Factorial design at two levels in Python

4 stages

1. Define variables and high- and low levels

	S	T	C
Low level	830	70	0.5
High level	910	120	0.7

1. Python input

```
inputs_labels = {'S' : 'Steel Temperature',
                 'T' : 'Oil Temperature',
                 'C' : 'Carbon Level'}

dat = [('S',830,910),
       ('T',70,120),
       ('C',0.5,0.7)]

inputs_df = pd.DataFrame(dat,columns=['index','low','high'])
inputs_df = inputs_df.set_index(['index'])
inputs_df['label'] = inputs_df.index.map( lambda z : inputs_labels[z] )

inputs_df
```

1. Python input

Dictionary

List

DataFrame

```
inputs_labels = {'S' : 'Steel Temperature',
                 'T' : 'Oil Temperature',
                 'C' : 'Carbon Level'}

dat = [('S',830,910),
       ('T',70,120),
       ('C',0.5,0.7)]

inputs_df = pd.DataFrame(dat,columns=['index','low','high'])
inputs_df = inputs_df.set_index(['index'])
inputs_df['label'] = inputs_df.index.map( lambda z : inputs_labels[z] )

inputs_df
```

1. Python input

Dictionary

List

DataFrame

```
inputs_labels = {'S' : 'Steel Temperature',
                 'T' : 'Oil Temperature',
                 'C' : 'Carbon Level'}

dat = [('S',830,910),
       ('T',70,120),
       ('C',0.5,0.7)]

inputs_df = pd.DataFrame(dat,columns=['index','low','high'])
inputs_df = inputs_df.set_index(['index'])
inputs_df['label'] = inputs_df.index.map( lambda z : inputs_labels[z] )

inputs_df
```

Takes index (S, T, C) and maps them to correct input labels.
If you write them explicitly:
inputs_df.index.map({S: ’Steel Temperature’)

1. Python input

Dictionary

List

DataFrame

```
inputs_labels = {'S' : 'Steel Temperature',
                 'T' : 'Oil Temperature',
                 'C' : 'Carbon Level'}

dat = [('S',830,910),
       ('T',70,120),
       ('C',0.5,0.7)]

inputs_df = pd.DataFrame(dat,columns=['index','low','high'])
inputs_df = inputs_df.set_index(['index'])
inputs_df['label'] = inputs_df.index.map( lambda z : inputs_labels[z] )

inputs_df
```

```
x = lambda a : a + 10
print(x(5))
```

Takes index (S, T, C) and maps them to correct input labels.
If you write them explicitly:
inputs_df.index.map({S: 'Steel Temperature')

1. Python output

Out[43]:

index	low	high	label
S	830.0	910.0	Steel Temperature
T	70.0	120.0	Oil Temperature
C	0.5	0.7	Carbon Level

2. Encode variables

2.Python input

$$x_i = \frac{\phi_i - avg(\phi)}{span(\phi)}$$

$$avg(\phi) = \frac{\phi_{high} + \phi_{low}}{2}$$

$$span(\phi) = \frac{\phi_{high} - \phi_{low}}{2}$$

```
inputs_df['average'] = inputs_df.apply( lambda z : ( z['high'] + z['low'])/2 , axis=1)
inputs_df['span'] = inputs_df.apply( lambda z : ( z['high'] - z['low'])/2 , axis=1)

inputs_df['encoded_low'] = inputs_df.apply( lambda z : ( z['low']  - z['average'] )/( z['span'] ), axis=1)
inputs_df['encoded_high'] = inputs_df.apply( lambda z : ( z['high'] - z['average'] )/( z['span'] ), axis=1)

inputs_df = inputs_df.drop(['average','span'],axis=1)

inputs_df
```

2.Python input

Computes average and span for every high and low value in each row (axis=1)

df.apply applies a function along an axis, axis=0 for columns and axis =1 for each row

```
inputs_df['average'] = inputs_df.apply( lambda z : ( z['high'] + z['low'])/2 , axis=1)
inputs_df['span'] = inputs_df.apply( lambda z : ( z['high'] - z['low'])/2 , axis=1)

inputs_df['encoded_low'] = inputs_df.apply( lambda z : ( z['low']  - z['average'] )/( z['span'] ), axis=1)
inputs_df['encoded_high'] = inputs_df.apply( lambda z : ( z['high'] - z['average'] )/( z['span'] ), axis=1)

inputs_df = inputs_df.drop(['average','span'],axis=1)

inputs_df
```

2.Python input

df.apply applies a function along an axis, axis=0 for columns and axis =1 for each rows

Computes average and span for every high and low value in each row (axis=1)

```
inputs_df['average'] = inputs_df.apply( lambda z : ( z['high'] + z['low'])/2 , axis=1)
inputs_df['span'] = inputs_df.apply( lambda z : ( z['high'] - z['low'])/2 , axis=1)

inputs_df['encoded_low'] = inputs_df.apply( lambda z : ( z['low']  - z['average'] )/( z['span'] ), axis=1)
inputs_df['encoded_high'] = inputs_df.apply( lambda z : ( z['high'] - z['average'] )/( z['span'] ), axis=1)

inputs_df = inputs_df.drop(['average','span'],axis=1)

inputs_df
```

Computes encoded xi-values high and low value in each row (axis=1)

df.drop removes specified labels from rows or columns, here axis=1 refers to columns.

2. Python output

Out[44]:

index	low	high	label	encoded_low	encoded_high
S	830.0	910.0	Steel Temperature	-1.0	1.0
T	70.0	120.0	Oil Temperature	-1.0	1.0
C	0.5	0.7	Carbon Level	-1.0	1.0

3. Create design matrix

A full 2^k factorial design consists of all 2^k trial points:

$$(x_1, x_2, \ldots, x_k) = (\pm 1, \pm 1, \ldots, \pm 1),$$

where every possible combination of +1/-1 is selected in turn

Trial #		Factor	
	S (°C)	T (°C)	C (%)
1	-1	-1	-1
2	+1	-1	-1
3	-1	+1	-1
4	+1	+1	-1
5	-1	-1	+1
6	+1	-1	+1
7	-1	+1	+1
8	+1	+1	+1

3. Python input

```python
import itertools
# we have four repetitions
encoded_inputs= list(itertools.product([-1,1],[-1,1],[-1,1]))
encoded_inputs
```

itertools le implements a number of iterator building blocks

Itertools.product() results in a cartesian product, equivalent to a nested for-loop

```python
aa = []; bb=[]; cc=[]
for i in [-1,1]:
  for j in [-1, 1]:
    for k in [-1, 1]:
      aa.append((i,j,k))
aa
```

3. Python input

```python
import itertools
# we have four repetitions
encoded_inputs= list(itertools.product([-1,1],[-1,1],[-1,1]))
encoded_inputs
```

itertools le implements a number of iterator building blocks

Itertools.product() results in a cartesian product, equivalent to a nested for-loop

```python
aa = []; bb=[]; cc=[]
for i in [-1,1]:
  for j in [-1, 1]:
    for k in [-1, 1]:
      aa.append((i,j,k))
aa
```

```python
results=pd.DataFrame(encoded_inputs)
results=results[results.columns[::-1]]
results.columns=['S','T','C']
results
```

Put data in to pandas dataframe in the right order (revert encoded_inputs to fit the data)

3. Python output

First box:

```
Out[45]: [(-1, -1, -1),
          (-1, -1, 1),
          (-1, 1, -1),
          (-1, 1, 1),
          (1, -1, -1),
          (1, -1, 1),
          (1, 1, -1),
          (1, 1, 1)]
```

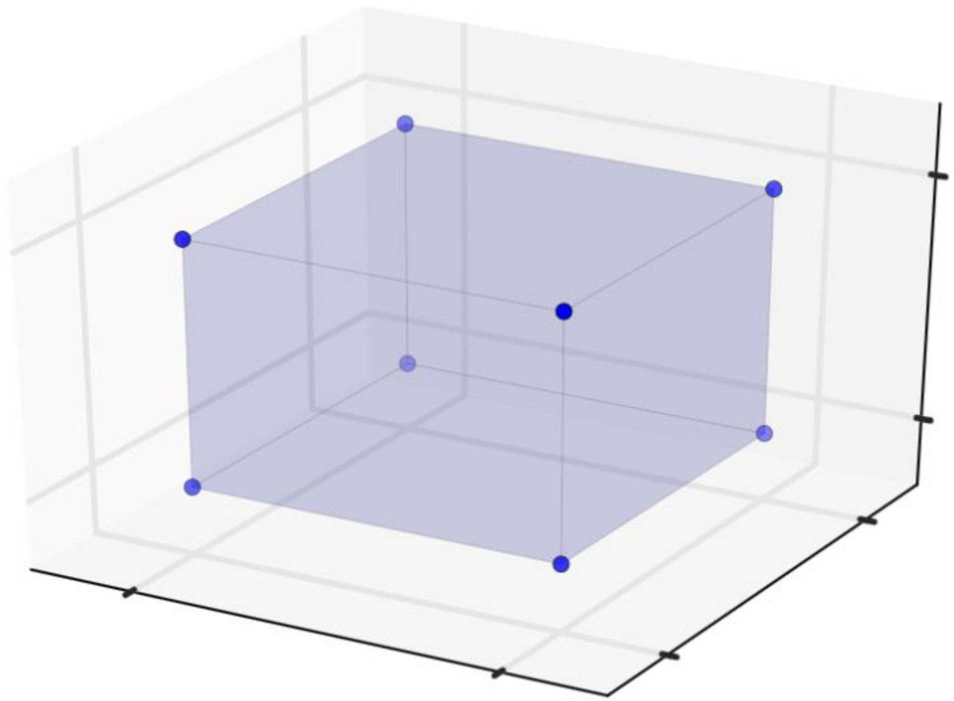
Second box:

Out[50]:

	S	T	C
0	-1	-1	-1
1	1	-1	-1
2	-1	1	-1
3	1	1	-1
4	-1	-1	1
5	1	-1	1
6	-1	1	1
7	1	1	1

3.5 Create matrix for experimental conditions

Trial #		Factor	
	S (°C)	T (°C)	C (%)
1	830	70	0.5
2	910	70	0.5
3	830	120	0.5
4	910	120	0.5
5	830	70	0.7
6	910	70	0.7
7	830	120	0.7
8	910	120	0.7



3.5 Create matrix for experimental conditions

```
real_experiment = results

var_labels = []
for var in ['S','T','C']:
    var_label = inputs_df.loc[var]['label']
    var_labels.append(var_label)
    real_experiment[var_label] = results.apply(
        lambda z : inputs_df.loc[var]['low'] if z[var]<0 else inputs_df.loc[var]['high'] ,
        axis=1)

print("The values of each real variable in the experiment:")
real_experiment[var_labels]
```

df.apply applies a function along an axis, axis=0 for columns and axis =1 for each rows

3.5 Python input

df.loc access a group of rows and columns by label(s) or a boolean array.

```
real_experiment = results

var_labels = []
for var in ['S','T','C']:
    var_label = inputs_df.loc[var]['label']
    var_labels.append(var_label)
    real_experiment[var_label] = results.apply(
        lambda z : inputs_df.loc[var]['low'] if z[var]<0 else inputs_df.loc[var]['high'] ,
        axis=1)

print("The values of each real variable in the experiment:")
real_experiment[var_labels]
```

input_df

index	low	high	label	encoded_low	encoded_high
S	830.0	910.0	Steel Temperature	-1.0	1.0
T	70.0	120.0	Oil Temperature	-1.0	1.0
C	0.5	0.7	Carbon Level	-1.0	1.0

3.5 Python output

```
The values of each real variable in the experiment:
```

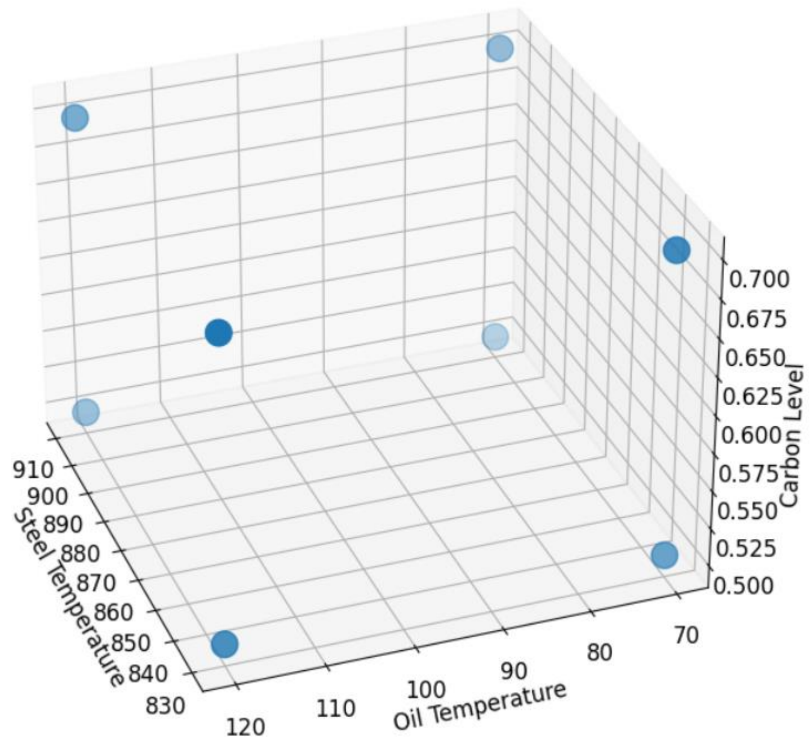
ut[58]:

	Steel Temperature	Oil Temperature	Carbon Level
0	830.0	70.0	0.5
1	910.0	70.0	0.5
2	830.0	120.0	0.5
3	910.0	120.0	0.5
4	830.0	70.0	0.7
5	910.0	70.0	0.7
6	830.0	120.0	0.7
7	910.0	120.0	0.7

3.6 Graphical representation of design matrix

```python
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

plt.rcParams.update({'font.size': 16})
# plot
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter3D(real_experiment['Steel Temperature'],
             real_experiment['Oil Temperature'],
             real_experiment['Carbon Level'],
           s=200)
ax.set_xlabel('Steel Temperature')
ax.set_ylabel('Oil Temperature')
ax.set_zlabel('Carbon Level');
ax.view_init(30, 160)
plt.show()
```



4. Make experiments ☺ and add data

Python input

```
y=[67, 79, 59, 90, 61, 75, 52, 87]
results['y']= y
results
```

Python output

	S	T	C	Steel Temperature	Oil Temperature	Carbon Level	y
0	-1	-1	-1	830.0	70.0	0.5	67
1	1	-1	-1	910.0	70.0	0.5	79
2	-1	1	-1	830.0	120.0	0.5	59
3	1	1	-1	910.0	120.0	0.5	90
4	-1	-1	1	830.0	70.0	0.7	61
5	1	-1	1	910.0	70.0	0.7	75
6	-1	1	1	830.0	120.0	0.7	52
7	1	1	1	910.0	120.0	0.7	87

5.1 Compute main effects

$$S \leftarrow \frac{12+31+14+35}{4} = 23$$

$$C \leftarrow \frac{61+75+52+87-67-79-59-90}{4} = -5$$

$$T \leftarrow \frac{59+90+52+87-67-79-61-75}{4} = 1.5$$

Trial #		**Factor**		**outcome**
	S (°C)	**T (°C)**	**C (%)**	
1	-	-	-	67
2	+	-	-	79
3	-	+	-	59
4	+	+	-	90
5	-	-	+	61
6	+	-	+	75
7	-	+	+	52
8	+	+	+	87
Estimated effect	**23**	**1.5**	**-5**	

5.1 Python input

```
labels = ['S','T','C']

main_effects = {}

print('main effects')
for key in labels:
        effects = results.groupby(key)['y'].mean()
        main_effects[key] = sum( [i*effects[i] for i in [-1,1]])
print(main_effects)
```

df.groupby
Group
DataFrame
using a mapper
or by a Series
of columns.

5.1 Python input

```
results.groupby('S')['y'].mean()
```

```
S
-1    59.75
 1    82.75
Name: y, dtype: float64
```

df.groupby
Group
DataFrame
using a mapper
or by a Series
of columns.

```
labels = ['S','T','C']

main_effects = {}

print('main effects')
for key in labels:
        effects = results.groupby(key)['y'].mean()
        main_effects[key] = sum( [i*effects[i] for i in [-1,1]])
print(main_effects)
```

Gives the average y-values for each label at high and low levels

5.1 Python input

```
results.groupby('S')['y'].mean()
```

```
S
-1    59.75
 1    82.75
Name: y, dtype: float64
```

df.groupby
Group DataFrame using a mapper or by a Series of columns.

```
labels = ['S','T','C']

main_effects = {}

print('main effects')
for key in labels:
        effects = results.groupby(key)['y'].mean()
        main_effects[key] = sum( [i*effects[i] for i in [-1,1]])
print(main_effects)
```

Gives the average y-values for each label at high and low levels

Compute the difference between the arithmetic means at low and high levels

5.1 Python input

```
results.groupby('S')['y'].mean()
```

```
S
-1    59.75
 1    82.75
Name: y, dtype: float64
```

df.groupby
Group DataFrame using a mapper or by a Series of columns.

```
labels = ['S','T','C']

main_effects = {}

print('main effects')
for key in labels:
        effects = results.groupby(key)['y'].mean()
        main_effects[key] = sum( [i*effects[i] for i in [-1,1]])
print(main_effects)
```

Gives the average y-values for each label at high and low levels

Compute the difference between the arithmetic means at low and high levels

Python output

```
Average main effects
{'S': 23.0, 'T': 1.5, 'C': -5.0}
```

5.2 two- and three-way interactions

Trial #		Factor						# outcome
	S	T	C	S x T	S x C	T x C	S x T x C	
1	-	-	-	+	+	+	-	67
2	+	-	-	-	-	+	+	79
3	-	+	-	-	+	-	+	59
4	+	+	-	+	-	-	-	90
5	-	-	+	+	-	-	+	61
6	+	-	+	-	+	-	-	75
7	-	+	+	-	-	+	-	52
8	+	+	+	+	+	+	+	87
Estimated effect	**23**	**-5**	**1.5**					

5.2 Python input

```
twoway_labels = list(itertools.combinations(labels, 2))


twoway_effects = {}
for key in twoway_labels:

    effects = results.groupby([key[0],key[1]])['y'].mean()

    twoway_effects[key] = sum([ i*j*effects[i][j]/2 for i in [-1,1] for j in [-1,1] ])
twoway_effects
```

5.2 Python input

Create all possible two-way interactions from labels (S, T, C)

```
twoway_labels = list(itertools.combinations(labels, 2))


twoway_effects = {}
for key in twoway_labels:

    effects = results.groupby([key[0],key[1]])['y'].mean()

    twoway_effects[key] = sum([ i*j*effects[i][j]/2 for i in [-1,1] for j in [-1,1] ])
twoway_effects
```

5.2 Python input

```
twoway_labels = list(itertools.combinations(labels, 2))


twoway_effects = {}
for key in twoway_labels:

    effects = results.groupby([key[0],key[1]])['y'].mean()

    twoway_effects[key] = sum([ i*j*effects[i][j]/2 for i in [-1,1] for j in |
twoway_effects
```

Create all possible two-way interactions from labels (S, T, C)

Gives the average y-values for high and low key[0] when key[1] is at high and low values

```
results.groupby(['S','C'])['y'].mean()
```

S	C	
-1	-1	63.0
	1	56.5
1	-1	84.5
	1	81.0

5.2 Python input

Create all possible two-way interactions from labels (S, T, C)

```
twoway_labels = list(itertools.combinations(labels, 2))


twoway_effects = {}
for key in twoway_labels:

    effects = results.groupby([key[0],key[1]])['y'].mean()

    twoway_effects[key] = sum([ i*j*effects[i][j]/2 for i in [-1,1] for j in [-1,1] ])
twoway_effects
```

Gives the average y-values for high and low key[0] when key[1] is at high and low values.

Compute the difference between the arithmetic means at low and high label 1 when label 2 is at low and high levels

$$13 = \tfrac{1}{2}\{(1|x_3 = 1) - (1|x_3 = -1)\} \leftarrow \tfrac{1}{2}\{0.72 - 0.78\} = -0.03.$$

5.3 Python input

```
threeway_labels = list(itertools.combinations(labels, 3))

threeway_effects = {}
for key in threeway_labels:

    effects = results.groupby([key[0],key[1],key[2]])['y'].mean()

    threeway_effects[key] = sum([ i*j*k*effects[i][j][k]/4 for i in [-1,1] for j in [-1,1] for k in [-1,1] ])

threeway_effects
```

5.2 Python output

```
Out[62]: {('S', 'T'): 10.0, ('S', 'C'): 1.5, ('T', 'C'): 0.0}
```

```
Out[64]: {('S', 'T', 'C'): 0.5}
```

6. summary of effects, analysis

Trial #		Factor						# outcome
	S	T	C	S x T	S x C	T x C	S x T x C	
1	-	-	-	+	+	+	-	67
2	+	-	-	-	-	+	+	79
3	-	+	-	-	+	-	+	59
4	+	+	-	+	-	-	-	90
5	-	-	+	+	-	-	+	61
6	+	-	+	-	+	-	-	75
7	-	+	+	-	-	+	-	52
8	+	+	+	+	+	+	+	87
Estimated effect	**23**	**-5**	**1.5**	**10**	**1.5**	**0**	**0.5**	**71.25**

6 Python input

```
effects=[] #pd.DataFrame({})
indexes=[]
for i,k in enumerate(main_effects.keys()):
    effects.append(abs(main_effects[k]))
    indexes.append(k)
for i,k in enumerate(twoway_effects.keys()):
    effects.append(abs(twoway_effects[k]))
    indexes.append(k)
for i,k in enumerate(threeway_effects.keys()):
    effects.append(abs(threeway_effects[k]))
    indexes.append(k)

effects_df=pd.DataFrame({"Standardized effect":effects})

# reset the indexes
effects_df.index=indexes
# Sort values in descending order
effects_df = effects_df.sort_values(by='Standardized effect', ascending=False)
# Add cumulative percentage column
effects_df["cum_percentage"] = round(effects_df["Standardized effect"].cumsum()/effects_df["Standardized effect"].sum()*100,2)

# Display data frame
effects_df
```

6. Python output

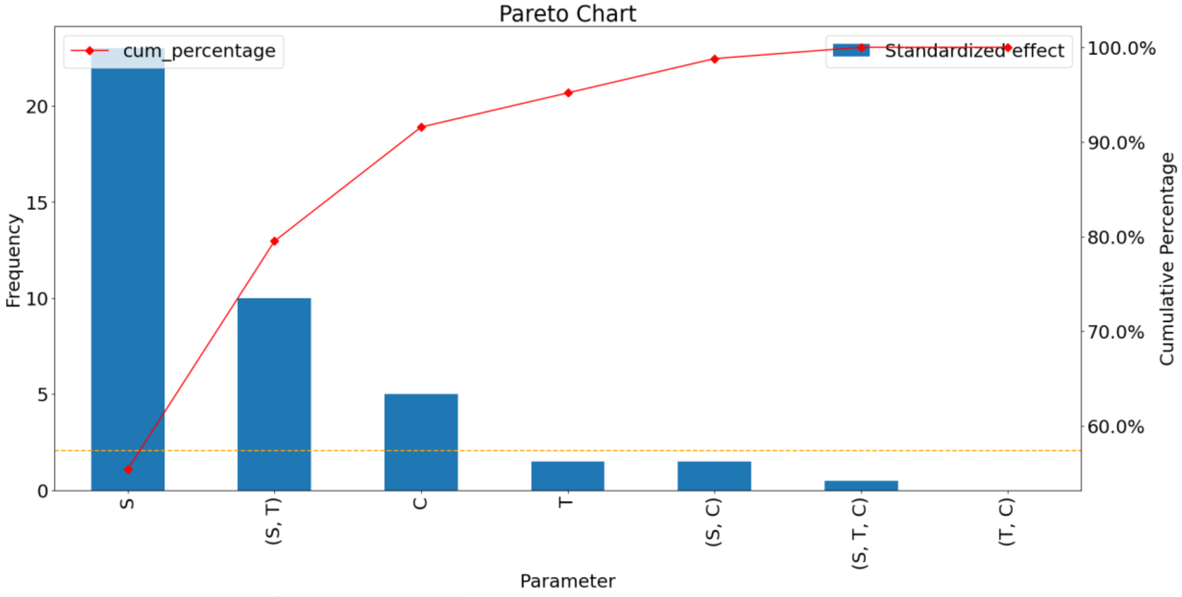
	Standardized effect	cum_percentage
S	23.0	55.42
(S, T)	10.0	79.52
C	5.0	91.57
T	1.5	95.18
(S, C)	1.5	98.80
(S, T, C)	0.5	100.00
(T, C)	0.0	100.00

6.1 graphical representation (Pareto chart)

```python
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
plt.rcParams.update({'font.size': 22})
# Set figure and axis
fig, ax = plt.subplots(figsize=(22,10))

# Plot bars (i.e. frequencies)
ax.set_title("Pareto Chart")
ax.set_xlabel("Parameter")
ax.set_ylabel("Frequency");
effects_df.plot.bar(y='Standardized effect', ax=ax)
ax.axhline(2.06, color="orange", linestyle="dashed")

# Second y axis (i.e. cumulative percentage)
ax2 = ax.twinx()
#ax2.plot(effects_df.index, effects_df["cum_percentage"], color="red", marker="D", ms=7)
effects_df.plot(y="cum_percentage", color="red", marker="D", ms=7, ax=ax2)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax2.set_ylabel("Cumulative Percentage");
```

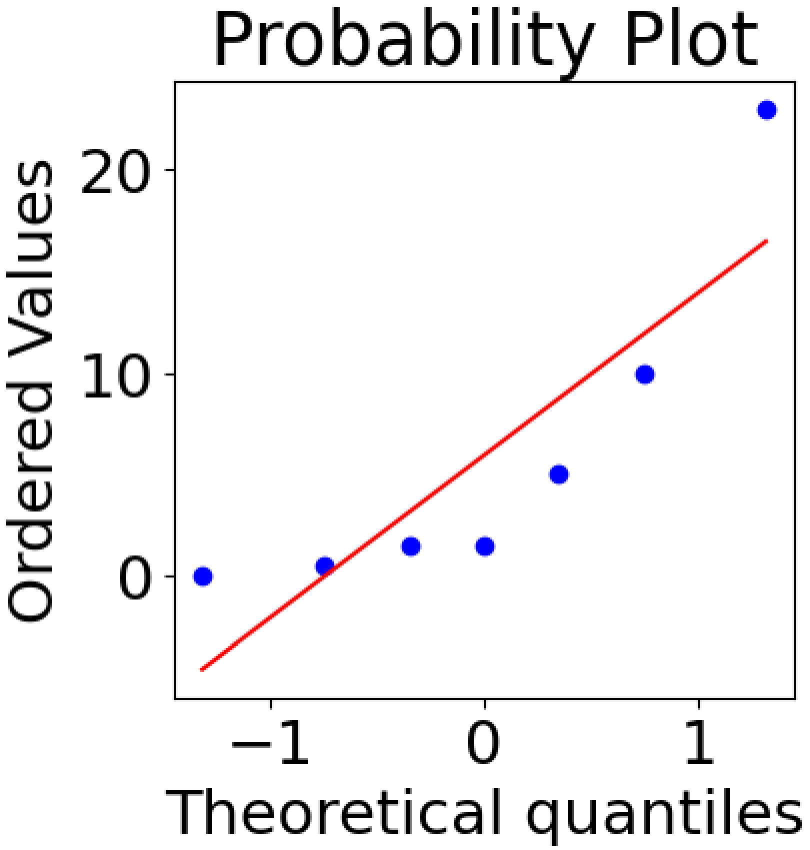


6.2 QQ-plot

```
from matplotlib.pyplot import *
import scipy.stats as stats
import statsmodels.api as sm
fig = figure(figsize=(4,4))


stats.probplot(effects_df["Standardized effect"], dist="norm", plot=plt)

#sm.qqplot(effects_df["Standardized effect"],line ='45')
```



7. Present polynomial response surface

```
s = "yhat = "

s += "%0.3f "%(results['y'].mean())

for i,k in enumerate(average_main_effects.keys()):
    if(average_main_effects[k]<0):
        s += "%0.3f %s "%( average_main_effects[k]/2.0, k )
    else:
        s += "+ %0.3f %s "%( average_main_effects[k]/2.0, k )

for i,k in enumerate(twoway_effects.keys()):
    if(twoway_effects[k]<0):
        s += " %0.3f %s %s"%( twoway_effects[k]/2.0, k[0],k[1])
    else:
        s += "+ %0.3f %s %s"%( twoway_effects[k]/2.0, k[0],k[1])

for i,k in enumerate(threeway_effects.keys()):
    if(threeway_effects[k]<0):
        s += " %0.3f %s %s %s"%( threeway_effects[k]/2.0, k[0],k[1], k[2])
    else:
        s += "+ %0.3f %s %s %s"%( threeway_effects[k]/2.0, k[0],k[1], k[2])


print(s)
```

```
yhat = 71.250 + 11.500 S + 0.750 T -2.500 C + 5.000 S T+ 0.750 S C+ 0.000 T C+ 0.250 S T C
```