In [55]:

```python
# Import necessary libraries
import numpy as np
import pandas as pd
import statistics
import math
from matplotlib import pyplot as plt
import statsmodels.formula.api as smf
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# Read .csv file as a dataframe
df = pd.read_csv('ColoradoRiverData.csv')
df.drop('delt', inplace=True, axis=1)
df.drop('SMI', inplace=True, axis=1)
df.drop('ET', inplace=True, axis=1)
df['prevflow'] = df['Flow_cfs'].shift()
df.at[0, 'prevflow'] = 0
df.head(10)
```

Out[55]:

	Date	PPT	Tmin	Tav	Tmax	Flow_cfs	Flow?	prevflow
0	1988-03	16.09	2.8	12.3	21.8	2.68	1	0.00
1	1988-04	17.63	7.9	17.0	26.1	2.33	1	2.68
2	1988-05	63.44	14.1	21.4	28.7	4.56	1	2.33
3	1988-06	58.01	18.6	25.8	33.0	18.20	1	4.56
4	1988-07	91.98	20.5	26.6	32.7	76.10	1	18.20
5	1988-08	8.09	20.5	27.4	34.3	0.28	1	76.10
6	1988-09	111.88	16.4	23.3	30.2	28.30	1	0.28
7	1988-10	0.00	10.2	17.9	25.7	0.40	1	28.30
8	1988-11	0.00	4.8	13.4	22.1	0.02	1	0.40
9	1988-12	15.88	-1.2	6.8	14.8	1.23	1	0.02

Model Building: Part 1 | Forecasting flowrates

In this part, the goal is to make models to predict flowrates in the Colorado River, and then evaluate their performance using appropriate goodness-of-fit measures, and analyze the outcomes. Use the first 75% of the dataset for training your models and the remaining 25% for testing.

- Build 3 data models that you see appropriate for this task. (*Please note that these models should be unique and this uniqueness can be defined based on using different algorithms, inputs, or both.*)
- Assess data model quality (decide which model is best)
- Build the input data interface for using the "best" model
- Using your best model determine the estimated flowrate for the hydro-meteorological conditions in the table below:

ppt	tmax	tmean	tmin	last month flowrate
0.0	113.0	99.0	85.0	0.0
4.5	95.0	85.0	75.0	74.5
2.2	20.0	10.0	0.0	55.0
1.0	80.0	60.0	40.0	36.3
0.0	80.0	60.0	40.0	12.0

note that you may not all the values for each case, depending on your best model.

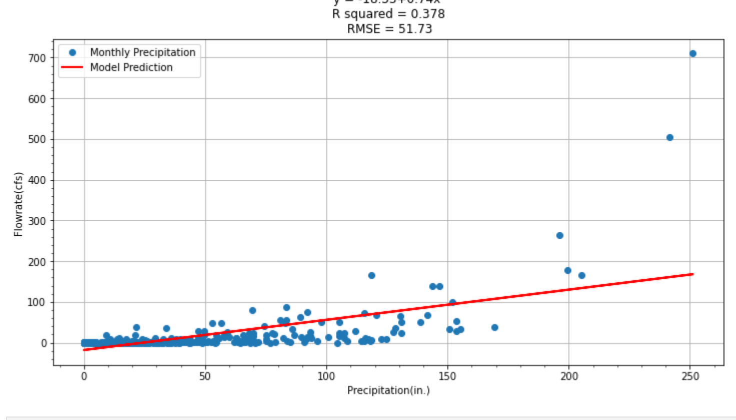
Your "Model Building: Part 1" section must include your answers for the following questions:

- What are the most important assumptions in your modeling?
- Is it benefitial to use the streamflow recorded in the previous step (a lagged streamflow value) as an input feature? why?
- Which parameter (between precipitation and temperature values) was be a better predictor for streamflow at the station of study? why?
- Is there a specific range of streamflow values that are harder to capture accurately for your data models? If yes, what range and why?

In [56]:

```python
# statsmodels package - linear

linmodel = smf.ols('Flow_cfs ~ PPT', data=df)
linmodel = linmodel.fit()
print(linmodel.summary())
# dir(model) # activate to find attributes
intercept = linmodel.params[0]
slope = linmodel.params[1]
Rsquare = linmodel.rsquared
RMSE = math.sqrt(linmodel.mse_total)

# Predict values
ppt_pred = linmodel.predict()

titleline = 'Precipitation effect on flow rate \n'  + \
            'y = ' + str(round(intercept,2)) + '+' + str(round(slope,2)) + 'x' + \
            ' \n R squared = ' + str(round(Rsquare,3)) + \
            ' \n RMSE = ' + \
            str(round(RMSE,2))
# Plot regression against actual data - What do we see?
plt.figure(figsize=(12, 6))
plt.plot(df['PPT'], df['Flow_cfs'], 'o')           # scatter plot showing actual data
plt.plot(df['PPT'], ppt_pred,color ='r', linewidth=2)   # regression line
plt.xlabel('Precipitation(in.)')
plt.ylabel('Flowrate(cfs)')
plt.legend(['Monthly Precipitation','Model Prediction'])
plt.title(titleline)

plt.minorticks_on()
plt.grid()
plt.show()
```

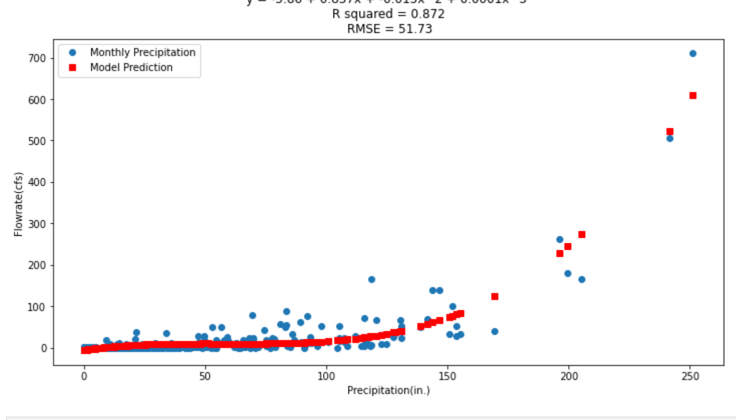
```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Flow_cfs   R-squared:                       0.378
Model:                            OLS   Adj. R-squared:                  0.376
Method:                 Least Squares   F-statistic:                     225.2
Date:                Tue, 03 May 2022   Prob (F-statistic):           4.08e-40
Time:                        23:07:27   Log-Likelihood:                -1912.2
No. Observations:                 373   AIC:                             3828.
Df Residuals:                     371   BIC:                             3836.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -18.3337      2.980     -6.153      0.000     -24.193     -12.474
PPT            0.7412      0.049     15.006      0.000       0.644       0.838
==============================================================================
Omnibus:                      566.188   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           145057.838
Skew:                           7.867   Prob(JB):                         0.00
Kurtosis:                      98.320   Cond. No.                         85.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```



In [57]:

```python
# statspackage model - cubic

df['XX']=df['PPT']**2  # add a column of X^2
df['XXX']=df['PPT']**3  # add a column of X^2

# Initialise and fit linear regression model using `statsmodels`
cubemodel = smf.ols('Flow_cfs ~ PPT+XX+XXX', data=df) # model object constructor syntax
cubemodel = cubemodel.fit()

# Predict values
cubicppt_pred = cubemodel.predict()
print(cubemodel.summary())

intercept = cubemodel.params[0]
slope1 = cubemodel.params[1]
slope2 = cubemodel.params[2]
slope3 = cubemodel.params[3]
Rsquare = cubemodel.rsquared
RMSE = math.sqrt(cubemodel.mse_total)

# Plot
titleline = 'Precipitation effect on flow rate \n'  + \
            'y = ' + str(round(intercept,2)) + ' + ' + str(round(slope1,3)) + 'x ' + '+ '+ str(round(slope2,3)) + 'x^2' + ' + '+ str(round(slope3,4
            ' \n R squared = ' + str(round(Rsquare,3)) + \
            ' \n RMSE = ' + \
            str(round(RMSE,2))

plt.figure(figsize=(12, 6))
plt.plot(df['PPT'], df['Flow_cfs'], 'o')           # scatter plot showing actual data
plt.plot(df['PPT'], cubicppt_pred, marker='s', color='r', linewidth=0 )   # regression line
plt.xlabel('Precipitation(in.)')
plt.ylabel('Flowrate(cfs)')
plt.legend(['Monthly Precipitation','Model Prediction'])
plt.title(titleline)
plt.show();
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Flow_cfs   R-squared:                       0.872
Model:                            OLS   Adj. R-squared:                  0.871
Method:                 Least Squares   F-statistic:                     834.7
Date:                Tue, 03 May 2022   Prob (F-statistic):          5.12e-164
Time:                        23:07:27   Log-Likelihood:                -1617.9
No. Observations:                 373   AIC:                             3244.
Df Residuals:                     369   BIC:                             3259.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -5.8596      1.936     -3.026      0.003      -9.667      -2.052
PPT            0.8368      0.108      7.749      0.000       0.624       1.049
XX            -0.0147      0.001    -10.546      0.000      -0.017      -0.012
XXX         8.409e-05   4.51e-06     18.643      0.000    7.52e-05     9.3e-05
==============================================================================
Omnibus:                      191.211   Durbin-Watson:                   1.910
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4658.266
Skew:                           1.606   Prob(JB):                         0.00
Kurtosis:                      20.012   Cond. No.                     3.03e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.03e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```



In [58]:

```python
# sklearn - train_test_split

feature_cols = ['PPT']
#feature_cols = ['Tmin', 'Tav', 'Tmax']
x = df[feature_cols] # Features
y = df['Flow_cfs'] # Target variable

# split x and y into training and testing sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=0)

# import the class
from sklearn.linear_model import LinearRegression

# instantiate the model (using the default parameters)
linreg = LinearRegression()

# fit the model with data
linreg.fit(x_train,y_train)

# run predictor on x_test
y_pred=linreg.predict(x_test)
```

In [60]:

```python
# use statspackage model cubic for testing

df['XX']=df['PPT']**2  # add a column of X^2
df['XXX']=df['PPT']**3  # add a column of X^2

# Initialise and fit linear regression model using `statsmodels`
cubemodel = smf.ols('Flow_cfs ~ PPT+XX+XXX', data=df) # model object constructor syntax
cubemodel = cubemodel.fit()
# Predict values
cubicppt_pred = cubemodel.predict()

intercept = cubemodel.params[0]
slope1 = cubemodel.params[1]
slope2 = cubemodel.params[2]
slope3 = cubemodel.params[3]
Rsquare = cubemodel.rsquared
RMSE = math.sqrt(cubemodel.mse_total)

# Apply equation from cubic model as function for interactive input
def cubic(i, intercept, slope1, slope2, slope3):
    ans = intercept + (slope1 * i) + (slope2 * math.pow(i, 2)) + (slope3 * math.pow(i,3))
    if(ans<0):
        ans = 0.0
    return(ans)

# Input data interface
i = input("Enter requested precipitation to test: ")
i = np.float64(i)
flowrate = cubic(i, intercept, slope1, slope2, slope3)
print("Expected flowrate is: ","{:.2f}".format(flowrate), " cfs")
```

```
Enter requested precipitation to test: 11
Expected flowrate is:  1.68  cfs
```

- What are the most important assumptions in your modeling?

- The most important assumption are to neglect the extreme values in the dataset that lead to outliers because the goal of the model is to create a pattern from the trends seen in the data, and that by eliminating the outliers, the data is more consistent with a wider range of data.

- Is it benefitial to use the streamflow recorded in the previous step (a lagged streamflow value) as an input feature? why?

- No because based on the correlation values, using the lagged streamflow value had a very miniscule correlation to the flow_cfs, being the lowest out of all of the parameters.

- Which parameter (between precipitation and temperature values) would be a better predictor for streamflow at the station of study? why?

- Using the precipitation parameter was a better predictor for streamflow due to how the precipitation has a bigger r-squared value compared to temperature, due to the r-squared value of precipitation being 0.378 out of 1, which is better than the r-squared value of temperature being 0.015.

- Is there a specific range of streamflow values that are harder to capture accurately for your data models? If yes, what range and why?

- The range of precipitation values from 0.0 to 8.0 were the hardest to capture accurately for the data models, likely due to the majority of those streamflow cases having a flowrate of 0 and the outliers had flowrates.

Model Building: Part 2 | Forecasting flow states

In this part, the goal is to make models to predict whether the Colorado River's flowstate is in the "Flow" or the "No-Flow" state. Then, evaluate their performance using appropriate goodness-of-fit measures, and analyze the outcomes. Use the first 75% of the dataset for training your models and the remaining 25% for testing.

- Add a column to the dataframe for the flow state: It should be 0 when the flowrate is equal to 0 and 1 when the flowrate is non-zero.
- Build 3 data models that you see appropriate for predicting the flow state. (*Please note that these models should be unique and this uniqueness can be defined based on using different algorithms, inputs, or both.*)
- Assess data model quality (decide which model is best)
- Build the input data interface for using the "best" model
- Using your best model determine the estimated flow state for the hydro-meteorological conditions in the table below: |ppt|tmax|tmean|tmin| |:---|:---|:---|:---| |0.0|113.0|99.0|85.0| |4.5|95.0|85.0|75.0| |2.2|20.0|10.0|0.0| |1.0|80.0|60.0|40.0| |0.0|80.0|60.0|40.0| *note that you may not all the values for each case, depending on your best model.*

In [61]:

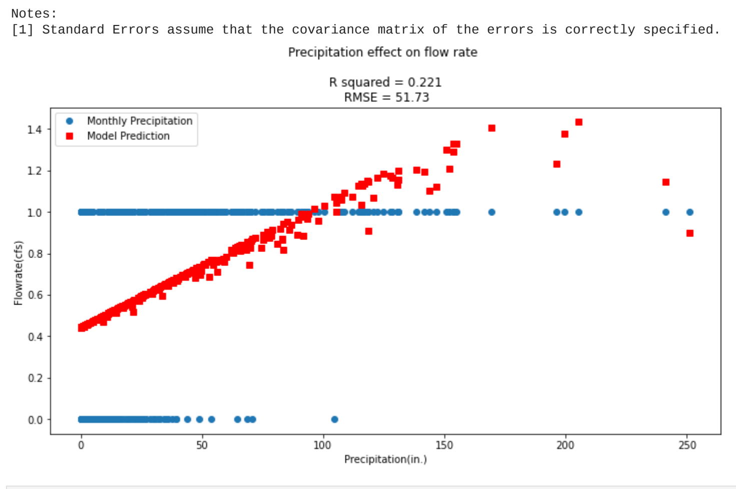
```python
# check the actual ratio of values in the dataset
print ('No flowrate % ',round(df['Flow?'].value_counts()[0]/len(df)*100,2))
print ('Fraud %     ',round(df['Flow?'].value_counts()[1]/len(df)*100,2))
```

```
No flowrate %  32.17
Fraud %     67.83
```

In [62]:

```python
# statspackage model - multi linear regression

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('Q("Flow?") ~ Flow_cfs + PPT', data=df) # model object constructor syntax
model = model.fit()
print(model.summary())

# Predict values
y_pred = model.predict()
Rsquared = model.rsquared
rmse = math.sqrt(model.mse_total)

# Plot
titleline = 'Precipitation effect on flow rate \n'  + \
            ' \n R squared = ' + str(round(Rsquared,3)) + \
            ' \n RMSE = ' + \
            str(round(RMSE,2))

plt.figure(figsize=(12, 6))
plt.plot(df['PPT'], df['Flow?'], 'o')           # scatter plot showing actual data
plt.plot(df['PPT'], y_pred, marker='s', color='r', linewidth=0 )   # regression line
plt.xlabel('Precipitation(in.)')
plt.ylabel('Flowrate(cfs)')
plt.legend(['Monthly Precipitation','Model Prediction'])
plt.title(titleline)
plt.show();
```

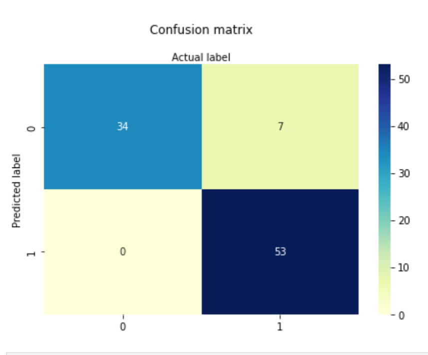
```
                            OLS Regression Results
==============================================================================
Dep. Variable:             Q("Flow?")   R-squared:                       0.221
Model:                            OLS   Adj. R-squared:                  0.217
Method:                 Least Squares   F-statistic:                     52.56
Date:                Tue, 03 May 2022   Prob (F-statistic):           8.12e-21
Time:                        23:07:33   Log-Likelihood:                -198.72
No. Observations:                 373   AIC:                             403.4
Df Residuals:                     370   BIC:                             415.2
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.4416      0.032     13.935      0.000       0.379       0.504
Flow_cfs      -0.0015      0.001     -2.832      0.005      -0.003      -0.000
PPT            0.0060      0.001      9.513      0.000       0.005       0.007
==============================================================================
Omnibus:                     1040.888   Durbin-Watson:                   1.293
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               35.731
Skew:                          -0.336   Prob(JB):                     1.74e-08
Kurtosis:                       1.641   Cond. No.                         107.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```



In [63]:

```python
# assign dataset into predictors and target
selectedcols = ['Flow_cfs']
y = df['Flow?']
x = df[selectedcols]

# split x and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)

# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
# logreg = LogisticRegression()
logreg = LogisticRegression(solver='lbfgs', max_iter=10000)
# fit the model with data  -TRAIN the model
logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)

# assess fitted results based on confusion matrix
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_pred, y_test)
print(cnf_matrix)
tpos = cnf_matrix[0][0]
fneg = cnf_matrix[1][1]
fpos = cnf_matrix[0][1]
tneg = cnf_matrix[1][0]
print("True Positive Cases are",tpos) #How many non-fraud cases were identified as non-fraud cases - GOOD
print("True Negative Cases are",tneg) #How many Fraud cases were identified as Fraud cases - GOOD
print("False Positive Cases are",fpos) #How many Fraud cases were identified as non-fraud cases - BAD | (type 1 error)
print("False Negative Cases are",fneg) #How many non-fraud cases were identified as Fraud cases - BAD | (type 2 error)

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# create heatmap
import seaborn as sns
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("F1-score:",metrics.f1_score(y_test, y_pred))
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
[[34  7]
 [ 0 53]]
True Positive Cases are 34
True Negative Cases are 0
False Positive Cases are 7
False Negative Cases are 53
Accuracy: 0.925531914893617
Precision: 1.0
Recall: 0.8833333333333333
F1-score: 0.9380530973451328
              precision    recall  f1-score   support

           0       0.83      1.00      0.91        34
           1       1.00      0.88      0.94        60

    accuracy                           0.93        94
   macro avg       0.91      0.94      0.92        94
weighted avg       0.94      0.93      0.93        94
```



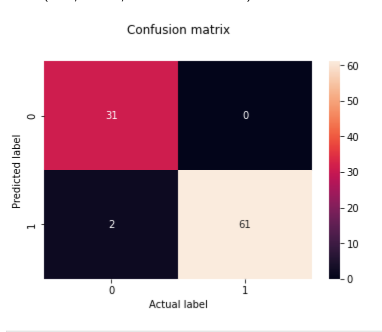
In [64]:

```python
# sklearn - K-Nearest Neighbors model

# split data into predictors and target
selectedcols = ['Flow_cfs']
X = df[selectedcols]
y = df['Flow?']

# split data into training and testing with 25:85 split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

#import KNeighborsClassifier class
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=4)
classifier.fit(X_train, y_train)

# make predictions on test data
y_pred = classifier.predict(X_test)

# assess fitted results using confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# apply to confusion matrix
cm = pd.DataFrame(confusion_matrix(y_test, y_pred))
sns.heatmap(cm, annot=True)
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')
```

```
[[31  0]
 [ 2 61]]
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        31
           1       1.00      0.97      0.98        63

    accuracy                           0.98        94
   macro avg       0.97      0.98      0.98        94
weighted avg       0.98      0.98      0.98        94
```

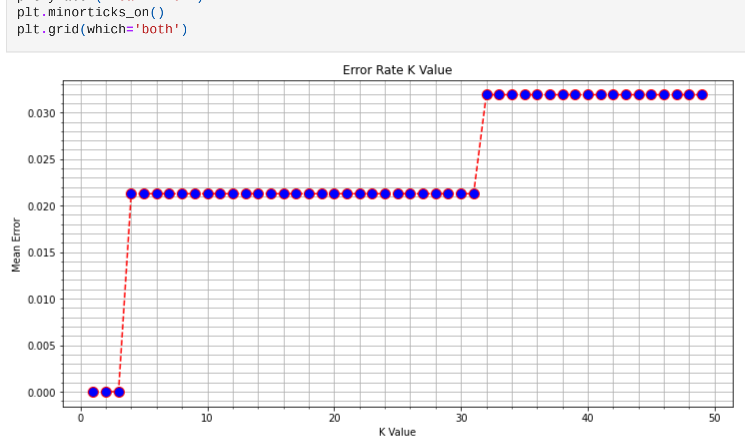
Out[64]:

```
Text(0.5, 15.0, 'Actual label')
```



In [65]:

```python
# finding the best possible K-value

error = []

# Calculating error for K values between 1 and 50
# In each iteration the mean error for predicted values of test set is calculated and
# the result is appended to the error list.
for i in range(1, 50):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

# plot the error values against K values:
plt.figure(figsize=(12, 6))
plt.plot(range(1, 50), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.minorticks_on()
plt.grid(which='both')
```



In [69]:

```python
# Input data interface
ppt = input("Enter requested precipitation to test: ")
ppt = np.float64(ppt)
flowrate = cubic(ppt, intercept, slope1, slope2, slope3)
print("Expected flowrate is: ","{:.2f}".format(flowrate), " cfs")
print(classifier.predict([[flowrate]]))
```

```
Enter requested precipitation to test: 200
Expected flowrate is:  246.72  cfs
[1]
```

- What are the most important assumptions in your modeling?

- The most important assumption was to assume any flowrate meant that Flow was true, since the very existence of a flowrate would mean that the river is flowing.

- Is it more difficult for the model to capture one flow state more than the other? If yes, which one? why?

- It was not difficult for the KNN model to capture a flowstate more than the other since it used the flow value as a parameter.

- Which parameter (between precipitation and temperature values) was be a better predictor for flow state at the station of study? why?

- The best predictor for flow state was precipitation because it relied on the model created for Model 1 as a basis on flow state outcome, and since the flow state had the most correlation with precipitation, it worked better than temperature.

- What can be some applications of this kind of streamflow state modeling?

- The streamflow state modelling can be used for agriculture and public safety in order to determine how much water would affect the area of study.