In [1]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in /Applications/anaconda3/lib/
python3.11/site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /Application
s/anaconda3/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Applications/anaconda
3/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /Applications/anacon
da3/lib/python3.11/site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in /Applications/anacond
a3/lib/python3.11/site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in /Applications/anaconda3/li
b/python3.11/site-packages (from python-dateutil>=2.8.2->pandas) (1.1
6.0)
```

In [2]:
```
import pandas as pd
```

In [164]:
```
import pandas as pd

# Read the CSV file, specifying which columns to use and setting the i
df = pd.read_csv('Sample Course Project (Dataset).csv', usecols=lambda

print(df)
```

```
      Close_ETF       oil      gold       JPM
0     97.349998  0.039242  0.004668  0.032258
1     97.750000  0.001953 -0.001366 -0.002948
2     99.160004 -0.031514 -0.007937  0.025724
3     99.650002  0.034552  0.014621  0.011819
4     99.260002  0.013619 -0.011419  0.000855
..          ...       ...       ...       ...
995  150.570007  0.009752  0.004634  0.003859
996  151.600006 -0.009341 -0.015325  0.018259
997  151.300003  0.036120 -0.006195 -0.007928
998  152.619995  0.001542  0.005778 -0.000381
999  152.539993  0.020330  0.001965  0.000381

[1000 rows x 4 columns]
```

Part-1

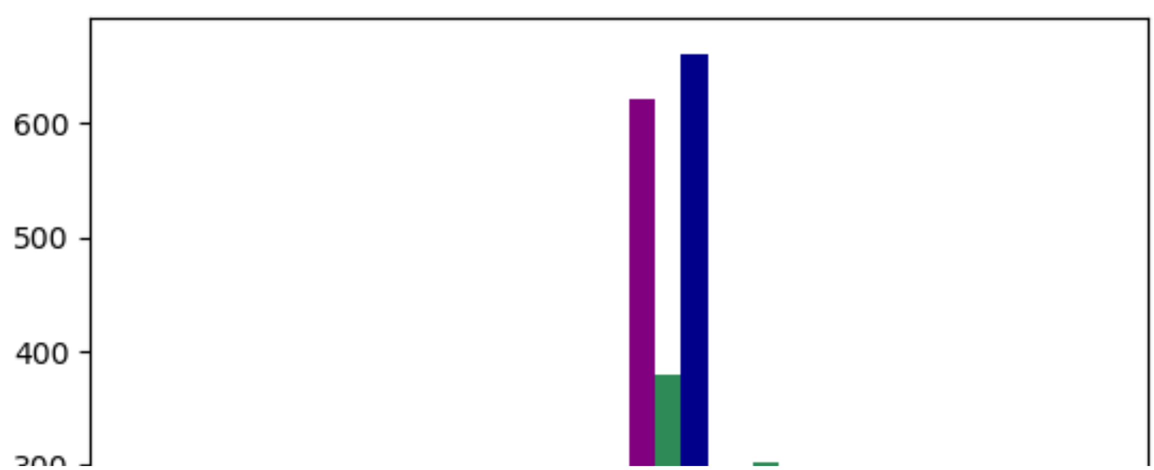
In [212]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
```

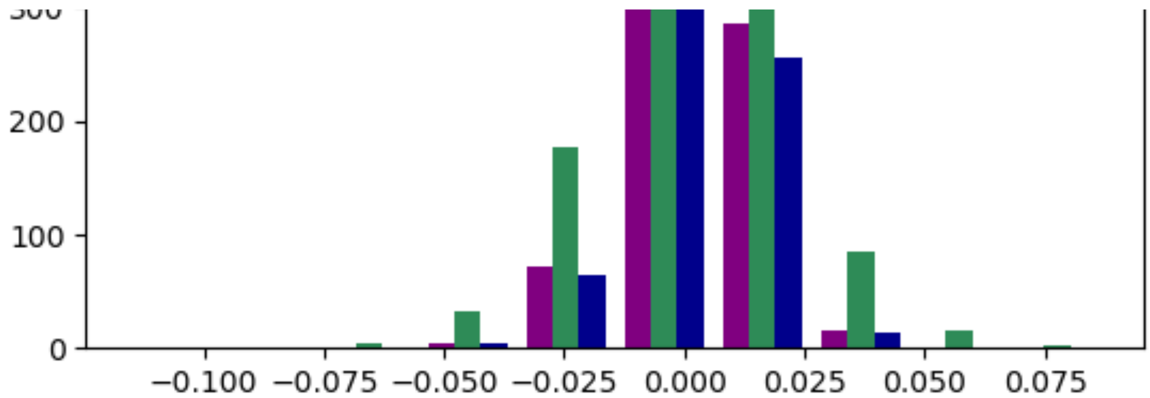
```
# Read the CSV file, excluding the "Unnamed: 4" column
data = pd.read_csv('Sample Course Project (Dataset).csv', usecols=lamb

# Display basic statistics of the data
print(data.describe())

# Plot a histogram of 'gold', 'oil', and 'JPM' columns with specified
plt.hist(data[['gold', 'oil', 'JPM']], color=['purple', 'seagreen', 'c

# Extract columns 'Close_ETF' and 'gold'
col1 = data['Close_ETF']
col2 = data['gold']

# Display the last few rows of the DataFrame
data.tail()
```

```
          Close_ETF          oil         gold          JPM
count  1000.000000  1000.000000  1000.000000  1000.000000
mean    121.152960     0.001030     0.000663     0.000530
std      12.569790     0.021093     0.011289     0.011017
min      96.419998    -0.116533    -0.065805    -0.048217
25%     112.580002    -0.012461    -0.004816    -0.005538
50%     120.150002     0.001243     0.001030     0.000386
75%     128.687497     0.014278     0.007482     0.006966
max     152.619995     0.087726     0.042199     0.057480
```

Out[212]:

	Close_ETF	oil	gold	JPM
995	150.570007	0.009752	0.004634	0.003859
996	151.600006	-0.009341	-0.015325	0.018259
997	151.300003	0.036120	-0.006195	-0.007928
998	152.619995	0.001542	0.005778	-0.000381
999	152.539993	0.020330	0.001965	0.000381

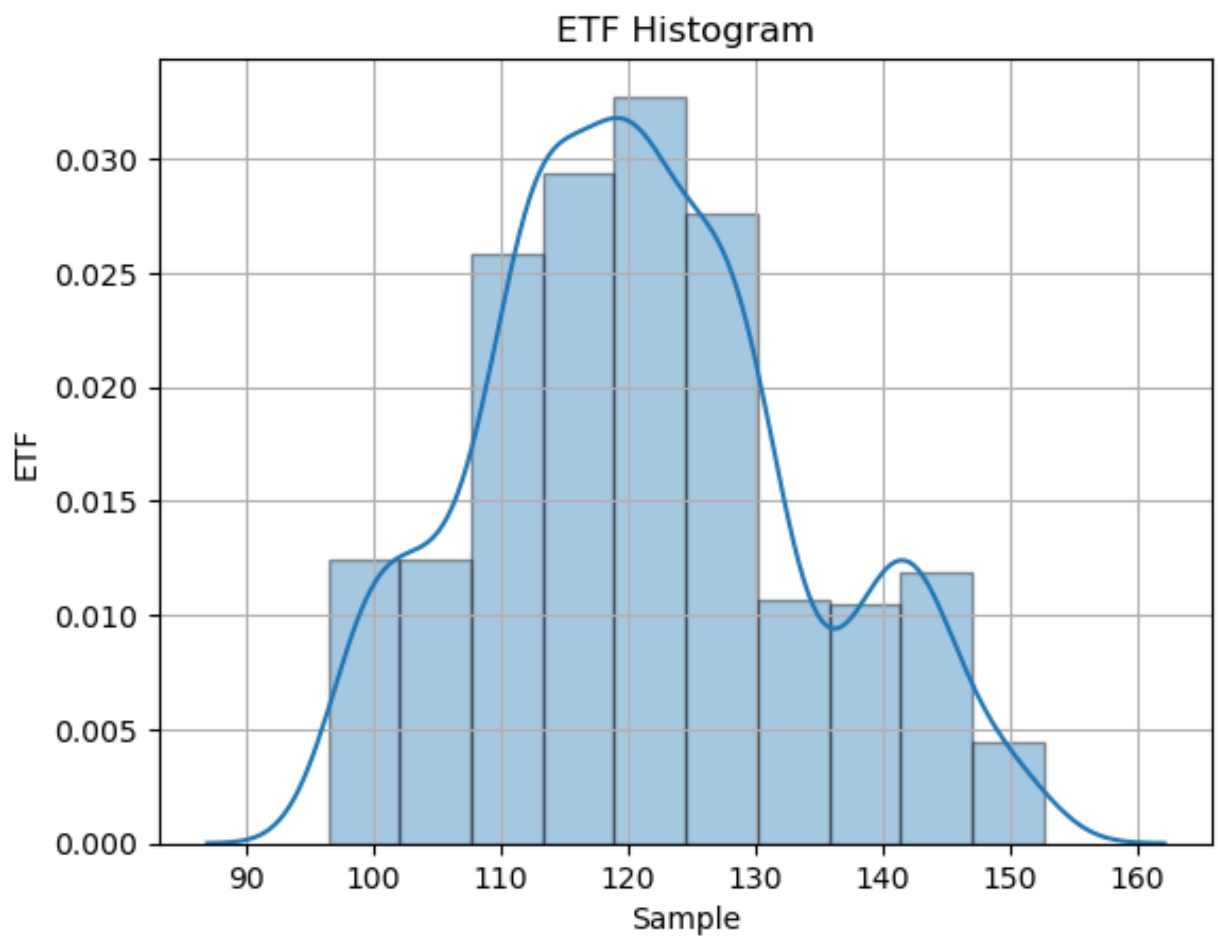




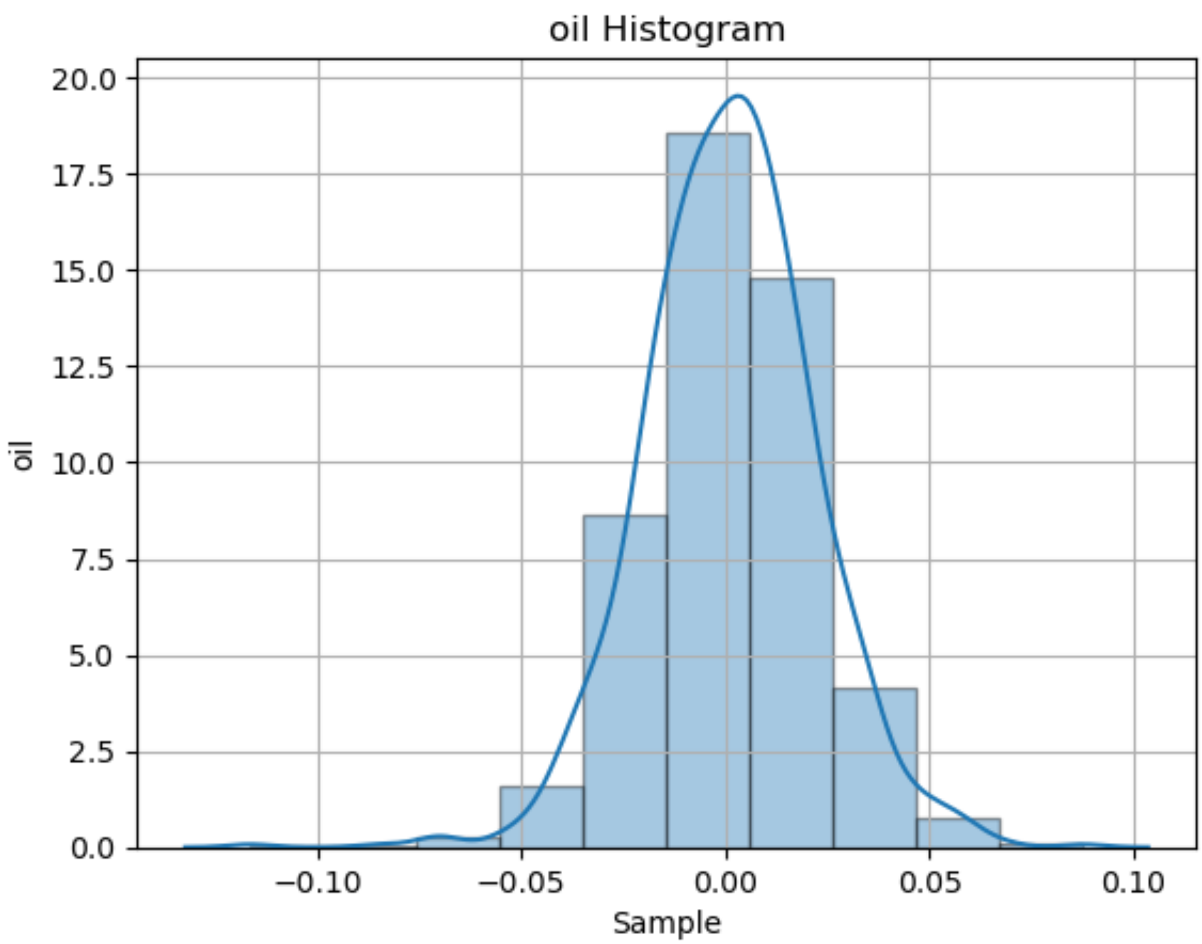
Part-2

In [166]:

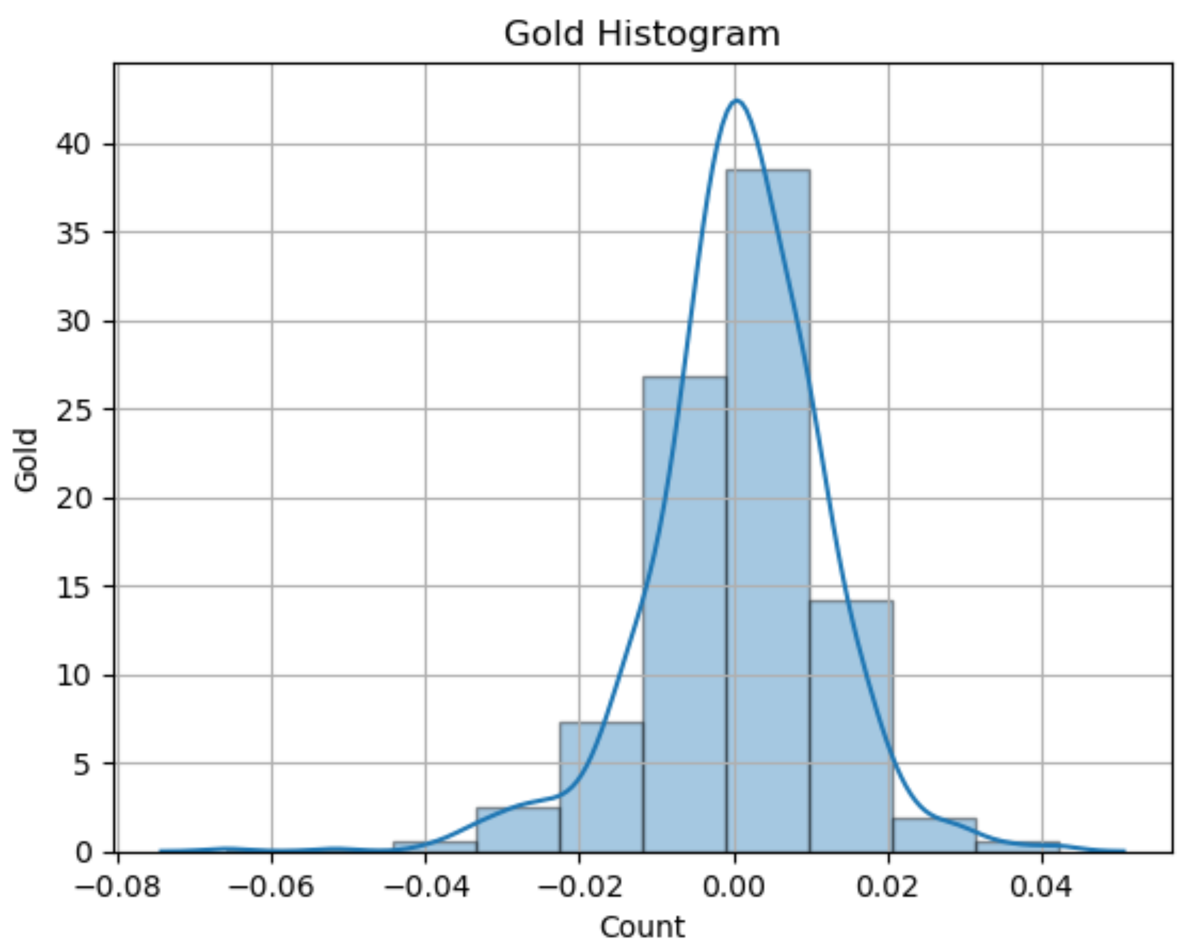
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a histogram using seaborn
sns.distplot(data['Close_ETF'], bins=10, hist_kws={'edgecolor': 'black

