Anushka Bajpai Follow
Mar 21 · 5 min read · Listen

101 DATA SCIENCE with Cheat Sheets (ML, DL, Scraping, Python, R, SQL, Maths & Statistics)

"Perhaps the best test of a man's intelligence is his capacity for making a summary" — **Lytton Strachey**

Couldn't find a better topic to begin my new 'Medium' journey with . . .

Photo by Annie Spratt on Unsplash

> *Data Science is an ever-growing field, there are numerous tools & techniques to remember. It is not*

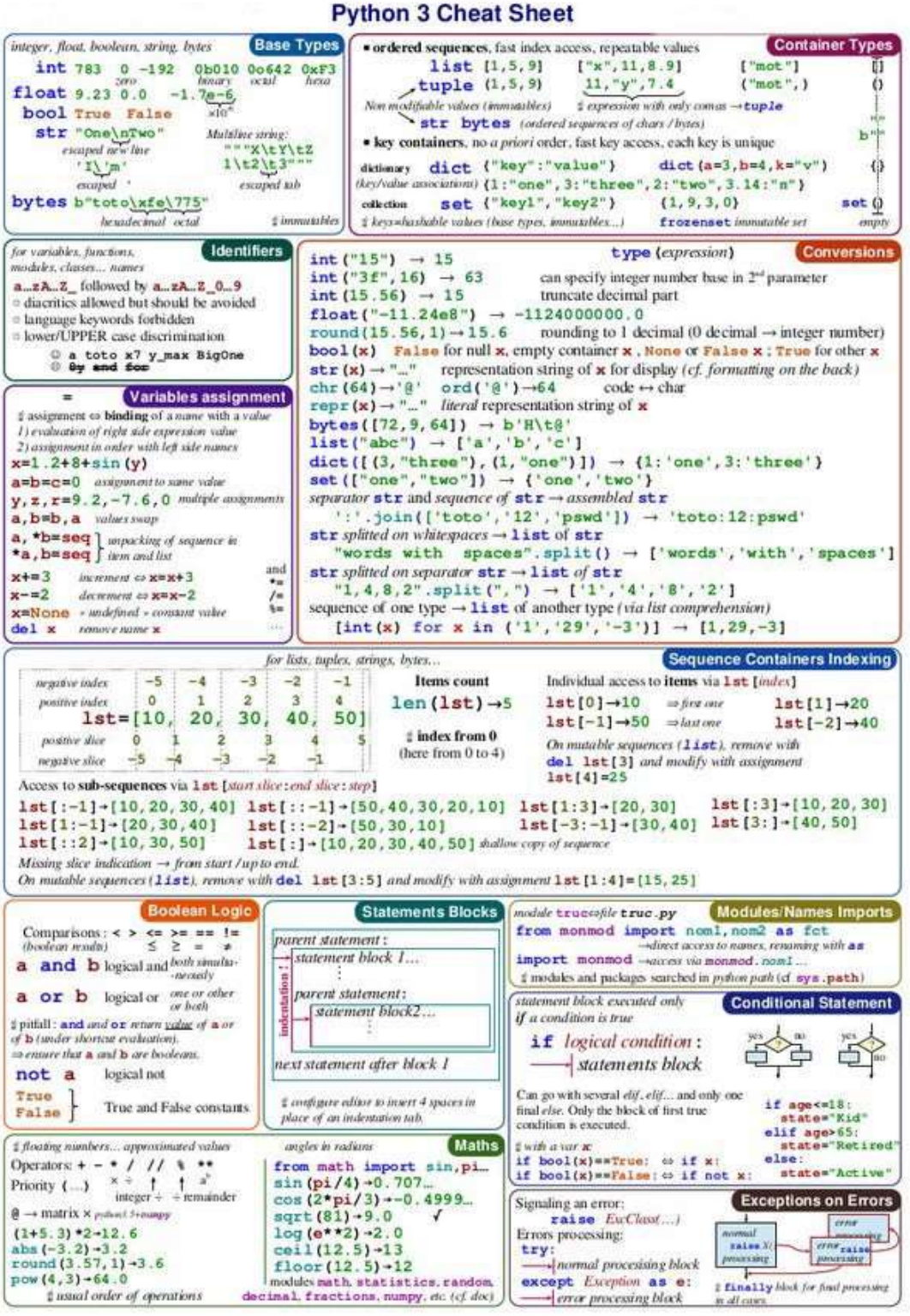
There are thousands of cheat sheets available out there. However, how many of them do you really find useful ?

It's a treasure to have good quality resources compiled at one place. Hence, I decided to go ahead and put in this compilation of Cheat Sheets for a quick yet thorough reference.

For my readers' convenience, I have segregated the best cheat sheets (comprehensive yet concise) separately for each of the below topics. . .

1. ***Python*** (*Basic python as well as ML libraries : Numpy /Pandas/Matplotlib/Seaborn/Spacy*)
2. ***R***
3. ***SQL***
4. ***Web scraping Tools*** (*Beautifulsoup, Scrapy and Selenium)*
5. ***Data Visualization Tools*** (*Tableau, Power BI*)
6. ***Maths and Statistics***
7. ***Machine Learning Algorithms*** *(Supervised and Unsupervised)*
8. ***Deep Learning and NLP*** *(Neural Networks, Keras, Tensorflow, Pytorch, NLTK, Spacy)*
9. ***IDE/OS*** *(GitHub, Linux, SkLearn, Jupyter Notebook)*

PYTHON



Source :https://www.reddit.com/

Basic Python

1. ***gto76 —*** *extensive Python cheat sheet offered by GitHub*

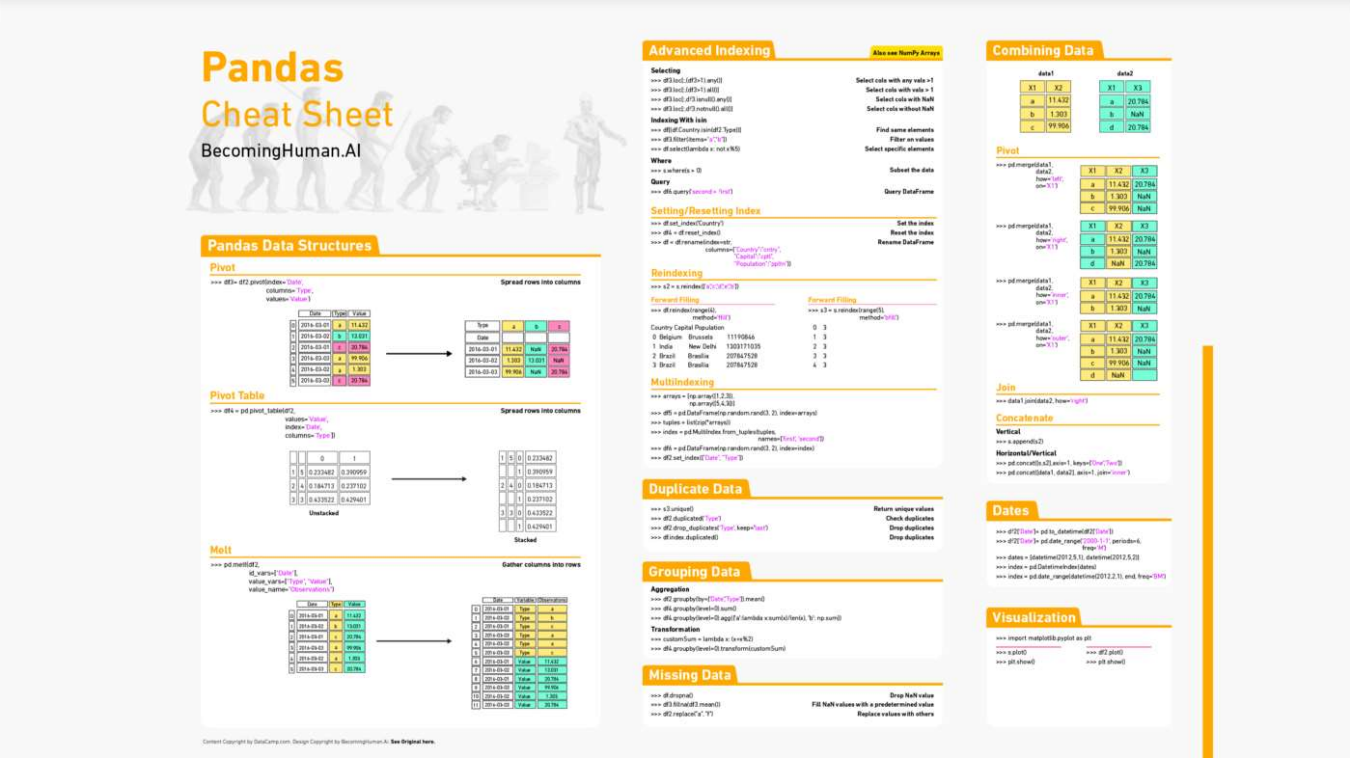
4. *Cheatography*
5. *Python crash course*
6. *DataQuest (basic and intermediate)*
7. *MementoPython3*

Numpy

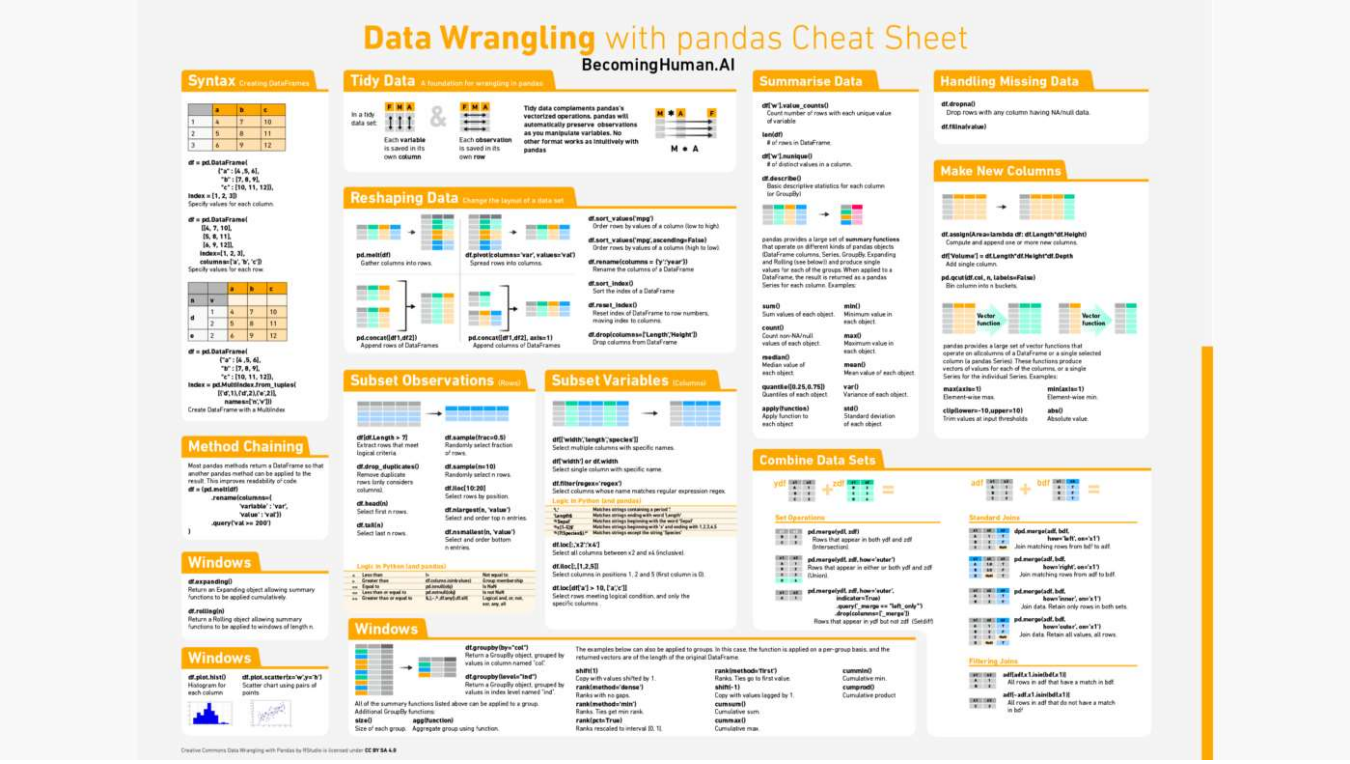
1. *DataCamp*
2. *Intellipat*
3. *A Little Bit of Everything*
4. *Data Quest*
5. *NumPy for R (and S-plus) Users*
6. *Ipgp_github*

Pandas

1. *The Most Comprehensive Cheat Sheet*
2. BecomingHuman.AI
3. *The Beginner's Cheat Sheet*
4. *DATACAMP*
5. *Data Science Central (compact)*
6. *INTELLIPAT*

Source : https://becominghuman.ai/



Source : https://becominghuman.ai/

Matplotlib

1. *Datacamp*

4. *Becominghuman.ai*
5. Travis_Blog

Seaborn

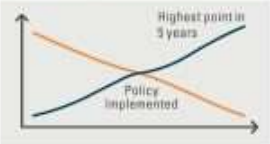
1. *Datacamp*
2. *Cheatography*
3. *MartinNormark*
4. *kaggle*

Core Principles of Data Visualization

Audience

Always consider your audience —whether they need a short, written report, a more in-depth paper, or an online exploratory data tool.

Include annotation

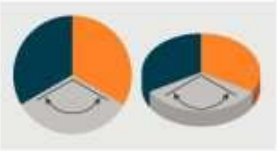


Add explanatory text to help the reader understand how to read or use the visualization (if necessary) and also to guide them through the content.

Use pie charts with care

We are not very good at discerning quantities from the slices of the pie chart. Other chart types—for example, bars, stacked bars, treemaps, or slope charts—may be a better choice.

Avoid 3D



Using 3D when you don't have a third variable will usually distort the perception of the data and should thus be avoided.

Start bar and column charts at zero

Bar and column charts that do not start at zero overemphasize the differences between the values. For small changes in quantities, consider visualizing the difference or the change in the values.

Make labels easy to read

When applicable, rotate bar and column charts to make the labels horizontal. If possible, make vertical axis labels horizontal, possibly below the title. In general, make labels clear, concise, and easy for your reader to understand.

Try small multiples

Breaking up a complicated chart into smaller chunks can be an effective way to visualize your data.

Use maps carefully

Use maps carefully, always being sure it is the geographic point you are trying to make. Column and bar charts, for example, are often better at enabling comparisons between geographic units.

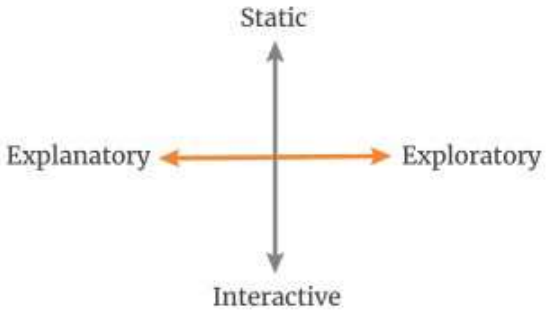
Color and font considerations

Avoid default colors and fonts—they all look the same and don't stand out.

Consider color blindness—about 10% of people (mostly men) have some form of color blindness.

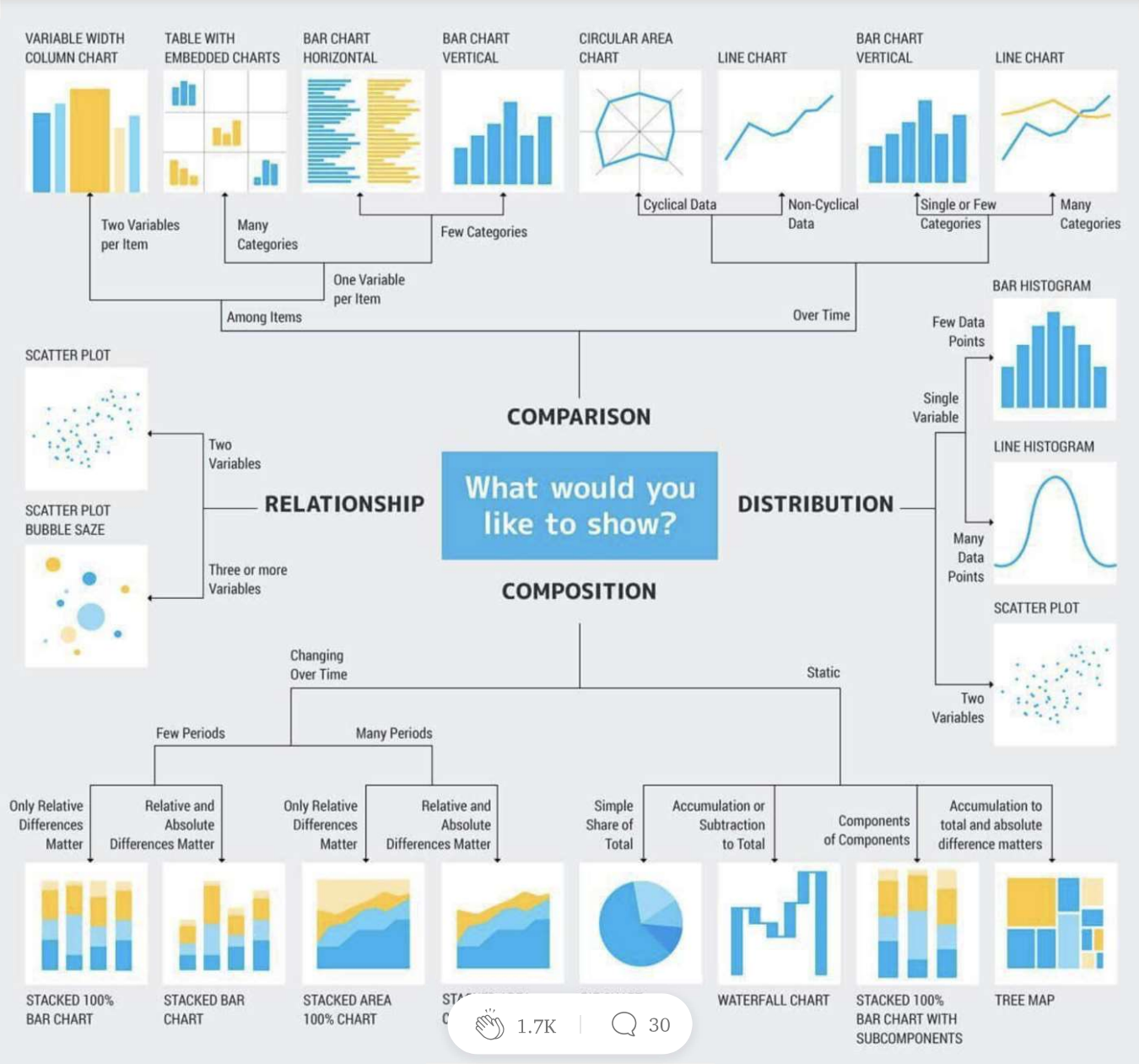
Avoid the rainbow color palette—it doesn't map to our number system and there is no logical ordering.

Visualization Mapping: Form and Function

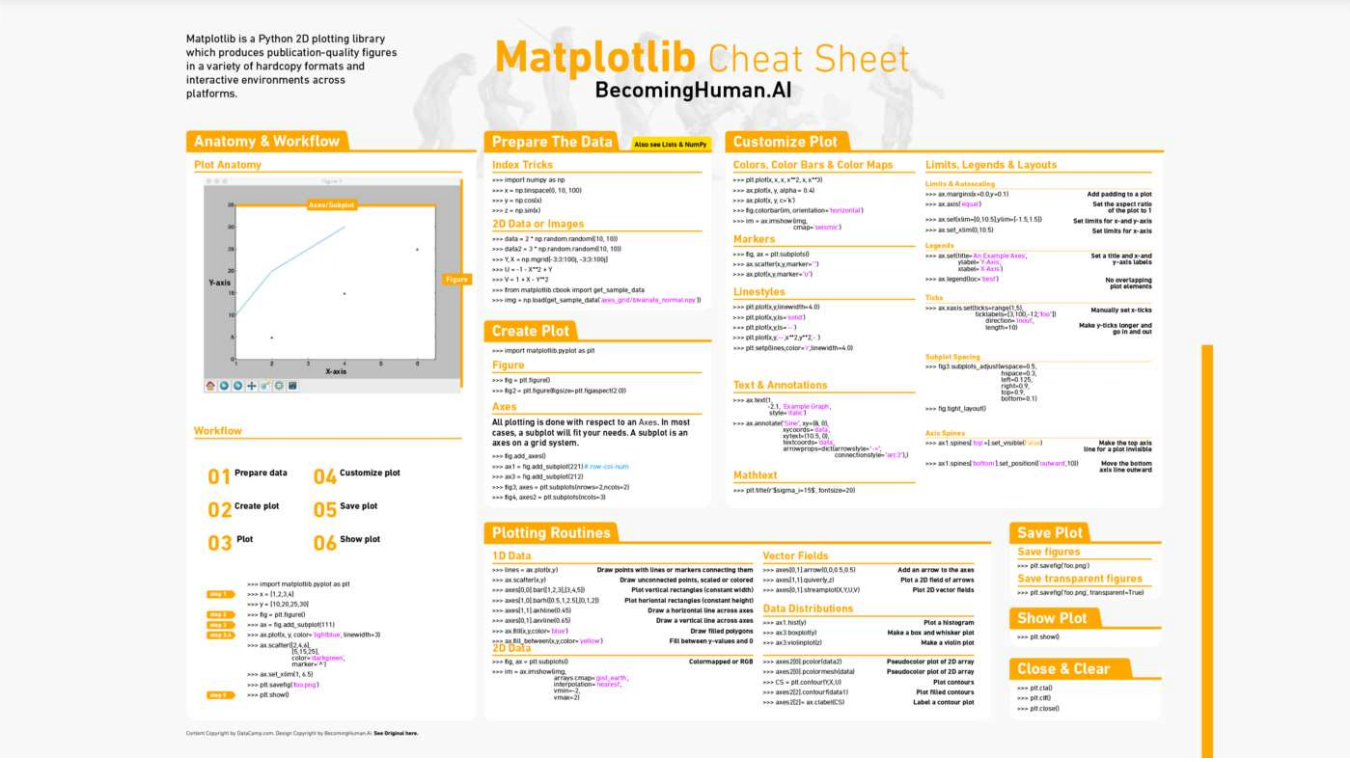


PolicyViz

Source : https://www.kdnuggets.com/2018/08/data-visualization-cheatsheet.html



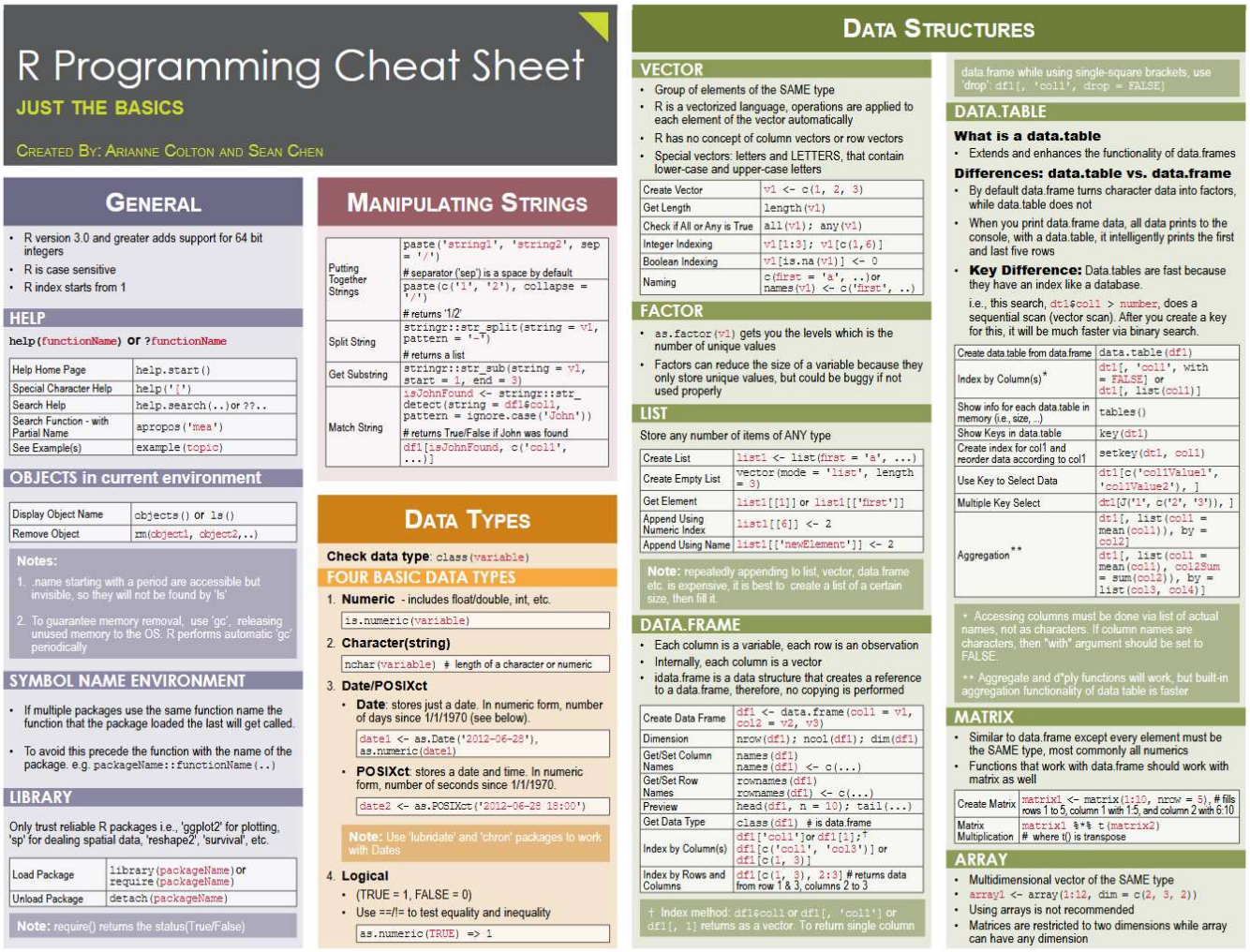
Source : https://www.kaggle.com/getting-started/160583



Source : https://becominghuman.ai/

R

1. *github*
2. *datacamp*
3. *rstudio*
4. *intellipat*



R Programming Cheat Sheet

JUST THE BASICS

CREATED BY: ARIANNE COLTON AND SEAN CHEN

GENERAL

- R version 3.0 and greater adds support for 64 bit integers
- R is case sensitive
- R index starts from 1

HELP

`help(functionName)` or `?functionName`

Help Home Page	`help.start()`
Special Character Help	`help('[')`
Search Help	`help.search(..)` or `??..`
Search Function - with Partial Name	`apropos('mea')`
See Example(s)	`example(topic)`

OBJECTS in current environment

Display Object Name	`objects()` or `ls()`
Remove Object	`rm(object1, object2,..)`

Notes:

1. .name starting with a period are accessible but invisible, so they will not be found by 'ls'
2. To guarantee memory removal, use 'gc', releasing unused memory to the OS. R performs automatic 'gc' periodically

SYMBOL NAME ENVIRONMENT

- If multiple packages use the same function name the function that the package loaded the last will get called.
- To avoid this precede the function with the name of the package. e.g. `packageName::functionName(..)`

LIBRARY

Only trust reliable R packages i.e., 'ggplot2' for plotting, 'sp' for dealing spatial data, 'reshape2', 'survival', etc.

Load Package	`library(packageName)` or `require(packageName)`
Unload Package	`detach(packageName)`

Note: require() returns the status(True/False)

MANIPULATING STRINGS

Putting Together Strings	`paste('string1', 'string2', sep = '/')`
	# separator ('sep') is a space by default
	`paste(c('1', '2'), collapse = '/')`
	# returns '1/2'

Split String	`stringr::str_split(string = v1, pattern = '-')` # returns a list
Get Substring	`stringr::str_sub(string = v1, start = 1, end = 3)`

Match String	`isJohnFound <- stringr::str_detect(string = df1$col1, pattern = ignore.case('John'))`
	# returns True/False if John was found
	`df1[isJohnFound, c('col1', ...)]`

DATA TYPES

Check data type: `class(variable)`

FOUR BASIC DATA TYPES

1. **Numeric** - includes float/double, int, etc.

```
is.numeric(variable)
```

2. **Character(string)**

```
nchar(variable)  # length of a character or numeric
```

3. **Date/POSIXct**
   - **Date**: stores just a date. In numeric form, number of days since 1/1/1970 (see below).

```
date1 <- as.Date('2012-06-28'),
as.numeric(date1)
```

   - **POSIXct**: stores a date and time. In numeric form, number of seconds since 1/1/1970.

```
date2 <- as.POSIXct('2012-06-28 18:00')
```

Note: Use 'lubridate' and 'chron' packages to work with Dates

4. **Logical**
   - (TRUE = 1, FALSE = 0)
   - Use ==/!= to test equality and inequality

```
as.numeric(TRUE) => 1
```

DATA STRUCTURES

VECTOR

- Group of elements of the SAME type
- R is a vectorized language, operations are applied to each element of the vector automatically
- R has no concept of column vectors or row vectors
- Special vectors: letters and LETTERS, that contain lower-case and upper-case letters

Create Vector	`v1 <- c(1, 2, 3)`
Get Length	`length(v1)`
Check if All or Any is True	`all(v1); any(v1)`
Integer Indexing	`v1[1:3]; v1[c(1,6)]`
Boolean Indexing	`v1[is.na(v1)] <- 0`
Naming	`c(first = 'a', ..)` or `names(v1) <- c('first', ..)`

FACTOR

- `as.factor(v1)` gets you the levels which is the number of unique values
- Factors can reduce the size of a variable because they only store unique values, but could be buggy if not used properly

LIST

Store any number of items of ANY type

Create List	`list1 <- list(first = 'a', ...)`
Create Empty List	`vector(mode = 'list', length = 3)`
Get Element	`list1[[1]]` or `list1[['first']]`
Append Using Numeric Index	`list1[[6]] <- 2`
Append Using Name	`list1[['newElement']] <- 2`

Note: repeatedly appending to list, vector, data.frame etc. is expensive, it is best to create a list of a certain size, then fill it.

DATA.FRAME

- Each column is a variable, each row is an observation
- Internally, each column is a vector
- idata.frame is a data structure that creates a reference to a data.frame, therefore, no copying is performed

Create Data Frame	`df1 <- data.frame(col1 = v1, col2 = v2, v3)`
Dimension	`nrow(df1); ncol(df1); dim(df1)`
Get/Set Column Names	`names(df1)` `names(df1) <- c(...)`
Get/Set Row Names	`rownames(df1)` `rownames(df1) <- c(...)`
Preview	`head(df1, n = 10); tail(...)`
Get Data Type	`class(df1)  # is data.frame`
Index by Column(s)	`df1['col1']` or `df1[1]`;† `df1[c('col1', 'col3')]` or `df1[c(1, 3)]`
Index by Rows and Columns	`df1[c(1, 3), 2:3]` # returns data from row 1 & 3, columns 2 to 3

† Index method: `df1$col1` or `df1[, 'col1']` or `df1[, 1]` returns as a vector. To return single column data.frame while using single-square brackets, use 'drop': `df1[, 'col1', drop = FALSE]`

DATA.TABLE

What is a data.table

- Extends and enhances the functionality of data.frames

Differences: data.table vs. data.frame

- By default data.frame turns character data into factors, while data.table does not
- When you print data.frame data, all data prints to the console, with a data.table, it intelligently prints the first and last five rows
- **Key Difference:** Data.tables are fast because they have an index like a database.

  i.e., this search, `dt1$col1 > number`, does a sequential scan (vector scan). After you create a key for this, it will be much faster via binary search.

Create data.table from data.frame	`data.table(df1)`
Index by Column(s)*	`dt1[, 'col1', with = FALSE]` or `dt1[, list(col1)]`
Show info for each data.table in memory (i.e., size, ...)	`tables()`
Show Keys in data.table	`key(dt1)`
Create index for col1 and reorder data according to col1	`setkey(dt1, col1)`
Use Key to Select Data	`dt1[c('col1Value1', 'col1Value2'), ]`
Multiple Key Select	`dt1[J('1', c('2', '3')), ]`
Aggregation**	`dt1[, list(col1 = mean(col1)), by = col2]` `dt1[, list(col1 = mean(col1), col2Sum = sum(col2)), by = list(col3, col4)]`

* Accessing columns must be done via list of actual names, not as characters. If column names are characters, then "with" argument should be set to FALSE.

** Aggregate and d*ply functions will work, but built-in aggregation functionality of data.table is faster

MATRIX

- Similar to data.frame except every element must be the SAME type, most commonly all numerics
- Functions that work with data.frame should work with matrix as well

Create Matrix	`matrix1 <- matrix(1:10, nrow = 5)`, # fills rows 1 to 5, column 1 with 1:5, and column 2 with 6:10
Matrix Multiplication	`matrix1 %*% t(matrix2)` # where t() is transpose

ARRAY

- Multidimensional vector of the SAME type
- `array1 <- array(1:12, dim = c(2, 3, 2))`
- Using arrays is not recommended
- Matrices are restricted to two dimensions while array can have any dimension

Source : https://intellipaat.com/

SQL

1. *SQL_Detailed*
2. *LearnSQL*
3. *SQL_Tutorial*
4. *SQL_Joins*
5. *Intellipat*
6. BuggyProgrammer

Sources

- W3Schools.com
- DataQuest.io

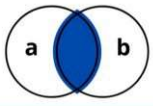
SQL CHEATSHEET

@AbzAaron

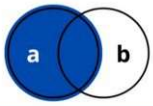
Commands / Clauses

SELECT	Select data from database
FROM	Specify table we're pulling from
WHERE	Filter query to match a condition
AS	Rename column or table with alias
JOIN	Combine rows from 2 or more tables
AND	Combine query conditions. All must be met
OR	Combine query conditions. One must be met
LIMIT	Limit rows returned. See also FETCH & TOP
IN	Specify multiple values when using WHERE
CASE	Return value on a specified condition
IS NULL	Return only rows with a NULL value
LIKE	Search for patterns in column
COMMIT	Write transaction to database
ROLLBACK	Undo a transaction block
ALTER TABLE	Add/Remove columns from table
UPDATE	Update table data
CREATE	Create TABLE, DATABASE, INDEX or VIEW
DELETE	Delete rows from table
INSERT	Add single row to table
DROP	Delete TABLE, DATABASE, or INDEX
GROUP BY	Group data into logical sets
ORDER BY	Set order of result. Use DESC to reverse order
HAVING	Same as WHERE but filters groups
COUNT	Count number of rows
SUM	Return sum of column
AVG	Return average of column
MIN	Return min value of column
MAX	Return max value of column

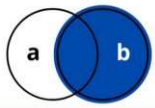
Joins



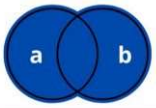
a INNER JOIN b



a LEFT JOIN b



a RIGHT JOIN b



a FULL OUTER JOIN b

Examples

Select all columns with filter applied

```
SELECT * FROM tbl
WHERE col > 5;
```

Select first 10 rows for two columns

```
SELECT col1, col2
FROM tbl LIMIT 10;
```

Select all columns with multiple filters

```
SELECT * FROM tbl
WHERE col1 > 5 OR col2 < 2;
```

Select all rows from col1 & col2 ordering by col1

```
SELECT col1, col2
FROM tbl ORDER BY 1;
```

Return count of rows in table

```
SELECT COUNT(*)
FROM tbl;
```

Return sum of col1

```
SELECT SUM(col1)
FROM tbl;
```

Return max value for col1

```
SELECT MAX(col1)
FROM tbl;
```

Compute summary stats by grouping col2

```
SELECT AVG(col1) FROM tbl
GROUP BY col2;
```

Combine data from 2 tables using left join

```
SELECT * FROM tbl1 AS t1 LEFT JOIN
tbl2 AS t2 ON t2.col1 = t1.col1;
```

Aggregate and filter result

```
SELECT col1,
       COUNT(*) AS total
FROM tbl
GROUP BY col1
HAVING COUNT(*) > 10;
```

Implementation of CASE statement

```
SELECT col1,
CASE
    WHEN col1 > 10 THEN 'more than 10'
    WHEN col1 < 10 THEN 'less than 10'
    ELSE '10'
END AS NewColumnName
FROM tbl;
```

Data Definition Language

CREATE

```
CREATE DATABASE MyDatabase;

CREATE TABLE MyTable (
    id int,
    name varchar(10));

CREATE INDEX IndexName
ON TableName(col1);
```

ALTER

```
ALTER TABLE MyTable
DROP COLUMN col5;

ALTER TABLE MyTable
ADD col5 int;
```

DROP

```
DROP DATABASE MyDatabase;
DROP TABLE MyTable;
```

Data Manipulation Language

UPDATE

```
UPDATE MyTable
SET col1 = 56
WHERE col2 = 'something';
```

INSERT

```
INSERT INTO MyTable (col1, col2)
VALUES ('value1', 'value2');
```

DELETE

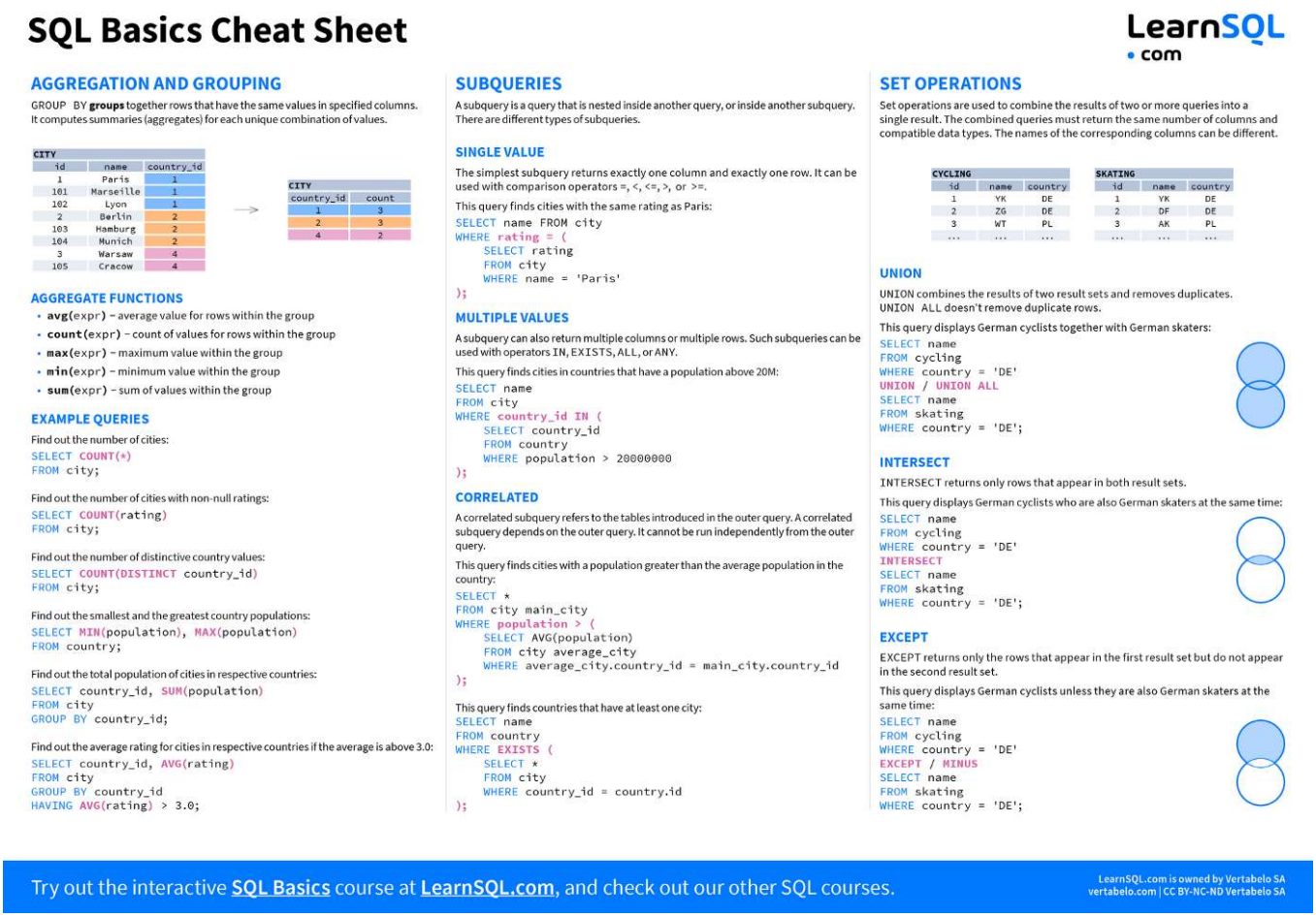
```
DELETE FROM MyTable
WHERE col1 = 'something';
```

SELECT

```
SELECT col1, col2
FROM MyTable;
```

Order Of Execution

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT



SQL Basics Cheat Sheet

LearnSQL.com

AGGREGATION AND GROUPING

GROUP BY **groups** together rows that have the same values in specified columns. It computes summaries (aggregates) for each unique combination of values.

CITY

id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4

→

CITY

country_id	count
1	3
2	3
4	2

AGGREGATE FUNCTIONS

- **avg**(expr) – average value for rows within the group
- **count**(expr) – count of values for rows within the group
- **max**(expr) – maximum value within the group
- **min**(expr) – minimum value within the group
- **sum**(expr) – sum of values within the group

EXAMPLE QUERIES

Find out the number of cities:

```
SELECT COUNT(*)
FROM city;
```

Find out the number of cities with non-null ratings:

```
SELECT COUNT(rating)
FROM city;
```

Find out the number of distinctive country values:

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Find out the smallest and the greatest country populations:

```
SELECT MIN(population), MAX(population)
FROM country;
```

Find out the total population of cities in respective countries:

```
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;
```

Find out the average rating for cities in respective countries if the average is above 3.0:

```
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;
```

SUBQUERIES

A subquery is a query that is nested inside another query, or inside another subquery. There are different types of subqueries.

SINGLE VALUE

The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators =, <, <=, >, or >=.

This query finds cities with the same rating as Paris:

```
SELECT name FROM city
WHERE rating = (
    SELECT rating
    FROM city
    WHERE name = 'Paris'
);
```

MULTIPLE VALUES

A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators IN, EXISTS, ALL, or ANY.

This query finds cities in countries that have a population above 20M:

```
SELECT name
FROM city
WHERE country_id IN (
    SELECT country_id
    FROM country
    WHERE population > 20000000
);
```

CORRELATED

A correlated subquery refers to the tables introduced in the outer query. A correlated subquery depends on the outer query. It cannot be run independently from the outer query.

This query finds cities with a population greater than the average population in the country:

```
SELECT *
FROM city main_city
WHERE population > (
    SELECT AVG(population)
    FROM city average_city
    WHERE average_city.country_id = main_city.country_id
);
```

This query finds countries that have at least one city:

```
SELECT name
FROM country
WHERE EXISTS (
    SELECT *
    FROM city
    WHERE country_id = country.id
);
```

SET OPERATIONS

Set operations are used to combine the results of two or more queries into a single result. The combined queries must return the same number of columns and compatible data types. The names of the corresponding columns can be different.

CYCLING

id	name	country
1	YK	DE
2	ZG	DE
3	WT	PL
...	...	...

SKATING

id	name	country
1	YK	DE
2	DF	DE
3	AK	PL
...	...	...

UNION

UNION combines the results of two result sets and removes duplicates. UNION ALL doesn't remove duplicate rows.

This query displays German cyclists together with German skaters:

```
SELECT name
FROM cycling
WHERE country = 'DE'
UNION / UNION ALL
SELECT name
FROM skating
WHERE country = 'DE';
```

INTERSECT

INTERSECT returns only rows that appear in both result sets.

This query displays German cyclists who are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';
```

EXCEPT

EXCEPT returns only the rows that appear in the first result set but do not appear in the second result set.

This query displays German cyclists unless they are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
EXCEPT / MINUS
SELECT name
FROM skating
WHERE country = 'DE';
```

Try out the interactive **SQL Basics** course at **LearnSQL.com**, and check out our other SQL courses.

Source : https://learnsql.com/

Web Scraping Tools

Source

1. *Data Camp (Python webscraping)*
2. *Beautifulsoup -Tutorialspoint*
3. *Web Scraping with Scrapy and MongoDB*
4. *Beautiful Soup — Akul's Blog*
5. *Beautiful Soup — Cheatography*
6. *Selenium — Intellipaat*
7. *Selenium — Automatetheplanet*

Some blogs and GitHub repositories with more details : —

Medium Blog by Frank

Blog by Hartley Brody

Use web scraping to become more competitive

Source

***Here** is am article on Legal issues associated with Web scraping.*

Data Visualization Tools

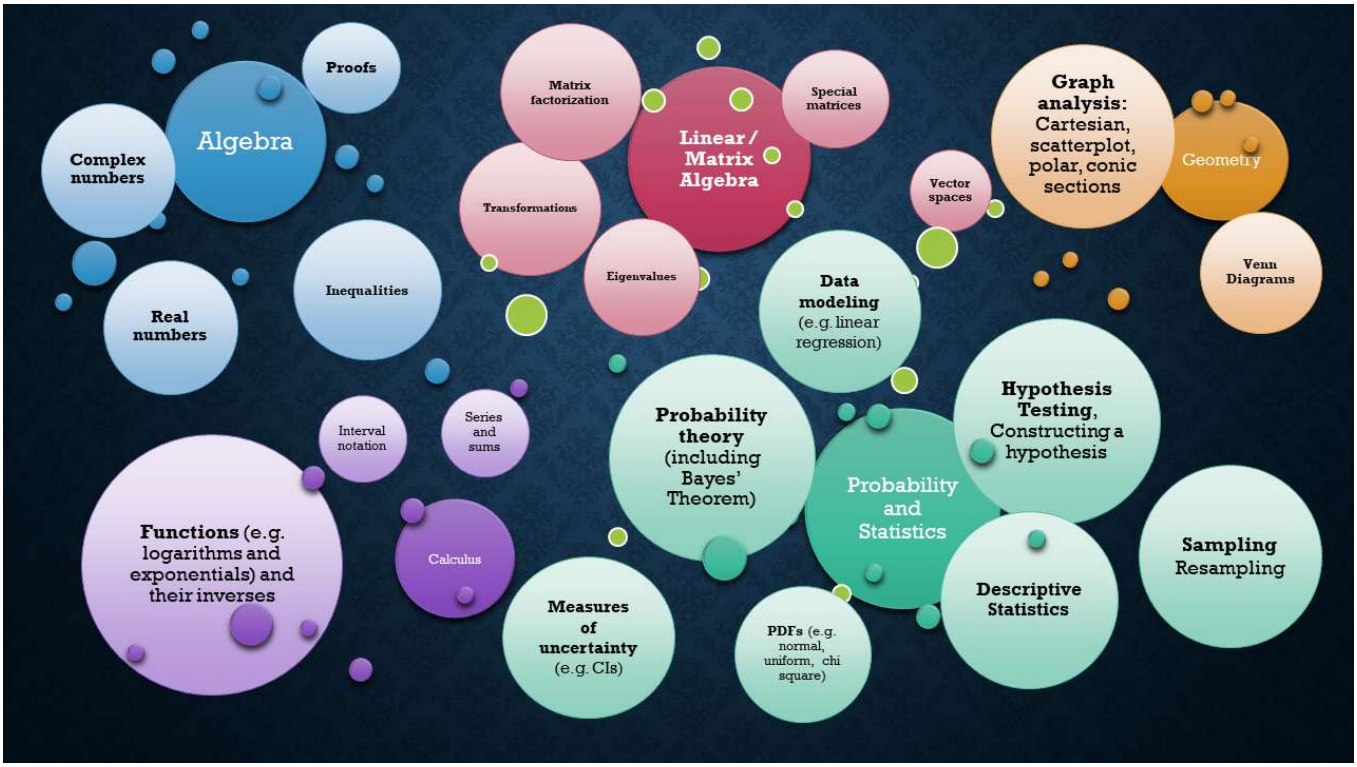
Tableau

1. *Intellipat*
2. *Tableau_community*
3. *howto*
4. *Datacamp*

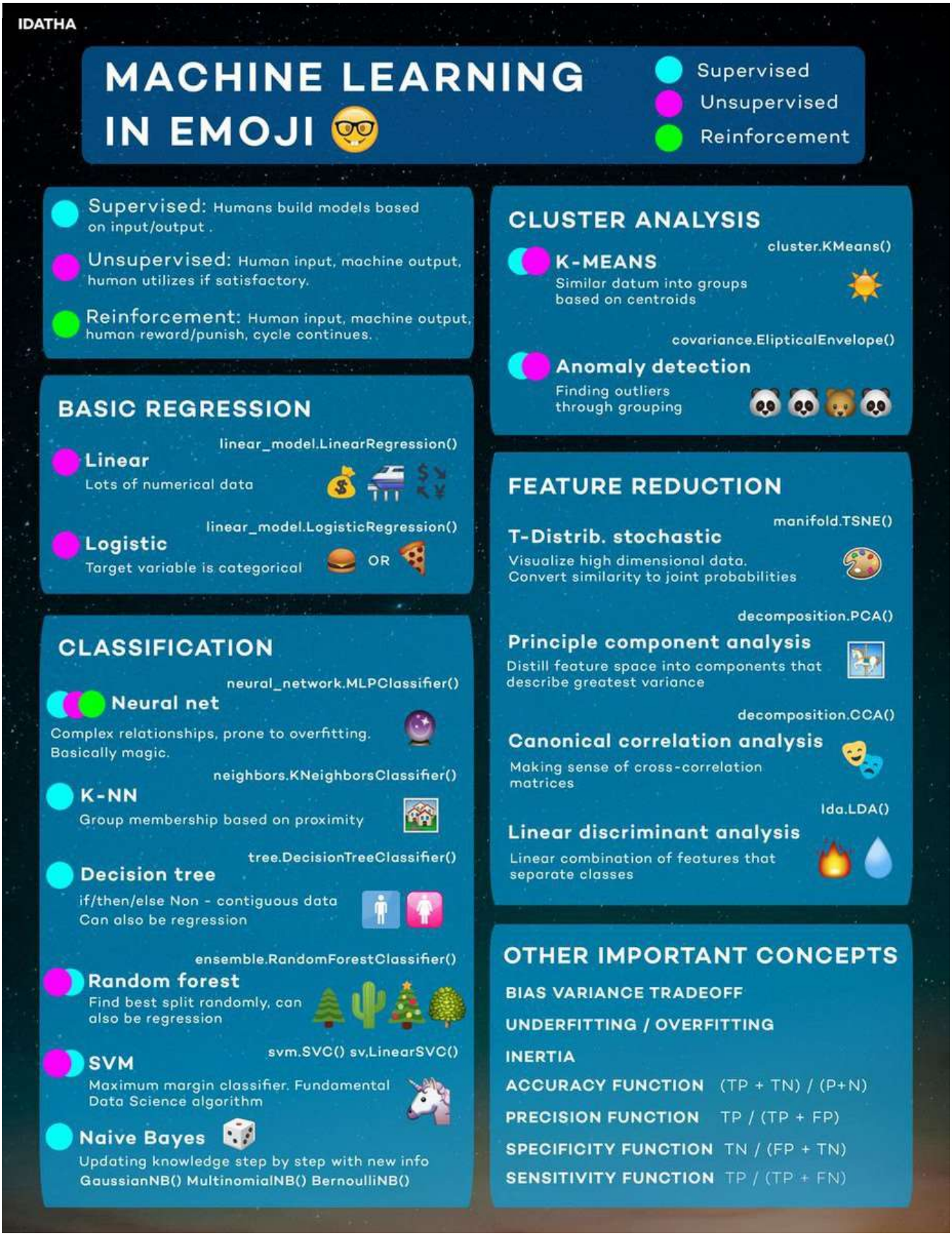
Maths and Statistics

1. *Probability*
2. *Statistics*
3. *Linear Algebra*

6. *Stats_for_interviews*



Machine Learning Algorithms

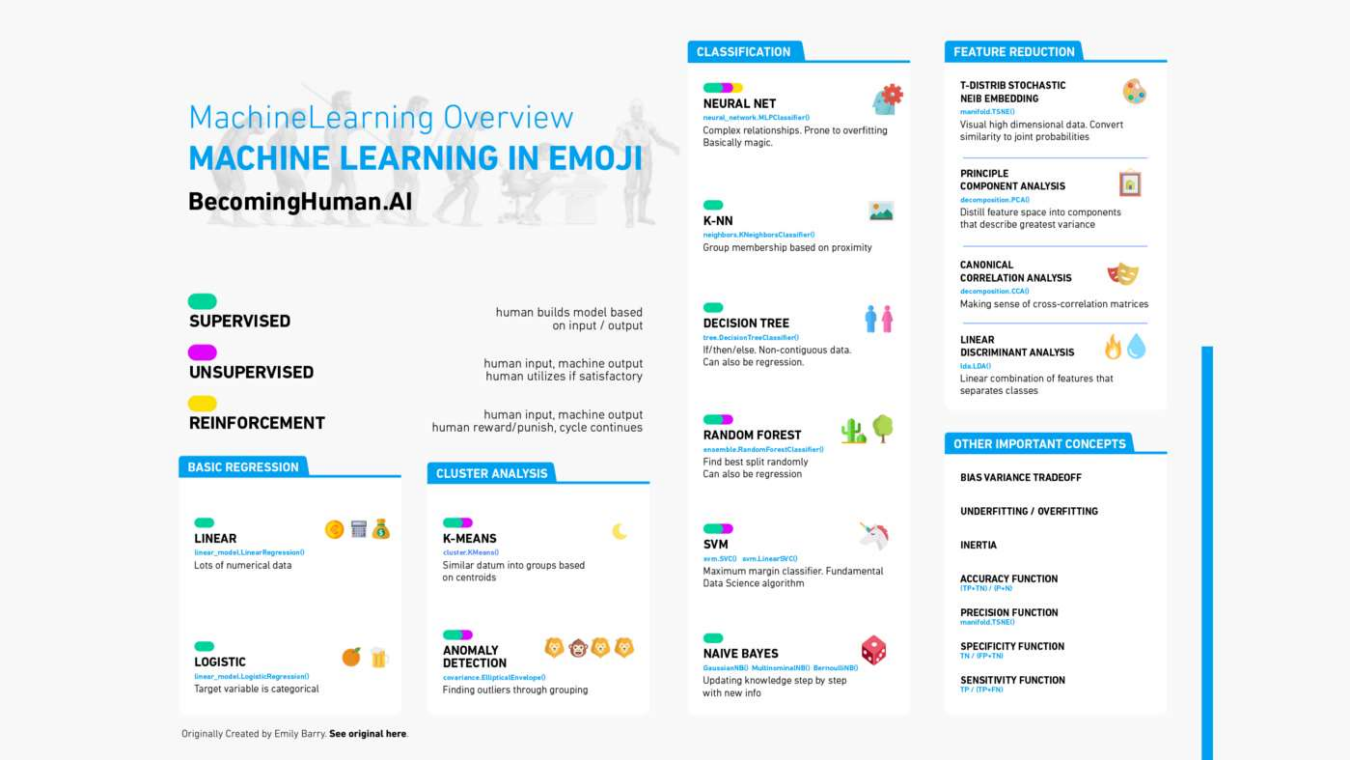


Source : https://twitter.com/IDATHAuy/status/1035531415893540864

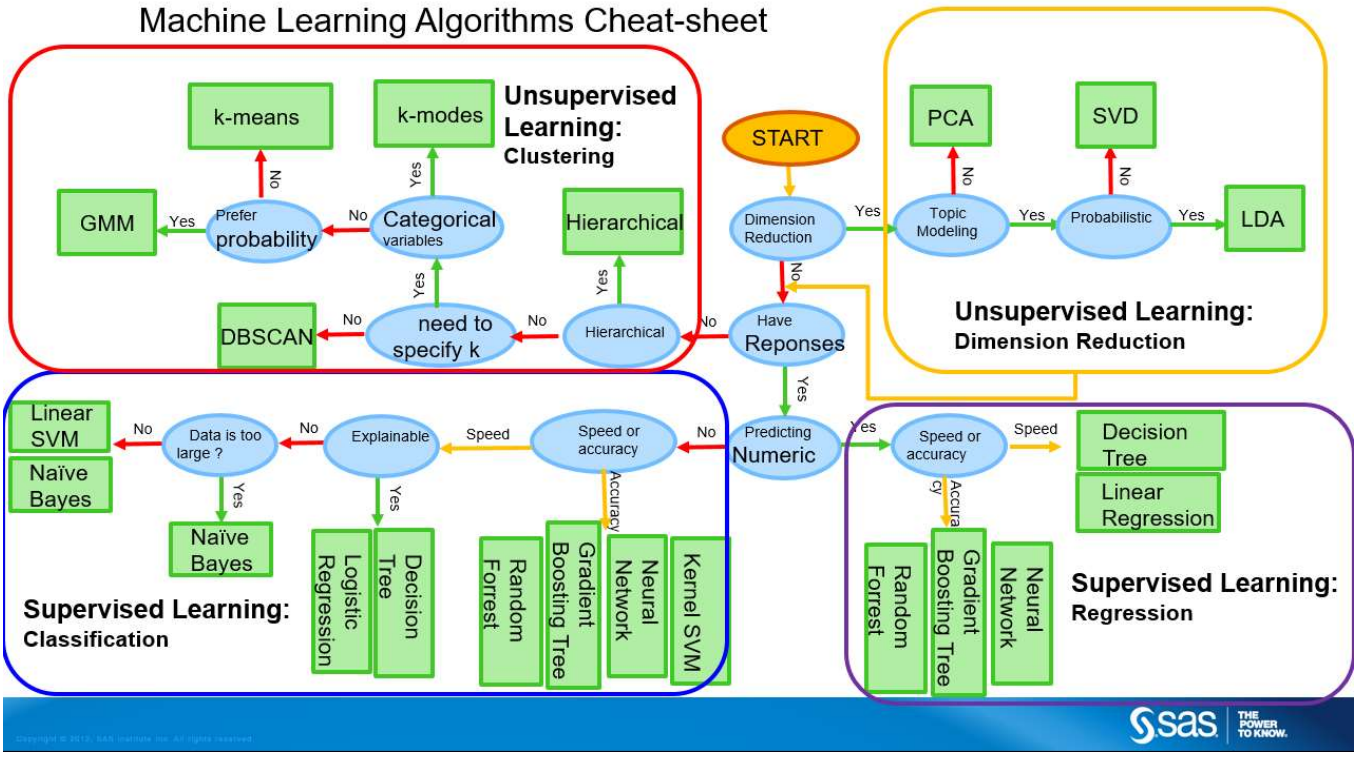
Machine Learning Algorithms

Sas_ml

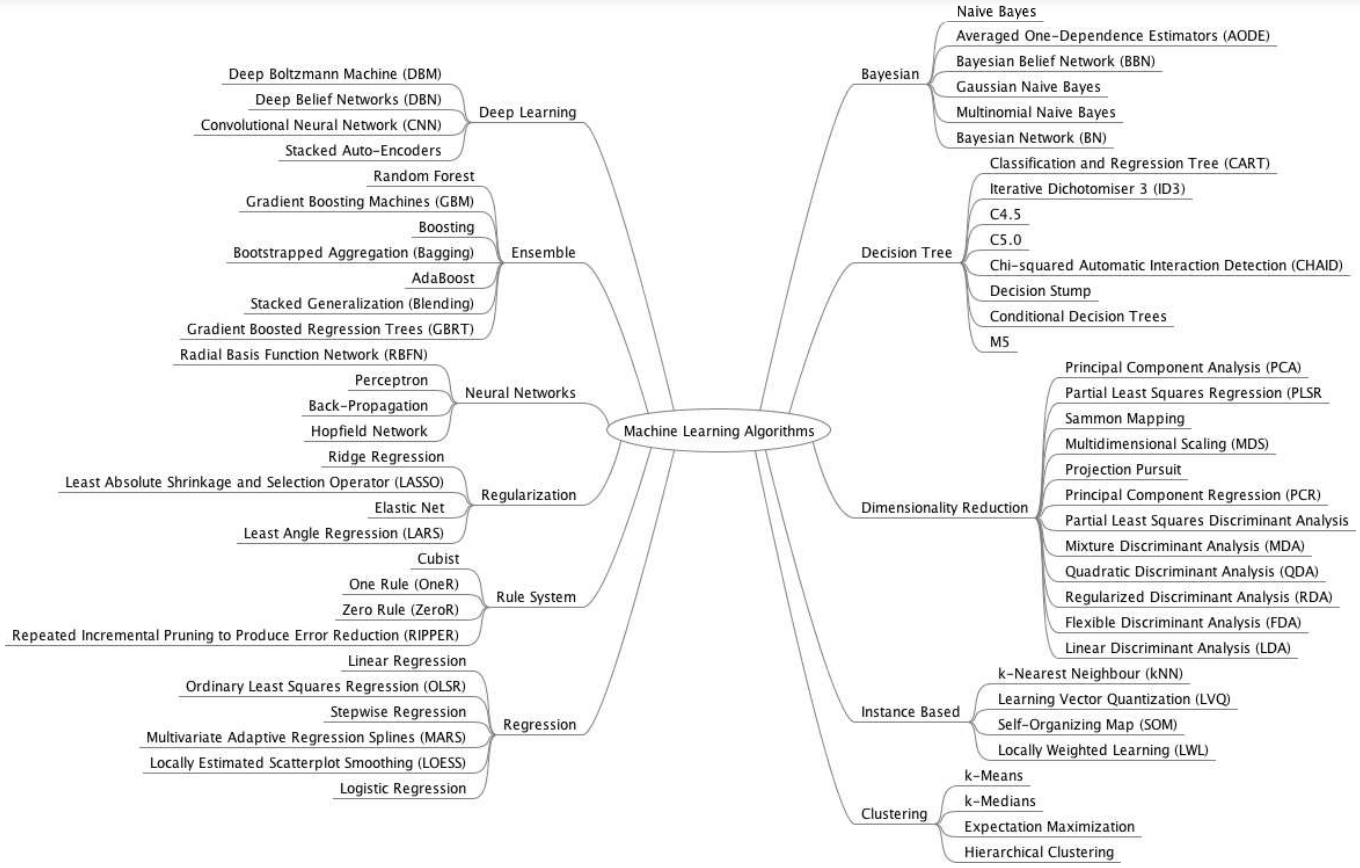
R_bloggers

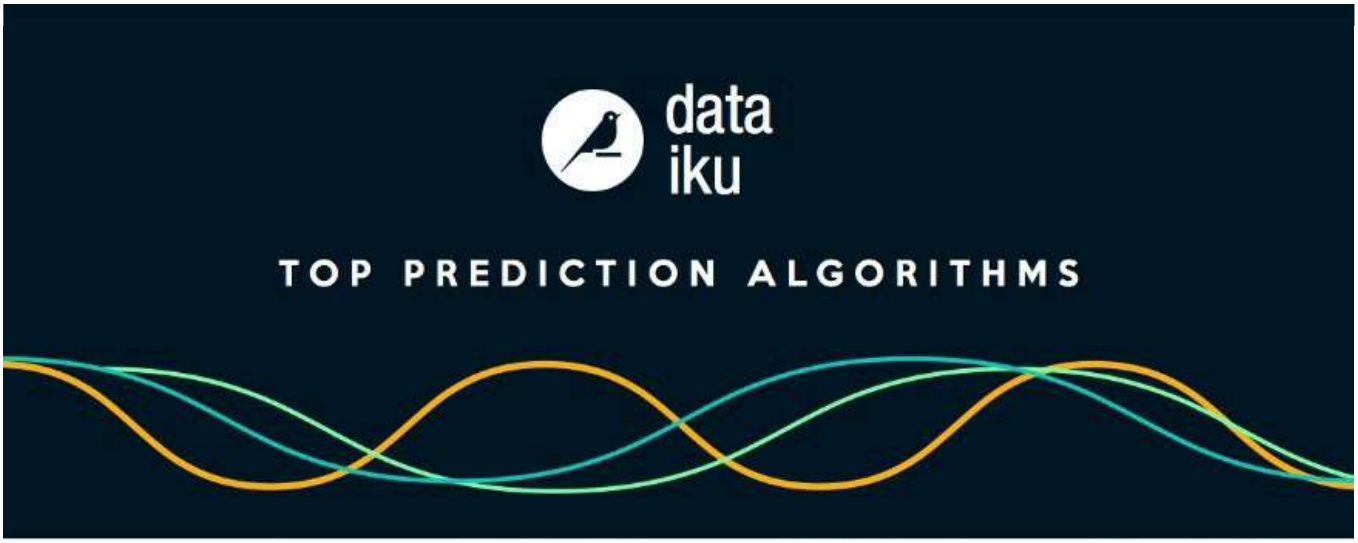


Source : https://becominghuman.ai/



Source : https://coursebapu.com/

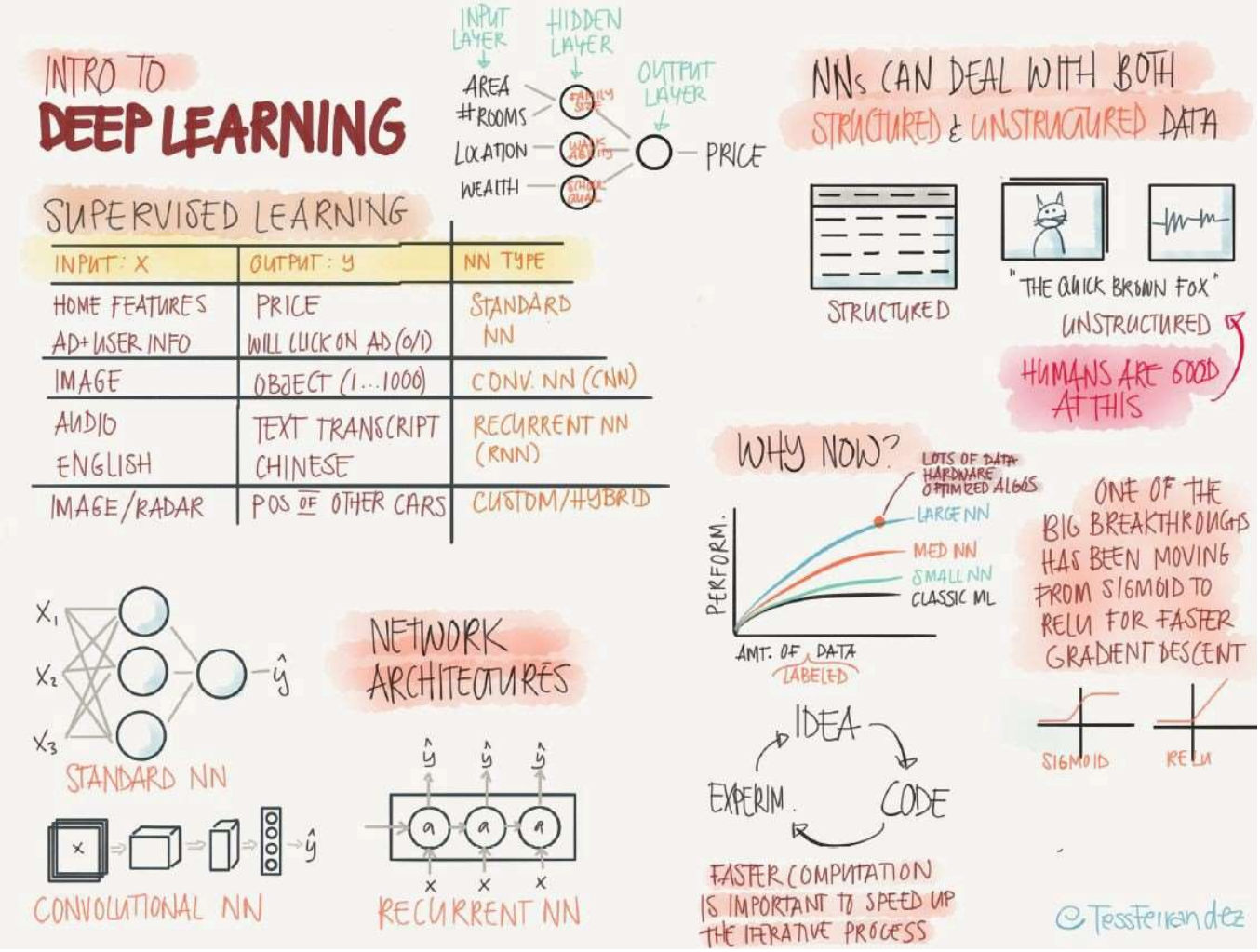




TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	X Can be slow to output predictions relative to other algorithms. X Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X A small change in the feature set or training set can create radical changes in the model. X Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	X Very, very slow to train, because they have so many layers. Require a lot of power. X Almost impossible to understand predictions.

Source :https://favouriteblog.com/

2. *Stanford(deep learning)*
3. *Stanford(neural_nets)*
4. *Datacamp (Keras)*
5. *Aimov_institute*



Source : https://sketchnotearmy.com/

Tensorflow

1. BecomingHuman.AI
2. Altoros
3. Github
4. Stanford.edu
5. Cheatography

About

TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

Skflow

Scikit Flow provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code. Scikit Flow is a simplified interface for TensorFlow, to get people started on predictive analytics and data mining. Scikit Flow has been merged into TensorFlow since version 0.8 and now called TensorFlow Learn.

Keras

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano

Installation

How to install new package in Python:

`pip install <package-name>`

Example: `pip install requests`

How to install tensorflow?

device = **cpu/gpu**

python_version = **cp27/cp34**

```
sudo pip install
https://storage.googleapis.com/
tensorflow/linux/$device/tensorflow-
0.8.0-$python_version-none-linux_x86
_64.whl
```

How to install Skflow

`pip install sklearn`

How to install Keras

`pip install keras`

update ~/.keras/keras.json - replace “theano” by “tensorflow”

Helpers

Python helper

Important functions

`type(object)`
Get object type

`help(object)`
Get help for object (list of available methods, attributes, signatures and so on)

`dir(object)`
Get list of object attributes (fields, functions)

`str(object)`
Transform an object to string

`object?`
Shows documentations about the object

`globals()`
Return the dictionary containing the current scope's global variables.

`locals()`
Update and return a dictionary containing the current scope's local variables.

`id(object)`
Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

```
import __builtin__
dir(__builtin__)
```

Other built-in functions

TensorFlow

Main classes

```
tf.Graph()
tf.Operation()
tf.Tensor()
tf.Session()
```

Some useful functions

```
tf.get_default_session()
tf.get_default_graph()
tf.reset_default_graph()
ops.reset_default_graph()
tf.device(“/cpu:0”)
tf.name_scope(value)
tf.convert_to_tensor(value)
```

TensorFlow Optimizers

```
GradientDescentOptimizer
AdadeltaOptimizer
AdagradOptimizer
MomentumOptimizer
AdamOptimizer
FtrlOptimizer
RMSPropOptimizer
```

Reduction

```
reduce_sum
reduce_prod
reduce_min
reduce_max
reduce_mean
reduce_all
reduce_any
accumulate_n
```

Activation functions

`tf.nn?`

relu
relu6
elu
softplus
softsign
dropout
bias_add
sigmoid
tanh
sigmoid_cross_entropy_with_logits
softmax
log_softmax
softmax_cross_entropy_with_logits
sparse_softmax_cross_entropy_with_logits
weighted_cross_entropy_with_logits
etc.

Skflow

Main classes

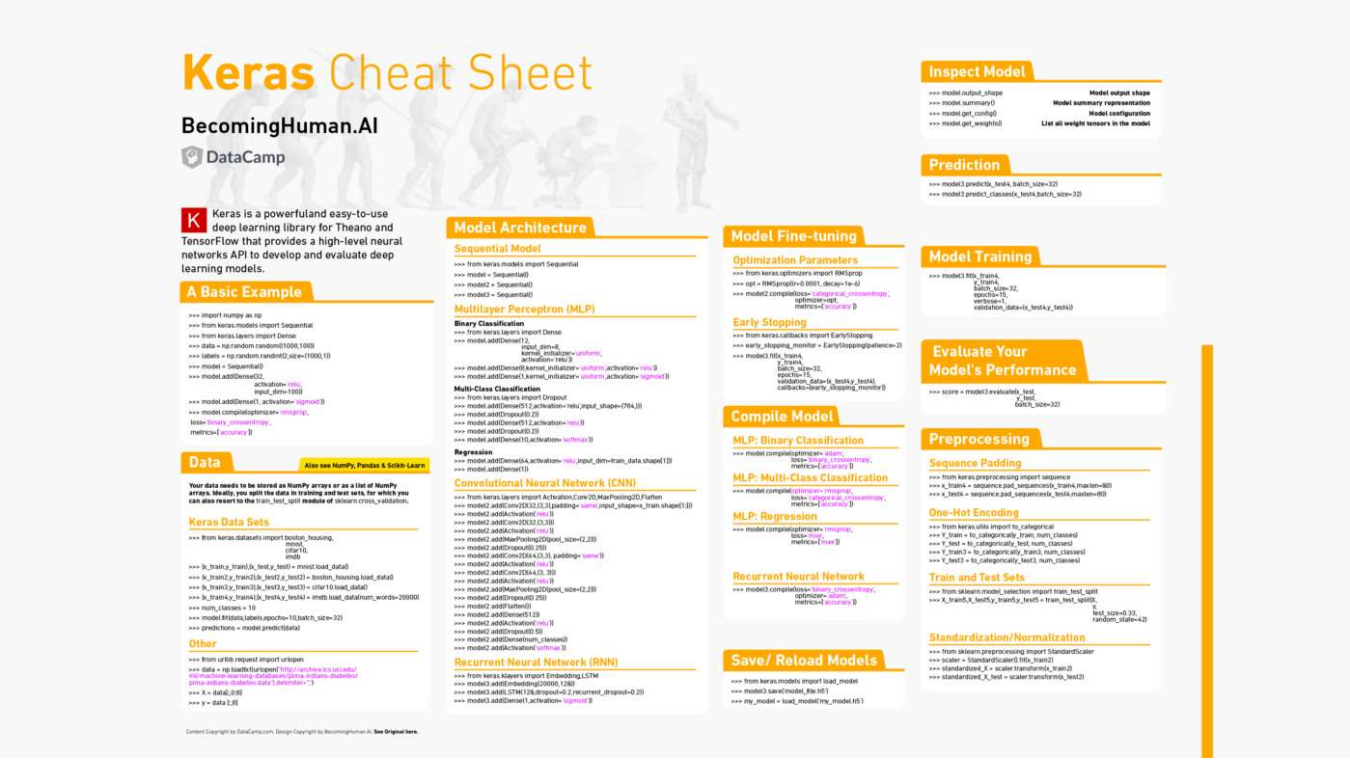
TensorFlowClassifier
TensorFlowRegressor
TensorFlowDNNClassifier
TensorFlowDNNRegressor
TensorFlowLinearClassifier
TensorFlowLinearRegressor
TensorFlowRNNClassifier
TensorFlowRNNRegressor
TensorFlowEstimator

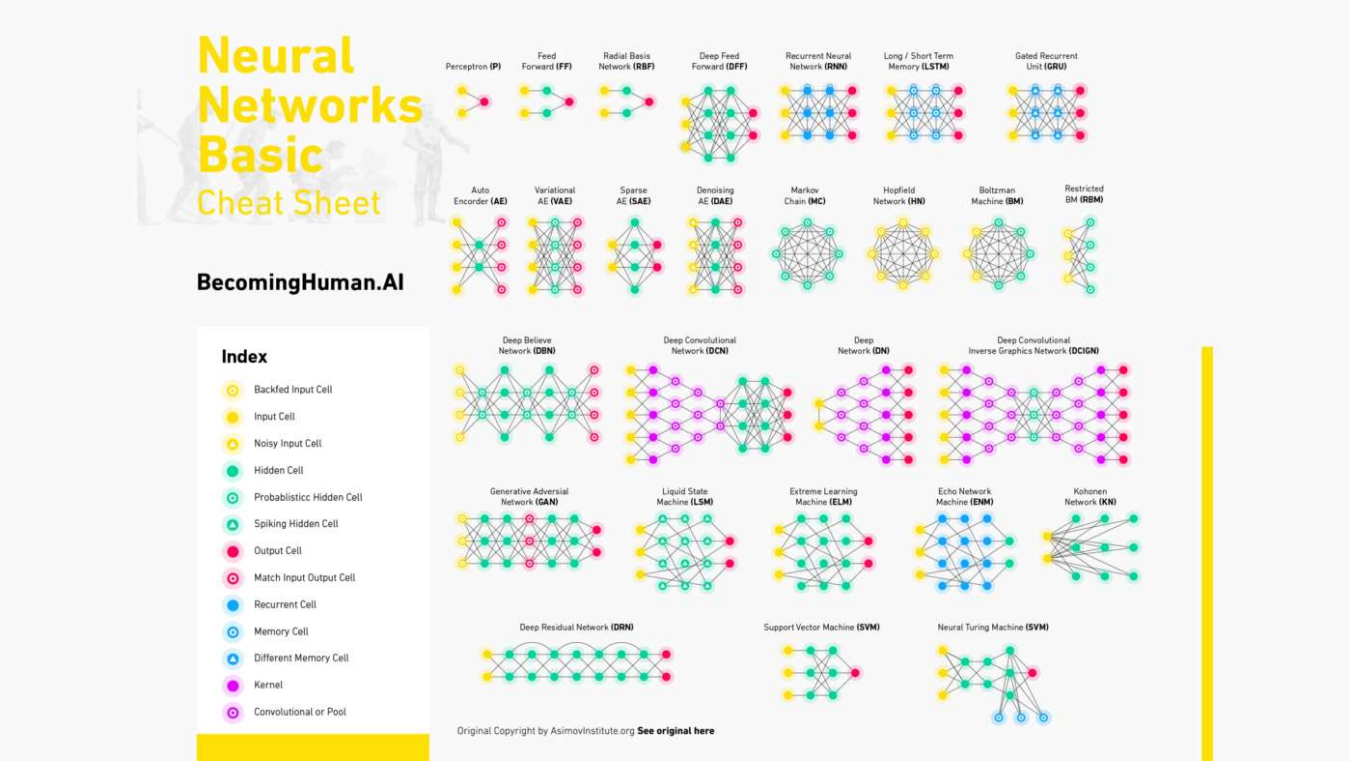
Each classifier and regressor have following fields

n_classes=0 (Regressor), n_classes are expected to be input (Classifiers)

batch_size=32,
steps=200, // except TensorFlowRNNClassifier - there is 50
optimizer='Adagrad',
learning_rate=0.1,


1. BuggyProgrammer
2. DataCamp
3. Quick_tutorial



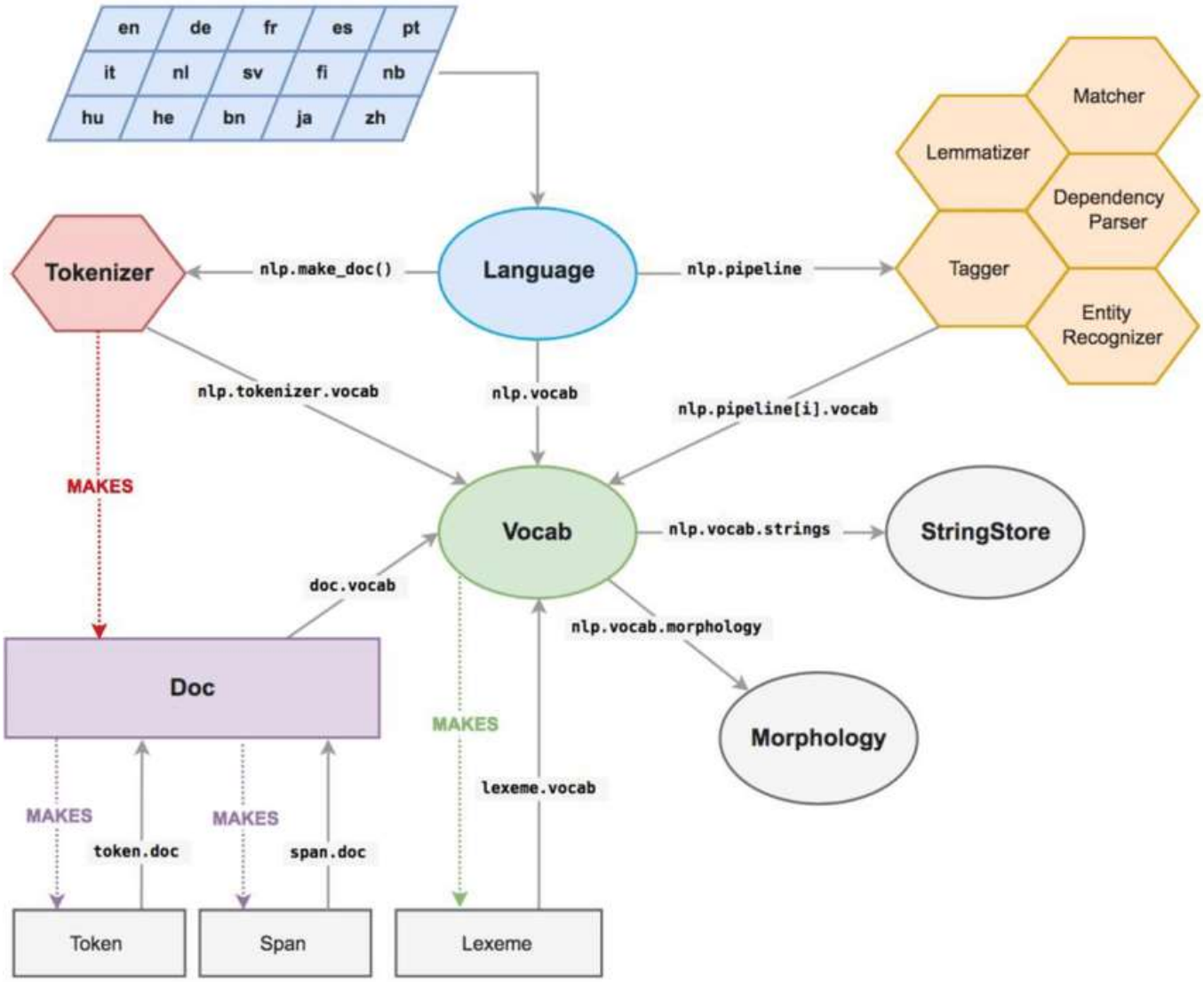


Source : https://becominghuman.ai/

2. Computingeverywhere
3. nltk.org

Spacy

1. Spacy.io
2. Datacamp
3. kaggle
4. Cheatography



Spacy Architecture (Source : https://www.researchgate.net)

IDE and OS

Linux

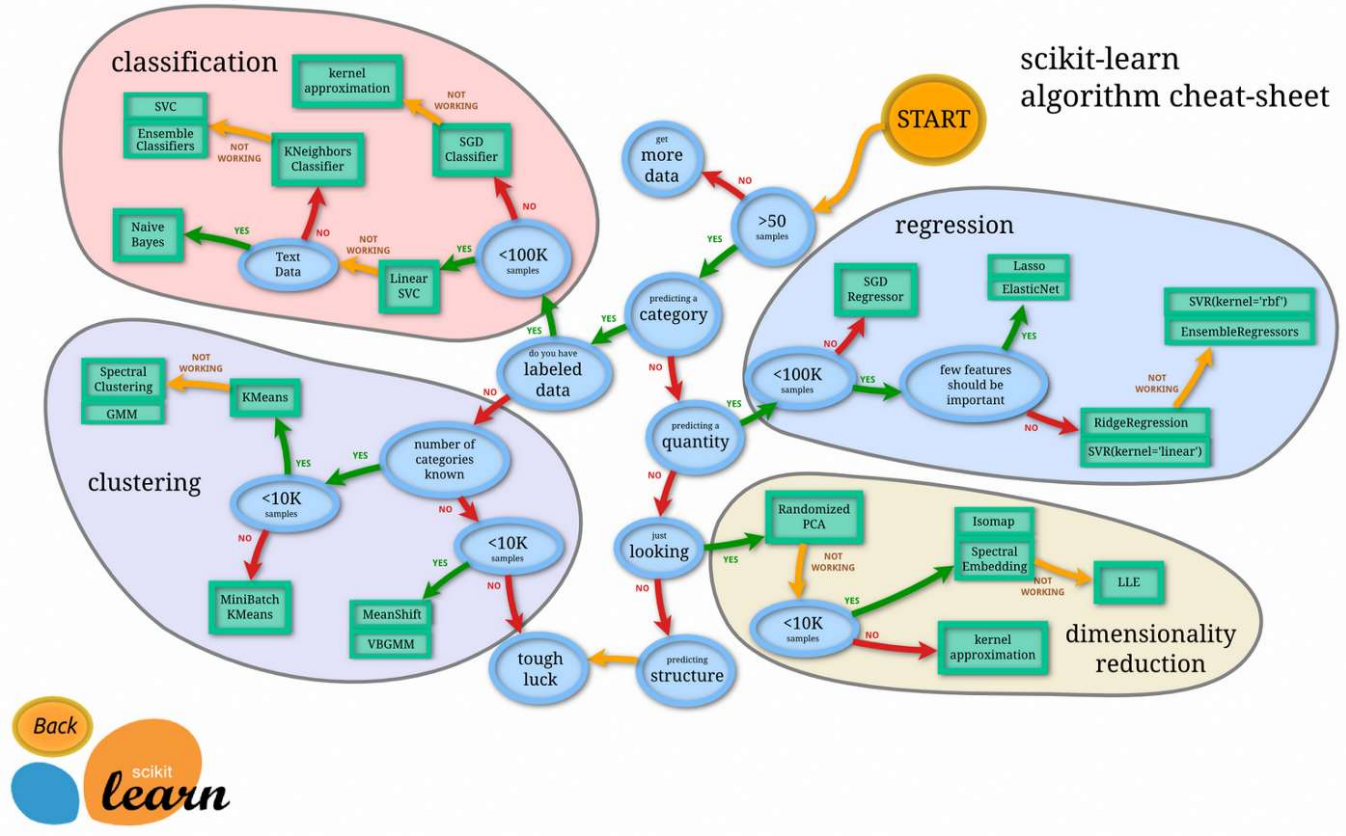
3. *Guru99*

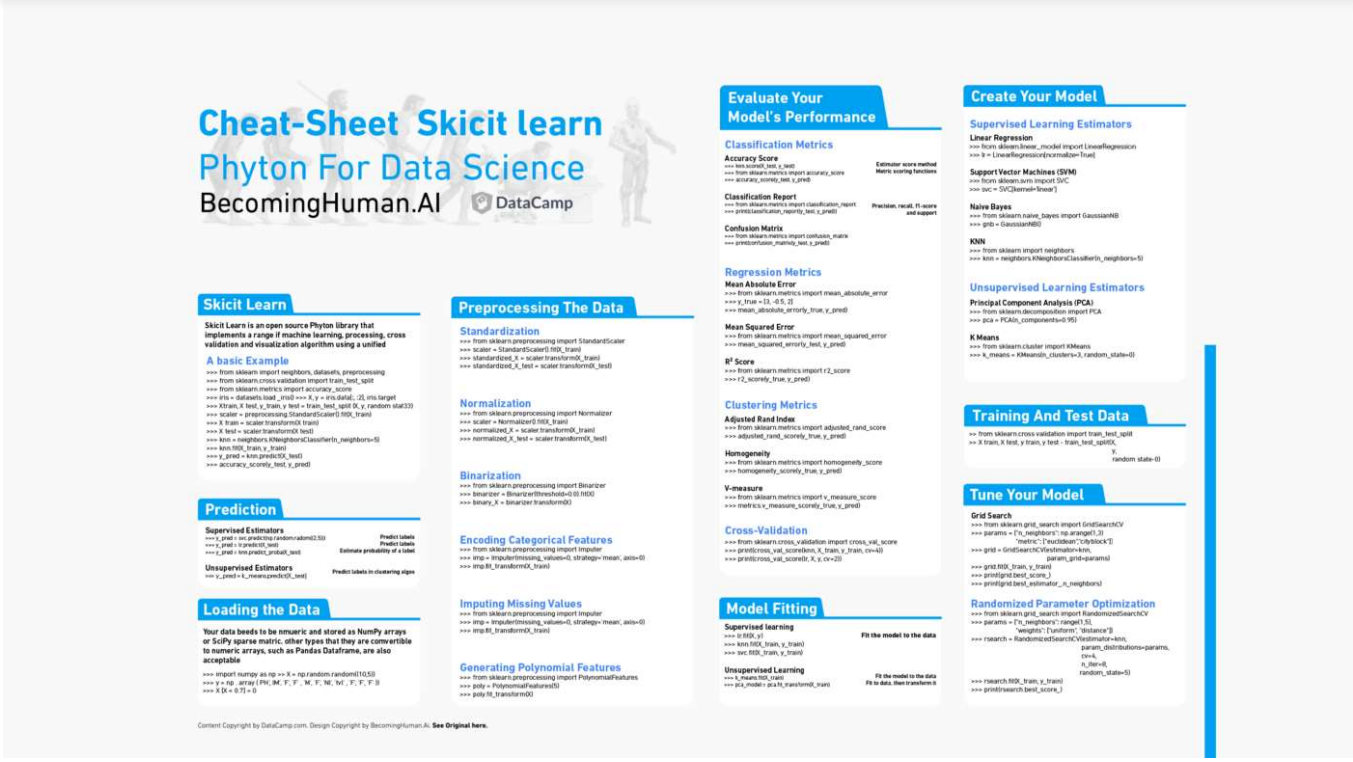
Jupyter Notebook

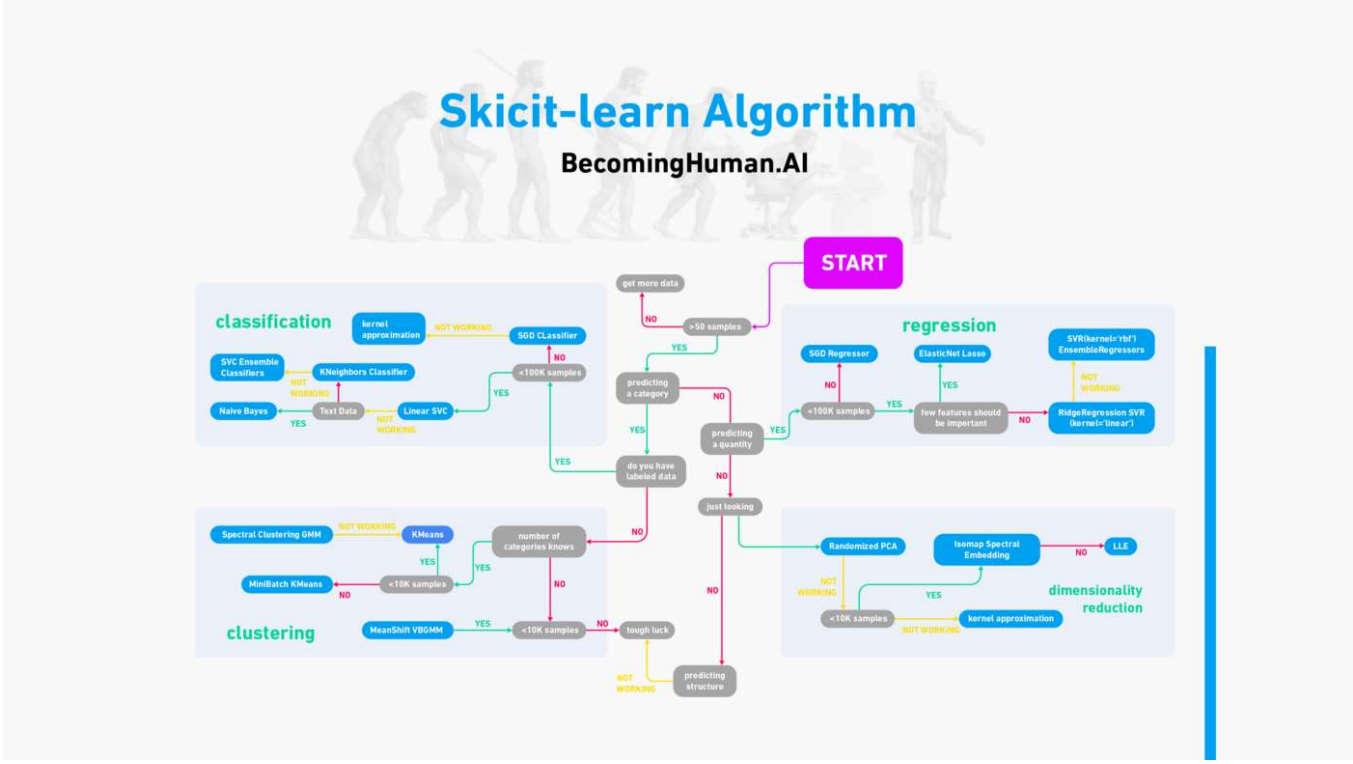
1. *Datacamp*
2. *Cheatography*
3. *Edureka*
4. *Anaconda*

ScikitLearn

1. *DataCamp*
2. *Edureka.co*
3. *Intellipaat*
4. *Cheatography*





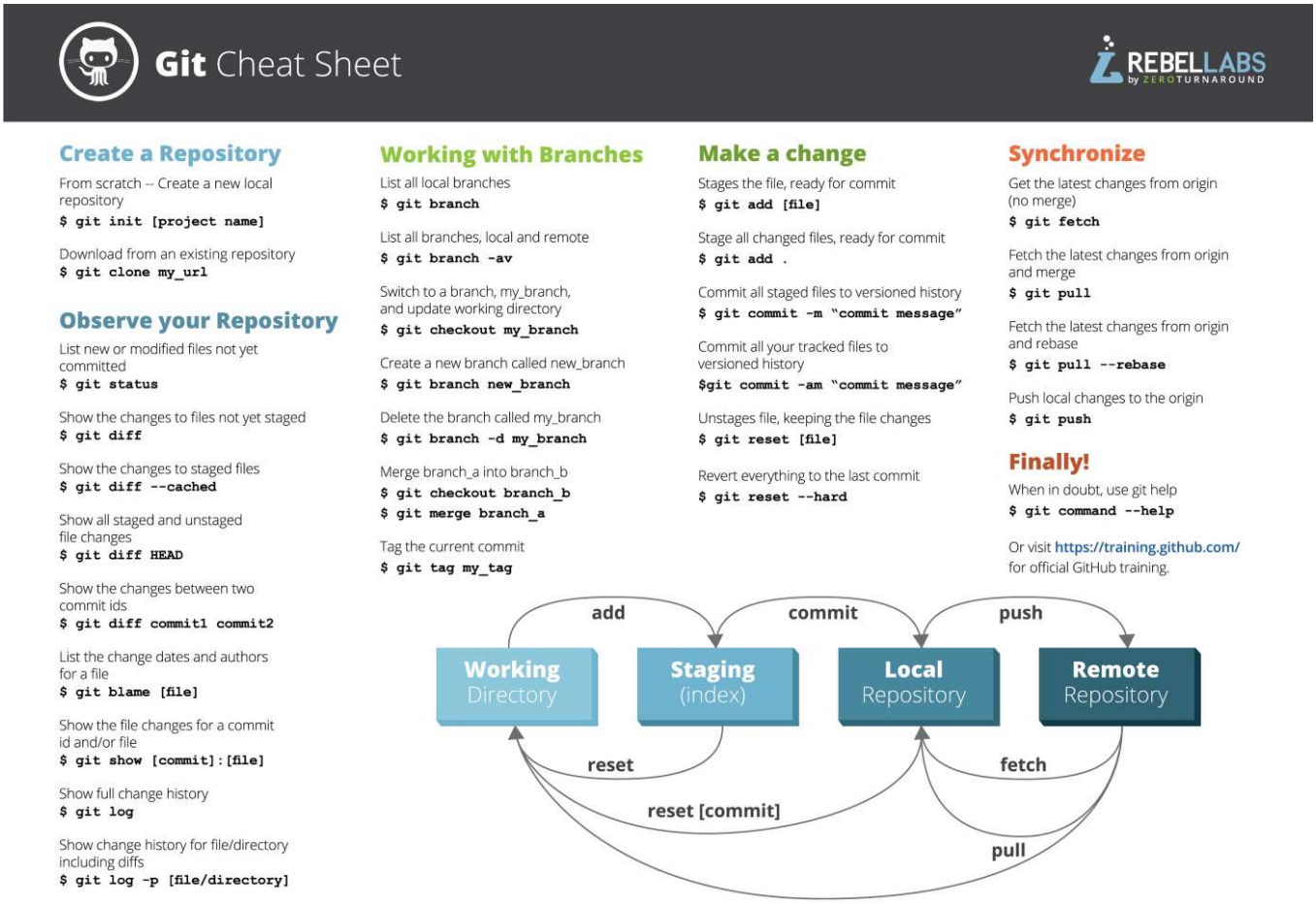


Source : https://becominghuman.ai/

GitHub

1. BitBucket
2. Freecodecamp

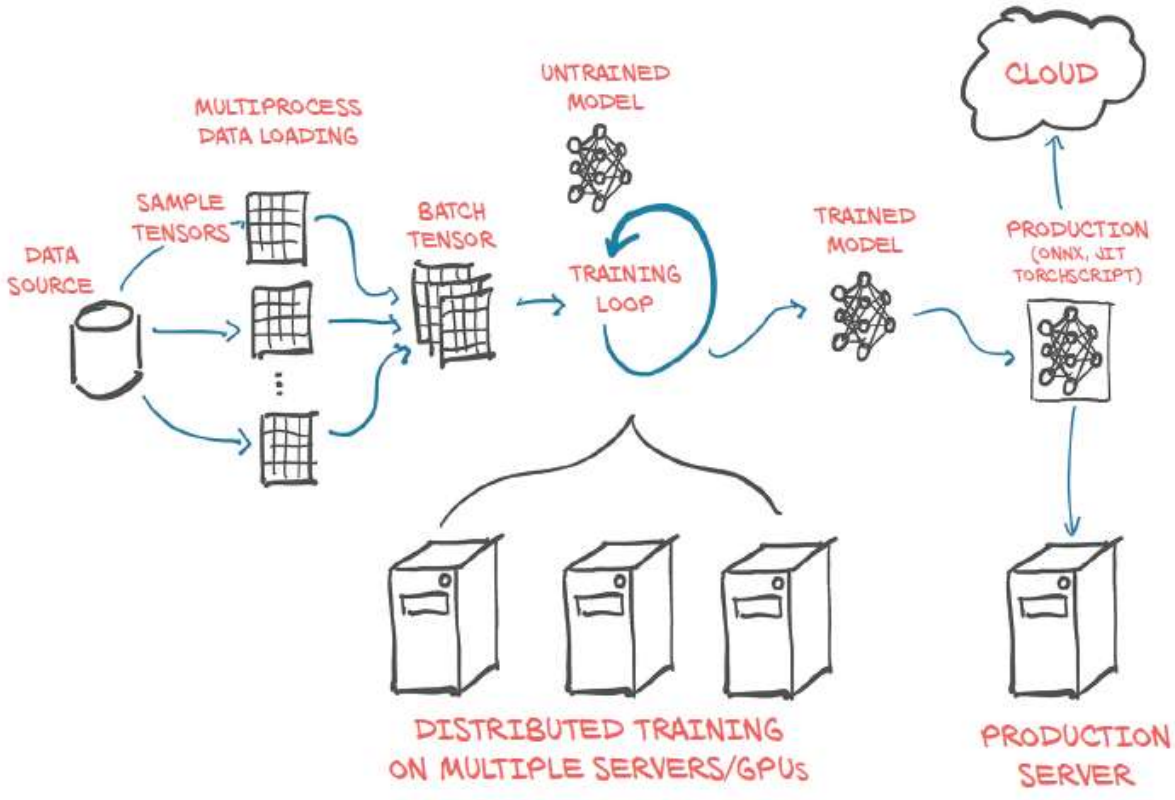
5. Git_Training
6. Intellipat



Source : REBELLABS

Pytorch

1. *Pytorch.org*
2. *Pytorch.org (for beginners)*
3. KDNuggets
4. *Cheatography*
5. *Github*
6. *ProgrammerSought*
7. *TechRepublic*



Pytorch Deep Learning Implementation (example)

Final Thoughts

So that was my effort in segregating and collecting the best cheat sheets on various Data Science facets. I hope it helps. In case I missed any or you guys have any suggestion or query, please do add your comments below.

I will be more than happy to add them to the list above!

Happy Learning!!

Cheers :)

Photo by Katrina Wright on Unsplash