In [5]:
```python
import seaborn as sns
import csv
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
import pandas as pd
from scipy.stats import multivariate_normal
import math
import random
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics import mean_squared_error, r2_score
%matplotlib inline
from sklearn.linear_model import LinearRegression
```

Question 4

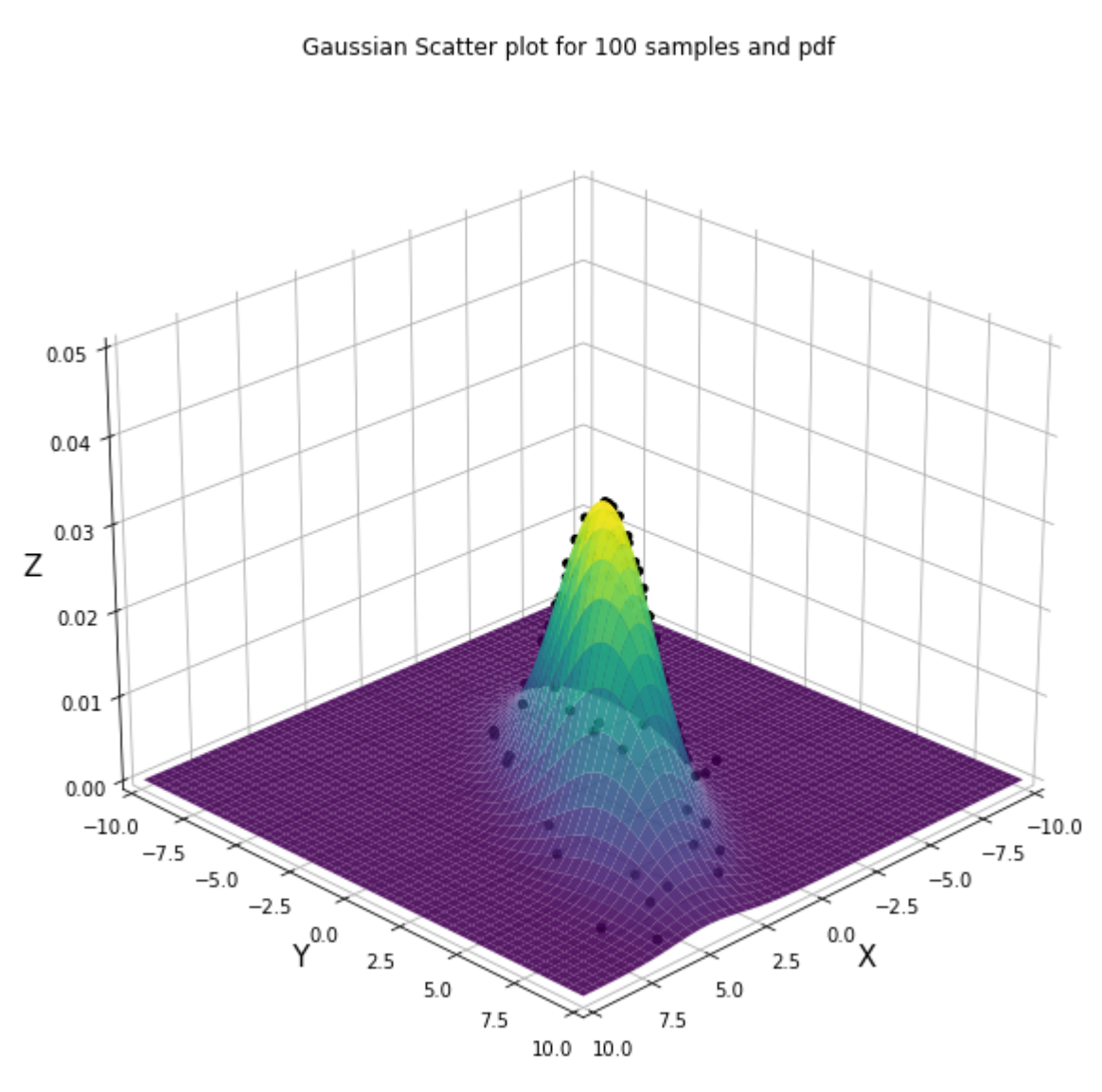
In [29]:
```python
men=np.array([11,13,24,14,20,34,25,22,49,8])
wom=np.array([9,41,10,21,23,25,37,27,15,7])
n1=len(men)
n2=len(wom)
m_mu=men.mean()
w_mu=wom.mean()
m_std=men.std()
w_std=wom.std()
print('Mean men:',m_mu)
print('Mean women:',w_mu)
print('Standard deviation men:',m_std)
print('Standard deviation women:',w_std)
print('Ratio: ' + str(m_std/w_std))
N=(n1-1)*m_std*m_std + (n2-1)*w_std*w_std
D = n1 + n2 -2
Sp=np.sqrt(N/D)
SE = Sp*np.sqrt(1/n1 + 1/n2)
l_95 = [(m_mu -w_mu) - 2.10092*SE,(m_mu-w_mu) + 2.10092*SE]
l_99 = [(m_mu-w_mu) - 2.87844*SE,(m_mu-w_mu) + 2.87844*SE]
print('95% interval: ',l_95)
print('99% interval: ',l_99)
print('Difference in mean:',m_mu-w_mu)
print('Standard Error:',SE)
```

```
Mean men: 22.0
Mean women: 21.5
Standard deviation men: 11.627553482998906
Standard deviation women: 10.984079387914129
Ratio: 1.0585824330250924
95% interval:  [-10.126793123757704, 11.126793123757704]
99% interval:  [-14.05961502539322, 15.05961502539322]
Difference in mean: 0.5
Standard Error: 5.058161721416191
```

