Imports and Functions

In [1]:
```
# Import Statements

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.dates import *
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import scipy.stats as stats
import statsmodels.graphics.gofplots as smg
```

Efficiency (Time) Data Analysis

In [2]:
```
time_data = pd.read_csv("2Dv1D CNB Efficiency Times.csv")
time_data
```

Out[2]:

	SID	Task	Condition	Layout	Time
0	1D_1128_1	Find	1Dv2D	1D	72
1	1D_1128_1	Graph Compare	1Dv2D	1D	171
2	1D_1128_1	Number Compare	1Dv2D	1D	195
3	1D_1128_1	Parameter Tuning	1Dv2D	1D	491
4	1D_1128_1	Code Comparison	1Dv2D	1D	333
...	...	...	...	...	...
295	2D_1210_1	Find	2Dv1D	2D	12
296	2D_1210_1	Graph Compare	2Dv1D	2D	42
297	2D_1210_1	Number Compare	2Dv1D	2D	24
298	2D_1210_1	Parameter Tuning	2Dv1D	2D	174
299	2D_1210_1	Code Comparison	2Dv1D	2D	69

300 rows × 5 columns

Find Task

In [3]:
```
find_task = time_data[time_data["Task"] == "Find"]
```

In [4]:
```
model = ols('Time ~ C(Condition) + C(Layout) + C(Condition):C(Layout)', data=find_task
```

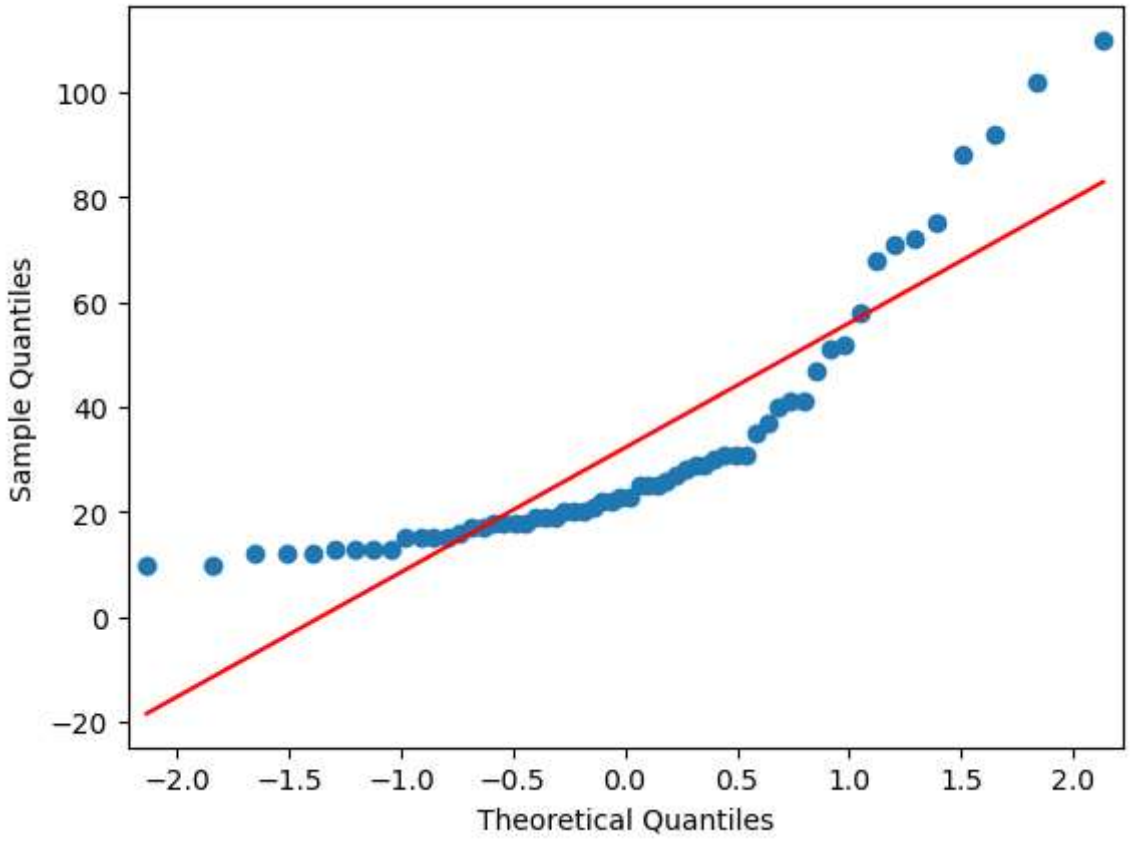
```
sm.stats.anova_lm(model, typ=2)
```

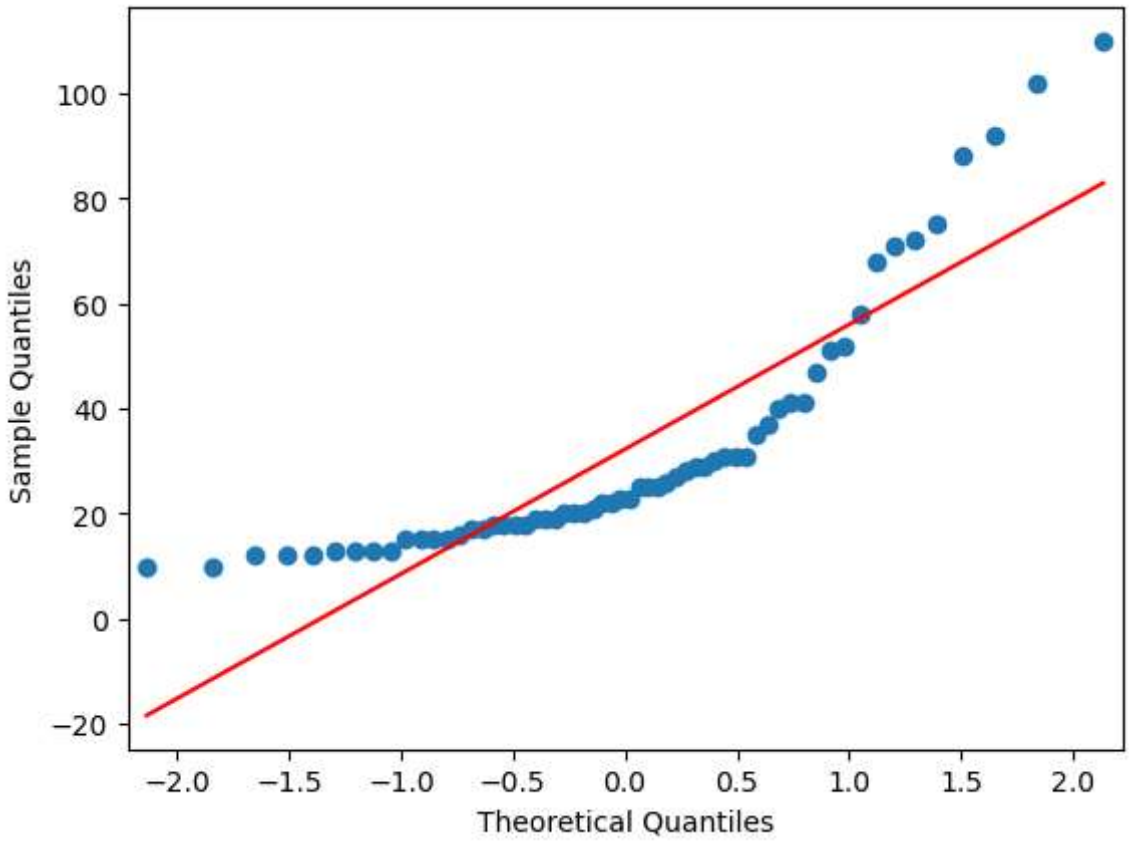
Out[4]:

	sum_sq	df	F	PR(>F)
C(Condition)	2870.416667	1.0	5.379366	0.024049
C(Layout)	660.016667	1.0	1.236918	0.270818
C(Condition):C(Layout)	421.350000	1.0	0.789640	0.378010
Residual	29881.466667	56.0	NaN	NaN

