In [90]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [91]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [92]:

```
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
```

In [93]:

```
print(X_train)
```

```
 [    43 129000]
 [    59  76000]
 [    18  44000]
 [    36 118000]
 [    42  90000]
 [    47  30000]
 [    26  43000]
 [    40  78000]
 [    46  59000]
 [    59  42000]
 [    46  74000]
 [    35  91000]
 [    28  59000]
 [    40  57000]
 [    59 143000]
 [    57  26000]
 [    52  38000]
 [    47 113000]
 [    53 143000]
 [    35  27000]
```

In [94]:

```
X_train.shape
```

Out[94]:

```
(300, 2)
```

In [95]:

```
sum_age, sum_sal=0,0
n=X_train.shape[0]

for data in X_train:
    sum_age+=data[0]
    sum_sal+=data[1]

#print(sum_age,sum_sal)

mean_age=sum_age/n
mean_sal=sum_sal/n

print("Mean: ",mean_age,mean_sal)

sd_age,sd_sal=0,0

for data in X_train:
    sd_age+= (abs(data[0]-mean_age))**2
    sd_sal+= (abs(data[1]-mean_sal))**2

sd_age=(sd_age/n)**(1/2)
sd_sal=(sd_sal/n)**(1/2)

print("Standard deviation: ",sd_age,sd_sal)
```

```
Mean:  38.126666666666665 69583.33333333333
Standard deviation:  10.097720314781727 34490.91265182113
```

In [96]:

```
X_train_std=[]

for val in X_train:
    X_train_std.append([(val[0]-mean_age)/sd_age, (val[1]-mean_sal)/sd_sal])

X_train_std=np.array(X_train_std)

print(X_train_std)
```

```
 [-0.30964085 -0.74174127]
 [-0.11157634  0.1570462 ]
 [-0.90383437 -0.65476184]
 [-0.70576986 -0.04590581]
 [ 0.38358493 -0.45180983]
 [-0.80480212  1.89663484]
 [ 1.37390747  1.28777882]
 [ 1.17584296 -0.97368642]
 [ 1.77003648  1.83864855]
 [-0.90383437 -0.24885782]
 [-0.80480212  0.56295021]
 [-1.20093113 -1.5535493 ]
 [-0.50770535 -1.11865214]
 [ 0.28455268  0.07006676]
 [-0.21060859 -1.06066585]
 [ 1.67100423  1.6067034 ]
 [ 0.97777845  1.78066227]
 [ 0.28455268  0.04107362]
 [-0.80480212 -0.21986468]
 [-0.11157634  0.07006676]
```

In [97]:

```
X_test_std=[]

for val in X_test:
    X_test_std.append([(val[0]-mean_age)/sd_age, (val[1]-mean_sal)/sd_sal])

X_test_std=np.array(X_test_std)
print(X_test_std)
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
 [ 1.07681071  2.07059371]
 [-1.00286662  0.50496393]
 [-0.90383437  0.30201192]
 [-0.11157634 -0.21986468]
 [-0.60673761  0.47597078]
 [-1.6960924   0.53395707]
 [-0.11157634  0.27301877]
 [ 1.86906873 -0.27785096]
 [-0.11157634 -0.48080297]
 [-1.39899564 -0.33583725]
 [-1.99318916 -0.50979612]
 [-1.59706014  0.33100506]
 [-0.4086731  -0.77073441]
 [-0.70576986 -1.03167271]
 [ 1.07681071 -0.97368642]
 [-1.10189888  0.53395707]
 [ 0.28455268 -0.50979612]
 [-1.10189888  0.41798449]
 [-0.30964085 -1.43757673]
 [ 0.48261718  1.22979253]
 [-1.10189888 -0.33583725]
 [-0.11157634  0.30201192]
 [ 1.37390747  0.59194336]
 [-1.20093113 -1.14764529]
 [ 1.07681071  0.47597078]
 [ 1.86906873  1.51972397]
 [-0.4086731  -1.29261101]
 [-0.30964085 -0.3648304 ]
 [-0.4086731   1.31677196]
 [ 2.06713324  0.53395707]
 [ 0.68068169 -1.089659  ]
 [-0.90383437  0.38899135]
 [-1.20093113  0.30201192]
 [ 1.07681071 -1.20563157]
 [-1.49802789 -1.43757673]
 [-0.60673761 -1.49556302]
```

```
 [ 2.1661655  -0.79972756]
 [-1.89415691  0.18603934]
 [-0.21060859  0.85288166]
 [-1.89415691 -1.26361786]
 [ 2.1661655   0.38899135]
 [-1.39899564  0.56295021]
 [-1.10189888 -0.33583725]
 [ 0.18552042 -0.65476184]
 [ 0.38358493  0.01208048]
 [-0.60673761  2.331532  ]
 [-0.30964085  0.21503249]
 [-1.59706014 -0.19087153]
 [ 0.68068169 -1.37959044]
 [-1.10189888  0.56295021]
 [-1.99318916  0.35999821]
 [ 0.38358493  0.27301877]
 [ 0.18552042 -0.27785096]
 [ 1.47293972 -1.03167271]
 [ 0.8787462   1.08482681]
 [ 1.96810099  2.15757314]
 [ 2.06713324  0.38899135]
 [-1.39899564 -0.42281668]
 [-1.20093113 -1.00267957]
 [ 1.96810099 -0.91570013]
 [ 0.38358493  0.30201192]
 [ 0.18552042  0.1570462 ]
 [ 2.06713324  1.75166912]
 [ 0.77971394 -0.8287207 ]
 [ 0.28455268 -0.27785096]
 [ 0.38358493 -0.16187839]
 [-0.11157634  2.21555943]
 [-1.49802789 -0.62576869]
 [-1.29996338 -1.06066585]
 [-1.39899564  0.41798449]
 [-1.10189888  0.76590222]
 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

In [98]:

```
def KNN(x1,x2,y,a,b,k,q):
    x1=np.array(x1)
    x2=np.array(x2)
    n=len(x1)


    diff=list( ((abs(x1-a))**q + (abs(x2-b))**q )**(1/q) )
    #print(diff)

    l=diff.copy()
    l.sort()
    l=l[:k]

    nearestNeighbor=[]
    for i in l:
        index=diff.index(i)
        nearestNeighbor.append(y[index])
    #print(nearestNeighbor)

    s=set(nearestNeighbor)
    maxCount=0
    res=''
    for i in s:
        c=nearestNeighbor.count(i)
        if c>maxCount:
            maxCount=c
            res=i
        #print(c,maxCount)
    return res
```

In [99]:

```
res=0
a,b=map(int,input("Enter query: ").split())
k=int(input("Enter value of k: "))
q=int(input("Enter value of q: "))

a=(a-mean_age)/sd_age
b=(b-mean_sal)/sd_sal
res=KNN(X_train_std[:,0],X_train_std[:,1],y_train,a,b,k,q)

print("Predicted value using Minkowski distance: ",res)
```

```
Enter query: 30 87000
Enter value of k: 5
Enter value of q: 2
Predicted value using Minkowski distance:  0
```

In [100]:

```
res,x=0,0
y_pred=[]


for val in X_test_std:
    res=KNN(X_train_std[:,0],X_train_std[:,1],y_train,val[0],val[1],5,2)
    y_pred.append([res, y_test[x]])
    x+=1

y_pred=np.array(y_pred)
print(y_pred)
```

```
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
```

```
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [1 1]
 [1 1]
 [1 0]
 [0 0]
 [0 0]
 [1 1]
 [0 1]
 [0 0]
 [1 1]
 [1 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [0 0]
 [1 1]
 [1 1]
 [1 1]]
```

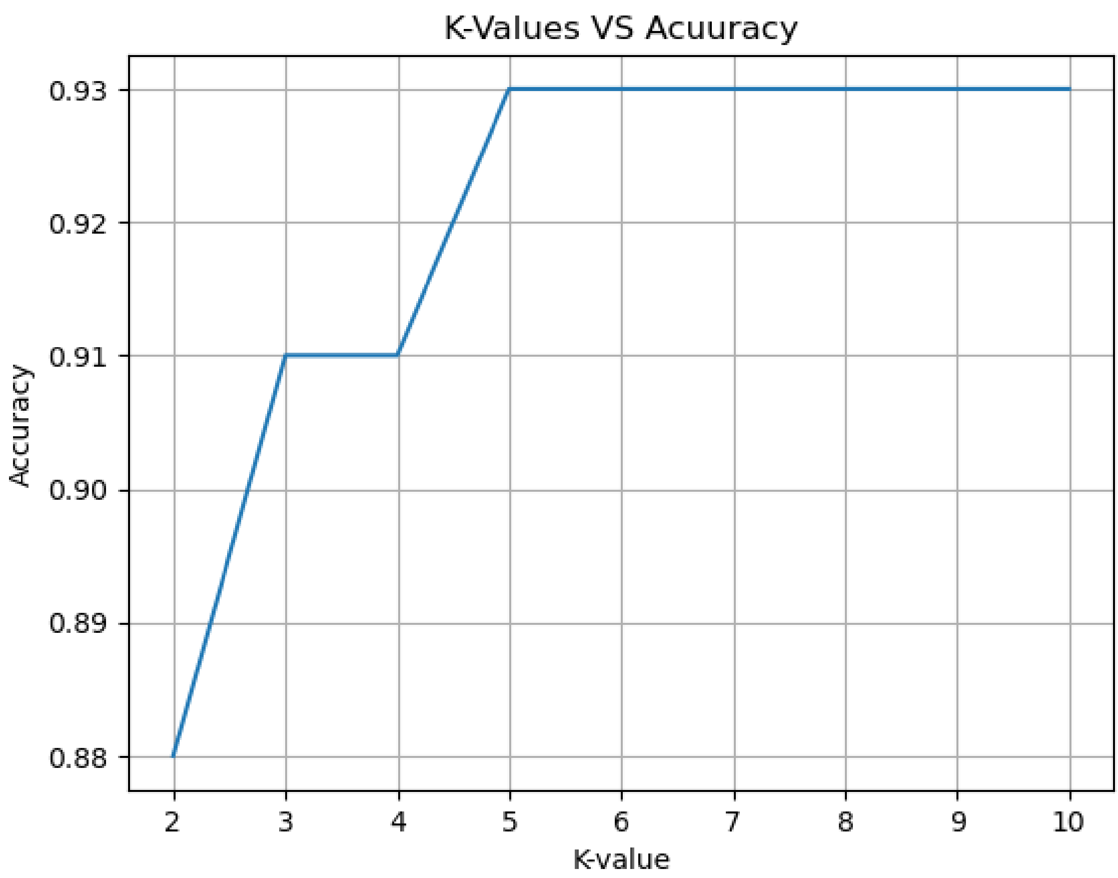
In [101]:

```
def confusion_matrix(y_pred):
    cm=np.zeros((2,2))
    for i in range(len(y_pred)): #the confusion matrix is for 2 classes: 1,0
        if (y_pred[i][0])==0 and (y_test[i])==0:
            cm[0,0]+=1
        elif (y_pred[i][0])==1 and (y_test[i])==0:
            cm[0,1]+=1
        elif (y_pred[i][0])==0 and (y_test[i])==1:
            cm[1,0]+=1
        elif (y_pred[i][0])==1 and (y_test[i])==1:
            cm[1,1]+=1
    return cm

cm=confusion_matrix(y_pred)
print(cm,"\n")

accuracy=(cm[0,0]+cm[1,1])/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])
print("Accuracy: ",accuracy)
```

```
[[64.  4.]
 [ 3. 29.]]

Accuracy:  0.93
```

In [102]:

```
res,accuracy=0,0
acc=[]
cm=cm=np.zeros((2,2))
k_values= []


for i in range(2,11):
    k_values.append(i)
    y_pred_i=[]
    x=0
    for val in X_test_std:
        res=KNN(X_train_std[:,0],X_train_std[:,1],y_train,val[0],val[1],i,2)
        y_pred_i.append([res, y_test[x]])
        x+=1
    cm=confusion_matrix(y_pred_i)
    accuracy=(cm[0,0]+cm[1,1])/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])
    acc.append(accuracy)

#print(acc)

plt.plot(k_values, acc)

plt.xlabel('K-value')
plt.ylabel('Accuracy')
plt.title('K-Values VS Acuuracy')

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



Accuracy is highest for k values from 5 to 10

In [ ]:

```

```