Question 5

In [31]:
```python
mean = [1,2]
cov = [[4,4],[4,9]]
val = np.random.multivariate_normal(mean, cov, 100)
print(val.shape)
print()
# generate once
sample_mean =[np.mean(val[:,0]),np.mean(val[:,1])]
print(sample_mean)
print()
temp=np.cov(val[:,0],val[:,1])
print(temp)
print()
# generate 10 times
sample_mean=np.zeros(2)
sample_cov=[[0,0],[0,0]]
sample_cov=np.array(sample_cov)
for i in range(10):
    val = np.random.multivariate_normal(mean, cov, 100)
    sample_mean =sample_mean + np.array([np.mean(val[:,0]),np.mean(val[:,1])])
    sample_cov = sample_cov + np.cov(val[:,0],val[:,1])

sample_mean = sample_mean/10
sample_cov = sample_cov/10
print(sample_mean)
print()
print(sample_cov)
print()
N=[20,40,60,80,100,200,300,400,500]
df = pd.DataFrame(columns=['Mean','Co-variance','Mean RMSE','Cov RMSE'],index=N)

for j in N:
    sample_mean=np.zeros(2)
    sample_cov=[[0,0],[0,0]]
    sample_cov=np.array(sample_cov)
    for i in range(int(j)):
        val = np.random.multivariate_normal(mean, cov, 100)
        sample_mean =sample_mean + np.array([np.mean(val[:,0]),np.mean(val[:,1])])
        sample_cov = sample_cov + np.cov(val[:,0],val[:,1])

    sample_mean = sample_mean/j
    sample_cov = sample_cov/j
    sample_mean = np.around(sample_mean,decimals=3)
    sample_cov = np.around(sample_cov,decimals=3)

    df.loc[j,'Mean']=sample_mean
    df.loc[j,'Co-variance']=sample_cov
    df.loc[j,'Mean RMSE']=np.sqrt(mean_squared_error(mean,sample_mean))
    df.loc[j,'Cov RMSE']=np.sqrt(mean_squared_error(cov,sample_cov))
    # print(sample_mean.shape)
    # print(sample_cov.shape)
pd.set_option('display.max_columns', None)
rv = multivariate_normal(mean, cov)
x_abs = 10
y_abs = 10
x_grid, y_grid = np.mgrid[-x_abs:x_abs:.1, -y_abs:y_abs:.1]
pos = np.empty(x_grid.shape + (2,))
pos[:, :, 0] = x_grid
pos[:, :, 1] = y_grid
fig = plt.figure(figsize=(10,10))
ax = fig.gca(projection='3d')
# # Removes the grey panes in 3d plots
ax.xaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
ax.yaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
ax.zaxis.set_pane_color((1.0, 1.0, 1.0, 0.0))
ax.plot_surface(x_grid, y_grid, rv.pdf(pos),cmap='viridis',linewidth=0,alpha=0.9,label='PDF function')
ax.scatter(val[:, 0], val[:, 1], rv.pdf(val), c='k',alpha=1,label='Simulated Sample')
# ax.legend(['PDF function','Simulated Sample'])
ax.set_title("Gaussian Scatter plot for 100 samples and pdf")
ax.set_xlim3d(-x_abs, x_abs)
ax.set_ylim3d(-y_abs, y_abs)
ax.set_zlim3d(0, 0.05)
ax.set_xlabel('X',fontsize=15)
ax.set_ylabel('Y',fontsize=15)
ax.set_zlabel('Z',fontsize=15)
# set viewing angle
ax.view_init(25, 45)
plt.show()
df
```

```
(100, 2)

[1.2605621383142365, 1.6906173278342917]

[[3.80323201 4.17095108]
 [4.17095108 9.73777845]]

[0.9366205  1.94547468]

[[4.09775728 4.21376932]
 [4.21376932 9.01550661]]
```



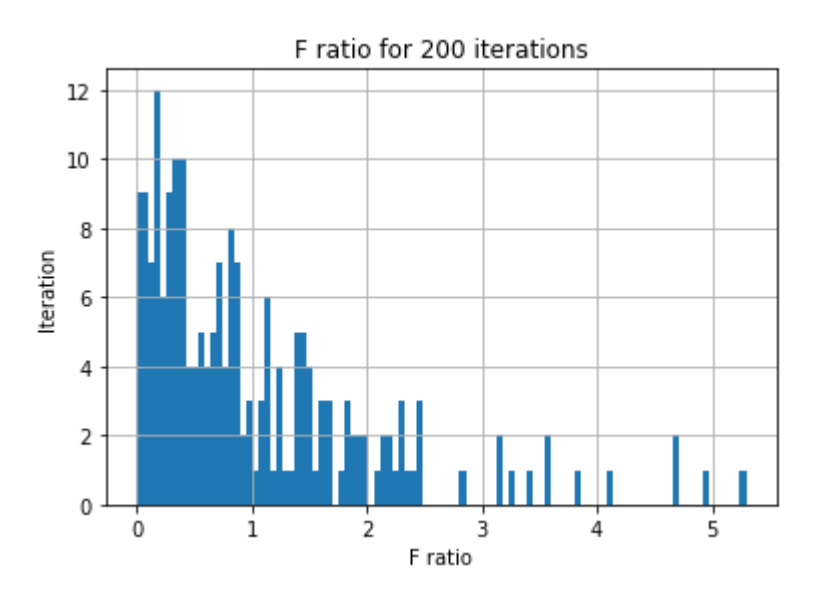
Out[31]:

	Mean	Co-variance	Mean RMSE	Cov RMSE
20	[0.994, 2.03]	[[3.956, 4.028], [4.028, 9.193]]	0.0216333	0.100937
40	[0.972, 1.986]	[[3.893, 3.825], [3.825, 8.787]]	0.0221359	0.171805
60	[1.03, 2.039]	[[4.033, 4.076], [4.076, 9.029]]	0.0347922	0.058056
80	[1.003, 2.013]	[[3.992, 3.986], [3.986, 8.991]]	0.00943398	0.0115866
100	[0.992, 1.989]	[[3.923, 3.912], [3.912, 8.986]]	0.00961769	0.0735068
200	[0.994, 1.979]	[[3.981, 3.956], [3.956, 8.849]]	0.0154434	0.0822101
300	[0.99, 1.979]	[[4.003, 3.997], [3.997, 8.98]]	0.0164469	0.010332
400	[1.0, 1.979]	[[4.002, 4.006], [4.006, 9.005]]	0.0148492	0.00502494
500	[0.988, 1.984]	[[3.994, 3.978], [3.978, 8.978]]	0.0141421	0.0192873