In [5]:
```
stats.chisquare(find_task["Time"])
smg.ProbPlot(find_task["Time"]).qqplot(line='s')
```

Out[5]:





In [6]:
```
find_task_1D = find_task[find_task["Layout"]=="1D"]
find_task_2D = find_task[find_task["Layout"] == "2D"]
find_task_2D
```

Out[6]:

	SID	Task	Condition	Layout	Time
5	1D_1128_1	Find	1Dv2D	2D	88
15	1D_1129_2	Find	1Dv2D	2D	102
25	1D_1129_4	Find	1Dv2D	2D	52
35	1D_1130_2	Find	1Dv2D	2D	51
45	1D_1130_6	Find	1Dv2D	2D	30
55	1D_1201_1	Find	1Dv2D	2D	10
65	1D_1202_1	Find	1Dv2D	2D	19
75	1D_1202_3	Find	1Dv2D	2D	12
85	1D_1203_1	Find	1Dv2D	2D	13
95	1D_1203_3	Find	1Dv2D	2D	25
105	1D_1205_1	Find	1Dv2D	2D	31
115	1D_1206_2	Find	1Dv2D	2D	15
125	1D_1207_2	Find	1Dv2D	2D	23
135	1D_1208_2	Find	1Dv2D	2D	12
145	1D_1212_1	Find	1Dv2D	2D	15
155	2D_1129_1	Find	2Dv1D	2D	68
165	2D_1129_3	Find	2Dv1D	2D	18
175	2D_1130_1	Find	2Dv1D	2D	31
185	2D_1130_3	Find	2Dv1D	2D	13
195	2D_1130_5	Find	2Dv1D	2D	13
205	2D_1130_7	Find	2Dv1D	2D	13
215	2D_1201_2	Find	2Dv1D	2D	58
225	2D_1202_2	Find	2Dv1D	2D	22
235	2D_1202_4	Find	2Dv1D	2D	29
245	2D_1203_2	Find	2Dv1D	2D	15
255	2D_1203_4	Find	2Dv1D	2D	18
265	2D_1206_1	Find	2Dv1D	2D	17
275	2D_1207_1	Find	2Dv1D	2D	22
285	2D_1208_1	Find	2Dv1D	2D	21
295	2D_1210_1	Find	2Dv1D	2D	12

In [7]:
```
stats.kruskal(find_task_1D['Time'], find_task_2D["Time"])
```

Out[7]:
```
KruskalResult(statistic=3.7282601435966094, pvalue=0.0534990929878251)
```

Graph Compare Task

In [8]:
```
graphCom_task = time_data[time_data["Task"] == "Graph Compare"]
```

In [9]:
```
model = ols('Time ~ C(Condition) + C(Layout) + C(Condition):C(Layout)', data=graphCom_
sm.stats.anova_lm(model, typ=2)
```

Out[9]:

	sum_sq	df	F	PR(>F)
C(Condition)	2160.000000	1.0	2.695556	0.106233
C(Layout)	14415.000000	1.0	17.989089	0.000084
C(Condition):C(Layout)	3872.066667	1.0	4.832116	0.032084
Residual	44873.866667	56.0	NaN	NaN

In [10]:
```
tukey_layout = pairwise_tukeyhsd(graphCom_task['Time'], graphCom_task['Layout'], 0.05)
print(tukey_layout)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper   reject
----------------------------------------------------
    1D     2D    -31.0 0.0002 -46.3118 -15.6882   True
----------------------------------------------------
```

In [11]:
```
graphCom_task['Combos'] = graphCom_task['Condition'] + "-" + graphCom_task['Layout']
```

```
C:\Users\harde\AppData\Local\Temp\ipykernel_18196\3858717447.py:1: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  graphCom_task['Combos'] = graphCom_task['Condition'] + "-" + graphCom_task['Layou
t']
```

In [12]:
```
tukey_combos = pairwise_tukeyhsd(graphCom_task['Time'], graphCom_task['Combos'], 0.05)
print(tukey_combos)
```

```
   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=========================================================
 group1   group2  meandiff p-adj   lower    upper   reject
---------------------------------------------------------
1Dv2D-1D 1Dv2D-2D -47.0667 0.0002 -74.4365 -19.6969   True
1Dv2D-1D 2Dv1D-1D -28.0667 0.0424 -55.4365  -0.6969   True
1Dv2D-1D 2Dv1D-2D    -43.0 0.0006 -70.3698 -15.6302   True
1Dv2D-2D 2Dv1D-1D     19.0 0.2666  -8.3698  46.3698  False
1Dv2D-2D 2Dv1D-2D   4.0667 0.9791 -23.3031  31.4365  False
2Dv1D-1D 2Dv1D-2D -14.9333 0.4774 -42.3031  12.4365  False
---------------------------------------------------------
```

NOTE: meandiff = group2 - group1, as seen here:

https://github.com/statsmodels/statsmodels/issues/8458

The practice effect seems to have overwhelmed the effect of the layout in the 2Dv1D condition, and when comparing the second item in each order (e.g. 2D for 1Dv2D and 1D for 2Dv1D), the results weren't significant. Also, for the second layout in each order, the results were not significant. Otherwise, 2D came out on top.

In [13]:
```
print("Overall Mean Time to Completion")
graphCom_task['Time'].mean()
```

```
Overall Mean Time to Completion
80.46666666666667
```

Out[13]:

In [14]:
```
print("Mean Time to Completion by Layout")
graphCom_task.groupby('Layout').mean()
```

```
Mean Time to Completion by Layout
```

Out[14]:

Layout	Time
1D	95.966667
2D	64.966667

In [15]:
```
print("Mean Time to Completion by Condition and Layout")
graphCom_task.groupby(['Condition', 'Layout']).mean()
```

