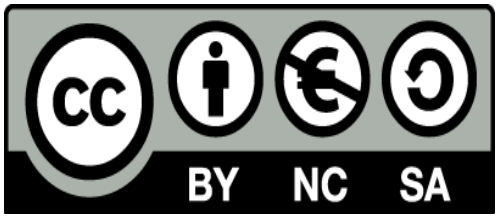
uc3m | Universidad Carlos III de Madrid



OPENCOURSEWARE
ADVANCED PROGRAMMING
STATISTICS FOR DATA SCIENCE

Ricardo Aler

MASTER IN STATISTICS FOR DATA SCIENCE. UC3M.

ADVANCED PROGRAMMING

2 HOURS

---

1. What is the expected result of this piece of code?

```
x=range(3)
y=x.append([4])
```

Answer:

It would raise an error because *range* does not return a list, but an iterator, and elements cannot be appended to iterators (they can be appended to lists).

2. What would be printed by this piece of code?

```
x = [1, 2, '3']
y = x
y[0:1]=x[1:2]
print(x)
```

Answer:

[2, 2, '3']

After y=x, y points/refers to x. y[0:1] refers to position 0 of y (0:1 is a range, and in Python, the last element in the range, 1 in this case, is excluded). x[1:2] refers to position 1 of x (which contains 2). Henceforth, this code puts 2 into the initial position of y, which is also the intial position of x.

Notice also that this is a Python list (not a numpy array), so it is ok for this list to contain elements belonging to different types (integers and strings in this case).

3. If *data* is a Pandas dataframe with a column named *'cn'*, would the following three options produce the same result? Why?

```
column_name = 'cn'

data[column_name] = 3             # option 1
data.column_name = 3              # option 2
data.loc[column_name] = 3         # option 3
```

Answer:

Option 1 would put number 3 into column named 'cn', as expected. In order for option 2 to work as option 1, the correct code would be data**.cn** = 3. Whatever goes after the '.' must be a column name, but not a variable containing the column name. In order for option 3 to work as option 1, the correct code would be data.loc[**:** , column_name]=3. That is, when loc is used in order to access a particular column, the range of rows must be specified first (in this case ':' means all rows).

4. What is the name of the Python concept that allows to do an operation like this:

```
np.array([1,2,3])+0
```

Answer:

Broadcasting. Broadcasting allows to operate two vectors (or matrices) of different sizes. In this case, Broadcasting allows to do np.array([1,2,3])+0 as if it were np.array([1,2,3])+ np.array([0,0,0]). While it is true that + is a universal function (element-wise + operation), without broadcasting it would not be possible to do the `np.array([1,2,3])+0` operation, so broadcasting is the main concept here.

5. In order to do automatic hyper-parameter tuning, it is necessary to define several elements. Describe all those elements

Answer:

- The search space (the hyper-parameters of the method and their allowed values or possible range of values)
- The search method: grid-search or random-search
- The evaluation method: basically, validation set (holdout) or crossvalidation
- The performance measure (or score function): missclassification error, balanced accuracy, RMSE, MAE, …

6. What would be printed by this piece of code?

```
data = pd.DataFrame({'a': [1,2,3,4,5,6], 'b':[4,5,6,7,8,9]})
print(data.loc[3:5,'a'])
```

Answer:

print(data.loc[3:5,'a'])
3 **4**
4 **5**
5 **6**
Name: a, dtype: int64

The reason is that we are using loc, not iloc. Iloc is for selecting rows via integer positions. loc is for selecting rows via row names (or column names, in the case of columns). The row names are contained in the index of the dataframe, which by default is just [0,1,2,3,4,5]

7. What would be the result of the following code?

```
mydictionary = {}
mydictionary[[1,2]] = 0
print(mydictionary)
```

Answer:

In principle, Python dictionaries can use any value as key, as far as they are non-mutable. If lists were non-mutable, this piece of code would just create a dictionary with key [1,2] and value 0. Something like this: {[1, 2]: 0}. However, Python dictionaries can only be non-mutable objects, therefore it would only work with a tuple like (1,2), but I've given half the marks to answers like this: {[1, 2]: 0}.

8. What would be the contents of variable *result* after executing this piece of code? Why?

```
import numpy as np
result = np.array([1,2,3]+[4,4,4])+6
```

Answer: array([7, 8, 9, 10, 10, 10])

[1,2,3]+[4,4,4] is an addition (+) of two base-Python lists, not numpy lists. For Python lists, the + operation actually concatenates the two lists, therefore [1,2,3]+[4,4,4] = [1,2,3, 4,4,4]. Finally, np.array([1,2,3]+[4,4,4])+6 is a summation of two numpy vectors, therefore the result would be array([7, 8, 9, 10, 10, 10]).

9. What would be the contents of variable *result* after executing this code?

```
result = 'abcdef'[5:1:-1]
```

Answer: 'fedc'

We start at position 5 (f) and go backwards down to position 1 (but excluding it).

10. Write two short pieces of code that do the same thing, but the first one uses a list, while the second one uses a tuple. The one for the list should work, the one for the tuple should be expected to raise an error.

Answer: the main difference between lists and tuples is that lists are mutable and tuples are non-mutable (they cannot be modified). So, something like this would work

a = [1,2,3]

a[0]=10

and something like this would not work:

a = (1,2,3)

a[0]=10

11. What follows is the code for defining a new transformer step for pipelines. Modify this code so that the transformer apparently does the same thing, but now with data-leakage. Explain why data-leakage is happening in your new code:

```
class f(TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        self.sum_ = np.nansum(X)
        return(self)
    def transform(self, X):
        for j in range(X.shape[0]):
            for i in range(X.shape[1]):
                if(np.isnan(X[i,j])):
                    X[i,j]=self.sum_
        return(X)
```

Answer: data-leakage happens when some information leaks from the test set to the training set, or in general, when some information is used where it shouldn't. In this case if X_test was a variable that contains the test set, we would get data-leakage by doing something like this (i.e. X_test, an external variable that contains the test set is used instead of X, which in a typical use of fit, would contain just the training partition):

```
    def fit(self, X, y=None):
        self.sum_ = np.nansum(X_test)
        return(self)
```

Another example of transformer that doesn't use properly the information is the following one. In this case, the test set is going to be imputed using information obtained from the test set (when doing the .transform), and it should be imputed with information obtained from the training set (when applying .fit). But the transformer below would impute the test set with information gathered from the test set, which is not correct. I've accepted other answers as correct if they were well justified.

```
class f(TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return(self)
    def transform(self, X):
        self.sum_ = np.nansum(X)
        for j in range(X.shape[0]):
            for i in range(X.shape[1]):
                if(np.isnan(X[i,j])):
                    X[i,j]=self.sum_
        return(X)
```

12. Stan returns posterior distributions. How does Stan represent them?

Answer:

Stan represents posterior distributions with lots of individual **samples** (out of which histograms and density plots can be constructed).

13. What is the main difference between *sort* and *sorted*?

Answer: sort modifies the list (it is a method), sorted does not (it is a function). Both are used to sort lists.

14. Why does Stan has a warmup period?

Answer: Stan works by iteratively sampling the posterior distribution, but the samples of the initial iterations are unstable and not very reliable, and they are usually discarded (that is the warmup period).

15. Write a piece of code that does the same thing than the one below, but it is much shorter:

```
mylist = [0,2,4,6]
result = []
for elem in mylist:
  result.append(elem+1)
print(result)
```

Answer:

[elem+1 for elem in mylist]

However, I've also accepted as solution *np.array(mylist)+1* because in fact it does the same thing as the initial code and it is also shorter.