Homework 3

Problem 1

Data:
HJAndrews_peakflow_WS1_WS2_WS3.xlsx

For this problem, consider only differences between WS1 and WS2. These two watersheds are adjacent to each other, and we expect they experience the same storms leading to peak runoff. WS2 is the control, but WS1 was actively 100% clearcut from late 1962 to 1966. Here we want to test whether the difference in peak flows between WS1 and WS2 is statistically different in 4 periods (labeled Index12), where 1 indicates the control period pre-treatment, 2 indicates the period of active treatment, 3 indicates the first 15 years post-treatment (when the forest would start to recover), and 4 indicates longer than 15 years post-treatment. We want to know whether the four periods are statistically different from each other, and if so, which one or ones are statistically different from which other ones.

In [1]:

```
import numpy as np
import scipy.stats as st
from scipy.io import loadmat
import statistics as stats
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
```

In [2]:

```
df=pd.read_excel('HJAndrews_peakflow_WS1_WS2_WS3.xlsx')
```

In [3]:

```
#rearrage the dataset
df_hja=df[2:].copy()
df_hja.columns=['year','WS1','WS2','WS3','I12','I23']
df_hja.head()
```

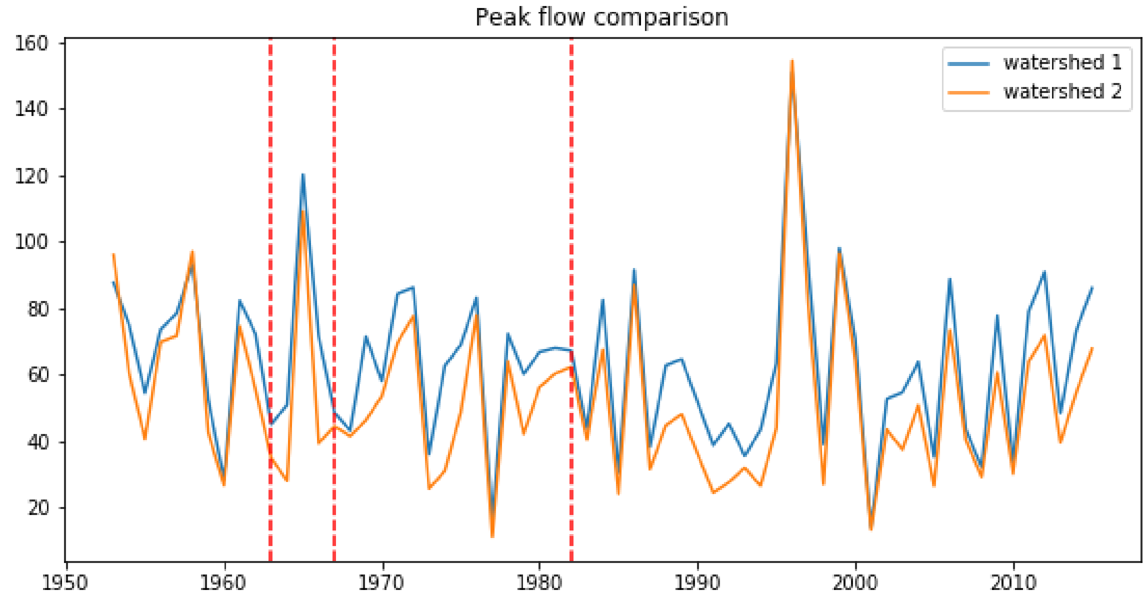
Out[3]:

	year	WS1	WS2	WS3	I12	I23
2	1953	87.5495	96.0073	81.9364	1	1
3	1954	74.7993	60.2205	50.7975	1	1
4	1955	54.4041	40.5364	35.1773	1	1
5	1956	73.5548	69.7704	54.6327	1	1
6	1957	78.3552	71.5483	57.1218	1	1

A) First, plot the timeseries of peakflow as a function of wateryear for both watershed 1 and watershed 2 on the same graph, with vertical dashed lines to indicate the different periods (put a vertical dashed line in 1963, in 1967, and in 1982).

In [4]:

```
plt.figure(figsize=(10,5))

plt.plot(df_hja['year'],df_hja['WS1'],label='watershed 1')
plt.plot(df_hja['year'],df_hja['WS2'],label='watershed 2')
plt.axvline(1963,linestyle='--',color='red')
plt.axvline(1967,linestyle='--',color='red')
plt.axvline(1982,linestyle='--',color='red')
plt.title('Peak flow comparison')

plt.legend(loc='best')
plt.show()
```



B) It has been suggested that paired data such as this can be made to be closer to normally distributed by taking the log of each value before subtracting. Create two series: Q12=Peakflow1-Peakflow2; and Qlog12=log(Peakflow1)- log(Peakflow2); and make graphs to demonstrate which is closer to normally distributed. Given that we want to use an ANOVA analysis, why is it important to do a transformation to get the data closer to normally distributed?

In [5]:

```
#Create columns with the Q12 and Qlog12
df_hja['Q12']=df_hja['WS1']-df_hja['WS2']
df_hja['Qlog12']=np.log(list(df_hja['WS1']))-np.log(list(df_hja['WS2']))
df_hja.head()
```

Out[5]:

	year	WS1	WS2	WS3	I12	I23	Q12	Qlog12
2	1953	87.5495	96.0073	81.9364	1	1	-8.4578	-0.092220
3	1954	74.7993	60.2205	50.7975	1	1	14.5788	0.216796
4	1955	54.4041	40.5364	35.1773	1	1	13.8677	0.294239
5	1956	73.5548	69.7704	54.6327	1	1	3.7844	0.052821
6	1957	78.3552	71.5483	57.1218	1	1	6.8069	0.090880

In [6]:

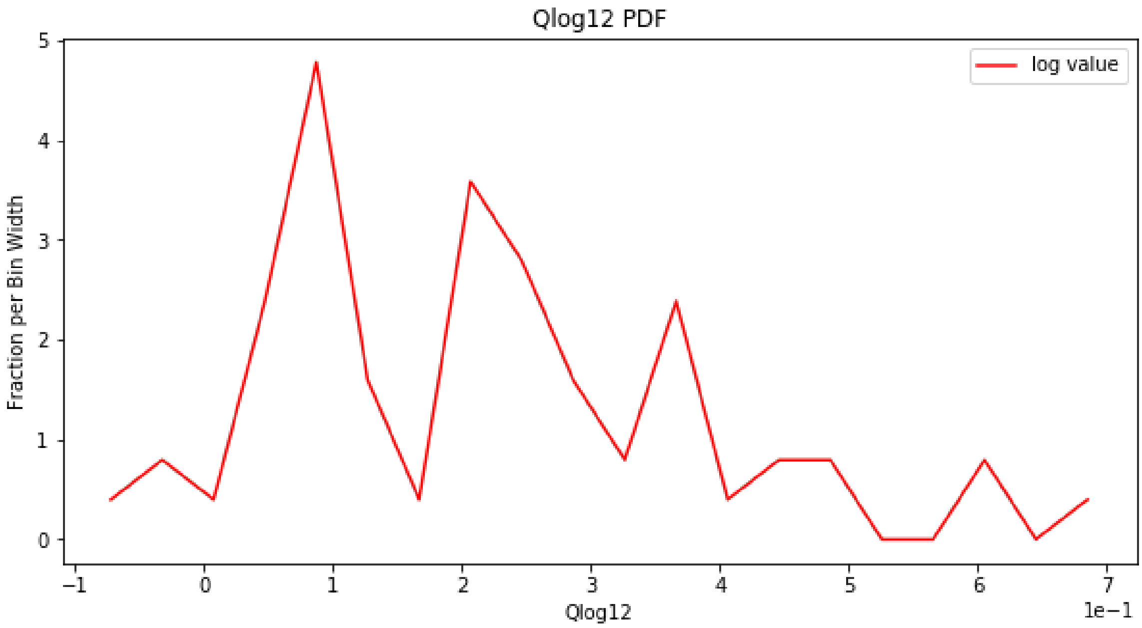
```
def pdf_fn(data, nbins):
    counts, bins, patches = plt.hist(data, nbins)
    plt.close()
    width = bins[2]-bins[1]
    centers = bins + width/2
    centers_list = np.array(centers).tolist()
    # Then, remove the last number from the list.
    centers_list.remove(centers_list[len(centers_list)-1])
    areas = [c * width for c in counts]
    area_under_curve = sum(areas)
    fractions = [c / area_under_curve for c in counts]
    return centers_list, fractions
```

In [7]:

```
#Create the comparison plot
nbins = 20
#Generate data for the plots,_w for whole period, _b for before, _a for after
cl, frac = pdf_fn(list(df_hja['Q12']), nbins)
cl_log,frac_log = pdf_fn(list(df_hja['Qlog12']), nbins)

plt.figure(1,figsize=(10,5))

plt.plot(cl, frac,'r',linestyle='-',label='original value')
plt.xlabel('Q12')
plt.ylabel('Fraction per Bin Width')
plt.title('Q12 PDF')
plt.ticklabel_format(axis='x', style='sci', scilimits=(0,0))
plt.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
plt.legend(loc='best')

plt.figure(2,figsize=(10,5))

plt.plot(cl_log, frac_log,'r',linestyle='-',label='log value')
plt.xlabel('Qlog12')
plt.ylabel('Fraction per Bin Width')
plt.title('Qlog12 PDF')
plt.ticklabel_format(axis='x', style='sci', scilimits=(0,0))
plt.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
plt.legend(loc='best')
```

Out[7]:

<matplotlib.legend.Legend at 0xb472e48>



Answer

As the result shows and according to the literature, Qlog12 is closer to the normal distribution. We tend to use the normal distribution is firstly because the anova test was built upon the hypothesis that our dataset follows normal distribution. Also because the non-normally distributed data will increase the chance of false positive results.

C) State the null and the alternative hypothesis for the question of whether the four periods are statistically different from each other. State the type I error (alpha value) that you are willing to accept.

Based on the lecture notes in class and from the Portland state university website, we propose a null and alternative hypothesis.

$H0$: The means of Qlog12(Peakflow1-Peakflow2) from 4 periods have the same central mean

$H1$: The means of Qlog12(Peakflow1-Peakflow2) from 4 periods are different from each other

In this case, we perform a one way ANOVA to determine whether our null hypothesis ($H0$) is true or not.

In this question the type I error(alpha value) = 0.05 is acceptable to me, which indicate the 95% confidence interval.

D) Perform an ANOVA test and discuss the results, related both to your hypothesis test listed above and to the more detailed question of which groups are statistically different from which other groups. Include graphs and/or tables that illustrate your results, and be sure to discuss what they mean. It's fine to use computer software here, but be sure that you understand what the code is doing and outputting.

In [8]:

```
#rebuild the data, choose Qlog12 as our dataset
df_anova=pd.DataFrame(columns=['index','treatments','Qlog12'])
df_anova['treatments']=df_hja['I12']
df_anova['Qlog12']=df_hja['Qlog12']
df_anova = df_anova.reset_index(drop=True)
df_anova.head()
```

Out[8]:

	index	treatments	Qlog12
0	NaN	1	-0.092220
1	NaN	1	0.216796
2	NaN	1	0.294239
3	NaN	1	0.052821
4	NaN	1	0.090880

In [9]:

```
#break down the data to create melted dataframe
df1=df_anova[df_anova['treatments']==1]
df1['treatments']='P1'
df2=df_anova[df_anova['treatments']==2]
df2['treatments']='P2'
df3=df_anova[df_anova['treatments']==3]
df3['treatments']='P3'
df4=df_anova[df_anova['treatments']==4]
df4['treatments']='P4'
df1['index']=list(df1.index)
df2 = df2.reset_index(drop=True)
df2['index']=list(df2.index)
df3 = df3.reset_index(drop=True)
df3['index']=list(df3.index)
df4 = df4.reset_index(drop=True)
df4['index']=list(df4.index)
df1=df1.append(df2,ignore_index=True)
df1=df1.append(df3,ignore_index=True)
df1=df1.append(df4,ignore_index=True)
```

```
E:\tools\py\lib\site-packages\ipykernel_launcher.py:3: 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
  This is separate from the ipykernel package so we can avoid doing imports until
E:\tools\py\lib\site-packages\ipykernel_launcher.py:5: 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
  """
E:\tools\py\lib\site-packages\ipykernel_launcher.py:7: 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
  import sys
E:\tools\py\lib\site-packages\ipykernel_launcher.py:9: 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
  if __name__ == '__main__':
E:\tools\py\lib\site-packages\ipykernel_launcher.py:10: 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
  # Remove the CWD from sys.path while we load stuff.
```

In [10]:

```
# get ANOVA table as R like output
import statsmodels.api as sm
from statsmodels.formula.api import ols
# reshape the d dataframe suitable for statsmodels package
#d_melt = pd.melt(df_hja.reset_index(), id_vars=['index'], value_vars=['WS2', 'WS1'])
# replace column names
df1.columns = ['index', 'treatments', 'value']
df1['value'] = df1['value'].astype(float)
# Ordinary Least Squares (OLS) model
model = ols('value ~ C(treatments)', data=df1).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
anova_table
```

Out[10]:

	sum_sq	df	F	PR(>F)
C(treatments)	0.226959	3.0	3.094333	0.033665
Residual	1.442482	59.0	NaN	NaN

In [11]:

```
# load packages
from statsmodels.stats.multicomp import pairwise_tukeyhsd
# perform multiple pairwise comparison (Tukey HSD)
m_comp = pairwise_tukeyhsd(endog=df1['value'], groups=df1['treatments'], alpha=0.05)
print(m_comp)
```

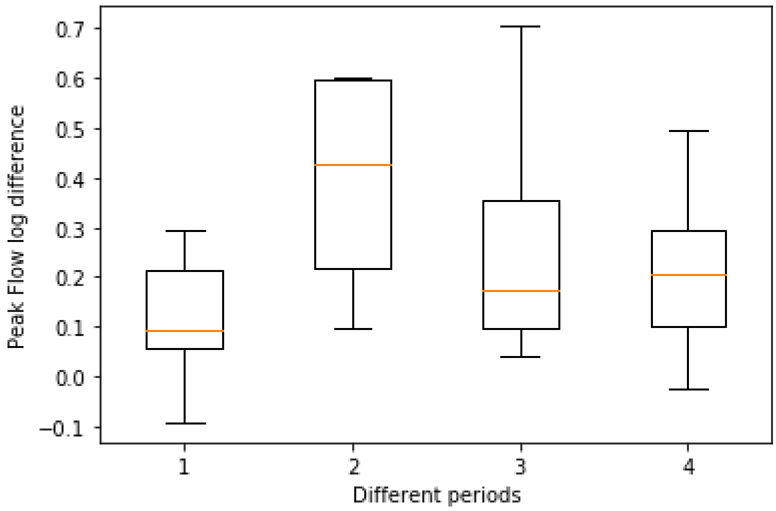
```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
    P1     P2   0.2724 0.0233  0.0278  0.517   True
    P1     P3   0.1233 0.2263 -0.0455 0.2921  False
    P1     P4   0.1019 0.2786 -0.0468 0.2506  False
    P2     P3  -0.1491  0.336 -0.3817 0.0835  False
    P2     P4  -0.1705 0.1773  -0.389  0.048  False
    P3     P4  -0.0214    0.9 -0.1496 0.1067  False
---------------------------------------------------
```

In [12]:

```
#Also make a boxplot for visualization
df1=df_anova[df_anova['treatments']==1]
df2=df_anova[df_anova['treatments']==2]
df3=df_anova[df_anova['treatments']==3]
df4=df_anova[df_anova['treatments']==4]

plt.boxplot((df1['Qlog12'],df2['Qlog12'],df3['Qlog12'],df4['Qlog12']))
plt.xlabel('Different periods')
plt.ylabel('Peak Flow log difference')
```

Out[12]:

```
Text(0, 0.5, 'Peak Flow log difference')
```



According to the results shows above, period 1 is different from period 2 while other comparions are basically giving there is not significant change occurs.

Problem 2

USGS gaged streamflow records for the Columbia River at The Dalles, OR began in 1878 and continues to the present day (one of the longest continuous records in the U.S.). Peak flow records (based on peak stage values recorded by railroad workers), however, extend back even farther, to 1858. Using coincident peak flow records from 1879-1932 (a period with no major storage dams on the Columbia):

A) First, isolate the period of relevant overlap (1879-1932) and plot the timeseries. Create a regression model for annual flow using spring peak flow as an explanatory variable.

In [13]:

```
import scipy.stats as st
import scipy.io as sio
import scipy.stats as st
%matplotlib inline
```

In [14]:

```
df=pd.read_excel('dalles_flow.xlsx')
```

In [15]:

```
#Rearrange the dataset

df_r=df.iloc[9:81,2:5].copy()
df_r.columns=['years','peakflow','annualmean']
df_r=df_r.reset_index()
df_r=pd.DataFrame(df_r,columns=['years','peakflow','annualmean'])
df_r=df_r.iloc[:54,:]
df_r.tail()
```

Out[15]:

	years	peakflow	annualmean
49	1928	766000	243395
50	1929	460000	142389
51	1930	332000	142941
52	1931	308000	131230
53	1932	578000	200003

In [16]:

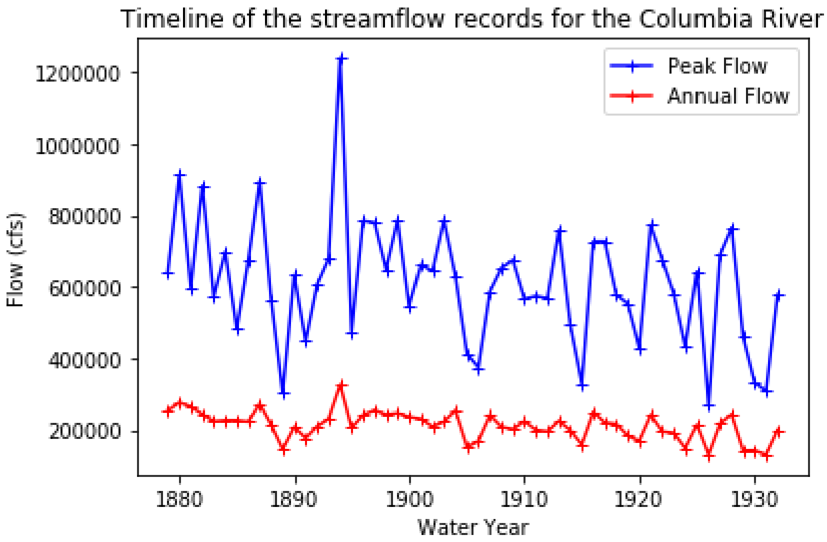
```
df_r['peakflow']=df_r['peakflow'].astype(int)
df_r['annualmean']=df_r['annualmean'].astype(int)
```

In [17]:

```
#First plot the data
plt.plot(df_r['years'],df_r['peakflow'],'b-+', label='Peak Flow');
plt.plot(df_r['years'],df_r['annualmean'],'r-+', label='Annual Flow');

plt.title('Timeline of the streamflow records for the Columbia River')
plt.xlabel('Water Year')
plt.ylabel('Flow (cfs)');
plt.legend(loc="best")
```

Out[17]:

```
<matplotlib.legend.Legend at 0xccd1048>
```

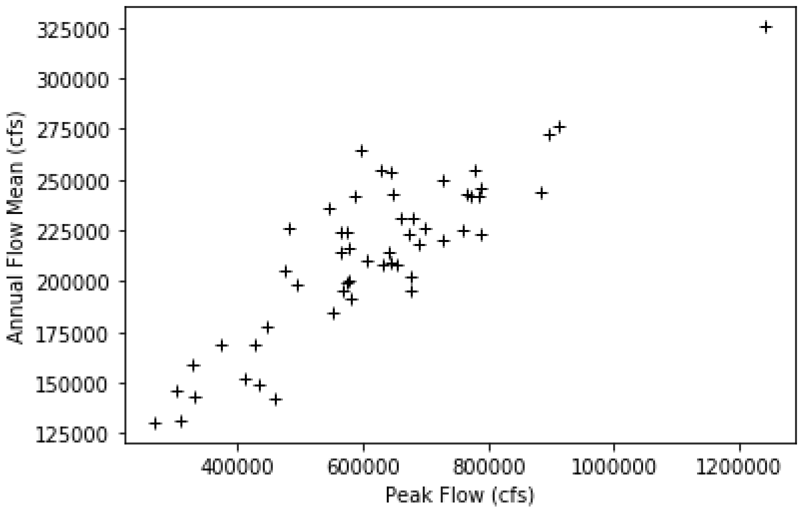


In [18]:

```
#Create scatter plot for the data

plt.plot(df_r['peakflow'],df_r['annualmean'],'k+')
plt.xlabel('Peak Flow (cfs)')
plt.ylabel('Annual Flow Mean (cfs)')
```

Out[18]:

```
Text(0, 0.5, 'Annual Flow Mean (cfs)')
```



In [19]:

```
n = df_r['peakflow'].size
#According to the scatter plot we choose linear regression model to fit the data
Bv2 = np.polyfit(df_r['peakflow'],df_r['annualmean'],1)
print(Bv2)

#n = df_r['peakflow'].size
x = np.linspace(np.min(df_r['peakflow']), np.max(df_r['peakflow']),n)
y = Bv2[1] + Bv2[0]*x
```

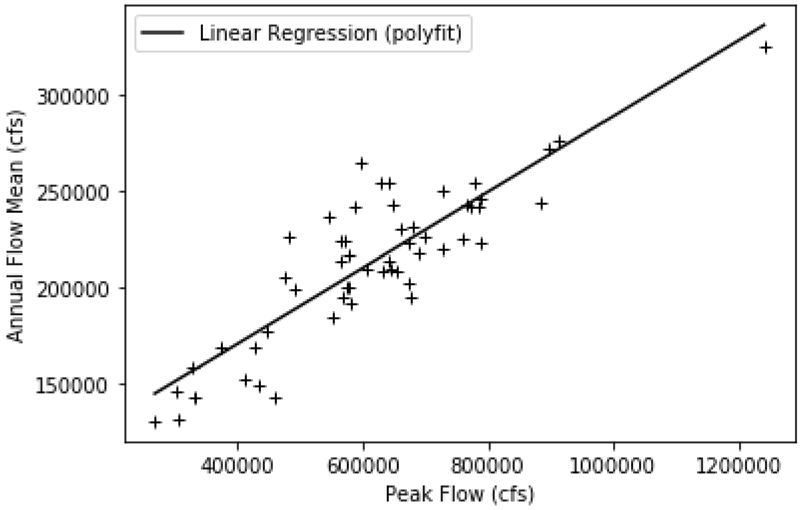
```
[1.97307224e-01 9.17957273e+04]
```

In [20]:

```
plt.plot(df_r['peakflow'],df_r['annualmean'],'k+')
plt.xlabel('Peak Flow (cfs)')
plt.ylabel('Annual Flow Mean (cfs)')
plt.plot(x,y,'k-',label='Linear Regression (polyfit)')
plt.legend()
```

Out[20]:

```
<matplotlib.legend.Legend at 0xcdc1d30>
```



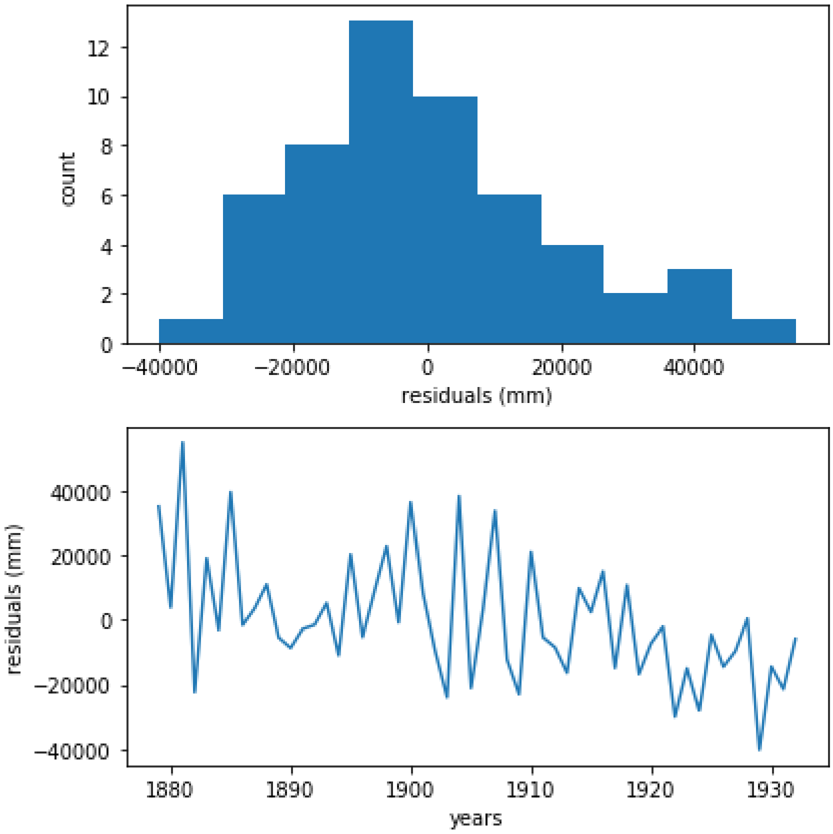
B) How much of the variance is explained by the resulting model?

In [21]:

```
resid = df_r['annualmean'] - (Bv2[1] + Bv2[0]*(df_r['peakflow']))
```

In [22]:

```
f, (ax1, ax2) = plt.subplots(2,1,figsize=(6,6))
ax1.hist(resid)
ax1.set_xlabel('residuals (mm)')
ax1.set_ylabel('count')


ax2.plot(df_r['years'],resid)
ax2.set_xlabel('years')
ax2.set_ylabel('residuals (mm)')

f.tight_layout()
```



In [23]:

```
#Calculate R P value from st.pearsonr()function
R, P = st.pearsonr(df_r['peakflow'],df_r['annualmean'])
print(R)
print(R*R)
print(P)
```

```
0.8712297880006823
0.7590413434997139
1.0707931769744675e-17
```

Answer

As the result shows, about 76% of variance were explained by the model.

C) Estimate the 95% confidence bounds for the annual flow estimates from 1858- 1877, and plot them with the central tendency (the prediction from the regression model).

In [24]:

```
#read the data for stream flow record
df_e=df.iloc[9:104,8:10].copy()
df_e.columns=['years','peakflow']
df_e=df_e.reset_index()
df_e=pd.DataFrame(df_e,columns=['years','peakflow'])
df_e['peakflow']=df_e['peakflow'].astype(np.float64)
df_e['years']=df_e['years'].astype(int)
df_e.tail()
```

Out[24]:

	years	peakflow
88	1946	583000.0
89	1947	542000.0
90	1948	1010000.0
91	1949	624000.0
92	1950	744000.0

In [25]:

```
#create subset from 1858-1877
df_s=df_e[df_e['years']<=1877]
n = df_s['peakflow'].size
x = np.linspace(np.min(df_s['peakflow']), np.max(df_s['peakflow']),n)
y = Bv2[1] + Bv2[0]*x
df_s['annualmean_pred']=Bv2[1] + Bv2[0]*(df_s['peakflow'])
```

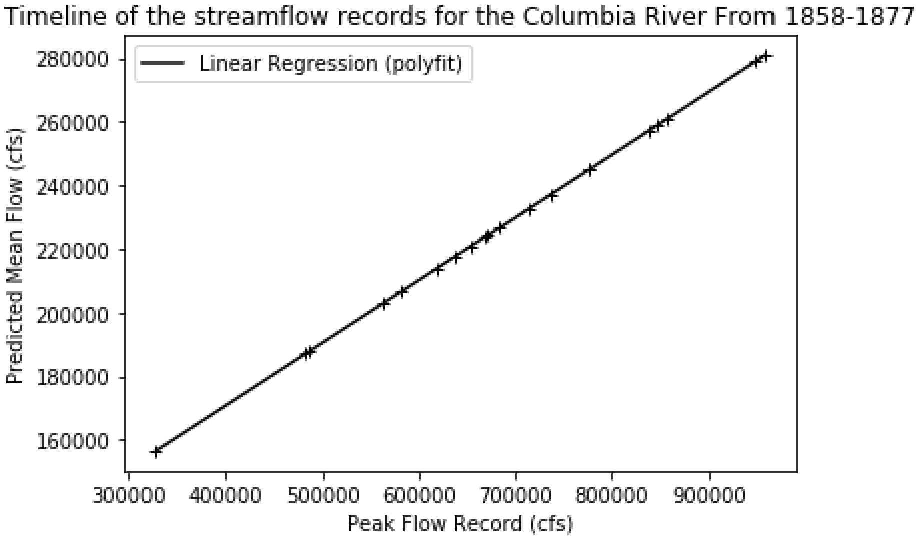
```
E:\tools\py\lib\site-packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

In [26]:

```
plt.plot(df_s['peakflow'],df_s['annualmean_pred'],'k+')
plt.plot(x,y,'k-',label='Linear Regression (polyfit)')
plt.title('Timeline of the streamflow records for the Columbia River From 1858-1877')
plt.xlabel('Peak Flow Record (cfs)')
plt.ylabel('Predicted Mean Flow (cfs)')
plt.legend()
```

Out[26]:

```
<matplotlib.legend.Legend at 0xce99198>
```



In [78]:

```
#set a=0.05
#calculate 95% confidence interval
n=df_s['years'].size
print(n)
t_095_18=1.734

SSE = np.sum((Bv2[1] + Bv2[0]*df_r['peakflow'] - df_r['annualmean'])**2);

standard_err_square = SSE/(df_r['peakflow'].size - 2)

x_avg=np.average(df_s['peakflow'])
com_var_x=standard_err_square*(1+(1/n)+(n*((df_s['peakflow']-x_avg)**2))/(n*(np.sum(df_s['peakfl
ow']**2))-((np.sum(df_s['peakflow']))**2)))
ci_95=t_095_18*np.sqrt(com_var_x)
df_s['upper']=df_s['annualmean_pred']+ci_95
df_s['lower']=df_s['annualmean_pred']-ci_95
```

```
20

E:\tools\py\lib\site-packages\ipykernel_launcher.py:14: 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

E:\tools\py\lib\site-packages\ipykernel_launcher.py:15: 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
  from ipykernel import kernelapp as app
```

In [79]:

```
df_s
```

Out[79]:

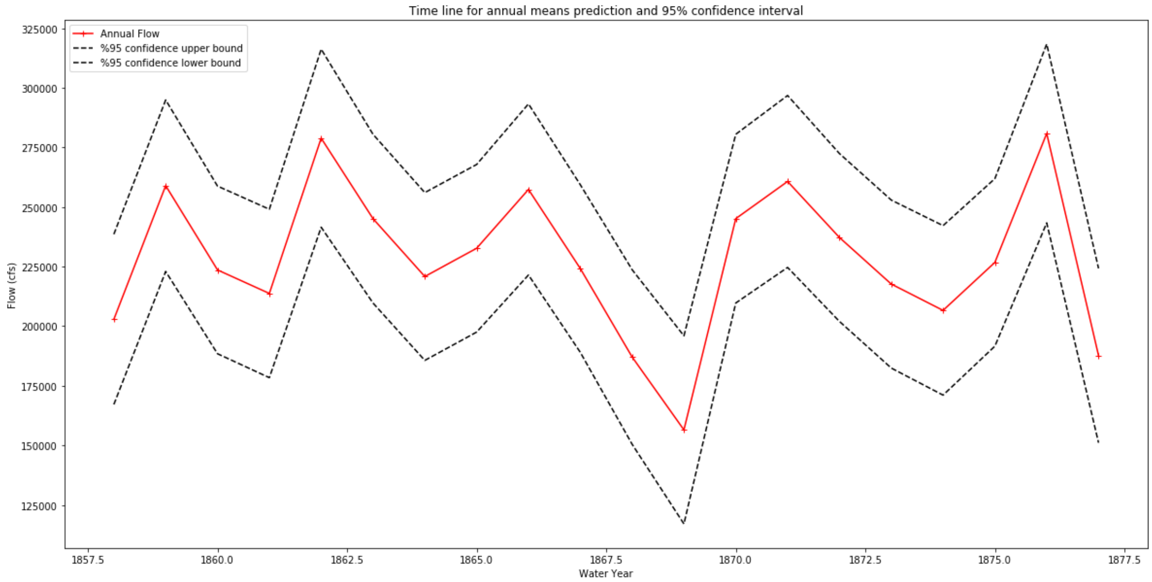
	years	peakflow	annualmean_pred	upper	lower
0	1858	563000.0	202879.694254	238572.264171	167187.124336
1	1859	847000.0	258914.945808	294869.277721	222960.613895
2	1860	668000.0	223596.952751	258743.254114	188450.651388
3	1861	618000.0	213731.591562	249044.665002	178418.518121
4	1862	948000.0	278842.975410	316176.466220	241509.484600
5	1863	777000.0	245103.440143	280483.132681	209723.747605
6	1864	654000.0	220834.651618	256010.340114	185658.963121
7	1865	714000.0	232673.085045	267818.115357	197528.054733
8	1866	839000.0	257336.488018	293208.831952	221464.144083
9	1867	671000.0	224188.874422	259330.637095	189047.111750
10	1868	483000.0	187095.116351	223692.585819	150497.646883
11	1869	328000.0	156512.496665	195940.280885	117084.712445
12	1870	777000.0	245103.440143	280483.132681	209723.747605
13	1871	856000.0	260690.710822	296742.208426	224639.213218
14	1872	737000.0	237211.151192	272410.319315	202011.983069
15	1873	638000.0	217677.736037	252903.524891	182451.947184
16	1874	582000.0	206628.531506	242167.104408	171089.958603
17	1875	684000.0	226753.868331	261883.144854	191624.591808
18	1876	958000.0	280816.047648	318319.272822	243312.822474
19	1877	486000.0	187687.038022	224243.307987	151130.768057

In [81]:

```
plt.figure(figsize=(20,10))
plt.plot(df_s['years'],df_s['annualmean_pred'],'r-+', label='Annual Flow')
plt.plot(df_s['years'],df_s['upper'],'k--', label='%95 confidence upper bound')
plt.plot(df_s['years'],df_s['lower'],'k--', label='%95 confidence lower bound')
plt.legend()
plt.title('Time line for annual means prediction and 95% confidence interval')
plt.xlabel('Water Year')
plt.ylabel('Flow (cfs)')
```

Out[81]:

```
Text(0, 0.5, 'Flow (cfs)')
```



D) Now create a non-parametric, quantile-based regression model using the same data.

In [46]:

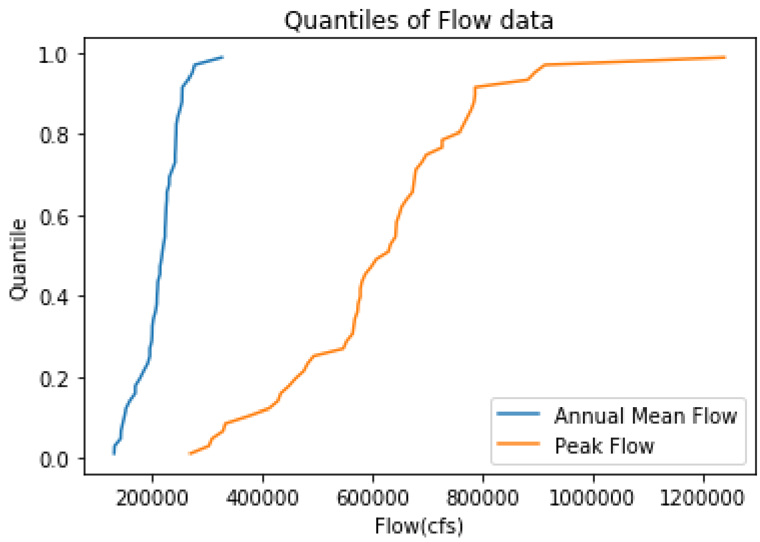
```
def quantile_fn(data):

    ordered_data = np.sort(data)
    n = len(ordered_data)

    rank = []
    plotting_position = []
    for i in range(n):
        rank.append(i+1)
        # Using the Cunnane plotting position.
        plotting_position.append((rank[i]-.4)/(n+.2))

    return ordered_data, plotting_position
```

In [47]:

```
y_ordered, y_quantile = quantile_fn(df_r['annualmean'])
x_ordered, x_quantile = quantile_fn(df_r['peakflow'])

#Let's try plotting these on the same plot and see how they compare.
plt.figure()
plt.plot(y_ordered, y_quantile, label='Annual Mean Flow')
plt.plot(x_ordered, x_quantile, label='Peak Flow')
plt.ylabel('Quantile')
plt.xlabel('Flow(cfs)')
plt.title('Quantiles of Flow data')
plt.legend(loc="best")
```

Out[47]:

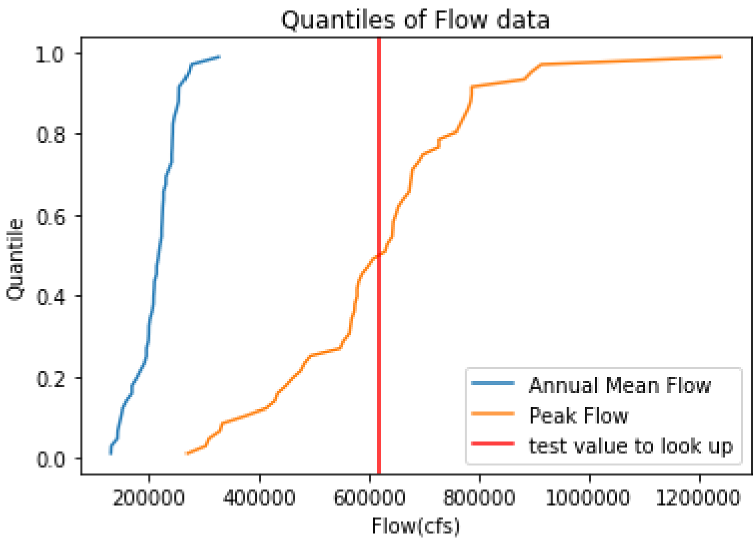
```
<matplotlib.legend.Legend at 0x93bf908>
```



In [49]:

```
from scipy.interpolate import interp1d
# Let's illustrate first by just looking up one value and plotting how we do it.
x_test=np.median(df_r['peakflow'])
print(x_test)

# Let's look up this physical value in the corresponding CDF
# first plot it for illustration
plt.figure()
plt.plot(y_ordered, y_quantile, label='Annual Mean Flow')
plt.plot(x_ordered, x_quantile, label='Peak Flow')
plt.ylabel('Quantile')
plt.xlabel('Flow(cfs)')
plt.title('Quantiles of Flow data')
plt.axvline(x_test,color='red',label='test value to look up')
plt.legend(loc="best")

# Now use the interpolation function to "look-up" what cdf-value that is at the intersection of
# our two lines in the graph.
f = interp1d(x_ordered, x_quantile)
print(f(x_test))
```

```
618000.0
0.5
```



In [50]:

```
# Let's resume where we were in the cell above.
f = interp1d(x_ordered, x_quantile)
print(f(x_test))
g = interp1d(y_quantile, y_ordered)
# Note the switch in the ordering of the variables in the interp1d function above
# Look up for the same value in other lines
print(g(f(x_test)))
```

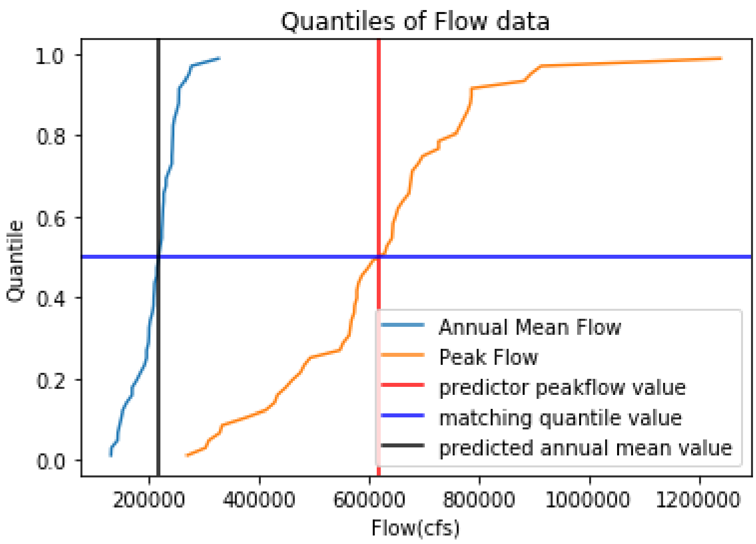
```
0.5
217500.5
```

In [51]:

```
# Let's plot all of that.
plt.figure()
plt.plot(y_ordered, y_quantile, label='Annual Mean Flow')
plt.plot(x_ordered, x_quantile, label='Peak Flow')
plt.ylabel('Quantile')
plt.xlabel('Flow(cfs)')
plt.title('Quantiles of Flow data')
plt.axvline(x_test,color='red',label='predictor peakflow value')
plt.axhline(f(x_test),color='blue',label='matching quantile value')
plt.axvline(g(f(x_test)),color='black',label='predicted annual mean value')
plt.legend(loc="best")
```

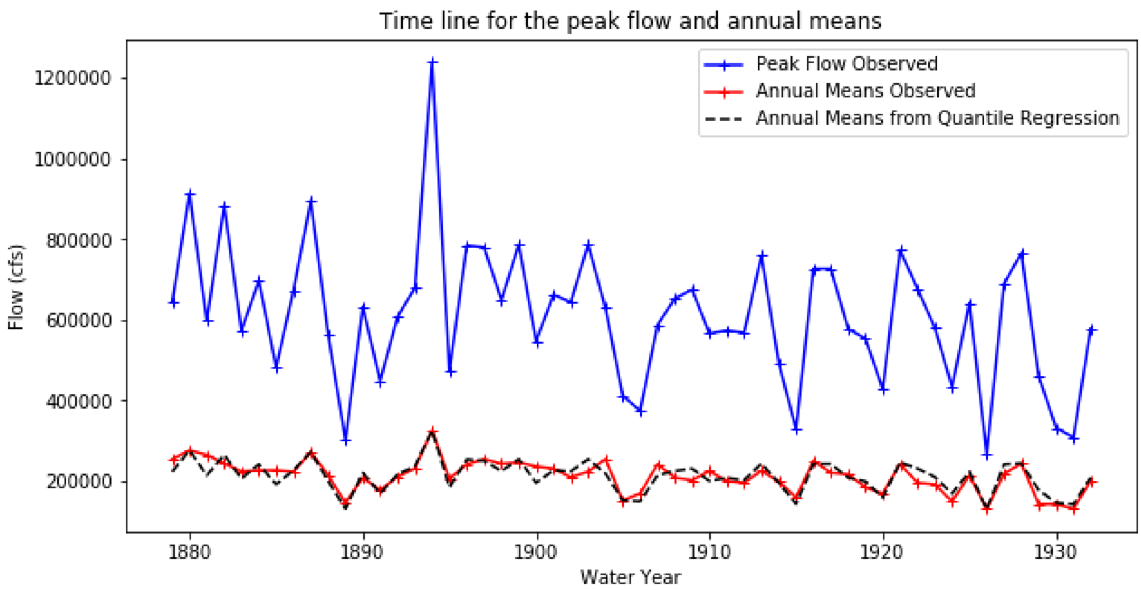
Out[51]:

<matplotlib.legend.Legend at 0xe02ccc0>



In [52]:

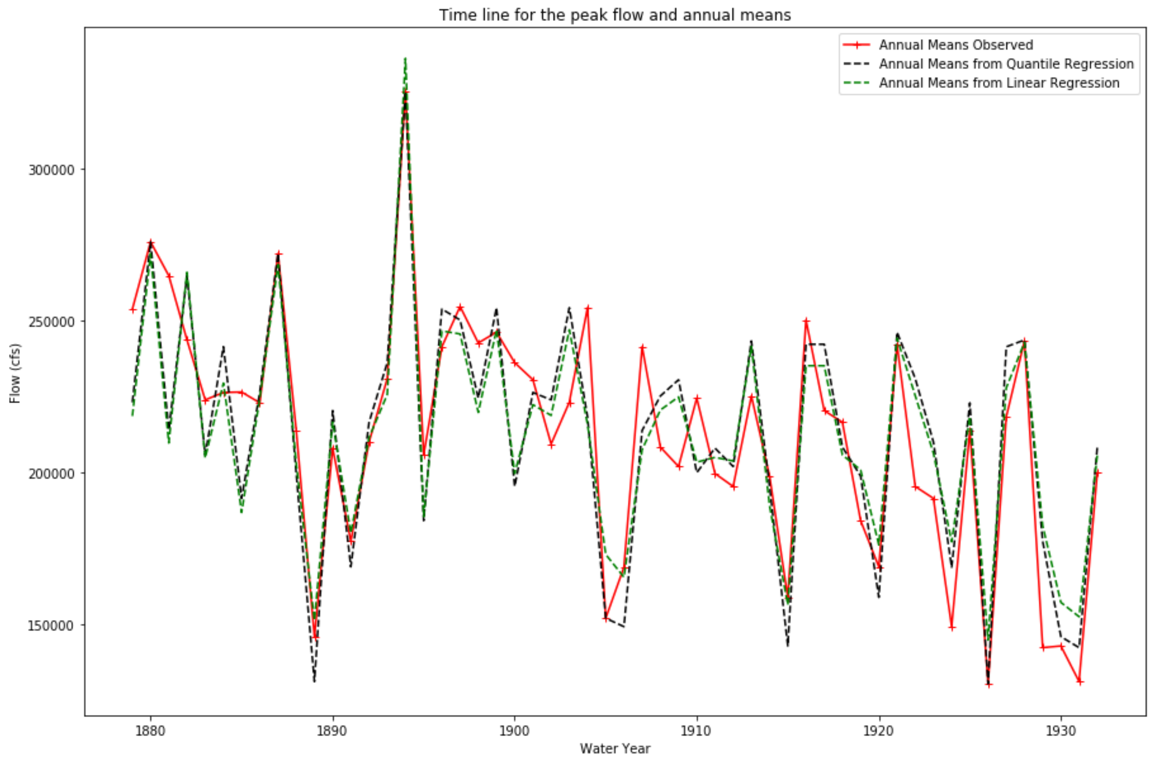
```
# Now, we can do this for every value in the Slide Canyon set to come up with a matching predict
ion for the Blue Canyon set
f = interp1d(x_ordered, x_quantile)

g = interp1d(y_quantile, y_ordered)

y_predicted=g(f(df_r['peakflow']))

# And we can see how well this did by making a time series plot of our actual and predicted valu
es
# Original data:
plt.figure(figsize=(10,5))
plt.plot(df_r['years'],df_r['peakflow'],'b-+', label='Peak Flow Observed');
plt.plot(df_r['years'],df_r['annualmean'],'r-+', label='Annual Means Observed');

# Predicted with linear regression between Slide Canyon and Blue Canyon
plt.plot(df_r['years'],y_predicted,'k--', label='Annual Means from Quantile Regression')
plt.legend()
plt.title('Time line for the peak flow and annual means')
plt.xlabel('Water Year')
plt.ylabel('Flow (cfs)');
```



E) Plot the predictions and residuals for the two different prediction models for the training period (1879-1932) and plot the model predictions for the pre-1878 data for the two different models. Is there a substantial difference between the two model formulations?

In [54]:

```
#Plot two different models on the same graph for the observed results
y_linear = Bv2[1] + Bv2[0]*df_r['peakflow']
plt.figure(figsize=(15,10))
plt.plot(df_r['years'],df_r['annualmean'],'r-+', label='Annual Means Observed')
plt.plot(df_r['years'],y_predicted,'k--', label='Annual Means from Quantile Regression')
plt.plot(df_r['years'],y_linear,'g--',label='Annual Means from Linear Regression')
plt.legend()
plt.title('Time line for the peak flow and annual means')
plt.xlabel('Water Year')
plt.ylabel('Flow (cfs)');
```



In [59]:

```
#calculate residues
col=['years','annualmean','resid_linear','resid_quantile']
df_resi=pd.DataFrame(df_r,columns=col)
df_resi['resid_linear'] = df_r['annualmean'] - (Bv2[1] + Bv2[0]*df_r['peakflow'])
df_resi['resid_quantile']=df_r['annualmean'] - y_predicted
df_resi
```

Out[59]:

	years	annualmean	resid_linear	resid_quantile
0	1879	253891	35226.727844	30697.0
1	1880	276025	3890.470199	0.0
2	1881	264888	55102.552914	50816.0
3	1882	243717	-22301.005864	-21171.0
4	1883	223983	19130.233509	18151.0
5	1884	226433	-3083.169464	-15052.0
6	1885	226570	39672.190873	35027.0
7	1886	223014	-1569.488870	-3556.0
8	1887	272184	3601.000227	0.0
9	1888	214072	10994.998523	14332.0
10	1889	146003	-5379.508846	14773.0
11	1890	208108	-8583.199918	-12361.0
12	1891	177420	-2769.363519	8415.0
13	1892	210089	-1472.212100	-6530.0
14	1893	231004	5236.667788	-5233.0
15	1894	325588	-10868.684754	0.0
16	1895	205832	20315.341439	21606.0
17	1896	241469	-5212.897933	-12422.0
18	1897	254739	9043.638186	4444.0
19	1898	242728	22879.884501	18146.0
20	1899	246303	-773.512381	-8045.0
21	1900	236237	36514.221327	40764.0
22	1901	230617	8203.890592	4184.0
23	1902	209451	-9410.579380	-14532.0
24	1903	223194	-23882.512381	-31154.0
25	1904	254348	38446.028977	35966.0
26	1905	152084	-21002.303463	0.0
27	1906	168880	3291.371041	19579.0
28	1907	241485	33869.932376	27556.0
29	1908	208443	-12194.344394	-16821.0
30	1909	202029	-22949.103317	-28588.0
31	1910	224582	21110.384075	24580.0
32	1911	199740	-5310.073715	-8368.0
33	1912	195459	-8407.230373	-6570.0
34	1913	225264	-16287.910115	-18131.0
35	1914	198896	9827.811411	3437.0
36	1915	158950	2437.503335	16009.0

	years	annualmean	resid_linear	resid_quantile
37	1916	250295	15056.921046	8003.0
38	1917	220469	-14769.078954	-21823.0
39	1918	216619	10779.697390	8176.0
40	1919	184226	-16680.622016	-14670.0
41	1920	169005	-7238.219043	10055.0
42	1921	242292	-2022.211248	-4011.0
43	1922	195473	-29899.717765	-35531.0
44	1923	191543	-14888.224282	-18546.0
45	1924	149301	-27928.755162	-19579.0
46	1925	213929	-4537.964932	-9085.0
47	1926	130459	-14412.370462	0.0
48	1927	218382	-9555.711674	-23087.0
49	1928	243395	461.939319	-322.0
50	1929	142388	-40169.050204	-35032.0
51	1930	142941	-14360.725560	-3062.0
52	1931	131230	-21336.352189	-11158.0
53	1932	200002	-5837.302610	-8441.0

In [64]:

```
col_simu=['years','peakflow','annualmean_pred','annualmean_pred_quantile']
df_simu=pd.DataFrame(df_s,columns=col_simu)
df_simu['annualmean_pred_quantile']=g(f(df_simu['peakflow']))
df_simu.head()
```

Out[64]:

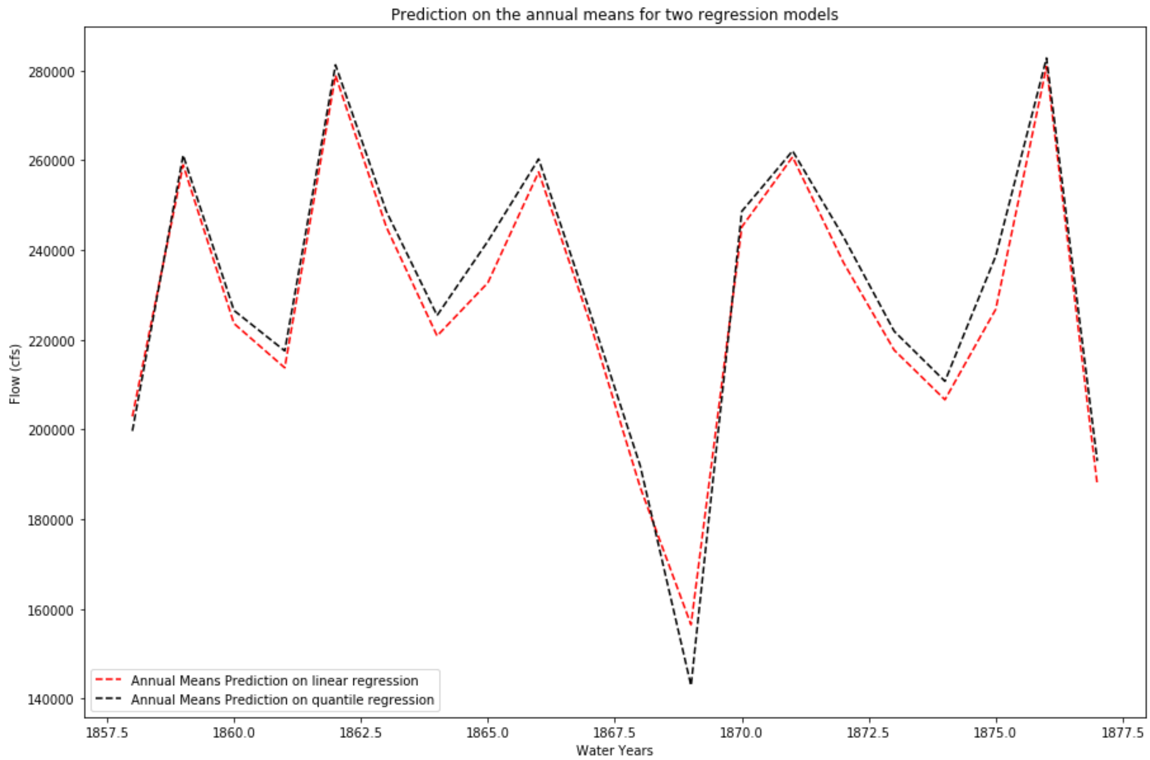
	years	peakflow	annualmean_pred	annualmean_pred_quantile
0	1858	563000.0	202879.694254	199663.272727
1	1859	847000.0	258914.945808	261082.125000
2	1860	668000.0	223596.952751	226507.727273
3	1861	618000.0	213731.591562	217500.500000
4	1862	948000.0	278842.975410	281194.147239

In [66]:

```
plt.figure(figsize=(15,10))
plt.plot(df_simu['years'],df_simu['annualmean_pred'],'r--', label='Annual Means Prediction on linear regression')
plt.plot(df_simu['years'],df_simu['annualmean_pred_quantile'],'k--', label='Annual Means Prediction on quantile regression')
plt.legend()
plt.xlabel('Water Years')
plt.ylabel('Flow (cfs)')
plt.title('Prediction on the annual means for two regression models')
```

Out[66]:

```
Text(0.5, 1.0, 'Prediction on the annual means for two regression models')
```



Answer

From the results above, as the Quantile Regression model have smaller residues seems like it did a better prediction than the linear regression model. However when we plot them out together there is not that much significant difference.