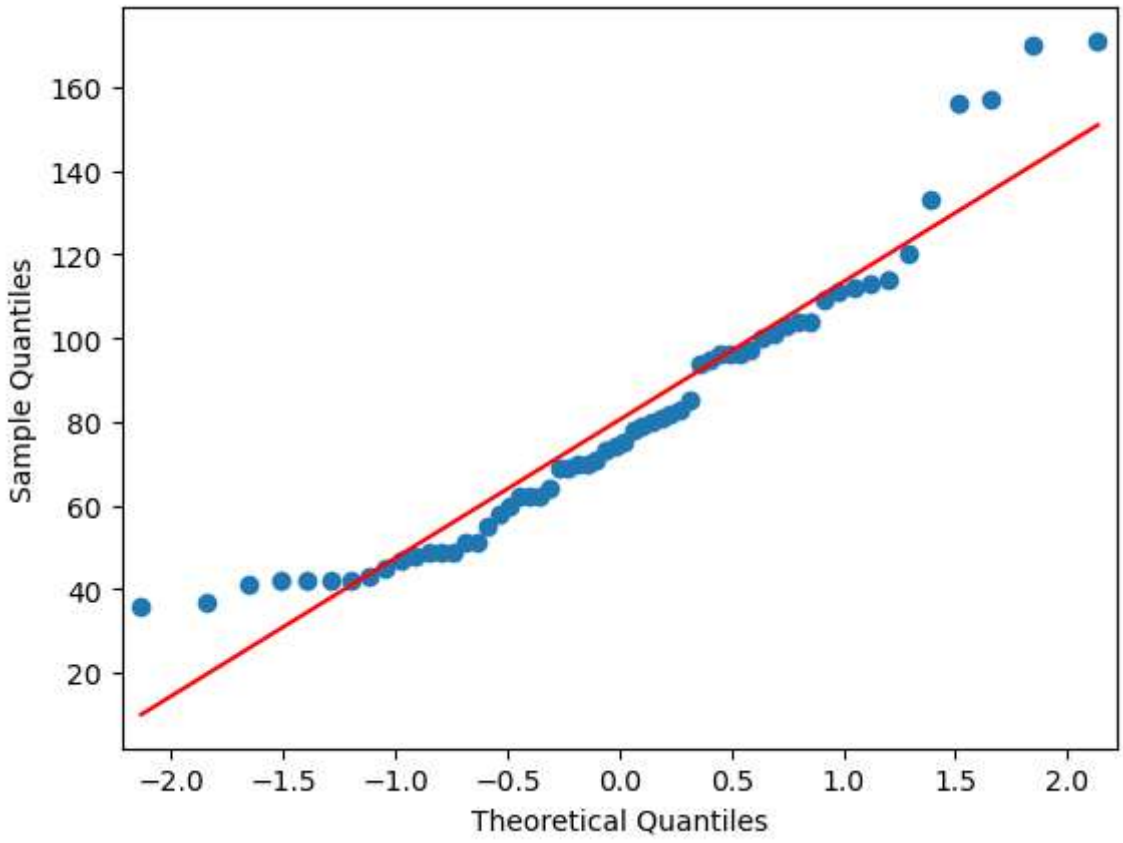
```
Mean Time to Completion by Condition and Layout
```

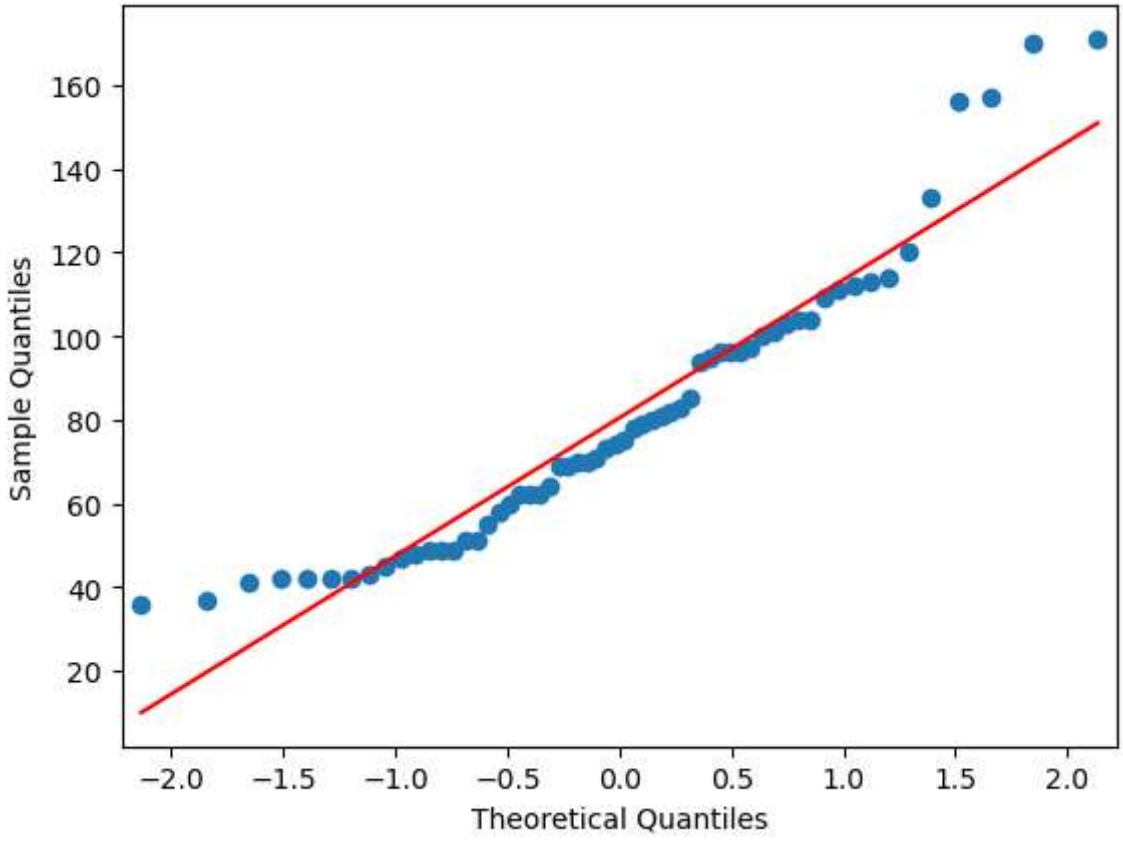
Out[15]:

Condition	Layout	Time
1Dv2D	**1D**	110.000000
	2D	62.933333
2Dv1D	**1D**	81.933333
	2D	67.000000

In [16]:
```
smg.ProbPlot(graphCom_task["Time"]).qqplot(line='s')
```

Out[16]:





In [17]:
```
stats.chisquare(graphCom_task["Time"])
```

Out[17]:
```
Power_divergenceResult(statistic=811.7763048881525, pvalue=7.63239003560691e-133)
```

In [18]:
```
graphCom_task_1D = graphCom_task[graphCom_task["Layout"]=="1D"]
graphCom_task_2D = graphCom_task[graphCom_task["Layout"] == "2D"]
```

In [19]:
```
stats.kruskal(graphCom_task_1D['Time'], graphCom_task_2D['Time'])
```

Out[19]:
```
KruskalResult(statistic=16.783490898305708, pvalue=4.189621859098256e-05)
```

Number Compare Task

In [20]:
```
numCom_task = time_data[time_data["Task"] == "Number Compare"]
model = ols('Time ~ C(Condition) + C(Layout) + C(Condition):C(Layout)', data=numCom_ta
sm.stats.anova_lm(model, typ=2)
```

Out[20]:

	sum_sq	df	F	PR(>F)
C(Condition)	3480.816667	1.0	5.435925	0.023350
C(Layout)	11234.016667	1.0	17.543948	0.000100
C(Condition):C(Layout)	5023.350000	1.0	7.844869	0.006983
Residual	35858.800000	56.0	NaN	NaN

In [21]:
```
numCom_task['Combos'] = numCom_task['Condition'] + "-" + numCom_task['Layout']
```

```
C:\Users\harde\AppData\Local\Temp\ipykernel_18196\2701913620.py:1: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  numCom_task['Combos'] = numCom_task['Condition'] + "-" + numCom_task['Layout']
```

In [22]:
```
tukey_layout = pairwise_tukeyhsd(numCom_task['Time'], numCom_task['Layout'], 0.05)
print(tukey_layout)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
====================================================
group1 group2 meandiff p-adj   lower    upper   reject
----------------------------------------------------
    1D     2D -27.3667 0.0003 -41.6607 -13.0727   True
----------------------------------------------------
```

In [23]:
```
tukey_combos = pairwise_tukeyhsd(numCom_task['Time'], numCom_task['Combos'], 0.05)
print(tukey_combos)
```

```
   Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
 group1   group2  meandiff p-adj   lower    upper   reject
-----------------------------------------------------
1Dv2D-1D 1Dv2D-2D -45.6667    0.0 -70.1332 -21.2001   True
1Dv2D-1D 2Dv1D-1D -33.5333 0.0034 -57.9999  -9.0668   True
1Dv2D-1D 2Dv1D-2D    -42.6 0.0001 -67.0665 -18.1335   True
1Dv2D-2D 2Dv1D-1D  12.1333 0.5586 -12.3332  36.5999  False
1Dv2D-2D 2Dv1D-2D   3.0667 0.9873 -21.3999  27.5332  False
2Dv1D-1D 2Dv1D-2D  -9.0667 0.7607 -33.5332  15.3999  False
-----------------------------------------------------
```

The practice effect (repeated measures issue) seems to have overwhelmed the 2D layout effect when comparing 2Dv1D condition 1D v 2D; this may explain the results not being significant for that pair. Otherwise, 2D came out on top!

In [24]:
```
print("Overall Mean Time to Completion")
numCom_task['Time'].mean()
```

```
Overall Mean Time to Completion
45.483333333333334
```

Out[24]:

In [25]:
```
print("Mean Time to Completion by Layout")
numCom_task.groupby('Layout').mean()
```

```
Mean Time to Completion by Layout
```

Out[25]:

Layout	Time
1D	59.166667
2D	31.800000

In [26]:
```
print("Mean Time to Completion by Condition and Layout")
numCom_task.groupby(['Condition', 'Layout']).mean()
```

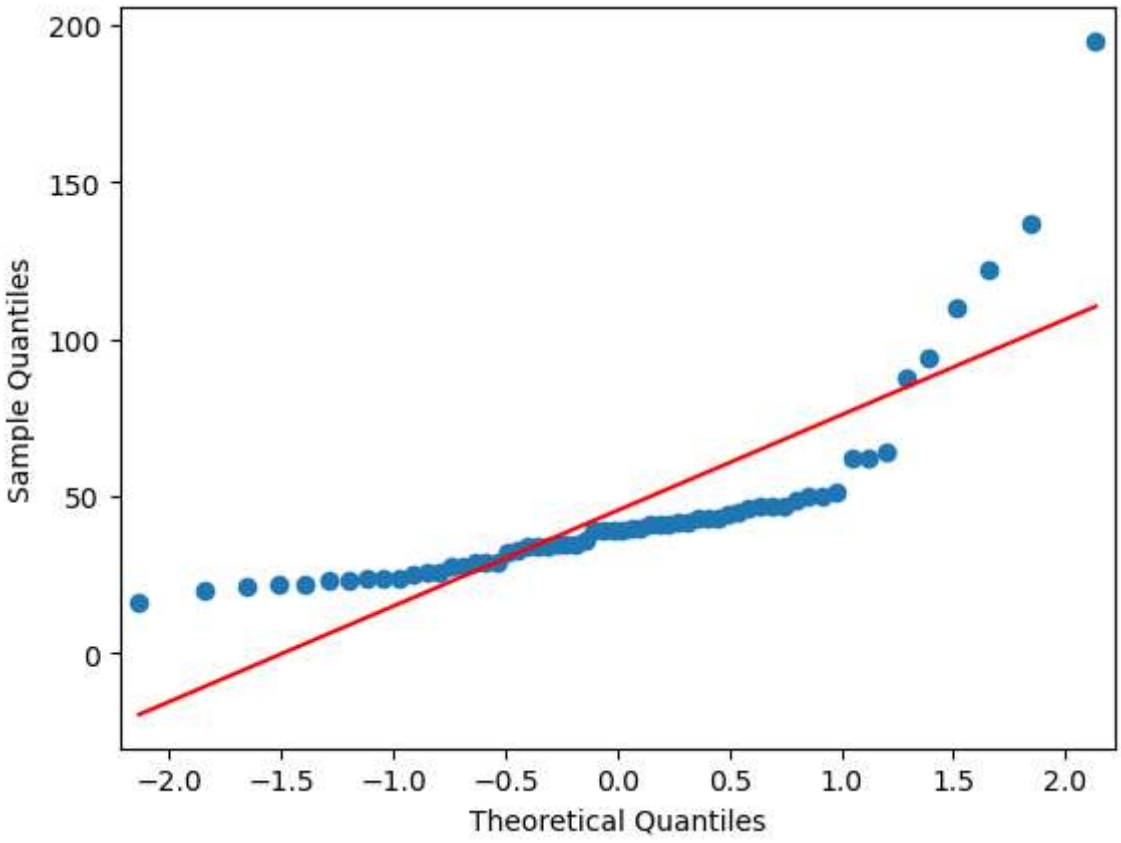
```
Mean Time to Completion by Condition and Layout
```

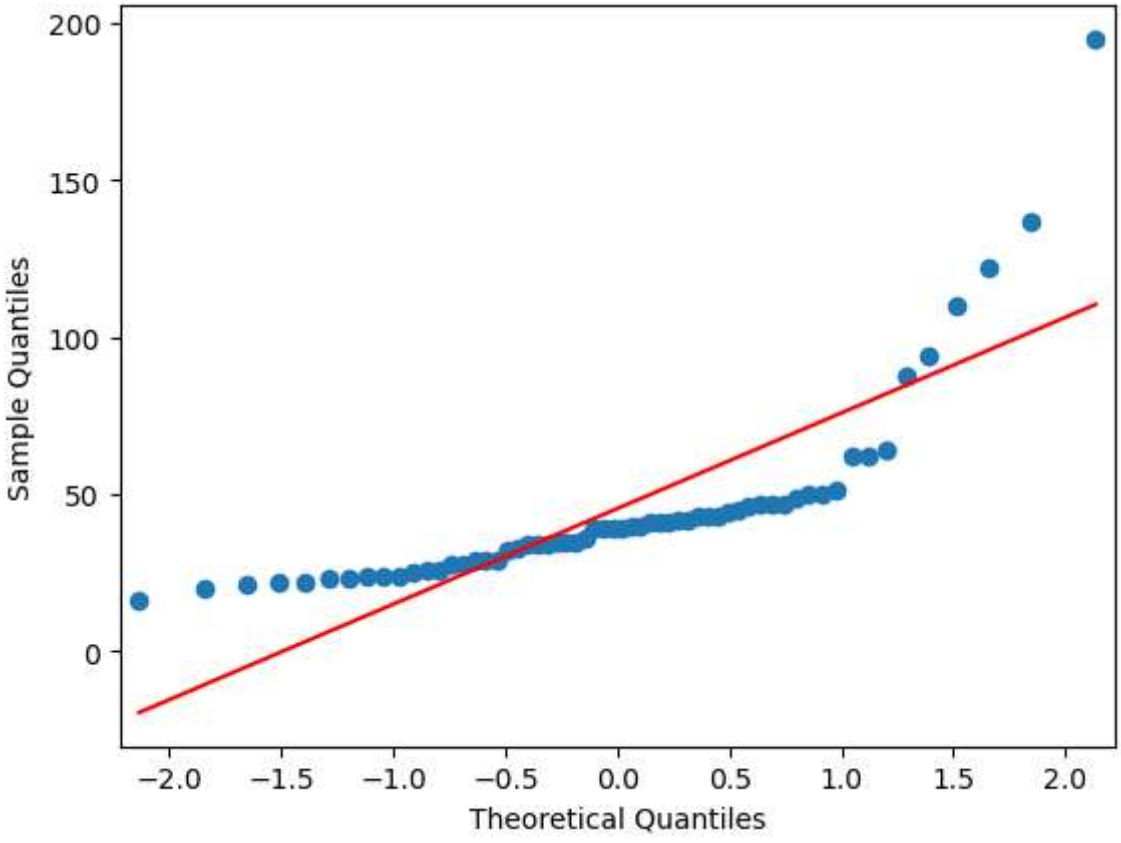
Out[26]:

Condition	Layout	Time
1Dv2D	**1D**	75.933333
	2D	30.266667
2Dv1D	**1D**	42.400000
	2D	33.333333

In [27]:
```
smg.ProbPlot(numCom_task["Time"]).qqplot(line='s')
```

Out[27]:





In [28]:
```
stats.chisquare(numCom_task["Time"])
```

Out[28]:
```
Power_divergenceResult(statistic=1222.3594723341885, pvalue=6.040360569828294e-217)
```

In [29]:
```
numCom_task_1D = numCom_task[numCom_task["Layout"]=="1D"]
numCom_task_2D = numCom_task[numCom_task["Layout"] == "2D"]
```

In [30]:
```
stats.kruskal(numCom_task_1D['Time'], numCom_task_2D['Time'])
```

Out[30]:
```
KruskalResult(statistic=20.560103679798207, pvalue=5.77879767610255e-06)
```

Parameter Tuning Task

In [31]:
```
pt_task = time_data[time_data["Task"] == "Parameter Tuning"]
model = ols('Time ~ C(Condition) + C(Layout) + C(Condition):C(Layout)', data=pt_task).
sm.stats.anova_lm(model, typ=2)
```

Out[31]:

	sum_sq	df	F	PR(>F)
C(Condition)	36.816667	1.0	0.006740	0.934860
C(Layout)	42933.750000	1.0	7.860318	0.006931
C(Condition):C(Layout)	8425.350000	1.0	1.542514	0.219420
Residual	305876.933333	56.0	NaN	NaN

In [32]:
```
tukey_layout = pairwise_tukeyhsd(pt_task['Time'], pt_task['Layout'], 0.05)
print(tukey_layout)
```

```
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj   lower    upper   reject
-----------------------------------------------------
    1D     2D    -53.5 0.0067 -91.5489 -15.4511   True
-----------------------------------------------------
```

2D came out on top in terms of efficiency for this task!

In [33]:
```
print("Overall Mean Time to Completion")
pt_task['Time'].mean()
```

Out[33]:
```
Overall Mean Time to Completion
250.95
```

In [34]:
```
print("Mean Time to Completion by Layout")
pt_task.groupby('Layout').mean()
```

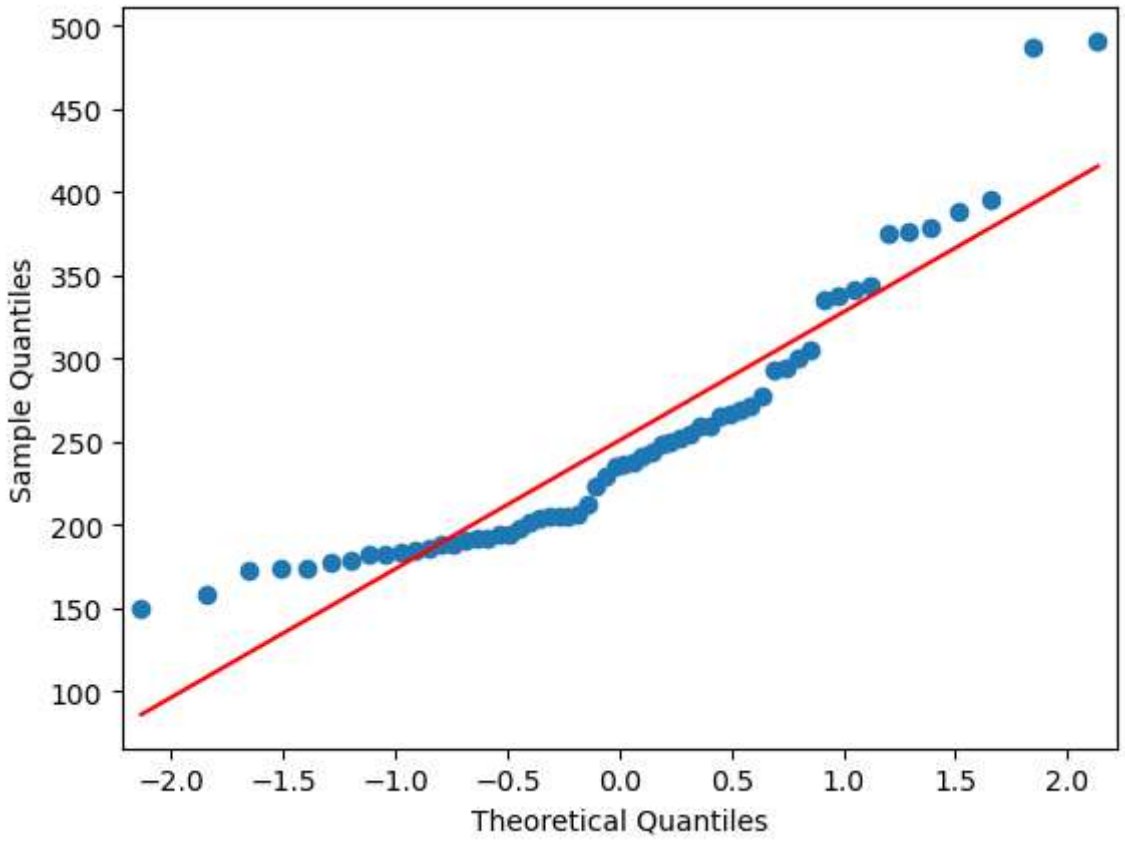
Mean Time to Completion by Layout

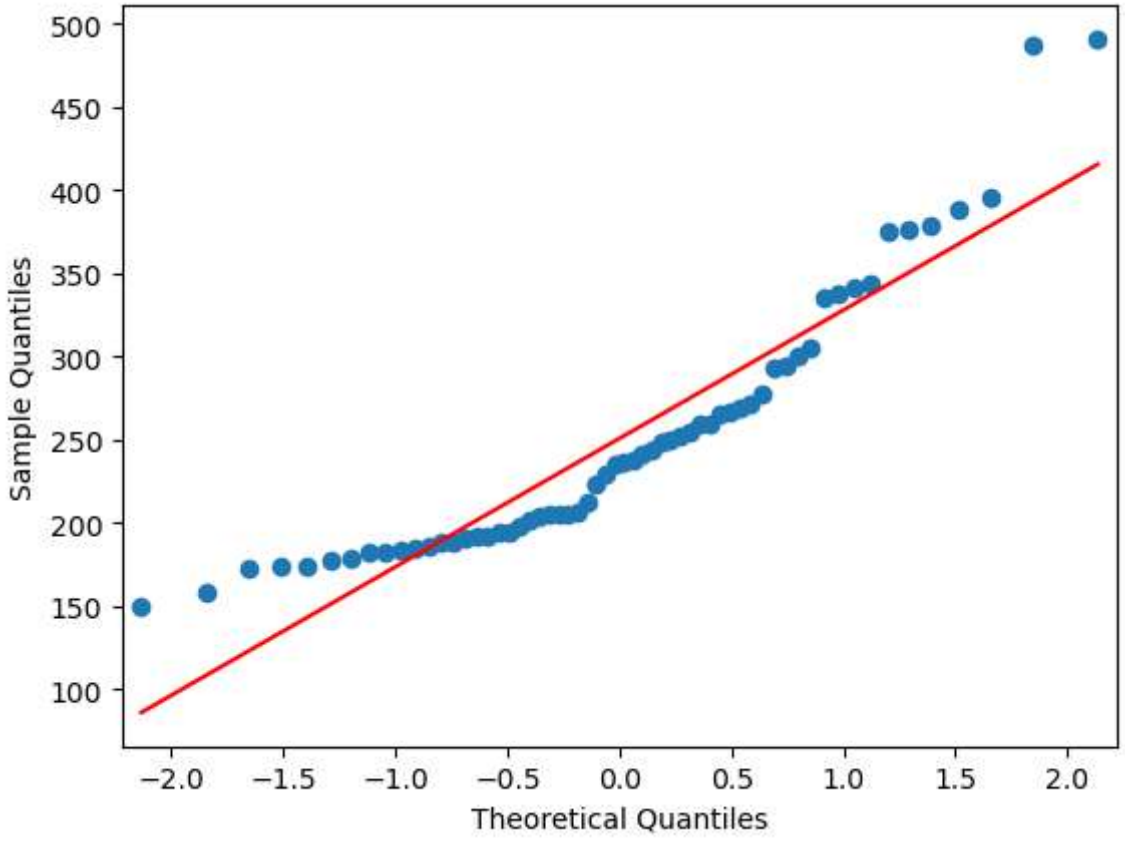
Out[34]:

