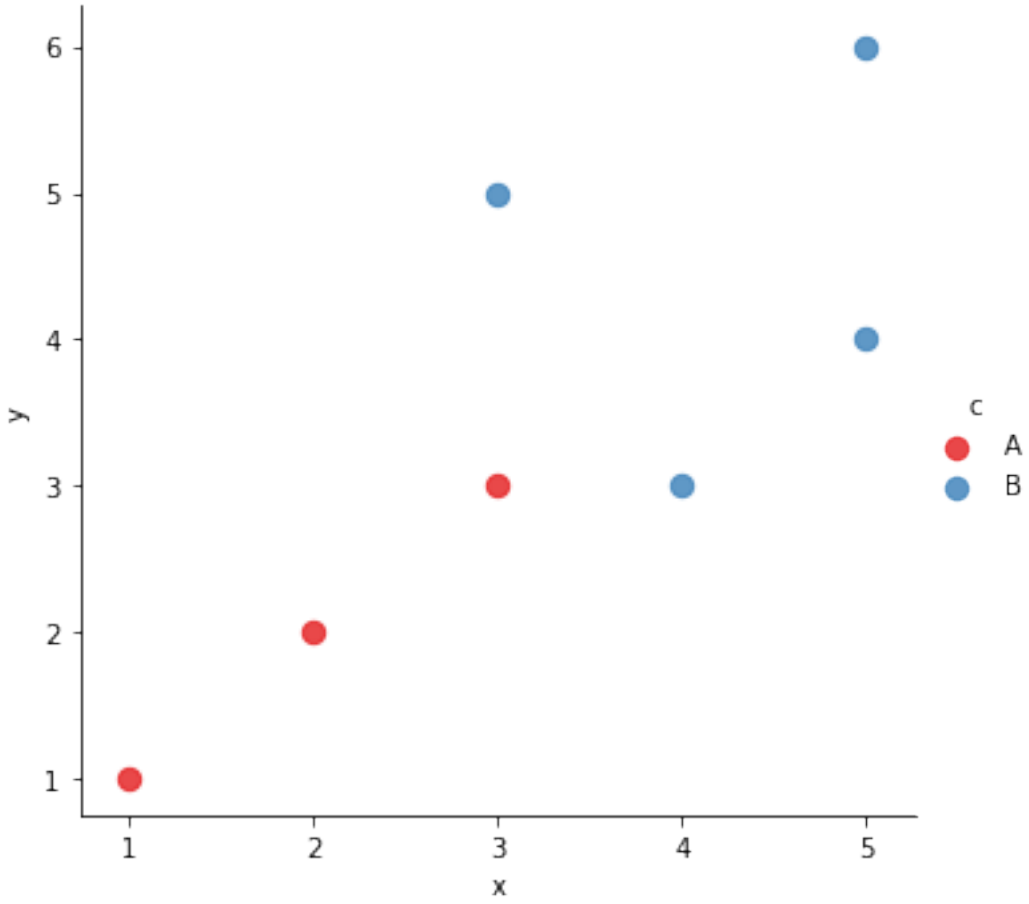
KNN

October 22, 2019

In [1]:
```
import pandas as pd
import numpy as np
import operator
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
data=pd.read_csv('knn.csv')
sns.lmplot('x', 'y', data=data,
           hue='c', palette='Set1',
           fit_reg=False, scatter_kws={"s":70})
plt.show()
```



In [14]:
```
def euclidean_distance(pt1, pt2, dimension):
    distance=0
    for x in range(dimension):
        distance+=np.square(pt1[x]-pt2[x])
    return np.sqrt(distance)

# KNN model
def knn(training_points, test_point, k):
    distances={}
    dimension = test_point.shape[1]
    for x in range(len(training_points)):
        dist = euclidean_distance(test_point, training_points.iloc[x], dimension)
        distances[x] = dist[0]

    sorted_d=sorted(distances.items(), key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(sorted_d[x][0])

    # for each neighbor found, find out its class
    class_counter = {}
    for x in range(len(neighbors)):
        # find out the class for that particular point
        cls = training_points.iloc[neighbors[x]][-1]
        if cls in class_counter:
            class_counter[cls]+=1
        else:
            class_counter[cls]=1
    # sort the class_counter in descending order
    sorted_counter = sorted(class_counter.items(),
                            key=operator.itemgetter(1),
                            reverse=True)
    return(sorted_counter[0][0], neighbors)
```

