In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics
from numpy import mean
from numpy import std
from statsmodels.graphics.gofplots import qqplot
import statsmodels.graphics.gofplots as sm
```

Part 1

In [2]:
```
project_data = pd.read_csv('data.csv')
project_data.head()
```

Out[2]:

	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

In [3]:
```
pop_means = project_data.mean()
print(pop_means)
```

```
Close_ETF    121.152960
oil            0.001030
gold           0.000663
JPM            0.000530
dtype: float64
```

In [4]:
```
pop_stdev = project_data.std()
print(pop_stdev)
```

```
Close_ETF    12.569790
oil           0.021093
gold          0.011289
JPM           0.011017
dtype: float64
```

In [5]:
```
project_data.corr(method='pearson')
```

Out[5]:

	Close_ETF	oil	gold	JPM
Close_ETF	1.000000	-0.009045	0.022996	0.036807
oil	-0.009045	1.000000	0.235650	-0.120849
gold	0.022996	0.235650	1.000000	0.100170

	Close_ETF	oil	gold	JPM
JPM	0.036807	-0.120849	0.100170	1.000000

Part 2

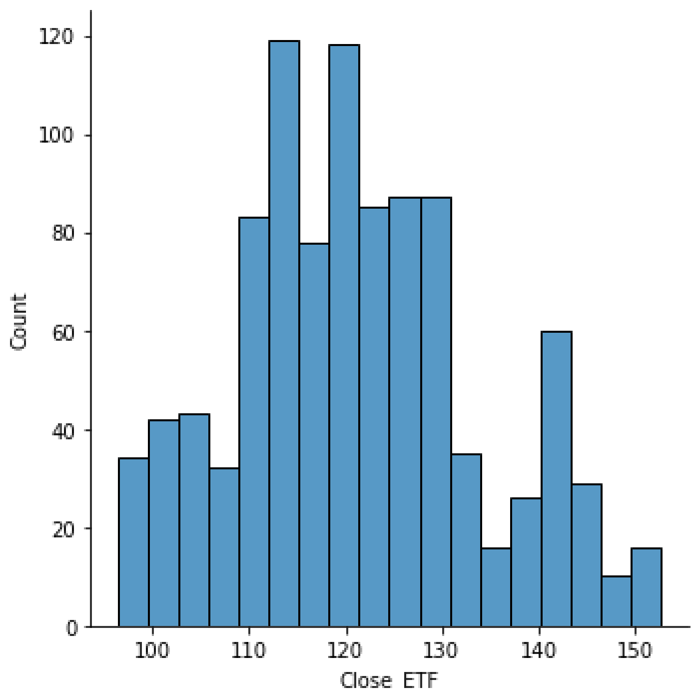
Histogram plots

In [6]:
```
project_data.head()
```

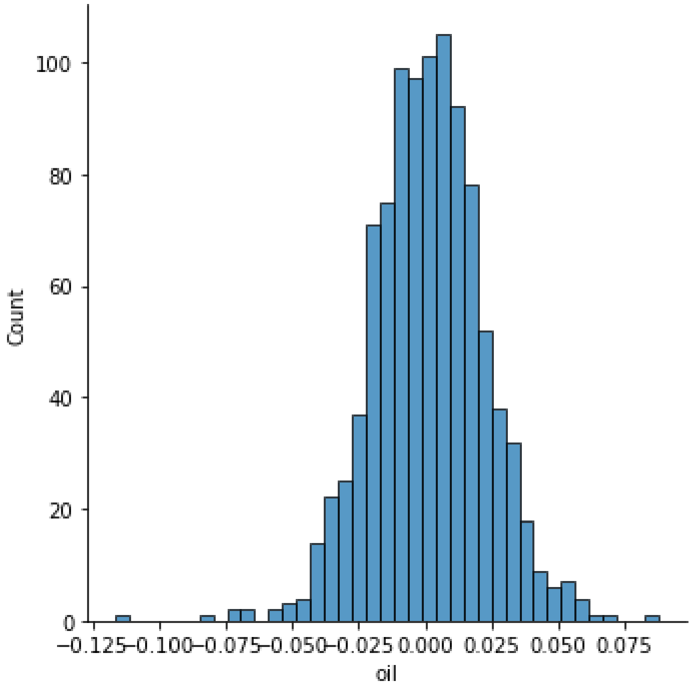
Out[6]:

	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

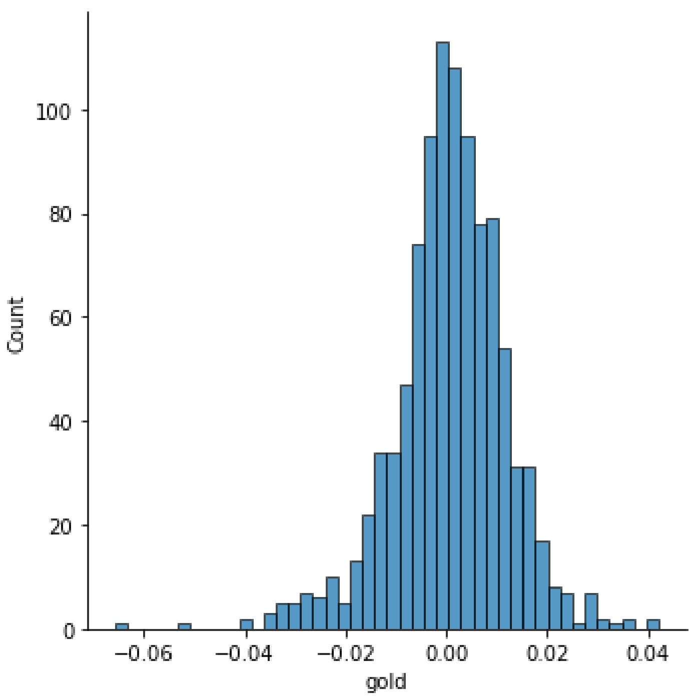
In [7]:
```
hist_Close_ETF = sns.displot(project_data, x="Close_ETF")
```



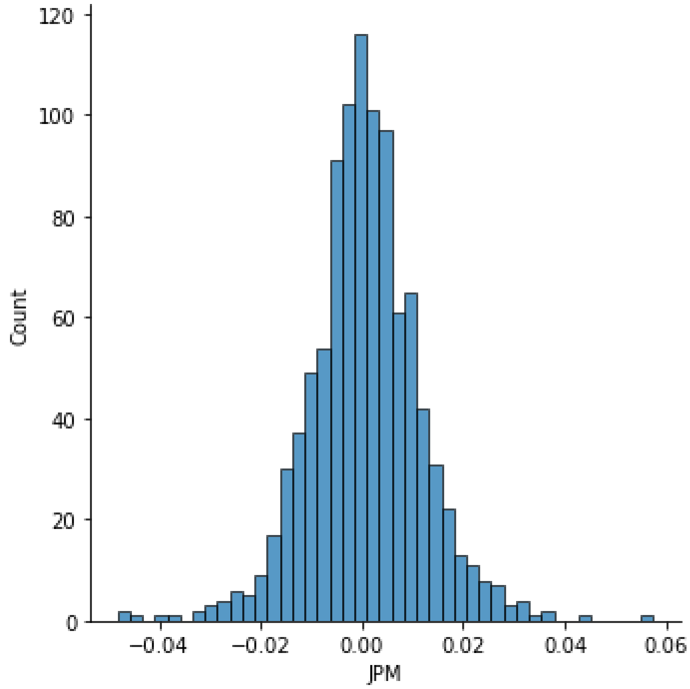
In [8]:
```
hist_oil = sns.displot(project_data, x="oil")
```



In [9]:
```
hist_gold = sns.displot(project_data, x="gold")
```



In [10]:
```
hist_JPM = sns.displot(project_data, x="JPM")
```



Time series plots

In [11]:

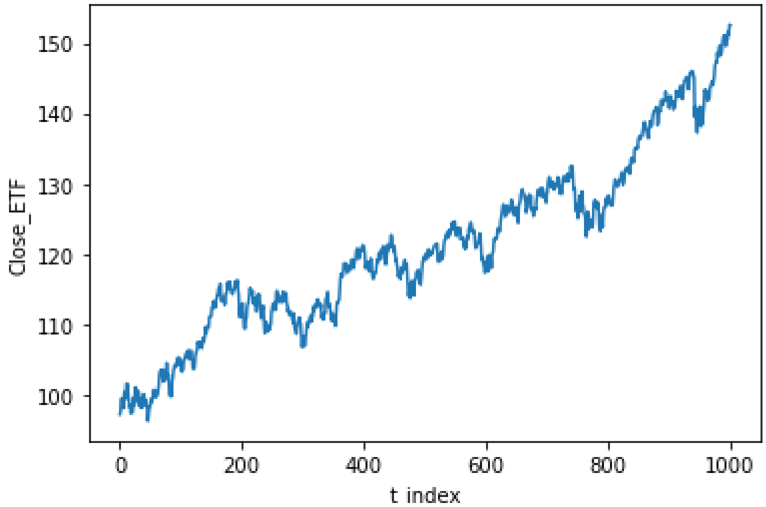
```
project_data.insert(
    loc=0,
    column='t_index',
    value=np.arange(1,1001)
)

project_data.head()
```

Out[11]:

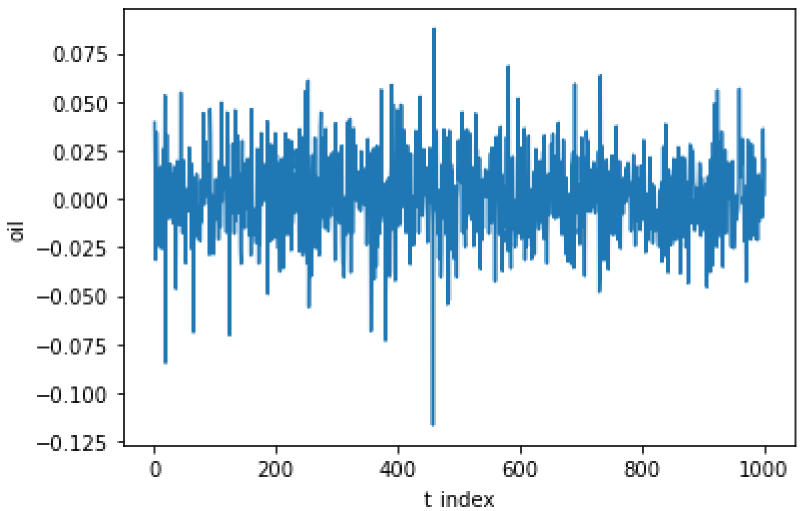
	t_index	Close_ETF	oil	gold	JPM
0	1	97.349998	0.039242	0.004668	0.032258
1	2	97.750000	0.001953	-0.001366	-0.002948
2	3	99.160004	-0.031514	-0.007937	0.025724
3	4	99.650002	0.034552	0.014621	0.011819
4	5	99.260002	0.013619	-0.011419	0.000855

In [12]:

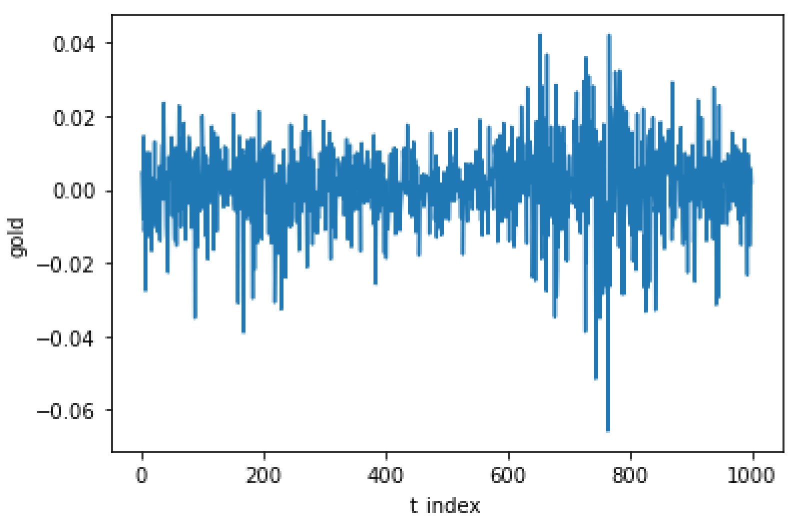
```
ts_Close_ETF = sns.lineplot(data=project_data, x="t_index", y="Close_ETF")
```



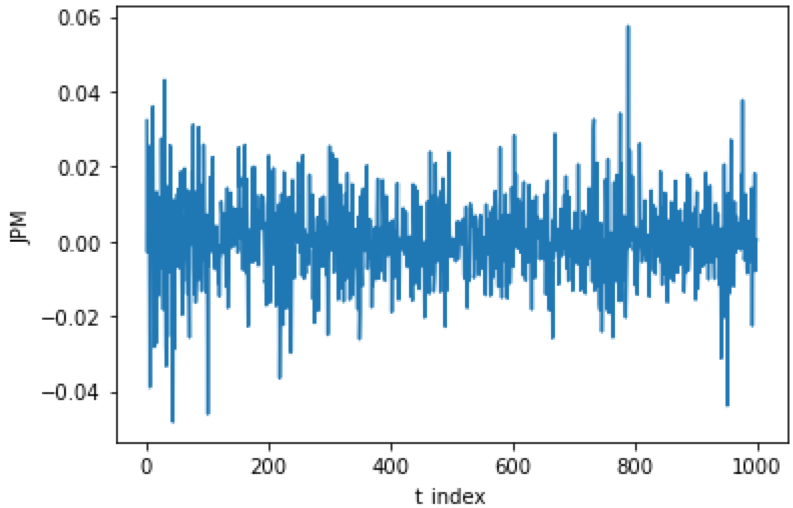
In [13]:
```
ts_oil = sns.lineplot(data=project_data, x="t_index", y="oil")
```



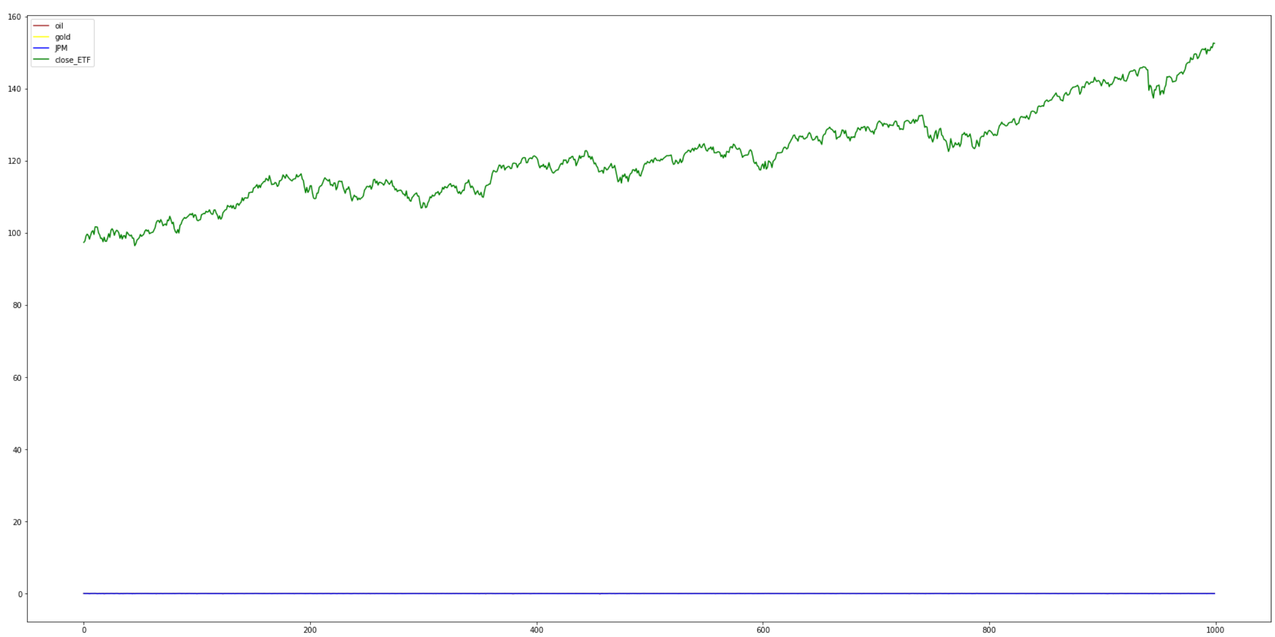
In [14]:
```
ts_gold = sns.lineplot(data=project_data, x="t_index", y="gold")
```



In [15]:
```
ts_JPM = sns.lineplot(data=project_data, x="t_index", y="JPM")
```

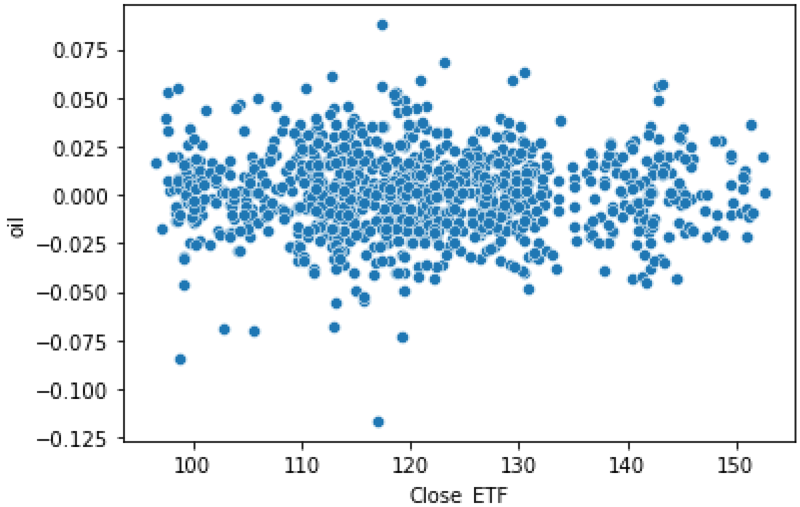


In [16]:
```
fig, ax = plt.subplots(figsize=(30,15))
ax.plot(project_data['oil'], color='brown', label='oil')
ax.plot(project_data['gold'], color='yellow', label='gold')
ax.plot(project_data['JPM'], color = 'blue', label='JPM')
ax.plot(project_data['Close_ETF'], color='green', label='close_ETF')
ax.legend(loc='upper left')
plt.show()
```



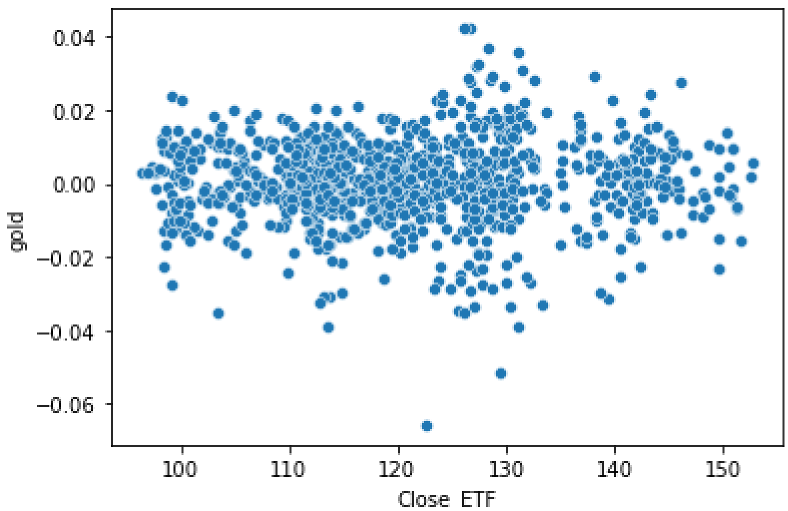
Scatter plots

In [17]:
```
scatter_ETF_oil = sns.scatterplot(data=project_data, x="Close_ETF", y="oil")
```



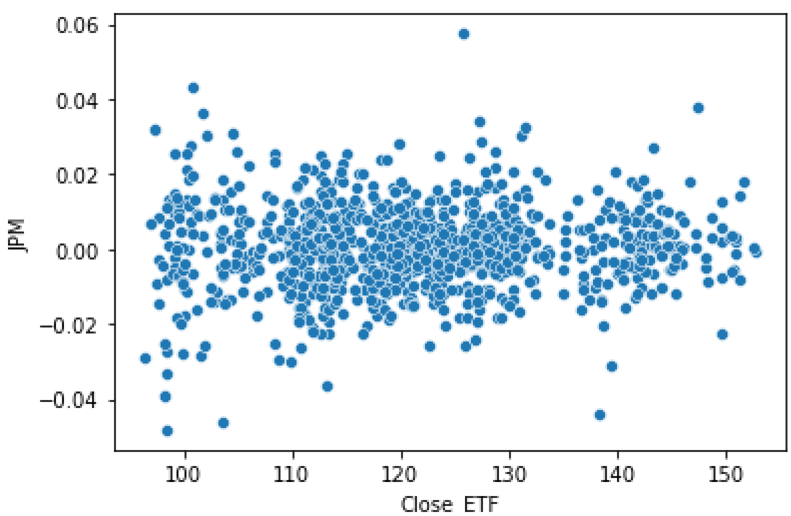
In [18]:

```
scatter_ETF_gold = sns.scatterplot(data=project_data, x="Close_ETF", y="gold")
```



In [19]:

```
scatter_ETF_JPM = sns.scatterplot(data=project_data, x="Close_ETF", y="JPM")
```



part 3

In [20]:
```
project_data.head()
```

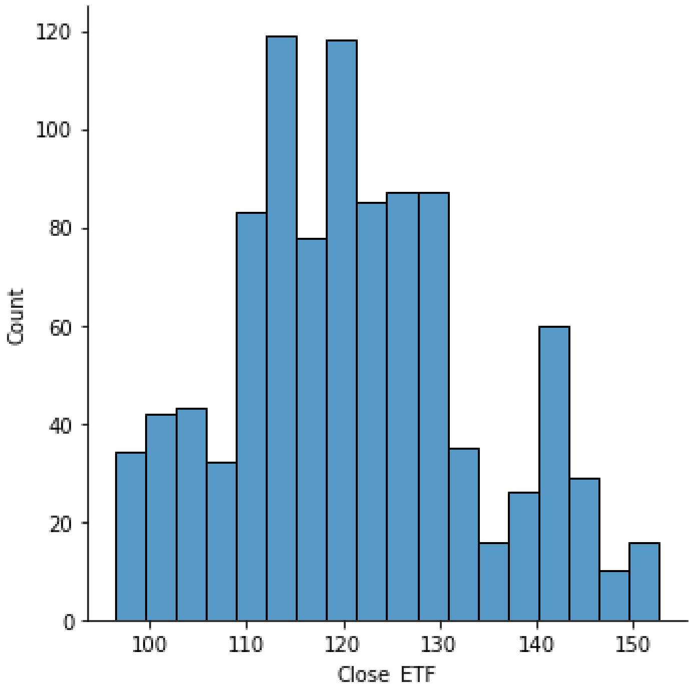
Out[20]:

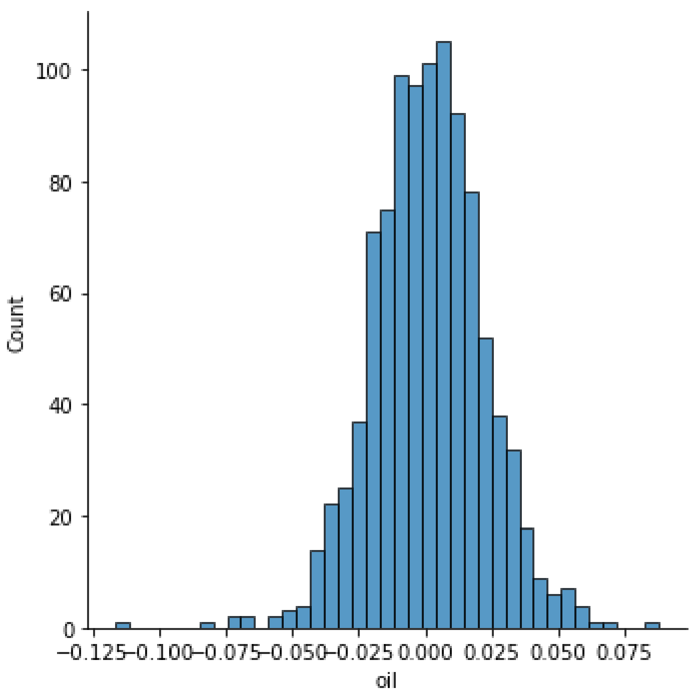
	t_index	Close_ETF	oil	gold	JPM
0	1	97.349998	0.039242	0.004668	0.032258
1	2	97.750000	0.001953	-0.001366	-0.002948
2	3	99.160004	-0.031514	-0.007937	0.025724
3	4	99.650002	0.034552	0.014621	0.011819
4	5	99.260002	0.013619	-0.011419	0.000855

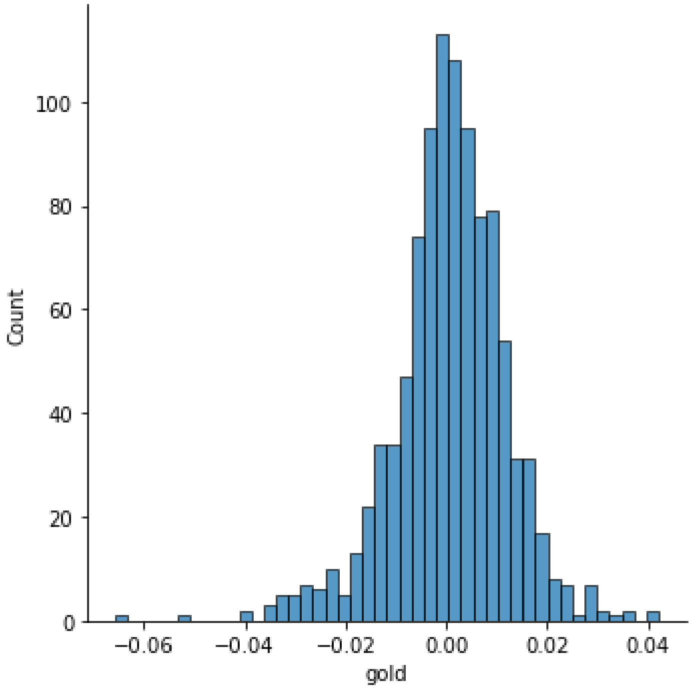
In [21]:
```
#mean_close_etf = project_data('Close_ETF').mean()

mean_project_data = project_data.mean()
std_project_data = project_data.std()
print('Mean is: \n',mean_project_data)
print('Standard Deviation is: \n',std_project_data)
```

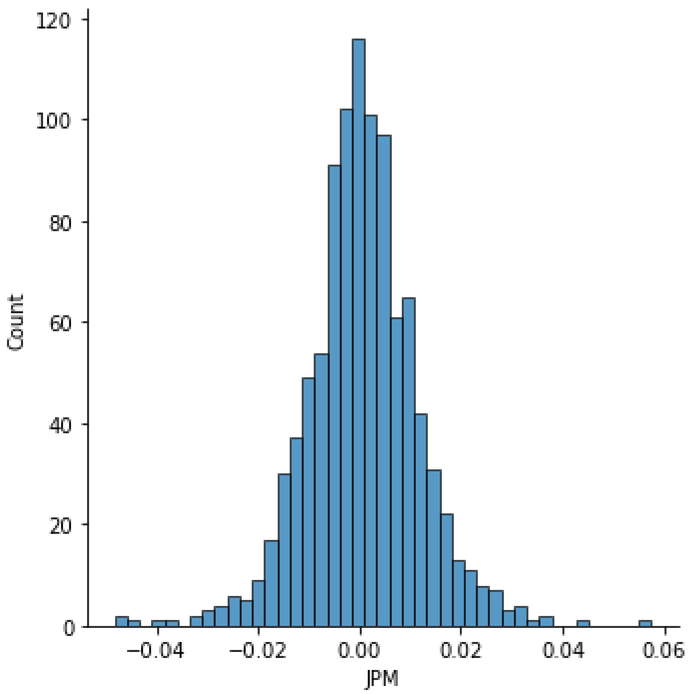
```
Mean is:
 t_index      500.500000
Close_ETF    121.152960
oil            0.001030
gold           0.000663
JPM            0.000530
dtype: float64
Standard Deviation is:
 t_index      288.819436
Close_ETF     12.569790
oil            0.021093
gold           0.011289
JPM            0.011017
dtype: float64
```

In [22]:
```
hist_Close_ETF = sns.displot(project_data, x="Close_ETF")
hist_oil = sns.displot(project_data, x="oil")
hist_gold = sns.displot(project_data, x="gold")
hist_JPM = sns.displot(project_data, x="JPM")
```


Count
Close_ETF

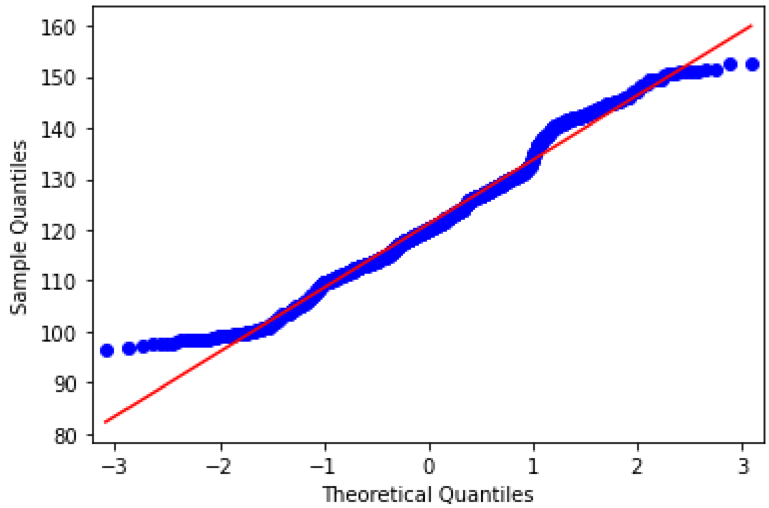

Count
oil



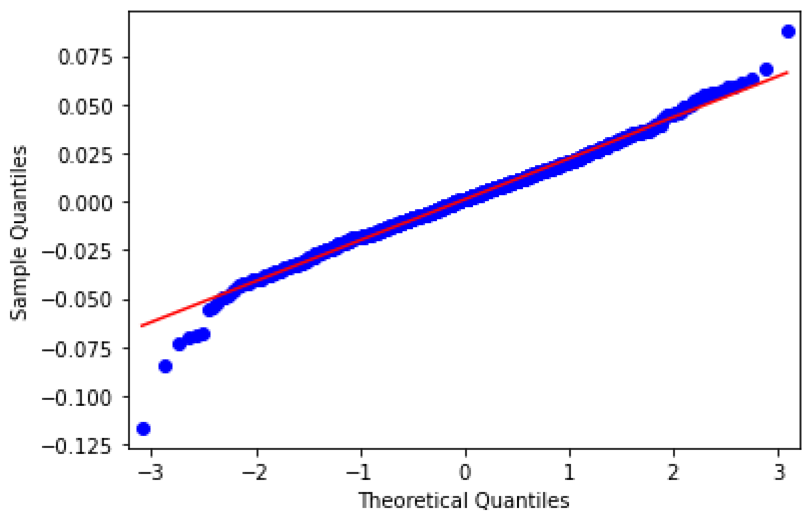


By observing the histogram, the variables are following normal distribution

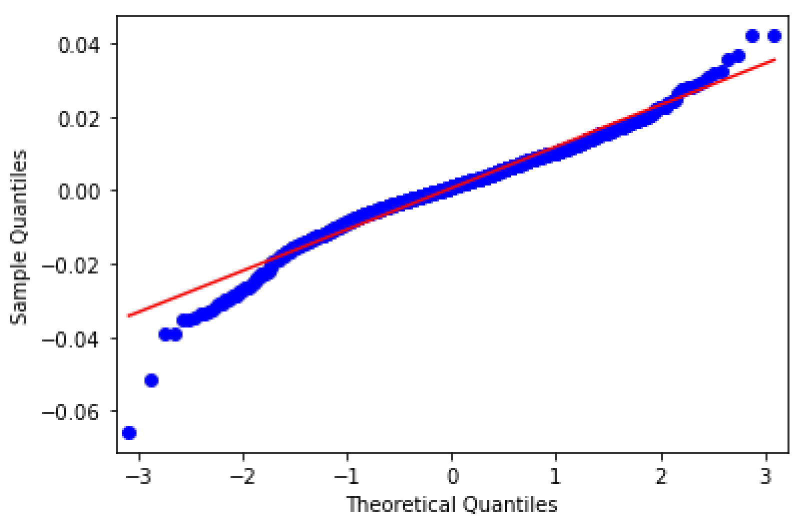
In [23]:
```
qqplot_project_data_close_etf = qqplot(project_data['Close_ETF'],line='s').gca()
```



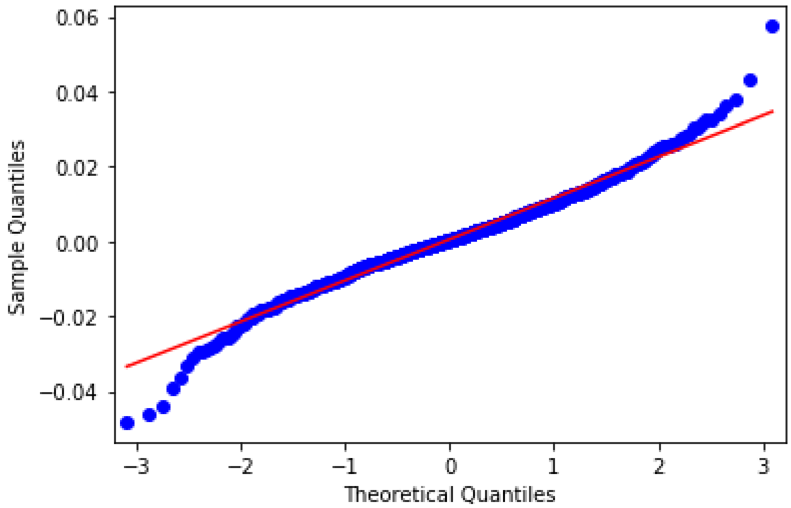
In [24]:
```
qqplot_project_data_oil = qqplot(project_data['oil'],line='s').gca().lines
```



In [25]:
```
qqplot_project_data_gold = qqplot(project_data['gold'],line='s').gca().lines
```



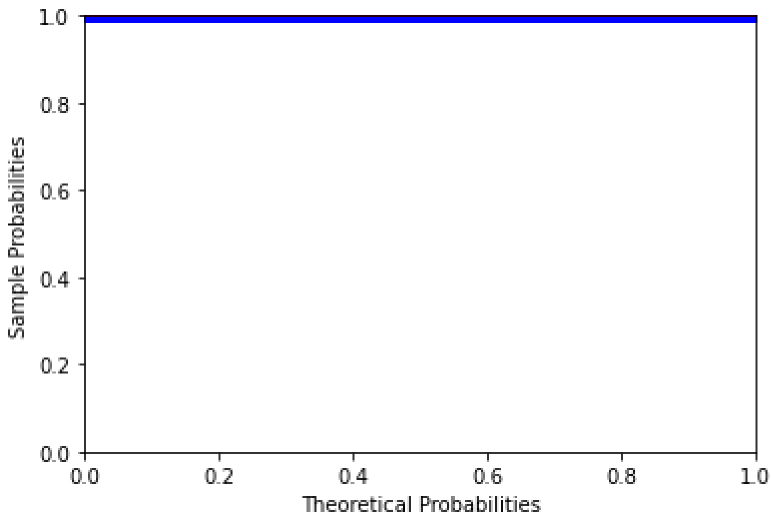
In [26]:
```
qqplot_project_JPM = qqplot(project_data['JPM'],line='s').gca().lines
```

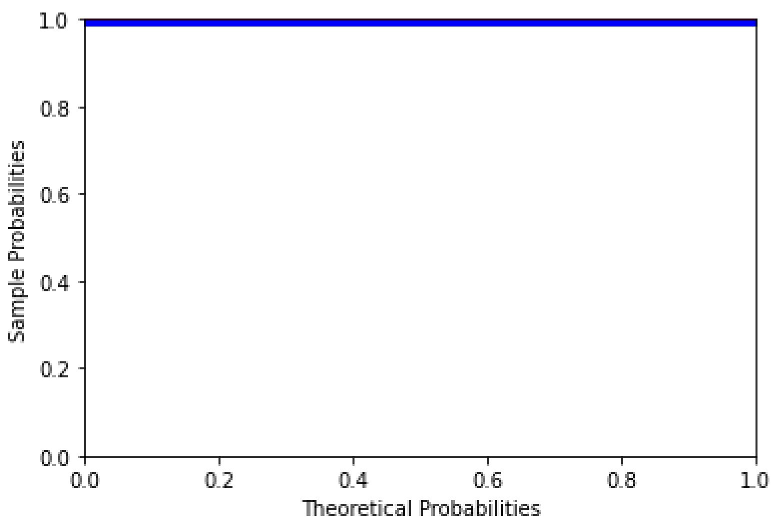


In [27]:

```
#ppplot_project_data_close_etf = ppplot(project_data['Close_ETF'],line='s').gca(
sm.ProbPlot(np.array(project_data['Close_ETF'])).ppplot(line='s')
```

Out[27]:

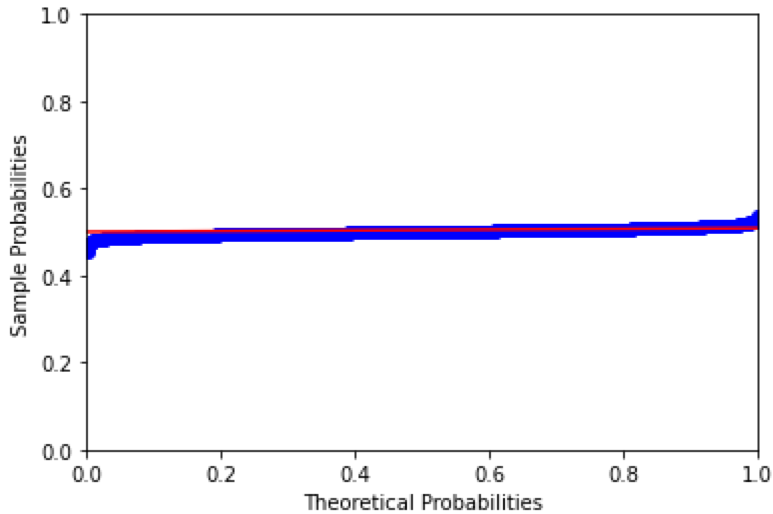


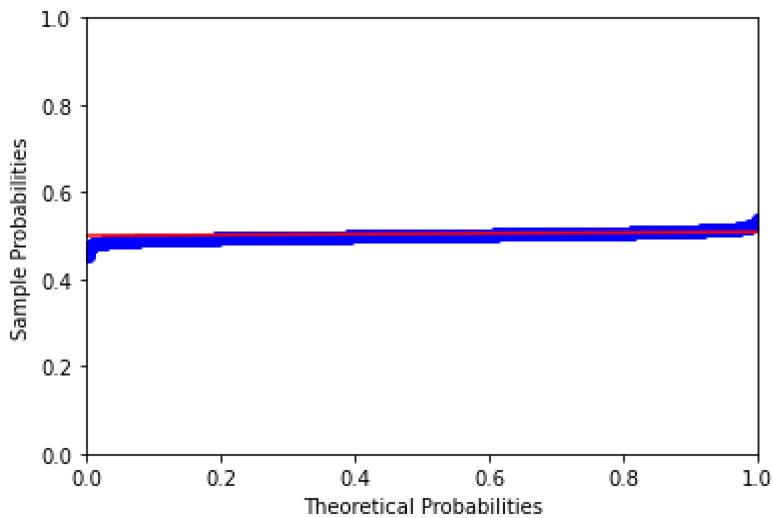


In [28]:

```
sm.ProbPlot(np.array(project_data['oil'])).ppplot(line='s')
```

Out[28]:

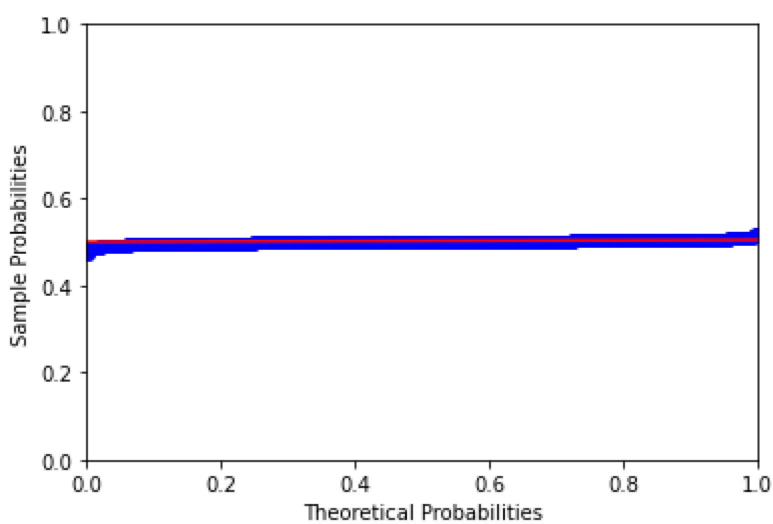


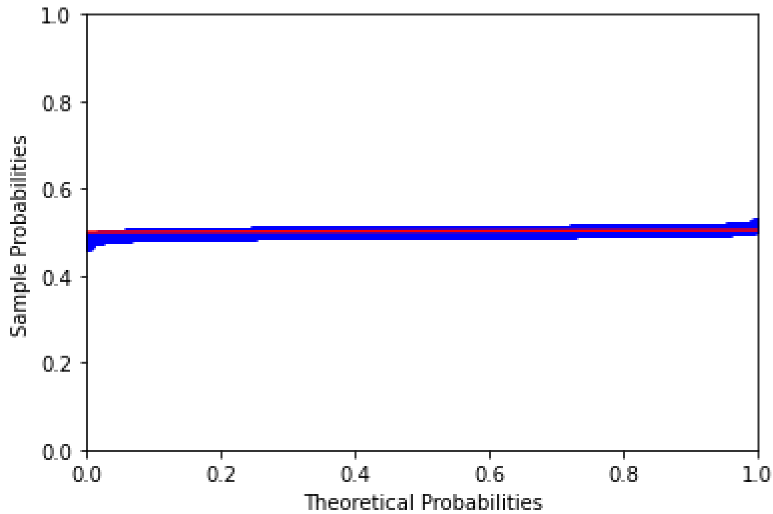


In [29]:

```
sm.ProbPlot(np.array(project_data['gold'])).ppplot(line='s')
```

Out[29]:

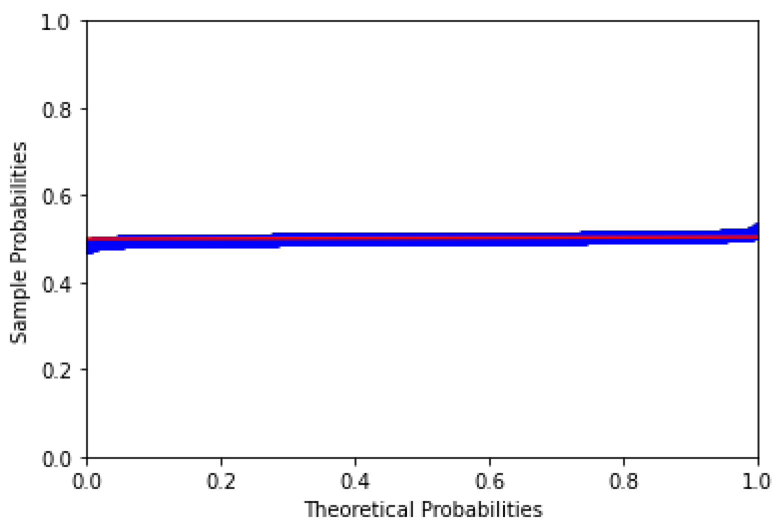


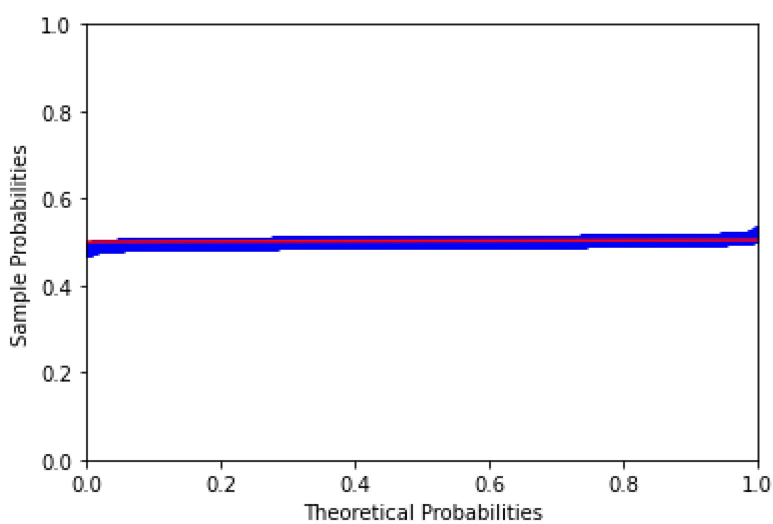


In [30]:

```
sm.ProbPlot(np.array(project_data['JPM'])).ppplot(line='s')
```

Out[30]:





In [31]:

```
from scipy.stats import shapiro
mean_closed_etf = project_data['Close_ETF'].mean()
std_closed_etf = project_data['Close_ETF'].std()
norm_closed_etf = (project_data['Close_ETF'] - mean_closed_etf)/std_closed_etf
stat_c, p_c = shapiro(norm_closed_etf)
```

```
# interpret
alpha = 0.0001
if p_c > alpha:
    msg_c = 'Closed-ETF looks Gaussian (fail to reject H0)'
else:
    msg_c = 'Closed-ETF does not look Gaussian (reject H0)'

print(msg_c)
```

```
Closed-ETF does not look Gaussian (reject H0)
```

In [32]:

```
mean_oil = project_data['oil'].mean()
std_oil = project_data['oil'].std()
norm_oil = (project_data['oil'] - mean_oil)/std_oil
stat_o, p_o = shapiro(norm_oil)

# interpret
alpha = 5.488e-08
if p_o > alpha:
    msg_o = 'Oil looks Gaussian (fail to reject H0)'
else:
    msg_o = 'Oil does not look Gaussian (reject H0)'

print(msg_o,p_o)
```

```
Oil looks Gaussian (fail to reject H0) 5.488897727445874e-07
```

In [33]:

```
mean_gold = project_data['gold'].mean()
std_gold = project_data['oil'].std()
norm_gold = (project_data['oil'] - mean_oil)/std_oil
stat_g, p_g = shapiro(project_data['gold'])

# interpret
alpha = 0.05
if p_g > alpha:
    msg_g = 'Gold looks Gaussian (fail to reject H0)'
else:
    msg_g = 'Gold does not look Gaussian (reject H0)'

print(msg_g)
```

```
Gold does not look Gaussian (reject H0)
```

In [34]:

```
stat_j, p_j = shapiro(project_data['JPM'])

# interpret
alpha = 0.05
if p_j > alpha:
    msg_j = 'JPM looks Gaussian (fail to reject H0)'
else:
    msg_j = 'JPM does not look Gaussian (reject H0)'

print(msg_j)
```

```
JPM does not look Gaussian (reject H0)
```

Part 4

In [35]:
```
x = project_data["Close_ETF"]
```

In [36]:
```
x.describe()
```

Out[36]:
```
count    1000.000000
mean      121.152960
std        12.569790
min        96.419998
25%       112.580002
50%       120.150002
75%       128.687497
max       152.619995
Name: Close_ETF, dtype: float64
```

In [37]:
```
x.mean()
```

Out[37]:
```
121.1529600120001
```

In [38]:
```
x.std()
```

Out[38]:
```
12.569790313110744
```

50 groups of 20

In [39]:
```
project_data['n20bins']=pd.qcut(project_data['Close_ETF'], q=50)
project_data.head()
```

Out[39]:

	t_index	Close_ETF	oil	gold	JPM	n20bins
0	1	97.349998	0.039242	0.004668	0.032258	(96.419, 98.799]
1	2	97.750000	0.001953	-0.001366	-0.002948	(96.419, 98.799]
2	3	99.160004	-0.031514	-0.007937	0.025724	(98.799, 99.856]
3	4	99.650002	0.034552	0.014621	0.011819	(98.799, 99.856]
4	5	99.260002	0.013619	-0.011419	0.000855	(98.799, 99.856]

In [40]:
```
y = project_data.groupby('n20bins').mean()['Close_ETF']
y
```

Out[40]:
```
n20bins
(96.419, 98.799]       98.050000
(98.799, 99.856]       99.367501
(99.856, 100.769]     100.287500
(100.769, 103.322]    101.928001
(103.322, 104.597]    103.781500
(104.597, 105.972]    105.211500
(105.972, 107.884]    106.854500
(107.884, 109.645]    108.826001
(109.645, 110.208]    109.869999
(110.208, 111.068]    110.654999
(111.068, 111.548]    111.280000
```

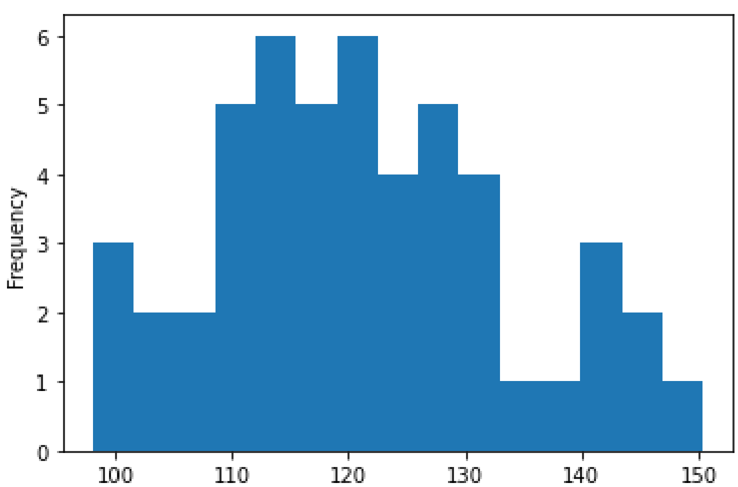
```
(111.548, 112.354]    111.890500
(112.354, 112.86]     112.628096
(112.86, 113.2]       113.030500
(113.2, 113.777]      113.451053
(113.777, 114.244]    113.944500
(114.244, 114.783]    114.498500
(114.783, 115.65]     115.122381
(115.65, 116.6]       116.137500
(116.6, 117.43]       117.152500
(117.43, 118.096]     117.767368
(118.096, 118.6]      118.329048
(118.6, 119.205]      118.914738
(119.205, 119.516]    119.355001
(119.516, 120.15]     119.902857
(120.15, 120.68]      120.412500
(120.68, 121.194]     120.957368
(121.194, 121.716]    121.414500
(121.716, 122.515]    122.244000
(122.515, 123.334]    122.912000
(123.334, 123.801]    123.563999
(123.801, 124.779]    124.277000
(124.779, 126.037]    125.519001
(126.037, 126.586]    126.273500
(126.586, 127.016]    126.762001
(127.016, 127.503]    127.281500
(127.503, 128.398]    128.032500
(128.398, 129.019]    128.686000
(129.019, 129.802]    129.460000
(129.802, 130.52]     130.198499
(130.52, 131.387]     130.893499
(131.387, 133.58]     132.313809
(133.58, 137.367]     135.891578
(137.367, 139.474]    138.411501
(139.474, 140.921]    140.317499
(140.921, 141.904]    141.466999
(141.904, 142.964]    142.362501
(142.964, 144.666]    143.815501
(144.666, 148.123]    146.003500
(148.123, 152.62]     150.375499
Name: Close_ETF, dtype: float64
```

In [41]:

```
y.plot.hist(bins=15)
```

Out[41]:

```
<AxesSubplot:ylabel='Frequency'>
```



from visual inspection not very normal

In [42]:
```
y.mean()
```

Out[42]: 121.16164592586216

In [43]:
```
y.std()
```

Out[43]: 12.686566498035301

In [44]:
```
x.mean()
```

Out[44]: 121.1529600120001

In [45]:
```
x.std()
```

Out[45]: 12.569790313110744

10 groups of 100

In [46]:
```
project_data['n100bins'] = pd.qcut(project_data['Close_ETF'], q=10)
project_data.head()
```

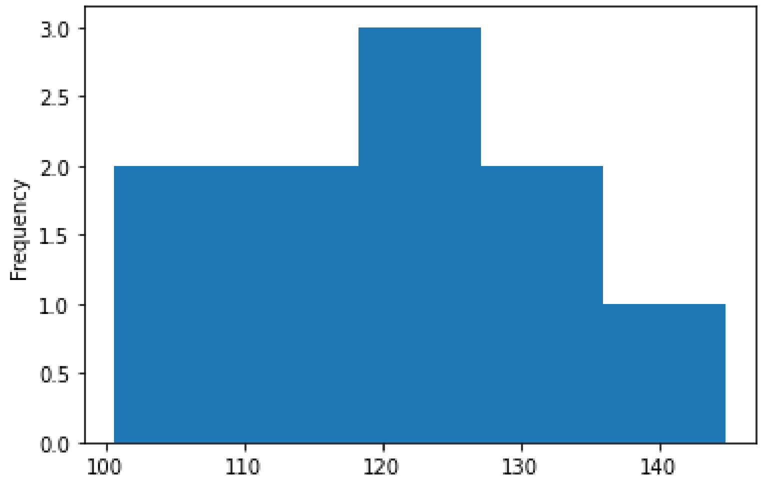
Out[46]:

	t_index	Close_ETF	oil	gold	JPM	n20bins	n100bins
0	1	97.349998	0.039242	0.004668	0.032258	(96.419, 98.799]	(96.419, 104.597]
1	2	97.750000	0.001953	-0.001366	-0.002948	(96.419, 98.799]	(96.419, 104.597]
2	3	99.160004	-0.031514	-0.007937	0.025724	(98.799, 99.856]	(96.419, 104.597]
3	4	99.650002	0.034552	0.014621	0.011819	(98.799, 99.856]	(96.419, 104.597]
4	5	99.260002	0.013619	-0.011419	0.000855	(98.799, 99.856]	(96.419, 104.597]

In [47]:
```
z=project_data.groupby('n100bins').mean()['Close_ETF']
```

In [48]:
```
z.plot.hist(bins=5)
```

Out[48]: <AxesSubplot:ylabel='Frequency'>



bit more normal but not by much

In [49]:
```
z.mean()
```

Out[49]: 121.15918584274345

In [50]:
```
z.std()
```

Out[50]: 13.080360155986197

In [51]:
```
x.std()
```

Out[51]: 12.569790313110744

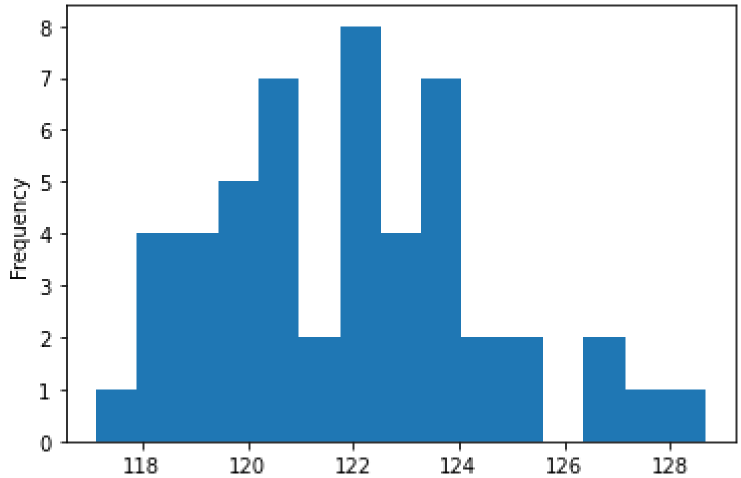
50 random samples (with replacement) groups of 20

In [52]:
```
randsamp20 = []
for i in range(50):
    randsamp20.append(pd.Series(project_data['Close_ETF'].sample(n=20, replace=T
```

In [53]:
```
randsamp20means = []
for i in range(50):
    randsamp20means.append(randsamp20[i].mean())
```

In [54]:
```
a = pd.Series(randsamp20means, dtype=float)
a.plot.hist(bins=15)
```

Out[54]: <AxesSubplot:ylabel='Frequency'>



Much more normal looking

In [55]:
```
a.mean()
```

Out[55]: 121.94334000300003

In [56]:
```
a.std()
```

Out[56]: 2.575016823332504

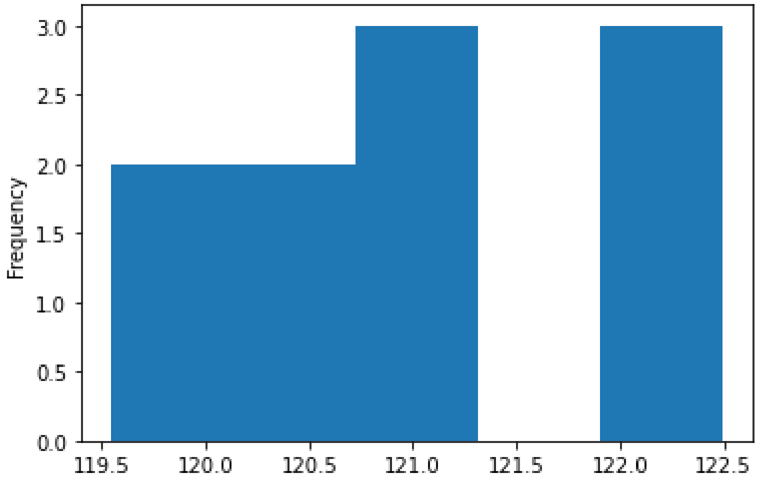
10 groups of 100

In [57]:
```
randsamp100 = []
for i in range(10):
    randsamp100.append(pd.Series(project_data['Close_ETF'].sample(n=100, replace
```

In [58]:
```
randsamp100means = []
for i in range(10):
    randsamp100means.append(randsamp100[i].mean())
```

In [59]:
```
b = pd.Series(randsamp100means, dtype=float)
b.plot.hist(bins=5)
```

Out[59]: <AxesSubplot:ylabel='Frequency'>



too few bins to judge normality

In [60]:
```
b.mean()
```

Out[60]: 121.01300993099998

In [61]:
```
b.std()
```

Out[61]: 1.0991791321617377

Part 5

In [62]:
```
import scipy.stats as st
```

In [63]:
```
sample_100 = pd.Series(x.sample(n=100, replace=True))
```

In [64]:
```
st.norm.interval(alpha=0.95, loc=sample_100.mean(), scale=sample_100.std())
```

Out[64]: (98.45737034928041, 148.1260296307196)

In [65]:
```
sample_50 = pd.Series(x.sample(n=20, replace=True))
```

In [66]:
```
st.t.interval(alpha=0.95, df = len(sample_50)-1, loc=sample_50.mean(), scale=sam
```

Out[66]: (102.6783334487439, 156.91566765125611)

In [67]:
```
pop_means = x.mean()
pop_means
```

Out[67]: 121.1529600120001

Part 6

In [68]:

```
from scipy import stats
```

In [69]:

```
#Use the same sample you picked up in Step1)of Part 5 to test H0: μ=100vs.Ha: μ
#What's your conclusion?

x = project_data["Close_ETF"]
sample_100 = pd.Series(x.sample(n=100, replace=True))
st.norm.interval(alpha=0.95, loc=sample_100.mean(), scale=sample_100.std())

print(sample_100.mean())
print()

mu = sample_100.mean()
std = sample_100.std()
n = 100
mu_0 = 100
S_x = std/np.sqrt(n)
print(S_x)

T = (mu - mu_0)/S_x
print("The values is",T)

pval = stats.t.sf(np.abs(T), n-1)*2
print("The p value is:",pval)

alpha_1 = 0.05

if pval>alpha_1:
    print("The test is failed to reject H0")
else:
    print("The test is reject H0")
```

```
120.76349952999995

1.3256164850999217
The values is 15.663278001883665
The p value is: 1.5069085351553728e-28
The test is reject H0
```

In [70]:

```
#Use the same sample you picked up in Step 2)of Part5 to testH0: μ=100vs. Ha: μ
#What's your conclusion?

x = project_data["Close_ETF"]
sample_50 = pd.Series(x.sample(n=50, replace=True))
st.t.interval(alpha=0.95, df = len(sample_50)-1, loc=sample_50.mean(), scale=sam
print(sample_50.mean())
print()

#ttest(mean_series_rand_50,100,.05)

mu = sample_100.mean()
std = sample_100.std()
n = 100
mu_0 = 100
S_x = std/np.sqrt(n)
print(S_x)
```

```
T = (mu - mu_0)/S_x
print("The values is",T)

pval = stats.t.sf(np.abs(T), n-1)*2
print("The p value is:",pval)

alpha_1 = 0.05

if pval>alpha_1:
    print("The test is failed to reject H0")
else:
    print("The test is reject H0")
```

```
120.89099996000003

1.3256164850999217
The values is 15.663278001883665
The p value is: 1.5069085351553728e-28
The test is reject H0
```

In [71]:

```
#Use the same sample you picked up in Step 2) of Part5 to test H0: σ=15 vs.Ha: σ
#What’s your conclusion?

st.t.interval(alpha=0.95, df = len(sample_50)-1, loc=sample_50.mean(), scale=sam
```

Out[71]: (95.1067511847364, 146.67524873526366)

In [72]:

```
#Use the same sample you picked up in Step2) of Part 5 to test H0: σ=15 vs.Ha: σ
#What’s your conclusion?
```

Part 7

In [73]:

```
#Consider the entire Gold column as a random sample from the first population,
#and the entire Oil column as a random sample from the second population.␣
#,→Assuming these two samples be
#drawn independently, form a hypothesis and test it to see if the Gold and Oil␣
#,→have equal means in the
#significance level 0.05.
```

In [74]:

```
significance_level = 0.05
gold_update = project_data['gold'].tolist()
oil_update = project_data['oil'].tolist()
```

In [75]:

```
t_test, p_value= stats.ttest_ind(gold_update, oil_update)
print("The p_value is: ", p_value)
```

```
The p_value is:  0.6274695292874639
```

In [76]:

```
if p_value<significance_level:
    print("The test is failed to reject H0")
else:
    print("The test is reject H0")
```

```
The test is reject H0
```

In [77]:

```
#Subtract the entire Gold column from the entire Oil column and generate a␣ ␣sa
#Consider this sample as a random sample from the target population of␣ ␣differ
#Form a hypothesis and test it to see if the Gold and Oil have equal means in␣ 
from scipy import stats
import scipy.stats as st
```

In [78]:

```
difference_gold_oil = (project_data['gold'] - project_data['oil'])
#print("The difference of the gold and oil is:",difference_gold_oil)
#diff_gold_oil = difference_gold_oil.tolist()
#print(difference_gold_oil)

sample_100_gold_oil = pd.Series(difference_gold_oil.sample(n=100, replace=True))
st.norm.interval(alpha=0.95, loc=sample_100_gold_oil.mean(), scale=sample_100_go

print("The sample of the mean is:",sample_100_gold_oil.mean())
print()

mu_diff = 0
std_diff= sample_100_gold_oil.std()
n_diff = 100
mu_0_diff = 100
S_x_diff = std_diff/np.sqrt(n_diff)
print("The result is:",S_x_diff)

T_test_diff = (mu_diff - mu_0_diff)/S_x_diff
print("The value is:",T_test_diff)

pval_diff = stats.t.sf(np.abs(T_test_diff), n_diff-1)*2
print("The p value is:",pval_diff)


significance_level = 0.05


t_test, p_value= stats.ttest_ind(difference_gold_oil,sample_100_gold_oil)
print("The p_value is: ", p_value)
if pval_diff>significance_level:
    print("The test is failed to reject H0")
else:
    print("The test is reject H0")
```

```
The sample of the mean is: 0.00395821875

The result is: 0.002021635028152294
The value is: -49464.91261154919
The p value is: 0.0
The p_value is:  0.0535698997541753
The test is reject H0
```

In [79]:

```
#Consider the entire Gold column as a random sample from the first population,
#and the entire Oil column as a random sample from the second population.
#Assuming these two samples be drawn independently, form a hypothesis and
#test it to see if the Gold and Oil have equal standard deviations in the signif

import scipy
significance_level = 0.05
gold_new_update = project_data['gold']
```

```
oil_new_update = project_data['oil']

sample_100_gold = pd.Series(gold_new_update.sample(n=100, replace=True))
st.norm.interval(alpha=0.95, loc=sample_100_gold.mean(), scale=sample_100_gold.s
sample_100_oil = pd.Series(oil_new_update.sample(n=100, replace=True))
st.norm.interval(alpha=0.95, loc=sample_100_oil.mean(), scale=sample_100_oil.std

print("The gold sample of the mean is:",sample_100_gold.mean())
print()

print("The oil sample of the mean is:",sample_100_oil.mean())
print()

f = np.var(project_data['gold']) / np.var(project_data['oil'])
n_oil = 50
n_gold = 50
result = 1-scipy.stats.f.cdf(f, n_oil - 1, n_gold -1)

print("The result is: ",result)

if p_value>significance_level:
    print("The test is failed to reject H0")
else:
    print("The test is reject H0")
```

```
The gold sample of the mean is: 0.00021816791000000017

The oil sample of the mean is: -0.003326272459999997

The result is:  0.999987979230873
The test is failed to reject H0
```

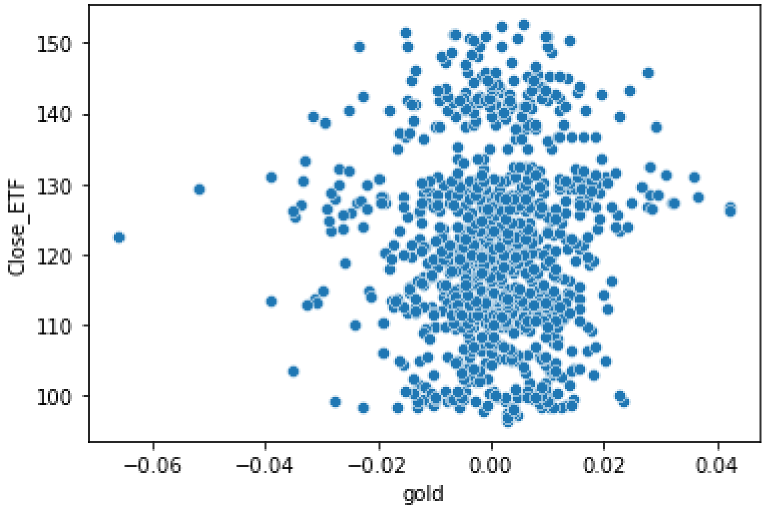
Part 8

In [80]:

```
from scipy.stats import pearsonr
```

In [81]:

```
#Draw a scatter plot of ETF (Y) vs. Gold (X).
#Is there any linear relationship between them which can be observed from the sc

scatter_ETF_gold = sns.scatterplot(data=project_data, x="gold", y="Close_ETF")
x_gold = project_data['gold']
y_close_etf = project_data['Close_ETF']
```

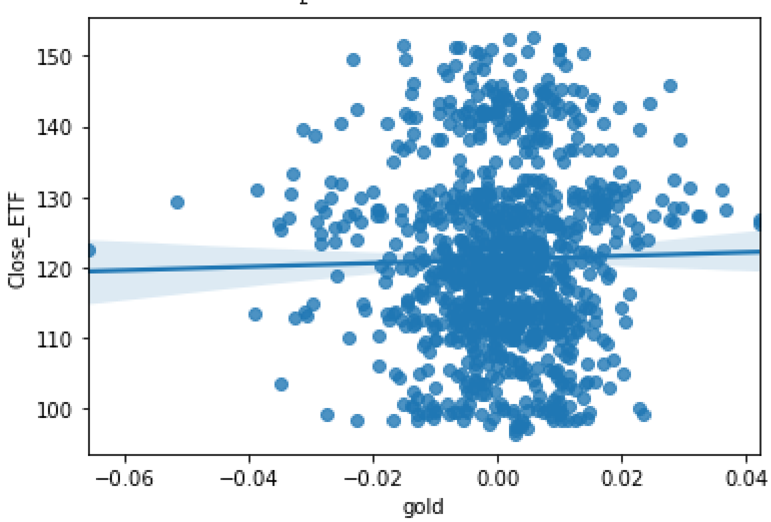


In [82]:
```
#Calculate the coefficient of correlation between ETF and Gold and interpret it.

corr, _ = pearsonr(x_gold,y_close_etf)
print('Pearsons correlation:' , corr)
```

```
Pearsons correlation: 0.022995570076054597
```

In [83]:
```
#Fit a regression line (or least squares line, best fitting line) to the scatter
#What are the intercept and slope of this line? How to interpret them?

regression_line_scatter = sns.regplot(x=x_gold, y=y_close_etf, data=project_data

slope, intercept, r_value, p_value, std_err = stats.linregress(x_gold,y_close_et

print('What is the slope:',slope)
print('What is the intercept:',intercept)
```

```
What is the slope: 25.604389324427277
What is the intercept: 121.13598849889819
```



In [84]:
```
#Conduct a two-tailed t-test with 〖H〗_0:  β_1=0.
#What is the P-value of the test?
#Is the linear relationship between ETF (Y) and Gold (X) significant at the sign
#Why or why not?
```

```
print('What is the p-value:',p_value)

#Suppose that you use the coefficient of determination to assess the quality of
#Is it a good model? Why or why not?


#What are the assumptions you made for this model fitting?


#Given the daily relative change in the gold price is 0.005127.
#Calculate the 99% confidence interval of the mean daily ETF return, and the 99%
#the individual daily ETF return.

st.t.interval(alpha=0.99, df=len(project_data['Close_ETF'])-1, loc=np.mean(proje
#st.norm.interval(alpha=0.99, loc=np.mean(project_data['Close_ETF']), scale=st.s
```

```
What is the p-value: 0.467611780618294
```

Out[84]:

```
(120.12712955132933, 122.17879047267085)
```

Part 9

In [85]:

```
#Consider the data including the ETF, Gold and Oil column.
#Using any software, fit a multiple linear regression model to the data with the
#Evaluate your model with adjusted R^2.

import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model



project_data = pd.read_csv('data.csv')
project_data.head()
```

Out[85]:

	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

In [86]:

```
X = project_data[['oil','gold']]
y = project_data['Close_ETF']

regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Out[86]:

```
LinearRegression()
```

In [91]:
```
regr.coef_
```

Out[91]:
```
array([-9.12610011, 29.62259192])
```

In [88]:
```
R_2 = 1 - (1-regr.score(X, y))*(len(y)-1)/(len(y)-X.shape[1]-1)
print('The value of R^2: ',R_2)
```

```
The value of R^2:  -0.0012542162682833702
```

Part 10

Check residuals

In [111…
```
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.nonparametric.smoothers_lowess import lowess
```

In [112…
```
model = ols('Close_ETF ~ oil + gold', data=project_data).fit()
model.summary()
```

Out[112…

OLS Regression Results

Dep. Variable:	Close_ETF	**R-squared:**	0.001
Model:	OLS	**Adj. R-squared:**	-0.001
Method:	Least Squares	**F-statistic:**	0.3743
Date:	Thu, 19 Aug 2021	**Prob (F-statistic):**	0.688
Time:	17:34:23	**Log-Likelihood:**	-3949.4
No. Observations:	1000	**AIC:**	7905.
Df Residuals:	997	**BIC:**	7919.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	121.1427	0.399	303.856	0.000	120.360	121.925
oil	-9.1261	19.413	-0.470	0.638	-47.221	28.968
gold	29.6226	36.272	0.817	0.414	-41.555	100.800

Omnibus:	26.565	**Durbin-Watson:**	0.005
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	22.981
Skew:	0.306	**Prob(JB):**	1.02e-05
Kurtosis:	2.579	**Cond. No.**	92.2

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Check the four assumptions

In [113…

```
project_data.head()
```

Out[113…

	t_index	Close_ETF	oil	gold	JPM
0	1	97.349998	0.039242	0.004668	0.032258
1	2	97.750000	0.001953	-0.001366	-0.002948
2	3	99.160004	-0.031514	-0.007937	0.025724
3	4	99.650002	0.034552	0.014621	0.011819
4	5	99.260002	0.013619	-0.011419	0.000855

Mean 0 assumption

In [116…

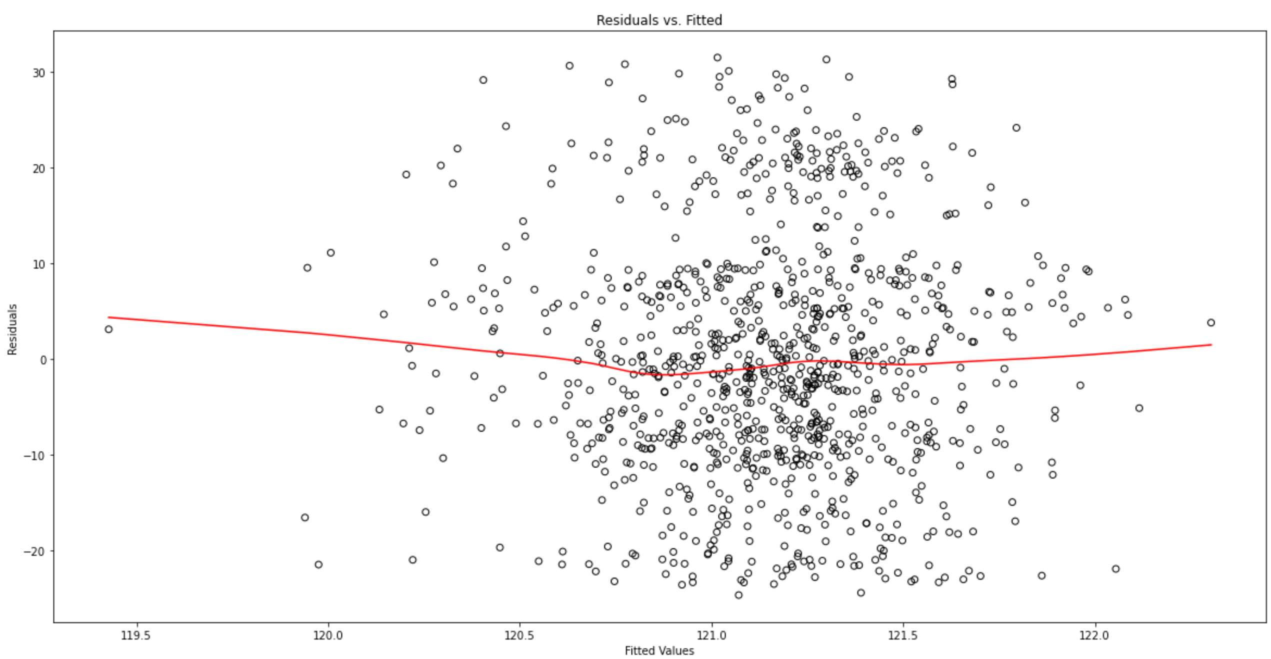
```
### plot residuals vs predictors
residuals = model.resid
fitted = model.fittedvalues
smoothed = lowess(residuals,fitted)
```

In [126…

```
fig, ax = plt.subplots(figsize=(20,10))
ax.scatter(fitted, residuals, edgecolors = 'k', facecolors = 'none')
ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')
ax.set_ylabel('Residuals')
ax.set_xlabel('Fitted Values')
ax.set_title('Residuals vs. Fitted')
```

Out[126…

```
Text(0.5, 1.0, 'Residuals vs. Fitted')
```



Because there is no pattern in the residuals plotted, but there is a small u-shape to the smoothed fitted data lane, we can say that there is not indication of non-linearity in the model data.

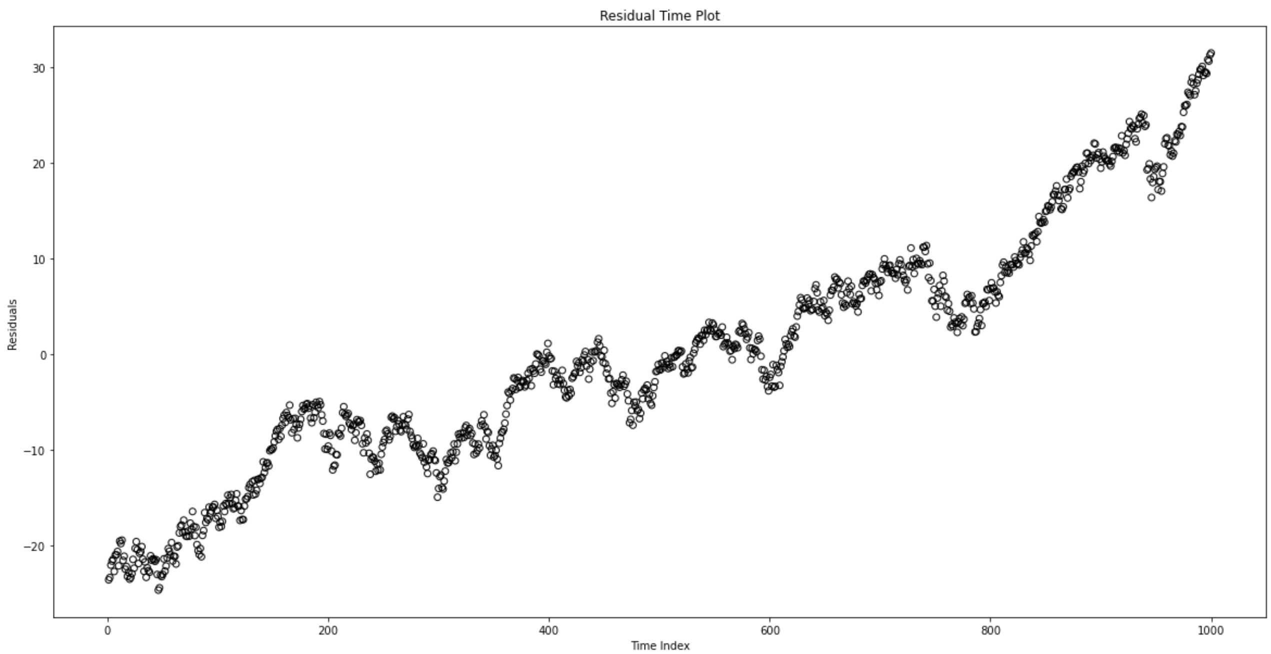
Independence assumption

In [135…

```
### plot residuals vs row number

fig2, ax2 = plt.subplots(figsize=(20,10))
ax2.scatter(project_data['t_index'], residuals, edgecolors = 'k', facecolors = '
ax2.set_ylabel('Residuals')
ax2.set_xlabel('Time Index')
ax2.set_title('Residual Time Plot')
```

Out[135…

```
Text(0.5, 1.0, 'Residual Time Plot')
```

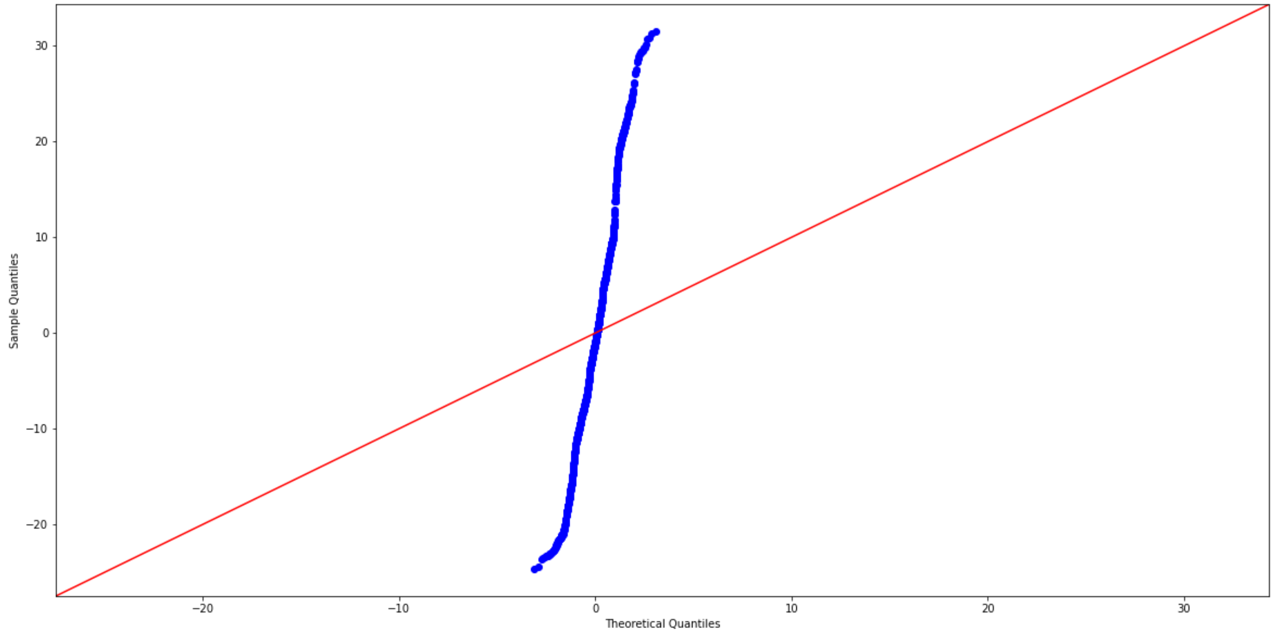


The residuals are showing a relationship over time, that is the variance is NOT consistent with time, so the model fails the test of independence.

Normality assumption

In [136…

```
from statsmodels.graphics.gofplots import qqplot

fig3, ax3 = plt.subplots(figsize=(20,10))
plot3 = qqplot(residuals, line="45", ax=ax3)
```



The model fails the test of normality because of the S-shape in the normality plot.

Variance assumption

Because the model fails the normality test and indepence, we can say that model is heteroscedastic and would fail the test of constant variance.

Discuss how you may improve the quality of your regression model according to the strategy of model selection.

There is a generally postive trend upward of the residuals with time, we can say with confidence that including time as a regressor will improve the model.