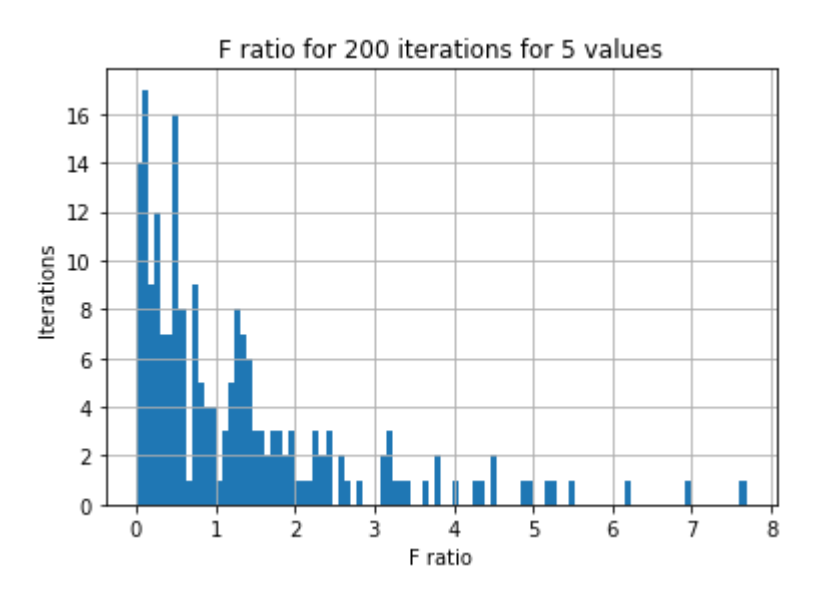
Question 6

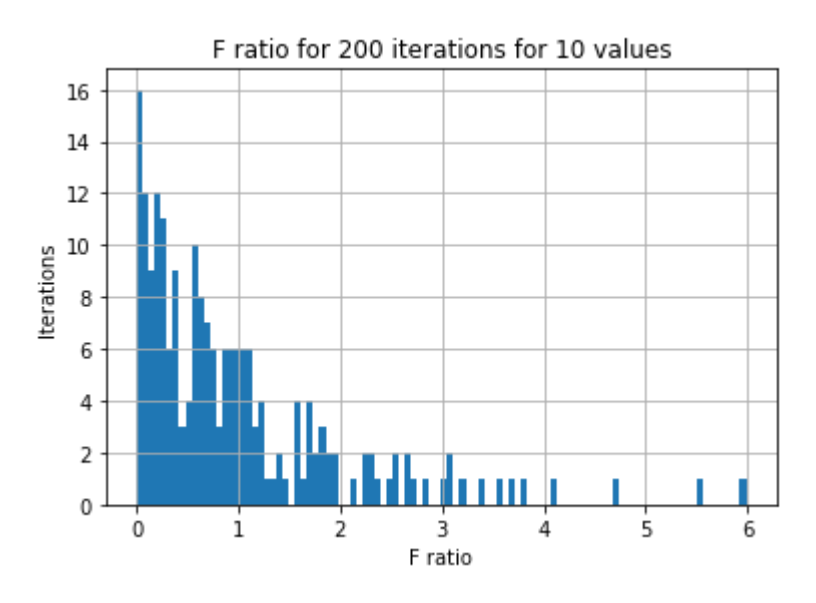
In [33]:
```python
def calculate_deviation(x):
    mean_x = sum(x)/len(x)
    t = [(i-mean_x)**2 for i in x]
    return sum(t)

mu = 60
sigma = 12
np.random.seed(10)
a = np.random.normal(mu, sigma, 10)
b = np.random.normal(mu, sigma, 10)
c = np.random.normal(mu, sigma, 10)

mean_a = np.mean(a)
mean_b = np.mean(b)
mean_c = np.mean(c)
grand_mean = (mean_a + mean_b + mean_c) / 3
SSB = 10*((mean_a - grand_mean)**2 + (mean_b - grand_mean)**2 + (mean_c - grand_mean)**2)
SSE = calculate_deviation(a) + calculate_deviation(b) + calculate_deviation(c)
MSB = SSB/2
MSE = SSE/27
F = MSB/MSE
print(mean_a,mean_b,mean_c,grand_mean,SSB,SSE,MSB,MSE,F)

mu = 60
sigma = 12
f_values = []
np.random.seed(10)
for i in range(200):
    a = np.random.normal(mu, sigma, 10)
    b = np.random.normal(mu, sigma, 10)
    c = np.random.normal(mu, sigma, 10)
    mean_a = np.mean(a)
    mean_b = np.mean(b)
    mean_c = np.mean(c)
    grand_mean = (mean_a + mean_b + mean_c) / 3
    SSB = 10*((mean_a - grand_mean)**2 + (mean_b - grand_mean)**2 + (mean_c - grand_mean)**2)
    SSE = calculate_deviation(a) + calculate_deviation(b) + calculate_deviation(c)
    MSB = SSB/2
    MSE = SSE/27
    F = MSB/MSE
    f_values.append(F)

plt.hist(f_values,bins=100)
plt.title('F ratio for 200 iterations')
plt.xlabel('F ratio')
plt.ylabel('Iteration')
plt.grid()
plt.show()

count = 0
for f in f_values:
    if(f>3.35): count=count+1
print(count/len(f_values))

count_vals = []
mu = 60
sigma = 12
vals = [5,10,15,25,30]
np.random.seed(10)
for i in vals:
    f_values = []
    for j in range(201):
        a = np.random.normal(mu, sigma, i)
        b = np.random.normal(mu, sigma, i)
        c = np.random.normal(mu, sigma, i)
        mean_a = np.mean(a)
        mean_b = np.mean(b)
        mean_c = np.mean(c)
        grand_mean = (mean_a + mean_b + mean_c) / 3
        SSB = i*((mean_a - grand_mean)**2 + (mean_b - grand_mean)**2 + (mean_c - grand_mean)**2)
        SSE = calculate_deviation(a) + calculate_deviation(b) + calculate_deviation(c)
        MSB = SSB/2
        MSE = SSE/(3*i - 3)
        F = MSB/MSE
        f_values.append(F)
    count = 0
    for f in f_values:
        if(f>3.35):count=count+1
    count_vals.append(count/len(f_values))
    plt.hist(f_values,bins=100)
    plt.title('F ratio for 200 iterations for ' + str(i) + ' values')
    plt.xlabel('F ratio')
    plt.ylabel('Iterations')
    plt.grid()
    plt.show()

plt.plot(vals,count_vals)
plt.title('Proportion of iterations for which F-ratio exceeds 3.35')
plt.xlabel('Data size')
plt.ylabel('Proportion')
plt.grid()
plt.show()
```

