In [30]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from half_normal_plotting import HalfNormPlot
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

1)

- a. There is definitely one factor and that is the trajectory at which the artillery shell is being fired. The type of target could be another factor differentiated by the target material, or it could be a block. If the experiment is to see what kind of damage each shell/salvo of shells does to each type of target then it would be a factor level I think. If the range facility only had those target types available or military protocol requires that all artillery tests evaluate efficacy against the designated target types then it would be a block.
- b. The possible treament combinations are as follows

Target Type	Firing Angle
Aluminum	45
Aluminum	60
Steel	45
Steel	60
Titanium	45
Titanium	60
Topsoil	45
Topsoil	60
Sand	45
Sand	60
Fuel Tank	45
Fuel Tank	60

- c. The experimental units can be two things in this experiment. It could be that the experimental units will be single artillery shells fired against each target type or it could be that the experimental unit will be salvos of artillery fired against each target type.

2)

- a. To completely test all of the possible combinations you need $2 * 3 * 3 * 2 * 2 * 2$, or 144 test runs

- b. To determine how many test runs are used to calculate the average corrosion resistance for each of the factor levels we can use the formula $\frac{TestRuns}{\#FactorLevels}$. Each factor in this experiment only has 2 or three factor levels so the number of test runs for each factor will either be $\frac{144}{2} = 72$ or $\frac{144}{3} = 48$

    - Bed Temp: 2 factor levels, 72 test runs
    - Tube Temp Below Bed: 3 factor levels, 48 test runs

- Particle Size: 3 factor levels, 48 test runs
- Environment: 2 factor levels, 72 test runs
- Particle Material: 2 factor levels, 72 test runs
- Tube Material: 2 factor levels, 72, test runs

- c. Assuming that the question is asking "If we investigated each factor level independently in a one-at-a-time fashion using the same number of experimental runs for each factor level identified in part b, how many test runs would we have to run?" We know that each factor level had 144 runs each in part b, so we can simply multiply 144 by the number of factors, 6, to get a total experiment size of 864 test runs. Comparing the two scenarios we can see the benefit of leveraging the techniques we're going to learn in this course to reduce experimental cost and decrease time to insight.

3)

There are four factors and 48 test runs in a 2 x 4 x 3 x 2 factorial design.

4)

The treatment design would be designated "3^4" and it will require 81 test runs for a complete replication.

5)

In [2]:
```
#make the dataframe to feed the Half Normal class
data = [23.0, -5.0, 1.5, 1.5, 10.0, 0.0, 0.5]
labels = [str(x) for x in data]

df = pd.DataFrame(data={'data':data, 'labels':labels})
df
```

Out[2]:

	data	labels
0	23.0	23.0
1	-5.0	-5.0
2	1.5	1.5
3	1.5	1.5
4	10.0	10.0
5	0.0	0.0
6	0.5	0.5

In [3]:
```
#instantiate the Half Normal class and show the columns calculated
half_norm = HalfNormPlot(data=df, data_col='data', label_col='label')
half_norm.half_norm_data
```

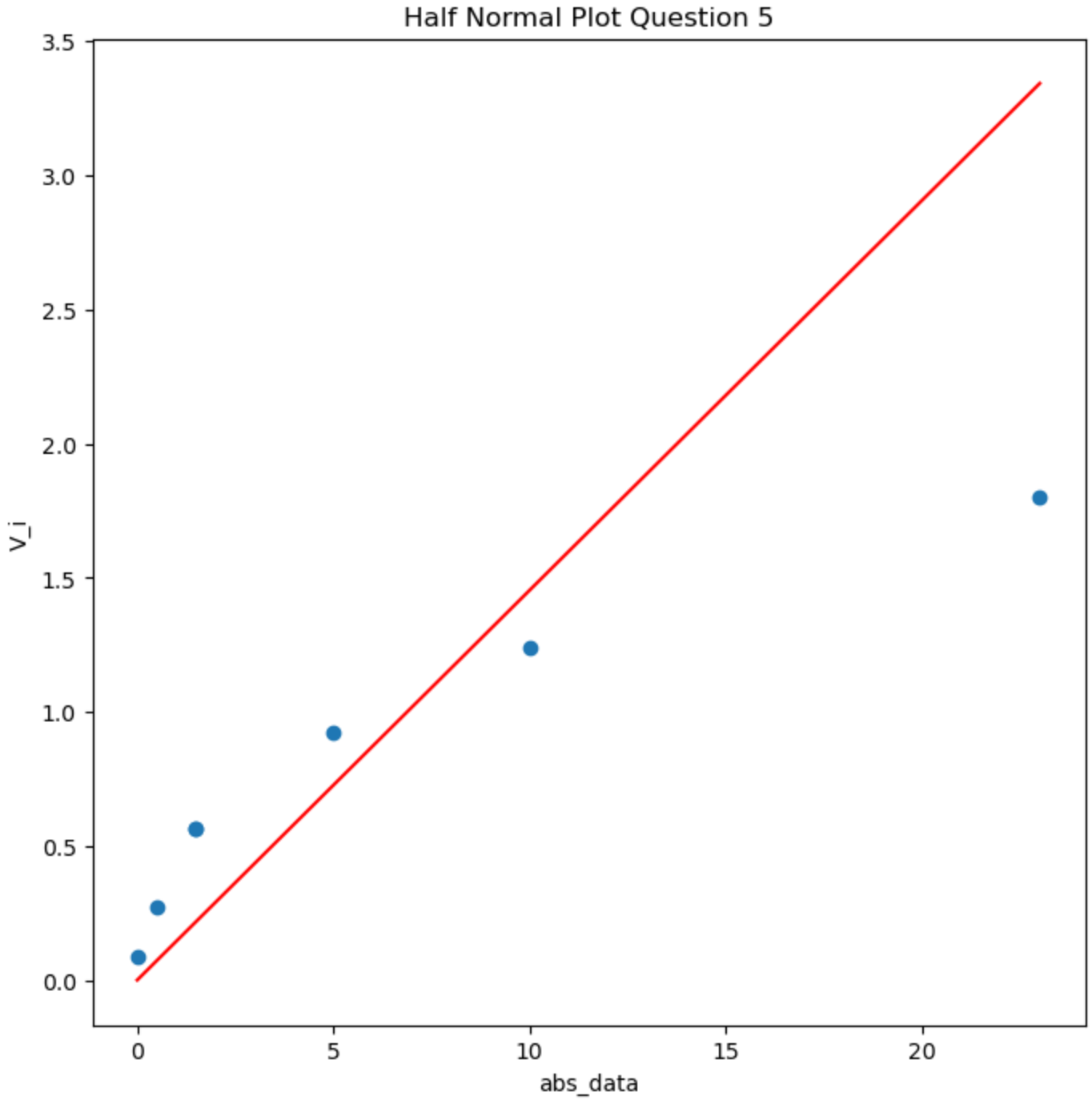
Out[3]:

	data	labels	abs_data	r_i	r_i*	p_i	v_i
5	0.0	0.0	0.0	1.0	0.071429	0.535714	0.089642
6	0.5	0.5	0.5	2.0	0.214286	0.607143	0.271880
2	1.5	1.5	1.5	3.5	0.428571	0.714286	0.565949
3	1.5	1.5	1.5	3.5	0.428571	0.714286	0.565949

	data	labels	abs_data	r_i	r_i*	p_i	v_i
1	-5.0	-5.0	5.0	5.0	0.642857	0.821429	0.920823
4	10.0	10.0	10.0	6.0	0.785714	0.892857	1.241867
0	23.0	23.0	23.0	7.0	0.928571	0.964286	1.802743

In [4]:
```
#create the half normal plot
half_norm.plot_half_norm(title="Half Normal Plot Question 5", data_percent=.8)
```

Out[4]:
```
<AxesSubplot: title={'center': 'Half Normal Plot Question 5'}, xlabel='abs_data', ylabel
='V_i'>
```



6)

In [5]:
```
#make data list and label list, turn into df
data = [38.7, 4.0, -54.6, 20.8, -13.3, 3.1, 55.9, -26.0, -13.9, -39.3, 1.1, 37.9, 28.2, 3.
labels = [str(x) for x in data]
```

```
df = pd.DataFrame(data={'data':data, 'label':labels})
df
```

Out[5]:

	data	label
0	38.7	38.7
1	4.0	4.0
2	-54.6	-54.6
3	20.8	20.8
4	-13.3	-13.3
5	3.1	3.1
6	55.9	55.9
7	-26.0	-26.0
8	-13.9	-13.9
9	-39.3	-39.3
10	1.1	1.1
11	37.9	37.9
12	28.2	28.2
13	3.1	3.1
14	15.0	15.0

In [6]:

```
#instantate new Half Normal class and populate the table
half_norm2 = HalfNormPlot(data=df, data_col='data', label_col='label')
half_norm2.half_norm_data
```

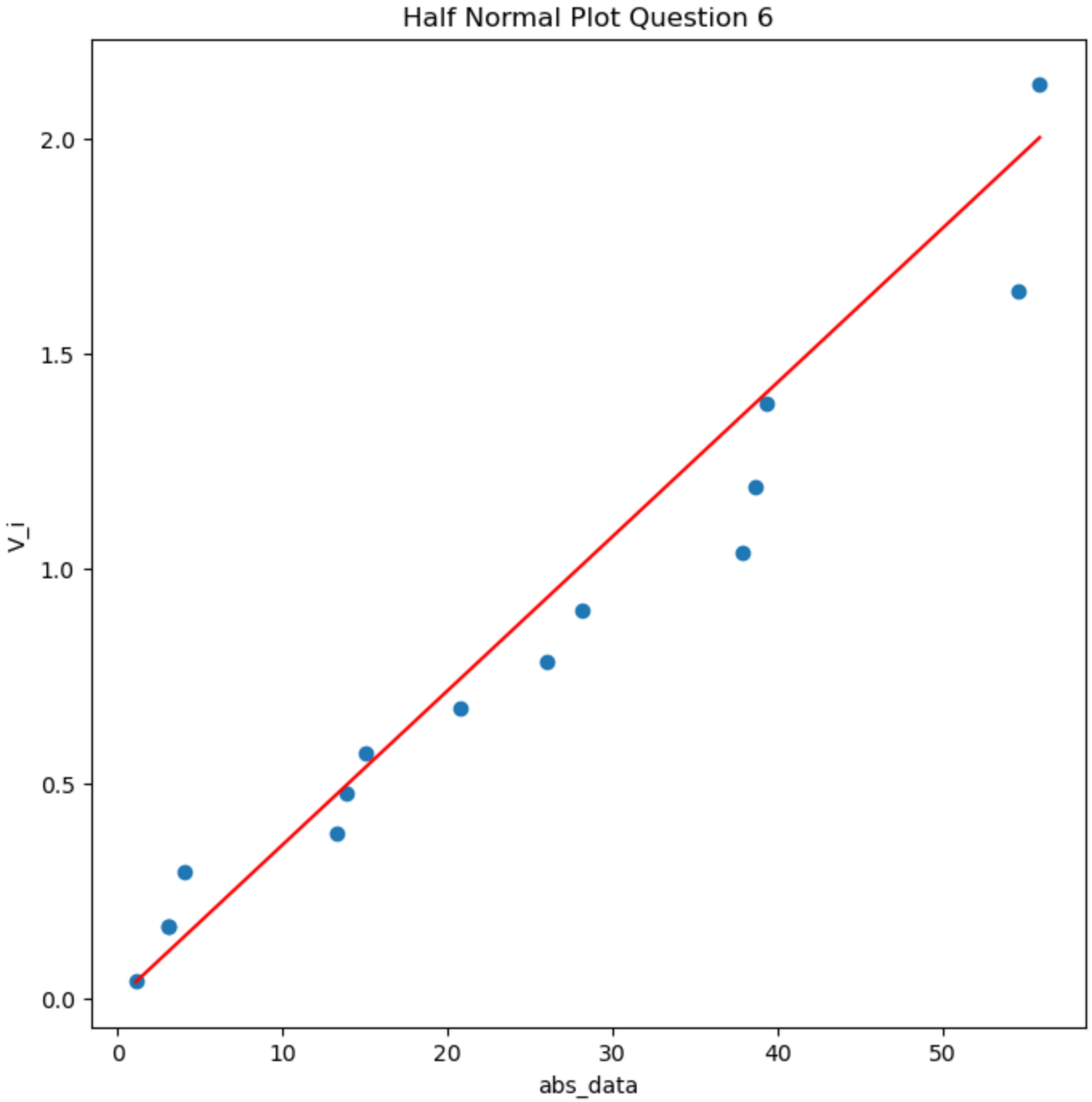
Out[6]:

	data	label	abs_data	r_i	r_i*	p_i	v_i
10	1.1	1.1	1.1	1.0	0.033333	0.516667	0.041789
5	3.1	3.1	3.1	2.5	0.133333	0.566667	0.167894
13	3.1	3.1	3.1	2.5	0.133333	0.566667	0.167894
1	4.0	4.0	4.0	4.0	0.233333	0.616667	0.296738
4	-13.3	-13.3	13.3	5.0	0.300000	0.650000	0.385320
8	-13.9	-13.9	13.9	6.0	0.366667	0.683333	0.477040
14	15.0	15.0	15.0	7.0	0.433333	0.716667	0.572968
3	20.8	20.8	20.8	8.0	0.500000	0.750000	0.674490
7	-26.0	-26.0	26.0	9.0	0.566667	0.783333	0.783500
12	28.2	28.2	28.2	10.0	0.633333	0.816667	0.902735
11	37.9	37.9	37.9	11.0	0.700000	0.850000	1.036433
0	38.7	38.7	38.7	12.0	0.766667	0.883333	1.191816
9	-39.3	-39.3	39.3	13.0	0.833333	0.916667	1.382994
2	-54.6	-54.6	54.6	14.0	0.900000	0.950000	1.644854

	data	label	abs_data	r_i	r_i*	p_i	v_i
6	55.9	55.9	55.9	15.0	0.966667	0.983333	2.128045

In [7]:
```
half_norm2.plot_half_norm(title="Half Normal Plot Question 6", data_percent=1)
```

Out[7]:
```
<AxesSubplot: title={'center': 'Half Normal Plot Question 6'}, xlabel='abs_data', ylabel
='V_i'>
```



7)

Using the half normal plot created above we can get two different estimates of sigma, one we can get by taking the absolute value of X whose V_i is closest to one, and the other we can get by calculating a regression line through all the points and taking the slope as the estimate of sigma.

In [8]:
```
#use class method to get estimate of sigma closest to one
half_norm2.sigma_closest_to_one()
```

Out[8]:
```
37.9
```

In [9]:
```
#use regression method to get estimate of sigma hat
half_norm2.sigma_from_regression()
```

Out[9]:
```
27.914162202401794
```

The estimate of sigma using the absolute value of X_i whose value of V_i is closest to one is 37.9, and the estimate of sigma using the slope of a regression line fitted to all points is 27.9.

8)

The sample standard deviation of numbers in question six is:

In [10]:
```
df.data.std()
```

Out[10]:
```
30.324833450647738
```

9)

In [39]:
```
data = [
(75, .03, 7.5),
(125, .03, 6.4),
(75, .06, 9.6),
(125, .06, 8.3)
]

df = pd.DataFrame(data=data, columns=['water', 'surfactant', 'volume'])
df['water x surfactant'] = df.water * df.surfactant
df = df.sort_values(by=['water','surfactant','water x surfactant'])
df = df[['water','surfactant','water x surfactant','volume']]
df
```

Out[39]:

	water	surfactant	water x surfactant	volume
0	75	0.03	2.25	7.5
2	75	0.06	4.50	9.6
1	125	0.03	3.75	6.4
3	125	0.06	7.50	8.3

In [40]:
```
#encode treatment design
df['water H/L'] = df.water.apply(lambda x: -1 if x == 75 else 1)
df['surfactant H/L'] = df.surfactant.apply(lambda x: -1 if x == .03 else 1)
df['water x surfactant H/L'] = df['water H/L'] * df['surfactant H/L']
df
```

Out[40]:

	water	surfactant	water x surfactant	volume	water H/L	surfactant H/L	water x surfactant H/L
0	75	0.03	2.25	7.5	-1	-1	1
2	75	0.06	4.50	9.6	-1	1	-1
1	125	0.03	3.75	6.4	1	-1	-1
3	125	0.06	7.50	8.3	1	1	1

In [45]:

```
water_effect = sum(df['volume'] * df['water H/L'])/2
surfactant_effect = sum(df['volume'] * df['surfactant H/L'])/2
water_x_surf_effect = sum(df['volume'] * df['water x surfactant H/L'])/2
print(f"Water effect size is {water_effect:.03f} \nSurfactant effect is {surfactant_effect
```

```
Water effect size is -1.200
Surfactant effect is 2.000
Interaction effect is -0.100
```