Question 1

```
import numpy as np
import pandas as pd
from scipy.stats import f, ncf, t
import matplotlib.pyplot as plt
```

```
N = 4
k = 3
sigma2 = 1
SSA = 2
nc = N/sigma2 * SSA
dfn = k - 1
dfd =  k*(N-1)
thresh = 0.01
```

Part A

```
def power(dfn, dfd, nc, thresh):
  central_F = f.ppf(1 - thresh, dfn, dfd)
  power = 1 - ncf.cdf(central_F, dfn, dfd, nc)
  return power
```

```
pow = power(dfn, dfd, nc, thresh)
print("The power for the one way ANOVA with N = {}, k = {}, p-val = {} is: {}".format
```

```
The power for the one way ANOVA with N = 4, k = 3, p-val = 0.01 is: 0.256916849€
```

Part B

```
pow = 0
N = 1
while pow < 0.8:
  N += 1
  nc = N/sigma2 * SSA
  dfd =  k*(N-1)
  pow = power(dfn, dfd, nc, thresh)
```

```
print("The smallest n that would give us a power of at least 80% is N = {} with power
```

```
The smallest n that would give us a power of at least 80% is N = 9 with power =
```

Part C

```
#params
k = 6
N = 10
sigma2 = 1
dfn = k - 1
dfd =  k*(N-1)
thresh = 0.01

#trial and error until we get a 0.8 power since there is no closed form solution to g
SSA = 0.4
nc = N/sigma2 * SSA


pow = 0
SSA = 0
while pow < 0.8:
  SSA += 0.000001
  nc = N/sigma2 * SSA
  pow = power(dfn, dfd, nc, thresh)


pow, SSA
```

```
(0.8000000155255544, 2.0536479999331503)
```

```
#get difference
a = [1, -1/5, -1/5, -1/5, -1/5, -1/5]
a2 = [i**2 for i in a]
alpha1 = np.sqrt(SSA/(np.sum(a2)))
alpha2 = -1/5 * alpha1
diff = alpha1 - alpha2


print("Max1≤i≤6 αi − min1≤i≤6 αi is = {} for us to get {}% power".format(diff, pow *
```

```
Max1≤i≤6 αi − min1≤i≤6 αi is = 1.5698336217318638 for us to get 80.0000015525554
```

Can we get exactly 80% power?

We theoretically can get 80% power.

PART D

```
#get difference
a = [1, 1, 1, -1, -1, -1]
a2 = [i**2 for i in a]
alpha1 = np.sqrt(SSA/(np.sum(a2)))
alpha4 = -alpha1
diff = alpha1 - alpha4


print("Max1≤i≤6 αi − min1≤i≤6 αi is = {} for us to get {}% power".format(diff, pow *
```

```
Max1≤i≤6 αi − min1≤i≤6 αi is = 1.1700848971857127 for us to get 80.0000015525554
```

QUESTION 3

Part A

```
%load_ext rpy2.ipython
```

```
%%R
foam_density <- c(rep("low", 2), rep("high", 2))
anchor_type <- c(rep("a", 4), rep("b", 4), rep("c", 4))
force <- c(190, 200, 241, 255, 185, 190, 230, 237, 210, 205, 256, 260)

my_data <- data.frame(anchor_type, foam_density, force)
my_data
```

	anchor_type	foam_density	force
1	a	low	190
2	a	low	200
3	a	high	241
4	a	high	255
5	b	low	185
6	b	low	190
7	b	high	230
8	b	high	237
9	c	low	210
10	c	low	205
11	c	high	256
12	c	high	260

```
%%R
analysis <- aov(force ~ anchor_type + foam_density, data = my_data)
summary(analysis)
```

```
             Df Sum Sq Mean Sq F value   Pr(>F)
anchor_type   2    990     495   17.17  0.00127 **
```

```
foam_density  1   7450    7450  258.38 2.25e-07 ***
Residuals     8    231      29
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

PART B