# Set labels and title
plt.xlabel('Sample')
plt.ylabel('ETF')
plt.title('ETF Histogram')

# Turn on grid
plt.grid(True)

# Uncomment the lines below to set specific y-axis and x-axis limits
# plt.ylim([0, 0.05])
# plt.xlim([0, 200])

# Display the plot
plt.show()
```



In [167]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a histogram using seaborn with specified edge color
sns.distplot(data['oil'], bins=10, hist_kws={'edgecolor': 'black'})

# Set labels and title
plt.xlabel('Sample')
plt.ylabel('oil')
plt.title('oil Histogram')

# Turn on grid
plt.grid(True)

# Display the plot
plt.show()
```



In [168]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a histogram using seaborn with specified edge color
sns.distplot(data['gold'], bins=10, hist_kws={'edgecolor': 'black'})

# Set labels and title
plt.xlabel('Count')
plt.ylabel('Gold')
plt.title('Gold Histogram')

# Turn on grid
plt.grid(True)

# Display the plot
plt.show()
```



Hypotheses:

ETF Null Hypothesis- H0- The distribution of the data is normal Alternate Hypothesis- H1- The distribution of the data is not normal

OIL Null Hypothesis- H0- The distribution of the data is normal Alternate Hypothesis- H1- The distribution of the data is not normal

Gold Null Hypothesis- H0- The distribution of the data is normal Alternate Hypothesis- H1- The distribution of the data is not normal

JPM Null Hypothesis- H0- The distribution of the data is normal Alternate Hypothesis- H1- The distribution of the data is not normal

Part-3

In [245]:
```
from scipy.stats import shapiro
from scipy.stats import anderson
from scipy.stats import kstest
```

In [246]:
```
from scipy.stats import shapiro

# Shapiro-Wilk test
stat, p = shapiro(data['Close_ETF'])
print('Statistics=%.3f, p=%.3f' % (stat, p))

# Interpretation
alpha = 0.05

if p > alpha:
    print('Fail to reject H0')
else:
    print('Reject H0')
```

```
Statistics=0.980, p=0.000
Reject H0
```

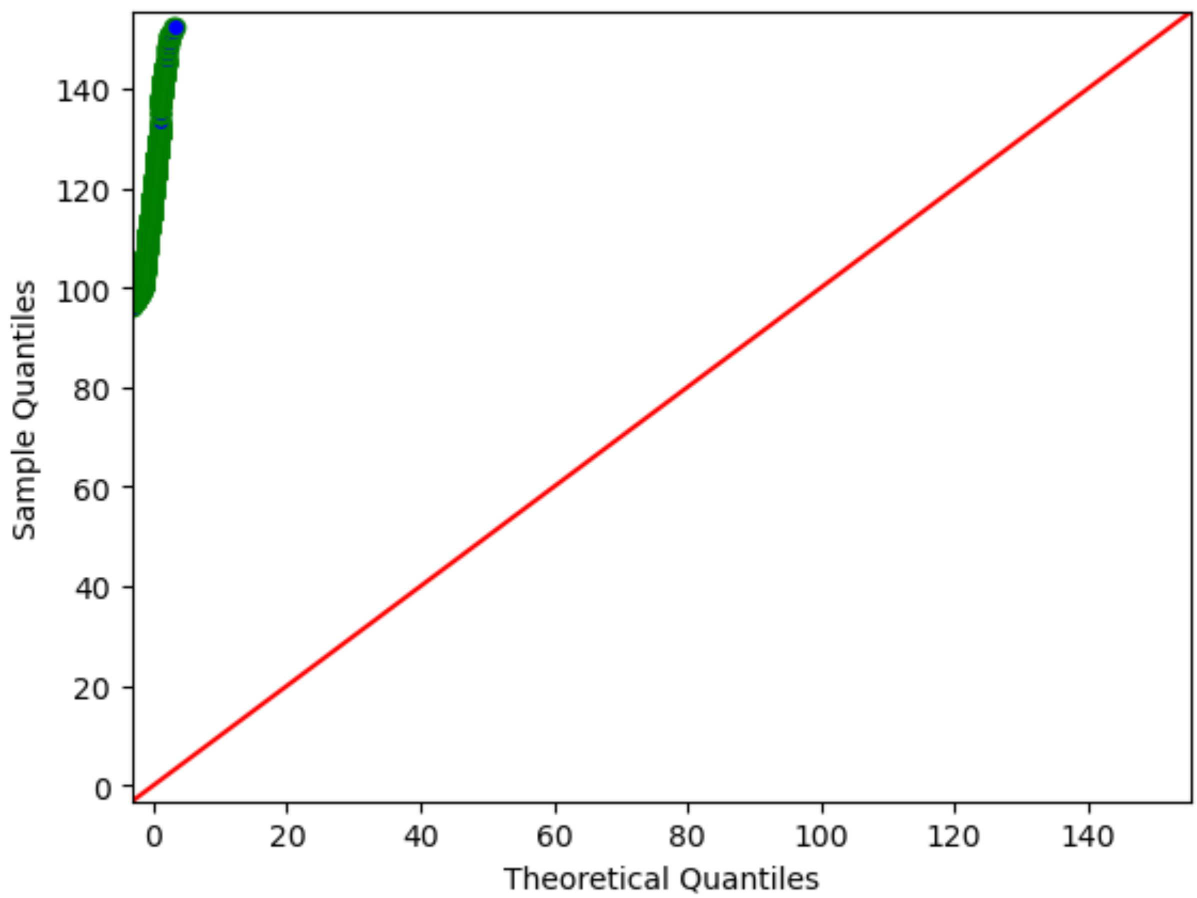
In [247]:
```
from scipy.stats import anderson

# Anderson-Darling test
result = anderson(data['Close_ETF'], dist='norm')
print('Statistic: %.3f' % result.statistic)

# Interpretation
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (s
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (s
```

```
Statistic: 4.693
15.000: 0.574, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.088, data does not look normal (reject H0)
```

In [248]:
```
from scipy.stats import kstest

# Smirnov-Kolmogorov test
result = kstest(data['Close_ETF'], 'norm')
print('Statistic: %.3f' % result.statistic)
print('p-value: %.3f' % result.pvalue)

# Interpretation
alpha = 0.05
if result.pvalue > alpha:
    print('Fail to reject H0')
else:
    print('Reject H0')
```

```
Statistic: 1.000
p-value: 0.000
Reject H0
```

In [249]:
```
import statsmodels.api as sm
import pylab as py

# QQ plot
sm.qqplot(data['Close_ETF'], line='45',markerfacecolor='blue',markerec
py.show()
```



Normality test for OIL

In [250]:
```
from scipy.stats import shapiro

# Shapiro-Wilk test
stat, p = shapiro(data['oil'])
oil_SWtest = 'Statistics=%.3f, p=%.3f' % (stat, p)
print(oil_SWtest)
```

```
Statistics=0.989, p=0.000
```

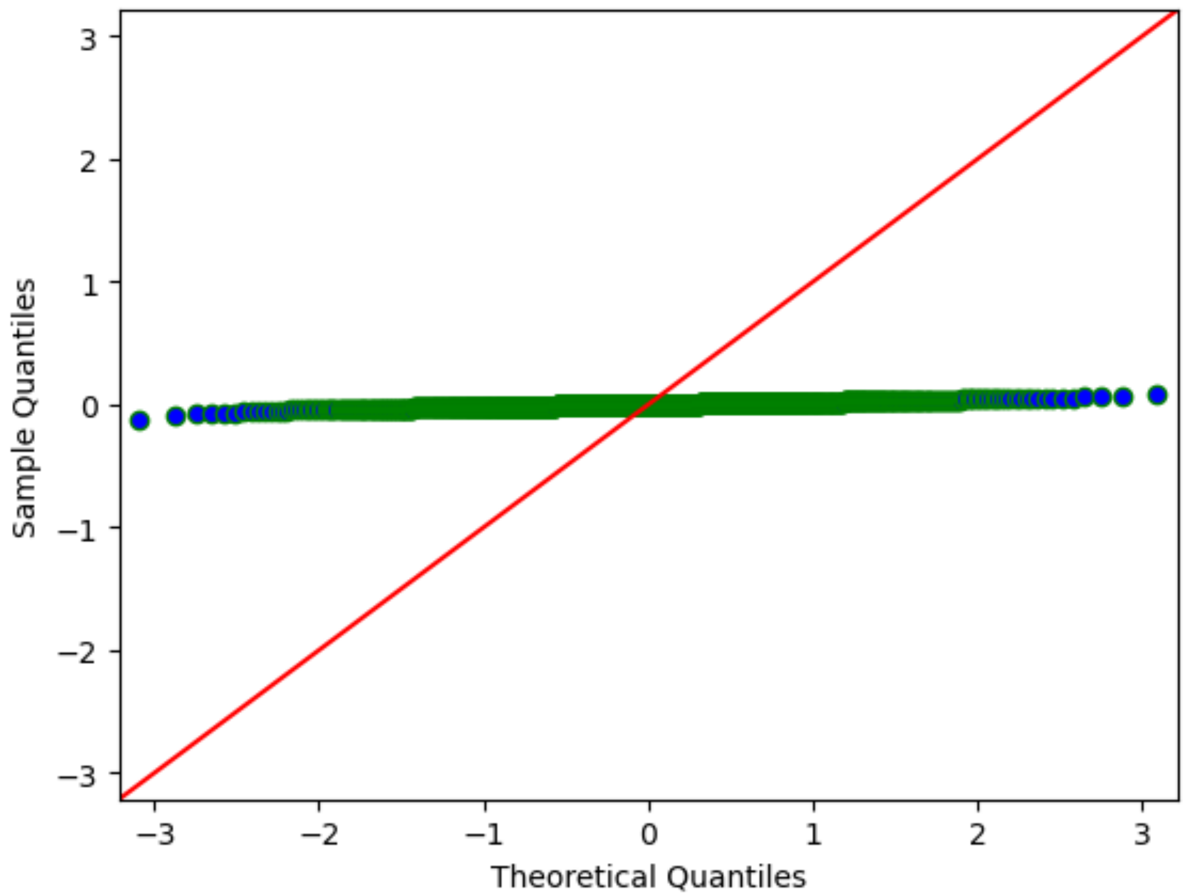
In [251]:
```
from scipy.stats import anderson

# Anderson-Darling test
result = anderson(data['oil'], dist='norm')
print('Statistic: %.3f' % result.statistic)

# Interpretation
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (s
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (s
```

```
Statistic: 1.143
15.000: 0.574, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.088, data does not look normal (reject H0)
```

In [252]:
```
#SmirnovKolmogorov test

kstest(data['oil'],'norm')
```

Out[252]:
```
KstestResult(statistic=0.4727185265212217, pvalue=1.2565304659417615e
-205, statistic_location=0.0609022556390977, statistic_sign=1)
```

In [253]:
```
import statsmodels.api as sm
import pylab as py

# QQ plot
sm.qqplot(data['oil'], line='45',markerfacecolor='blue',markeredgecolo
py.show()
```



Normality test for GOLD

In [254]:
```
from scipy.stats import shapiro

# Shapiro-Wilk test
stat, p = shapiro(data['gold'])
gold_SWtest = 'Statistics=%.3f, p=%.3f' % (stat, p)
print(gold_SWtest)
```

```
Statistics=0.969, p=0.000
```

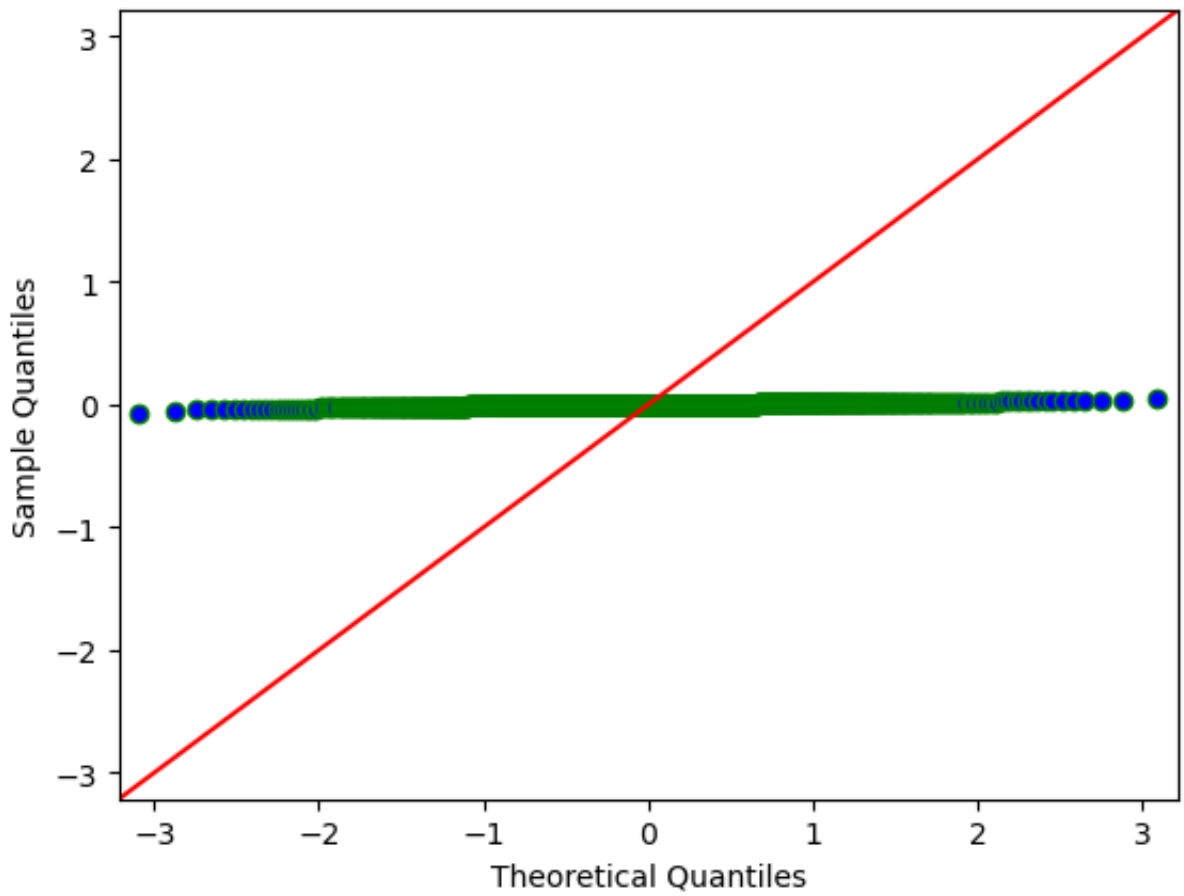
In [255]:
```
from scipy.stats import anderson

# Anderson-Darling test
result = anderson(data['gold'], dist='norm')
print('Statistic: %.3f' % result.statistic)

# Interpretation
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (s
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (s
```

```
Statistic: 6.377
15.000: 0.574, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.088, data does not look normal (reject H0)
```

In [256]:
```
from scipy.stats import kstest

# Smirnov-Kolmogorov test
result = kstest(data['gold'], 'norm')
print('Statistic: %.3f' % result.statistic)
print('p-value: %.3f' % result.pvalue)

# Interpretation
alpha = 0.05
if result.pvalue > alpha:
    print('Fail to reject H0')
else:
    print('Reject H0')
```

```
Statistic: 0.483
p-value: 0.000
Reject H0
```

In [257]:
```
import statsmodels.api as sm
import pylab as py

# QQ plot with custom colors
sm.qqplot(data['gold'], line='45', markerfacecolor='blue', markeredgec
py.show()
```



Normality test for JPM

In [258]:
```
from scipy.stats import shapiro

# Shapiro-Wilk test
stat, p = shapiro(data['JPM'])
JPM_SWtest = 'Statistics=%.3f, p=%.3f' % (stat, p)
print(JPM_SWtest)
```

```
Statistics=0.980, p=0.000
```

In [259]:
```
from scipy.stats import anderson

# Anderson-Darling test
result = anderson(data['JPM'], dist='norm')
print('Statistic: %.3f' % result.statistic)

# Interpretation
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (s
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (s
```

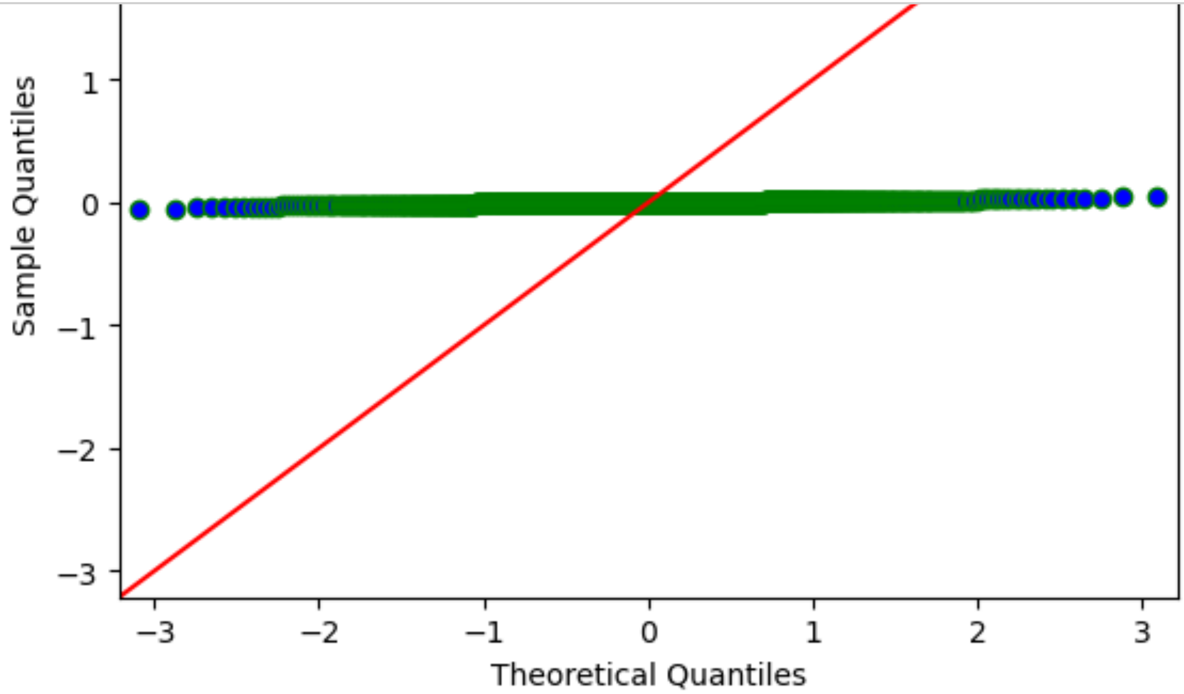
```
Statistic: 3.883
15.000: 0.574, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.088, data does not look normal (reject H0)
```

In [260]:
```
from scipy.stats import kstest

# Smirnov-Kolmogorov test
result = kstest(data['JPM'], 'norm')
print('Statistic: %.3f' % result.statistic)
print('p-value: %.3f' % result.pvalue)

# Interpretation
alpha = 0.05
if result.pvalue > alpha:
    print('Fail to reject H0')
else:
    print('Reject H0')
```

```
Statistic: 0.483
p-value: 0.000
Reject H0
```

In [261]:

```
import statsmodels.api as sm
import pylab as py

# QQ plot
sm.qqplot(data['JPM'], line='45',markerfacecolor='blue', markeredgecol
py.show()
```



Part-4

In [262]:

```
# Importing pandas library
import pandas as pd

# Creating or loading your DataFrame 'df'
df = pd.read_csv('Sample Course Project (Dataset).csv')

# Now you can proceed with the rest of your code
d1 = df[['Close_ETF']]

i2 = [seq for seq in range(0, 1000, 20)]
j2 = [seq1 for seq1 in range(20, 1020, 20)]
n2 = []

for k in range(0, 50):
    nn = d1[i2[k]:j2[k]]
    n2.append(nn)

print(n2)
```

```
[     Close_ETF
0    97.349998
1    97.750000
2    99.160004
3    99.650002
4    99.260002
5    98.250000
6    99.250000
7   100.300003
8   100.610001
9    99.559998
10  101.660004
11  101.660004
12  101.570000
13  100.019997
14   99.440002
15   98.419998
16   98.519997
17   97.529999
18   98.800003
19   97.660004,      Close_ETF
20   97.629997
21   98.529999
22   99.769997
23   98.739998
24  100.699997
25  101.150002
26  100.580002
27   99.300003
28  100.239998
29  100.730003
30  100.510002
31   99.919998
32   98.500000
33   99.510002
34   98.279999
35   99.169998
36   99.239998
37   98.489998
38  100.230003
39   99.860001,      Close_ETF
40   99.400002
41   99.160004
42   99.389999
43   98.510002
44   98.510002
45   96.419998
```

```
46     96.980003
47     98.000000
48     98.279999
49     98.650002
50     99.550003
51     99.040001
52     99.309998
53     99.620003
54    100.480003
55    100.860001
56    100.449997
57    100.769997
58     99.769997
59     99.930000,      Close_ETF
60    100.110001
61    100.139999
62    100.760002
63    101.440002
64    102.800003
65    103.360001
66    103.410004
67    102.830002
68    103.680000
69    103.000000
70    101.959999
71    102.260002
72    102.449997
73    102.089996
74    103.580002
75    103.379997
76    104.599998
77    103.669998
78    102.550003
79    102.940002,      Close_ETF
80    101.110001
81    100.279999
82     99.949997
83    100.930000
84     99.949997
85    102.080002
86    102.449997
87    103.389999
88    103.860001
89    104.260002
90    104.000000
91    104.279999
92    104.570000
93    104.900002
94    105.269997
95    104.989998
```

```
96  105.410004
97  104.260002
98  105.040001
99  104.860001,       Close_ETF
100  103.540001
101  103.349998
102  103.580002
103  103.629997
104  105.040001
105  105.180000
106  105.400002
107  105.300003
108  105.989998
109  105.760002
110  105.839996
111  106.400002
112  105.610001
113  105.180000
114  105.150002
115  106.330002
116  106.360001
117  105.459999
118  104.930000
119  103.839996,       Close_ETF
120  104.720001
121  103.779999
122  104.209999
123  105.589996
124  105.989998
125  106.370003
126  106.449997
127  107.599998
128  107.330002
129  107.160004
130  107.599998
131  106.849998
132  107.570000
133  106.739998
134  106.730003
135  107.930000
136  108.139999
137  107.599998
138  108.160004
139  108.500000,       Close_ETF
140  109.720001
141  108.900002
142  109.660004
143  109.730003
144  109.620003
145  109.699997
```

```
146  111.160004
147  111.180000
148  111.279999
149  111.230003
150  112.440002
151  112.550003
152  112.930000
153  113.379997
154  112.389999
155  113.220001
156  112.559998
157  113.500000
158  113.779999
159  114.230003,       Close_ETF
160  114.199997
161  115.099998
162  114.800003
163  114.430000
164  115.870003
165  114.680000
166  113.370003
167  113.480003
168  113.480003
169  113.970001
170  113.779999
171  112.849998
172  113.180000
173  114.449997
174  114.480003
175  114.849998
176  116.070000
177  115.650002
178  115.129997
179  116.169998,       Close_ETF
180  115.660004
181  115.230003
182  114.879997
183  114.589996
184  114.389999
185  114.870003
186  114.940002
187  115.019997
188  116.160004
189  115.480003
190  115.690002
191  115.989998
192  116.379997
193  114.959999
194  114.500000
195  112.580002
```

```
196  111.120003
197  112.580002
198  111.199997
199  111.790001,       Close_ETF
200  113.040001
201  113.070000
202  111.059998
203  109.650002
204  109.459999
205  109.550003
206  111.000000
207  111.029999
208  112.589996
209  112.970001
210  113.099998
211  113.779999
212  114.639999
213  115.269997
214  114.900002
215  114.629997
216  114.370003
217  114.820000
218  113.209999
219  113.389999,       Close_ETF
220  112.959999
221  113.830002
222  113.830002
223  111.919998
224  112.669998
225  114.250000
226  114.360001
227  114.199997
228  114.300003
229  112.820000
230  111.830002
231  110.959999
232  112.150002
233  112.059998
234  112.779999
235  111.809998
236  109.959999
237  108.830002
238  109.750000
239  110.449997,       Close_ETF
240  109.989998
241  110.040001
242  109.099998
243  109.650002
244  109.269997
245  109.620003
```

```
246  109.809998
247  110.269997
248  111.849998
249  112.239998
250  112.870003
251  112.860001
252  112.709999
253  113.129997
254  112.089996
255  112.980003
256  114.699997
257  114.860001
258  113.790001
259  114.349998,       Close_ETF
260  113.220001
261  114.019997
262  114.000000
263  113.830002
264  113.629997
265  113.199997
266  113.769997
267  114.750000
268  114.389999
269  113.839996
270  113.449997
271  113.919998
272  114.529999
273  112.940002
274  112.879997
275  111.889999
276  112.220001
277  111.440002
278  111.730003
279  111.779999,       Close_ETF
280  111.860001
281  111.519997
282  110.800003
283  110.709999
284  110.239998
285  111.639999
286  109.580002
287  109.879997
288  108.959999
289  108.750000
290  109.769997
291  110.099998
292  110.570000
293  110.839996
294  111.070000
295  110.209999
```

```
296  110.199997
297  108.400002
298  106.849998
299  107.000000,       Close_ETF
300  108.379997
301  108.160004
302  106.980003
303  107.190002
304  108.300003
305  108.910004
306  110.029999
307  109.709999
308  110.480003
309  110.199997
310  110.349998
311  111.099998
312  111.099998
313  111.449997
314  110.529999
315  111.110001
316  111.320000
317  112.580002
318  112.120003
319  112.860001,       Close_ETF
320  112.580002
321  112.480003
322  113.059998
323  113.430000
324  113.660004
325  112.800003
326  113.139999
327  113.150002
328  112.470001
329  112.959999
330  111.550003
331  110.949997
332  111.459999
333  110.750000
334  111.279999
335  111.839996
336  111.760002
337  113.650002
338  113.839996
339  113.900002,       Close_ETF
340  114.680000
341  113.449997
342  112.510002
343  112.970001
344  112.529999
345  111.540001
```

```
346  110.639999
347  111.260002
348  111.680000
349  110.739998
350  110.519997
351  111.239998
352  109.989998
353  109.860001
354  111.540001
355  112.879997
356  113.220001
357  113.199997
358  113.510002
359  113.550003,       Close_ETF
360  114.980003
361  116.550003
362  117.279999
363  117.110001
364  116.879997
365  116.970001
366  117.860001
367  118.790001
368  118.730003
369  117.879997
370  118.580002
371  118.739998
372  117.419998
373  117.980003
374  118.160004
375  118.440002
376  118.349998
377  117.809998
378  117.889999
379  119.230003,       Close_ETF
380  119.330002
381  119.250000
382  119.209999
383  118.099998
384  118.790001
385  119.209999
386  119.330002
387  120.370003
388  120.790001
389  120.879997
390  120.809998
391  119.440002
392  119.470001
393  120.389999
394  120.680000
395  120.769997
```

```
396  120.519997
397  121.180000
398  121.360001
399  121.129997,        Close_ETF
400  120.870003
401  120.300003
402  118.830002
403  118.010002
404  118.610001
405  118.440002
406  119.000000
407  118.180000
408  118.570000
409  117.620003
410  118.239998
411  119.470001
412  118.220001
413  117.500000
414  116.779999
415  116.550003
416  116.879997
417  117.230003
418  117.430000
419  117.430000,        Close_ETF
420  118.160004
421  118.910004
422  119.269997
423  118.959999
424  120.230003
425  120.070000
426  120.209999
427  119.309998
428  119.739998
429  120.769997
430  120.680000
431  121.129997
432  121.209999
433  120.230003
434  120.389999
435  118.599998
436  119.449997
437  120.239998
438  121.430000
439  120.629997,        Close_ETF
440  121.230003
441  121.169998
442  121.220001
443  122.730003
444  122.790001
445  122.330002
```

```
446  120.970001
447  121.239998
448  120.389999
449  121.139999
450  120.139999
451  119.120003
452  119.360001
453  118.540001
454  118.099998
455  116.900002
456  117.000000
457  117.139999
458  117.309998
459  116.529999,       Close_ETF
460  118.180000
461  117.959999
462  117.430000
463  117.629997
464  118.190002
465  118.599998
466  119.239998
467  118.000000
468  118.089996
469  118.699997
470  117.300003
471  115.769997
472  114.150002
473  114.500000
474  115.410004
475  113.800003
476  116.010002
477  115.570000
478  116.330002
479  115.199997,       Close_ETF
480  115.650002
481  114.199997
482  115.750000
483  116.400002
484  116.599998
485  117.500000
486  117.459999
487  117.089996
488  117.820000
489  116.599998
490  117.239998
491  115.949997
492  115.720001
493  116.800003
494  117.580002
495  118.790001
```

```
496  119.290001
497  119.120003
498  119.779999
499  119.500000,      Close_ETF
500  119.410004
501  120.050003
502  120.250000
503  119.480003
504  120.500000
505  120.760002
506  120.150002
507  120.040001
508  120.129997
509  119.910004
510  120.480003
511  120.199997
512  120.580002
513  120.860001
514  121.089996
515  121.400002
516  121.360001
517  121.400002
518  121.470001
519  121.570000,      Close_ETF
520  119.860001
521  118.980003
522  119.150002
523  120.150002
524  119.830002
525  119.180000
526  119.529999
527  120.489998
528  119.480003
529  119.949997
530  121.320000
531  121.940002
532  122.260002
533  122.430000
534  122.910004
535  122.839996
536  122.349998
537  123.019997
538  123.440002
539  122.720001,      Close_ETF
540  123.540001
541  123.190002
542  123.339996
543  123.790001
544  124.570000
545  123.739998
```

```
546  123.650002
547  124.389999
548  124.720001
549  123.720001
550  122.879997
551  122.650002
552  123.389999
553  123.330002
554  123.820000
555  123.059998
556  123.820000
557  122.209999
558  122.199997
559  122.190002,       Close_ETF
560  122.470001
561  122.470001
562  122.239998
563  121.150002
564  121.589996
565  120.760002
566  121.690002
567  121.050003
568  122.580002
569  122.489998
570  122.269997
571  123.699997
572  123.910004
573  123.500000
574  124.599998
575  124.349998
576  123.660004
577  123.209999
578  123.150002
579  123.500000,       Close_ETF
580  123.089996
581  122.050003
582  120.910004
583  121.339996
584  121.440002
585  121.580002
586  121.550003
587  121.669998
588  122.660004
589  123.040001
590  122.599998
591  121.220001
592  119.629997
593  119.199997
594  119.610001
595  118.599998
```

```
596  118.430000
597  117.500000
598  117.430000
599  118.669998,        Close_ETF
600  119.110001
601  117.820000
602  119.779999
603  117.669998
604  118.129997
605  119.959999
606  119.720001
607  119.370003
608  118.099998
609  119.800003
610  120.129997
611  120.489998
612  121.750000
613  122.269997
614  122.110001
615  122.230003
616  122.230003
617  122.389999
618  123.339996
619  123.760002,        Close_ETF
620  123.690002
621  123.239998
622  123.489998
623  124.639999
624  125.129997
625  125.760002
626  126.300003
627  127.029999
628  127.129997
629  126.230003
630  126.089996
631  125.410004
632  126.690002
633  126.849998
634  126.580002
635  126.820000
636  126.080002
637  126.000000
638  126.330002
639  126.449997,        Close_ETF
640  127.309998
641  127.809998
642  127.440002
643  126.360001
644  125.709999
645  125.830002
```

```
646  126.029999
647  126.690002
648  126.760002
649  125.470001
650  125.750000
651  125.190002
652  124.510002
653  126.699997
654  127.300003
655  127.379997
656  128.440002
657  128.770004
658  128.899994
659  129.309998,        Close_ETF
660  128.800003
661  128.679993
662  128.330002
663  127.820000
664  128.309998
665  125.970001
666  126.419998
667  126.550003
668  126.660004
669  127.360001
670  128.539993
671  128.440002
672  127.500000
673  128.389999
674  126.900002
675  126.269997
676  126.599998
677  125.480003
678  126.620003
679  126.410004,        Close_ETF
680  126.639999
681  126.410004
682  127.750000
683  128.199997
684  129.160004
685  128.809998
686  128.490005
687  129.270004
688  129.080002
689  129.410004
690  129.460007
691  128.229996
692  129.369995
693  129.360001
694  128.759995
695  128.169998
```

```
696  127.970001
697  128.240005
698  127.379997
699  128.589996,       Close_ETF
700  128.830002
701  130.179993
702  130.759995
703  131.029999
704  130.619995
705  130.410004
706  129.589996
707  130.380005
708  130.110001
709  130.210007
710  130.020004
711  129.220001
712  130.029999
713  129.800003
714  129.830002
715  129.729996
716  130.559998
717  131.009995
718  130.869995
719  129.539993,       Close_ETF
720  129.740005
721  128.639999
722  128.880005
723  128.710007
724  128.660004
725  130.699997
726  130.949997
727  131.130005
728  131.149994
729  130.910004
730  130.369995
731  130.399994
732  131.029999
733  131.470001
734  130.399994
735  131.380005
736  130.889999
737  131.360001
738  132.520004
739  132.360001,       Close_ETF
740  132.619995
741  132.550003
742  130.949997
743  129.240005
744  129.500000
745  129.309998
```

```
746  126.849998
747  126.209999
748  127.099998
749  126.129997
750  125.169998
751  126.169998
752  127.730003
753  128.380005
754  126.099998
755  127.510002
756  128.729996
757  129.000000
758  127.120003
759  126.809998,         Close_ETF
760  125.860001
761  125.750000
762  125.349998
763  123.989998
764  122.550003
765  123.500000
766  126.120003
767  124.650002
768  123.669998
769  124.089996
770  125.010002
771  124.459999
772  124.440002
773  124.989998
774  123.910004
775  124.750000
776  127.269997
777  127.279999
778  127.800003
779  127.070000,         Close_ETF
780  127.440002
781  126.610001
782  126.849998
783  127.410004
784  126.050003
785  124.000000
786  123.519997
787  123.339996
788  123.970001
789  125.690002
790  124.830002
791  123.949997
792  126.209999
793  126.660004
794  126.830002
795  126.709999
```

```
796  127.980003
797  127.849998
798  127.220001
799  128.080002,        Close_ETF
800  128.419998
801  128.199997
802  127.900002
803  127.410004
804  126.980003
805  127.370003
806  127.010002
807  127.110001
808  128.630005
809  129.699997
810  130.029999
811  130.690002
812  130.130005
813  130.119995
814  129.759995
815  129.649994
816  129.809998
817  130.429993
818  130.580002
819  130.660004,        Close_ETF
820  130.639999
821  131.419998
822  131.669998
823  130.509995
824  129.910004
825  130.279999
826  130.410004
827  131.690002
828  132.220001
829  132.229996
830  131.960007
831  132.139999
832  131.809998
833  132.509995
834  131.869995
835  131.470001
836  132.479996
837  133.580002
838  133.740005
839  133.690002,        Close_ETF
840  133.580002
841  133.080002
842  133.360001
843  134.919998
844  135.179993
845  135.009995
```

```
846  135.089996
847  135.270004
848  135.110001
849  136.279999
850  136.630005
851  136.839996
852  136.410004
853  136.589996
854  136.809998
855  136.839996
856  137.470001
857  137.880005
858  138.350006
859  138.779999,       Close_ETF
860  137.910004
861  137.809998
862  137.789993
863  136.860001
864  136.779999
865  136.539993
866  138.080002
867  138.610001
868  138.910004
869  138.179993
870  138.240005
871  138.580002
872  139.619995
873  140.020004
874  140.380005
875  140.419998
876  140.500000
877  140.639999
878  140.919998
879  140.350006,       Close_ETF
880  138.419998
881  139.020004
882  140.470001
883  140.529999
884  140.220001
885  141.289993
886  141.899994
887  141.779999
888  141.160004
889  141.419998
890  141.830002
891  141.720001
892  141.869995
893  143.119995
894  142.339996
895  141.949997
```

```
896  142.220001
897  142.139999
898  141.619995
899  140.750000,        Close_ETF
900  141.580002
901  142.509995
902  142.210007
903  141.619995
904  141.369995
905  141.669998
906  140.539993
907  141.190002
908  141.070007
909  141.539993
910  142.160004
911  143.240005
912  142.960007
913  143.020004
914  142.539993
915  142.820007
916  142.380005
917  142.800003
918  143.949997
919  142.259995,        Close_ETF
920  142.130005
921  142.050003
922  142.789993
923  143.750000
924  144.610001
925  144.809998
926  144.850006
927  144.889999
928  145.210007
929  145.020004
930  143.940002
931  143.449997
932  144.660004
933  145.610001
934  145.800003
935  145.729996
936  146.039993
937  145.979996
938  145.869995
939  145.300003,        Close_ETF
940  145.169998
941  139.500000
942  140.929993
943  140.509995
944  138.669998
945  137.350006
```

```
946  139.699997
947  139.559998
948  140.740005
949  140.779999
950  140.990005
951  138.250000
952  139.279999
953  139.470001
954  138.529999
955  140.199997
956  140.970001
957  143.289993
958  143.179993
959  143.389999,        Close_ETF
960  143.199997
961  142.860001
962  141.820007
963  141.970001
964  142.000000
965  142.160004
966  143.690002
967  143.850006
968  144.240005
969  144.440002
970  144.610001
971  144.020004
972  144.660004
973  145.320007
974  146.699997
975  147.089996
976  147.270004
977  147.229996
978  148.619995
979  148.059998,        Close_ETF
980  148.119995
981  149.479996
982  149.649994
983  149.529999
984  148.289993
985  148.669998
986  149.539993
987  150.350006
988  150.919998
989  150.949997
990  150.750000
991  151.160004
992  149.580002
993  150.860001
994  150.529999
995  150.570007
```

```
996  151.600006
997  151.300003
998  152.619995
999  152.539993]
```

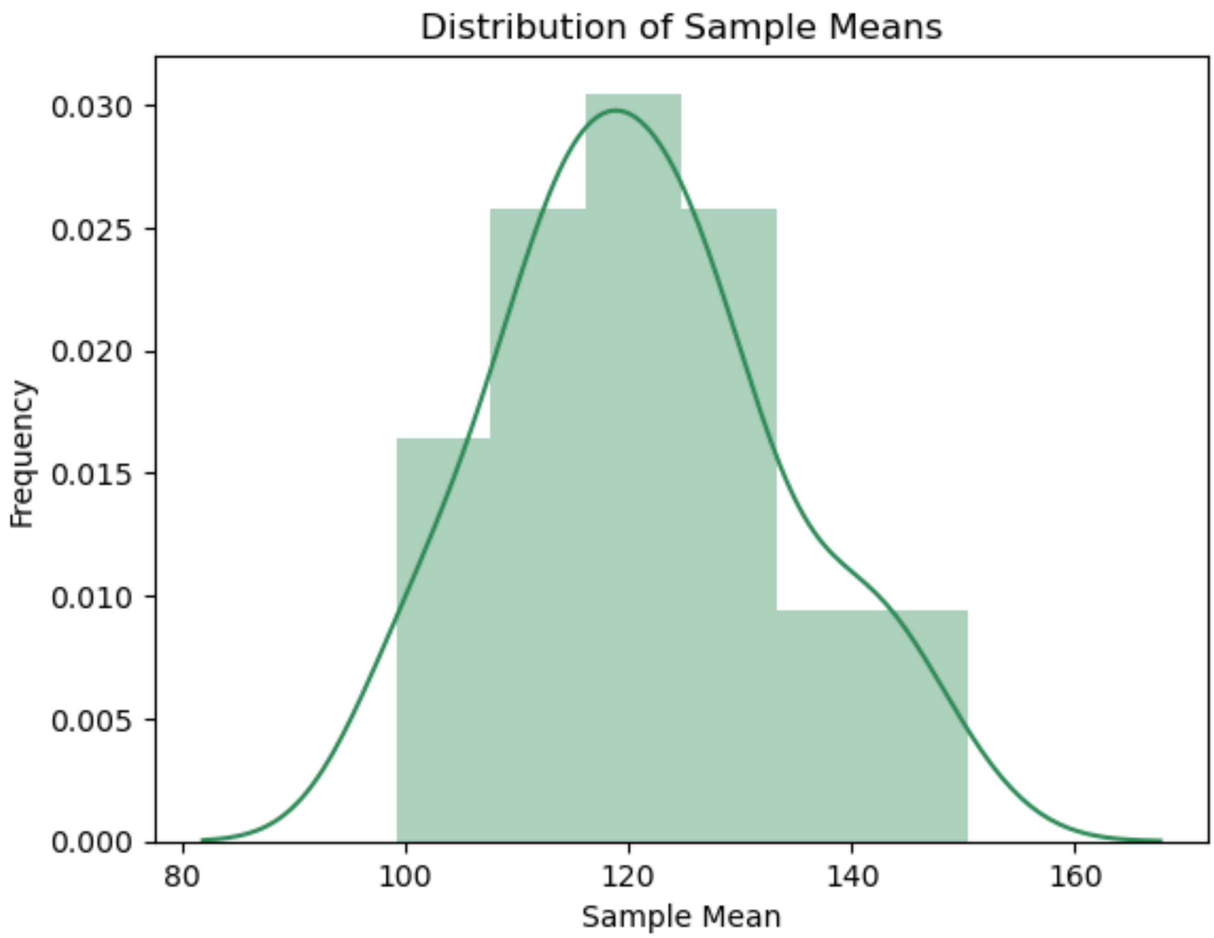
In [74]:
```
import numpy as np

mean = []
for i in range(len(n2)):
    sample_mean = np.mean(n2[i])
    mean.append(sample_mean)

    print("####################")
    print("Sample:", i+1)
    print("Close_ETF Mean:", sample_mean)
    print("Mean Value Data Type:", type(sample_mean))
```

```
Mean Value Data Type: <class 'numpy.float64'>
####################
Sample: 46
Close_ETF Mean: 142.17150034999997
Mean Value Data Type: <class 'numpy.float64'>
####################
Sample: 47
Close_ETF Mean: 144.6245003
Mean Value Data Type: <class 'numpy.float64'>
####################
Sample: 48
Close_ETF Mean: 140.5229988
Mean Value Data Type: <class 'numpy.float64'>
####################
Sample: 49
Close_ETF Mean: 144.69050135
Mean Value Data Type: <class 'numpy.float64'>
####################
Sample: 50
Close_ETF Mean: 150.35049895
```

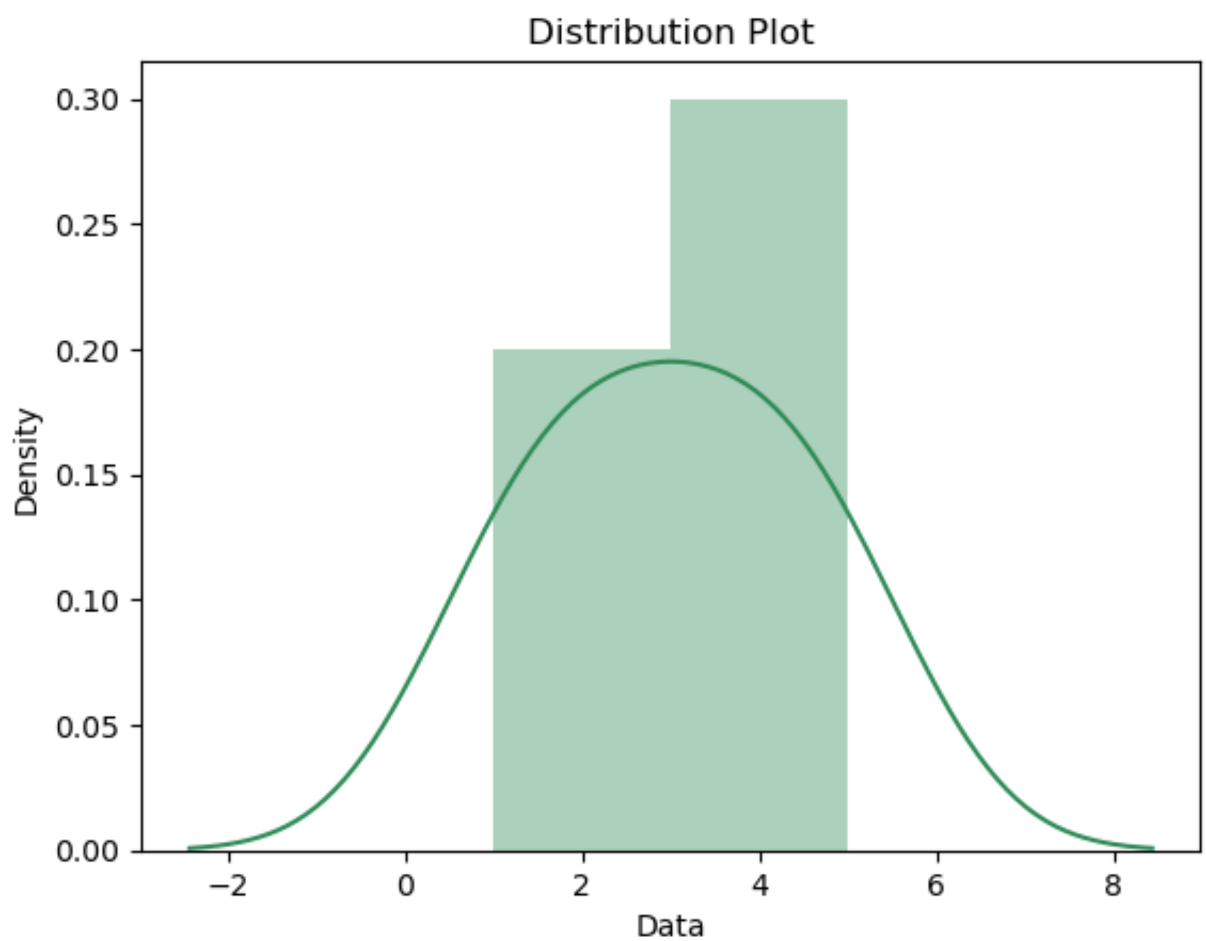
In [75]:
```
import numpy as np

mean = []
for i in range(len(n2)):
    sample_mean = np.mean(n2[i])
    mean.append(sample_mean)

    print("####################")
    print("Standard deviation for sample number:", i+1)
    print("Close_ETF Mean:", sample_mean)
    print("Standard deviation Value Data Type:", type(sample_mean))
```

