Question 2 - Data Merge

In [1]:

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

In [2]:

```
sb_cleaning = pd.read_csv('SB_cleaning.csv')  # Read cleaned SB dataset
gb_cleaning = pd.read_csv('GB_cleaning.csv')  # Read cleaned GB dataset
```

In [3]:

```
sb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52 entries, 0 to 51
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      52 non-null     int64  
 1   FJELLSE_45_     52 non-null     int64  
 2   HEMNES_150_     52 non-null     int64  
 3   HEMNES_220_31   52 non-null     float64
 4   MALM_125_36     52 non-null     float64
 5   MALM_139_30     52 non-null     float64
 6   NORDLI_189_     52 non-null     float64
 7   TARVA_75_       52 non-null     float64
 8   type            52 non-null     object 
 9   sbquantity      52 non-null     float64
 10  FJELLSE         52 non-null     int64  
 11  HEMNES          52 non-null     float64
 12  MALM            52 non-null     float64
 13  NORDLI          52 non-null     float64
 14  TARVA           52 non-null     float64
 15  Part_No         52 non-null     int64  
dtypes: float64(10), int64(5), object(1)
memory usage: 6.6+ KB
```

- As I will do the analysis of each type's part number in their quantity usage, I will select Part_No, type, and quantity, and combined product names from two dataset. My data merge will based on these variables.

In [4]:

```
# Select combined products from SB dataset.
sb = sb_cleaning[['Part_No','type','sbquantity', 'FJELLSE', 'HEMNES', 'MALM', 'NORDI
sb
```

Out[4]:

	Part_No	type	sbquantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA
0	100001	Single_beds	2.0	1	0.0	0.0	0.0	1.0
1	100006	Single_beds	1.0	0	0.0	1.0	0.0	0.0
2	100049	Single_beds	2.0	0	0.0	2.0	0.0	0.0
3	100087	Single_beds	4.0	0	4.0	0.0	0.0	0.0
4	100089	Single_beds	1.0	0	1.0	0.0	0.0	0.0
5	100092	Single_beds	1.0	0	1.0	0.0	0.0	0.0
6	100224	Single_beds	18.0	0	0.0	18.0	0.0	0.0
7	100349	Single_beds	12.0	0	0.0	0.0	12.0	0.0
8	100514	Single_beds	14.0	0	0.0	0.0	0.0	14.0
9	101345	Single_beds	48.0	0	40.0	8.0	0.0	0.0
10	101350	Single_beds	44.0	18	0.0	0.0	0.0	26.0
11	101352	Single_beds	18.0	0	18.0	0.0	0.0	0.0
12	101357	Single_beds	4.0	4	0.0	0.0	0.0	0.0
13	101359	Single_beds	36.0	0	12.0	24.0	0.0	0.0
14	101367	Single_beds	6.0	0	0.0	6.0	0.0	0.0
15	101372	Single_beds	12.0	0	0.0	0.0	12.0	0.0
16	101385	Single_beds	6.0	6	0.0	0.0	0.0	0.0
17	102267	Single_beds	16.0	0	4.0	8.0	0.0	4.0
18	102335	Single_beds	4.0	0	4.0	0.0	0.0	0.0
19	104875	Single_beds	4.0	0	4.0	0.0	0.0	0.0
20	105163	Single_beds	16.0	0	4.0	8.0	0.0	4.0
21	105307	Single_beds	16.0	0	0.0	0.0	0.0	16.0
22	105330	Single_beds	14.0	0	0.0	0.0	0.0	14.0
23	106569	Single_beds	8.0	8	0.0	0.0	0.0	0.0
24	109041	Single_beds	20.0	0	20.0	0.0	0.0	0.0
25	110519	Single_beds	12.0	0	0.0	0.0	12.0	0.0
26	110630	Single_beds	38.0	0	22.0	16.0	0.0	0.0
27	110789	Single_beds	48.0	0	16.0	32.0	0.0	0.0
28	111401	Single_beds	8.0	0	0.0	8.0	0.0	0.0
29	111402	Single_beds	10.0	0	10.0	0.0	0.0	0.0
30	111451	Single_beds	4.0	0	4.0	0.0	0.0	0.0
31	113453	Single_beds	2.0	0	0.0	2.0	0.0	0.0

	Part_No	type	sbquantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA
32	114254	Single_beds	6.0	0	0.0	6.0	0.0	0.0
33	114334	Single_beds	6.0	0	0.0	6.0	0.0	0.0
34	114670	Single_beds	16.0	0	0.0	16.0	0.0	0.0
35	117228	Single_beds	4.0	0	2.0	0.0	0.0	2.0
36	117327	Single_beds	20.0	0	5.0	10.0	0.0	5.0
37	119030	Single_beds	12.0	0	0.0	0.0	12.0	0.0
38	121214	Single_beds	4.0	4	0.0	0.0	0.0	0.0
39	122628	Single_beds	8.0	0	0.0	8.0	0.0	0.0
40	122998	Single_beds	6.0	0	0.0	6.0	0.0	0.0
41	123491	Single_beds	16.0	0	0.0	16.0	0.0	0.0
42	123492	Single_beds	16.0	0	0.0	16.0	0.0	0.0
43	123502	Single_beds	16.0	0	0.0	16.0	0.0	0.0
44	128780	Single_beds	8.0	0	0.0	0.0	0.0	8.0
45	139163	Single_beds	3.0	0	0.0	0.0	3.0	0.0
46	139164	Single_beds	3.0	0	0.0	0.0	3.0	0.0
47	139251	Single_beds	12.0	0	0.0	0.0	12.0	0.0
48	113434	Single_beds	8.0	0	8.0	0.0	0.0	0.0
49	122332	Single_beds	8.0	0	8.0	0.0	0.0	0.0
50	118331	Single_beds	42.0	0	30.0	0.0	12.0	0.0
51	112996	Single_beds	42.0	0	30.0	0.0	12.0	0.0