```
%%R
foam_density <- c(rep("low", 2), rep("high", 2))
anchor_type <- c(rep("a", 4), rep("b", 4), rep("c", 4))
lab <- c(rep("lab1", 1), rep("lab2", 1))
force <- c(190, 200, 241, 255, 185, 190, 230, 237, 210, 205, 256, 260)

my_data2 <- data.frame(anchor_type, foam_density, lab, force)
my_data2
```

```
   anchor_type foam_density  lab force
1            a          low lab1   190
2            a          low lab2   200
3            a         high lab1   241
4            a         high lab2   255
5            b          low lab1   185
6            b          low lab2   190
7            b         high lab1   230
8            b         high lab2   237
9            c          low lab1   210
10           c          low lab2   205
11           c         high lab1   256
12           c         high lab2   260
```

```
%%R
analysis <- aov(force ~ anchor_type + foam_density + lab, data = my_data2)
summary(analysis)
```

```
             Df Sum Sq Mean Sq F value   Pr(>F)    
anchor_type   2    990     495  26.952 0.000515 ***
foam_density  1   7450    7450 405.578 1.86e-07 ***
lab           1    102     102   5.557 0.050534 .  
Residuals     7    129      18                     
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

PART C

```
%%R
foam_density <- c(rep("low", 1), rep("high", 1))
anchor_type <- c(rep("a", 2), rep("b", 2), rep("c", 2))
```

```
lab <- c(rep("lab1", 1), rep("lab2", 1))
vec1 <- c(190, 200)
vec2 <- c(241, 255)
vec3 <- c(185, 190)
vec4 <- c(230, 237)
vec5 <- c(210, 205)
vec6 <- c(256, 260)

force <- c(mean(vec1), mean(vec2), mean(vec3), mean(vec4), mean(vec5), mean(vec6))

my_data3 <- data.frame(anchor_type, foam_density, lab, force)
my_data3
```

```
  anchor_type foam_density  lab force
1           a          low lab1 195.0
2           a         high lab2 248.0
3           b          low lab1 187.5
4           b         high lab2 233.5
5           c          low lab1 207.5
6           c         high lab2 258.0
```

```
%%R
analysis <- aov(force ~ anchor_type + foam_density + lab, data = my_data3)
summary(analysis)
```

```
             Df Sum Sq Mean Sq F value  Pr(>F)
anchor_type   2    495     248   39.34 0.02479 *
foam_density  1   3725    3725  592.06 0.00168 **
Residuals     2     13       6
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Question 2

```
catapult = pd.read_csv('data.txt', delim_whitespace = True)
```

```
main_eff = catapult.columns.drop('Dist')
main_eff
```

```
Index(['Front', 'Back', 'Fixed', 'Moving', 'Bucket'], dtype='object')
```

```
catapult
```

	Front	Back	Fixed	Moving	Bucket	Dist
0	1	-1	-1	1	1	210.3
1	1	1	1	1	1	343.0
2	1	-1	-1	-1	-1	50.0
3	-1	-1	1	1	1	263.5
4	-1	-1	-1	1	-1	134.5
5	-1	-1	-1	-1	1	94.5
6	-1	1	-1	1	1	310.8
7	1	1	-1	-1	1	94.8
8	-1	1	-1	-1	-1	91.5
9	1	1	-1	1	-1	168.5
10	-1	1	1	-1	1	277.4
11	1	1	1	-1	-1	145.5
12	1	-1	1	-1	1	157.5

```
effects = {}

for eff in main_eff:
  effects[eff] = np.mean(catapult[catapult[eff] == 1]['Dist']) - np.mean(catapult[cat

effects
```

```
{'Front': -27.887499999999999,
 'Back': 62.587500000000006,
 'Fixed': 73.1875,
 'Moving': 103.98749999999998,
 'Bucket': 76.0375}
```

```
from itertools import combinations
list_combinations = list()

for n in range(len(main_eff) + 1):
    list_combinations += list(combinations(main_eff, n))


interact = [i for i in list_combinations if len(i)  == 2]


for inter in interact:
    pos_effect = catapult[(catapult[inter[0]] == 1) & (catapult[inter[1]] == 1)]['Dis
    pos_neg_effect = catapult[(catapult[inter[0]] == 1) & (catapult[inter[1]] == -1)]
    neg_pos_effect = catapult[(catapult[inter[0]] == -1) & (catapult[inter[1]] == 1)]
```

