Week 5: Final project by Antonio Gálvez

Introduction

This notebook will guide you through the development of my final project. For my final project I have used the Student Performance Data set from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Student+Performance). Originally, the Student Performance Data set was shared by Paulo Cortez from the University of Minho (http://www3.dsi.uminho.pt/pcortez/Home.html).

Part 1: Dataset description

The Student Performance dataset containts data related to student achievement in secondary education of two Portuguese schools. The dataset attributes include student grades, demographic, social and school related features. It was collected by using school reports and questionnaires in the subjects of Mathematics (mat).

The original dataset contains information of 395 students. The information taken from each student is listed below: (the dataset description comes from the UCI Machine Learning Repository)

Id	Noum	Data	Description
1	school	binary	student's school: "GP" - Gabriel Pereira or "MS" - Mousinho da Silveira
2	sex	binary	student's sex: "F" - female or "M" - male
3	age	numeric	student's age: from 15 to 22
4	address	binary	student's home address type: "U" - urban or "R" - rural
5	famsize	binary	family size: "LE3" - less or equal to 3 or "GT3" - greater than 3
6	Pstatus	binary	parent's cohabitation status: "T" - living together or "A" - apart
7	Medu	numeric	mother's education: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education
8	Fedu	numeric	father's education: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education
9	Mjob	string	mother's job: "teacher", "health" care related, civil "services", "at_home" or "other"
10	Fjob	string	father's job: "teacher", "health" care related, civil "services", "at_home" or "other"
11	reason	string	reason to choose this school: close to "home", school "reputation", "course" preference or "other"
12	guardian	string	student's guardian: "mother", "father" or "other"
13	traveltime	numeric	home to school travel time: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour
14	studytime	numeric	weekly study time: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours
15	failures	numeric	number of past class failures: n if 1<=n<3, else 4
16	schoolsup	binary	extra educational support: yes or no
17	famsup	binary	family educational support: yes or no
18	paid	binary	extra paid classes within Math: yes or no
19	activities	binary	extra-curricular activities: yes or no
20	nursery	binary	attended nursery school: yes or no
21	higher	binary	wants to take higher education: yes or no
22	internet	binary	Internet access at home: yes or no
23	romantic	binary	with a romantic relationship: yes or no

Id	Noum	Data	Description
24	famrel	numeric	quality of family relationships: from 1 - very bad to 5 - excellent
25	freetime	numeric	free time after school: from 1 - very low to 5 - very high
26	goout	numeric	going out with friends: from 1 - very low to 5 - very high
27	Dalc	numeric	workday alcohol consumption: from 1 - very low to 5 - very high
28	Walc	numeric	weekend alcohol consumption: from 1 - very low to 5 - very high
29	health	numeric	current health status: from 1 - very bad to 5 - very good
30	absences	numeric	number of school absences: from 0 to 93

In [1]:

```
import csv
import matplotlib.pyplot as plt
import pandas
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.datasets import make_blobs
```

In [2]:

```
file = open("./student-mat.csv", 'r')
reader = csv.reader(file, delimiter = ';')
header1 = next(reader)
```

Part 2: Data preparation

In the following piece of code is defined the type of some variables to ensure that all of these parameters are properly identified. The parameter of this dataset was defined as data type of string, integer or logical.

In [3]:

```
data = []
for line in reader:
    d = dict(zip(header1, line))
    for field in ['school', 'sex', 'address', 'famsize', 'Pstatus',
                  'Mjob', 'Fjob','reason', 'guardian']:
        d[field] = str(d[field])

    for field in ['age', 'Medu', 'Fedu', 'traveltime', 'studytime',
                  'failures', 'famrel', 'freetime', 'goout', 'Dalc',
                  'Walc', 'health', 'absences', 'failures']:
        d[field] = int(d[field])

    for field in ['schoolsup', 'famsup', 'activities', 'nursery',
                  'higher', 'internet', 'romantic']:
        if d[field] == 'yes':
            d[field] = True
        else:
            d[field] = False

    data.append(d)
```

Part 3: Simple statistics development

The statistics developed has two main parts. The first one evaluates the number of past class failures making a comparison between two genders. The second one sums weekly study time agrupping by student's guards.

The following piece of code give the percentage of failures by genders.

In [4]:

```
datasetF = [d for d in data if d['sex'] == 'F']
len(datasetF)

datasetM = [d for d in data if d['sex'] == 'M']

students = len(data)
studentsM = len(datasetM)
studentsF = len(datasetF)
```

Percertange of student in the gender "Male"

In [5]:

```
PerM = 100*studentsM/students
PerM
```

Out[5]:

```
47.34177215189873
```

Percertange of student in the gender "Female"

In [6]:

```
PerF = 100*studentsF/students
PerF
```

Out[6]:

```
52.65822784810127
```

In [7]:

```
failuresF = []

for k in datasetF:
    failuresF.append(k['failures'])

PerFailuresF = 100*(sum(failuresF)/students)
PerFailuresF
```

Out[7]:

```
15.949367088607595
```

In [8]:

```
PerFailuresM = 100 - PerFailuresF
PerFailuresM
```

Out[8]:

```
84.0506329113924
```

The following piece of code makes groups by student's guardian (mother, father and other). Then, it is summed the weekly study time.

In [9]:

```
for d in data:
    if d['studytime'] == 2:
        d['studytime'] = 2.5
    elif d['studytime'] == 3:
        d['studytime'] = 7.5
    elif d['studytime'] == 4:
        d['studytime'] = 10

data.append(d)
data[0]
```

Out[9]:

```
{'school': 'GP',
 'sex': 'F',
 'age': 18,
 'address': 'U',
 'famsize': 'GT3',
 'Pstatus': 'A',
 'Medu': 4,
 'Fedu': 4,
 'Mjob': 'at_home',
 'Fjob': 'teacher',
 'reason': 'course',
 'guardian': 'mother',
 'traveltime': 2,
 'studytime': 2.5,
 'failures': 0,
 'schoolsup': True,
 'famsup': False,
 'paid': 'no',
 'activities': False,
 'nursery': True,
 'higher': True,
 'internet': False,
 'romantic': False,
 'famrel': 4,
 'freetime': 3,
 'goout': 4,
 'Dalc': 1,
 'Walc': 1,
 'health': 3,
 'absences': 6,
 'G1': '5',
 'G2': '6',
 'G3': '6'}
```

In [10]:

```
AllStudytime = []

for k in data:
    AllStudytime.append(k['studytime'])

totalstudytime = sum(AllStudytime)
totalstudytime
```

Out[10]:

```
1358.5
```

In [11]:

```
mother = [d for d in data if d['guardian'] == 'mother']
stuM = len(mother)
stuM
```

Out[11]:

```
273
```

In [12]:

```
StudytimeM = []
for k in mother:
    StudytimeM.append(k['studytime'])

StudytimeperSM = sum(StudytimeM)/stuM
StudytimeperSM
```

Out[12]:

```
3.399267399267399
```

In [13]:

```
father = [d for d in data if d['guardian'] == 'father']
stuF = len(father)
stuF
```

Out[13]:

```
91
```

In [14]:

```
StudytimeF = []
for k in father:
    StudytimeF.append(k['studytime'])

StudytimeperSF = sum(StudytimeF)/stuF
StudytimeperSF
```

Out[14]:

```
3.4285714285714284
```

In [15]:

```
other  = [d for d in data if d['guardian'] == 'other']
stuO = len(other)
```

In [16]:

```
StudytimeO = []
for k in other:
    StudytimeO.append(k['studytime'])

StudytimeperSO = sum(StudytimeO)/stuO
StudytimeperSO
```

Out[16]:

```
3.703125
```

Part 4: Plotting results of simple statistics development

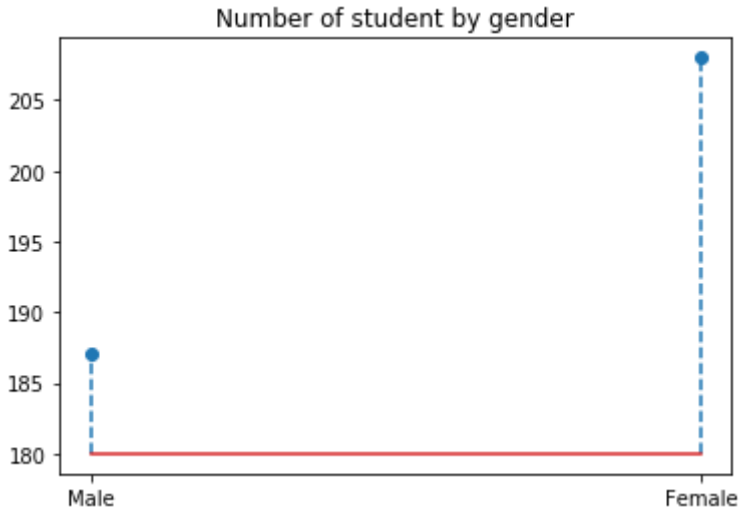
In this section you will find some plots where you can easily compare the results.

In [17]:

```
import matplotlib.pyplot as plt
from matplotlib import colors
import numpy
from collections import defaultdict
```

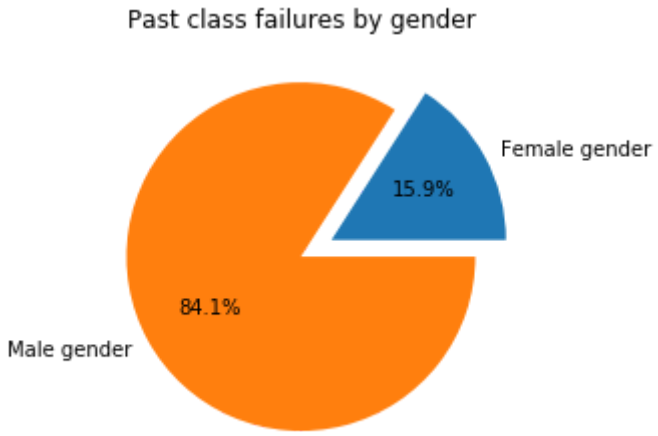
In [18]:

```
x = ["Male", "Female"]
y = [studentsM, studentsF]
plt.stem(x, y, linefmt = '--', bottom = 180 ,use_line_collection = True)
plt.title('Number of student by gender')
plt.show()
```



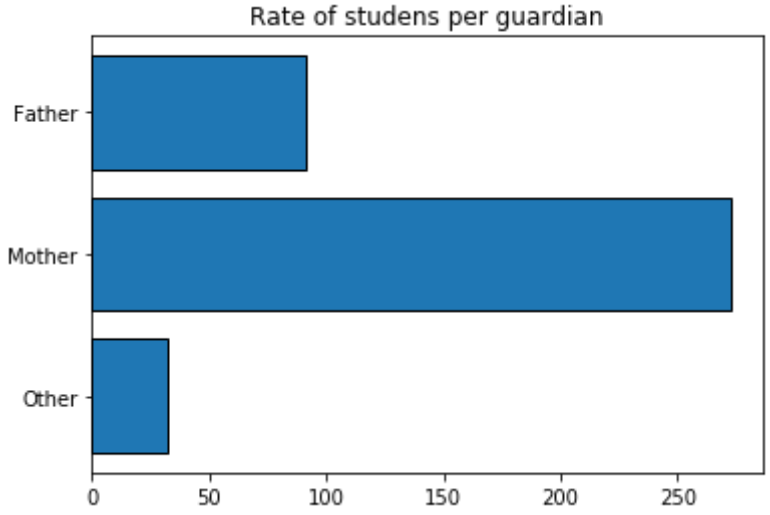
In [19]:

```
failuresGender = [PerFailuresF, PerFailuresM]
labelFa = ['Female gender', 'Male gender']
plt.pie(failuresGender, explode = [0.2, 0],
        autopct = '%1.1f%%', labels = labelFa)
plt.title('Past class failures by gender')
plt.show()
```



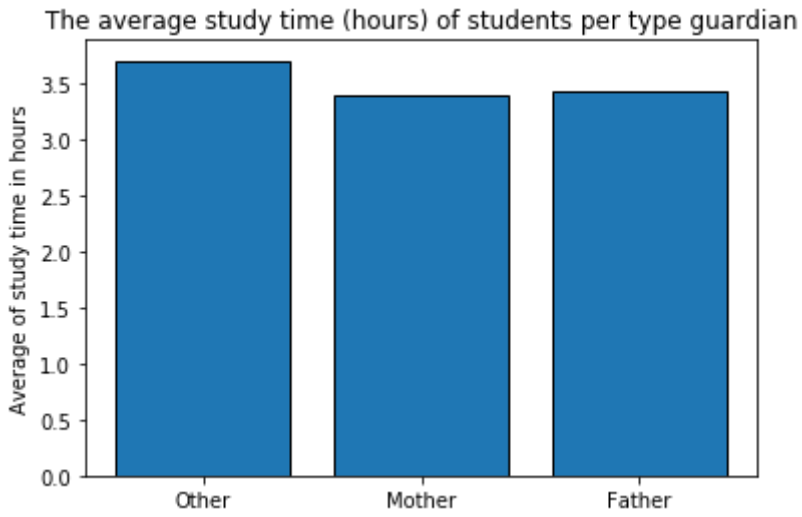
In [20]:

```
numGuar = [stuO, stuM, stuF]
nameGuar = ['Other', 'Mother', 'Father']
plt.barh(range(3), numGuar, edgecolor = 'black')
plt.yticks(range(3), nameGuar, rotation = 0)
plt.title('Rate of studens per guardian')
plt.show()
```



In [21]:

```
AvrStudyTime = [StudytimeperSO, StudytimeperSM, StudytimeperSF]
nameGuar = ['Other', 'Mother', 'Father']
plt.bar(range(3), AvrStudyTime, edgecolor = 'black')
plt.xticks(range(3), nameGuar)
plt.ylabel('Average of study time in hours')
plt.title('The average study time (hours) of students per type guardian')
plt.show()
```



Part 5: Classification and logistic Regression

In the following piece of code are defined and defined the type of the parameters used for the logistic regression model. Afer these lines, it ensures that all of these parameters are correctly cleaned and identified.

In [22]:

```
dataset = []
file = open("./student-mat.csv", 'r')
header = file.readline().strip().split(';')
for line in file:
    line = line.strip().split(';')
    dataset.append(line)
```

In [23]:

```
header.index('Fedu')
```

Out[23]:

7

In [24]:

```
header.index('Medu')
```

Out[24]:

6

In [25]:

```
header.index('failures')
```

Out[25]:

```
14
```

In [26]:

```
y = [int(d[14]) for d in dataset]
```

In [27]:

```
def feature(datum):
    feat = [1, int(datum[6]), int(datum[7])]
    return feat
```

In [28]:

```
X = [feature(d) for d in dataset]
```

The variables selected were already divided by group. Thus, the original matrix of features and the outputs are directly divided in a set of parameters for training and another for testing. The 70% of the parameters are taken for training and the rest for testing.

At the end of this, you can find the accuracy of the predictions obtained by the logistic refression model. Futhermore, there are the confusion matrix to validate the tru positives and fault positivies.

In [29]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
= 1)
```

In [30]:

```
modelm = LogisticRegression()
modelm.fit(X_train, y_train)
```

Out[30]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [46]:

```
predictions = modelm.predict(X_test)
```

In [47]:

```
accuracy_score(y_test, predictions)
```

Out[47]:

```
0.8151260504201681
```

In [48]:

```
confusion_matrix(y_test,predictions)
```

Out[48]:

```
array([[97,  0,  0,  1],
       [11,  0,  0,  0],
       [ 3,  0,  0,  0],
       [ 6,  1,  0,  0]], dtype=int64)
```

Part 6: Splitting the dataset

THe following piece of code splits the dataset into training, validation, and testing sets. The splitting is developed considering mother and father's education as a list of features and the number of past class failures as a list og labels. These both list are splits into training, validation and test sets at a 60/20/20 ratio.

In [49]:

```
from collections import defaultdict
from sklearn import linear_model
import string

wordCount = defaultdict(int)
punctuation = set(string.punctuation)

y = [int(d[14]) for d in dataset]

def feature(datum):
    feat = [1, int(datum[6]), int(datum[7])]
    return feat

X = [feature(d) for d in dataset]

def split_dataset(X, y):
    N = len(X)
    X_train = X[:3*N//5]
    X_validation = X[3*N//5:4*N//5]
    X_test  = X[4*N//5:]
    y_train = y[:3*N//5]
    y_validation = y[3*N//5:4*N//5]
    y_test  = y[4*N//5:]
    results = [X_train, X_validation, X_test, y_train, y_validation, y_test]
    return results

SetSplit = split_dataset(X, y)
```

In [50]:

```
X_tr = SetSplit[0]
X_va = SetSplit[1]
X_te = SetSplit[2]
y_tr = SetSplit[3]
y_va = SetSplit[4]
y_te = SetSplit[5];
```

In [51]:

```
bestModel = None
bestMSE = None
```

In [53]:

```
def MSE(model, X, y):
    predictions = model.predict(X)
    differences = [(a-b)**2 for (a, b) in zip(predictions, y)]
    return sum(differences) / len(differences)

for lamb in [0.01, 0.1, 1, 10, 100]:
    model = linear_model.Ridge(lamb, fit_intercept = False)
    model.fit(X_tr, y_tr)

    mseTrain = MSE(model, X_tr, y_tr)
    mseValid = MSE(model, X_va, y_va)

    print("lambda = " + str(lamb) + ", training/validation error = " +
            str(mseTrain) + '/' + str(mseValid))
    if not bestModel or mseValid < bestMSE:
        bestModel = model
        bestMSE = mseValid
```

```
lambda = 0.01, training/validation error = 0.5998444884366753/0.3709981212
89538
lambda = 0.1, training/validation error = 0.5998462522984568/0.37050350483
4589
lambda = 1, training/validation error = 0.6000106242708517/0.3659166665788
125
lambda = 10, training/validation error = 0.6090527221975848/0.342129064171
96626
lambda = 100, training/validation error = 0.672945036794862/0.344319766625
82675
```

In [54]:

```
mseTest = MSE(bestModel, X_te, y_te)
print("test error = " + str(mseTest))
```

```
test error = 0.3985556474535689
```