2. target 1.0 i.e. 'versicolor' flower has avergae value for mean sepal length, mean petal length and petal width compared to other targets in the dataset and smallest sepal width among all targets.
3. target 2.0 i.e. 'virginica' flower has largest value for mean sepal length, mean petal length and mean petal width.

In [12]:
```
#making a copy of dataset to train the model
iris_df2 = iris_df.copy(deep =True)
```

In [13]:
```
#importing useful libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [14]:
```
#preprocessing data
X = iris_df2.drop(['target'], axis =1)
y = iris_df2.target
print ("shape of X is :", X.shape)
print("shape of y is :", y.shape)
```

```
shape of X is : (150, 4)
shape of y is : (150,)
```

KNN Classifier

In [15]:
```
#splitting data into training, validation and test sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(X, y, test_size = 0.
X_train_k, X_valid_k, y_train_k, y_valid_k = train_test_split(X_train_k, y_train_
print( "X_train_k.shape is :", X_train_k.shape)
print( "X_valid_k.shape is :", X_valid_k.shape)
print( "X_valid_k.shape is :", X_valid_k.shape)
print( "y_train_k.shape is :", y_train_k.shape)
print( "y_valid_k.shape is :", y_valid_k.shape)
print( "y_valid_k.shape is :", y_valid_k.shape)
```

```
X_train_k.shape is : (90, 4)
X_valid_k.shape is : (30, 4)
X_valid_k.shape is : (30, 4)
y_train_k.shape is : (90,)
y_valid_k.shape is : (30,)
y_valid_k.shape is : (30,)
```

In [16]:
```
# lets have a look at X_trian_k X_test_k and X_valid_k to make sure that they ar
print("\n")
print("Description of training set feature values ")
print(X_train_k.describe().T)
print("\n")
print("Description of validation set feature values ")
print(X_valid_k.describe().T)
print("\n")
print("Description of testing set feature values ")
print(X_test_k.describe().T)
```

```
Description of training set feature values
                   count      mean       std  min  25%  50%  75%  max
sepal length (cm)   90.0  5.846667  0.835074  4.3  5.1  5.8  6.4  7.7
sepal width (cm)    90.0  3.112222  0.462000  2.0  2.8  3.0  3.4  4.4
petal length (cm)   90.0  3.727778  1.797753  1.1  1.5  4.3  5.1  6.7
petal width (cm)    90.0  1.188889  0.769568  0.1  0.3  1.3  1.8  2.5


Description of validation set feature values
                   count      mean       std  min    25%   50%    75%  max
sepal length (cm)   30.0  5.696667  0.791978  4.4  5.200  5.70  6.225  7.7
sepal width (cm)    30.0  2.910000  0.375408  2.2  2.625  3.00  3.100  3.6
petal length (cm)   30.0  3.723333  1.637636  1.0  1.750  4.10  4.975  6.1
petal width (cm)    30.0  1.166667  0.710189  0.2  0.225  1.35  1.750  2.3


Description of testing set feature values
                   count      mean       std  min    25%   50%    75%  max
sepal length (cm)   30.0  5.980000  0.845026  4.7  5.425  6.05  6.500  7.9
sepal width (cm)    30.0  3.040000  0.384708  2.2  2.800  3.00  3.200  3.8
petal length (cm)   30.0  3.883333  1.841305  1.3  1.600  4.50  5.175  6.9
petal width (cm)    30.0  1.263333  0.810910  0.1  0.325  1.35  2.000  2.3
```

In [17]:
```
#Lets see performance of KNN classifier, default case on this dataset
Knn_default = KNeighborsClassifier()
Knn_default.fit(X_train_k, y_train_k)
y_pred_test = Knn_default.predict(X_test_k)
score_default = accuracy_score(y_test_k, y_pred_test)
print(" accuracy on default Knn is : ", score_default , sep ="\t")
```

```
 accuracy on default Knn is :   0.9666666666666667
```

The KNN on default case gives an approximate accuracy of 0.967 on the test set

In [18]:
```
def K_performance(k):
    """
    this function takes input k, ie which is fed to KNN as numbers of neighbours
    account while modelling the model
    it returns the accuracy on the validation set of the model
    """
    Knn_model = KNeighborsClassifier(k)
    Knn_model.fit(X_train_k, y_train_k)
    y_pred_cv = Knn_model.predict(X_valid_k)
    score = accuracy_score(y_valid_k,y_pred_cv)
    return score
```

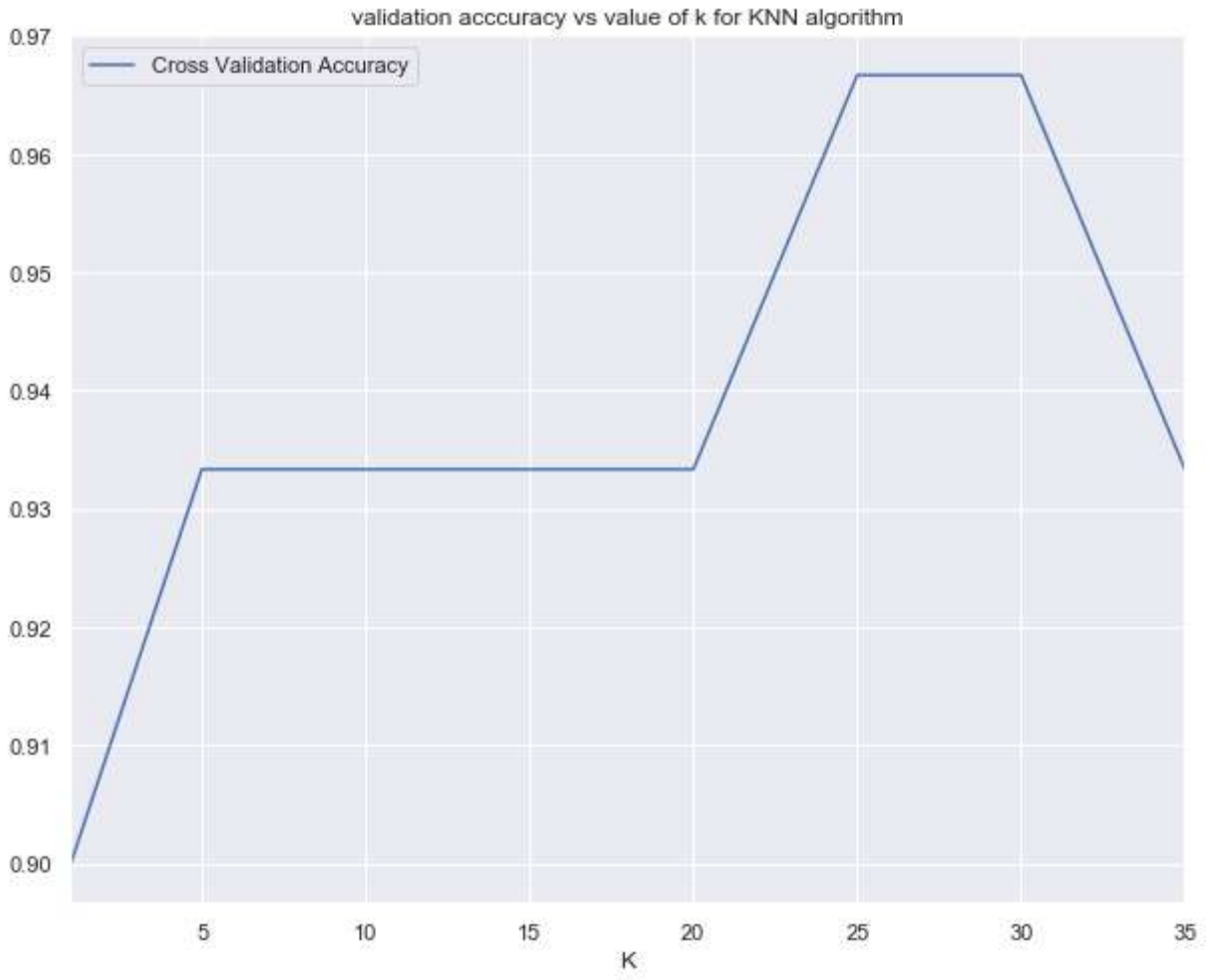
In [19]:
```
# storing KNN model performance (i.e. accuracy score) for various
#k values in the dictionary dict_k
K_vals = [1,5,10,15,20,25,30,35]
dict_k ={}
for k in K_vals:
    temp = K_performance(k)
    dict_k[k] = temp

print ("The dictionary storing accuracy for different values of K is")
print(dict_k)
```

```
The dictionary storing accuracy for different values of K is
{1: 0.9, 5: 0.9333333333333333, 10: 0.9333333333333333, 15: 0.9333333333333333,
20: 0.9333333333333333, 25: 0.9666666666666667, 30: 0.9666666666666667, 35: 0.9
33333333333333}
```

In [20]:
```
#lets see the graph for k vs performance in the validation set
#(I would convert dictionary to dataframe, so as to plot it)
K_val_data = pd.DataFrame(data = list(dict_k.items()))
K_val_data.rename(columns = {0 : "K", 1: "Cross Validation Accuracy"}, inplace =
K_val_data.plot(x= "K", y="Cross Validation Accuracy", title = "validation acccu
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1fdb084b940>



From the Graph, it can be seen, the performance accuracy on validation is highest for k = 25 and 30 ,

1. When K is small, the algorithm is looking only for few neighbors around and hence is performing mediore on validation set, this can be thought as overfitting.
2. From the graph, we can see KNN algorithm perfromance on validation set increases as value of K increase upto a certain value of K.(these values of K seem optimized values)
3. After A certain value of K, increase in K did not result in increase in accuracy (here K = 30), seems like it does provide an overly smoothening effect .

In [46]:
```
# writing a function to see performance of Hypertuned model on test set
def K_performance_test(k):
    """
    this function takes input k, ie which is fed to KNN as numbers of neighbours
    account while modelling the model
    it returns the accuracy on the test set of the model
    """
    Knn_model = KNeighborsClassifier(k)
    Knn_model.fit(X_train_k, y_train_k)
    y_pred_test = Knn_model.predict(X_test_k)
    score = accuracy_score(y_test_k,y_pred_test)
    return score
```

In [47]:
```
# lets see performance of KNN for various values of K
K_test = [25,30]
dict_k_t ={}
for k in K_test:
    temp = K_performance_test(k)
    dict_k_t[k] = temp

print("test set accuracy dictionary with key values as k and accuracy as values
```

```
test set accuracy dictionary with key values as k and accuracy as values  {25:
1.0, 30: 1.0}
```

In [65]:
```
import time
K_time = [25,30]
dict_time_t ={}
for k in K_time:
    start = time.time()
    K_performance_test(k)
    stop = time.time()
    dict_time_t[k] = stop - start
print("test set accuracy dictionary with key values as k and approx fit time (in
```

```
test set accuracy dictionary with key values as k and approx fit time (in secon
ds) as values  {25: 0.002956390380859375, 30: 0.004007816314697266}
```

Both for K= 25 and k =30 the model is able to achieve 100% accuracy on the test set. Since K =30 will take more time and would be computationally expensive compared to the case for K = 25. Hence we choose K = 25. Also, K =30 which is even, which is not preffered over an alternative which is odd (K=25).
Thus the best K parameter to consider would be K=25

SVM classifier

In [68]:
```
#importing useful libraries
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

In [69]:
```
#preprocessing the dataset,printing shapes of data to be fed
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X,y,test_size = 0.20
print( "X_train_s.shape is :", X_train_s.shape)
print( "X_test_s.shape is :", X_test_s.shape)
print( "y_train_s.shape is :", y_train_s.shape)
print( "y_test_s.shape is :", y_test_s.shape)
```

```
X_train_s.shape is : (120, 4)
X_test_s.shape is : (30, 4)
y_train_s.shape is : (120,)
y_test_s.shape is : (30,)
```

In [70]:
```
#defining parameters for which are to be optimized
parameters_SVM = {'C':[0.1, 0.5, 1, 2, 5, 10, 20, 50]}
```

In [145]:
```
#writing function that takes C values as input and returns mean accuracy on 10 f
def SVM_performance(C):
    """
    this function takes input C, ie which is fed to SVC parameter C to take into
    account while modelling the model
    it returns the mean accuracy of 10-fold validation sets of the model
    """
    SVM_model = SVC(C=C, kernel = 'linear', random_state = 42)
    scores_SVM = cross_val_score(SVM_model, X_train_s, y_train_s, cv=10)
    return scores_SVM.mean()
```