Assignment 2

DSC 478

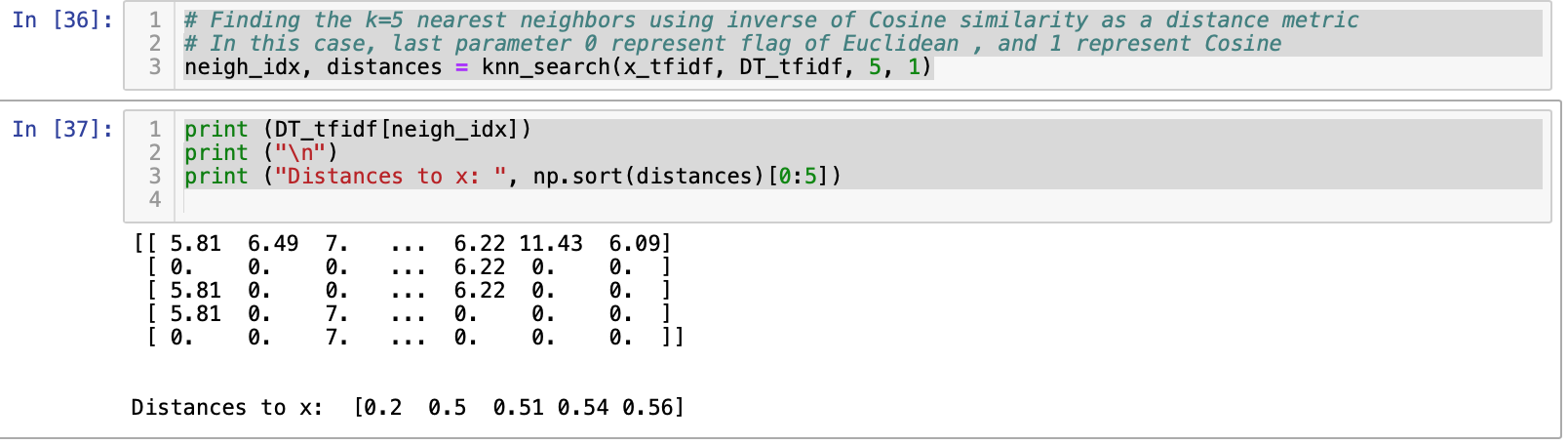
Chaonan Shi

1901412

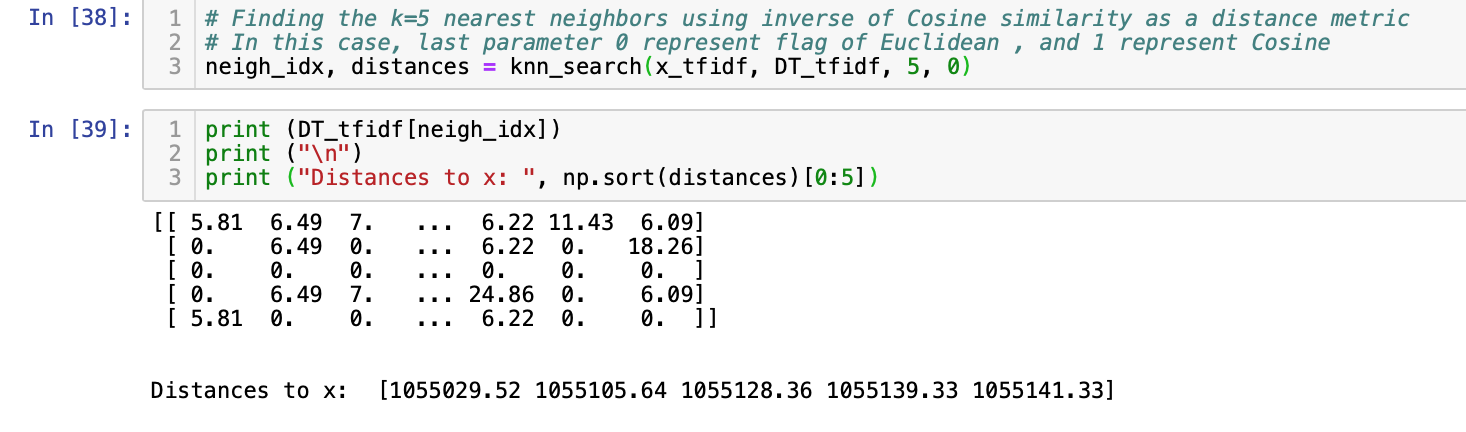
**20 Newsgroup data set:**

1. Create your own KNN classifier function. Your classifier should allow as input the training data matrix, the training labels, the instance to be classified, the value of K, and should return the predicted class for the instance and the top K neighbors. Your classifier should work with Euclidean distance as well as Cosine Similarity. You may create two separate classifiers, or add this capability as a parameter for the classifier function.

*Figure 1:*



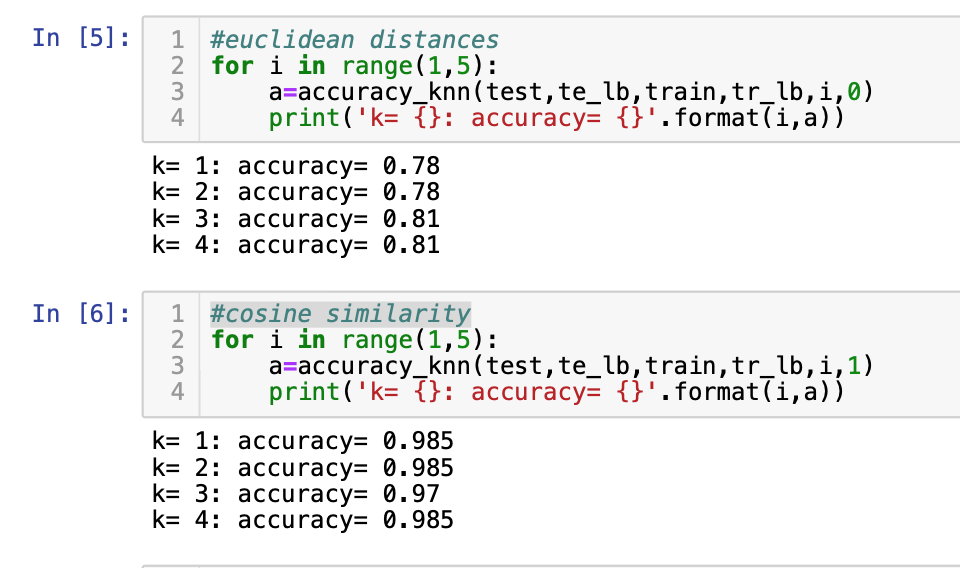
*Figure 2:*



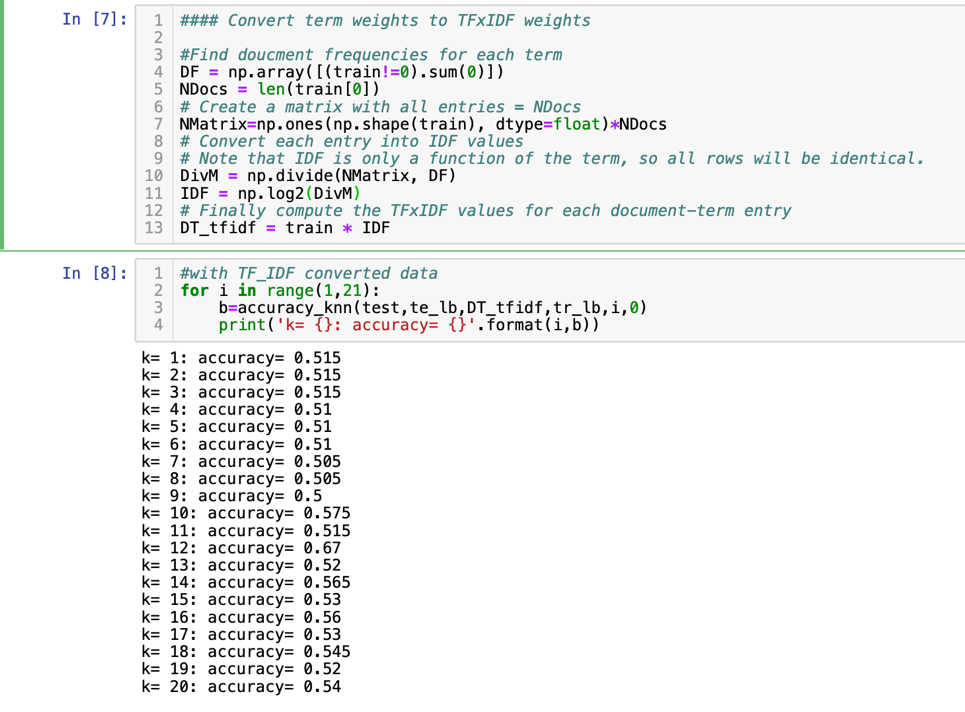
**Analysis:**

From above analysis, the figure1 represents the distance calculated by Cosine, and figure2 represents the distance calculated by Euclidean. As result, we can conclude that since the measurements of the distance are different, then results of tf\*idf is different either.

1. Create a function to compute the classification accuracy over the test data set (ratio of correct predictions to the number of test instances). This function will call the classifier function in part **a** on all the test instances and in each case compares the actual test class label to the predicted class label.
2. Run your accuracy function on a range of values for K in order to compare accuracy values for different numbers of neighbors. Do this both using Euclidean Distance as well as Cosine similarity measure. [For example, you can try evaluating your classifiers on a range of values of K from 1 through 20 and present the results as a table or a graph].



1. Using Python, modify the training and test data sets so that term weights are converted to TFxIDF weights (instead of raw term frequencies). [See class notes on [text categorization](https://d2l.depaul.edu/d2l/common/dialogs/quickLink/quickLink.d2l?ou=706272&type=content&rcode=depaul-3314410)]. Then, rerun your evaluation on the range of K values (as above) and compare the results to the results without using TFxIDF weights.



**Analysis:**

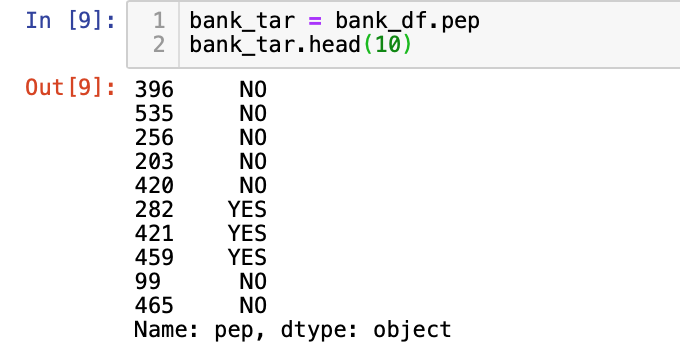
According to the accuracy reuslt of those three, runing knn with cosine similarity method has the highest accuracy then knn with Euclidean similarity method ranked second and knn with tf\_idf data converted ranked third.

1. Create a new classifier based on the **Rocchio Method adapted for text categorization [See**[**class notes on text categorization**](https://d2l.depaul.edu/d2l/common/dialogs/quickLink/quickLink.d2l?ou=706272&type=content&rcode=depaul-3314410)**]**. You should separate the training function from the classification function. The training part for the classifier can be implemented as a function that takes as input the training data matrix and the training labels, returning the prototype vectors for each class. The classification part can be implemented as another function that would take as input the prototypes returned from the training function and the instance to be classified. This function should measure Cosine similarity of the test instance to each prototype vector. Your output should indicate the predicted class for the test instance and the similarity values of the instance to each of the category prototypes. Finally, compute the classification accuracy using the test instances and compare your results to the best KNN approach you tried earlier.

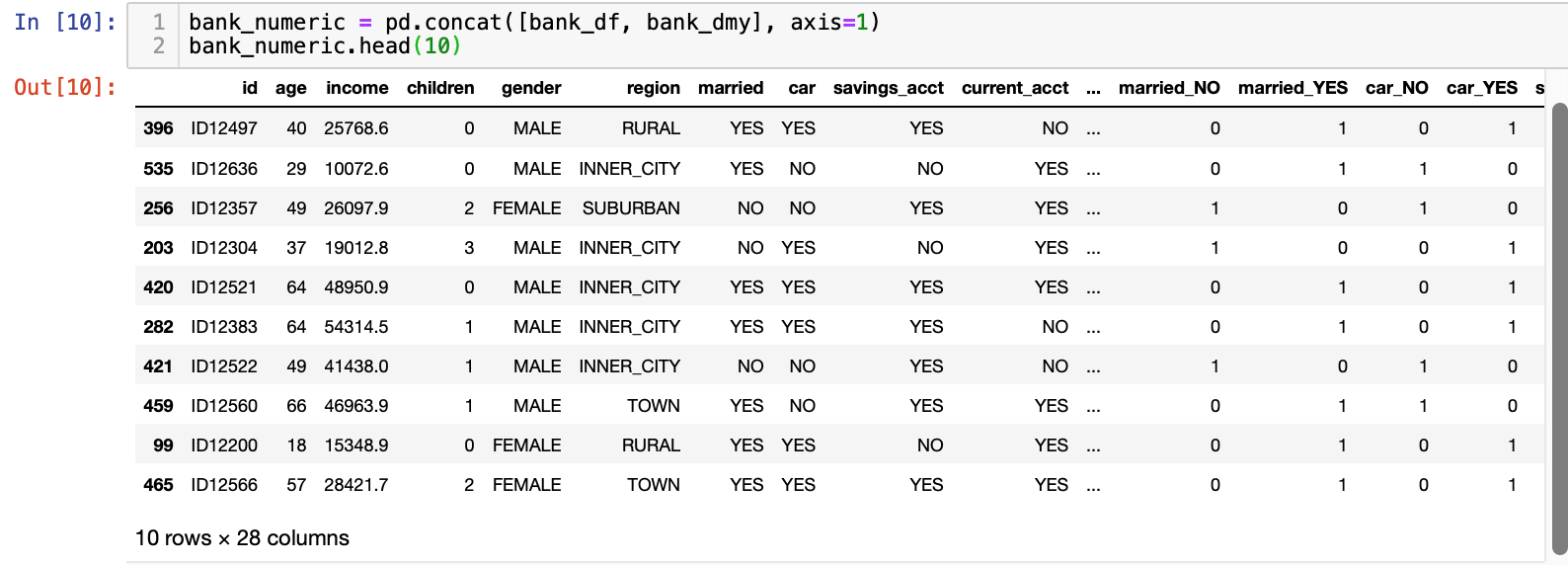
**Dataset: bank\_data.csv:**

1. Load and preprocess the data using Numpy or Pandas and the preprocessing functions from scikit-learn. Specifically, you need to separate the target attribute ("pep") from the portion of the data to be used for training and testing. You will need to convert the selected dataset into the Standard Spreadsheet format (scikit-learn functions generally assume that all attributes are in numeric form). Finally, you need to split the transformed data into training and test sets (using 80%-20% randomizedsplit). [Review Ipython Notebook examples from Week 4 for different ways to perform these tasks.]

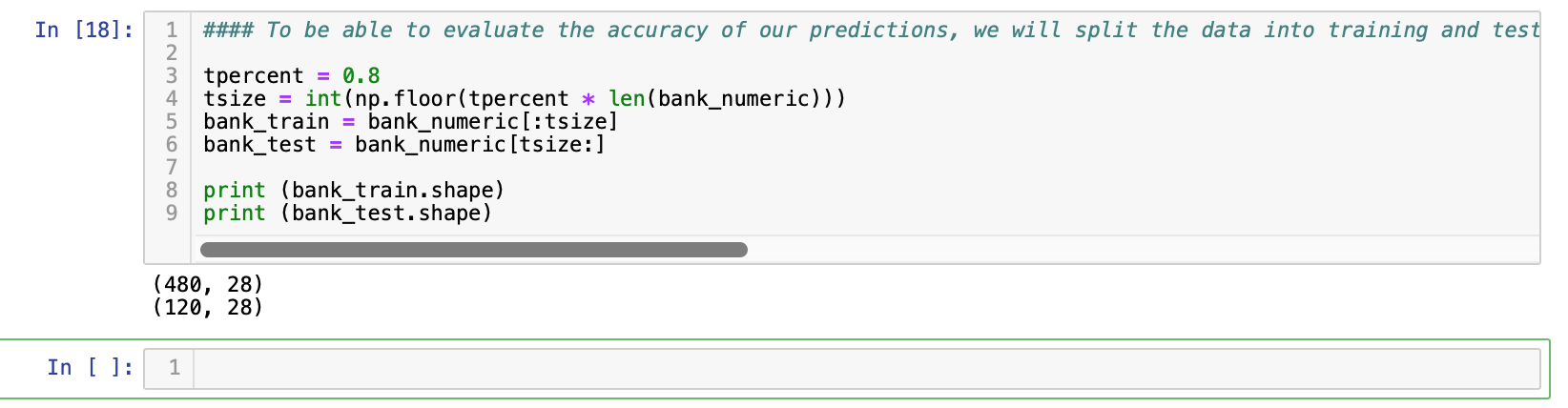
*Figure 2.1:*



*Figure 2.2:*

**

*Figure 2.3:*



**Analysis:**

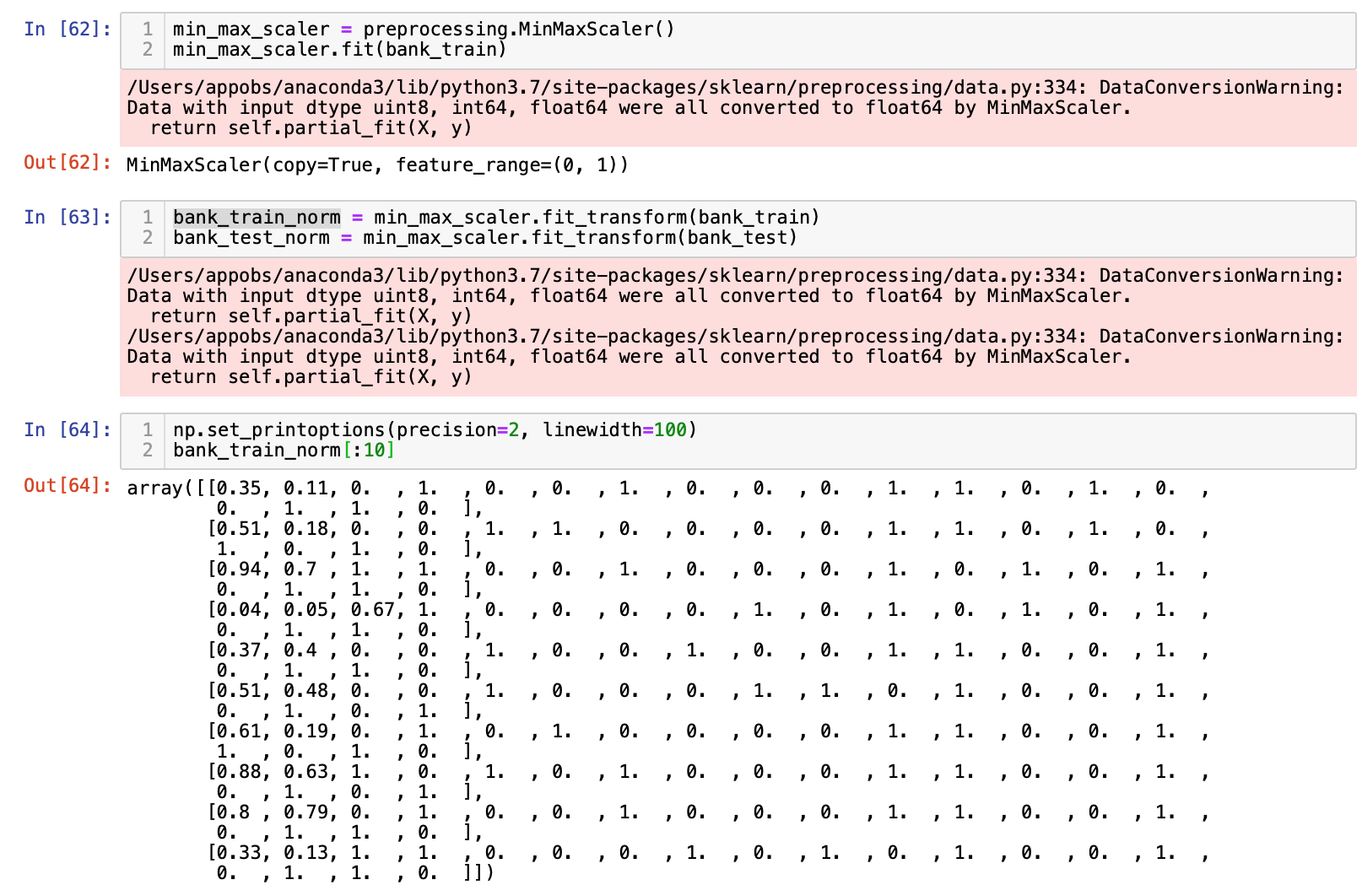
Figure2.1 shows separate the target attribute ("pep") from the portion of the data to be used for training and testing.

Figure2.2 shows convert the selected dataset into the Standard Spreadsheet format;

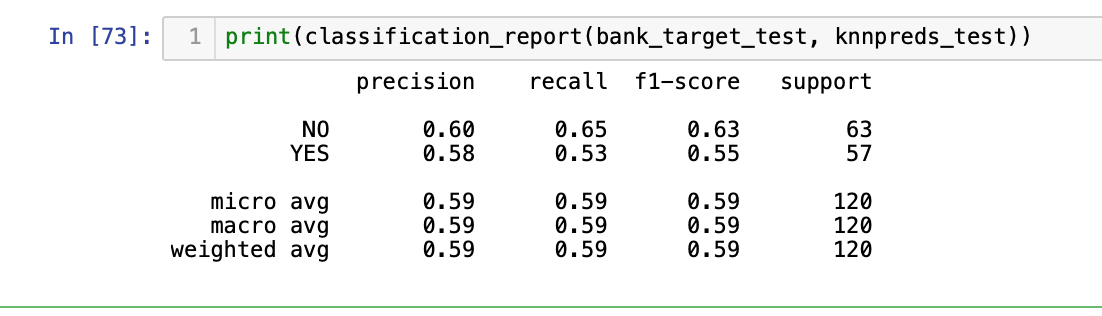
Fugure2.3 shows split the transformed data into training and test sets (using 80%-20% randomizedsplit).

1. Run scikit-learn's KNN classifier on the test set. Note: in the case of KNN, you should first normalize the data so that all attributes are in the same scale (normalize so that the values are between 0 and 1). Generate the confusion matrix (visualize it using Matplotlib), as well as the classification report. Also, compute the average accuracy score. Experiment with different values of K and the weight parameter (i.e., with or without distance weighting) for KNN to see if you can improve accuracy (you do not need to provide the details of all of your experimentation, but provide a short discussion on what parameters worked best as well as your final results).

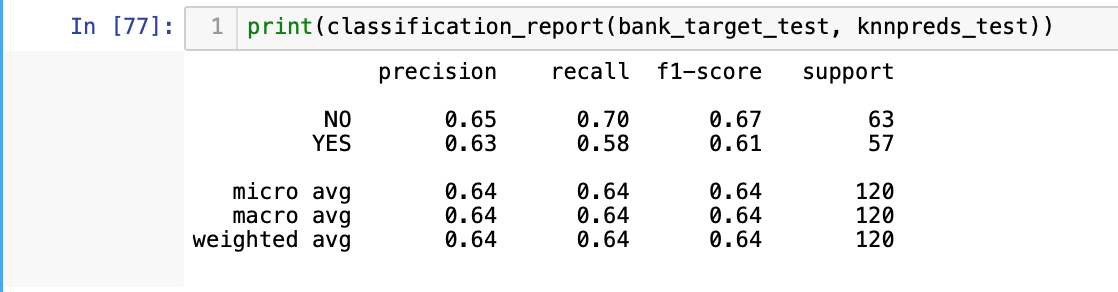
*Figure 2.4:*

**

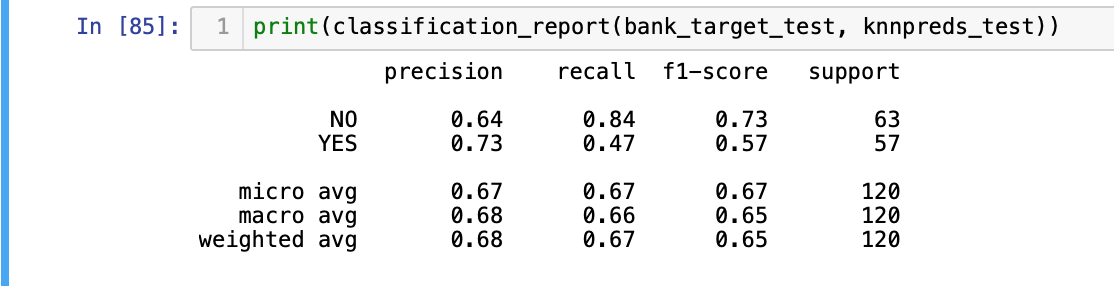
*Figure 2.5:*

**

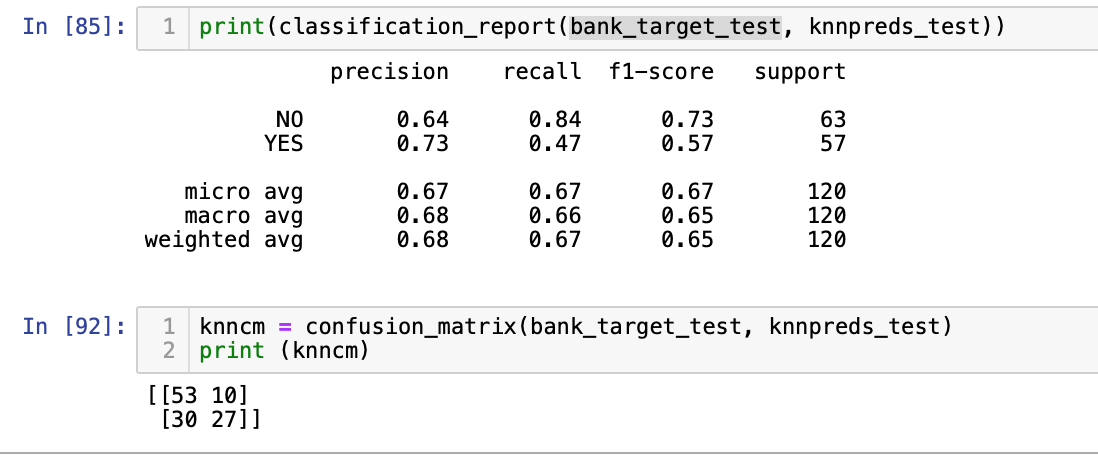
*Figure 2.6:*

**

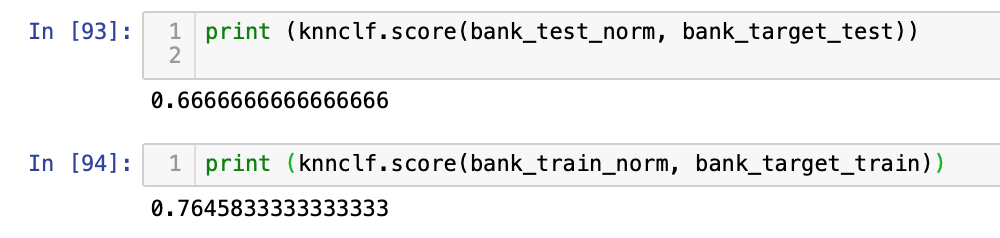
*Figure 2.7:*

**

*Figure 2.8:*

**

*Figure 2.9:*

**

**Analysis:**

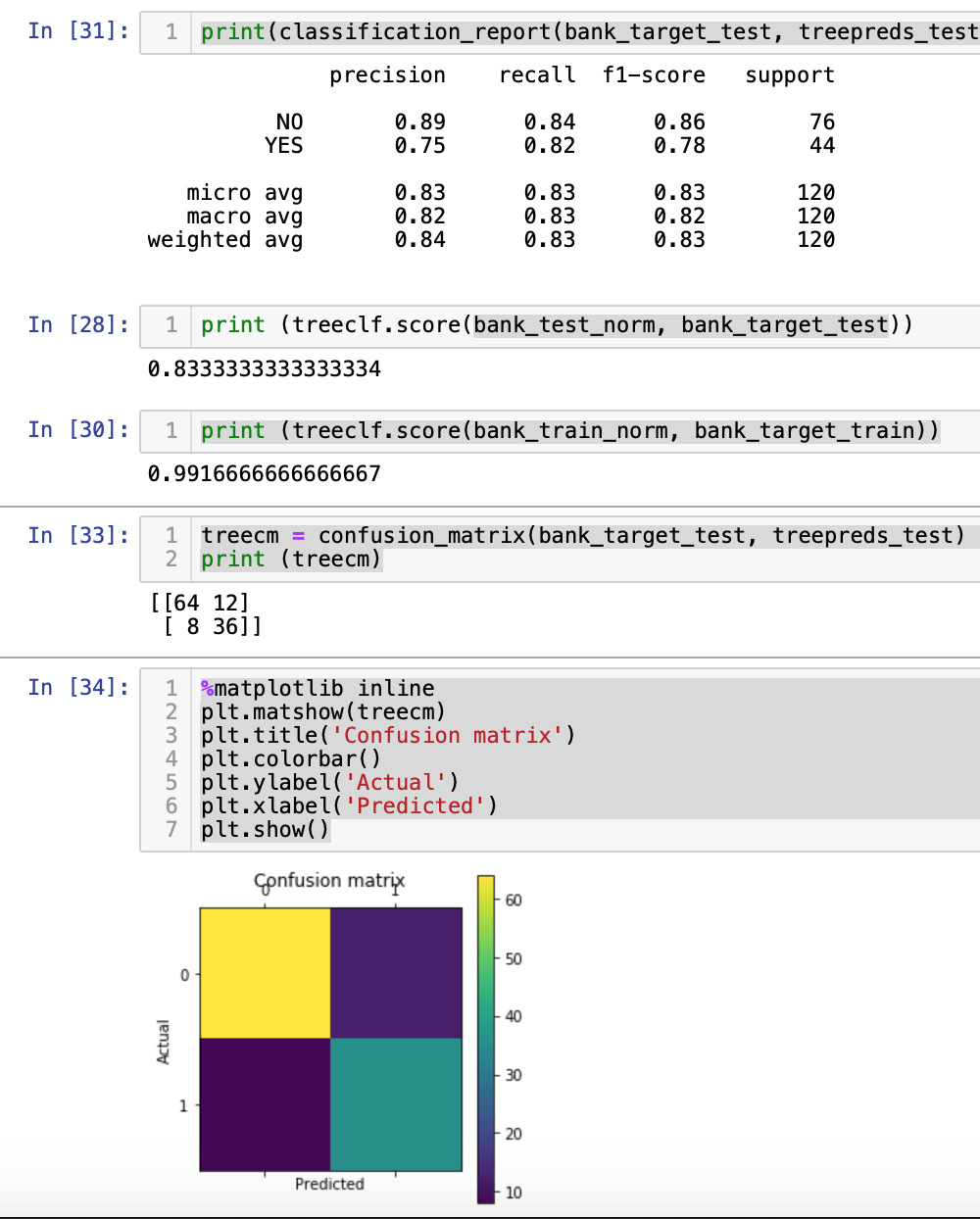
Figure2.4 shows I scaled both training and testing;

Figure2.5 – Figure2.7 show different accuracy scores under different Ks. For instance, the Figure2.5’s result with K =5, Figure2.6’s result with K =10, Figure2.7’s result with K =10 and without ‘distance’;

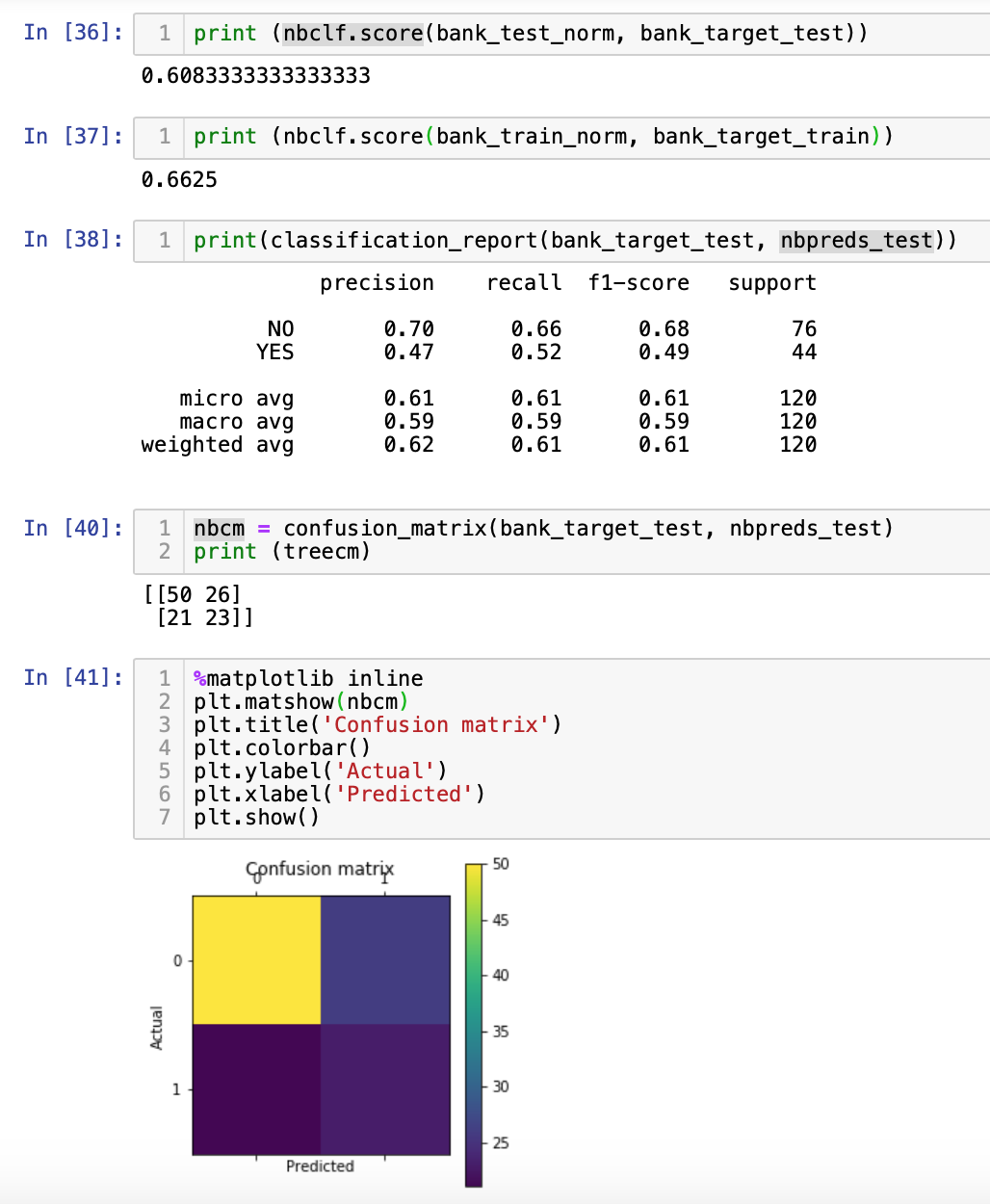
As result, we can summarize that the best prediction for KNN is setting with k=10 and without distances. In this case, increasing K does not improve the results since if k is relatively large, than it possible some of irrelated records been included, and that results the worse prediction score. Moreover, if we do not including parameter of weight, the default setting is ‘uniform’, which meaning of  All points in each neighborhood are weighted equally. Thus, it makes sense that larger K results in worse accuracy score, since irrelevant records may been weighted larger than others which relevant.

1. Repeat the classification using scikit-learn's decision tree classifier (using the default parameters) and the Naive Bayes (Gaussian) classifier. As above, generate the confusion matrix, classification report, and average accuracy scores for each classifier. For each model, compare the average accuracy scores on the test and the training data sets. What does the comparison tell you in terms of bias-variance trade-off?

*Figure 2.10:*



*Figure 2.11:*

**

**Analysis:**

Figure2.10 shows accuracy report for Decision Tree algorithm;

Figure2.11 shows accuracy report for Naïve Bayes algorithm;

As conclusion, we can summarize that the result from decision tree is way better that others. For accuracy score of test and training, if accuracy on train set significantly higher than the test set, which gap between train and test data is caused by fitting:

1. noises in train data (hence overfitting)
2. features exist only in train data

The regularisation can control fitting to noises. But for the second point, we may need a process like feature selection.

1. Discuss your observations based on the above experiments.

**Analysis:**

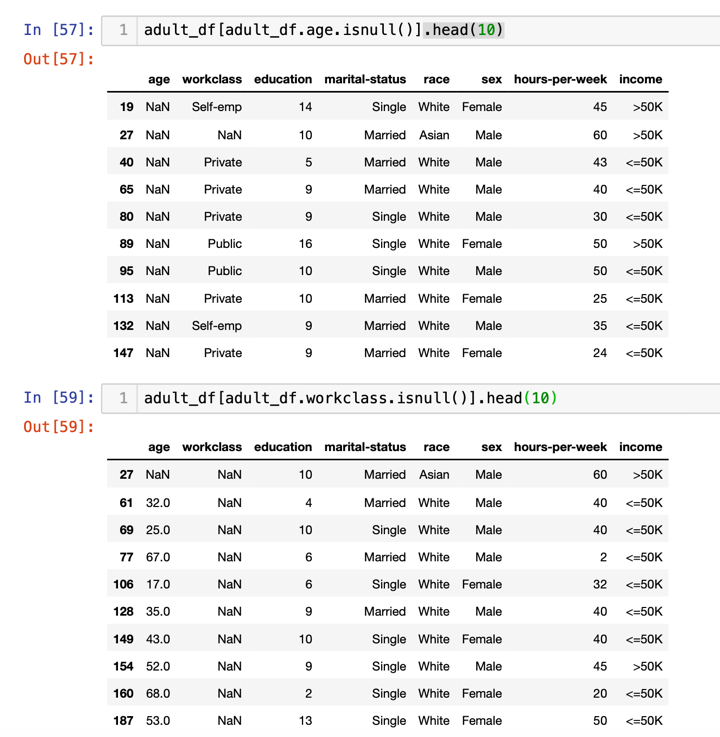
From above observation, we can summarize that the decision tree perform way better that KNN and Naïve Bayes algorithm. From my point of view, the KNN algorithm is easy but risky, since it may contains too much irrelevant records based on the large number of K. On the other hand, the Naïve Bayes been limited on feature selection. For instance, more irrelevant features been included, the more less accuracy score Naïve Bayes performed.

However, even the Decision Tree performed far better than others, we can still observe overfitting problems on it. The test accuracy is 0.83, but train accuracy is 0.99, meaning the algorithm trying too hard to learn from noise. In this case, we may need to adjust the parameter inside of decision tree function, for instance, adjust the measurement from ‘entropy’ to ‘Gini’. Moreover, take heavier regularization to train set.

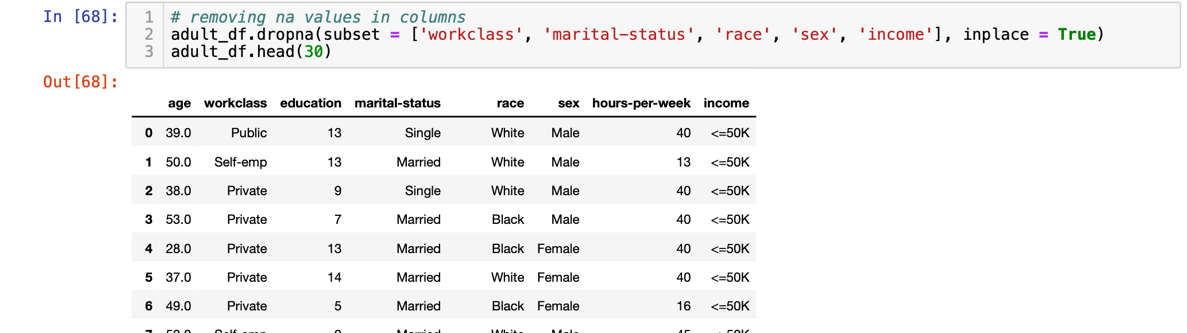
[**Adult Census Data Set**](https://archive.ics.uci.edu/ml/datasets/Adult)**:**

1. **Preprocessing and data analysis:**
2. Examine the data for missing values. In case of categorical attributes, remove instances with missing values. In the case of numeric attributes, impute and fill-in the missing values using the attribute mean.

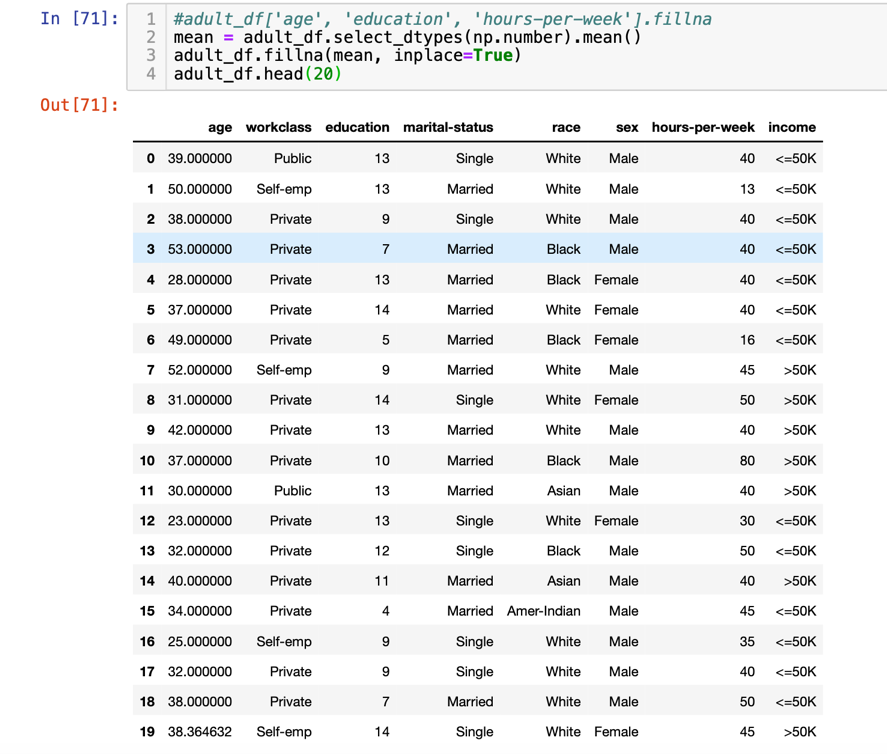
*Figure 3.01:*

**

*Figure 3.02:*

**

*Figure 3.03:*

**

**Analysis:**

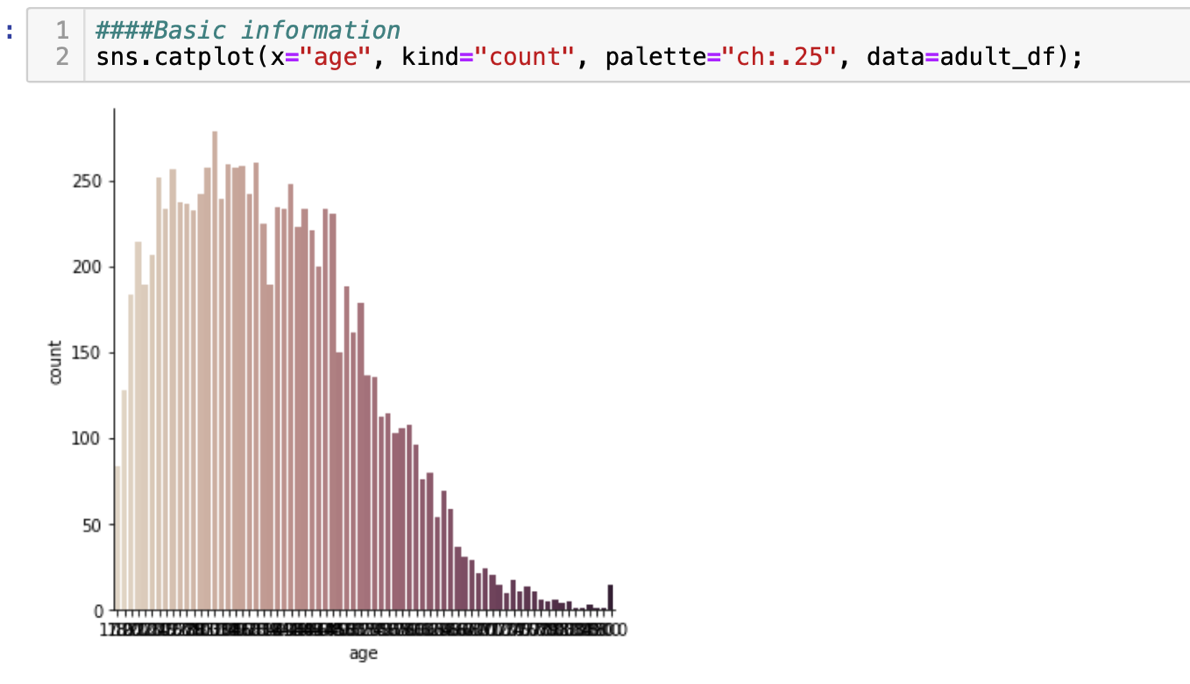
From above graphs, we can summarize that from figure 3.01, features: age and workclass contains missing value;

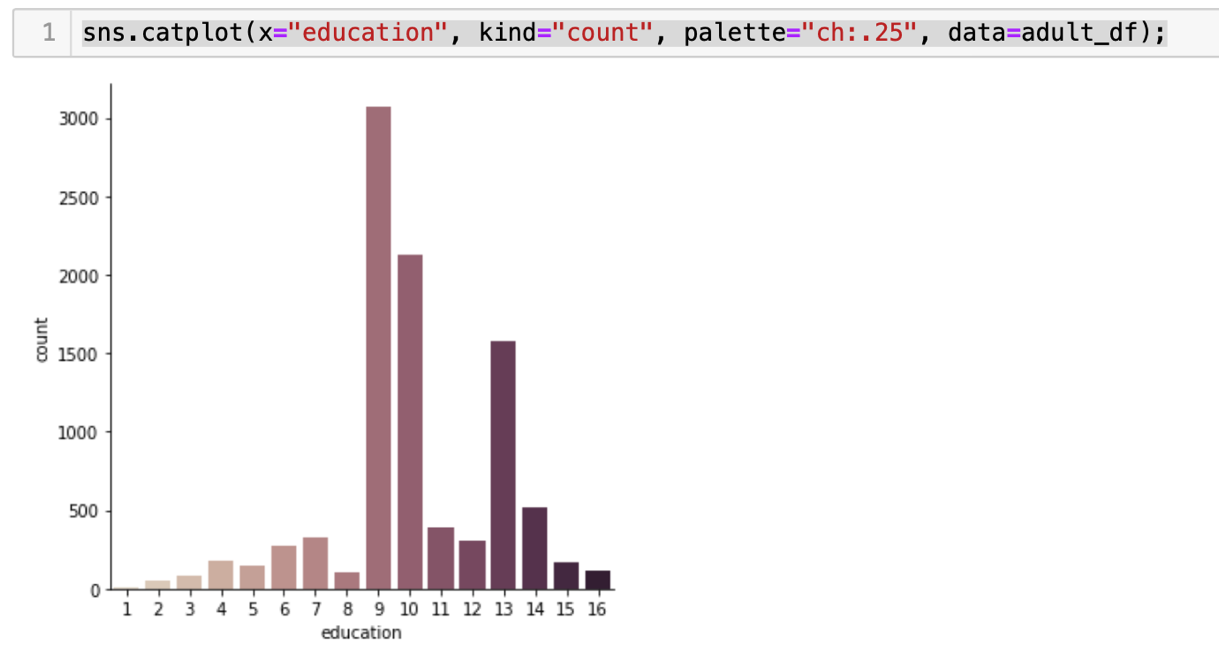
Figure 3.02 shows that remove instances with missing values for categorical features;

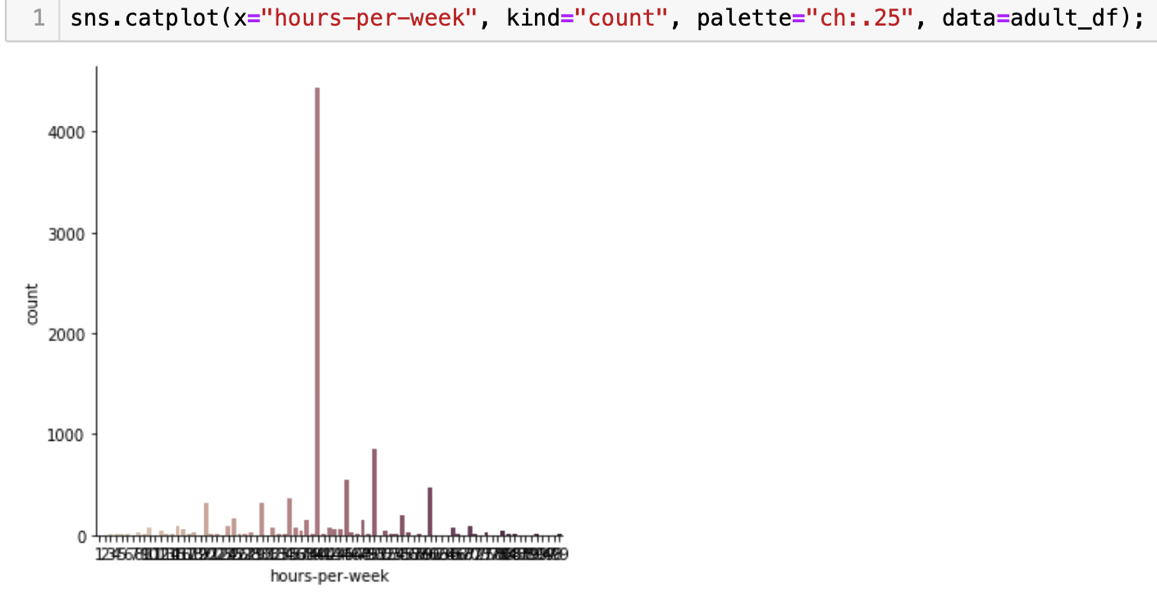
Figure 3.03 shows that replace instances with missing values for numeric features;

1. Examine the characteristics of the attributes, including relevant statistics for each attribute, histograms illustrating the distributions of numeric attributes, bar graphs showing value counts for categorical attributes, etc.

*Figure 3.04:*

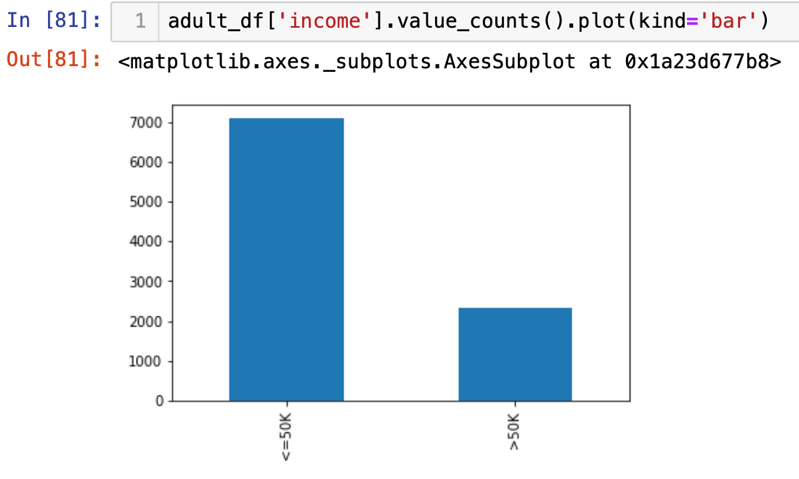
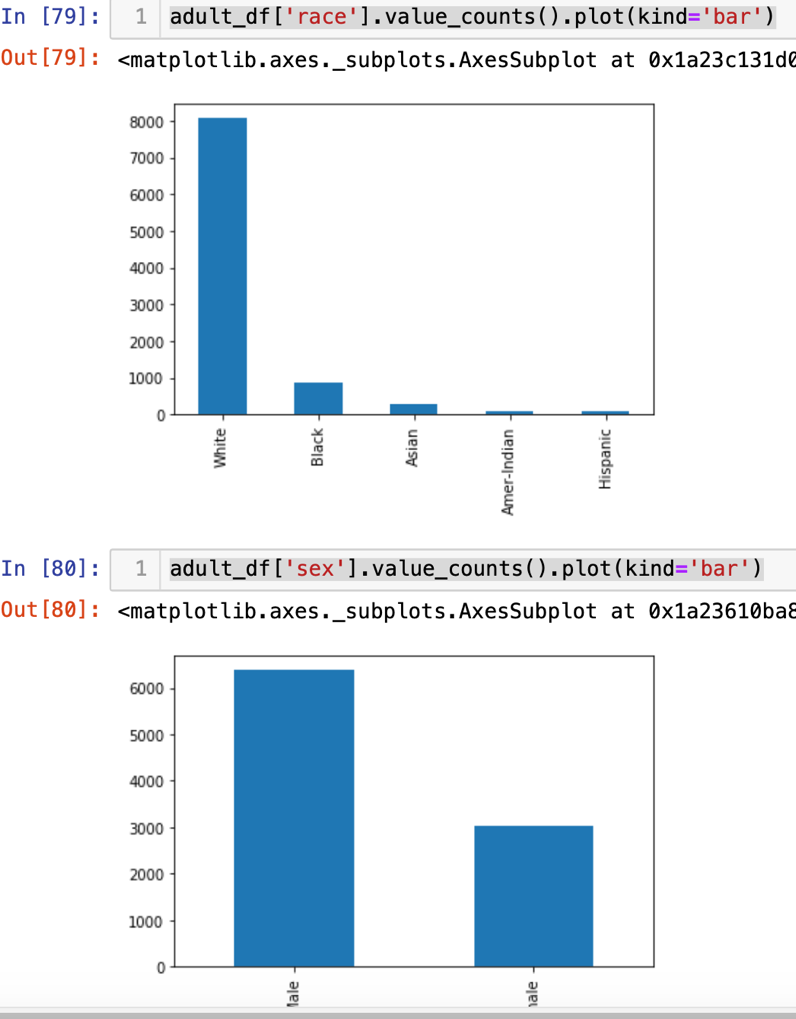
**

**



*Figure 3.05:*

**

**

*Figure 3.06:*

**Analysis:**

From above graphs, the numeric features of age tends to be right skew; education tends to be left skew, and hours-per-week close to normal distribution, but with high kurtosis.

For categorical variable, the number of workclass of private higher than others;

the number of both levels of marital-status stay at same levels;

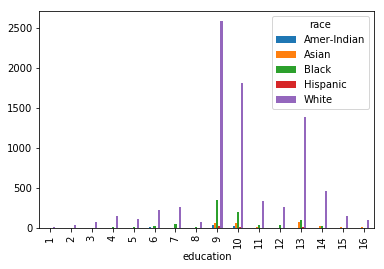
the number of White levels of race is significant higher than others;

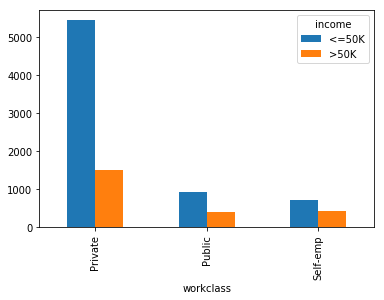
the number of Male levels of sex is significant higher than others;

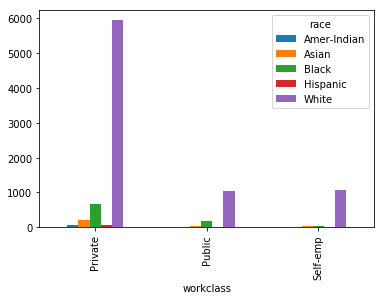
the number of <=50k levels of income is significant higher than others;

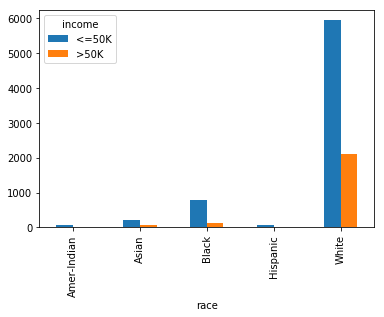
1. Perform the following cross-tabulations (including generating bar charts): education+race, work-class+income, work-class+race, and race+income. In the latter case (race+income) also create a table or chart showing percentages of each race category that fall in the low-income group. Discuss your observations from this analysis.

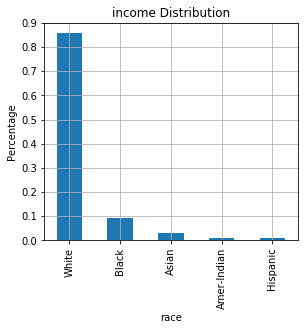
*Figure 3.06:*











**Analysis:**

From above graphs, we can summarize that in term of education + race, the white is significant higher than others;

in term of work-class+income, private class is significant lower than others;

in term of work-class+race, more white higher than others;

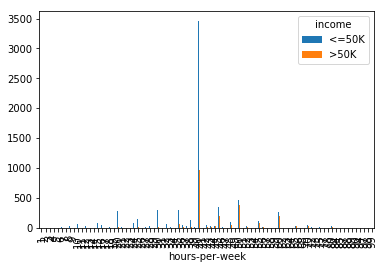
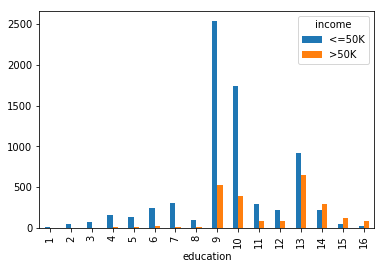
in term of race+income, both levels of income for White is significant higher than others;

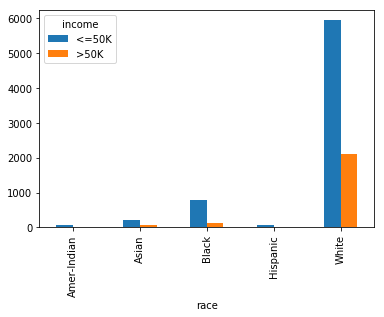
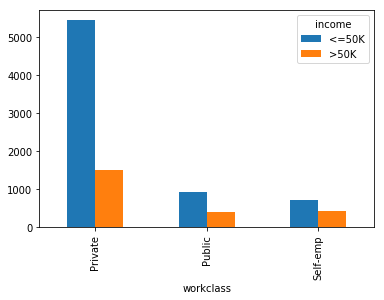
For percentages of each race category, the white has highest percentage than others;

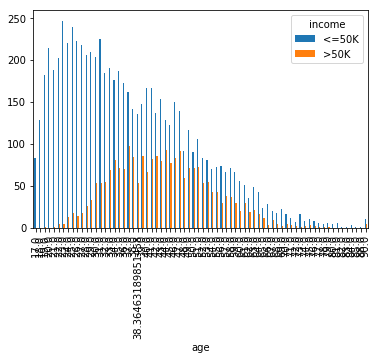
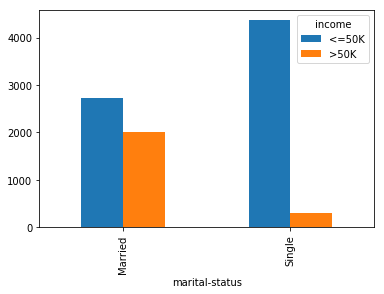
From my point of view, I believe this is because more white which data contains, so that white race more higher/lower that others.

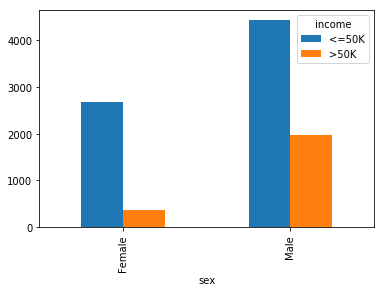
1. Compare and contrast the characteristics of the low-income and high-income categories across the different attributes.

*Figure 3.07:*



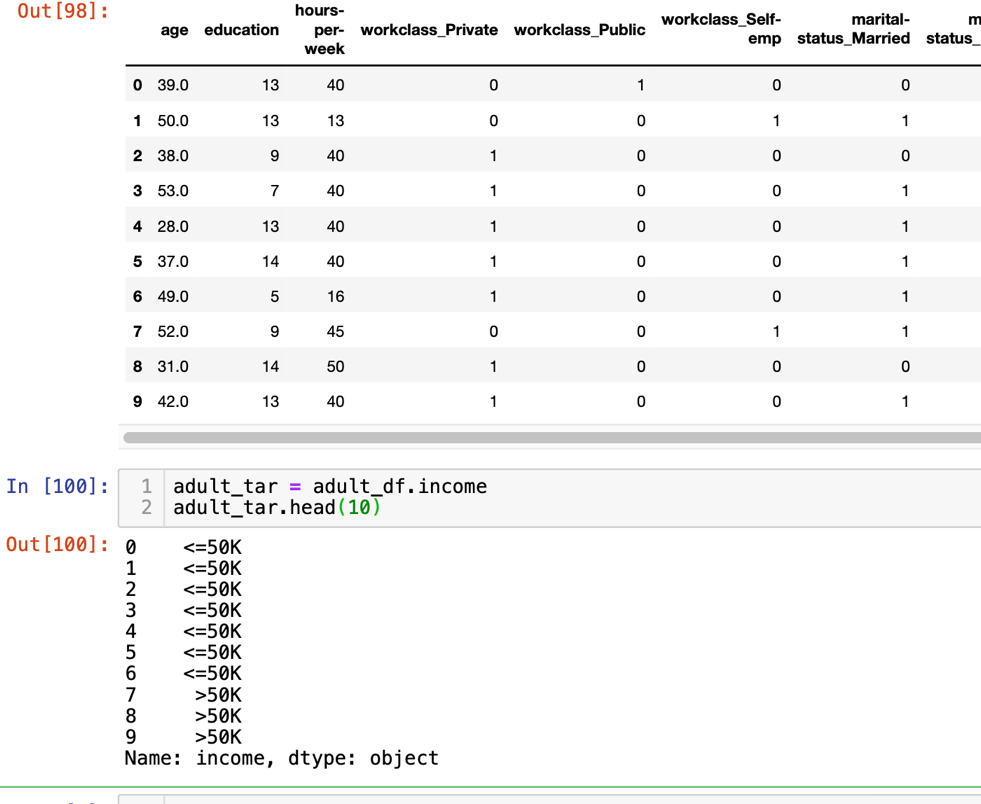




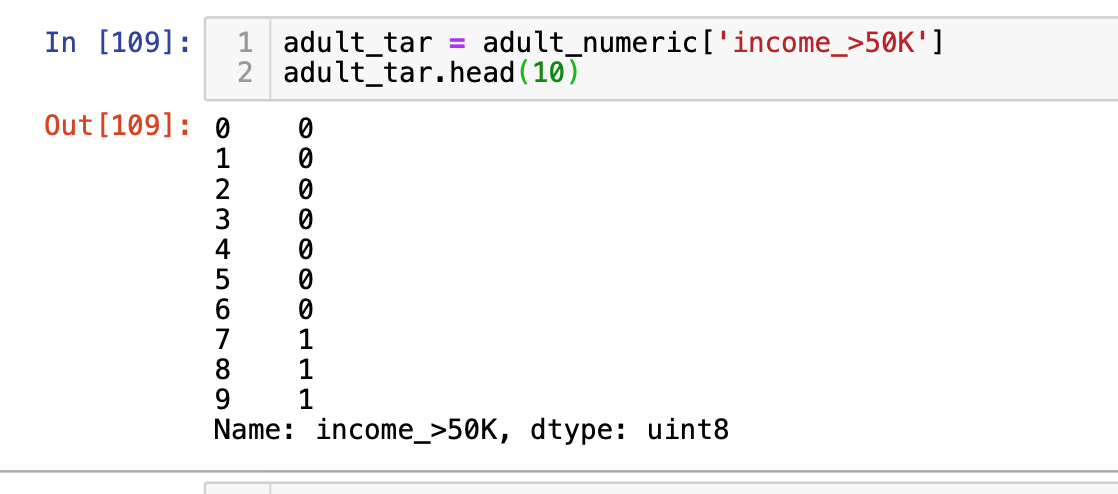


1. **Predictive Modeling and Model Evaluation:**
   1. Using either Pandas or Scikit-learn, create dummy variables for the categorical attributes. Then separate the target attribute ("income\_>50K") from the attributes used for training. [Note: you need to drop "income\_<=50K" which is also created as a dummy variable in earlier steps).

*Figure 3.08:*

**

*Figure 3.09:*

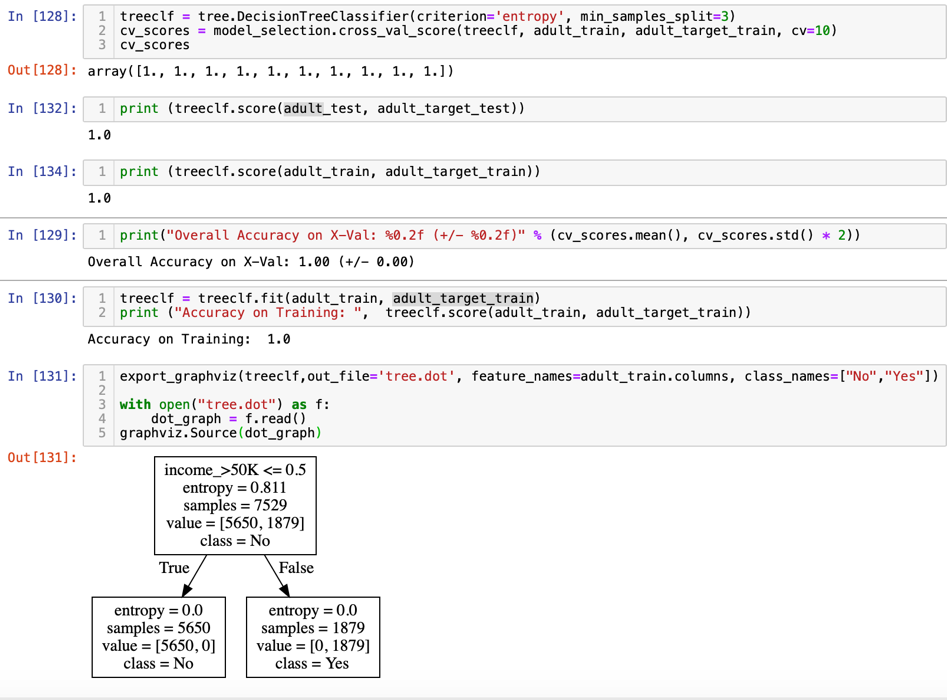
**

**Analysis:**

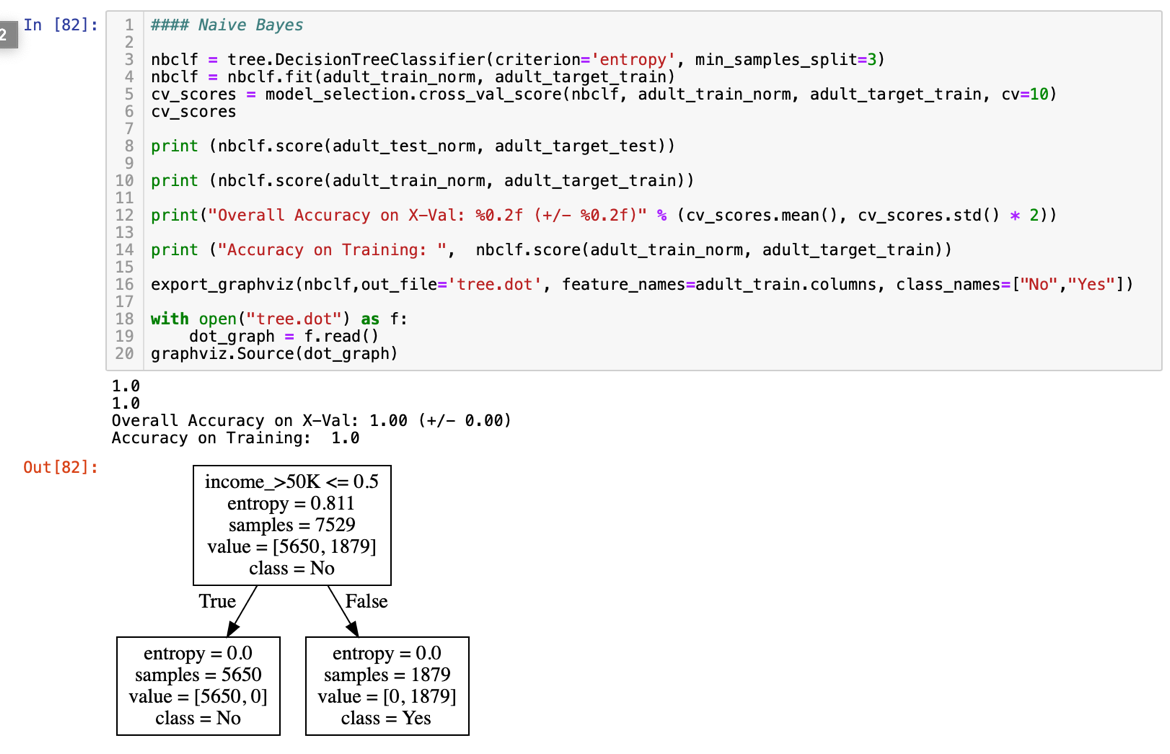
From second graphs, I slide attribute of ‘=> 50K’ from ‘income’ as my target;

* 1. Use scikit-learn to build classifiers uisng Naive Bayes (Gaussian), decision tree (using "entropy" as selection criteria), and linear discriminant analysis (LDA). For each of these perform 10-fold cross-validation (using cross-validation module in scikit-learn) and report the overall average accuracy.

*Figure 3.10:*

**

*Figure 3.11:*

**

*Figure 3.12:*

**

**Analysis:**

From above graphs, the figure 3.10 shows the result of decision tree, the figure 3.11 shows the result of Naïve Bayes, and the figure 3.11 shows the result of LDA.

In this case, we can summarize that both decision tree and Naïve Bayes facing overfitting problem, since the accuracy on train set is 1.00; on the other hand, the LDA perform better result, the accuracy score around 0.81

* 1. For the decision tree model (generated on the full training data), generate a visualization of tree and submit it as a separate file (png, jpg, or pdf) or embed it in the Jupyter Notebook.

*Figure 3.13:*

