Lab13

October 15, 2020

1 Laboratory 13 Probability Modeling

1.1 Full name: Maximus Rodriguez

1.2 R#: 11712928

1.3 HEX: 0xb2b9a0

1.4 Title of the notebook: Laboratory 13 Worksheet

1.5 Date: 10/8/2020

1.5.1 Important Steps:

1. **Get descriptive statistics- mean, variance, std. dev.**
2. **Use plotting position formulas (e.g., weibull, gringorten, cunnane) and plot the SAMPLES (data you already have)**
3. **Use different data models (e.g., normal, log-normal, Gumbell) and find the one that better FITs your samples- Visual or Numerical**
4. **Use the data model that provides the best fit to infer about the POPULATION**

2 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, Texas. The first few lines of the file look like:

```
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:

[214]:
```
# 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 dataframem
```

[215]:
```
import pandas
df = pandas.DataFrame(col0)
df['date']= col1
df['flow']= col2
```

[216]:
```
df.head()
```

[216]:

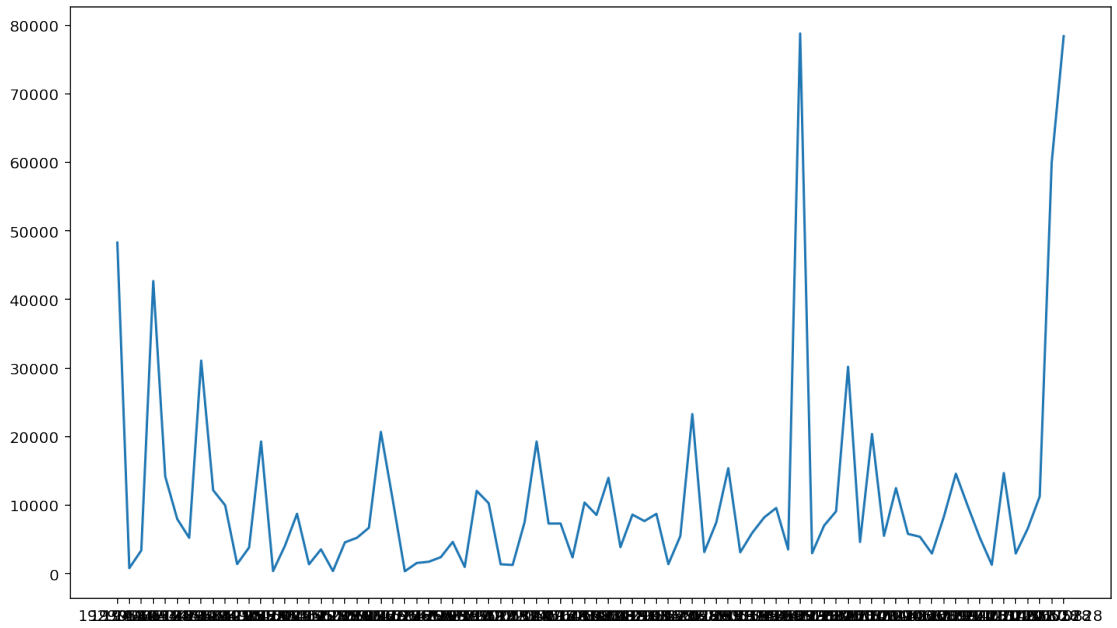
	0	date	flow
0	308068500	19290530	48300
1	308068500	19390603	838
2	308068500	19400612	3420
3	308068500	19401125	42700
4	308068500	19420409	14200

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

[217]:
```
# Plot here
matplotlib.pyplot.plot(df["date"],df["flow"])
```

[217]:
```
[<matplotlib.lines.Line2D at 0x7f86924327f0>]
```

[217]:



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

[218]:
```
# Descriptive Statistics
df["flow"].describe()
```

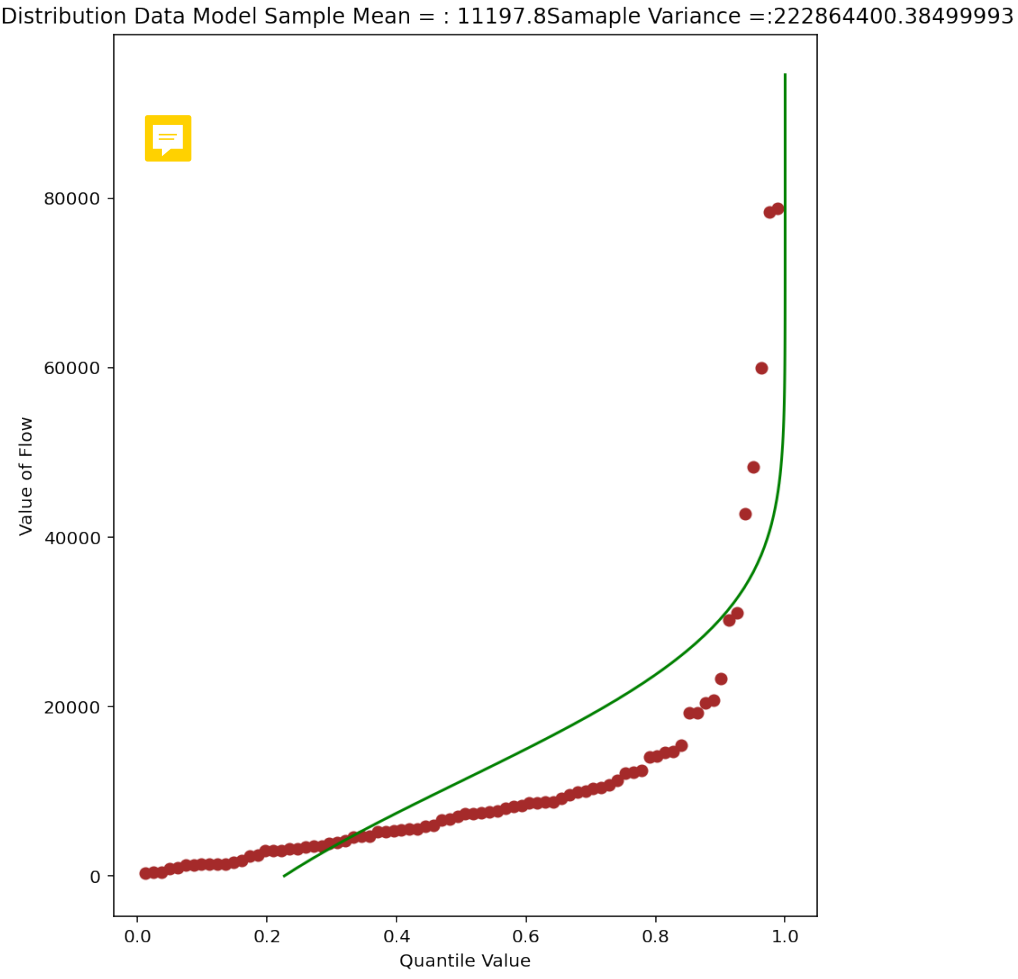
[218]:
```
count       80.000000
mean     11197.800000
std      15022.831582
min        381.000000
25%       3360.000000
50%       7190.000000
75%      11500.000000
max      78800.000000
Name: flow, dtype: float64
```

[219]:
```
import numpy
import math
#Weibull Plotting Position Function
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))
```

[220]:
```
# 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
sigma = math.sqrt(flow1_variance)

x = []; ypdf = []; 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)
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)
```

[221]:
```
# Fitting Data to Normal Data Model
myfigure = matplotlib.pyplot.figure(figsize = (7,9))
matplotlib.pyplot.scatter(weibull_pp, flow1, color = 'brown')
matplotlib.pyplot.plot(ycdf, x, color = 'green')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of Flow")
mytitle = "Normal Distribution Data Model Sample Mean = : " + str(mu)+ "Samaple␣
 ↪Variance =:" + str(sigma**2)
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[221]:



[222]:
```
# Log-Normal Quantile Function
from scipy.optimize import newton

myguess = 2000
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))
```

```
11197.800000000001
```

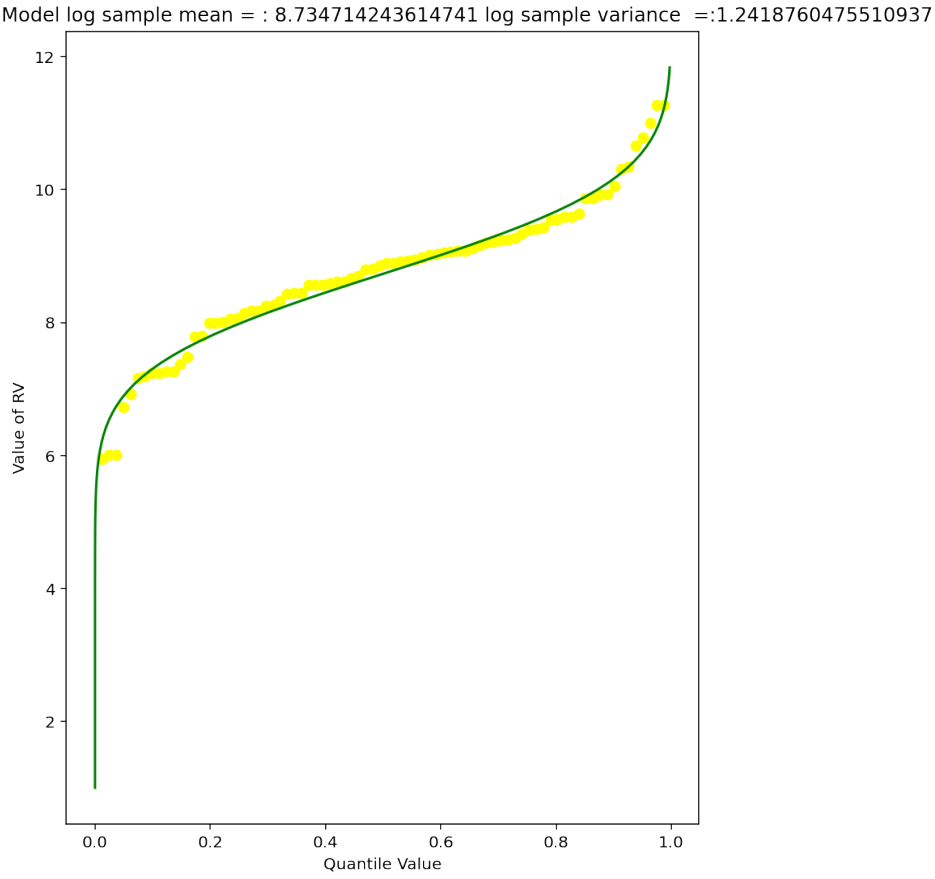
2.1 Normal Distribution Data Model

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)

[223]:
```
sample = df['flow'].tolist()

def loggit(x):  # A prototype function to log transform x
    return(math.log(x))

logsample = df['flow'].apply(loggit).tolist() # put the peaks into a list
sample_mean = numpy.array(logsample).mean()
sample_variance = numpy.array(logsample).std()**2
logsample.sort() # sort the sample in place!
weibull_pp = [] # built a relative frequency approximation to probability,␣
 ↪assume each pick is equally likely
for i in range(0,len(sample),1):
    weibull_pp.append((i+1)/(len(sample)+1))
#################
mu = sample_mean # Fitted Model in Log Space
sigma = math.sqrt(sample_variance)
```

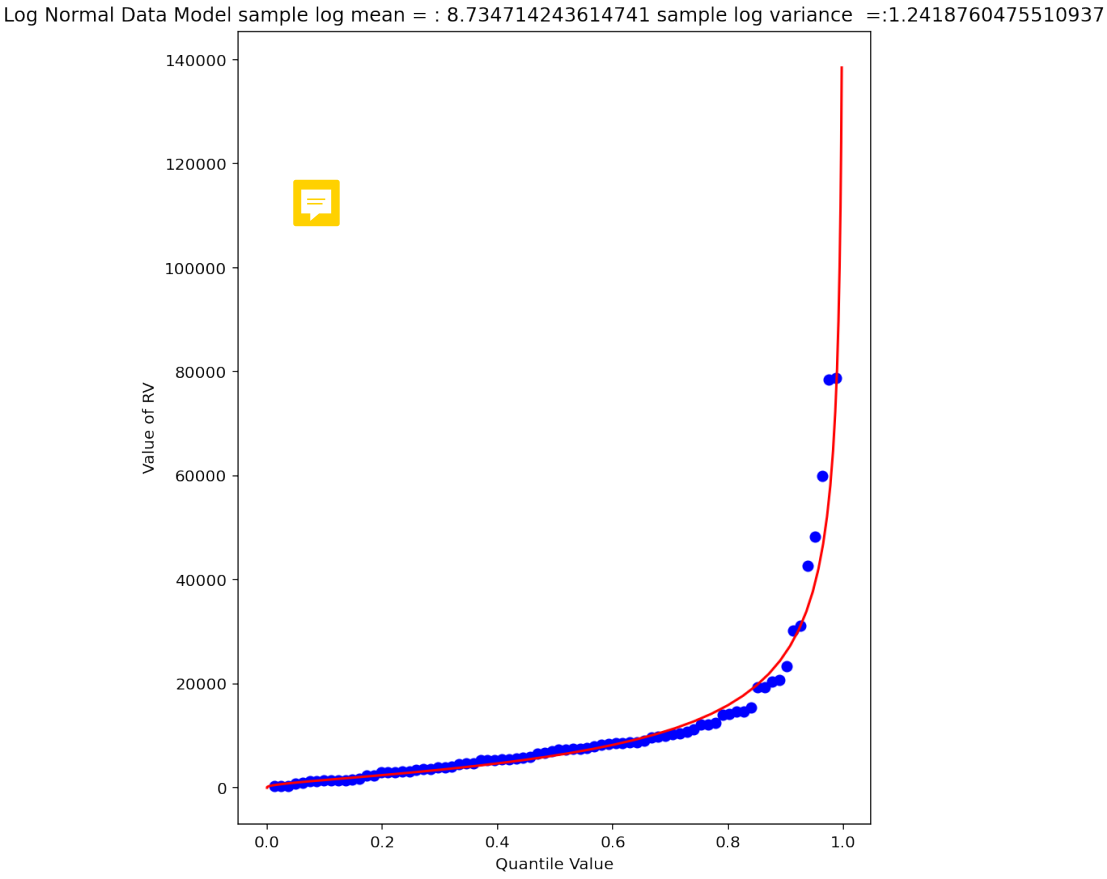
```
x = []; ycdf = []
xlow = 1; xhigh = 1.05*max(logsample) ; howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)
# Now plot the sample values and plotting position
myfigure = matplotlib.pyplot.figure(figsize = (7,9)) # generate a object from␣
 ↪the figure class, set aspect ratio
matplotlib.pyplot.scatter(weibull_pp, logsample ,color ='yellow')
matplotlib.pyplot.plot(ycdf, x, color ='green')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of RV")
mytitle = "Log Normal Data Model log sample mean = : " + str(sample_mean)+ "␣
 ↪log sample variance  =:" + str(sample_variance)
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[223]:



[224]:
```
def antiloggit(x):  # A prototype function to log transform x
    return(math.exp(x))

