The Use of a Naive Bayes Classifier on Social Services Data

Bruce Marron
Portland State University

Abstract

A Naive Bayes classifier (MultinomialNB) was applied to a typical social services dataset as part of the Machine Learning Pilot Project (MLPP) sponsored by the United Way of the Columbia-Willamette. The original dataset was provided by Metropolitan Family Service (MFS) and contained demographic data for 6587 MFS students enrolled in the Schools Uniting Neighborhoods (SUN) program sponsored by Multnomah County, Oregon. The data were managed in compliance with all provisions, chain of custody procedures, and privacy safeguards defined by the MLPP. After substantial pre-processing the data contained 6544 records each with a feature vector of 16 variates plus 1, binary class attribute (attendance >= 30-days = 1; otherwise = 0). The data were split 75:25 for training and testing. The overall performance of the classifier was poor as judged by standard performance measures of accuracy, precision, recall, F1 score, and ROC curve. The investigation suggests that portability of machine learning methodologies to individual social service organizations may be difficult because of the effort and expertise required for data pre-processing. The results suggest that at the first level of program component assessment, machine learning tools may provide discriminant power to determine which datasets are valuable for outcome prediction and which are not.

I. Introduction

Advances in machine learning algorithms have led to a rich selection of tools for tackling tough problems of inference (Alpaydin, 2014). A surprisingly simple and effective machine learning tool is the Naive Bayes classifier (Hand & Yu, 2001; Kotsiantis, 2007). Naive Bayes classifiers are supervised learning algorithms that apply Bayes' theorem with the "naive" assumption of independence between every pair of features. That is, Naive Bayes classifiers assume that each of the measurements in a feature vector is independent of every other measurement. Feature vectors are vectors of multivariate data that are associated with a perceived event outcome. Naive Bayes classifiers can be used with categorical data (e.g., ethnicity or school types), continuous data (e.g., age or time spent in program), or mixtures of both data types. Naive Bayes classifiers are fast, do not require much training data, and can be used as a baseline or first-step classifier.

This report documents the results of an investigation into the use of a Naive Bayes classifier on a typical social services dataset. As detailed in the Machine Learning Pilot Project (MLPP) proposal document, "Machine Learning Tools for Social Service Providers Funded by the United Way of the Columbia-Willamette," the purpose of this investigation is (1) to determine if simple machine learning tools will prove useful to individual social service organizations in helping them assess their own system of social service provisioning, and (2) to determine if the general methodology developed will be portable and applicable to the unique datasets maintained by different social service organizations.

II. Methods and Materials

A typical social services dataset was obtained from Metropolitan Family Service (MFS) in compliance with all provisions and using the chain of custody procedures and privacy safeguards detailed in the document, "Project Plan for the Machine Learning Pilot Project." A copy of the signed Data Request and Use Form is provided in the Appendix. The original dataset (hereafter known as, original-MFS-dataset1) contained data for 6587 MFS students enrolled in the Schools Uniting Neighborhoods (SUN) program sponsored by Multnomah County, Oregon. In 2016 there were 80 SUN Community Schools in 6 school districts across Multnomah County: 36 elementary, 19 middle, 16 K-8, and 9 high schools. As defined below, the single class attribute considered for this analysis was SUN program attendance.

A feature vector of 16 demographic variates and 1 class attribute on 6544 records were ultimately obtained from original-MFS-dataset1 after formatting and pre-processing. Note that rarely are data "clean" and ready for analysis; original-MFS-dataset1 was no exception. The details of formatting and pre-processing are provided in the document, LOG_FormatProcessing_MFS_dataset1.txt which is included in the Appendix. Formatting and pre-processing created a 6544 x 17 data matrix of "clean" data. These data are contained in the file, FormatProcessedLevel3_MFS_dataset1.csv. The "cleaned" data (all categorical data numerically codified, extraneous data fields removed, no missing values) are summarized in Table 1.

The single class attribute (target) considered for this analysis, SUN program attendance, was determined by evaluating records against a 30-day threshold. If the value in the attendance field of original-MFS-dataset1 was equal to or greater than 30, then the record received a "1" in the class attribute field (Field 17). Otherwise, the record received a "0" in Field 17. Of the 6544 records submitted for machine learning analysis, 2800 received class attribute "1" and 3744 received class attribute "0".

Table 1: *Field characteristics of the Level3 formatted and pre-processed Metropolitan Family Service (MFS) dataset.*

Field	Data Type	Values Range
School Name	Categorical	0 - 22
School District	Categorical	0 - 4
School Type	Categorical	0 - 2
Age	Continuous	6 - 22
Gender	Categorical	0 - 3
Language	Categorical	0 - 57
African	Binary	0 - 1
Asian	Binary	0 - 1
Black/ African American	Binary	0 - 1
Latino/Hispanic	Binary	0 - 1
Middle Eastern	Binary	0 - 1
Native American/ Alaska Native	Binary	0 - 1
Native Hawaiian/ Pacific Islander	Binary	0 - 1
Slavic	Binary	0 - 1
White	Binary	0 - 1
Declined	Binary	0 - 1
Days Attended (Class attribute)	Binary	0 - 1

The categorical data in FormatProcessedLevel3_MFS_dataset1.csv were transformed with a pre-processing algorithm called the OneHotEncoder[1] which is part of the Python-based ma-

[1] http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-categorical-features

chine learning library, *sci-kit learn*.[2]. Such a transformation is often necessary because the integer representation of categorical (discrete) features cannot be used directly with many of the scikit-learn estimators and classifiers. These classifiers expect either (1) continuous input thus interpreting the categories as being ordered, or (2) count input thus interpreting the categories as counts of a single discrete variate. The OneHotEncoder uses a one-of-K or one-hot encoding to convert each categorical feature with m possible values into m binary features with only one feature active. For example, records in Level3 data were initially coded with the integers 1, 2, or 3 for the field 'School Type' depending on whether the student attended a grade school, a middle school, or a high school, respectively. In place of a single column for school type, the OneHotEncoder generates three separate columns where each column corresponds to one possible value of one of the features. Thus, the record of a middle school student would be represented as a (0,1,0) vector. The transformation of categorical data with the OneHotEncoder generated a Level4 dataset (`FormatProcessedLevel4_MFS_dataset1.csv`). The process of transforming the Level3 dataset to the Level4 dataset is detailed in the module (Python script), `FormatProcessingLevel3-4_MFS_dataset1_OneHotEncoder_v2.py` which is provided in the Appendix.

The analysis of the data in `FormatProcessedLevel4_MFS_dataset1.csv` was performed with a Naive Bayes classifier called, "MultinomialNB" which also is part of the Python-based machine learning library, *sci-kit learn*."Multinomial" refers to the multinomial distribution, a multidimensional generalization of the binomial distribution. Multinomial means that instead of only two (binary) possible distinct outcomes for an event or trial, there are some number k of distinct outcomes. The multinomial distribution reduces to the binomial distribution if there are only two possible outcomes for an event or trial. "NB" refers to Naive Bayes. The multinomial Naive Bayes classifier is suitable for classification of k classes from n events or trials. MultinomialNB requires discrete features and (normally) integer feature counts for each trial. As noted above, the categorical data in the Level3 MFS dataset were transformed prior to MultinomialNB analysis into a Level4 dataset precisely because the integers represented categories, not counts. Additionally, the Level4 data were randomly split (75:25) into a training dataset (n = 4908 records) and a test dataset (n = 1636 records). The details of the final analytical procedure are documented in the module, `FormattedProcessedLevel4_MFS_dataset1_MNBayes.py` which is included in the Appendix.

All data formatting, pre-processing, analyses were performed with readily available hardware and freely available, open-source software. Hardware included a HP Compaq 6710b (32-bit) machine from the non-profit, "Free Geek" located in Portland, OR. The machine used a Linux-based operating system:

```
Distro: Ubuntu 14.04 trusty
Kernel: 3.16.0-77-generic i686 (32 bit, gcc: 4.8.4)
Desktop:  Xfce 4.11.8 (Gtk 2.24.23)
```

The Python-based suite, *sci-kit learn* is a set of 'simple and efficient tools for data mining and data analysis, accessible to everybody.' Python scripts for the investigation were written an executed in Spyder (Spyder 2.3.8), an integrated development environment (IDE) that is included as part of the Anaconda distribution (Anaconda 2.5.0) of Python (Python 2.7.12).

[2]`http://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes`

Table 2: *Confusion matrix of the MultinomialNB classifier applied to the test dataset. The test dataset was randomly selected from the Level4 formatted and pre-processed Metropolitan Family Service (MFS) dataset. The test dataset contained 1636 records.*

	1	0
1	463	273
0	373	527

III. Results

There are multiple performance measures for machine learning classifiers many of which are derived from the confusion matrix. The confusion matrix for the MultinomialNB classifier as applied to the test dataset is presented numerically in Table 2 and graphically in Figure 1. From this table, the number of true positives (TP) is 463, the number of true negatives (TN) is 527, the number of false negatives (FN) is 273, and the number of false positives (FP) is 373. Accuracy (TP + TN/Total scores), precision (TP/TP + FP), recall (TP/TP + FN), and F1 measures are given in Table 3. The F1-score is the harmonic mean of precision and recall with a range of [0,1]. The F-measure is another measure of accuracy and is suitable when (1) mistakes are considered equally bad, whether they are false positives or false negatives, and (2) the number of mistakes is measured relative to the number of true positives, neglecting true negatives. To have a high F1 score, values of high precision and high recall are required.

Another measure of performance is the receiver operator characteristic (ROC) curve. ROC curves typically plot true positive rate or recall (TP/TP + FN) on the Y-axis, and false positive rate (FP/FP + TN) on the X-axis. The top left corner of the plot is thus a "perfect point" with a false positive rate of '0' and a true positive rate of '1'. A general rubric for evaluating ROC curves, which uses the area under the ROC curve, is

- 1.0 = perfect prediction
- 0.9 = excellent prediction
- 0.8 = good prediction
- 0.7 = mediocre prediction
- 0.6 = poor prediction
- 0.5 = random prediction
- <0.5 = something is wrong!

The ROC curve for the MultinomialNB classifier as applied to the test dataset is displayed in Figure 2.

IV. Discussion

The MultinomialNB preformed poorly given the Level4 MFS dataset. Possible data-related reasons include (1) too many non-responses (22 for Field 5, 36 for Field 6, 133 for Field 16), (2) the segregation of languages into too many (58) categories, and perhaps most importantly, 3) the lack

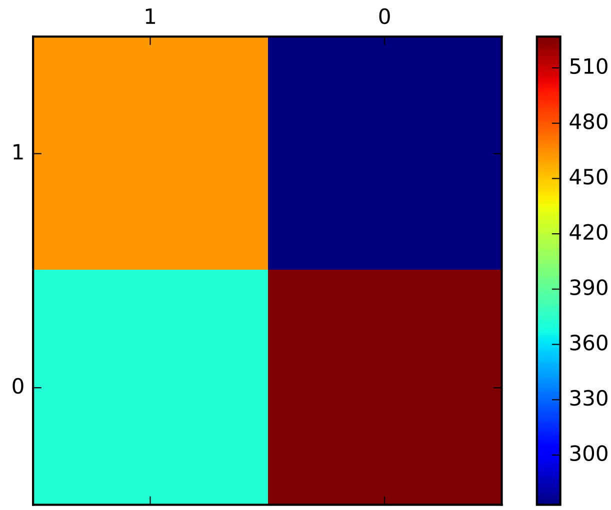

Figure 1: *Confusion matrix of the MultinomialNB classifier applied to the test dataset. The test dataset was randomly selected from the Level4 formatted and pre-processed Metropolitan Family Service (MFS) dataset. The test dataset contained 1636 records.*

Table 3: *Classification report of the MultinomialNB classifier applied to the test dataset. The test dataset was randomly selected from the Level4 formatted and pre-processed Metropolitan Family Service (MFS) dataset.*

Accuracy	Precision	Recall	F1 Score
0.61	0.61	0.61	0.61

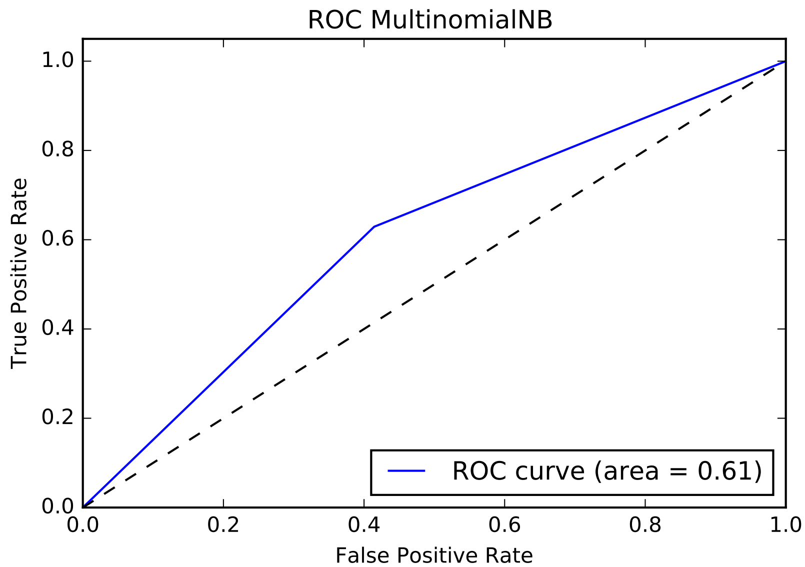

Figure 2: *Receiver operating characteristic (ROC) curve for the MultinomialNB classifier applied to the test dataset. The test dataset was randomly selected from the Level4 formatted and pre-processed Metropolitan Family Service (MFS) dataset.*

of truly predictive data. Certainly, the analysis could be repeated with modifications to handle non-responses and excessive categories and the performance is likely to improve somewhat. But this analysis may in fact be quite useful exactly because it is so poorly predictive. The results raise two complementary questions: Are the data fields which were provided actually poor predictors of success in the SUN program? And if so, What are the missing, truly predictive data fields? Additional sleuthing and analyses would be required to answer these questions.

The pre-processing of the original data required substantial effort and expertise. This suggests that the portability of the Naive Bayes machine learning methodology may be difficult regardless of the availability of pre-packaged Python scripts for actualizing the machine learning tools themselves. There are, however, options that could increase the adoption of machine learning methodologies. Two such options are first, to create and maintain a library of "data cleaning" scripts to allow for efficient data pre-processing from typical data streams, and second, to implement some level of data entry quality assurance/quality control in those social service organizations wanting to use machine learning tools.

Lastly, this investigation has found that machine learning tools may prove useful to individual social service organizations in surprising ways. At the first level of program component assessment, machine learning tools may provide discriminant power to determine which datasets are valuable for outcome prediction and which are not.

References

Alpaydin, E. (2014). *Introduction to machine learning* (Third edition ed.). Cambridge, Massachusetts: The MIT Press.

Hand, D. J., & Yu, K. (2001). Idiot's bayes - not so stupid after all? *International statistical review*, *69*(3), 385–398.

Kotsiantis, S. B. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, *31*, 249–268.

Appendix

Data Request and Use Form

Data-Holding Organization: Metropolitan Family Service (MFS)

Description of Data Requested (be specific):
SUN school programmatic and demographic data for students and families for school years 2014 and 2015.

Reason for Request:
Trial development of Naive Bayes classifier

Intended Use of the Dataset(s) (be specific):
To ascertain the viability of machine learning tools and actual social services data.

By accepting receipt of the dataset(s) described above, I expressly agree to be legally bound by all Terms and Conditions of Data Use as defined in the Machine Learning Pilot Project Project Plan document.

B Marron (Bruce Marron) 19 July 2016

Data User Date
(Printed Name and Signature)

Approval

Data-Holding Organization: METROPOLITAN FAMILY SERVICE

Description of Data Delivered: DEMOGRAPHIC AND ATTENDANCE DATA FOR MFS SUN STUDENTS

Mechanism and Date of Delivery: SECURE CSV FILE, 7.19.16

The Data User above is approved for the use of the delivered data for the period 7.19.16 through 12.31.16.

GEOFFREY BRUSCA 7.19.16

Organization Representative Date
(Printed Name and Signature)

```
Title:                  Data formatting and preprocessing of the original MFS dataset1 for input
                        to a Multinomial Naive Bayes classifier
Project Descriptor:     Consulting to United Way of Columbia Willamette  (STAT 570)
Project ID:             2016SoE013_STAT_570_Consulting
Record:
Author:                 bmarron
Origin Date:            18 Aug 2016
Final Date:             28 Aug 2016



##########################################
MFS dataset formatting and pre-processing
############################################

Rarely are data "clean" and ready for analysis. The following formatting and pre-processing steps prepare
the MFS daataset for Multinomial Bayes analysis.
The pre-processing uses Linux shell scripting commands and two, open-source text file editors, "sed" and
"awk." Data scientists typically keep data in
simple .txt or .csv files for cleaning before importing the correctly formatted data into R, Python, or
any other data analysis package.

===== I. Make a saved copy of the original dataset
=========================================================================

        a. Use Linux copy command to make backup of original (just in case!)
$ cp MFS_dataset1.csv original_MFS_dataset1.csv


==== II. Document the size of original dataset
=================================================================================

        a. Use Linux to perform a line count of the original data file
$ wc -l < original_MFS_dataset1.csv
6587


==== III. Delete unncessary data columns
=================================================================================

        a. Often data have unnecessary commas w/in fields. This will cause problems as a .csv data file
inputted into Python for machine learning.

        b. Delete unnecessary fields or fields lacking substantial data. Luckily, these are the fields
that contain the spurious commas.
        b1. Use LibreOffice Calc or Gnumeric spreadsheets to delete Fields 6 (Grade), 7 (Teacher), 9
(Inclusive Identity), and 10 (Person of Color?)


==== IV. Assign unique digits to nominal (categorical) data
=====================================================================

+++++++++++++
Field 2 ==>
School Name
++++++++++++++

        a. Use Linux to provide a list of unique instances of the nominal data in Field 2 (School Name)
        a1. The following Linux command string does the following:
                deletes the first line of the file, MFS_dataset1.csv using sed (Stream EDitor);
                pipes the comma delimited file to 'cut' to pull the strings of data in Field 2;
                pipes to 'sort' to alphabetize the srrings;
                pipes to 'unique' to keep one instance of each unique string and count the number of each
instance

$ sed '1d' MFS_dataset1.csv | cut -d',' -f2 | sort | uniq -c
```

```
    472 Cherry Park Elementary
    218 Clear Creek Middle
    866 David Douglas High
    251 Davis Elementary
    286 Earl Boyles Elementary
    269 Glenfair Elementary
    421 Gordon Russell Middle
    408 Gresham High
    231 Hartley Elementary
    293 H.B. Lee Middle
    263 Highland Elementary
    263 Lincoln Park Elementary
    214 Lynch Meadows Elementary
    190 Lynch View Elementary
    274 Lynch Wood Elementary
    292 McCarty Middle
    152 North Gresham Elementary
    241 Oliver Elementary
    214 Parklane Elementary
      7 Reynolds Middle
     89 Salish Ponds Elementary
    176 Shaver Elementary
    496 West Powellhurst Elementary


        b. Assign a unique digits (1-23) to each school name

Cherry Park Elementary          = 1
Clear Creek Middle              = 2
David Douglas High              = 3
Davis Elementary                = 4
Earl Boyles Elementary          = 5
Glenfair Elementary             = 6
Gordon Russell Middle           = 7
Gresham High                    = 8
Hartley Elementary              = 9
H.B. Lee Middle                 = 10
Highland Elementary             = 11
Lincoln Park Elementary         = 12
Lynch Meadows Elementary        = 13
Lynch View Elementary           = 14
Lynch Wood Elementary           = 15
McCarty Middle                  = 16
North Gresham Elementary        = 17
Oliver Elementary               = 18
Parklane Elementary             = 19
Reynolds Middle                 = 20
Salish Ponds Elementary         = 21
Shaver Elementary               = 22
West Powellhurst Elementary     = 23

        c. Use sed (Stream EDitor) to find and replace each school name with its assigned digit
        c1. Send output to a check file (test.txt) BEFORE changing the original file

$ sed "
s/Cherry Park Elementary/1/;
s/Clear Creek Middle/2/;
s/David Douglas High/3/;
s/Davis Elementary/4/;
s/Earl Boyles Elementary/5/;
s/Glenfair Elementary/6/;
s/Gordon Russell Middle/7/;
s/Gresham High/8/;
s/Hartley Elementary/9/;
s/H.B. Lee Middle/10/;
s/Highland Elementary/11/;
s/Lincoln Park Elementary/12/;
```

```
s/Lynch Meadows Elementary/13/;
s/Lynch View Elementary/14/;
s/Lynch Wood Elementary/15/;
s/McCarty Middle/16/;
s/North Gresham Elementary/17/;
s/Oliver Elementary/18/;
s/Parklane Elementary/19/;
s/Reynolds Middle/20/;
s/Salish Ponds Elementary/21/;
s/Shaver Elementary/22/;
s/West Powellhurst Elementary/23/
" MFS_dataset1.csv >> test.txt

        c2. change original file

$ sed -i "
s/Cherry Park Elementary/1/;
s/Clear Creek Middle/2/;
s/David Douglas High/3/;
s/Davis Elementary/4/;
s/Earl Boyles Elementary/5/;
s/Glenfair Elementary/6/;
s/Gordon Russell Middle/7/;
s/Gresham High/8/;
s/Hartley Elementary/9/;
s/H.B. Lee Middle/10/;
s/Highland Elementary/11/;
s/Lincoln Park Elementary/12/;
s/Lynch Meadows Elementary/13/;
s/Lynch View Elementary/14/;
s/Lynch Wood Elementary/15/;
s/McCarty Middle/16/;
s/North Gresham Elementary/17/;
s/Oliver Elementary/18/;
s/Parklane Elementary/19/;
s/Reynolds Middle/20/;
s/Salish Ponds Elementary/21/;
s/Shaver Elementary/22/;
s/West Powellhurst Elementary/23/
" MFS_dataset1.csv




+++++++++++++
Field 3 ==>
School District
++++++++++++++

        a. Use Linux to provide a list of unique instances of the nominal data in Field 3 (School
District)
        a1. The following Linux command string does the following:
                pipes the comma delimited file to 'cut' to pull the strings of data in Field 3;
                pipes to 'sort' to alphabetize the srrings;
                pipes to 'unique' to keep one instance of each unique string and count the number of each
instance

$ sed '1d' MFS_dataset1.csv | cut -d',' -f3 | sort | uniq -c

   1133 CSD
   2383 DDSD
   1754 GBSD
    176 PSD
   1140 RSD


        b. Assign a unique digits (1-5) to each school district
```

```
CSD     = 1
DDSD    = 2
GBSD    = 3
PSD     = 4
RSD     = 5

        c. Use sed (Stream EDitor) to find and replace each school district with its assigned digit
        c1. Send output to a check file (test.txt) BEFORE changing the original file

$ sed "
s/CSD/1/;
s/DDSD/2/;
s/GBSD/3/;
s/PSD/4/;
s/RSD/5/
" MFS_dataset1.csv >> test.txt


        c2. change original file

$ sed -i "
s/CSD/1/;
s/DDSD/2/;
s/GBSD/3/;
s/PSD/4/;
s/RSD/5/
" MFS_dataset1.csv


+++++++++++++
Field 4 ==>
School Type
++++++++++++++

        a. Use Linux to provide a list of unique instances of the nominal data in Field 4 (School Type)
        a1. The following Linux command string does the following:
                pipes the comma delimited file to 'cut' to pull the strings of data in Field 4;
                pipes to 'sort' to alphabetize the srrings;
                pipes to 'unique' to keep one instance of each unique string and count the number of each
instance

$ sed '1d' MFS_dataset1.csv | cut -d',' -f4 | sort | uniq -c
   4081 Elementary School
   1274 High School
    939 Middle School
    292 Middle School

        a2. Note there may be a keyboard entry error for the 'Middle School' assignment b/c multiple
'Middle School' instances are reported
        That is, there is some type of symbol difference between the  '939 Middle School' and the '292
Middle School' entries
        a3. Substitute "MiddleSchool" for 'Middle[...]School' and re-count

        a3i. The following Linux command string uses the 'awk' programming language and does the following
                notes that the File Separator and Output File Separator are both commas
                finds lines where the 4th field contains a staring "M" followed by "idd" and whatever else
                for those lines meeting the find criteria, does a global substitution in Field 4 of
"MiddleSchool" for anything in the field
                prints everything
                operates on the MFS_dataset1.csv file, sends the output to temp.csv, and then replace the
files

$ awk 'BEGIN{FS=OFS=","} $4 ~ /^M[idd]/ {gsub(/.*/,"MiddleSchool", $4)}; {print $0}' MFS_dataset1.csv >
tmp.csv && mv tmp.csv MFS_dataset1.csv

        a3ii.
```

```
$ sed '1d' MFS_dataset1.csv | cut -d',' -f4 | sort | uniq -c
   4081 Elementary School
   1274 High School
   1231 MiddleSchool


        b. Assign a unique digits (1-3) to each school type

Elementary School       = 1
High School             = 2
MiddleSchool            = 3


        c. Use sed (Stream EDitor) to find and replace each school type with its assigned digit
        c1. Send output to a check file (test.txt) BEFORE changing the original file

$ sed "
s/Elementary School/1/;
s/High School/2/;
s/MiddleSchool/3/
" MFS_dataset1.csv >> test.txt


        c2. change original file

$ sed -i "
s/Elementary School/1/;
s/High School/2/;
s/MiddleSchool/3/
" MFS_dataset1.csv


+++++++++++++
Field 6 ==>
Gender
++++++++++++++

        a. Use Linux to provide a list of unique instances of the nominal data in Field 6 (Gender)
        a1. The following Linux command string does the following:
                pipes the comma delimited file to 'cut' to pull the strings of data in Field 6;
                pipes to 'sort' to alphabetize the srrings;
                pipes to 'unique' to keep one instance of each unique string and count the number of each
instance

$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c

      24
      1 Client refused
     10 Data not collected
   3267 Female
   3283 Male
      1 Transgender female to male



        a2. Data missing and redundant. Let there be one category for No Data (empty, Client refused,
Data not collected) = 0
        a2i.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "") {$6 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
     24 0
      1 Client refused
     10 Data not collected
   3267 Female
   3283 Male
      1 Transgender female to male
```

```
        a2ii.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "Client refused") {$6 = "0"}}; {print $0}' MFS_dataset1.csv >
tmp.csv && mv tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
     25 0
     10 Data not collected
   3267 Female
   3283 Male
      1 Transgender female to male


        a2iii.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "Data not collected") {$6 = "0"}}; {print $0}' MFS_dataset1.csv >
tmp.csv && mv tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
     35 0
   3267 Female
   3283 Male
      1 Transgender female to male


        b. Assign a unique digits (0-3) gender

No data                = 0
Female                 = 1
Male                   = 2
Transgender            = 3


        b1i.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "Female") {$6 = "1"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
    35 0
   3267 1
   3283 Male
      1 Transgender female to male

        b1ii.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "Male") {$6 = "2"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
     35 0
   3267 1
   3283 2
      1 Transgender female to male

        b1iii.
$ awk 'BEGIN{FS=OFS=","} { if ($6 == "Transgender female to male") {$6 = "3"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ sed '1d' MFS_dataset1.csv | cut -d',' -f6 | sort | uniq -c
     35 0
   3267 1
   3283 2
      1 3


+++++++++++++
Field 7 ==>
Language
++++++++++++++

        a. Use Linux to provide a list of unique instances of the nominal data in Field 7 (Language)
        a1. The following Linux command string does the following:
                pipes the comma delimited file to 'cut' to pull the strings of data in Field 7;
                pipes to 'sort' to alphabetize the srrings;
```

```
                pipes to 'unique' to keep one instance of each unique string and count the number of each
instance

$ sed '1d' MFS_dataset1.csv | cut -d',' -f7 | sort | uniq -c

     47 
      1 Afan Oromo
      1 Albanian
     10 Amharic
     75 Arabic
      1 Armenian
      1 ASL
      4 Bosnian
     47 Burmese
      6 Cambodian
    108 Cantonese
      1 Chechen
      2 Chinese
      1 Chukese
     12 Chuukese
      1 Creole
      3 Dari
      1 Davi
      1 Declined to answer
   3814 English
      5 Farsi
      7 French
      4 Hindi
     22 Hmong
      3 Igbo
      2 Japanese
      1 Kachin
      1 karen
     61 Karen
      1 Kareni
      1 Khmer
      1 Kinyarwanda
      1 korean
      3 Korean
      1 Koren
      1 Kyinarwanda
      1 Laos
      8 Laotian
      2 Lisu
      2 Maay Maay
      1 Maay-Maay
      1 Malay
      2 Malaysian
      4 Mamy
     15 Mandarin
      1 Maymay
      2 Micronesian
      7 Mien
      2 Nepalese
     39 Nepali
      1 Nothing selected
      1 Omromo
      2 Oromo
      1 Other
      1 Perisa
      1 Persian
      1 Pingelapese
      1 Pinglapese
      4 Rhohingya
      1 Rohingya
     35 Romanian
    142 Russian
```

```
      4 Samoan
     97 Somali
   1668 Spanish
     10 Swahili
      8 Tagalog
      1 Tagalog (Filipino)
      1 Tagalong/Filipino
      1 Tagglog
      4 Thai
      8 Tibetan
      1 Tibetian
      1 Tigirigina
      1 Tigriga
      1 Tigrigna
      3 Tigrina
      3 Tigrinya
      1 Tongan
      1 TONGAN
      6 Turkish
     18 Ukrainian
      1 Uzbek
    185 Vietnamese
      1 "Yap
      1 Yappese/Palauan
      2 Yoruba
     27 Zomi
      1 Zotung


        b. Lots of variations and missing data!
        b1. Use Internet to check 'most likely' language and define. Then make changes.
    47                                  = No data       = 0
      1 Afan Oromo                      = Oromo         = 1
      1 Albanian                        = Albanian      = 2
     10 Amharic                         = Amharic       = 3
     75 Arabic                          = Arabic        = 4
      1 Armenian                        = Armenian      = 5
      1 ASL                             = ASL           = 6
      4 Bosnian                         = Bosnian       = 7
     47 Burmese                         = Burmese       = 8
      6 Cambodian                       = Cambodian     = 9
    108 Cantonese                       = Cantonese     = 10
      1 Chechen                         = Chechen       = 11
      2 Chinese                         = Cantonese     = 10
      1 Chukese                         = Chuukese      = 12
     12 Chuukese                        = Chuukese      = 12
      1 Creole                          = Creole        = 13
      3 Dari                            = Dari          = 14
      1 Davi                            = Dari          = 14
      1 Declined to answer              = No data       = 0
   3814 English                         = English       = 15
      5 Farsi                           = Farsi         = 16
      7 French                          = French        = 17
      4 Hindi                           = Hindi         = 18
     22 Hmong                           = Hmong         = 19
      3 Igbo                            = Igbo          = 20
      2 Japanese                        = Japanese      = 21
      1 Kachin                          = Kachin        = 22
      1 karen                           = Karen         = 23
     61 Karen                           = Karen         = 23
      1 Kareni                          = Karenni       = 24
      1 Khmer                           = Khmer         = 25
      1 Kinyarwanda                     = Kinyarwanda   = 26
      1 korean                          = Korean        = 27
      3 Korean                          = Korean        = 27
      1 Koren                           = Korean        = 27
      1 Kyinarwanda                     = Kinyarwanda   = 26
```

```
      1 Laos                          = Lao          = 28
      8 Laotian                       = Lao          = 28
      2 Lisu                          = Lisu         = 29
      2 Maay Maay                     = Maay         = 30
      1 Maay-Maay                     = Maay         = 30
      1 Malay                         = Malay        = 31
      2 Malaysian                     = Malay        = 31
      4 Mamy                          = Maymay       = 32
     15 Mandarin                      = Mandarin     = 33
      1 Maymay                        = Maymay       = 32
      2 Micronesian                   = Micronesian  = 34
      7 Mien                          = Mien         = 35
      2 Nepalese                      = Nepali       = 36
     39 Nepali                        = Nepali       = 36
      1 Nothing selected              = No data      = 0
      1 Omromo                        = Oromo        = 1
      2 Oromo                         = Oromo        = 1
      1 Other                         = No Data      = 0
      1 Perisa                        = Farsi        = 16
      1 Persian                       = Farsi        = 16
      1 Pingelapese                   = Pingelapese  = 37
      1 Pinglapese                    = Pingelapese  = 37
      4 Rhohingya                     = Rohingya     = 38
      1 Rohingya                      = Rohingya     = 38
     35 Romanian                      = Romanian     = 39
    142 Russian                       = Russian      = 40
      4 Samoan                        = Samoan       = 41
     97 Somali                        = Somali       = 42
   1668 Spanish                       = Spanish      = 43
     10 Swahili                       = Swahili      = 44
      8 Tagalog                       = Tagalog      = 45
      1 Tagalog (Filipino)            = Tagalog      = 45
      1 Tagalong/Filipino             = Tagalog      = 45
      1 Tagglog                       = Tagalog      = 45
      4 Thai                          = Thai         = 46
      8 Tibetan                       = Tibetan      = 47
      1 Tibetian                      = Tibetan      = 47
      1 Tigirigina                    = Tigrinya     = 48
      1 Tigriga                       = Tigrinya     = 48
      1 Tigrigna                      = Tigrinya     = 48
      3 Tigrina                       = Tigrinya     = 48
      3 Tigrinya                      = Tigrinya     = 48
      1 Tongan                        = Tongan       = 49
      1 TONGAN                        = Tongan       = 49
      6 Turkish                       = Turkish      = 50
     18 Ukrainian                     = Ukrainian    = 51
      1 Uzbek                         = Uzbek        = 52
    185 Vietnamese                    = Vietnamese   = 53
      1 "Yap                          = Yapese       = 54
      1 Yappese/Palauan               = Yapese       = 54
      2 Yoruba                        = Yoruba       = 55
     27 Zomi                          = Zou          = 56
      1 Zotung                        = Zotung       = 57

      b1i. Re-code the data file
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "" || $7 == "Other" || $7 == "Declined to answer" || $7 == "Nothing
selected") {$7 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Afan Oromo" || $7 == "Omromo"  || $7 == "Oromo") {$7 = "1"}};
{print $0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Albanian") {$7 = "2"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Amharic") {$7 = "3"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Arabic") {$7 = "4"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Armenian") {$7 = "5"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
```

```
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "ASL") {$7 = "6"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Bosnian") {$7 = "7"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Burmese") {$7 = "8"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Cambodian") {$7 = "9"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Cantonese" || $7 == "Chinese") {$7 = "10"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Chechen") {$7 = "11"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Chuukese" || $7 == "Chukese") {$7 = "12"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Creole") {$7 = "13"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Dari" || $7 == "Davi") {$7 = "14"}}; {print $0}' MFS_dataset1.csv >
tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "English") {$7 = "15"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Farsi" || $7 == "Perisa" || $7 == "Persian") {$7 = "16"}}; {print
$0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "French") {$7 = "17"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Hindi") {$7 = "18"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Hmong") {$7 = "19"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Igbo") {$7 = "20"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Japanese") {$7 = "21"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Kachin") {$7 = "22"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "karen" || $7 == "Karen") {$7 = "23"}}; {print $0}' MFS_dataset1.csv
> tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Kareni") {$7 = "24"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Khmer") {$7 = "25"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Kinyarwanda" || $7 == "Kyinarwanda") {$7 = "26"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Korean" || $7 == "korean"  || $7 == "Koren") {$7 = "27"}}; {print
$0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Laos" || $7 == "Laotian") {$7 = "28"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Lisu") {$7 = "29"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Maay-Maay" || $7 == "Maay Maay") {$7 = "30"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Malay" || $7 == "Malaysian") {$7 = "31"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Mamy" || $7 == "Maymay") {$7 = "32"}}; {print $0}' MFS_dataset1.csv
> tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Mandarin") {$7 = "33"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Micronesian") {$7 = "34"}}; {print $0}' MFS_dataset1.csv > tmp.csv
&& mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Mien") {$7 = "35"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
```

```
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Nepalese" || $7 == "Nepali") {$7 = "36"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Pingelapese" || $7 == "Pinglapese") {$7 = "37"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Rhohingya" || $7 == "Rohingya") {$7 = "38"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Romanian") {$7 = "39"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Russian") {$7 = "40"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Samoan") {$7 = "41"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Somali") {$7 = "42"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Spanish") {$7 = "43"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Swahili") {$7 = "44"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Tagalog" || $7 == "Tagalog (Filipino)" || $7 == "Tagalong/Filipino"
|| $7 == "Tagglog") {$7 = "45"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Thai") {$7 = "46"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Tibetan" || $7 == "Tibetian") {$7 = "47"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Tigirigina" || $7 == "Tigriga" || $7 == "Tigrigna" || $7 ==
"Tigrina" || $7 == "Tigrinya") {$7 = "48"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv tmp.csv
MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Tongan" || $7 == "TONGAN") {$7 = "49"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Turkish") {$7 = "50"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Ukrainian") {$7 = "51"}}; {print $0}' MFS_dataset1.csv > tmp.csv &&
mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Uzbek") {$7 = "52"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Vietnamese") {$7 = "53"}}; {print $0}' MFS_dataset1.csv > tmp.csv
&& mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "\"Yap" || $7 == "Yappese/Palauan") {$7 = "54"}}; {print $0}'
MFS_dataset1.csv > tmp.csv && mv tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Yoruba") {$7 = "55"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Zomi") {$7 = "56"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv
$ awk 'BEGIN{FS=OFS=","} { if ($7 == "Zotung") {$7 = "57"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
$ sed '1d' MFS_dataset1.csv | cut -d',' -f7 | sort | uniq -c

     50 0
      1 1
    110 10
      1 11
     13 12
      1 13
      4 14
   3814 15
      7 16
      7 17
      4 18
     22 19
      1 2
      3 20
```

```
      2 21
      1 22
     62 23
      1 24
      1 25
      2 26
      8 27
      9 28
      2 29
     10 3
      3 30
      3 31
      5 32
     15 33
      2 34
      7 35
     41 36
      2 37
      5 38
     35 39
     75 4
    142 40
      4 41
     97 42
   1668 43
     10 44
     11 45
      4 46
      9 47
      9 48
      2 49
      1 5
      6 50
     18 51
      1 52
    185 53
      2 54
      2 55
     27 56
      1 57
      1 6
      4 7
     47 8
      6 9
```

```
==== V. Transfer target data field and create binary target data
=====================================================================

        a. For ease of machine learning programming, move the target data field (Days Attended) to be the
LAST field of data (Field 18).
        a1. Use LibreOffice Calc or any other spreadsheet program to make the change.

        b. check number of counts above and below the MFS threshold of 30-day attendance
$ awk 'BEGIN {FS =","} $18>=30 {count++}; END {print count}' MFS_dataset1.csv
2805

$ awk 'BEGIN {FS =","} $18<30 {count++}; END {print count}' MFS_dataset1.csv
3782

        c. create binary target data in Field 18
$ awk 'BEGIN {FS=OFS=","} {if ($18 >=30 ) $18 = "1"; else $18 = "0"} {print $0}' MFS_dataset1.csv >
tmp.csv && mv tmp.csv MFS_dataset1.csv


        d. check contents of Field 18
$ cut -d',' -f18 MFS_dataset1.csv | sort | uniq -c
   3782 0
```

```
   2805 1


==== VI. Fill in missing binary data in Fields 8 - 17
=====================================================================

+++++++++++++
Field 8 ==>
African
++++++++++++++

        a. check contents of Field 8
$ cut -d',' -f8 MFS_dataset1.csv | sort | uniq -c
   6376
    210 1
      1 African

        b. make the changes
$ awk 'BEGIN {FS=OFS=","} { if ($8 == "") {$8 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
$ cut -d',' -f8 MFS_dataset1.csv | sort | uniq -c
   6376 0
    210 1
      1 African


+++++++++++++
Field 9 ==>
Asian
++++++++++++++

        a. check contents of Field 9
$ cut -d',' -f9 MFS_dataset1.csv | sort | uniq -c
   5724
    862 1
      1 Asian

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($9 == "") {$9 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
$ cut -d',' -f9 MFS_dataset1.csv | sort | uniq -c
   5724 0
    862 1
      1 Asian


+++++++++++++
Field 10 ==>
Black/African American
++++++++++++++

        a. check contents of Field 10
$ cut -d',' -f10 MFS_dataset1.csv | sort | uniq -c
   5768
    818 1
      1 Black/ African American


        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($10 == "") {$10 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
```

```
$ cut -d',' -f10 MFS_dataset1.csv | sort | uniq -c
    5768 0
    818 1
      1 Black/ African American


+++++++++++++
Field 11 ==>
Latino/Hispanic
++++++++++++++

        a. check contents of Field 11
$ cut -d',' -f11 MFS_dataset1.csv | sort | uniq -c
   4224
   2362 1
      1 Latino/Hispanic

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($11 == "") {$11 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
$ cut -d',' -f11 MFS_dataset1.csv | sort | uniq -c
    4224 0
   2362 1
      1 Latino/Hispanic


+++++++++++++
Field 12 ==>
Middle Eastern
++++++++++++++

        a. check contents of Field 12 before and after changes
$ cut -d',' -f12 MFS_dataset1.csv | sort | uniq -c
   6515
     71 1
      1 Middle Eastern


        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($12 == "") {$12 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   6515 0
     71 1
      1 Middle Eastern


+++++++++++++
Field 13 ==>
Native American/Alaska Native
++++++++++++++

        a. check contents of Field 13 before and after changes
$ cut -d',' -f13 MFS_dataset1.csv | sort | uniq -c
   6382
    204 1
      1 Native American/ Alaska Native

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($13 == "") {$13 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   6382 0
```

```
    204 1
      1 Native American/ Alaska Native

+++++++++++++
Field 14 ==>
Native Hawaiian/Pacific Islander
++++++++++++++

        a. check contents of Field 14 before and after changes
$ cut -d',' -f14 MFS_dataset1.csv | sort | uniq -c
   6432
    154 1
      1 Native Hawaiian/ Pacific Islander

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($14 == "") {$14 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   6432 0
    154 1
      1 Native Hawaiian/ Pacific Islander

+++++++++++++
Field 15 ==>
Slavic
++++++++++++++

        a. check contents of Field 15 before and after changes
$ cut -d',' -f15 MFS_dataset1.csv | sort | uniq -c
   6474
    112 1
      1 Slavic

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($15 == "") {$15 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   6474 0
    112 1
      1 Slavic

+++++++++++++
Field 16 ==>
White
++++++++++++++

        a. check contents of Field 16 before and after changes
$ cut -d',' -f16 MFS_dataset1.csv | sort | uniq -c
   4101
   2485 1
      1 White

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($16 == "") {$16 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   4101 0
   2485 1
      1 White

+++++++++++++
Field 17 ==>
Declined
++++++++++++++
```

```
        a. check contents of Field 17 before and after changes
$ cut -d',' -f17 MFS_dataset1.csv | sort | uniq -c
   6453
    133 1
      1 Declined

        b. make the changes
$ awk 'BEGIN{FS=OFS=","} { if ($17 == "") {$17 = "0"}}; {print $0}' MFS_dataset1.csv > tmp.csv && mv
tmp.csv MFS_dataset1.csv

        c. check
   6453 0
    133 1
      1 Declined


==== VII. Delete first data column or Field 1 (SvcPt)
=================================================================================

        a. This data column was maintained during dataset formatting and pre-processing as a reference
anchor for dataset changes.
        Ultimately these data are unnecessary.
        a1. Use LibreOffice Calc or Gnumeric spreadsheets to delete Field 1 (SvcPt)
        a2. Note that all fields are now shifted by one column. Field 1 is now "School Name"


==== VIII. Delete massively incomplete records
=========================================================================================

        a. Records 6584, 6585, 6586, and 6587 are missing data for Age, Gender, Language, Ethnicity, and
Attendance.
        These are the last four records in the dataset corresponding to SvcPt numbers 719126, 719008,
719126, 719008.
        a1. Delete these four records using LibreOffice.


==== IX. Final Field Names (Column Headers) Level 2 data
==================================================================================

        a. The final version of the formatted and preprocessed dataset has the following column headers

Field 1         School Name
Field 2         District
Field 3         ES/MS/HS
Field 4         Age
Field 5         Gender
Field 6         Language
Field 7         African
Field 8         Asian
Field 9         Black/ African American
Field 10        Latino/Hispanic
Field 11        Middle Eastern
Field 12        Native American/ Alaska Native
Field 13        Native Hawaiian/ Pacific Islander
Field 14        Slavic
Field 15        White
Field 16        Declined
Field 17        Days Attended

        b. The column headers must be deleted for MN Bayes analysis
        b1. Delete the column headers using LibreOffice
        b2. Rename formatted and pre-processed datast == FormatProcessed_MFS_dataset1.csv

        c. check that all records have 17 fields
$ awk 'BEGIN {FS =","} END {print FNR}' FormatProcessed_MFS_dataset1.csv
6582
```

```
$ awk 'BEGIN {FS =","} NF = 17 {count++} END {print count}' FormatProcessed_MFS_dataset1.csv
6582

$ awk 'BEGIN {FS =","} NF != 17 {count++} END {print count}' FormatProcessed_MFS_dataset1.csv

        d. check contents of Field 17
$ cut -d',' -f17 FormatProcessed_MFS_dataset1.csv | sort | uniq -c
   3778 0
   2804 1


==== X. Formatted and pre-processed dataset (Level 2) ready for Python testing
===================================================================

The comma-separated dataset is now formatted and pre-processed correctly for
Dataset Error Check using in Python.

1. Number of Instances
        FormatProcessed_MFS_dataset1.csv        6582

2. Number of Attributes
        17 input + 1 class attribute

3. For Each Attribute:
        All input attributes are integers in the range 0...57
        The last attribute is the class code 0,1

4. Missing Attribute Values
        None (??)

5. Class Distribution
        Class:  No of examples
        0:  3778
        1:  2804


==== XI.  Feedback from Python testing
=================================================================================

The data in "FormatProcessedLevel2_MFS_dataset1.csv" indicated a problem:
ValueError: Input contains NaN, infinity or a value too large for dtype('float64')

>>> np.where(np.isnan(full_MFS_dataset1)
(array([ 696, 1432, 1664, 1924, 2221, 2363, 2370, 3272, 3408, 3820, 4013, 4145, 4789, 5096, 5309]),
 array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]))
Appears that there are problems in Field 4


        a. check if any entries in Field 4 are blank
$ awk 'BEGIN {FS =","} {if ($4 =="") print $0}' FormatProcessed_MFS_dataset1.csv
2,3,3,,2,0,0,0,0,0,0,0,0,0,0,0,0
3,2,2,,0,0,0,0,0,0,0,0,0,0,0,0,0
3,2,2,,0,0,0,0,0,0,0,0,0,0,0,0,0
4,5,1,,2,4,0,0,0,0,1,0,0,0,0,0,0
4,5,1,,2,43,0,0,0,0,0,0,0,0,0,0,0
4,5,1,,2,15,0,0,0,0,0,0,0,0,0,0,0
4,5,1,,1,15,0,0,0,0,0,0,0,0,1,0,0
10,5,3,,0,0,0,0,0,0,0,0,0,0,0,0,0
12,2,1,,2,43,0,0,0,1,0,0,0,0,0,0,0
12,2,1,,0,0,0,0,0,0,0,0,0,0,0,0,0
13,1,1,,0,0,0,0,0,0,0,0,0,0,0,0,0
19,1,1,,0,0,0,0,0,0,0,0,0,0,0,0,0
19,1,1,,0,0,0,0,0,0,0,0,0,0,0,0,0
22,4,1,,0,0,0,0,0,0,0,0,0,0,0,0,0
22,4,1,,0,15,0,0,0,0,0,0,0,0,1,0,0
```

```
        b. check if ANY fields have blank entries
$ awk 'BEGIN {FS =","} {if ($1 =="" || $2 =="" || $3 =="" || $4 =="" || $5 =="" || $6 =="" || $7 =="" ||
$8 =="" || $9 =="" || $10 =="" || $11 =="" || $12 =="" || $13 =="" || $14 =="" || $15 =="" || $16 =="" ||
$17 =="") print NR}' FormatProcessed_MFS_dataset1.csv
690
1555
1556
1804
1805
1806
1807
3484
4240
4241
4455
5817
5818
6089
6090

        c. Delete these records
        c1. Delete these 15 records using LibreOffice.

        d. check if ANY fields have blank entries
$ awk 'BEGIN {FS =","} {if ($1 =="" || $2 =="" || $3 =="" || $4 =="" || $5 =="" || $6 =="" || $7 =="" ||
$8 =="" || $9 =="" || $10 =="" || $11 =="" || $12 =="" || $13 =="" || $14 =="" || $15 =="" || $16 =="" ||
$17 =="") print NR}' FormatProcessed_MFS_dataset1.csv


        e. check that all records have 17 fields
$ awk 'BEGIN {FS =","} END {print FNR}' FormatProcessed_MFS_dataset1.csv
6567

$ awk 'BEGIN {FS =","} NF = 17 {count++} END {print count}' FormatProcessed_MFS_dataset1.csv
6567

$ awk 'BEGIN {FS =","} NF != 17 {count++} END {print count}' FormatProcessed_MFS_dataset1.csv

        f. check contents of Field 17
$ cut -d',' -f17 FormatProcessed_MFS_dataset1.csv | sort | uniq -c
   3763 0
   2804 1


==== XII. Additional checks on Field 4 (Age) per Python testing
==============================================

        a. check min and max ages
min value:
$ cut -f4 -d"," FormatProcessed_MFS_dataset1.csv | sort -n | head -1
1

max value:
$ cut -f4 -d"," FormatProcessed_MFS_dataset1.csv | sort -n | tail -1
22

        b. Ages less than 6 seem problematic; ages greater than 18 may be acceptable
        b1. check if any entries in Field 4 are less than 6
$ awk 'BEGIN {FS =","} {if ($4 < 6) print $0}' FormatProcessed_MFS_dataset1.csv
2,3,3,5,1,15,0,0,0,0,0,0,0,0,0,0,1
3,2,2,2,1,43,0,0,0,1,0,0,0,0,0,0,0
3,2,2,2,1,43,0,0,0,1,0,0,0,0,0,0,0
3,2,2,1,2,43,0,0,0,1,0,0,0,0,0,0,0
5,2,1,3,2,43,0,0,0,1,0,0,0,0,0,0,0
5,2,1,2,1,15,0,0,0,1,0,0,0,0,1,0,0
6,5,1,5,2,15,0,0,0,0,0,0,0,0,1,0,0
6,5,1,3,2,15,0,0,0,0,0,1,0,0,0,0,0
```

```
6,5,1,3,1,15,0,0,0,0,0,0,0,0,1,0,0
6,5,1,1,2,15,0,0,0,0,0,1,0,0,0,0,0
6,5,1,1,2,15,0,0,0,0,0,1,0,0,0,0,0
13,1,1,4,1,0,0,0,0,1,0,0,0,0,0,0,0
13,1,1,2,2,18,0,1,0,0,0,0,0,0,0,0,0
14,1,1,5,2,15,0,0,0,1,0,0,0,0,0,0,0
14,1,1,5,2,43,0,0,0,1,0,0,0,0,0,0,0
14,1,1,5,1,43,0,0,0,1,0,0,0,0,0,0,0
14,1,1,5,2,15,0,0,0,0,0,0,0,0,1,0,0
14,1,1,5,1,15,0,0,0,0,0,0,0,0,1,0,0
14,1,1,3,1,43,0,0,0,1,0,0,0,0,0,0,0
17,3,1,1,2,43,0,0,0,1,0,0,0,0,0,0,1
18,1,1,1,2,43,0,0,0,1,0,0,0,0,0,0,1
21,5,1,1,2,15,0,0,0,0,0,0,0,0,1,0,0
21,5,1,1,1,15,0,0,1,0,0,0,0,0,0,0,1

        b2. record numbers for entries in Field 4 less than 6
$ awk 'BEGIN {FS =","} {if ($4 < 6) print NR}' FormatProcessed_MFS_dataset1.csv

689
1551
1552
1553
2085
2086
2351
2352
2353
2354
2355
4443
4444
4629
4630
4631
4632
4633
4634
5352
5593
5900
5901


        c. Delete these records
        c1. Delete these 23 records using LibreOffice.

        d. check number of records after deletions
$ awk 'BEGIN {FS =","} END {print FNR}' FormatProcessed_MFS_dataset1.csv
6544

        f. check contents of Field 17
$ cut -d',' -f17 FormatProcessed_MFS_dataset1.csv | sort | uniq -c
   3744 0
   2800 1


==== XII. Formatted and pre-processed dataset Level 3
========================================================================

The comma-separated dataset is now formatted and pre-processed correctly for
sklearn.preprocessing.OneHotEncoder to re-code categorical data for input to
sklearn.naive_bayes.MultinomialNB in Python.

FormatProcessedLevel3_MFS_dataset1.csv

1. Number of Instances
```

```
        FormatProcessed_MFS_dataset1.csv        6544

2. Number of Attributes
        17 input + 1 class attribute

3. For Each Attribute:
        All input attributes are integers in the range 0...57
        The last attribute is the class code 0,1

4. Missing Attribute Values
        None

5. Class Distribution
        Class:  No of examples
        0:  3744
        1:  2800

==== XIII. Transforming the Level 3 dataset to the Level 4 dataset
======================================================================
http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-categorical-features
http://scikit-learn.org/stable/modules/generated/
sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder

Often features are not given as continuous values but categorical. For example a person could have
features
Field 1 == ["male", "female"]
Field 2 == ["from Europe", "from US", "from Asia"]
Field 3 == ["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]
Such features can be efficiently coded as integers. For instance,
["male", "from US", "uses Internet Explorer"] could be expressed as ==> [0, 1, 3]
["female", "from Asia", "uses Chrome"] could be expressed as ==>  [1, 2, 1].

Such integer representation can not be used directly with scikit-learn estimators, as these expect either
1) continuous input (and would interpret
the categories as being ordered), or 2) count input (and would interpret the categories as counts of a
single discrete variate). One possibility
to convert categorical features to features that can be used with scikit-learn estimators is to use a one-
of-K or one-hot encoding, which is implemented
in OneHotEncoder. This transforms each categorical feature with 'm' possible values into 'm' binary
features, with only one active.


        a. Identify the fields with categorical and *continuous data

Field 1         School Name
Field 2         District
Field 3         ES/MS/HS
*Field 4        Age
Field 5         Gender
Field 6         Language
Field 7         African
Field 8         Asian
Field 9         Black/ African American
Field 10        Latino/Hispanic
Field 11        Middle Eastern
Field 12        Native American/ Alaska Native
Field 13        Native Hawaiian/ Pacific Islander
Field 14        Slavic
Field 15        White
Field 16        Declined
* Field 17      Days Attended (class attribute / target data)


        b. Process the Level 3 data to Level 4 data
        b1. see "FormatProcessingLevel3-4_MFS_dataset1_OneHotEncoder.py" for processing details
```

```
==== XIV. Formatted and pre-processed dataset Level 4
==========================================================================

The comma-separated dataset is now formatted and pre-processed correctly for
input to sklearn.naive_bayes.MultinomialNB in Python.


FormatProcessedLevel4_MFS_dataset1.csv

1. Number of Instances
        FormatProcessed_MFS_dataset1.csv        6544

2. Number of Attributes
        114 input + 1 class attribute

3. For Each Attribute:
        All input attributes are integers in the range 0,1
        The last attribute is the class code 0,1

4. Missing Attribute Values
        None

5. Class Distribution
        Class:  No of examples
        0:  3744
        1:  2800
```

```
Title:                  Data formatting and preprocessing of the original MFS dataset1 for input
                        to a Multinomial Naive Bayes classifier
Project Descriptor:     Consulting to United Way of Columbia Willamette  (STAT 570)
Project ID:             2016SoE013_STAT_570_Consulting
Record:
Author:                 bmarron
Origin Date:            29 Aug 2016
Final Date:             29 Aug 2016



##########################################
Possible glitch with OneHotEncoder
############################################
http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

Concerned that OneHotEncoder's addition of an additional field for
Field 1         School Name
Field 2         District
Field 3         ES/MS/HS

may decrease MultinomialNB effectiveness ==> "It is assumed that input features take on values in the
range [0, n_values)."



==== I. Re-code Fields 1, 2, and 3 starting from 0 NOT 1 for submittal to the OneHotEncoder
=====================

$ cd /home/bmarron/Desktop/PSU/PhD_EES/2016SoE013_STAT_570_Consulting/Works_InProgress/
UWCWDataset1_MNBayes_20160818/Data/FormattedProcessed_Data/

++++++
Field 1
++++++++
$ cut -d',' -f1 FormatProcessedLevel3_MFS_dataset1a.csv | sort | uniq -c

   472 1
    292 10
    263 11
    261 12
    211 13
    184 14
    274 15
    292 16
    151 17
    240 18
    212 19
    216 2
      7 20
     87 21
    174 22
    492 23
    861 3
    247 4
    284 5
    264 6
    421 7
    408 8
    231 9

$ cd /home/bmarron/Desktop/PSU/PhD_EES/2016SoE013_STAT_570_Consulting/Works_InProgress/
UWCWDataset1_MNBayes_20160818/Data/FormattedProcessed_Data/
$ awk 'BEGIN{FS=OFS=","} {if ($1 == "1") {$1 = "0"}} {print $0}' FormatProcessedLevel3_MFS_dataset1a.csv
> FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "2") {$1 = "1"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
```

```
awk 'BEGIN{FS=OFS=","} {if ($1 == "3") {$1 = "2"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "4") {$1 = "3"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "5") {$1 = "4"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "6") {$1 = "5"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "7") {$1 = "6"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "8") {$1 = "7"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "9") {$1 = "8"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "10") {$1 = "9"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv

awk 'BEGIN{FS=OFS=","} {if ($1 == "11") {$1 = "10"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "12") {$1 = "11"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "13") {$1 = "12"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "14") {$1 = "13"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "15") {$1 = "14"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "16") {$1 = "15"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "17") {$1 = "16"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "18") {$1 = "17"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "19") {$1 = "18"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "20") {$1 = "19"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "21") {$1 = "20"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "22") {$1 = "21"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv
awk 'BEGIN{FS=OFS=","} {if ($1 == "23") {$1 = "22"}} {print $0}' FormatProcessedLevel3_MFS_dataset1b.csv
> tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv



$ cut -d',' -f1 FormatProcessedLevel3_MFS_dataset1b.csv | sort | uniq -c

    472 0
    216 1
    263 10
    261 11
    211 12
    184 13
    274 14
    292 15
    151 16
    240 17
    212 18
      7 19
    861 2
     87 20
    174 21
    492 22
    247 3
    284 4
    264 5
```

```
    421 6
    408 7
    231 8
    292 9

+++++++
Field 2
++++++++

$ cut -d',' -f2 FormatProcessedLevel3_MFS_dataset1a.csv | sort | uniq -c

   1121 1
   2370 2
   1751 3
    174 4
   1128 5

$ awk 'BEGIN{FS=OFS=","} {if ($2 == "1") {$2 = "0"}} {print $0}' FormatProcessedLevel3_MFS_dataset1a.csv
> FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($2 == "2") {$2 = "1"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($2 == "3") {$2 = "2"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($2 == "4") {$2 = "3"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($2 == "5") {$2 = "4"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv

$ cut -d',' -f2 FormatProcessedLevel3_MFS_dataset1c.csv | sort | uniq -c

   1121 0
   2370 1
   1751 2
    174 3
   1128 4


$ awk 'BEGIN{FS=OFS=","} FNR==NR{a[NR]=$2;next}{$2=a[FNR]}1' FormatProcessedLevel3_MFS_dataset1c.csv
FormatProcessedLevel3_MFS_dataset1b.csv > tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv

$ cut -d',' -f2 FormatProcessedLevel3_MFS_dataset1b.csv | sort | uniq -c

   1121 0
   2370 1
   1751 2
    174 3
   1128 4







+++++++
Field 3
+++++++

$ cut -d',' -f3 FormatProcessedLevel3_MFS_dataset1a.csv | sort | uniq -c

   4047 1
   1269 2
   1228 3

$ awk 'BEGIN{FS=OFS=","} {if ($3 == "1") {$3 = "0"}} {print $0}' FormatProcessedLevel3_MFS_dataset1a.csv
> FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($3 == "2") {$3 = "1"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
```

```
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($3 == "3") {$3 = "2"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv
awk 'BEGIN{FS=OFS=","} {if ($3 == "4") {$3 = "3"}} {print $0}' FormatProcessedLevel3_MFS_dataset1c.csv >
tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1c.csv


$ cut -d',' -f3 FormatProcessedLevel3_MFS_dataset1c.csv | sort | uniq -c

   4047 0
   1269 1
   1228 2



$ awk 'BEGIN{FS=OFS=","} FNR==NR{a[NR]=$3;next}{$3=a[FNR]}1' FormatProcessedLevel3_MFS_dataset1c.csv
FormatProcessedLevel3_MFS_dataset1b.csv > tmp.csv && mv tmp.csv FormatProcessedLevel3_MFS_dataset1b.csv

$ cut -d',' -f3 FormatProcessedLevel3_MFS_dataset1b.csv | sort | uniq -c

   4047 0
   1269 1
   1228 2


+++++++
Field 5
+++++++

        a. No changes needed
$ cd /home/bmarron/Desktop/PSU/PhD_EES/2016SoE013_STAT_570_Consulting/Works_InProgress/
UWCWDataset1_MNBayes_20160818/Data/FormattedProcessed_Data/

$ cut -d',' -f5 FormatProcessedLevel3_MFS_dataset1b.csv | sort | uniq -c

     22 0
   3256 1
   3265 2
      1 3



==== II. Transforming the Level 3 dataset to the Level 4 dataset
==========================================================================

        a. new Level3 data:
FormatProcessedLevel3_MFS_dataset1b.csv

        b. Process the new Level 3 data to Level 4 data
        b1. see "FormatProcessingLevel3-4_MFS_dataset1_OneHotEncoder_v2.py" for processing details



==== XIV. Formatted and pre-processed dataset Level 4
==========================================================================

The comma-separated dataset is now formatted and pre-processed correctly for
input to sklearn.naive_bayes.MultinomialNB in Python.


FormatProcessedLevel4_MFS_dataset1b.csv

1. Number of Instances
        FormatProcessed_MFS_dataset1.csv        6544

2. Number of Attributes
        114 input + 1 class attribute
```

```
3. For Each Attribute:
        All input attributes are integers in the range 0,1
        The last attribute is the class code 0,1

4. Missing Attribute Values
        None

5. Class Distribution
        Class:  No of examples
        0:  3744
        1:  2800
```

United Way MLPP: Data checks on Level 2 MFS dataset1

```
# -*- coding: utf-8 -*-


"""
Title:                    Apply np.isnan().any() to
                          Formatted/Processed Level2 MFS_dataset1
                          as Dataset Error Check
                          (Python script)
Project Descriptor:       Machine Learning Pilot Project (United Way)
Document ID:
Author:                   Bruce Marron, Portland State University
Date:                     25 Aug 2016

"""


"""
Background:
This script provides an example of the use of machine learning tools (the
Multinomial Naive Bayes algorithm) as applied to a dataset obtained from
Metropolitan Family Service (MFS) under the Machine Learning Pilot Project
sponsored by the United Way of the Columbia-Willamette (UWCW). The goals and
implementation of the Machine Learning Pilot Project, as well as privacy and
data safeguards, are detailed in two documents,

(1) "Machine Learning Tools for Social Service Providers Funded by the
United Way of the Columbia-Willamette" and
(2) "Project Plan for the Machine Learning Pilot Project"

These documents and additional information about the pilot project are
available from Alejandro Queral, Director of Systems Planning and Performance
at UWCW (email: AlejandroQ@Unitedway-pdx.org).


The MFS data used in this example consist of demographic and attendance data
for MFS students enrolled in the Schools Uniting Neighborhoods (SUN) program
sponsored by Multnomah County in Oregon. In 2016 there were 80 SUN Community
Schools in 6 school districts across Multnomah County: 36 elementary,
19 middle, 16 K-8, and 9 high schools. For more information about the data and
to obtain a copy of the dataset, contact Geoff Brusca, Database and Impact
Manager at MFS (email: geoffb@mfs.email)

"""


"""
Important and helpful websites:
https://www.python.org/
http://scikit-learn.org/
http://scikit-learn.org/stable/modules/naive_bayes.html
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
https://www.linuxfoundation.org/
https://docs.continuum.io/anaconda/
http://stackoverflow.com/questions/14254203/mixing-categorial-and-continuous-data-in-naive-bayes-classifier-using-scikit-learn
"""

#%%

"""
On a Linux-based machine with Anaconda installed, open a command line terminal
and load Spyder (Scientific PYthon Development EnviRonment) by typing 'spyder'
at the Linux prompt,
```

```
Note the organization of the Spyder user interface:
(1) a code testing environment compartmentalized with '#%%' symbols into cells
(2) options for variable explorer, object inspector, file explorer environments
(3) an enhanced Python interpreter environment (IPython console)

Spyder allows you to try out the pieces of code within a '#%%' cell without
having to run the entire script (see the options marked with a green arrowpoint
in the tool bar below the pull-down menus).

Spyder uses colors to denote the functionality associated with various words
and symbols. For example, any words or symbols typed within a matched set of
three quotes (ie, """                    ..... """) will not be taken as
    Python code or syntax.
Similarly, any words or symbols following the pound sign (#) will not be
interpreted as Python code and will not be executed.
"""
#%%

"""
Following this cell is the step-by-step procedure for using the (multinomial)
Naive Bayes classifier machine learning tool, "MultinomialNB" as a data check
on a dataset obtained from Metropolitan Family Service (MFS). The MFS dataset is
labeled, "FormatProcessedLevel2_MFS_dataset1.csv".

IMPORTANT NOTE:  The MFS dataset has been correctly formatted and
pre-processed for input into the "MultinomialNB" algorithm (Level 2).
Formatting and pre-processing a raw dataset is generally non-trivial.
The  details of the formatting and pre-processing of the raw MFS dataset to
Level 2 is detailed in the file,
"LOG_FormatProcessing_MFS_dataset1.txt".

"""

#%% STEP 1: Setup the required Python environment

"""
Python requires that certain packages be loaded to provide the necessary
functionalities called for by the command syntax in the script. Note that the
words 'import', 'as', and 'from' are all command words in Python.

"""

import cPickle
import urllib
import csv
import operator
import numpy as np
import pylab as plt
import pandas as pd
from time import time
from numpy import genfromtxt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.naive_bayes import MultinomialNB

```

```


#%% STEP 2. Import MFS data (Level 2) into Python


# load data

Level2_MFS_dataset1 = genfromtxt("/home/bmarron/Desktop/
    FormatProcessedLevel2_MFS_dataset1.csv", delimiter=",")


#%% STEP 3. Check dataset for NaN, infinty, or values that are too large

# Data are checked for inappropriate values (Not A Number == NaN)


np.isnan(Level2_MFS_dataset1).any()
# If True (this is NOT good)

# If False (this is good) ==> skip STEP 4


#%% STEP 4. Only if STEP 3 returns, "True"


# find offending records

np.where(np.isnan(Level2_MFS_dataset1))
# For example when True got this output

#(array([ 696, 1432, 1664, 1924, 2221, 2363, 2370, 3272, 3408, 3820, 4013,

#         4145, 4789, 5096, 5309]),

# array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]))

# Appears that records 696, 1432, ... have problems in Field 4

# Go back and repair datset (see XI. Feedback from Python in,

# "LOG_FormatProcessing_MFS_dataset1.txt"


#%% STEP 5: Confirm data content with info about MFS dataset


"""
Check the characteristics of "Level2_MFS_dataset1" as compared to the known
characteristics of the dataset in "FormatProcessedLevel2_MFS_dataset1.csv".

No. rows = 6544
No. columns = 17

No. records >= 30-day attendance = 2800
No. records < 30-day attendance = 3744

Recall from "LOG_FormatProcessing_MFS_dataset1.txt", there are the following
number of target values (zeroes and ones),

==== XII. Completely formatted and pre-processed dataset ==============
```

```

5. Class Distribution
        Class:  No. of examples
        0:  3744
        1:  2800

"""

# check dataset size

Level2_MFS_dataset1.shape
# Out[3]: (6544, 17)


# check for correct targets

test0 = np.where(Level2_MFS_dataset1[:, 16] == 0)
test0 = list(test0[0])

test1 = np.where(Level2_MFS_dataset1[:, 16] == 1)
test1 = list(test1[0])

print "zeros: %s" %(len(test0))
print "ones: %s" %(len(test1))

# zeros: 3744

# ones: 2800
```

UWCW MLPP: Transform categorical Level3 data to Level4 data

```
# -*- coding: utf-8 -*-


"""
Title:                      Apply sklearn.preprocessing.OneHotEncoder to
                            Formatted/Processed Level3 MFS_dataset1b.csv
                            to Generate Formatted/Processed Level4 MFS_dataset1
                            (Python script)
Project Descriptor:         Machine Learning Pilot Project (United Way)
Document ID:
Author:                     Bruce Marron, Portland State University
Date:                       27 Aug 2016

"""


"""
Background:
This script provides an example of the use of machine learning tools
as applied to a dataset obtained from Metropolitan Family Service (MFS)
under the Machine Learning Pilot Project (MLPP) sponsored by the United Way
of the Columbia-Willamette. The goals and implementation of the MLPP, as well
as privacy and data safeguards, are detailed in two documents,

(1) "Machine Learning Tools for Social Service Providers Funded by the
United Way of the Columbia-Willamette" and
(2) "Project Plan for the Machine Learning Pilot Project"

These documents and additional information about the pilot project are
available from Alejandro Queral, Director of Systems Planning and Performance
at UWCW (email: AlejandroQ@Unitedway-pdx.org).


The MFS data used in this example consist of demographic and attendance data
for MFS students enrolled in the Schools Uniting Neighborhoods (SUN) program
sponsored by Multnomah County in Oregon. In 2016 there were 80 SUN Community
Schools in 6 school districts across Multnomah County: 36 elementary,
19 middle, 16 K-8, and 9 high schools. For more information about the data and
to obtain a copy of the dataset, contact Geoff Brusca, Database and Impact
Manager at MFS (email: geoffb@mfs.email)

"""

"""
Important and helpful websites:
https://www.python.org/
http://scikit-learn.org/
http://scikit-learn.org/stable/modules/naive_bayes.html
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.
    MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
https://www.linuxfoundation.org/
https://docs.continuum.io/anaconda/
http://stackoverflow.com/questions/14254203/mixing-categorial-and-continuous-
    data-in-naive-bayes-classifier-using-scikit-learn
http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.
    OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder
"""

#%% Preliminaries

"""
On a Linux-based machine with Anaconda installed, open a command line terminal
and load Spyder (Scientific PYthon Development EnviRonment) by typing 'spyder'
at the Linux prompt,
```

```

Note the organization of the Spyder user interface:
(1) a code testing environment compartmentalized with '#%%' symbols into cells
(2) options for variable explorer, object inspector, file explorer environments
(3) an enhanced Python interpreter environment (IPython console)

Spyder allows you to try out the pieces of code within a '#%%' cell without
having to run the entire script (see the options marked with a green arrowpoint
in the tool bar below the pull-down menus).

Spyder uses colors to denote the functionality associated with various words
and symbols. For example, any words or symbols typed within a matched set of
three quotes (ie, """                    ..... """) will not be taken as
    Python code or syntax.
Similarly, any words or symbols following the pound sign (#) will not be
interpreted as Python code and will not be executed.
"""
#%%

"""
Following this cell is the step-by-step procedure for using the
sklearn.preprocessing.OneHotEncoder to encode categorical integer features
in the dataset obtained from Metropolitan Family Service (MFS) using a
one-hot (aka one-of-K) scheme. This encoding is needed for feeding
categorical data to many scikit-learn estimators like MultinomialNB

The MFS dataset is labeled, "FormatProcessedLevel3_MFS_dataset1b.csv".

IMPORTANT NOTE:  The MFS dataset has been correctly formatted and
pre-processed for input into the OneHotEncoder algorithm. Formatting and
pre-processing a raw dataset is generally non-trivial. The details of the
formatting and pre-processing of the MFS dataset are provided in the file,
"LOG_FormatProcessing_MFS_dataset1.txt".

"""

#%% STEP 1: Setup the required Python environment

"""
Python requires that certain packages be loaded to provide the necessary
functionalities called for by the command syntax in the script. Note that the
words 'import', 'as', and 'from' are all command words in Python.

"""

import cPickle
import csv
import operator
import numpy as np
import pylab as plt
import pandas as pd
from time import time
from numpy import genfromtxt
from sklearn.preprocessing import OneHotEncoder



#%% Import MFS data into Python


```

```python
# load data

Level3_MFS_dataset1b = genfromtxt("/home/bmarron/Desktop/PSU/PhD_EES/2016SoE013_STAT_570_Consulting/Works_InProgress/UWCWDataset1_MNBayes_20160818/Data/FormattedProcessed_Data/FormatProcessedLevel3_MFS_dataset1b.csv", delimiter=",")

# check data

Level3_MFS_dataset1b.shape

#%% Split the data in Level3_MFS_dataset1b into three separate datasets


# Dataset "X" contains all the input data

# Dataaset "y" contains all the attribute data

# Dataset "a" contains all Field 4 (age) data

X, y, a = Level3_MFS_dataset1b[:,0:16], Level3_MFS_dataset1b[:, 16], Level3_MFS_dataset1b[:, 3]

#remove Field 4 from X

Xminus = np.delete(X, 3, axis=1)



#%% Save the data

# all categorical data (all Fields except Field 4 and Field 17) == Xminus

# age (Field 4)== a

# target data (Field 17)== y


# Save X

with open("Level3_MFS_dataset1b_X.pkl", 'wb') as f:
    cPickle.dump(X, f, protocol=2)

# Save Xminus

with open("Level3_MFS_dataset1b_Xminus.pkl", 'wb') as f:
    cPickle.dump(Xminus, f, protocol=2)
# Save a

with open("Level3_MFS_dataset1b_y.pkl", 'wb') as f:
    cPickle.dump(y, f, protocol=2)

# Save y

with open("Level3_MFS_dataset1b_a.pkl", 'wb') as f:
    cPickle.dump(a, f, protocol=2)


#%% Run the encoder

```

```

Enc = OneHotEncoder()

Xminus_csrf=Enc.fit_transform(Xminus)

Xminus_onehot=Xminus_csrf.toarray()

#%% Check encoder outputs


# check immediate encoder output

In [32]: Xminus_csrf
Out[32]:
<6544x113 sparse matrix of type '<type 'numpy.float64'>'
        with 98160 stored elements in Compressed Sparse Row format>:

# check transform from Compressed Sparse Row format back to array

In [34]: Xminus_onehot.shape
Out[34]: (6544, 113)


In [8]: Enc.feature_indices_
    ...:
Out[8]:
array([  0,  23,  28,  31,  35,  93,  95,  97,  99, 101, 103, 105, 107,
       109, 111, 113])

# check that the correct number of columns were added

# 23-0 = 23 (School Name)

# 28-23 = 5 (School District)

# 31-28 = 3 (school Type)

# 35-31 = 4  (Gender)

# 93-35 = 58 (Language)

# 95-93 = 2 (African)

# 97-95 = 2 (Asian)

# 99-97 = 2 (Black)

# 101-99 = 2 (Latino)

# 103-101 = 2 (Middle Eastern)

# 105-103 = 2 (Native American)

# 107-105 =2 (Hawaiian)

# 109-107 = 2 (Slavic)

# 111-109 = 2 (White)

# 113-111 = 2 (Declined)
```

```


#%% add age data back in


# reshape vector to match Xminus_onehot shape

a = a.reshape(6544,1)

X_transform = np.hstack((Xminus_onehot, a))

#check

In [42]: X_transform.shape
Out[42]: (6544, 114)

#%% Save Level 4 data


# Save X_transform

with open("Level4_MFS_dataset1b_X_transform.pkl", 'wb') as f:
    cPickle.dump(X_transform, f, protocol=2)


# reshape y

y_reshape = y.reshape(6544,1)


# Save y_reshape

with open("Level4_MFS_dataset1b_y_reshape.pkl", 'wb') as f:
    cPickle.dump(y_reshape, f, protocol=2)

#%% Generate the complete Level 4 dataset and save


FormatProcessedLevel4_MFS_dataset1 = np.hstack((X_transform, y_reshape))

#save

with open("Level4_MFS_dataset1b_onehot.pkl", 'wb') as f:
    cPickle.dump(FormatProcessedLevel4_MFS_dataset1, f, protocol=2)


#%% save Level 4 data to .csv


np.savetxt("FormatProcessedLevel4_MFS_dataset1b.csv",
   FormatProcessedLevel4_MFS_dataset1, delimiter=",")
```

UWCW MLPP: Executing MultinomialNB on Level4 data

```
# -*- coding: utf-8 -*-


"""
Title:                  Apply sklearn.naive_bayes.MultinomialNB to
                        Formatted/Processed Level4 MFS_dataset1
                        as Naive Bayes Classifier
                        (Python script)
Project Descriptor:     Machine Learning Pilot Project (United Way)
Document ID:
Author:                 Bruce Marron, Portland State University
Date:                   28 Aug 2016

"""


"""
Background:
This script provides an example of the use of machine learning tools (the
Multinomial Naive Bayes algorithm) as applied to a dataset obtained from
Metropolitan Family Service (MFS) under the Machine Learning Pilot Project
sponsored by the United Way of the Columbia-Willamette (UWCW). The goals and
implementation of the Machine Learning Pilot Project, as well as privacy and
data safeguards, are detailed in two documents,

(1) "Machine Learning Tools for Social Service Providers Funded by the
United Way of the Columbia-Willamette" and
(2) "Project Plan for the Machine Learning Pilot Project"

These documents and additional information about the pilot project are
available from Alejandro Queral, Director of Systems Planning and Performance
at UWCW (email: AlejandroQ@Unitedway-pdx.org).


The MFS data used in this example consist of demographic and attendance data
for MFS students enrolled in the Schools Uniting Neighborhoods (SUN) program
sponsored by Multnomah County in Oregon. In 2016 there were 80 SUN Community
Schools in 6 school districts across Multnomah County: 36 elementary,
19 middle, 16 K-8, and 9 high schools. For more information about the data and
to obtain a copy of the dataset, contact Geoff Brusca, Database and Impact
Manager at MFS (email: geoffb@mfs.email)

"""


"""
Software and hardware:
The Machine Learning Pilot Project is committed to the sole use of readily
available, open-source software and hardware. No proprietary software is
required for this script or for any other statistical analysis performed in
support of the Machine Learning Pilot Project. All calculations and analyses
carried out by the Machine Learning Pilot Project are therefore completely
transparent and available for review and validation.

This script uses a machine learning tool called, "MultinomialNB" which is part
of the Python-based machine learning library, "sci-kit learn." The development
of this script as well as all subsequent computations were performed on a used,
HP Compaq 6710b (32-bit) machine obtained from the non-profit, "Free Geek"
located in Portland, OR. The HP Compaq 6710b machine was running the following
Linux-based operating system:

Distro: Ubuntu 14.04 trusty
Kernel: 3.16.0-77-generic i686 (32 bit, gcc: 4.8.4)
Desktop: Xfce 4.11.8 (Gtk 2.24.23)
```

```
"""


"""
Important and helpful websites:
https://www.python.org/
http://scikit-learn.org/
http://scikit-learn.org/stable/modules/naive_bayes.html
http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
https://www.linuxfoundation.org/
https://docs.continuum.io/anaconda/
http://stackoverflow.com/questions/14254203/mixing-categorial-and-continuous-data-in-naive-bayes-classifier-using-scikit-learn
"""

#%%

"""
On a Linux-based machine with Anaconda installed, open a command line terminal
and load Spyder (Scientific PYthon Development EnviRonment) by typing 'spyder'
at the Linux prompt,
$ spyder

If Anaconda Navigator is installed (https://docs.continuum.io/anaconda/navigator)
type 'anaconda-navigator' at the Linux prompt,
$ anaconda-navigator

and then launch Spyder through the Anaconda Navigator window.


If you are running Python (Anaconda distribution) from a Windows or Mac, the
process is similar (https://docs.continuum.io/anaconda/ide_integration).
"""
#%% Preliminaries

"""
Import THIS script into Spyder using the pull-down menus,
File --> Open --> <this file>.py

Note the organization of the Spyder user interface:
(1) a code testing environment compartmentalized with '#%%' symbols into cells
(2) options for variable explorer, object inspector, file explorer environments
(3) an enhanced Python interpreter environment (IPython console)

Spyder allows you to try out the pieces of code within a '#%%' cell without
having to run the entire script (see the options marked with a green arrowpoint
in the tool bar below the pull-down menus).

Spyder uses colors to denote the functionality associated with various words
and symbols. For example, any words or symbols typed within a matched set of
three quotes (ie, """                         ..... """) will not be taken as Python code or syntax.
Similarly, any words or symbols following the pound sign (#) will not be
interpreted as Python code and will not be executed.
"""
#%%

"""
```

```
Following this cell is the step-by-step procedure for using the (multinomial)
Naive Bayes classifier machine learning tool, "MultinomialNB" to evaluate a
dataset obtained from Metropolitan Family Service (MFS). The MFS dataset is
labeled, "FormatProcessedLevel4_MFS_dataset1.csv".

IMPORTANT NOTE:  The MFS dataset has been correctly formatted and
pre-processed for input into the "MultinomialNB" algorithm (Level 3).
Formatting and pre-processing a raw dataset is generally non-trivial.
The  details of the formatting and pre-processing of the original MFS dataset
to Level 3 are provided in the file,
"LOG_FormatProcessing_MFS_dataset1.txt".

"""

#%% STEP 1: Setup the required Python environment

"""
Python requires that certain packages be loaded to provide the necessary
functionalities called for by the command syntax in the script. Note that the
words 'import', 'as', and 'from' are all command words in Python.

"""

import cPickle
import urllib
import csv
import operator
import numpy as np
import pylab as plt
import pandas as pd
from time import time
from numpy import genfromtxt
from sklearn.cross_validation import train_test_split
from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, roc_auc_score



#%% STEP 2: Import, randomize, and save the formatted and pre-processed

# MFS data into Python


"""
Import the properly formatted and pre-processed MFS data into Python and
re-name the dataset for running MultinomialNB. Because MultinomialNB requires
splitting up the original data into training data and test data, the MFS
datset contained in, "FormatProcessedLevel4_MFS_dataset1.csv" will be
re-named to, "Level4_MFS_dataset1". The dataset is imported into a Numpy array.
Details of the data import are shown in the "Variable explorer" window.

The data are shuffled 2x.

The data are saved in .pkl format for ease of re-loading into Python.
"""
```

```
# load Level4 data: FormatProcessedLevel4_MFS_dataset1b.csv

Level4_MFS_dataset1 = genfromtxt("/home/bmarron/Desktop/PSU/PhD_EES/2016
    SoE013_STAT_570_Consulting/Works_InProgress/UWCWDataset1_MNBayes_20160818/
    Data/FormattedProcessed_Data/FormatProcessedLevel4_MFS_dataset1b.csv",
    delimiter=",")

# shuffle data 2x

Level4_MFS_dataset1 = np.random.permutation(Level4_MFS_dataset1)
Level4_MFS_dataset1 = np.random.permutation(Level4_MFS_dataset1)

# Save the data in Python-ready format

with open("Level4_MFS_dataset1b.pkl", 'wb') as f:
    cPickle.dump(Level4_MFS_dataset1, f, protocol=2)



#%% STEP 3: Split full dataset into inputs and attributes to generate training

# and test sets


"""
Naive Bayes classifiers need to be trained on one set of data and then tested
on a second, different set of data. The outputs (results) of the classifier
applied to the test data are used to evaluate classifier performance.

"""

# Split the data in Level4_MFS_dataset1 into two separate datasets,

# Dataset "X" contains all the input data

# Dataaset "y" contains all the attribute data

X, y = Level4_MFS_dataset1[:,0:114], Level4_MFS_dataset1[:, 114]

#Use the built-in function, "train_test_split()" to generate a training dataset

# and a test dataset (75:25 split).

#The training dataset has two subsets: "tr_d_X" plus its associated "tr_d_y".

#The test dataset has two subsets: "te_d_X" plus its associated "te_d_y".

tr_d_X, te_d_X, tr_d_y, te_d_y = train_test_split(X, y, test_size=.25,
    random_state=74)


#%% STEP 5a: Save the training data and the test data


# Save tr_d_X

with open("tr_d_X.pkl", 'wb') as f:
    cPickle.dump(tr_d_X, f, protocol=2)

# Save tr_d_y
```

```

with open("tr_d_y.pkl", 'wb') as f:
    cPickle.dump(tr_d_y, f, protocol=2)

# Save te_d_X

with open("te_d_X.pkl", 'wb') as f:
    cPickle.dump(te_d_X, f, protocol=2)

# Save te_d_y

with open("te_d_y.pkl", 'wb') as f:
    cPickle.dump(te_d_y, f, protocol=2)

#%% STEP 5b: Reload the training and test data into Python, if needed


tr_d_X = cPickle.load(open("/home/bmarron/Desktop/tr_d_X.pkl","rb"))

#Reload the processed data

tr_d_y = cPickle.load(open("/home/bmarron/Desktop/tr_d_y.pkl","rb"))

#Reload the processed data

te_d_X = cPickle.load(open("/home/bmarron/Desktop/te_d_X.pkl","rb"))

#Reload the processed data

te_d_y = cPickle.load(open("/home/bmarron/Desktop/te_d_y.pkl","rb"))


#%% STEP 6: Build and run the classifier


# Build (and time) the classifier

print ("Building the model from training data")
t0 = time()
classifier = MultinomialNB(alpha=1, class_prior=None, fit_prior=True)
#classifier = MultinomialNB(alpha=0, class_prior=None, fit_prior=True)

classifier_model = classifier.fit(tr_d_X, tr_d_y)
print ("done in %fs" %(time() - t0))

# Run (and time) the classifier

print("Using the model to predict the outcomes in the test set")
t0 = time()
classifier_pred_y = classifier.predict(te_d_X)
print("done in %fs" % (time() - t0))

#%% STEP 7a: Evaluate the MultinomialNB classifier


# Accuracy score

print ("Accuracy:")
print accuracy_score(te_d_y, classifier_pred_y)

```

```python
#%% STEP 7b: Evaluate the MultinomialNB classifier


# Precision and recall

precision, recall, thresholds = precision_recall_curve(te_d_y, classifier_pred_y)
av_precision = average_precision_score(te_d_y, classifier_pred_y)

print("Classification report on test set for classifier:")
print(classifier)
clfreport = classification_report(te_d_y, classifier_pred_y)
print(clfreport)


#%% STEP 7c: Evaluate the MultinomialNB classifier


# Confusion matrix (standard orientation)

cm = confusion_matrix(te_d_y, classifier_pred_y, labels=[1, 0])
print('Confusion matrix:')
print(cm)

#%% STEP 7d: Evaluate the MultinomialNB classifier


fpr, tpr, _ = roc_curve(te_d_y, classifier_pred_y)

roc_auc = roc_auc_score(te_d_y, classifier_pred_y)



#%% STEP 8a: Generate evaluation plots

# http://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels

# http://stackoverflow.com/questions/11244514/modify-tick-label-text


# Confusion matrix plot

plt.clf() # clear the plot fxn
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
a=ax.get_xticks().tolist()
b=ax.get_yticks().tolist()
a[1]='1'
a[2]='0'
b[1]='1'
b[2]='0'
ax.set_xticklabels(a)
ax.set_yticklabels(b)
#plt.title('Confusion matrix for the MultinomialNB Classifier', fontsize=10)

fig.colorbar(cax)
plt.savefig("MNBayes1.pdf")

```

```
#%% STEP 8b: Generate evaluation plots

# Precision-recall curve

plt.clf() # clear the plot fxn
plt.plot(recall, precision, label='Precision-Recall curve (area = {0:0.2f})'.
    format(av_precision))
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall MultinomialNB')
plt.legend(loc="lower right")
plt.savefig("MNBayes2.pdf")

#%% Generate evaluation plots

# Plot of a ROC curve for a specific class

plt.clf()
#plt.figure()

plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC MultinomialNB ')
plt.legend(loc="lower right")
plt.savefig("MNBayes3.pdf")
plt.show()

#%%Output for LaTex reports


df = pd.DataFrame(cm)
print df.to_latex()
```