In [15]:
```
# test point
test_set = [[3,3.9]]
test = pd.DataFrame(test_set)
cls,neighbors = knn(data, test, 5)
print("Predicted Class: " + cls)
```

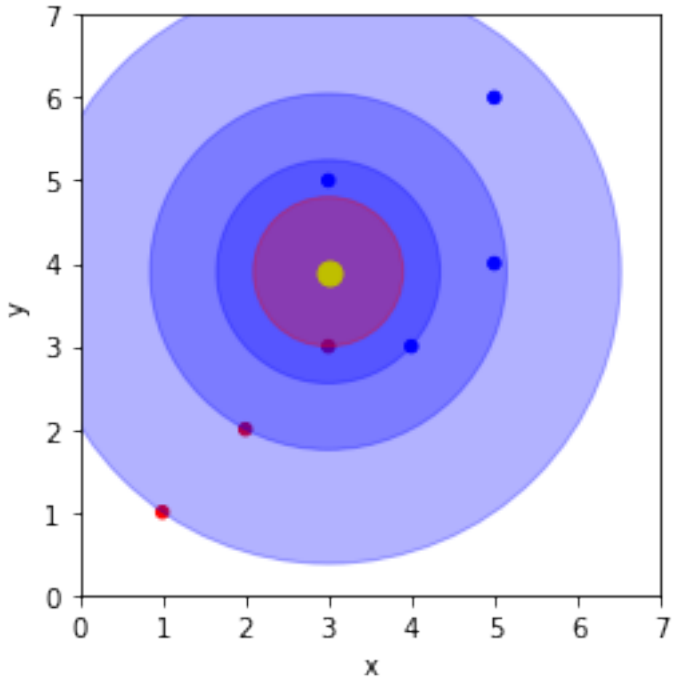
```
Predicted Class: B
```

In [19]:
```
# generate the color map for the scatter plot
# if column 'c' is A, then use Red, else use Blue
colors = ['r' if i == 'A' else 'b' for i in data['c']]
```

```
        ax = data.plot(kind='scatter', x='x', y='y', c = colors)
        plt.xlim(0,7)
        plt.ylim(0,7)

        # plot the best point
        plt.plot(test_set[0][0], test_set[0][1], "yo", markersize='9')

        for k in range(7,0,-2):
            cls,neighbors = knn(data, test, k)
            print("k=", k)
            print("Class", cls)
            print("Neighbors")
            print(data.iloc[neighbors])

            furthest_point = data.iloc[neighbors].tail(1)

            # draw a circle connecting the test point
            # and the furthest point
            radius = euclidean_distance(test, furthest_point.iloc[0], 2)

            # display the circle in red if classfification is A, else display circle in blue
            c = 'r' if cls=='A' else 'b'
            circle = plt.Circle((test_set[0][0], test_set[0][1]),
                                radius, color=c, alpha=0.3)
            ax.add_patch(circle)

        plt.gca().set_aspect('equal', adjustable='box')
        plt.show()
```

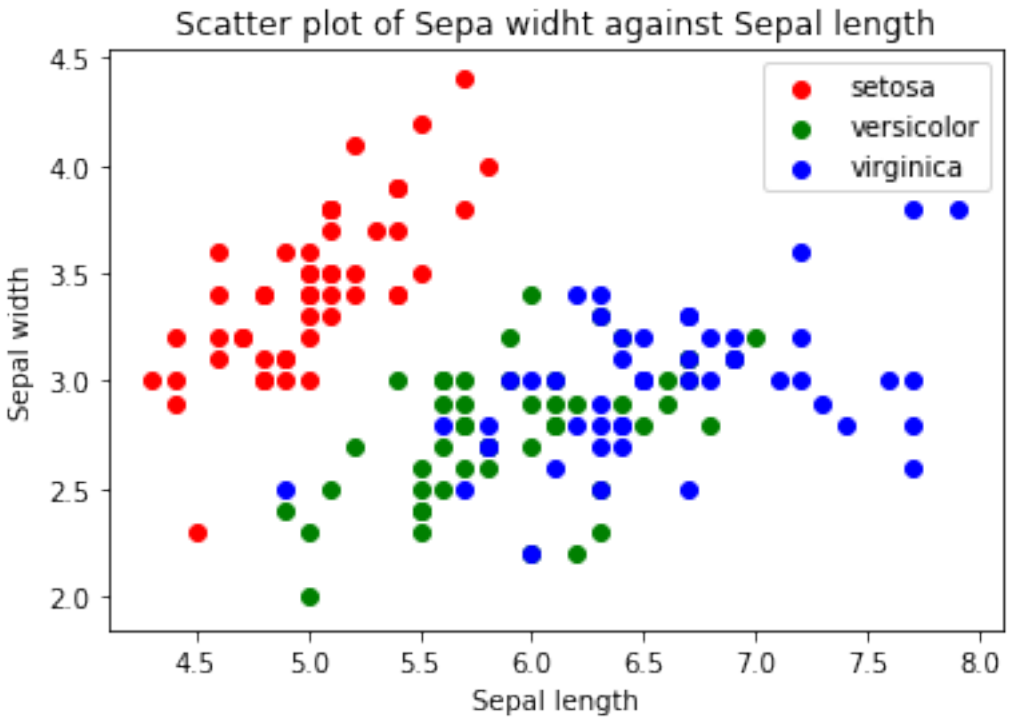
```
k= 7
Class B
Neighbors
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
6  5  4  B
1  2  2  A
5  5  6  B
0  1  1  A
k= 5
Class B
Neighbors
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
6  5  4  B
```

```
1  2  2  A
k= 3
Class B
Neighbors
   x  y  c
3  3  3  A
4  3  5  B
2  4  3  B
k= 1
Class A
Neighbors
   x  y  c
3  3  3  A
```




```
In [38]: #Scikit-Learn's KNeighborsClassifier Class for KNN
         %matplotlib inline
         import pandas as pd
         import numpy as np
         import matplotlib.patches as mpatches
         from sklearn import svm, datasets
         import matplotlib.pyplot as plt

         iris = datasets.load_iris()
```

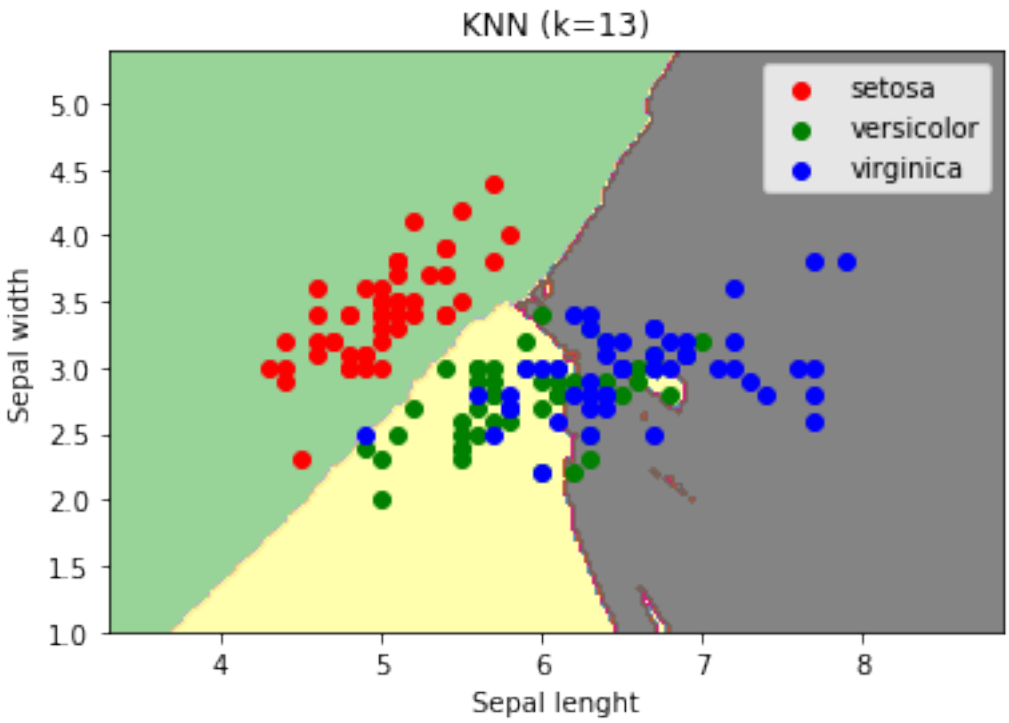
```
X = iris.data[:, :2]
y = iris.target

colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0,1,2], iris.target_names):
    plt.scatter(X[y==i,0], X[y==i,1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.title("Scatter plot of Sepa widht against Sepal length")
plt.show()
```



In [40]:
```
from sklearn.neighbors import KNeighborsClassifier
k=13
# instantiate learning model
knn = KNeighborsClassifier(n_neighbors=k)
# fitting the mdoel
knn.fit(X, y)
# min and max for the first feature
x_min, x_max = X[:, 0].min()-1, X[:,0].max()+1
# min and amx for the second feature
y_min, y_max = X[:, 1].min()-1, X[:,1].max()+1
```

```
        # step size in the mesh
        h = (x_max/x_min)/100
        # make predictions for each of the points in xx,yy
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))
        Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])

        # draw the result using a color plot
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)

        # plot the training points
        colors = ['red', 'green', 'blue']
        for color, i, target in zip(colors, [0,1,2], iris.target_names):
            plt.scatter(X[y==i, 0], X[y==i, 1], color=color, label=target)

        plt.xlabel("Sepal lenght")
        plt.ylabel("Sepal width")
        plt.title(f'KNN (k={k})')
        plt.legend(loc='best', shadow=False, scatterpoints=1)

        predictions = knn.predict(X)

        # classifications based on predictions
        print(np.unique(predictions, return_counts=True))
```

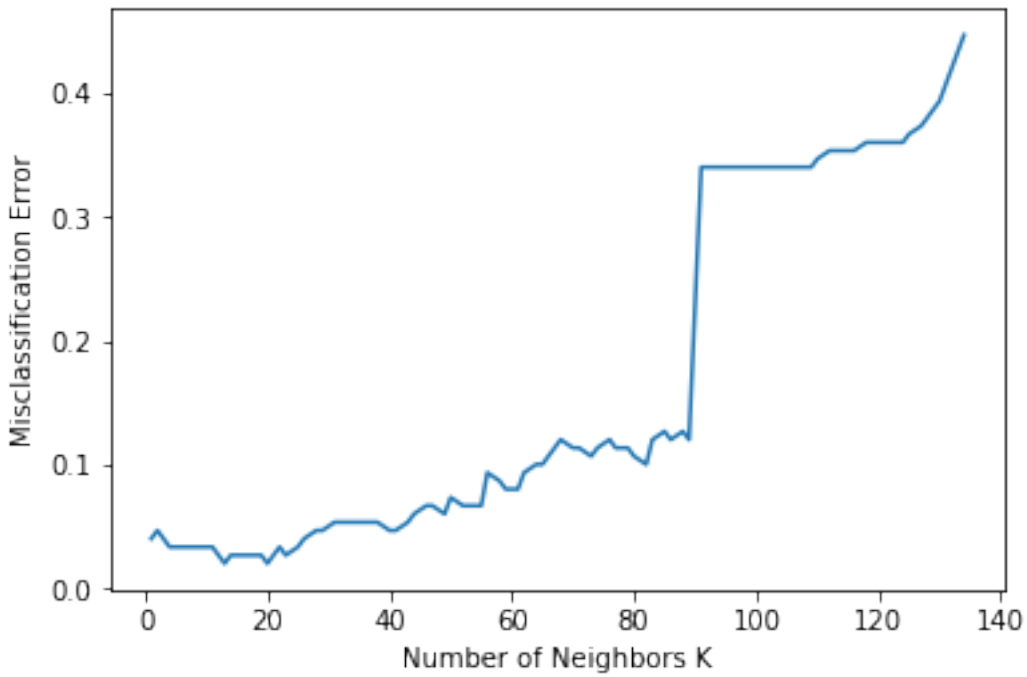
```
(array([0, 1, 2]), array([51, 46, 53], dtype=int64))
```



In [35]:
```
# Parameter tuning K
from sklearn.model_selection import cross_val_score

# holds the cv(cross validation) scores
cv_scores = []

# use all features
X = iris.data[:, :4]
y = iris.target

# number of flods
folds = 10
# createing odd list of K for KNN
ks = list(range(1,int(len(X)*((folds-1)/folds))))

# remove all multiples of 3
ks = [k for k in ks if k % 3 != 0]

# perform k-fold cross validation
for k in ks:
    knn = KNeighborsClassifier(n_neighbors=k)

    # perorms cross-validation and return the average accuracy
    scores = cross_val_score(knn, X, y, cv=folds, scoring='accuracy')
```

```
            mean = scores.mean()
            cv_scores.append(mean)
            #print(k, mean)

        # calculate misclassification error for each k
        MSE = [1-x for x in cv_scores]
        # determining best k (min. MSE)
        optimal_k = ks[MSE.index(min(MSE))]
        print(f"The optimal number of neighbors is {optimal_k}")

        # plot misclassification error vs k
        plt.plot(ks, MSE)
        plt.xlabel('Number of Neighbors K')
        plt.ylabel('Misclassification Error')
        plt.show()
```

```
The optimal number of neighbors is 13
```



In [ ]: