APPLIED MACHINE LEARNING SDU

Lecture 2

Python Basics for Data Analysis
SciPy + NumPy + Pandas + Matplotlib

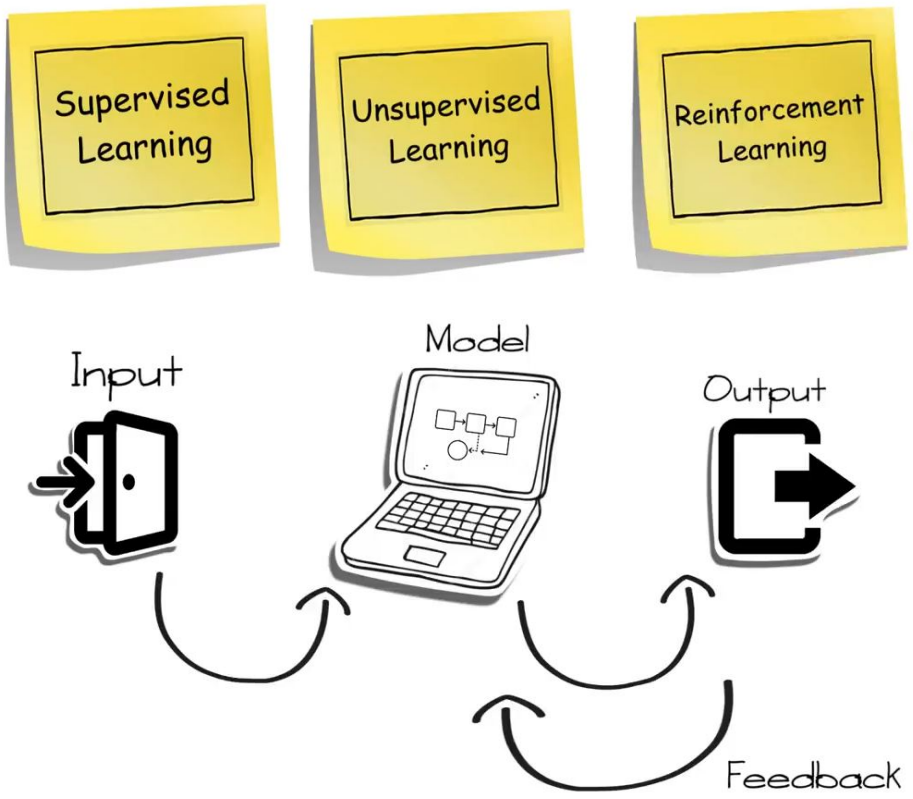
13 February 2023
Rahman Peimankar

Agenda

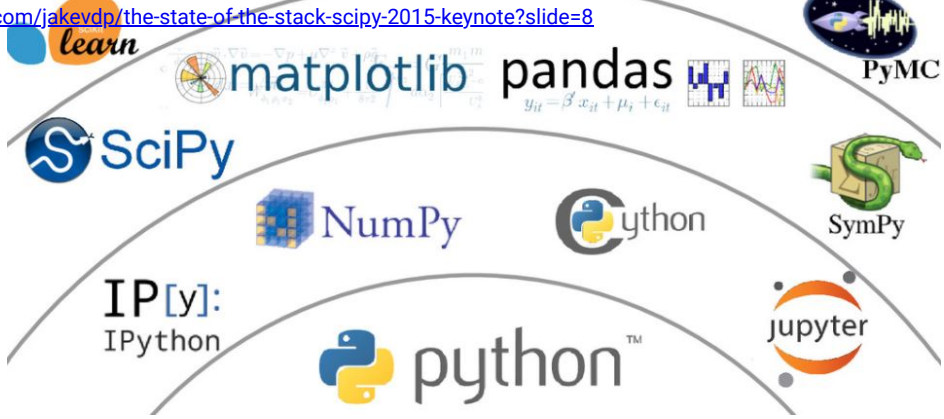
- Why we use Python as a programming language for ML development
- Python Crash course
- NumPy package
- Matplotlib package
- Pandas package
- Understand Your Data With Visualization

Recap of Last Week

There are many ways in which a machine learns:



https://speakerdeck.com/jakevdp/the-state-of-the-stack-scipy-2015-keynote?slide=8

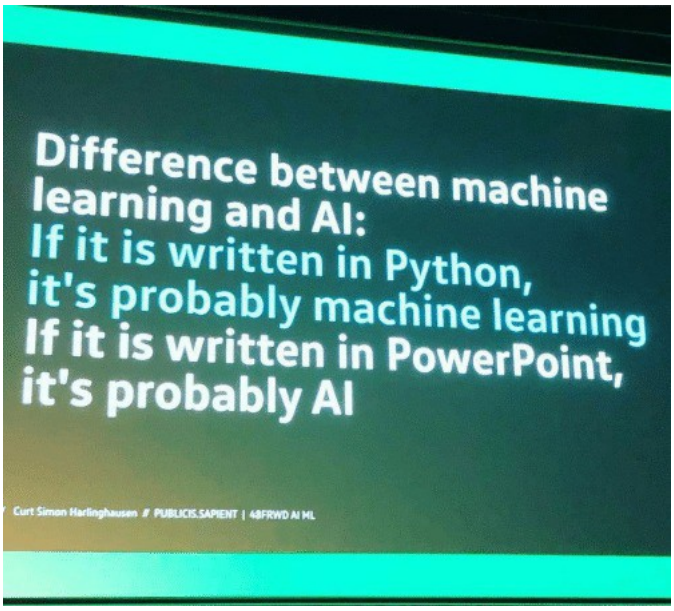


Learn Machine Learning The Wrong Way

- Get really good at Python programming
- Start with the theory
- Study all of machine learning
- Implement everything from scratch

Python Ecosystem for Machine Learning

- Dominant platform for machine learning
- A general purpose programming language (unlike R and Matlab)
- Implement whatever you need



Python

- Easy to learn and use!
- The philosophy of Python are:
    - Beautiful is better than ugly.
    - Explicit is better than implicit.
    - Simple is better than complex.
    - Complex is better than complicated.
    - Flat is better than nested.
    - Sparse is better than dense.

- Readability counts.

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

https://www.python.org/dev/peps/pep-0020/

SciPy

- Python libraries for mathematics, science and engineering.
- An add-on to Python that you will need for machine learning.
- The SciPy ecosystem is comprised of the following core modules relevant to machine learning:
    1. **NumPy**: work with data in arrays.
    2. **Matplotlib**: create 2D charts and plots from data.
    3. **Pandas**: Tools and data structures to organize and analyze your data.

And **scikit-learn** library is how you can develop and practice machine learning in Python.

Ckeck the Version of Your Packages

```
# scipy
import scipy
print('scipy: {}'.format(scipy.__version__))
# numpy
import numpy
print('numpy: {}'.format(numpy.__version__))
# matplotlib
import matplotlib
print('matplotlib: {}'.format(matplotlib.__version__))
# pandas
import pandas
print('pandas: {}'.format(pandas.__version__))
# scikit-learn
import sklearn
print('sklearn: {}'.format(sklearn.__version__))
```

```
scipy: 1.7.2
numpy: 1.18.5
matplotlib: 3.5.0
pandas: 1.2.4
sklearn: 1.0.1
```

Python Crash Course

A few key details about the language syntax such as:

- Assignment
- Flow Control
- Data Structures

- Functions

1. Assignment

String

```
data = 'Hello World'
print(data)
```

```
Hello World
```

Numbers

```
Float = 123.1
print(Float)
Integer = 10
print(Integer)
type(Float)
```

```
123.1
10
float
```

Boolean

```
a = True
b = False
print(a, b)
type(a)
```

```
True False
bool
```

Multiple Assignment

```
a, b, c = 1, 2, 3
print(a, b, c)
```

```
1 2 3
```

2. Flow Control

If-Then-Else Conditional

```
speed = 50
if  200 > speed > 99:
    print('That is fast')
elif speed > 200:
    print('That is too fast')
else:
    print('That is safe')
```

```
That is safe
```

For-Loop

```
for i in range(6):
    print(i)
```

```
0
1
2
3
4
5
```

While-Loop

```
# While-Loop
i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

3. Data Structures

There are three data structures in Python **tuples**, **lists** and **dictionaries**.

Tuple

Tuples are read-only collections of items.

```
a = (1, 2, 3)
print(a)
type(a)
```

```
(1, 2, 3)
tuple
```

List

Lists use the square bracket notation and can be index using array notation.

```
mylist = [1, 2, 3]
print("Zeroth Value: {}".format(mylist[0]))
mylist.append(4)
print("List Length: {}".format(len(mylist)))
for value in mylist:
    print(value)
```

```
Zeroth Value: 1
List Length: 4
1
2
3
4
```

Dictionary

Dictionaries are mappings of names to values, like key-value pairs.

```
mydict = {'a': 1, 'b': 2, 'c': 3}
print("A value: {}".format(mydict['a']))
mydict['a'] = 11
print("A value: {}".format(mydict['a']))
print("Keys: {}".format(mydict.keys()))
print("Values: {}".format(mydict.values()))
for key in mydict.keys():
    print(mydict[key])
mydictict
```

```
A value: 1
A value: 11
Keys: dict_keys(['a', 'b', 'c'])
Values: dict_values([11, 2, 3])
11
2
3
{'a': 11, 'b': 2, 'c': 3}
```

4. Functions

The example below defines a new function to calculate the sum of two values and calls the function with two arguments.

```
# Sum function
def mysum(x, y):
    return x + y
# Test sum function
result = mysum(1, 3)
print(result)
```

```
4
```

NumPy Crash Course

NumPy provides the foundation data structures and operations for SciPy. These are arrays (ndarrays) that are efficient to defne and manipulate.

1. Create Array

```
import numpy as np
mylist = [1, 2, 3]
myarray = np.array(mylist)
print(myarray)
print(myarray.shape)
```

```
[1 2 3]
(3,)
```

2. Access Data

```
import numpy
mylist = [[1, 2, 3], [3, 4, 5]]
myarray = numpy.array(mylist)
print(myarray)
print(myarray.shape)
print("First row: {}".format(myarray[0]))
print("Last row: {}".format(myarray[1]))
print("Specific row and col: {}".format(myarray[0, 2]))
print("Whole col: {}".format(myarray[:, 2]))
```

```
[[1 2 3]
 [3 4 5]]
(2, 3)
First row: [1 2 3]
Last row: [3 4 5]
Specific row and col: 3
Whole col: [[2]
 [4]]
```

3. Arithmetic

```
myarray1 = numpy.array([2, 2, 1])
myarray2 = numpy.array([3, 3, 3])
print("Addition: {}".format(myarray1 + myarray2))
print("Multiplication: {}".format(myarray1 * myarray2))
```

```
Addition: [5 5 4]
Multiplication: [6 6 3]
Multiplication: [-1 -1 -2]
```

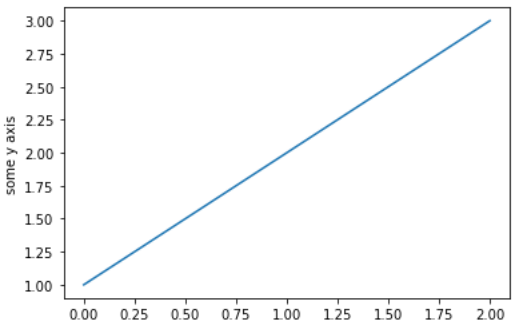
Matplotlib Crash Course

Matplotlib can be used for creating plots and charts. The library is generally used as follows:

- Call a plotting function with some data (e.g. **.plot()**).
- Call many functions to setup the properties of the plot (e.g. labels and colors).
- Make the plot visible (e.g. **.show()**).

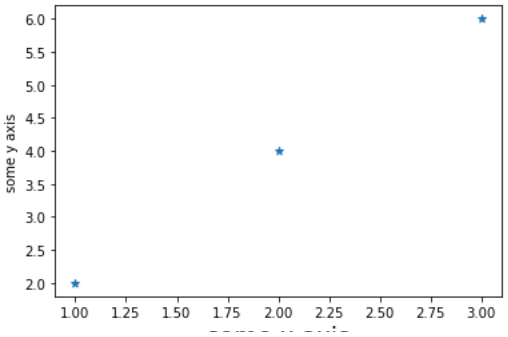
1. Line Plot

```
import matplotlib.pyplot as plt
import numpy
myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```



2. Scatter Plot

```
x = numpy.array([1, 2, 3])
y = numpy.array([2, 4, 6])
plt.scatter(x,y, marker='*')
plt.xlabel('some x axis', fontsize=18)
plt.ylabel('some y axis')
plt.show()
```



Pandas Crash Course

- Pandas provides data structures and functionality to quickly manipulate and analyze data.
- The key to understanding Pandas for machine learning is understanding the Series and DataFrame data structures.

1. Series

A series is a **one dimensional array** where the rows and columns can be labeled.

```
import pandas
myarray = numpy.array([1, 2, 3])
rownames = ['a', 'b', 'c']
myseries = pandas.Series(myarray, index=rownames)
print(myseries)
```

```
aa    1
bb    2
cc    3
dtype: int32
pandas.core.series.Series
```

You can access the data in a series like a NumPy array and like a dictionary, for example:

```
print(myseries[0])
print(myseries['a'])
```

```
1
1
```

2. DataFrame

A data frame is a multi-dimensional array where the rows and the columns can be labeled.

```
myarray = numpy.array([[1, 2, 3], [4, 5, 6]])
rownames = ['a', 'b']
colnames = ['one', 'two', 'three']
mydataframe = pandas.DataFrame(myarray, index=rownames, columns=colnames)
print(mydataframe)
type(mydataframe)
```

```
   one  two  three
a    1    2      3
b    4    5      6
pandas.core.frame.DataFrame
```

```
print("method 1:")
print("one column:\n{}".format(mydataframe['one']))
print("method 2:")
print("one column:\n{}".format(mydataframe.two))
```

```
method 1:
one column:
a    1
b    4
Name: one, dtype: int32
method 2:
one column:
a    2
b    5
Name: two, dtype: int32
a    1
Name: one, dtype: int32
```

Summary

We have covered a lot of Python basics so far. We learned basic syntax and usage of three key Python libraries used for machine learning:

- NumPy
- Matplotlib
- Pandas

Next, we are going to know how to easily and quickly load standard ML datasets to start our projects.

How To Load Machine Learning Data

- Load CSV Files with the Python Standard Library.
- Load CSV Files with NumPy.
- Load CSV Files with Pandas.

Considerations When Loading CSV Data

1. File Header?
    - If yes, these can be assigned automatically as names of column in the dataset
    - If no, we may need to name our attributes manually.
2. Comments?
    - Comments in a CSV file are indicated by a hash (#) at the start of a line.
    - We may need to indicate whether or not to expect comments and the character to expect to signify a comment line.
3. Delimiter
    - The standard delimiter that separates values in fields is the comma (,) character.
    - Your file could use a different delimiter like **tab** or **white space** in which case you must specify it explicitly.
4. Quotes

- Sometimes field values can have spaces. In these CSV files the values are often quoted.
- The default quote character is the double quotation marks character.
- Other characters can be used, and you must specify the quote character used in your file.

Pima Indians Dataset

- This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes withinfive years.
- It is classification problem.
- A good dataset for demonstration since it has only numeric inputs and the output variable to be predicted is binary (0 or 1).
- The dataset is freely available at https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

1. Load CSV Files with the Python Standard Library

```
import csv
import numpy
filename = 'diabetes.csv'
with open('diabetes.csv') as f:
    reader = csv.reader(f)
    x = list(reader)
data = numpy.array(x)
print(data.shape)
```

```
(769, 9)
```

For more information about the **csv.reader()** method, see here:
https://docs.python.org/2/library/csv.html

2. Load CSV Files with NumPy

You can load your CSV data using NumPy and the numpy.loadtxt() function. This function assumes no header row and all data has the same format.

```
from numpy import loadtxt

filename = 'diabetes.csv'
with open('diabetes.csv') as f:
    data = loadtxt(f, delimiter=",", skiprows=1)
print(data.shape)
```

```
(768, 9)
```

3. Load CSV Files with Pandas

You can load your CSV data using Pandas and the **pandas.read_csv()** function. The function returns a pandas DataFrame that you can immediately start summarizing and plotting. You can learn more about **pandas.read_csv()** function here:
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

```
from pandas import read_csv
filename = 'diabetes.csv'
colNmaes = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=colNmaes)
print(data.shape)
```

(769, 9)

	preg	plas	pres	skin	test	mass	
0	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
1	6	148	72	35	0	33.6	
2	1	85	66	29	0	26.6	
3	8	183	64	0	0	23.3	
4	1	89	66	23	94	28.1	
...	...	...	...	...	...	...	
764	10	101	76	48	180	32.9	

Now, let's look at at the data itself and what is insde it.

766	5	121	72	23	112	26.2

Understand Your Data With Descriptive Statistics

Some common ways to underestand your ML data:

- Take a peek at your raw data.
- Review the dimensions of your dataset.
- Review the data types of attributes in your data.
- Summarize the distribution of instances across classes in your dataset.
- Summarize your data using descriptive statistics.
- Understand the relationships in your data using correlations.
- Review the skew of the distributions of each attribute.

1. Peek at Your Data

- Looking at the raw data can reveal insights that you cannot get any other way.
- It can also plant seeds that may later grow into ideas on how to better handle the data

```
from pandas import read_csv
filename = "diabetes.csv"
#names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename)
data.tail(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Di
758	1	106	76	0	0	37.5	
759	6	190	92	0	0	35.5	
760	2	88	58	26	16	28.4	
761	9	170	74	31	0	44.0	
762	9	89	62	0	0	22.5	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	

2. Dimensions of Your Data

- Too many rows and algorithms may take too long to train.
- Too few rows and perhaps you do not have enough data to train the algorithms.
- Too many features (columns) and some algorithms can be distracted or suffer poor performance due to the curse of dimensionality.

```
from pandas import read_csv
filename = "diabetes.csv"
data = read_csv(filename)
shape = data.shape
```

```
print(shape)
data.head(5)
```

```
(768, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	

3. Data Type For Each Attribute

- The type of each attribute is important.
- Strings may need to be converted to floating point values or integers to represent categorical or ordinal values.

```
from pandas import read_csv
filename = "diabetes.csv"
data = read_csv(filename)
types = data.dtypes
types
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

4. Descriptive Statistics

- The **describe()** function on the Pandas DataFrame lists 8 statistical properties of each attribute.

```
from pandas import read_csv
from pandas import set_option
filename = "diabetes.csv"
data = read_csv(filename)
set_option('precision', 3)
description = data.describe()
description
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BM
count	768.000	768.000	768.000	768.000	768.000	768.00
mean	3.845	120.895	69.105	20.536	79.799	31.99
std	3.370	31.973	19.356	15.952	115.244	7.88
min	0.000	0.000	0.000	0.000	0.000	0.00
25%	1.000	99.000	62.000	0.000	0.000	27.30
50%	3.000	117.000	72.000	23.000	30.500	32.00
75%	6.000	140.250	80.000	32.000	127.250	36.60

5. Class Distribution (Classification Only)

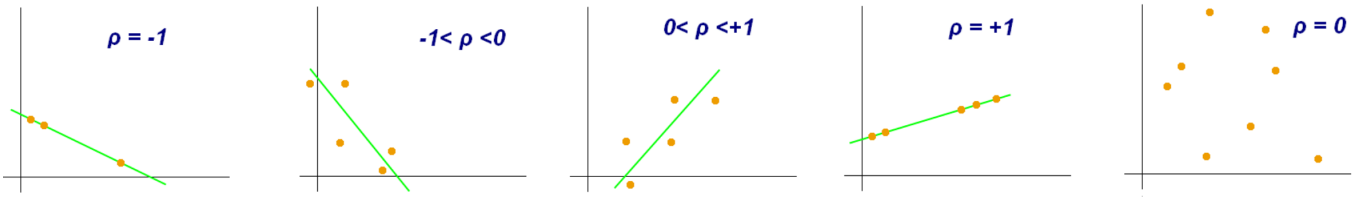
- On classification problems you need to know how balanced the class values are.
- Highly imbalanced problems are common and may need special handling in the data preparation stage of your project.

```
from pandas import read_csv
filename = "diabetes.csv"
data = read_csv(filename)
class_counts = data.groupby('Outcome').size()
print(class_counts)
```

```
Outcome
0    500
1    268
dtype: int64
(268, 9)
```

6. Correlations Between Attributes

- Relationship between two variables and how they may or may not change together.
- The most common method for calculating correlation is Pearson's Correlation Coefficient.




```
from pandas import read_csv
from pandas import set_option
filename = "diabetes.csv"
data = read_csv(filename)
set_option('precision', 3)
correlations = data.corr(method='pearson')
correlations
```

	Pregnancies	Glucose	BloodPressure	SkinThickne
Pregnancies	1.000	0.129	0.141	-0.0
Glucose	0.129	1.000	0.153	0.0
BloodPressure	0.141	0.153	1.000	0.2
SkinThickness	-0.082	0.057	0.207	1.0
Insulin	-0.074	0.331	0.089	0.4
BMI	0.018	0.221	0.282	0.3
DiabetesPedigreeFunction	-0.034	0.137	0.041	0.1
Age	0.544	0.264	0.240	-0.1

7. Skew of Univariate Distributions

Skew refers to a distribution that is assumed Gaussian (normal or bell curve) that is shifted or squashed in one direction or another.

```
from pandas import read_csv
filename = "diabetes.csv"
data = read_csv(filename)
skew = data.skew()
print(skew)
```

```
Pregnancies                0.902
Glucose                    0.174
BloodPressure             -1.844
SkinThickness              0.109
Insulin                    2.272
BMI                       -0.429
DiabetesPedigreeFunction   1.920
Age                        1.130
Outcome                    0.635
dtype: float64
```

Tips To Remember

- **Review the numbers:** Generating the summary statistics is not enough. Take a moment to pause, read and really think about the numbers you are seeing.
- **Ask why:** Review your numbers and ask a lot of questions. How and why are you seeing specific numbers.
- **Write down ideas:** Write down your observations and ideas. Keep a small text file or note pad and write down all of the ideas for how variables may relate, for what numbers mean, and ideas for techniques to try later.

Understand Your Data With Visualization

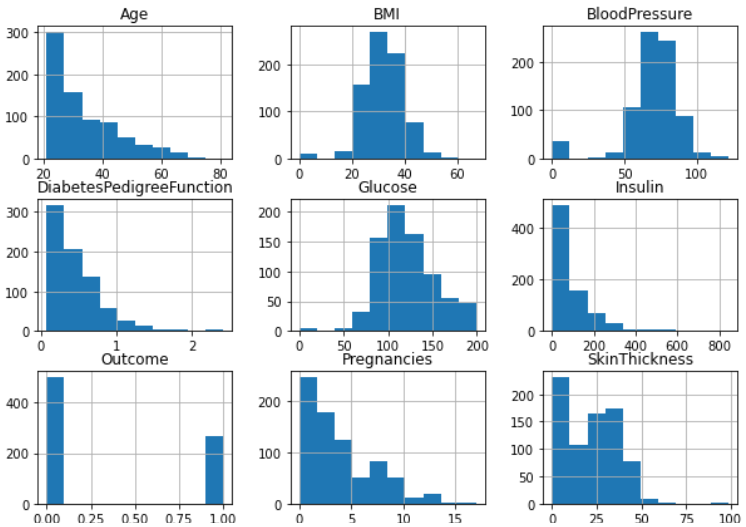
You must understand your data in order to get the best results from machine learning algorithms.

- Univariate Plots
    1. Histograms
    2. Density Plots
    3. Box and Whisker Plots
- Multivariate Plots
    4. Correlation Matrix Plot
    5. Scatter Plot Matrix

1. Histograms

A fast way to get an idea of the distribution of each attribute.

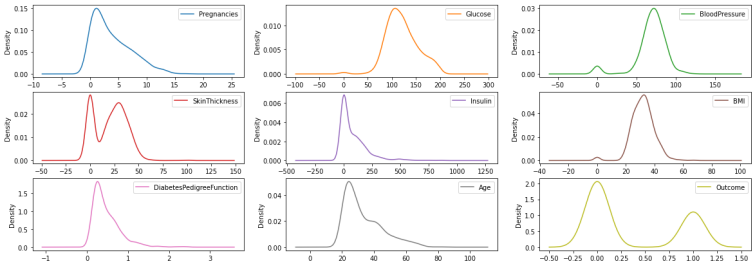
```
from matplotlib import pyplot as plt
from pandas import read_csv
filename = 'diabetes.csv'
data = read_csv(filename)
data.hist(figsize=(10, 7))
plt.show()
```



2. Density Plots

Density plots show a quick overview of the distribution of each attribute.

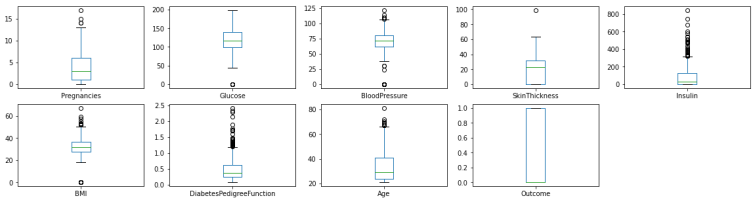
```
from matplotlib import pyplot
from pandas import read_csv
filename = 'diabetes.csv'
data = read_csv(filename)
data.plot(kind='density', subplots=True, layout=(3,3), sharex=False, figsize=(20,7))
pyplot.show()
```



3. Box and Whisker Plots

- Another way to review the distribution of each attribute.
- Boxplots draws a line for the median (mid. value) and a box around the 25^{th} and 75^{th} percentiles.

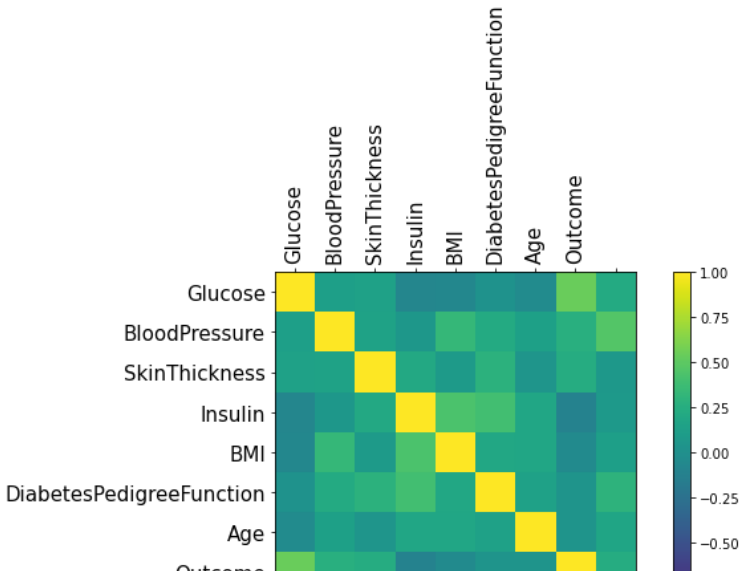
```
from matplotlib import pyplot
from pandas import read_csv
filename = "diabetes.csv"
data = read_csv(filename)
data.plot(kind='box', subplots=True, layout=(2,5), sharex=False, sharey=False, figsize=(20,5))
pyplot.show()
```



4. Correlation Matrix Plot

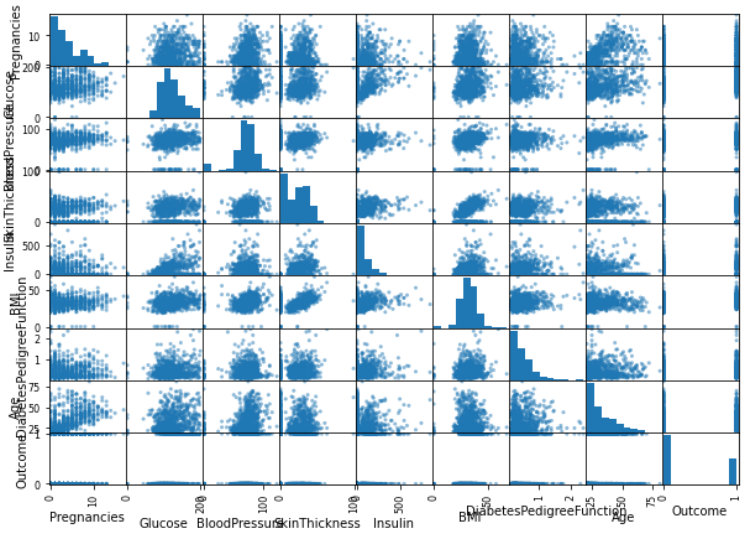
Some machine learning algorithms can have poor performance if there are highly correlated input variables in your data.

```
from matplotlib import pyplot
from pandas import read_csv
import numpy
filename = 'diabetes.csv'
data = read_csv(filename)
correlations = data.corr()
# plot correlation matrix
fig = pyplot.figure(figsize=(10, 5))
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ax.set_xticklabels(data.columns, rotation=90, fontsize=15)
ax.set_yticklabels(data.columns, fontsize=15)
pyplot.show()
```



5. Scatter Plot Matrix

```
from matplotlib import pyplot
from pandas import read_csv
from pandas.plotting import scatter_matrix
filename = "diabetes.csv"
data = read_csv(filename)
axes = scatter_matrix(data, figsize=(10,7))
# [ax.set_xlabel(ax.get_xlabel(), fontsize = 10, rotation = 90) for ax in axes.ravel()]
# [ax.set_ylabel(ax.get_ylabel(), fontsize = 10, rotation = 0) for ax in axes.ravel()]
pyplot.show()
```



Feedback during the semester.

Please provide your anonymous feedback here.

Thank you!