In [5]:

```
gb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 85 entries, 0 to 84
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       85 non-null     int64  
 1   BRIMNES_329_22   85 non-null     float64
 2   FLEKKE_399_      85 non-null     float64
 3   FYRESDAL_299_    85 non-null     float64
 4   HEMNES_409_24    85 non-null     float64
 5   TARVA_119_       85 non-null     float64
 6   UT_ER_299_       85 non-null     float64
 7   UT_ER_299_.1     85 non-null     float64
 8   UT_ER_299_.2     85 non-null     float64
 9   GB_unnamed_part  85 non-null     float64
 10  type             85 non-null     object 
 11  Part_No          85 non-null     int64  
 12  gbquantity       85 non-null     float64
 13  BRIMNES          85 non-null     float64
 14  FLEKKE           85 non-null     float64
 15  FYRESDAL         85 non-null     float64
 16  HEMNES           85 non-null     float64
 17  TARVA            85 non-null     float64
 18  UT               85 non-null     float64
dtypes: float64(16), int64(2), object(1)
memory usage: 12.7+ KB
```

In [6]:

```
# Select combined products from GB dataset.
gb = gb_cleaning[['Part_No', 'type', 'gbquantity','BRIMNES', 'FLEKKE', 'FYRESDAL', '
gb
```

Out[6]:

	Part_No	type	gbquantity	BRIMNES	FLEKKE	FYRESDAL	HEMNES	TARVA	UT
0	100001	Guest_beds	5.0	0.0	1.0	1.0	1.0	1.0	0.0
1	100027	Guest_beds	2.0	0.0	0.0	0.0	0.0	1.0	0.0
2	100049	Guest_beds	3.0	1.0	1.0	0.0	0.0	0.0	0.0
3	100089	Guest_beds	2.0	0.0	0.0	0.0	0.0	1.0	0.0
4	100211	Guest_beds	9.0	0.0	0.0	0.0	0.0	8.0	0.0
...	...	...	...	...	...	...	...	...	...
80	119030	Guest_beds	31.0	22.0	8.0	0.0	0.0	0.0	0.0
81	118224	Guest_beds	31.0	22.0	8.0	0.0	0.0	0.0	0.0
82	117434	Guest_beds	31.0	22.0	8.0	0.0	0.0	0.0	0.0
83	124328	Guest_beds	3.0	2.0	0.0	0.0	0.0	0.0	0.0
84	128763	Guest_beds	3.0	2.0	0.0	0.0	0.0	0.0	0.0

85 rows × 9 columns

2.1 Combine SB and GB datasets

As I consider the identical parts for two bed types, I will not use join function because it will replace the quantity from one type. Instead, I will use concat to combine two datasets.

In this part, the analysis will based on:

- Identical part No. and their ranking based on total quantity usage.
- All parts and their ranking based on total quantity usage.

In [7]:

```
# Combine gb and sb dataset, and reset index values.
sgb = pd.concat([sb,gb],ignore_index=True).fillna(0)
sgb
```

Out[7]:

	Part_No	type	sbquantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	gbquantity	BRIMNES	FLE
0	100001	Single_beds	2.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
2	100049	Single_beds	2.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	
3	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	
4	100089	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
132	119030	Guest_beds	0.0	0.0	0.0	0.0	0.0	0.0	31.0	22.0	
133	118224	Guest_beds	0.0	0.0	0.0	0.0	0.0	0.0	31.0	22.0	
134	117434	Guest_beds	0.0	0.0	0.0	0.0	0.0	0.0	31.0	22.0	
135	124328	Guest_beds	0.0	0.0	0.0	0.0	0.0	0.0	3.0	2.0	

In [8]:

```
sgb.columns
```

Out[8]:

```
Index(['Part_No', 'type', 'sbquantity', 'FJELLSE', 'HEMNES', 'MALM',
'NORDLI',
       'TARVA', 'gbquantity', 'BRIMNES', 'FLEKKE', 'FYRESDAL', 'UT'],
      dtype='object')
```

In [9]:

```
# Based on part number, check duplicates of components.
sgb['Part_No'].duplicated().sum()
```

Out[9]:

```
24
```

There are 24 duplicates components, meaning these part No. components are both used in two bed types. Thus, I will analyse these identical part No. in the next sector.

2.2 Merge Identical part No. are handled by two types of beds

In [10]:

```
# Use inner join to check the identical parts are handled by both two types.
identical = pd.merge(gb,sb, how='inner', on = 'Part_No')
identical
```

Out[10]:

	Part_No	type_x	gbquantity	BRIMNES	FLEKKE	FYRESDAL	HEMNES_x	TARVA_x	U
0	100001	Guest_beds	5.0	0.0	1.0	1.0	1.0	1.0	0.
1	100049	Guest_beds	3.0	1.0	1.0	0.0	0.0	0.0	0.
2	100089	Guest_beds	2.0	0.0	0.0	0.0	0.0	1.0	0.
3	100514	Guest_beds	11.0	0.0	0.0	0.0	0.0	10.0	0.
4	101345	Guest_beds	41.0	32.0	8.0	0.0	0.0	0.0	0.
5	101350	Guest_beds	65.0	0.0	24.0	0.0	34.0	6.0	0.
6	101352	Guest_beds	15.0	0.0	0.0	0.0	0.0	0.0	14.
7	101359	Guest_beds	51.0	4.0	0.0	0.0	0.0	46.0	0.
8	101367	Guest_beds	9.0	0.0	8.0	0.0	0.0	0.0	0.
9	104875	Guest_beds	5.0	0.0	0.0	0.0	0.0	4.0	0.
10	105163	Guest_beds	3.0	2.0	0.0	0.0	0.0	0.0	0.
11	105307	Guest_beds	78.0	0.0	58.0	0.0	0.0	19.0	0.
12	105330	Guest_beds	7.0	0.0	0.0	0.0	0.0	6.0	0.
13	110630	Guest_beds	17.0	0.0	16.0	0.0	0.0	0.0	0.
14	111401	Guest_beds	23.0	6.0	8.0	0.0	8.0	0.0	0.
15	111451	Guest_beds	3.0	0.0	0.0	0.0	0.0	2.0	0.
16	113453	Guest_beds	2.0	0.0	1.0	0.0	0.0	0.0	0.
17	114670	Guest_beds	13.0	4.0	8.0	0.0	0.0	0.0	0.
18	122628	Guest_beds	5.0	4.0	0.0	0.0	0.0	0.0	0.
19	128780	Guest_beds	9.0	0.0	0.0	0.0	0.0	8.0	0.
20	110519	Guest_beds	17.0	8.0	8.0	0.0	0.0	0.0	0.
21	118331	Guest_beds	87.0	22.0	24.0	0.0	40.0	0.0	0.
22	112996	Guest_beds	87.0	22.0	24.0	0.0	40.0	0.0	0.
23	119030	Guest_beds	31.0	22.0	8.0	0.0	0.0	0.0	0.

Check datatype os identical dataset, and change dtype

In [11]:

```python
# Check and change datatype
identical.dtypes
```

Out[11]:

```
Part_No          int64
type_x          object
gbquantity     float64
BRIMNES        float64
FLEKKE         float64
FYRESDAL       float64
HEMNES_x       float64
TARVA_x        float64
UT             float64
type_y          object
sbquantity     float64
FJELLSE          int64
HEMNES_y       float64
MALM           float64
NORDLI         float64
TARVA_y        float64
dtype: object
```

In [12]:

```python
# Change Part_No's data type
identical['Part_No'] = identical['Part_No'].astype('object')
identical.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24 entries, 0 to 23
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Part_No     24 non-null     object 
 1   type_x      24 non-null     object 
 2   gbquantity  24 non-null     float64
 3   BRIMNES     24 non-null     float64
 4   FLEKKE      24 non-null     float64
 5   FYRESDAL    24 non-null     float64
 6   HEMNES_x    24 non-null     float64
 7   TARVA_x     24 non-null     float64
 8   UT          24 non-null     float64
 9   type_y      24 non-null     object 
 10  sbquantity  24 non-null     float64
 11  FJELLSE     24 non-null     int64  
 12  HEMNES_y    24 non-null     float64
 13  MALM        24 non-null     float64
 14  NORDLI      24 non-null     float64
 15  TARVA_y     24 non-null     float64
dtypes: float64(12), int64(1), object(3)
memory usage: 3.2+ KB
```

From table above

- We can see there are 24 identical parts, and their usage quantity in each bed type. In order to analyse which identical parts are most popular, I will calculate the sum quantity of each identical parts.

- As these identical parts appear in both single and guest beds, I will change their type name as 'Guest_Single_beds'.
- There are two products appear in both bed types, HEMNES and TARVA. So we also need to calculate the sum quantity of these two products.

In [13]:

```
# Calculate the sum quantity uage of identical parts
identical['quantity'] = identical['sbquantity'] + identical['gbquantity']

# Create new 'type' value for identical parts, named as 'Guests_Single_beds'
identical['type'] = 'Guest_Single_beds'

# Calculate identical quantity of HEMNES and TARVA.
identical['TARVA'] = identical[list(identical.filter(regex='TARVA'))].sum(axis=1)
identical['HEMNES'] = identical[list(identical.filter(regex='HEMNES'))].sum(axis=1)
```

In [14]:

```
# Select combined parts from old 'identical' dataset
identical = identical[['Part_No', 'type', 'quantity','FJELLSE', 'HEMNES', 'MALM', 'N
       'TARVA', 'BRIMNES', 'FLEKKE', 'FYRESDAL', 'UT']]
```

In [15]:

```
identical
```

Out[15]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNE
0	100001	Guest_Single_beds	7.0	1	1.0	0.0	0.0	2.0	0
1	100049	Guest_Single_beds	5.0	0	0.0	2.0	0.0	0.0	1
2	100089	Guest_Single_beds	3.0	0	1.0	0.0	0.0	1.0	0
3	100514	Guest_Single_beds	25.0	0	0.0	0.0	0.0	24.0	0
4	101345	Guest_Single_beds	89.0	0	40.0	8.0	0.0	0.0	32
5	101350	Guest_Single_beds	109.0	18	34.0	0.0	0.0	32.0	0
6	101352	Guest_Single_beds	33.0	0	18.0	0.0	0.0	0.0	0
7	101359	Guest_Single_beds	87.0	0	12.0	24.0	0.0	46.0	4
8	101367	Guest_Single_beds	15.0	0	0.0	6.0	0.0	0.0	0
9	104875	Guest_Single_beds	9.0	0	4.0	0.0	0.0	4.0	0
10	105163	Guest_Single_beds	19.0	0	4.0	8.0	0.0	4.0	2
11	105307	Guest_Single_beds	94.0	0	0.0	0.0	0.0	35.0	0
12	105330	Guest_Single_beds	21.0	0	0.0	0.0	0.0	20.0	0
13	110630	Guest_Single_beds	55.0	0	22.0	16.0	0.0	0.0	0
14	111401	Guest_Single_beds	31.0	0	8.0	8.0	0.0	0.0	6
15	111451	Guest_Single_beds	7.0	0	4.0	0.0	0.0	2.0	0
16	113453	Guest_Single_beds	4.0	0	0.0	2.0	0.0	0.0	0
17	114670	Guest_Single_beds	29.0	0	0.0	16.0	0.0	0.0	4
18	122628	Guest_Single_beds	13.0	0	0.0	8.0	0.0	0.0	4
19	128780	Guest_Single_beds	17.0	0	0.0	0.0	0.0	16.0	0
20	110519	Guest_Single_beds	29.0	0	0.0	0.0	12.0	0.0	8
21	118331	Guest_Single_beds	129.0	0	70.0	0.0	12.0	0.0	22
22	112996	Guest_Single_beds	129.0	0	70.0	0.0	12.0	0.0	22
23	119030	Guest_Single_beds	43.0	0	0.0	0.0	12.0	0.0	22

2.3 Rank the identical components (based on Part No.) by the quantity of their use.

In [16]:

```python
# Rank quantity of identical parts
identical.sort_values(by=['quantity'], ascending = False).set_index('quantity')
```

