***Data Science Nigeria: Taking Artificial Intelligence to every city in Nigeria***

# *AI Invasion 2019 Curriculum of Study*

***March 25 to April 26, 2019***

Data Science Nigeria is a non-profit driven by a vision to build a **world-class Artificial Intelligence knowledge, research and innovation ecosystem** that delivers high impact research, business use applications, locally-relevant AI-first start-ups, increase employability and drive for social good use cases. Our drive is to accelerate Nigeria's development through the solution-oriented application of machine learning to solving social/business problems and to galvanize a data science knowledge revolution. We believe that in 10 years, Nigeria will become one of the top 10 AI talent/knowledge destinations with **20% GDP multiplier impact** to attain a **1 million AI talents in 10 years mandate**

**Our Successes**

- 300,000+ direct download of our ***Artificial Intelligence for Starter free ebook*** to stimulate interest in AI through use cases and inspirational contribution from leading AI experts including recognized leaders like Prof Yoshua Bengio

- 10,212 participated in the 1st ever Intercampus Machine Learning competition that involved 95 universities and a great opportunity to incentivize Nigerian students to learn Introduction to Machine Learning with Python, R and MS Azure
- 12,234 online participants in Data Science course – from registration to at least one class
- Over 1,000 have participated in our face to face classes via fully residential all-expense-paid bootcamp, free meet-ups and weekly classes at the AI Hub
- 203 direct job placement, project participation and internship
- 4 innovative product ideas that are being pre-incubated
- The largest convergence of academia and industry practitioners as mentors to support real-world application of learning
- High impact learning bootcamps, academic engagement and direct job placements (full time, freelance and internships) for young Nigerian data scientists
- Strategic partnership with leading firms and organisations including Microsoft, Bill and Melinda Gates Foundation, KPMG, Access Bank, African Fintech Foundry, BlueChip Technology, Nigerian Bureau of Statistics, Terragon Group, Proshare Nigeria, FCMB Bank, Guardian Newspapers, Diamond Bank, InstaDeep, PayLater OneFi, Keoun, Axa Mansar, Interswitch etc.

# 30 Days AI Invasion Curriculum

## Day 1 – Introduction to Machine Learning Part 1

- What is Machine Learning?
- But what it does actually or why do we need ML or AI?
- Types of Machine Learning algorithms
- Python Master Sheet

## Day 2 – Introduction to Machine Learning Part 2

- Jupyter Notebooks
- Machine Learning and Concept of Maths and Programming
- Datasets for Machine Learning
- Some more Machine Learning Resources

## Day 3– Introduction to Regression Analysis

- Regression Analysis
- Concept of Linear Regression
- A Little Bit About the Math
- Activity 1: Predicting Boston Housing Prices

## Day 4 – Assumptions of Regression and Tree-based Regression Model

- Assumptions of Linear Regression
- Non- Linear Regression
- Tree Based Regression
- Activity 2: Predicting Boston Housing Prices with Advanced Regression Techniques

## Day 5 - Regression and classification Comparison

- Comparison of Regression and classification
- Binary classification

- Multi-Class Classification
- Algorithms for Classification
- Activity 3 – Titanic Passenger Survival Prediction
- What is Clustering
- Use of Clustering Algorithms
- K-means Clustering
- Hierarchical Clustering
- More Clustering Algorithms
- Activity 4 – Clustering on Breast Cancer Data and World Happiness Data

## AI Invasion Curriculum - Extras

End of Course

# AI Invasion Curriculum - Day 1

## What is Machine Learning?

Machine Learning is a branch of Artificial Intelligence (AI), which helps making machines capable of learning from observational data without being explicitly programmed. It's not like that, Machine learning or ML and Artificial Intelligence or AI are the same. There are a lot of differences as well a lot of similarities. Alternatively, we can also say that, Machine learning is a component of AI.

Tom Mitchell gave a clearer definition of Machine Learning:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*

## But what it does actually or why do we need ML or AI?

Machine Learning and Artificial Intelligence (AI) are based on automated and self-training algorithm to learn from prior data in order to find the pattern, which exists within, and then help machines to make decisions in situations they have never seen.

Machine learning is expanding its reach to various domains.

# Interdisciplinary field


Spam Detection: When Google Mail detects your spam mail automatically, it is as a result of applying machine learning techniques.

Link for more Reading

Credit Card Frauds: Identifying 'unusual' activities on a credit card is often a machine-learning problem involving anomaly detection.

Link for more Reading

Product Recommendation: Whether Netflix was suggesting you watch House of Cards or Amazon was insisting you finally buy those Bose headphones, it's not magic, but machine learning at work.

Link for more Reading

Medical Diagnosis: Using a database of symptoms and treatments of patients, a popular machine learning problem is to predict if a patient has a particular illness.

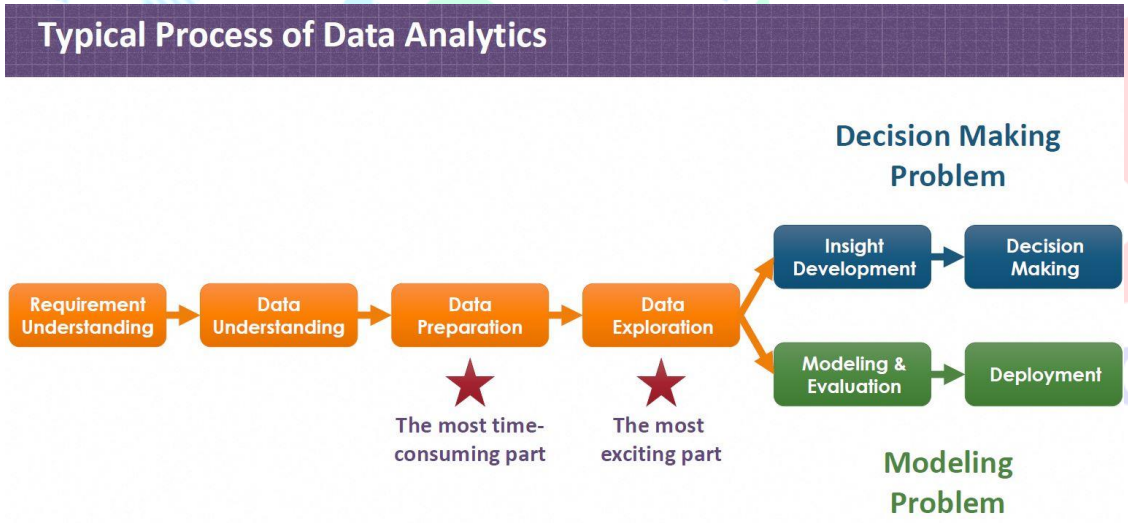
Link for more Reading

Face Detection: When Facebook automatically recognizes faces of your friends in a photo, a machine learning process is what's running in the background.

Link for more Reading

Customer Segmentation: Using the data for usage during a trial period of a product, identifying the users who will go onto subscribe for the paid version is a learning problem.

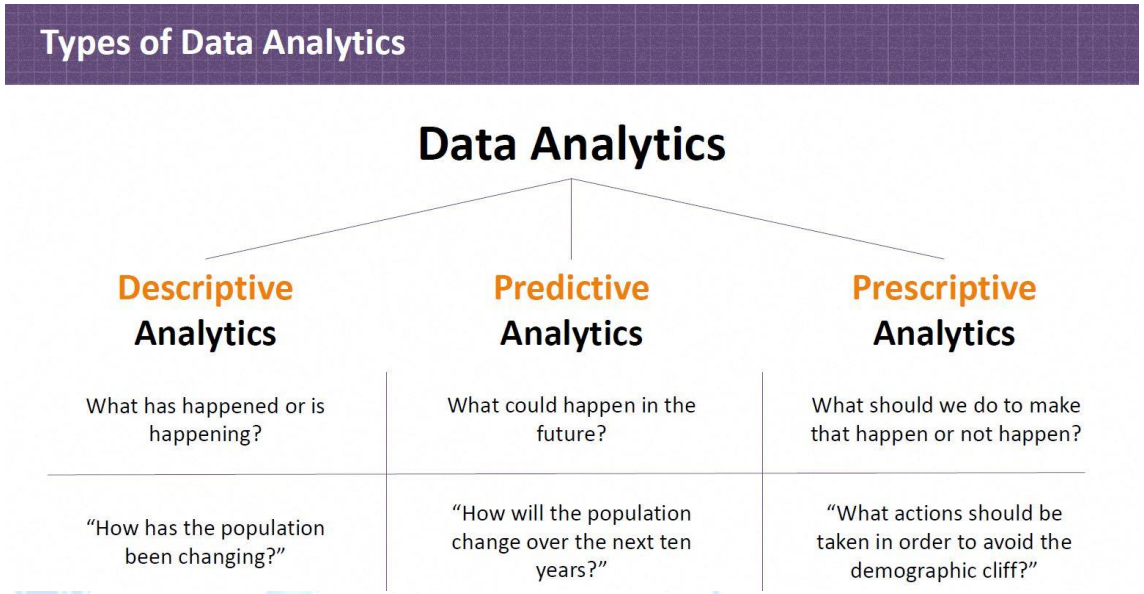
Link for more Reading



## Types of Machine Learning algorithms

All of the applications of Machine Learning mentioned above differ from each other. For example, the customer segmentation problem *clusters* the customers into two segments - customers who will pay or the others who won't. Another example like, face recognition problem aims to *classify* a face. So, we can say that, there are broadly two types of machine learning algorithms under which a number of algorithms are present respectively.

- **Supervised Learning**: It consists of Classification and Regression Algorithms. The target variable is known and so it is called supervised learning. For example, when you rate a movie after you view it on Netflix, the suggestion which follows is predicted using a database of your ratings (known as the training data). When the problem is based on continuous variables, (maybe predicting the stock price) then it falls under *regression*. With class labels (such as in the Netflix problem), the learning problem is called a *classification* problem.
- **Unsupervised Learning**: It consists of Clustering Algorithms. The target variable is unknown and so it is called un-supervised learning. In the case of unsupervised learning, there is no labelled data set which can be used for making further predictions. The learning algorithm tries to find patterns or associations in the given data set. Identifying clusters in the data (as in the customer segmentation problem) and reducing dimensions of the data falls in the unsupervised learning category.




So, that was a quick introduction to Machine Learning and different types of Machine Learning algorithms and a few applications as well. In this course, we will be exploring all that about Machine Learning and Artificial Intelligence. Python and its open source technologies will be used as the primary programming language within this course.

Python is the world largest growing programming language and until before 2007, there existed no built-in machine learning package in Python when David Cournapeau (Developer of Numpy/Scipy) developed **Scikit-learn** as a part of a Google Summer of Code project. The project now has 30 contributors and many sponsors including Google and Python Software Foundation. Scikit-learn provides a range of supervised and unsupervised learning algorithms via a Python interface. In the same way TensorFlow also built by google provides machines the power to handle deep learning stuff. We will be exploring all these packages and libraries one by one in this course from the grassroot level, by first learning how to represent data in scikit-learn and then building highly optimised and efficient machine learning models.

# Python Master Sheet

## Python Cheat Sheet

JUST THE BASICS

Created By: Arianne Colton and Sean Chen

### General

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

#### HELP

| | |
|---|---|
| Help Home Page | `help()` |
| Function Help | `help(str.replace)` |
| Module Help | `help(re)` |

#### MODULE (AKA LIBRARY)

Python module is simply a '.py' file

| | |
|---|---|
| List Module Contents | `dir(module1)` |
| Load Module | `import module1` * |
| Call Function from Module | `module1.func1()` |

* import statement creates a new namespace and executes all the statements in the associated *.py* file within that namespace. If you want to load the module's content into current namespace, use `'from module1 import *'`

### Scalar Types

**Check data type :** `type(variable)`

#### SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via \ or preface with r

```
str1 = r'this\f?ff'
```

   - String formatting can be done in a number of ways

```
template = '%.2f %s haha $%d';
str1 = template % (4.88, 'hola', 2)
```

### Scalar Types

* str(), bool(), int() and float() are also explicit type cast functions.

5. **NoneType(None)** - Python 'null' value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments :

```
def func1(a, b, c = None)
```

   - Common usage of None :

```
if variable is None :
```

6. **datetime** - built-in python 'datetime' module provides 'datetime', 'date', 'time' types.
   - 'datetime' combines information stored in 'date' and 'time'

| | |
|---|---|
| Create datetime from String | `dt1 = datetime.strptime('20091031', '%Y%m%d')` |
| Get 'date' object | `dt1.date()` |
| Get 'time' object | `dt1.time()` |
| Format datetime to String | `dt1.strftime('%m/%d/%Y %H:%M')` |
| Change Field Value | `dt2 = dt1.replace(minute = 0, second = 30)` |
| Get Difference | `diff = dt1 - dt2`<br># diff is a 'datetime.timedelta' object |

**Note :** Most objects in Python are mutable except for 'strings' and 'tuples'

### Data Structures

**Note :** All non-Get function call i.e. `list1.sort()` examples below are in-place (without creating a new object) operations unless noted otherwise.

#### TUPLE

One dimensional, fixed-length, **immutable** sequence of Python objects of ANY type.

| | |
|---|---|
| Create Tuple | `tup1 = 4, 5, 6` or `tup1 = (6,7,8)` |
| Create Nested Tuple | `tup1 = (4,5,6), (7,8)` |
| Convert Sequence or Iterator to Tuple | `tuple([1, 0, 2])` |
| Concatenate Tuples | `tup1 + tup2` |
| Unpack Tuple | `a, b, c = tup1` |

**Application of Tuple**

| | |
|---|---|
| Swap variables | `b, a = a, b` |

### Data Structures

| | |
|---|---|
| Create Tuple | `tup1 = 4, 5, 6` or `tup1 = (6,7,8)` |
| Create Nested Tuple | `tup1 = (4,5,6), (7,8)` |
| Convert Sequence or Iterator to Tuple | `tuple([1, 0, 2])` |
| Concatenate Tuples | `tup1 + tup2` |
| Unpack Tuple | `a, b, c = tup1` |

**Application of Tuple**

| | |
|---|---|
| Swap variables | `b, a = a, b` |

#### LIST

One dimensional, variable length, **mutable** (i.e. contents can be modified) sequence of Python objects of ANY type.

| | |
|---|---|
| Create List | `list1 = [1, 'a', 3]` or `list1 = list(tup1)` |
| Concatenate Lists* | `list1 + list2` or `list1.extend(list2)` |
| Append to End of List | `list1.append('b')` |
| Insert to Specific Position | `list1.insert(posIdx, 'b')` ** |
| Inverse of Insert | `valueAtIdx = list1.pop(posIdx)` |
| Remove First Value from List | `list1.remove('a')` |
| Check Membership | `3 in list1 => True` *** |
| Sort List | `list1.sort()` |
| Sort with User-Supplied Function | `list1.sort(key = len)`<br># sort by length |

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, `extend()` is preferable.

** Insert is computationally expensive compared with append.

*** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

**Built-in 'bisect module‡**

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insort' actually inserts into that location.

‡ WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

#### SLICING FOR SEQUENCE TYPES†

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

| | |
|---|---|
| Notation | `list1[start:stop]` |
| | `list1[start:stop:step]` (If step is used) § |

**Note :**
- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

§ Application of 'step' :

| | |
|---|---|
| Take every other element | `list1[::2]` |
| Reverse a string | `str1[::-1]` |

#### DICT (HASH MAP)

| | |
|---|---|
| Create Dict | `dict1 = {'key1' :'value1', 2 :[3, 2]}` |
| Create Dict from Sequence | `dict(zip(keyList, valueList))` |
| Get/Set/Insert Element | `dict1['key1']`*<br>`dict1['key1'] = 'newValue'` |
| Get with Default Value | `dict1.get('key1', defaultValue)`** |
| Check if Key Exists | `'key1' in dict1` |
| Delete Element | `del dict1['key1']` |
| Get Key List | `dict1.keys()` *** |
| Get Value List | `dict1.values()` *** |
| Update Values | `dict1.update(dict2)`<br># dict1 values are replaced by dict2 |

* 'KeyError' exception if the key does not exist.

** 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

**Valid dict key types**

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string'), hash([1, 2])` - this would fail.

#### SET

- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

| | |
|---|---|
| Create Set | `set([3, 6, 3])` or `{3, 6, 3}` |
| Test Subset | `set1.issubset(set2)` |
| Test Superset | `set1.issuperset(set2)` |
| Test sets have same content | `set1 == set2` |

- **Set operations :**

| | |
|---|---|
| Union(aka 'or') | `set1 \| set2` |
| Intersection (aka 'and') | `set1 & set2` |
| Difference | `set1 - set2` |
| Symmetric Difference (aka 'xor') | `set1 ^ set2` |

# Python For Data Science *Cheat Sheet*

## PySpark Basics

Learn Python for data science Interactively at www.DataCamp.com

## Spark

**PySpark** is the Spark Python API that exposes the Spark programming model to Python

## Initializing Spark

### SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

### Inspect SparkContext

```
>>> sc.version                  Retrieve SparkContext version
>>> sc.pythonVer                Retrieve Python version
>>> sc.master                   Master URL to connect to
>>> str(sc.sparkHome)           Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())         Retrieve name of the Spark User running SparkContext
>>> sc.appName                  Return application name
>>> sc.applicationId            Retrieve application ID
>>> sc.defaultParallelism       Return default level of parallelism
>>> sc.defaultMinPartitions     Default minimum number of partitions for RDDs
```

### Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

### Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

## Loading Data

### Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

### External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

## Retrieving RDD Information

### Basic Information

```
>>> rdd.getNumPartitions()          List the number of partitions
>>> rdd.count()                     Count RDD instances
 3
>>> rdd.countByKey()                Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()              Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()              Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()                      Sum of RDD elements
 4950
>>> sc.parallelize([]).isEmpty()    Check whether RDD is empty
 True
```

### Summary

```
>>> rdd3.max()                      Maximum value of RDD elements
  99
>>> rdd3.min()                      Minimum value of RDD elements
  0
>>> rdd3.mean()                     Mean value of RDD elements
  49.5
>>> rdd3.stdev()                    Standard deviation of RDD elements
  28.866070047722118
>>> rdd3.variance()                 Compute variance of RDD elements
  833.25
>>> rdd3.histogram(3)               Compute histogram by bins
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                    Summary statistics (count, mean, stdev, max & min)
```

## Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))          Apply a function to each RDD element
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element and flatten the result

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)           Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

## Selecting Data

### Getting

```
>>> rdd.collect()                   Return a list with all RDD elements
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                     Take first 2 RDD elements
  [('a', 7), ('a', 2)]
>>> rdd.first()                     Take first RDD element
  ('a', 7)
>>> rdd.top(2)                      Take top 2 RDD elements
  [('b', 2), ('a', 7)]
```

### Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()   Return sampled subset of rdd3
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

### Filtering

```
>>> rdd.filter(lambda x: "a" in x)  Filter the RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()       Return distinct RDD values
  ['a',2,'b',7]
>>> rdd.keys().collect()            Return (key,value) RDD's keys
  ['a', 'a', 'b']
```

## Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)                  Apply a function to all RDD elements
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

## Reshaping Data

### Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)     Merge the rdd values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)        Merge the rdd values
  ('a',7,'a',2,'b',2)
```

### Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)         Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()                      Group rdd by key
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

### Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)    Aggregate RDD elements of each partition and then the results
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)   Aggregate values of each RDD key
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)                      Aggregate the elements of each partition, and then the results
  4950
>>> rdd.foldByKey(0, add)                 Merge the values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)             Create tuples of RDD elements by applying a function
       .collect()
```

## Mathematical Operations

```
>>> rdd.subtract(rdd2)                    Return each rdd value not contained in rdd2
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)               Return each (key,value) pair of rdd2 with no matching key in rdd
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()         Return the Cartesian product of rdd and rdd2
```

## Sort

```
>>> rdd2.sortBy(lambda x: x[1])           Sort RDD by given function
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()                      Sort (key, value) RDD by key
       .collect()
  [('a',2),('b',1),('d',1)]
```

## Repartitioning

```
>>> rdd.repartition(4)              New RDD with 4 partitions
>>> rdd.coalesce(1)                 Decrease the number of partitions in the RDD to 1
```

## Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

## Stopping SparkContext

```
>>> sc.stop()
```

## Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

edureka!

# PYTHON FOR DATA SCIENCE CHEAT SHEET

Learn Python for Data Science at www.edureka.co

## Scikit-learn

**Scikit-learn** is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train,X_test,y_train,y_test= train_test_split(X,y,random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

## Loading The Data

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

## Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)
```

PYTHON FOR DATA SCIENCE

## Create Your Model

### Supervised Learning Estimators

**Linear Regression**

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

**Support Vector Machines (SVM)**

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

**Naive Bayes**

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

**KNN**

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### Unsupervised Learning Estimators

**K Means**

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

**Principal Component Analysis (PCA)**

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

## Model Fitting

**Supervised learning**

```
>>> lr.fit(X, Y)
>>> knn.fit(X_train, Y_train)
>>> svc.fit(X_train, Y_train)
```

#Fit the model to the data

**Unsupervised Learning**

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

#Fit the model to the data
#Fit to data, then transform it

## Prediction

**Supervised Estimators**

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test))
```

#Predict labels
#Predict labels
#Estimate probability of a label

**Unsupervised Estimators**

```
>>> y_pred = k_means.predict(X_test)
```

#Predict labels in clustering algos

## Evaluate Your Model's Performance

### Classification Metrics

**Accuracy Score**

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

#Estimator score method

**Classification Report**

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

#Metric scoring functions

**Confusion Matrix**

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

#Precision, recall, f1-score and support

### Regression Metrics

**Mean Absolute Error**

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

**Mean Squared Error**

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

**R² Score**

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

### Clustering Metrics

**Adjusted Rand Index**

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

**Homogeneity**

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

**V-measure**

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

### Cross-Validation

**Adjusted Rand Index**

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

## Tune Your Model

### Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3), "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

### Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5), "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

## Scikit-learn

### Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

### Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

### Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

### Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

### Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

### Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

# Python For Data Science *Cheat Sheet*

## NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

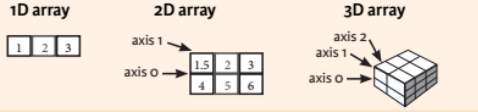
## NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

### NumPy Arrays




## Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

### Initial Placeholders

| | |
|---|---|
| `>>> np.zeros((3,4))` | Create an array of zeros |
| `>>> np.ones((2,3,4),dtype=np.int16)` | Create an array of ones |
| `>>> d = np.arange(10,25,5)` | Create an array of evenly spaced values (step value) |
| `>>> np.linspace(0,2,9)` | Create an array of evenly spaced values (number of samples) |
| `>>> e = np.full((2,2),7)` | Create a constant array |
| `>>> f = np.eye(2)` | Create a 2X2 identity matrix |
| `>>> np.random.random((2,2))` | Create an array with random values |
| `>>> np.empty((3,2))` | Create an empty array |

## I/O

### Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

### Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

## Data Types

| | |
|---|---|
| `>>> np.int64` | Signed 64-bit integer types |
| `>>> np.float32` | Standard double-precision floating point |
| `>>> np.complex` | Complex numbers represented by 128 floats |
| `>>> np.bool` | Boolean type storing `TRUE` and `FALSE` values |
| `>>> np.object` | Python object type |
| `>>> np.string_` | Fixed-length string type |
| `>>> np.unicode_` | Fixed-length unicode type |

## Inspecting Your Array

| | |
|---|---|
| `>>> a.shape` | Array dimensions |
| `>>> len(a)` | Length of array |
| `>>> b.ndim` | Number of array dimensions |
| `>>> e.size` | Number of array elements |
| `>>> b.dtype` | Data type of array elements |
| `>>> b.dtype.name` | Name of data type |
| `>>> b.astype(int)` | Convert an array to a different type |

## Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

## Array Mathematics

### Arithmetic Operations

```
>>> g = a - b
array([[-0.5,  0. ,  0. ],
       [-3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
array([[ 2.5,  4. ,  6. ],
       [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
array([[ 0.66666667,  1.        ,  1.        ],
       [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
array([[  1.5,   4. ,   9. ],
       [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
array([[ 7.,  7.],
       [ 7.,  7.]])
```

Subtraction

Subtraction
Addition

Addition
Division

Division
Multiplication

Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

### Comparison

```
>>> a == b
array([[False,  True,  True],
       [False, False, False]], dtype=bool)
>>> a < 2
array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison

Element-wise comparison

Array-wise comparison

### Aggregate Functions

| | |
|---|---|
| `>>> a.sum()` | Array-wise sum |
| `>>> a.min()` | Array-wise minimum value |
| `>>> b.max(axis=0)` | Maximum value of an array row |
| `>>> b.cumsum(axis=1)` | Cumulative sum of the elements |
| `>>> a.mean()` | Mean |
| `>>> b.median()` | Median |
| `>>> a.corrcoef()` | Correlation coefficient |
| `>>> np.std(b)` | Standard deviation |

## Copying Arrays

| | |
|---|---|
| `>>> h = a.view()` | Create a view of the array with the same data |
| `>>> np.copy(a)` | Create a copy of the array |
| `>>> h = a.copy()` | Create a deep copy of the array |

## Sorting Arrays

| | |
|---|---|
| `>>> a.sort()` | Sort an array |
| `>>> c.sort(axis=0)` | Sort the elements of an array's axis |

## Subsetting, Slicing, Indexing

Also see Lists

### Subsetting

```
>>> a[2]
3
```
Select the element at the 2nd index

```
>>> b[1,2]
6.0
```
Select the element at row 0 column 2 (equivalent to `b[1][2]`)

### Slicing

```
>>> a[0:2]
array([1, 2])
```
Select items at index 0 and 1

```
>>> b[0:2,1]
array([ 2.,  5.])
```
Select items at rows 0 and 1 in column 1

```
>>> b[:1]
array([[1.5, 2., 3.]])
```
Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]
array([[[ 3.,  2.,  1.],
        [ 4.,  5.,  6.]]])
```
Same as `[1,:,:]`

```
>>> a[ : :-1]
array([3, 2, 1])
```
Reversed array `a`

### Boolean Indexing

```
>>> a[a<2]
array([1])
```
Select elements from `a` less than 2

### Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
array([ 4. , 2. , 6. , 1.5])
```
Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
array([[ 4. ,5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5],
       [ 4. , 5. , 6. , 4. ],
       [ 1.5, 2. , 3. , 1.5]])
```
Select a subset of the matrix's rows and columns

## Array Manipulation

### Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```
Permute array dimensions
Permute array dimensions

### Changing Array Shape

```
>>> b.ravel()
```
Flatten the array

```
>>> g.reshape(3,-2)
```
Reshape, but don't change data

### Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```
Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

### Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
array([ 1,  2,  3, 10, 15, 20])
```
Concatenate arrays

```
>>> np.vstack((a,b))
array([[ 1. ,  2. ,  3. ],
       [ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```
Stack arrays vertically (row-wise)

```
>>> np.r_[e,f]
>>> np.hstack((e,f))
array([[ 7.,  7.,  1.,  0.],
       [ 7.,  7.,  0.,  1.]])
```
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)

```
>>> np.column_stack((a,d))
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
```
Create stacked column-wise arrays

```
>>> np.c_[a,d]
```
Create stacked column-wise arrays

### Splitting Arrays

```
>>> np.hsplit(a,3)
[array([1]),array([2]),array([3])]
```
Split the array horizontally at the 3rd index

```
>>> np.vsplit(c,2)
[array([[[ 1.5,  2. ,  1. ],
        [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]]])]
```
Split the array vertically at the 2nd index

# Python For Data Science *Cheat Sheet*

## Matplotlib

Learn Python Interactively at www.DataCamp.com

### Matplotlib

**Matplotlib** is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

## 1 Prepare The Data

Also see Lists & NumPy

### 1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

### 2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

## 2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

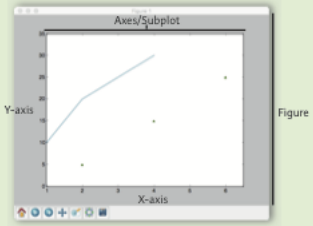
### Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

## Plot Anatomy & Workflow

### Plot Anatomy




### Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3,4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

## 4 Customize Plot

### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

### Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

### Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

### Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

### Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

### Limits, Legends & Layouts

#### Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

#### Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

#### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

#### Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

#### Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

## 3 Plotting Routines

### 1D Data

```
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and 0
```

### Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot 2D vector fields
```

### Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

### 2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(X,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

## 5 Save Plot

### Save figures

```
>>> plt.savefig('foo.png')
```

### Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

## 6 Show Plot

```
>>> plt.show()
```

## Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

DataCamp
Learn Python for Data Science Interactively

# Python For Data Science *Cheat Sheet*

## Pandas

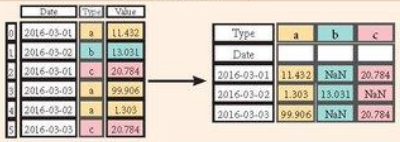
Learn Python for Data Science Interactively at www.DataCamp.com

## Reshaping Data

### Pivot

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

Spread rows into columns



### Pivot Table

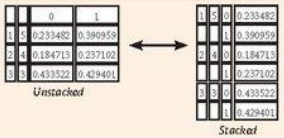
```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

Spread rows into columns

### Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels



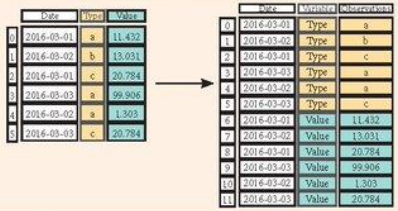
*Unstacked*

*Stacked*

### Melt

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

Gather columns into rows



## Iteration

```
>>> df.iteritems()
>>> df.iterrows()
```

(Column-index, Series) pairs
(Row-index, Series) pairs

## Advanced Indexing

Also see NumPy Arrays

### Selecting

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

Select cols with any vals >1
Select cols with vals > 1
Select cols with NaN
Select cols without NaN

### Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

Find same elements
Filter on values
Select specific elements

### Where

```
>>> s.where(s > 0)
```

Subset the data

### Query

```
>>> df6.query('second > first')
```

Query DataFrame

## Setting/Resetting Index

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

Set the index
Reset the index
Rename DataFrame

## Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

### Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

| | Country | Capital | Population |
|---|---|---|---|
| 0 | Belgium | Brussels | 11190846 |
| 1 | India | New Delhi | 1303171035 |
| 2 | Brazil | Brasilia | 207847528 |
| 3 | Brazil | Brasilia | 207847528 |

### Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

| | |
|---|---|
| 0 | 3 |
| 1 | 3 |
| 2 | 3 |
| 3 | 3 |
| 4 | 3 |

## MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

## Duplicate Data

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

## Grouping Data

### Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                              'b': np.sum})
```

### Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

## Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

Drop NaN values
Fill NaN values with a predetermined value
Replace values with others

## Combining Data

*data1*

| | X1 | X2 |
|---|---|---|
| a | 11.432 | |
| b | 1.303 | |
| c | 99.906 | |

*data2*

| | X1 | X3 |
|---|---|---|
| a | 20.784 | |
| b | NaN | |
| d | 20.784 | |

### Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

| | X1 | X2 | X3 |
|---|---|---|---|
| a | 11.432 | | 20.784 |
| b | 1.303 | | NaN |
| c | 99.906 | | NaN |

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

| | X1 | X2 | X3 |
|---|---|---|---|
| a | 11.432 | | 20.784 |
| b | 1.303 | | NaN |
| d | NaN | | 20.784 |

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

| | X1 | X2 | X3 |
|---|---|---|---|
| a | 11.432 | | 20.784 |
| b | 1.303 | | NaN |

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

| | X1 | X2 | X3 |
|---|---|---|---|
| a | 11.432 | | 20.784 |
| b | 1.303 | | NaN |
| c | 99.906 | | NaN |
| d | NaN | | 20.784 |

### Join