```
    neg_effect = catapult[(catapult[inter[0]] == -1) & (catapult[inter[1]] == -1)]['D
    effects[inter] = (np.mean(pos_effect) - np.mean(pos_neg_effect) - np.mean(neg_pos

effects
```

```
{'Front': -27.88749999999999,
 'Back': 62.587500000000006,
 'Fixed': 73.1875,
 'Moving': 103.98749999999998,
 'Bucket': 76.0375,
 ('Front', 'Back'): -20.712500000000006,
 ('Front', 'Fixed'): -0.9625000000000057,
 ('Front', 'Moving'): 6.137500000000003,
 ('Front', 'Bucket'): -7.262500000000003,
 ('Back', 'Fixed'): 18.51250000000001,
 ('Back', 'Moving'): 15.912499999999994,
 ('Back', 'Bucket'): 12.462500000000006,
 ('Fixed', 'Moving'): -19.3375,
 ('Fixed', 'Bucket'): 9.562500000000014,
 ('Moving', 'Bucket'): 21.862499999999983}
```

```
s0 = 1.5 * np.median(list(map(abs, effects.values())))


def get_PSE(c_list, s0):
  med_list = []
  for cj in c_list:
    if np.abs(cj) < 2.5 * s0:
      med_list.append(np.abs(cj))
  return 1.5 * np.median(med_list)


PSE = get_PSE(effects.values(), s0)


t_stats = {}

for eff in effects.keys():
  t_stats[eff] = effects[eff] / PSE


#find the t statistics
t_stats
```

```
{'Front': -1.080125877511498,
 'Back': 2.4241103848946985,
 'Fixed': 2.8346647300895667,
 'Moving': 4.027596223674654,
 'Bucket': 2.9450496247881865,
 ('Front', 'Back'): -0.8022270636649723,
 ('Front', 'Fixed'): -0.03727910917453423,
 ('Front', 'Moving'): 0.23771483902202867,
 ('Front', 'Bucket'): -0.281287823771484,
 ('Back', 'Fixed'): 0.7170176712660377,
 ('Back', 'Moving'): 0.6163156620672958,
```

```
 ('Back', 'Bucket'): 0.48269184216896655,
 ('Fixed', 'Moving'): -0.7489711934156378,
 ('Fixed', 'Bucket'): 0.3703703703703709,
 ('Moving', 'Bucket'): 0.8467683369644147}
```

```
#find the p_values
n = 16
p_vals = {}
for eff in effects.keys():
  p_vals[eff] = (1 - t.cdf(abs(t_stats[eff]), df=(n - 1)/3))*2
```

```
#find the p values
p_vals
```

```
{'Front': 0.3294099342475638,
 'Back': 0.059813387730317524,
 'Fixed': 0.03647425206594668,
 'Moving': 0.010045032094806894,
 'Bucket': 0.032068581491171866,
 ('Front', 'Back'): 0.4588371186158433,
 ('Front', 'Fixed'): 0.9717050655713386,
 ('Front', 'Moving'): 0.8215361020566676,
 ('Front', 'Bucket'): 0.7897589884500309,
 ('Back', 'Fixed'): 0.505452463966575,
 ('Back', 'Moving'): 0.5646605580510182,
 ('Back', 'Bucket'): 0.6497059311457751,
 ('Fixed', 'Moving'): 0.48759176418446115,
 ('Fixed', 'Bucket'): 0.7262783317524306,
 ('Moving', 'Bucket'): 0.43576770470352333}
```

Is it possible that the PSE would be the median of an empty set? If so, construct an example. If not, prove it cannot happen.

Yes, it is possible that the PSE would be the median of an empty set. Say we had three values in our effect (c1, c2, c3) = {0, 0, 0}. Then, the median would be 0 as well and s0 would be 0. In our PSE algorithm, we must check the condition : abs(cj) < 2.5*s0. If s0 is 0 and all the cj was 0 for all effects, then we would never have a case that cj < 0 and we would get that the PSE is the median of an empty set.

✓ 0s completed at 4:48 PM