# LESSON PLAN

## LESSON #5

**Aim:** Data reading and selection

**Objective:** After a lesson on reading and selection of data, students will be able to use Python to perform the same procedures on two other datasets.

**Do Now:** Statistically Funny https://www.youtube.com/watch?v=UASWcDsH_nM

**Standards:** 9-12. CT.2 Computational Thinking, Data Analysis, and Visualization

9-12. CT.3 Computational Thinking, Data Analysis, and Visualization

9-12. CT.10 Computational Thinking, Algorithms, and Programming

**Mini-Lesson:**

1. Manipulating data
   a. List of most commonly used aggregation function

| Function | Description |
|---|---|
| count() | Number of non-null observations |
| sum() | Sum of values |
| mean() | Mean of values |
| median() | Arithmetic median of values |
| min() | Minimum |
| max() | Maximum |
| prod() | Product of values |
| std() | Unbiased standard deviation |
| var() | Unbiased variance |

   b. The above functions can either be applied on the rows for each column (axis = 0) or it should be applied on the columns of each row (axis = 1)
   c. The difference between pandas max function and standard Python max function
   d. Applying operations over all the values in rows, columns, or a selection or both. We can apply any binary arithmetical operations on an entire row.

e. Applying any function to a data frame or series by just setting its name as an argument of the apply method. Example: applying the sqrt function from the Numpy library to perform the square root of each value in the Value column
f. Using an in-line-function or λ-function
g. Setting values in a data frame. Warning: Use a new column name!
h. Dropping the column or row by using the drop function by setting axis = 1 or axis = 0, respectively. Note: If you do not want to keep the old data frame values set inplace = True
i. Inserting a row at the bottom of the data frame
j. Removing a row
k. The drop function and removing missing values
l. Removing missing values
m. Filling/replacing missing values

2. Sorting Data
   a. Sorting a column in a data frame by column in descending order
   b. Sorting a column in ascending order, while over writing the data frame
3. Conclusion and summary

   Discussion:
   - What are some of the problems or challenges you encountered?
   - How did you resolve them?
   - What did you learn from this lesson?
   - Do you have any lingering questions on today's lesson or data science in general?

# CODE

## Data Manipulation

```
edu.max(axis = 0)
```

```
TIME          2011
GEO          Spain
Value         8.81
dtype: object
```

**A Word of Note**

```
print "Pandas max function:", edu['Value'].max()
print "Python max function:", max(edu['Value'])
```

```
Pandas max function: 8.81
Python max function: nan
```

## *Apply aggregate functions over all the values in rows, columns, or a selection of both*

```
s = edu["Value"]/100
s.head()
```

```
0          NaN
1          NaN
2       0.0500
3       0.0503
4       0.0495
Name: Value, dtype: float64
```

## *Using the apply method*

```
s = edu["Value"].apply(np.sqrt)
s.head()
```

```
0           NaN
1           NaN
2      2.236068
3      2.242766
4      2.224860
Name: Value, dtype: float64
```

### *Using an in-line-function or λ-function*

```
s = edu["Value"].apply(lambda d: d**2)
s.head()
```

```
0           NaN
1           NaN
2       25.0000
3       25.3009
4       24.5025
Name: Value, dtype: float64
```

### *Setting new values in a data frame*

```
edu['ValueNorm'] = edu['Value']/edu['Value'].max()
edu.tail()
```

| | TIME | GEO | Value | ValueNorm |
|---|---|---|---|---|
| 379 | 2007 | Finland | 5.90 | 0.669694 |
| 380 | 2008 | Finland | 6.10 | 0.692395 |
| 381 | 2009 | Finland | 6.81 | 0.772985 |
| 382 | 2010 | Finland | 6.85 | 0.777526 |
| 383 | 2011 | Finland | 6.76 | 0.767310 |

### *Deleting a column or row*

```
edu.drop('ValueNorm', axis = 1, inplace = True)
edu.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 0 | 2000 | European Union (28 countries) | NaN |
| 1 | 2001 | European Union (28 countries) | NaN |
| 2 | 2002 | European Union (28 countries) | 5 |
| 3 | 2003 | European Union (28 countries) | 5.03 |
| 4 | 2004 | European Union (28 countries) | 4.95 |

### *Inserting a new row at the bottom of the data frame*

```
edu = edu.append({"TIME": 2000,"Value": 5.00,"GEO": 'a'},
                 ignore_index = True)
edu.tail()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 380 | 2008 | Finland | 6.1 |
| 381 | 2009 | Finland | 6.81 |
| 382 | 2010 | Finland | 6.85 |
| 383 | 2011 | Finland | 6.76 |
| 384 | 2000 | a | 5 |

### *Removing a row*

```
edu.drop(max(edu.index), axis = 0, inplace = True)
edu.tail()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 379 | 2007 | Finland | 5.9 |
| 380 | 2008 | Finland | 6.1 |
| 381 | 2009 | Finland | 6.81 |
| 382 | 2010 | Finland | 6.85 |
| 383 | 2011 | Finland | 6.76 |

### *The drop function and removing missing values*

```
eduDrop = edu.drop(edu["Value"].isnull(), axis = 0)
eduDrop.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 2 | 2002 | European Union (28 countries) | 5.00 |
| 3 | 2003 | European Union (28 countries) | 5.03 |
| 4 | 2004 | European Union (28 countries) | 4.95 |
| 5 | 2005 | European Union (28 countries) | 4.92 |
| 6 | 2006 | European Union (28 countries) | 4.91 |

### *Removing missing values*

```
eduDrop = edu.dropna(how = 'any', subset = ["Value"])
eduDrop.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 2 | 2002 | European Union (28 countries) | 5.00 |
| 3 | 2003 | European Union (28 countries) | 5.03 |
| 4 | 2004 | European Union (28 countries) | 4.95 |
| 5 | 2005 | European Union (28 countries) | 4.92 |
| 6 | 2006 | European Union (28 countries) | 4.91 |

### *Filling/replacing missing values*

```
eduFilled = edu.fillna(value = {"Value": 0})
eduFilled.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 0 | 2000 | European Union (28 countries) | 0.00 |
| 1 | 2001 | European Union (28 countries) | 0.00 |
| 2 | 2002 | European Union (28 countries) | 5.00 |
| 3 | 2003 | European Union (28 countries) | 4.95 |
| 4 | 2004 | European Union (28 countries) | 4.95 |

## Sorting Data

### *Sorting a column in a data frame by column in descending order*

```
edu.sort_values(by = 'Value', ascending = False,
                inplace = True)
edu.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 130 | 2010 | Denmark | 8.81 |
| 131 | 2011 | Denmark | 8.75 |
| 129 | 2009 | Denmark | 8.74 |
| 121 | 2001 | Denmark | 8.44 |
| 122 | 2002 | Denmark | 8.44 |

### *Sorting a column in ascending order, while over writing the data frame*

```
edu.sort_index(axis = 0, ascending = True, inplace = True)
edu.head()
```

| | TIME | GEO | Value |
|---|---|---|---|
| 0 | 2000 | European Union ... | NaN |
| 1 | 2001 | European Union ... | NaN |
| 2 | 2002 | European Union ... | 5.00 |
| 3 | 2003 | European Union ... | 5.03 |
| 4 | 2004 | European Union ... | 4.95 |