University of Canberra

Faculty of Science and Technology

**Programming for Data Science G (11521)**

**Week 11 Tutorial**

**Pandas and Scikit-learn Packages**

**Objectives**

* To use Pandas for database and data manipulation
* To use Scikit-learn for classification in machine learning

**Pandas Package**

* Create a new Python project and name it **PandasScikitlearn**.
* You will work on this online CSV file <https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv> . Have a look at the data in this file on a web browser.

**Example 1: Read data from that CSV file**

* Check if Pandas is available in Python Environments in your project. Install it if it is not.
* Enter the following to the **PandasDatabase.py** file to use Pandas

import pandas as pd

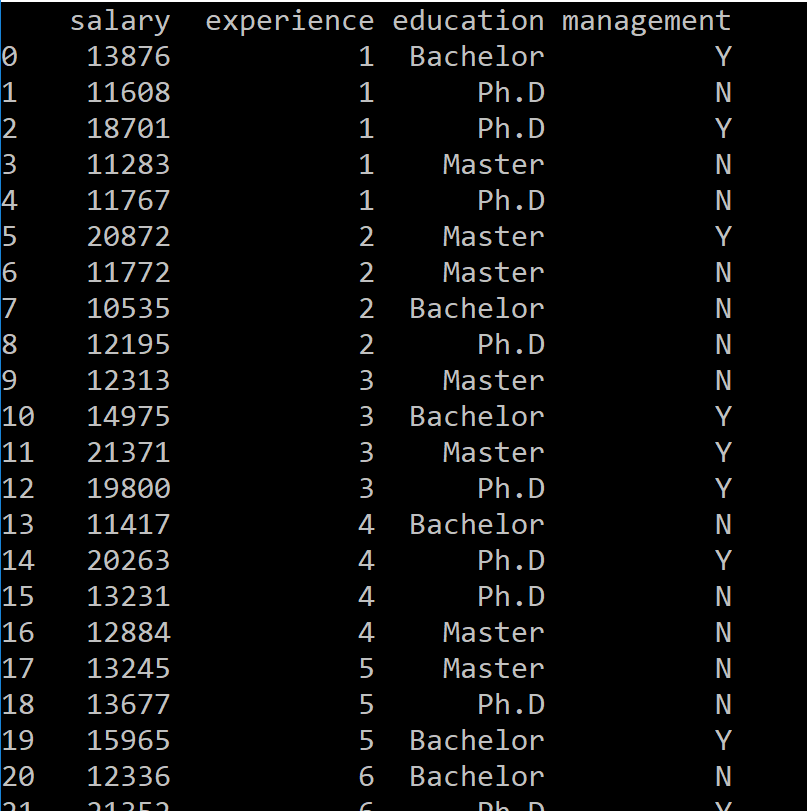
#Example 1: Read data from an online CSV file

url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary\_table.csv'

salary = pd.read\_csv(url)

print(salary)

Your program will output the following



**Example 2: Change the order of columns in the salary table**

* Enter the following code example, below the code for Example 1

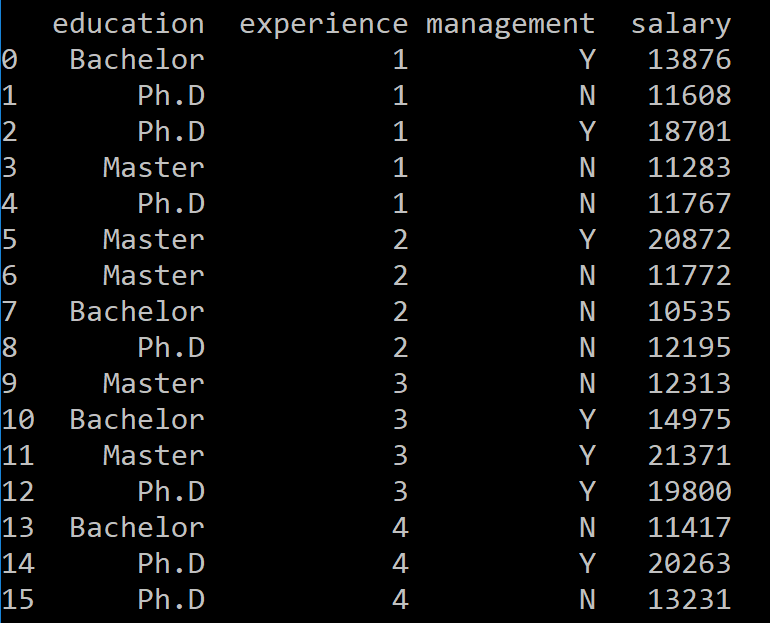
#Example 2: Change the order of columns in the salary table

cols = ['education', 'experience', 'management', 'salary']

salary2 = salary[cols] # or salary2 = salary.loc[:, cols]

print(salary2)

You can see the order of columns is different from that in Example 1.



**Example 3: Filter the salary table**

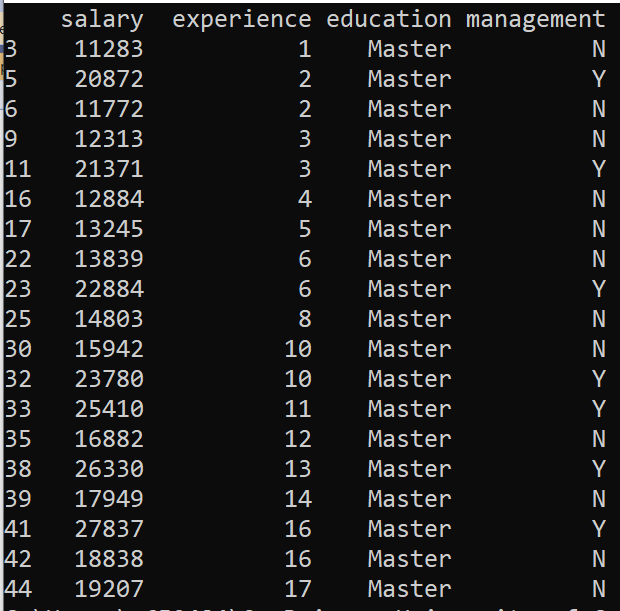
* You will output the data for Master only. Add the following code example to do that

#Example 3: Select rows for Master

rows = salary['education']=='Master'

master\_salary = salary[rows] # or master\_salary = salary.loc[rows, :]

print(master\_salary)



**Example 4: Select rows and columns for Bachelor**

* You output the data for Bachelor only and do not print the education column. There are 3 options to output the same result. Add the following code example to do that

#Example 4: Select rows and columns for Bachelor

rows = salary['education']=='Bachelor'

cols = ['experience', 'management', 'salary']

print('1. Select rows then cols:')

salary\_rows = salary[rows]

salary\_rows\_cols = salary\_rows[cols]

print(salary\_rows\_cols)

print('2. Select rows and cols:')

salary\_rows\_cols = salary[rows][cols]

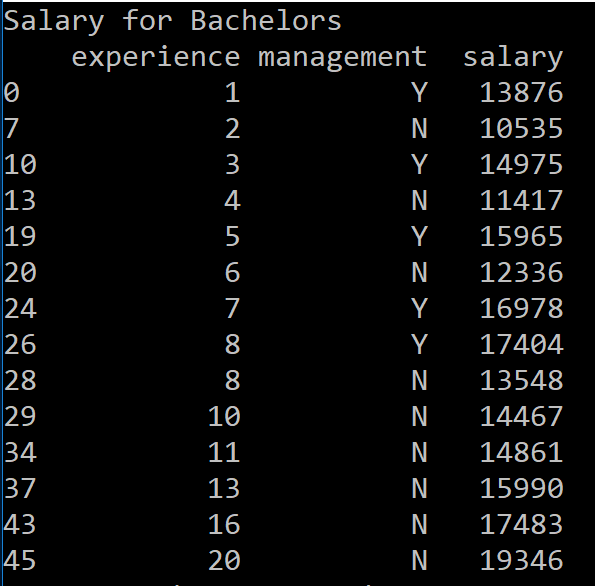
print(salary\_rows\_cols)

print('3. Use loc function:')

salary\_loc = salary.loc[rows, cols]

print(salary\_loc)

The three outputs have the same result as follows:



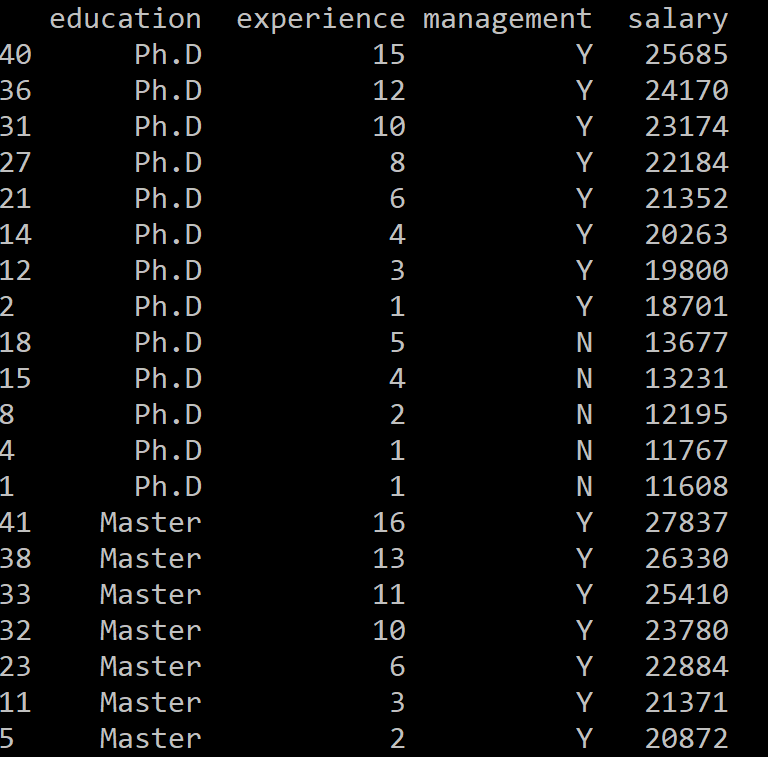
**Example 5: Sort the salary table**

* You sort the data in salary table by education then salary in descending order. Add the following code example to do that

#Example 5: Sort the salary table

sorted\_salary = salary.sort\_values(by=['education', 'salary'], ascending=False)

print(sorted\_salary)



**Example 6: Output to a CSV file**

* You output the sorted list to a CSV file then open that CSV file to read data in. Use the following code example:

#Example 6: Output data to a CSV file

import tempfile, os.path

tmpdir = tempfile.gettempdir()

csv\_filename = os.path.join(tmpdir, "sorted\_salary.csv")

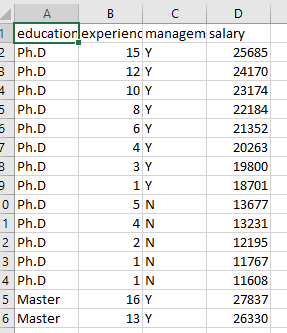
print(tmpdir)

sorted\_salary.to\_csv(csv\_filename, index=False)

input\_csv = pd.read\_csv(csv\_filename)

print(input\_csv)

* Run the project. The output would be the same as that in Example 5.
* Below is part of the data in that CSV file:



**Example 7: Output to an Excel file**

* You output the sorted list to an Excel file then open that Excel file to read data in. Use the following code example:

#Example 7: Output data to a Excel file

import tempfile, os.path

tmpdir = tempfile.gettempdir()

excel\_filename = os.path.join(tmpdir, "salary.xlsx")

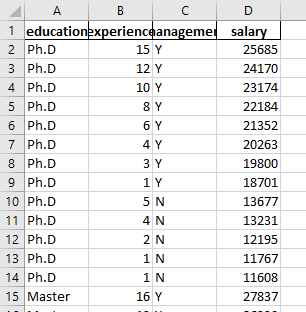
print(tmpdir)

sorted\_salary.to\_excel(excel\_filename, sheet\_name='Sorted salary', index=False)

input\_excel = pd.read\_excel(excel\_filename)

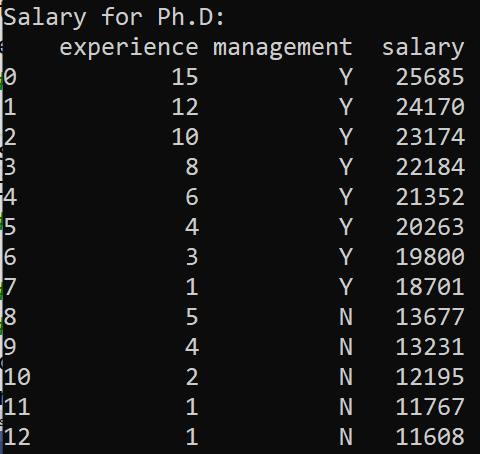
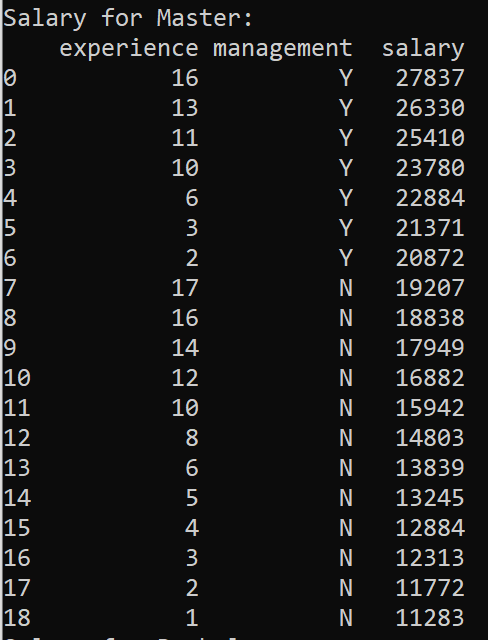
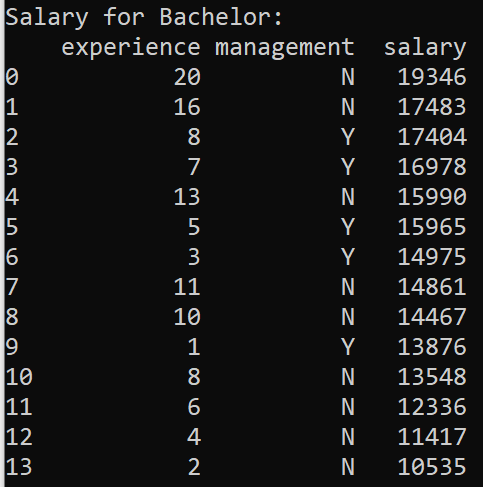
print(input\_excel)

* Run the project. The output would be the same as that in Example 6.
* The data in that Excel file:



**Question 1: Output data in the sorted salary table to an Excel file in 3 sheets for Ph.D, Master and Bachelor.**

* Write a program that outputs data from **sorted salary** in **Example 5** to an Excel file named **salary3.xlsx**. This file has 3 sheets and the sheet names are **Ph.D**, **Master** and **Bachelor**. The data in the sheet name Ph.D are for PhD only and presented in 3 columns **experience**, **management** and **salary**. The same requirement for the data in two sheets for Master and Bachelor. You also need to open this Excel file and read data from the 3 sheets in and output data to screen as seen in the screenshots below.

**Question 2: Output data in the sorted salary table to an SQLite database file.**

Write a program that reads data from the **sorted\_salary.csv** file in **Example 6** and outputs these data to an SQLite database file named **salary.db**. You also write code to open this database file and reads data in and output the data on screen.

**Scikit-learn package**

* Scikit-learn provides simple and efficient tools for predictive data analysis
* Scikit-learn provides built-in estimators or classifiers (machine learning techniques and models) that can be fitted to some data using its fit method.
* Once the estimator is fitted, you can use it for predicting target values of unknown data samples without re-training the estimator.

**Load Iris dataset**

* Add the following line to have modules in scikit-learn package

from sklearn import datasets, neighbors, metrics, svm

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

import matplotlib.pyplot as plt

* Add the following line to get iris dataset

#Load data set

dataset = datasets.load\_iris()

print(dataset)

* Run your program to see what is included in the iris dataset. You will only need to get **data** (data samples in an np-array), **target** (class index of data samples in a list), and **target\_names** (class names or class labels in a list). Add the following code and run to output them.

X = dataset.data

print('Array of data samples:')

print(X)

print()

n\_samples, n\_features = X.shape

print('Number of data samples: ', n\_samples)

print('Dimensionality (Number of features): ', n\_features)

print()

y = dataset.target

print('True class index of data samples:')

print(y)

print()

class\_names = dataset.target\_names

print('Array of class names:', class\_names)

print('Number of classes:', len(class\_names))

print()

* You will split the dataset into training set and testing set using train\_test\_split function. Add the following code and run

#Split dataset into training and testing sets

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

print(X\_train)

print()

print(X\_test)

print()

print(y\_train)

print()

print(y\_test)

print()

test\_size=0.25 means 25% data samples are in testing set (X\_test) and 75% data samples are in training set (X\_train). The same is applied to y\_train and y\_test lists comparing with the y list.

random\_state=0 is to have the same data in X\_train, X\_test, y\_train, y\_test every time you run your program. To have different data for each run change to large number, for example random\_state=42.

**Load classifier (machine learning technique and model) for training and testing**

* Add a classifier (machine learning technique and model) to your program

#Load classifier containing classification technique and model

classifier = neighbors.KNeighborsClassifier(n\_neighbors=3)

If n\_neighbors=1 this KNeighborsClassifier will be the *nearest neighbour classifier* you implemented in Assignment 1.

* Now you use y\_train to train the classifier and use y\_test to test that classifier. Add the following code to do that

#Training

classifier.fit(X\_train, y\_train)

#Testing

y\_pred = classifier.predict(X\_test)

print(y\_pred)

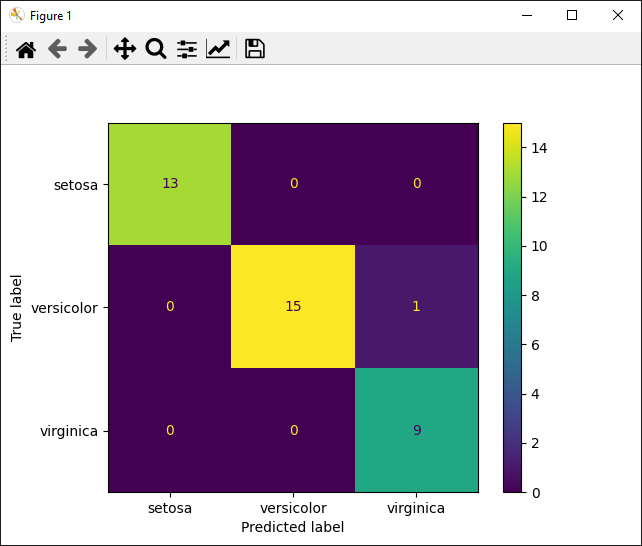
**Output confusion matrix**

* The confusion\_matrix function is used to evaluate classification accuracy of a classifier. Add the following code and run to plot confusion matrix.

#Plot confusion matrix

metrics.plot\_confusion\_matrix(classifier, X\_test, y\_test, display\_labels=class\_names)

plt.show()



* Numbers in the plot are numbers of samples classified (predicted) for each class. For example, number 1 in the plot shows there is 1 sample of versicolor that is incorrectly classified as virgina.

**Change parameter and classifier**

* **Change parameter**: replace **n\_neighbors=3** with **n\_neighbors=1** then run your program again. Compare numbers in the confusion matrix with the numbers when you run your program with n\_neighbors=3.
* **Change classifier**: replace the line

classifier = neighbors.KNeighborsClassifier(n\_neighbors=1)

with

classifier = svm.SVC(gamma=0.5)

to use **Support Vector Machine** classifier then run your program.

* **Change parameter**: replace **gamma=0.5** with **gamma=0.1** then run your program again. Compare numbers in the confusion matrix with the numbers when you run your program with **gamma=0.5**.

**Classification accuracy**

* Change dataset back to **Iris**

dataset = datasets.load\_iris()

and change classifier back to **K-Neighbour classifier**

classifier = neighbors.KNeighborsClassifier(n\_neighbors=3)

* Use **print(y\_pred)** and **print(y\_test)** to print out values. Below are their values (the first line is from y\_pred)

**[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]**

**[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 1]**

* The **y\_test** list contains indices of true classes (0 for Setosa, 1 for Versicolour and 2 for Virginica). The **y\_pred** list contains indices of predicted classes. Compare these values you will see that only the last number (2) in **y\_pred** is different from the last index in **y\_test** (1). This means that the last data sample in X\_test having true class 1 (Versicolour) is correctly classified as 2 (Virginica). Using the following

accuracy (in %) = 100% \* number of correctly classified samples in y\_predict / number of all samples in y\_predict

you will have the accuracy = 100% \* 37 / 38 = 97.37%

**Question 3:**

* Write a function to calculate classification accuracy using true class index list in **y\_test** and predicted class index list in **y\_pred**. The function is as follows

#############################################

def get\_accuracy(true\_list, predicted\_list):

#add your code here to calculate accuracy

return #accuracy in %

#############################################

* After you finish implementing the function, add the following to call the function to get the accuracy

accuracy = get\_accuracy(y\_test, y\_pred)

print('Accuracy: ' + str(accuracy) + '%')

* You will get the output **Accuracy: 97.36842105263158%**. Change parameter and classifier to get accuracy of each case.

**Total mark for assessment: 3%. Complete and submit all questions: 2%, and lab attendance: 1%. Submit after 23:59 (midnight): -0.5% and -0.5% for each day after.**