sample = df['flow'].tolist() # pull original list
sample.sort() # sort in place
################
mu = sample_mean # Fitted Model in Log Space
sigma = math.sqrt(sample_variance)
x = []; ycdf = []
xlow = 1; xhigh = 1.05*max(logsample) ; howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(antiloggit(xlow + i*xstep))
    yvalue = normdist(xlow + i*xstep,mu,sigma)
    ycdf.append(yvalue)
# Now plot the sample values and plotting position
myfigure = matplotlib.pyplot.figure(figsize = (7,9)) # generate a object from␣
 ↪the figure class, set aspect ratio
matplotlib.pyplot.scatter(weibull_pp, sample ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of RV")
mytitle = "Log Normal Data Model sample log mean = : " + str((sample_mean))+ "␣
 ↪sample log variance  =:" + str((sample_variance))
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[224]:



[225]:
```
myguess = 90000
print(mu,sigma)
print(normdist(loggit(myguess),mu,sigma)) # mu, sigma already in log space -␣
↪convert myguess
```

```
8.734714243614741 1.1143949244101454
0.9917682970396343
```

[226]:
```
from scipy.optimize import newton

def f(x):
    mu = 8.734714243614741
    sigma = 1.1143949244101454
    quantile = 0.999
    argument = (loggit(x) - mu)/(math.sqrt(2.0)*sigma)
    normdist = (1.0 + math.erf(argument))/2.0
    return normdist - quantile

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

194551.74365123696

2.2 Log-Normal Distribution Data Model

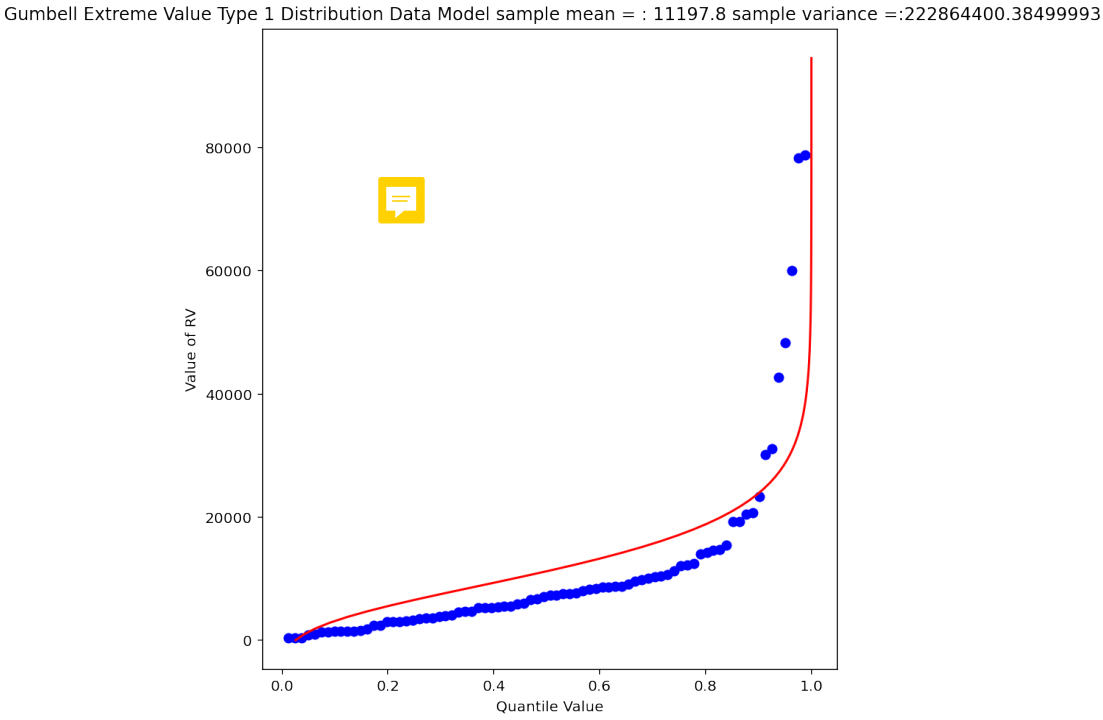
Exceedence Probability	Flow Value	Remarks
25%	2930.9043006567777	75% chance of greater value
50%	6214.957984690637	50% chance of greater value
75%	13178.766274563905	25% chance of greater value
90%	25922.557216077774	10% chance of greater value
99%	83048.86984807048	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	153602.45417873585	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	194551.74365123696	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

[227]:
```
# Gumbell EV1 Quantile Function
def ev1dist(x,alpha,beta):
    argument = (x - alpha)/beta
    constant = 1.0/beta
    ev1dist = math.exp(-1.0*math.exp(-1.0*argument))
    return ev1dist
```

[228]:
```
# Fitting Data to Gumbell EV1 Data Model
sample =df['flow'].tolist() # put the peaks into a list
sample_mean = numpy.array(sample).mean()
sample_variance = numpy.array(sample).std()**2
alpha_mom = sample_mean*math.sqrt(6)/math.pi
beta_mom = math.sqrt(sample_variance)*0.45

sample = df['flow'].tolist() # put the peaks into a list
sample_mean = numpy.array(sample).mean()
sample_variance = numpy.array(sample).std()**2
alpha_mom = sample_mean*math.sqrt(6)/math.pi
beta_mom = math.sqrt(sample_variance)*0.45
sample.sort() # sort the sample in place!
weibull_pp = [] # built a relative frequency approximation to probability,␣
 ↪assume each pick is equally likely
for i in range(0,len(sample),1):
    weibull_pp.append((i+1)/(len(sample)+1))
################
```

```
mu = sample_mean # Fitted Model
sigma = math.sqrt(sample_variance)
x = []; ycdf = []
xlow = 0; xhigh = 1.2*max(sample) ; howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = ev1dist(xlow + i*xstep,alpha_mom,beta_mom)
    ycdf.append(yvalue)
# Now plot the sample values and plotting position
myfigure = matplotlib.pyplot.figure(figsize = (7,8)) # generate a object from␣
 ↪the figure class, set aspect ratio
matplotlib.pyplot.scatter(weibull_pp, sample ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of RV")
mytitle = " Gumbell Extreme Value Type 1 Distribution Data Model sample mean = :
 ↪ " + str(sample_mean)+ " sample variance =:" + str(sample_variance)
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[228]:



[229]:

```
myguess = 10000
print(alpha_mom,beta_mom)
```

```
print(ev1dist(myguess,alpha_mom,beta_mom)) #
```

```
8730.888840854457 6717.889629784229
0.43698485405073
```

[230]:
```
from scipy.optimize import newton

def f(x):
    alpha = 8730.888840854457
    beta = 6717.889629784229
    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))
```

```
6536.595933014119
```

2.3 Gumbell Double Exponential (EV1) Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	6536.595933014118	75% chance of greater value
50%	11193.082189211951	50% chance of greater value
75%	17100.702987342494	25% chance of greater value
90%	23848.608172211992	10% chance of greater value
99%	39634.183626876766	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	50473.21664381972	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	55133.06604940008	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

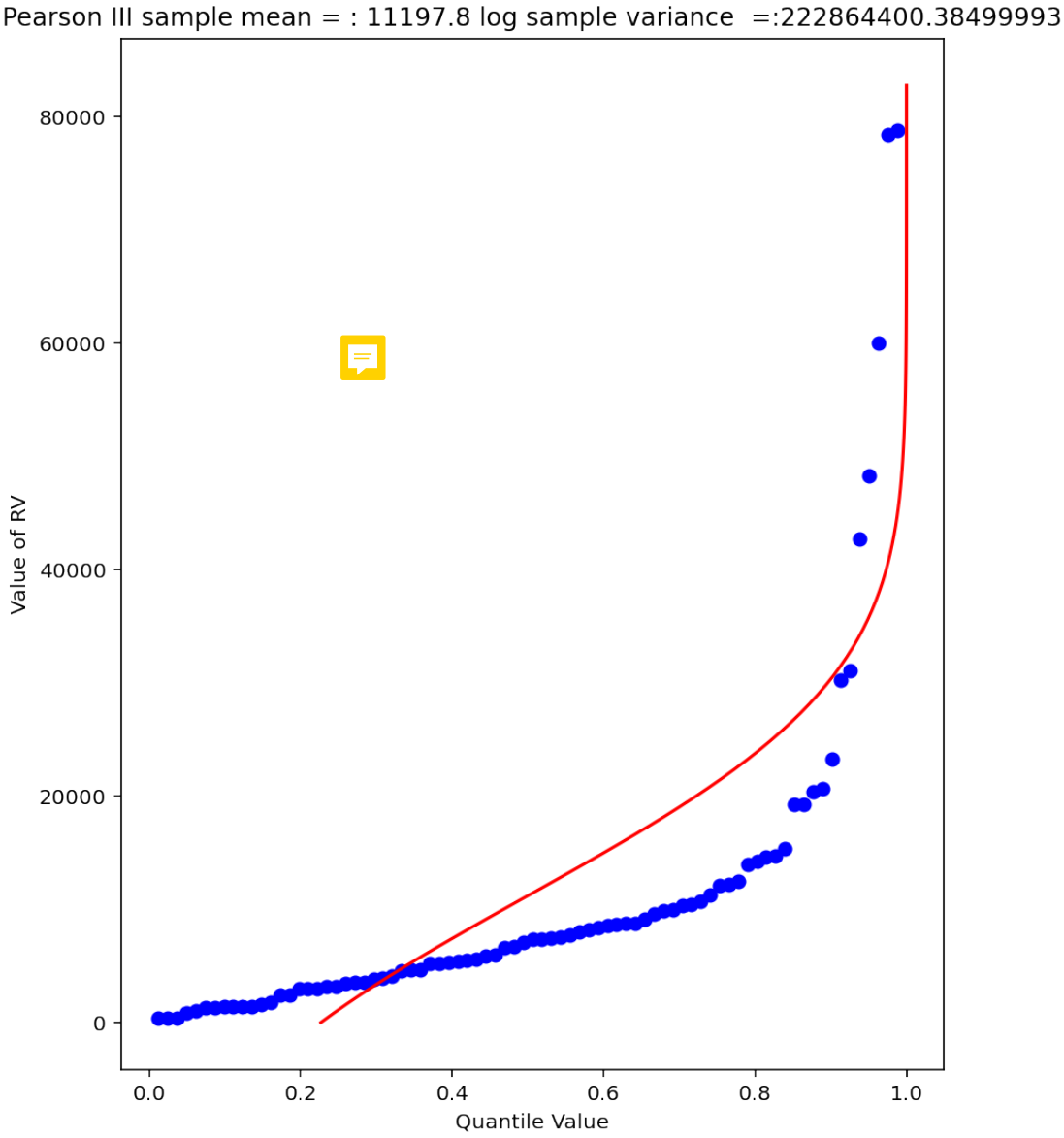
[231]:
```
# Gamma (Pearson Type III) Quantile Function
```

[232]:
```
def gammacdf(x,tau,alpha,beta): # Gamma Cumulative Density function - with␣
↪three parameter to one parameter convert
    xhat = x-tau
    lamda = 1.0/beta
    gammacdf = scipy.stats.gamma.cdf(lamda*xhat, alpha)
```

```
    return gammacdf
```

[233]:
```
# Fitting Data to Pearson (Gamma) III Data Model
# This is new, in lecture the fit was to log-Pearson, same procedure, but not␣
 ↪log transformed
import scipy.stats # import scipy stats package
import math        # import math package
import numpy       # import numpy package
# log and antilog
def loggit(x):  # A prototype function to log transform x
    return(math.log(x))
def antiloggit(x):  # A prototype function to log transform x
    return(math.exp(x))
logsample = df['flow'].tolist() # put the peaks into a list
sample_mean = numpy.array(logsample).mean()
sample_variance = numpy.array(logsample).std()**2
sample_stdev = numpy.array(logsample).std()
sample_skew  = scipy.stats.skew(logsample)
sample_alpha = 4.0/(sample_skew**2)
sample_beta = numpy.sign(sample_skew)*math.sqrt(sample_stdev**2/sample_alpha)
sample_tau   = sample_mean - sample_alpha*sample_beta
logsample.sort() # sort the sample in place!
weibull_pp = [] # built a relative frequency approximation to probability,␣
 ↪assume each pick is equally likely
for i in range(0,len(logsample),1):
    weibull_pp.append((i+1)/(len(logsample)+1))
#################
mu = sample_mean # Fitted Model in Log Space
sigma = math.sqrt(sample_variance)
x = []; ycdf = []
xlow = 1; xhigh = 1.05*max(logsample) ; 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)
# Now plot the sample values and plotting position
myfigure = matplotlib.pyplot.figure(figsize = (7,9)) # generate a object from␣
 ↪the figure class, set aspect ratio
matplotlib.pyplot.scatter(weibull_pp, logsample ,color ='blue')
matplotlib.pyplot.plot(ycdf, x, color ='red')
matplotlib.pyplot.xlabel("Quantile Value")
matplotlib.pyplot.ylabel("Value of RV")
mytitle = "Pearson III sample mean = : " + str(sample_mean)+ " log sample␣
 ↪variance  =:" + str(sample_variance)
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[233]:



[234]:
```
print(sample_tau)
print(sample_alpha)
print(sample_beta)
```

```
1356.6347332854966
0.43456260236933913
22646.13938948754
```

[235]:
```
from scipy.optimize import newton

def f(x):
```

```
    sample_tau = 1356.6347332854966
    sample_alpha = 0.43456260236933913
    sample_beta = 22646.13938948754
    quantile = 0.999
    argument = x
    gammavalue = gammacdf(argument,sample_tau,sample_alpha,sample_beta)
    return gammavalue - quantile

myguess =  60000
print(newton(f, myguess))
```

```
118564.83951972675
```

2.4 Pearson III Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	2077.9070960392946	75% chance of greater value
50%	5267.0259459809095	50% chance of greater value
75%	14029.404138740409	25% chance of greater value
90%	28734.91475283843	10% chance of greater value
99%	71869.96373024615	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	104294.44062157601	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	118564.83951972675	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

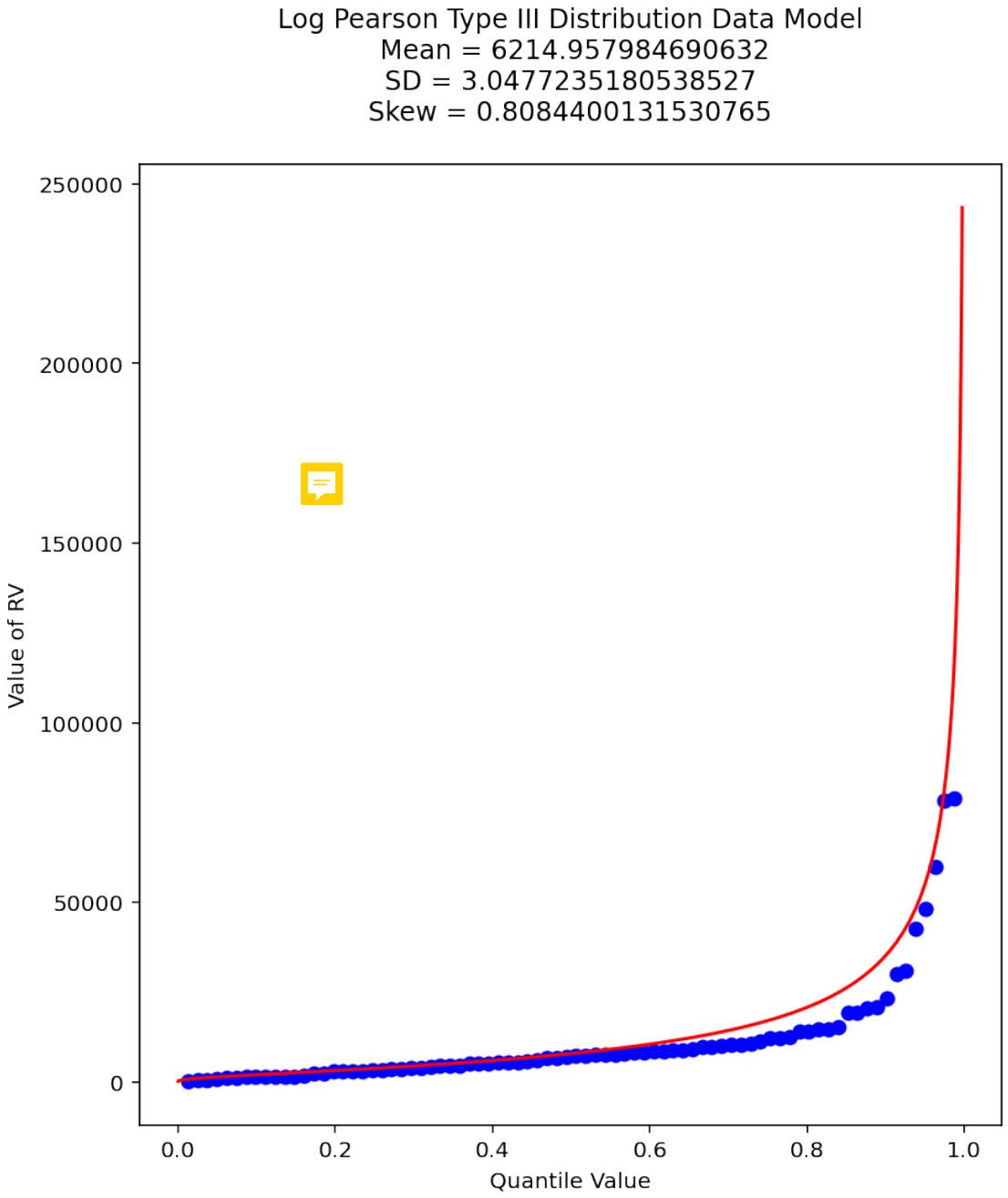
[236]:
```
def gammacdf(x,tau,alpha,beta): # Gamma Cumulative Density function - with␣
 ↪three parameter to one parameter convert
    xhat = x-tau
    lamda = 1.0/beta
    gammacdf = scipy.stats.gamma.cdf(lamda*xhat, alpha)
    return gammacdf
#do 1 min percentile
```

[237]:
```
import scipy.stats # import scipy stats package
import math        # import math package
import numpy      # import numpy package
# log and antilog
def loggit(x):  # A prototype function to log transform x
    return(math.log(x))
```

```
def antiloggit(x):  # A prototype function to log transform x
    return(math.exp(x))
def weibull_pp(sample): # plotting position function
# returns a list of plotting positions; sample must be a numeric list
    weibull_pp = [] # null list to return after fill
    sample.sort() # sort the sample list in place
    for i in range(0,len(sample),1):
        weibull_pp.append((i+1)/(len(sample)+1))
    return weibull_pp
def gammacdf(x,tau,alpha,beta): # Gamma Cumulative Density function - with␣
 ↪three parameter to one parameter convert
    xhat = x-tau
    lamda = 1.0/beta
    gammacdf = scipy.stats.gamma.cdf(lamda*xhat, alpha)
    return gammacdf
#samp = numpy.sign(sample_skew)*math.sqrt(sample_stdev**2/sample_alpha)le =␣
 ↪beargrass['Peak'].tolist() # put the peaks into a list
sample = df['flow'].apply(loggit).tolist() # put the log peaks into a list
sample_mean = numpy.array(sample).mean()
sample_stdev = numpy.array(sample).std()
sample_skew  = scipy.stats.skew(sample)
sample_alpha = 4.0/(sample_skew**2)
sample_beta = numpy.sign(sample_skew)*math.sqrt(sample_stdev**2/sample_alpha)
sample_tau = sample_mean - sample_alpha*sample_beta
plotting = weibull_pp(sample)
x = []; ycdf = []
xlow = (0.9*min(sample)); xhigh = (1.1*max(sample)) ; howMany = 100
xstep = (xhigh - xlow)/howMany
for i in range(0,howMany+1,1):
    x.append(xlow + i*xstep)
    yvalue = gammacdf(xlow + i*xstep,sample_tau,sample_alpha,sample_beta)
    ycdf.append(yvalue)
# reverse transform the peaks, and the data model peaks
for i in range(len(sample)):
    sample[i] = antiloggit(sample[i])
for i in range(len(x)):
    x[i] = antiloggit(x[i])
rycdf = ycdf[::-1]
myfigure = matplotlib.pyplot.figure(figsize = (7,8)) # generate a object from␣
 ↪the figure class, set aspect ratio
matplotlib.pyplot.scatter(plotting, sample ,color ='blue')
matplotlib.pyplot.plot(rycdf, 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(sample_mean)) + "\n"
mytitle += "SD = " + str(antiloggit(sample_stdev)) + "\n"
```

```
mytitle += "Skew = " + str(antiloggit(sample_skew)) + "\n"
matplotlib.pyplot.title(mytitle)
matplotlib.pyplot.show()
```

[237]:



[238]:

```
lp3 = round(gammacdf(loggit(2999),sample_tau,sample_alpha,sample_beta),4)
print(1-lp3)
```

```
75000
```

```
0.25
```

[238]: 75000

2.5 Log-Pearson III Distribution Data Model

Exceedence Probability	Flow Value	Remarks
25%	2999	75% chance of greater value
50%	6466	50% chance of greater value
75%	13446	25% chance of greater value
90%	25210	10% chance of greater value
99%	69691.44888201726	1% chance of greater value (in flood statistics, this is the 1 in 100-yr chance event)
99.8%	115401.36641105622	0.002% chance of greater value (in flood statistics, this is the 1 in 500-yr chance event)
99.9%	139261.65142255824	0.001% chance of greater value (in flood statistics, this is the 1 in 1000-yr chance event)

3 Summary of "Best" Data Model based on Graphical Fit

The "best" visual fit for the data is the log-normal data distribution model in my opinion. A close second is the Log-Pearson III graph has a so-so visual fit, since it follows the earlier percentile but, after the 60% percentile it goes above the plotted points and it on top of them for the remainder of the graph. Gumbell was in the middle of the all the graphical fits because it outlined the graphs nearly perfectly if the points were shifted vertically, until the higher percentiles. Lastly, The Normal and Pearson III had the similar distributions that never stayed on the line so the visual fit was the worse out of all of the graphs. Additionally, I had to use the guess function to fill out the table for Log-Pearson so that is why there is no decimals included.