Part 1: Meet the data

Data description – This data includes four columns/random variables: the daily ETF return; the daily relative change in the price of the crude oil; the daily relative change in the gold price; and the daily return of the JPMorgan Chase & Co stock. The sample size is 1000. Requirements – Use any software to obtain the sample mean and sample standard deviation for each random variable (column) of the data; the sample correlations among each pair of the four random variables (columns) of the data.

In [1]:
```
# importing pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
# read an excel file and convert
# into a dataframe object
df = pd.DataFrame(pd.read_excel("/Users/yashadmuthe/Desktop/541/Data.xlsx"))

# show the dataframe
df
```

Out[1]:

	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 × 4 columns

In [2]:
```
df.describe()
```

Out[2]:

	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

	Close_ETF	oil	gold	JPM
max	152.619995	0.087726	0.042199	0.057480

In [3]:
```
# Mean and standard deviation for
# ETF = 121.152960 , 12.569790
# Oil = 0.001030 , 0.021093
# Gold = 0.000663 , 0.011289
# JPM = 0.000530 , 0.011017
```

In [4]:
```
# Correlation between among each pair of 4 random variable
df.corr()
```

Out[4]:

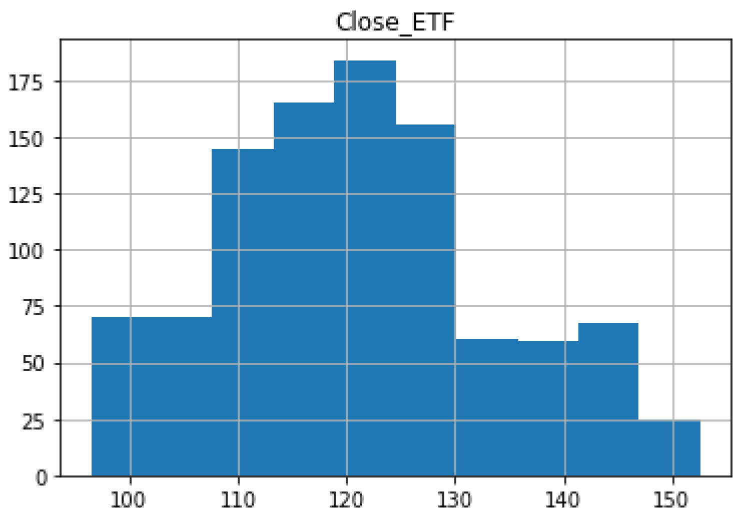
	Close_ETF	oil	gold	JPM
Close_ETF	1.000000	-0.009045	0.022996	0.036807
oil	-0.009045	1.000000	0.235650	-0.120849
gold	0.022996	0.235650	1.000000	0.100170
JPM	0.036807	-0.120849	0.100170	1.000000

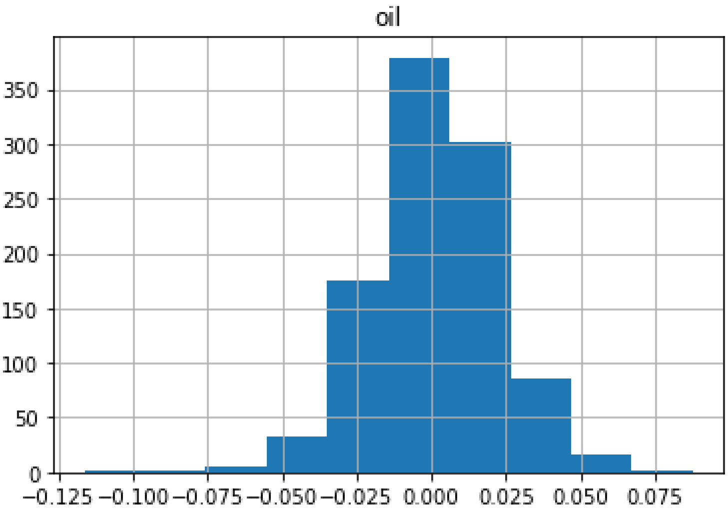
Part 2: Describe your data

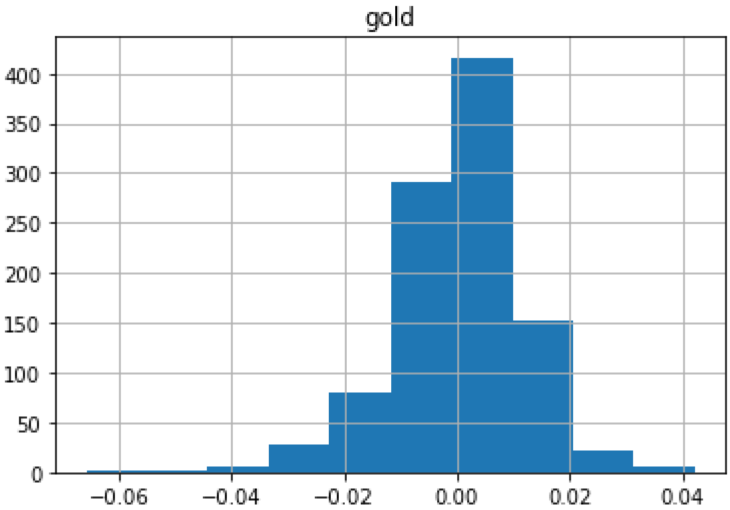
a) A histogram for each column (hint: four histograms total) b) A time series plot for each column (hint: use the series “1, 2, 3, ..., 1000” as the horizontal axis; four plots total) c) A time series plot for all four columns (hint: one plot including four “curves” and each “curve” describes one column) d) Three scatter plots to describe the relationships between the ETF column and the OIL column; between the ETF column and the GOLD column; between the ETF column and the JPM column, respectively

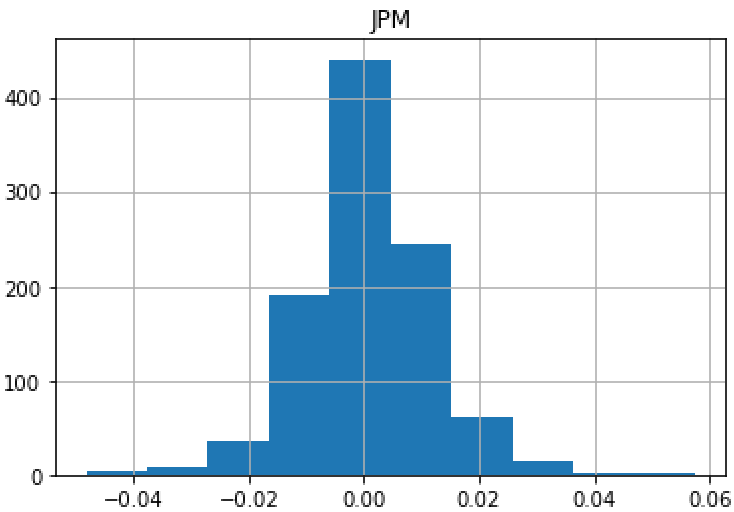
In [5]:
```
#a) Hist for each columns

df.hist('Close_ETF')
df.hist('oil')
df.hist('gold')
df.hist('JPM')
```

Out[5]:
```
array([[<AxesSubplot:title={'center':'JPM'}>]], dtype=object)
```







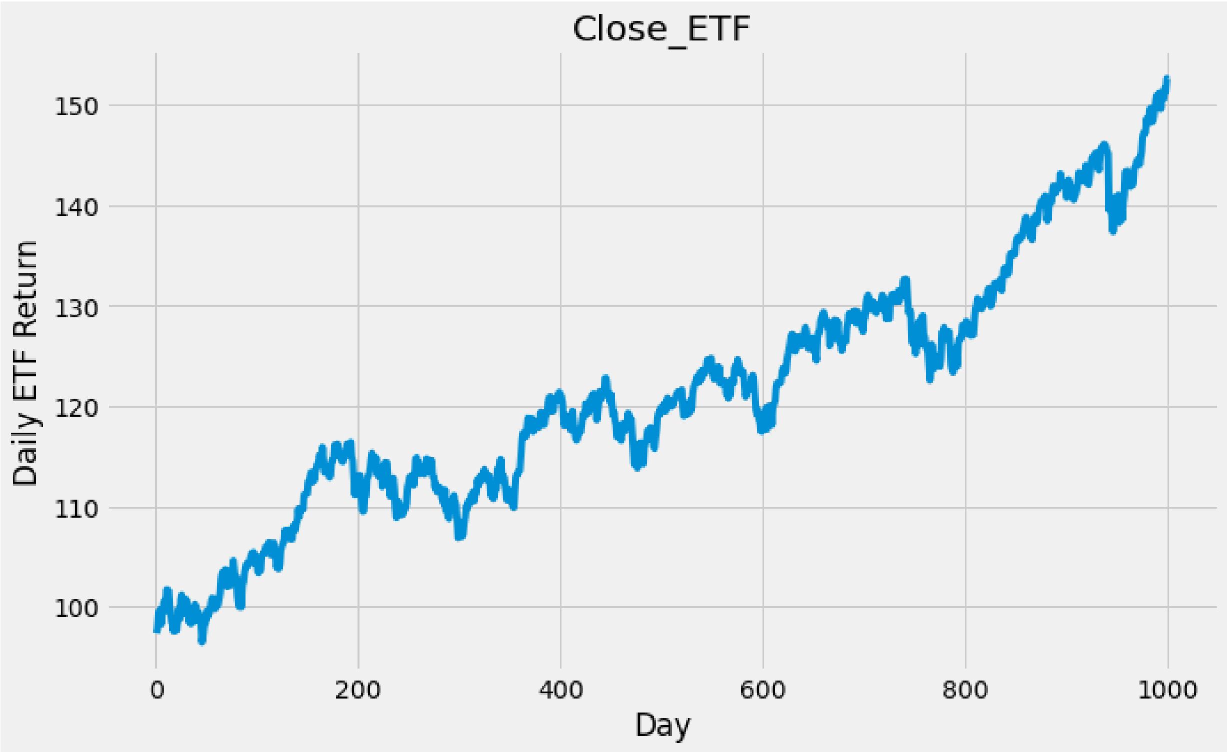


In [6]:

```
#b) A time series plot for each column

# creating series of number from 1 to 1000
x_col = np.linspace(1,1000,1000)


plt.style.use("fivethirtyeight")

plt.figure(figsize=(10, 6))

# Labelling the axes and setting
# a title
plt.xlabel("Day")
plt.ylabel("Daily ETF Return")
```

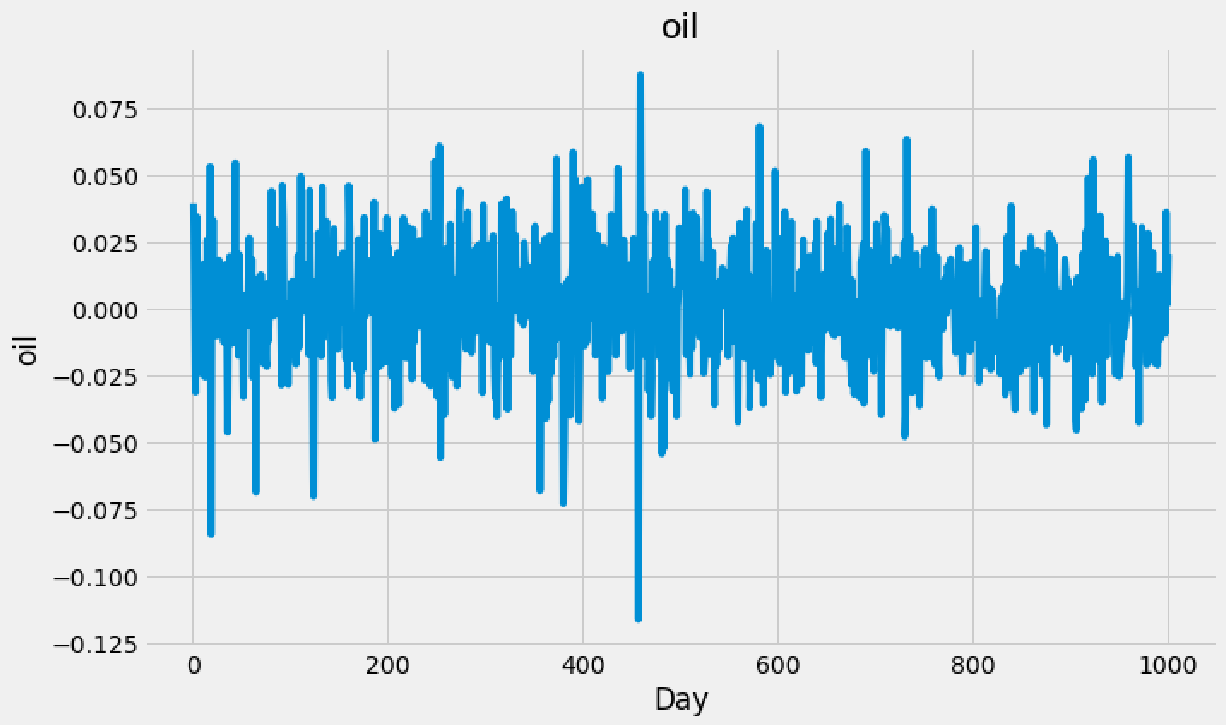
```
plt.title("Close_ETF")

plt.plot(x_col,df['Close_ETF'])
```

Out[6]: [<matplotlib.lines.Line2D at 0x7fdce832af70>]



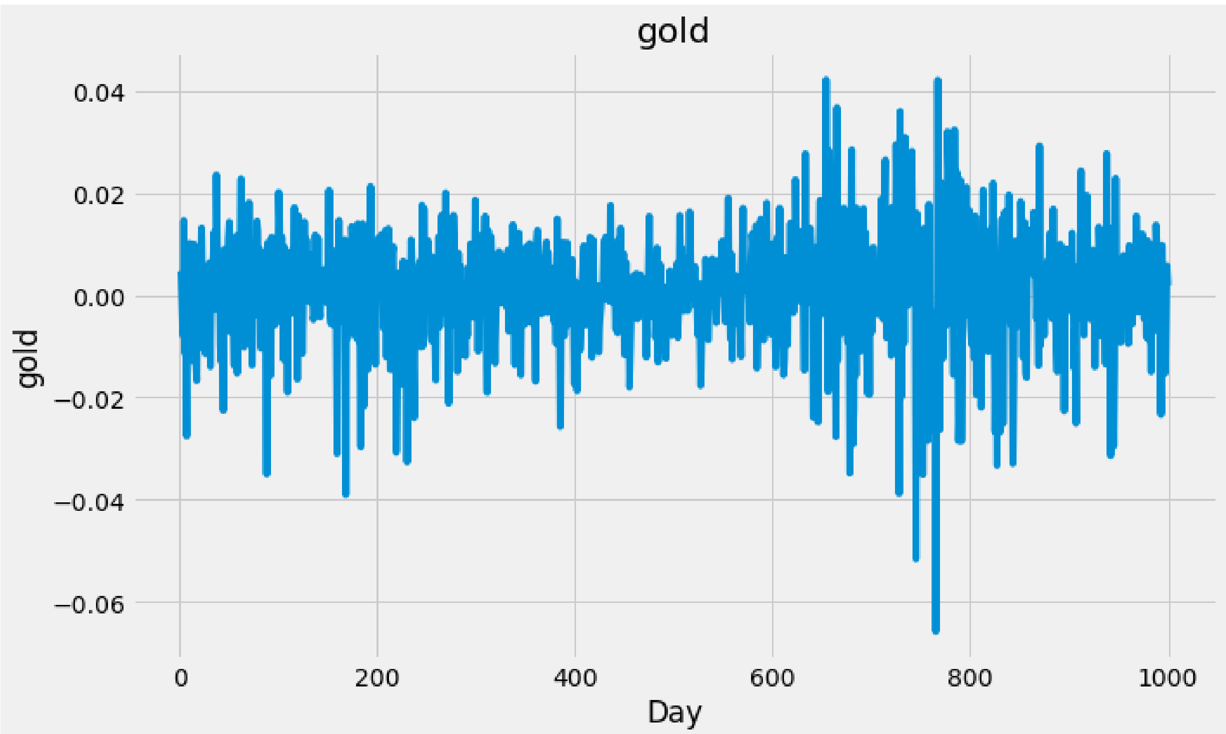
In [7]:
```
plt.style.use("fivethirtyeight")

plt.figure(figsize=(10, 6))

# Labelling the axes and setting
# a title
plt.xlabel("Day")
plt.ylabel("oil")
plt.title("oil")

plt.plot(x_col,df['oil'])
```

Out[7]: [<matplotlib.lines.Line2D at 0x7fdcd819a640>]



In [8]:
```
plt.style.use("fivethirtyeight")

plt.figure(figsize=(10, 6))

# Labelling the axes and setting
# a title
plt.xlabel("Day")
plt.ylabel("gold")
plt.title("gold")

plt.plot(x_col,df['gold'])
```

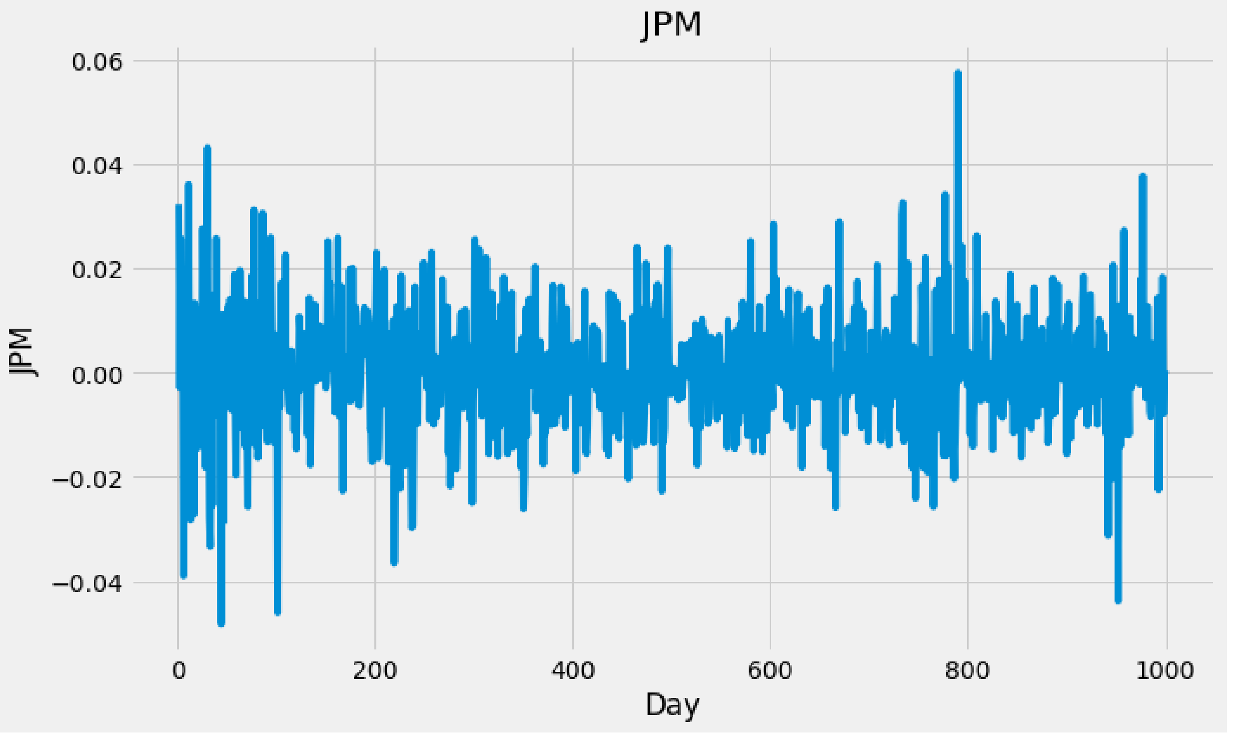
Out[8]: `[<matplotlib.lines.Line2D at 0x7fdcf8cbfdc0>]`



In [9]:
```
plt.style.use("fivethirtyeight")

plt.figure(figsize=(10, 6))
```

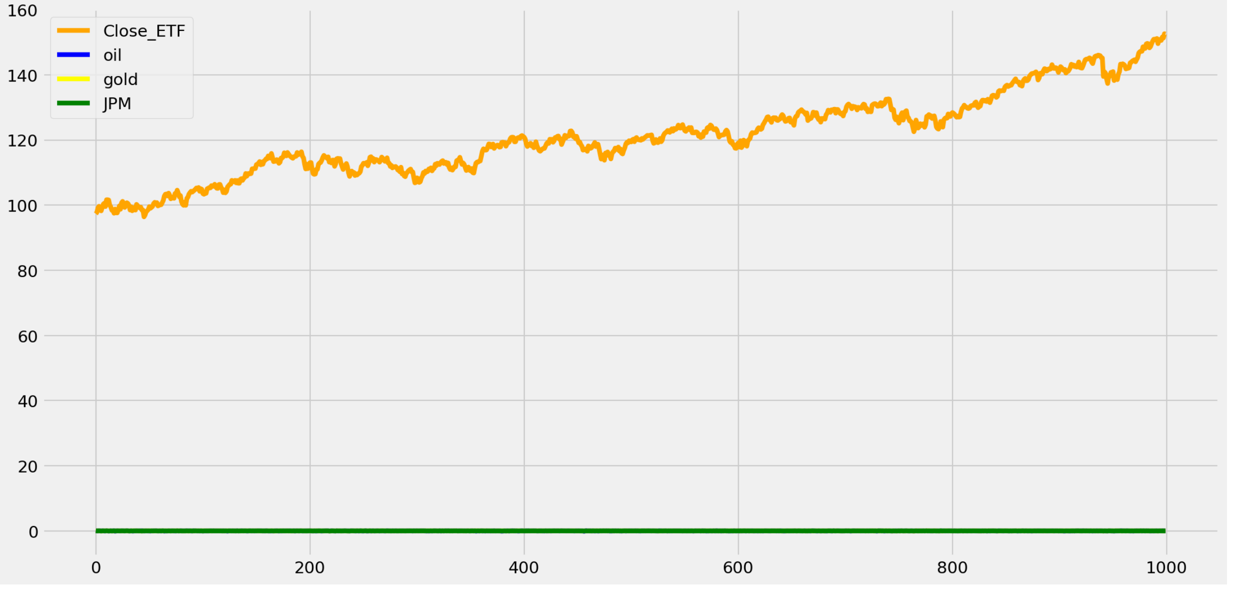
```
# Labelling the axes and setting
# a title
plt.xlabel("Day")
plt.ylabel("JPM")
plt.title("JPM")

plt.plot(x_col,df['JPM'])
```

Out[9]: [<matplotlib.lines.Line2D at 0x7fdcd8303760>]



In [10]:
```
#c) A time series plot for all four columns
plt.figure(figsize=(16, 8), dpi=150)
df['Close_ETF'].plot(label='Close_ETF', color='orange')
df['oil'].plot(label= 'oil', color='blue')
df['gold'].plot(label= 'gold', color='yellow')
df['JPM'].plot(label= 'JPM', color='green')
plt.legend()
```

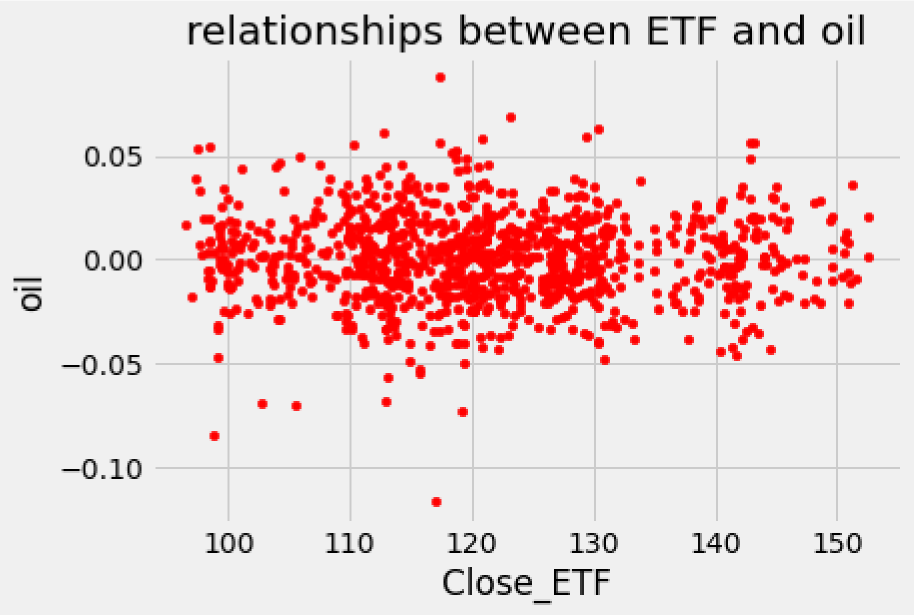
Out[10]: <matplotlib.legend.Legend at 0x7fdcf93ba220>



In [11]:
```
""""d)Three scatter plots to describe the relationships between the ETF colum
```

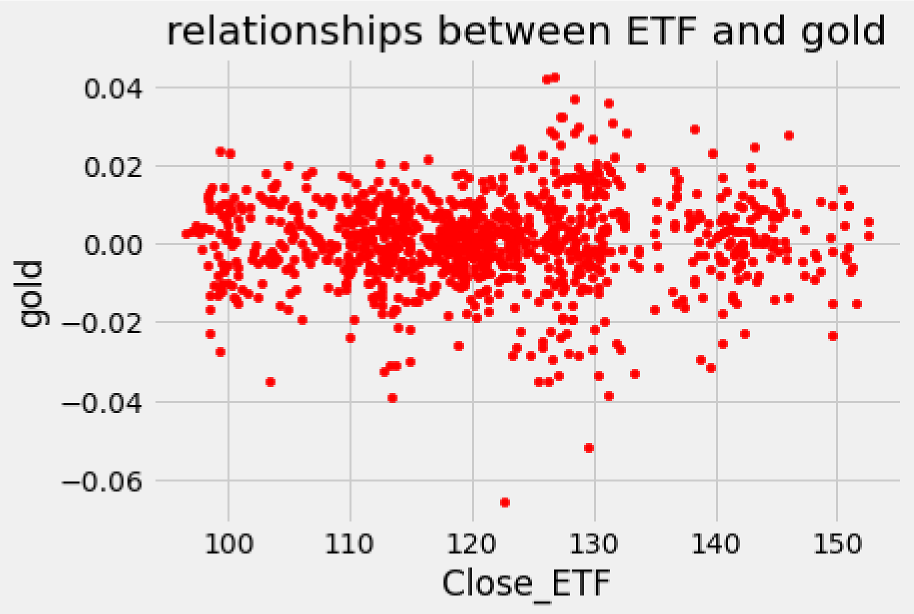
```
column; between the ETF column and the GOLD column; between the ETF column an
the JPM column, respectively
plt.figure(figsize=(10, 6))"""

df.plot.scatter(x='Close_ETF',
                y='oil',
                c='red')
plt.title('relationships between ETF and oil')
```

Out[11]: Text(0.5, 1.0, 'relationships between ETF and oil')



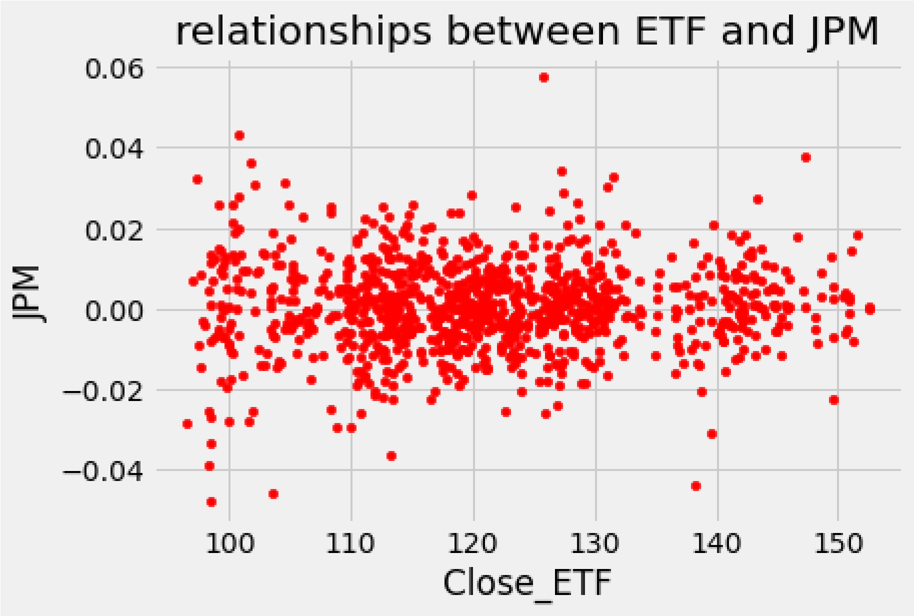
In [12]:
```
df.plot.scatter(x='Close_ETF',
                y='gold',
                c='red')
plt.title('relationships between ETF and gold')
```

Out[12]: Text(0.5, 1.0, 'relationships between ETF and gold')



In [13]:
```
df.plot.scatter(x='Close_ETF',
                y='JPM',
                c='red')
plt.title('relationships between ETF and JPM')
```

Out[13]: Text(0.5, 1.0, 'relationships between ETF and JPM')



Part 3: What distribution does your data follo

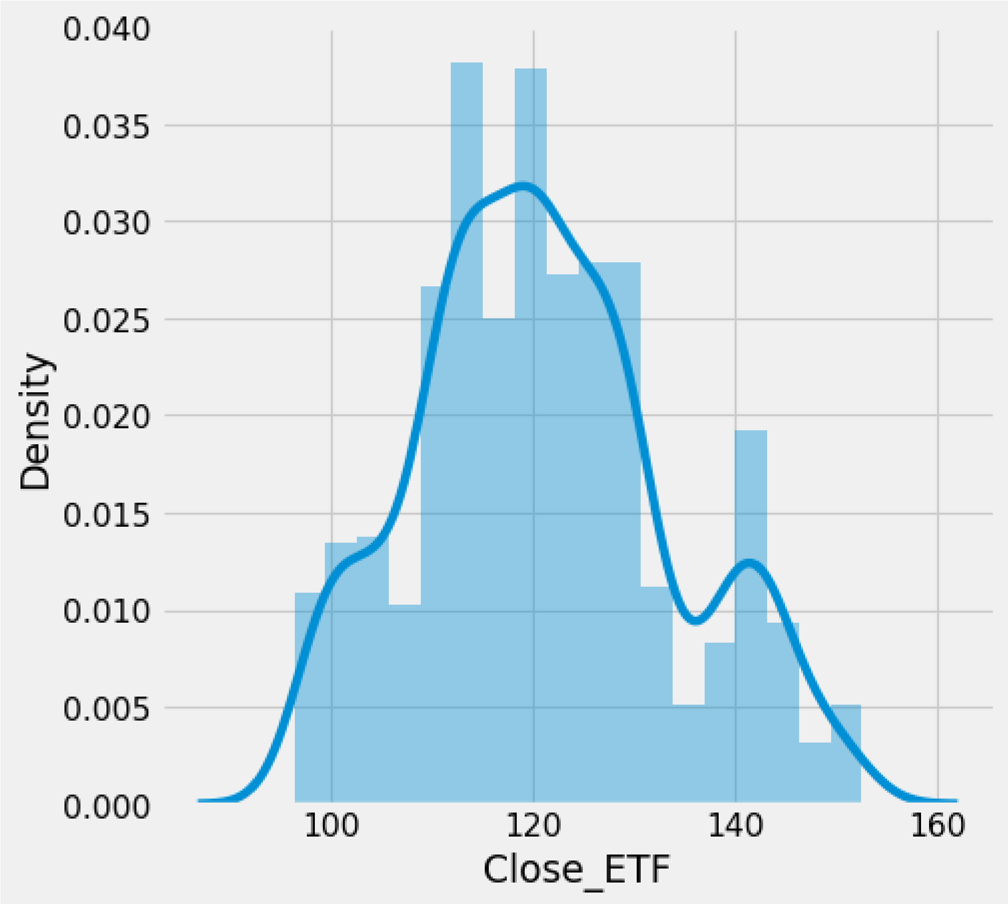
Propose an assumption/a hypothesis regarding the type of distribution each column of the data set may follow (i.e., the ETF, OIL, GOLD, and JPM column), based on the plots from Part 2. Then verify or object that assumption/hypothesis with appropriate tests (for example, normality test). You may use any software to perform those tests.

In [14]:
```
import numpy as np
import scipy.stats as stats
import seaborn as sns
from scipy.stats import chisquare
from pylab import*
```

In [15]:
```
print("                    Normality test for ETF             ")
print("                       ETF Histogram                   ")

plt.figure(figsize =(6,6), dpi=80)
sns.distplot(df['Close_ETF'], hist=True, kde=True)
```

```
                    Normality test for ETF
                       ETF Histogram
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```

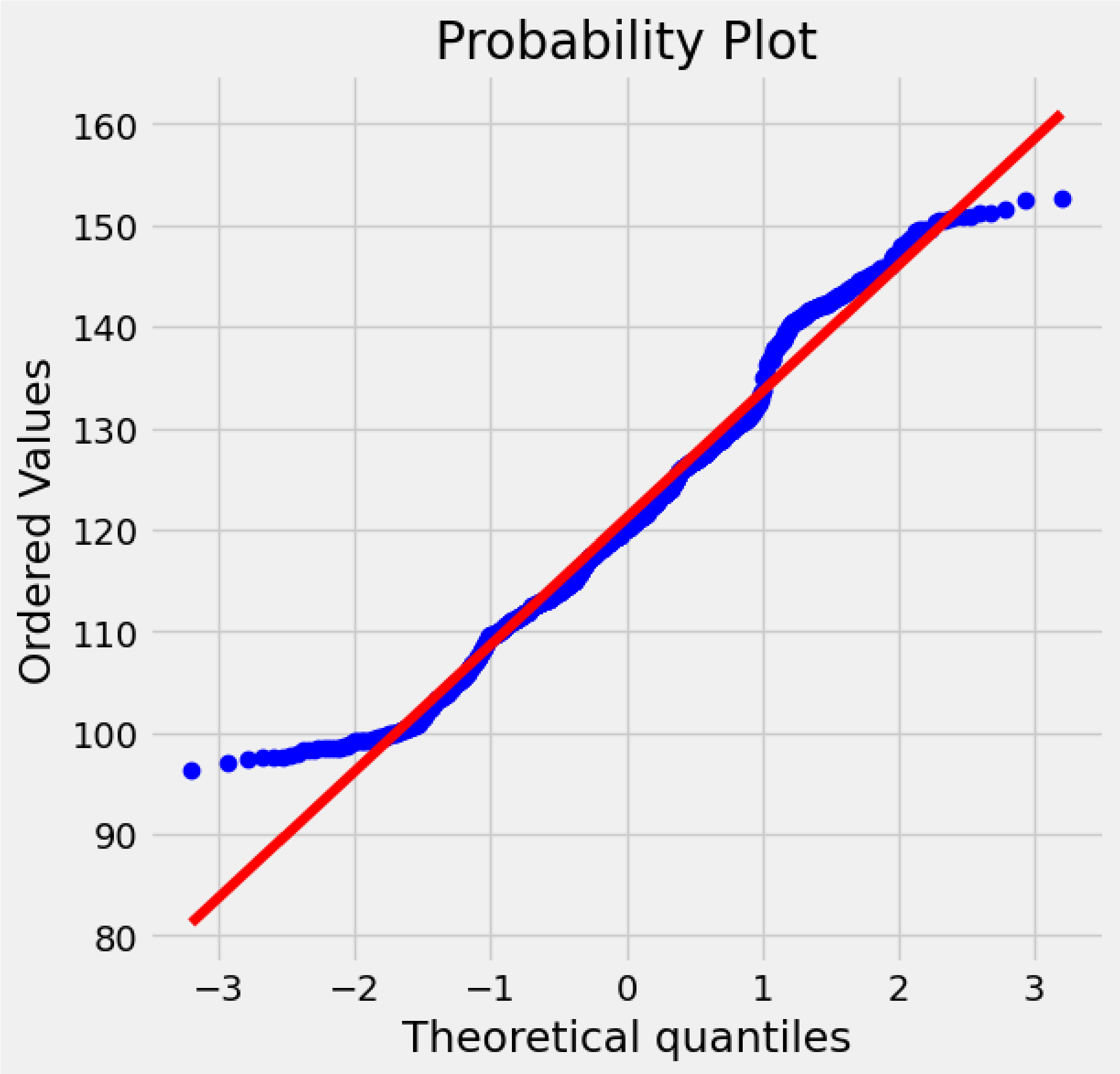
Out[15]:
```
<AxesSubplot:xlabel='Close_ETF', ylabel='Density'>
```



In [16]:

```
print("                                    QQ Plot                                    ")
plt.figure(figsize=(6,6), dpi=100)
stats.probplot(df["Close_ETF"],dist="norm",plot=plt)
plt.show()
```

```
                                    QQ Plot
```



In [17]:
```
print("    Chisquare   ")
stat , p1 = stats.chisquare(df["Close_ETF"])
print('Statistics=%.3f, p=%.3f' % (stat,p1))
alpha = 0.05
if p1 > alpha :
    print( 'Sample looks Gaussian ( fail to rejectH0)' )
else :
    print('Sample does not look Gaussian ( rejectH0)')
```

```
    Chisquare
Statistics=1302.829, p=0.000
Sample does not look Gaussian ( rejectH0)
```

In [18]:
```
print( '------------------ Shapiro-Wilk ------------------------' )
stat , p2 = stats.shapiro(df["Close_ETF"])
print('Statistics=%.3f , p=%.3f ' % ( stat , p2))
alpha = 0.05
if p2 > alpha :
    print('Sample looks Gaussian (failed to reject H0)')
else :
    print('Sample does not look Gaussian ( reject H0)')
```

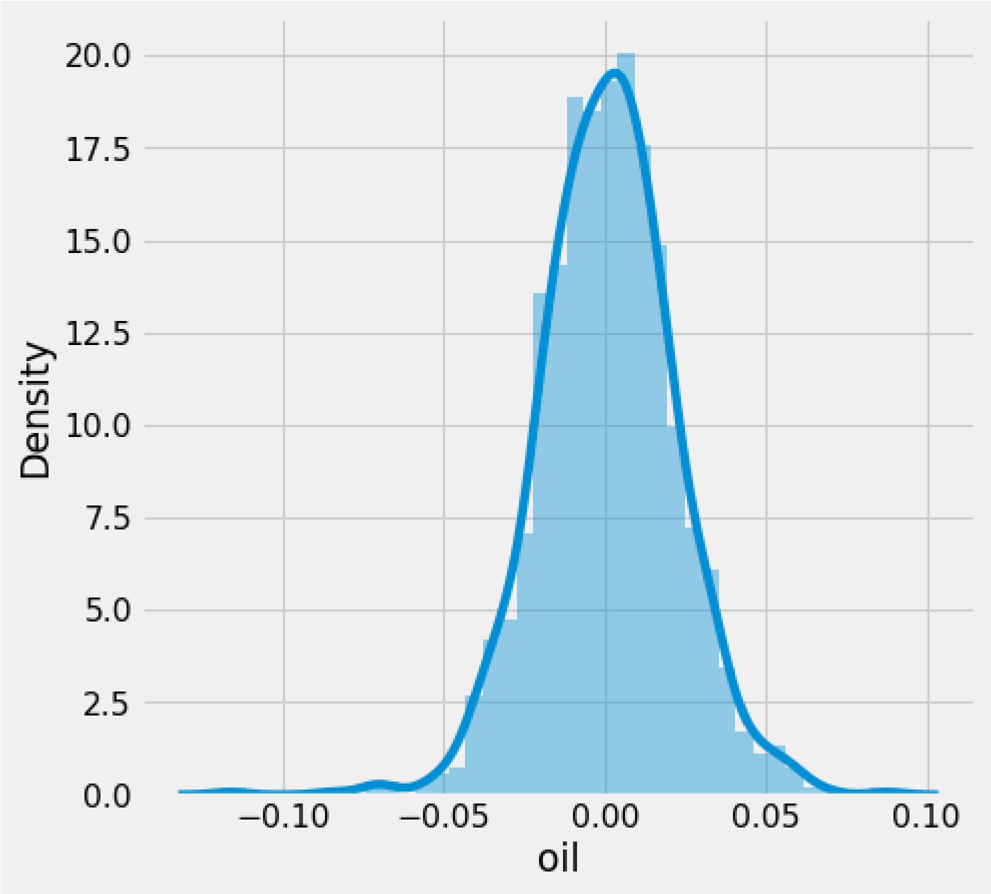
```
------------------ Shapiro-Wilk ------------------------
Statistics=0.980 , p=0.000
Sample does not look Gaussian ( reject H0)
```

In [19]:
```
print("                    Normality test for oil             ")
print("                       Oil Histogram                   ")
```

```
plt.figure(figsize =(6,6), dpi=80)
sns.distplot(df['oil'], hist=True, kde=True)
```

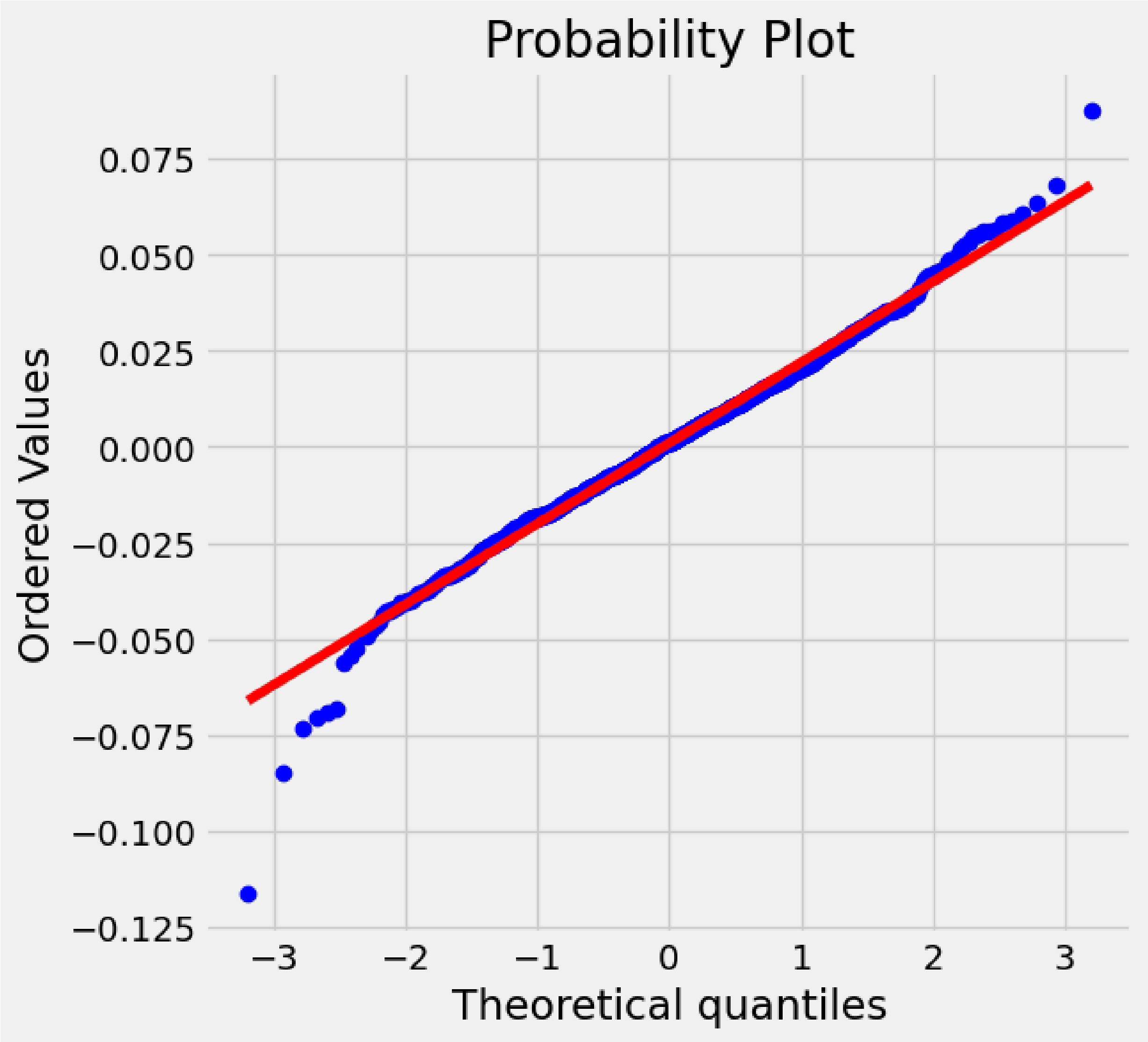
```
                     Normality test for oil
                        Oil Histogram
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]: `<AxesSubplot:xlabel='oil', ylabel='Density'>`



In [20]:
```
print("                      QQ Plot                    ")
plt.figure(figsize=(6,6), dpi=100)
stats.probplot(df["oil"],dist="norm",plot=plt)
plt.show()
```

```
                      QQ Plot
```



In [21]:

```
print("    Chisquare    ")
stat , p1 = stats.chisquare(df["oil"])
print('Statistics=%.3f, p=%.3f' % (stat,p1))
alpha = 0.05
if p1 > alpha :
    print( 'Sample looks Gaussian ( fail to rejectH0)' )
else :
    print('Sample does not look Gaussian ( rejectH0)')
```

```
    Chisquare
Statistics=431.505, p=1.000
Sample looks Gaussian ( fail to rejectH0)
```

In [22]:

```
print( '------------------ Shapiro-Wilk ------------------------' )
stat , p2 = stats.shapiro(df["oil"])
print('Statistics=%.3f , p=%.3f ' % ( stat , p2))
alpha = 0.05
if p2 > alpha :
    print('Sample looks Gaussian (failed to reject H0)')
else :
    print('Sample does not look Gaussian ( reject H0)')
```

```
------------------ Shapiro-Wilk ------------------------
Statistics=0.989 , p=0.000
Sample does not look Gaussian ( reject H0)
```

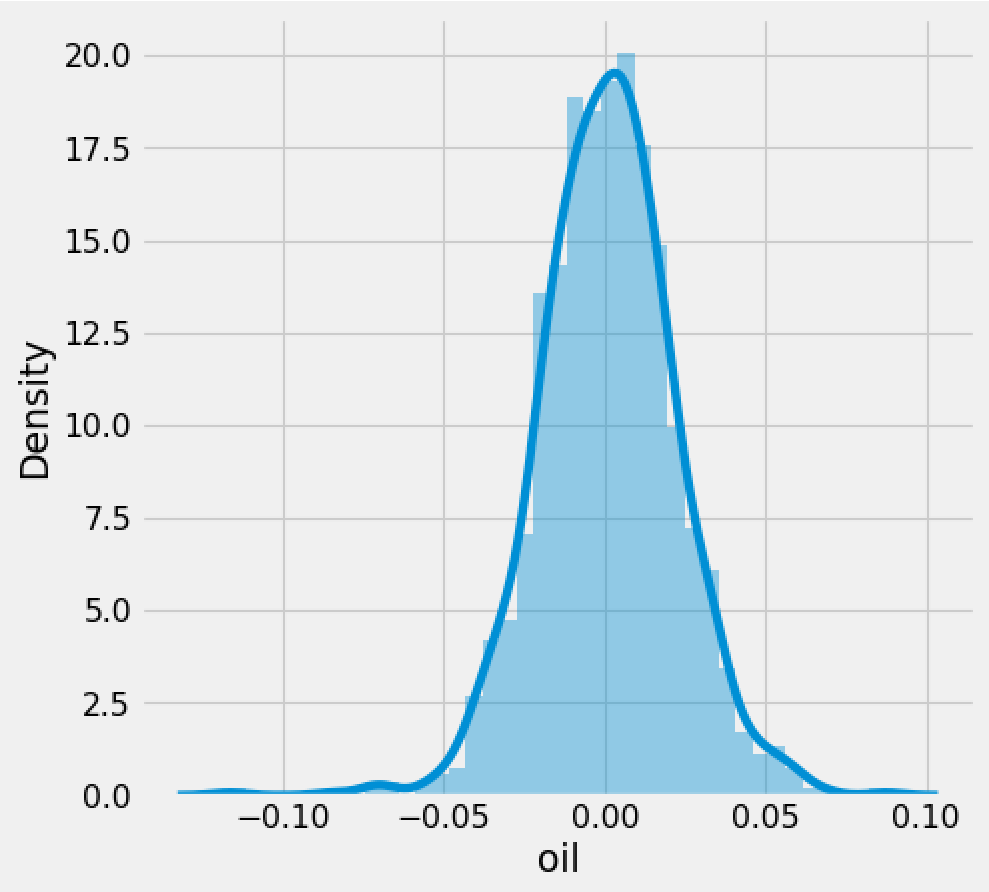
In [23]:

```
print("                    Normality test for gold              ")
print("                       gold Histogram                    ")
```

```
plt.figure(figsize =(6,6), dpi=80)
sns.distplot(df['oil'], hist=True, kde=True)
```

```
                  Normality test for gold
                     gold Histogram
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```

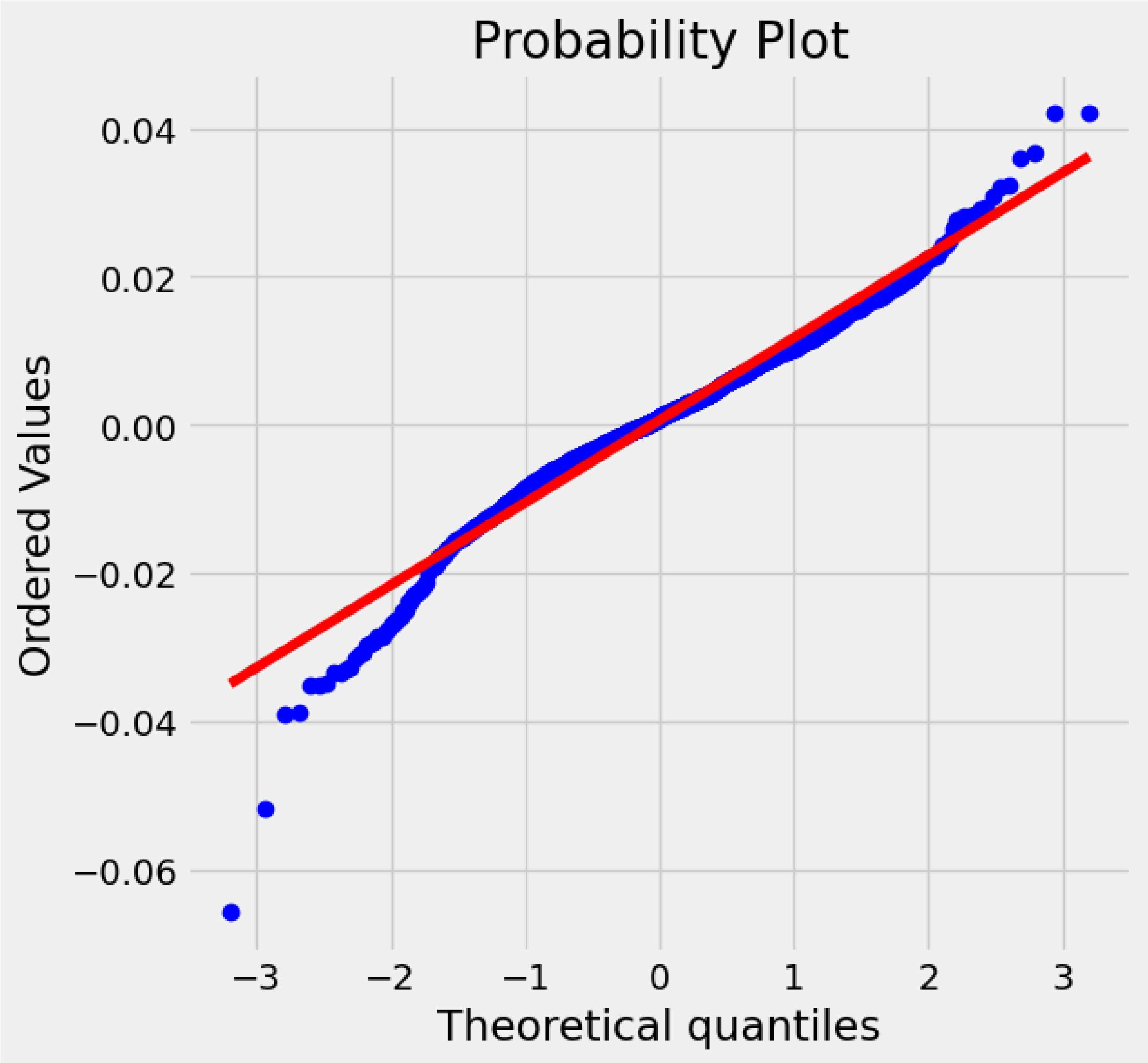
Out[23]: <AxesSubplot:xlabel='oil', ylabel='Density'>



In [24]:

```
print("                              QQ Plot                                ")
plt.figure(figsize=(6,6), dpi=100)
stats.probplot(df["gold"],dist="norm",plot=plt)
plt.show()
```

```
                              QQ Plot
```



In [25]:
```
print("     Chisquare    ")
stat , p1 = stats.chisquare(df["gold"])
print('Statistics=%.3f, p=%.3f' % (stat,p1))
alpha = 0.05
if p1 > alpha :
    print( 'Sample looks Gaussian ( fail to rejectH0)' )
else :
    print('Sample does not look Gaussian ( rejectH0)')
```

```
    Chisquare
Statistics=192.077, p=1.000
Sample looks Gaussian ( fail to rejectH0)
```

In [26]:
```
print( '------------------ Shapiro-Wilk ------------------------' )
stat , p2 = stats.shapiro(df["gold"])
print('Statistics=%.3f , p=%.3f ' % ( stat , p2))
alpha = 0.05
if p2 > alpha :
    print('Sample looks Gaussian (failed to reject H0)')
else :
    print('Sample does not look Gaussian ( reject H0)')
```

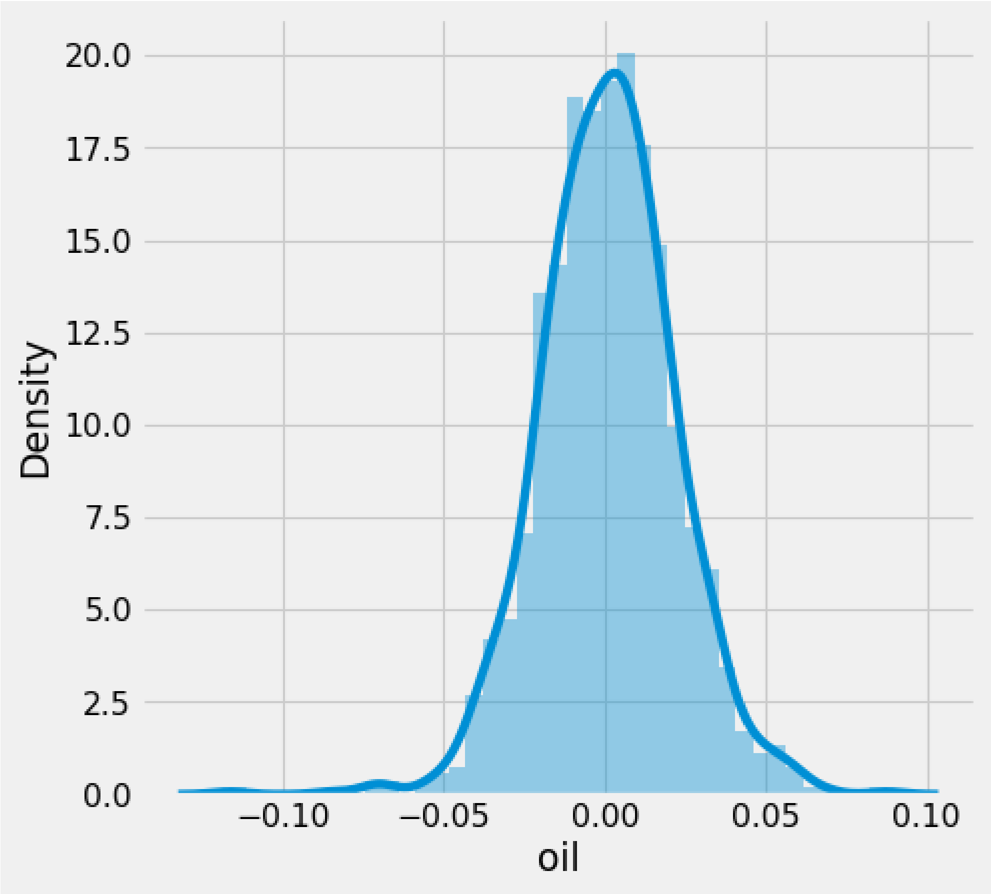
```
------------------ Shapiro-Wilk ------------------------
Statistics=0.969 , p=0.000
Sample does not look Gaussian ( reject H0)
```

In [27]:
```
print("                    Normality test for JPM             ")
print("                       JPM Histogram                   ")
```

```
plt.figure(figsize =(6,6), dpi=80)
sns.distplot(df['oil'], hist=True, kde=True)
```

```
                    Normality test for JPM
                       JPM Histogram
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```

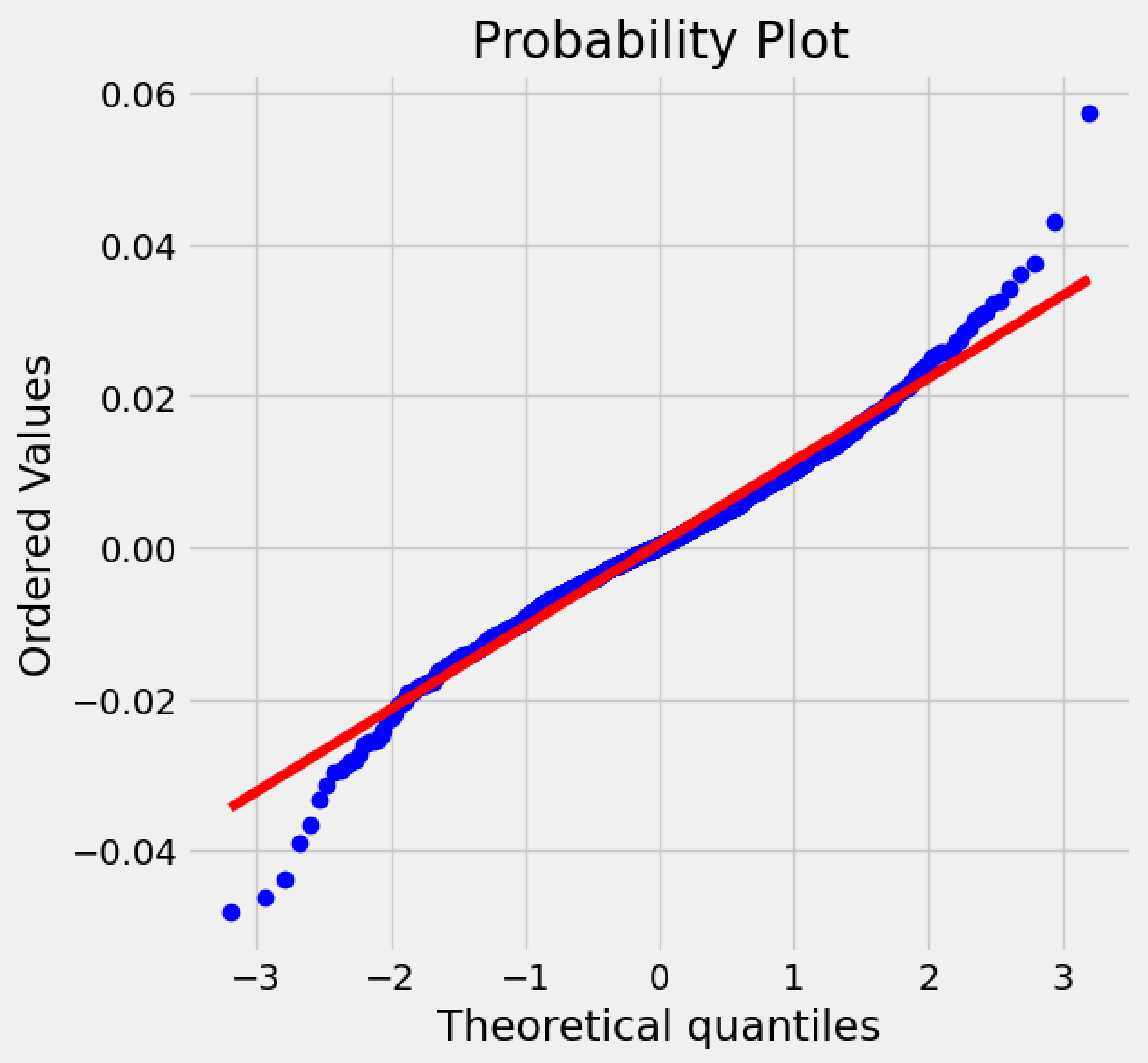
Out[27]: <AxesSubplot:xlabel='oil', ylabel='Density'>



In [28]:

```
print("                          QQ Plot                              ")
plt.figure(figsize=(6,6), dpi=100)
stats.probplot(df["JPM"],dist="norm",plot=plt)
plt.show()
```

```
                          QQ Plot
```



In [29]:

```
print("    Chisquare   ")
stat , p1 = stats.chisquare(df["JPM"])
print('Statistics=%.3f, p=%.3f' % (stat,p1))
alpha = 0.05
if p1 > alpha :
    print( 'Sample looks Gaussian ( fail to rejectH0)' )
else :
    print('Sample does not look Gaussian ( rejectH0)')
```

```
    Chisquare
Statistics=228.584, p=1.000
Sample looks Gaussian ( fail to rejectH0)
```

In [30]:

```
print( '                 Shapiro-Wilk                ' )
stat , p2 = stats.shapiro(df["JPM"])
print('Statistics=%.3f , p=%.3f ' % ( stat , p2))
alpha = 0.05
if p2 > alpha :
    print('Sample looks Gaussian (failed to reject H0)')
else :
    print('Sample does not look Gaussian ( reject H0)')
```

```
               Shapiro-Wilk
Statistics=0.980 , p=0.000
Sample does not look Gaussian ( reject H0)
```

Part 4: Break your data into small groups and

let them discuss the importance of the

Central Limit Theorem

Consider the ETF column (1000 values) as the population (x), and do the follows.1) Calculate the mean μx and the standard deviation σx of the population. 2) Break the population into 50 groups sequentially and each group includes 20 values. 3) Calculate the sample mean (x) of each group. Draw a histogram of all the sample means. Comment on the distribution of these sample means, i.e., use the histogram to assess the normality of the data consisting of these sample means. 4) Calculate the mean (μx) and the standard deviation (σx) of the data including these sample means. Make a comparison between μx and μx , between $\sigma x\ n$ and σx . Here, n is the number of sample means calculated from Item 3) above. 5) Are the results from Items 3) and 4) consistent with the Central Limit Theorem? Why? 6) Break the population into 10 groups sequentially and each group includes 100 values. 7) Repeat Items 3) ~ 5). 8) Generate 50 simple random samples or groups (with replacement) from the population. The size of each sample is 20, i.e., each group includes 20 values. 9) Repeat Items 3) ~ 5). 10) Generate 10 simple random samples or groups (with replacement) from the population. The size of each sample is 100, i.e., each group includes 100 values. 11) Repeat Items 3) ~ 5). 12) In Part 3 of the project, you have figured out the distribution of the population (the entire ETF column). Does this information have any impact on the distribution of the sample mean(s)? Explain your answer.

In [31]:
```
#1)Sequential  split  of  data
def split_data_seq(data, size):
    return np.array_split(data, size)
```

In [32]:
```
#To print sample mean
def print_mean(data):
    for index, value in enumerate(data):
        print("group" + str(index+1) + "---> "+str(np.mean(value)))
```

In [33]:
```
#Return array of samplemean
def mean_array(data):
    mean_value = []
    for value in data:
        mean_value.append(np.mean(value))
    return mean_value
```

In [34]:
```
#mean of sample means
def mean_mean(data):
    return np.mean(mean_array(data))
```

In [35]:
```
#standard deviation of mean
def std_of_samples(data):
    return np.std(mean_array(data))
```

In [36]:
```
#For spliting data randomly
def split_data_random(data, size, groups):
    random_array = []
    for i in range(groups):
        random_array.append(choices(data, k=size))
    return random_array
```

In [37]:
```
#1)  Calculate  the mean and  the  standard  deviation  of  the  population .
```

```
print("Mean_of_ETF_column", np.mean(df['Close_ETF']))
print('Standard_Deviation_ETF_column', np.std(df['Close_ETF']))

#2)  Break  the  population  into  50  groups  sequentially  and  each

seq_data_50 = split_data_seq(df['Close_ETF'], 50)
```

```
Mean_of_ETF_column 121.1529600120001
Standard_Deviation_ETF_column 12.563503845944297
```

In [38]:

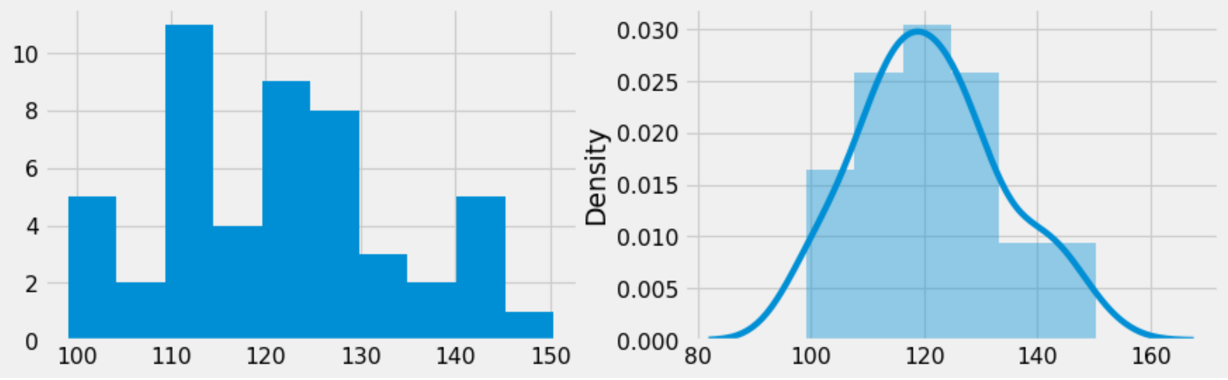
```
#3)Calculate  the  sample mean  of  each  group .Draw a histogram of all the
#of  these  sample means.
#sample mean
print('Sample_mean_of_each_group')
print_mean(seq_data_50)
```

```
Sample_mean_of_each_group
group1---> 99.32100080000002
group2---> 99.55399975000002
group3---> 99.15400055
group4---> 102.55050039999999
group5---> 103.29199995000002
group6---> 105.09350015
group7---> 106.75099974999998
group8---> 111.6580009
group9---> 114.49950014999997
group10---> 114.40050045000001
group11---> 112.77649960000001
group12---> 112.28599980000001
group13---> 111.80899929999998
group14---> 113.27149915
group15---> 109.9474991
group16---> 110.14300039999998
group17---> 112.53550034999998
group18---> 112.0754997
group19---> 117.78150055
group20---> 120.0504997
group21---> 118.20800089999997
group22---> 119.98099934999998
group23---> 119.76750025000001
group24---> 116.80299985000003
group25---> 117.24199984999998
group26---> 120.55450105
group27---> 121.09150044999998
group28---> 123.40999985
group29---> 122.7170002
group30---> 120.61099994999998
group31---> 120.50799975000002
group32---> 125.79700005
group33---> 126.88300015
group34---> 127.30250020000003
group35---> 128.43750040000003
group36---> 130.13649915
group37---> 130.58250049999998
group38---> 128.15899955
group39---> 125.12550015
group40---> 126.06000055000001
group41---> 129.02949995
group42---> 131.8114998
group43---> 135.97399985
group44---> 138.857
group45---> 141.28849860000003
group46---> 142.17150035
group47---> 144.62450029999997
group48---> 140.5229988
group49---> 144.69050135000003
group50---> 150.35049894999997
```

In [39]:
```
#Histogram of mean of samples
mean_seq_data_50 = mean_array(seq_data_50)
figure(figsize=(12, 8), dpi=80)
plt.subplot(2,2,1)
plt.hist(mean_seq_data_50)
plt.subplot(2,2,2)
sns.distplot(mean_seq_data_50 , hist=True, kde=True)
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[39]:
```
<AxesSubplot:ylabel='Density'>
```

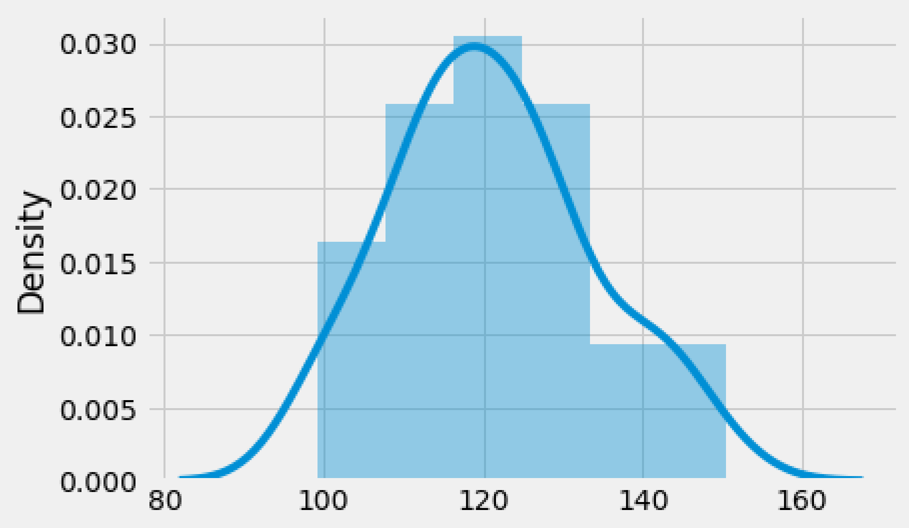


In [40]:
```
#Histogram of mean of samples for 1000 sample
array_of_mean = []
for i in range(20):
        seq_data = split_data_seq(df['Close_ETF'] , 50)
for value in seq_data:
        array_of_mean.append(np.mean(value))
        print("Total_Sample : ", len(array_of_mean))
sns.distplot(array_of_mean , hist=True, kde=True)
```

```
Total_Sample :  1
Total_Sample :  2
Total_Sample :  3
Total_Sample :  4
Total_Sample :  5
Total_Sample :  6
Total_Sample :  7
Total_Sample :  8
Total_Sample :  9
Total_Sample :  10
Total_Sample :  11
Total_Sample :  12
Total_Sample :  13
Total_Sample :  14
Total_Sample :  15
Total_Sample :  16
Total_Sample :  17
Total_Sample :  18
Total_Sample :  19
Total_Sample :  20
Total_Sample :  21
Total_Sample :  22
Total_Sample :  23
Total_Sample :  24
Total_Sample :  25
Total_Sample :  26
Total_Sample :  27
Total_Sample :  28
Total_Sample :  29
```

```
Total_Sample :  30
Total_Sample :  31
Total_Sample :  32
Total_Sample :  33
Total_Sample :  34
Total_Sample :  35
Total_Sample :  36
Total_Sample :  37
Total_Sample :  38
Total_Sample :  39
Total_Sample :  40
Total_Sample :  41
Total_Sample :  42
Total_Sample :  43
Total_Sample :  44
Total_Sample :  45
Total_Sample :  46
Total_Sample :  47
Total_Sample :  48
Total_Sample :  49
Total_Sample :  50
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```

Out[40]: <AxesSubplot:ylabel='Density'>



In [41]:

```
#4)  Calculate  the mean and  the  standard  deviation  of  the
#data including these sample means. Make a comparison between
#sd/ sqrt (n)  and  standard  deviation .
print("Mean : "+str(mean_mean(seq_data_50)))
print("Standard_deviation : "+str(std_of_samples(seq_data_50)))
print("Mean : ", np.mean(df['Close_ETF'] , axis=0) , "and_mean_of_samples : "
print("sd/sqrt(n)" , (np.std(df['Close_ETF'])/math.sqrt(50)) ,"and_standard_d
```

```
Mean : 121.15296001199998
Standard_deviation : 12.489175897769007
Mean :  121.1529600120001 and_mean_of_samples :  121.15296001199998
sd/sqrt(n) 1.7767477529860964 and_standard_deviation_of_samples :  12.48917589
7769007
```

In [42]:

```
#5) Are the res ul ts from Items 3) and 4) consistent with the Central Limit
#6)  Break  the  population  into  10  groups  sequentially  and  each  group
#Spliting  10 groups  sequentially
seq_data_100 = split_data_seq(df['Close_ETF'] , 10)
```

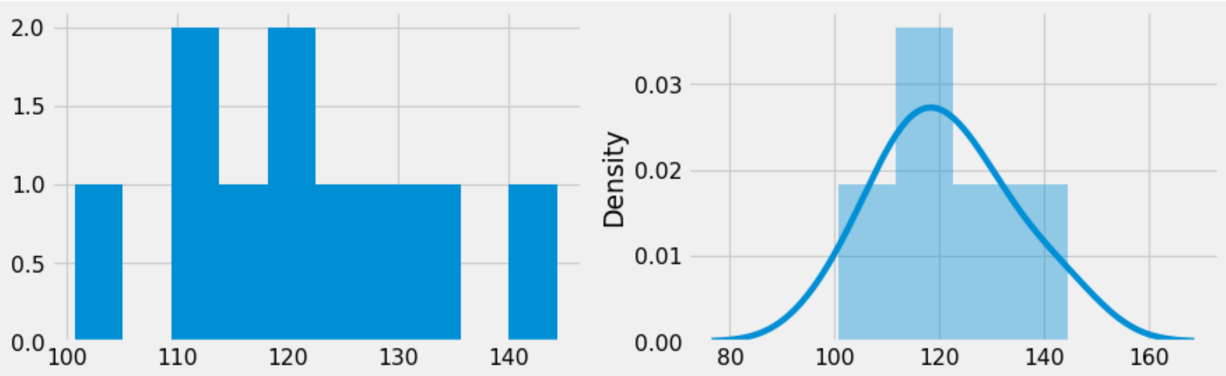
In [43]:
```
#7)  Repeat  Items  3)  ~  5).
print("Sample_mean_of_every_group")
print_mean(seq_data_100)
```

```
Sample_mean_of_every_group
group1---> 100.77430028999999
group2---> 110.48050028
group3---> 112.01809938999999
group4---> 114.51720014000003
group5---> 118.40030003999999
group6---> 121.67680029999993
group7---> 125.78560010999992
group8---> 128.01269997999995
group9---> 135.3920996399999
group10---> 144.47199995
```

In [44]:
```
#Histogram  of  sample mean
mean_seq_data_100 = mean_array(seq_data_100)
figure(figsize=(12, 8), dpi=80)
plt.subplot(2,2,1)
plt.hist(mean_seq_data_100)
plt.subplot(2,2,2)
sns.distplot(mean_seq_data_100 , hist=True, kde=True)
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```

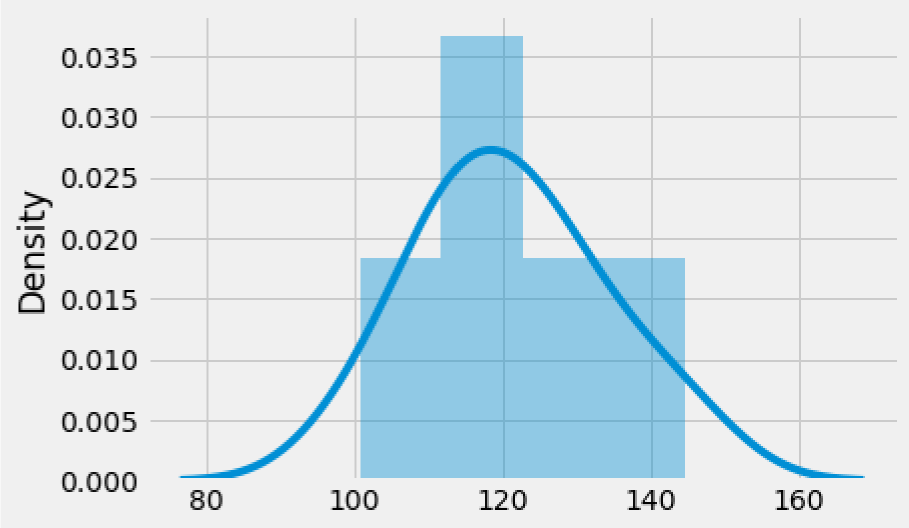
Out[44]: <AxesSubplot:ylabel='Density'>



In [45]:
```
#Histogram  of  sample mean  for  1000  size
array_of_mean = []
for i in range(100):
    seq_data = split_data_seq(df['Close_ETF'] , 10)
for value in seq_data:
    array_of_mean.append(np.mean(value))
print("Total_Sample : ", len(array_of_mean))
sns.distplot(array_of_mean , hist=True, kde=True)
print("Mean : "+str(mean_mean(seq_data_100)))
print("Standard_deviation : "+str(std_of_samples(seq_data_100)))
print("Mean : ", np.mean(df['Close_ETF'] , axis=0) ,"and_mean_of_samples : ",
print("sd/sqrt(n) : ", (np.std(df['Close_ETF'])/math.sqrt(100)) ,"and_standar
```

```
Total_Sample :  10
Mean : 121.15296001199997
Standard_deviation : 12.16375686089257
Mean :  121.1529600120001 and_mean_of_samples :  121.15296001199997
sd/sqrt(n) :  1.2563503845944297 and_standard_deviation_of_samples :  12.16375
686089257
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
```

```
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```



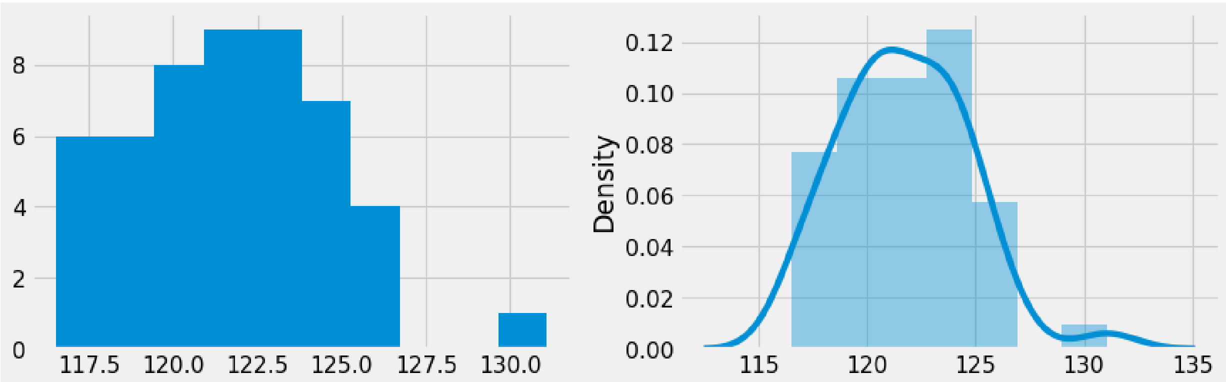
In [46]:

```python
#8) Generate 50 simple random samples or groups ( with replacement )
#population . The size of each sample is 20, i . e . each
#group includes 20 values .
#Spliting 50 simple random groups
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from random import choices
import scipy.stats as stats
from scipy.stats import shapiro , normaltest
from matplotlib.pyplot import figure
random_data_50 = split_data_random(df['Close_ETF'] , 20, 50)
#9) Repeat Items 3) ~ 5)
print("Sample_mean_of_each_group")
print_mean(random_data_50)
#Histogram of sample mean
mean_random_data_50 = mean_array(random_data_50)
figure(figsize=(12, 8), dpi=80)
plt.subplot(2,2,1)
plt.hist(mean_random_data_50)
plt.subplot(2,2,2)
sns.distplot(mean_random_data_50, hist=True, kde=True)
```

```
Sample_mean_of_each_group
group1---> 122.09400065
group2---> 118.41899865
group3---> 125.59350009999999
group4---> 124.8624996
group5---> 118.3629997
group6---> 124.2880001
group7---> 124.18850089999998
group8---> 119.26050024999999
group9---> 124.02899974999998
group10---> 121.97600179999999
group11---> 121.07699960000002
group12---> 122.80399910000001
group13---> 122.85900005
group14---> 123.3195004
group15---> 120.39150079999999
group16---> 123.73150065
group17---> 120.75900085
group18---> 119.42650110000002
group19---> 123.32900079999999
```

```
group20---> 121.05400010000001
group21---> 123.77449915
group22---> 121.19300035
group23---> 125.16749989999998
group24---> 123.6034995
group25---> 120.19699929999999
group26---> 116.52800105000001
group27---> 125.26600034999998
group28---> 124.3630009
group29---> 121.58199900000002
group30---> 117.47799995
group31---> 120.66700065
group32---> 121.14099959999999
group33---> 131.08850105
group34---> 126.69299955
group35---> 119.59800109999999
group36---> 122.74849965000001
group37---> 117.91949965
group38---> 116.75450014999998
group39---> 116.98049964999998
group40---> 119.36900035000001
group41---> 119.4749995
group42---> 122.83250040000003
group43---> 120.11800070000001
group44---> 118.78399960000002
group45---> 120.12650024999998
group46---> 121.4454991
group47---> 117.74649935
group48---> 123.92250019999999
group49---> 121.3625004
group50---> 126.05799979999999
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[46]: <AxesSubplot:ylabel='Density'>

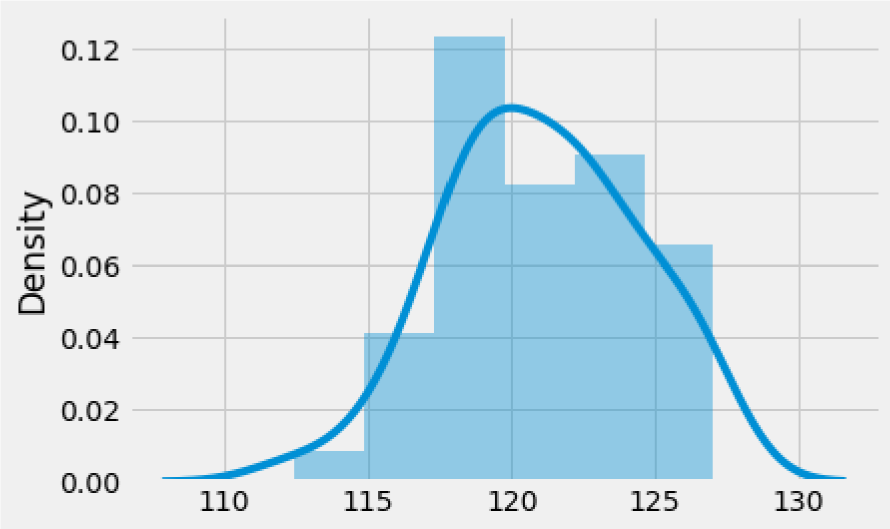


In [47]:

```
#Histogram of sample mean of 1000 sample
array_of_mean = []
for i in range(20):
    seq_data = split_data_random(df['Close_ETF'] , 20, 50)
for value in seq_data:
    array_of_mean.append(np.mean(value))
print("Total_Sample : ", len(array_of_mean))
sns.distplot(array_of_mean , hist=True, kde=True)
print("Mean: "+str(mean_mean(random_data_50)))
print("Standard_deviation : "+str(std_of_samples(random_data_50)))
print("Mean: ", np.mean(df['Close_ETF'] , axis=0), "and_mean_of_samples_: ", 
print("sd/sqrt(n) ",(np.std(df['Close_ETF'])/math.sqrt(50)) ,"and_standard_de
```

```
Total_Sample :  50
```

```
Mean: 121.71616010199999
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
Standard_deviation : 2.914893164696741
Mean:  121.1529600120001 and_mean_of_samples_:  121.71616010199999
sd/sqrt(n)  1.7767477529860964 and_standard_deviation_of_samples :  2.91489316
4696741
```



In [48]:

```
#10)  Generate  10  simple  random  samples  or  groups  ( with  replacement
#population .The  size  of  each  sample  is  100,  i . e .  each  group
#includes  100  values .
#Spliting  10 simple random samples or groups
random_data_100 = split_data_random(df['Close_ETF'] , 100, 10)
#11)  Repeat Items 3) 5).
print('Sample_mean_of_each_group')
print_mean(random_data_100)
```

```
Sample_mean_of_each_group
group1---> 119.52159982999997
group2---> 122.06269973
group3---> 120.80799978
group4---> 120.00519999999999
group5---> 121.34169988000002
group6---> 121.91139976999997
group7---> 121.26779988
group8---> 123.12549995
group9---> 121.84260025999998
group10---> 121.75780012
```

In [49]:

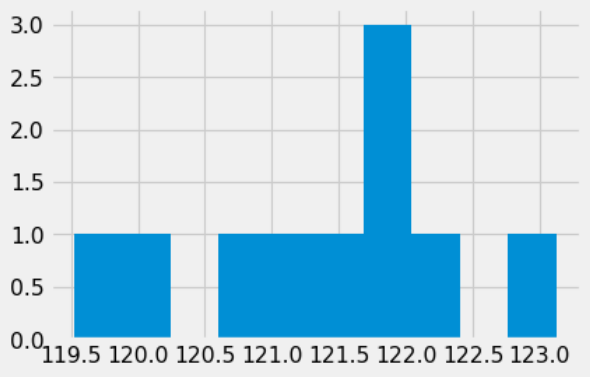
```
#Histogram  of  mean  of  samples
import math
mean_random_data_100 = mean_array(random_data_100)
figure(figsize=(12, 8), dpi=80)
plt.subplot(2,2,1)
plt.hist(mean_random_data_100)
plt.subplot(2,2,2)
sns.distplot(mean_random_data_100, hist=True, kde=True)
#Histogram  of  mean  of  1000  samples
array_of_mean = []
for i in range(100):
    seq_data = split_data_random(df['Close_ETF'] , 100, 10)
for value in seq_data:
    array_of_mean.append(np.mean(value))
    print("Total_Sample : ", len(array_of_mean))
    sns.distplot(array_of_mean , hist=True, kde=True)
```

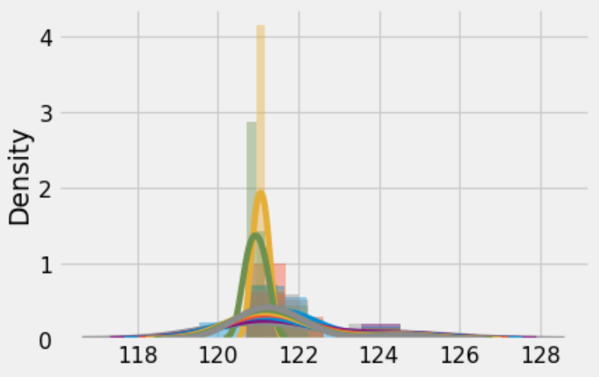
```
print("Mean: "+str(mean_mean(random_data_100)))
print("Standard_deviation : "+str(std_of_samples(random_data_100)))
print("Mean : ", np.mean(df['Close_ETF'] , axis=0) ,"and_mean__samples :
mean_mean(random_data_100))

print("sd/sqrt(n)", (np.std(df['Close_ETF'])/math.sqrt(100)) ,"and_standa
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
Total_Sample :  1
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  2
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  3
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  4
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  5
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  6
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  7
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  8
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
Total_Sample :  9
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
```

```
Total_Sample :  10
Mean: 121.36442991999999
Standard_deviation : 0.9916909104815054
Mean :  121.1529600120001 and_mean__samples :  121.36442991999999
sd/sqrt(n) 1.2563503845944297 and_standard_deviation_of_samples :  0.991690910
4815054
```

```
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:305: UserW
arning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: Futu
reWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
```





In [50]:
```
import scipy.stats as st

df_etf = df['Close_ETF']
```

In [51]:
```
df_etf = df['Close_ETF']
etf_sample_100 = df_etf.sample(n=100, replace=True, random_state=100)
```

In [52]:
```
# Confidence Interval is given by:- x +/- t*(s/√n)

# where
#    x: sample mean(122.156)
#    t: t-value that corresponds to the confidence level 0.05 (1.960)
#    s: sample standard deviation(13.64)
#    n: sample size(100)

122.156 - 1.960*(np.std(etf_sample_100)/np.sqrt(len(etf_sample_100))),122.156

# 95% Confidence Interval: 122.15 ± 2.67
```

Out[52]: (119.48113668779801, 124.830863312202)

In [53]:
```
st.norm.interval(alpha=0.95, loc=np.mean(etf_sample_100), scale=st.sem(etf_sa

# We chose the scipy.stats.norm.interval() method since the sample size is gr

# The location (loc) keyword specifies the mean. The scale (scale) keyword sp
# the square root of sample length.

# reference https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.
# https://www.statology.org/confidence-intervals-python/
```

Out[53]: (119.46781113944346, 124.84438990055662)

In [54]:
```
#part 5.2
etf_sample_20 = df_etf.sample(n=20, replace=True,  random_state=100)
```

In [55]:
```
# Using the same formula as above

125.422 - 1.960*(np.std(etf_sample_20)/np.sqrt(len(etf_sample_20))),125.422 +
```

Out[55]: (120.07492723467257, 130.76907276532742)

In [56]:
```
# using the scipy.norm.interval() method

st.norm.interval(alpha=0.95, loc=np.mean(etf_sample_20), scale=st.sem(etf_sam
```

Out[56]: (119.93612040193646, 130.90788059806357)

In [57]:
```
# we used the scipy.t.interval() method here since, the number of data points

st.t.interval(alpha=0.95, df = 19, loc=np.mean(etf_sample_20), scale=st.sem(e
```

Out[57]: (119.56368927825525, 131.28031172174477)

In [58]:
```
np.std(etf_sample_20)
```

Out[58]: 12.200426718275422

In [59]:
```
# wilcoxon test since the sample doesn't follow a normal distribution

#x <- c(119.860001,
# 126.209999,
# 131.470001,
# 138.580002,
# 136.83999599999999,
# 102.940002,
# 138.669998,
# 140.53999299999998,
# 110.519997,
# 140.740005,
# 138.080002,
# 99.620003,
# 123.150002,
# 132.520004,
# 119.529999,
# 127.900002,
# 127.730003,
# 111.860001,
# 127.379997,
# 114.300003)

# > wilcox.test(x, mu=122.156100, conf.int = T)

# alternative hypothesis: true location is not equal to 122.1561
# 95 percent confidence interval:
# 119.620 <-> 132.395

# reference https://www.rdocumentation.org/packages/stats/versions/3.6.2/topi
```

In [60]:
```
# the range is 132.395 - 119.62 = 12.77
```

In [61]:
```
# 6th
meanSampData = np.mean(etf_sample_100)
hypMean = 100
n = 100
std_pop = np.std(df_etf)
```

In [62]:
```
np.mean(etf_sample_100)
```

Out[62]: 122.15610052000004

In [63]:
```
# We went with the z-test since the popluation std deviation is known which i

# even though z-test assumes normal distribution and the data is not normally
# (100 in this case) is large enough to conduct the test

# Now the formula for z - value is
```

```
z = (meanSampData-hypMean)/(std_pop/np.sqrt(n))
z
```

Out[63]: 17.635287728393052

In [64]:
```
# Method 1(using p-value)

# Using the P-value approach: The p-value is p=0 and since 0<0.05

# it is concluded that the null hypothesis is rejected.

# https://mathcracker.com/z-test-for-one-mean
```

In [65]:
```
# Method 2(using critical values):

# this is a 2 sided test

# value of z at .05 making it .025 for 2 sided we know from z table z = (+ 1.

# as calculated z score 17.63 is greater than 1.96 (tabular z score), we reje

# Observed z-value = 17.63

# Critical value = 1.96

# Reference : https://github.com/sharmasw/Data-Science-with-python/blob/maste
#             https://www.youtube.com/watch?v=kd6zKBa9Rfk
#             https://www.statisticshowto.com/probability-and-statistics/find

# Online calculator : https://mathcracker.com/z-test-for-one-mean
```

In [66]:
```
#6.2

meanSampData = np.mean(etf_sample_20)
hypMean = 100
n = 20
std_sam = np.std(etf_sample_20)
pop_mean = 121.152
```

In [67]:
```
T = (meanSampData-pop_mean)/(std_sam/np.sqrt(n))
T
```

Out[67]: 1.5651930219220662

In [68]:
```
# Observations: for t test

# this is a 2 sided test

# value of t at .05 making it .025 for 2 sided we know from t table t = (+ -)

# as calculated t value 1.565 is lesser than 2.093, we failed to reject the n

# Reference https://www.statisticshowto.com/probability-and-statistics/t-dist

# Observations : for wilcoxon test

# References https://sixsigmastudyguide.com/1-sample-wilcoxon-non-parametric-
#            http://www.sthda.com/english/wiki/one-sample-wilcoxon-signed-ran

# x <- c(119.860001,
# 126.209999,
# 131.470001,
```

```
# 138.580002,
# 136.83999599999999,
# 102.940002,
# 138.669998,
# 140.53999299999998,
# 110.519997,
# 140.740005,
# 138.080002,
# 99.620003,
# 123.150002,
# 132.520004,
# 119.529999,
# 127.900002,
# 127.730003,
# 111.860001,
# 127.379997,
# 114.300003)
# > wilcox.test(x, mu = 100, alternative = "two.sided")
#    Wilcoxon signed rank test
# data:  x
# V = 209, p-value = 3.815e-06
# alternative hypothesis: true location is not equal to 100
```

In [69]:
```
#6.3
# https://www.itl.nist.gov/div898/handbook/eda/section3/eda358.htm
# Using the Chi-Square method(two tailed)
etf_sample_20 = df_etf.sample(n=20, replace=True,  random_state=100)
N = len(etf_sample_20)
stdSampData = np.std(etf_sample_20)
hypStd = 15

T = [(N-1) * ((stdSampData/hypStd)**2)]
T
```

Out[69]: [12.569590355787406]

In [70]:
```
# https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm
# reject if greater than 32.852  and less than 8.907
# Hence we failed to reject the null hypothesis
# But for random_state = 0 the null hypothesis is getting rejected
```

In [71]:
```
##Part 6.4 (not two tailed)
# Using the Chi-Square method one tailed

N = 20
stdSampData = np.std(etf_sample_20)
hypStd = 15

T = [(N-1) * ((stdSampData/hypStd)**2)]
T
```

Out[71]: [12.569590355787406]

In [72]:
```
# https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm

# reject if less than 10.117
```

In [73]:
```
# Observations: Failed to reject the Null hypothesis
```

In [74]:
```
#Part 7.1
```

In [75]:
```
# manual formula method
x = np.array(df['oil'])
y = np.array(df['gold'])

t = (np.mean(x)-np.mean(y))/np.sqrt(((np.std(x)*np.std(x))/len(x))+((np.std(y
pval = st.t.sf(np.abs(t), 1000-1)*2

t , pval
# https://www.socscistatistics.com/pvalues/tdistribution.aspx
```

Out[75]: (0.4856094792948105, 0.6273505577888034)

In [76]:
```
# References:-
# https://www.youtube.com/watch?v=0Pd3dc1GcHc
# https://www.youtube.com/watch?v=8aaIdXENNJI
# https://github.com/bhattbhavesh91/GA_Sessions/blob/master/t_test_independen

# Observations:-
# We failed to reject the null hypothesis (the means of oil and gold are equa
```

In [118…
```
#!pip install researchpy
#from researchpy import ttest as rpTtest
researchpy.ttest(df['oil'], df['gold'])
```

In [80]:
```
# manual formual mehtod

d = df['oil'] - df['gold']
mean_d = np.mean(d)
std_d = np.std(d)
d_sqrd = d*d

t = np.sum(d)/np.sqrt((1000*np.sum(d_sqrd)-(np.sum(d)*np.sum(d)))/(998))
t
```

Out[80]: 0.5410599236152893

In [81]:
```
# the critical value for significance level 0.05 and dof 999 is 1.96

# since 0.541 is between + or - 1.96 (we failed to reject the null hypothesis
```

In [82]:
```
var_oil = np.var(x)
var_gold = np.var(y)
var_oil, var_gold
```

Out[82]: (0.00044446545891371905, 0.00012731543865693258)

In [83]:
```
F = (np.power(var_oil,2))/np.power(var_gold,2)
F
```

Out[83]: 12.187479283391776

In [84]:
```
# Reference : https://www.statisticshowto.com/probability-and-statistics/hypo
#             https://mathcracker.com/f-critical-values#results
# Observations

# Critical f-values:    FL=0.883 and FU=1.132

# since the F value is towards the right of the critical value, we are in the

# hence we reject the null hypothesis
```

In [85]:
```
import seaborn as sns
import statsmodels.api as sm
import statsmodels.tsa.api as smt
import scipy.stats as stats
import warnings
import pylab

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.stats.diagnostic import linear_harvey_collier

warnings.filterwarnings("ignore")
%matplotlib inline
```

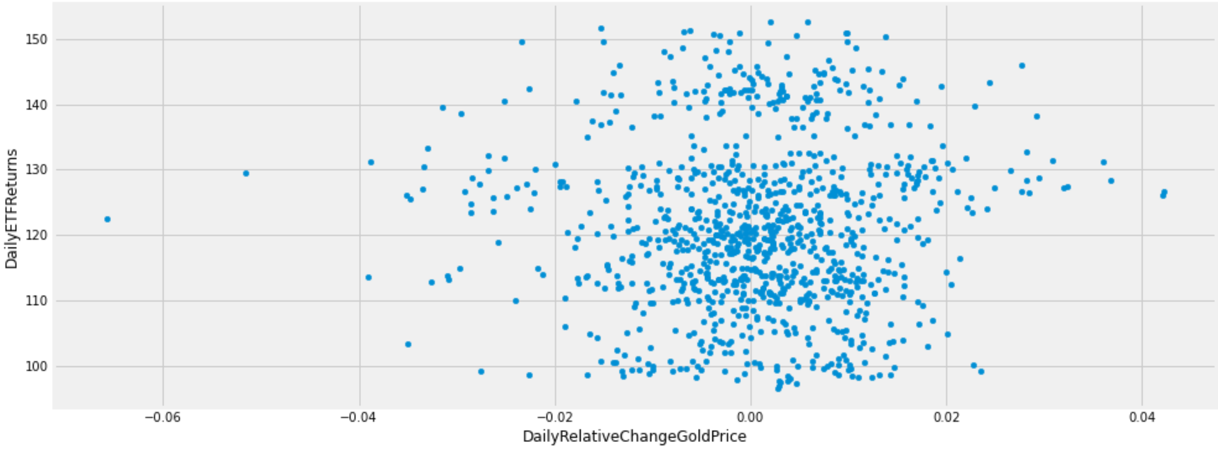
In [86]:
```
pdInputData = pd.read_excel("Data.xlsx")
pdInputData.rename(columns={"Close_ETF" : "DailyETFReturns", "oil": "DailyRel
                            "gold": "DailyRelativeChangeGoldPrice", "JPM": "D
pdInputDataP8 = pdInputData[['DailyETFReturns', 'DailyRelativeChangeGoldPrice
pdInputDataP8.head()
```

Out[86]:

	DailyETFReturns	DailyRelativeChangeGoldPrice
0	97.349998	0.004668
1	97.750000	-0.001366
2	99.160004	-0.007937
3	99.650002	0.014621
4	99.260002	-0.011419

In [87]:
```
#1) Draw a scatter plot of ETF (Y) vs. Gold (X). Is there any linear relation
pdInputDataP8.plot.scatter(x="DailyRelativeChangeGoldPrice", y ="DailyETFRetu
print("Starting points of the graph are on x axis:{} on y axis: {}".format(pd
```

```
Starting points of the graph are on x axis:-0.065804741 on y axis: 96.419998
```



In [88]:
```
#2) Calculate the coefficient of correlation between ETF and Gold and interpr
def get_Correlation_coefficient(x, y):
    n = len(x)
    return (   ( n * np.sum(x*y) )  -  ( np.sum(x) * np.sum(y) )   ) \
           / np.sqrt( ( ( n * np.sum( x**2) ) - (np.sum(x))**2 ) * ( ( n * np

# Get x and y input values
x = pdInputDataP8.DailyRelativeChangeGoldPrice
y = pdInputDataP8.DailyETFReturns
```

```
r = get_Correlation_coefficient(x, y)
print("The Pearson’s Correlation coefficient r between ETF and Gold is: ", r)
```

```
The Pearson’s Correlation coefficient r between ETF and Gold is:  0.0229955700
7605459
```

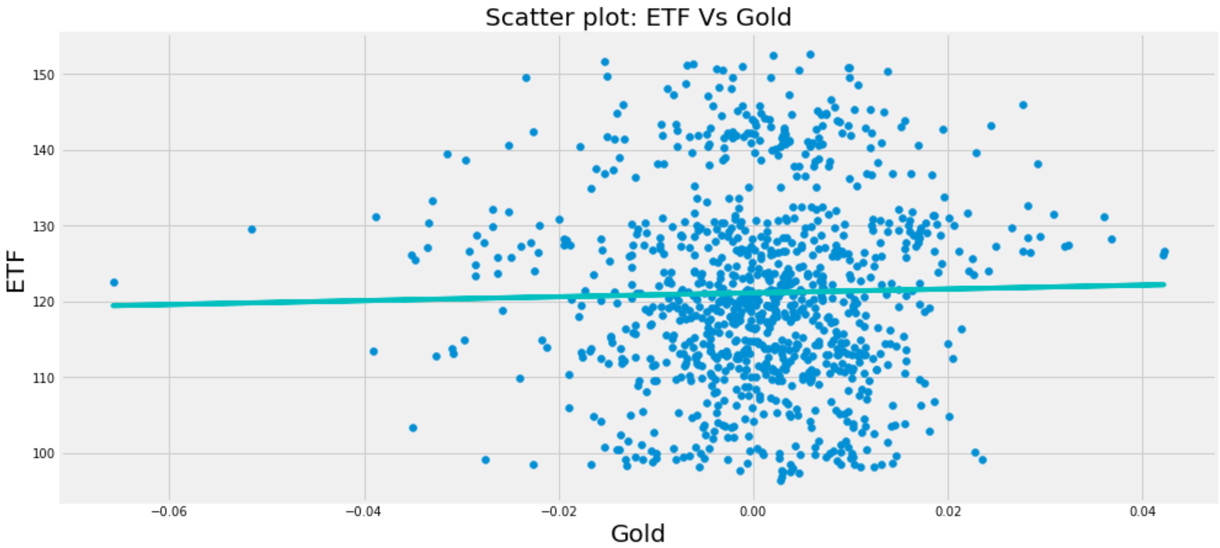
In [89]:

```
#3) Fit a regression line (or least squares line, best fitting line) to the s
def get_mean(serVariableValues):
    return serVariableValues.mean()

def get_slope(x, y):

    # Calculate x bar and y bar
    x_bar = get_mean(x)   # x bar
    y_bar = get_mean(y)   # y bar

    return  np.sum( (x-x_bar) * (y - y_bar) ) / np.sum( (x-x_bar)**2 )

def calculate_intercept_of_the_line(y, x, floarSlope):
    return np.mean(y - floarSlope * x) #b0 = y - b1x

def get_regression_line(x, floatIntercept, floarSlope):
    return floatIntercept + floarSlope * x

def calculate_regression_line(x, y):

    # Calculate b0 and b1 that is intercept and slope of the line
    floarSlope = get_slope(x, y)
    floatIntercept = calculate_intercept_of_the_line(y, x, floarSlope)
    y_hat = get_regression_line(x, floatIntercept, floarSlope)


    return floarSlope, floatIntercept, y_hat
# Get x and y input values
x = pdInputDataP8.DailyRelativeChangeGoldPrice
y = pdInputDataP8.DailyETFReturns
floarSlope, floatIntercept, y_hat = calculate_regression_line(x, y)
print("The slope of line={} and the intercept={}".format(floarSlope, floatInt

#pdInputDataP8.plot.scatter(x="DailyRelativeChangeGoldPrice", y ="DailyETFRet
plt.figure(figsize=(15,7))
plt.scatter(x, y)
plt.xlabel("Gold", fontsize = 20)
plt.ylabel("ETF", fontsize = 20)
plt.plot(x, y_hat, 'c')
plt.title("Scatter plot: ETF Vs Gold", fontsize = 20)
```

```
The slope of line=25.604389324427277 and the intercept=121.13598849889823
```

Out[89]:

```
Text(0.5, 1.0, 'Scatter plot: ETF Vs Gold')
```



In [90]:
```
#4) Conduct a two-tailed t-test with H0: β1=0.
b1=floarSlope
Beta1 = 0       # Null Hypothesis: When B1 = 0
n = len(x)
alpha = 0.01  # Is the linear relationship between ETF (Y) and Gold (X) signi
def get_mean_square_error( y, y_hat ):
    return np.square(np.subtract(y,y_hat)).mean()

#### Method 2:
#from sklearn.metrics import mean_squared_error
#def get_mean_square_error( y,y_hat ):
#    return mean_squared_error(y,y_hat)
def get_square_error(MSE, x):
    x_bar = get_mean(x)
    return np.sqrt(MSE) / np.sqrt(np.sum(np.square(x-x_bar)))

def get_t_score(x, y, y_hat, b1, Beta1):
    # Calculation of Mean Squared Error (MSE)
    MSE = get_mean_square_error( y, y_hat )
    # Calculation of Squared Error
    SEb1 = get_square_error(MSE, x)
    return (b1 - Beta1) / SEb1

def get_p_value_from_t_score(t_score):
    return stats.t.sf(np.abs(t_score), n-1)*2  # two-sided pvalue = Prob(abs(

t_score =  get_t_score(x, y, y_hat, b1, Beta1) # t-statistic for H0: B1 = 0
print("Hence t_score is: ", t_score)

p_value = get_p_value_from_t_score(t_score)
print("Got p value using t-score as: ", p_value)

def check_assumption(alpha, p_value):

    if (p_value > alpha) :
        print('Same distributions (failed to reject H0)')
    else:
        print('Different distributions (reject H0)')

check_assumption(alpha, p_value)
```

```
Hence t_score is:  0.727376117653451
Got p value using t-score as:  0.4671660043870999
Same distributions (failed to reject H0)
```

In [91]:
```
#5) Suppose that you use the coefficient of determination to assess the quali
```

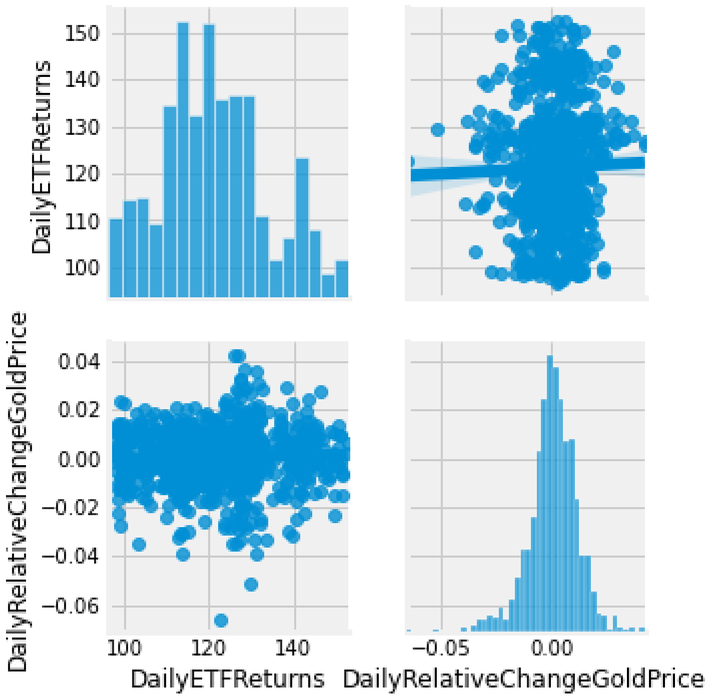
```
def get_residuals(y,y_hat):
    return y-y_hat

def squared_error(y,y_hat):
    return sum( np.square( get_residuals(y, y_hat) ) )

def total_sum_of_squares(y, y_bar):
    return np.sum( np.square( y - y_bar ) )

def coefficient_of_determination(y,y_hat):
    #coefficient of determination(R^2) =  1 - (RSS/ TSS)

    #RSS = sum of squares of residual errors
    floatRSS = squared_error(y, y_hat)

    y_bar = get_mean(y)
    #TSS = total sum of squares (proportional to the variance of the data)
    floatTSS = total_sum_of_squares(y, y_bar)

    return 1 - (floatRSS /  floatTSS)

# Get x and y input values
x = pdInputDataP8.DailyRelativeChangeGoldPrice
y = pdInputDataP8.DailyETFReturns

floarSlope, floatIntercept, y_hat = calculate_regression_line(x, y)

R2 = coefficient_of_determination(y,y_hat)
print("The coefficient of determination (R^2) score for a model to assess the
```

```
The coefficient of determination (R^2) score for a model to assess the quality
of this fitting is:  0.0005287962431228532
```

In [92]:

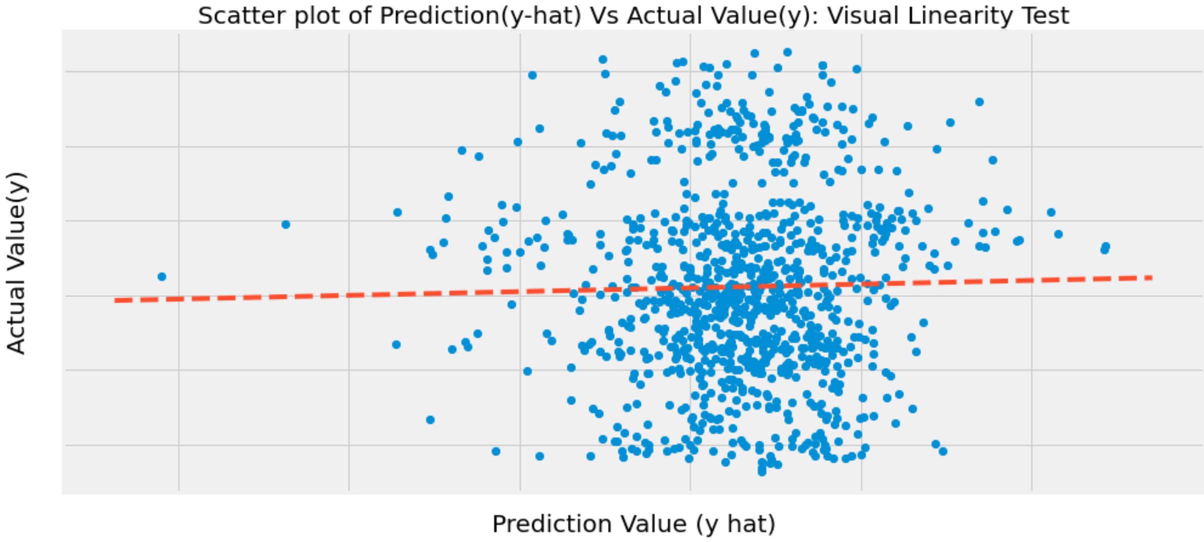
```
#6) What are the assumptions you made for this model fitting?
#Plot pairwise relationships in a dataset with one independent vairable xi an
plt.figure(figsize=(15,7))
sns.pairplot(pdInputDataP8[["DailyETFReturns", "DailyRelativeChangeGoldPrice"

# Save the file
plt.savefig('Part8Q6_1.png', bbox_inches='tight')
plt.show()
```

```
<Figure size 1080x504 with 0 Axes>
```

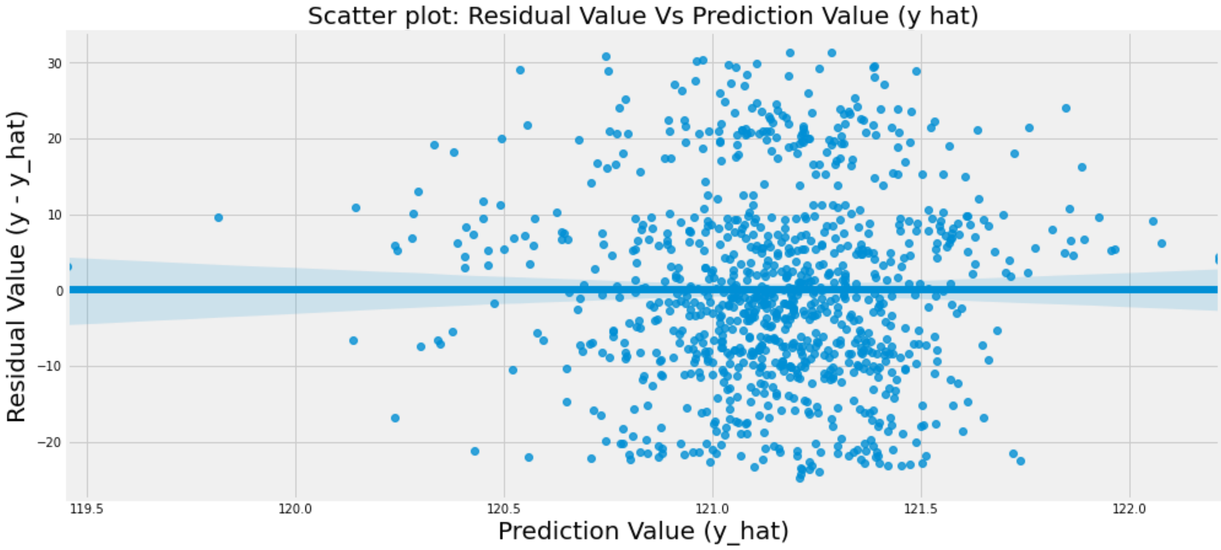


In [93]:

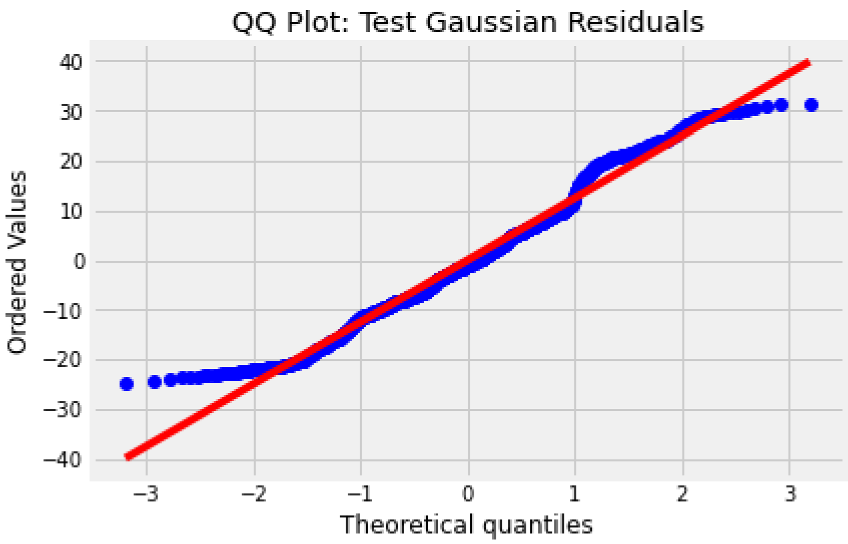
```
def abline(slope, intercept):
    # """Plot a line from slope and intercept, borrowed from https://stackove
    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = intercept + slope * x_vals
    plt.plot(x_vals, y_vals, '--')


#plot predicted vs actual
plt.figure(figsize=(15,7))
plt.scatter(y_hat, y)
plt.xlabel("Prediction Value (y hat)" , fontsize = 20)
plt.ylabel("Actual Value(y)", fontsize = 20)
plt.title("Scatter plot of Prediction(y-hat) Vs Actual Value(y): Visual Linea


plt.plot(y_hat, y,'o')
plt.tick_params(axis='x', colors='white')
plt.tick_params(axis='y', colors='white')
abline(1,0)
plt.show()

#fit an OLS model to data
model = sm.OLS(y, sm.tools.add_constant(x))
results = model.fit()
#predict y values for training data
y_hat = model.predict(results.params)


ttest, pval = sm.stats.diagnostic.linear_rainbow(res=results)
def check_assumption(alpha, p_value):

    if (p_value > alpha) :
        print('Same distributions (failed to reject H0)')
    else:
        print('Different distributions (reject H0)')

alpha = 0.05
check_assumption(alpha, pval)

ttest, pval = linear_harvey_collier(results)
check_assumption(alpha, pval)

### # Plot Predict Vs Residual To Check Linearity
serResidual=get_residuals(y,y_hat)
plt.figure(figsize=(15,7))
sns.regplot(x=y_hat,y=serResidual)
plt.xlabel("Prediction Value (y_hat)", fontsize = 20)
plt.ylabel("Residual Value (y - y_hat)", fontsize = 20)
plt.title("Scatter plot: Residual Value Vs Prediction Value (y hat)", fontsiz
plt.show()
```




```
Different distributions (reject H0)
Different distributions (reject H0)
```



In [94]:
```
stats.probplot(y-y_hat, dist="norm", plot=pylab)
pylab.title('QQ Plot: Test Gaussian Residuals')
pylab.show()
```



In [95]:
```
#7) Given the daily relative change in the gold price is 0.005127. Calculate
def get_confidance_intervals(floatAlpha, n_1, pop_mean):
    return stats.t.interval(alpha=floatAlpha, df=n_1, loc=pop_mean)
# 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
```

```
n_1 = len(pdInputDataP8)-1
daily_relative_cahnge = 0.005127
floatAlpha = 0.99

confidence_interval = get_confidance_intervals(floatAlpha, n_1, daily_relativ
print("Confidance interval for gold is (with daily relative change in the gol


# Method 1: Without scaled

import scipy.stats as st
#create 99% confidence interval for same sample

n_1 = len(pdInputDataP8)-1
pop_mean = np.mean(pdInputDataP8.DailyETFReturns)
floatAlpha = 0.99
confidence_interval  = get_confidance_intervals(floatAlpha, n_1, pop_mean)
print("Confidance interval for Close_ETF is ", confidence_interval, "For alph

#create 99% confidence interval for same sample

n_1 = len(pdInputDataP8)-1
pop_mean = np.mean(pdInputDataP8)
floatAlpha = 0.99

confidence_interval = get_confidance_intervals(floatAlpha, n_1, pop_mean)
print("Confidance interval for Close_ETF & gold is ", confidence_interval, "F
```

```
Confidance interval for gold is (with daily relative change in the gold price
is 0.005127)  (-2.575632637267628, 2.585886637267628) For alpha:  0.99
Confidance interval for Close_ETF is  (118.57220037473246, 123.73371964926773)
For alpha:  0.99
Confidance interval for Close_ETF & gold is  (array([118.57220037,  -2.5800968
]), array([123.73371965,   2.58142247])) For alpha:  0.99
```

In [96]:

```
# PART 9
pdInputData = pd.read_excel("Data.xlsx")

X = pdInputData[['gold', "oil", "JPM"]]
y = pdInputData['Close_ETF']

# Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    random_state=42)


X_with_constant = sm.add_constant(X_train)
model = sm.OLS(y_train, X_with_constant)

results = model.fit()
print(results.params)
print(results.summary())
X_test = sm.add_constant(X_test)
y_pred = results.predict(X_test)
```

```
const     121.046690
gold       18.293780
oil        -2.101395
JPM        30.555632
dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:              Close_ETF   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.003
Method:                 Least Squares   F-statistic:                    0.2948
```

```
Date:                Fri, 10 Dec 2021   Prob (F-statistic):                0.829
Time:                        17:27:23   Log-Likelihood:                  -3155.4
No. Observations:                 800   AIC:                               6319.
Df Residuals:                     796   BIC:                               6338.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        121.0467      0.445    272.164      0.000     120.174     121.920
gold          18.2938     41.186      0.444      0.657     -62.553      99.141
oil           -2.1014     21.508     -0.098      0.922     -44.320      40.117
JPM           30.5556     39.947      0.765      0.445     -47.859     108.970
==============================================================================
Omnibus:                       18.936   Durbin-Watson:                   1.931
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               16.314
Skew:                           0.283   Prob(JB):                     0.000287
Kurtosis:                       2.588   Cond. No.                         98.1
==============================================================================

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

In [97]:

```
# assumptions:
from sklearn.metrics import r2_score
R2 = r2_score(y_test, y_pred)
R2
```

Out[97]: 0.0031383509671115695

In [98]:

```
VIF = 1 / (1 - R2)
VIF
```

Out[98]: 1.0031482312216107

In [99]:

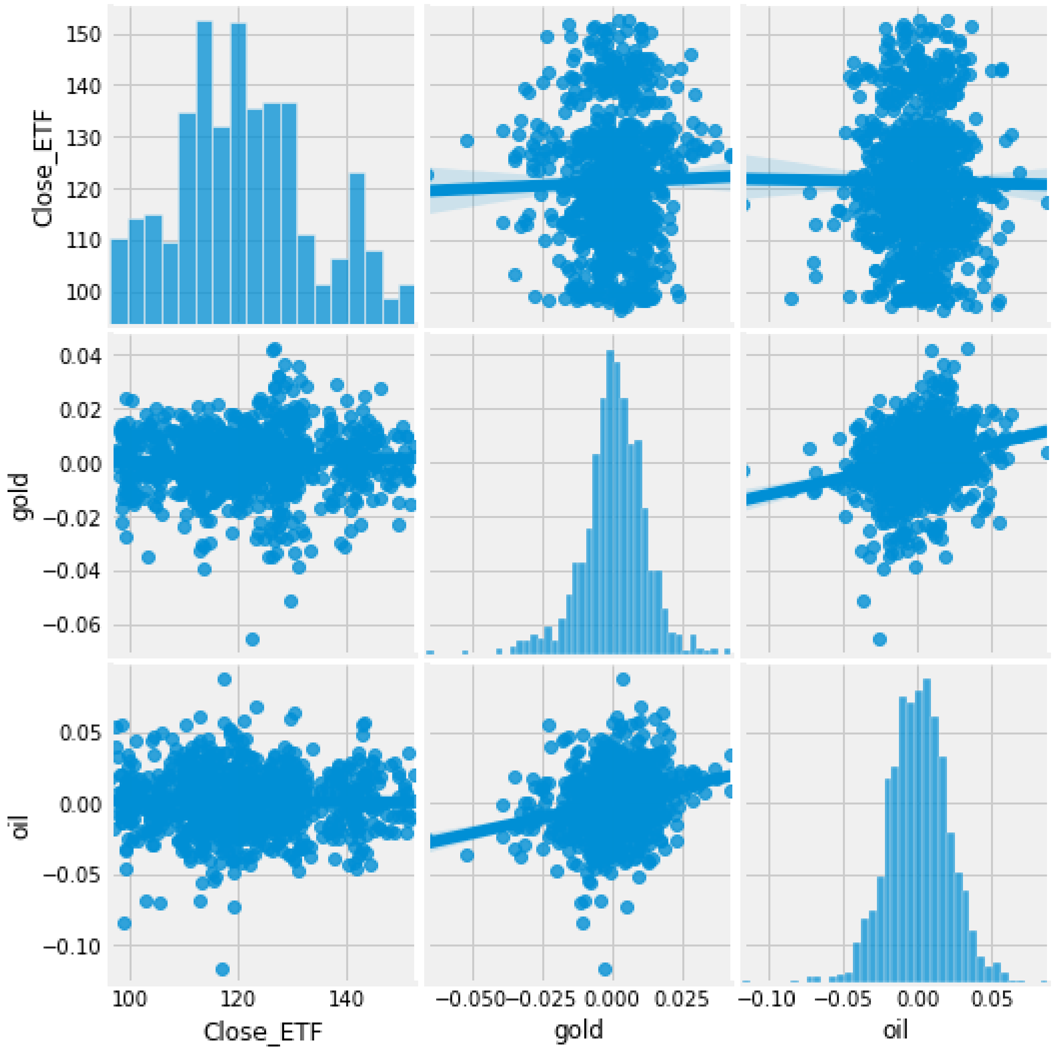
```
# Incase only one input variable and output variable
pdInputData[["Close_ETF", "gold", "oil"]].corr()
```

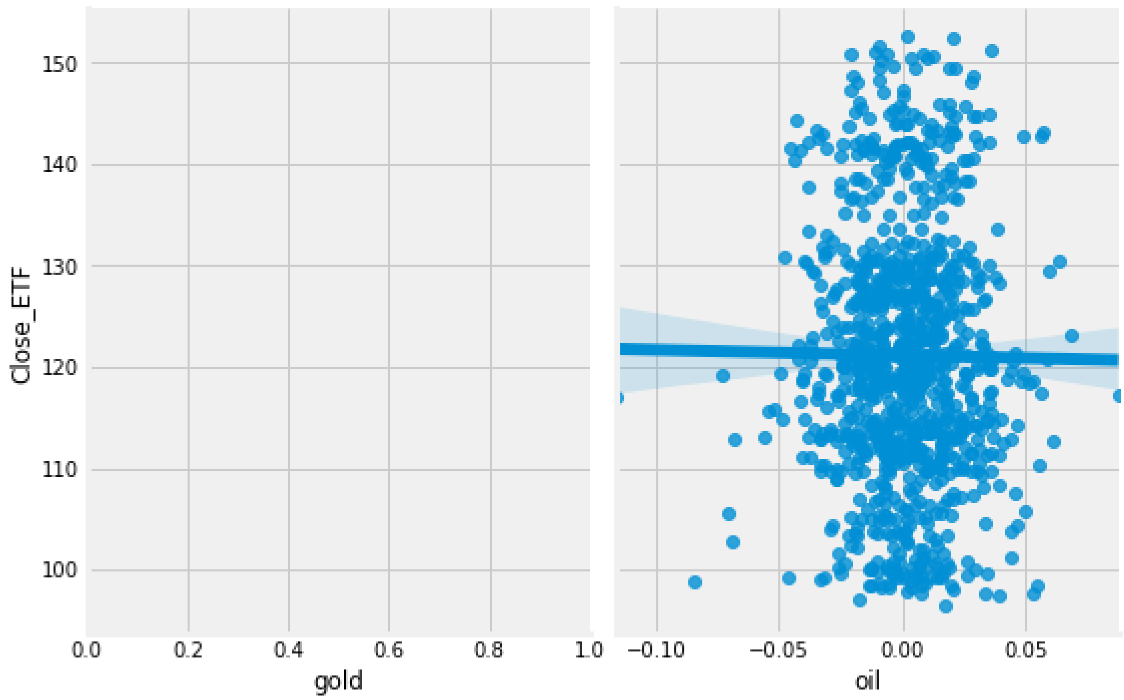
Out[99]:

	Close_ETF	gold	oil
Close_ETF	1.000000	0.022996	-0.009045
gold	0.022996	1.000000	0.235650
oil	-0.009045	0.235650	1.000000

In [100…

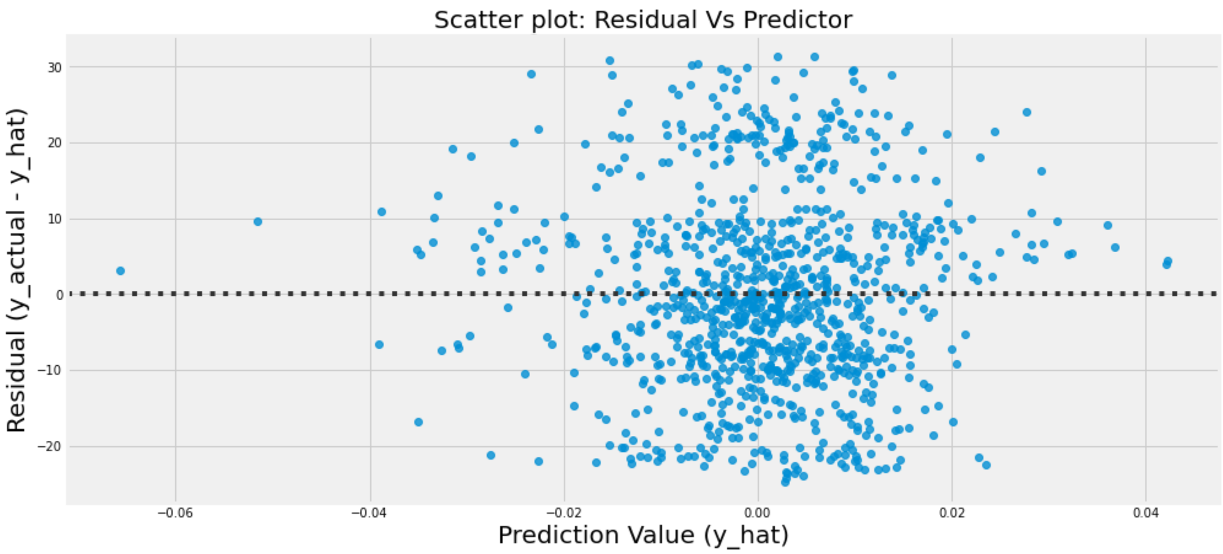
```
#Plot pairwise relationships in a dataset
sns.pairplot(pdInputData[["Close_ETF", "gold", "oil"]], kind ='reg')
sns.pairplot(pdInputData , x_vars=["gold", "oil"], y_vars=["Close_ETF"],
             height=5, aspect=.8, kind="reg");
```





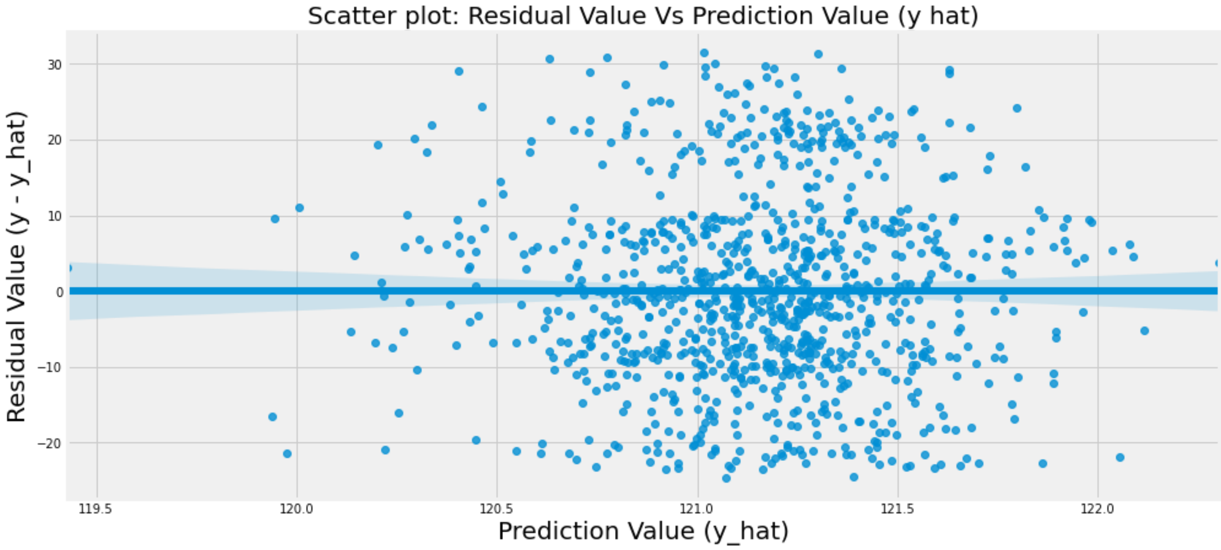
In [101…

```
plt.figure(figsize=(15,7))
sns.residplot(x = 'gold',
              y = "Close_ETF",
              data = pdInputData)
plt.xlabel("Prediction Value (y_hat)", fontsize = 20)
plt.ylabel("Residual (y_actual - y_hat)", fontsize = 20)
plt.title("Scatter plot: Residual Vs Predictor", fontsize = 20)
plt.show()
```

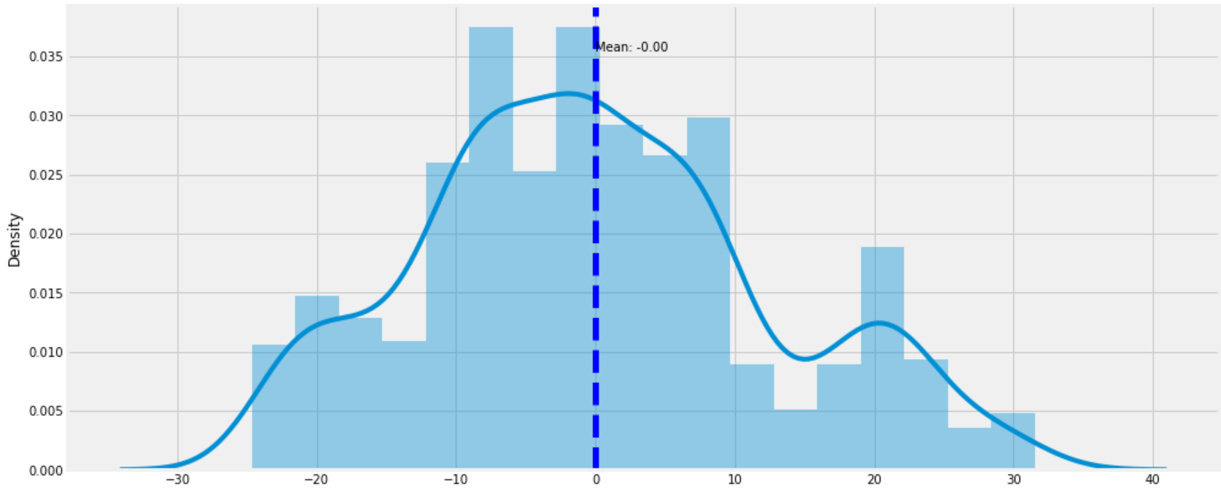


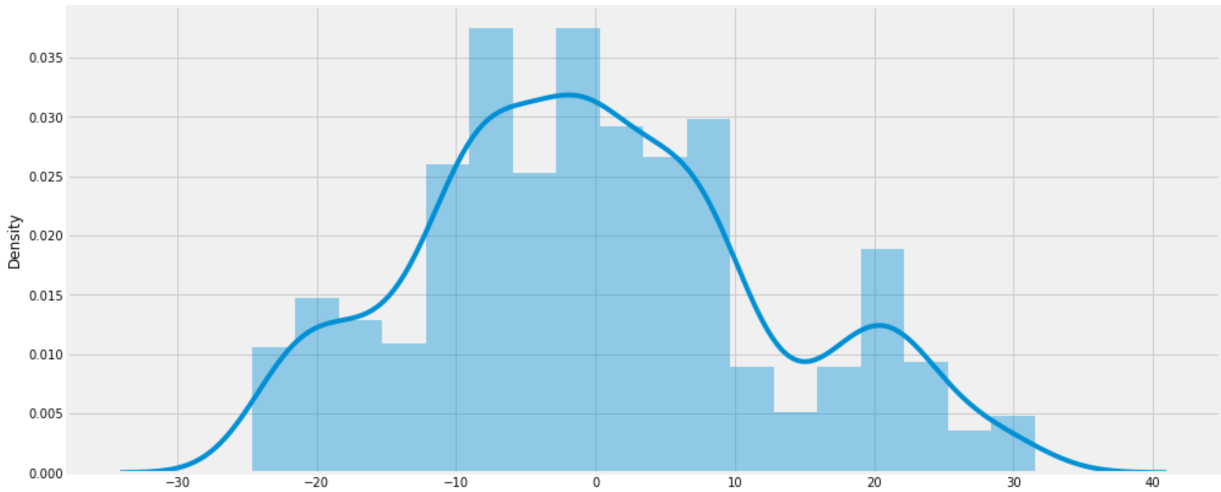
In [102…

```
X = pdInputData[["gold", "oil"]]
y = pdInputData['Close_ETF']


X_with_constant = sm.add_constant(X)
model = sm.OLS(y, X_with_constant)

results = model.fit()
results.params

# Now making prediction for the test data
# align test data for the prediction
X= sm.add_constant(X)
y_pred = results.predict(X)
serResidual = y - y_pred
plt.figure(figsize=(15,7))
sns.regplot(x=y_pred,y=serResidual)
plt.xlabel("Prediction Value (y_hat)", fontsize = 20)
plt.ylabel("Residual Value (y - y_hat)", fontsize = 20)
plt.title("Scatter plot: Residual Value Vs Prediction Value (y hat)", fontsiz
plt.show()


plt.figure(figsize=(15,7))
ax = sns.distplot(serResidual)
plt.axvline(np.mean(serResidual), color="b", linestyle="dashed", linewidth=5)
_, max_ = plt.ylim()
plt.text(        serResidual.mean() + serResidual.mean() / 10, max_ - max_ /
    )

plt.figure(figsize=(15,7))
sns.distplot(serResidual)
print("Mean of resudual is:", serResidual.mean())
```



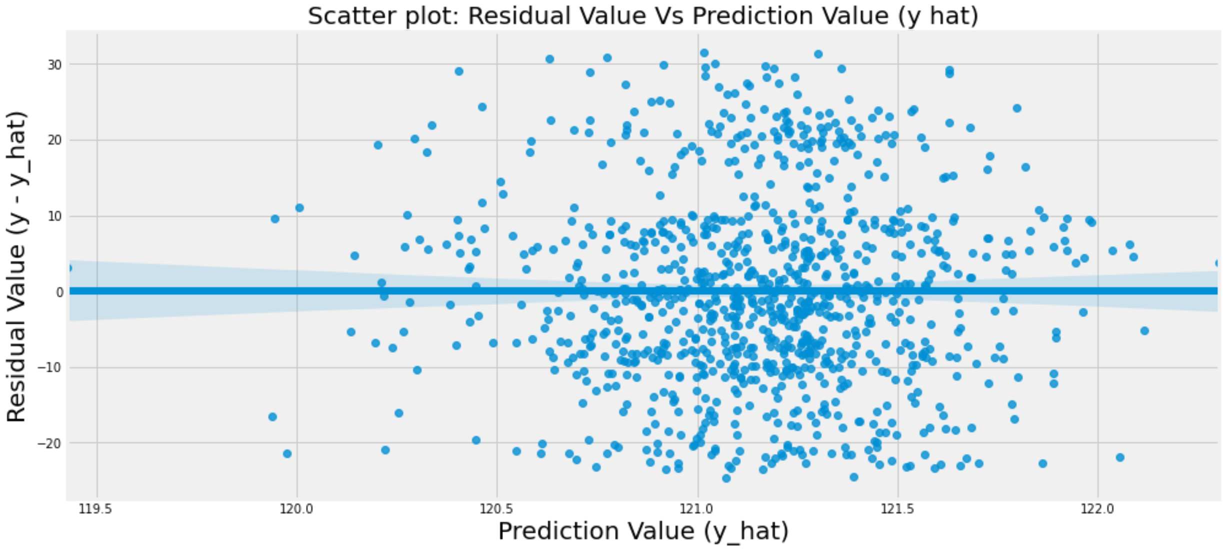

```
Mean of resudual is: -7.761968845443334e-14
```





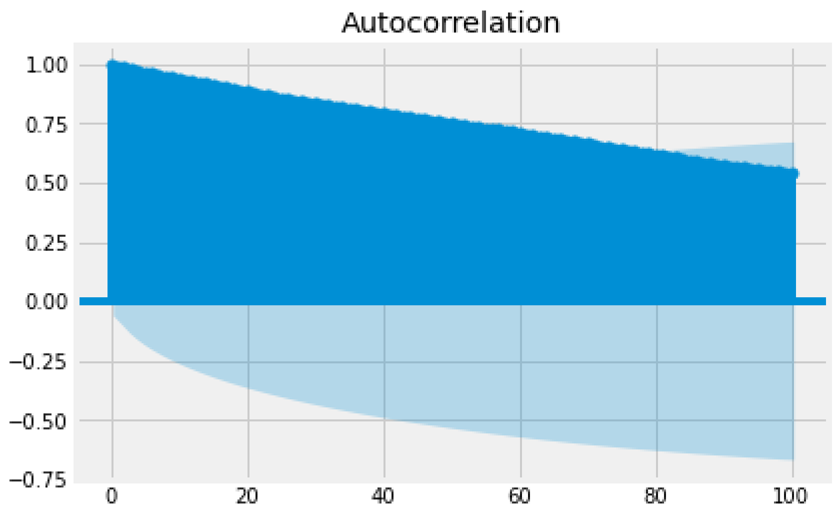
In [103…

```
plt.figure(figsize=(15,7))
sns.regplot(x=y_pred,y=serResidual)
plt.xlabel("Prediction Value (y_hat)", fontsize = 20)
plt.ylabel("Residual Value (y - y_hat)", fontsize = 20)
plt.title("Scatter plot: Residual Value Vs Prediction Value (y hat)", fontsiz
plt.show()
```



In [104…

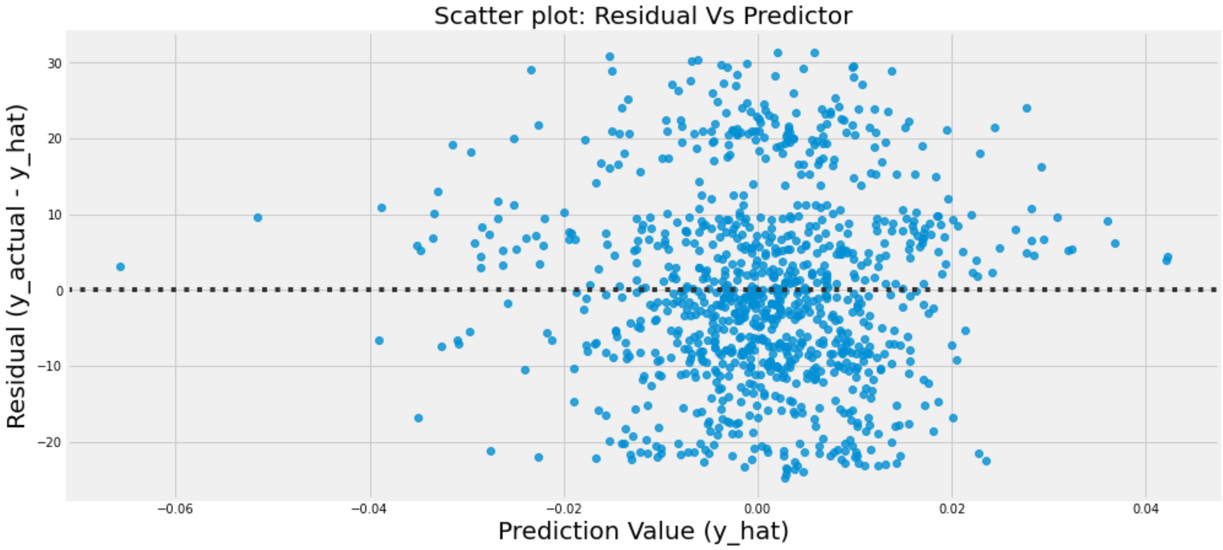
```
# In addition to above we can use following too:
acf = smt.graphics.plot_acf(serResidual, lags=100, alpha=0.05)
acf.show()
```



In [105…

```
def homoscedasticity_assumption(model, features, label):
    """
    Homoscedasticity: Assumes that the errors exhibit constant variance
    """
    print('Assumption 5: Homoscedasticity of Error Terms', '\n')

    print('Residuals should have relative constant variance')

    # Plotting the residuals

    plt.figure(figsize=(15,7))
    sns.residplot(x = 'gold',
                  y = "Close_ETF",
                  data = pdInputData)
    plt.xlabel("Prediction Value (y_hat)", fontsize = 20)
    plt.ylabel("Residual (y_actual - y_hat)", fontsize = 20)
    plt.title("Scatter plot: Residual Vs Predictor", fontsize = 20)
    plt.show()
homoscedasticity_assumption(model, pdInputData[["gold", "oil"]],
                            pdInputData["Close_ETF"])
```

```
Assumption 5: Homoscedasticity of Error Terms

Residuals should have relative constant variance
```



In [106…

```
R2 = r2_score(y, y_pred)
print(R2)

VIF = 1 / (1 - R2)
print(VIF)

#calculate VIF for each explanatory variable
def get_vif(X_features):
    pdVif = pd.DataFrame()
    pdVif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.s
    pdVif['variable'] = X.columns
    return pdVif

def get_tolerance_value(pdVif):
    pdVif = get_vif(X_features)
    return 1/pdVif['VIF']
X_features = pdInputData[["gold", "oil"]]
pdVif = get_vif(X_features)
pdVif['Tolerance'] = get_tolerance_value(pdVif)
pdVif
```

```
0.0007502966608660122
1.0007508600286383
```

Out[106…

	VIF	variable	Tolerance
0	1.004749	const	0.995273
1	1.058796	gold	0.944469
2	1.058796	oil	0.944469

In [107…

```
X = pdInputData[['gold', "oil"]]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
pdVif = pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
pdVif
```

Out[107…

	gold	oil
vif	1.059952	1.059952

In [108…

```
print("For Gold the tolerance is: ", 1/pdVif['gold']['vif'])
print("For Oil the tolerance is: ", 1/pdVif['oil']['vif'])
```

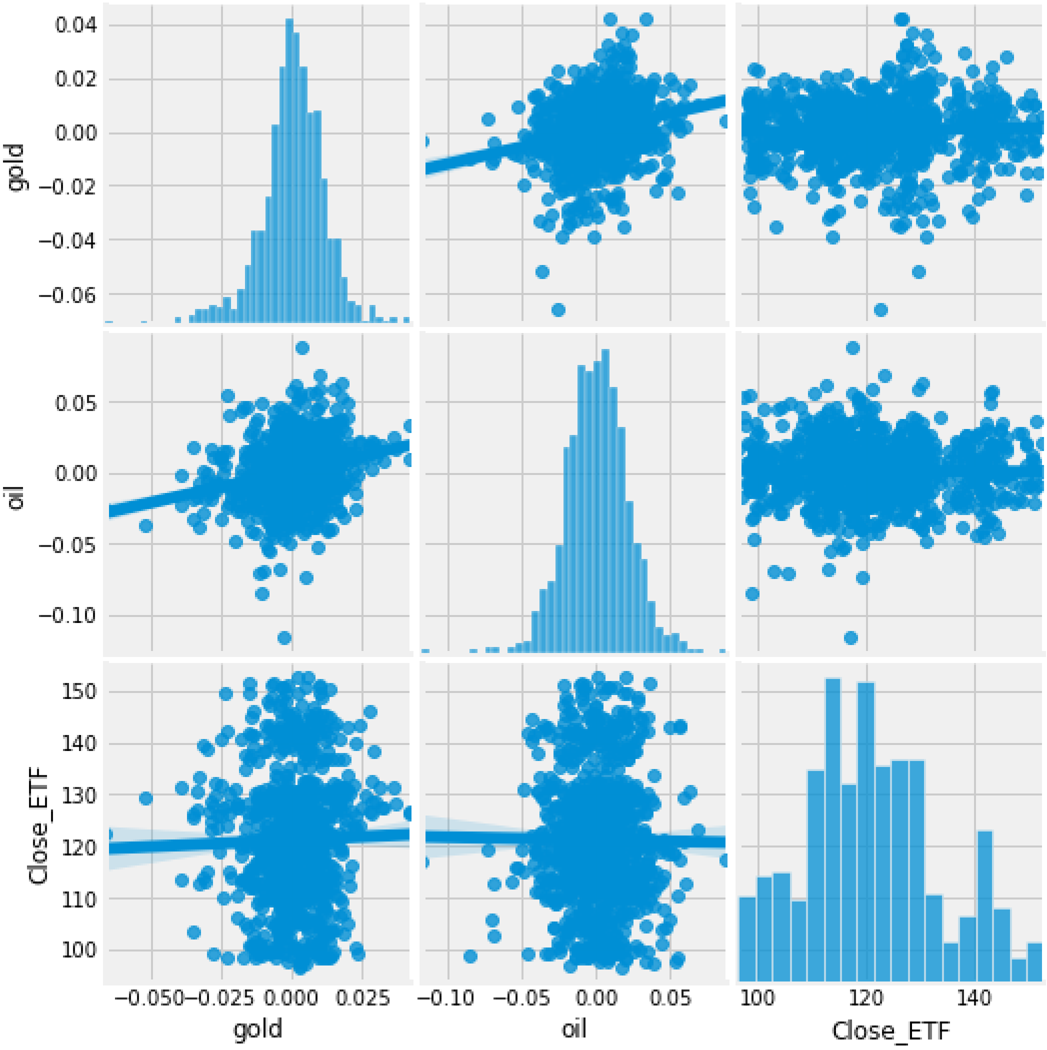
```
For Gold the tolerance is:  0.9434386482198437
For Oil the tolerance is:  0.9434386482198437
```

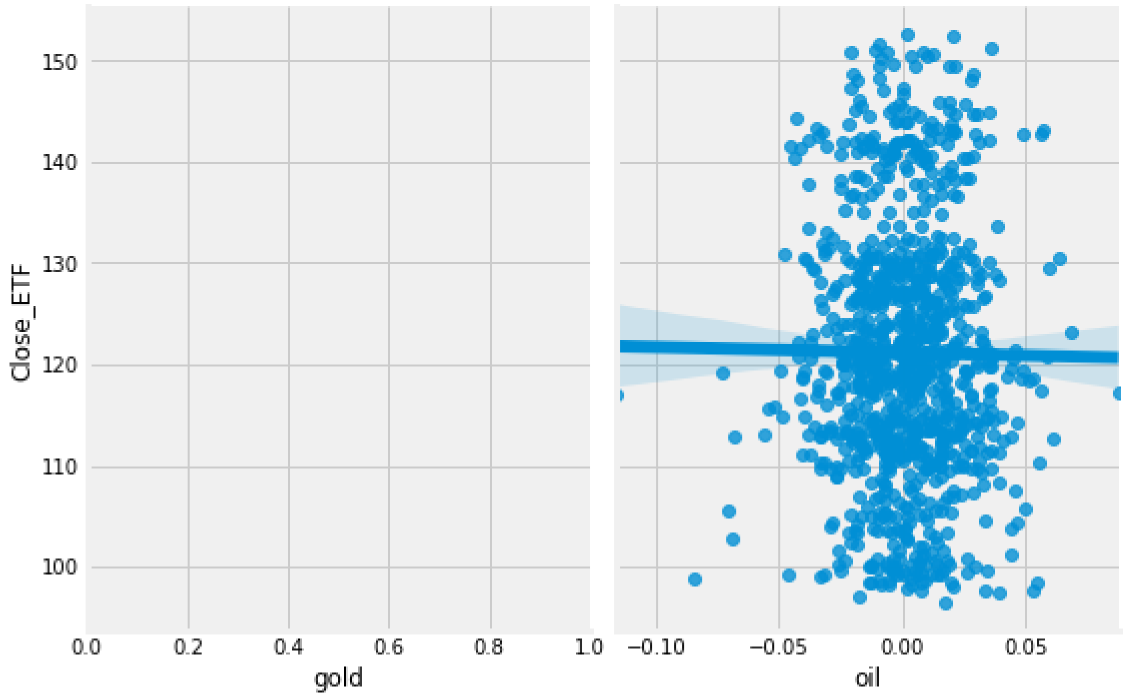
In [121…

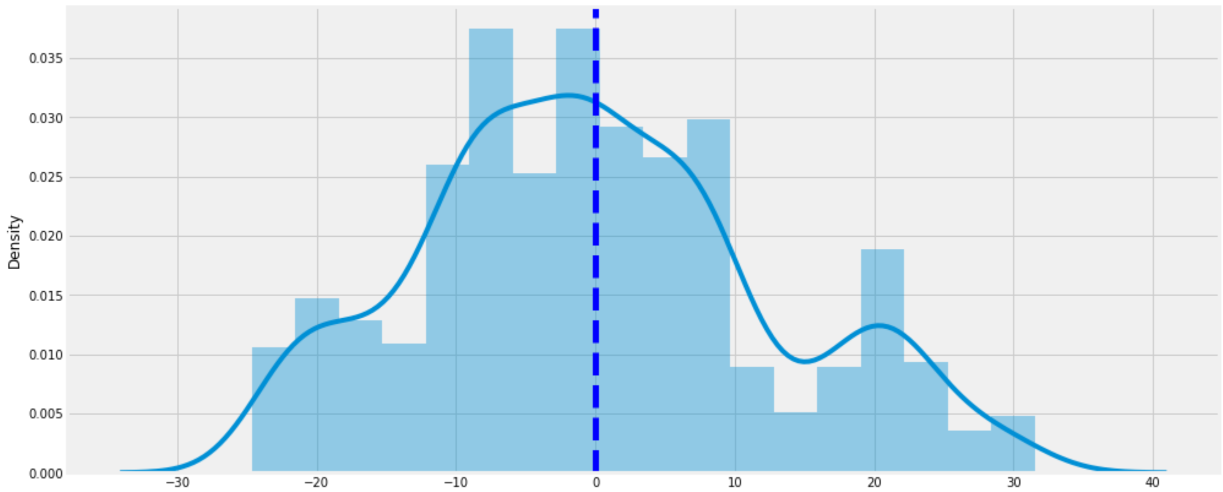
```
#part 10
#Plot  pairwise  relationships  in a  dataset .
print(sns.pairplot(pdInputData[['gold', 'oil', 'Close_ETF']] , kind ='reg') )
#Or just use first column which gives more clear picture:
sns.pairplot(pdInputData , x_vars=['gold', 'oil'],
y_vars=['Close_ETF'] , height=5, aspect=.8, kind='reg');
#Corelation between predictorsandoutputCloseETF + among the
#predictors
print(pdInputData[["Close_ETF", "gold", "oil"]]. corr())
#normality of residuals
plt.figure(figsize=(15,7))
ax = sns.distplot(serResidual)
plt.axvline(np.mean(serResidual), color="b", linestyle="dashed", linewidth=5)
max =plt.ylim()
plt.text(serResidual.mean() + serResidual.mean() / 10, max - max /10, "Mean:{
```

```
<seaborn.axisgrid.PairGrid object at 0x7fdcf97e3070>
            Close_ETF      gold       oil
Close_ETF    1.000000  0.022996 -0.009045
gold         0.022996  1.000000  0.235650
oil         -0.009045  0.235650  1.000000
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-121-2ff7090bc590> in <module>
     13 plt.axvline(np.mean(serResidual), color="b", linestyle="dashed", linew
idth=5),
     14 max =plt.ylim()
---> 15 plt.text(serResidual.mean() + serResidual.mean() / 10, max - max /10,
"Mean:{:.2f}".format(serResidual.mean()), )

TypeError: unsupported operand type(s) for /: 'tuple' and 'int'
```







In [ ]:

```
# Split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    random_state=42)


X_with_constant = sm.add_constant(X_train)
model = sm.OLS(y_train, X_with_constant)

results = model.fit()
results.params
```

In [ ]:

```
vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.sh
pd.DataFrame({'vif': vif[0:]}, index=X_train.columns).T
```

In [110…

```
X = pdInputData[['oil' , 'gold']]
y = pdInputData['Close_ETF']
X_with_constant = sm.add_constant(X)
model = sm.OLS(y, X_with_constant)
results = model.fit()
results.params
```

Out[110…

```
const     121.142725
oil        -9.126100
gold       29.622592
dtype: float64
```

In [111…
```
results.summary()
```

Out[111…

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:	Fri, 10 Dec 2021	**Prob (F-statistic):**	0.688
Time:	17:27:41	**Log-Likelihood:**	-3949.4
No. Observations:	1000	**AIC:**	7905.
Df Residuals:	997	**BIC:**	7919.
Df Model:	2		
Covariance Type:	nonrobust		

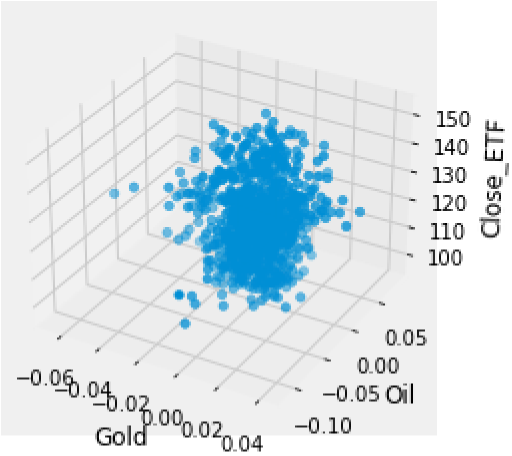
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	121.1427	0.399	303.856	0.000	120.360	121.925
oil	-9.1261	19.413	-0.470	0.638	-47.221	28.968
gold	29.6226	36.272	0.817	0.414	-41.555	100.800

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.

In [112…
```
fig = plt.figure(1)
ax = fig.add_subplot(111, projection='3d')
#ax.scatter(X[:, 0], X[:, 1], Y)
ax.scatter(pdInputData['gold'], pdInputData["oil"], pdInputData['Close_ETF'])
ax.set_xlabel('Gold')
ax.set_ylabel('Oil')
ax.set_zlabel('Close_ETF')
```

Out[112…
```
Text(0.5, 0, 'Close_ETF')
```



In [113…

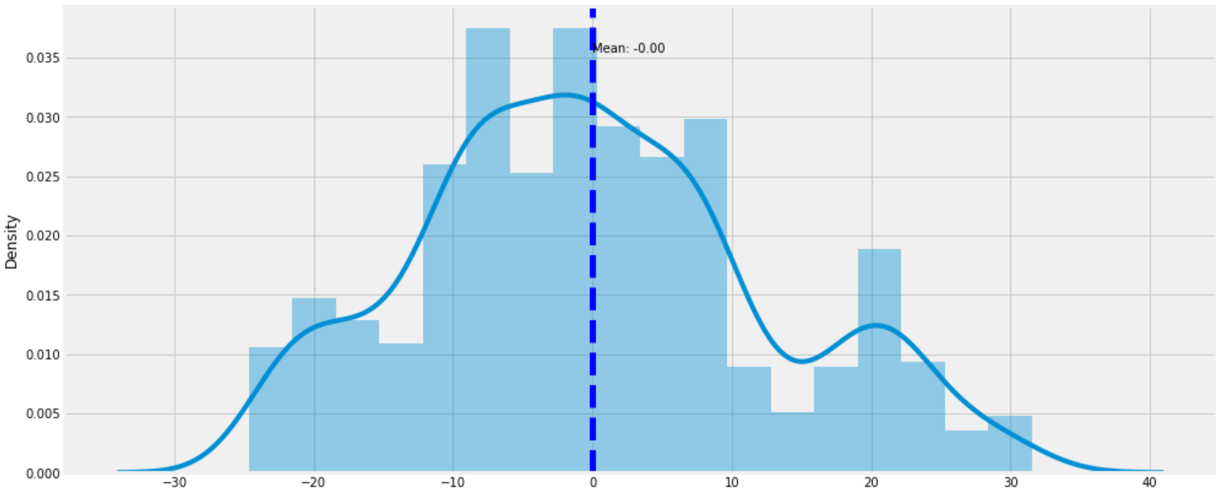
```
y_pred = results.predict()
# multicolinearity/independence

vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
pd.DataFrame({'vif': vif[0:]}, index=X.columns).T
```

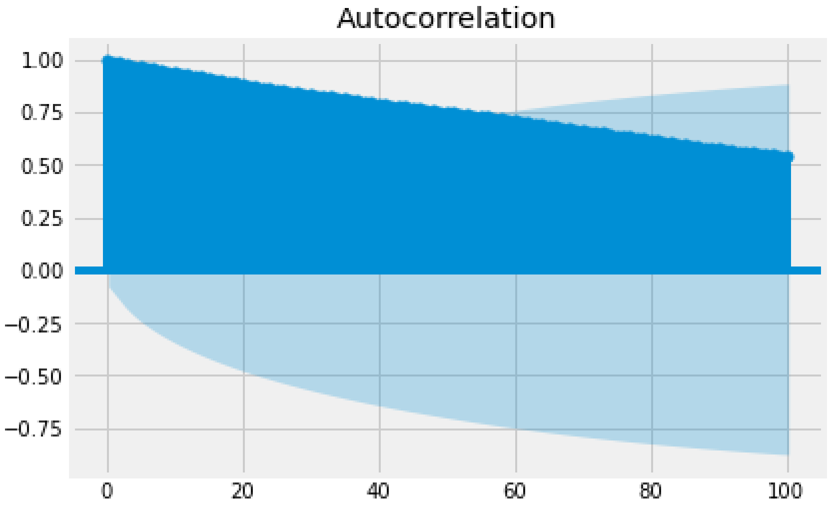
Out[113…

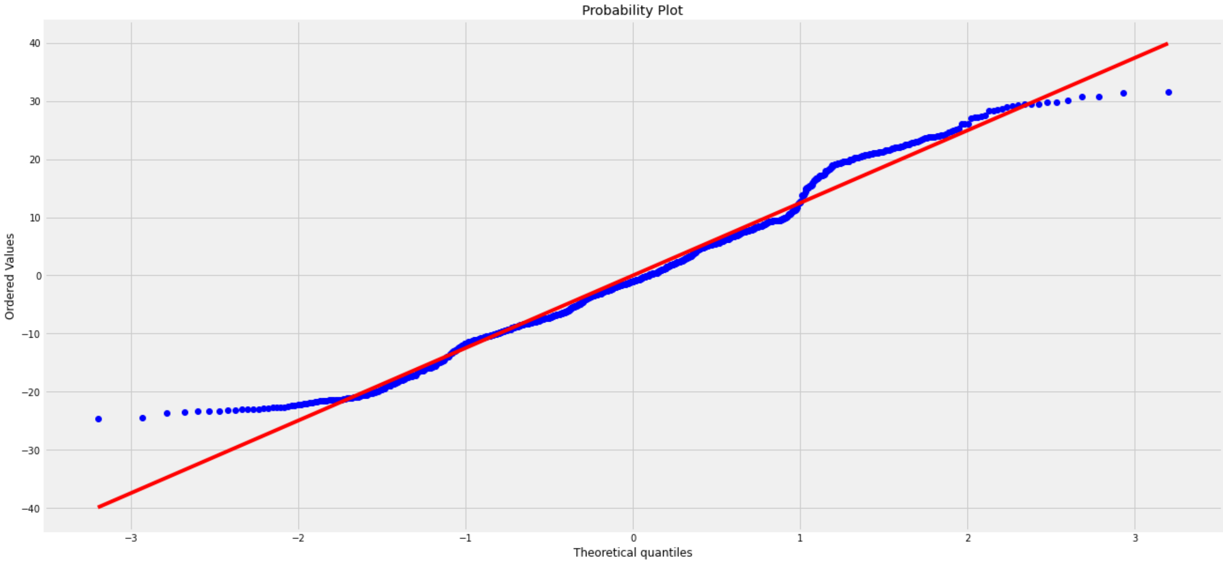
	oil	gold
vif	1.059952	1.059952

In [114…

```
# normality of residuals
import scipy
plt.figure(figsize=(15,7))
serResidual = results.resid
ax = sns.distplot(serResidual)
plt.axvline(np.mean(serResidual), color="b", linestyle="dashed", linewidth=5)
_, max_ = plt.ylim()
plt.text(        serResidual.mean() + serResidual.mean() / 10, max_ - max_ /
    )
acf = smt.graphics.plot_acf(serResidual, lags=100, alpha=0.01)

fig, ax = plt.subplots(figsize=(20,10))
_, (__, ___, r) = scipy.stats.probplot(serResidual, plot=ax, fit=True)
```







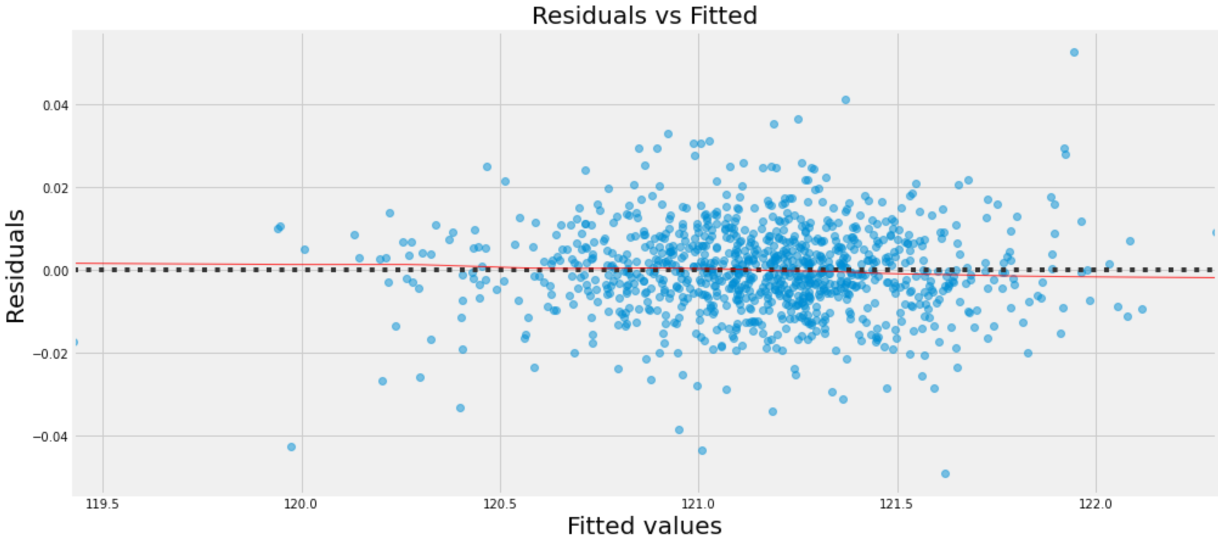
In [115…
```
np.mean(serResidual)
```

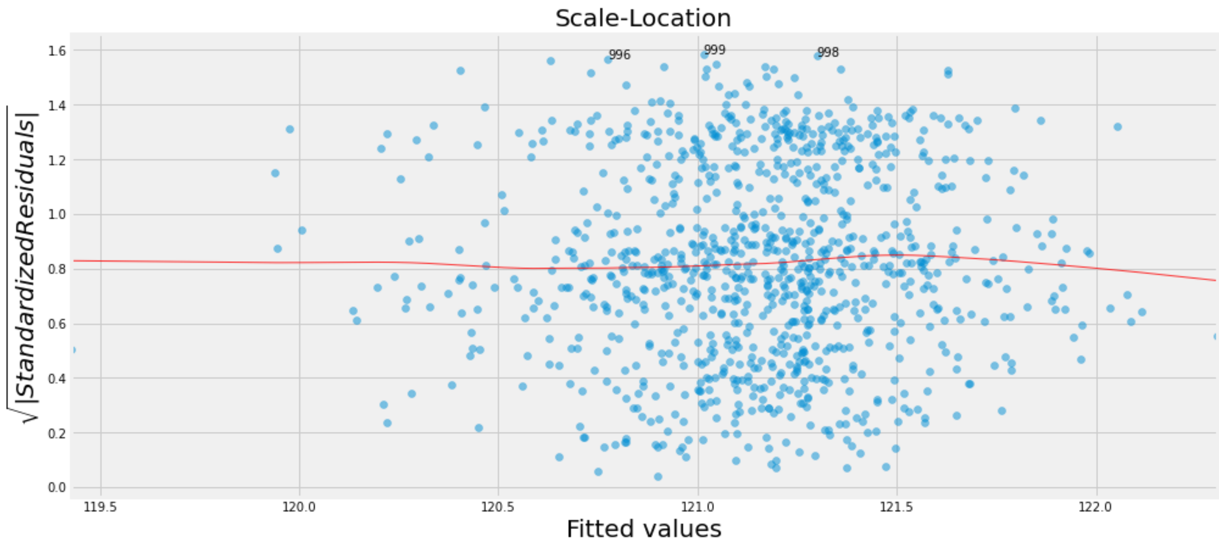
Out[115… -7.258904588525183e-14

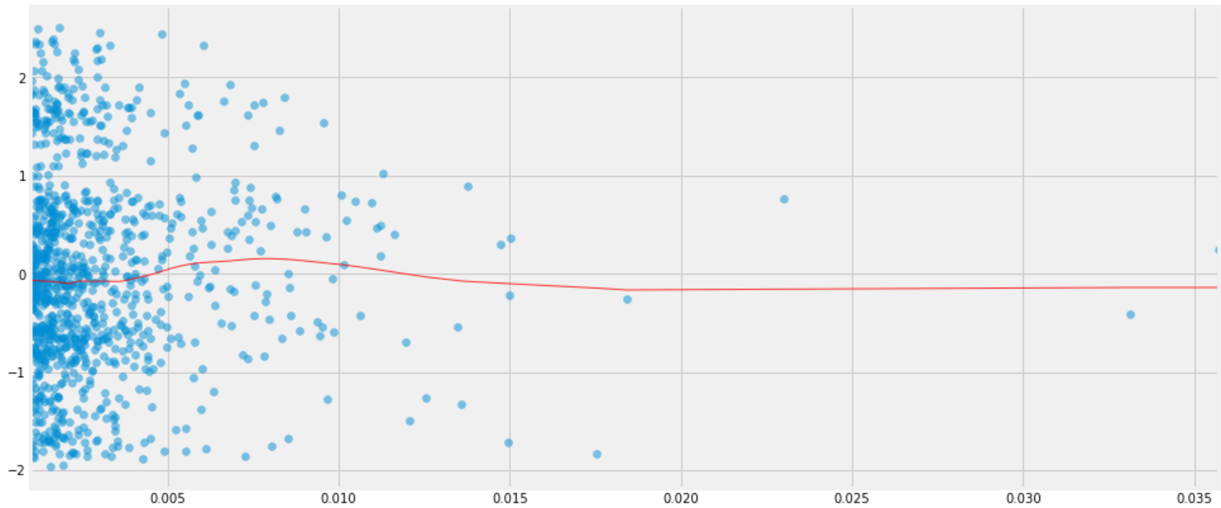
In [116…
```
# Residuals vs Fitted
model_fitted_y = results.predict()
model_residuals = results.resid
model_norm_residuals = results.get_influence().resid_studentized_internal
model_norm_residuals_abs_sqrt = np.sqrt(np.abs(model_norm_residuals))
model_abs_resid = np.abs(model_residuals)
model_leverage = results.get_influence().hat_matrix_diag
model_cooks = results.get_influence().cooks_distance[0]


plot_lm_1 = plt.figure(figsize=(15,7))
plot_lm_1.axes[0] = sns.residplot(model_fitted_y, pdInputData.columns[-1], \
                          data=pdInputData,
                          lowess=True,
                          scatter_kws={'alpha': 0.5},
                          line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})

plot_lm_1.axes[0].set_title('Residuals vs Fitted', size = 20)
plot_lm_1.axes[0].set_xlabel('Fitted values', size = 20)
plot_lm_1.axes[0].set_ylabel('Residuals', size = 20)


plot_lm_3 = plt.figure(figsize=(15,7))
plt.scatter(model_fitted_y, model_norm_residuals_abs_sqrt, alpha=0.5);
```

```
sns.regplot(model_fitted_y, model_norm_residuals_abs_sqrt,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8});
plot_lm_3.axes[0].set_title('Scale-Location', size = 20)
plot_lm_3.axes[0].set_xlabel('Fitted values', size = 20)
plot_lm_3.axes[0].set_ylabel('$\sqrt{|Standardized Residuals|}$', size = 20);

# annotations
abs_sq_norm_resid = np.flip(np.argsort(model_norm_residuals_abs_sqrt), 0)
#abs_norm_resid_top_3 = abs_norm_resid[:3]
abs_sq_norm_resid_top_3 = abs_sq_norm_resid[:3]
for i in abs_sq_norm_resid_top_3:
    plot_lm_3.axes[0].annotate(i,
                               xy=(model_fitted_y[i],
                                   model_norm_residuals_abs_sqrt[i]));


plot_lm_4 = plt.figure(figsize=(15,7))
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
sns.regplot(model_leverage, model_norm_residuals,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
plot_lm_4.axes[0].set_xlim(0, max(model_leverage)+0.01)
plot_lm_4.axes[0].set_ylim(-3, 5)
plot_lm_4.axes[0].set_title('Residuals vs Leverage', size = 20)
plot_lm_4.axes[0].set_xlabel('Leverage', size = 20)
plot_lm_4.axes[0].set_ylabel('Standardized Residuals', size = 20)

# annotations
leverage_top_3 = np.flip(np.argsort(model_cooks), 0)[:3]
for i in leverage_top_3:
    plot_lm_4.axes[0].annotate(i,
                               xy=(model_leverage[i],
                                   model_norm_residuals[i]))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-116-499c0b522c37> in <module>
     49                  lowess=True,
     50                  line_kws={'color': 'red', 'lw': 1, 'alpha': 0.8})
---> 51 plot_lm_4.axes[0].set_xlim(0, max(model_leverage)+0.01)
     52 plot_lm_4.axes[0].set_ylim(-3, 5)
     53 plot_lm_4.axes[0].set_title('Residuals vs Leverage', size = 20)

TypeError: 'tuple' object is not callable
```


Residuals vs Fitted
Residuals
0.04
0.02
0.00
−0.02
−0.04
119.5
120.0
120.5
121.0
121.5
122.0
Fitted values


Scale-Location
$\sqrt{|Standardized Residuals|}$
996
999
998
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
119.5
120.0
120.5
121.0
121.5
122.0
Fitted values


2
1
0
−1
−2
0.005
0.010
0.015
0.020
0.025
0.030
0.035

In [ ]:
```
#END
```