Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
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)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

From the above output image, we can see that our data is successfully scaled.

- **Fitting K-NN classifier to the Training data:**
  Now we will fit the K-NN classifier to the training data.
- To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class.
- The Parameter of this class will be
  - **n_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
  - **metric='minkowski':** This is the default parameter and it decides the distance between the points.
  - **p=2:** It is equivalent to the standard Euclidean metric.

  And then we will fit the classifier to the training data. Below is the code for it:

1. **#Fitting K-NN classifier to the training set**
2. from sklearn.neighbors **import** KNeighborsClassifier
3. classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
4. classifier.fit(x_train, y_train)

Output: By executing the above code, we will get the output as:

```
Out[10]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

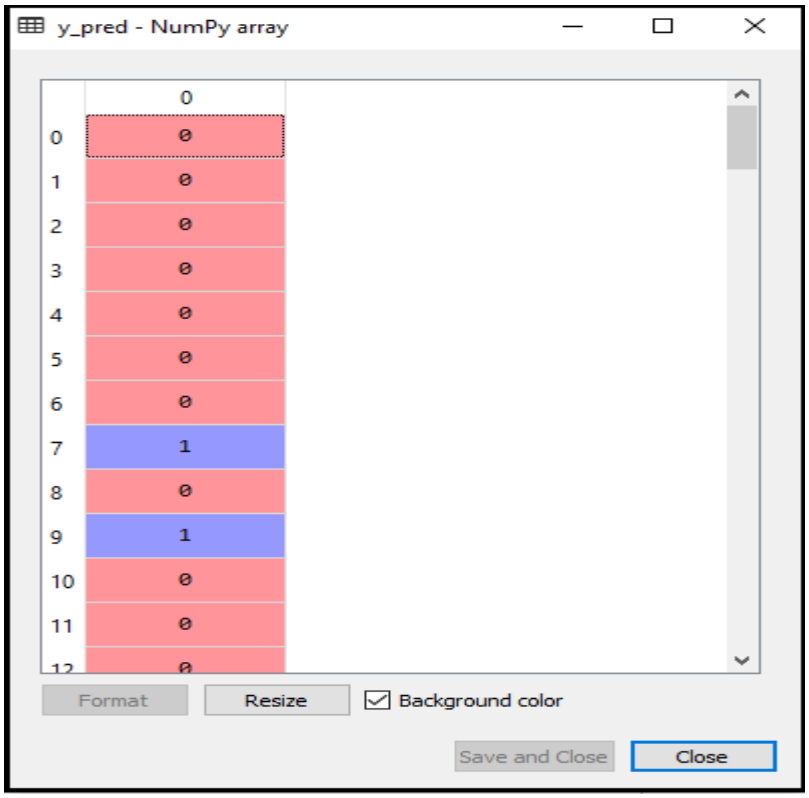
- **Predicting the Test Result:** To predict the test set result, we will create a **y_pred** vector as we did in Logistic Regression. Below is the code for it:

#Predicting the test set result

1. y_pred= classifier.predict(x_test)

Output:

The output for the above code will be:



- **Creating the Confusion Matrix:**
  Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

```
1. #Creating the Confusion matrix
2.     from sklearn.metrics import confusion_matrix
3.     cm= confusion_matrix(y_test, y_pred)
```

In above code, we have imported the confusion_matrix function and called it using the variable cm.

Output: By executing the above code, we will get the matrix as below:

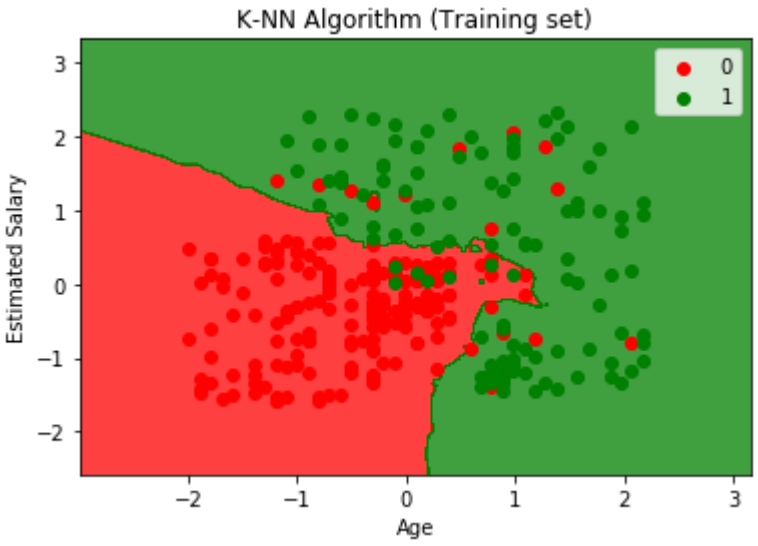
In the above image, we can see there are 64+29= 93 correct predictions and 3+4= 7 incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

- **Visualizing the Training set result:**
  Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic Regression, except the name of the graph. Below is the code for it:

```
1. #Visulaizing the trianing set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_train, y_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, ste
   p  =0.01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('red','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.         c = ListedColormap(('red', 'green'))(i), label = j)
13. mtp.title('K-NN Algorithm (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:

By executing the above code, we will get the below graph:



The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

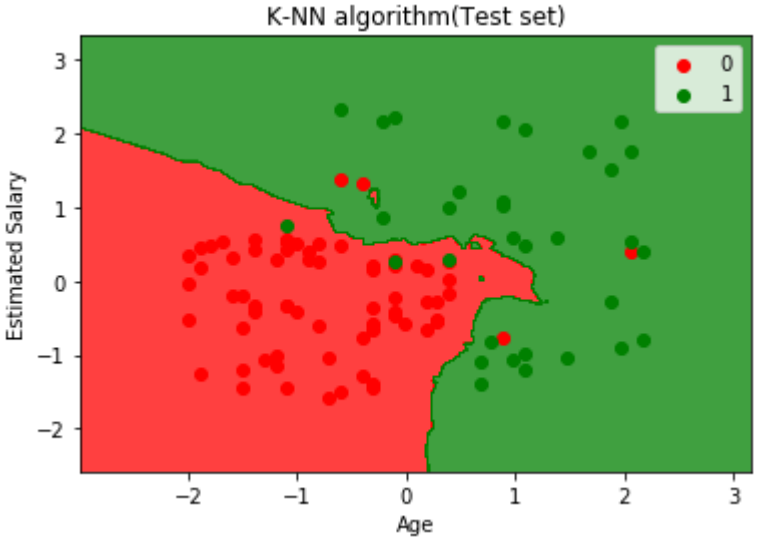
- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.

- **Visualizing the Test set result:**
  After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x_train and y_train** will be replaced by **x_test and y_test**.
  Below is the code for it:

#Visualizing the test set result

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, ste
p  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



- The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.
- However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).