Assessment 1

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as plt
from matplotlib import pyplot
```

In [2]:

```
gb_cleaning = pd.read_csv('GBmatrixc.csv')
sb_cleaning = pd.read_csv('SBmatrixc.csv')
```

Question 1 - Data Cleaning

1. For GBMatrixc dataset

1.1 Checking unnamed column

We want to know th type of bed, so I add a new column called type. Besides, we also want to know the total quantity of each part.

In [3]:

```
# Create new column 'type' for GB and SB.
gb_cleaning['type'] = 'Guest_beds'
sb_cleaning['type'] = 'Single_beds'
```

In [4]:

```
gb_cleaning.head()
```

Out[4]:

	Unnamed: 0	BRIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24	TARVA_119_
0	100001	0	1.0	1.0	1.0	1.0
1	100027	0	NaN	NaN	NaN	1.0
2	100049	1	1.0	NaN	NaN	NaN
3	100089	0	NaN	NaN	NaN	1.0
4	100211	0	NaN	NaN	NaN	8.0

In [5]:

```
gb_cleaning.tail()
```

Out[5]:

	Unnamed: 0	BRIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24	T
73	100602/117056	2	NaN	NaN	NaN	
74	110519/118149	8	8.0	NaN	NaN	
75	118331/112996	22	24.0	NaN	40.0	
76	119030/118224/117434	22	8.0	NaN	NaN	
77	124328/128763	2	NaN	NaN	NaN	

- Rename column 'Unnamed: 0' as 'Part_No' because it shows the part codes for each component.
- For column 'Unamed:9', I assume to rename this column as'GB_unamed_part' because all its value is 1, meaning this part might be important that will be used in all part number.

In [6]:

```
# Rename columns
gb_cleaning.rename(columns = {'Unnamed: 0':'Part_No', 'Unnamed: 9':'GB_unnamed_part'
```

In [7]:

```
gb_cleaning.columns
```

Out[7]:

```
Index(['Part_No', 'BRIMNES_329_22', 'FLEKKE_399_', 'FYRESDAL_299_',
       'HEMNES_409_24', 'TARVA_119_', 'UT_ER_299_', 'UT_ER_299_.1',
       'UT_ER_299_.2', 'GB_unnamed_part', 'type'],
      dtype='object')
```

1.2 Change data type

In [8]:

```
gb_cleaning.info() # Find that 'UT_ER_299_ ' is object data type.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Part_No          78 non-null     object 
 1   BRIMNES_329_22   78 non-null     int64  
 2   FLEKKE_399_      28 non-null     float64
 3   FYRESDAL_299_    9 non-null      float64
 4   HEMNES_409_24    20 non-null     float64
 5   TARVA_119_       24 non-null     float64
 6   UT_ER_299_       66 non-null     object 
 7   UT_ER_299_.1     0 non-null      float64
 8   UT_ER_299_.2     0 non-null      float64
 9   GB_unnamed_part  78 non-null     int64  
 10  type             78 non-null     object 
dtypes: float64(6), int64(2), object(3)
memory usage: 6.8+ KB
```

In [9]:

```
# Transform dtype to float64, for 'BRIMNES_329_22','UT_ER_299_ ','Unnamed: 9'columns
gb_cleaning['BRIMNES_329_22'] = gb_cleaning['BRIMNES_329_22'].astype('float')
gb_cleaning['UT_ER_299_'] = gb_cleaning['UT_ER_299_'].astype('float')
gb_cleaning['GB_unnamed_part'] = gb_cleaning['GB_unnamed_part'].astype('float')
```

In [10]:

```
gb_cleaning.dtypes
```

Out[10]:

```
Part_No             object
BRIMNES_329_22     float64
FLEKKE_399_        float64
FYRESDAL_299_      float64
HEMNES_409_24      float64
TARVA_119_         float64
UT_ER_299_         float64
UT_ER_299_.1       float64
UT_ER_299_.2       float64
GB_unnamed_part    float64
type                object
dtype: object
```

In [11]:

```
gb_cleaning.describe()
```

Out[11]:

	BRIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24	TARVA_119_	UT_ER_
count	78.000000	28.000000	9.000000	20.000000	24.000000	6.00
mean	3.487179	10.071429	8.444444	15.150000	8.458333	7.83
std	8.687569	12.498677	12.319812	15.435008	10.910622	4.66
min	0.000000	1.000000	1.000000	1.000000	1.000000	1.00
25%	0.000000	2.000000	2.000000	3.000000	2.000000	5.00
50%	0.000000	6.500000	2.000000	6.000000	6.000000	9.00
75%	2.000000	10.000000	8.000000	26.000000	8.250000	10.00
max	54.000000	58.000000	37.000000	50.000000	46.000000	14.00

In [12]:

```
# There are 78 rows and 11 columns in GB dataset.
gb_cleaning.shape
```

Out[12]:

```
(78, 11)
```

1.3 Check duplicates

In [13]:

```
# For GBmatrixc: Check duplicates from columns
gb_cleaning.columns
```

Out[13]:

```
Index(['Part_No', 'BRIMNES_329_22', 'FLEKKE_399_', 'FYRESDAL_299_',
       'HEMNES_409_24', 'TARVA_119_', 'UT_ER_299_', 'UT_ER_299_.1',
       'UT_ER_299_.2', 'GB_unnamed_part', 'type'],
      dtype='object')
```

In [14]:

```
# Checking for duplicates in columns using the 'duplicated' method
gb_cleaning.index.duplicated(keep=False)

# check row duplicated. As all values are False, meaning no duplicated rows.
```

Out[14]:

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False])
```

In [15]:

```
gb_cleaning['Part_No'].duplicated().sum()  # there is no duplicated parts.
```

Out[15]:

```
0
```

In [16]:

```
gb_cleaning.index.duplicated().sum()  # there is no duplicated index for columns
```

Out[16]:

```
0
```

There are no duplicated rows and index in GBMatrixc dataset, so don't need to drop duplicates.

1.4 Check missing data and fill NA

In [17]:

```
# GBmatrixc: #Fill 0 into NA
gb_cleaning = gb_cleaning.fillna(0)
```

In [18]:

```
gb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78 entries, 0 to 77
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Part_No          78 non-null     object 
 1   BRIMNES_329_22   78 non-null     float64
 2   FLEKKE_399_      78 non-null     float64
 3   FYRESDAL_299_    78 non-null     float64
 4   HEMNES_409_24    78 non-null     float64
 5   TARVA_119_       78 non-null     float64
 6   UT_ER_299_       78 non-null     float64
 7   UT_ER_299_.1     78 non-null     float64
 8   UT_ER_299_.2     78 non-null     float64
 9   GB_unnamed_part  78 non-null     float64
 10  type             78 non-null     object 
dtypes: float64(9), object(2)
memory usage: 6.8+ KB
```

In [19]:

```
gb_cleaning
```

Out[19]:

	Part_No	BRIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24
0	100001	0.0	1.0	1.0	1.0
1	100027	0.0	0.0	0.0	0.0
2	100049	1.0	1.0	0.0	0.0
3	100089	0.0	0.0	0.0	0.0
4	100211	0.0	0.0	0.0	0.0
...	...	...	...	...	...
73	100602/117056	2.0	0.0	0.0	0.0
74	110519/118149	8.0	8.0	0.0	0.0
75	118331/112996	22.0	24.0	0.0	40.0
76	119030/118224/117434	22.0	8.0	0.0	0.0
77	124328/128763	2.0	0.0	0.0	0.0

78 rows × 11 columns

1.5 Split multiple part codes

In the multiple part codes, I assume that each unique 'Part No' need the same amount of quantities. This assumption is to avoid the lack of inventory for the multiple part.

In [20]:

```
# Check unique values of 'Part No' columns
gb_cleaning['Part_No'].unique()
```

Out[20]:

```
array(['100001', '100027', '100049', '100089', '100211', '100347',
       '100359', '100365', '100372', '100514', '101345', '101350',
       '101352', '_101359', '_101367', '_102138', '_103114', '_10343
0',
       '_104012', '_104875', '_105163', '_105236', '_105307', '10533
0',
       '106698', '107091', '108458', '108490', '108903', '109048',
       '109049', '109060', '109542', '109567', '110525', '110630',
       '110646', '111401', '111451', '111631', '112975', '112977',
       '113453', '114670', '114671', '115348', '115349', '116894',
       '117229', '117494', '117615', '118231', '118301', '119976',
       '120030', '122628', '122898', '123660', '128605', '128780',
       '128874', '130485', '130787', '131182', '139430', '139505',
       '146767', '146865', '146967', '151641', '152746', '153552',
       '100215/105251', '100602/117056', '110519/118149', '118331/1129
96',
       '119030/118224/117434', '124328/128763'], dtype=object)
```

In [21]:

```
# create a new column 'gb_part' that split '/'
gb_part = gb_cleaning['Part_No'].str.split('/', expand = True)
gb_part
```

Out[21]:

	0	1	2
0	100001	None	None
1	100027	None	None
2	100049	None	None
3	100089	None	None
4	100211	None	None
...	...	...	...
73	100602	117056	None
74	110519	118149	None
75	118331	112996	None
76	119030	118224	117434
77	124328	128763	None

78 rows × 3 columns

In [22]:

```
# Stack splited parts into one new column
gb_part = gb_part.stack()
gb_part
```

Out[22]:

```
0   0    100001
1   0    100027
2   0    100049
3   0    100089
4   0    100211
           ...
76  0    119030
    1    118224
    2    117434
77  0    124328
    1    128763
Length: 85, dtype: object
```

In [23]:

```
# Reset index
gb_part = gb_part.reset_index(level = 1, drop=True)
gb_part
```

Out[23]:

```
0      100001
1      100027
2      100049
3      100089
4      100211
        ...
76     119030
76     118224
76     117434
77     124328
77     128763
Length: 85, dtype: object
```

In [24]:

```
# Modify the dataframe and rename the new column as a new 'Part No'
gb_part = gb_part.to_frame().rename({0:'Part_No'},axis =1)
gb_part
```

Out[24]:

	Part_No
0	100001
1	100027
2	100049
3	100089
4	100211
...	...
76	119030
76	118224
76	117434
77	124328
77	128763

85 rows × 1 columns

In [25]:

```
# Join a new gb_cleaning table.
# Remove old 'Part No' column in gb_cleaning and join new gb_part which has new 'Par
gb_cleaning = gb_cleaning.drop(['Part_No'],axis = 1).join(gb_part)
```

In [26]:

```
gb_cleaning
```

]:

RIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24	TARVA_119_	UT_ER_299_	UT_ER_299_.1	UT_ER
0.0	1.0	1.0	1.0	1.0	0.0	0.0	
0.0	0.0	0.0	0.0	1.0	0.0	0.0	
1.0	1.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	1.0	0.0	0.0	
0.0	0.0	0.0	0.0	8.0	0.0	0.0	
...	...	...	...	...	...	...	
22.0	8.0	0.0	0.0	0.0	0.0	0.0	
22.0	8.0	0.0	0.0	0.0	0.0	0.0	
22.0	8.0	0.0	0.0	0.0	0.0	0.0	
2.0	0.0	0.0	0.0	0.0	0.0	0.0	

1.6 Replace punctuation

In [27]:

```
# Check the values of 'Part No', I find that there are punctuation.
gb_cleaning['Part_No'].unique()
```

Out[27]:

```
array(['100001', '100027', '100049', '100089', '100211', '100347',
       '100359', '100365', '100372', '100514', '101345', '101350',
       '101352', '_101359', '_101367', '_102138', '_103114', '_10343
0',
       '_104012', '_104875', '_105163', '_105236', '_105307', '10533
0',
       '106698', '107091', '108458', '108490', '108903', '109048',
       '109049', '109060', '109542', '109567', '110525', '110630',
       '110646', '111401', '111451', '111631', '112975', '112977',
       '113453', '114670', '114671', '115348', '115349', '116894',
       '117229', '117494', '117615', '118231', '118301', '119976',
       '120030', '122628', '122898', '123660', '128605', '128780',
       '128874', '130485', '130787', '131182', '139430', '139505',
       '146767', '146865', '146967', '151641', '152746', '153552',
       '100215', '105251', '100602', '117056', '110519', '118149',
       '118331', '112996', '119030', '118224', '117434', '124328',
       '128763'], dtype=object)
```

In [28]:

```
# Replace punctuation to '', and delete all space of Part_No.
gb_cleaning['Part_No']= gb_cleaning['Part_No'].str.replace(r'_', '').str.strip()
```

In [29]:

```
gb_cleaning['Part_No'].unique()
```

Out[29]:

```
array(['100001', '100027', '100049', '100089', '100211', '100347',
       '100359', '100365', '100372', '100514', '101345', '101350',
       '101352', '101359', '101367', '102138', '103114', '103430',
       '104012', '104875', '105163', '105236', '105307', '105330',
       '106698', '107091', '108458', '108490', '108903', '109048',
       '109049', '109060', '109542', '109567', '110525', '110630',
       '110646', '111401', '111451', '111631', '112975', '112977',
       '113453', '114670', '114671', '115348', '115349', '116894',
       '117229', '117494', '117615', '118231', '118301', '119976',
       '120030', '122628', '122898', '123660', '128605', '128780',
       '128874', '130485', '130787', '131182', '139430', '139505',
       '146767', '146865', '146967', '151641', '152746', '153552',
       '100215', '105251', '100602', '117056', '110519', '118149',
       '118331', '112996', '119030', '118224', '117434', '124328',
       '128763'], dtype=object)
```

1.7 Change columns

In [30]:

```
gb_cleaning.columns
```

Out[30]:

```
Index(['BRIMNES_329_22', 'FLEKKE_399_', 'FYRESDAL_299_', 'HEMNES_409_2
4',
       'TARVA_119_', 'UT_ER_299_', 'UT_ER_299_.1', 'UT_ER_299_.2',
       'GB_unnamed_part', 'type', 'Part_No'],
      dtype='object')
```

In [31]:

```
# Create new column 'quantity' for GB and SB, calculating the total quantity of each
gb_cleaning['gbquantity'] = np.sum(gb_cleaning, axis = 1)
sb_cleaning['sbquantity'] = np.sum(sb_cleaning, axis = 1)
```

```
/Users/lisahu/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fro
mnumeric.py:85: FutureWarning: Dropping of nuisance columns in DataFra
me reductions (with 'numeric_only=None') is deprecated; in a future ve
rsion this will raise TypeError.  Select only valid columns before cal
ling the reduction.
  return reduction(axis=axis, out=out, **passkwargs)
```

In [32]:

```
# Combine columns which start as the same product names
gb_cleaning['BRIMNES']  = gb_cleaning[list(gb_cleaning.filter(regex='BRIMNES'))].sum
gb_cleaning['FLEKKE'] = gb_cleaning[list(gb_cleaning.filter(regex='FLEKKE'))].sum(ax
gb_cleaning['FYRESDAL'] = gb_cleaning[list(gb_cleaning.filter(regex='FYRESDAL'))].su
gb_cleaning['HEMNES'] = gb_cleaning[list(gb_cleaning.filter(regex='HEMNES'))].sum(ax
gb_cleaning['TARVA'] = gb_cleaning[list(gb_cleaning.filter(regex='TARVA'))].sum(axis
gb_cleaning['UT'] = gb_cleaning[list(gb_cleaning.filter(regex='UT'))].sum(axis=1)
```

In [33]:

```
gb_cleaning.head()
```

Out[33]:

```
RVA_119_  UT_ER_299_  UT_ER_299_.1  UT_ER_299_.2  GB_unnamed_part        type  Part_No  gb
     1.0         0.0           0.0           0.0              1.0  Guest_beds   100001
     1.0         0.0           0.0           0.0              1.0  Guest_beds   100027
     0.0         0.0           0.0           0.0              1.0  Guest_beds   100049
     1.0         0.0           0.0           0.0              1.0  Guest_beds   100089
     8.0         0.0           0.0           0.0              1.0  Guest_beds   100211
```

In [34]:

```
# Set Part number as index.

gb_cleaning.set_index('Part_No', inplace = False)

# In the last column 'Unnamed:9', I would keep this column because it has values 1.
```

Out[34]:

```
         BRIMNES_329_22  FLEKKE_399_  FYRESDAL_299_  HEMNES_409_24  TARVA_119_  UT_E
Part_No
 100001             0.0          1.0            1.0            1.0         1.0
 100027             0.0          0.0            0.0            0.0         1.0
 100049             1.0          1.0            0.0            0.0         0.0
 100089             0.0          0.0            0.0            0.0         1.0
 100211             0.0          0.0            0.0            0.0         8.0
    ...             ...          ...            ...            ...         ...
 119030            22.0          8.0            0.0            0.0         0.0
 118224            22.0          8.0            0.0            0.0         0.0
 117434            22.0          8.0            0.0            0.0         0.0
 124328             2.0          0.0            0.0            0.0         0.0
 128763             2.0          0.0            0.0            0.0         0.0

85 rows × 17 columns
```

In [35]:

```
gb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 85 entries, 0 to 77
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   BRIMNES_329_22    85 non-null     float64
 1   FLEKKE_399_       85 non-null     float64
 2   FYRESDAL_299_     85 non-null     float64
 3   HEMNES_409_24     85 non-null     float64
 4   TARVA_119_        85 non-null     float64
 5   UT_ER_299_        85 non-null     float64
 6   UT_ER_299_.1      85 non-null     float64
 7   UT_ER_299_.2      85 non-null     float64
 8   GB_unnamed_part   85 non-null     float64
 9   type              85 non-null     object 
 10  Part_No           85 non-null     object 
 11  gbquantity        85 non-null     float64
 12  BRIMNES           85 non-null     float64
 13  FLEKKE            85 non-null     float64
 14  FYRESDAL          85 non-null     float64
 15  HEMNES            85 non-null     float64
 16  TARVA             85 non-null     float64
 17  UT                85 non-null     float64
dtypes: float64(16), object(2)
memory usage: 12.6+ KB
```

1.8 Outliers

In [36]:

```
# Detect outliers
for column in gb_cleaning.columns:
    print(column)
    print(gb_cleaning[column].value_counts())
    print('')
```

```
BRIMNES_329_22
0.0     50
2.0      9
4.0      5
22.0     5
6.0      4
1.0      3
32.0     2
8.0      2
12.0     1
10.0     1
16.0     1
3.0      1
54.0     1
Name: BRIMNES_329_22, dtype: int64

FLEKKE_399_
0.0     52
8.0     10
```

Find outliers using z-scores

Calculating mean and standard deviation for each column.

In [37]:

```
# Calculate mean & std for quantity

mean = np.mean(gb_cleaning.gbquantity)
std = np.std(gb_cleaning.gbquantity)
print('mean of guest beds\' parts quantity is ', mean)
print('std. deviation of guest beds\' is ', std)



# Calculate Z score. If Z score > 3, print it as an outlier

threshold = 3
outlier =[]
for value in gb_cleaning.gbquantity:
    z = (value - mean)/std
    if z > threshold:
        outlier.append(value)
print('outlier in guest beds quantity is ', outlier)
```

```
mean of guest beds' parts quantity is  16.905882352941177
std. deviation of guest beds' is  23.18854865935631
outlier in guest beds quantity is  [109.0, 105.0, 87.0, 87.0]
```

In [38]:

```
# Apply function in each row.

def is_outlier(value, mean=1, std=1, threshold=1):
    threshold=3
    z = (value - mean)/std
    return z > threshold

gb_cleaning[gb_cleaning['gbquantity'].apply(is_outlier, mean=mean, std=std, threshol
```

Out[38]:

	BRIMNES_329_22	FLEKKE_399_	FYRESDAL_299_	HEMNES_409_24	TARVA_119_	UT_ER_29
34	32.0	30.0	0.0	32.0	14.0	0.
47	54.0	0.0	0.0	50.0	0.0	0.
75	22.0	24.0	0.0	40.0	0.0	0.
75	22.0	24.0	0.0	40.0	0.0	0.

From the table above, it shows there are four outliers in 'quantity'. Quantity refers to the total quantity that is used in each part of guest beds. I assume that these four components are highly frequently used parts and thus key parts. So I will keep these outliers and consider them as key parts of guest beds.

In [39]:

```
gb_cleaning.to_csv('GB_cleaning.csv')
```

In [ ]:

```

```

- Data Cleaning

2. For SBMatrixc dataset

2.1 Checking unnamed column

In [40]:

```
# For column 'Unnamed: 0', I want to change its name as 'Part_No' because it shows
sb_cleaning.rename(columns = {'Unnamed: 0':'Part_No'}, inplace=True)
```

In [41]:

```
sb_cleaning.columns
```

Out[41]:

```
Index(['Part_No', 'FJELLSE_45_', 'HEMNES_150_', 'HEMNES_220_31', 'MALM
_125_36',
       'MALM_139_30', 'NORDLI_189_', 'TARVA_75_', 'Nan', 'type', 'sbqu
antity'],
      dtype='object')
```

In [42]:

```
sb_cleaning.head()
```

Out[42]:

	Part_No	FJELLSE_45_	HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORD
0	100001	1	Nan	NaN	NaN	NaN	
1	100006	0	Nan	NaN	NaN	1.0	
2	100049	0	Nan	NaN	1.0	1.0	
3	100087	0	4	4.0	NaN	NaN	
4	100089	0	1	1.0	NaN	NaN	

In [43]:

```
# Combine columns which start as the same product names
sb_cleaning['FJELLSE']  = sb_cleaning[list(sb_cleaning.filter(regex='FJELLSE'))].sum
sb_cleaning['HEMNES']  = sb_cleaning[list(sb_cleaning.filter(regex='HEMNES'))].sum(a
sb_cleaning['MALM']  = sb_cleaning[list(sb_cleaning.filter(regex='MALM'))].sum(axis=
sb_cleaning['NORDLI']  = sb_cleaning[list(sb_cleaning.filter(regex='NORDLI'))].sum(a
sb_cleaning['TARVA']  = sb_cleaning[list(sb_cleaning.filter(regex='TARVA'))].sum(axi
```

```
/var/folders/t5/vfwg7gv55v9_n8vzkxs4bf900000gn/T/ipykernel_908/2919582
642.py:3: FutureWarning: Dropping of nuisance columns in DataFrame red
uctions (with 'numeric_only=None') is deprecated; in a future version
this will raise TypeError.  Select only valid columns before calling t
he reduction.
  sb_cleaning['HEMNES']  = sb_cleaning[list(sb_cleaning.filter(regex
='HEMNES'))].sum(axis=1)
```

2.2 Change data type

In [44]:

```
sb_cleaning.dtypes
```

Out[44]:

```
Part_No           object
FJELLSE_45_        int64
HEMNES_150_       object
HEMNES_220_31    float64
MALM_125_36      float64
MALM_139_30      float64
NORDLI_189_      float64
TARVA_75_        float64
Nan               object
type              object
sbquantity       float64
FJELLSE            int64
HEMNES           float64
MALM             float64
NORDLI           float64
TARVA            float64
dtype: object
```

Investigating data

In [45]:

```
sb_cleaning.describe()
```

Out[45]:

	FJELLSE_45_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_
count	50.000000	19.000000	14.000000	21.000000	8.00000	10.000000
mean	0.820000	11.000000	5.357143	7.523810	9.75000	9.400000
std	2.939457	10.801234	4.271680	5.793264	4.16619	7.933053
min	0.000000	1.000000	1.000000	1.000000	3.00000	1.000000
25%	0.000000	4.000000	3.000000	3.000000	9.75000	4.000000
50%	0.000000	5.000000	4.000000	5.000000	12.00000	6.500000
75%	0.000000	17.000000	7.250000	12.000000	12.00000	14.000000
max	18.000000	40.000000	16.000000	18.000000	12.00000	26.000000

In [46]:

```
sb_cleaning.info()  #find there is a "Nan" column key with all 'Nan' values, which
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Part_No         50 non-null     object 
 1   FJELLSE_45_     50 non-null     int64  
 2   HEMNES_150_     50 non-null     object 
 3   HEMNES_220_31   19 non-null     float64
 4   MALM_125_36     14 non-null     float64
 5   MALM_139_30     21 non-null     float64
 6   NORDLI_189_     8 non-null      float64
 7   TARVA_75_       10 non-null     float64
 8   Nan             50 non-null     object 
 9   type            50 non-null     object 
 10  sbquantity      50 non-null     float64
 11  FJELLSE         50 non-null     int64  
 12  HEMNES          50 non-null     float64
 13  MALM            50 non-null     float64
 14  NORDLI          50 non-null     float64
 15  TARVA           50 non-null     float64
dtypes: float64(10), int64(2), object(4)
memory usage: 6.4+ KB
```

In [47]:

```
# There are 50 rows and 10 columns in SB dataset.
sb_cleaning.shape
```

Out[47]:

```
(50, 16)
```

2.3 Check duplicates

In [48]:

```
# For SBmatrixc: check duplicates from columns
sb_cleaning.columns
```

Out[48]:

```
Index(['Part_No', 'FJELLSE_45_', 'HEMNES_150_', 'HEMNES_220_31', 'MALM
_125_36',
       'MALM_139_30', 'NORDLI_189_', 'TARVA_75_', 'Nan', 'type', 'sbqu
antity',
       'FJELLSE', 'HEMNES', 'MALM', 'NORDLI', 'TARVA'],
      dtype='object')
```

In [49]:

```
# Check row duplicated. As all values are False, meaning no duplicated rows.
sb_cleaning.index.duplicated(keep=False)
```

Out[49]:

```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False])
```

In [50]:

```
# Calculate the toal quantity of dulicates for rows and columns
sb_cleaning.duplicated().sum()  # No duplicates in rows
```

Out[50]:

```
0
```

In [51]:

```
sb_cleaning.index.duplicated().sum()  # No duplicates in columns
```

Out[51]:

```
0
```

As there are no duplicates in SBmatrixc dataset, we don't need to drop duplicates.

2.4 Check missing data and fill NA

In [52]:

```
# Fill 0 into NA
sb_cleaning = sb_cleaning.fillna(0)
```

In [53]:

```
sb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Part_No         50 non-null     object 
 1   FJELLSE_45_     50 non-null     int64  
 2   HEMNES_150_     50 non-null     object 
 3   HEMNES_220_31   50 non-null     float64
 4   MALM_125_36     50 non-null     float64
 5   MALM_139_30     50 non-null     float64
 6   NORDLI_189_     50 non-null     float64
 7   TARVA_75_       50 non-null     float64
 8   Nan             50 non-null     object 
 9   type            50 non-null     object 
 10  sbquantity      50 non-null     float64
 11  FJELLSE         50 non-null     int64  
 12  HEMNES          50 non-null     float64
 13  MALM            50 non-null     float64
 14  NORDLI          50 non-null     float64
 15  TARVA           50 non-null     float64
dtypes: float64(10), int64(2), object(4)
memory usage: 6.4+ KB
```

In [54]:

```
# Because there is a "Nan" column, check unique values of column "Nan"
Nan = sb_cleaning.drop_duplicates(['Nan']) # In column "Nan", all values are "Nan"
Nan
```

Out[54]:

INES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	Nan	
Nan	0.0	0.0	0.0	0.0	1.0	Nan	Single

In [55]:

```
# Since there are no values in "Nan" column, delete column "Nan"
sb_cleaning = sb_cleaning.drop(columns = ['Nan'])
sb_cleaning
```

Out[55]:

HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	
Nan	0.0	0.0	0.0	0.0	1.0	Single
Nan	0.0	0.0	1.0	0.0	0.0	Single
Nan	0.0	1.0	1.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
1	1.0	0.0	0.0	0.0	0.0	Single
1	1.0	0.0	0.0	0.0	0.0	Single
Nan	0.0	0.0	18.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	12.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	14.0	Single
Nan	40.0	0.0	8.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	26.0	Single
18	18.0	0.0	0.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	0.0	Single
12	12.0	12.0	12.0	0.0	0.0	Single
Nan	0.0	3.0	3.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	12.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	0.0	Single
4	4.0	4.0	4.0	0.0	4.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
4	4.0	4.0	4.0	0.0	4.0	Single
Nan	0.0	0.0	0.0	0.0	16.0	Single
Nan	0.0	0.0	0.0	0.0	14.0	Single
Nan	0.0	0.0	0.0	0.0	0.0	Single
Nan	20.0	0.0	0.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	12.0	0.0	Single
Nan	22.0	0.0	16.0	0.0	0.0	Single
16	16.0	16.0	16.0	0.0	0.0	Single
Nan	0.0	0.0	8.0	0.0	0.0	Single
Nan	10.0	0.0	0.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
Nan	0.0	1.0	1.0	0.0	0.0	Single

HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	
Nan	0.0	3.0	3.0	0.0	0.0	Single
Nan	0.0	3.0	3.0	0.0	0.0	Single
Nan	0.0	8.0	8.0	0.0	0.0	Single
2	2.0	0.0	0.0	0.0	2.0	Single
5	5.0	5.0	5.0	0.0	5.0	Single
Nan	0.0	0.0	0.0	12.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	0.0	Single
Nan	0.0	4.0	4.0	0.0	0.0	Single
Nan	0.0	3.0	3.0	0.0	0.0	Single
Nan	0.0	8.0	8.0	0.0	0.0	Single
Nan	0.0	0.0	16.0	0.0	0.0	Single
Nan	0.0	0.0	16.0	0.0	0.0	Single
Nan	0.0	0.0	0.0	0.0	8.0	Single
Nan	0.0	0.0	0.0	3.0	0.0	Single
Nan	0.0	0.0	0.0	3.0	0.0	Single
Nan	0.0	0.0	0.0	12.0	0.0	Single
8	8.0	0.0	0.0	0.0	0.0	Single
8	30.0	0.0	0.0	12.0	0.0	Single

In [56]:

```
sb_cleaning.shape
```

Out[56]:

```
(50, 15)
```

In [57]:

```
# From the table, find there are "Nan", replace "Nan" to 0.
sb_cleaning = sb_cleaning.replace('Nan', 0)
sb_cleaning
```

Out[57]:

HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	
0	0.0	0.0	0.0	0.0	1.0	Single
0	0.0	0.0	1.0	0.0	0.0	Single
0	0.0	1.0	1.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
1	1.0	0.0	0.0	0.0	0.0	Single
1	1.0	0.0	0.0	0.0	0.0	Single
0	0.0	0.0	18.0	0.0	0.0	Single
0	0.0	0.0	0.0	12.0	0.0	Single
0	0.0	0.0	0.0	0.0	14.0	Single
0	40.0	0.0	8.0	0.0	0.0	Single
0	0.0	0.0	0.0	0.0	26.0	Single
18	18.0	0.0	0.0	0.0	0.0	Single
0	0.0	0.0	0.0	0.0	0.0	Single
12	12.0	12.0	12.0	0.0	0.0	Single
0	0.0	3.0	3.0	0.0	0.0	Single
0	0.0	0.0	0.0	12.0	0.0	Single
0	0.0	0.0	0.0	0.0	0.0	Single
4	4.0	4.0	4.0	0.0	4.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
4	4.0	4.0	4.0	0.0	4.0	Single
0	0.0	0.0	0.0	0.0	16.0	Single
0	0.0	0.0	0.0	0.0	14.0	Single
0	0.0	0.0	0.0	0.0	0.0	Single
0	20.0	0.0	0.0	0.0	0.0	Single
0	0.0	0.0	0.0	12.0	0.0	Single
0	22.0	0.0	16.0	0.0	0.0	Single
16	16.0	16.0	16.0	0.0	0.0	Single
0	0.0	0.0	8.0	0.0	0.0	Single
0	10.0	0.0	0.0	0.0	0.0	Single
4	4.0	0.0	0.0	0.0	0.0	Single
0	0.0	1.0	1.0	0.0	0.0	Single

HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	
0	0.0	3.0	3.0	0.0	0.0	Single
0	0.0	3.0	3.0	0.0	0.0	Single
0	0.0	8.0	8.0	0.0	0.0	Single
2	2.0	0.0	0.0	0.0	2.0	Single
5	5.0	5.0	5.0	0.0	5.0	Single
0	0.0	0.0	0.0	12.0	0.0	Single
0	0.0	0.0	0.0	0.0	0.0	Single
0	0.0	4.0	4.0	0.0	0.0	Single
0	0.0	3.0	3.0	0.0	0.0	Single
0	0.0	8.0	8.0	0.0	0.0	Single
0	0.0	0.0	16.0	0.0	0.0	Single
0	0.0	0.0	16.0	0.0	0.0	Single
0	0.0	0.0	0.0	0.0	8.0	Single
0	0.0	0.0	0.0	3.0	0.0	Single
0	0.0	0.0	0.0	3.0	0.0	Single
0	0.0	0.0	0.0	12.0	0.0	Single
8	8.0	0.0	0.0	0.0	0.0	Single
8	30.0	0.0	0.0	12.0	0.0	Single

2.5 Split multiple part codes

In the multiple part codes, I assume that each unique Part_No need the same amount of quantities. This assumption is to avoid the lack of inventory for those overlaopped components. So sufficient inventory level for these parts should be considered as priority.

In [58]:

```
# Check unique values of 'Part_No' columns
sb_cleaning['Part_No'].unique()
```

Out[58]:

```
array(['100001', '100006', '100049', '100087', '100089', '100092',
       '100224', '100349', '100514', '101345', '101350', '101352',
       '101357', '101359', '101367', '101372', '101385', '102267',
       '102335', '104875', '105163', '105307', '105330', '106569',
       '109041', '110519', '110630', '110789', '_111401', '_111402',
       '_111451', '_113453', '_114254', '_114334', '_114670', '_11722
8',
       '*117327', '*119030', '*121214', '*122628', '122998', '123491',
       '123492', '123502', '128780', '139163', '139164', '139251',
       '113434/122332', '118331/112996'], dtype=object)
```

From the columns name above, we can see that there are '/' that need to be splitted.

In [59]:

```
sb_part = sb_cleaning['Part_No'].str.split('/', expand=True)
sb_part
```

Out[59]:

	0	1
0	100001	None
1	100006	None
2	100049	None
3	100087	None
4	100089	None
5	100092	None
6	100224	None
7	100349	None
8	100514	None
9	101345	None
10	101350	None
11	101352	None
12	101357	None
13	101359	None
14	101367	None
15	101372	None
16	101385	None
17	102267	None
18	102335	None
19	104875	None
20	105163	None
21	105307	None
22	105330	None
23	106569	None
24	109041	None
25	110519	None
26	110630	None
27	110789	None
28	_111401	None
29	_111402	None
30	_111451	None
31	_113453	None
32	_114254	None

	0	1
33	_114334	None
34	_114670	None
35	_117228	None
36	*117327	None
37	*119030	None
38	*121214	None
39	*122628	None
40	122998	None
41	123491	None
42	123492	None
43	123502	None
44	128780	None
45	139163	None
46	139164	None
47	139251	None
48	113434	122332
49	118331	112996

In [60]:

```
# Stack splited parts into one new column
sb_part = sb_part.stack()
sb_part
```

Out[60]:

```
0   0     100001
1   0     100006
2   0     100049
3   0     100087
4   0     100089
5   0     100092
6   0     100224
7   0     100349
8   0     100514
9   0     101345
10  0     101350
11  0     101352
12  0     101357
13  0     101359
14  0     101367
15  0     101372
16  0     101385
17  0     102267
18  0     102335
19  0     104875
20  0     105163
21  0     105307
22  0     105330
23  0     106569
24  0     109041
25  0     110519
26  0     110630
27  0     110789
28  0    _111401
29  0    _111402
30  0    _111451
31  0    _113453
32  0    _114254
33  0    _114334
34  0    _114670
35  0    _117228
36  0    *117327
37  0    *119030
38  0    *121214
39  0    *122628
40  0     122998
41  0     123491
42  0     123492
43  0     123502
44  0     128780
45  0     139163
46  0     139164
47  0     139251
48  0     113434
    1     122332
49  0     118331
    1     112996
dtype: object
```

In [61]:

```
# Reset index
sb_part = sb_part.reset_index(level=1, drop=True)
sb_part
```

Out[61]:

```
0      100001
1      100006
2      100049
3      100087
4      100089
5      100092
6      100224
7      100349
8      100514
9      101345
10     101350
11     101352
12     101357
13     101359
14     101367
15     101372
16     101385
17     102267
18     102335
19     104875
20     105163
21     105307
22     105330
23     106569
24     109041
25     110519
26     110630
27     110789
28    _111401
29    _111402
30    _111451
31    _113453
32    _114254
33    _114334
34    _114670
35    _117228
36    *117327
37    *119030
38    *121214
39    *122628
40     122998
41     123491
42     123492
43     123502
44     128780
45     139163
46     139164
47     139251
48     113434
48     122332
49     118331
49     112996
dtype: object
```

In [62]:

```
# Modify the dataframe and rename the new column as a new 'Part_No'
sb_part = sb_part.to_frame().rename({0:'Part_No'}, axis=1)
sb_part
```

Out[62]:

	Part_No
0	100001
1	100006
2	100049
3	100087
4	100089
5	100092
6	100224
7	100349
8	100514
9	101345
10	101350
11	101352
12	101357
13	101359
14	101367
15	101372
16	101385
17	102267
18	102335
19	104875
20	105163
21	105307
22	105330
23	106569
24	109041
25	110519
26	110630
27	110789
28	_111401
29	_111402
30	_111451
31	_113453

	Part_No
32	_114254
33	_114334
34	_114670
35	_117228
36	*117327
37	*119030
38	*121214
39	*122628
40	122998
41	123491
42	123492
43	123502
44	128780
45	139163
46	139164
47	139251
48	113434
48	122332
49	118331
49	112996

In [63]:

```
# Join a new sb_cleaning table by removing old 'Part_No' and add new 'sb_part'.
sb_cleaning = sb_cleaning.drop(['Part_No'], axis=1).join(sb_part)
sb_cleaning
```

Out[63]:

_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	type	sb
0	0.0	0.0	0.0	0.0	1.0	Single_beds	
0	0.0	0.0	1.0	0.0	0.0	Single_beds	
0	0.0	1.0	1.0	0.0	0.0	Single_beds	
4	4.0	0.0	0.0	0.0	0.0	Single_beds	
1	1.0	0.0	0.0	0.0	0.0	Single_beds	
1	1.0	0.0	0.0	0.0	0.0	Single_beds	
0	0.0	0.0	18.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	12.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	14.0	Single_beds	
0	40.0	0.0	8.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	26.0	Single_beds	
18	18.0	0.0	0.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	0.0	Single_beds	
12	12.0	12.0	12.0	0.0	0.0	Single_beds	
0	0.0	3.0	3.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	12.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	0.0	Single_beds	
4	4.0	4.0	4.0	0.0	4.0	Single_beds	
4	4.0	0.0	0.0	0.0	0.0	Single_beds	
4	4.0	0.0	0.0	0.0	0.0	Single_beds	
4	4.0	4.0	4.0	0.0	4.0	Single_beds	
0	0.0	0.0	0.0	0.0	16.0	Single_beds	
0	0.0	0.0	0.0	0.0	14.0	Single_beds	
0	0.0	0.0	0.0	0.0	0.0	Single_beds	
0	20.0	0.0	0.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	12.0	0.0	Single_beds	
0	22.0	0.0	16.0	0.0	0.0	Single_beds	
16	16.0	16.0	16.0	0.0	0.0	Single_beds	
0	0.0	0.0	8.0	0.0	0.0	Single_beds	
0	10.0	0.0	0.0	0.0	0.0	Single_beds	
4	4.0	0.0	0.0	0.0	0.0	Single_beds	
0	0.0	1.0	1.0	0.0	0.0	Single_beds	

_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_	TARVA_75_	type	sb
0	0.0	3.0	3.0	0.0	0.0	Single_beds	
0	0.0	3.0	3.0	0.0	0.0	Single_beds	
0	0.0	8.0	8.0	0.0	0.0	Single_beds	
2	2.0	0.0	0.0	0.0	2.0	Single_beds	
5	5.0	5.0	5.0	0.0	5.0	Single_beds	
0	0.0	0.0	0.0	12.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	0.0	Single_beds	
0	0.0	4.0	4.0	0.0	0.0	Single_beds	
0	0.0	3.0	3.0	0.0	0.0	Single_beds	
0	0.0	8.0	8.0	0.0	0.0	Single_beds	
0	0.0	0.0	16.0	0.0	0.0	Single_beds	
0	0.0	0.0	16.0	0.0	0.0	Single_beds	
0	0.0	0.0	0.0	0.0	8.0	Single_beds	
0	0.0	0.0	0.0	3.0	0.0	Single_beds	
0	0.0	0.0	0.0	3.0	0.0	Single_beds	
0	0.0	0.0	0.0	12.0	0.0	Single_beds	
8	8.0	0.0	0.0	0.0	0.0	Single_beds	
8	8.0	0.0	0.0	0.0	0.0	Single_beds	
8	30.0	0.0	0.0	12.0	0.0	Single_beds	
8	30.0	0.0	0.0	12.0	0.0	Single_beds	

In [64]:

```
# In the new sb_cleaning dataset, there are 56 rows in new dataset, instead of 50 ro
sb_cleaning.shape
```

Out[64]:

```
(52, 15)
```

2.6 Replace punctuation

In [65]:

```
# Check the values of 'Part_No', I find that there are punctuation '_' and '*'.
sb_cleaning['Part_No'].unique()
```

Out[65]:

```
array(['100001', '100006', '100049', '100087', '100089', '100092',
       '100224', '100349', '100514', '101345', '101350', '101352',
       '101357', '101359', '101367', '101372', '101385', '102267',
       '102335', '104875', '105163', '105307', '105330', '106569',
       '109041', '110519', '110630', '110789', '_111401', '_111402',
       '_111451', '_113453', '_114254', '_114334', '_114670', '_11722
8',
       '*117327', '*119030', '*121214', '*122628', '122998', '123491',
       '123492', '123502', '128780', '139163', '139164', '139251',
       '113434', '122332', '118331', '112996'], dtype=object)
```

In [66]:

```
# Replace punctuation to '', and delete all space of Part_No.
sb_cleaning['Part_No'] = sb_cleaning['Part_No'].replace(r'[_*]','',regex=True).str.s
```

In [67]:

```
sb_cleaning['Part_No'].unique()
```

Out[67]:

```
array(['100001', '100006', '100049', '100087', '100089', '100092',
       '100224', '100349', '100514', '101345', '101350', '101352',
       '101357', '101359', '101367', '101372', '101385', '102267',
       '102335', '104875', '105163', '105307', '105330', '106569',
       '109041', '110519', '110630', '110789', '111401', '111402',
       '111451', '113453', '114254', '114334', '114670', '117228',
       '117327', '119030', '121214', '122628', '122998', '123491',
       '123492', '123502', '128780', '139163', '139164', '139251',
       '113434', '122332', '118331', '112996'], dtype=object)
```

In [68]:

```
sb_cleaning.dtypes
```

Out[68]:

```
FJELLSE_45_          int64
HEMNES_150_         object
HEMNES_220_31      float64
MALM_125_36        float64
MALM_139_30        float64
NORDLI_189_        float64
TARVA_75_          float64
type                object
sbquantity         float64
FJELLSE              int64
HEMNES             float64
MALM               float64
NORDLI             float64
TARVA              float64
Part_No             object
dtype: object
```

2.7 Check outliers

Find outliers using z-scores

Calculating mean and standard deviation for each column.

In [69]:

```
# Calculate mean and std for quantity
sbmean = np.mean(sb_cleaning.sbquantity)
sbstd = np.std(sb_cleaning.sbquantity)
print('mean of single beds\' parts quantity is ', sbmean)
print('std. deviation of single beds\' is ', sbstd)



# Calculate Z score. If Z score > 3, print it as an outlier

threshold = 3
outlier =[]
for value in sb_cleaning.sbquantity:
    z = (value - sbmean)/sbstd
    if z > threshold:
        outlier.append(value)
print('outlier in single beds quantity is ', outlier)
```

```
mean of single beds' parts quantity is  13.557692307692308
std. deviation of single beds' is  12.720990318153545
outlier in single beds quantity is  []
```

From the result above, we can see that there are no outliers in single bed's component quantity.

In [70]:

```
# Here is the final cleaning SB dataset.
sb_cleaning
```

Out[70]:

	FJELLSE_45_	HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_
0	1	0	0.0	0.0	0.0	0.0
1	0	0	0.0	0.0	1.0	0.0
2	0	0	0.0	1.0	1.0	0.0
3	0	4	4.0	0.0	0.0	0.0
4	0	1	1.0	0.0	0.0	0.0
5	0	1	1.0	0.0	0.0	0.0
6	0	0	0.0	0.0	18.0	0.0
7	0	0	0.0	0.0	0.0	12.0
8	0	0	0.0	0.0	0.0	0.0
9	0	0	40.0	0.0	8.0	0.0
10	18	0	0.0	0.0	0.0	0.0
11	0	18	18.0	0.0	0.0	0.0
12	4	0	0.0	0.0	0.0	0.0
13	0	12	12.0	12.0	12.0	0.0
14	0	0	0.0	3.0	3.0	0.0
15	0	0	0.0	0.0	0.0	12.0
16	6	0	0.0	0.0	0.0	0.0
17	0	4	4.0	4.0	4.0	0.0
18	0	4	4.0	0.0	0.0	0.0
19	0	4	4.0	0.0	0.0	0.0
20	0	4	4.0	4.0	4.0	0.0
21	0	0	0.0	0.0	0.0	0.0
22	0	0	0.0	0.0	0.0	0.0
23	8	0	0.0	0.0	0.0	0.0
24	0	0	20.0	0.0	0.0	0.0
25	0	0	0.0	0.0	0.0	12.0
26	0	0	22.0	0.0	16.0	0.0
27	0	16	16.0	16.0	16.0	0.0
28	0	0	0.0	0.0	8.0	0.0
29	0	0	10.0	0.0	0.0	0.0
30	0	4	4.0	0.0	0.0	0.0
31	0	0	0.0	1.0	1.0	0.0

	FJELLSE_45_	HEMNES_150_	HEMNES_220_31	MALM_125_36	MALM_139_30	NORDLI_189_
32	0	0	0.0	3.0	3.0	0.0
33	0	0	0.0	3.0	3.0	0.0
34	0	0	0.0	8.0	8.0	0.0
35	0	2	2.0	0.0	0.0	0.0
36	0	5	5.0	5.0	5.0	0.0
37	0	0	0.0	0.0	0.0	12.0
38	4	0	0.0	0.0	0.0	0.0
39	0	0	0.0	4.0	4.0	0.0
40	0	0	0.0	3.0	3.0	0.0
41	0	0	0.0	8.0	8.0	0.0
42	0	0	0.0	0.0	16.0	0.0
43	0	0	0.0	0.0	16.0	0.0
44	0	0	0.0	0.0	0.0	0.0
45	0	0	0.0	0.0	0.0	3.0
46	0	0	0.0	0.0	0.0	3.0
47	0	0	0.0	0.0	0.0	12.0
48	0	8	8.0	0.0	0.0	0.0
48	0	8	8.0	0.0	0.0	0.0
49	0	8	30.0	0.0	0.0	12.0
49	0	8	30.0	0.0	0.0	12.0

In [71]:

```
# Save cleaned SB dataset
sb_cleaning.to_csv('SB_cleaning.csv')
```

In [ ]:

```

```

In [ ]:

```

```