Out[16]:

quantity	Part_No	type	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES
129.0	112996	Guest_Single_beds	0	70.0	0.0	12.0	0.0	22.0
129.0	118331	Guest_Single_beds	0	70.0	0.0	12.0	0.0	22.0
109.0	101350	Guest_Single_beds	18	34.0	0.0	0.0	32.0	0.0
94.0	105307	Guest_Single_beds	0	0.0	0.0	0.0	35.0	0.0
89.0	101345	Guest_Single_beds	0	40.0	8.0	0.0	0.0	32.0
87.0	101359	Guest_Single_beds	0	12.0	24.0	0.0	46.0	4.0
55.0	110630	Guest_Single_beds	0	22.0	16.0	0.0	0.0	0.0
43.0	119030	Guest_Single_beds	0	0.0	0.0	12.0	0.0	22.0
33.0	101352	Guest_Single_beds	0	18.0	0.0	0.0	0.0	0.0
31.0	111401	Guest_Single_beds	0	8.0	8.0	0.0	0.0	6.0
29.0	110519	Guest_Single_beds	0	0.0	0.0	12.0	0.0	8.0
29.0	114670	Guest_Single_beds	0	0.0	16.0	0.0	0.0	4.0
25.0	100514	Guest_Single_beds	0	0.0	0.0	0.0	24.0	0.0
21.0	105330	Guest_Single_beds	0	0.0	0.0	0.0	20.0	0.0
19.0	105163	Guest_Single_beds	0	4.0	8.0	0.0	4.0	2.0
17.0	128780	Guest_Single_beds	0	0.0	0.0	0.0	16.0	0.0
15.0	101367	Guest_Single_beds	0	0.0	6.0	0.0	0.0	0.0
13.0	122628	Guest_Single_beds	0	0.0	8.0	0.0	0.0	4.0
9.0	104875	Guest_Single_beds	0	4.0	0.0	0.0	4.0	0.0
7.0	111451	Guest_Single_beds	0	4.0	0.0	0.0	2.0	0.0
7.0	100001	Guest_Single_beds	1	1.0	0.0	0.0	2.0	0.0
5.0	100049	Guest_Single_beds	0	0.0	2.0	0.0	0.0	1.0
4.0	113453	Guest_Single_beds	0	0.0	2.0	0.0	0.0	0.0
3.0	100089	Guest_Single_beds	0	1.0	0.0	0.0	1.0	0.0

From the table above, we can see that there components number No.112996 and No.118331 are highest usage in both two bed types, with 129 quantity in total. In addition, No.101350 is also highly frequency usage with 109 total quantity.

On the contrary, there are six parts (No.104875, 111451, 100001, 100049, 113453,100089) are least used in both two types, below 10 quantity usage in total.

2.4 Combine all components by the total quantity of their

use.

Merge new sgb_cleaning dataset that only shows all non-duplicated values

- Drop duplicate rows from old 'sgb' dataset, and combine the 'identical' dataset into a new sgb_cleaning dataset. In new sgb_cleaning dataset, previous duplicate/identical parts' type is 'Guest_Single_beds', and their quantity is the total usage quantity in each product series.

In [17]:

```
# Drop all duplicates part numbers, and create dataset for unique components
sgb_uni = sgb.drop_duplicates('Part_No', keep=False)

# Calculate the total quantity
sgb_uni['quantity'] = sgb_uni['sbquantity'] + sgb_uni['gbquantity']

sgb_uni = sgb_uni[['Part_No', 'type', 'quantity','FJELLSE', 'HEMNES', 'MALM', 'NORDI
       'TARVA', 'BRIMNES', 'FLEKKE', 'FYRESDAL', 'UT']] # this dataframe is all 89 u
```

```
/var/folders/t5/vfwg7gv55v9_n8vzkxs4bf900000gn/T/ipykernel_944/3584426
87.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas
-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
  sgb_uni['quantity'] = sgb_uni['sbquantity'] + sgb_uni['gbquantity']
```

In [18]:

```
sgb_uni # this dataframe is all 89 unique parts.
```

Out[18]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FI
1	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
3	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
5	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
6	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	
7	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	
129	118149	Guest_beds	17.0	0.0	0.0	0.0	0.0	0.0	8.0	
133	118224	Guest_beds	31.0	0.0	0.0	0.0	0.0	0.0	22.0	
134	117434	Guest_beds	31.0	0.0	0.0	0.0	0.0	0.0	22.0	
135	124328	Guest_beds	3.0	0.0	0.0	0.0	0.0	0.0	2.0	
136	128763	Guest_beds	3.0	0.0	0.0	0.0	0.0	0.0	2.0	

89 rows × 12 columns

In [19]:

```
sgb_uni.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 89 entries, 1 to 136
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Part_No   89 non-null     int64  
 1   type      89 non-null     object 
 2   quantity  89 non-null     float64
 3   FJELLSE   89 non-null     float64
 4   HEMNES    89 non-null     float64
 5   MALM      89 non-null     float64
 6   NORDLI    89 non-null     float64
 7   TARVA     89 non-null     float64
 8   BRIMNES   89 non-null     float64
 9   FLEKKE    89 non-null     float64
 10  FYRESDAL  89 non-null     float64
 11  UT        89 non-null     float64
dtypes: float64(10), int64(1), object(1)
memory usage: 9.0+ KB
```

In [20]:

```
# Combine identical dataset(sub) with dropped datasets(sgb_uni)
sgb_cleaning = pd.concat([sgb_uni,identical],ignore_index=True)

sgb_cleaning
```

Out[20]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLEKKE	FY
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
1	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
3	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	0.0	
4	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
108	128780	Guest_Single_beds	17.0	0.0	0.0	0.0	0.0	16.0	0.0	0.0	
109	110519	Guest_Single_beds	29.0	0.0	0.0	0.0	12.0	0.0	8.0	8.0	
110	118331	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	22.0	24.0	
111	112996	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	22.0	24.0	

In [21]:

```
sgb_cleaning.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113 entries, 0 to 112
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Part_No   113 non-null    object 
 1   type      113 non-null    object 
 2   quantity  113 non-null    float64
 3   FJELLSE   113 non-null    float64
 4   HEMNES    113 non-null    float64
 5   MALM      113 non-null    float64
 6   NORDLI    113 non-null    float64
 7   TARVA     113 non-null    float64
 8   BRIMNES   113 non-null    float64
 9   FLEKKE    113 non-null    float64
 10  FYRESDAL  113 non-null    float64
 11  UT        113 non-null    float64
dtypes: float64(10), object(2)
memory usage: 10.7+ KB
```

In [22]:

```
# From the table, we can see that No.112996 and No.119030 all value are correct.
sgb_cleaning.tail()
```

Out[22]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMN
108	128780	Guest_Single_beds	17.0	0.0	0.0	0.0	0.0	16.0	
109	110519	Guest_Single_beds	29.0	0.0	0.0	0.0	12.0	0.0	
110	118331	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	2
111	112996	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	2
112	119030	Guest_Single_beds	43.0	0.0	0.0	0.0	12.0	0.0	2

2.5 Rank total quantity of parts.

In [23]:

```
# Sort values of total quantity
sgb_sorted = sgb_cleaning.sort_values(by=['quantity'], ascending = False)
sgb_sorted
```

Out[23]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMN
111	112996	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	2
110	118331	Guest_Single_beds	129.0	0.0	70.0	0.0	12.0	0.0	2
94	101350	Guest_Single_beds	109.0	18.0	34.0	0.0	0.0	32.0	
49	110525	Guest_beds	109.0	0.0	32.0	0.0	0.0	14.0	3
57	116894	Guest_beds	105.0	0.0	50.0	0.0	0.0	0.0	5
...	...	...	...	...	...	...	...	...	
51	111631	Guest_beds	2.0	0.0	0.0	0.0	0.0	0.0	
77	151641	Guest_beds	2.0	0.0	0.0	0.0	0.0	1.0	
28	100027	Guest_beds	2.0	0.0	0.0	0.0	0.0	1.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	

113 rows × 12 columns

From the table above, we can see that the highest frequency use is Part No.112996 and No.118331, both 129 total usage quantity. And top 3 usage quantity are all 'guest_single_beds', meaning top 3 components are used in both bed types. Following top 3, rank 4 and 5 are used over 100 quantities, both are unique guest beds.

On the contrary, the lowest usage quantity appears in single bed type, with only 1 quantity in No.100092 and No.100092. And No.111631, No.151641, and 100027 are belong to unique guest beds, with very low usage as 2 quantity.

Question 3 Data Analysis

Calculate the percentage of unique components in each bed type

As each component has different quantity, I assume higher quantity means higher weight. Thus, I will calculate the percentage based on the total quantity of unique components in each bed type.

In [24]:

```
# Split unique single bed
unisingle = sgb_cleaning[sgb_cleaning['type'] == 'Single_beds']
unisingle
```

Out[24]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLI
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	
4	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	
5	101357	Single_beds	4.0	4.0	0.0	0.0	0.0	0.0	0.0	
6	101372	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	
7	101385	Single_beds	6.0	6.0	0.0	0.0	0.0	0.0	0.0	
8	102267	Single_beds	16.0	0.0	4.0	8.0	0.0	4.0	0.0	
9	102335	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
10	106569	Single_beds	8.0	8.0	0.0	0.0	0.0	0.0	0.0	
11	109041	Single_beds	20.0	0.0	20.0	0.0	0.0	0.0	0.0	
12	110789	Single_beds	48.0	0.0	16.0	32.0	0.0	0.0	0.0	
13	111402	Single_beds	10.0	0.0	10.0	0.0	0.0	0.0	0.0	
14	114254	Single_beds	6.0	0.0	0.0	6.0	0.0	0.0	0.0	
15	114334	Single_beds	6.0	0.0	0.0	6.0	0.0	0.0	0.0	
16	117228	Single_beds	4.0	0.0	2.0	0.0	0.0	2.0	0.0	
17	117327	Single_beds	20.0	0.0	5.0	10.0	0.0	5.0	0.0	
18	121214	Single_beds	4.0	4.0	0.0	0.0	0.0	0.0	0.0	
19	122998	Single_beds	6.0	0.0	0.0	6.0	0.0	0.0	0.0	
20	123491	Single_beds	16.0	0.0	0.0	16.0	0.0	0.0	0.0	
21	123492	Single_beds	16.0	0.0	0.0	16.0	0.0	0.0	0.0	
22	123502	Single_beds	16.0	0.0	0.0	16.0	0.0	0.0	0.0	
23	139163	Single_beds	3.0	0.0	0.0	0.0	3.0	0.0	0.0	
24	139164	Single_beds	3.0	0.0	0.0	0.0	3.0	0.0	0.0	
25	139251	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	
26	113434	Single_beds	8.0	0.0	8.0	0.0	0.0	0.0	0.0	
27	122332	Single_beds	8.0	0.0	8.0	0.0	0.0	0.0	0.0	

In [25]:

```
# Calculate the total quantity of unique components of single beds.
unisingle = unisingle['quantity'].sum()
```

In [26]:

```
unisingle # There are 292 unique components in total usage quantity of single beds.
```

Out[26]:

```
292.0
```

From the previous calculation, we have got identical dataset. So we only need to calculate the total quantity of identical components.

In [27]:

```
iden = identical['quantity'].sum()
```

In [28]:

```
iden # There are 1002 components in total quantity that are used in both single and
```

Out[28]:

```
1002.0
```

In [29]:

```
# Calculate the percentage of unique components in single bed, and keep two decimal
single_per = round(unisingle / (unisingle + iden) * 100, 2)
print('The percentage of unique components quantity in single bed is:', single_per,
```

```
The percentage of unique components quantity in single bed is: 22.57 %
```

For single beds, the percentage of unique components (in terms of their quantity usage) is:

22.57 %

In [30]:

```
# Split unique guest bed
uniguest = sgb_cleaning[sgb_cleaning['type'] == 'Guest_beds']

# Calculate the total quantity of unique components of guest beds.
uniguest = uniguest['quantity'].sum()
```

In [31]:

```
uniguest # There are 848 unique components in total usage quantity of guest beds.
```

Out[31]:

```
848.0
```

In [32]:

```
# Calculate the percentage of unique components in single bed, and keep two decimal
guest_per = round(uniguest / (uniguest + iden) * 100, 2)
print('The percentage of unique components quantity in single bed is:', guest_per, '
```

```
The percentage of unique components quantity in single bed is: 45.84 %
```

For guest beds, the percentage of unique components (in terms of their quantity usage) is:

45.84 %

Question 4 Data Discovery

- PCA Project to 2D

In [33]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
%matplotlib notebook
```

In [34]:

```
sgb_cleaning.head()
```

Out[34]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLE
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	
4	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	

Name columns and normalise values in [0,1]

In [35]:

```
sgb_cleaning.columns
```

Out[35]:

```
Index(['Part_No', 'type', 'quantity', 'FJELLSE', 'HEMNES', 'MALM', 'NO
RDLI',
       'TARVA', 'BRIMNES', 'FLEKKE', 'FYRESDAL', 'UT'],
      dtype='object')
```

In [36]:

```
# Define quantity as parameters
features = ['FJELLSE', 'HEMNES', 'MALM', 'NORDLI',
       'TARVA', 'BRIMNES', 'FLEKKE', 'FYRESDAL', 'UT']

x = sgb_cleaning.loc[:,features].values
```

In [37]:

```
# Define target parameter, type as independent variable
y = sgb_cleaning.loc[:,['type']].values
```

In [38]:

```
# Scale the parameter values
x = StandardScaler().fit_transform(x)
```

In [39]:

```
pd.DataFrame(data = x, columns = features).head()
```

Out[39]:

	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLEKKE	FYRESDAL	
0	-0.183431	-0.416015	-0.196612	-0.274067	-0.343329	-0.388029	-0.378393	-0.168287	-0.20
1	-0.183431	-0.097305	-0.381756	-0.274067	-0.343329	-0.388029	-0.378393	-0.168287	-0.20
2	-0.183431	-0.336338	-0.381756	-0.274067	-0.343329	-0.388029	-0.378393	-0.168287	-0.20
3	-0.183431	-0.416015	2.950826	-0.274067	-0.343329	-0.388029	-0.378393	-0.168287	-0.20
4	-0.183431	-0.416015	-0.381756	3.855208	-0.343329	-0.388029	-0.378393	-0.168287	-0.20

- PCA Project to 2D

In [40]:

```
# Create PCA by projecting the 4D parameters onto a 2D circular
pca = PCA(n_components =2)
```

In [41]:

```
# Scaling the data onte 2D.
principalComponents = pca.fit_transform(x)
principalComponents
```

Out[41]:

```
array([[-0.76190387, -0.15199766],
       [-0.57895611, -0.22294592],
       [-0.71540392, -0.19831215],
       [-0.77919878,  0.49575729],
       [ 0.55386302, -1.69839256],
       [-0.43335128,  0.83407245],
       [ 0.55386302, -1.69839256],
       [-0.26958365,  1.34615912],
       [-0.48444831,  0.41313036],
       [-0.57895611, -0.22294592],
       [-0.10581602,  1.85824579],
       [ 0.14876556, -0.35432605],
       [-0.06571998,  0.89782242],
       [-0.30606048, -0.27221347],
       [-0.76699061,  0.0385185 ],
       [-0.76699061,  0.0385185 ],
       [-0.61859803, -0.0408982 ],
       [-0.41533875,  0.56393817],
```

In [42]:

```
# Extract principle components
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal compone
```

In [43]:

```
principalDf.head()
```

Out[43]:

	principal component 1	principal component 2
0	-0.761904	-0.151998
1	-0.578956	-0.222946
2	-0.715404	-0.198312
3	-0.779199	0.495757
4	0.553863	-1.698393

In [44]:

```
sgb_cleaning[['type']]
```

Out[44]:

	type
0	Single_beds
1	Single_beds
2	Single_beds
3	Single_beds
4	Single_beds
...	...
108	Guest_Single_beds
109	Guest_Single_beds
110	Guest_Single_beds
111	Guest_Single_beds
112	Guest_Single_beds

113 rows × 1 columns

In [45]:

```
finalDf = pd.concat([principalDf, sgb_cleaning[['type']]], axis = 1)
finalDf.head()
```

Out[45]:

	principal component 1	principal component 2	type
0	-0.761904	-0.151998	Single_beds
1	-0.578956	-0.222946	Single_beds
2	-0.715404	-0.198312	Single_beds
3	-0.779199	0.495757	Single_beds
4	0.553863	-1.698393	Single_beds

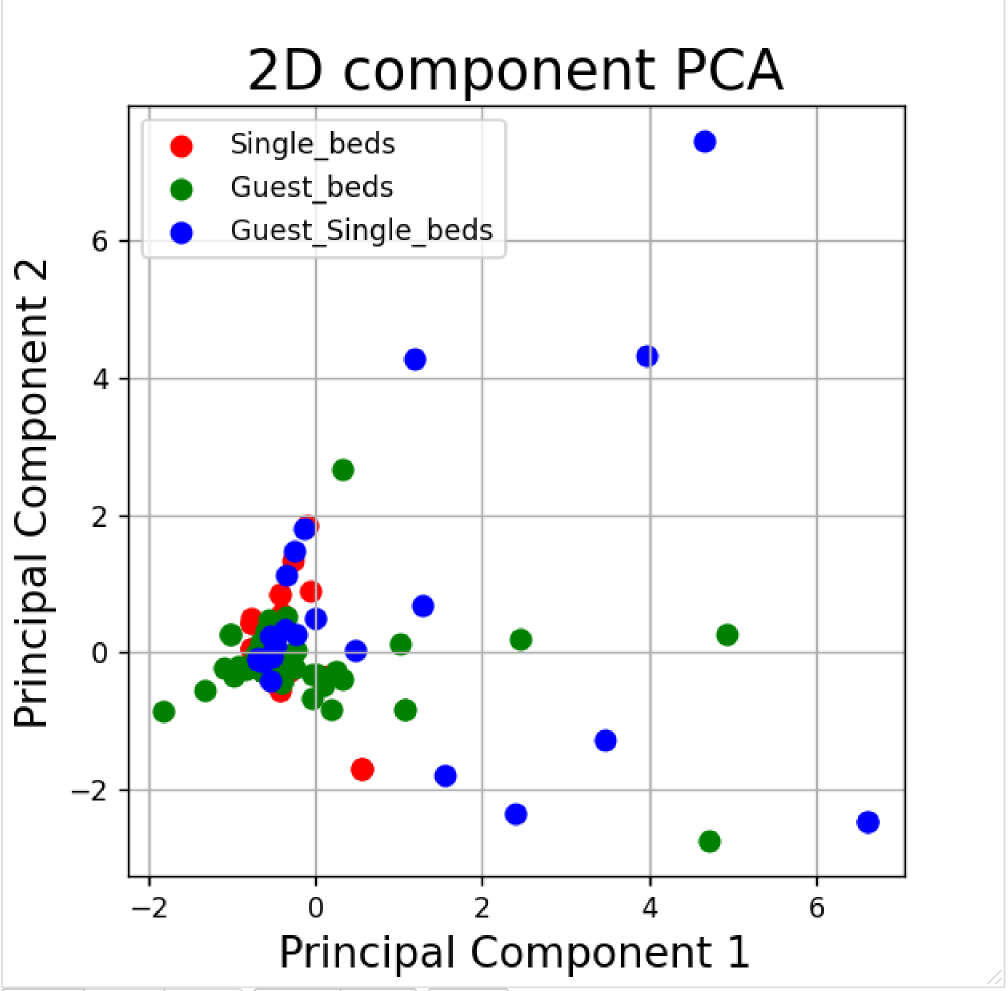
In [58]:

