BaseBlockManager.apply(self, f, align_keys, **kwargs)
    361         applied = b.apply(f, **kwargs)
    362     else:
--> 363         applied = getattr(b, f)(**kwargs)
    364     result_blocks = extend_blocks(applied, result_blocks)
    366 out = type(self).from_blocks(result_blocks, self.axes)

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\interna
ls\blocks.py:758, in Block.astype(self, dtype, copy, errors, using_cow, squeeze)
    755         raise ValueError("Can not squeeze with more than one column.")
    756     values = values[0, :]  # type: ignore[call-overload]
--> 758 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    760 new_values = maybe_coerce_values(new_values)
    762 refs = None

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\dtypes
\astype.py:237, in astype_array_safe(values, dtype, copy, errors)
    234     dtype = dtype.numpy_dtype
    236 try:
--> 237     new_values = astype_array(values, dtype, copy=copy)
    238 except (ValueError, TypeError):
    239     # e.g. _astype_nansafe can fail on object-dtype of strings
    240     #  trying to convert to float
    241     if errors == "ignore":
```

```
File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\dtypes
\astype.py:182, in astype_array(values, dtype, copy)
    179     values = values.astype(dtype, copy=copy)
    181 else:
--> 182     values = _astype_nansafe(values, dtype, copy=copy)
    184 # in pandas we don't store numpy str dtypes, so convert to object
    185 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\pandas\core\dtypes
\astype.py:133, in _astype_nansafe(arr, dtype, copy, skipna)
    129     raise ValueError(msg)
    131 if copy or arr.dtype == object or dtype == object:
    132     # Explicit copy, or required since NumPy can't view from / to object.
--> 133     return arr.astype(dtype, copy=True)
    135 return arr.astype(dtype, copy=copy)

ValueError: could not convert string to float: '0.xf'
```

```
In [10]: df['Sales'].value_counts()
```

Out[10]:

```
Sales
$228.00     3823
$85.00      2827
$159.00     2796
$224.00     2795
$213.00     2795
$122.00     2795
$109.00     2795
$62.00      2795
$248.00     2794
$196.00     2794
$218.00     2794
$211.00     1853
$250.00     1114
$133.00     1053
$70.00      1029
$119.00     1029
$124.00     1029
$67.00      1029
$78.00      1029
$34.00      1028
$216.00     1027
$231.00      829
$114.00      827
$140.00      826
$54.00       826
$72.00       826
$117.00      826
$118.00      826
$130.00      261
$192.00      224
$83.00       224
$65.00       224
$199.00      221
$33.00       221
$104.00      221
$220.00      221
$111.00      221
$222.00      221
$149.00      221
0.xf           1
Name: count, dtype: int64
```

In [11]:

```python
df['Sales'] = df['Sales'].replace('0.xf', np.nan)
```

In [12]:

```python
df = (df
        .replace({'Sales': {'0.xf': np.nan},
                   'Shipping Cost': {'test': np.nan},
                   'Region': {'So3th': 'South', '4orth': 'North'},
                   'Quantity': {'abc': np.nan},
                   'Discount': {'xxx': np.nan}
                 }))
```

In [13]:

```python
data = (data
        .replace({'Sales': {'0.xf': np.nan},
                    'Shipping Cost': {'test': np.nan},
                    'Region': {'So3th': 'South', '4orth': 'North'},
                    'Quantity': {'abc': np.nan}})
         .fillna({'Order Date': '2015-04-17'}))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 data = (data
      2         .replace({'Sales': {'0.xf': np.nan},
      3                    'Shipping Cost': {'test': np.nan},
      4                    'Region': {'So3th': 'South', '4orth': 'North'},
      5                    'Quantity': {'abc': np.nan}})
      6         .fillna({'Order Date': '2015-04-17'}))

NameError: name 'data' is not defined
```

In [14]:

```
df[['Sales', 'Shipping Cost', 'Profit']] = (df[['Sales', 'Shipping Cost', 'Profit']
                                             .astype(float))
df.head(2)
```

Out[14]:

	Order ID	Order Date	Ship Date	Aging	Ship Mode	Product Category	Product	Sales	Quantity	Discount	...
0	AU-2015-1	2015-11-09	2015-11-17	8.0	First Class	Auto & Accessories	Car Media Players	140.0	2	0.05	...
1	AU-2015-2	2015-06-30	2015-07-02	2.0	First Class	Auto & Accessories	Car Speakers	211.0	3	0.03	...

2 rows × 21 columns

In [15]:

```
df[df.columns[0:4]].head(5)
```

Out[15]:

	Order ID	Order Date	Ship Date	Aging
0	AU-2015-1	2015-11-09	2015-11-17	8.0
1	AU-2015-2	2015-06-30	2015-07-02	2.0
2	AU-2015-3	2015-12-05	2015-12-13	8.0
3	AU-2015-4	2015-05-09	2015-05-16	7.0
4	AU-2015-5	2015-07-09	2015-07-18	9.0

In [16]:

```
df[['Quantity', 'Discount']] = df[['Quantity', 'Discount']].astype(float)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   Order ID          51290 non-null  object        
 1   Order Date        51289 non-null  datetime64[ns]
 2   Ship Date         51290 non-null  datetime64[ns]
 3   Aging             51289 non-null  float64       
 4   Ship Mode         51290 non-null  object        
 5   Product Category  51290 non-null  object        
 6   Product           51290 non-null  object        
 7   Sales             51289 non-null  float64       
 8   Quantity          51288 non-null  float64       
 9   Discount          51289 non-null  float64       
 10  Profit            51290 non-null  float64       
 11  Shipping Cost     51289 non-null  float64       
 12  Order Priority    51288 non-null  object        
 13  Customer ID       51289 non-null  object        
 14  Customer Name     51290 non-null  object        
 15  Segment           51289 non-null  object        
 16  City              51290 non-null  object        
 17  State             51290 non-null  object        
 18  Country           51290 non-null  object        
 19  Region            51289 non-null  object        
 20  Months            51290 non-null  object        
dtypes: datetime64[ns](2), float64(6), object(13)
memory usage: 8.2+ MB
```

In [17]:

```
df.isnull().sum()
```

Out[17]:

```
Order ID            0
Order Date          1
Ship Date           0
Aging               1
Ship Mode           0
Product Category    0
Product             0
Sales               1
Quantity            2
Discount            1
Profit              0
Shipping Cost       1
Order Priority      2
Customer ID         1
Customer Name       0
Segment             1
City                0
State               0
Country             0
Region              1
Months              0
dtype: int64
```

In [18]:

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

```
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split


# Using .loc for label-based indexing
column_name = 'Sales'
row_label = 793
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")

# Encode the 'Product' column
le = LabelEncoder()
df['Product'] = le.fit_transform(df['Product'])

# Create a validation set by setting known values to NaN
validation_indices = [1, 3]  # Indices of known values to set as NaN for validation
df_validation = df.copy()
df_validation.loc[validation_indices, 'Sales'] = np.nan

# Separate features and target for the imputer
features = df[['Product', 'Quantity', 'Discount']]
target = df[['Sales']]

# Combine features and target for the imputer
data_for_imputer = pd.concat([features, target], axis=1)

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Fit and transform the data
imputed_data = imputer.fit_transform(data_for_imputer)

# Update the dataframe with imputed values
df['Sales'] = imputed_data[:, -1]

# Create a new DataFrame with imputed values
df_imputed = df_validation.copy()
df_imputed['Sales'] = imputed_data[:, -1]


# Reverse the encoding for the 'Product' column
df['Product'] = le.inverse_transform(df['Product'])
df_imputed['Product'] = le.inverse_transform(df_imputed['Product'].astype(int))

# Convert the 'Product' column back to object type for both DataFrames
df['Product'] = df['Product'].astype(object)
df_imputed['Product'] = df_imputed['Product'].astype(object)

# Verify that 'Product' column is now correctly decoded
print("Data after converting 'Product' column to object:\n", df['Product'].head(5))
# print("Imputed DataFrame after converting 'Product' column to object:\n", df_impu

# Calculate the mean squared error and R-squared for the imputed values
actual_values = df.loc[validation_indices, 'Sales']
```

```
imputed_values = df_imputed.loc[validation_indices, 'Sales']
mse = mean_squared_error(actual_values, imputed_values)
r2 = r2_score(actual_values, imputed_values)

print("Accuracy of Model")
print(f'Mean Squared Error of the imputation: {mse}')
print(f'R-squared of the imputation: {r2}')



# Using .loc for label-based indexing
column_name = 'Sales'
row_label = 793
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")
```

```
Value at column 'Sales' and row 793: nan
Data after converting 'Product' column to object:
 0    Car Media Players
1          Car Speakers
2       Car Body Covers
3       Car & Bike Care
4                  Tyre
Name: Product, dtype: object
Accuracy of Model
Mean Squared Error of the imputation: 0.0
R-squared of the imputation: 1.0

Value at column 'Sales' and row 793: 211.0
```

In [19]:

```
df.isnull().sum()
```

Out[19]:

```
Order ID            0
Order Date          1
Ship Date           0
Aging               1
Ship Mode           0
Product Category    0
Product             0
Sales               0
Quantity            2
Discount            1
Profit              0
Shipping Cost       1
Order Priority      2
Customer ID         1
Customer Name       0
Segment             1
City                0
State               0
Country             0
Region              1
Months              0
dtype: int64
```

In [20]:

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split


# Using .loc for label-based indexing
column_name = 'Shipping Cost'
row_label = 535
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")

# Encode the 'Product' column
le = LabelEncoder()
df['Product'] = le.fit_transform(df['Product'])

# Create a validation set by setting known values to NaN
validation_indices = [1, 3]  # Indices of known values to set as NaN for validation
df_validation = df.copy()
df_validation.loc[validation_indices, 'Shipping Cost'] = np.nan

# Separate features and target for the imputer
features = df[['Product', 'Quantity', 'Discount']]
target = df[['Shipping Cost']]

# Combine features and target for the imputer
data_for_imputer = pd.concat([features, target], axis=1)

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Fit and transform the data
imputed_data = imputer.fit_transform(data_for_imputer)

# Update the dataframe with imputed values
df['Shipping Cost'] = imputed_data[:, -1]

# Create a new DataFrame with imputed values
df_imputed = df_validation.copy()
df_imputed['Shipping Cost'] = imputed_data[:, -1]


# Reverse the encoding for the 'Product' column
df['Product'] = le.inverse_transform(df['Product'])
df_imputed['Product'] = le.inverse_transform(df_imputed['Product'].astype(int))

# Convert the 'Product' column back to object type for both DataFrames
df['Product'] = df['Product'].astype(object)
df_imputed['Product'] = df_imputed['Product'].astype(object)

# Verify that 'Product' column is now correctly decoded
print("Data after converting 'Product' column to object:\n", df['Product'].head(5))
# print("Imputed DataFrame after converting 'Product' column to object:\n", df_impu
# Calculate the mean squared error and R-squared for the imputed values
actual_values = df.loc[validation_indices, 'Shipping Cost']
imputed_values = df_imputed.loc[validation_indices, 'Shipping Cost']
mse = mean_squared_error(actual_values, imputed_values)
r2 = r2_score(actual_values, imputed_values)

print("Accuracy of Model")
print(f'Mean Squared Error of the imputation: {mse}')
print(f'R-squared of the imputation: {r2}')



# Using .loc for label-based indexing
column_name = 'Shipping Cost'
row_label = 535
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")
```

```
Value at column 'Shipping Cost' and row 535: nan
Data after converting 'Product' column to object:
 0    Car Media Players
1         Car Speakers
2      Car Body Covers
3      Car & Bike Care
4                 Tyre
Name: Product, dtype: object
Accuracy of Model
Mean Squared Error of the imputation: 0.0
R-squared of the imputation: 1.0

Value at column 'Shipping Cost' and row 535: 15.0
```

In [21]:

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split


# Using .loc for label-based indexing
column_name = 'Quantity'
row_label = 95
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")

# Encode the 'Product' column
le = LabelEncoder()
df['Product'] = le.fit_transform(df['Product'])

# Create a validation set by setting known values to NaN
validation_indices = [1, 3]  # Indices of known values to set as NaN for validation
df_validation = df.copy()
df_validation.loc[validation_indices, 'Quantity'] = np.nan
```

```python
# Separate features and target for the imputer
features = df[['Product', 'Quantity', 'Discount']]
target = df[['Quantity']]

# Combine features and target for the imputer
data_for_imputer = pd.concat([features, target], axis=1)

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Fit and transform the data
imputed_data = imputer.fit_transform(data_for_imputer)

# Update the dataframe with imputed values
df['Quantity'] = imputed_data[:, -1]

# Create a new DataFrame with imputed values
df_imputed = df_validation.copy()
df_imputed['Quantity'] = imputed_data[:, -1]


# Reverse the encoding for the 'Product' column
df['Product'] = le.inverse_transform(df['Product'])
df_imputed['Product'] = le.inverse_transform(df_imputed['Product'].astype(int))

# Convert the 'Product' column back to object type for both DataFrames
df['Product'] = df['Product'].astype(object)
df_imputed['Product'] = df_imputed['Product'].astype(object)

# Verify that 'Product' column is now correctly decoded
print("Data after converting 'Product' column to object:\n", df['Product'].head(5))
# print("Imputed DataFrame after converting 'Product' column to object:\n", df_impu

# Calculate the mean squared error and R-squared for the imputed values
actual_values = df.loc[validation_indices, 'Quantity']
imputed_values = df_imputed.loc[validation_indices, 'Quantity']
mse = mean_squared_error(actual_values, imputed_values)
r2 = r2_score(actual_values, imputed_values)

print("Accuracy of Model")
print(f'Mean Squared Error of the imputation: {mse}')
print(f'R-squared of the imputation: {r2}')

# Using .loc for label-based indexing
column_name = 'Quantity'
row_label = 95
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")
```

```
Value at column 'Quantity' and row 95: nan
Data after converting 'Product' column to object:
 0    Car Media Players
1        Car Speakers
2      Car Body Covers
3      Car & Bike Care
4                 Tyre
Name: Product, dtype: object
Accuracy of Model
Mean Squared Error of the imputation: 0.0
R-squared of the imputation: 1.0

Value at column 'Quantity' and row 95: 3.6
```

In [22]:

```python
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split


# Using .loc for label-based indexing
column_name = 'Discount'
row_label = 211
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")

# Encode the 'Product' column
le = LabelEncoder()
df['Product'] = le.fit_transform(df['Product'])

# Create a validation set by setting known values to NaN
validation_indices = [1, 3]  # Indices of known values to set as NaN for validation
df_validation = df.copy()
df_validation.loc[validation_indices, 'Discount'] = np.nan

# Separate features and target for the imputer
features = df[['Product', 'Quantity', 'Shipping Cost']]
target = df[['Discount']]

# Combine features and target for the imputer
data_for_imputer = pd.concat([features, target], axis=1)

# Initialize KNNImputer
imputer = KNNImputer(n_neighbors=5)

# Fit and transform the data
imputed_data = imputer.fit_transform(data_for_imputer)

# Update the dataframe with imputed values
df['Discount'] = imputed_data[:, -1]

# Create a new DataFrame with imputed values
df_imputed = df_validation.copy()
df_imputed['Discount'] = imputed_data[:, -1]
```

```
# Reverse the encoding for the 'Product' column
df['Product'] = le.inverse_transform(df['Product'])
df_imputed['Product'] = le.inverse_transform(df_imputed['Product'].astype(int))

# Convert the 'Product' column back to object type for both DataFrames
df['Product'] = df['Product'].astype(object)
df_imputed['Product'] = df_imputed['Product'].astype(object)

# Verify that 'Product' column is now correctly decoded
print("Data after converting 'Product' column to object:\n", df['Product'].head(5))
# print("Imputed DataFrame after converting 'Product' column to object:\n", df_impu

# Calculate the mean squared error and R-squared for the imputed values
actual_values = df.loc[validation_indices, 'Discount']
imputed_values = df_imputed.loc[validation_indices, 'Discount']
mse = mean_squared_error(actual_values, imputed_values)
r2 = r2_score(actual_values, imputed_values)

print("Accuracy of Model")
print(f'Mean Squared Error of the imputation: {mse}')
print(f'R-squared of the imputation: {r2}')

# Using .loc for label-based indexing
column_name = 'Discount'
row_label = 211
value_loc = df.loc[row_label, column_name]
print(f"\nValue at column '{column_name}' and row {row_label}: {value_loc}")
```

```
Value at column 'Discount' and row 211: nan
Data after converting 'Product' column to object:
 0    Car Media Players
1          Car Speakers
2       Car Body Covers
3       Car & Bike Care
4                  Tyre
Name: Product, dtype: object
Accuracy of Model
Mean Squared Error of the imputation: 0.0
R-squared of the imputation: 1.0

Value at column 'Discount' and row 211: 0.03
```

In [23]:
```
df.isna().sum()
```

Out[23]:

```
Order ID            0
Order Date          1
Ship Date           0
Aging               1
Ship Mode           0
Product Category    0
Product             0
Sales               0
Quantity            0
Discount            0
Profit              0
Shipping Cost       0
Order Priority      2
Customer ID         1
Customer Name       0
Segment             1
City                0
State               0
Country             0
Region              1
Months              0
dtype: int64
```

In [24]:

```
df[df['Region'].isna()]
```

Out[24]:

	Order ID	Order Date	Ship Date	Aging	Ship Mode	Product Category	Product	Sales	Quantity	Discount
117	AU-2015-118	2015-08-16	2015-08-17	1.0	First Class	Auto & Accessories	Car Media Players	140.0	1.0	0.04

1 rows × 21 columns

In [25]:

```
df[df.isna().any(axis=1)]
```

Out[25]:

	Order ID	Order Date	Ship Date	Aging	Ship Mode	Product Category	Product	Sales	Quantity	Discount
27	AU-2015-28	2015-09-29	2015-10-05	NaN	First Class	Auto & Accessories	Car Media Players	140.0	1.0	0.03
100	AU-2015-101	NaT	2015-04-23	6.0	First Class	Auto & Accessories	Car Speakers	211.0	5.0	0.03
117	AU-2015-118	2015-08-16	2015-08-17	1.0	First Class	Auto & Accessories	Car Media Players	140.0	1.0	0.04
131	AU-2015-132	2015-08-12	2015-08-15	3.0	First Class	Auto & Accessories	Bike Tyres	72.0	5.0	0.05
370	AU-2015-371	2015-11-21	2015-11-25	4.0	First Class	Auto & Accessories	Car Speakers	211.0	4.0	0.02
625	AU-2015-626	2015-02-03	2015-02-05	2.0	First Class	Auto & Accessories	Tyre	250.0	4.0	0.03
791	AU-2015-792	2015-04-27	2015-05-01	4.0	First Class	Auto & Accessories	Car Pillow & Neck Rest	231.0	1.0	0.01

7 rows × 21 columns

In [26]:

```
df.loc[(df['Country'] == 'Italy') & (df['Region'].isna()), 'Region'] = 'South'
df.loc[117]
```

```
Out[26]: Order ID                      AU-2015-118
         Order Date            2015-08-16 00:00:00
         Ship Date             2015-08-17 00:00:00
         Aging                                 1.0
         Ship Mode                     First Class
         Product Category        Auto & Accessories
         Product                  Car Media Players
         Sales                               140.0
         Quantity                              1.0
         Discount                             0.04
         Profit                               54.4
         Shipping Cost                         5.4
         Order Priority                       High
         Customer ID                      RI-00118
         Customer Name             Ayala Molinari
         Segment                          Consumer
         City                                Turin
         State                            Piedmont
         Country                             Italy
         Region                              South
         Months                                Aug
         Name: 117, dtype: object
```

```
In [27]: df[df['Aging'].isna()]
```

Out[27]:

	Order ID	Order Date	Ship Date	Aging	Ship Mode	Product Category	Product	Sales	Quantity	Discount	..
27	AU-2015-28	2015-09-29	2015-10-05	NaN	First Class	Auto & Accessories	Car Media Players	140.0	1.0	0.03	.

1 rows × 21 columns

```
In [28]: df.loc[(df['Aging'].isna()), 'Aging'] = (df['Ship Date'] - df['Order Date']).dt.day
         print(df.loc[27])
```

```
Order ID                      AU-2015-28
Order Date           2015-09-29 00:00:00
Ship Date            2015-10-05 00:00:00
Aging                                6.0
Ship Mode                    First Class
Product Category      Auto & Accessories
Product                Car Media Players
Sales                              140.0
Quantity                             1.0
Discount                            0.03
Profit                              55.8
Shipping Cost                        5.6
Order Priority                      High
Customer ID                      NG-0028
Customer Name           Harris Armstrong
Segment                        Corporate
City                               Jinan
State                           Shandong
Country                            China
Region                        North Asia
Months                               Sep
Name: 27, dtype: object
```

In [29]:

```
df.loc[27, 'Aging'] = np.nan
```

In [30]:

```
df.loc[(df['Order Date'].isna()), 'Order Date'] = df['Ship Date'] - pd.to_timedelta
df.loc[100]
```

Out[30]:

```
Order ID                     AU-2015-101
Order Date           2015-04-17 00:00:00
Ship Date            2015-04-23 00:00:00
Aging                                6.0
Ship Mode                    First Class
Product Category      Auto & Accessories
Product                     Car Speakers
Sales                              211.0
Quantity                             5.0
Discount                            0.03
Profit                              99.4
Shipping Cost                        9.9
Order Priority                  Critical
Customer ID                     RN-00101
Customer Name                  Cook Bern
Segment                         Consumer
City                           Amsterdam
State                      North Holland
Country                      Netherlands
Region                           Central
Months                               Apr
Name: 100, dtype: object
```

In [ ]: