In [1]:

```python
# Standard imports
import numpy as np
import pandas as pd

# matplot
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import matplotlib.ticker as ticker
%matplotlib inline

# Scikit-learn imports
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

In [2]:

```python
# Load the data
df = pd.read_csv("data_sets/titanic.csv")
df.describe()
```

Out[2]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [21]:

```python
# Preprocession and train test split

# Build a wrangle function that processes the data and
# split the data into X and y dataframes
def titanic_tripulation(df):
    # Make a copy of the data
    df = df.copy()

    # Drop Name because it doesn't help predict
    # Drop Fare because it correlates heavily with Pclass
    drop_columns = ['Name', 'Pclass']
    df = df.drop(drop_columns, axis=1)


    # One hot encode the sex column to split it into a Male and Female colum
    dummies = pd.get_dummies(data=df, prefix=['Sex'])
    concatenation = pd.concat([df, dummies], axis='columns')

    # Separate the target and the rest of the features
    concatenation = concatenation.drop(['Survived', 'Sex'], axis=1)
    concatenation = concatenation.loc[:,~concatenation.columns.duplicated()]
    features = concatenation.columns
    target = 'Survived'

    # Create X and y dataframes
    X = concatenation[features]
    y = df[target]

    return X, y

X,y = titanic_tripulation(df)
```

In [4]:

```python
# Perform the train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

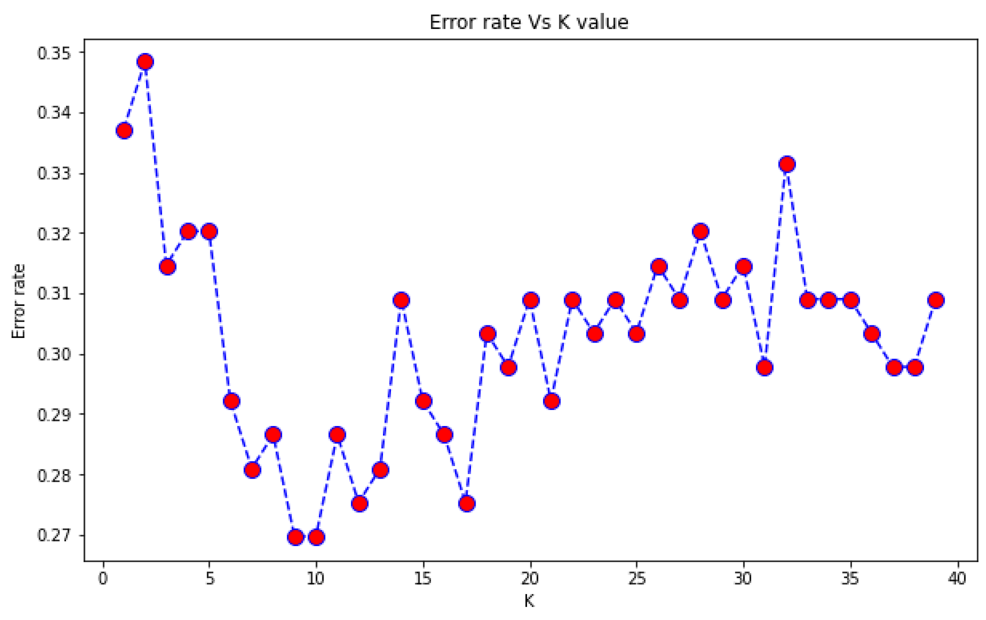
In [5]:

```python
# Improve the model and find out the optimal K value
error_rate = []

"""
Where 40 is the top for K, selected for us.
"""
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i).fit(X_train, y_train)
    predict_y = knn.predict(X_test)
    error_rate.append(np.mean(predict_y != y_test))


'''
plot error rate Vs K value
'''
plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed',
         marker='o', markerfacecolor='red', markersize = 10)
plt.title('Error rate Vs K value')
plt.xlabel('K')
plt.ylabel('Error rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
```

```
Minimum error:- 0.2696629213483146 at K = 8
```



In [6]:

```python
# Create KNN classifier object and determine the number of neighbors (n_neighbors) parameter
model = KNeighborsClassifier(n_neighbors=8)

# Fit the model to the data
model.fit(X_train, y_train)

# Make our predictions on the X_test set
predict_y = model.predict(X_test)

print("Acuaricy of model with sklearn at K=8 is:",metrics.accuracy_score(y_test, predict_y))
```

```
Acuaricy of model at K=8 is: 0.7134831460674157
```

In [7]:

```python
# KNN algorithm by hand

# Helper functions
def _sqrt(x):
    return x**.5


# calculate Euclidean distance
def euclideanDistance(row1, row2):
    """
    Finds the Euclidean distance between two rows. I.e. Get the diference between two features,
    apply the square and attach that value to a general distance value and
    at the end apply the square root to the accumulated distance.
    """
    # Each time we call the function we are setting the "distance" variable
    # to 0.0
    distance = 0.0

    for i in range(len(row1) -1):
        # Add the distance between each feature in the two rows
        distance += (row1[i] - row2[i])**2


    # Return the square root of the distance between the two rows
    return _sqrt(distance)
```

In [19]:

```python
class KNNHomebrew:

    def __init__(self, k=3):
        self.k = k # number of neighbors

    def model_fit(self, X, y):
        """
        Fits the training data to the model.

        KNN simply memorizes the data. So fitting the data is simple creating class
        variables for the X_train and y_train.
        """
        self.X_train = X
        self.y_train = y

    def model_predict_all(self, X_text):

        '''
        usign model_predict_all we can make prediction on an array of new data
        '''
        self.predictions = []
        for i in range(len(X_test)):
            x = self.model_predict(X_test.iloc[i])
            self.predictions.append(x)

        return np.array(self.predictions)

    def model_predict(self, row):
        """
        This method lets us make a prediction on one new row of data.
        """

        # First need to find Euclidean distance between the incoming row
        # and all the other rows that belong to the train set
        all_distances = {i: euclideanDistance(row, self.X_train.iloc[i]) for i in range(len(self.X_train))}

        # Get the k closest position of incoming row
        sort_orders = [k for k, v in sorted(all_distances.items(), key=lambda item: item[1])][:self.k]

        # Make the prediction, taking the output of the k closest positions (rows).
        # Get the value "Survived" and attach
        output_values = []
        for k in sort_orders:
            output_values.append(self.y_train.iloc[k])

        # Take as predict the major number of occurrences.
        # I.e. if major numer of occurrences = 1 -> prediction = 1 (survive)
        prediction = max(set(output_values), key=output_values.count)

        return prediction

    def model_accuracy(self, y_test, predict_y):
        '''
        Calculate the accuracy score of the new data
        '''
        return sum(predict_y == y_test) / len(y_test)
```

In [20]:

```python
# Create KNN instance and indicate the number of neighbors to evaluate
Knn = KNNHomebrew(8)

# Fit the model to the train data
Knn.model_fit(X_train, y_train)

# Make predictions on the test set
predict_y = Knn.model_predict_all(X_test)

print("manual model accuracy: {}".format(Knn.model_accuracy(y_test, predict_y)))

print("Acuaricy of model at K=8 is:",metrics.accuracy_score(y_test, predict_y))
```

```
manual model accuracy: 0.6910112359550562
Acuaricy of model at K=8 is: 0.6910112359550562
```

In [ ]: