

Problem 1: A Bayesian Example

A medical testing company has developed a two-stage, non-invasive test for Down's syndrome in the first two months of pregnancy. Initial tests have shown that when the fetus actually has Down's syndrome, the test correctly identifies this by giving a positive result 98% of the time (i.e. the observed probability of a false negative is 2%). Furthermore, in cases when the fetus does not have Down's syndrome, the test result is negative 97% of the time (i.e. the observed probability of a false positive is 3%). Armed with this data, the company's statisticians are charged with estimating the overall probability that a given fetus has Down's syndrome if the test shows positive. Assume the observed probability of Down's syndrome in a random sample from a large number of observations is about 1 in 1,200 (this is the reported value for a 25-year-old mother; the risk is higher for older mothers).

A. ¶

Using Bayes' theorem, estimate the overall probability that a given fetus has Down's syndrome if the test shows positive. Are you surprised by the answer? What is the fundamental problem with this test? How accurate would the test have to be to achieve a 50% probability of having the disease if the test showed a positive result? (For the last one, experiment with the accuracies of both the positive and negative test results.)

Answer

According to the Bayes' theorem

$$P(H|E) = P(E|H) * P(H) / P(E)$$

Where

For positive results:

$P(H|E)$ =probability that given fetus has syndrome when test shows positive

$P(E|H)$ = probability that test shows positive when given fetus has syndrome=0.98

$P(H)$ =probability that some one have a syndrome=1/1200

$P(E)$ =probability that have a positive result=0.98/1200+0.03*1199/1200

In [2]:

```
print('Probability that a given fetus has Downs syndrome if test shows positive=', (0.98/1200)/(0.98/1200+0.03*1199/1200))
```

Probability that a given fetus has Downs syndrome if test shows positive= 0.02652232746955345

The result was suprisingly low than my expectation. The fundamental problem is that the calculated results from the given condition is actually much lower than our expected.

To reach 50% accuracy, we have the equation from above as:

$$0.5 = (x/1200)/(x/1200 + (1 - x) * 1199/1200)$$

Where x = true positive probability from the test

Solving the equation we get $x=1199/1200$ is the accuracy we need to achieve 50% probability for pos result

For the negative one we also have the equation as:

$$0.5 = (x * 1199/1200)/(x * 1199/1200 + (1 - x)/1200)$$

Where x = true positive probability from the test

Solving the equation we get $x=1/1200$ is the accuracy we need to achieve 50% probability for neg result

B.

Estimate the likelihood that the fetus does not have Down's Syndrome if the test shows a negative result. Is the test more useful in this framework as a screening step when trying to decide to use more accurate (but also more invasive) tests?

Using the same equation from A we have:

$$P(H|E) = P(E|H) * P(H)/P(E)$$

Where

For negative results:

$P(H|E)$ =probability that given fetus doesn't have syndrome when test shows negative

$P(E|H)$ = probability that test shows negative when given fetus doesn't have syndrome=0.97

$P(H)$ =probability that some one don't have a syndrome=1199/1200

$P(E)$ =probability that have a negative result=0.97 1199/1200+0.031/1200

In [3]:

```
print('Probability that a given fetus dont have Downs syndrome if test shows negative=', (0.97*1199/1200)/(0.97*1199/1200+0.03*1/1200))
```

Probability that a given fetus dont have Downs syndrome if test shows negative= 0.9999742059738964

From the result, this test is more useful during the screening processes.

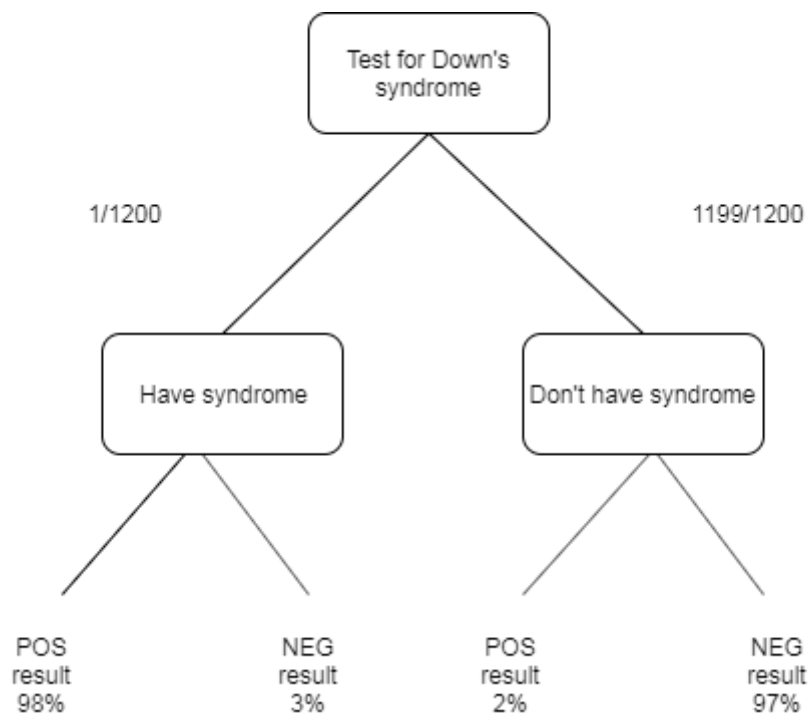
C.

Finally, draw a tree diagram for this problem.

In [4]:

```
from IPython.display import Image  
Image(filename='Tree diagram.png')
```

Out[4]:



Problem 2: Continuous Bayesian Estimation of Flood Frequency

Data:

NYC_precip_priors.xlsx Based on long-term climate records, the 24-hour duration, 100-year return period rainfall in New York City was previously estimated at 7.2 inches. 100-year rainfall indicates a storm with a rainfall total that would on average be met or exceeded only once every 100 years. In a given year, the storm has a 0.01 chance of occurrence, i.e., for any k -year storm, its probability of occurrence in one year is $1/k$, where k is the return period. In fact, during the 20th century, this total was exceeded only once, during Hurricane Floyd in 1999. However, a storm in 2007 produced 8 inches of rainfall in a 24-hour period. Another storm in August 2011 again exceeded the 7.2-inch total. Hurricane Irene in September 2011 exceeded the total yet again, and so did Hurricane Sandy in October 2012. This happened again in September 2018 from Hurricane Florence. [Note that in 2019, there was flooding in July, but not of this magnitude. Also, subtropical Storm Melissa brought minor flooding to the New York coastline on 11 October 2019 but not of this magnitude. Therefore, we will not update the number of floods in 2019 but will instead add one more year without a flood.]

In this problem we will use Bayes' Theorem to estimate the probability that the 7.2-inch storm is no longer the 100-year storm, that is, the probability that 7.2 inches of rainfall in a 24-hour period is likely to occur more often than every 100 years. In this problem, A is the true likelihood of the storm's occurrence, and B are the events we have observed. [Note that unlike what you will do in many hydrology classes, which is to calculate the return period across a range of different precipitation or streamflow values, we are focusing here only on the probability of 24-hour storms exceeding 7.2 inches of rainfall because this is a critical design number for New York City.]

A.

Download and plot the prior likelihood of a ≥ 7.2 -inch storm occurring in a given year from the data file "NYC_precip_priors" (.xlsx or .mat). This dataset shows the prior distribution of the likelihood of the 7.2-inch storm in a given year, with mean $p = 0.01$. Note that the variable labeled "pdf" represents the probability of the storm frequency or return interval falling within that interval (value noted and the next value) and therefore includes the interval width, such that the sum of the pdf values alone sums to 1, and cumulative sum of the pdf is the cdf. Note that there is some uncertainty about the original likelihood. What are the mean and 95% confidence interval of the probability, i.e., where does 95% of the PDF fall between? To what k -year storm (expected return period) range do these values correspond?

In [5]:

```
import numpy as np
import scipy.stats as st
import statistics as stats
import matplotlib.pyplot as plt
import pandas as pd

%matplotlib inline
```

In [6]:

```
df=pd.read_excel('NYC_precip_priors.xlsx')
df=df.iloc[:, :3]
```

In [7]:

df.head()

Out[7]:

	PDF	Storm Probability	Return Period (years)
0	0.038961	0.000	NaN
1	0.038922	0.001	1000.000000
2	0.038786	0.002	500.000000
3	0.038554	0.003	333.333333
4	0.038228	0.004	250.000000

In [8]:

```

# number of periods
m = 3
# number of years on each periods
n = [100, 11, 9]

# number of 7.2inch-storm on each periods
c = [1, 1, 4]

nx=100

# create timeseries x evenly spaced with 100 values between 0 and 1
x = np.linspace(0, 0.1, num=nx+1)
x = np.delete(x, -1)

p_bar = sum(c) / sum(n)

ones_and_zeros = []
for i in range(sum(c)):
    ones_and_zeros.append(1)
for j in range(sum(n)-sum(c)):
    ones_and_zeros.append(0)
s_bar = stats.stdev(ones_and_zeros)
# Note that s_bar determines the standard deviation in the number of right
# answers, not the standard deviation of the probability of success.

# Over n trials, the variance of the number of successes/failures
# is measured by
#  $\sigma^2 = n * p * (1-p)$ 
# and the variance of the probability of success is the above variance
# divided by  $n^2$ 
# so  $\sigma_p = \sqrt{\sigma^2 / n^2} = \sqrt{p * (1-p) / n}$  % as written below
nonBayes_pdf = st.norm.pdf(x, p_bar, np.sqrt(p_bar * (1-p_bar) / (sum(n))))
nonBayes_cdf = (x[1] - x[0]) * np.cumsum(nonBayes_pdf)

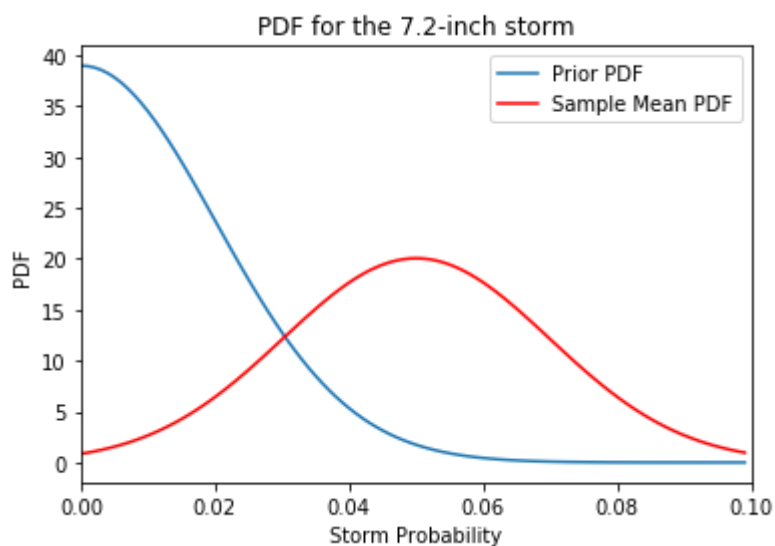
```

In [9]:

```
plt.plot(df['Storm Probability'], df['PDF']*1000, label='Prior PDF')
plt.plot(x, nonBayes_pdf, color='red', label='Sample Mean PDF')
plt.xlabel('Storm Probability')
plt.ylabel('PDF')
plt.title('PDF for the 7.2-inch storm')
plt.xlim(0, 0.1)
plt.legend()
```

Out[9]:

<matplotlib.legend.Legend at 0xada0c50>



In [10]:

```
x_df=np.array(df['Storm Probability'])
cdf = np.cumsum(df['PDF'])
```

In [11]:

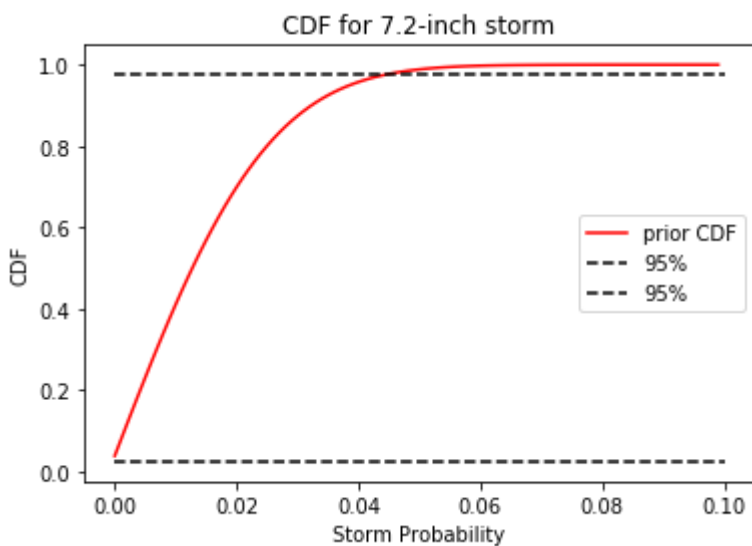
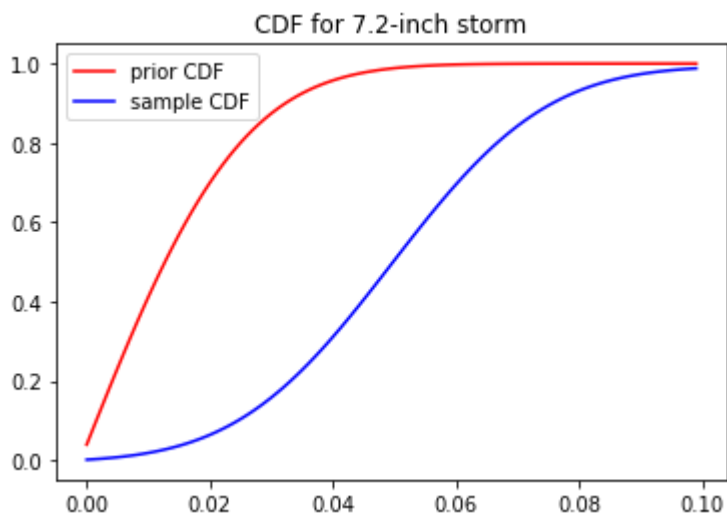
```
# Look for 95% Confidence intervals
# 95% range is [0.000025, 0.000975]

plt.figure()
plt.plot(x_df, cdf, 'r', label='prior CDF')
plt.plot(x, nonBayes_cdf, 'b', label='sample CDF')
plt.title('CDF for 7.2-inch storm')
plt.legend()

plt.figure()
plt.plot(x_df, cdf, 'r', label='prior CDF')
plt.xlabel('Storm Probability')
plt.ylabel('CDF')
plt.title('CDF for 7.2-inch storm')
plt.plot([0, 0.1], [0.025, 0.025], 'k--', label='95%')
plt.plot([0, 0.1], [0.975, 0.975], 'k--', label='95%')
plt.legend()
```

Out[11]:

<matplotlib.legend.Legend at 0xb0bd908>



In [12]:

```

from scipy.interpolate import interp1d
f2 = interp1d(cdf, x)
print('mean=', f2(0.5))
print('mean return period=', 1/f2(0.5))

print(f2(0.975))
print(1/f2(0.975))
#print(f2(0.025)) is out of range

```

```

mean= 0.012717473075032229
mean return period= 78.63197304213406
0.044250694049779014
22.59851560463816

```

According to the graph above, the 95% confidence interval falls between 0-0.04425 of storm probability, the returning period for 95% confidence interval is 22.59 years(23 years)~unlimited.

B.

At each time period of interest, count the number of times the threshold has been exceeded and the number of years, beginning with the period 1900-1999. The likelihood $P(B|A = p)$ that the storm would be exceeded m times in n years (event B) for a storm probability p (event $A = p$) is given by:

$$P(B|A = p) = \text{binomdist}(m, n, p)$$

Calculate the posterior likelihood of the 7.2-inch storm's probability after the period 1900-1999, which includes only Hurricane Floyd. Use the continuous version of Bayes' Theorem to calculate the probability of A at all values of p . What are the mean and the 95% confidence interval of the return period for the 7.2-inch storm after 1999?

In [13]:

```

# information in just the first test
likelihood = []
for i in range(0, nx):
    # calculate likelihood of test score given student's "true" fraction correct
    likelihood.append(st.binom.pmf(c[0], n[0], x[i]))
    # Note, this is P(B|A)

```

In [14]:

```

prior_pdf=df['PDF']
prior_cdf=cdf

```

In [15]:

```

# Use that information to update the prior for a new, posterior pdf
# Calculate the posterior likelihood using Bayes' Theorem

post_pdf1 = []
for j in range(0, nx-1):
    post_pdf1.append(likelihood[j] * prior_pdf[j])
    # Note that I'm not normalizing yet. I'm calculating P(B|A)*P(A)

```


In [16]:

```
# Now consider each test in sequence (some of this is repeat from above;
# above was for illustration)

post_pdf = np.empty([m+1, nx])
post_cdf = np.empty([m+1, nx])
post_pdf.fill(np.nan)
post_cdf.fill(np.nan)

post_pdf[0] = prior_pdf # This is P(A) for all x
post_cdf[0] = prior_cdf

likelihood = np.empty([m, nx])
likelihood.fill(np.nan)

for i in range(0, m): # loop through each test
    for j in range(0, nx): # loop through each incremental fraction correct
        # calculate likelihood of test score given students' "true" fraction correct
        likelihood[i, j] = st.binom.pmf(c[i], n[i], x[j])
        #now calculate the posterior likelihood using Bayes' Theorem
        post_pdf[i+1, j] = likelihood[i, j] * post_pdf[i, j]

# normalize so that area under curve is 1 and then calculate CDF
post_pdf[i + 1] = post_pdf[i + 1]/((x[1] - x[0])*np.nansum(post_pdf[i + 1]))
# right here we are normalizing by P(B) which is SUM of P(B/A)*P(A)
# over all values of A, applying the chain rule
post_cdf[i + 1] = (x[2] - x[1])*np.cumsum(post_pdf[i + 1])
```

In [17]:

```
# Make the plot comparing the first and second steps

y_likelihood = np.zeros_like(likelihood)
for i in range(likelihood.shape[0]):
    for j in range(likelihood.shape[1]):
        y_likelihood[i, j] = likelihood[i, j] / (np.sum(likelihood[i]) * (x[1] - x[0]))

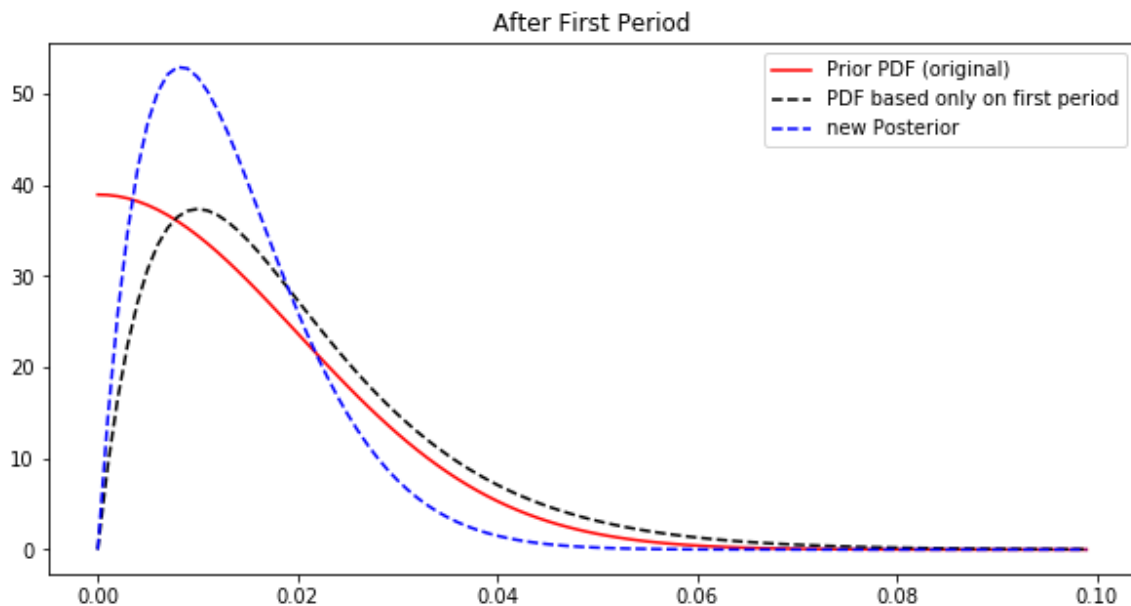
#f, ax = plt.subplots(1, 1, figsize=(7, 7))

plt.figure(figsize=(10, 5))
plt.plot(x, post_pdf[0]*1000, 'r', label='Prior PDF (original)')
plt.plot(x, y_likelihood[0], 'k--', label='PDF based only on first period')
plt.plot(x, post_pdf[1], 'b--', label='new Posterior')
plt.title('After First Period')
plt.legend(loc='best')

#ax[1].plot(x, post_pdf[1]*0.001, 'r', label='Prior PDF (posterior from plot above)')
#ax[1].plot(x, y_likelihood[1]*0.001, 'k--', label='PDF based only on second period')
#ax[1].plot(x, post_pdf[2]*0.001, 'b--', label='new Posterior')
#ax[1].set_title('After Second Period')
#ax[1].legend(loc='best');
```

Out[17]:

<matplotlib.legend.Legend at 0xb160fd0>

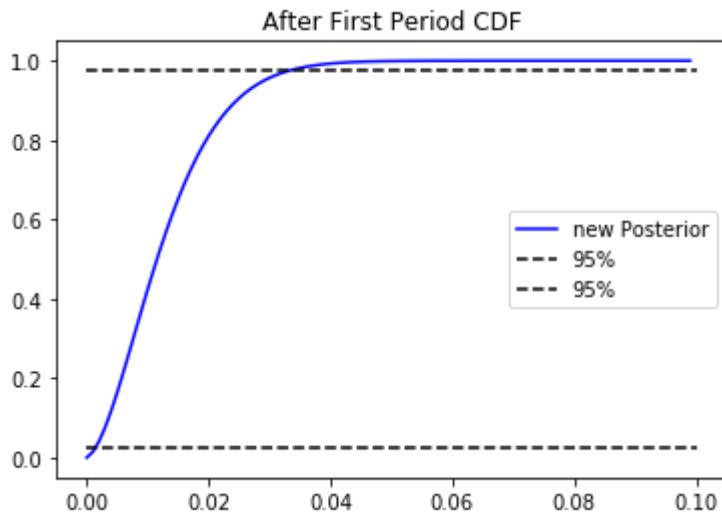


In [18]:

```
plt.plot(x, post_cdf[1], 'b', label='new Posterior')
plt.title('After First Period CDF')
plt.plot([0, 0.1], [0.025, 0.025], 'k--', label='95%')
plt.plot([0, 0.1], [0.975, 0.975], 'k--', label='95%')
plt.legend(loc='best')
```

Out[18]:

<matplotlib.legend.Legend at 0xbla3f98>



In [19]:

```
f_p = interp1d(post_cdf[1], x)
print('mean=', f_p(0.5))
print('mean return period=', 1/f_p(0.5))

print('lower 95% confidence interval value=', f_p(0.025))
print('lower 95% confidence interval return period=', 1/f_p(0.025))
print('higher 95% confidence interval value=', f_p(0.975))
print('higher 95% confidence interval return period=', 1/f_p(0.975))
```

```
mean= 0.011536773823693483
mean return period= 86.67934513427527
lower 95% confidence interval value= 0.0014116352846001673
lower 95% confidence interval return period= 708.398274617541
higher 95% confidence interval value= 0.0332312992175132
higher 95% confidence interval return period= 30.092112663262675
```

C.

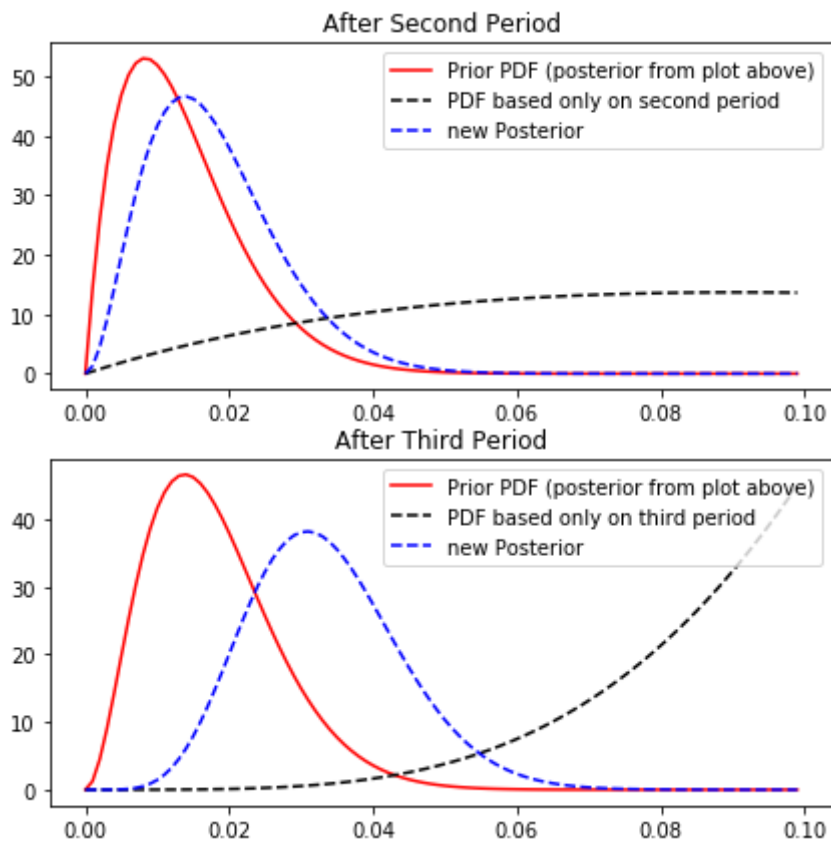
Now update the posterior likelihood of the storm's probability by considering the period from 2000-2010 (storm 2), and then the period 2011-2019 (storms 3, 4, 5, and 6). Remember to change the number of storms (m) and the number of years (n) at each step. Use the previous step's posterior distribution as the new prior distribution at each step, and update only with the new years. Plot the posterior distributions after each update, and state the mean and 95% confidence interval of the return period each time (as in B but for the updated distributions).

In [20]:

```
f, ax = plt.subplots(2, 1, figsize=(7, 7))

ax[0].plot(x, post_pdf[1], 'r', label='Prior PDF (posterior from plot above)')
ax[0].plot(x, y_likelihood[1], 'k--', label='PDF based only on second period')
ax[0].plot(x, post_pdf[2], 'b--', label='new Posterior')
ax[0].set_title('After Second Period')
ax[0].legend(loc='best');

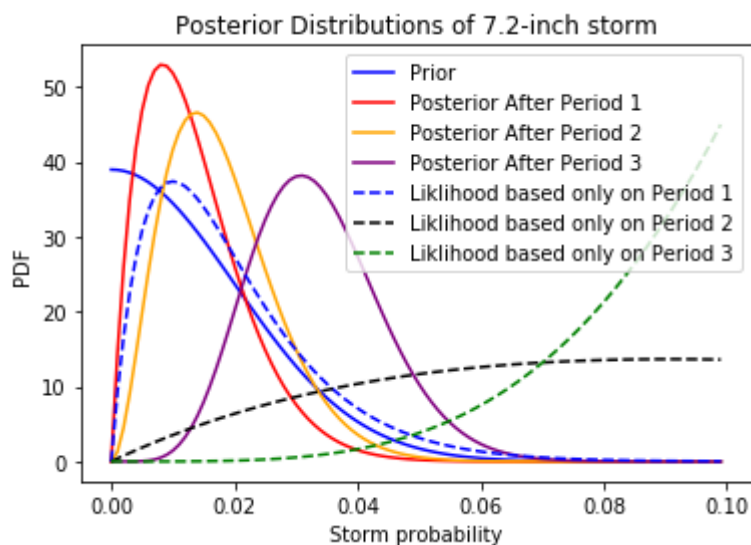
ax[1].plot(x, post_pdf[2], 'r', label='Prior PDF (posterior from plot above)')
ax[1].plot(x, y_likelihood[2], 'k--', label='PDF based only on third period')
ax[1].plot(x, post_pdf[3], 'b--', label='new Posterior')
ax[1].set_title('After Third Period')
ax[1].legend(loc='best');
```



In [21]:

```
# plot posterior distributions
```

```
plt.figure()
plt.plot(x, post_pdf[0]*1000, 'b', label='Prior')
plt.plot(x, post_pdf[1], 'r', label='Posterior After Period 1')
plt.plot(x, post_pdf[2], 'orange', label='Posterior After Period 2')
plt.plot(x, post_pdf[3], 'purple', label='Posterior After Period 3')
plt.plot(x, y_likelihood[0], 'b--', label='Likelihood based only on Period 1')
plt.plot(x, y_likelihood[1], 'k--', label='Likelihood based only on Period 2')
plt.plot(x, y_likelihood[2], 'g--', label='Likelihood based only on Period 3')
plt.xlabel('Storm probability')
plt.ylabel('PDF')
plt.legend(loc='best')
plt.title('Posterior Distributions of 7.2-inch storm');
```

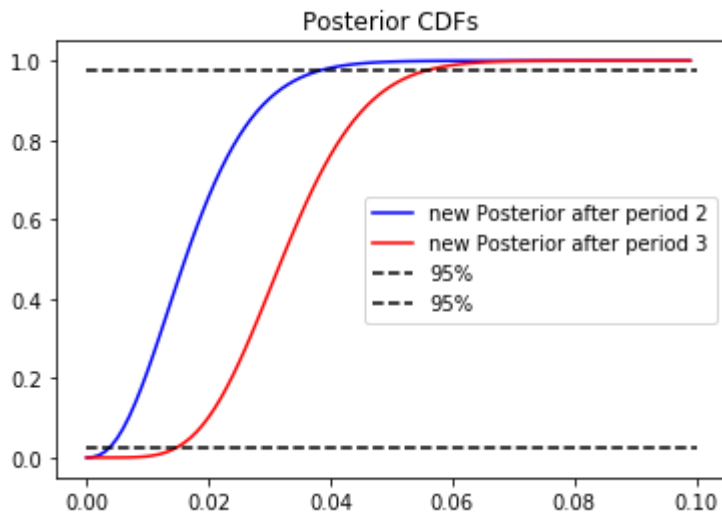


In [22]:

```
plt.plot(x, post_cdf[2], 'b', label='new Posterior after period 2')
plt.plot(x, post_cdf[3], 'r', label='new Posterior after period 3')
plt.title('Posterior CDFs')
plt.plot([0, 0.1], [0.025, 0.025], 'k--', label='95%')
plt.plot([0, 0.1], [0.975, 0.975], 'k--', label='95%')
plt.legend(loc='best')
```

Out[22]:

<matplotlib.legend.Legend at 0x6b19ac8>



In [23]:

```
f_p = interp1d(post_cdf[2], x)

print('for period 2')
print('mean=', f_p(0.5))
print('mean return period=', 1/f_p(0.5))
print('lower 95% confidence interval value=', f_p(0.025))
print('lower 95% confidence interval return period=', 1/f_p(0.025))
print('higher 95% confidence interval value=', f_p(0.975))
print('higher 95% confidence interval return period=', 1/f_p(0.975))
```

```
for period 2
mean= 0.016161580168471204
mean return period= 61.875137800624756
lower 95% confidence interval value= 0.003715286355511129
lower 95% confidence interval return period= 269.15825707933226
higher 95% confidence interval value= 0.038200559034140845
higher 95% confidence interval return period= 26.177627377292403
```

In [24]:

```
f_p = interp1d(post_cdf[3], x)

print('for period 3')
print('mean=', f_p(0.5))
print('mean return period=', 1/f_p(0.5))
print('lower 95% confidence interval value=', f_p(0.025))
print('lower 95% confidence interval return period=', 1/f_p(0.025))
print('higher 95% confidence interval value=', f_p(0.975))
print('higher 95% confidence interval return period=', 1/f_p(0.975))
```

```
for period 3
mean= 0.032048767447938845
mean return period= 31.202448007538372
lower 95% confidence interval value= 0.014712740465401226
lower 95% confidence interval return period= 67.96830286999352
higher 95% confidence interval value= 0.0556213741240648
higher 95% confidence interval return period= 17.97870001862011
```

D.

How much did the mean and 95% confidence interval return period change from the original prior return period to 2019? What does this indicate about the “true” frequency of the 7.2-inch-storm?

For original mean return period= 86.67934513427527

lower 95% confidence interval return period= 708.398274617541

higher 95% confidence interval return period= 30.092112663262675

For posterior

mean return period= 31.202448007538372

lower 95% confidence interval return period= 67.96830286999352

higher 95% confidence interval return period= 17.97870001862011

The result indicate that the ture frequency of the 7.2-inch-storm is much higher than we predicted after we taking the observations into account. It happens more frequently than we expected.

Problem 3

Update on your term project. By now you should have acquired all of the data you need for your term project.

Create 2-3 plots that illustrate your data. These can be time-series plots, histograms, CDFs, whatever is relevant to your data and your problem.

Discuss the quality of your data. Do you need to take into account any erroneous values or uncertain numbers before you start your statistics?

Write down at least two (more are also okay) very specific questions that you will answer with your data. For each question, write down which statistical tools you will use.

Based on what you have so far, do you think that your project scope is about right for a 10-page paper, or do you feel that you need to either simplify things to make it shorter or add complexity to make it longer? If you feel adjustments are needed, how to you plan to address this issue?

At this point, what do you anticipate being the most difficult parts of your term project? Do you have questions you would like help with or advice on? Write them here.

In [25]:

```
df_Q1=pd.read_csv("Lyell_blw_Maclure_timeseries_stage_Q_T_2005_2015.csv", low_memory=False)
```


In [26]:

#Q1_Data missing value check and Data cleaning

```
print (df_Q1.isnull().sum())
```

```

date_time(inGMT_PDTplus7)          0
raw_pressure(cm)                   0
barocorrected_pressure(cm)         0
stage_in_feet(used_in_rating_curve_ft_referenced_to_bolt)  0
estimated_discharge(cms)           913
lower_confidence_discharge_cms_bestestimate  28298
upper_confidence_discharge_cms_bestestimate  28871
water_temperature(deg_C)           0
instrument_ID                       0
offset_cm(cm_to_correct_solinst_to_bolt)  0
offset_ft(ft_to_correct_solinst_to_bolt)  0
dtype: int64

```

Since we are dealing with Temperature and Discharge, thus we remove all rows which contains null discharge value (95% confidence estimate not considered at the moment)

In [27]:

```

df_Q01=df_Q1[df_Q1[' estimated_discharge(cms)'].isna()==False]
df_Q01['date_time(inGMT_PDTplus7)']=pd.to_datetime(df_Q01['date_time(inGMT_PDTplus7)'])

```

E:\tools\py\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

In [28]:

```
plt.figure(figsize=(15, 10))
plt.plot(df_Q01['date_time(inGMT_PDTplus7)'], df_Q01['estimated_discharge(cms)'])
plt.xlabel('Date')
plt.ylabel('Discharge(cms)')
plt.title('Q01 site discharge time series plot')

plt.figure(figsize=(15, 10))
plt.plot(df_Q01['date_time(inGMT_PDTplus7)'], df_Q01['water_temperature(deg_C)'])
plt.xlabel('Date')
plt.ylabel('Water temperature')
plt.title('Q01 site water temperature time series plot')
```

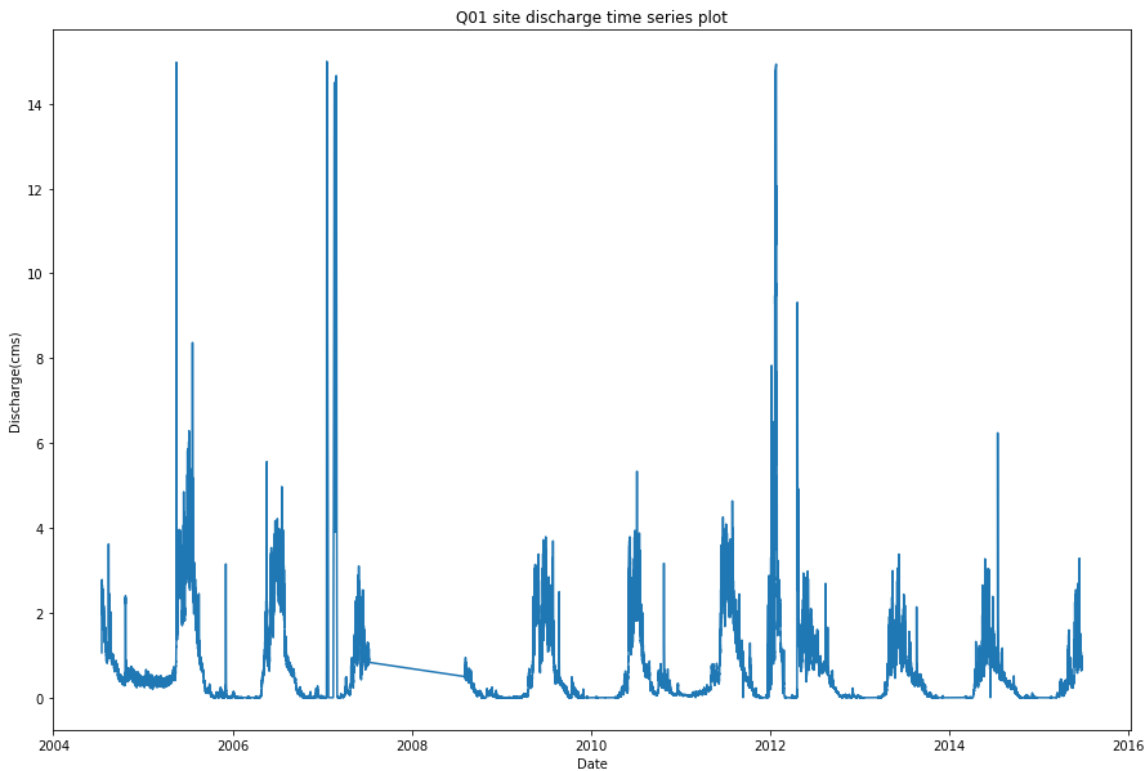
E:\tools\py\lib\site-packages\pandas\plotting_converter.py:129: FutureWarning: Using an implicitly registered datetime converter for a matplotlib plotting method. The converter was registered by pandas on import. Future versions of pandas will require you to explicitly register matplotlib converters.

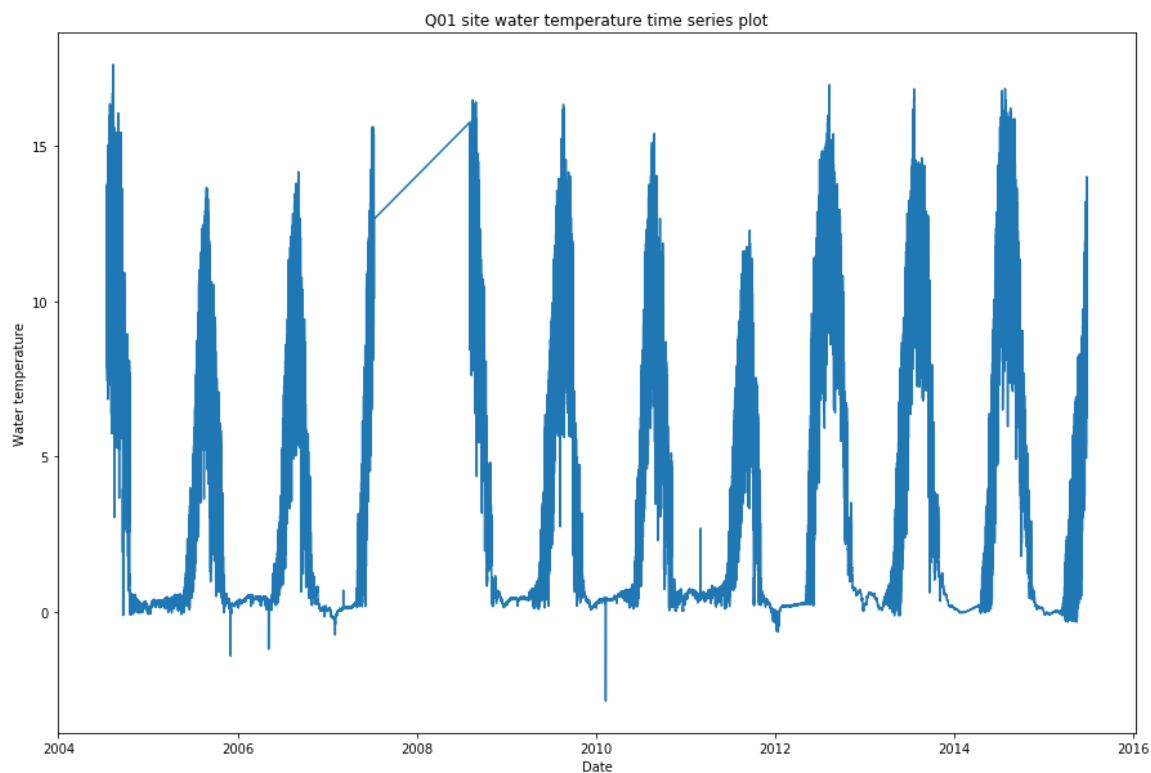
To register the converters:

```
>>> from pandas.plotting import register_matplotlib_converters
>>> register_matplotlib_converters()
warnings.warn(msg, FutureWarning)
```

Out[28]:

Text(0.5, 1.0, 'Q01 site water temperature time series plot')





In [29]:

```
df_Q4=pd.read_csv('Tuolumnel20_timeseries_stage_Q_T_2002_2015.csv', low_memory=False)
```

In [30]:

```
#Q4_Data missing value check and Data cleaning
```

```
print (df_Q4.isnull().sum())
```

```
date_time(UTC:LDT+7)          0
raw_pressure(cm)              0
barocorrected_pressure(cm)    0
offset(cm)                   0
stage(cm)                    0
estimated_discharge(cms)      0
lower_confidence_discharge_cms_bestestimate  370
upper_confidence_discharge_cms_bestestimate  370
instrument_ID                 0
water_temperature(deg_C)      0
dtype: int64
```

In [31]:

```
df_Q4['date_time(UTC:LDT+7)']=pd.to_datetime(df_Q4['date_time(UTC:LDT+7)'])
```

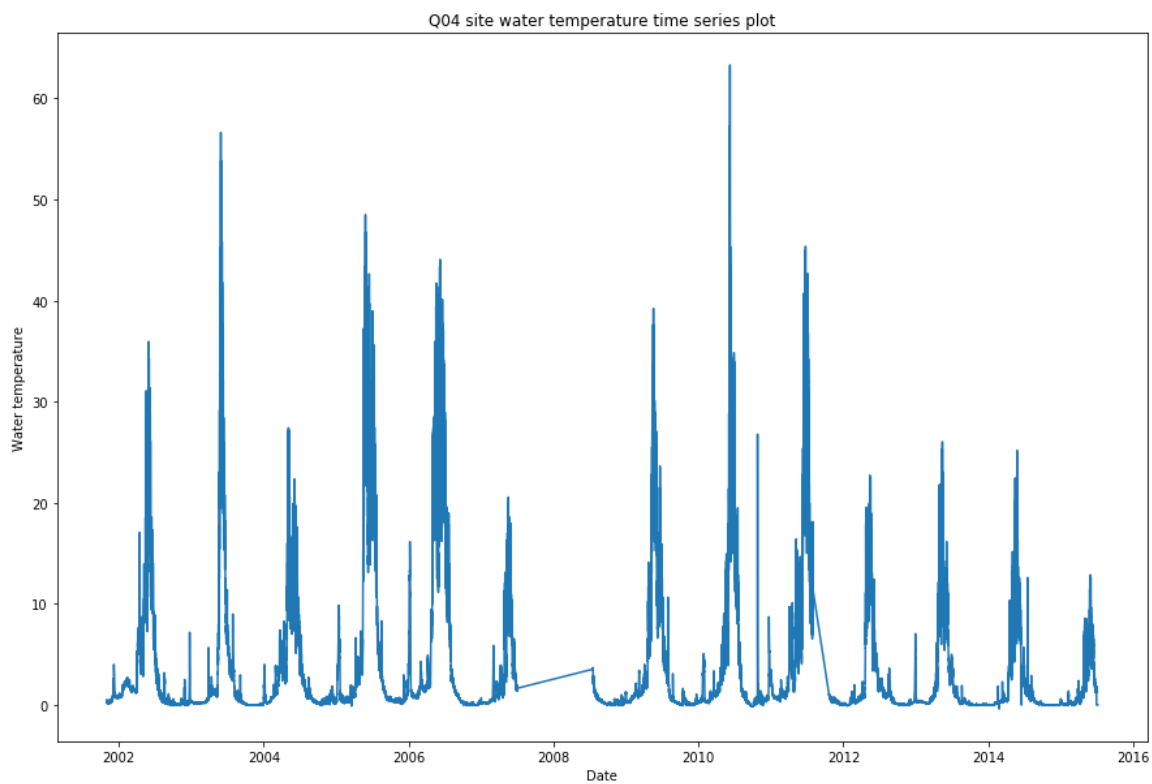
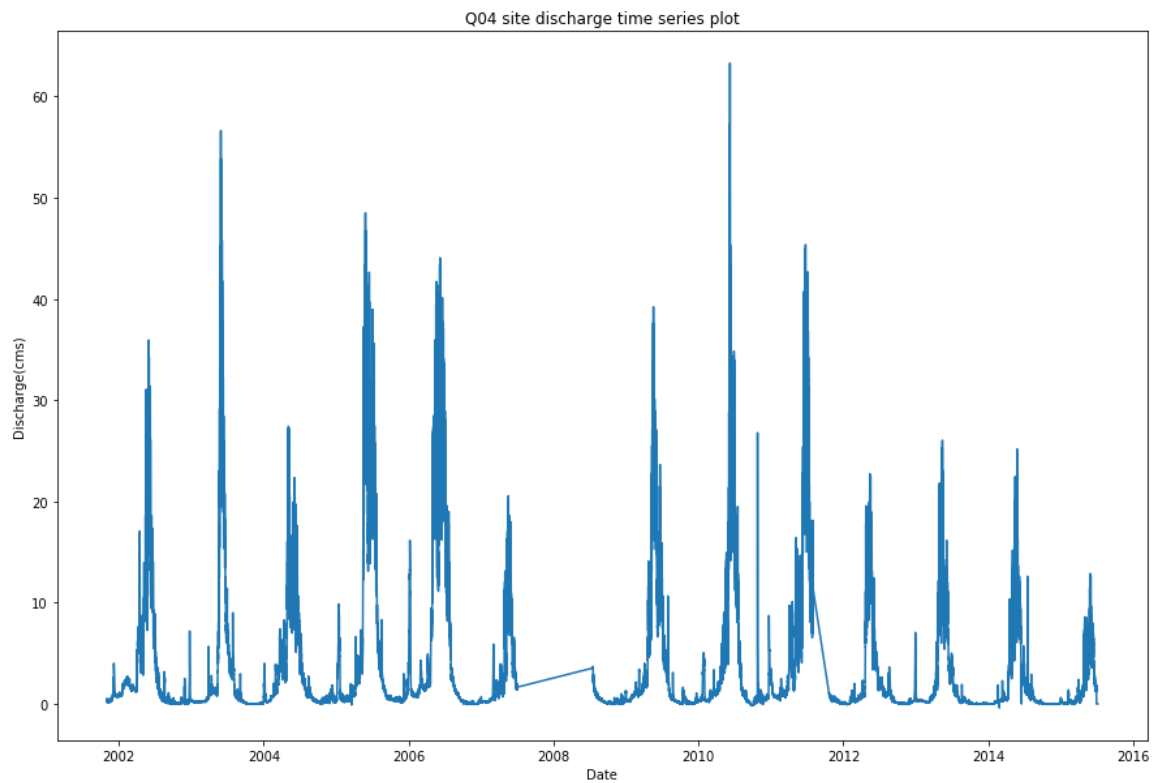
In [32]:

```
plt.figure(figsize=(15, 10))
plt.plot(df_Q4['date_time(UTC:LDT+7)'], df_Q4['estimated_discharge(cms)'])
plt.xlabel('Date')
plt.ylabel('Discharge(cms)')
plt.title('Q04 site discharge time series plot')

plt.figure(figsize=(15, 10))
plt.plot(df_Q4['date_time(UTC:LDT+7)'], df_Q4['estimated_discharge(cms)'])
plt.xlabel('Date')
plt.ylabel('Water temperature')
plt.title('Q04 site water temperature time series plot')
```

Out[32]:

Text(0.5, 1.0, 'Q04 site water temperature time series plot')



Q:

Discuss the quality of your data. Do you need to take into account any erroneous values or uncertain numbers before you start your statistics?

A:

According to the data plotted above, we can tell that there are part of the time series data missed during the whole process, but the rest of them have a rather good quality. Before further data analysis, we also need to clean the dataset, remove some abnormal points or outliers in order to reduce the effects of these noises.

Q:

Write down at least two (more are also okay) very specific questions that you will answer with your data. For each question, write down which statistical tools you will use.

A:

Q1: When is the onset date of snowmelt runoff?

For question1, we will use ANOVA test to determind when the streams start rising.

Q2: Whether stream temperature is significantly changed by streamflow timing?

For question2, we will use several hypothesis tests like Monte Cario Test and fit the data into regression models to determine whether stream temperature is significantly changed by streamflow timing.

Q:

Based on what you have so far, do you think that your project scope is about right for a 10-page paper, or do you feel that you need to either simplify things to make it shorter or add complexity to make it longer? If you feel adjustments are needed, how to you plan to address this issue?

A:

From the website we can pull down 6 rivers stream and temperature data for the Tuolumne River Watershed and surroundings, which is far enough for our analysis. As the data is a bit of too volumy we decided to start with Q01 and Q04 data to narrow down the range for the 10-page paper. After few try out on the datasets we can do additional adjustment but so far it looks fine.

Q:

At this point, what do you anticipate being the most difficult parts of your term project? Do you have questions you would like help with or advice on? Write them here.

A:

As at the beginning point, we think the main challenge now is to adapt to this large dataset for the analysis, the stream and temperature data is recorded by every half hour for more than 5 years thus we need to figure out how to either simplify the data or pick up some specific subsets for further analysis.

Also we bit of confused about our 'ultimate' goal of the project, after we get to the conclusion of the relationship between temperature and runoff, should we continue exploring why stream temperatures being unhealthy for fish if snow melts earlier? Or it is OK for us to just get this relationship and end up there.