Layout	Time
1D	277.7
2D	224.2

In [35]:
```
smg.ProbPlot(pt_task["Time"]).qqplot(line='s')
```

Out[35]:





In [36]:
```
stats.chisquare(pt_task["Time"])
```

Out[36]:
```
Power_divergenceResult(statistic=1423.6814106395695, pvalue=8.885280973267302e-259)
```

In [37]:
```
pt_task_1D = pt_task[pt_task["Layout"]=="1D"]
pt_task_2D = pt_task[pt_task["Layout"] == "2D"]
stats.kruskal(pt_task_1D['Time'], pt_task_2D['Time'])
```

Out[37]:
```
KruskalResult(statistic=8.57156197887718, pvalue=0.0034145409792734004)
```

Code Comparison Task

In [38]:
```
cc_task = time_data[time_data["Task"] == "Code Comparison"]
model = ols('Time ~ C(Condition) + C(Layout) + C(Condition):C(Layout)', data=cc_task).
sm.stats.anova_lm(model, typ=2)
```

Out[38]:

	sum_sq	df	F	PR(>F)
C(Condition)	183.750000	1.0	0.041191	0.839904
C(Layout)	58844.016667	1.0	13.191141	0.000611
C(Condition):C(Layout)	65538.150000	1.0	14.691774	0.000323
Residual	249808.933333	56.0	NaN	NaN

In [39]:
```
tukey_layout = pairwise_tukeyhsd(cc_task['Time'], cc_task['Layout'], 0.05)
print(tukey_layout)
```

```
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj    lower    upper   reject
-----------------------------------------------------
    1D     2D -62.6333 0.0017 -100.7543 -24.5124   True
-----------------------------------------------------
```

In [40]:
```
cc_task['Combos'] = cc_task['Condition'] + "-" + cc_task['Layout']
```

```
C:\Users\harde\AppData\Local\Temp\ipykernel_18196\1895316253.py:1: SettingWithCopyWar
ning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cc_task['Combos'] = cc_task['Condition'] + "-" + cc_task['Layout']
```

In [41]:
```
tukey_combos = pairwise_tukeyhsd(cc_task['Time'], cc_task['Combos'], 0.05)
print(tukey_combos)
```

```
    Multiple Comparison of Means - Tukey HSD, FWER=0.05
=======================================================
 group1   group2   meandiff p-adj    lower    upper   reject
-------------------------------------------------------
1Dv2D-1D 1Dv2D-2D -128.7333    0.0 -193.3105 -64.1562   True
1Dv2D-1D 2Dv1D-1D     -69.6 0.0299 -134.1771  -5.0229   True
1Dv2D-1D 2Dv1D-2D  -66.1333 0.0428 -130.7105  -1.5562   True
1Dv2D-2D 2Dv1D-1D   59.1333  0.084   -5.4438 123.7105  False
1Dv2D-2D 2Dv1D-2D      62.6 0.0607   -1.9771 127.1771  False
2Dv1D-1D 2Dv1D-2D    3.4667  0.999  -61.1105  68.0438  False
-------------------------------------------------------
```

The practice effect (repeated measures issue) seems to have overwhelmed the effect of 2D layouts for 2Dv1D. Also, the results of the second layout in each order were not significant. Still, 2D came out on top!

In [42]:
```
print("Overall Mean Time to Completion")
cc_task['Time'].mean()
```

```
Overall Mean Time to Completion
150.95
```

Out[42]:

In [43]:
```
print("Mean Time to Completion by Layout")
cc_task.groupby('Layout').mean()
```

```
Mean Time to Completion by Layout
```

Out[43]:

Layout	Time
1D	182.266667
2D	119.633333

In [44]:
```
print("Mean Time to Completion by Condition and Layout")
cc_task.groupby(['Condition', 'Layout']).mean()
```

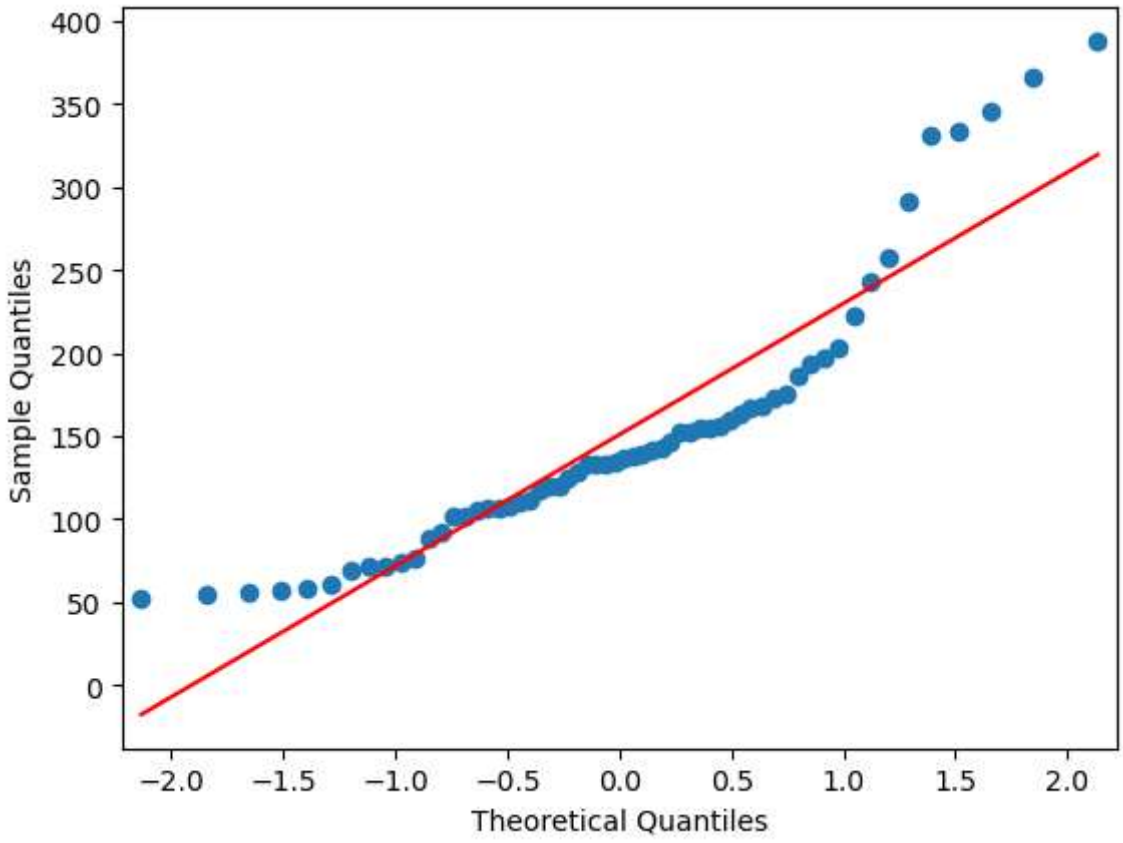
```
Mean Time to Completion by Condition and Layout
```

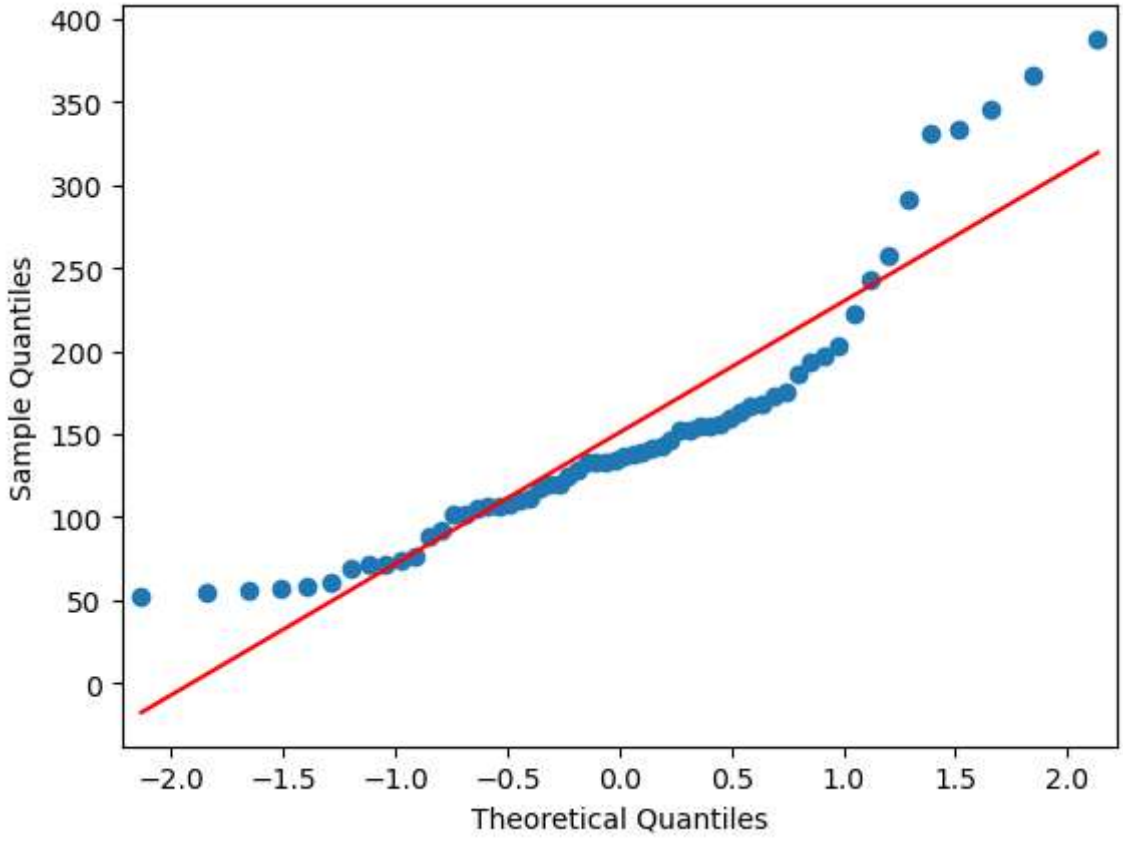
Out[44]:

Condition	Layout	Time
1Dv2D	**1D**	217.066667
	2D	88.333333
2Dv1D	**1D**	147.466667
	2D	150.933333

In [45]:
```
smg.ProbPlot(cc_task["Time"]).qqplot(line='s')
```

Out[45]:





In [46]:
```
stats.chisquare(cc_task["Time"])
```

Out[46]:
```
Power_divergenceResult(statistic=2480.1248757866842, pvalue=0.0)
```

In [47]:
```
cc_task_1D = cc_task[cc_task["Layout"]=="1D"]
cc_task_2D = cc_task[cc_task["Layout"] == "2D"]
stats.kruskal(cc_task_1D['Time'], cc_task_2D['Time'])
```

Out[47]:
```
KruskalResult(statistic=12.804076225415258, pvalue=0.00034586495138787547)
```

Post-1D & 2D Likert Data Analysis

In [48]:
```
post1d2d_data = pd.read_csv("Post1D2D_Likert_data.csv")
post1d2d_data
```

Out[48]:

	SID	Condition	Layout	Question	Rating
0	1D_1128_1	1Dv2D	1D	Easy Nav	2
1	1D_1128_1	1Dv2D	1D	Quick Find Info	2
2	1D_1128_1	1Dv2D	1D	Easy VizCompare	2
3	1D_1128_1	1Dv2D	1D	Easy NumCompare	2
4	1D_1128_1	1Dv2D	1D	Easy CodeCompare	1
...	...	...	...	...	...
295	2D_1210_1	2Dv1D	2D	Easy Nav	3
296	2D_1210_1	2Dv1D	2D	Quick Find Info	3
297	2D_1210_1	2Dv1D	2D	Easy VizCompare	3
298	2D_1210_1	2Dv1D	2D	Easy NumCompare	3
299	2D_1210_1	2Dv1D	2D	Easy CodeCompare	3

300 rows × 5 columns

Easy Navigation

In [49]:
```
easyNav_data = post1d2d_data[post1d2d_data['Question'] == 'Easy Nav']
```

In [50]:
```
easyNav_1d = easyNav_data[easyNav_data['Layout'] == '1D']
easyNav_2d = easyNav_data[easyNav_data['Layout'] == '2D']
```

In [51]:
```
t_results = stats.ttest_rel(easyNav_1d['Rating'], easyNav_2d['Rating'])
t_results
```

Out[51]:
```
Ttest_relResult(statistic=-5.520616976049565, pvalue=5.975635924933263e-06)
```

In [52]:
```
w_results = stats.wilcoxon(easyNav_1d['Rating'], easyNav_2d['Rating'])
w_results
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\_morestats.py:3255: UserWarnin
g: Exact p-value calculation does not work if there are zeros. Switching to normal ap
proximation.
  warnings.warn("Exact p-value calculation does not work if there are "
```

Out[52]:
```
WilcoxonResult(statistic=9.0, pvalue=4.9034562984353145e-05)
```

Both the paired t-test and the Wilcoxon test found that there was a statistically significant difference between the layout groups. Now to calculate the average difference and see if it is positive or negative.

In [53]:
```
easyNav_diff = easyNav_1d['Rating'].to_numpy() - easyNav_2d['Rating'].to_numpy()
easyNav_diff
```

Out[53]:
```
array([-1, -2, -5, -3,  0, -1,  0, -2,  0, -5,  1, -1, -1, -2, -4,  0,  0,
       -1, -2, -3, -1, -4, -6, -4, -4, -2,  0,  1, -2, -2], dtype=int64)
```

In [54]:
```
easyNav_mean = np.mean(easyNav_diff)
easyNav_med = np.median(easyNav_diff)
print("Mean: " + str(easyNav_mean))
print("Median: " + str(easyNav_med))
```

```
Mean: -1.8666666666666667
Median: -2.0
```

Since both the mean and median are negative, and our equation was 1D - 2D, this means that 2D layouts were generally considered more usable for ease of navigation.

Quick Find Info

In [55]:
```
quickFind_data = post1d2d_data[post1d2d_data['Question'] == 'Quick Find Info']
```

In [56]:
```
quickFind_1d = quickFind_data[quickFind_data['Layout'] == '1D']
quickFind_2d = quickFind_data[quickFind_data['Layout'] == '2D']
```

In [57]:
```
t_results = stats.ttest_rel(quickFind_1d['Rating'], quickFind_2d['Rating'])
t_results
```

Out[57]:
```
Ttest_relResult(statistic=-5.571719022297886, pvalue=5.187110861774126e-06)
```

In [58]:
```
w_results = stats.wilcoxon(quickFind_1d['Rating'], quickFind_2d['Rating'])
w_results
```

Out[58]:
```
WilcoxonResult(statistic=12.0, pvalue=6.891893067383647e-05)
```

Both the paired t-test and the Wilcoxon test found that there was a statistically significant difference between the layout groups. Now to calculate the average difference and see if it is positive or negative.

In [59]:
```
quickFind_diff = quickFind_1d['Rating'].to_numpy() - quickFind_2d['Rating'].to_numpy()
quickFind_diff
```

Out[59]:
```
array([-1, -2, -4, -2,  0,  0,  0, -2, -1, -5,  0,  0, -1, -3, -5,  0, -4,
       -1, -2, -1, -2, -3, -6, -2, -3, -2, -1, -1,  2, -2], dtype=int64)
```

In [60]:
```
quickFind_mean = np.mean(quickFind_diff)
quickFind_med = np.median(quickFind_diff)
print("Mean: " + str(quickFind_mean))
print("Median: " + str(quickFind_med))
```

```
Mean: -1.8
Median: -2.0
```

Since both the mean and median are negative, and our equation was 1D - 2D, this means that 2D layouts were generally considered more usable in terms of quickly finding information.

Easy VizCompare

In [61]:
```
vizCom_data = post1d2d_data[post1d2d_data['Question'] == 'Easy VizCompare']
vizCom_1d = vizCom_data[vizCom_data['Layout'] == '1D']
vizCom_2d = vizCom_data[vizCom_data['Layout'] == '2D']
```

In [62]:
```
t_results = stats.ttest_rel(vizCom_1d['Rating'], vizCom_2d['Rating'])
t_results
```

Out[62]:
```
Ttest_relResult(statistic=-8.31259446888417, pvalue=3.6560239119618315e-09)
```

In [63]:
```
w_results = stats.wilcoxon(vizCom_1d['Rating'], vizCom_2d['Rating'])
w_results
```

Out[63]:
```
WilcoxonResult(statistic=0.0, pvalue=1.1061759441432599e-05)
```

Both the paired t-test and the Wilcoxon test found that there was a statistically significant difference between the layout groups. Now to calculate the average difference and see if it is positive or negative.

In [64]:
```
vizCom_diff = vizCom_1d['Rating'].to_numpy() - vizCom_2d['Rating'].to_numpy()
vizCom_diff
```

Out[64]:
```
array([ 0, -3, -5, -1, -2,  0, -4, -3, -3, -5,  0,  0, -1, -4, -5, -2, -4,
       -2,  0, -4, -4, -4, -6, -3, -5, -1, -4, -5, -1, -5], dtype=int64)
```

In [65]:
```
vizCom_mean = np.mean(vizCom_diff)
vizCom_med = np.median(vizCom_diff)
print("Mean: " + str(vizCom_mean))
print("Median: " + str(vizCom_med))
```

```
Mean: -2.8666666666666667
Median: -3.0
```

Since both the mean and median are negative, and our equation was 1D - 2D, this means that 2D layouts were generally considered more usable in terms of comparing visualizations. This difference was also more significant (-3.0) than easy navigation or quickly finding information, which may indicate that the effects of a 2D layout on visual comparisons are stronger in usability than in the latter two.

Easy NumCompare

In [66]:
```
numCom_data = post1d2d_data[post1d2d_data['Question'] == 'Easy NumCompare']
numCom_1d = numCom_data[numCom_data['Layout'] == '1D']
numCom_2d = numCom_data[numCom_data['Layout'] == '2D']
```

In [67]:
```
t_results = stats.ttest_rel(numCom_1d['Rating'], numCom_2d['Rating'])
t_results
```

Out[67]:
```
Ttest_relResult(statistic=-7.623691083853344, pvalue=2.0948890712051675e-08)
```

In [68]:
```
w_results = stats.wilcoxon(numCom_1d['Rating'], numCom_2d['Rating'])
w_results
```

Out[68]:
```
WilcoxonResult(statistic=4.0, pvalue=7.7560571055044e-06)
```

Both the paired t-test and the Wilcoxon test found that there was a statistically significant difference between the layout groups. Now to calculate the average difference and see if it is positive or negative.

In [69]:
```
numCom_diff = numCom_1d['Rating'].to_numpy() - numCom_2d['Rating'].to_numpy()
numCom_diff
```

Out[69]:
```
array([ 0, -4, -5, -1, -2, -1, -2, -2, -4, -4,  1, -1, -1, -5, -4,  0, -1,
       -1,  0, -2, -4, -5, -6, -2, -6, -4, -4, -5, -5, -5], dtype=int64)
```

In [70]:
```
numCom_mean = np.mean(numCom_diff)
numCom_med = np.median(numCom_diff)
print("Mean: " + str(numCom_mean))
print("Median: " + str(numCom_med))
```

```
Mean: -2.8333333333333335
Median: -3.0
```

Since both the mean and median are negative, and our equation was 1D - 2D, this means that 2D layouts were generally considered more usable in terms of comparing numerical results. This difference was also more significant (-3.0) than easy navigation or quickly finding information, which may indicate that the effects of a 2D layout on visual comparisons are stronger in usability than in the latter two.

Easy CodeCompare

In [71]:
```
codeCom_data = post1d2d_data[post1d2d_data['Question'] == 'Easy CodeCompare']
codeCom_1d = codeCom_data[codeCom_data['Layout'] == '1D']
codeCom_2d = codeCom_data[codeCom_data['Layout'] == '2D']
```

In [72]:
```
t_results = stats.ttest_rel(codeCom_1d['Rating'], codeCom_2d['Rating'])
t_results
```

Out[72]:
```
Ttest_relResult(statistic=-10.661467764759559, pvalue=1.513277414009814e-11)
```

In [73]:
```
w_results = stats.wilcoxon(codeCom_1d['Rating'], codeCom_2d['Rating'])
w_results
```

Out[73]:
```
WilcoxonResult(statistic=0.0, pvalue=1.862645149230957e-09)
```

Both the paired t-test and the Wilcoxon test found that there was a statistically significant difference between the layout groups. Now to calculate the average difference and see if it is positive or negative.

In [74]:
```
codeCom_diff = codeCom_1d['Rating'].to_numpy() - codeCom_2d['Rating'].to_numpy()
codeCom_diff
```

Out[74]:
```
array([-2, -5, -5, -1, -2, -1, -2, -3, -4, -5, -1, -2, -1, -6, -4, -4, -1,
       -3, -1, -5, -4, -5, -6, -2, -6, -4, -5, -6, -5, -6], dtype=int64)
```

In [75]:
```
codeCom_mean = np.mean(codeCom_diff)
codeCom_med = np.median(codeCom_diff)
print("Mean: " + str(codeCom_mean))
print("Median: " + str(codeCom_med))
```

```
Mean: -3.566666666666667
Median: -4.0
```

Since both the mean and median are negative, and our equation was 1D - 2D, this means that 2D layouts were generally considered more usable in terms of comparing code. This difference was also more significant (-4.0) than all other ratings, which may indicate that the effects of a 2D layout on visual comparisons can be very helpful in comparing code.