```
>>> data1.join(data2, how='right')
```

### Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

## Dates

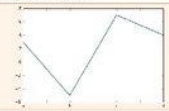
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

## Visualization

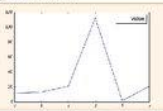
Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



## Links and Resources to Learn in Day 1:

All the links provided are for better and enhanced learning. These all are free resources. Learning should never end, we assume you will go through a few of these specifically to understand the basic concept explained in this lesson. A few of them are courses which are of longer duration, you can continue with them along with the course.

1. Artificial Intelligence, Revealed a quick introduction by Yann LeCun, mostly about Machine Learning ideas, Deep Learning, and convolutional neural network
2. Intro to Machine Learning - Udacity hands on scikit-learn (python) programming learning on core ML concepts
3. Machine Learning: Supervised, Unsupervised & Reinforcement - Udacity
4. Machine Learning Mastery very carefully laid out step-by-step guide to some particular algorithms.
5. Andrew Ng's Course on Coursera, one of the best courses for Machine Learning and Deep Learning in the world exploring all the maths and theories behind it)
6. Machine Learning is Fun Part 1 simple approach to machine learning for non-maths people
7. Machine Learning with Python - Video Playlist

# AI Invasion Curriculum - Day 2

Today, we will be exploring the various open source libraries for data loading, data manipulation and exploratory data visualization.

The best all in one IDE for learning ML and Data Science with python is Jupyter notebook which comes from Anaconda Distribution. Jupyter Notebook provides highly interactive notebook type interface where inputs as well as the outputs are displayed on the same cell.

## Jupyter Notebooks

Here is a video link, explaining the use and importance of Jupyter Notebook. To install Jupyter Notebook, go to this link, and download Anaconda, the one which matches your OS requirements (Win/Mac/Linux). We will be using Python 3.6 version. Jupyter Notebooks have a lot of advantages which are the following:

1) **High-Performance Distribution:** Easily install 1,000+ data science packages.
2) **Package Management:** Manage packages, dependencies and environments with Conda.
3) **Portal to Data Science:** Uncover insights in your data and create interactive visualizations.
4) **Interactive:** Coding and Visualization

**Machine Learning and Concept of Maths and Programming:** To master Machine Learning (ML) one has to be good at maths and programming.



## Getting Started With Jupyter Notebook for Python

In the following tutorial, you will be guided through the process of installing Jupyter Notebook. Furthermore, we'll explore the basic functionality of Jupyter Notebook and you'll be able to try out first examples.

This is at the same time the beginning of a series of Python-related tutorial on CodingTheSmartWay.com. From the very beginning you'll learn everything to need to know to use Python for scientific computing and machine learning use cases.

Jupyter Notebook is a web application that allows you to create and share documents that contain:

- live code (e.g. Python code)
- visualizations
- explanatory text (written in markdown syntax)

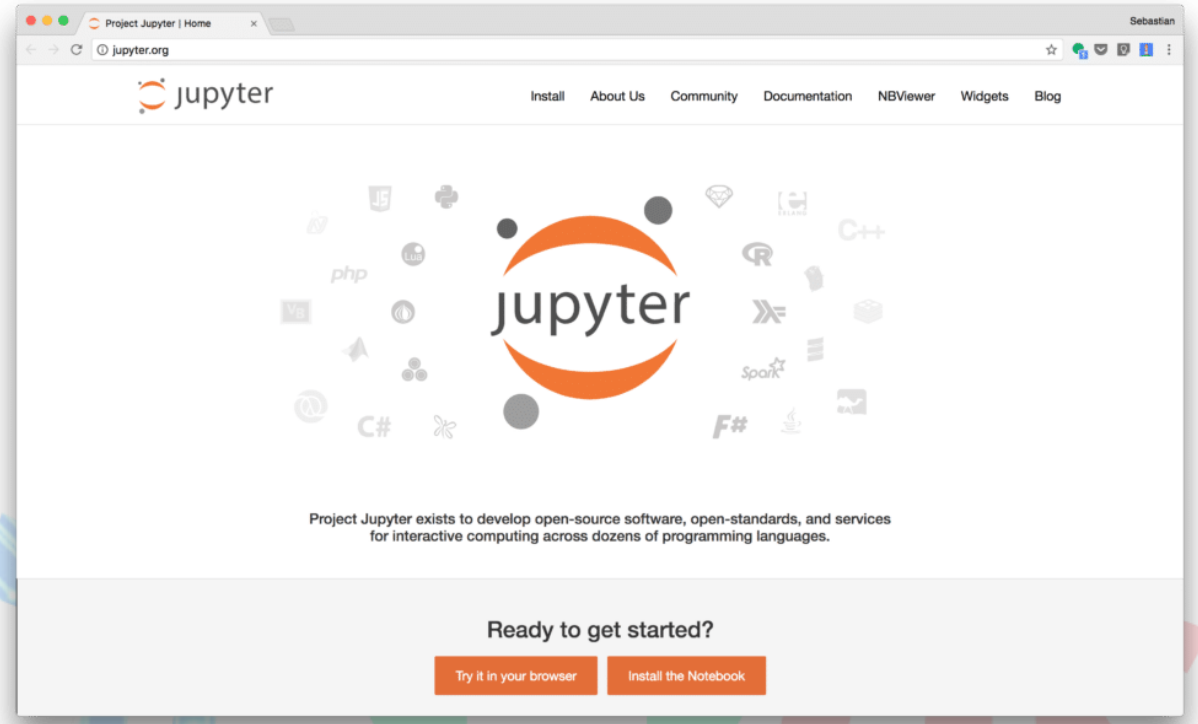
Jupyter Notebook is great for the following use cases:

- learn and try out Python
- data processing / transformation
- numeric simulation
- statistical modeling
- machine learning

Let's get started and install Jupyter Notebook on your computer …

## Setting up Jupyter Notebook

The first step to get started is to visit the project's website at http://www.jupyter.org:




Here you'll find two options:

- Try it in your browser
- Install the Notebook

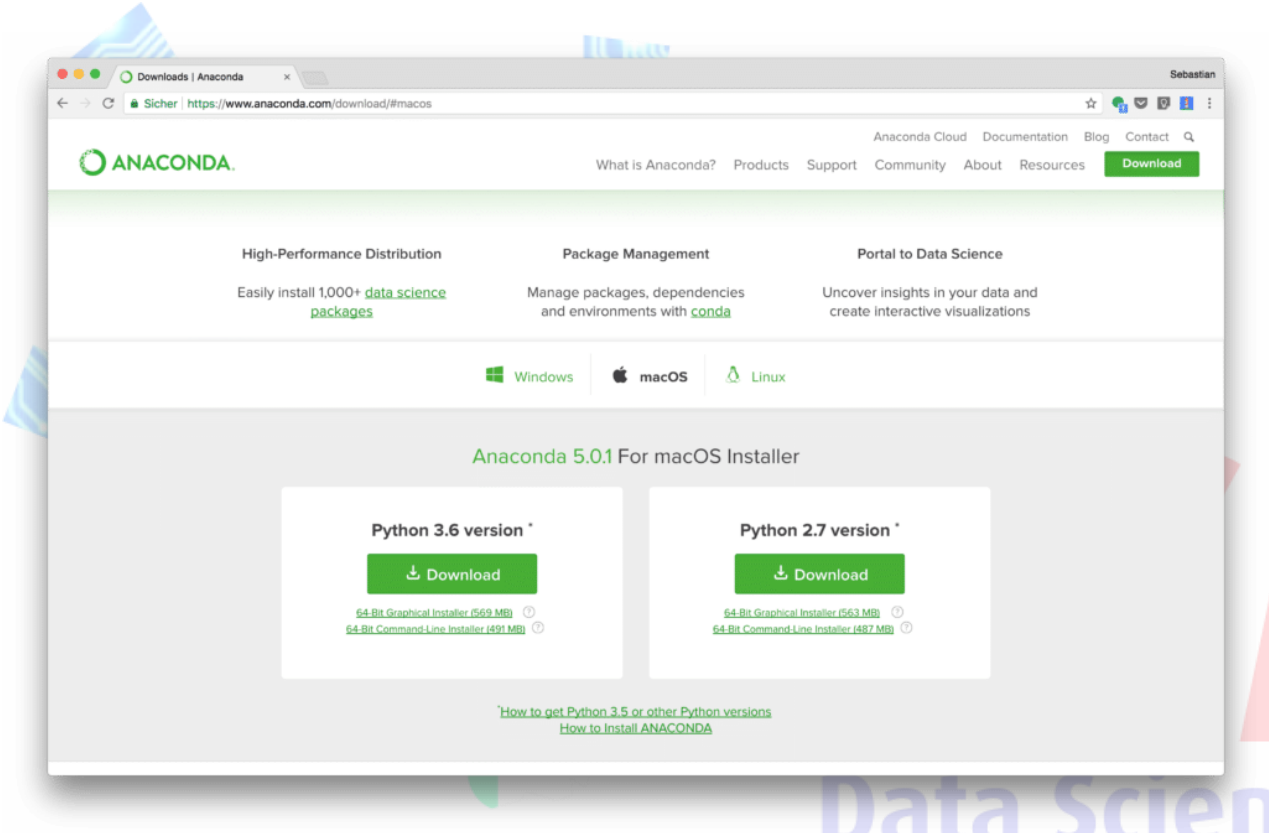
With the first option *Try it in your browser* you can access a hosted version of Jupyter Notebook. This will get you direct access without needing to install it on your computer.

The second option *Install the Notebook* will take you to another page which gives you detailed instruction for the installation. There are two different ways:

- Installing Jupyter Notebook by using the Python's package manager *pip*

- Installing Jupyter Notebook by installing the Anaconda distribution

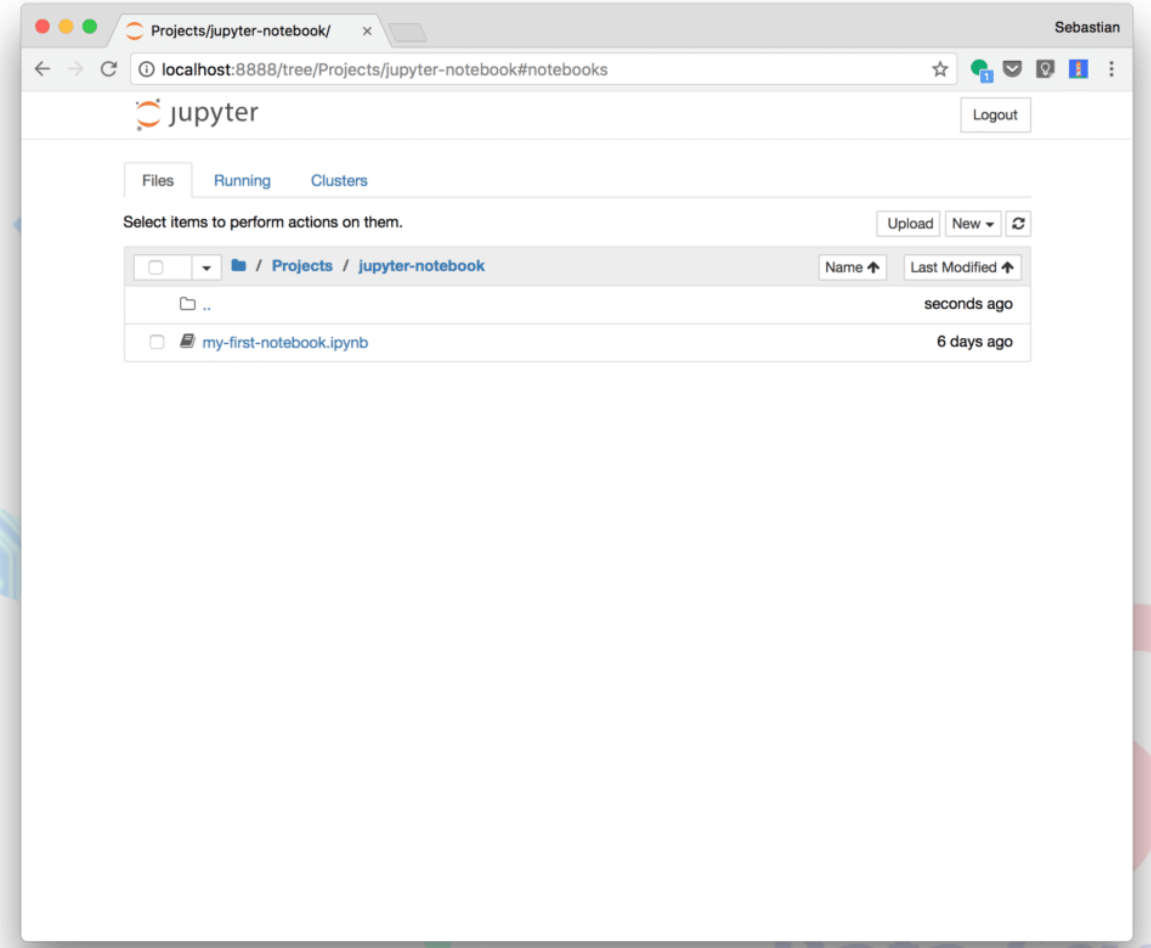
Especially if you're new to Python and would like to set up your development environment from scratch using the Anaconda distribution is a great choice. If you follow the link (https://www.anaconda.com/download/) to the Anaconda download page you can choose between installers for Windows, macOS, and Linux:






Download and execute the installer of your choice. Having installed the Anaconda distribution we can now start Jupyter Notebook by clicking the Jupyter n

The web server is started and the Jupyter Notebook application is opened in your default browser automatically. You should be able to see a browser output, which is similar to the following screenshot:






As you can see the user interface of Jupyter Notebook is split up into three sections (tabs):

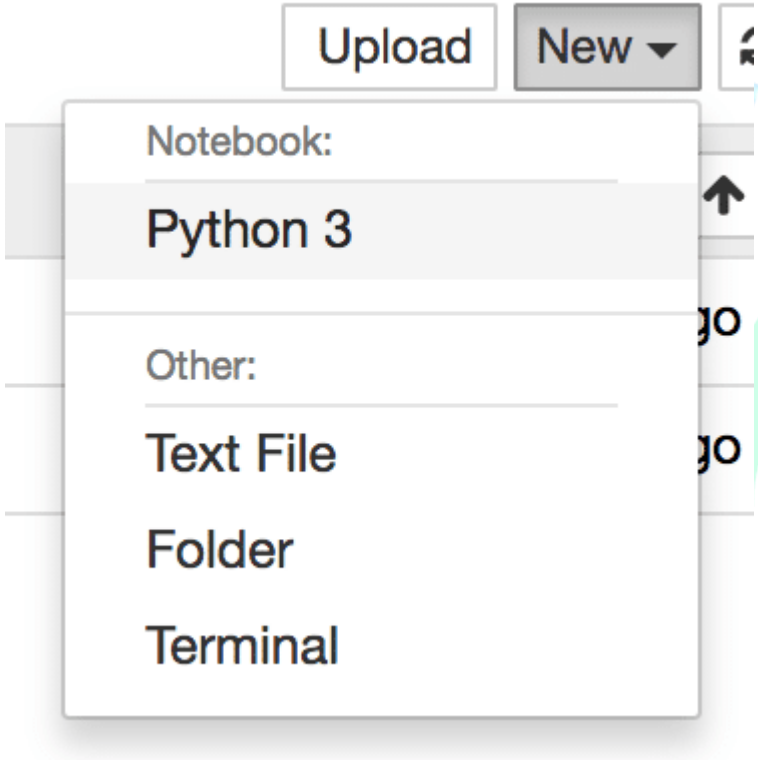
- Files
- Running

- Clusters

The default view is the *Files* tab from where you can open or create notebooks.

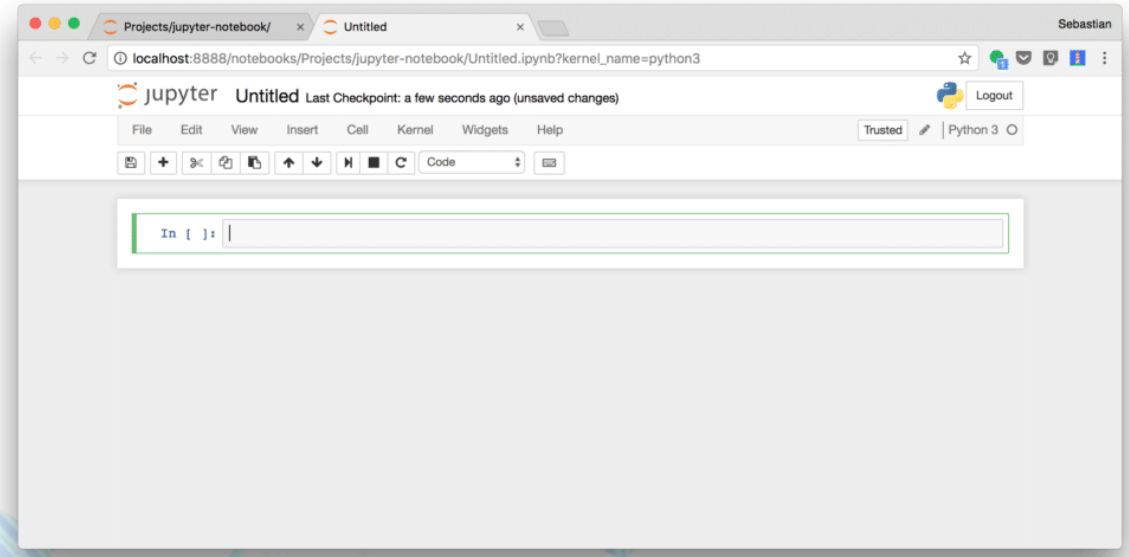
## Creating A New Notebook

Creating a new Jupyter Notebook is easy. Just use the *New* dropdown menu and you'll see the following options:

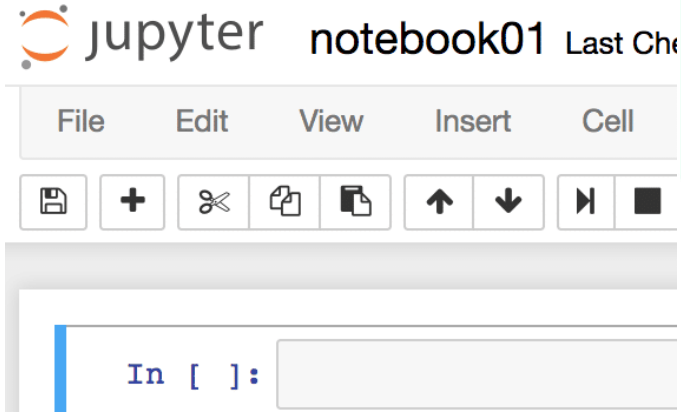





Select option *Python 3* to open a new Jupyter Notebook for Python. The notebook is created and you should be able to see something similar to:




The notebook is created but still untitled. By clicking into the text “Untitled” on the top, you can give it a name. By giving it a name the notebook will also be saved as a file of the same name with extension *.ipynb*. E.g. name the notebook *notebook01*:



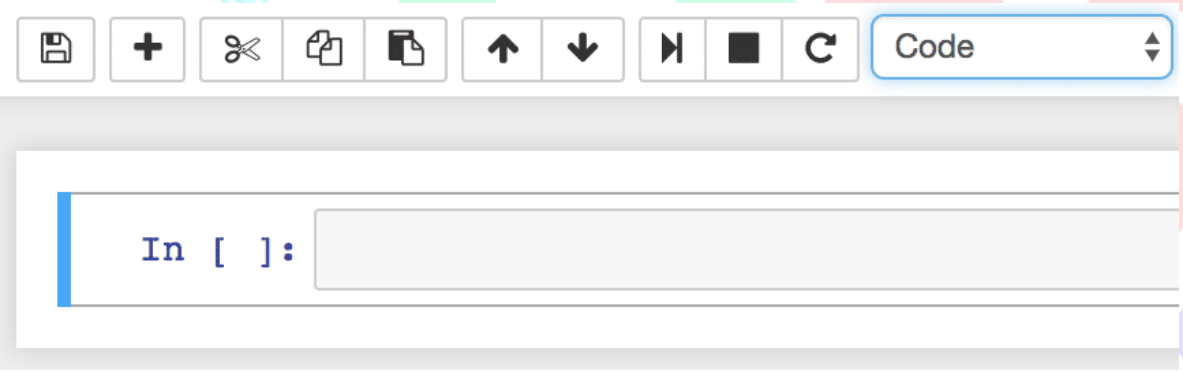

Switching back to the Files tab you’ll be able to see a new file *notebook01.ipynb*:


Because this notebook file is opened right now the file is marked with status *Running*. From here you can decided to shutdown this notebook by clicking on button *Shutdown*.

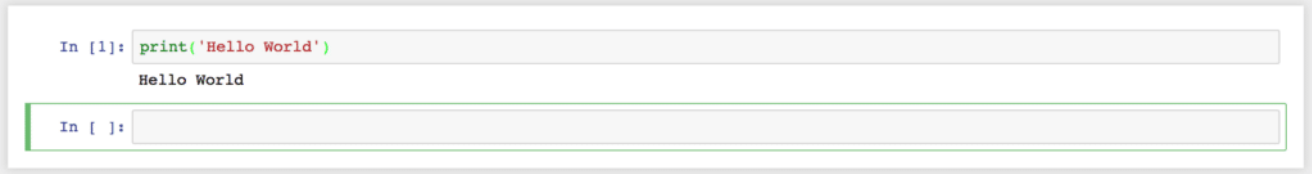
However before shutting down the notebook let's switch back to the notebook view and try out a few things to get familiar with the notebook concept.

## Working with the Notebook

The notebook itself consists of cells. A first empty cell is already available after having created the new notebook:

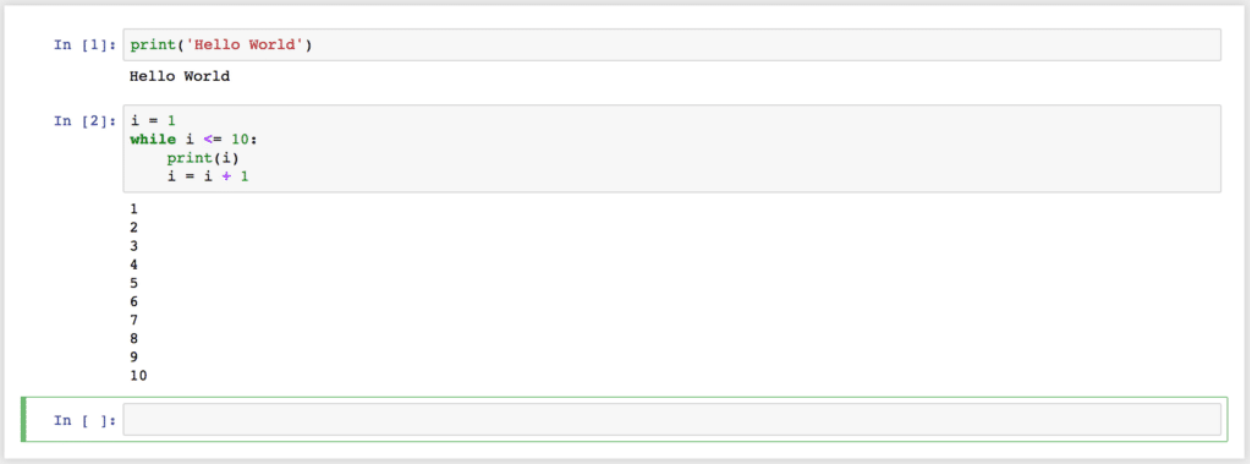



This cell is of type "Code" and you can start typing in Python code directly. Executing code in this cell can be done by either clicking on the *run cell* button or hitting Shift + Return keys:

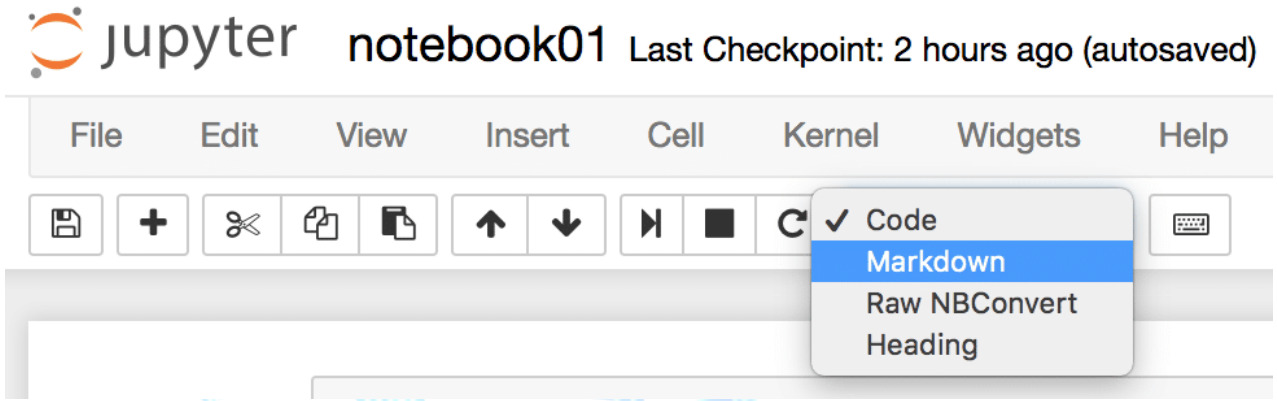



The resulting output becomes visible right underneath the cell.

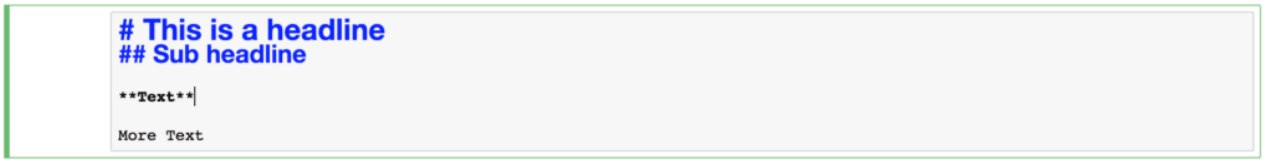
The next empty code cell is created automatically and you can continue to add further code to that cell. Just another example:



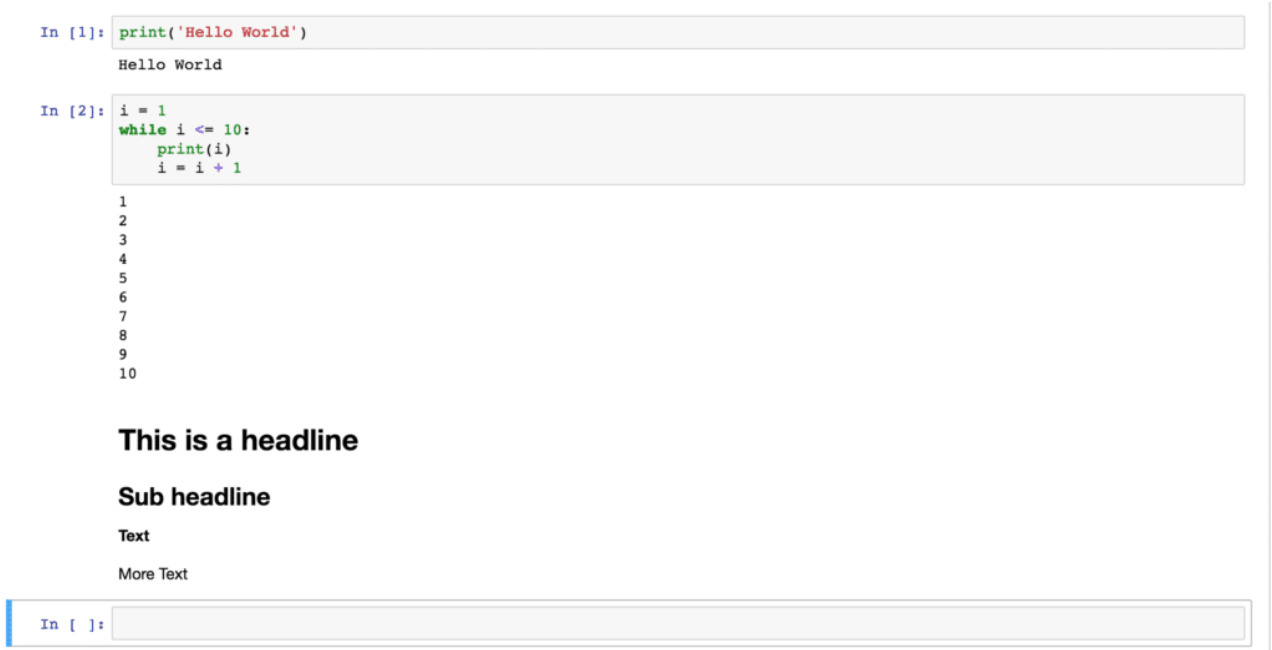

You can change the cell type from *Code* to *Markdown* to include explanatory text in your notebook. To change the type you can use the dropdown input control:




Once switched the type to *Markdown* you can start typing in markdown code:




After having entered the markdown code you can compile the cell by hitting Shift + Return once again. The markdown editor cell is then replaced with the output:




If you want to change the markdown code again you can simply click into the compiled result and the editor mode opens again.

# Edit And Command Mode

If a cell is active, two modes distinguished:

- edit mode
- command mode

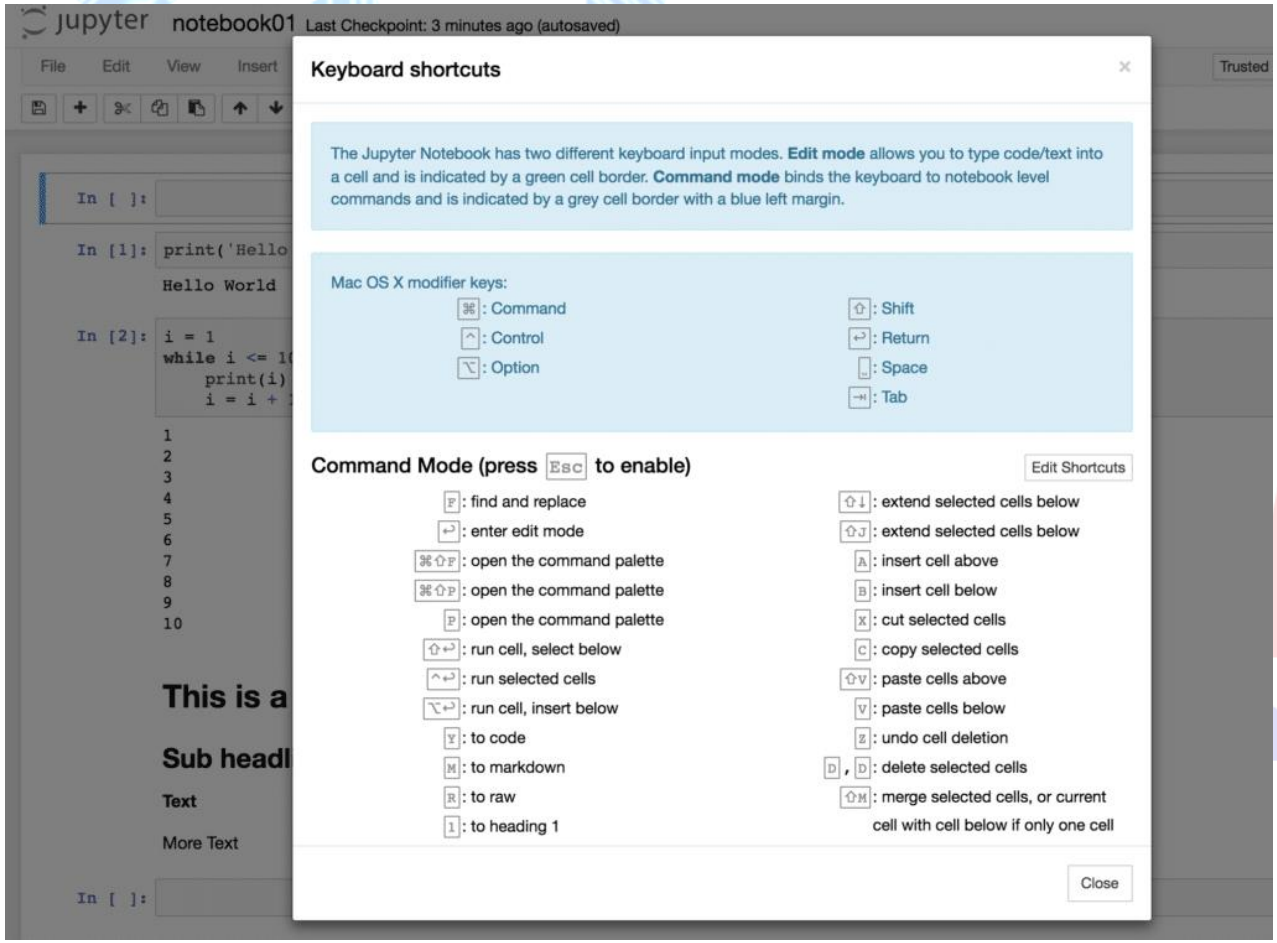
If you just click in one cell the cell is opened in command mode which is indicated by a blue border on the left:


The edit mode is entered if you click into the code area of that cell. This mode is indicated by a green border on the left side of the cell:

```
In [1]: print('Hello World')
        Hello World
```

If you'd like to leave edit mode and return to command mode again you just need to hit ESC.

To get an overview of functions which are available in command and in edit mode you can open up the overview of key shortcuts by using menu entry *Help → Keyboard Shortcuts*:



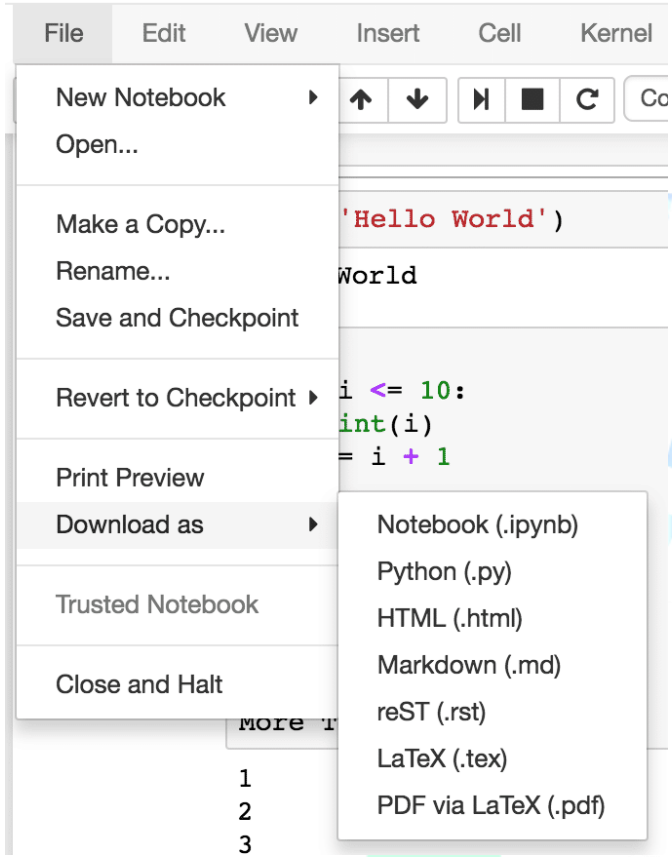
## Checkpoints

Another cool function of Jupyter Notebook is the ability to create checkpoint. By creating a checkpoint you're storing the current state of the notebook so that you can later on go back to this checkpoint and revert changes which have been made to the notebook in the meantime.

To create a new checkpoint for your notebook select menu item *Save and Checkpoint* from the *File* menu. The checkpoint is created and the notebook file is saved. If you want to go back to that checkpoint at a later point in time you need to select the corresponding checkpoint entry from menu *File → Revert to Checkpoint*.

## Exporting The Notebook

Jupyter Notebook gives you several options to export your notebook. Those options can be found in menu *File → Download as*:




**Maths**: To understand the machine learning algorithms and to **choose the right model**, one needs to understand the maths behind. But, you don't need all the maths but only some sub-branches:

- Linear algebra
- Probability theory
- Optimization

- Calculus
- Information theory and decision theory

**Programming:** Programming is needed for the following tasks.

1. Using ML models
2. Build new one
3. Get the data from various sources
4. Clean the data
5. Choose the right features and to validate

Some programming languages are preferred for doing ML than others because they have large number of libraries with most of the ML models already implemented.

- Python
- R (good but slow run time)
- MATLAB (good but costly and slow)
- JULIA (Future best! very fast, good, limited libraries as it is new)

As mentioned, we will be using Python throughout this course. Some good books in ML are the following:

Books:

- Elements of Statistical Learning
- Pattern Recognition and Machine Learning

Courses:

- Machine Learning in Coursera
- Deep Learning in Coursera

Practice:

- Kaggle
- Analytics Vidhya
- Driven Data

If you read and implement lot of good papers (say 100) you will become an expert in ML/ DL. After this point you can create your own algorithms and start publishing your work.

## Popular Python Data Analytics Libraries

| Library | Usage |
|---|---|
| numpy, scipy | Scientific & technical computing |
| pandas | Data manipulation & aggregation |
| mlpy, scikit-learn | Machine learning |
| theano, tensorflow, keras | Deep learning |
| statsmodels | Statistical analysis |
| nltk, gensim | Text processing |
| networkx | Network analysis & visualization |
| bokeh, matplotlib, seaborn, plotly | Visualization |
| beautifulsoup, scrapy | Web scraping |

### Datasets for Machine Learning

Another important consideration while getting into Machine Learning and Deep Learning is the dataset that you are going to use. Below is a list of free datasets for data science and machine learning, organized by their use cases.

### Datasets for Exploratory Analysis

- Game of Thrones
- World University Rankings
- IMDB 5000 Movie Dataset
- Kaggle Datasets

### Datasets for General Machine Learning

- Wine Quality (Regression)
- Credit Card Default (Classification)
- US Census Data (Clustering)
- UCI Machine Learning Repository

Datasets for Deep Learning

- MNIST
- CIFAR
- ImageNet
- YouTube 8M
- Deeplearning.net
- DeepLearning4J.org
- Analytics Vidhya

Datasets for Natural Language Processing

- Enron Dataset
- Amazon Reviews
- Newsgroup Classification
- NLP-datasets (Git-Hub)
- Quora Answer

Datasets for Cloud Machine Learning

- AWS Public Datasets
- Google Cloud Public Datasets
- Microsoft Azure Public Datasets

Datasets for Time Series Analysis

- EOD Stock Prices
- Zillow Real Estate Research
- Global Education Statistics

Datasets for Recommender Systems

- MovieLens
- Jester
- Million Song Dataset

## Some more Machine Learning Resources

From the next lesson, we will be studying algorithms specifically such as for regression, classification, clustering etc. In this lesson, we provided you resources to all the common machine-learning algorithms as a whole to explore and also links to a few highly rated and renowned courses from top professionals in Machine Learning as well as Deep Learning. All of the resources are available free online.

### Machine Learning Theory

- Machine Learning, Stanford University
- Machine Learning, Carnegie Mellon University
- Machine Learning, MIT
- Machine Learning, California Institute of Technology
- Machine Learning, Oxford University
- Machine Learning, Data School

### General Machine Learning with Python and Scikit-learn

*Machine Learning with scikit-learn, Data School*

- Machine Learning with scikit-learn
- Comparing Supervised Learning Algorithm
- Machine Learning with text

*Machine Learning with scikit-learn, Jake Vanderplas*

- Introduction to scikit-learn
- Basic Principles
- Linear Regression
- Support Vector Machines
- Regression Forests
- Dimensionality Reduction
- k-means Clustering
- Density Estimation
- Validation and Model Selection

Decision Trees, The Grimm Scientist

Machine Learning with scikit-learn, Andreas Mueller

- Introduction to Machine Learning with Python
- Scikit-learn tutorial
- Advanced scikit-learn

References

- **Datasets for Data Science and Machine Learning**
- **Top 15 Python Libraries for Data Science in 2017**

# AI Invasion Curriculum - Day 3

## Regression Analysis

This is the first lesson in which we will be specifically focusing on Machine Learning Algorithms and its practical implementation in Python using real world data sets. We will start with Regression Analysis and Predictive Modelling.

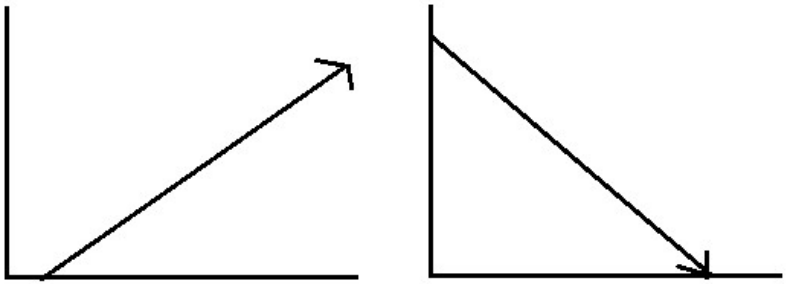
We assume that you have basic knowledge of python programming and have used the basic libraries such as Numpy, SciPy, Pandas, Matplotlib and Sklearn also mentioned in the last lesson's notes. If you haven't, here are the links for you.

1. **Numpy**
2. **SciPy**
3. **Pandas**
4. **Matplotlib**
5. **Sklearn**

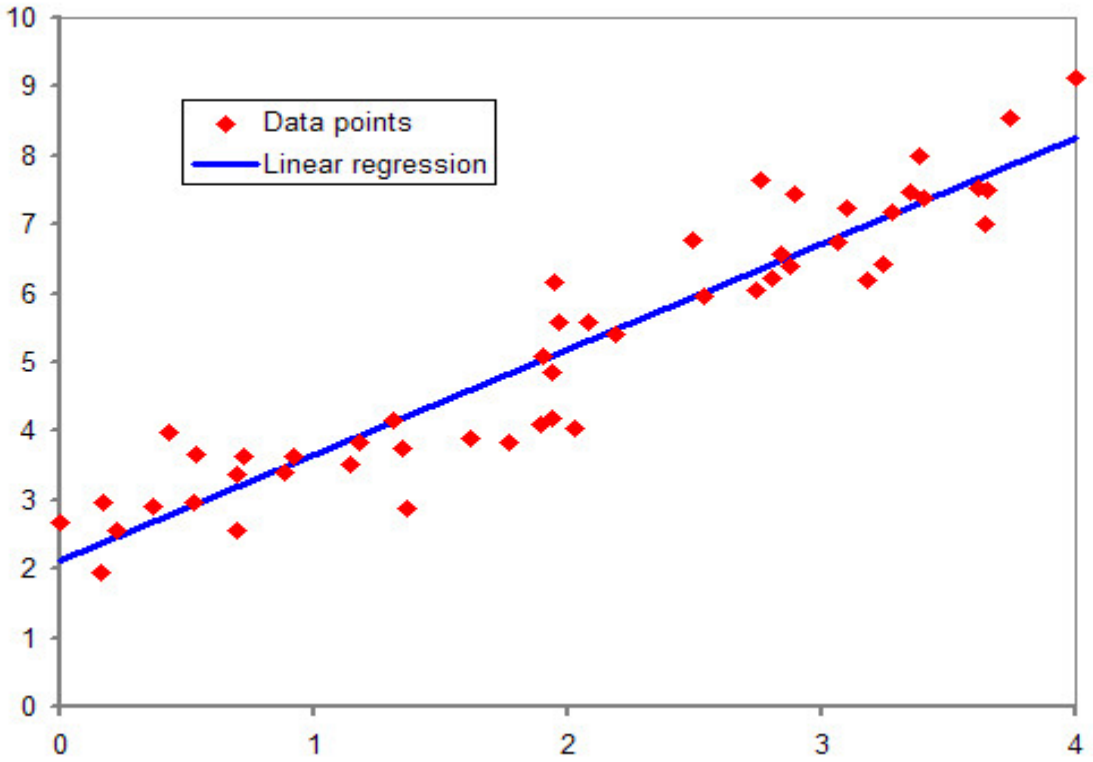
Firstly, we will have a look at a quick introduction to building models in Python, and what better way to start than one of the very basic models, linear regression? Linear Regression will be the first algorithm about regression analysis and we plan to make you learn more complex regression models.

## Concept of Linear Regression

Linear regression is a statistical model that examines the linear relationship between two (Simple Linear Regression) or more (Multiple Linear Regression) variables - a dependent variable and independent variable(s). Linear relationship basically means that when one (or more) independent variables increases (or decreases), the dependent variable increases (or decreases) too.




As you can see, a linear relationship can be positive (independent variable goes up, dependent variable goes up) or negative (independent variable goes up, dependent variable goes down) or other.




## A Little Bit About the Math

A relationship between variables Y and X is represented by this equation:

$$Y = mX + b$$

In this equation,

*Y* is the dependent variable — or the variable we are trying to predict or estimate;

X is the independent variable — the variable we are using to make predictions;

m is the slope of the regression line — it represents the effect *X* has on *Y*.

Simple Linear Regression(SLR) models also include the errors in the data (also known as residuals). We won't go too much into it now, but residuals are basically the differences between the true value of Y and the predicted/estimated value of Y. It is important to note that in a linear regression, we are trying to predict a continuous variable. In a regression model, we are trying to minimize these errors by finding the "line of best fit" — the regression line from the errors would be minimal. We are trying **to minimize the distance of the red dots from the blue line** — as close to zero as possible. It is related to (or equivalent to) minimizing the mean squared error (MSE) or the sum of squares of error (SSE), also called the "residual sum of squares." (RSS).

In most cases, we will have more than one independent variable — we'll have multiple variables; it can be as little as two independent variables and up to hundreds (or theoretically even thousands) of variables. in those cases, we will use a Multiple Linear Regression model (MLR). The regression equation is pretty much the same as the simple regression equation, just with more variables:

$$Y = b0 + b1X1 + b2X2$$

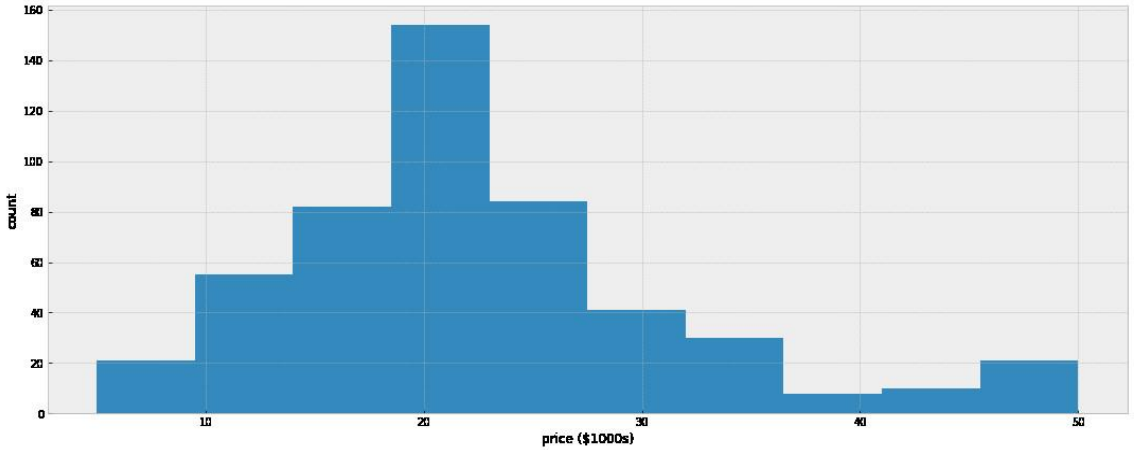
**Activity 1: Predicting Boston Housing Prices**

Here we perform a simple regression analysis on the Boston housing data, exploring simple types of linear regression model. We will use Boston Housing data set, the data set contains information about the housing values in suburbs of Boston. This dataset was originally taken from the StatLib library which is maintained at Carnegie Mellon University and is now available on the UCI Machine Learning Repository. UCI machine learning repository contains many interesting data sets, I encourage you to go through it. We will be using Sklearn to import the Boston data as it contains a bunch of useful datasets to practise along and we will also import Linear Regression Model from Sklearn. Although, you can also code your Linear Regression model as a function or class in python and it's easy.

```
from sklearn.datasets import load_boston
data = load_boston()
```

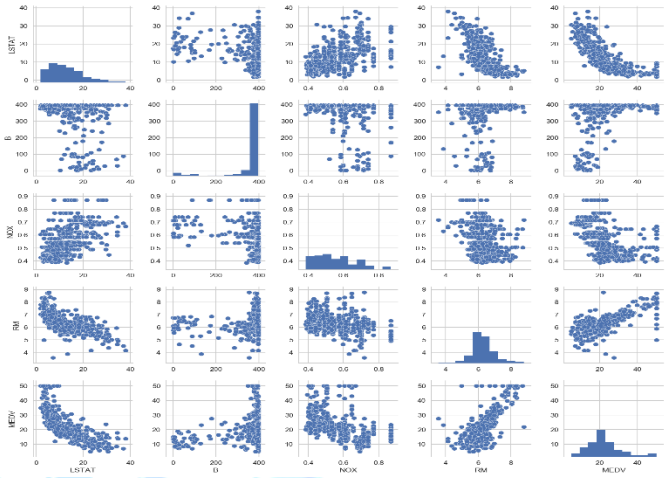
Print a histogram of the quantity to predict: price

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
plt.figure(figsize=(15, 6))
plt.hist(data.target)
plt.xlabel('price ($1000s)')
plt.ylabel('count')
```




Print the join histogram for each feature

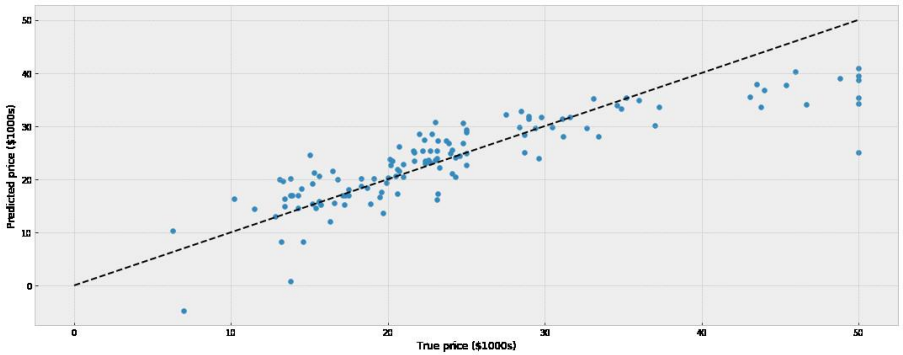
```
for index, feature_name in enumerate(data.feature_names):
    plt.figure(figsize=(4, 3))
    plt.scatter(data.data[:, index], data.target)
    plt.ylabel('Price', size=15)
    plt.xlabel(feature_name, size=15)
    plt.tight_layout()
```




Prediction

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target)
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(X_train, y_train)
predicted = clf.predict(X_test)
expected = y_test

plt.figure(figsize=(15, 6))
plt.scatter(expected, predicted)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.xlabel('True price ($1000s)')
plt.ylabel('Predicted price ($1000s)')
plt.tight_layout()
```




As we can see from the above results, the linear regression model is able to predict the values in a good manner, fitting the linear line in best manner.

Next, we will also understand the **various assumptions in Regression model**. Moreover, we will also discuss what if these assumptions get violated, how do you build your linear regression model then.

Regression is not just fitting a line to the predicted values or defining it as an equation (y = m*x + b) like we just did, there is so much into it. Regression is considered to be the simplest algorithm in Machine Learning, when we start playing with ML, most of us start with regression, but it is not understood well by the beginners. It is important and these things describe how well you understand the math behind ML. Please note that Machine Learning is not just loading classes from sci-kit learn and fitting data and predicting targets, it is something more than that.

**Regression is a parametric approach** by saying it is a 'Parametric' approach, we mean it is going to make assumptions about your data for the purpose of analysis. And, because of this reason, it has some limited uses and other regression techniques like tree-based regression and deep nets are used practically. **Linear Regression surely fails to deliver good results with data sets which doesn't fulfil its assumptions.** Therefore, for a successful regression analysis, it's essential to validate these assumptions.

So, how would you check if your data Set follows all regression assumptions?

To understand all the regression assumptions, here is a link, all the assumptions are explained in the best manner with how to overcome techniques as well.

More Links for Learning

Simple and Multiple Linear Regression in Python

Regression analysis using Python

7 Types of Regression Techniques you should know

Step-by-step guide to execute Linear Regression in Python

Linear Regression (Python Implementation)

Linear Regression in Python: A Tutorial

How to run Linear regression in Python scikit-Learn

How to Implement Simple Linear Regression from Scratch with Python

8 ways to perform simple linear regression and measure their speed using Python

**Exercises**

As for the practice for this lesson, you have to test your hands on the below mentioned Kaggle Competition.

Sberbank Russian Housing Market

In the last topic, we discussed about what Regression is and we learnt about the assumptions or so-called limitations of linear regression. Linear Regression is assumed to be the simplest machine-learning algorithm the world has ever seen, and yes! it is. Anyways, we also discussed how your model can give you poor predictions in real time if you don't obey the assumptions of linear regression. Whatever you are going to predict, whether it is stock value, sales or some revenue, linear regression must be handled with care if you want to get best predictions from it. Linear regression says, the data should be linear in nature, there must be a linear relationship. But, wait! the real-world data is always non-linear. So, what we should we do, should we try to bring non-linearity into the regression model, or check out the residuals and fitted values, keep applying transformations and working harder and harder to get the best predictive model using linear regression. This would be a pain and it is going to a take a good amount of time.

Now, the question is, should it be considered as the solution or is there any other way to deal with this, so that we can get a better predictive model without getting into these assumptions of linear regression.

Yes! There is a solution, in fact a bunch of solutions.

***There are many different analytic procedures for fitting regressive models of nonlinear nature (e.g., Generalized Linear/Nonlinear Models (GLZ), Generalized Additive Models (GAM), etc.), or some other better models called tree based regressive models.***

Most of us know about Random Forest and Decision Trees, it is very common, in case of classification or regression, they often perform far better than other models. In this lesson, we will be specifically learning tree-based models such as Decision Trees and ensemble tree-based models like Random forests (RF), Gradient Boosted Tree (GBT), Ada Boost Tree, Extreme Boosted tree for regression analysis. Tree-based model have proven themselves to be both reliable and effective and are now part of any modern predictive modeler’s toolkit.

But, there are some cases when a linear regression is assumed better than a tree-based models like the following cases.

1. *When the underlying function is truly linear*
2. *When there are a very large number of features, especially with very low signal to noise ratio. Tree based model have a little trouble modelling linear combinations of a large number of features.*

The point is, there are probably only a few cases in which linear models like SLR is better than tree-based models or other non-linear models as these fits the data better from the get-go without transformations.

Tree-Based models are more forgiving in almost every way. You don't need to scale your data, you don't need to do any monotonic transformations (log square root etc). You often don't even need to remove outliers. You can throw in features, and it'll automatically partition the data if it aids the fit. You don't have to spend any time generating interaction terms as in case of linear models and perhaps most importantly: in most cases, it will probably be notably more accurate.

***The bottom line is you can spend 3 hours playing with the data, generating features and interaction variables and get a 77% r-squared; and I can "from sklearn.ensemble import RandomForestRegressor" and in 3 minutes get an 82% r-squared.***

There is not a hype of these tree model, it's the ground, check out this link. Let me explain it to you using some examples for clear intuition with an example. Linear regression is a linear model, which means it works really nicely when the data has a linear shape. But, when the data has a non-linear shape, then a linear model cannot capture the non-linear features. So, in this case, you can use the decision trees, which will do a better job at capturing the non-linearity in the data by dividing the space into smaller sub-spaces depending on the questions asked.

Now, the question is when do you use linear regression vs Decision Trees? I guess the Quora answer here would do a better job than me, at explaining the difference between them and their applications. Let me quote that for you. Suppose you are trying to predict the income of certain individuals, the predictor variables that are available are education, age, and city. Now in a linear regression model, you have an equation with these three attributes. Fine. You'd expect higher degrees of education, higher "age" and larger cities to be

associated with higher income. But what about an individual who has a PhD and is 40 years old and living in Scranton Pennsylvania? Is he likely to earn more than a BS holder who is 35 and living in Upper West Side NYC? Maybe not. Maybe education totally loses its predictive power in a city like Scranton? Maybe age is a very ineffective, weak variable in a city like NYC?

This is where decision trees are handy. The tree be can split by city and you get to use a different set of variables for each city. Maybe Age will be a strong second-level split variable in Scranton, but it might not feature at all in the NYC branch of the tree as education may be a stronger variable in NYC. Decision Trees, be it Random Forest or Gradient Boosting Model (GBM), handle messier data and messier relationships better than regression models and there is seldom a dataset in the real world where relationships are not messy. No wonder you will seldom see a linear regression model outperforming RF or GBM. So, this is the main idea behind tree (Decision Tree Regression) and ensemble-based models (Random Forest Regression/Gradient Boosting Regression/ Extreme Boosting Regression).

In the GitHub link, you might have seen a number of other models too and their comparison as well, which are all the different available regressive models present in the Sklearn. As you can see GBM/RF are performing the best.

Below are the links to almost all the regression techniques in scikit-learn.

1. **Ordinary least squares Linear Regression**
2. **Linear least squares with l2 regularization**
3. **Linear Model trained with L1 prior as regularised (aka the Lasso)**
4. **Support Vector Regression**
5. **Decision Tree Regressor**
6. **Random Forest Regressor**
7. **Linear model fitted by minimizing a regularized empirical loss with SGD**
8. **Gradient Boosting for regression**

There is a lot more you need to explore in regression analysis, here are links for the same.

1. Bias and Variance Trade off in Regression models
2. Under fitting and over fitting in regression models
3. How can we optimize our model to avoid under fitting and over fitting
4. Regularization techniques
5. L1 and L2 – Ridge and Lasso Regression

**Exercises**

As for the practice for this lesson, you have to test your hands on the below mentioned Kaggle Competition.

Sberbank Russian Housing Market

House Prices: Advanced Regression Techniques

# AI Invasion Curriculum - Day 5

---

In the last topic, we discussed about Regression modelling and got into the different types of regression models, today, we will compare the regression analysis with the classification analysis and learn a few classification algorithms with their implementation as well.

***Regression and classification are both related to prediction, where regression predicts a value from a continuous set, whereas classification predicts the 'belonging' to the class.***

Let me explain this with the help of an example, In the last posts, we saw how we can use regression to predict the price of a house depending on the 'size' (sq. feet or whatever unit) and say 'location' of the house, can be some 'numerical value' (the house price) - This relates to regression.

On the other hand, classification works in the following manner.

Classification can be built on top of regression:

```
if score > 0.5
   class = 'a'
else
   class = 'b'
end
```

Though usually the difference is in what's called the loss function. In regression and classification, the goal of the optimization algorithm (linear, support vector, decision trees, etc.) is to optimize the output of the loss function.

To understand it much better way, here are a few links to dive deep into.

## Difference Between Classification and Regression in Machine Learning

## Classification and Regression

## What is the main difference between classification and regression problems?

Similarly, if instead of the prediction of price, we try to predict the classes such as the price can be classified in labels such as, 'very costly', 'costly', 'affordable', 'cheap', and 'very cheap' - This relates to classification. Each class may correspond to some range of values as explained here.

***Classification*** and ***Regression*** come under the same umbrella of Supervised Machine Learning and share the common concept of using past data to make predictions, or take decisions, that's where their similarity ends. Let me explain with an example.

***Have you ever thought how your mail engine is able to separate out something as spam, or ham (not spam)?***

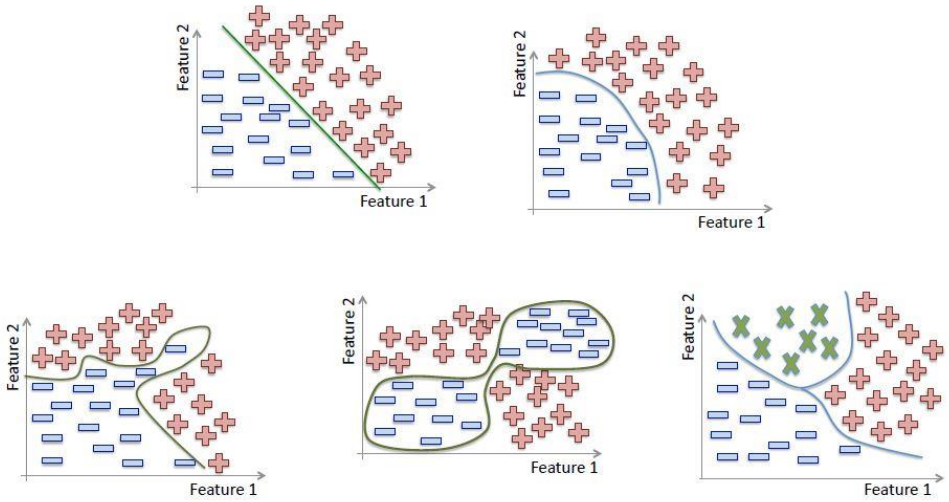
The process behind it, is to teach a model to identify any incoming mail, by training it with millions of emails that have already been determined as spam or ham. To classify mail as spam, the following things are taken into consideration:

1. Whether the mail contains spam related terms like 'lottery', 'free' etc.
2. Whether a similar mail has been classified as spam by the user.
3. How often they are received

**CLASSIFICATION**: Now with these emails, the model is trained to identify new emails.

So here, after the system has been trained to identify emails, when new emails strike your inbox, it'll automatically be classified as spam or not spam. Classification problems, requires items to be divided into different categories, based on past data. In a way, we're solving a yes/no problem. Classification can be multi label as well as discussed above in case of category of house prices.

## Classification:




**REGRESSION**: Now with regression problem, the system attempts to predict a value for an input based on past data. Unlike classification, we are predicting a value based on past data, rather than classifying them into different categories.

Say you wanted to predict whether it would rain, and if it does, how much rain you would get, maybe in centimetres based on the atmospheric variables such as humidity, temperature pressure wind-speed, wind-direction.

Let's see some more common examples of classification, which we will be working with in our python implementation.

**Given set of input features predict whether a Breast Cancer is Benign or Malignant.**

**Given an image correctly classify as containing Cats or Dogs.**

**From a given email predict whether it's spam email or not.**

Let's also discuss the types of classification:

**(1).** **Binary classification** — when there are only two classes to predict, usually 1 or 0 values.

**(2).** **Multi-Class Classification** — When there are more than two class labels to predict we call multi-classification task. E.g. predicting 3 types of iris species, image classification problems where there are more than thousands of classes (cat, dog, fish and car).

**Algorithms for classification**

- *Decision Trees*
- *Logistic Regression*
- *Naive Bayes*
- *K Nearest Neighbors*
- *Support vector Machines*
- *Random Forests*
- *many more*

**Further Reading**

This section provides more resources on the topic if you are looking to go deeper.

- How to Build a Machine Learning Classifier in Python with Scikit-learn
- Solving Multi-Label Classification problems (Case studies included)
- Machine Learning Classification Strategy Python
- Python Machine Learning: Scikit-Learn Tutorial
- Spot-Check Classification Machine Learning Algorithms in Python with scikit-learn

## Exercises

As for the practice for today, you have to test your hands on the below mentioned Kaggle Competition.

Titanic: Machine Learning from Disaster

We have explored classification algorithms and that was a part of Supervised Machine Learning which also consists of regression analysis and predictive modelling. There is another type of Machine Learning algorithm which are known as Unsupervised Machine Learning algorithms. Today, we will explore unsupervised Machine Learning algorithms such as Clustering.

**Supervised Learning**

Machine learning can be categorized as supervised and un-supervised machine learning. Some of the well know supervised machine learning algorithms are SVM (Support Vector Machine), Linear Regression, Neural Network (NN), Naive Bayes (NB). In supervised learning, the training data is labelled, that means we already know the target variable we are going to predict while we test the model.

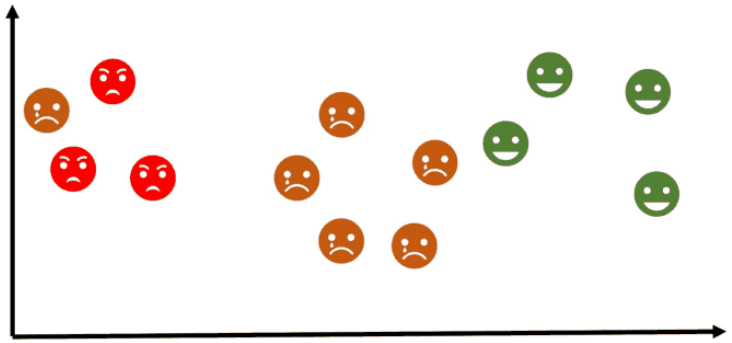
**Unsupervised Classification**

In unsupervised learning, the training data is unlabelled and the system tries to learn without a trainer. Some of the most important un-supervised algorithms are clustering, k-means, Association rule learning etc.

**What Is Clustering?**

**Cluster analysis** or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

Clustering is widely used in marketing to find naturally occurring groups of customers with similar characteristics, resulting in customer segmentation that more accurately depicts and predicts customer behaviour, leading to more personalized sales and customer service efforts.



There are a lot of clustering algorithms each serving a specific purpose and having its own use cases. To look out clustering and its definition in a deeper aspect, here are a few links that you can go through as well.

[What is Clustering in Data Mining?]

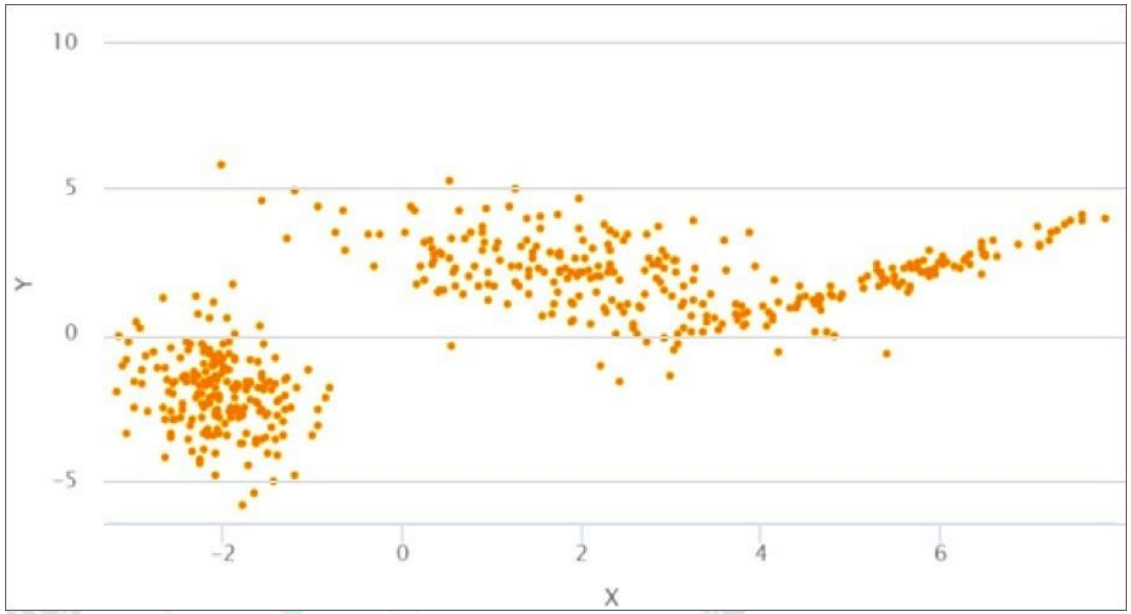
[Data Mining - Cluster Analysis]

[Clustering in Data Mining]

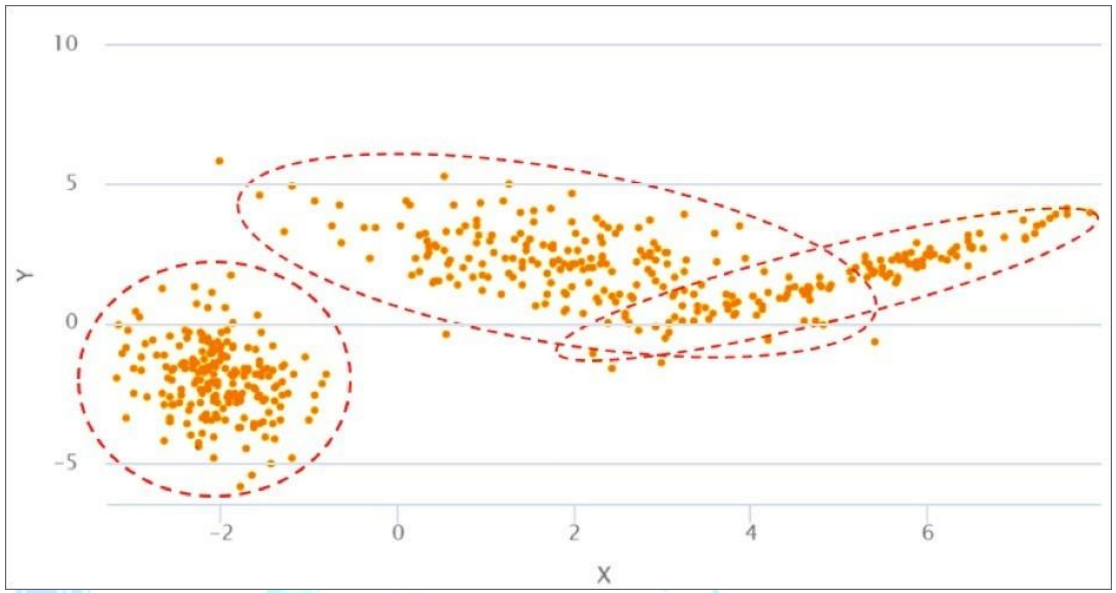
[Data Mining Concepts]

[How Businesses Can Use Clustering in Data Mining]

Numerous Clustering techniques work best for different types of data. Let's assume that your data is a numeric and continuous two-dimensional data as shown in figure below in form of a scatter plot.

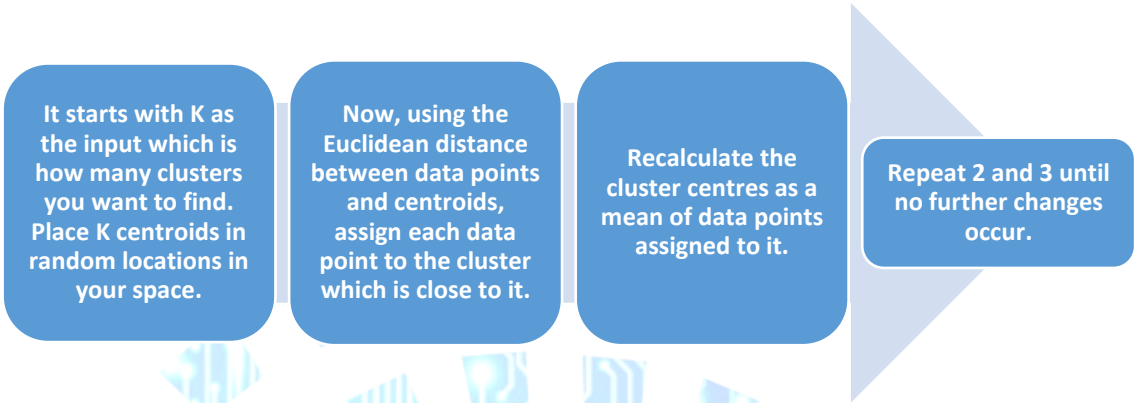



This is another scatter plot that is created from several "blobs" of different sizes and shapes which shows the clusters that exists in the data.




We will discuss a few Clustering algorithms which are Kmeans, Hierarchical Clustering.

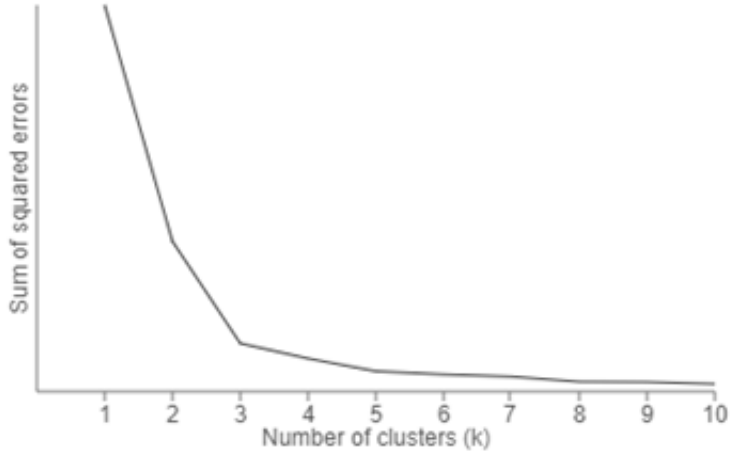
**K-means**




*You might be thinking that how do I decide the value of K in the first step.*

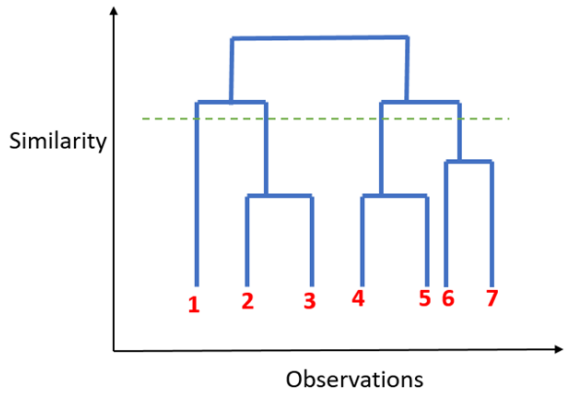
One of the methods is called Elbow method can be used to decide an optimal number of clusters. Here you would run K-mean clustering on a range of K values and plot the "*percentage of variance explained*" on the Y-axis and "*K*" on X-axis as shown in the figure below. As we add more clusters after 3 it doesn't affect the variance explained.

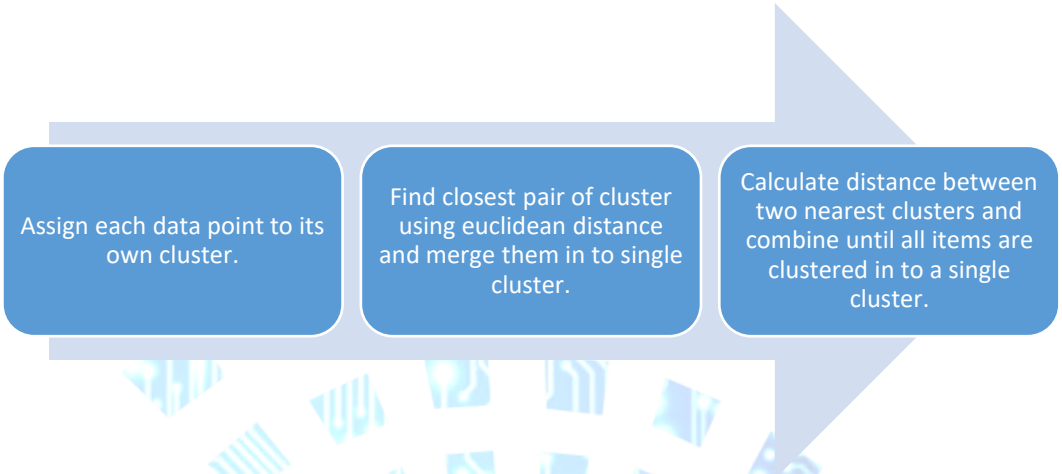
Here is another link for you to explore the same.




**Hierarchical Clustering**

Unlike K-mean clustering, *Hierarchical* clustering starts by assigning all data points as their own cluster building the hierarchy and it combines the two nearest data point and merges it together to one cluster as shown in the Dendrogram below.






**More Algorithms to Learn**

- **Mean-Shift Clustering**
- **Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM)**
- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**

**More resources for this Lesson:**

- **The 5 Clustering Algorithms Data Scientists Need to Know**
- **As for the practise today, you have to implement all the clustering algorithms available in Sklearn on these two Kaggle datasets.**
- **Breast Cancer Wisconsin (Diagnostic) Data Set**
- **World Happiness Report**

# AI Invasion Curriculum - Extras

---

Here, we will provide you the best Machine Learning and Deep Learning Resources to explore. I hope you found this course exciting and you learnt and explore a lot through the contents and resources we shared with you. Here are the awesome ML/DL resources that you can look into to enhance your knowledge.

**Free Online Books**

1. Deep Learning by Yoshua Bengio, Ian Goodfellow and Aaron Courville (May 2015)
2. Neural Networks and Deep Learning by Michael Nielsen (Dec 2014)
3. Deep Learning by Microsoft Research (2013)
4. Deep Learning Tutorial by LISA lab, University of Montreal (Jan 6 2015)
5. neuraltalk by Andrej Karpathy : numpy-based RNN/LSTM implementation
6. An introduction to genetic algorithms
7. Artificial Intelligence: A Modern Approach
8. Deep Learning in Neural Networks: An Overview

**Courses**

1. Machine Learning - Stanford by Andrew Ng in Coursera (2010-2014)
2. Machine Learning - Caltech by Yaser Abu-Mostafa (2012-2014)
3. Machine Learning - Carnegie Mellon by Tom Mitchell (Spring 2011)
4. Neural Networks for Machine Learning by Geoffrey Hinton in Coursera (2012)
5. Neural networks class by Hugo Larochelle from Université de Sherbrooke (2013)

6. Deep Learning Course by CILVR lab @ NYU (2014)
7. A.I - Berkeley by Dan Klein and Pieter Abbeel (2013)
8. A.I - MIT by Patrick Henry Winston (2010)
9. Vision and learning - computers and brains by Shimon Ullman, Tomaso Poggio, Ethan Meyers @ MIT (2013)
10. Convolutional Neural Networks for Visual Recognition - Stanford by Fei-Fei Li, Andrej Karpathy (2017)
11. Deep Learning for Natural Language Processing - Stanford
12. Neural Networks - usherbrooke
13. Machine Learning - Oxford (2014-2015)
14. Deep Learning - Nvidia (2015)
15. Graduate Summer School: Deep Learning, Feature Learning by Geoffrey Hinton, Yoshua Bengio, Yann LeCun, Andrew Ng, Nando de Freitas and several others @ IPAM, UCLA (2012)
16. Deep Learning - Udacity/Google by Vincent Vanhoucke and Arpan Chakraborty (2016)
17. Deep Learning - UWaterloo by Prof. Ali Ghodsi at University of Waterloo (2015)
18. Statistical Machine Learning - CMU by Prof. Larry Wasserman
19. Deep Learning Course by Yann LeCun (2016)
20. Designing, Visualizing and Understanding Deep Neural Networks-UC Berkeley
21. UVA Deep Learning Course MSc in Artificial Intelligence for the University of Amsterdam.
22. MIT 6.S094: Deep Learning for Self-Driving Cars
23. MIT 6.S191: Introduction to Deep Learning
24. Berkeley CS 294: Deep Reinforcement Learning
25. Keras in Motion video course

26. Practical Deep Learning For Coders by Jeremy Howard - Fast.ai
27. Introduction to Deep Learning by Prof. Bhiksha Raj (2017)

**Videos and Lectures**

1. Deep Learning, Self-Taught Learning and Unsupervised Feature Learning By Andrew Ng
2. Recent Developments in Deep Learning By Geoff Hinton
3. The Unreasonable Effectiveness of Deep Learning by Yann LeCun
4. Deep Learning of Representations by Yoshua Bengio
5. Principles of Hierarchical Temporal Memory by Jeff Hawkins
6. Machine Learning Discussion Group - Deep Learning w/ Stanford AI Lab by Adam Coates
7. Making Sense of the World with Deep Learning By Adam Coates
8. Demystifying Unsupervised Feature Learning By Adam Coates
9. Visual Perception with Deep Learning By Yann LeCun
10. The Next Generation of Neural Networks By Geoffrey Hinton at GoogleTechTalks
11. The wonderful and terrifying implications of computers that can learn By Jeremy Howard at TEDxBrussels
12. Unsupervised Deep Learning - Stanford by Andrew Ng in Stanford (2011)
13. Natural Language Processing By Chris Manning in Stanford
14. A beginners Guide to Deep Neural Networks By Natalie Hammel and Lorraine Yurshansky
15. Deep Learning: Intelligence from Big Data by Steve Jurvetson (and panel) at VLAB in Stanford.
16. Introduction to Artificial Neural Networks and Deep Learning by Leo Isikdogan at Motorola Mobility HQ
17. NIPS 2016 lecture and workshop videos - NIPS 2016
18. Deep Learning Crash Course: a series of mini-lectures by Leo Isikdogan on YouTube (2018)

**Papers**

*You can also find the most cited deep learning papers from here*

1. ImageNet Classification with Deep Convolutional Neural Networks
2. Using Very Deep Autoencoders for Content Based Image Retrieval
3. Learning Deep Architectures for AI
4. CMU's list of papers
5. Neural Networks for Named Entity Recognition zip
6. Training tricks by YB
7. Geoff Hinton's reading list (all papers)
8. Supervised Sequence Labelling with Recurrent Neural Networks
9. Statistical Language Models based on Neural Networks
10. Training Recurrent Neural Networks
11. Recursive Deep Learning for Natural Language Processing and Computer Vision
12. Bi-directional RNN
13. LSTM
14. GRU - Gated Recurrent Unit
15. GFRNN . .
16. LSTM: A Search Space Odyssey
17. A Critical Review of Recurrent Neural Networks for Sequence Learning
18. Visualizing and Understanding Recurrent Networks
19. Wojciech Zaremba, Ilya Sutskever, An Empirical Exploration of Recurrent Network Architectures

20. Recurrent Neural Network based Language Model

**Tutorials**

1. Deep Learning for NLP (without Magic)
2. A Deep Learning Tutorial: From Perceptrons to Deep Networks
3. Deep Learning from the Bottom up
4. Theano Tutorial
5. Neural Networks for Matlab
6. Using convolutional neural nets to detect facial keypoints tutorial
7. Torch7 Tutorials
8. The Best Machine Learning Tutorials On The Web
9. VGG Convolutional Neural Networks Practical
10. TensorFlow tutorials
11. More TensorFlow tutorials
12. TensorFlow Python Notebooks
13. Keras and Lasagne Deep Learning Tutorials
14. Classification on raw time series in TensorFlow with a LSTM RNN
15. Using convolutional neural nets to detect facial keypoints tutorial
16. TensorFlow-World
17. Deep Learning with Python
18. Grokking Deep Learning
19. Deep Learning for Search

20. Keras Tutorial: Content Based Image Retrieval Using a Convolutional Denoising Autoencoder
21. Pytorch Tutorial by Yunjey Choi

**Websites**

1. deeplearning.net
2. deeplearning.stanford.edu
3. nlp.stanford.edu
4. ai-junkie.com
5. cs.brown.edu/research/ai
6. eecs.umich.edu/ai
7. cs.utexas.edu/users/ai-lab
8. cs.washington.edu/research/ai
9. aiai.ed.ac.uk
10. www-aig.jpl.nasa.gov
11. csail.mit.edu
12. cgi.cse.unsw.edu.au/~aishare
13. cs.rochester.edu/research/ai
14. ai.sri.com
15. isi.edu/AI/isd.htm
16. nrl.navy.mil/itd/aic
17. hips.seas.harvard.edu

18. AI Weekly
19. stat.ucla.edu
20. deeplearning.cs.toronto.edu
21. jeffdonahue.com/lrcn/
22. visualqa.org
23. www.mpi-inf.mpg.de/departments/computer-vision...
24. Deep Learning News
25. Machine Learning is Fun! Adam Geitgey's Blog
26. Guide to Machine Learning
27. Deep Learning for Beginners

**Datasets**

1. MNIST Handwritten digits
2. Google House Numbers from street view
3. CIFAR-10 and CIFAR-100
4. IMAGENET
5. Tiny Images 80 Million tiny images6.
6. Flickr Data 100 Million Yahoo dataset
7. Berkeley Segmentation Dataset 500
8. UC Irvine Machine Learning Repository
9. Flickr 8k
10. Flickr 30k

11. Microsoft COCO
12. VQA
13. Image QA
14. AT&T Laboratories Cambridge face database
15. AVHRR Pathfinder

**Conferences**

1. CVPR - IEEE Conference on Computer Vision and Pattern Recognition
2. AAMAS - International Joint Conference on Autonomous Agents and Multiagent Systems
3. IJCAI - International Joint Conference on Artificial Intelligence
4. ICML - International Conference on Machine Learning
5. ECML - European Conference on Machine Learning
6. KDD - Knowledge Discovery and Data Mining
7. NIPS - Neural Information Processing Systems
8. O'Reilly AI Conference - O'Reilly Artificial Intelligence Conference
9. ICDM - International Conference on Data Mining
10. ICCV - International Conference on Computer Vision
11. AAAI - Association for the Advancement of Artificial Intelligence

**Frameworks**

1. Caffe
2. Torch7
3. Theano

4. cuda-convnet
5. convetjs
6. Ccv
7. NuPIC
8. DeepLearning4J
9. Brain
10. DeepLearnToolbox
11. Deepnet
12. Deeppy
13. JavaNN
14. hebel
15. Mocha.jl
16. OpenDL
17. cuDNN
18. MGL
19. Knet.jl
20. Nvidia DIGITS - a web app based on Caffe
21. Neon - Python based Deep Learning Framework
22. Keras - Theano based Deep Learning Library
23. Chainer - A flexible framework of neural networks for deep learning
24. RNNLM Toolkit

25. RNNLIB - A recurrent neural network library
26. char-rnn
27. MatConvNet: CNNs for MATLAB
28. Minerva - a fast and flexible tool for deep learning on multi-GPU
29. Brainstorm - Fast, flexible and fun neural networks.
30. Tensorflow - Open source software library for numerical computation using data flow graphs
31. DMTK - Microsoft Distributed Machine Learning Tookit
32. Scikit Flow - Simplified interface for TensorFlow (mimicking Scikit Learn)
33. MXnet - Lightweight, Portable, Flexible Distributed/Mobile Deep Learning framework
34. Veles - Samsung Distributed machine learning platform
35. Marvin - A Minimalist GPU-only N-Dimensional ConvNets Framework
36. Apache SINGA - A General Distributed Deep Learning Platform
37. DSSTNE - Amazon's library for building Deep Learning models
38. SyntaxNet - Google's syntactic parser - A TensorFlow dependency library
39. mlpack - A scalable Machine Learning library
40. Torchnet - Torch based Deep Learning Library
41. Paddle - PArallel Distributed Deep LEarning by Baidu
42. NeuPy - Theano based Python library for ANN and Deep Learning
43. Lasagne - a lightweight library to build and train neural networks in Theano
44. nolearn - wrappers and abstractions around existing neural network libraries, most notably Lasagne
45. Sonnet - a library for constructing neural networks by Google's DeepMind

46. PyTorch - Tensors and Dynamic neural networks in Python with strong GPU acceleration
47. CNTK - Microsoft Cognitive Toolkit
48. Serpent.AI - Game agent framework: Use any video game as a deep learning sandbox
49. Caffe2 - A New Lightweight, Modular, and Scalable Deep Learning Framework
50. deeplearn.js - Hardware-accelerated deep learning and linear algebra (NumPy) library for the web

# 25 MAJOR MILESTONES THAT SET DATA SCIENCE NIGERIA APART

1. 1st of its kind Knowledge Box of 10,000 AI videos distributed free to Nigerian students and AI enthusiasts
   *https://guardian.ng/technology/data-science-nigeria-democratises-artificial-intelligence-learning/*
2. 300,000+ download of our free ebook on Artificial Intelligence for Starters
   *https://goo.gl/QJio3W*
3. Publication of Annual Report as a registered non-profit
   *https://goo.gl/iQPfy4*
4. Development of locally-relevant data collection apps
   - Check4Me:*https://www.linkedin.com/feed/update/urn:li:activity:6501532932623081472*
   - MarkeTell: *goo.gl/XYhvs9*
5. 1st ever InterCampus Machine Learning competition for 95 Nigerian universities and polytechnics
   *https://techcabal.com/2018/10/19/oau-and-unilag-students-win-the-data-science-nigeria-bluechip-inter-campus-machine-learning-competition/*
6. Artificial Intelligence Hub Formal Opening and Mini Hackathon
   *https://www.youtube.com/watch?v=X6To4gRQ5VQ*
7. 2018 Artificial Intelligence Summit & all-expense paid residential Bootcamp
   *https://www.youtube.com/watch?v=7k2Qu5nvilQ*
8. Pan Nigerian AI Knowledge sharing via FREE Introduction to Machine learning classes in 30 cities
   *https://www.linkedin.com/feed/update/urn:li:activity:6506851307486216192*
9. Over 10,000 participants in our bootcamp qualification online study
   *https://nigeriacommunicationsweek.com.ng/oau-unilag-students-win-bluechip-inter-campus-machine-learning-competition/*

10. Industry-level Hackathon with Access Bank and African Fintech Foundry
    https://www.linkedin.com/feed/update/urn:li:activity:6478573643499597824
11. 50+ direct job/internship placement & strategic partnership with KPMG
    https://guardian.ng/business-services/communications/kpmg-offers-internship-to-10-data-science-nigeria-hackathon-participants/
12. Data Science Nigeria member, Olamide Oyediran wins Zindi United Nation's Sustainable Development Goals Text Classification Challenge
    https://technext.ng/2018/11/27/datascience-nigerias-winner-emerges-zindi-africas-sustainable-development-goals-text-classification-challenge/
13. Data Science Nigeria's Zainab Ishaq Musa Wins 2018 Hult Prize Regional Competition
    https://technext.ng/2018/04/06/data-science-nigeria-zainab-ishaq-musa-wins-2018-hult-prize-regional-competition-compete-1m-prize-finals/
14. Artificial Intelligence for Executives and Business Leaders - Masterclass
    https://goo.gl/EMfXFw
15. Formal validation as a US-verified charity via Equivalency Determination by NGOsource
    https://goo.gl/P3nSJH
16. Development of Curriculum for intro Machine Learning & Deep Learning https://drive.google.com/file/d/1h-RxyjEdIKnN2iAf0_rUlqrNb5ofkW2I/view
17. Publication of our 3 years Artificial Intelligence Roadmap (Vision 2016 to 2019)
    https://drive.google.com/file/d/1KKr1r0BmKyQcr62cVVafv3WTCDOJAX36/view?usp=sharing
18. Industry Meet-Up on Natural Language Processing with prof Raj Khrishnan
    https://www.youtube.com/watch?v=hwJG6PAtxvw
19. Artificial Intelligence for Kids Summer School
    https://www.youtube.com/watch?v=jk3K1pjp53c

20. Japanese envoy visit to the AI Hub
    *https://technext.ng/2018/11/05/data-science-nigeria-robotic-lab-getting-support-japan/*
21. Artificial Intelligence for Agriculture with Prof Ndubuisi Ekekwe
    *https://www.youtube.com/watch?v=-zkHFeEMoH8*
22. 2017 Summit and Bootcamp
    *https://www.youtube.com/watch?v=_z_7hB-27A4*
23. AI Masterclass with visiting experts in the diaspora
    *https://www.linkedin.com/feed/update/urn:li:activity:6505299530441846784*
24. Global mentorship programme
    *https://technext.ng/2018/05/07/data-science-nigeria-dsn-launches-mentorship-initiative-40-mentors/*
25. Weekly free 10-week Introductory Machine Learning course
    *https://www.linkedin.com/feed/update/urn:li:activity:6500005848848936960*

**Artificial Intelligence Bootcamps 2017/2018 – all expense paid and fully residential for local talent development**

**First-of-its-kind Knowledge Box- a 2 Terabyte USB of over 10,000 videos from top AI schools and conferences with a view to eliminating the constraints of expensive and low-quality internet access and electricity challenges**


**Inclusive learning through 5-day FREE Introductory Machine Learning classes in 30 Nigerian cities**

## Regular Dial-in Classes with global learning experts


## Free 10-week Face-to-Face class on Introduction Machine learning at a fully dedicated AI Hub

**Hackathon to deploy learning in solving business problems – partnership with leading companies in Nigeria**

**Contacts**

Website: www.datasciencenigeria.org

Email: info@datasciencenigeria.org

Twitter: Datasciencenig

Instagram: Datasciencenigeria

Facebook: facebook.com/datasciencenig

YouTube: https://goo.gl/Vcjjyp