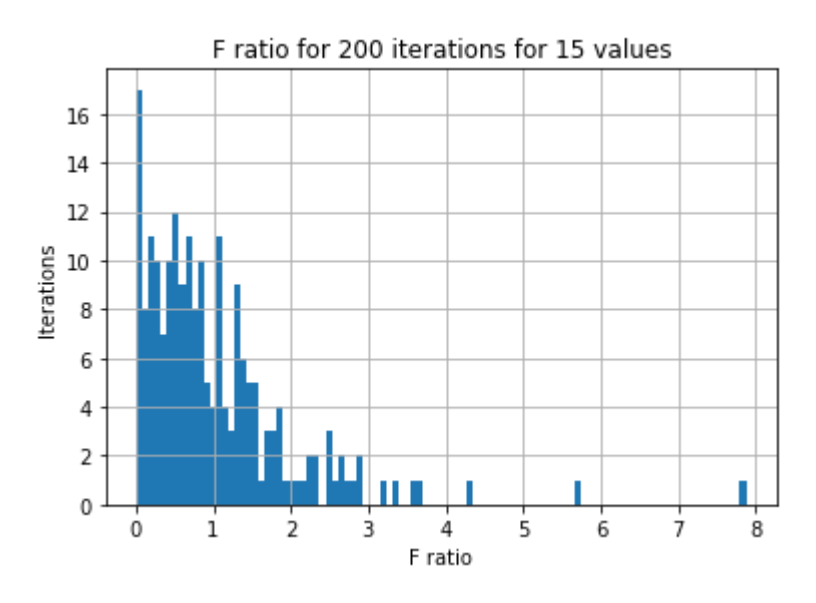
```
60.71769969768949 62.13156779622678 64.27842260963371 62.37589670118333 64.28918748518151 4580.0125084983265 32.14459
374259076 169.63009290734541 0.18949817919482384
```

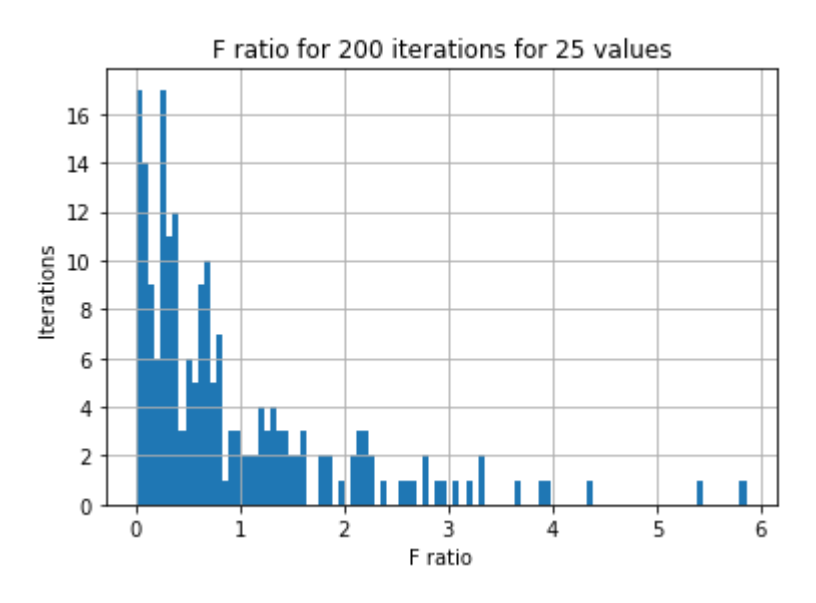


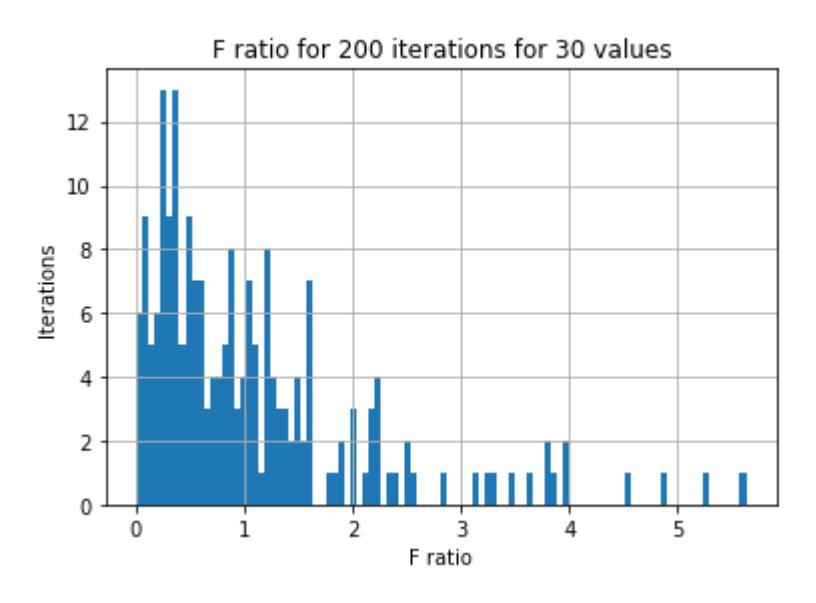

```
0.045
```

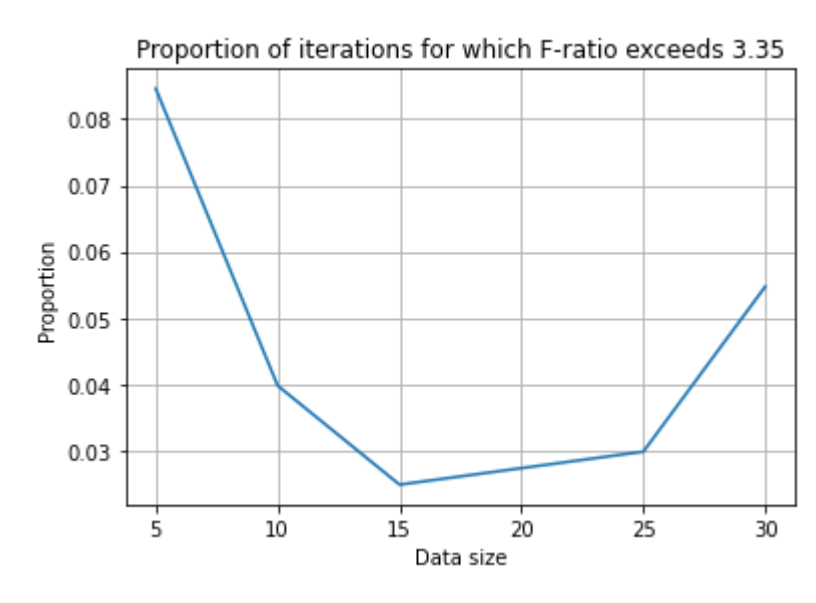






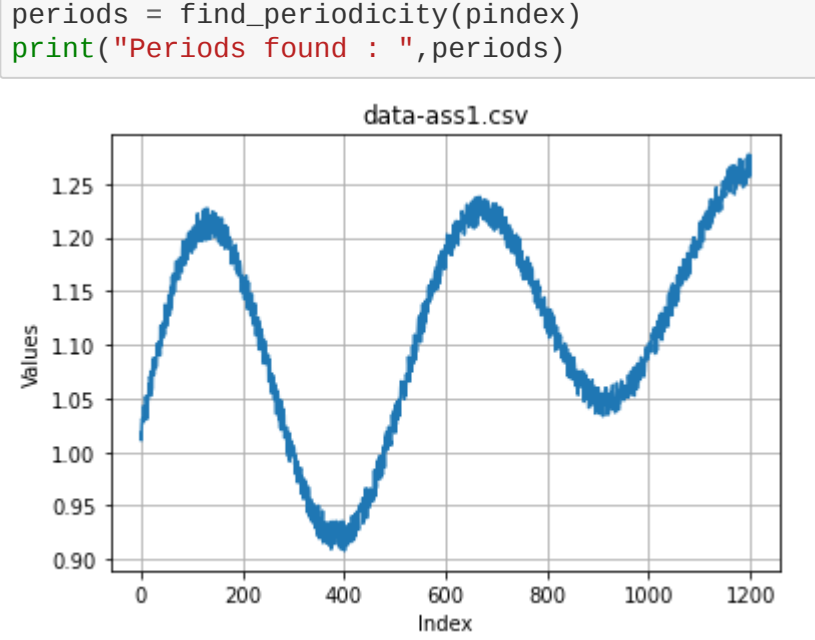




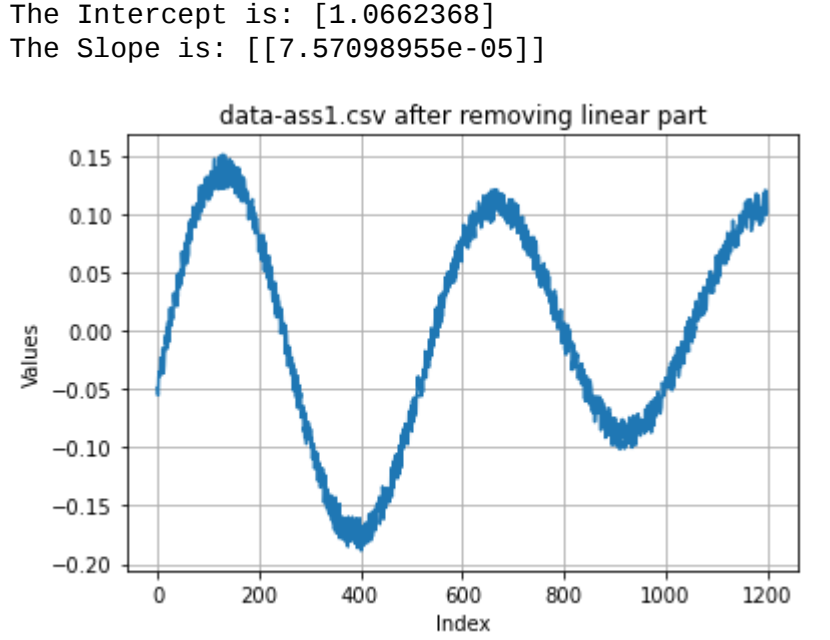


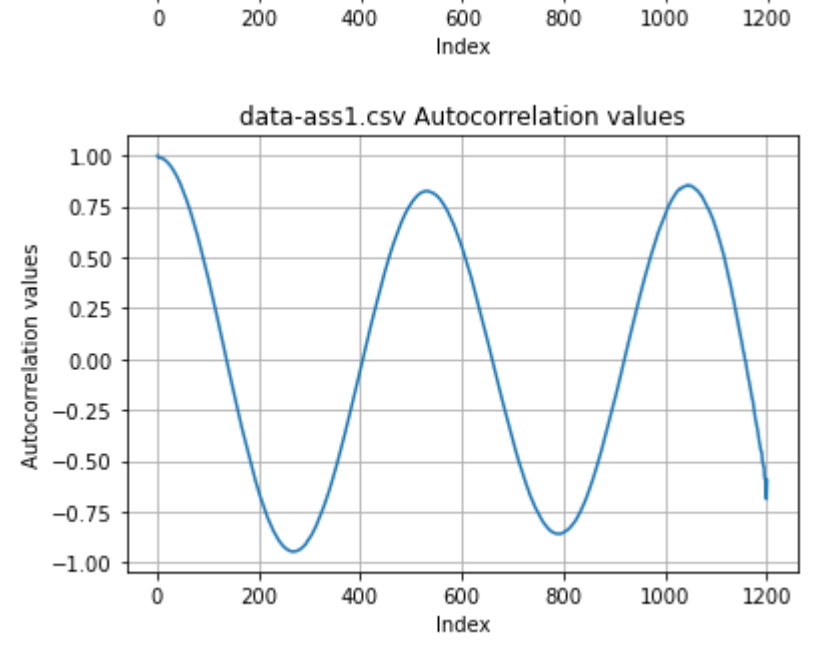
Question 7

In [11]:
```python
df = pd.read_csv(r'data-ass1.csv')
y = df["x"].values
plt.plot(y)
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('data-ass1.csv')
plt.grid()
plt.show()
def calculate_autocorrelation(x):
    n = len(x)
    variance = x.var()
    x = x-x.mean()
    r = np.correlate(x, x, mode = 'full')[-n:]
    result = r/(variance*(np.arange(n, 0, -1)))
    return result

def find_peak_indexes(x):
    y = max(x)
