In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from half_normal_plotting import HalfNormPlot
from effects_table import yates_step, generate_standard_table, Experiment2N
from statsmodels.formula.api import ols
import statsmodels.api as sm
from scipy.stats import t
```

1)

In [2]:

```
#read in data
data = pd.read_excel("Assignment 3 Data.xlsx", sheet_name="question_1")
data['Block'] = data['Block'].apply(lambda x: "Block 1" if x == 1 else "Block 2")
data
```

Out[2]:

	Block	A	B	C	y
0	Block 1	0	0	0	3
1	Block 1	1	0	0	6
2	Block 1	0	1	0	8
3	Block 1	1	1	0	10
4	Block 1	0	0	1	4
5	Block 1	1	0	1	6
6	Block 1	0	1	1	11
7	Block 1	1	1	1	9
8	Block 2	0	0	0	4
9	Block 2	1	0	0	4
10	Block 2	0	1	0	7
11	Block 2	1	1	0	7
12	Block 2	0	0	1	4
13	Block 2	1	0	1	3
14	Block 2	0	1	1	10
15	Block 2	1	1	1	8

In [3]:

```
q1_lm = ols('y ~ Block + A*B*C', data=data).fit()
anova = sm.stats.anova_lm(q1_lm)
anova['sd_effect'] = anova['mean_sq']**.5
anova
```

Out[3]:

	df	sum_sq	mean_sq	F	PR(>F)	sd_effect
Block	1.0	6.25	6.250000	6.481481	0.038334	2.500000
A	1.0	0.25	0.250000	0.259259	0.626283	0.500000
B	1.0	81.00	81.000000	84.000000	0.000038	9.000000
A:B	1.0	2.25	2.250000	2.333333	0.170471	1.500000

	df	sum_sq	mean_sq	F	PR(>F)	sd_effect
C	1.0	2.25	2.250000	2.333333	0.170471	1.500000
A:C	1.0	4.00	4.000000	4.148148	0.081107	2.000000
B:C	1.0	2.25	2.250000	2.333333	0.170471	1.500000
A:B:C	1.0	1.00	1.000000	1.037037	0.342410	1.000000
Residual	7.0	6.75	0.964286	NaN	NaN	0.981981

Based on the results of the Anova analysis we can see that only the B main effect is significant at the 5% level. Looking at the data and keeping the goal of minimizing y in mind we can see that whenever B is at the high level the response variable tends to be higher. Lets formalize this insight by looking at this as a 2^1 experiment with only factor B as the main effect and the other test runs considered replications of the experiment.

In [4]:

```
data_B = data[['B','y']]
data_B
```

Out[4]:

	B	y
0	0	3
1	0	6
2	1	8
3	1	10
4	0	4
5	0	6
6	1	11
7	1	9
8	0	4
9	0	4
10	1	7
11	1	7
12	0	4
13	0	3
14	1	10
15	1	8

In [5]:

```
data_B.groupby(['B']).agg(['mean','count'])
```

Out[5]:

	y	
	mean	count
B		
0	4.25	8
1	8.75	8

In [6]:
```
#calculate the LSD using t distribution and variance of difference in two means
LSD = t.ppf(1-.025, 7) * (anova.loc['Residual','mean_sq'] * (1/8 + 1/8))
print(f"LSD for this experiment is {t.ppf(1-.025, 7):.04f}*({anova.loc['Residual','mean_sc
```

```
LSD for this experiment is 2.3646*(0.9643*(1/8+1/8))=0.5700
```

We can see that for both blocks the difference in means is greater than the LSD so we can conclude that B does have a large and positive impact on y. Knowing this and our goal of minimizing y the recommendation from this experiment is to leave all treatment factors at the low level.

2)

a.

In [7]:
```
#read in data, and sort into standard order
q2_data = pd.read_excel("Assignment 3 Data.xlsx", sheet_name="question_2")

#make standard table of effects to sort data correctly
standard_table = generate_standard_table(n=5, columns=['A','B','C','D','E']).set_index(['A

#index both datasets on A,B,C,D,E
q2_data = q2_data.set_index(['A','B','C','D','E'])

#merge data on standard table format and reset index
q2_fixed = standard_table.merge(q2_data, left_index=True, right_index=True)
q2_fixed = q2_fixed.reset_index()
q2_fixed
```

Out[7]:

	A	B	C	D	E	y
0	0	0	0	0	0	23
1	1	0	0	0	0	15
2	0	1	0	0	0	27
3	1	1	0	0	0	27
4	0	0	1	0	0	24
5	1	0	1	0	0	20
6	0	1	1	0	0	30
7	1	1	1	0	0	31
8	0	0	0	1	0	23
9	1	0	0	1	0	24
10	0	1	0	1	0	28
11	1	1	0	1	0	33
12	0	0	1	1	0	29
13	1	0	1	1	0	26
14	0	1	1	1	0	31
15	1	1	1	1	0	33
16	0	0	0	0	1	20
17	1	0	0	0	1	18

	A	B	C	D	E	y
18	0	1	0	0	1	25
19	1	1	0	0	1	32
20	0	0	1	0	1	21
21	1	0	1	0	1	25
22	0	1	1	0	1	24
23	1	1	1	0	1	34
24	0	0	0	1	1	14
25	1	0	0	1	1	22
26	0	1	0	1	1	24
27	1	1	0	1	1	36
28	0	0	1	1	1	22
29	1	0	1	1	1	31
30	0	1	1	1	1	25
31	1	1	1	1	1	36

In [8]:

```
q2_lm = ols('y ~ A + B + C + D + E + A:B + A:C + A:D + A:E + B:C + B:D + B:E + C:D + C:E 
q2_anova = sm.stats.anova_lm(q2_lm)
q2_anova['significant'] = q2_anova['PR(>F)'].apply(lambda x: True if x < .05 else False)
q2_anova['std_effects'] = q2_anova['mean_sq']**.5
q2_anova
```

Out[8]:

	df	sum_sq	mean_sq	F	PR(>F)	significant	std_effects
A	1.0	87.78125	87.781250	26.130233	1.045294e-04	True	9.369165
B	1.0	442.53125	442.531250	131.730233	3.911175e-09	True	21.036427
C	1.0	81.28125	81.281250	24.195349	1.541882e-04	True	9.015611
D	1.0	52.53125	52.531250	15.637209	1.135937e-03	True	7.247845
E	1.0	7.03125	7.031250	2.093023	1.672800e-01	False	2.651650
A:B	1.0	57.78125	57.781250	17.200000	7.576240e-04	True	7.601398
A:C	1.0	1.53125	1.531250	0.455814	5.092267e-01	False	1.237437
A:D	1.0	42.78125	42.781250	12.734884	2.563782e-03	True	6.540738
A:E	1.0	132.03125	132.031250	39.302326	1.119183e-05	True	11.490485
B:C	1.0	22.78125	22.781250	6.781395	1.917995e-02	True	4.772971
B:D	1.0	2.53125	2.531250	0.753488	3.982071e-01	False	1.590990
B:E	1.0	1.53125	1.531250	0.455814	5.092267e-01	False	1.237437
C:D	1.0	1.53125	1.531250	0.455814	5.092267e-01	False	1.237437
C:E	1.0	0.28125	0.281250	0.083721	7.760325e-01	False	0.530330
D:E	1.0	11.28125	11.281250	3.358140	8.555052e-02	False	3.358757
Residual	16.0	53.75000	3.359375	NaN	NaN	False	1.832860

In [9]:

```
q2_anova[q2_anova['significant']==True]
```

Out[9]:

	df	sum_sq	mean_sq	F	PR(>F)	significant	std_effects
A	1.0	87.78125	87.78125	26.130233	1.045294e-04	True	9.369165
B	1.0	442.53125	442.53125	131.730233	3.911175e-09	True	21.036427
C	1.0	81.28125	81.28125	24.195349	1.541882e-04	True	9.015611
D	1.0	52.53125	52.53125	15.637209	1.135937e-03	True	7.247845
A:B	1.0	57.78125	57.78125	17.200000	7.576240e-04	True	7.601398
A:D	1.0	42.78125	42.78125	12.734884	2.563782e-03	True	6.540738
A:E	1.0	132.03125	132.03125	39.302326	1.119183e-05	True	11.490485
B:C	1.0	22.78125	22.78125	6.781395	1.917995e-02	True	4.772971

An estimate of the experimental error is 1.832 with 16 degrees of freedom, and the main effects A, B, C, D are significant with interaction effects $A*B,\ \ A*D,\ \ A*E,\ \ B*C$ significant at the 5% alpha level.

In [10]:

```
def encode_effect(X):
    A, B, C, D, E = X
    effect = []
    if A == 1:
        effect.append('a')
    if B == 1:
        effect.append('b')
    if C == 1:
        effect.append('c')
    if D == 1:
        effect.append('d')
    if E == 1:
        effect.append('e')
    if len(effect) > 0:
        return ''.join(effect)
    else:
        return '(1)'
```

In [11]:

```
q2_yates = q2_fixed.copy()
q2_yates['step_1'] = yates_step(q2_yates['y'].tolist())
q2_yates['step_2'] = yates_step(q2_yates['step_1'].tolist())
q2_yates['step_3'] = yates_step(q2_yates['step_2'].tolist())
q2_yates['step_4'] = yates_step(q2_yates['step_3'].tolist())
q2_yates['step_5'] = yates_step(q2_yates['step_4'].tolist())
q2_yates['std_effect'] = q2_yates['step_5'] / (len(q2_data)**.5)
q2_yates['effect'] = q2_yates[['A','B','C','D','E']].apply(encode_effect, axis=1)
q2_yates
```

Out[11]:

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect
0	0	0	0	0	0	23	38	92	197	424	833	147.254987	(1)
1	1	0	0	0	0	15	54	105	227	409	53	9.369165	a
2	0	1	0	0	0	27	44	108	199	-6	119	21.036427	b
3	1	1	0	0	0	27	61	119	210	59	43	7.601398	ab
4	0	0	1	0	0	24	47	95	-11	56	51	9.015611	c
5	1	0	1	0	0	20	61	104	5	63	7	1.237437	ac

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect
6	0	1	1	0	0	30	55	96	19	22	-27	-4.772971	bc
7	1	1	1	0	0	31	64	114	40	21	-7	-1.237437	abc
8	0	0	0	1	0	23	38	-8	33	24	41	7.247845	d
9	1	0	0	1	0	24	57	-3	23	27	37	6.540738	ad
10	0	1	0	1	0	28	46	6	31	-2	-9	-1.590990	bd
11	1	1	0	1	0	33	58	-1	32	9	-13	-2.298097	abd
12	0	0	1	1	0	29	36	5	13	-4	7	1.237437	cd
13	1	0	1	1	0	26	60	14	9	-23	-21	-3.712311	acd
14	0	1	1	1	0	31	53	20	15	-2	-15	-2.651650	bcd
15	1	1	1	1	0	33	61	20	6	-5	5	0.883883	abcd
16	0	0	0	0	1	20	-8	16	13	30	-15	-2.651650	e
17	1	0	0	0	1	18	0	17	11	11	65	11.490485	ae
18	0	1	0	0	1	25	-4	14	9	16	7	1.237437	be
19	1	1	0	0	1	32	1	9	18	21	-1	-0.176777	abe
20	0	0	1	0	1	21	1	19	5	-10	3	0.530330	ce
21	1	0	1	0	1	25	5	12	-7	1	11	1.944544	ace
22	0	1	1	0	1	24	-3	24	9	-4	-19	-3.358757	bce
23	1	1	1	0	1	34	2	8	0	-9	-3	-0.530330	abce
24	0	0	0	1	1	14	-2	8	1	-2	-19	-3.358757	de
25	1	0	0	1	1	22	7	5	-5	9	5	0.883883	ade
26	0	1	0	1	1	24	4	4	-7	-12	11	1.944544	bde
27	1	1	0	1	1	36	10	5	-16	-9	-5	-0.883883	abde
28	0	0	1	1	1	22	8	9	-3	-6	11	1.944544	cde
29	1	0	1	1	1	31	12	6	1	-9	3	0.530330	acde
30	0	1	1	1	1	25	9	4	-3	4	-3	-0.530330	bcde
31	1	1	1	1	1	36	11	2	-2	1	-3	-0.530330	abcde

Lets validate the above analysis by using contrast analysis to determine the effects.

In [12]:

```
#make the experiment and validate the standard order
exp2 = Experiment2N(factors=['A','B','C','D','E'])
exp2_table = exp2.generate_table()
exp2_table
```

Out[12]:

	A	B	C	D	E	A_contrast	B_contrast	C_contrast	D_contrast	E_contrast	...	BCD_contrast	BCE_contrast	BDE_
0	0	0	0	0	0	-1	-1	-1	-1	-1	...	-1	-1	
1	1	0	0	0	0	1	-1	-1	-1	-1	...	-1	-1	
2	0	1	0	0	0	-1	1	-1	-1	-1	...	1	1	
3	1	1	0	0	0	1	1	-1	-1	-1	...	1	1	

	A	B	C	D	E	A_contrast	B_contrast	C_contrast	D_contrast	E_contrast	...	BCD_contrast	BCE_contrast	BDE_
4	0	0	1	0	0	-1	-1	1	-1	-1	...	1	1	
5	1	0	1	0	0	1	-1	1	-1	-1	...	1	1	
6	0	1	1	0	0	-1	1	1	-1	-1	...	-1	-1	
7	1	1	1	0	0	1	1	1	-1	-1	...	-1	-1	
8	0	0	0	1	0	-1	-1	-1	1	-1	...	1	-1	
9	1	0	0	1	0	1	-1	-1	1	-1	...	1	-1	
10	0	1	0	1	0	-1	1	-1	1	-1	...	-1	1	
11	1	1	0	1	0	1	1	-1	1	-1	...	-1	1	
12	0	0	1	1	0	-1	-1	1	1	-1	...	-1	1	
13	1	0	1	1	0	1	-1	1	1	-1	...	-1	1	
14	0	1	1	1	0	-1	1	1	1	-1	...	1	-1	
15	1	1	1	1	0	1	1	1	1	-1	...	1	-1	
16	0	0	0	0	1	-1	-1	-1	-1	1	...	-1	1	
17	1	0	0	0	1	1	-1	-1	-1	1	...	-1	1	
18	0	1	0	0	1	-1	1	-1	-1	1	...	1	-1	
19	1	1	0	0	1	1	1	-1	-1	1	...	1	-1	
20	0	0	1	0	1	-1	-1	1	-1	1	...	1	-1	
21	1	0	1	0	1	1	-1	1	-1	1	...	1	-1	
22	0	1	1	0	1	-1	1	1	-1	1	...	-1	1	
23	1	1	1	0	1	1	1	1	-1	1	...	-1	1	
24	0	0	0	1	1	-1	-1	-1	1	1	...	1	1	
25	1	0	0	1	1	1	-1	-1	1	1	...	1	1	
26	0	1	0	1	1	-1	1	-1	1	1	...	-1	-1	
27	1	1	0	1	1	1	1	-1	1	1	...	-1	-1	
28	0	0	1	1	1	-1	-1	1	1	1	...	-1	-1	
29	1	0	1	1	1	1	-1	1	1	1	...	-1	-1	
30	0	1	1	1	1	-1	1	1	1	1	...	1	1	
31	1	1	1	1	1	1	1	1	1	1	...	1	1	

32 rows × 36 columns

Now that we have our contrast table lets calculate the effects and compare to before

In [13]:

```
#subset to main effects columns to avoid index errors
exp2_table = exp2_table[['A','B','C','D','E']]
exp2_table['y'] = q2_fixed['y'].values
exp2_effects = exp2.calculate_effects(exp2_table, response_col='y')
exp2_effects
```

```
C:\Users\aburtnerabt\AppData\Local\Temp\ipykernel_15848\2448447328.py:3: SettingWithCopyWa
rning:
```

```
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_gu
ide/indexing.html#returning-a-view-versus-a-copy
  exp2_table['y'] = q2_fixed['y'].values
```

Out[13]:

	A_effect	B_effect	C_effect	D_effect	E_effect	AB_effect	AC_effect	AD_effect	AE_effect	BC_e
effect	53.000000	119.000000	51.000000	41.000000	-15.00000	43.000000	7.000000	37.000000	65.000000	-27.0(
std_effect	9.369165	21.036427	9.015611	7.247845	-2.65165	7.601398	1.237437	6.540738	11.490485	-4.77

2 rows × 31 columns

In [14]:

```
#lets iterate over both and verify that they are the same
contrast_comparison = exp2_effects.T.drop('effect', axis=1)
yates_comparison = q2_yates[['std_effect','effect']]
yates_comparison.loc[:,'effect'] = yates_comparison['effect'].apply(lambda x: f"{x.upper()
yates_comparison.set_index('effect', inplace=True)
compare_table = contrast_comparison.merge(yates_comparison, how='inner', left_index=True,
compare_table = compare_table.rename(columns={"std_effect_x":"std_effect_contrast", "std_e
compare_table
```

```
C:\Users\aburtnerabt\AppData\Local\Temp\ipykernel_15848\2893471803.py:4: SettingWithCopyWa
rning:
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_gu
ide/indexing.html#returning-a-view-versus-a-copy
  yates_comparison.loc[:,'effect'] = yates_comparison['effect'].apply(lambda x: f"{x.upper
()}_effect")
```

Out[14]:

	std_effect_contrast	std_effect_yates
A_effect	9.369165	9.369165
B_effect	21.036427	21.036427
C_effect	9.015611	9.015611
D_effect	7.247845	7.247845
E_effect	-2.651650	-2.651650
AB_effect	7.601398	7.601398
AC_effect	1.237437	1.237437
AD_effect	6.540738	6.540738
AE_effect	11.490485	11.490485
BC_effect	-4.772971	-4.772971
BD_effect	-1.590990	-1.590990
BE_effect	1.237437	1.237437
CD_effect	1.237437	1.237437
CE_effect	0.530330	0.530330
DE_effect	-3.358757	-3.358757
ABC_effect	-1.237437	-1.237437
ABD_effect	-2.298097	-2.298097

	std_effect_contrast	std_effect_yates
ABE_effect	-0.176777	-0.176777
ACD_effect	-3.712311	-3.712311
ACE_effect	1.944544	1.944544
ADE_effect	0.883883	0.883883
BCD_effect	-2.651650	-2.651650
BCE_effect	-3.358757	-3.358757
BDE_effect	1.944544	1.944544
CDE_effect	1.944544	1.944544
ABCD_effect	0.883883	0.883883
ABCE_effect	-0.530330	-0.530330
ABDE_effect	-0.883883	-0.883883
ACDE_effect	0.530330	0.530330
BCDE_effect	-0.530330	-0.530330
ABCDE_effect	-0.530330	-0.530330

And look at that! The Yates analysis and the Contrast analysis match up exactly!

In [15]:

```
q2_hn = HalfNormPlot(data=q2_yates.iloc[1:,:], data_col='std_effect', label_col='effect')
q2_hn.half_norm_data
```

```
C:\Users\aburtnerabt\Documents\K-State\Experimental Design\half_normal_plotting.py:45: Set
tingWithCopyWarning:
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_gu
ide/indexing.html#returning-a-view-versus-a-copy
  self.half_norm_data[self.abs_col] = self.half_norm_data[self.data_col].apply(lambda x: a
bs(x))
```

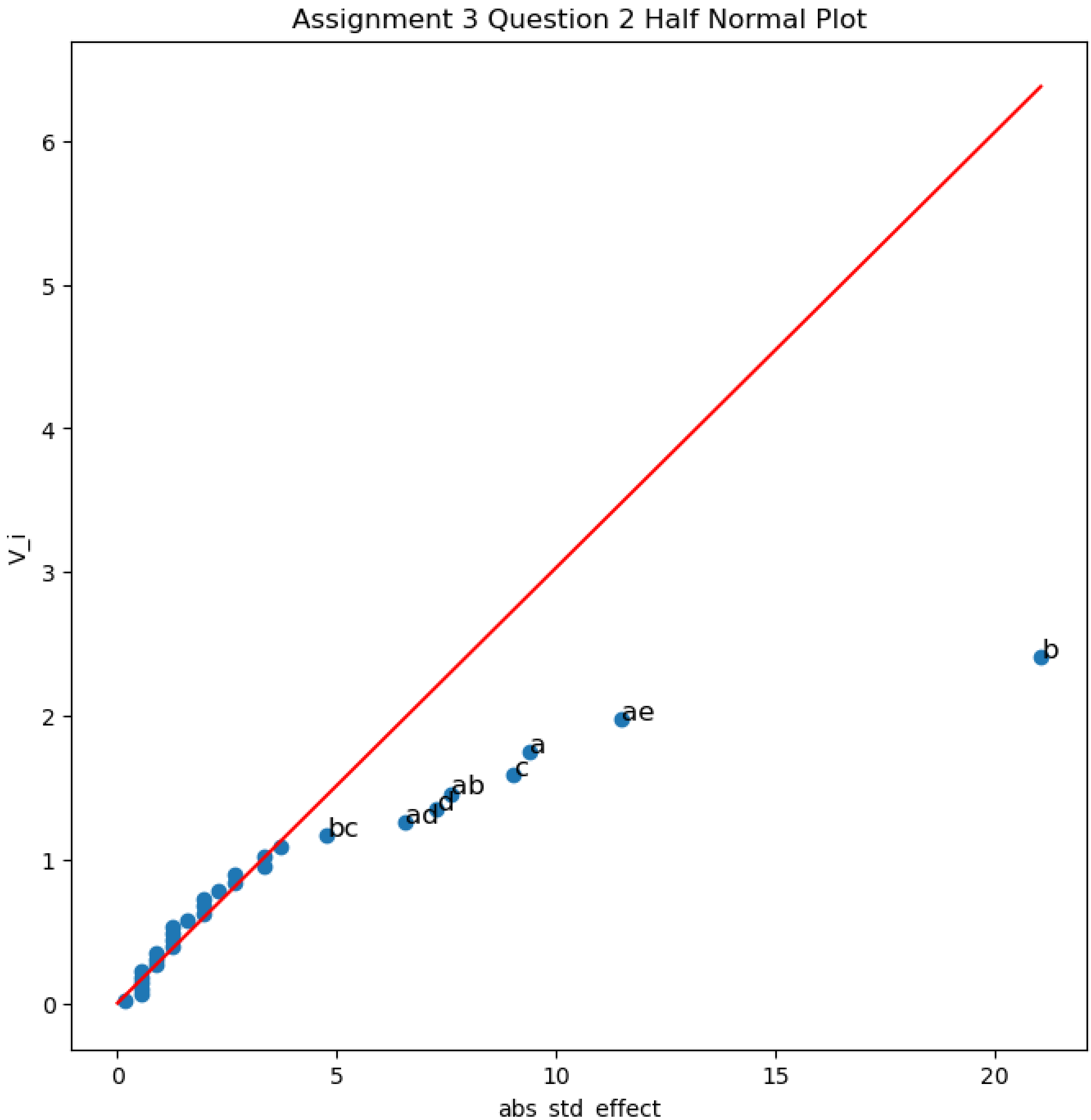
Out[15]:

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect	abs_std_effect	r_i	r_i*	
0	1	1	0	0	1	32	1	9	18	21	-1	-0.176777	abe	0.176777	1.0	0.016129	0.5
1	1	1	1	1	1	36	11	2	-2	1	-3	-0.530330	abcde	0.530330	2.0	0.048387	0.5
2	0	1	1	1	1	25	9	4	-3	4	-3	-0.530330	bcde	0.530330	3.0	0.080645	0.5
3	0	0	1	0	1	21	1	19	5	-10	3	0.530330	ce	0.530330	4.0	0.112903	0.5
4	1	1	1	0	1	34	2	8	0	-9	-3	-0.530330	abce	0.530330	5.0	0.145161	0.5
5	1	0	1	1	1	31	12	6	1	-9	3	0.530330	acde	0.530330	6.0	0.177419	0.5
6	1	0	0	1	1	22	7	5	-5	9	5	0.883883	ade	0.883883	7.0	0.209677	0.6
7	1	1	0	1	1	36	10	5	-16	-9	-5	-0.883883	abde	0.883883	8.0	0.241935	0.6
8	1	1	1	1	0	33	61	20	6	-5	5	0.883883	abcd	0.883883	9.0	0.274194	0.6
9	1	1	1	0	0	31	64	114	40	21	-7	-1.237437	abc	1.237437	10.0	0.306452	0.6
10	0	0	1	1	0	29	36	5	13	-4	7	1.237437	cd	1.237437	11.0	0.338710	0.6
11	1	0	1	0	0	20	61	104	5	63	7	1.237437	ac	1.237437	12.0	0.370968	0.6

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect	abs_std_effect	r_i	r_i*	
12	0	1	0	0	1	25	-4	14	9	16	7	1.237437	be	1.237437	13.0	0.403226	0.7
13	0	1	0	1	0	28	46	6	31	-2	-9	-1.590990	bd	1.590990	14.0	0.435484	0.7
14	0	1	0	1	1	24	4	4	-7	-12	11	1.944544	bde	1.944544	15.0	0.467742	0.7
15	1	0	1	0	1	25	5	12	-7	1	11	1.944544	ace	1.944544	16.0	0.500000	0.7
16	0	0	1	1	1	22	8	9	-3	-6	11	1.944544	cde	1.944544	17.0	0.532258	0.7
17	1	1	0	1	0	33	58	-1	32	9	-13	-2.298097	abd	2.298097	18.0	0.564516	0.7
18	0	0	0	0	1	20	-8	16	13	30	-15	-2.651650	e	2.651650	19.0	0.596774	0.7
19	0	1	1	1	0	31	53	20	15	-2	-15	-2.651650	bcd	2.651650	20.0	0.629032	0.8
20	0	1	1	0	1	24	-3	24	9	-4	-19	-3.358757	bce	3.358757	21.0	0.661290	0.8
21	0	0	0	1	1	14	-2	8	1	-2	-19	-3.358757	de	3.358757	22.0	0.693548	0.8
22	1	0	1	1	0	26	60	14	9	-23	-21	-3.712311	acd	3.712311	23.0	0.725806	0.8
23	0	1	1	0	0	30	55	96	19	22	-27	-4.772971	bc	4.772971	24.0	0.758065	0.8
24	1	0	0	1	0	24	57	-3	23	27	37	6.540738	ad	6.540738	25.0	0.790323	0.8
25	0	0	0	1	0	23	38	-8	33	24	41	7.247845	d	7.247845	26.0	0.822581	0.9
26	1	1	0	0	0	27	61	119	210	59	43	7.601398	ab	7.601398	27.0	0.854839	0.9
27	0	0	1	0	0	24	47	95	-11	56	51	9.015611	c	9.015611	28.0	0.887097	0.9
28	1	0	0	0	0	15	54	105	227	409	53	9.369165	a	9.369165	29.0	0.919355	0.9
29	1	0	0	0	1	18	0	17	11	11	65	11.490485	ae	11.490485	30.0	0.951613	0.9
30	0	1	0	0	0	27	44	108	199	-6	119	21.036427	b	21.036427	31.0	0.983871	0.9

In [16]:

```
q2_hn.plot_half_norm(title="Assignment 3 Question 2 Half Normal Plot", data_percent=.75,
```

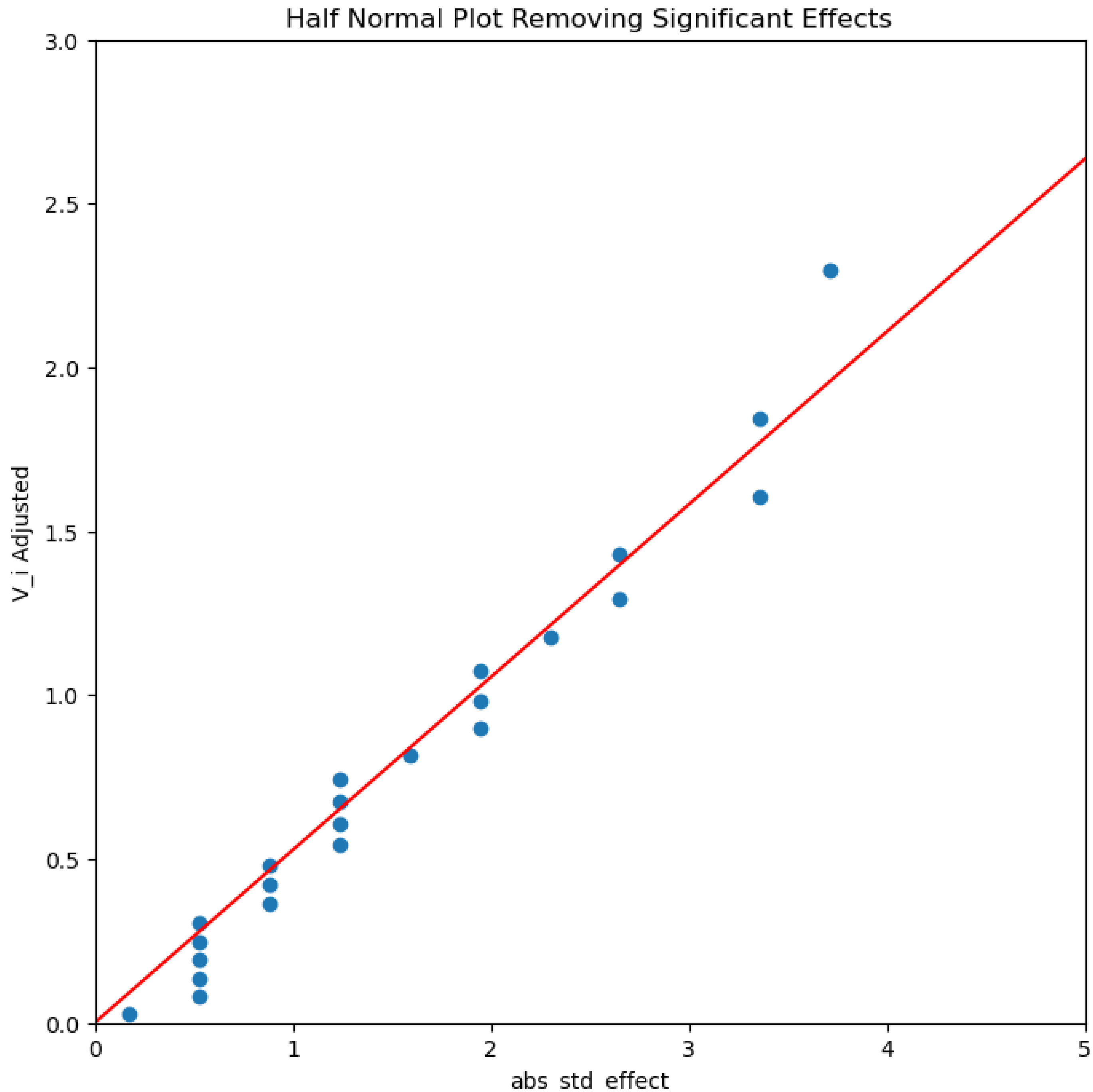


In [17]:

```
figure = q2_hn.plot_half_norm(title="Half Normal Plot Removing Significant Effects", num_a
figure.set_ylim(0,3)
figure.set_xlim(0,5)
```

Out[17]:

```
(0.0, 5.0)
```



Looking at the half-normal plot for all 31 factors it looks like A, B, C, D and $A*B, \quad A*D, \quad A*E, \quad B*C$ are significant. These results align with the ANOVA analysis.

In [18]:

```
#experimental error estimate
q2_hn.sigma_from_adjusted_regression()
```

Out[18]: 1.897000833090486

An estimate of the experimental error using slope of the regression line through non-significant points is 1.897, which is close the ANOVA estimate of 1.832.

c.

We will now include the effects of the blocks. Assuming the blocks are sized 2^{n-1} we will have two blocks. We will use the method of assigning even/odd treatment combinations for the treatment selected to be confounded with blocks, which in this case is $A*B*C*D*E$.

In [19]:

```python
#get sum of treatment combination
q2_fixed['treatment_sum'] = q2_fixed[['A','B','C','D','E']].sum(axis=1)
q2_fixed['block'] = q2_fixed.treatment_sum.apply(lambda x: 'block_1' if x % 2 == 0 else 'b
print(q2_fixed['block'].value_counts())
q2_fixed = q2_fixed.sort_values(by="block")
q2_fixed
```

```
block_1    16
block_2    16
Name: block, dtype: int64
```

Out[19]:

	A	B	C	D	E	y	treatment_sum	block
0	0	0	0	0	0	23	0	block_1
29	1	0	1	1	1	31	4	block_1
27	1	1	0	1	1	36	4	block_1
24	0	0	0	1	1	14	2	block_1
23	1	1	1	0	1	34	4	block_1
20	0	0	1	0	1	21	2	block_1
18	0	1	0	0	1	25	2	block_1
17	1	0	0	0	1	18	2	block_1
30	0	1	1	1	1	25	4	block_1
12	0	0	1	1	0	29	2	block_1
15	1	1	1	1	0	33	4	block_1
9	1	0	0	1	0	24	2	block_1
6	0	1	1	0	0	30	2	block_1
3	1	1	0	0	0	27	2	block_1
5	1	0	1	0	0	20	2	block_1
10	0	1	0	1	0	28	2	block_1
11	1	1	0	1	0	33	3	block_2
1	1	0	0	0	0	15	1	block_2
28	0	0	1	1	1	22	3	block_2
2	0	1	0	0	0	27	1	block_2
26	0	1	0	1	1	24	3	block_2
25	1	0	0	1	1	22	3	block_2
4	0	0	1	0	0	24	1	block_2
21	1	0	1	0	1	25	3	block_2
19	1	1	0	0	1	32	3	block_2
7	1	1	1	0	0	31	3	block_2
16	0	0	0	0	1	20	1	block_2
8	0	0	0	1	0	23	1	block_2
14	0	1	1	1	0	31	3	block_2
13	1	0	1	1	0	26	3	block_2

	A	B	C	D	E	y	treatment_sum	block
22	0	1	1	0	1	24	3	block_2
31	1	1	1	1	1	36	5	block_2

The above table provides the test runs and their treatment structures allocated to each block.

In [20]:

```
q2_lm2 = ols('y ~ block + A + B + C + D + E + A:B + A:C + A:D + A:E + B:C + B:D + B:E + C:
q2_anova2 = sm.stats.anova_lm(q2_lm2)
q2_anova2['significant'] = q2_anova2['PR(>F)'].apply(lambda x: True if x < .05 else False)
q2_anova2['std_effects'] = q2_anova2['mean_sq']**.5
q2_anova2
```

Out[20]:

	df	sum_sq	mean_sq	F	PR(>F)	significant	std_effects
block	1.0	0.28125	0.281250	0.078901	7.826294e-01	False	0.530330
A	1.0	87.78125	87.781250	24.625950	1.703352e-04	True	9.369165
B	1.0	442.53125	442.531250	124.146698	1.183118e-08	True	21.036427
C	1.0	81.28125	81.281250	22.802455	2.455562e-04	True	9.015611
D	1.0	52.53125	52.531250	14.736996	1.610081e-03	True	7.247845
E	1.0	7.03125	7.031250	1.972531	1.805473e-01	False	2.651650
A:B	1.0	57.78125	57.781250	16.209819	1.099308e-03	True	7.601398
A:C	1.0	1.53125	1.531250	0.429573	5.221242e-01	False	1.237437
A:D	1.0	42.78125	42.781250	12.001753	3.468159e-03	True	6.540738
A:E	1.0	132.03125	132.031250	37.039743	2.083599e-05	True	11.490485
B:C	1.0	22.78125	22.781250	6.390999	2.318464e-02	True	4.772971
B:D	1.0	2.53125	2.531250	0.710111	4.126425e-01	False	1.590990
B:E	1.0	1.53125	1.531250	0.429573	5.221242e-01	False	1.237437
C:D	1.0	1.53125	1.531250	0.429573	5.221242e-01	False	1.237437
C:E	1.0	0.28125	0.281250	0.078901	7.826294e-01	False	0.530330
D:E	1.0	11.28125	11.281250	3.164816	9.550809e-02	False	3.358757
Residual	15.0	53.46875	3.564583	NaN	NaN	False	1.888010

The same main and interaction effects as before are significant though the experimental error has a decreased degrees of freedom.

In [21]:

```
q2_hn2 = HalfNormPlot(data=q2_yates.iloc[1:,:], data_col='std_effect', label_col='effect')
q2_hn2.half_norm_data
```

```
C:\Users\aburtnerabt\Documents\K-State\Experimental Design\half_normal_plotting.py:45: Set
tingWithCopyWarning:
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_gu
ide/indexing.html#returning-a-view-versus-a-copy
  self.half_norm_data[self.abs_col] = self.half_norm_data[self.data_col].apply(lambda x: a
bs(x))
```

Out[21]:

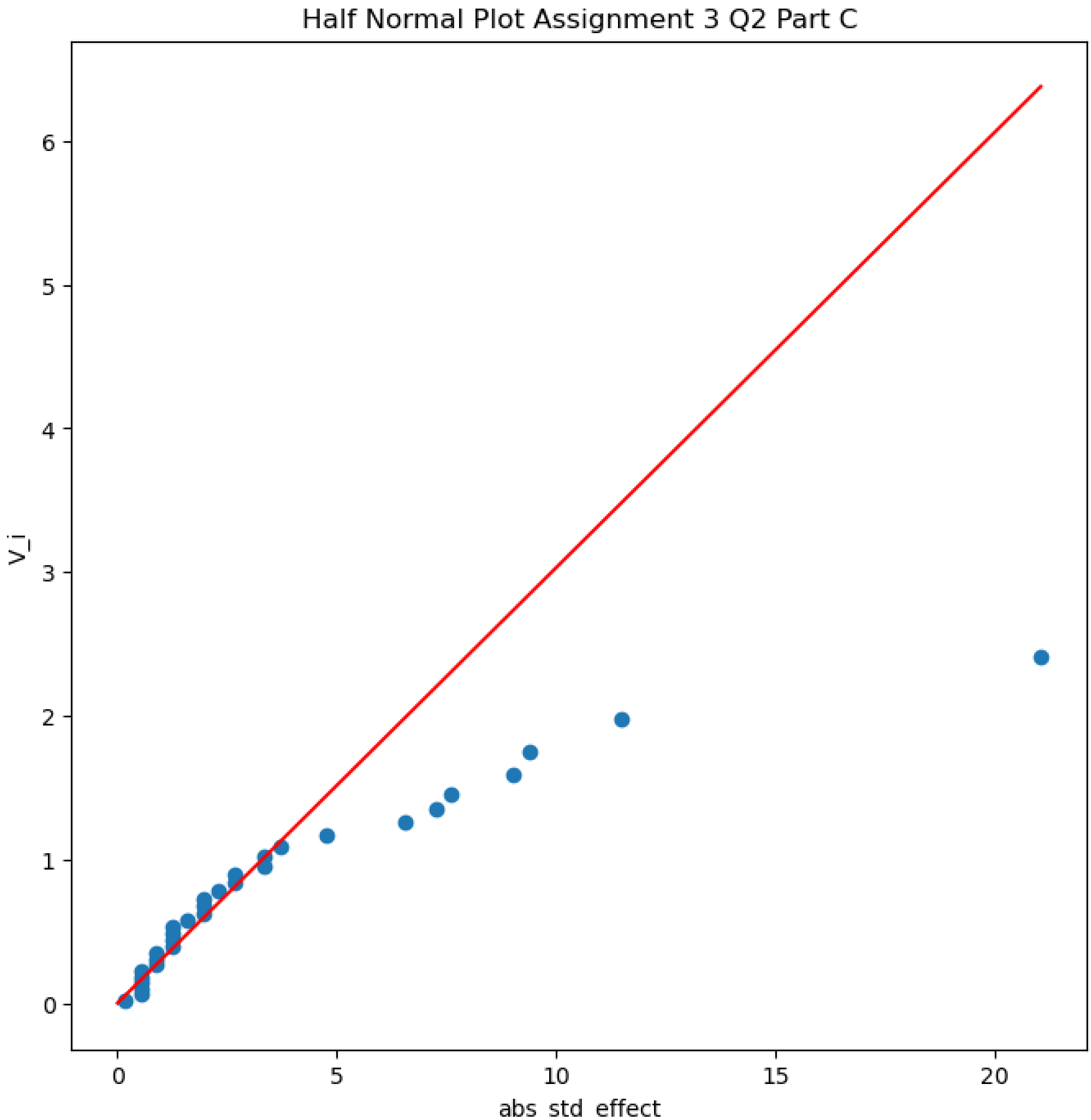
	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect	abs_std_effect	r_i	r_i*	
0	1	1	0	0	1	32	1	9	18	21	-1	-0.176777	abe	0.176777	1.0	0.016129	0.5
1	1	1	1	1	1	36	11	2	-2	1	-3	-0.530330	abcde	0.530330	2.0	0.048387	0.5
2	0	1	1	1	1	25	9	4	-3	4	-3	-0.530330	bcde	0.530330	3.0	0.080645	0.5
3	0	0	1	0	1	21	1	19	5	-10	3	0.530330	ce	0.530330	4.0	0.112903	0.5
4	1	1	1	0	1	34	2	8	0	-9	-3	-0.530330	abce	0.530330	5.0	0.145161	0.5
5	1	0	1	1	1	31	12	6	1	-9	3	0.530330	acde	0.530330	6.0	0.177419	0.5
6	1	0	0	1	1	22	7	5	-5	9	5	0.883883	ade	0.883883	7.0	0.209677	0.6
7	1	1	0	1	1	36	10	5	-16	-9	-5	-0.883883	abde	0.883883	8.0	0.241935	0.6
8	1	1	1	1	0	33	61	20	6	-5	5	0.883883	abcd	0.883883	9.0	0.274194	0.6
9	1	1	1	0	0	31	64	114	40	21	-7	-1.237437	abc	1.237437	10.0	0.306452	0.6
10	0	0	1	1	0	29	36	5	13	-4	7	1.237437	cd	1.237437	11.0	0.338710	0.6
11	1	0	1	0	0	20	61	104	5	63	7	1.237437	ac	1.237437	12.0	0.370968	0.6
12	0	1	0	0	1	25	-4	14	9	16	7	1.237437	be	1.237437	13.0	0.403226	0.7
13	0	1	0	1	0	28	46	6	31	-2	-9	-1.590990	bd	1.590990	14.0	0.435484	0.7
14	0	1	0	1	1	24	4	4	-7	-12	11	1.944544	bde	1.944544	15.0	0.467742	0.7
15	1	0	1	0	1	25	5	12	-7	1	11	1.944544	ace	1.944544	16.0	0.500000	0.7
16	0	0	1	1	1	22	8	9	-3	-6	11	1.944544	cde	1.944544	17.0	0.532258	0.7
17	1	1	0	1	0	33	58	-1	32	9	-13	-2.298097	abd	2.298097	18.0	0.564516	0.7
18	0	0	0	0	1	20	-8	16	13	30	-15	-2.651650	e	2.651650	19.0	0.596774	0.7
19	0	1	1	1	0	31	53	20	15	-2	-15	-2.651650	bcd	2.651650	20.0	0.629032	0.8
20	0	1	1	0	1	24	-3	24	9	-4	-19	-3.358757	bce	3.358757	21.0	0.661290	0.8
21	0	0	0	1	1	14	-2	8	1	-2	-19	-3.358757	de	3.358757	22.0	0.693548	0.8
22	1	0	1	1	0	26	60	14	9	-23	-21	-3.712311	acd	3.712311	23.0	0.725806	0.8
23	0	1	1	0	0	30	55	96	19	22	-27	-4.772971	bc	4.772971	24.0	0.758065	0.8
24	1	0	0	1	0	24	57	-3	23	27	37	6.540738	ad	6.540738	25.0	0.790323	0.8
25	0	0	0	1	0	23	38	-8	33	24	41	7.247845	d	7.247845	26.0	0.822581	0.9
26	1	1	0	0	0	27	61	119	210	59	43	7.601398	ab	7.601398	27.0	0.854839	0.9
27	0	0	1	0	0	24	47	95	-11	56	51	9.015611	c	9.015611	28.0	0.887097	0.9
28	1	0	0	0	0	15	54	105	227	409	53	9.369165	a	9.369165	29.0	0.919355	0.9
29	1	0	0	0	1	18	0	17	11	11	65	11.490485	ae	11.490485	30.0	0.951613	0.9
30	0	1	0	0	0	27	44	108	199	-6	119	21.036427	b	21.036427	31.0	0.983871	0.9

In [22]:

```
q2_hn2.plot_half_norm(title="Half Normal Plot Assignment 3 Q2 Part C", data_percent=.75)
```

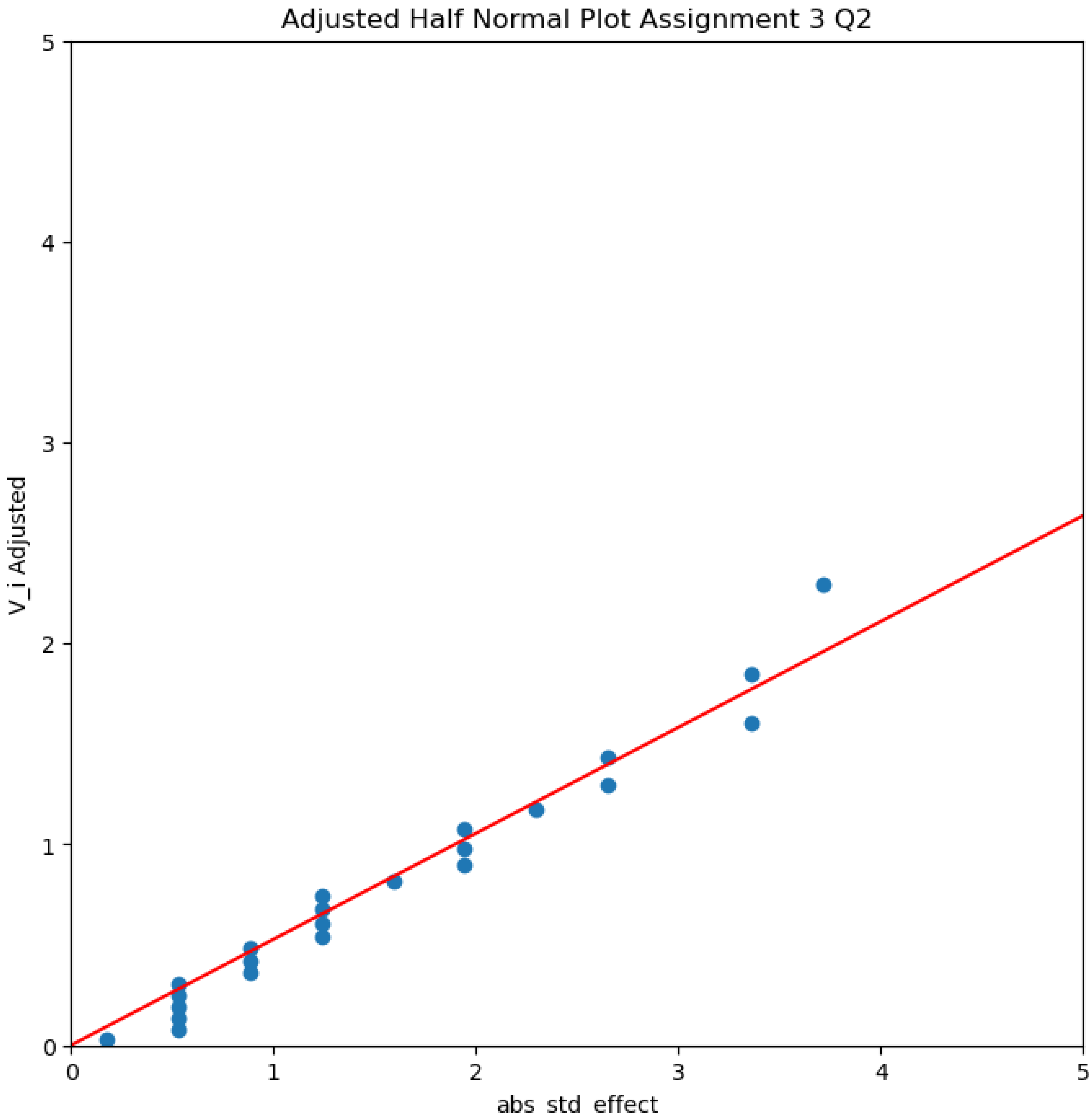
Out[22]:

```
<AxesSubplot: title={'center': 'Half Normal Plot Assignment 3 Q2 Part C'}, xlabel='abs_std_effect', ylabel='V_i'>
```



In [28]:
```
figure = q2_hn2.plot_half_norm(title="Adjusted Half Normal Plot Assignment 3 Q2", num_adju
figure.set_ylim(0,5);
figure.set_xlim(0,5)
```

Out[28]:
```
(0.0, 5.0)
```



Once again the same factors are significant.

In [27]:
```
q2_hn2.half_norm_data_adj
```

Out[27]:

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect	abs_std_effect	r_i	r_i*	
0	1	1	0	0	1	32	1	9	18	21	-1	-0.176777	abe	0.176777	1.0	0.021739	0.5
1	1	1	1	1	1	36	11	2	-2	1	-3	-0.530330	abcde	0.530330	2.0	0.065217	0.5
2	0	1	1	1	1	25	9	4	-3	4	-3	-0.530330	bcde	0.530330	3.0	0.108696	0.5
3	0	0	1	0	1	21	1	19	5	-10	3	0.530330	ce	0.530330	4.0	0.152174	0.5
4	1	1	1	0	1	34	2	8	0	-9	-3	-0.530330	abce	0.530330	5.0	0.195652	0.5
5	1	0	1	1	1	31	12	6	1	-9	3	0.530330	acde	0.530330	6.0	0.239130	0.6
6	1	0	0	1	1	22	7	5	-5	9	5	0.883883	ade	0.883883	7.0	0.282609	0.6
7	1	1	0	1	1	36	10	5	-16	-9	-5	-0.883883	abde	0.883883	8.0	0.326087	0.6

	A	B	C	D	E	y	step_1	step_2	step_3	step_4	step_5	std_effect	effect	abs_std_effect	r_i	r_i*	
8	1	1	1	1	0	33	61	20	6	-5	5	0.883883	abcd	0.883883	9.0	0.369565	0.6
9	0	1	0	0	1	25	-4	14	9	16	7	1.237437	be	1.237437	10.0	0.413043	0.7
10	1	0	1	0	0	20	61	104	5	63	7	1.237437	ac	1.237437	11.0	0.456522	0.7
11	1	1	1	0	0	31	64	114	40	21	-7	-1.237437	abc	1.237437	12.0	0.500000	0.7
12	0	0	1	1	0	29	36	5	13	-4	7	1.237437	cd	1.237437	13.0	0.543478	0.7
13	0	1	0	1	0	28	46	6	31	-2	-9	-1.590990	bd	1.590990	14.0	0.586957	0.7
14	0	1	0	1	1	24	4	4	-7	-12	11	1.944544	bde	1.944544	15.0	0.630435	0.8
15	1	0	1	0	1	25	5	12	-7	1	11	1.944544	ace	1.944544	16.0	0.673913	0.8
16	0	0	1	1	1	22	8	9	-3	-6	11	1.944544	cde	1.944544	17.0	0.717391	0.8
17	1	1	0	1	0	33	58	-1	32	9	-13	-2.298097	abd	2.298097	18.0	0.760870	0.8
18	0	0	0	0	1	20	-8	16	13	30	-15	-2.651650	e	2.651650	19.0	0.804348	0.9
19	0	1	1	1	0	31	53	20	15	-2	-15	-2.651650	bcd	2.651650	20.0	0.847826	0.9
20	0	0	0	1	1	14	-2	8	1	-2	-19	-3.358757	de	3.358757	21.0	0.891304	0.9
21	0	1	1	0	1	24	-3	24	9	-4	-19	-3.358757	bce	3.358757	22.0	0.934783	0.9
22	1	0	1	1	0	26	60	14	9	-23	-21	-3.712311	acd	3.712311	23.0	0.978261	0.9

In [26]:

```
#estimate error
q2_hn2.sigma_from_adjusted_regression()
```

Out[26]:

```
1.897000833090486
```

The experimental error estimate is the same as before.

In [ ]:

```
```