MA541 - Statistical Methods Project

Reeva Andipara - 10474756

Introduction:

- This report summarizes the statistical analysis and modeling results associated with the JPMorgan Chase & Co's stock dataset. The prupose of this report is to document the implemented analysis, hypothesis testing for the data and model fitting along with other inference techniques used during the statistical analysis.
- Initially the data is described by different plots and the analysis is done to figure out what distribution does the data follow. Then the data is broken in different samples and consistency of Central Limit Theorem is discussed. After constructing the confidence interval for, in part-6 a hypothesis is tested. Hypothesis test is used to compare the data with different datasets. In part-8, we fit the line to the data and check whether the model can predict or not. At last we check for the residuals and model selection

Implementation with output:

In [13]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
from scipy.stats import norm
from scipy import stats
import statsmodels.api as sm
warnings.filterwarnings('ignore')
```

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 [14]:
```
df = pd.read_excel (r'E:\MS DS\MA 541\Project\MA 541 Course Project Data.xlsx')
print("Close_ETF: mean=%.3f stdv=%.3f" % (df['Close_ETF'].mean(),df['Close_ETF'].std())
print("oil: mean=%.3f stdv=%.3f" % (df['oil'].mean(),df['oil'].std()))
print("gold: mean=%.3f stdv=%.3f" % (df['gold'].mean(),df['gold'].std()))
print("JPM: mean=%.3f stdv=%.3f" % (df['JPM'].mean(),df['JPM'].std()))
```

```
Close_ETF: mean=121.153 stdv=12.570
oil: mean=0.001 stdv=0.021
```

```
gold: mean=0.001 stdv=0.011
JPM: mean=0.001 stdv=0.011
```

In [15]:

```
# Correlation between two variables

print("Close_ETF-oil: ", df['Close_ETF'].corr(df['oil']))
print("Close_ETF-gold: ", df['Close_ETF'].corr(df['gold']))
print("Close_ETF-JPM: ", df['Close_ETF'].corr(df['JPM']))
print("oil-gold: ", df['oil'].corr(df['gold']))
print("oil-JPM: ", df['oil'].corr(df['JPM']))
print("gold-JPM: ", df['gold'].corr(df['JPM']))
```

```
Close_ETF-oil:  -0.009044842009622087
Close_ETF-gold:  0.022995570076054607
Close_ETF-JPM:  0.03680705773259183
oil-gold:  0.23565037184144094
oil-JPM:  -0.12084893009495938
gold-JPM:  0.10016984211388412
```

Part 2: Describe your data

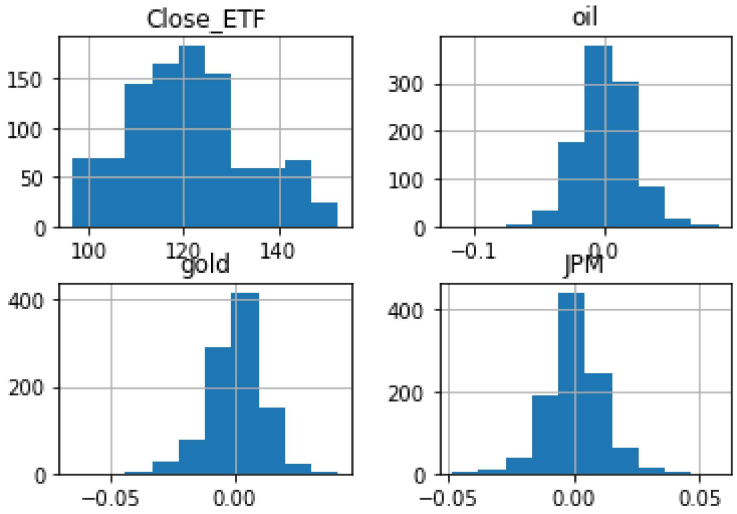
1) A histogram for each column (hint: four histograms total)

In [16]:

```
# Part-2
# 1) Histogram for each column

df.hist()
```

Out[16]:

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



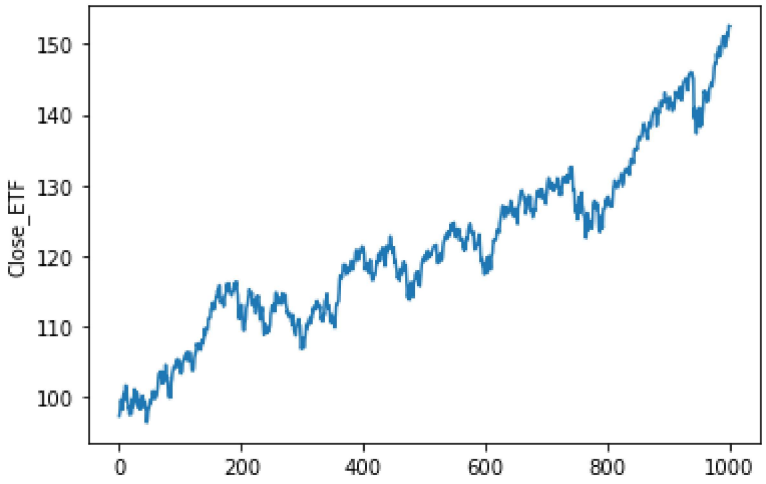
2) A time series plot for each column (hint: use the series “1, 2, 3, …, 1000” as the horizontal axis; four plots total)

In [17]:

```
x=np.arange(0,1000,1)
sns.lineplot(x,df['Close_ETF'])
```

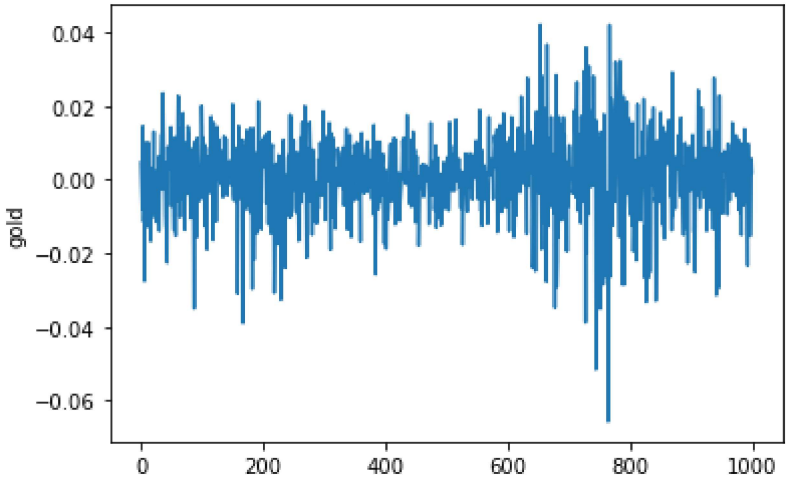
Out[17]:

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



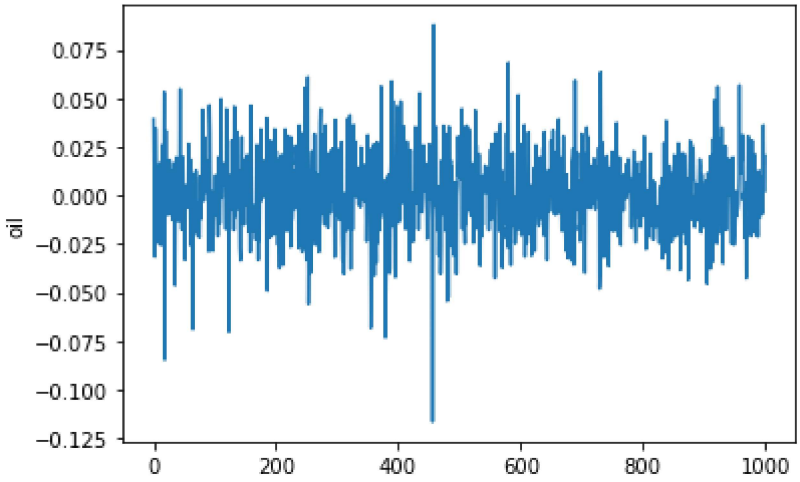
In [18]:
```
sns.lineplot(x,df['gold'])
```

Out[18]: <AxesSubplot:ylabel='gold'>



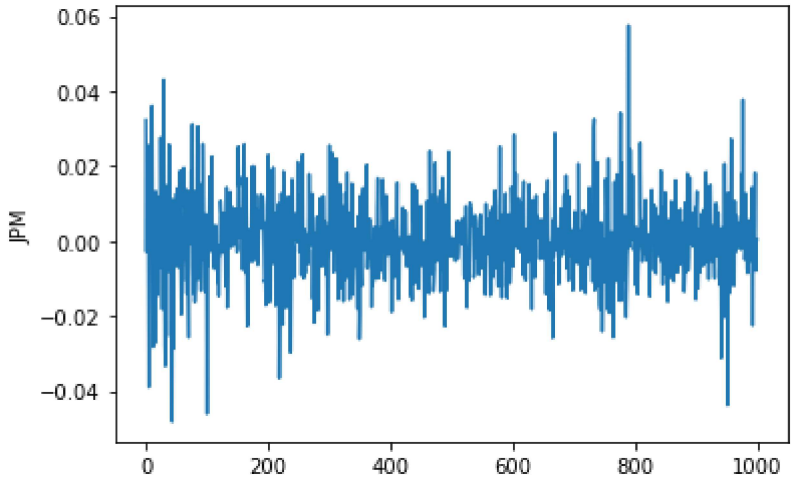
In [19]:
```
sns.lineplot(x,df['oil'])
```

Out[19]: <AxesSubplot:ylabel='oil'>



In [20]:
```
sns.lineplot(x,df['JPM'])
```

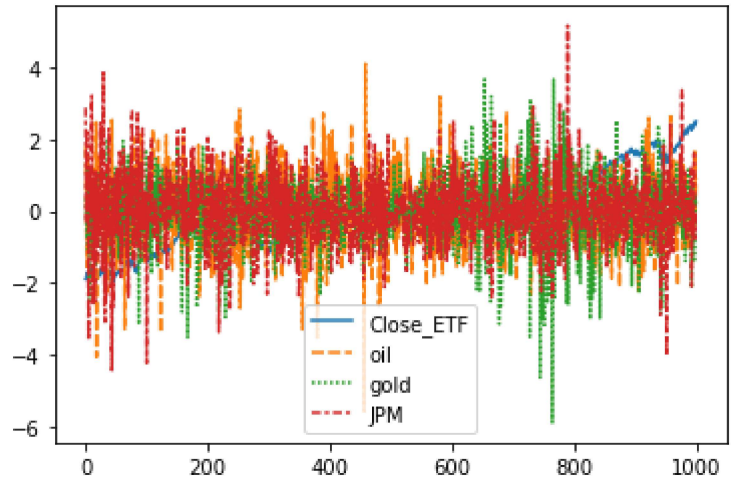
Out[20]: <AxesSubplot:ylabel='JPM'>



3) A time series plot for all four columns (hint: one plot including four "curves" and each "curve" describes one column)

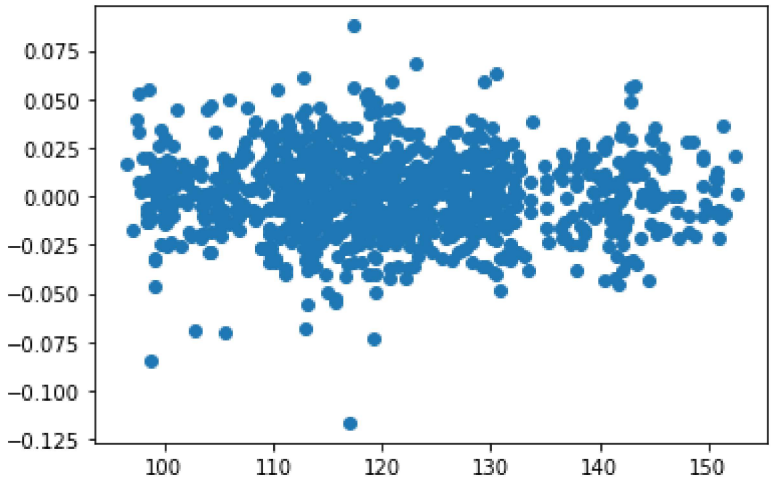
In [23]:
```
y = (df-df.mean())/df.std()
sns.lineplot(data=y)
```

Out[23]: <AxesSubplot:>

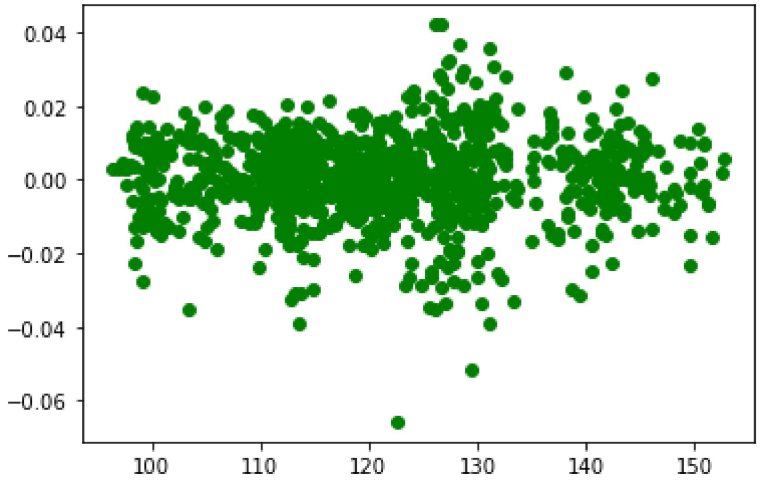


4) Three scatter plots to describe the relationships between the ETF column and the OIL column; between the ETF column and the GOLD column; between the ETF column and the JPM column, respectively

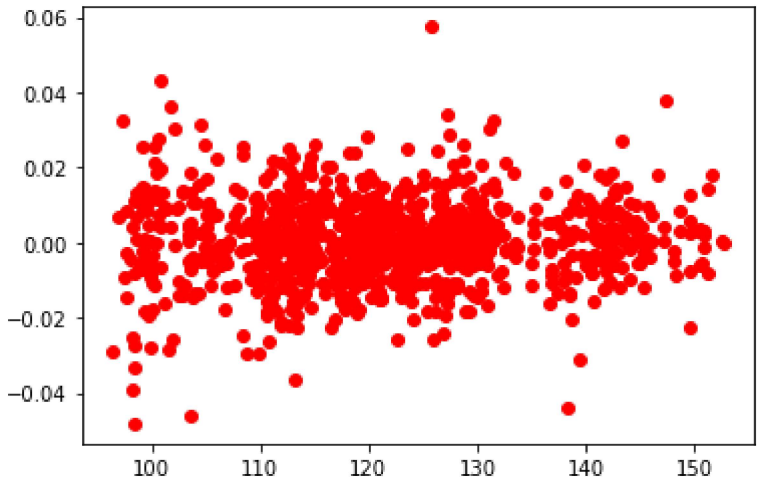
In [13]:
```
plt.scatter(a, b)
plt.show()
plt.scatter(a, c, Color='green')
plt.show()
plt.scatter(a, d, Color='red')
plt.show()
```




```
<ipython-input-13-3e526fe5dde8>:3: MatplotlibDeprecationWarning:

Case-insensitive properties were deprecated in 3.3 and support will be removed two minor
releases later
```




```
<ipython-input-13-3e526fe5dde8>:5: MatplotlibDeprecationWarning:

Case-insensitive properties were deprecated in 3.3 and support will be removed two minor
releases later
```



Part 3

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.

Assumptions for ETF, Oil, Gold and JPM:

H0: Follows Normal Distribution

H1: Doesn't follow Normal Distribution

In [14]:
```
from scipy.stats import chisquare

stat, p_value = chisquare(np.array(df['Close_ETF']))
if(p_value>0.1):
    print("Normal")
else:
    print("Reject H0")
```

```
Reject H0
```

In [15]:
```
stat, p_value = chisquare(np.array(df['oil']))
if(p_value>0.1):
    print("Normal")
else:
    print("Reject H0")
```

```
Normal
```

In [16]:
```
stat, p_value = chisquare(np.array(df['gold']))
if(p_value>0.1):
    print("Normal")
else:
    print("Reject H0")
```

```
Normal
```

In [17]:
```
stat, p_value = chisquare(np.array(df['JPM']))
if(p_value>0.1):
    print("Normal")
else:
    print("Reject H0")
```

```
Normal
```

Part 4: Break your data into small groups and let them discuss the importance of the Central Limit Theorem

1) Calculate the mean and the standard deviation of the population.

In [18]:

```
import statistics as st
p_mean = st.mean(df['Close_ETF'])
print("The mean of the population is %s" %p_mean)
p_stdev = st.stdev(df['Close_ETF'])
print("The standard deviation of the population is %s" %p_stdev)
```

```
The mean of the population is 121.152960012
The standard deviation of the population is 12.569790313110744
```

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

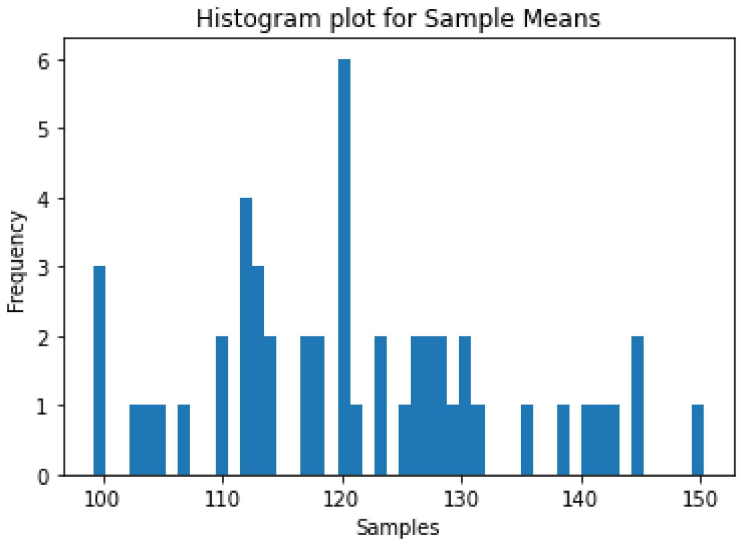
In [61]:

```
data = df['Close_ETF'].tolist()
n = 20

sample=[]
for i in range(0, len(df),n):
    group = data[i:i+20]
    sample.append(group)

# for i in range (0, 50):
#     print("Sample %s is : \n %s" %(i+1 ,sample[i]))
```

3) Calculate the sample mean 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 [20]:

```
sample_mean = []
for i in range (0,50):
    group1 = st.mean(sample[i])
    sample_mean.append(group1)

# for i in range (0, 50):
#     print("Mean of Sample %s is : \n %s" %(i+1 ,sample_mean[i]))

plt.hist(sample_mean,50)
plt.title("Histogram plot for Sample Means")
plt.xlabel("Samples")
plt.ylabel("Frequency")
plt.show()
```



- If the normal probability plot is linear, then the normal distribution is a good model for the data. Since the plot is close to a Normal Distribution Graph, hence it is a good model.

4) Calculate the mean and the standard deviation of the data including these sample means.

In [21]:
```
sample_mean = []
for i in range (0, 50):
    group1 = st.mean(sample[i])
    sample_mean.append(group1)

total_sample_mean = st.mean(sample_mean)
total_sample_std_deviation = st.stdev(sample_mean)

print("The mean of the sample is %s" %total_sample_mean)
print("The standard deviation of the sample is %s" %total_sample_std_deviation)
```

```
The mean of the sample is 121.15296001200001
The standard deviation of the sample is 12.615972812491503
```

- The mean of sample mean ($\mu\bar{x}$) is 121.15296001200001.
- $\sigma x/\sqrt{n}$ is 1.7767477529860964
- The mean (μx) and the mean of sample means ($\mu\bar{x}$) is equal because the samples are chosen sequentially.

5) Are the results from Items 3) and 4) consistent with the Central Limit Theorem? Why?

- It is not consistent because:
  - a. Sampling Distribution Mean($\mu\bar{x}$) = Mean(μx) - True
  - b. Sampling distribution's standard deviation (Standard error) = $\sigma\bar{x}/\sqrt{n}$ is almost equal to $\sigma\bar{x}$ - Not True
  - c. For n > 30, the sampling distribution becomes a normal distribution.- True
- It doesn't meet the requirements.

6) Break the population into 10 groups sequentially and each group includes 100 values.

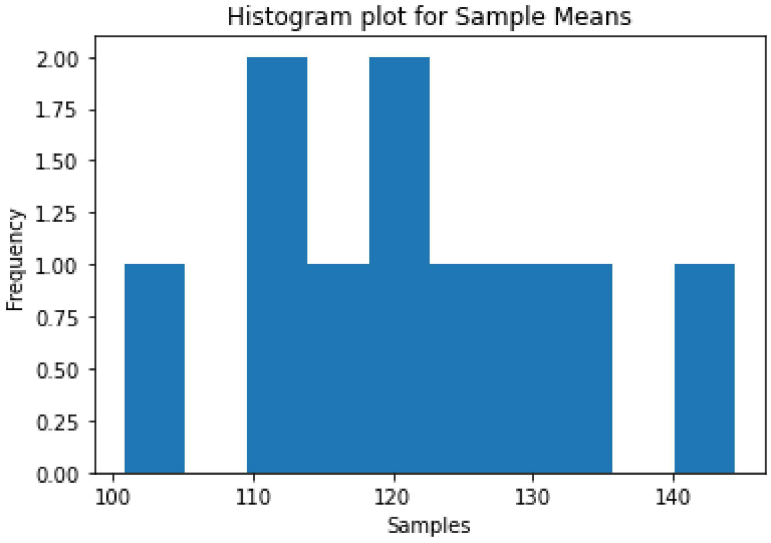
In [22]:
```
data = df['Close_ETF'].tolist()
n = 100

sample=[]
for i in range(0, len(df),n):
    group = data[i:i+100]
    sample.append(group)

# for i in range (0, 10):
#     print("Sample %s is : \n %s" %(i+1 ,sample[i]))
```

7) Repeat Items 3) ~ 5).

In [23]:
```
sample_mean = []
for i in range (0, 10):
    group1 = st.mean(sample[i])
    sample_mean.append(group1)

for i in range (0, 10):
```

```
    print("Mean of Sample %s is : \n %s" %(i+1 ,sample_mean[i]))

plt.hist(sample_mean,10)
plt.title("Histogram plot for Sample Means")
plt.xlabel("Samples")
plt.ylabel("Frequency")
plt.show()
```

```
Mean of Sample 1 is :
 100.77430029
Mean of Sample 2 is :
 110.48050028
Mean of Sample 3 is :
 112.01809939
Mean of Sample 4 is :
 114.51720014
Mean of Sample 5 is :
 118.40030004
Mean of Sample 6 is :
 121.6768003
Mean of Sample 7 is :
 125.78560011
Mean of Sample 8 is :
 128.01269998
Mean of Sample 9 is :
 135.39209964
Mean of Sample 10 is :
 144.47199995
```



In [24]:

```
total_sample_mean = st.mean(sample_mean)
total_sample_std_deviation = st.stdev(sample_mean)

print("The mean of the sample is %s" %total_sample_mean)
print("The standard deviation of the sample is %s" %total_sample_std_deviation)
```

```
The mean of the sample is 121.152960012
The standard deviation of the sample is 12.821725528306825
```

- It is consistent because:
  - a. Sampling Distribution Mean($\mu\bar{x}$) = Mean(μx) - True

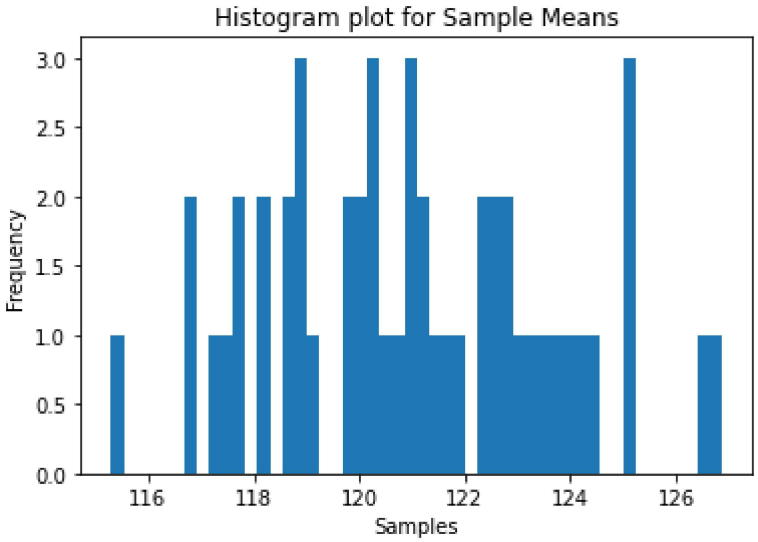
    - c. For n > 30, the sampling distribution becomes a normal distribution.- True
- It meets the requirements.

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.

In [25]:
```
import random
df = pd.read_excel (r'/Users/reevaandipara/Assignments/MA541_Course_Project_Data.xlsx')
data = df['Close_ETF'].tolist()

sample=[]
for i in range(0, 50):
    group = random.sample(data,20)
    sample.append(group)

# for i in range (0, 50):
#     print("Sample %s is : \n %s" %(i+1 ,sample[i]))
```

9) Repeat Items 3) ~ 5)

In [26]:
```
sample_mean = []
for i in range (0, 50):
    group1 = st.mean(sample[i])
    sample_mean.append(group1)

# for i in range (0, 50):
#     print("Mean of Sample %s is : \n %s" %(i+1 ,sample_mean[i]))

plt.hist(sample_mean,50)
plt.title("Histogram plot for Sample Means")
plt.xlabel("Samples")
plt.ylabel("Frequency")
plt.show()
```



In [27]:
```
total_sample_mean = st.mean(sample_mean)
total_sample_std_dev = st.stdev(sample_mean)
print(total_sample_mean)
print(total_sample_std_dev)
```

```
121.032060117
2.676564015622219
```

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.

In [28]:

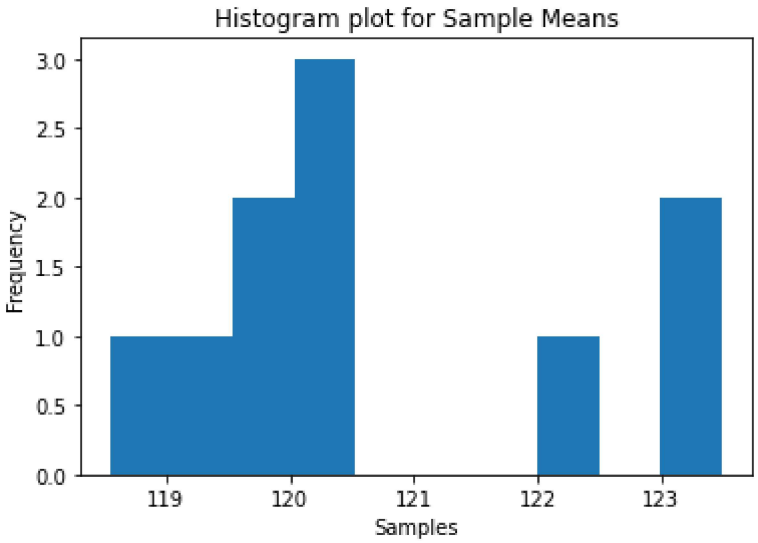
```
sample=[]
for i in range(0, 10):
    group = random.sample(data,100)
    sample.append(group)

# for i in range (0, 10):
#     print("Sample %s is : \n %s" %(i+1 ,sample[i]))
```

11) Repeat Items 3) ~ 5)

In [29]:

```
sample_mean = []
for i in range (0, 10):
    group1 = st.mean(sample[i])
    sample_mean.append(group1)

for i in range (0, 10):
    print("Mean of Sample %s is : \n %s" %(i+1 ,sample_mean[i]))

plt.hist(sample_mean,10)
plt.title("Histogram plot for Sample Means")
plt.xlabel("Samples")
plt.ylabel("Frequency")
plt.show()
```

```
Mean of Sample 1 is :
 119.3641999
Mean of Sample 2 is :
 123.48439985
Mean of Sample 3 is :
 120.32880047
Mean of Sample 4 is :
 123.16389991
Mean of Sample 5 is :
 120.40849959
Mean of Sample 6 is :
 119.81640007
Mean of Sample 7 is :
 120.37850014
Mean of Sample 8 is :
 118.55520004
Mean of Sample 9 is :
 120.02310005
Mean of Sample 10 is :
 122.47930006
```



In [30]:
```
total_sample_mean = st.mean(sample_mean)
total_sample_std_dev = st.stdev(sample_mean)
print(total_sample_mean)
print(total_sample_std_dev)
```

```
120.800230008
1.661282247205508
```

Conclusion:

- The sample size should be large to approach normality.
- The sampling distribution of sample means approaches the normalistribution as thesample size gets larger.It is true especially for sample size>30. Therefore, as the sample size in 1oth part is 100 that is large as compared to other sample sizes and so it is more normal then other.
- Hence, large sample size (as per this project sample size of 100) is consistent with the Central Limit Theorem.

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.

- The distribution of population has no impact.
- The sample mean distribution is affected by sample size number. It is not affected by distribution of population.
- As per Central Limit Theorem, when there are more samples, distribution will approach a normal distribution.
- Also, even if the opulation distribution is normal, it is not compulsary that all sample meanistributions will be normal.

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 . μ

In [31]:
```
n=100
confidence = 0.95
alpha = 1 - confidence
print(alpha)
```

```
0.050000000000000044
```

In [32]:
```
# Consider Sample 6 for analysis
x = st.mean(sample[5])
total = st.stdev(sample[5])
x
```

Out[32]:
```
119.81640007
```

In [33]:
```
z_criti =stats.norm.ppf(q=0.975)
z_int = stats.norm.interval(alpha = confidence)
print(z_int)
```

```
(-1.959963984540054, 1.959963984540054)
```

In [34]:
```
import math
stand_error = total / math.sqrt(n)
print(stand_error)
```

```
1.2056010568106443
```

In [35]:
```
CI_lowerlimit = x - z_criti * stand_error
CI_upperlimit = x + z_criti * stand_error
CI_lowerlimit, CI_upperlimit
```

Out[35]:
```
(117.4534654189277, 122.1793347210723)
```

- 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 .μ

In [36]:
```
n=20
confidence = 0.95
alpha = 1 - confidence
x = st.mean(sample[6])
total = st.stdev(sample[6])
z_criti =stats.norm.ppf(q=0.975)
z_int = stats.norm.interval(alpha = confidence)
stand_error = total / math.sqrt(n)
CI_lowerlimit = x - z_criti * stand_error
CI_upperlimit = x + z_criti * stand_error
CI_lowerlimit, CI_upperlimit
```

Out[36]:
```
(115.01015294344866, 125.74684733655134)
```

- 3) In Part 1, you have calculated the mean of the population (the entire ETF column) μ using Excel function. Do the two intervals from 1) and 2) above include (the true value of) the mean ? Which one is more accurate? Why?

In [37]:
```
n=20
confidence = 0.95
alpha = 1 - confidence
x = st.mean(df['Close_ETF'])
total = st.stdev(df['Close_ETF'])
z_criti =stats.norm.ppf(q=0.975)
z_int = stats.norm.interval(alpha = confidence)
stand_error = total / math.sqrt(n)
CI_lowerlimit = x - z_criti * stand_error
CI_upperlimit = x + z_criti * stand_error
CI_lowerlimit, CI_upperlimit
```

Out[37]:
```
(115.64410774211862, 126.66181228188137)
```

- The first part is more accurate. Simce the mean is 123.10749988 that lies in the confidence interval that is small interval and hence gives less chance of getting an error unlike the other two confidence intervals.

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 $H0$: $\mu = 100$ vs. Ha: $\mu \neq 100$ at the significance level 0.05. What's your conclusion?

In [38]:
```
import scipy.stats
x = st.mean(sample[5])
u = st.mean(data)
s = st.stdev(data)
n = len(sample[5])
alpha = 0.05

z_test = (x-u)/(s/math.sqrt(n))
z_critical1 = scipy.stats.norm.ppf(alpha/2)
z_critical2 = scipy.stats.norm.ppf(1-alpha/2)

print("z critical 1 =", z_critical1)
print("z critical 2 =", z_critical2)
print("z_test =", z_test)
if(z_test>z_critical1 or z_test<z_critical2):
    print("Reject H0")
else:
    print("Accept H0")
```

```
z critical 1 = -1.9599639845400545
z critical 2 = 1.959963984540054
z_test = -1.063311247607618
Reject H0
```

- As the test statistics value is in critical rejion we reject null hypothesis.

2) Use the same sample you picked up in Step 2) of Part 5 to test $H0 : \mu = 100$ vs. $Ha : \mu \neq 100$ at the significance level 0.05. What's your conclusion?

In [39]:
```
# u = 100
# z_score = (x - u)/stand_error
# p_value = scipy.stats.norm.sf(abs(z_score))
# print('P-value =',(p_value))
# if p_value<alpha :
#     print('p_value is less than alpha. Therefore we reject the Null hypothesis')
# else:
#     print('p_value is greater than alpha. We do not reject Null hypothesis')

x = st.mean(sample[6])
u = st.mean(data)
s = st.stdev(data)
n = len(sample[6])
alpha = 0.05

z_test = (x-u)/(s/math.sqrt(n))
z_critical1 = scipy.stats.norm.ppf(alpha/2)
z_critical2 = scipy.stats.norm.ppf(1-alpha/2)

print("z critical 1 =", z_critical1)
print("z critical 2 =", z_critical2)
print("z_test =", z_test)
if(z_test>z_critical1 or z_test<z_critical2):
    print("Reject H0")
else:
    print("Accept H0")
```

```
z critical 1 = -1.9599639845400545
z critical 2 = 1.959963984540054
z_test = -0.616127916781718
Reject H0
```

- As the test statistics is in the critical region, we reject null hypothesis.

3) Use the same sample you picked up in Step2) of Part5 to test $H0 : \sigma=15$ vs $Ha : \sigma \neq 15$ at the significance level 0.05. What's your conclusion?

In [40]:
```
from scipy.stats import chi2
from scipy.stats import chi2_contingency

n = len(sample[5])
s = st.stdev(sample[5])
sigma = 15
alpha = 0.05
dof = n-1

test_stat = (n-1)/(s/sigma)

critical_val = chi2.ppf(alpha/2, dof)
critical_val = chi2.ppf(1-alpha/2, dof)
print("test_statisics: ", test_stat)
print("critical_val: ", critical_val)
```

```
if(test_stat<critical_val or test_stat>critical_val):
    print("Reject H0")
else:
    print("Accept H0")
```

```
test_statisics:  123.17507450835281
critical_val:  128.4219886438403
Reject H0
```

4) Use the same sample you picked up in Step2) of Part5 to test $H0$:σ=15 vs. Ha: σ < 15 at the significance level 0.05. What's your conclusion?

In [41]:
```
test_stat = (n-1)/(s/sigma)

critical_val = chi2.ppf(1-alpha, dof)
print("test_statisics: ", test_stat)
print("critical_val: ", critical_val)
if(test_stat<critical_val):
    print("Reject H0")
else:
    print("Accept H0")
```

```
test_statisics:  123.17507450835281
critical_val:  123.2252214533618
Reject H0
```

- The value of test statistics is less than the critical value and hence we reject confidence interval for standard deviation

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.

In [42]:
```
from statsmodels.stats import weightstats
sample_gold = df['gold'].tolist()
sample_oil = df['gold'].tolist()
zstat,p= weightstats.ztest(sample_gold, sample_oil, alternative='two-sided',usevar='poo
if(p<0.05):
    print("Reject H0")
else:
    print("Accept H0")
```

```
Accept H0
```

- Because the P-value is greater than alpha, the null hypothesis is accepted. The mean of two samples is consideredd to be equal

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.

In [43]:
```
from scipy.stats import ttest_1samp,f,t
random_sample = []
n = len(sample_gold)
for i in range(n):
    random_sample.append(sample_gold[i]-sample_oil[i])
tstati, p = ttest_1samp(random_sample,0)
if(p<0.05):
    print("Reject H0")
else:
    print("Accept H0")
```

```
Accept H0
```

- We accept Null hypothesis because the P-value is greater than alpha.

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.

In [44]:
```
fstat = (st.stdev(sample_gold)/st.stdev(sample_oil))
p_value = f.sf(fstat,999,999)
if(p_value<0.05):
    print("Reject H0")
else:
    print("Accept H0")
```

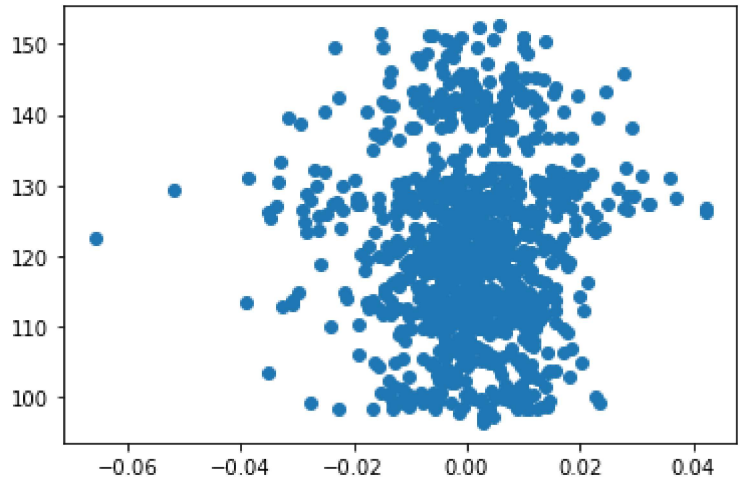
```
Accept H0
```

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?

In [45]:
```
x = df['Close_ETF']
y = df['gold']
plt.scatter(y,x)
plt.show()
```



- From this scatter plot it seems difficult to find a linear relationship between ETF and gold.

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

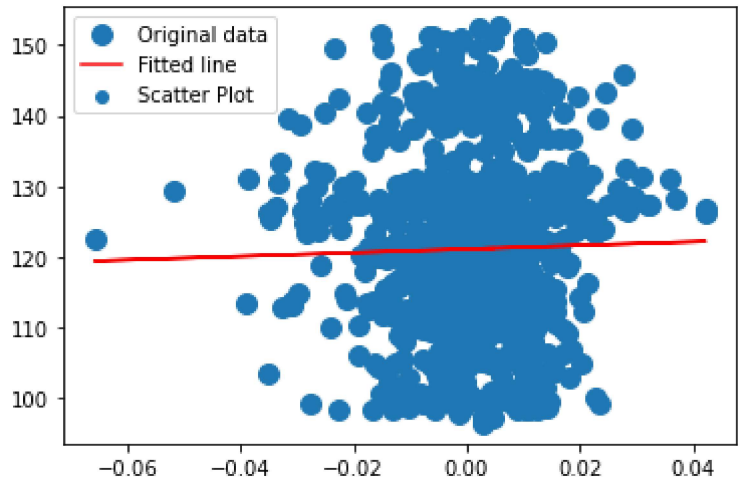
In [46]:
```
correlation = df['Close_ETF'].corr(df['gold'])
print(correlation)
```

```
0.022995570076054603
```

- There's a positive correlation between ETF and Gold but they are weakly correlated

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?

In [47]:
```
y = df['Close_ETF']
x = df['gold']

plt.scatter(x,y, label='Scatter Plot')
A = np.vstack([x, np.ones(len(x))]).T
m, c = np.linalg.lstsq(A, y, rcond=None)[0]
plt.plot(x, y, 'o', label='Original data', markersize=10)
plt.plot(x, m*x + c, 'r', label='Fitted line')
plt.legend()
plt.show()
m, c
```



Out[47]: (25.604389324427057, 121.13598849889821)

- ETF seems increasing as the slpoe is positive (slope is 25.604389324427057) , daily relativechange of gold increases as well and the data have a positive intercept of 121.13598849889821.

4) Conduct a two-tailed t-test with $H0$: $\beta 1=0$. 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 [48]:
```
import math
x = df['Close_ETF']
y = df['gold']
mean_x = np.mean(x)
n = len(y)
ssxx = sum(x*x) - (sum(x)*sum(x)/n)
ssyy = sum(y*y) - (sum(y)*sum(y)/n)
ssxy = sum(x*y) - (sum(x)*sum(y)/n)
b = ssxx/ssxy
s = np.sqrt((ssxx-(b*ssyy))/(n-2))
seb = s/np.sqrt(ssyy)
test_stat = m/seb

p_value = t.sf(test_stat,999)
if(p_value<0.02):
    print("Reject H0")
else:
    print("Accept H0")
```

```
Accept H0
```

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?

In [49]:
```
from sklearn.metrics import r2_score

x = df['Close_ETF']
y = df['gold']
r2 = r2_score(x,y)
r2
```

Out[49]: -92.99113815628421

6) What are the assumptions you made for this model fitting?

- For this model fitting we assume that there is a significant linear relationship between ETF column and gold column.
- The residuals are independent.
- Consecutive residuals in time series data have no correlation.
- The residuals are normally distributed.
- Residuals have constant variance.

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.

In [51]:
```
CI_lower = b - (t*seb)
CI_upper = b + (t*seb)
CI_lower, CI_upper
```

- For ETF and gold, this is not a good fit because there's no relationship between them.

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 $R2$.

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

y = df['Close_ETF']
x = df[['gold', 'oil']]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state
mlr = LinearRegression()
mlr.fit(x_train, y_train)
print("Intercept: ", mlr.intercept_)
print("Coefficients:",list(zip(x, mlr.coef_)))
adj_r2 = 1 - (1-mlr.score(x_train, y_train))*(len(y_train)-1)/(len(y_train)-x_train.sha
print("adjusterd r2:", adj_r2)
```

- oil and gold are not significant in this model. The performance of the model is poor.

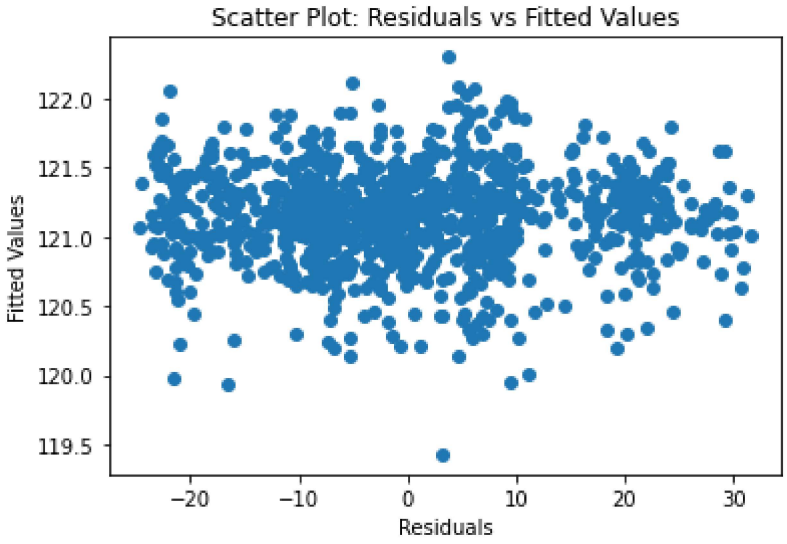
Part 10: Checking residuals and model selection

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.

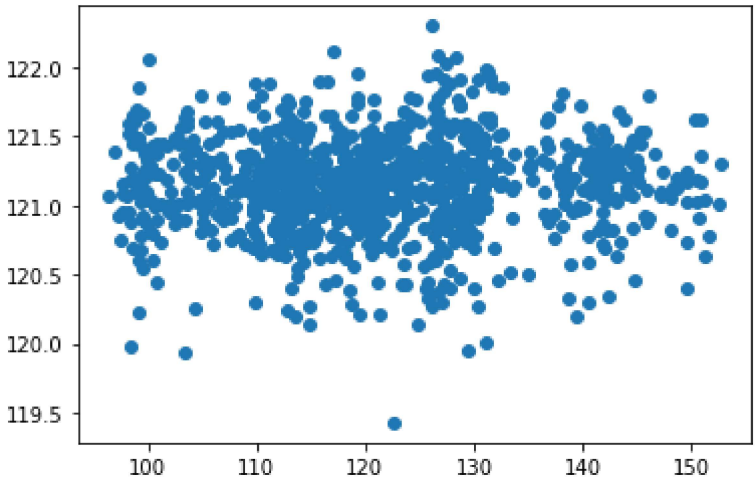
In [58]:
```
# import statsmodels.api as sm

# mlr = sm.OLS(y, x).fit()
# influence = mlr.get_influence()
# standardized_residuals = influence.resid_studentized_internal
# #print(standardized_residuals)


residuals = df['Close_ETF'] - result.fittedvalues
x = df.iloc[ : , 1:3] # Splicing to Only gold and oil
y = df.iloc[ : , 0] # ETF

x = sm.add_constant(x)
model = sm.OLS(y, x)

result = model.fit()

plt.scatter(residuals, result.fittedvalues)
plt.xlabel('Residuals')
plt.ylabel('Fitted Values')
plt.title('Residuals vs Fitted Values')
```

Out[58]:
```
Text(0.5, 1.0, 'Scatter Plot: Residuals vs Fitted Values')
```



In [59]:
```
plt.scatter(df['Close_ETF'], result.fittedvalues)
```

Out[59]:
```
<matplotlib.collections.PathCollection at 0x7fc8505acbe0>
```



Conclusion

- Residual of model do not approve the independence and normality assumptions
- The response variable ETF, do not fit the normal distribution. And hence This data did not give the desired result for linear regression

References:

- Book: Mathematical Statistics and Data Analysis 3rd ed by John A. Rice (z-lib.org)
- https://calcworkshop.com/linear-regression/t-test/
- www.statology.org/standardized-residuals-python/
- http://atomic.phys.uni-sofia.bg/local/nist-e-handbook/e-handbook/eda/section3/eda358.htm
- https://www.analystsoft.com/en/products/statplus/content/help/analysis_basic_statistics_two_samp test.html

In [ ]: