# MA541 - Statistical Methods

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import pandas as pd
import seaborn as sb
import os

import warnings
warnings.filterwarnings("ignore")

import random

random.seed(1234)

%matplotlib inline
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [5]:
```
!ls
```

```
drive  sample_data
```

In [84]:
```
path = '/content/MA 541 Course Project Data (1).xlsx'
df = pd.read_excel(path)
df.head()
```

Out[84]:

| | 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]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Close_ETF  1000 non-null   float64
 1   oil        1000 non-null   float64
 2   gold       1000 non-null   float64
 3   JPM        1000 non-null   float64
dtypes: float64(4)
memory usage: 31.4 KB
```

In [7]:
```
```

# 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 [8]:
```
df.describe()
```

Out[8]:

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

In [9]:
```
df.corr()
```

Out[9]:

| | 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

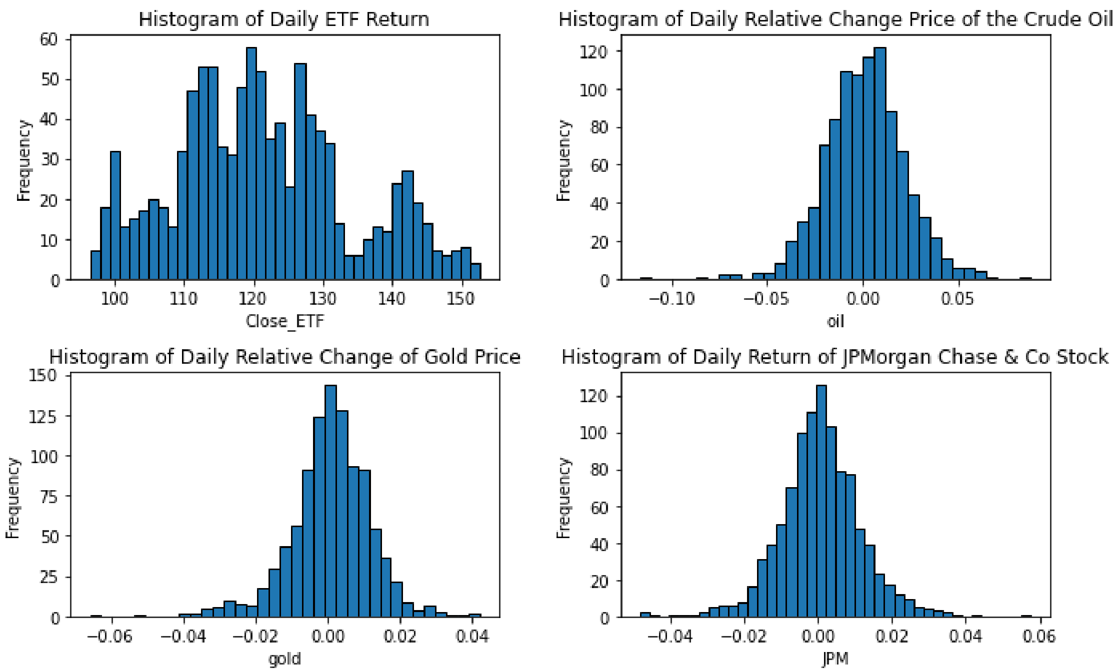
**Requirements** – Use any software to draw the following plots:

1. A histogram for each column (hint: four histograms total)
2. A time series plot for each column (hint: use the series "1, 2, 3, ..., 1000" as the horizontal axis; four plots total)
3. A time series plot for all four columns (hint: one plot including four "curves" and each "curve" describes one column)
4. 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

## Histograms

In [10]:
```
plt.figure(figsize = [10, 6])

# Close_ETF
plt.subplot(2,2,1);
x = 'Close_ETF'
plt.hist(data = df, x = x, edgecolor='black', bins=40);
plt.title(f'Histogram of Daily ETF Return')
plt.xlabel(x)
plt.ylabel('Frequency')

# oil
plt.subplot(2,2,2);
x = 'oil'
plt.hist(data = df, x = x, edgecolor='black', bins=35);
plt.title(f'Histogram of Daily Relative Change Price of the Crude Oil')
plt.xlabel(x)
```

```python
plt.ylabel('Frequency')

# gold
plt.subplot(2,2,3);
x = 'gold'
plt.hist(data = df, x = x, edgecolor='black', bins=35);
plt.title(f'Histogram of Daily Relative Change of Gold Price')
plt.xlabel(x)
plt.ylabel('Frequency')

# JPM
plt.subplot(2,2,4);
x = 'JPM'
plt.hist(data = df, x = x, edgecolor='black', bins=40);
plt.title(f'Histogram of Daily Return of JPMorgan Chase & Co Stock')
plt.xlabel(x)
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

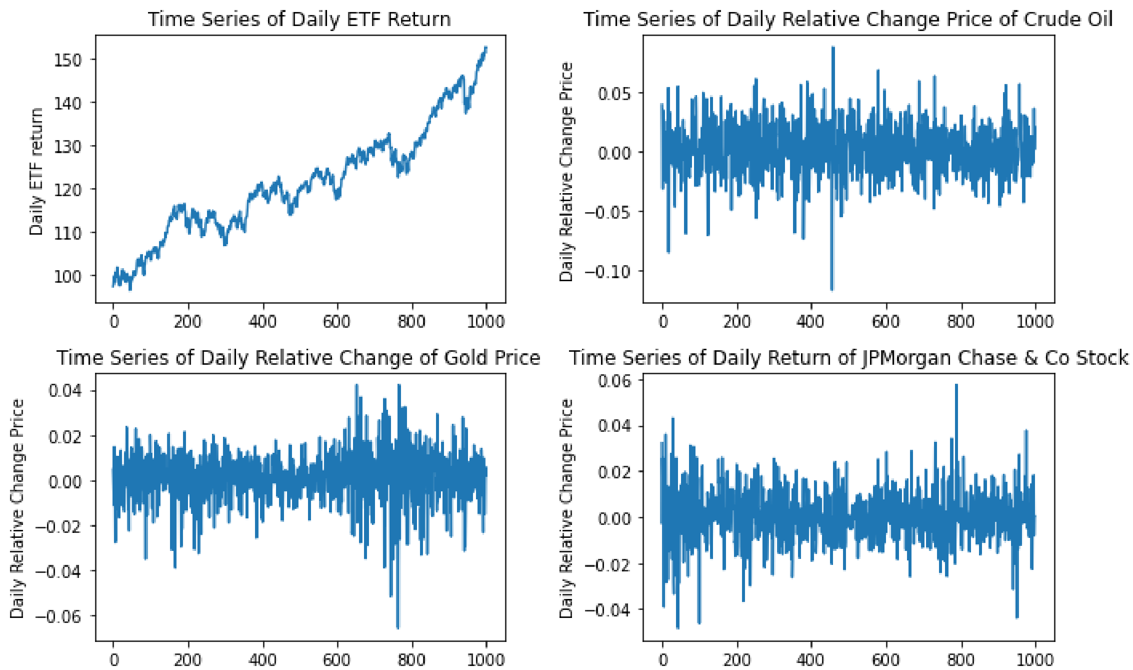



## Time Series Plots

In [11]:

```python
plt.figure(figsize = [10, 6])

# Close_ETF
plt.subplot(2,2,1);
x = 'Close_ETF'
plt.plot(df[x]);
plt.title(f'Time Series of Daily ETF Return')
plt.ylabel('Daily ETF return')

# oil
plt.subplot(2,2,2);
x = 'oil'
plt.plot(df[x]);
plt.title(f'Time Series of Daily Relative Change Price of Crude Oil')
plt.ylabel('Daily Relative Change Price')

# gold
plt.subplot(2,2,3);
x = 'gold'
plt.plot(df[x]);
plt.title(f'Time Series of Daily Relative Change of Gold Price')
plt.ylabel('Daily Relative Change Price')
```

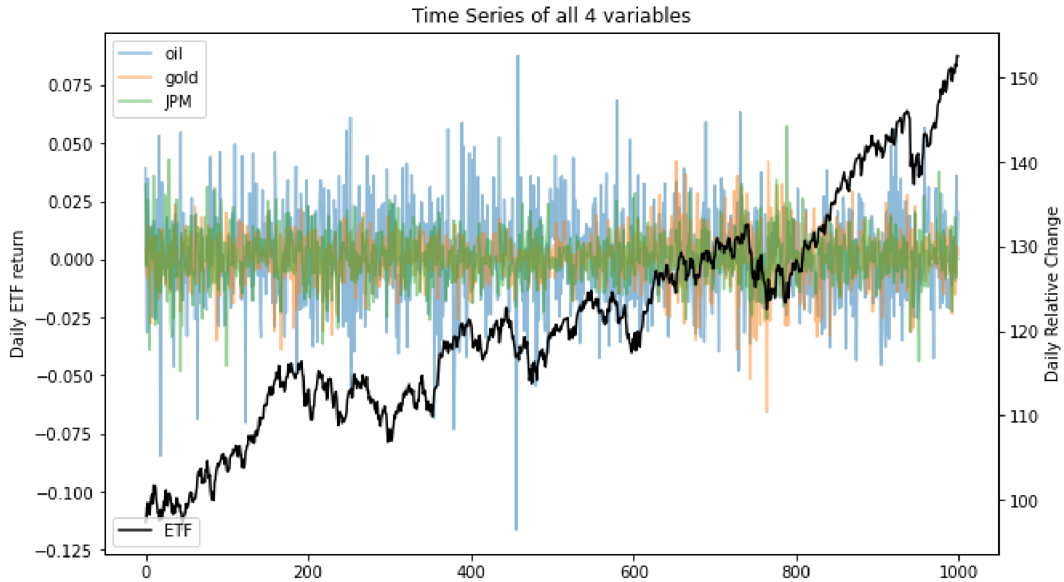
```python
plt.subplot(2,2,4);
x = 'JPM'
plt.plot(df[x]);
plt.title(f'Time Series of Daily Return of JPMorgan Chase & Co Stock')
plt.ylabel('Daily Relative Change Price')

plt.tight_layout()
plt.show()
```




# Time Series Plot with all the series in one Plot

In [12]:

```python
fig ,ax = plt.subplots(figsize=(10,6));

# oil
x = 'oil'
ax.plot(df[x], alpha = 10/20);

# gold
x = 'gold'
ax.plot(df[x], alpha = 10/20);

# JPM
x = 'JPM'
ax.plot(df[x], alpha = 10/20);
ax.set_ylabel('Daily ETF return')
ax.legend(['oil','gold','JPM'], loc=2)
# 2nd axis
x = 'Close_ETF'

ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(df[x], color = 'black');
ax2.set_ylabel("Daily ETF Return")

plt.legend(['ETF'], loc=3)

plt.title(f'Time Series of all 4 variables')
plt.ylabel('Daily Relative Change')

plt.show()
```

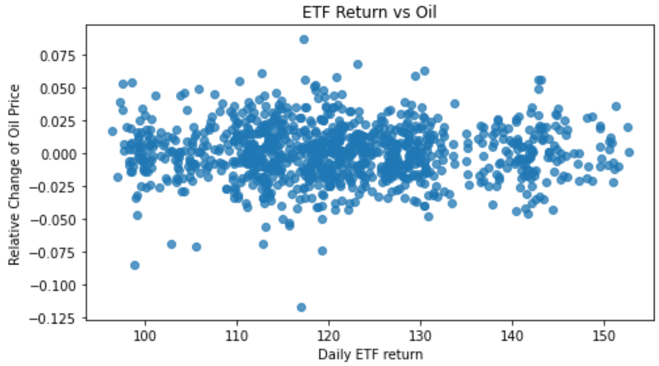


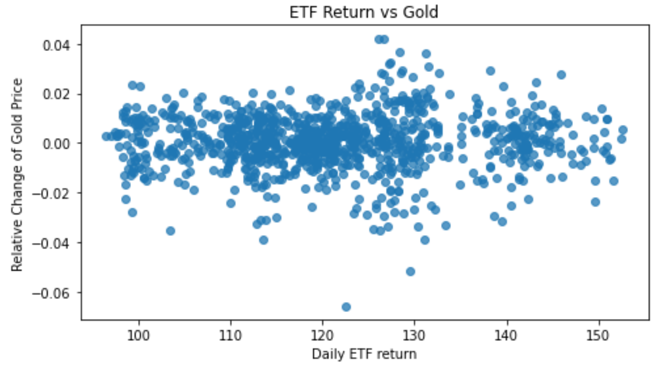

## Scatter plots to describe the relationships

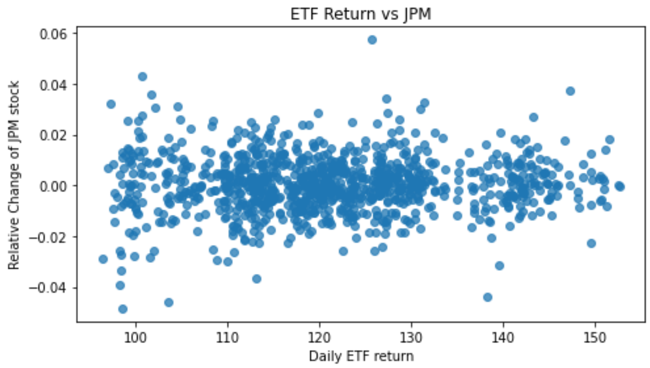
- between the ETF column and the OIL column
- between the ETF column and the GOLD column
- between the ETF column and the JPM column

In [13]:

```
plt.figure(figsize = [14, 8])
# Oil
plt.subplot(2,2,1);
x = 'Close_ETF'
y = 'oil'
plt.scatter(data = df, x = x, y = y, alpha=15/20);
plt.title(f'ETF Return vs Oil')
plt.ylabel('Relative Change of Oil Price')
plt.xlabel('Daily ETF return')

# Gold
plt.subplot(2,2,2);
x = 'Close_ETF'
y = 'gold'
plt.scatter(data = df, x = x, y = y, alpha=15/20);
plt.title(f'ETF Return vs Gold')
plt.ylabel('Relative Change of Gold Price')
plt.xlabel('Daily ETF return')

#JPM
plt.subplot(2,2,3);
x = 'Close_ETF'
y = 'JPM'
plt.scatter(data = df, x = x, y = y, alpha=15/20);
plt.title(f'ETF Return vs JPM')
plt.ylabel('Relative Change of JPM stock')
plt.xlabel('Daily ETF return')

plt.tight_layout()
plt.show()
```



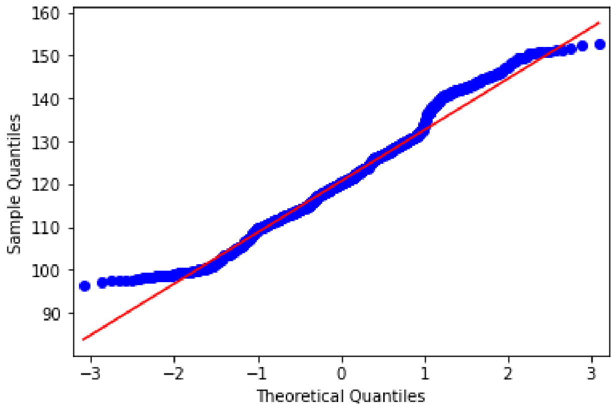



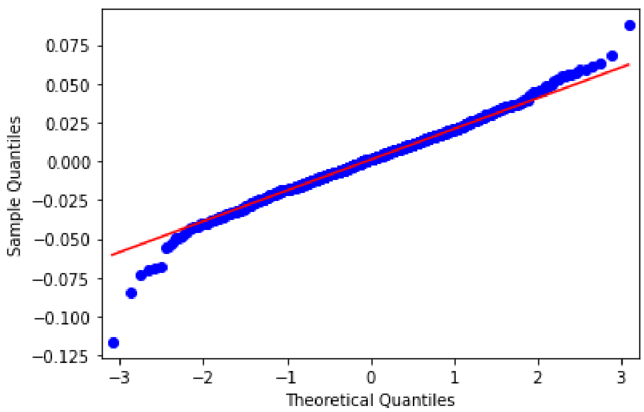

# Part 3: Distribution of the data

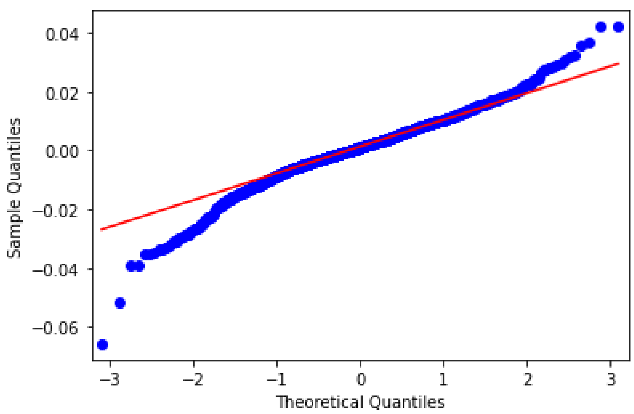
**Requirements** – Propose an assumption/a hypothesis regarding the type of distribution each column of the data set may follow (i.e., the ETF, OIL, GOLD, and JPM column), based on the plots from Part 2. Then verify or object that assumption/hypothesis with appropriate tests (for example, normality test).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

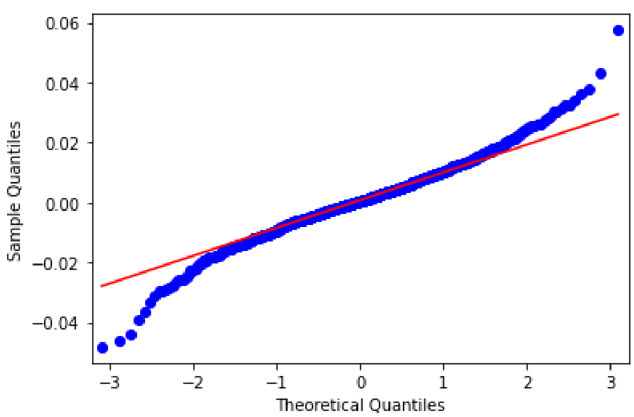
In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

In [15]:
```
# Close_ETF
# plt.subplot(2,2,1);
x = 'Close_ETF'
sm.qqplot(df[x], line = 'q');


# oil
# plt.subplot(2,2,2);
x = 'oil'
sm.qqplot(df[x], line = 'q');
# plt.title(f'Time Series of Daily ETF Return')
# plt.ylabel('Daily ETF return')


# gold
# plt.subplot(2,2,3);
x = 'gold'
sm.qqplot(df[x], line = 'q');
# plt.title(f'Time Series of Daily ETF Return')
# plt.ylabel('Daily ETF return')


# plt.subplot(2,2,4);
x = 'JPM'
sm.qqplot(df[x], line = 'q');
# plt.title(f'Time Series of Daily ETF Return')
# plt.ylabel('Daily ETF return')
```




## Normality Test

In [16]:

```
from scipy.stats import kstest
```

In [17]:

```
def kstest_(col):
  statistic, pvalue = kstest(col, 'norm')
```

```python
print("ks statistic={}, pvalue={}".format(statistic, pvalue))
```

> We do not believe that the Daily ETF Return follows a normal distribution base on its histrogram
>
> $H_0$ : Daily ETF Return Follows Normal Distribution
>
> $H_a$ : Daily ETF Return DOES NOT Follow Normal Distribution

In [18]:

```python
x = 'Close_ETF'
print(x)
kstest_(df[x])
# Rejecting the null hypthesis in the 0.01 alpha level in favor of the
# alternative hypothesis that daily etf return DOES NOT follow a normal
# distribution.
```

```
Close_ETF
ks statistic=1.0, pvalue=0.0
```

> We believe that the oil daily change follows a normal distribution based on the observed histogram
>
> $H_0$ : Oil daily changes Follows Normal Distribution
>
> $H_a$ : Oil daily changes DOES NOT Follow Normal Distribution

In [19]:

```python
x = 'oil'
print(x)
kstest_(df[x])
# Rejecting the null hypthesis in the 0.01 alpha level in favor of the
# alternative hypothesis that daily
# changes of oil price DOES NOT follow a normal distribution
```

```
oil
ks statistic=0.4727185265212217, pvalue=1.2565304659417615e-205
```

> We believe that the gold daily price change follows a normal distribution based on the observed histogram
>
> $H_0$ : gold daily changes Follows Normal Distribution
>
> $H_a$ : gold daily changes DOES NOT Follow Normal Distribution

In [20]:

```python
x = 'gold'
kstest_(df[x])
# Rejecting the null hypthesis in the 0.01 alpha level in favor of the
# alternative hypothesis that daily
# changes of gold price DOES NOT follow a normal distribution
```

```
ks statistic=0.48333847934283236, pvalue=1.4922487931964242e-215
```

> We believe that the JPM daily price change follows a normal distribution based on the observed histogram
>
> $H_0$ : JPM daily changes Follows Normal Distribution
>
> $H_a$ : JPM daily changes DOES NOT Follow Normal Distribution

In [21]:

```python
x = 'JPM'
kstest_(df[x])
# Rejecting the null hypthesis in the 0.01 alpha level in favor of the
# alternative hypothesis that daily
# changes of JPM price DOES NOT follow a normal distribution
```

```
ks statistic=0.4829776270752645, pvalue=3.278870501508125e-215
```

# Other Normality Tests: Normality Test based on based on skewness and kurtosis

In [22]:
```
from scipy.stats import shapiro
```

In [23]:
```
def shapiro_(col):
  statistic, pvalue = shapiro(col)
  print("shapiro-wilk statistic={}, pvalue={}".format(statistic, pvalue))
```

In [24]:
```
x = 'Close_ETF'
shapiro_(df[x])
x = 'oil'
shapiro_(df[x])
x = 'gold'
shapiro_(df[x])
x = 'JPM'
shapiro_(df[x])
```

```
shapiro-wilk statistic=0.9795047640800476, pvalue=1.1655147680311728e-10
shapiro-wilk statistic=0.9886544346809387, pvalue=5.487161729433865e-07
shapiro-wilk statistic=0.9692050814628601, pvalue=1.0206207623833508e-13
shapiro-wilk statistic=0.9798560738563538, pvalue=1.5373954886932495e-10
```

In [25]:
```
from scipy.stats import normaltest
```

In [26]:
```
def normaltest_(col):
  statistic, pvalue = normaltest(col)
  print("normaltest statistic={}, pvalue={}".format(statistic, pvalue))
```

In [27]:
```
x = 'Close_ETF'
normaltest_(df[x])
x = 'oil'
normaltest_(df[x])
x = 'gold'
normaltest_(df[x])
x = 'JPM'
normaltest_(df[x])
```

```
normaltest statistic=27.147577721224625, pvalue=1.2734397418438873e-06
normaltest statistic=41.4478074658443, pvalue=9.993623074366447e-10
normaltest statistic=105.7598369986937, pvalue=1.0827873971023125e-23
normaltest statistic=52.29823459337726, pvalue=4.4013170216572694e-12
```

In [28]:
```
from scipy import stats

x = 'Close_ETF'
print('Skewness is {0} and Kurtosis is {1}'.format(stats.skew(df[x]), stats.kurtosis(df[x], fisher=False)))
x = 'oil'
print('Skewness is {0} and Kurtosis is {1}'.format(stats.skew(df[x]), stats.kurtosis(df[x], fisher=False)))
x = 'gold'
print('Skewness is {0} and Kurtosis is {1}'.format(stats.skew(df[x]), stats.kurtosis(df[x], fisher=False)))
x = 'JPM'
print('Skewness is {0} and Kurtosis is {1}'.format(stats.skew(df[x]), stats.kurtosis(df[x], fisher=False)))
```

```
Skewness is 0.30598669028843434 and Kurtosis is 2.5710948445680577
Skewness is -0.15618809219158547 and Kurtosis is 4.563030265583717
Skewness is -0.5232400185875281 and Kurtosis is 5.557555193051716
Skewness is 0.011025631535947643 and Kurtosis is 5.0928717453849695
```

> **All statistical tests indicated that NONE of the variables were normally distributed**

# Part 4: Breaking data into small groups and to discuss the importance of the Central Limit Theorem

**Requirements** – Consider the ETF column (1000 values) as the population (x), and do the follows. Any software may be used.

1. Calculate the mean $\mu_x$ and the standard deviation $\sigma_x$ of the population.
2. Break the population into 50 groups **sequentially** and each group includes 20 values.
3. Calculate the sample mean ($\bar{x}$) 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 $\mu_{\bar{x}}$ and the standard deviation ($\sigma_{\bar{x}}$) of the data including these sample means. Make a comparison between $\mu_x$ and $\mu_{\bar{x}}$ , between $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ . 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.

## 1. Calculate the mean $\mu_x$ and the standard deviation $\sigma_x$ of the population.

In [29]:
```
etf = df[['Close_ETF']]
```

In [30]:
```
mu_x = np.mean(etf)[0]
sigma_x = np.std(etf)[0]
print("Population Mean:\n",mu_x)
print("Population Standard Deviation:\n",sigma_x)
```

```
Population Mean:
 121.1529600120001
Population Standard Deviation:
 12.563503845944297
```

## 2. Break the population into 50 groups sequentially and each group includes 20 values.

## 3. Calculate the sample mean ($\bar{x}$) 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.

In [31]:
```
g = []
for i in np.arange(1,51):
  num = [i]*20
  g.extend(num)
```

In [32]:
```
g = np.array(g)
```

In [33]:
```
etf['group'] = g.tolist()
```

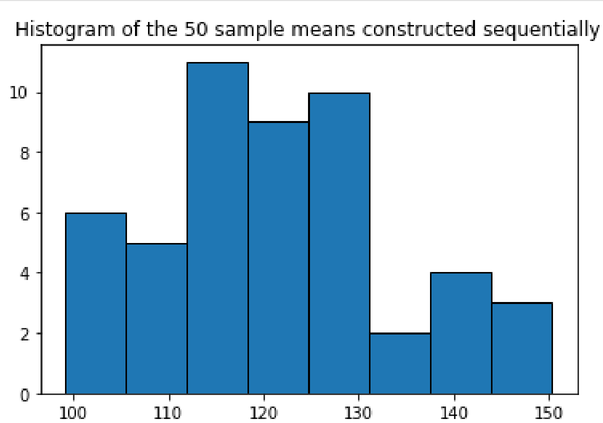
In [34]:
```
seq_means = etf.groupby('group')['Close_ETF'].mean()
```

In [35]:
```
print(seq_means)
```

```
group
1      99.321001
2      99.554000
3      99.154001
4     102.550500
5     103.292000
6     105.093500
7     106.751000
8     111.658001
9     114.499500
10    114.400500
11    112.776500
12    112.286000
13    111.808999
14    113.271499
15    109.947499
16    110.143000
17    112.535500
18    112.075500
19    117.781501
20    120.050500
21    118.208001
22    119.980999
23    119.767500
24    116.803000
25    117.242000
26    120.554501
27    121.091500
28    123.410000
29    122.717000
30    120.611000
31    120.508000
32    125.797000
33    126.883000
34    127.302500
35    128.437500
36    130.136499
37    130.582500
38    128.159000
39    125.125500
40    126.060001
41    129.029500
42    131.811500
43    135.974000
44    138.857000
45    141.288499
46    142.171500
47    144.624500
48    140.522999
49    144.690501
50    150.350499
Name: Close_ETF, dtype: float64
```

In [36]:

```python
plt.hist(x = seq_means, edgecolor='black', bins=8);
plt.title(f'Histogram of the 50 sample means constructed sequentially');
# plt.xlabel(x)
# plt.ylabel('Frequency')
```



> The histogram does not look normal. There's lack of symmetry and there's heavy tails

1. Calculate the mean $\mu_{\bar{x}}$ and the standard deviation ($\sigma_{\bar{x}}$) of the data including these sample means. Make a comparison between $\mu_x$ and $\mu_{\bar{x}}$ , between $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ . Here, n is the number of sample means calculated from Item 3) above.
2. Are the results from Items 3) and 4) consistent with the Central Limit Theorem? Why?

In [37]:
```
mu_xbar = np.mean(seq_means)
sigma_xbar = np.std(seq_means)

print("\u03BC xbar:\n",mu_xbar)
print("\u03C3 xbar:\n",sigma_xbar)
print("\u03BC x:\n", mu_x)
print("\u03C3 x / sqrt(n):\n", sigma_x/(20**0.5))
```

```
μ xbar:
 121.15296001200001
σ xbar:
 12.489175897769007
μ x:
 121.1529600120001
σ x / sqrt(n):
 2.809284863511149
```

- $\mu_x$ and $\mu_{\bar{x}}$ are very close to each other
- $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ are NOT close to each other.
- The results are NOT consistent with the Central Limit Theorem because we did not satisfy the condition for it to be applicable here. Central limit theorem is applicable for when *large* enough *random* samples with *replacement*. Our samples were NOT random and they were not with replacement and they were not large enough.
- The sampling distribution does not look normal

1. Break the population into 10 groups **sequentially** and each group includes 100 values.
2. Repeat Items 3) ~ 5).

In [38]:
```
g = []
for i in np.arange(1,11):
  num = [i]*100
  g.extend(num)
```

In [39]:
```
g = np.array(g)
```

In [40]:
```
etf['group'] = g.tolist()
```

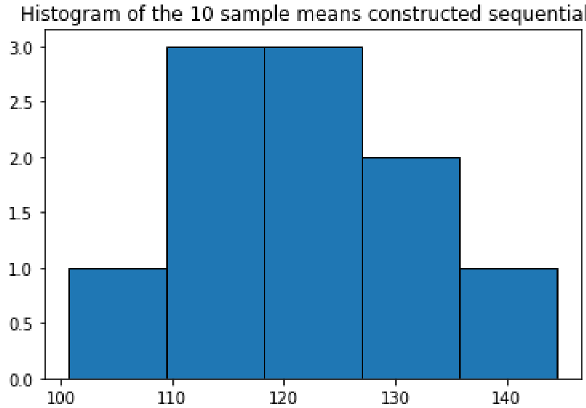
In [41]:
```
seq_means = etf.groupby('group')['Close_ETF'].mean()
```

In [42]:
```
print(seq_means)
```

```
group
1     100.774300
2     110.480500
3     112.018099
4     114.517200
5     118.400300
6     121.676800
7     125.785600
8     128.012700
9     135.392100
10    144.472000
Name: Close_ETF, dtype: float64
```

In [43]:
```
plt.hist(x = seq_means, edgecolor='black', bins=5);
plt.title(f'Histogram of the 10 sample means constructed sequentially');
```

```
plt.show()

mu_xbar = np.mean(seq_means)
sigma_xbar = np.std(seq_means)

print("\u03BC xbar:\n",mu_xbar)
print("\u03C3 xbar:\n",sigma_xbar)
print("\u03BC x:\n", mu_x)
print("\u03C3 x / sqrt(n):\n", sigma_x/(100**0.5))
```




```
μ xbar:
 121.152960012
σ xbar:
 12.16375686089257
μ x:
 121.1529600120001
σ x / sqrt(n):
 1.2563503845944297
```

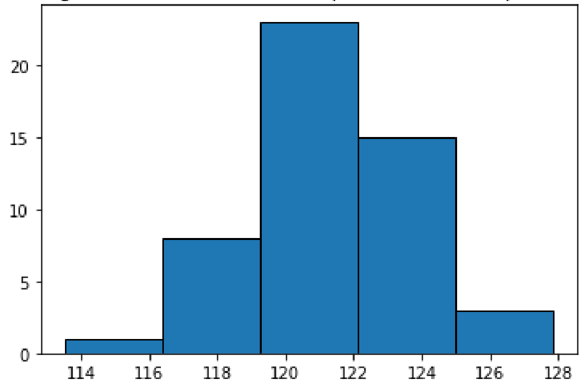
- $\mu_x$ and $\mu_{\overline{x}}$ are very close to each other
- $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\overline{x}}$ are NOT close to each other.
- The results are NOT consistent with the Central Limit Theorem because we did not satisfy the condition for it to be applicable here. Central limit theorem is applicable for when *large* enough *random* samples with *replacement*. Our samples were NOT random and they were not with replacement
- Distribution still does not look normal

1. 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.
2. Repeat Items 3) ~ 5).

In [44]:
```
bootstrap_means_20_50 = []
bootstrap_stds_20_50 = []
for _ in range(50):
  bootstrap_means_20_50.append(etf.sample(20, replace = True).Close_ETF.mean())
  bootstrap_stds_20_50.append(etf.sample(20, replace = True).Close_ETF.std())
```

In [45]:
```
plt.hist(x = bootstrap_means_20_50, edgecolor='black', bins=5);
plt.title(f'Histogram of the 50 random sample means with replacement');
plt.show()

mu_xbar_20_50 = np.mean(bootstrap_means_20_50)
sigma_xbar_20_50 = np.std(bootstrap_means_20_50)

print("\u03BC xbar:\n",mu_xbar_20_50)
print("\u03C3 xbar:\n",sigma_xbar_20_50)
print("\u03BC x:\n", mu_x)
print("\u03C3 x / sqrt(n):\n", sigma_x/(20**0.5))
```

Histogram of the 50 random sample means with replacement



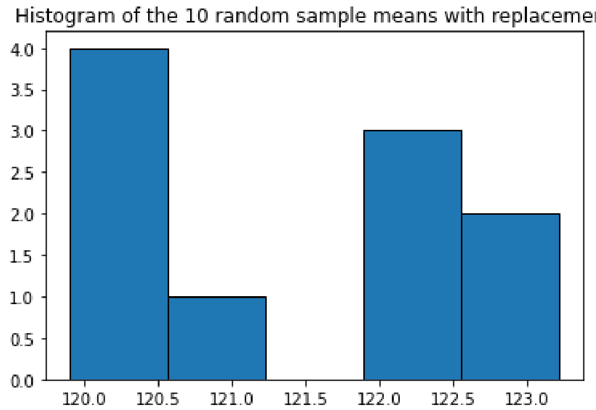

```
μ xbar:
 121.31533984099998
σ xbar:
 2.5629472056243063
μ x:
 121.1529600120001
σ x / sqrt(n):
 2.809284863511149
```

- $\mu_x$ and $\mu_{\bar{x}}$ are very close to each other
- $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ are closer now.
- The results are becoming more consistent with the Central Limit Theorem because we are satisfying most of the condition for it to be applicable here. Central limit theorem is applicable for when *large* enough *random* samples with *replacement*. Our samples were random and with replacement. However, they were not large enough.
- Sampling Distribution does not look normal

1. 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.
2. Repeat Items 3) ~ 5).

In [46]:

```python
bootstrap_means_100_10 = []
bootstrap_stds_100_10 = []
for _ in range(10):
  bootstrap_means_100_10.append(etf.sample(100, replace = True).Close_ETF.mean())
  bootstrap_stds_100_10.append(etf.sample(100, replace = True).Close_ETF.std())

plt.hist(x = bootstrap_means_100_10, edgecolor='black', bins=5);
plt.title(f'Histogram of the 10 random sample means with replacement');
plt.show()

mu_xbar_100_10 = np.mean(bootstrap_means_100_10)
sigma_xbar_100_10 = np.std(bootstrap_means_100_10)

print("\u03BC xbar:\n",mu_xbar_100_10)
print("\u03C3 xbar:\n",sigma_xbar_100_10)
print("\u03BC x:\n", mu_x)
print("\u03C3 x / sqrt(n):\n", sigma_x/(100**0.5))
```

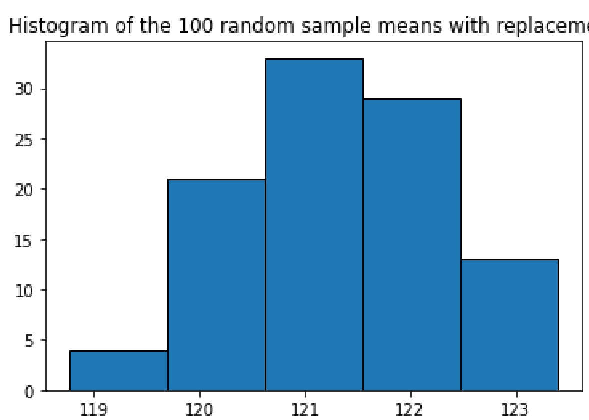



```
μ xbar:
 121.42593993799998
σ xbar:
 1.1605259000575416
μ x:
 121.1529600120001
σ x / sqrt(n):
 1.2563503845944297
```

- $\mu_x$ and $\mu_{\bar{x}}$ are very close to each other
- $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ are closer now.
- Sampling Distribution does not look normal
- Although we have large enough samples and they are random with replacement. We should generate more than just 10 samples so we have enough data points for the histogram and sampling distribution. **For example see below**. Generating 100 random samples with replacements give us a better histogram and now we see that $\frac{\sigma_x}{\sqrt{n}}$ and $\sigma_{\bar{x}}$ are really close

In [47]:

```
bootstrap_means = []
for _ in range(100):
  bootstrap_means.append(etf.sample(100, replace = True).Close_ETF.mean())

plt.hist(x = bootstrap_means, edgecolor='black', bins=5);
plt.title(f'Histogram of the 100 random sample means with replacement');
plt.show()

mu_xbar = np.mean(bootstrap_means)
sigma_xbar = np.std(bootstrap_means)

print("\u03BC xbar:\n",mu_xbar)
print("\u03C3 xbar:\n",sigma_xbar)
print("\u03BC x:\n", mu_x)
print("\u03C3 x / sqrt(n):\n", sigma_x/(100**0.5))
```




```
μ xbar:
 121.31497701949998
σ xbar:
 1.0023685571375862
μ x:
 121.1529600120001
```

```
σ x / sqrt(n):
 1.2563503845944297
```

1. 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 this case, it did not matter what the population distribution of ETF was. The sampling distribution of xbar will follow a normal distribution for large enough samples with replacement. This holds true because of the Central Limit Theorem which states that sampling distribution of xbar will be normal for many types of probability distributions.. There are execeptions, such as cauchy distribution.

# Part 5: Construct a confidence interval with your data

**Requirements**

1. Pick up one of the 10 simple random samples you generated in Step 10) of **Part 4**, construct an appropriate 95% confidence interval of the mean $\mu$.
2. Pick up one of the 50 simple random samples you generated in Step 8) of **Part 4**, construct an appropriate 95% confidence interval of the mean $\mu$.
3. In **Part 1**, you have calculated the mean $\mu$ of the population (the entire ETF column). Do the two intervals from 1) and 2) above include (the true value of) the mean $\mu$? Which one is more accurate? Why?

In [48]:

```
#  95% CI using one of the 10 SRS with size 100 for mu
margin_error_100 = 1.96*sigma_x/(100**0.5)
xbar_100 = bootstrap_means_100_10[9]
lower_100, upper_100 = (xbar_100 - margin_error_100, xbar_100 + margin_error_100)
print("95% CI for \u03BC: ({},{})".format(lower_100, upper_100))
```

```
95% CI for μ: (119.71365301619488,124.63854652380505)
```

In [49]:

```
#  95% CI using one of the 50 SRS with size 20 for mu
margin_error_20 = 1.96*sigma_x/(20**0.5)
xbar_20 = bootstrap_means_20_50[9]
lower_20, upper_20 = (xbar_20 - margin_error_20, xbar_20 + margin_error_20)
print("95% CI for \u03BC: ({},{})".format(lower_20, upper_20))
```

```
95% CI for μ: (116.23230151751814,127.24469818248186)
```

In [50]:

```
print("Population Mean \u03BC = {}".format(mu_x))
```

```
Population Mean μ = 121.1529600120001
```

> The population mean falls on the both confidence intervals. The confidence intervals constructed from a sample size of 100 is narrower than that of the sample size of 20. This provides more accuracy when it is narrower.

# Part 6: Form a hypothesis and test it with your data

**Requirements** –

1. Use the same sample you picked up in **Step 1) of Part 5** to test $H_0$:$\mu = 100$ vs. $H_a$:$\mu \neq 100$ at the significance level 0.05. What's your conclusion?
2. Use the same sample you picked up in **Step 2) of Part 5** to test $H_0$:$\mu = 100$ vs. $H_a$:$\mu \neq 100$ at the significance level 0.05. What's your conclusion?
3. Use the same sample you picked up in **Step 2) of Part 5** to test $H_0$:$\sigma = 15$ vs. $H_a$:$\sigma \neq 15$ at the significance level 0.05. What's your conclusion?
4. Use the same sample you picked up in **Step 2) of Part 5** to test $H_0$:$\sigma = 15$ vs. $H_a$:$\sigma < 15$ at the significance level 0.05. What's your conclusion?

Formulas:

$$z = \frac{\sqrt{n}(\bar{x} - \mu_0)}{\sigma_{\bar{x}}}$$

$$\chi^2 = \Sigma \frac{(n-1)s^2}{\sigma_0^2}$$

In [51]:

```
!pip install scipy
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (1.4.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scipy) (1.19.5)
```

In [52]:

```
import scipy.stats as st
# >>> st.norm.ppf(.95)
# 1.6448536269514722
# >>> st.norm.cdf(1.64)
# 0.94949741652589625
```

In [53]:

```
# H0: mu = 100
# Ha: mu <> 100
print(bootstrap_means_100_10[9])
test_statistic = (100.0**0.5)*(bootstrap_means_100_10[9] - 100)/(sigma_x)
pvalue = (1 - st.norm.cdf(test_statistic))*2
print("test statistic:\n{}".format(test_statistic))
print("pvalue:\n{}".format(pvalue))
```

```
122.17609976999996
test statistic:
17.65120625736806
pvalue:
0.0
```

The p value is less than the chosen significance level, so we reject $H_0$ and conclude $\mu \neq 100$

In [54]:

```
# H0: mu = 100
# Ha: mu <> 100
print(bootstrap_means_20_50[9])
test_statistic = (20.0**0.5)*(bootstrap_means_20_50[9] - 100)/(sigma_x)
pvalue = (1 - st.norm.cdf(test_statistic))*2
print("test statistic:\n{}".format(test_statistic))
print("pvalue:\n{}".format(pvalue))
```

```
121.73849985
test statistic:
7.738090263594845
pvalue:
9.992007221626409e-15
```

The p value is less than the chosen significance level, so we reject $H_0$ and conclude $\mu \neq 100$

In [55]:

```
# H0: sigma = 15
# Ha: sigma <> 15
print(bootstrap_stds_20_50[9])
test_statistic = (20 - 1)*(bootstrap_stds_20_50[9]/15)**2
print("test statistic:\n{}".format(test_statistic))
print(st.chi2.ppf(0.025, 20-1), st.chi2.ppf(0.975, 20-1))
```

```
14.318775406830976
test statistic:
17.31341890610704
8.906516481987971 32.85232686172969
```

The p value is not less than the chosen significance level, so we fail to reject $H_0$. In otherwords, the test statistic falls on the interval, so we failed to reject the null hypothesis.

In [56]:

```
# H0: sigma = 15
# Ha: sigma < 15
print(bootstrap_stds_20_50[9])
test_statistic = (20 - 1)*(bootstrap_stds_20_50[9]/15)**2
pvalue = (st.chi2.cdf(test_statistic, 20 - 1))
print("test statistic:\n{}".format(test_statistic))
print("pvalue:\n{}".format(pvalue))
```

```
14.318775406830976
test statistic:
17.31341890610704
pvalue:
0.43135644911238136
```

In [57]:
```python
st.chi2.ppf(0.05, 20-1)
```

Out[57]:
```
10.117013063859044
```

The p value is not less than the chosen significance level, so we fail to reject $H_0$

# Part 7: Compare your data with a different data set

**Requirements**

1. 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.
2. Subtract the entire Gold column from the entire Oil column and generate a sample of differences. Consider this sample as a random sample from the target population of differences between Gold and Oil. Form a hypothesis and test it to see if the Gold and Oil have equal means in the significance level 0.05.
3. 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 significance level 0.05.

Hypothesis:

Let $\mu_1$ and $\mu_2$ be the population means of oil and gold, respectively.

> $H_0 : \mu_1 = \mu_2 --> \mu_1$ - $\mu_2 = 0$
>
> $H_a : \mu_1 \neq \mu_2 --> \mu_1$ - $\mu_2 \neq 0$
>
> $\alpha = 0.05$

In [58]:
```python
# Determine whether to use pooled vs unpooled t-test
# As long as the ratio of their standard deviations is betwween 0.5 to 2, we can safely assume equal variance
# The ratio is 1.87 so will proceed with pooled
df['oil'].std() / df['gold'].std()
```

Out[58]:
```
1.8684370591076154
```

In [59]:
```python
import statsmodels.stats.weightstats as stats
```

In [60]:
```python
tstat, pvalue, n = stats.ttest_ind(x1=df['oil'], x2=df['gold'], value=0, alternative='two-sided',usevar='pooled')
print("test statistic:", tstat)
print("pvalue:        ", pvalue)
```

```
test statistic: 0.485366613823608
pvalue:         0.627469525830638
```

Since our resulting p-value satisifies p-value = 0.627 > $\alpha$ = 0.05, we cannot reject the null hypothesis and can conclude that the means are equal.

In [61]:
```python
import scipy.stats as stats2
```

Hypothesis:

Let $D_i$ be the difference between the ith oil and gold. Let $\overline{D}$ be the mean of the sample difference. So the population mean of differences is $\mu_D$

> $H_0 : \mu_D = 0$
>
> $H_a : \mu_D \neq 0$
>
> $\alpha = 0.05$

In [62]:
```
df['diff_oil_gold'] = df['oil'] - df['gold']
```

In [63]:
```
stats2.ttest_1samp(a = df['diff_oil_gold'], popmean = 0)
```

Out[63]:
```
Ttest_1sampResult(statistic=0.5413309278514735, pvalue=0.5884002009146817)
```

Since our resulting p-value satisifies p-value = 0.588 > $\alpha$ = 0.05, we cannot reject the null hypothesis and can conclude that the mean of the difference is zero.

Hypothesis:

Let $\sigma_1$ and $\sigma_2$ be the population standard deviation of oil and gold, respectively

> $H_0 : \sigma_1^2 = \sigma_2^2$
>
> $H_a : \sigma_1^2 \neq \sigma_2^2$
>
> $\alpha = 0.05$

In [64]:
```
import scipy.stats as st

def f_test(x, y, alt="two_sided"):
    """
    Calculates the F-test.
    :param x: The first group of data
    :param y: The second group of data
    :param alt: The alternative hypothesis, one of "two_sided" (default), "greater" or "less"
    :return: a tuple with the F statistic value and the p-value.
    """
    df1 = len(x) - 1
    df2 = len(y) - 1
    f = x.var() / y.var()
    if alt == "greater":
        p = 1.0 - st.f.cdf(f, df1, df2)
    elif alt == "less":
        p = st.f.cdf(f, df1, df2)
    else:
        # two-sided by default
        # Crawley, the R book, p.355
        p = 2.0*(1.0 - st.f.cdf(f, df1, df2))
    return f, p
```

In [65]:
```
tstat, pvalue = f_test(df['oil'], df['gold'])
print("test statistic:", tstat)
print("pvalue:        ", pvalue)
```

```
test statistic: 3.491057043846715
pvalue:         2.220446049250313e-16
```

In this case, the F-test resulted in a very low p-value (less than $\alpha$), meaning that we do not believe the standard deviations are equal between the two samples. Referring back to the summary statistics of both samples, it can be seen that the standard deviation of oil is about twice that of gold.

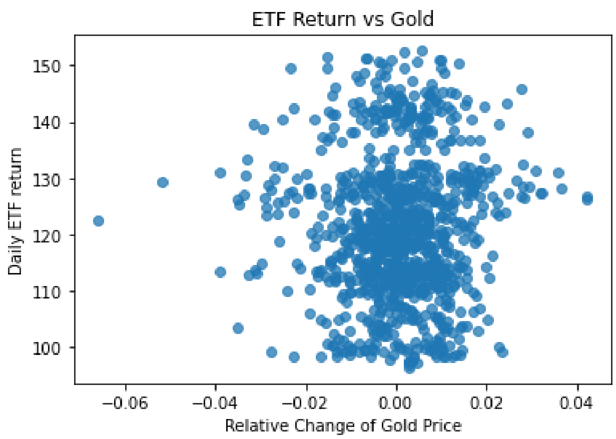
# Part 8: Fitting the line to the data

**Requirements** Consider the data including the ETT column and Gold column only. Using any software,

1. Draw a scatter plot of ETF (Y) vs. Gold (X). Is there any linear relationship between them which can be observed from the scatter plot?
2. Calculate the coefficient of correlation between ETF and Gold and interpret it.
3. Fit a regression line (or least squares line, best fitting line) to the scatter plot. What are the intercept and slope of this line? How to interpret them?
4. Conduct a two-tailed t-test with $H_0$: $\beta_1 = 0$. What is the P-value of the test? Is the linear relationship between ETF (Y) and Gold (X) significant at the significance level 0.01? Why or why not?
5. Suppose that you use the coefficient of determination to assess the quality of this fitting. Is it a good model? Why or why not?
6. What are the assumptions you made for this model fitting?
7. 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% prediction interval of the individual daily ETF return.

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

In [66]:
```
# Gold
x = 'gold'
y = 'Close_ETF'
plt.scatter(data = df, x = x, y = y, alpha=15/20);
plt.title(f'ETF Return vs Gold')
plt.xlabel('Relative Change of Gold Price')
plt.ylabel('Daily ETF return')

plt.show()
```




There doesn't appear to be a linear relationship between ETF and gold price when looking at the scatter plot

1. Calculate the coefficient of correlation between ETF and Gold and interpret it.

Correlation Coefficient

In [67]:
```
# THe correlation coefficient is 0.022996. There appears to be no linear relationship between gold's daily change
df[['Close_ETF', 'gold']].corr()
```

Out[67]:

| | Close_ETF | gold |
|---|---|---|
| **Close_ETF** | 1.000000 | 0.022996 |
| **gold** | 0.022996 | 1.000000 |

Gold and ETF price have a very weak positive correlation (0.0230). Because Pearson's coefficient is so close to 0, we shouldn't expect to see an evident increase in gold's price change or ETF price when the other variable increases.

1. Fit a regression line (or least squares line, best fitting line) to the scatter plot. What are the intercept and slope of this line? How to interpret them?

In [68]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm

np.random.seed(1234)
```

In [69]:
```
X = df['gold']
X = sm.add_constant(X)
y = df['Close_ETF']
```

In [70]:
```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Close_ETF   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                 -0.000
Method:                 Least Squares   F-statistic:                    0.5280
Date:                Sun, 05 Dec 2021   Prob (F-statistic):              0.468
Time:                        19:21:42   Log-Likelihood:                -3949.5
No. Observations:                1000   AIC:                             7903.
Df Residuals:                     998   BIC:                             7913.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        121.1360      0.398    304.155      0.000     120.354     121.918
gold          25.6044     35.236      0.727      0.468     -43.541      94.750
==============================================================================
Omnibus:                       26.752   Durbin-Watson:                   0.005
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               23.045
Skew:                           0.305   Prob(JB):                     9.91e-06
Kurtosis:                       2.576   Cond. No.                         88.6
==============================================================================

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

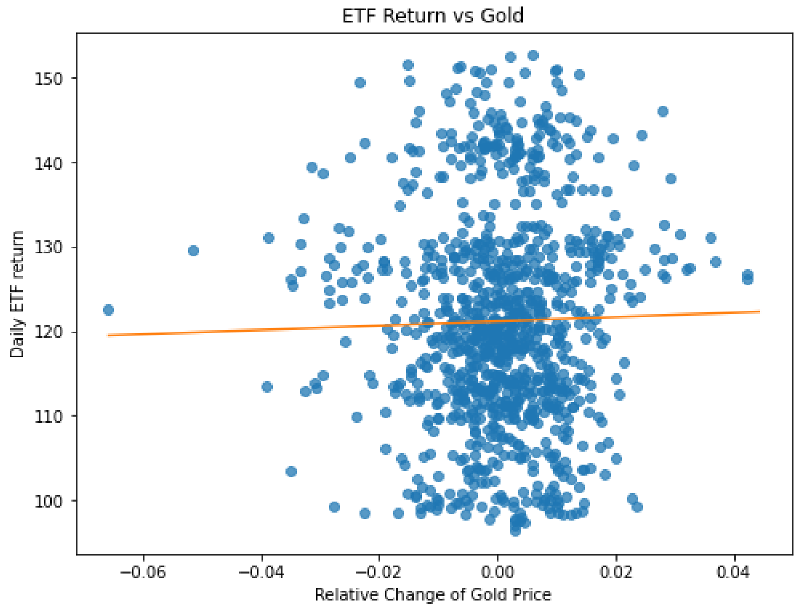
In [71]:
```
const, m = results.params[0], results.params[1]
print('intercept:', const)
print('slope    :', m)
```

```
intercept: 121.13598849889824
slope    : 25.604389324427302
```

The intercept represents the price of an ETF when the price of gold doesn't experience a price change. The coefficient, on the other hand, represents the incremental fluctuation of the ETF price with respect to gold's daily price change. When the price of gold increases or decreases by 100% (1.0, not 100.0), the ETF price increases/decreases by $25.60.

> For each change

In [72]:
```
# Gold
plt.figure(figsize = (8,6))
x = df['gold']
plt.plot(x, df['Close_ETF'], 'o', alpha=15/20);
plt.title(f'ETF Return vs Gold')
plt.xlabel('Relative Change of Gold Price')
plt.ylabel('Daily ETF return')

xx = np.arange(min(x), max(x)+.01, 0.01)
plt.plot(xx, const + m*xx)

plt.show()
```




1. Conduct a two-tailed t-test with $H_0$: $\beta_1 = 0$. What is the P-value of the test? Is the linear relationship between ETF (Y) and Gold (X) significant at the significance level 0.01? Why or why not?

In [73]:
```
results.pvalues
```

Out[73]:
```
const    0.000000
gold     0.467612
dtype: float64
```

> The linear relationship between ETF and Gold is NOT significant at the 0.01 signfcance level because the p-value = 0.467612 > 0.01.

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

Rsquare

In [74]:
```
results.rsquared
```

Out[74]:
```
0.0005287962431228532
```

No, this model is not a good one using the coefficient of determination. From the coefficient of determination, the model suggests that it can account for ~0.05% of ETF's variance, which has very poor predictive capability.

1. What are the assumptions you made for this model fitting?

Using a linear regression model, four key assumptions were made:

> - the relationship between the daily percent change of the gold price and the ETF price was linear
> - model errors are normally distributed
> - the observations are all independent from each other where the residuals are all independet
> - there is a constant variance across the residuals

1. 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% prediction interval of the individual daily ETF return.

In [75]:
```
new_data = {'gold':[0.005127]}
X = pd.DataFrame.from_dict(new_data)['gold']
type(X)
X = df['gold'].sample(n=1)
print(X)
```

```
X = sm.add_constant(X)
print(X.shape)
print(type(X))
```

```
681   -0.022073
Name: gold, dtype: float64
(1, 1)
<class 'pandas.core.frame.DataFrame'>
```

In [76]:

```
new_data = {'gold':[0.005127, 0, -1]}
X = pd.DataFrame.from_dict(new_data)['gold']
type(X)
# X = df['gold'].sample(n=2)
print(X)
X = sm.add_constant(X)
print(X.shape)
print(type(X))
```

```
0    0.005127
1    0.000000
2   -1.000000
Name: gold, dtype: float64
(3, 2)
<class 'pandas.core.frame.DataFrame'>
```

In [77]:

```
# print(results.get_prediction().summary_frame())
predictions = results.get_prediction(X)
(predictions.summary_frame(alpha=0.01)).loc[0,:]
```

Out[77]:

```
mean             121.267262
mean_se            0.427572
mean_ci_lower    120.163800
mean_ci_upper    122.370725
obs_ci_lower      88.801169
obs_ci_upper     153.733355
Name: 0, dtype: float64
```

```
[1] "Prediction Interval generated by R:"
       fit      lwr      upr
1 121.2673 88.80117 153.7334
[1] "Confidence Interval generated by R:"
       fit      lwr      upr
1 121.2673 120.1638 122.3707
```

---

# Part 9: Does your model predict?

**Requirements** –

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

In [77]:

```

```

In [78]:

```
X = df[['gold', 'oil']]
X = sm.add_constant(X)
y = df['Close_ETF']
```

In [79]:

```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            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:                Sun, 05 Dec 2021   Prob (F-statistic):              0.688
Time:                        19:21:42   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
gold          29.6226     36.272      0.817      0.414     -41.555     100.800
oil           -9.1261     19.413     -0.470      0.638     -47.221      28.968
==============================================================================
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
==============================================================================

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

In [80]:

```
results.rsquared
```

Out[80]:

```
0.0007502966608659012
```

> The $R^2$ is nearly zero! Furthermore, it is worth pointing out that the p-values for the coefficients for oil and gold are very high and greater than 0.05. These variables do not appear to be useful in predicting ETF.

# Part 10: Checking residuals and model selection

**Requirements** – Calculate the residuals of the model fitting you did in Part 9. Check the four assumptions made for the error terms of the multiple regression model using these residuals (mean 0; constant variance; normality; and the independence). You may draw some plots over the residuals to check these assumptions. For example, draw a Normal Probability Plot to check the normality assumption; draw a scatter plot of Residuals vs. Fitted Values to check the constant variance assumption and the independence assumption; and so on. You may refer to the following link https://www.youtube.com/watch?v=4zQkJw73U6I for some hints. In your project report, all the relevant plots and at least one paragraph of summary of checking the four assumptions using those plots must be included.

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

In [ ]:

```
proc reg data=sasdf;
model Close_ETF = gold oil;
run;
```

**The SAS System**

**The REG Procedure**
**Model: MODEL1**
**Dependent Variable: Close_ETF**

| | |
|---|---|
| **Number of Observations Read** | 1000 |
| **Number of Observations Used** | 1000 |

| Analysis of Variance | | | | | |
|---|---|---|---|---|---|
| **Source** | **DF** | **Sum of Squares** | **Mean Square** | **F Value** | **Pr > F** |
| **Model** | 2 | 118.42805 | 59.21402 | 0.37 | 0.6879 |
| **Error** | 997 | 157723 | 158.19779 | | |

| Analysis of Variance | | | | | |
|---|---|---|---|---|---|
| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
| Corrected Total | 999 | 157842 | | | |

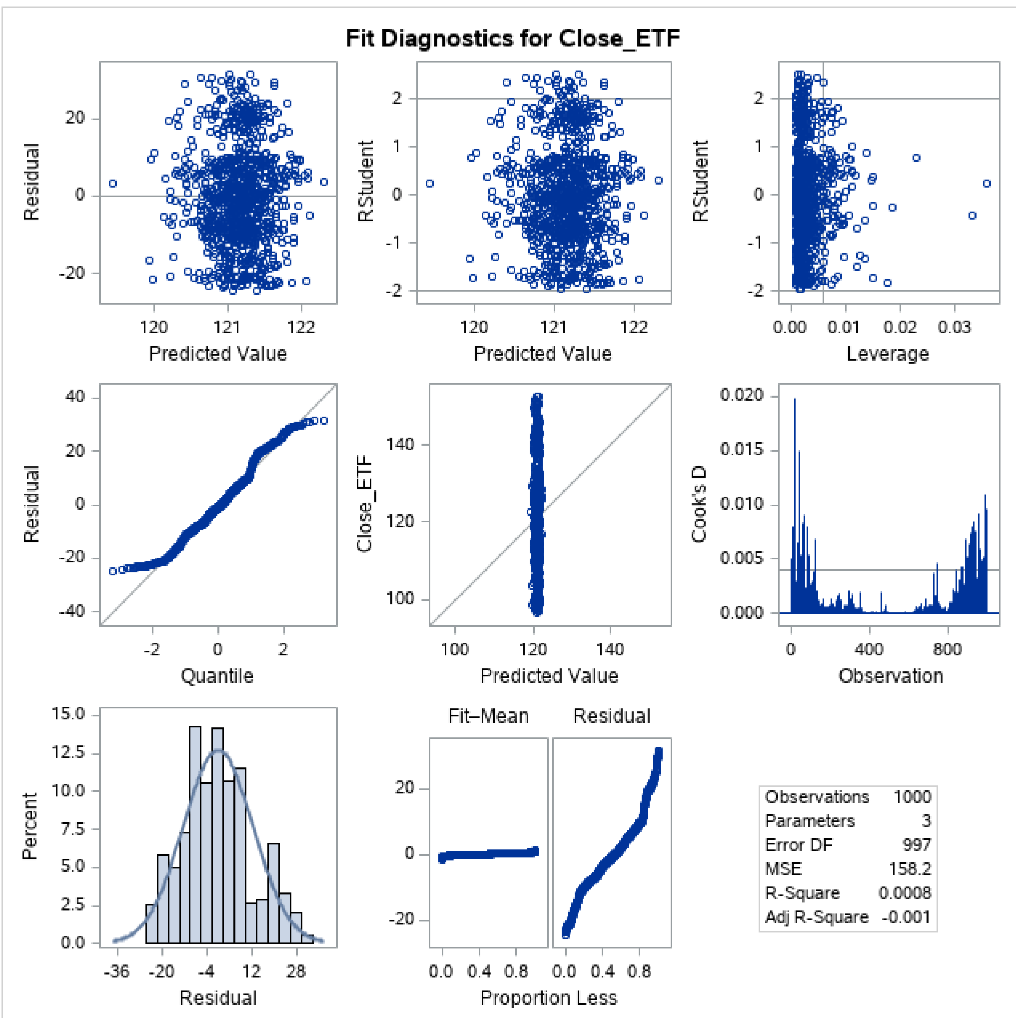
| | | | |
|---|---|---|---|
| Root MSE | 12.57767 | R-Square | 0.0008 |
| Dependent Mean | 121.15296 | Adj R-Sq | -0.0013 |
| Coeff Var | 10.38165 | | |

| Parameter Estimates | | | | | |
|---|---|---|---|---|---|
| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > \|t\| |
| Intercept | 1 | 121.14273 | 0.39868 | 303.86 | <.0001 |
| gold | 1 | 29.62259 | 36.27153 | 0.82 | 0.4143 |
| oil | 1 | -9.12610 | 19.41276 | -0.47 | 0.6384 |

---



## The REG Procedure
## Model: MODEL1
## Dependent Variable: Close_ETF




**Discussion of the Assumptions**

> The residuals are independent. The scatter plot has a random pattern and the mean is zero. The variance appears to be constant so we satisfy the homoskedasticity assumption. We also see that the residuals' distribution are somewhat normal. Having said that, this regression model has virtually no predictive power based on the R square and the fact that non of the predictors are significant!

**Strategy to improve model**

> We initially had calculated the correlations between the dependent variable and the predictors, which we saw very weak correlations. Thus, we had already anticipated that the ordinary least squares regression model would not perform well. Interestingly, if we instead calculate the relative change of the dependent variable (ETF) and run the regression model, we may get a better model. We calculated the correlations (see below) between the relative change of ETF vs the relative change of the predictors (which are the original variables) and have better correlations between them since we are predicting relative change with relative predictors.
>
> Another approach is to use time series techniques because this data is time series data. Using ARIMA for this data.

In [86]:
```
df['etf_change'] = (df['Close_ETF'] - df['Close_ETF'].shift())/df['Close_ETF'].shift()
```

In [87]:
```
df.corr()
```

Out[87]:

| | Close_ETF | oil | gold | JPM | etf_change |
|---|---|---|---|---|---|
| **Close_ETF** | 1.000000 | -0.009045 | 0.022996 | 0.036807 | 0.040974 |
| **oil** | -0.009045 | 1.000000 | 0.235650 | -0.120849 | -0.071179 |
| **gold** | 0.022996 | 0.235650 | 1.000000 | 0.100170 | 0.089717 |
| **JPM** | 0.036807 | -0.120849 | 0.100170 | 1.000000 | 0.705986 |
| **etf_change** | 0.040974 | -0.071179 | 0.089717 | 0.705986 | 1.000000 |

In [ ]:
```
```