    print(y)
    pindex = []
    pindex.append(0)
    for i in range(1,len(x)-1):
        if(x[i-1]<x[i] and x[i+1]<x[i]):
               pindex.append(i)
    return pindex
def find_periodicity(x):
    ans = []
    for i in range(1,len(x)):
        ans.append(x[i] - x[i-1])
    return ans

index = np.arange(1,len(y)+1,1).reshape(-1,1)
y = y.reshape(-1,1)
linear_regressor = LinearRegression().fit(index,y)
beta_o = linear_regressor.intercept_
beta_1 = linear_regressor.coef_
#linear_regressor.fit(x, y)
print('The Intercept is:',beta_o )
print('The Slope is:', beta_1)

y_predicted = linear_regressor.predict(index)
y = y - y_predicted
plt.title("data-ass1.csv after removing linear part")
plt.xlabel("Index")
plt.ylabel("Values")
plt.grid()
plt.plot(y)
plt.show()

y = np.ndarray.flatten(y)
auto_corr = calculate_autocorrelation(y)
plt.plot(index,auto_corr)
plt.xlabel('Index')
plt.ylabel('Autocorrelation values')
plt.title('data-ass1.csv Autocorrelation values')
plt.grid()
plt.show()

pindex = find_peak_indexes(auto_corr)
print("Peak Indices : ",pindex)

periods = find_periodicity(pindex)
print("Periods found : ",periods)
```




```
The Intercept is: [1.0662368]
The Slope is: [[7.57098955e-05]]
```






```
1.0000000000000002
Peak Indices :  [0, 3, 265, 269, 521, 528, 531, 533, 784, 786, 791, 1029, 1036, 1042, 1045, 1049]
Periods found :  [3, 262, 4, 252, 7, 3, 2, 251, 2, 5, 238, 7, 6, 3, 4]
```

In [ ]: