Laboratory 13 Probability Modeling

Full name: Mariah Herrera

R#: 11723240

HEX: 0xb2e1e8

Lab 13

Date: 10/08/2020

In [1]:
```
RN= 11723240
HRN= hex(RN)
print(HRN)
```

```
0xb2e1e8
```

Important Terminology:

Population: In statistics, a population is the entire pool from which a statistical sample is drawn. A population may refer to an entire group of people, objects, events, hospital visits, or measurements.
Sample: In statistics and quantitative research methodology, a sample is a set of individuals or objects collected or selected from a statistical population by a defined procedure. The elements of a sample are known as sample points, sampling units or observations.
Distribution (Data Model): A data distribution is a function or a listing which shows all the possible values (or intervals) of the data. It also (and this is important) tells you how often each value occurs.

From https://www.investopedia.com/terms
https://www.statisticshowto.com/data-distribution/

In [0]:
```
### Important Steps:
1. __Get descriptive statistics- mean, variance, std. dev.__
2. __Use plotting position formulas (e.g., weibull, gringorten, cunnane) and plot the S
3. __Use different data models (e.g., normal, log-normal, Gumbell) and find the one tha
4. __Use the data model that provides the best fit to infer about the POPULATION__
```

Estimate the magnitude of the annual peak flow at Spring Ck near Spring, TX.

The file `08068500.pkf` is an actual WATSTORE formatted file for a USGS gage at Spring Creek,

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Z08068500                       USGS
H08068500       30063709526100048483395W12040102409    409     72.6
N08068500       Spring Ck nr Spring, TX
Y08068500
308068500       19290530  483007              34.30        1879
308068500       19390603    838               13.75
308068500       19400612   3420               21.42
308068500       19401125  42700               33.60
308068500       19420409  14200               27.78
308068500       19430730   8000               25.09
308068500       19440319   5260               23.15
308068500       19450830  31100               32.79
308068500       19460521  12200               27.97
```

The first column are some agency codes that identify the station , the second column after the fourth row is a date in YYYYMMDD format, the third column is a discharge in CFS, the fourth and fifth column are not relevant for this laboratory exercise. The file was downloadef from

https://nwis.waterdata.usgs.gov/tx/nwis/peak?site_no=08068500&agency_cd=USGS&format=hn2

In the original file there are a couple of codes that are manually removed:

- 19290530 483007; the trailing 7 is a code identifying a break in the series (non-sequential)
- 20170828 784009; the trailing 9 identifies the historical peak

The laboratory task is to fit the data models to this data, decide the best model from visual perspective, and report from that data model the magnitudes of peak flow associated with the probebilitiess below (i.e. populate the table)

Exceedence Probability	Flow Value	Remarks
25%	????	75% chance of greater value
50%	????	50% chance of greater value
75%	????	25% chance of greater value
90%	????	10% chance of greater value
99%	????	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	????	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	????	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

The first step is to read the file, skipping the first part, then build a dataframe:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# Read the data file
```

```
amatrix = [] # null list to store matrix reads
rowNumA = 0
matrix1=[]
col0=[]
col1=[]
col2=[]
with open('08068500.pkf','r') as afile:
    lines_after_4 = afile.readlines()[4:]
afile.close() # Disconnect the file
howmanyrows = len(lines_after_4)
for i in range(howmanyrows):
    matrix1.append(lines_after_4[i].strip().split())
for i in range(howmanyrows):
    col0.append(matrix1[i][0])
    col1.append(matrix1[i][1])
    col2.append(int(matrix1[i][2]))
# col2 is date, col3 is peak flow
#now build a datafranem
```

In [15]:

```
import pandas
df = pandas.DataFrame(col0)
df['date']= col1
df['flow']= col2
```

In [16]:

```
df.head()
```

Out[16]:

	0	date	flow
0	308068500	19290530	48300
1	308068500	19390603	838
2	308068500	19400612	3420
3	308068500	19401125	42700
4	308068500	19420409	14200

Now explore if you can plot the dataframe as a plot of peaks versus date.

In [33]:

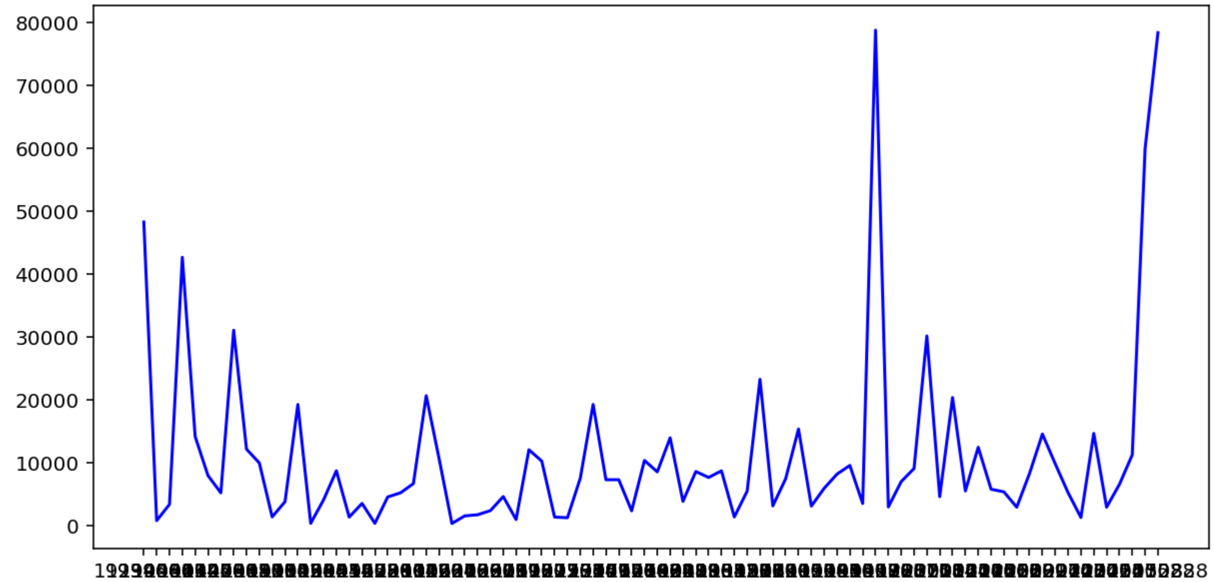
```
# Plot here
import matplotlib.pyplot

date = df['date']
flow = df['flow']

myfigure = matplotlib.pyplot.figure(figsize = (10,5))

matplotlib.pyplot.plot(date,flow, color= 'blue')
```

Out[33]:

```
[<matplotlib.lines.Line2D at 0x7f2688163160>]
```

Out[33]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



From here on you can proceede using the lecture notebook as a go-by, although you should use functions as much as practical to keep your work concise

In [34]:
```
# Descriptive Statistics
import pandas as pd
flow.describe()
```

Out[34]:
```
count       80.000000
mean     11197.800000
std      15022.831582
min        381.000000
25%       3360.000000
50%       7190.000000
75%      11500.000000
max      78800.000000
Name: flow, dtype: float64
```

In [41]:
```
# Weibull Plotting Position Function
import numpy
flow1 = df['flow'].tolist()
flow1_mean = numpy.array(flow1).mean()
flow1_variance = numpy.array(flow1).std()**2
flow1.sort()
weibull_pp = []
for i in range(0,len(flow1),1):
    weibull_pp.append((i+1)/(len(flow1)+1))
```

In [52]:
```
# Normal Quantile Function
import math
def normdist(x,mu,sigma):
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist

mu = flow1_mean
```

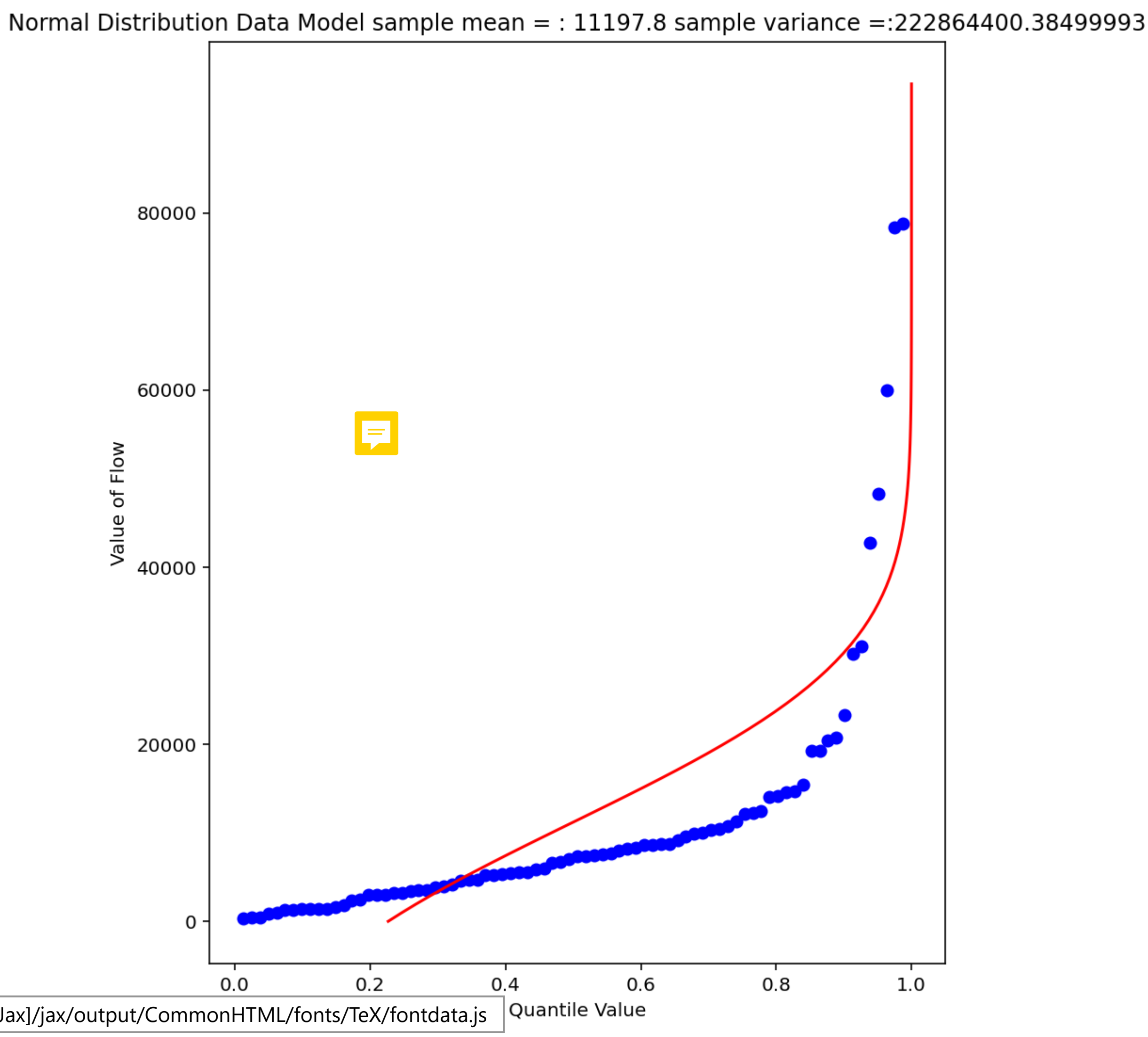
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
x = []
ycdf = []

xlow = 0
xhigh = 1.2*max(flow1)
howMany = 100
xstep = (xhigh - xlow)/howMany

for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)
```

In [53]:

```
# Fitting Data to Normal Data Model
myfigure = matplotlib.pyplot.figure(figsize = (7,9))
matplotlib.pyplot.scatter(weibull_pp, flow1 ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of Flow")
mytitle = "Normal Distribution Data Model sample mean = : " + str(mu)+ " sample varianc
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

Out[53]:



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Normal Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	1128.5828928044155	75% chance of greater value
50%	11197.800000000001	50% chance of greater value
75%	21267.017107195585	25% chance of greater value
90%	30329.62660490181	10% chance of greater value
99%	45927.018351754574	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	54164.85088869193	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	57330.776807175745	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

In [66]:

```
# Log-Normal Quantile Function
from scipy.optimize import newton

myguess = 2000

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.25
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.50
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.75
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.9
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.99
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.99
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.998
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

def f(x):
    mu = flow1_mean
    sigma = math.sqrt(flow1_variance)
    quantile = 0.999
    argument = (x - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))
```

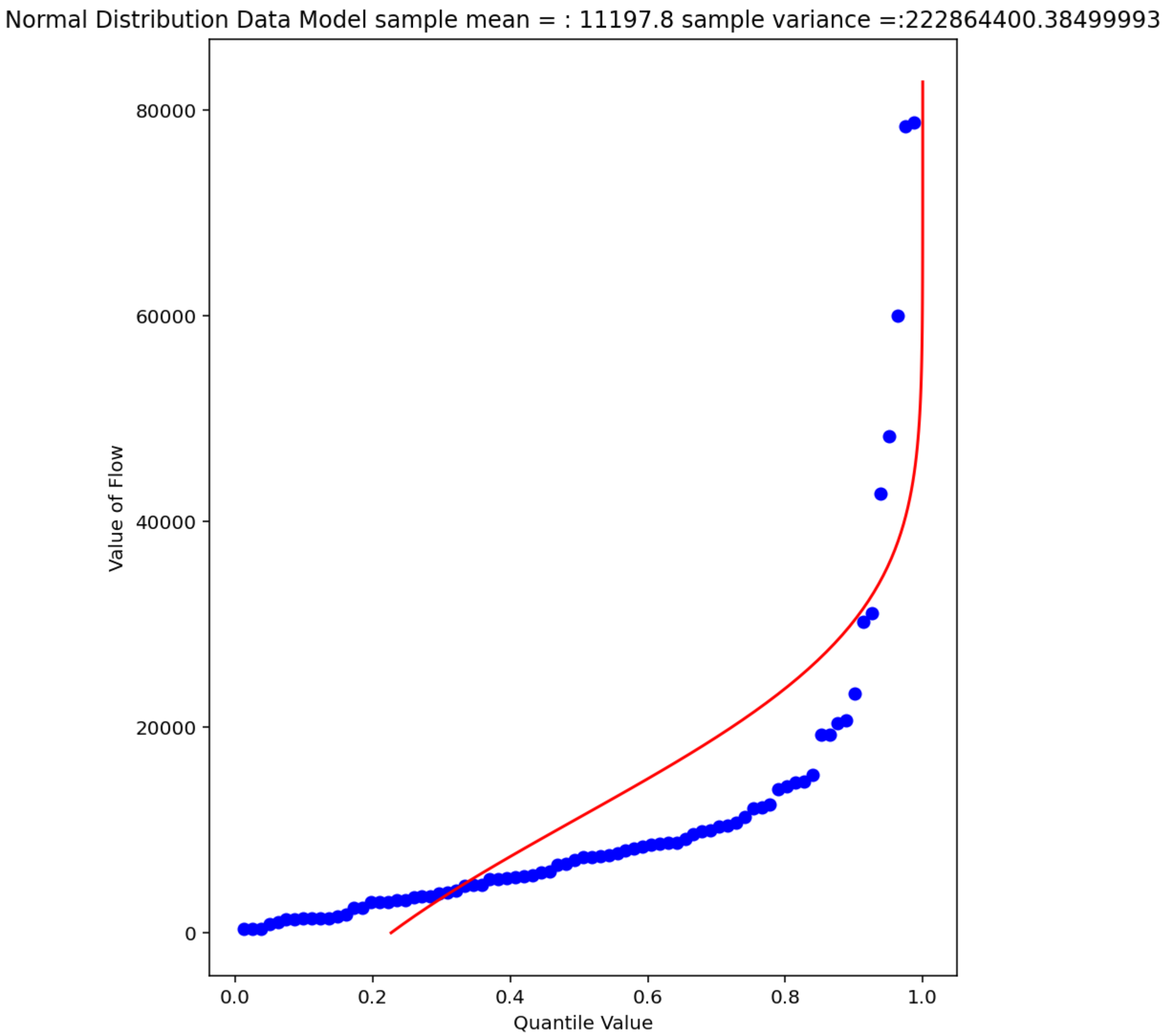
```
1128.5828928044155
0.25
11197.800000000001
0.5
21267.017107195585
0.75
30329.62660490181
0.8999999999999999
45927.018351754574
0.99
45927.018351754574
0.99
54164.85088869193
0.998
57330.776807175745
0.9990000000000001
```

In [71]:

```python
# Fitting Data to Normal Data Model
import math
mu = flow1_mean
sigma = math.sqrt(flow1_variance)

x = []
ycdf = []

xlow = 1
xhigh = 1.05*max(flow1)
howMany = 100
xstep = (xhigh - xlow)/howMany

for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)

myfigure = matplotlib.pyplot.figure(figsize = (7,9))
matplotlib.pyplot.scatter(weibull_pp, flow1 ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of Flow")
mytitle = "Normal Distribution Data Model sample mean = : " + str(mu)+ " sample varianc
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

Out[71]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [0]:

Log-Normal Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	1101.4389442765275	75% chance of greater value
50%	1410.1975824983385	50% chance of greater value
75%	1801.9154685618557	25% chance of greater value
90%	2249.3502726390875	10% chance of greater value
99%	3296.047827999136	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	4014.753927367107	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	4323.735566660218	0.001% chance of greater value (in flood statistics, this is the 1 in [illegible]0-yr chance event)

In [86]:

```
# Gumbell EV1 Quantile Function
from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.25
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.5
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.75
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.90
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(normdist(newton(f,myguess),mu,sigma))
```

```
from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.99
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.998
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))

from scipy.optimize import newton

def f(x):
    alpha = 1246.9363972503857
    beta = 445.4445561942363
    quantile = 0.999
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist - quantile

print(newton(f, myguess))
print(normdist(newton(f,myguess),mu,sigma))
```

```
1101.4389442765275
1.0
1410.1975824983385
1.0
1801.9154685618557
1.0
2249.3502726390875
1.0
3296.047827999136
1.0
4014.753927367107
1.0
4323.735566660218
1.0
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

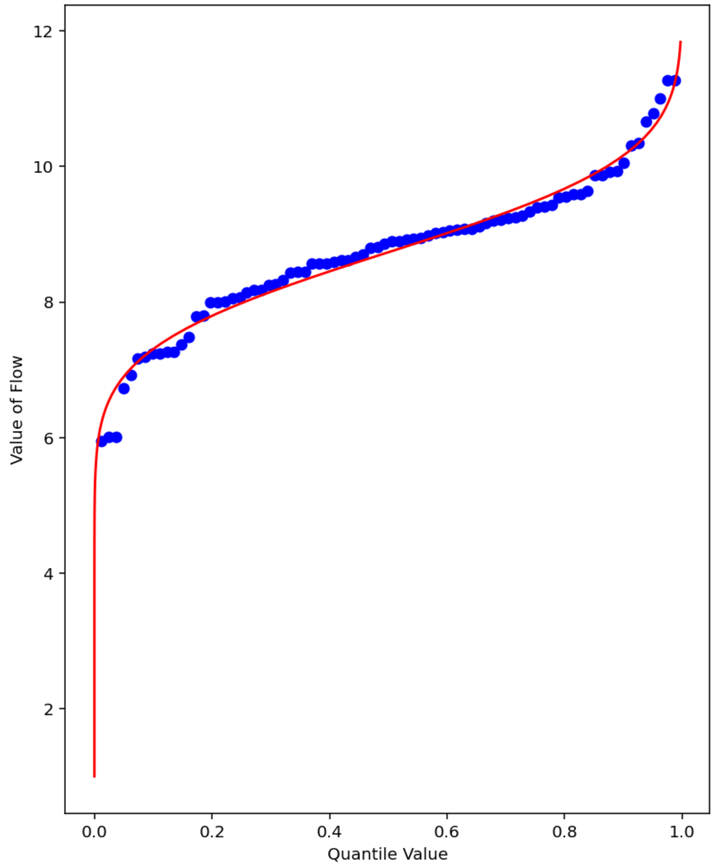
In [79]:

```
# Fitting Data to Gumbell EV1 Data Model
def loggit(x):
    return(math.log(x))

flow2 = df['flow'].apply(loggit).tolist()
flow2_mean = numpy.array(flow2).mean()
flow2_variance = numpy.array(flow2).std()**2
flow2.sort()
weibull_pp = []
for i in range(0,len(flow2),1):
    weibull_pp.append((i+1)/(len(flow2)+1))

mu = flow2_mean # Fitted Model in Log Space
sigma = math.sqrt(flow2_variance)
x = []
ycdf = []
xlow = 1
xhigh = 1.05*max(flow2)
howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)

myfigure = matplotlib.pyplot.figure(figsize = (7,9))
matplotlib.pyplot.scatter(weibull_pp, flow2 ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of Flow")
mytitle = "Log Normal Distribution Data Model log sample mean = : " + str(flow2_mean)+
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

Out[79]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Log Normal Distribution Data Model log sample mean = : 8.734714243614741 log sample variance =:1.2418760475510937



Gumbell Double Exponential (EV1) Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	968.1512046699919	75% chance of greater value
50%	1302.814639184079	50% chance of greater value
75%	1860.5263696955876	25% chance of greater value
90%	2706.313223303496	10% chance of greater value
99%	5856.109131583644	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	9420.653537187907	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	11455.308202234171	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

In [120…

```
# Gamma (Pearson Type III) Quantile Function

from scipy.optimize import newton

    flow3_tau = 5.976005311346212
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.25
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.50
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.75
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.90
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.99
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))
```

```
def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.998
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.999
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))
```

```
968.1512046699919
1302.814639184079
1860.5263696955876
2706.313223303496
5856.109131583644
9420.653537187907
11455.308202234171
```

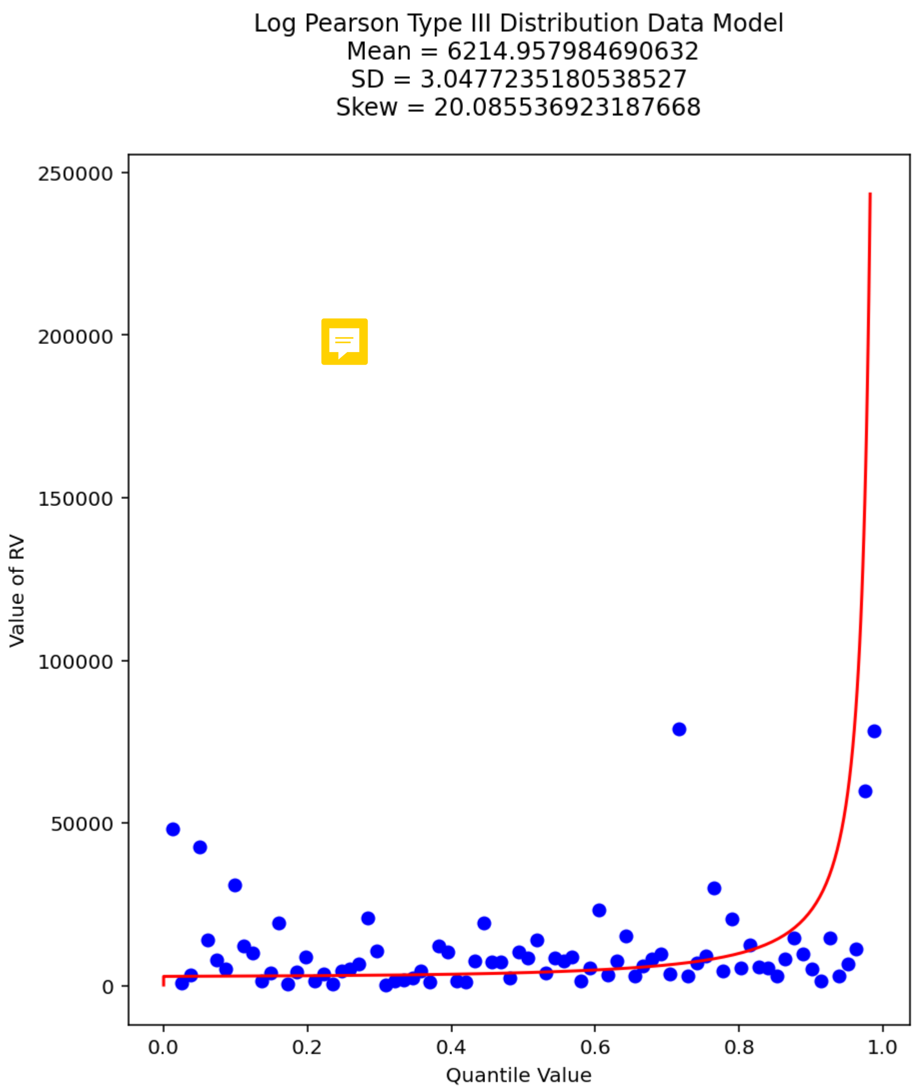
In [98]:

```
# Fitting Data to Pearson (Gamma) III Data Model
# This is new, in lecture the fit was to Log-Pearson, same procedure, but not log trans
def gammacdf(x,tau,alpha,beta):
    xhat = x-tau
    lamda = 1.0/beta
    gammacdf = scipy.stats.gamma.cdf(lamda*xhat, alpha)
    return gammacdf

flow3 = df['flow'].apply(loggit).tolist()
flow3_mean  = numpy.array(flow3).mean()
flow3_stdev = numpy.array(flow3).std()
flow3_skew  = 3.0
flow3_alpha = 4.0/(flow3_skew**2)
flow3_beta  = numpy.sign(flow3_skew)*math.sqrt(flow3_stdev**2/flow3_alpha)
flow3_tau   = flow3_mean - flow3_alpha*flow3_beta

x = []
ycdf = []
xlow = (0.9*min(flow3))
xhigh = (1.1*max(flow3))
howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    yvalue = gammacdf(xlow + i*xstep,flow3_tau,flow3_alpha,flow3_beta)
    ycdf.append(yvalue)

for i in range(len(flow3)):
    flow3[i] = antiloggit(flow3[i])
for i in range(len(x)):
    x[i] = antiloggit(x[i])
myfigure = matplotlib.pyplot.figure(figsize = (7,8))
matplotlib.pyplot.scatter(weibull_pp, flow3 ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of RV")
mytitle = "Log Pearson Type III Distribution Data Model\n "
mytitle += "Mean = " + str(antiloggit(flow3_mean)) + "\n"
mytitle += "SD = " + str(antiloggit(flow3_stdev)) + "\n"
mytitle += "Skew = " + str(antiloggit(flow3_skew)) + "\n"
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

Out[98]:



Pearson III Distribution Data Model

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Exceedence Probability	Flow Value	Remarks
25%	????	75% chance of greater value
50%	????	50% chance of greater value
75%	????	25% chance of greater value
90%	????	10% chance of greater value
99%	????	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	????	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	????	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

In [129…

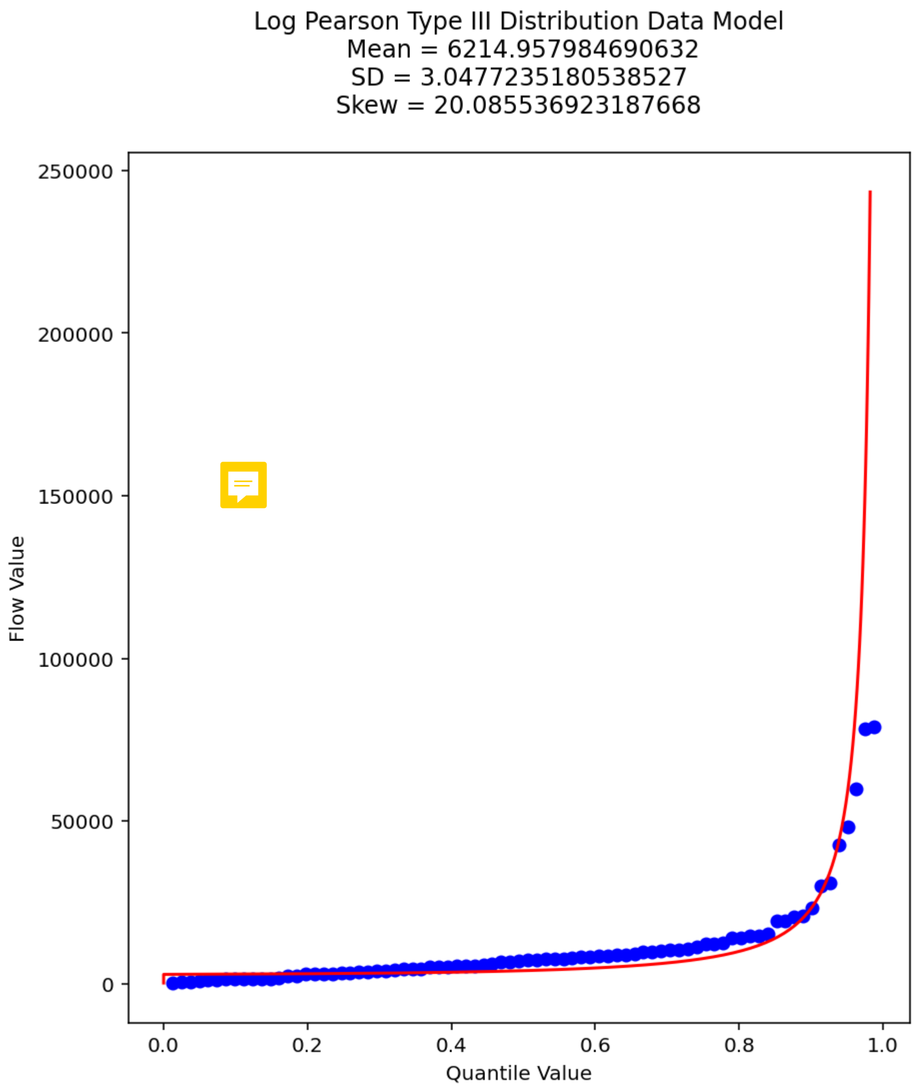
```
# Fitting Data to Log-Pearson (Log-Gamma) III Data Model
import scipy.stats
import math
import numpy
flow4 = df['flow'].apply(loggit).tolist()
flow4_mean  = numpy.array(flow4).mean()
flow4_stdev = numpy.array(flow4).std()
flow4_skew  = 3.0
flow4_alpha = 4.0/(flow4_skew**2)
flow4_beta  = numpy.sign(flow4_skew)*math.sqrt(flow4_stdev**2/flow4_alpha)
flow4_tau   = flow4_mean - flow4_alpha*flow4_beta
def loggit(x):
    return(math.log(x))
def antiloggit(x):
    return(math.exp(x))
def weibull_pp(flow4):
    weibull_pp = []
    flow4.sort()
    for i in range(0,len(flow4),1):
        weibull_pp.append((i+1)/(len(flow4)+1))
    return weibull_pp

def gammacdf(x,tau,alpha,beta):
    xhat = x-tau
    lamda = 1.0/beta
    gammacdf = scipy.stats.gamma.cdf(lamda*xhat, alpha)
    return gammacdf
plotting = weibull_pp(flow4)

x = []; ycdf = []
xlow = (0.9*min(flow4)); xhigh = (1.1*max(flow4)) ; howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = gammacdf(xlow + i*xstep,flow4_tau,flow4_alpha,flow4_beta)
    ycdf.append(yvalue)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
        flow4[i] = antiloggit(flow4[i])
```

```
for i in range(len(x)):
    x[i] = antiloggit(x[i])
myfigure = matplotlib.pyplot.figure(figsize = (7,8))
matplotlib.pyplot.scatter(plotting, flow4 ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Flow Value")
mytitle = "Log Pearson Type III Distribution Data Model\n "
mytitle += "Mean = " + str(antiloggit(flow4_mean)) + "\n"
mytitle += "SD = " + str(antiloggit(flow4_stdev)) + "\n"
mytitle += "Skew = " + str(antiloggit(flow4_skew)) + "\n"
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

Out[129…



Log-Pearson III Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	968.1512046699919	75% chance of greater value
50%	[illegible]	50% chance of greater value

Exceedence Probability	Flow Value	Remarks
75%	1860.5263696955876	25% chance of greater value
90%	2706.313223303496	10% chance of greater value
99%	5856.109131583644	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	9420.653537187907	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	11455.308202234171	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

In [134…

```
from scipy.optimize import newton

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.25
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.50
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.75
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
```

```
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.90
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.99
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.998
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))

def f(x):
    flow3_tau = 5.976005311346212
    flow3_alpha = 6.402272915026134
    flow3_beta = 0.1970087438569494
    quantile = 0.999
    argument = loggit(x)
    gammavalue = gammacdf(argument,flow3_tau,flow3_alpha,flow3_beta)
    return gammavalue - quantile

myguess =  1000
print(newton(f, myguess))
```

```
968.1512046699919
1302.814639184079
1860.5263696955876
2706.313223303496
5856.109131583644
9420.653537187907
11455.308202234171
```

Summary of "Best" Data Model based on Graphical Fit

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

The best fit data model was Log-Normal distribution model. It hit the most points and followed the slope and curve of the scatter plot points the best.

In [0]:

```
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js