```
fig = plt.figure(figsize = (5,5))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title ( '2D component PCA', fontsize = 20)

targets = ['Single_beds', 'Guest_beds','Guest_Single_beds']
colors = ['r','g','b']
for target, color in zip(targets, colors):
    indicesToKeep = finalDf['type'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)

ax.legend(targets)
ax.grid()
```

Figure 6



Conclusion

Plot the distribution of diameter

In [47]:

```
import seaborn as sns
```

In [48]:

```
sgb_cleaning.head()
```

Out[48]:

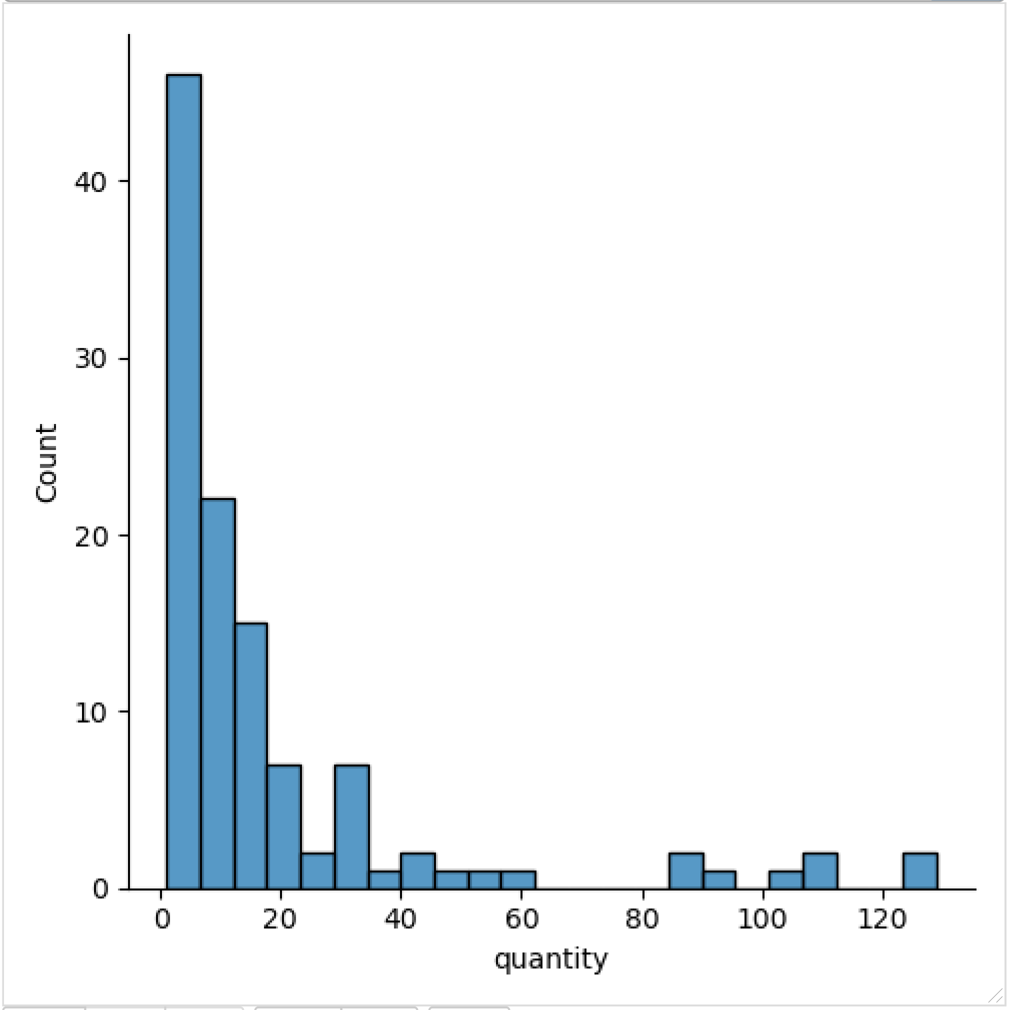
	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLE
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	
4	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	

- Histogram distribution of total quantity in each Part No.

In [49]:

```
# Histogram distribution of total quantity in each part No.
chart = sns.displot(sgb_cleaning['quantity'], kde=False)
chart
```

Figure 2



Out[49]:

```
<seaborn.axisgrid.FacetGrid at 0x7f808f9545e0>
```

From the histogram chart above, most parts are used below 20 quantity. But there are some parts are frequent used with over 80 quantity usage.

Bart plots: component quantity of types of IKEA beds

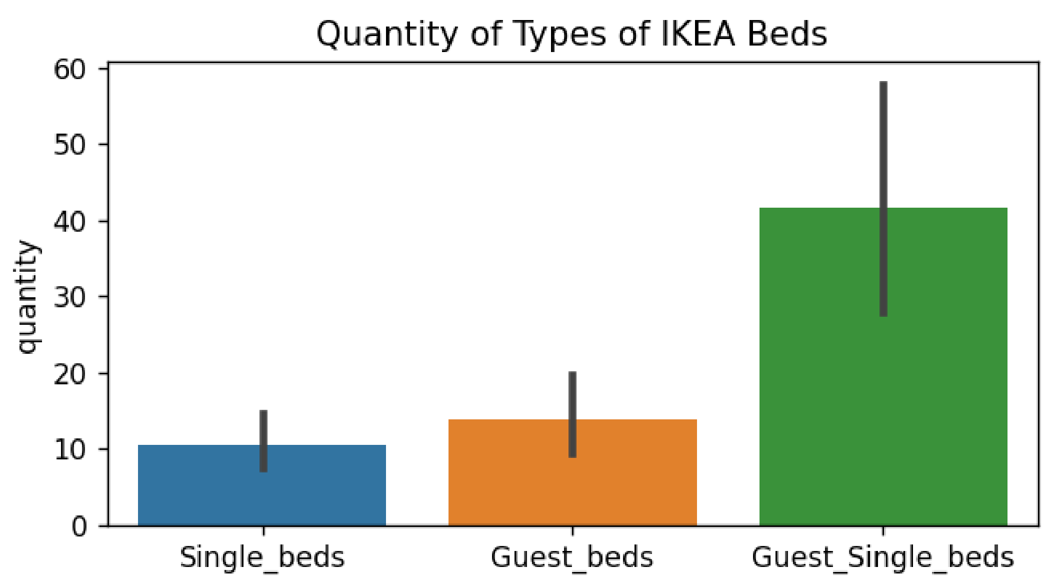
In [55]:

```
# Show the type's distribute of bar plots
plt.figure(figsize=(6,3))

# Add title and axis
plt.title('Quantity of Types of IKEA Beds')
plt.ylabel('Quantity')

# Bar chart showing diameter for each screw type
sns.barplot(x=sgb_cleaning['type'], y=sgb_cleaning['quantity'])
```

Figure 5



Out[55]:

```
<AxesSubplot:title={'center':'Quantity of Types of IKEA Beds'}, xlabel
='type', ylabel='quantity'>
```

From the bar chart, we can see that the quantity of guest_single dual components is over 40 in total, meaning that many parts are used in two bed types. in contrary, single beds' part quantity is the lowest, with 10 in total usage.

- Lineplot for all product series of IKEA bed

In [51]:

```
sgb_cleaning.head()
```

Out[51]:

	Part_No	type	quantity	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLE
0	100006	Single_beds	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	100087	Single_beds	4.0	0.0	4.0	0.0	0.0	0.0	0.0	
2	100092	Single_beds	1.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	100224	Single_beds	18.0	0.0	0.0	18.0	0.0	0.0	0.0	
4	100349	Single_beds	12.0	0.0	0.0	0.0	12.0	0.0	0.0	

In [52]:

```
# Slice the columns of all product series, which start from 'FJELLSE' till 'UT'
products = sgb_cleaning.iloc[:,3:]
products
```

Out[52]:

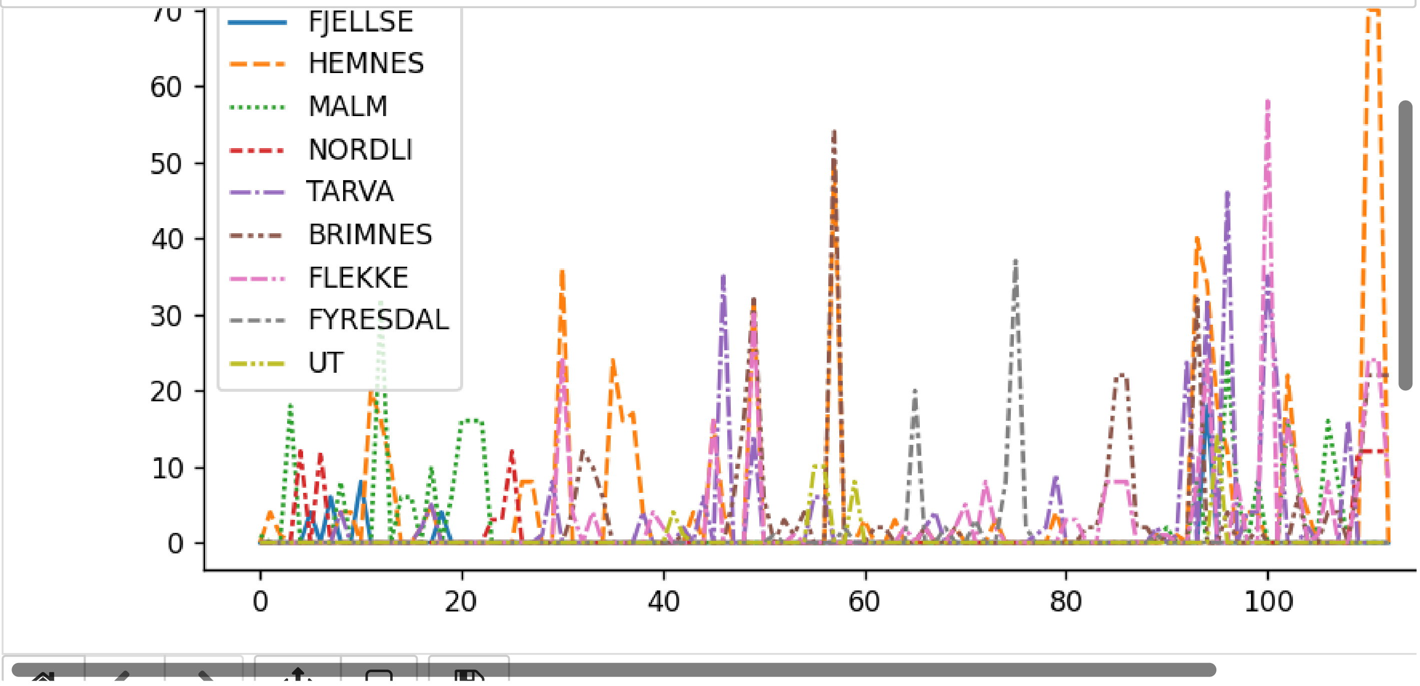
	FJELLSE	HEMNES	MALM	NORDLI	TARVA	BRIMNES	FLEKKE	FYRESDAL	UT
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	18.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	12.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
108	0.0	0.0	0.0	0.0	16.0	0.0	0.0	0.0	0.0
109	0.0	0.0	0.0	12.0	0.0	8.0	8.0	0.0	0.0
110	0.0	70.0	0.0	12.0	0.0	22.0	24.0	0.0	0.0
111	0.0	70.0	0.0	12.0	0.0	22.0	24.0	0.0	0.0
112	0.0	0.0	0.0	12.0	0.0	22.0	8.0	0.0	0.0

113 rows × 9 columns

In [59]:

```
# Set the width and height of the figure
plt.figure(figsize=(8,3.8))

# Line char showing how single and bed products differ
sns.lineplot(data=products)
```



From the lineplot, we can see that HEMNES has high quantity of components, with some parts may over 50 quantities. In some parts, HEMNES even amost 70 quantity.

In sum, most components are used under 10 quantity in total, but there are still some parts have very high usage with over 80 quantity. In addition, there are 24 identical parts are used in guest and single beds, and those identical parts accounts the most percentage of total parts quantity usage. For single beds, the percentage of unique components (in terms of their quantity usage) is 22.57%. For guest beds, the percentage of unique components (in terms of their quantity usage) is 45.84%.

In [ ]: