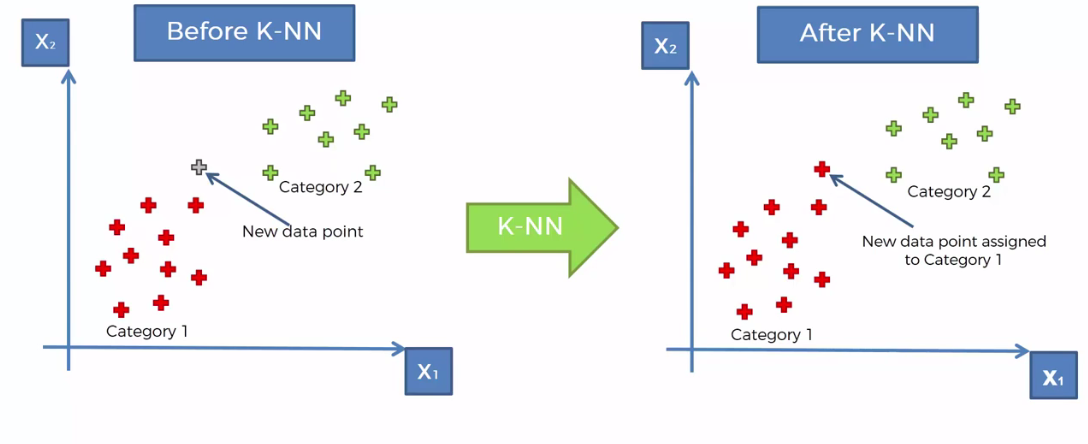
Chapter 3 : part 2

**Logistic Regression**

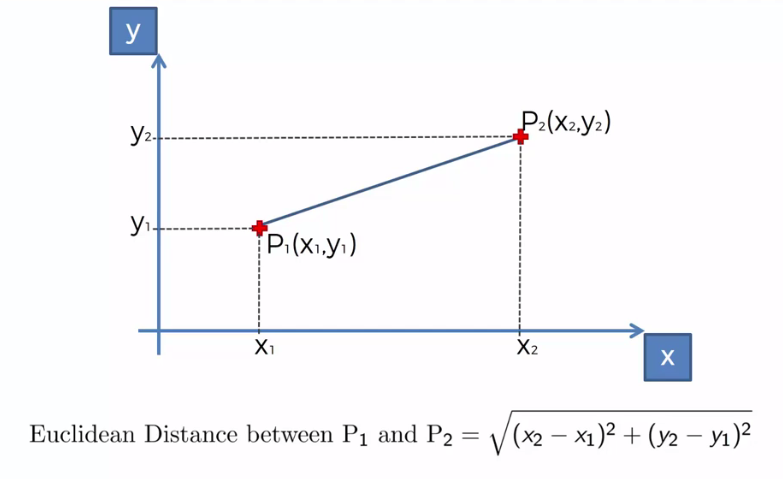
**3.1.1 Logistic Regression**

* Let's imagine that we have a scenario where we have two categories *already* *present* in our data set so we've identified two categories: Green and Red.
* For simplicity's let's consider two variables or two columns and in our data-set so all of this grouping is happening based on these two columns.
* Now let's say we *add a new data point* into our data set. The question is should it fall into the *Red* *category* or should fall to the *Green* *category*.



* K-Nearest Neighborhoods: The steps are given below:

1. STEP 1: Choose the number ***K*** of neighbors (select the no. of points which will be the Neighborhood).
2. STEP 2: Calculate the *Euclidean distance* of *K number of neighbors*
3. STEP 3: Take the ***K nearest neighbors*** of the ***new*** ***data*** ***point***, according to the Euclidean distance (we can also use other kind of distances)
4. STEP 4: Among these ***K neighbors***, *count* *the* *Number* of data points in each category (you might even have more than two categories in your data set).
5. STEP 5: *Assign* the *new* *data* *point* to the category where you counted the *most* *neighbors*
6. Final: Your Model is Ready



* Euclidean distance : The formula is , whwere the points are
* How we classify: Let's no. of points in our nbd is **K=5**.
* Now take the K nearest neighbors of the new data point according to their Euclidean distance.
* Once you've taken the nearest neighbors among these K neighbors you need to count the number of data points in each category.
* So how many data points fell into each category (you might even have more than two categories in your data set).
* Just calculate how many fall into each category and then you need to assign the new data point to the category where you counted the most neighbors.

|  |  |
| --- | --- |
|  |  |
|  |  |

* How to select the value of K in the K-NN Algorithm?: Below are some points to remember while selecting the value of K in the K-NN algorithm:
* There is ***no*** ***particular*** ***way*** to determine the best value for "K", so we need to ***try some values*** to find the best out of them. The most ***preferred*** value for ***K*** is ***5***.
* A very low value for ***K*** such as ***K=1*** or ***K=2***, can be ***noisy*** and lead to the effects of ***outliers*** in the model.
* Large values for K are good, but it may find some difficulties.
* Advantages of KNN Algorithm:
* It is simple to implement.
* It is robust to the noisy training data
* It can be more effective if the training data is large.
* Disadvantages of KNN Algorithm:
* Always needs to determine the value of K which may be complex some time.
* The computation cost is high because of calculating the distance between the data points for all the training samples.

Notes:

* K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on *Supervised* *Learning* technique.
* K-NN algorithm assumes the *similarity* between the *new case/data* and available cases and put the new case into the *category* that is *most* *similar* to the *available* *categories*.
* K-NN algorithm stores all the available data and classifies a new data point based on the *similarity*. This means when *new data* appears then it can be easily *classified* into a *well suite category* by using K- NN algorithm.
* K-NN algorithm can be used for ***Regression*** as well as for ***Classification*** but mostly it is used for the ***Classification*** problems.
* Non-Parametric Algorithm: K-NN is a non-parametric algorithm, which means it ***does not make any assumption on underlying data***.
* Lazy learner algorithm: It is also called a *lazy* *learner* *algorithm* because it *does not learn from the training set immediately instead it stores the dataset and at the time of classification*, it performs an action on the dataset.
* KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

|  |  |
| --- | --- |
| * Example: Suppose, we have an image of a *creature* that looks similar to *cat* and *dog*, but we want to know either it is a *cat* or *dog*. So for this *identification*, we can use the *KNN algorithm*, as it works on a similarity measure. Our *KNN model will find the similar features* of the *new data set* to the *cats* and *dogs* images and based on the *most similar features* it will put it in either cat or dog category. | K-Nearest Neighbor(KNN) Algorithm for Machine Learning |

**3.1.2 Python implementation of the KNN algorithm**

To do the ***Python implementation*** of the ***K-NN*** algorithm, we will use the *same* *problem* and *dataset* which we have used in *Logistic* *Regression*. But here we will improve the performance of the model. Below is the problem description:

|  |  |
| --- | --- |
| * Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the Estimated ***Salary*** and ***Age*** we will consider for the independent variable and the ***Purchased*** variable is for the dependent variable. Below is the dataset: * Steps to implement the K-NN algorithm:  1. Data Pre-processing step 2. Fitting the K-NN algorithm to the Training set 3. Predicting the test result 4. Test accuracy of the result(Creation of Confusion matrix) 5. Visualizing the test set result. | K-Nearest Neighbor(KNN) Algorithm for Machine Learning |

* Data Pre-Processing Step: The ***Data Pre-processing*** step will remain exactly the ***same*** as ***Logistic Regression***.

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*data Extract*

dtaSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dtaSet**.**iloc[:, [2, 3]]**.**values

y = dtaSet**.**iloc[:, 4]**.**values

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

#*0.25 test\_size means "1/4"th of the total observation*

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

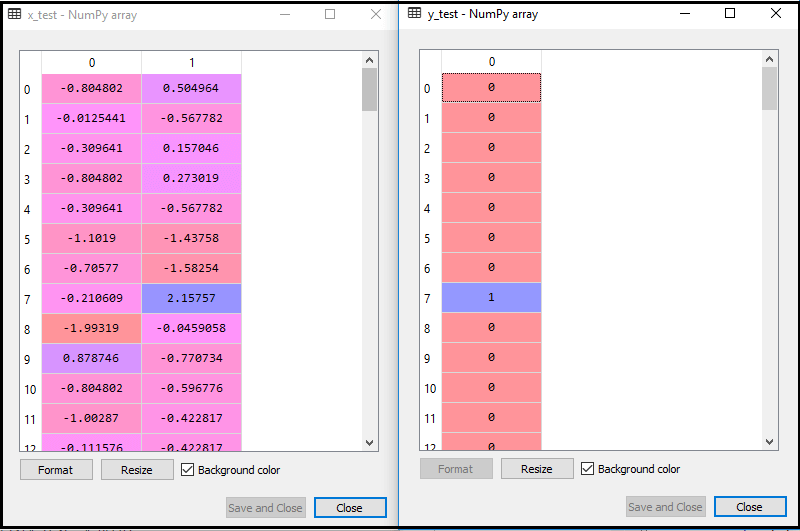
#*Feature-Scaling*

**from** sklearn**.**preprocessing **import** StandardScaler

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)



* 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:

#*Fitting K-NN classifier to the training set*

**from** sklearn**.**neighbors **import** KNeighborsClassifier

clsFier = **KNeighborsClassifier**(p = 2, metric= "minkowski", n\_neighbors=5)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

|  |  |
| --- | --- |
| * 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*  y\_prd = clsFier**.predict**(X\_test)   * Creating the Confusion Matrix: In below code, we have imported the confusion\_matrix function and called it using the variable cm.   #*Making the confusion matrix use the function "confusion\_matrix"*  **from** sklearn**.**metrics **import** confusion\_matrix  cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)  #*parameters of cm: y\_true: Real values, y\_pred: Predicted value*    We can see there are ***64+29= 93 correct predictions*** and ***3+4= 7 incorrect predictions***, whereas, in ***Logistic Regression***, there were ***11 incorrect prediction***s. 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:

#*Visualising the Training set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

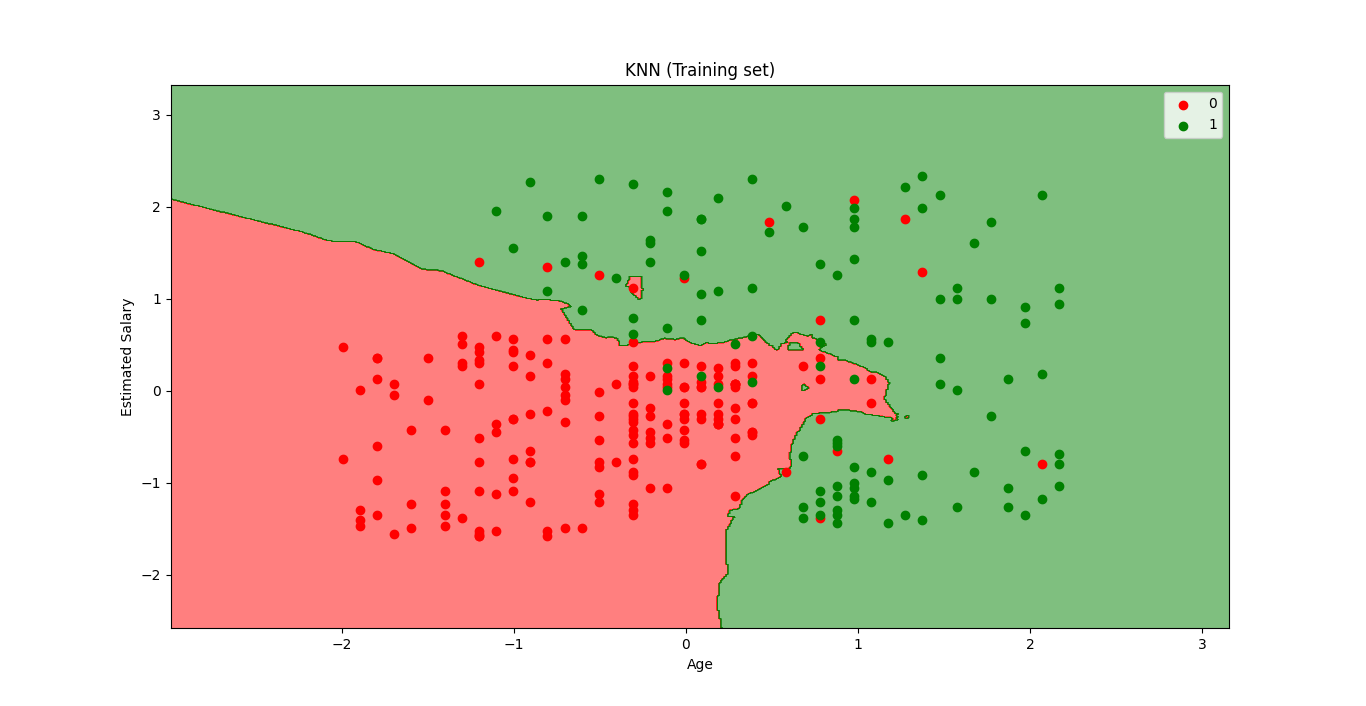
pLt**.title**('Logistic Regression (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()



* 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***.

#*Visualising the Test set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

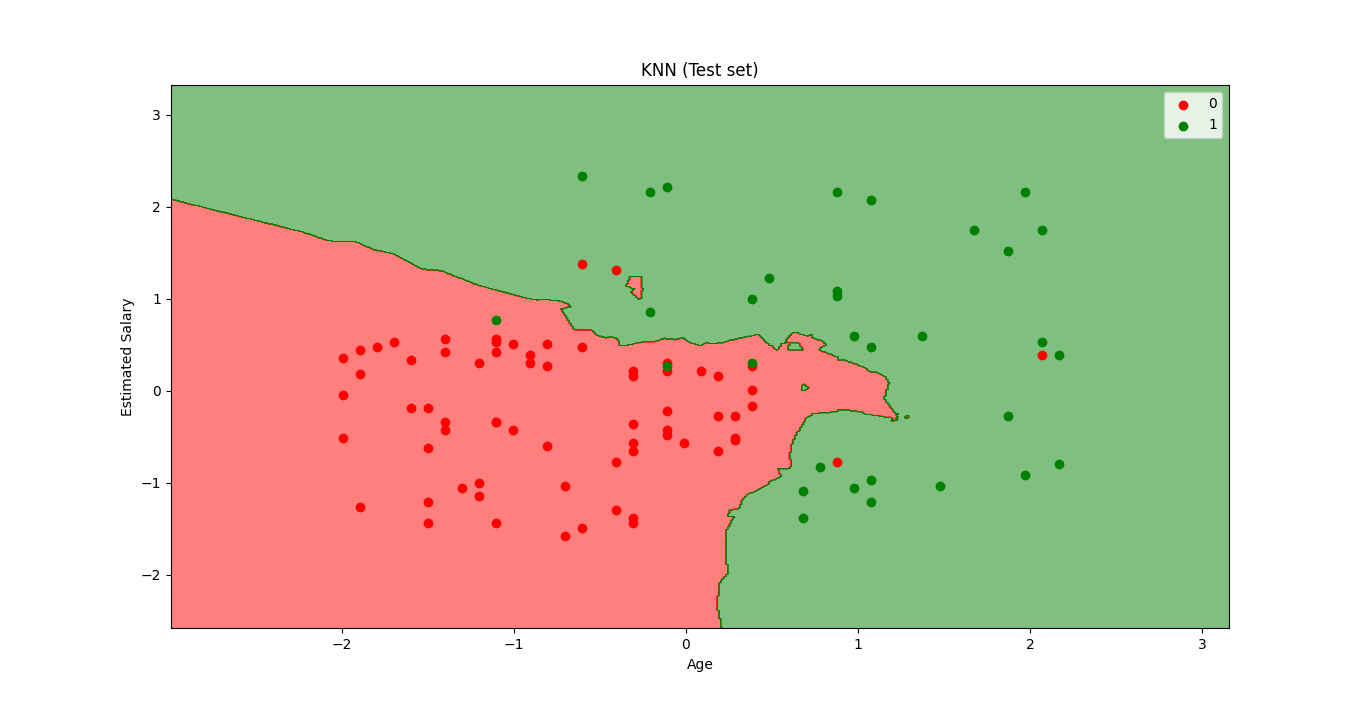
pLt**.title**('Logistic Regression (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()



* 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).
* Nlinear – Non\_Linear Classifier: If the *"Prediction Boundary"* is linear then the classification model is called linear model.

Practiced version

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*data Extract*

dtaSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dtaSet**.**iloc[:, [2, 3]]**.**values

y = dtaSet**.**iloc[:, 4]**.**values

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

#*0.25 test\_size means "1/4"th of the total observation*

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)

#*Fitting K-NN classifier to the training set*

**from** sklearn**.**neighbors **import** KNeighborsClassifier

clsFier = **KNeighborsClassifier**(p = 2, metric= "minkowski", n\_neighbors=5)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualising the Training set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('KNN (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('KNN (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*python prctc\_knn.py*