```
Standard deviation Value Data Type: <class 'numpy.float64'>
#####################
Standard deviation for sample number: 46
Close_ETF Mean: 142.17150034999997
Standard deviation Value Data Type: <class 'numpy.float64'>
#####################
Standard deviation for sample number: 47
Close_ETF Mean: 144.6245003
Standard deviation Value Data Type: <class 'numpy.float64'>
#####################
Standard deviation for sample number: 48
Close_ETF Mean: 140.5229988
Standard deviation Value Data Type: <class 'numpy.float64'>
#####################
Standard deviation for sample number: 49
Close_ETF Mean: 144.69050135
Standard deviation Value Data Type: <class 'numpy.float64'>
#####################
Standard deviation for sample number: 50
Close_ETF Mean: 150.35049895
```

In [230]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a histogram of the sample means with a specified color
sns.distplot(mean, color='seagreen')

# Set labels and title
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.title('Distribution of Sample Means')

# Display the plot
plt.show()
```



In [231]:
```
# Assuming 'stand' is a variable containing your data
# Define or load your data into the 'stand' variable here
# For example:
stand = [1, 2, 3, 4, 5]

# After defining 'stand', you can proceed with the rest of your code
import seaborn as sns
import matplotlib.pyplot as plt

# Plot a histogram of the 'stand' data with a specified color
sns.distplot(stand, color='seagreen')

# Set labels and title
plt.xlabel('Data')
plt.ylabel('Density')
plt.title('Distribution Plot')

# Display the plot
plt.show()
```



part-5: (1)

In [238]:
```
import random

# Creating a population, replace with your own
population = data['Close_ETF'].tolist()

sample_size = 10
value = 100

for x in range(sample_size):
    # Creating a random sample of the population with size 100
    sample = random.sample(population, value)
```

In [239]:
```
arr1 = np.array(sample)
arr1
```

Out[239]:
```
array([128.800003, 148.119995, 113.43    , 122.550003, 123.519997,
       109.660004, 114.230003, 129.5     , 119.25    , 147.270004,
       120.489998, 118.540001, 123.190002, 109.75    , 136.839996,
       119.5     , 128.440002, 115.410004, 126.410004, 119.830002,
       111.830002,  99.650002, 143.449997, 124.440002, 141.820007,
       149.580002, 119.480003, 122.190002, 128.770004, 121.18    ,
       142.539993, 101.959999, 119.480003, 143.179993, 120.480003,
       118.      , 100.139999,  99.300003, 124.639999, 136.779999,
       121.239998, 120.150002, 143.940002, 108.139999, 142.960007,
       120.370003, 128.440002, 112.589996, 109.989998, 126.      ,
       113.900002, 147.089996, 117.5     , 143.119995, 115.129997,
       116.970001, 102.550003, 109.580002, 113.830002, 117.43    ,
       111.860001, 150.919998, 107.190002, 131.809998, 114.589996,
       120.199997, 111.919998, 115.769997, 115.480003, 110.839996,
       110.239998, 131.380005, 108.75    , 119.949997, 109.650002,
       127.440002, 146.039993, 108.959999, 117.089996, 112.059998,
       123.970001,  99.550003,  99.860001, 124.650002, 140.639999,
       126.169998, 118.790001, 118.349998, 144.809998, 107.160004,
       119.610001, 100.239998, 136.860001, 141.720001, 128.589996,
       130.210007, 121.209999, 111.540001, 111.779999, 113.839996])
```

In [240]:
```
import numpy as np
import scipy.stats

def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m-h, m+h
```

Part-5: (2)

95% confidence interval for one of the10 simple random samples

In [241]:
```
mean_confidence_interval(sample, confidence=0.95)
```

Out[241]:
```
(121.76230015000002, 119.16491746308576, 124.35968283691427)
```

In [242]:
```
import random

sample_size = 50
value = 20
sample_list = []

for x in range(sample_size):
    # Creating a random sample of the population with size 20:
    sample2 = random.sample(population, value)
    sample_list.append(sample)
```

In [243]:
```
sample2
```

Out[243]:
```
[117.580002,
 117.089996,
 112.059998,
 128.440002,
 120.629997,
 124.830002,
 128.440002,
 127.849998,
 144.889999,
 137.470001,
 123.5,
 117.309998,
 115.870003,
 120.910004,
 147.270004,
 97.75,
 140.529999,
 127.5,
 140.380005,
 110.739998]
```

95% confidence interval for one of the 50 simple random samples

In [244]:
```
mean_confidence_interval(sample2, confidence=0.95)
```

Out[244]:
```
(125.0520004, 119.19630297969665, 130.90769782030335)
```

```
Part-6
(T-test): (1) & (2)
```

In [99]:
```
import scipy.stats as stats

# Assuming 'sample' is your sample data
t_statistic, p_value = stats.ttest_ind(data['Close_ETF'], sample)

print("t-statistic:", t_statistic)
print("p-value:", p_value)
```

```
t-statistic: 0.2923053565320816
p-value: 0.7701127251079036
```

In [105]:
```
import scipy.stats as stats

# Assuming 'sample2' is your second sample data
t_statistic, p_value = stats.ttest_ind(data['Close_ETF'], sample2)

print("t-statistic:", t_statistic)
print("p-value:", p_value)
```

```
t-statistic: -0.5755347479614691
p-value: 0.5650568897123266
```

Part-6 (T-test): (3) & (4)

In [106]:
```
import scipy.stats as stats

# Assuming 'sample' and 'sample2' are your sample data
f_statistic, p_value = stats.f_oneway(sample, sample2)

print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

```
F-statistic: 0.6180564126862451
p-value: 0.43664565909324193
```

In [107]:
```
import scipy.stats as stats

# Assuming 'sample' is your sample data
f_statistic, p_value = stats.f_oneway(sample, data['Close_ETF'])

print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

```
F-statistic: 0.08544242145735152
p-value: 0.7701127251078093
```

In [108]:
```
import scipy.stats as stats

# Assuming 'sample2' is your sample data
f_statistic, p_value = stats.f_oneway(sample2, data['Close_ETF'])

print("F-statistic:", f_statistic)
print("p-value:", p_value)
```

```
F-statistic: 0.3312402461110693
p-value: 0.5650568897123198
```

part-8

In [109]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score

data=data.iloc[:999]
```

In [110]:
```
import numpy as np

# Assuming 'data' is your DataFrame containing 'gold' and 'Close_ETF'

# Extracting 'gold' data and reshaping it
X = np.array(data['gold']).reshape(-1, 1)

# Extracting 'Close_ETF' data and reshaping it
y = np.array(data['Close_ETF']).reshape(-1, 1)
```

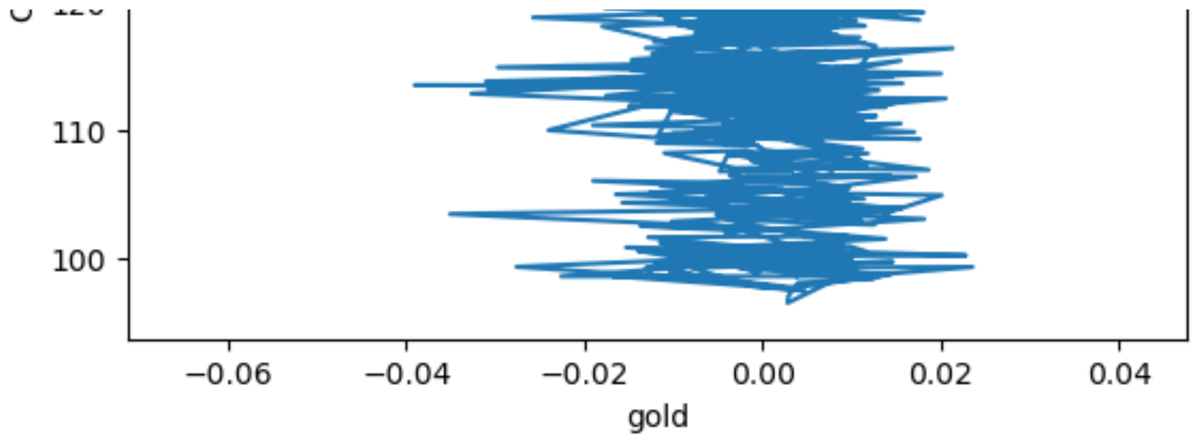
In [111]:
```
from sklearn.model_selection import train_test_split

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

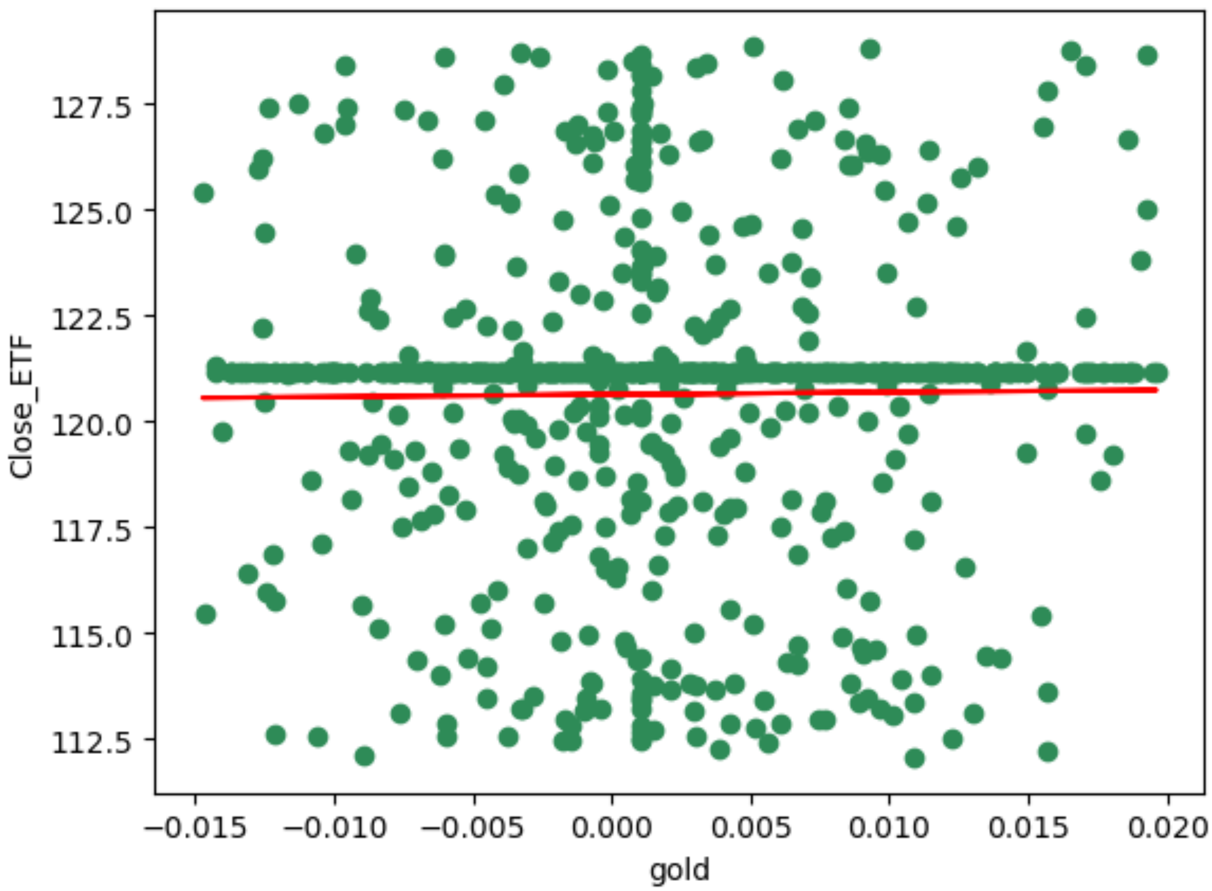
In [112]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

# Creating a Linear Regression model
model = LinearRegression()

# Performing cross-validation with 5 folds
scores = cross_val_score(model, X_train, y_train, cv=5)
print(scores)
```

```
[ 0.00141353 -0.00131514 -0.00281045 -0.0052275  -0.00902953]
```

In [113]:
```
# Fitting the model to the training data
model = model.fit(X_train, y_train)

# Making predictions on the test data
y_pred = model.predict(X_test)
```

In [267]:
```
import matplotlib.pyplot as plt

# Plotting gold vs. Close_ETF
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(data['gold'], data['Close_ETF'])
ax.set_xlabel('gold')
ax.set_ylabel('Close_ETF')
plt.show()

# Plotting the training data and the regression line
plt.scatter(X_train, y_train, color='seagreen')
plt.plot(X_train, model.predict(X_train), color='red')
plt.xlabel('gold')
plt.ylabel('Close_ETF')
plt.show()
```


110
100
−0.06
−0.04
−0.02
0.00
0.02
0.04
gold


127.5
125.0
122.5
120.0
117.5
115.0
112.5
Close_ETF
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020
gold

In [116]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Calculating Mean Squared Error
MSE = mean_squared_error(y_test, y_pred)

# Calculating R2 Score
r2 = r2_score(y_test, y_pred)

# Printing the results
print('Mean squared error: ', MSE)
print('R2 Score: ', r2)
```

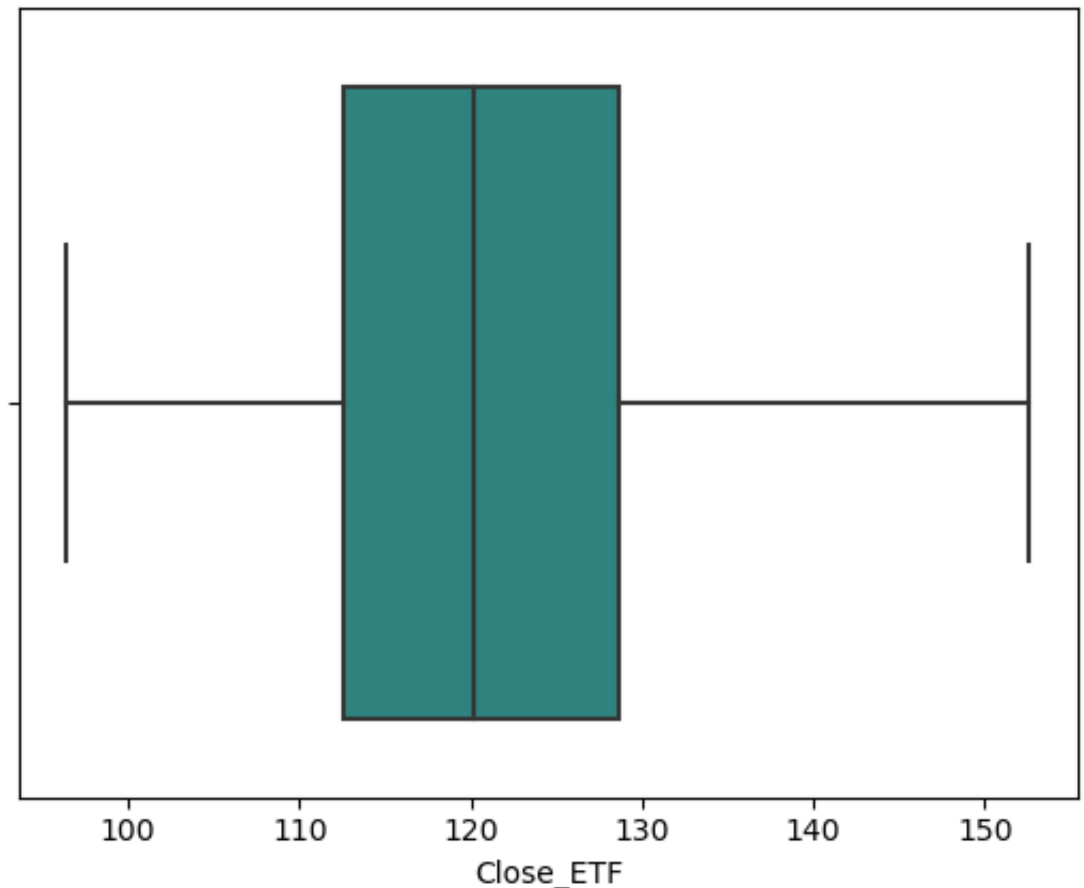
```
Mean squared error:  145.5942952742069
R2 Score:  -0.007500697440531834
```

Type Markdown and LaTeX: $\alpha 2$

Removing outliers from gold and ETF

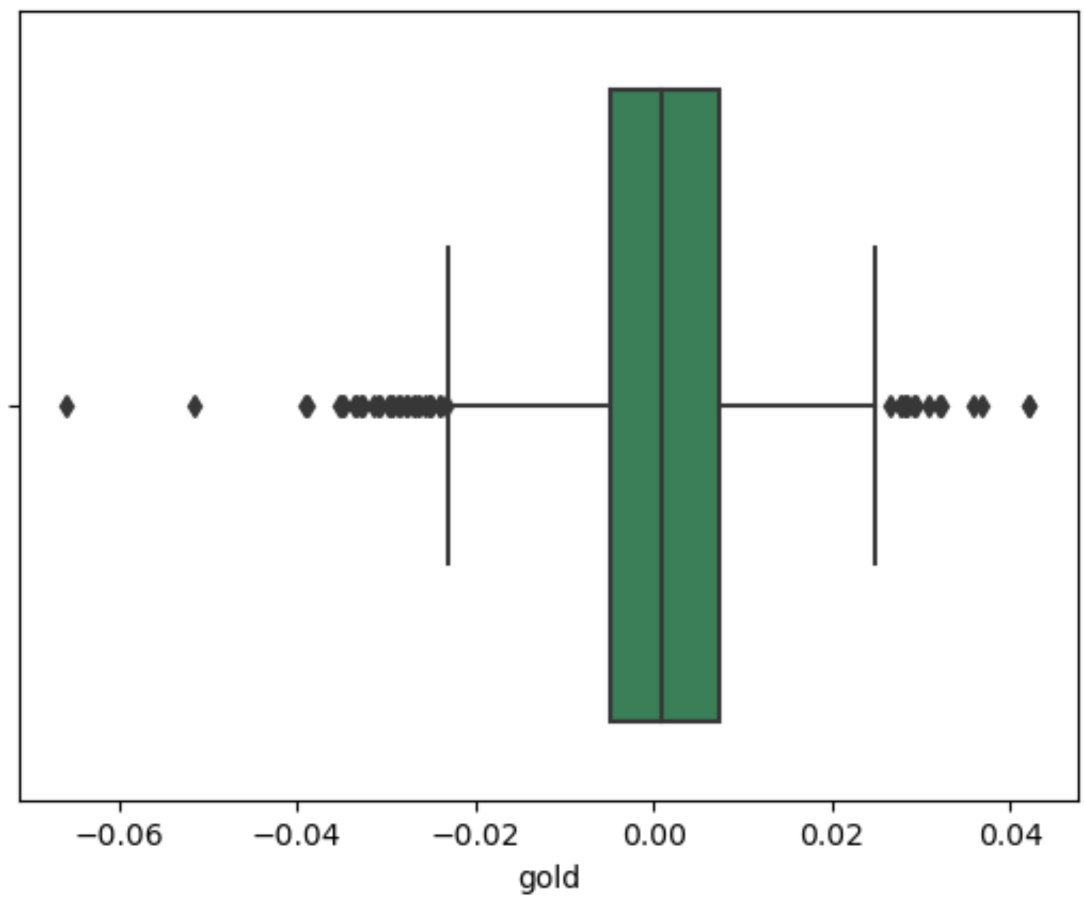
In [135]:
```
import seaborn as sns

# Creating a boxplot of 'Close_ETF' with custom colors
sns.boxplot(x=data['Close_ETF'], palette='viridis')  # Example using a
```

Out[135]: <Axes: xlabel='Close_ETF'>



In [270]:
```
import seaborn as sns

# Creating a boxplot of 'gold' with a specified color
sns.boxplot(x=data['gold'], color='seagreen')
```

Out[270]: <Axes: xlabel='gold'>

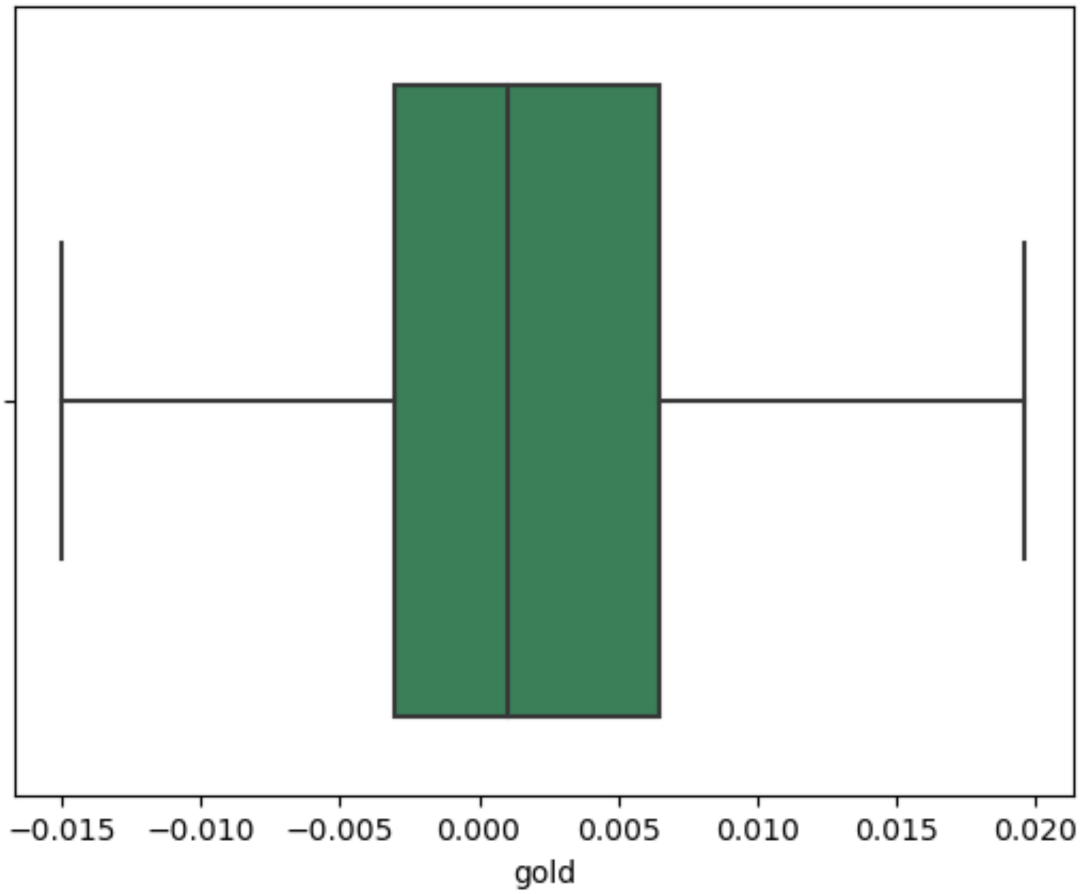


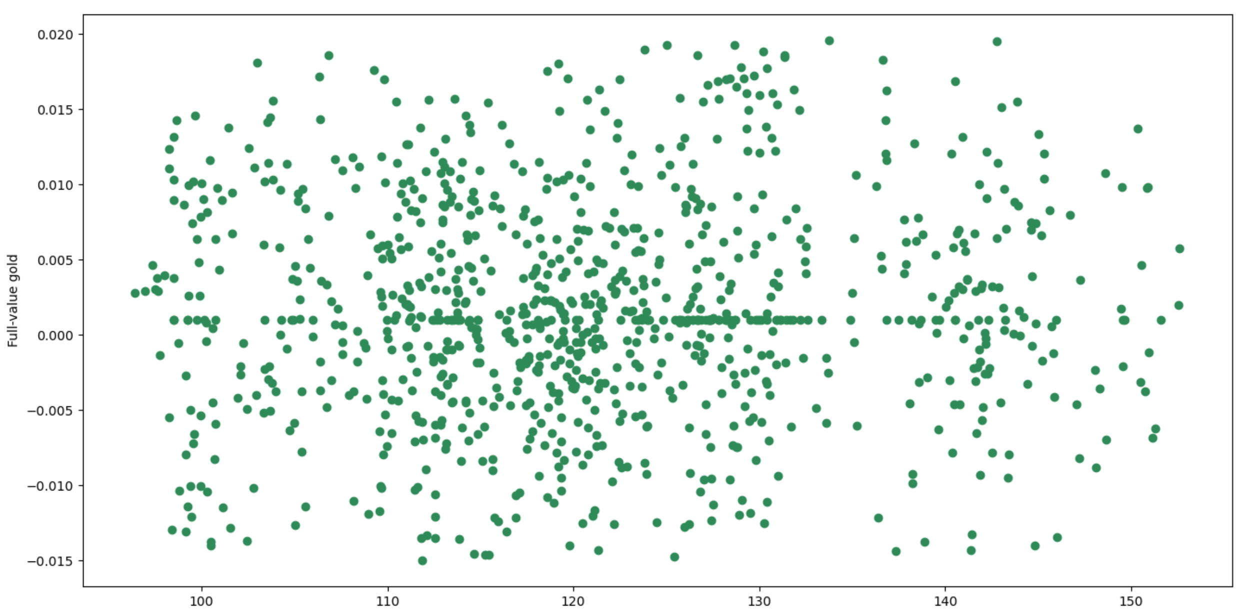
In [271]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

# Scatter plot of 'Close_ETF' vs 'gold' with a specified color
fig, ax = plt.subplots(figsize=(16, 8))
ax.scatter(data['Close_ETF'], data['gold'], color='purple')
ax.set_xlabel('Close_ETF')
ax.set_ylabel('Full-value gold')
plt.show()

# Calculating z-scores for 'gold'
z = np.abs(stats.zscore(data['gold']))

# Printing all the z-scores
for score in z:
    print(score)
```

```
0.04123954903300669
0.08107536356203009
0.4398249305061804
0.35771104423547123
0.07848272314511096
0.5305984441956728
0.5744237281117054
1.0054730420168219
0.8175806239620799
0.8282835899795846
0.22763764989564855
1.1497009285615836
1.326032055651459
0.34444920417612546
0.3654664981733328
0.7091450209089452
1.2631688881694803
0.3187878080901175
0.35948880081125867
0.837684393762613
```

In [272]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Modify 'gold' values based on conditions
data['gold'] = np.where(data['gold'] > 0.02, 0.001030, data['gold'])
data['gold'] = np.where(data['gold'] < -0.015, 0.001030, data['gold'])

# Create a boxplot of 'gold' with a specified color
sns.boxplot(x=data['gold'], color='seagreen')
```

```
# Scatter plot of 'Close_ETF' vs 'gold' with a specified color
fig2, ax = plt.subplots(figsize=(16, 8))
ax.scatter(data['Close_ETF'], data['gold'], color='seagreen')
ax.set_xlabel('Close_ETF')
ax.set_ylabel('Full-value gold')
plt.show()
```



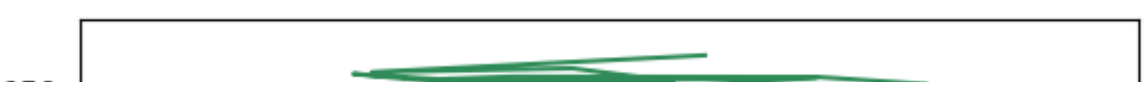


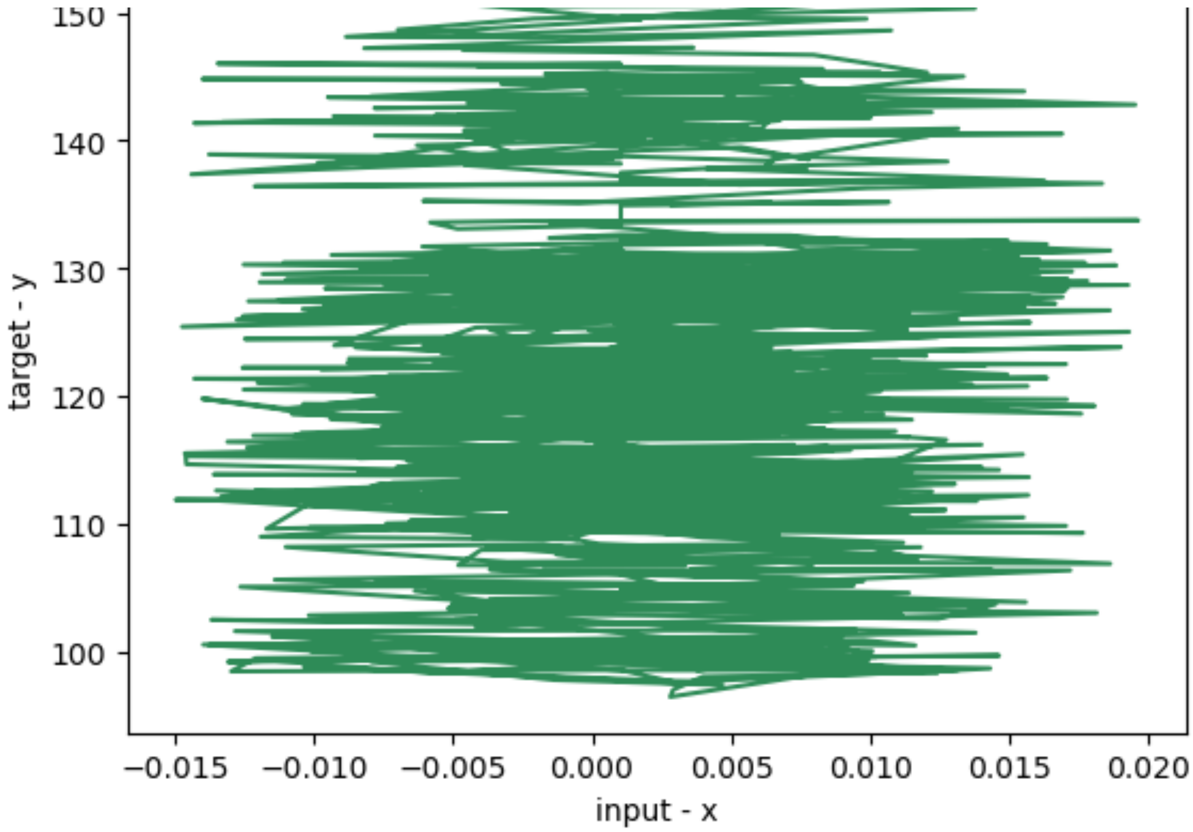
Close_ETF

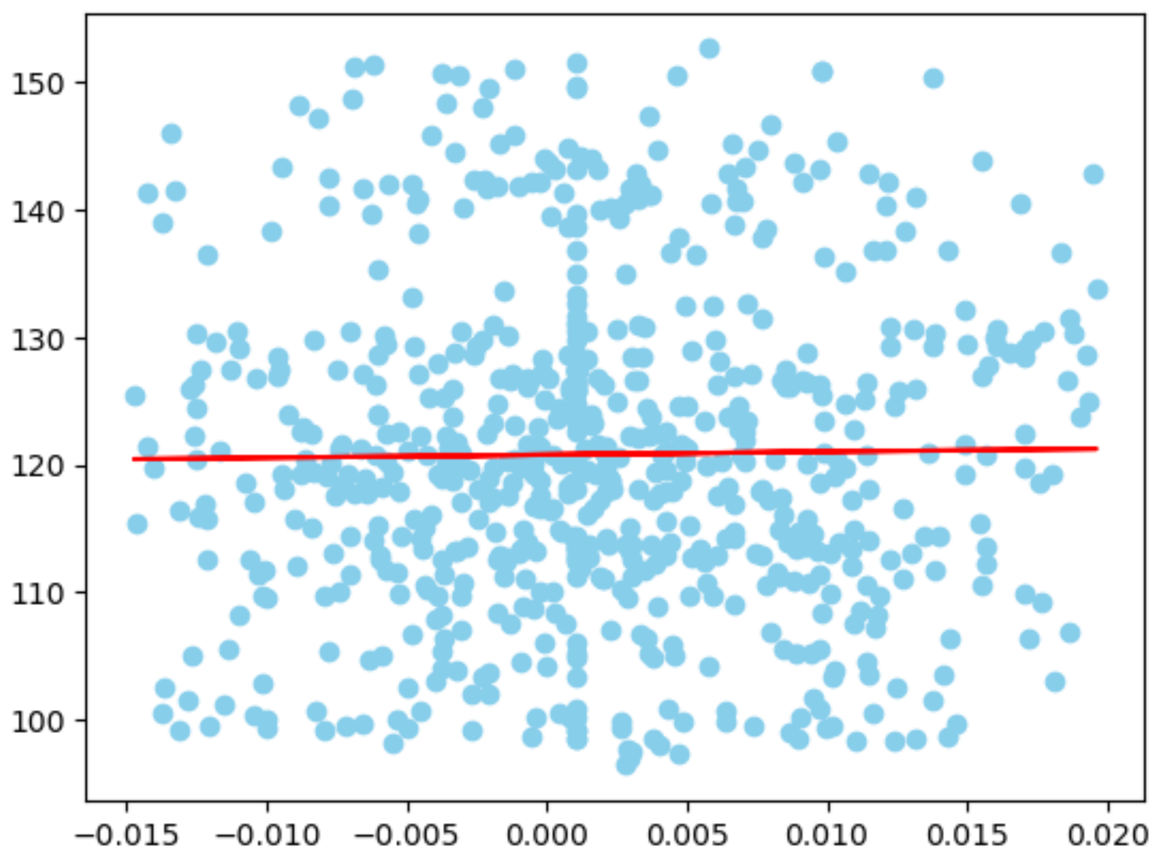
In [150]:
```
import numpy as np

# Modify 'Close_ETF' values based on conditions
data['Close_ETF'] = np.where(data['Close_ETF'] > 129, 121.152960, data
data['Close_ETF'] = np.where(data['Close_ETF'] < 112, 121.152960, data
```

In [275]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Assuming data is already loaded or created
data = data.iloc[:999]

# Preparing data
X = np.array(data['gold']).reshape(-1, 1)
y = np.array(data['Close_ETF']).reshape(-1, 1)

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Creating and training the model
model = LinearRegression()
model.fit(X_train, y_train)

# Cross-validation scores
scores = cross_val_score(model, X_train, y_train, cv=5)
print(scores)

# Plotting gold vs. Close_ETF
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(data['gold'], data['Close_ETF'], color='seagreen')  # Change c
ax.set_xlabel('input - x')
ax.set_ylabel('target - y')
plt.show()

# Plotting the training data and the regression line
plt.scatter(X_train, y_train, color='skyblue')  # Change color to oran
plt.plot(X_train, model.predict(X_train), color='red')
plt.show()
```

```
[-0.00168621 -0.00262029 -0.00780704 -0.00357968 -0.00072264]
```




150
140
130
120
110
100
target - y
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020
input - x


150
140
130
120
110
100
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015
0.020

In [278]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Calculating Mean Squared Error
MSE = mean_squared_error(y_test, y_pred)

# Calculating R2 Score
r2 = r2_score(y_test, y_pred)

# Printing Mean Squared Error and R2 Score
print('Mean squared error:', MSE)
print('R2 Score:', r2)
```

```
Mean squared error: 145.5942952742069
R2 Score: -0.007500697440531834
```

In [279]:
```
minvalueIndexLabel = data['Close_ETF'].idxmin()
print(minvalueIndexLabel)
```

```
45
```

Removing outliers from Close_ETF

In [280]:
```
import numpy as np

data['Close_ETF'] = np.where(data['Close_ETF'] > 129, 121.152960, data
data['Close_ETF'] = np.where(data['Close_ETF'] < 112, 121.152960, data
```

part 8 (7)

99% confidence interval of the mean daily ETF return,

In [281]:
```
import numpy as np
import scipy.stats as stats

def mean_confidence_interval(data, confidence=0.99):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = se * stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m - h, m + h

mean_confidence_interval(col1, confidence=0.99)
```

Out[281]:
```
(121.152960012, 120.12712955132923, 122.17879047267076)
```

In [282]:
```
import numpy as np
import scipy.stats as stats

def mean_confidence_interval(data, confidence=0.99):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = se * stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m - h, m + h

mean_confidence_interval(col2, confidence=0.99)
```

Out[282]:
```
(0.0006628360819999999, -0.0002584729930030417, 0.001584145157003041
6)
```

99% prediction interval of the individual daily ETF return.

In [283]:
```
import numpy as np
import scipy.stats as stats

def mean_confidence_interval(data, confidence=0.99):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = se * stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m - h, m + h

mean_confidence_interval(y_pred, confidence=0.99)
```

Out[283]:
```
(120.77166782216734, array([120.71860059]), array([120.82473506]))
```

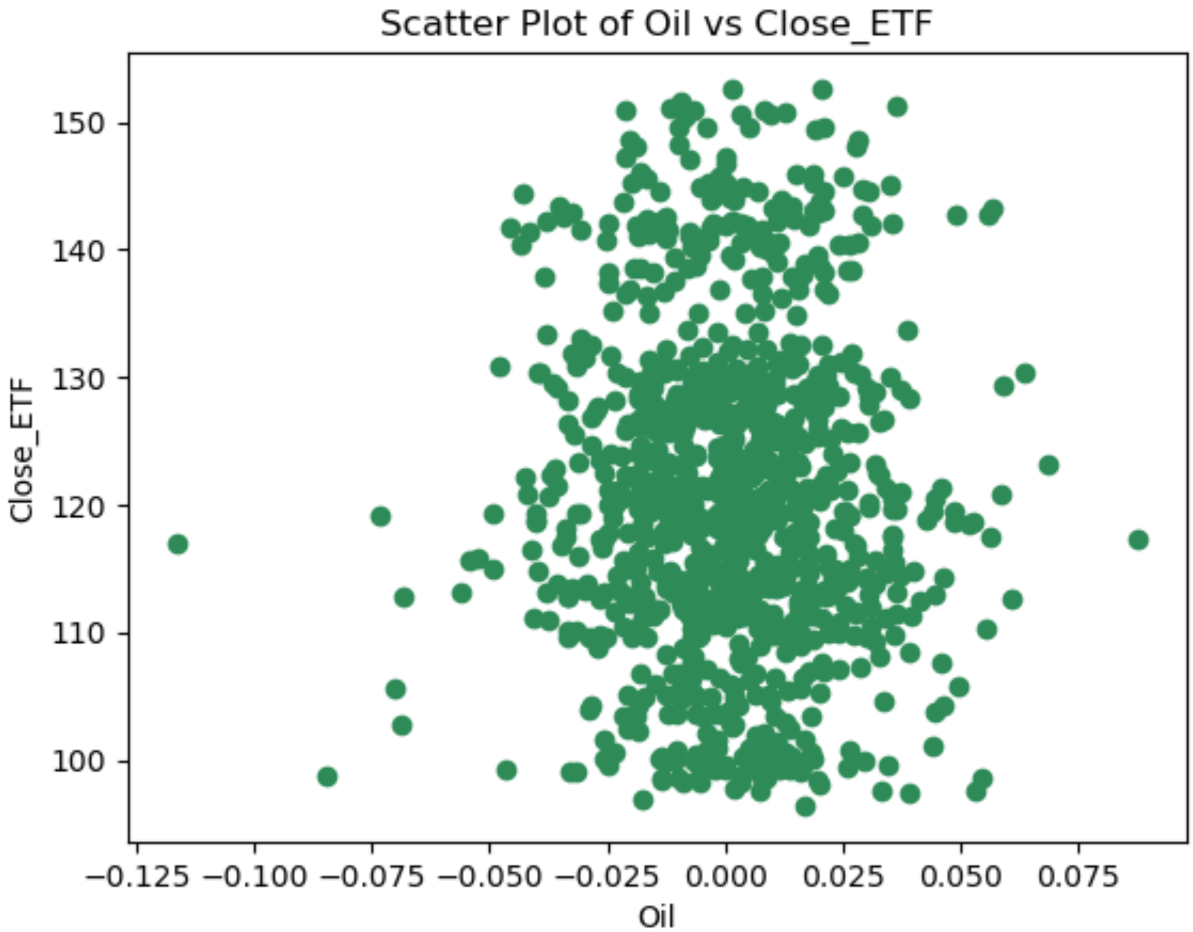
In [284]:
```
import numpy as np
import scipy.stats as stats

def mean_confidence_interval(data, confidence=0.99):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), stats.sem(a)
    h = se * stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m - h, m + h

mean_confidence_interval(X_test, confidence=0.99)
```

Out[284]:
```
(0.0013872294, array([0.00034262]), array([0.00243184]))
```

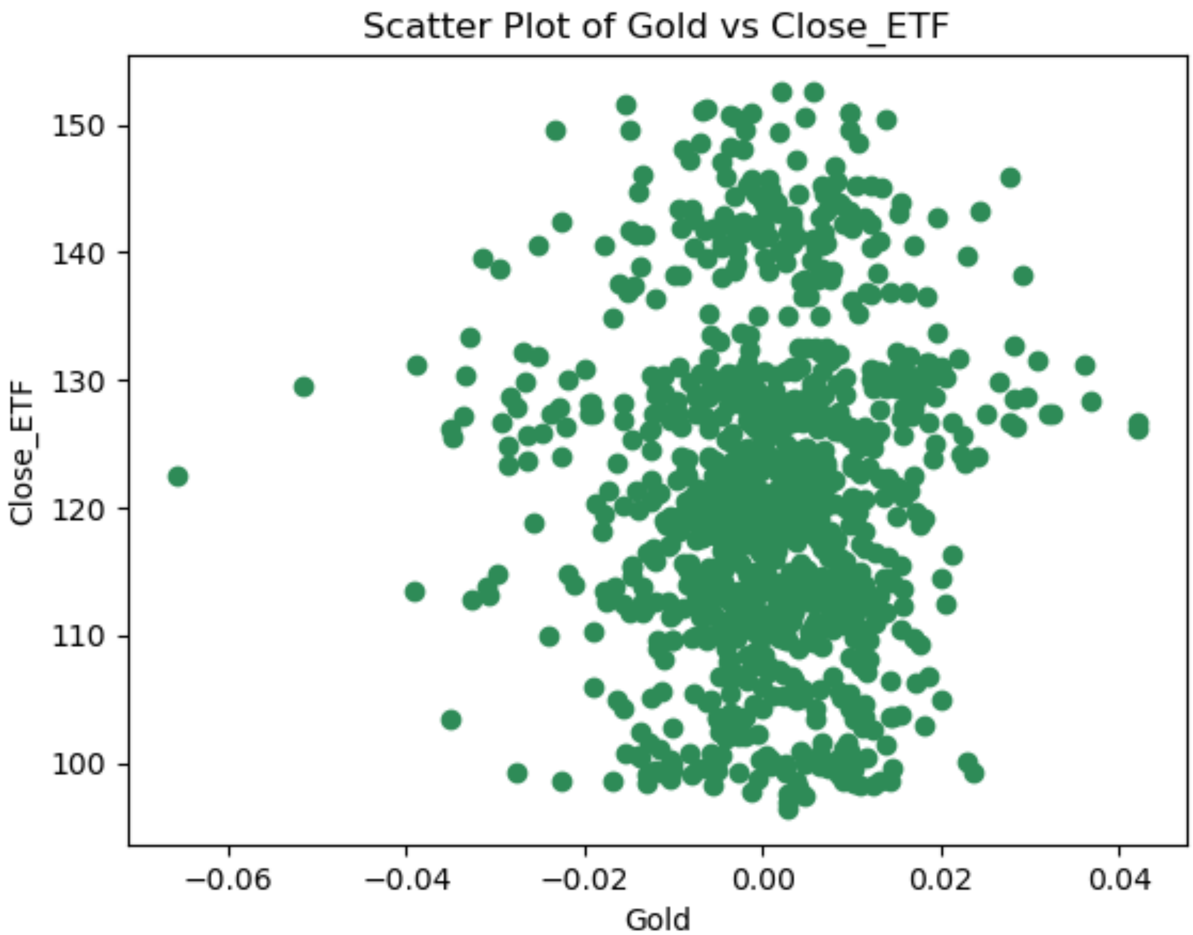
In [285]:
```
import pandas as pd

# Read the CSV file, specifying which columns to use and setting the i
df = pd.read_csv('Sample Course Project (Dataset).csv', usecols=lambda

print(df)
```

```
      Close_ETF       oil      gold       JPM
0     97.349998  0.039242  0.004668  0.032258
1     97.750000  0.001953 -0.001366 -0.002948
2     99.160004 -0.031514 -0.007937  0.025724
3     99.650002  0.034552  0.014621  0.011819
4     99.260002  0.013619 -0.011419  0.000855
..          ...       ...       ...       ...
995  150.570007  0.009752  0.004634  0.003859
996  151.600006 -0.009341 -0.015325  0.018259
997  151.300003  0.036120 -0.006195 -0.007928
998  152.619995  0.001542  0.005778 -0.000381
999  152.539993  0.020330  0.001965  0.000381

[1000 rows x 4 columns]
```

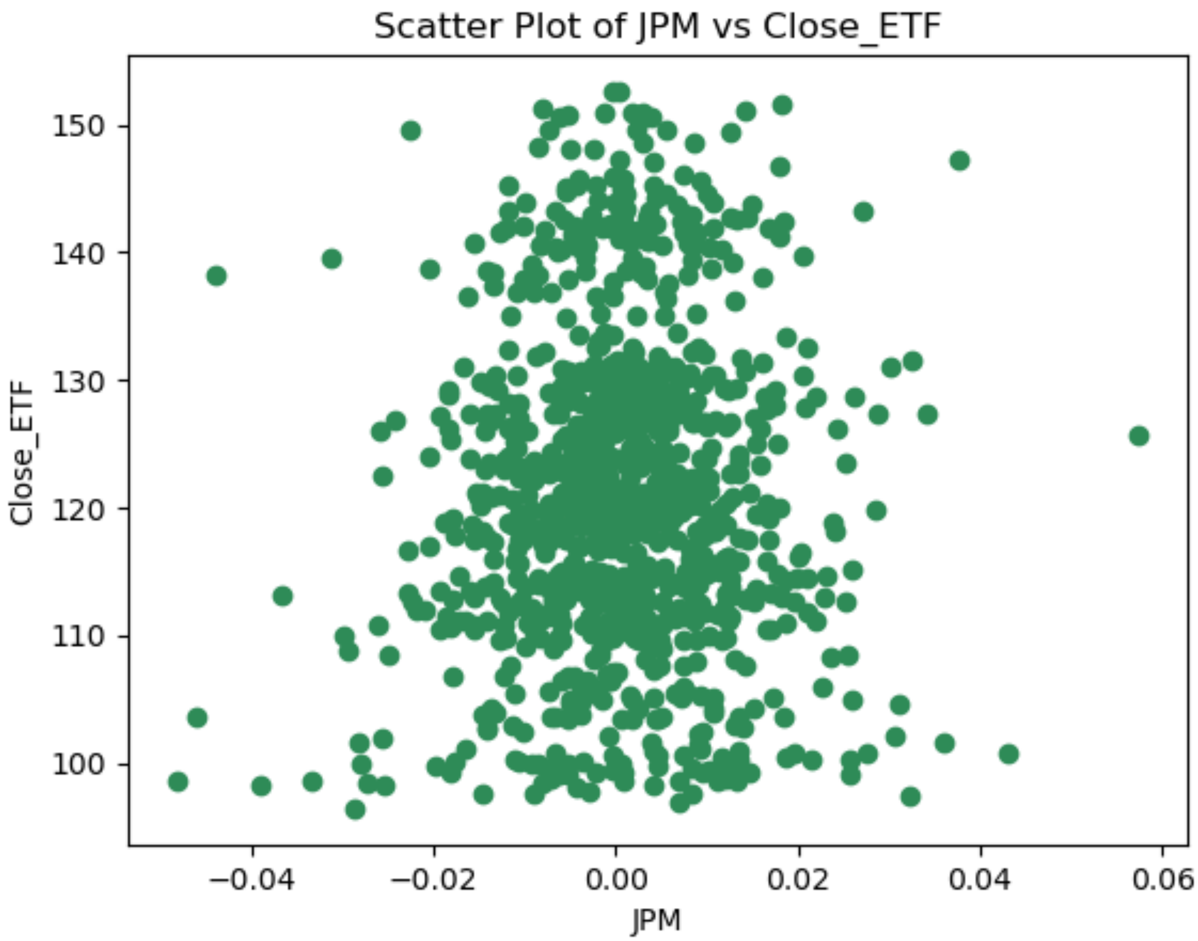
In [286]:
```
import matplotlib.pyplot as plt

# Scatter plot of 'oil' vs 'Close_ETF'
plt.scatter(df['oil'], df['Close_ETF'], color='seagreen')
plt.xlabel('Oil')
plt.ylabel('Close_ETF')
plt.title('Scatter Plot of Oil vs Close_ETF')
plt.show()
```



In [288]:
```
import matplotlib.pyplot as plt

# Scatter plot of 'gold' vs 'Close_ETF'
plt.scatter(df['gold'], df['Close_ETF'], color='seagreen')
plt.xlabel('Gold')
plt.ylabel('Close_ETF')
plt.title('Scatter Plot of Gold vs Close_ETF')
plt.show()
```



In [289]:
```
import matplotlib.pyplot as plt

# Scatter plot of 'JPM' vs 'Close_ETF'
plt.scatter(df['JPM'], df['Close_ETF'], color='seagreen')
plt.xlabel('JPM')
plt.ylabel('Close_ETF')
plt.title('Scatter Plot of JPM vs Close_ETF')
plt.show()
```



In [290]:
```
import pandas as pd
df = pd.read_csv('Sample Course Project (Dataset).csv', usecols=lambda
print(df)
```

```
      Close_ETF       oil      gold       JPM
0     97.349998  0.039242  0.004668  0.032258
1     97.750000  0.001953 -0.001366 -0.002948
2     99.160004 -0.031514 -0.007937  0.025724
3     99.650002  0.034552  0.014621  0.011819
4     99.260002  0.013619 -0.011419  0.000855
..          ...       ...       ...       ...
995  150.570007  0.009752  0.004634  0.003859
996  151.600006 -0.009341 -0.015325  0.018259
997  151.300003  0.036120 -0.006195 -0.007928
998  152.619995  0.001542  0.005778 -0.000381
999  152.539993  0.020330  0.001965  0.000381

[1000 rows x 4 columns]
```

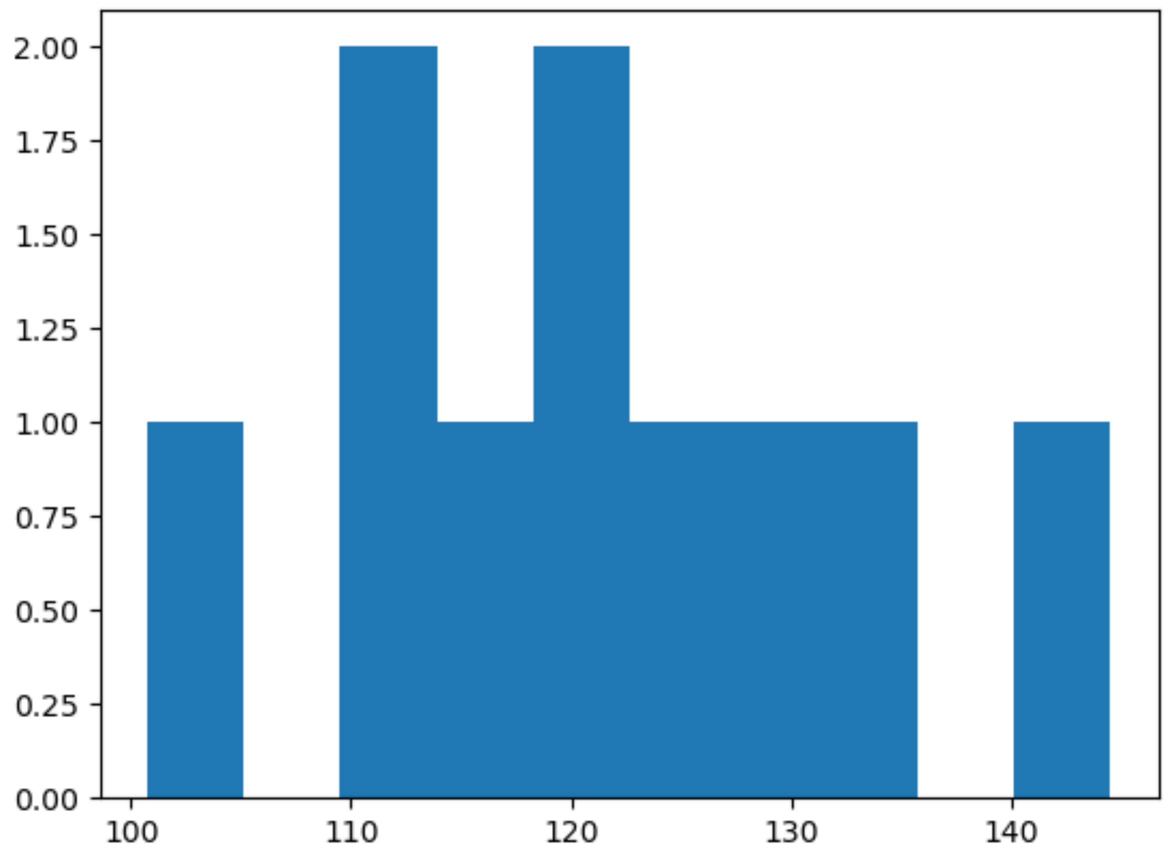
In [291]:
```
print("Mean of population x:", df['Close_ETF'].mean())
print("Std of population x:", df['Close_ETF'].std())
```

```
Mean of population x: 121.152960012
Std of population x: 12.569790313110744
```

In [292]:
```
import matplotlib.pyplot as plt
import numpy as np
import statistics

# Calculate the size of each index
index_size = len(df) // 100

x = 0
y = 100
histogram = {}

# Iterate through 10 samples
for i in range(10):
    approx_1 = df.iloc[x:y]  # Splitting into 100 values
    x += 100
    y += 100
    print(str(i) + "th mean is: " + str(approx_1['Close_ETF'].mean()))
    histogram[i] = approx_1['Close_ETF'].mean()

# Plotting histogram
plt.hist(list(histogram.values()))
plt.show()

print(list(histogram.values()))
```

```
print("\nMean of Sample means: " + str(statistics.mean(histogram.value
print("Median of Sample means: " + str(statistics.median(histogram.val
print("Mode of Sample means: " + str(statistics.mode(histogram.values(
print("Standard deviation of sample means: " + str(statistics.stdev(hi
```

```
0th mean is: 100.77430028999999
1th mean is: 110.48050028
2th mean is: 112.01809938999999
3th mean is: 114.51720014
4th mean is: 118.40030004
5th mean is: 121.67680030000001
6th mean is: 125.78560011000002
7th mean is: 128.01269998
8th mean is: 135.39209964
9th mean is: 144.47199995
```

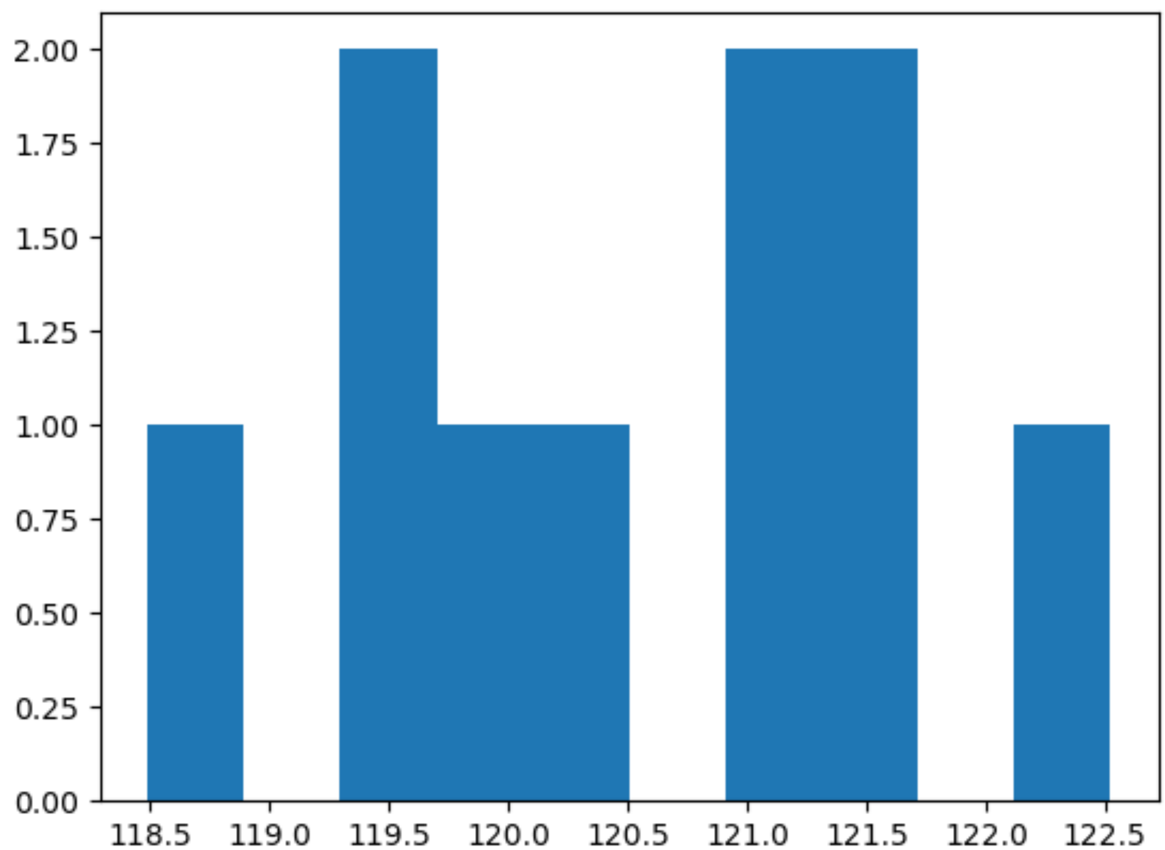



```
[100.77430028999999, 110.48050028, 112.01809938999999, 114.51720014,
118.40030004, 121.67680030000001, 125.78560011000002, 128.01269998, 1
35.39209964, 144.47199995]

Mean of Sample means: 121.152960012
Median of Sample means: 120.03855017000001
Mode of Sample means: 100.77430028999999
Standard deviation of sample means: 12.821725528306828
```

In [203]:

In [293]:

```
import matplotlib.pyplot as plt
import random
import statistics

# Creating a population (replace 'df.Close_ETF.tolist()' with your own
population = df['Close_ETF'].tolist()

sample_size = 10
value = 100
histogram = {}

for x in range(sample_size):
    # Creating a random sample of the population with size 10
    sample = random.sample(population, value)  # With Replacement mean
    histogram[x] = statistics.mean(sample)

# Plotting histogram
plt.hist(list(histogram.values()))
plt.show()

print(list(histogram.values()))
print("\nMean of Sample means:", statistics.mean(histogram.values()))
print("Median of Sample means:", statistics.median(histogram.values())
print("Mode of Sample means:", statistics.mode(histogram.values()))
print("Standard deviation of sample means:", statistics.stdev(histogra
```



```
[120.14869966, 121.46130024, 118.49240016, 121.04559972999999, 119.30
369977, 119.91599957, 121.4094999, 122.52070036, 121.14399955, 119.36
639984]

Mean of Sample means: 120.480829878
Median of Sample means: 120.597149695
Mode of Sample means: 120.14869966
Standard deviation of sample means: 1.2362293398418325
```

In [294]:

```
import numpy as np
import pandas as pd
from scipy import stats
import statistics

# Load data from Excel file
df = pd.read_csv('Sample Course Project (Dataset).csv')

# Define F-test function
def f_test(x, y):
    x = np.array(x)
    y = np.array(y)
    f = np.var(x, ddof=1) / np.var(y, ddof=1)  # Calculate F test stat
    dfn = x.size - 1  # Define degrees of freedom numerator
    dfd = y.size - 1  # Define degrees of freedom denominator
    p = 1 - stats.f.cdf(f, dfn, dfd)  # Find p-value of F test statist
    return f, p

# Perform F-test
f_val, p_val = f_test(df['oil'], df['gold'])
print("Variance of oil:", statistics.variance(df['oil']))
print("Variance of gold:", statistics.variance(df['gold']))
print("F value:", f_val)
print("p value:", p_val)
if p_val < 0.05:
    print("Null hypothesis is rejected")
else:
    print("Null hypothesis is accepted")
```

```
Variance of oil: 0.0004449103692830018
Variance of gold: 0.00012744288153847104
F value: 3.491057043846716
p value: 1.1102230246251565e-16
Null hypothesis is rejected
```

In [295]:

```
import numpy as np

x = [18, 19, 22, 25, 27, 28, 41, 45, 51, 55,14, 15, 15, 17, 18, 22, 25
y = [14, 15, 15, 17, 18, 22, 25, 25, 27, 34,18, 19, 22, 25, 27, 28, 41

#define F-test function
def f_test(x, y):
    x = np.array(x)
    y = np.array(y)
    f = np.var(x, ddof=1)/np.var(y, ddof=1) #calculate F test statisti
    dfn = x.size-1 #define degrees of freedom numerator
    dfd = y.size-1 #define degrees of freedom denominator
    p = 1-scipy.stats.f.cdf(f, dfn, dfd) #find p-value of F test stati
    return f, p

#perform F-test
fVal,pVal=f_test(x, y)
print("F value :"+str(fVal));
print("p value :"+str(pVal));
print(statistics.variance(x))
print(statistics.variance(y))
```

```
F value :1.1992714779857212
p value :0.18435452419273057
168.47790780988774
140.48354430379746
```

In [296]:

```
import numpy as np
import pandas as pd
import scipy.stats
import statistics

# Read the data from csv
df = pd.read_csv('Sample Course Project (Dataset).csv')

# Set the significance level
alpha = 0.05

# Define null and alternative hypotheses
H0 = "Standard deviations are the same"
Ha = "Standard deviations are different"

# Define F-test function
def f_test(x, y):
    x = np.array(x)
    y = np.array(y)
    f = np.std(x, ddof=1) / np.std(y, ddof=1)  # Calculate F test stat
    dfn = x.size - 1  # Define degrees of freedom numerator
    dfd = y.size - 1  # Define degrees of freedom denominator
```

```
    p = 1 - scipy.stats.f.cdf(f, dfn, dfd)  # Find p-value of F test s
    return f, p

# Perform F-test
fVal, pVal = f_test(df.oil, df.gold)

# Print standard deviations
print("Standard deviation of Oil:", statistics.stdev(df.oil))
print("Standard deviation of Gold:", statistics.stdev(df.gold))

# Check null hypothesis
if statistics.stdev(df.oil) == statistics.stdev(df.gold):
    print(H0)
else:
    print(Ha)

# Print F-value and p-value
print("F value:", fVal)
print("p value:", pVal)

# Check the decision
if pVal < alpha:
    print("Null hypothesis is rejected")
else:
    print("Null hypothesis is accepted")
```

```
Standard deviation of Oil: 0.021092898551005531
Standard deviation of Gold: 0.011289060259316142
Standard deviations are different
F value: 1.8684370591076158
p value: 1.1102230246251565e-16
Null hypothesis is rejected
```

In [297]:
```
import statistics as s
import matplotlib.pyplot as plt
import random
from scipy import stats
from statsmodels.stats import weightstats as stests
import numpy as np

# Creating a population replace with your own:
goldData = df.gold.tolist()
oilData = df.oil.tolist()
apprix_1 = df.iloc[0:100:]

value = 10
alpha = 0.05

goldMean = s.mean(goldData)
oilMean = s.mean(oilData)
#print("Gold's mean is:", goldMean)
#print("Oil's mean is:", oilMean)
ttest, pval = stests.ztest(apprix_1.gold, apprix_1.oil)
print("p-value:", pval)
if pval < alpha:
    print("We reject the null hypothesis")
else:
    print("We accept the null hypothesis")
```

```
p-value: 0.9831528153294554
We accept the null hypothesis
```

In [ ]: