Essential Cheat Sheets for Machine Learning and Deep Learning Engineers

Kailash Ahirwar · Follow
Published in Startups & Venture Capital · 4 min read · May 28, 2017

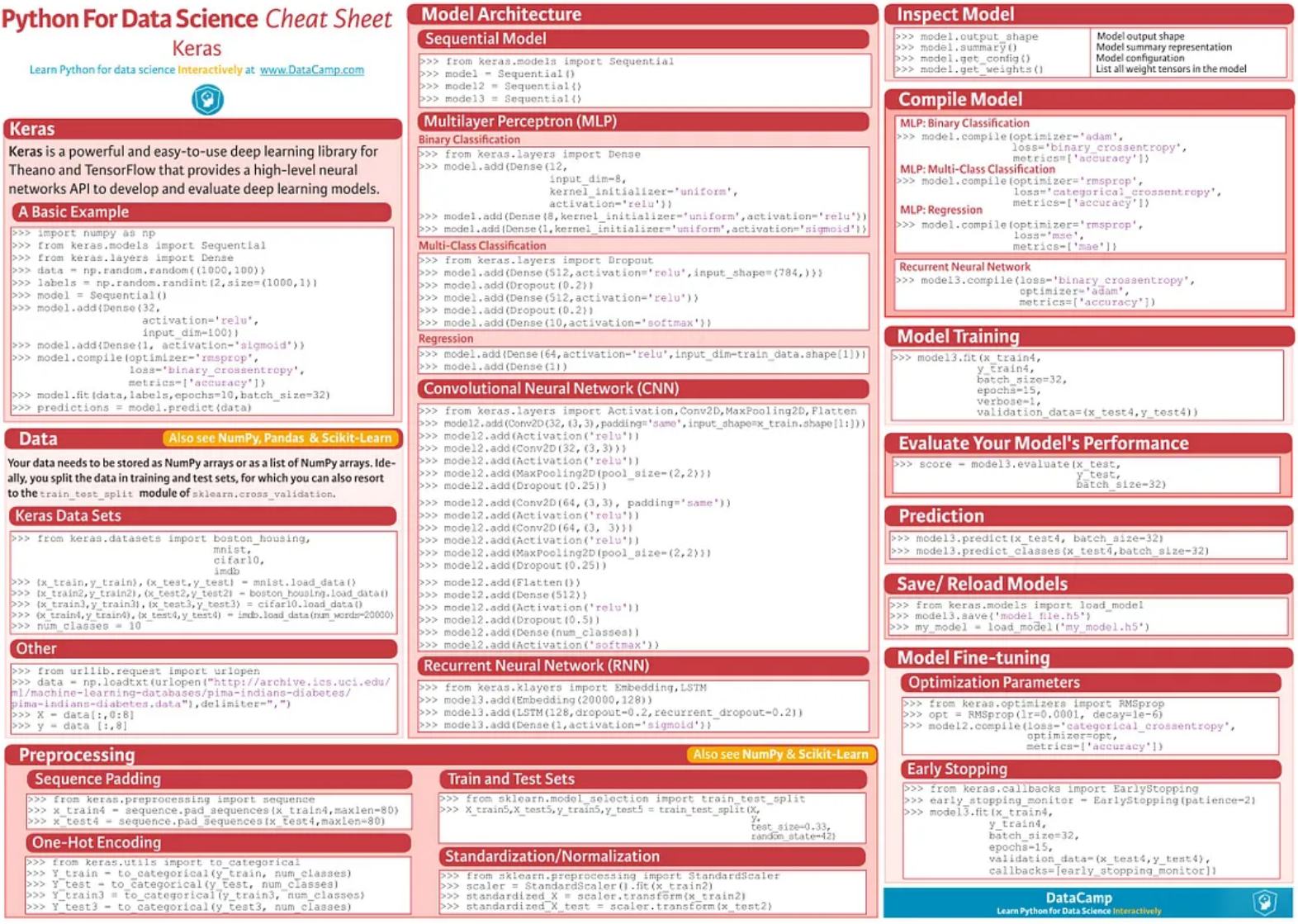
Machine learning is complex. For newbies, starting to learn machine learning can be painful if they don't have right resources to learn from. Most of the machine learning libraries are difficult to understand and learning curve can be a bit frustrating. I am creating a repository on Github(cheatsheets-ai) containing cheatsheets for different machine learning frameworks, gathered from different sources. Do visit the Github repository, also, contribute cheat sheets if you have any. Thanks.

List of Cheatsheets:

1. Keras
2. Numpy
3. Pandas
4. Scipy
5. Matplotlib
6. Scikit-learn

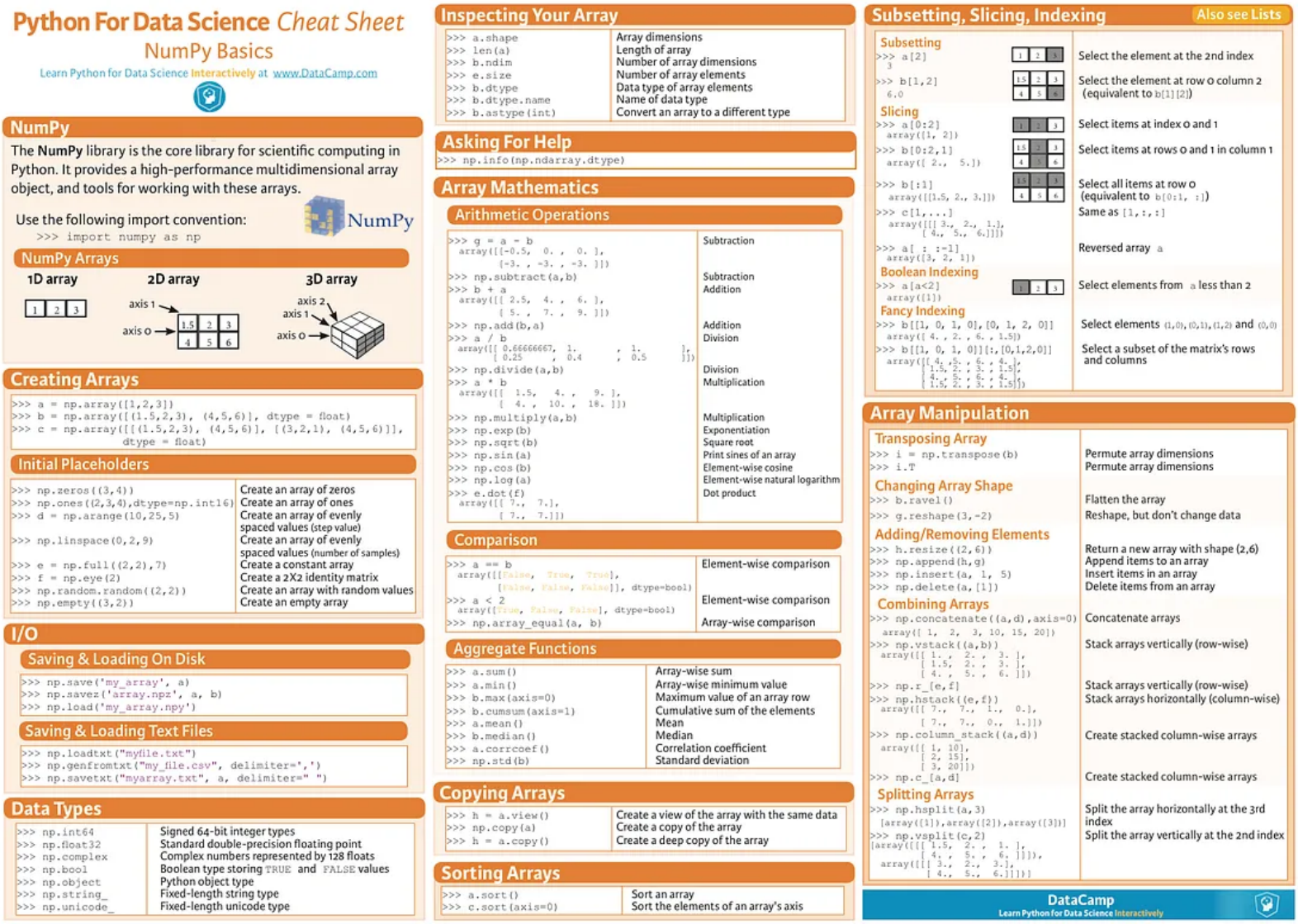
7. Neural Networks Zoo
8. ggplot2
9. PySpark
10. R Studio
11. Jupyter Notebook
12. Dask

1. Keras



Source — https://www.datacamp.com/community/blog/keras-cheat-sheet#gs.DRKeNMs

2. Numpy



Source — https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.AK5ZBgE

3. Pandas

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

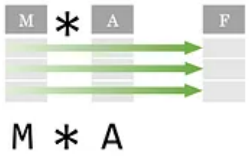
In a tidy data set:

&

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v'])))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

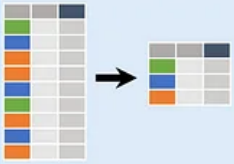
`max(axis=1)`
Element-wise max.

`min(axis=1)`
Element-wise min.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`abs()`
Absolute value.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`shift(-1)`
Copy with values lagged by 1.

`rank(method='dense')`
Ranks with no gaps.

`cumsum()`
Cumulative sum.

`rank(method='min')`
Ranks. Ties get min rank.

`cummax()`
Cumulative max.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`cummin()`
Cumulative min.

`rank(method='first')`
Ranks. Ties go to first value.

`cumprod()`
Cumulative product.

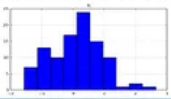
Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

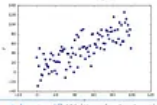
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(['_merge'],axis=1)
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Source — https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#gs.HPFoRIc

Python For Data Science *Cheat Sheet*

Pandas Basics

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

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

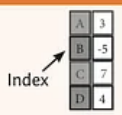
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

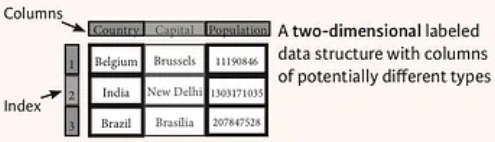
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-5
```
Get one element

```
>>> df[1:]
   Country     Capital  Population
1   India   New Delhi  1303171035
2  Brazil    Brasilia   207847528
```
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat[[0],[0]]
'Belgium'
```
Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[[0], ['Country']]
'Belgium'
```
Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country       Brazil
Capital     Brasilia
Population  207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasilia
```
Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```
Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```
Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```
Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```
Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```
Sort by row or column index
Sort a series by its values
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```
(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```
Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```
Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

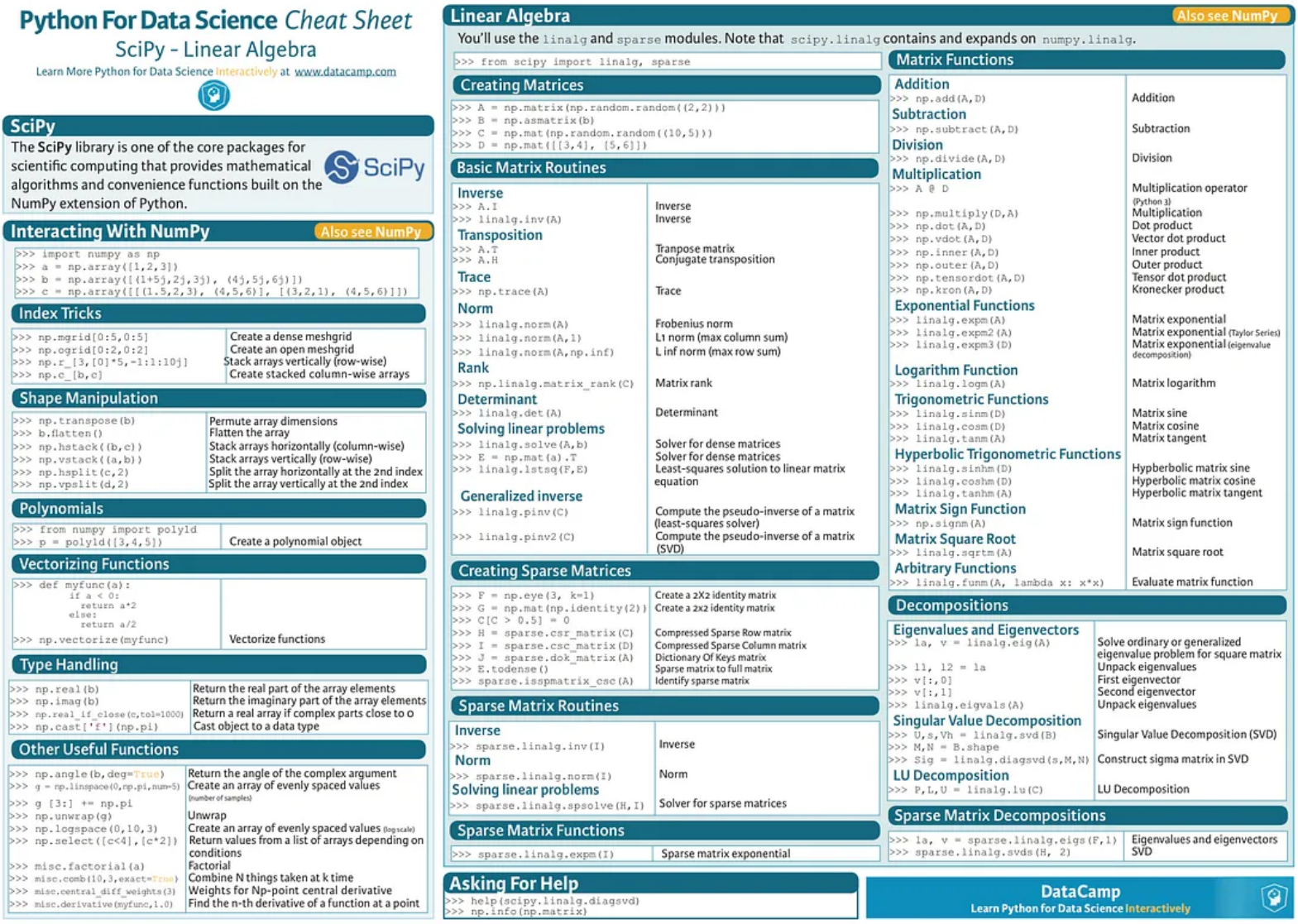
`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

DataCamp
Learn Python for Data Science Interactively

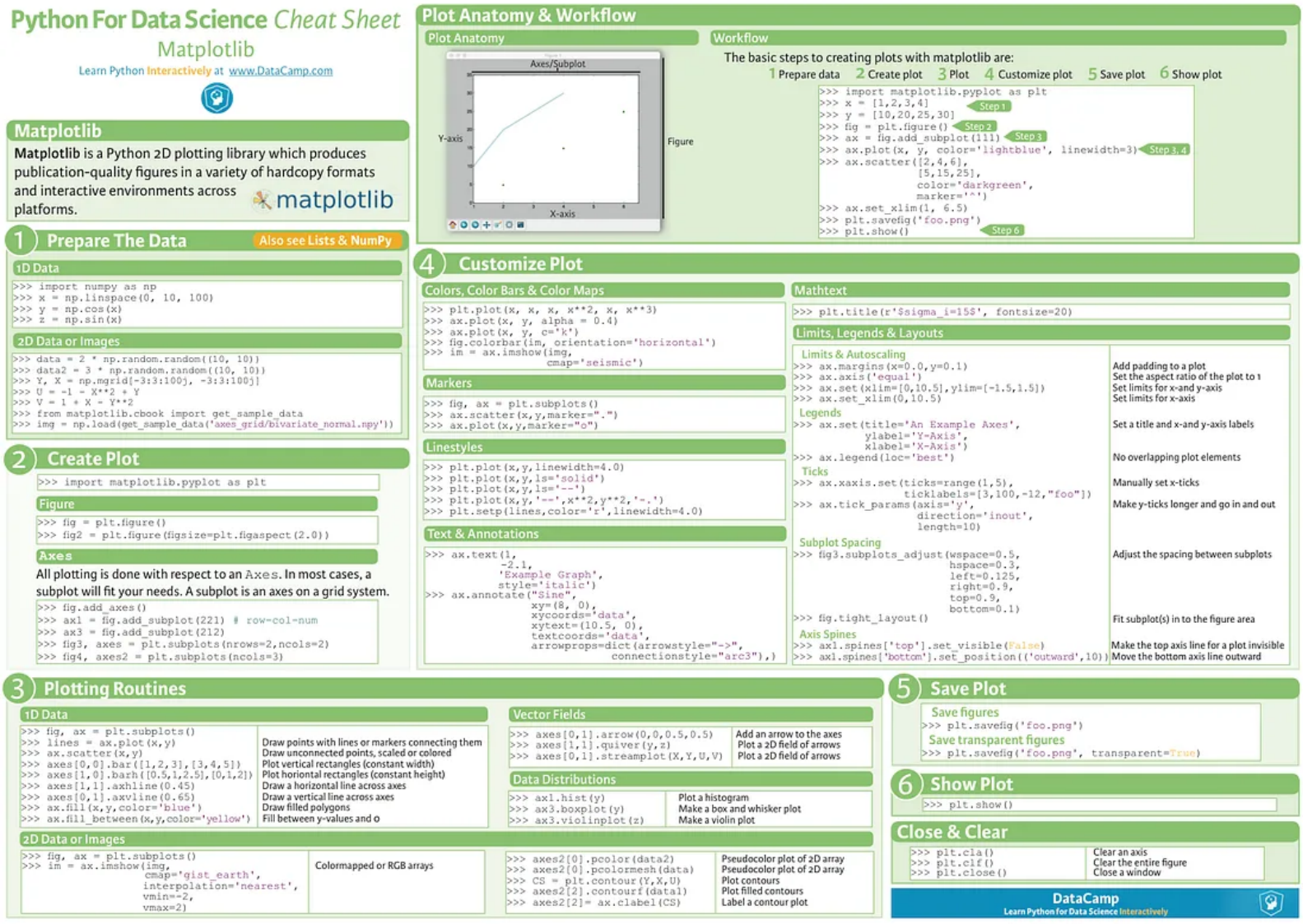
Source — https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.oundfxM

4. Scipy



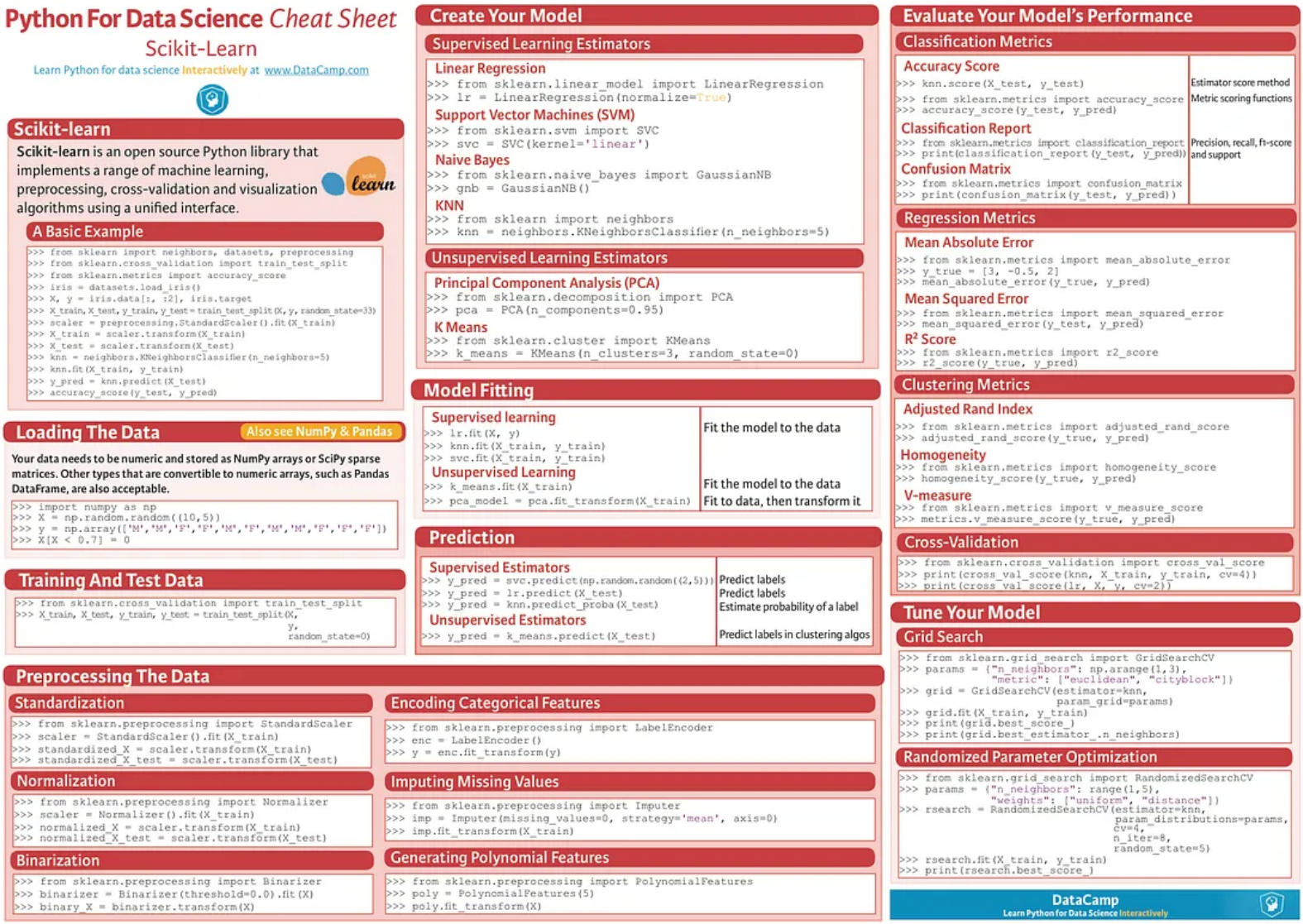
Source — https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3OI

5. Matplotlib



Source — https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet#gs.uEKySpY

6. Scikit-learn



Source — https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet

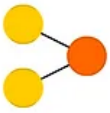
7. Neural Networks Zoo

A mostly complete chart of

Neural Networks

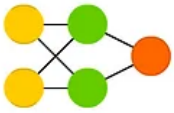
- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

Perceptron (P)

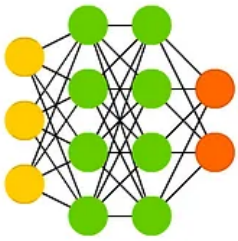


Feed Forward (FF)

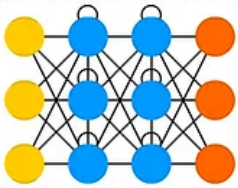
Radial Basis Network (RBF)



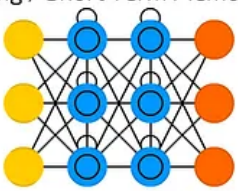
Deep Feed Forward (DFF)



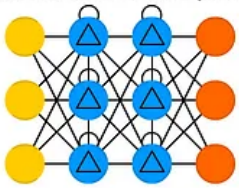
Recurrent Neural Network (RNN)



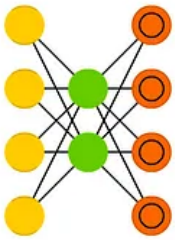
Long / Short Term Memory (LSTM)



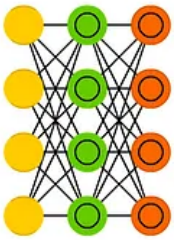
Gated Recurrent Unit (GRU)



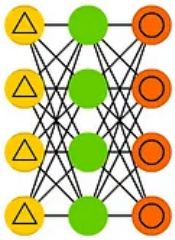
Auto Encoder (AE)



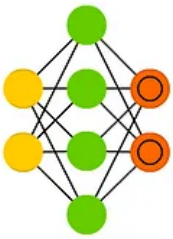
Variational AE (VAE)



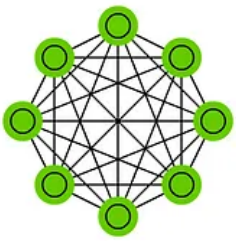
Denoising AE (DAE)



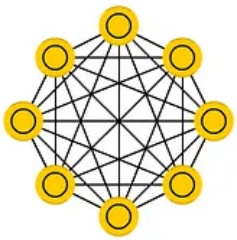
Sparse AE (SAE)



Markov Chain (MC)

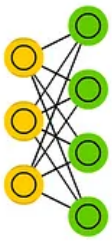


Hopfield Network (HN)

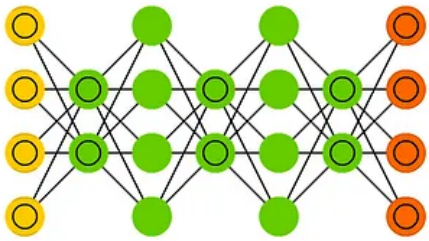


Boltzmann Machine (BM)

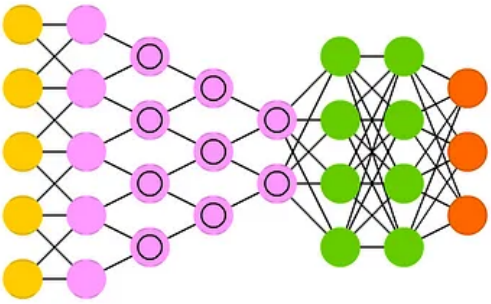
Restricted BM (RBM)



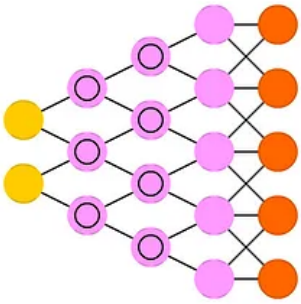
Deep Belief Network (DBN)



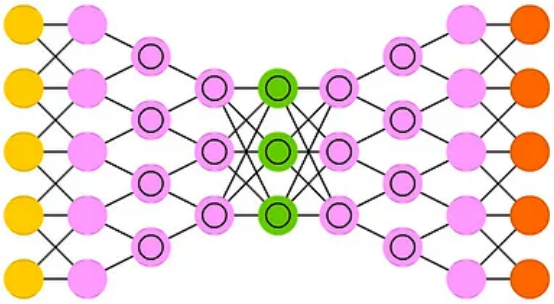
Deep Convolutional Network (DCN)



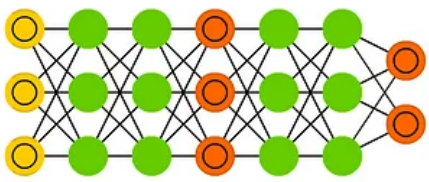
Deconvolutional Network (DN)



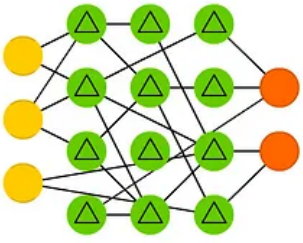
Deep Convolutional Inverse Graphics Network (DCIGN)



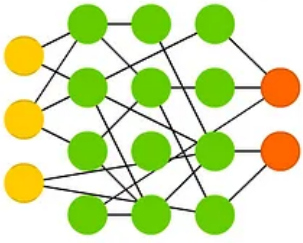
Generative Adversarial Network (GAN)



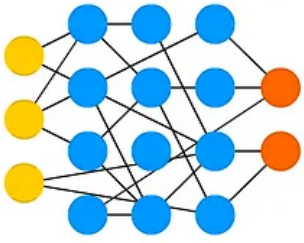
Liquid State Machine (LSM)



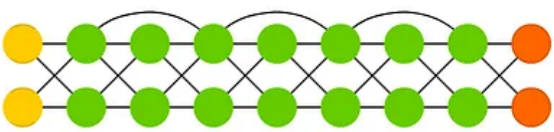
Extreme Learning Machine (ELM)



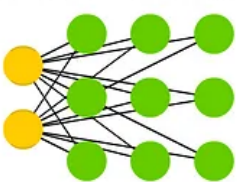
Echo State Network (ESN)



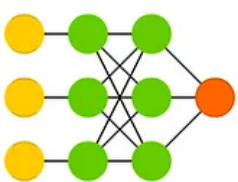
Deep Residual Network (DRN)



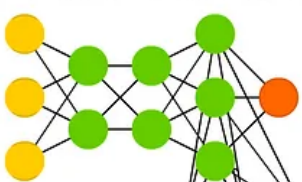
Kohonen Network (KN)



Support Vector Machine (SVM)

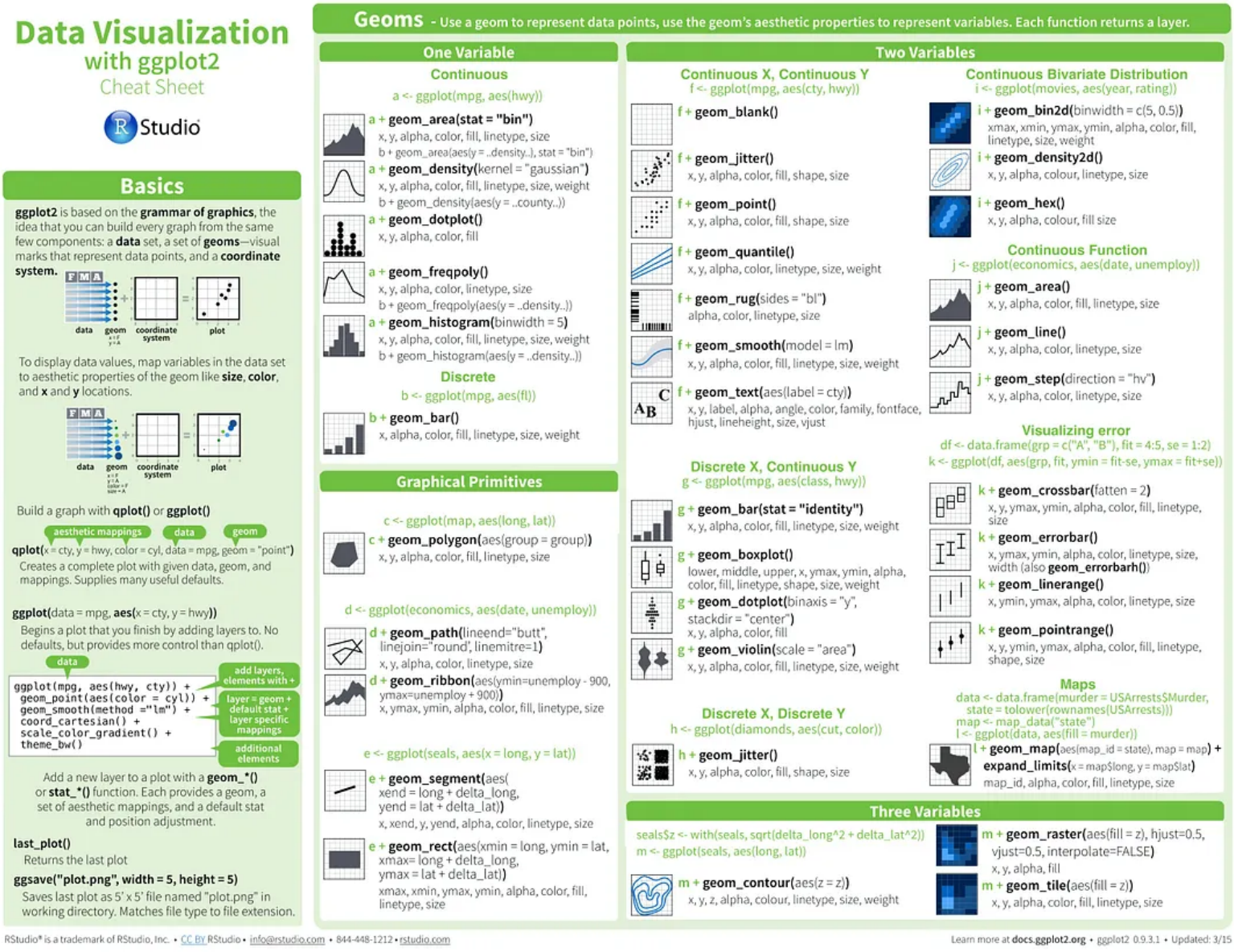


Neural Turing Machine (NTM)



Source — http://www.asimovinstitute.org/neural-network-zoo/

8. ggplot2



Data Visualization with ggplot2 Cheat Sheet

RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system.**

data · geom · coordinate system · plot

To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.

data · geom · coordinate system · plot

Build a graph with **qplot()** or **ggplot()**

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes**(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +
layer = geom + default stat + layer specific mappings
additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()
Returns the last plot

ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

f + geom_smooth(model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map**) +**
expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

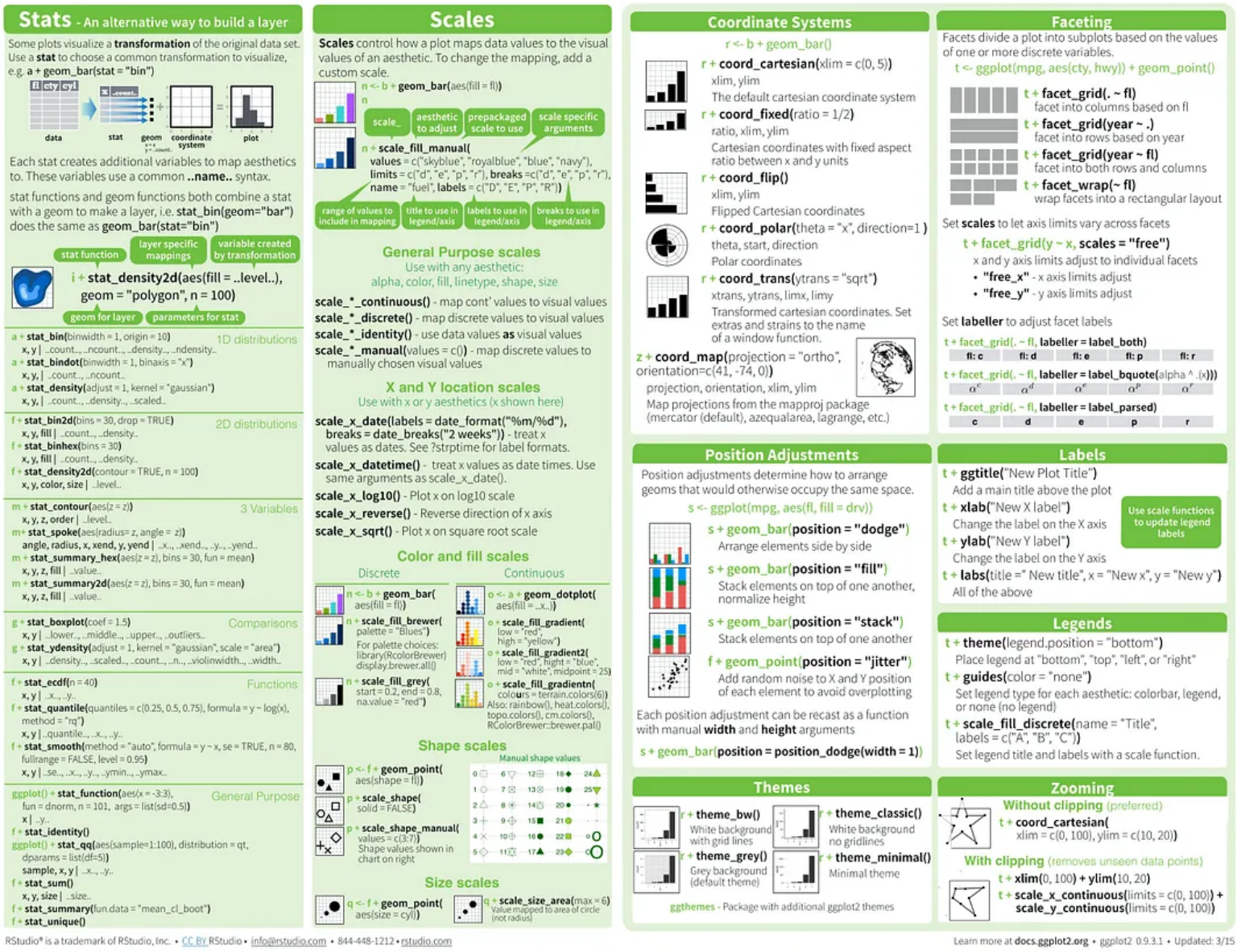
Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size



Source — https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

9. PySpark

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])).collect()
 [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
 ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x).collect()
 [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Apply a function to each RDD element

Apply a function to each RDD element and flatten the result

Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

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).collect()   Filter the RDD
 [('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
```

DataCamp
Learn Python for Data Science Interactively

Source — https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ

Python For Data Science *Cheat Sheet*

PySpark - RDD 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

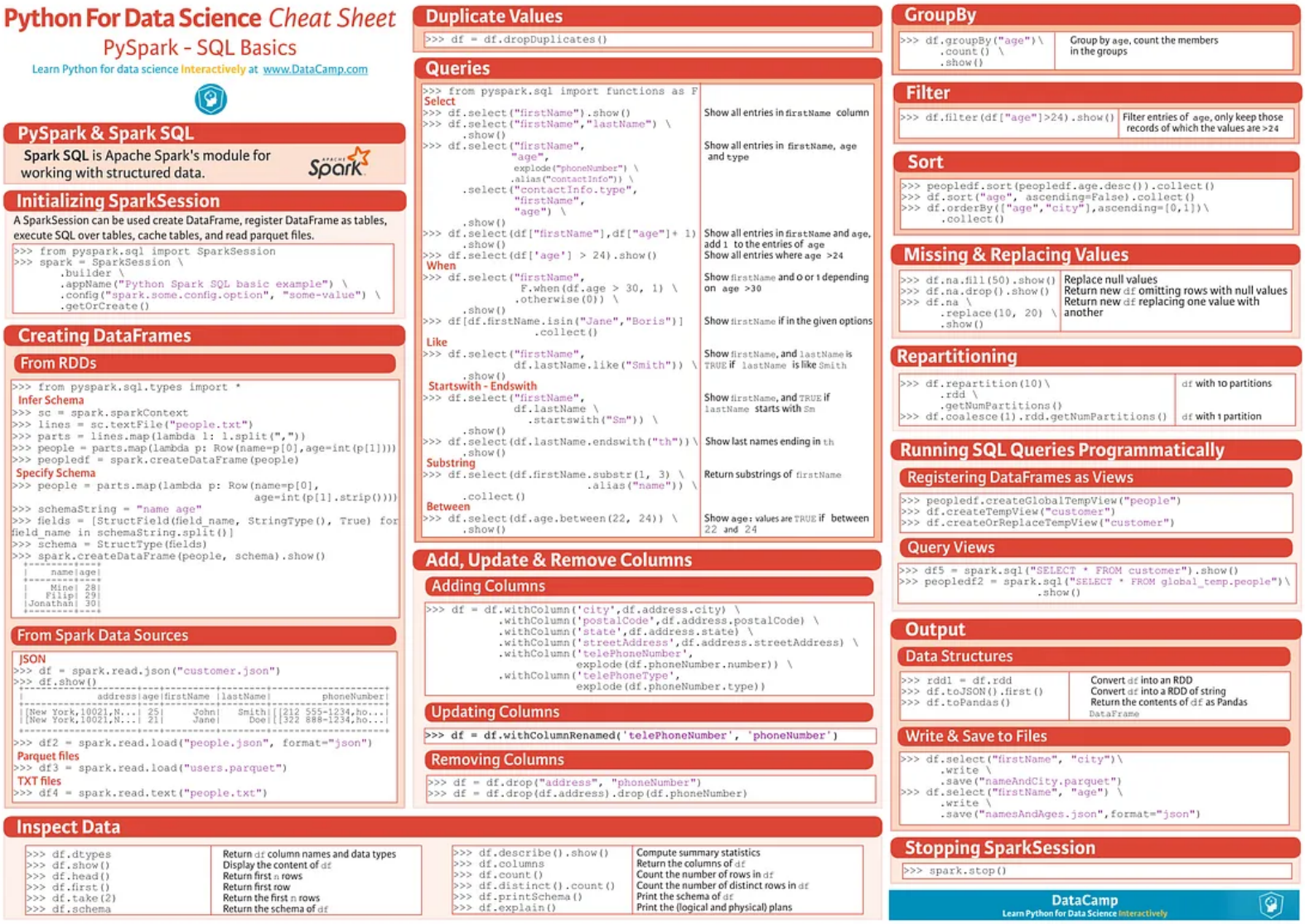
```
>>> sc.stop()
```

Execution

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

DataCamp
Learn Python for Data Science Interactively

Source — https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ



Source — https://www.datacamp.com/community/blog/pyspark-sql-cheat-sheet

10. R Studio (dplyr and tidyr)

Data Wrangling with dplyr and tidyr Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::tbl_df(iris)

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

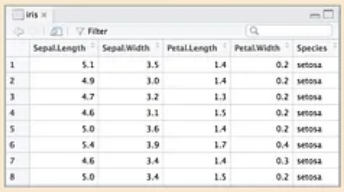
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::glimpse(iris)

Information dense summary of tbl data.

utils::View(iris)

View data set in spreadsheet-like display (note capital V).



dplyr::%>%

Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
x %>% f(y)  is the same as  f(x, y)
y %>% f(x, .., z)  is the same as  f(x, y, z )
```

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

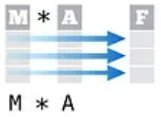
Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Reshaping Data - Change the layout of a data set

tidyr::gather(cases, "year", "n", 2:4)

Gather columns into rows.

tidyr::spread(pollution, size, amount)

Spread rows into columns.

tidyr::separate(storms, date, c("y", "m", "d"))

Separate one column into several.

tidyr::unite(data, col, ..., sep)

Unite several columns into one.

dplyr::data_frame(a = 1:3, b = 4:6)

Combine vectors into data frame (optimized).

dplyr::arrange(mtcars, mpg)

Order rows by values of a column (low to high).

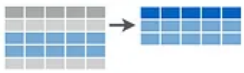
dplyr::arrange(mtcars, desc(mpg))

Order rows by values of a column (high to low).

dplyr::rename(tb, y = year)

Rename the columns of a data frame.

Subset Observations (Rows)



dplyr::filter(iris, Sepal.Length > 7)

Extract rows that meet logical criteria.

dplyr::distinct(iris)

Remove duplicate rows.

dplyr::sample_frac(iris, 0.5, replace = TRUE)

Randomly select fraction of rows.

dplyr::sample_n(iris, 10, replace = TRUE)

Randomly select n rows.

dplyr::slice(iris, 10:15)

Select rows by position.

dplyr::top_n(storms, 2, date)

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

dplyr::select(iris, Sepal.Width, Petal.Length, Species)

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

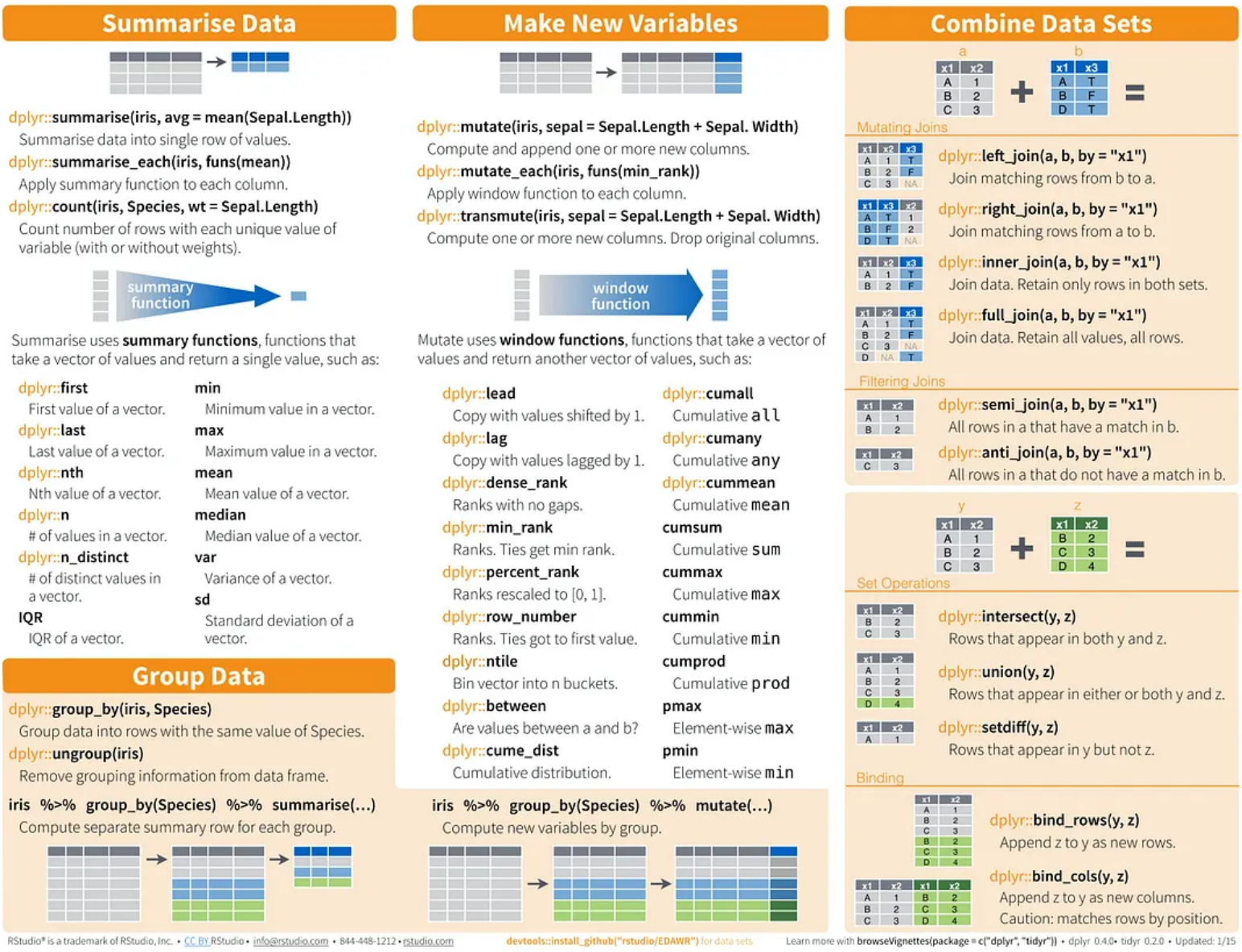
select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

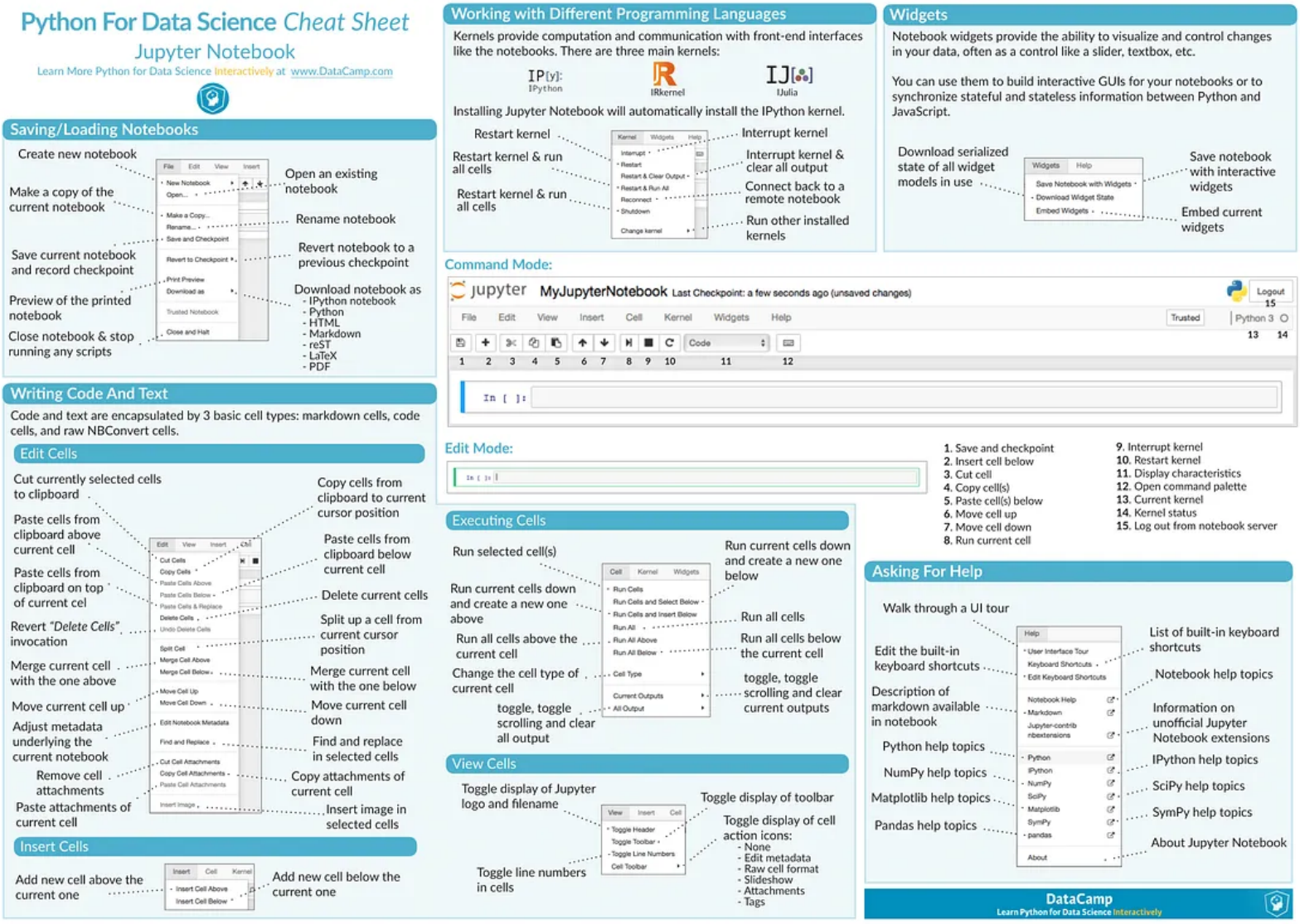
devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0 • tidyr 0.2.0 • Updated: 1/15



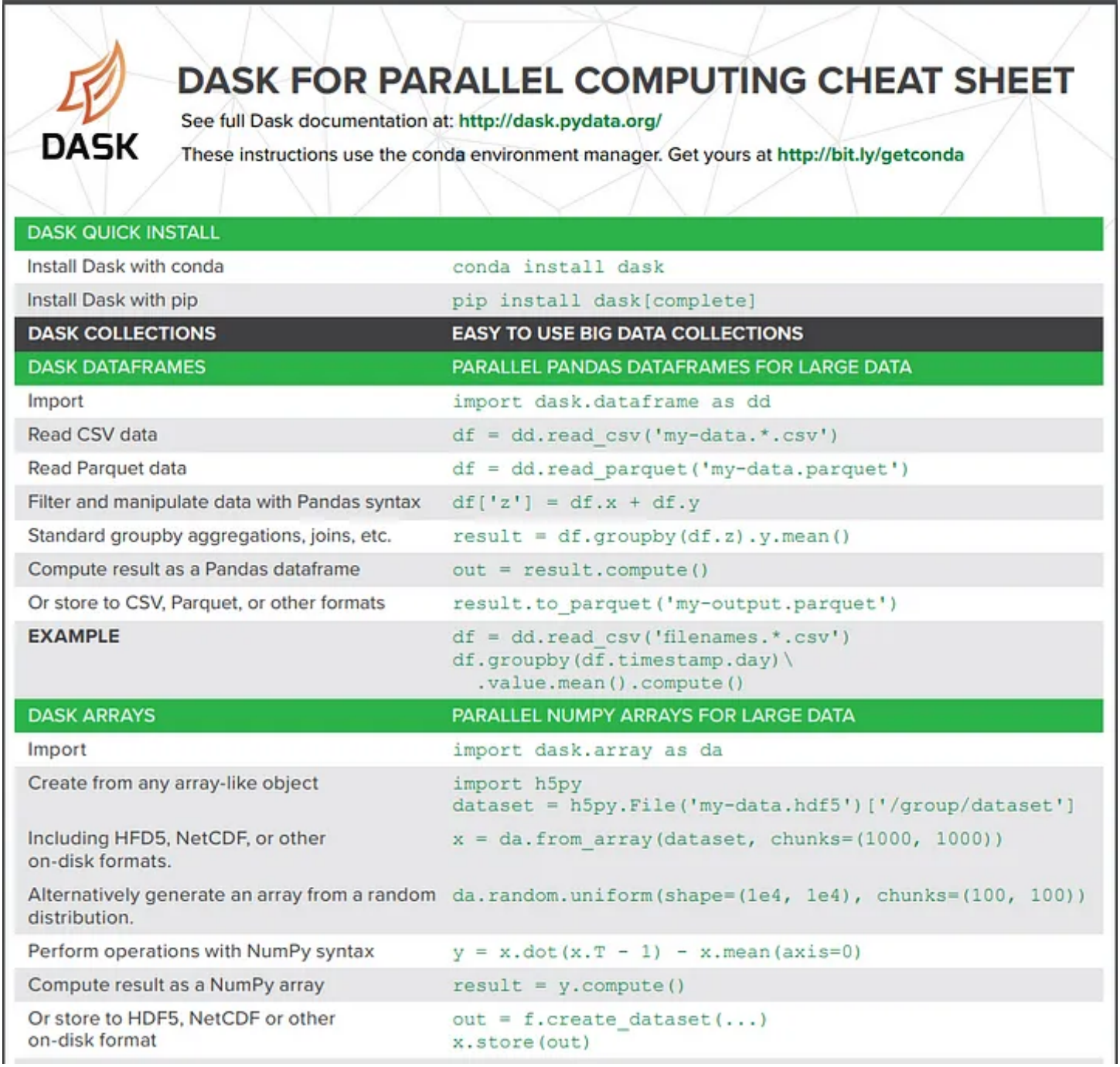
Source — https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

11. Jupyter Notebook



Source — https://www.datacamp.com/community/blog/jupyter-notebook-cheat-sheet

12. Dask



Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

EXAMPLE	`with h5py.File('my-data.hdf5') as f:`
	`  x = da.from_array(f['/path'], chunks=(1000, 1000))`
	`  x -= x.mean(axis=0)`
	`  out = f.create_dataset(...)`
	`  x.store(out)`

DASK BAGS	**PARELLEL LISTS FOR UNSTRUCTURED DATA**
Import	`import dask.bag as db`
Create Dask Bag from a sequence	`b = db.from_sequence(seq, npartitions)`
Or read from text formats	`b = db.read_text('my-data.*.json')`

Map and filter results	`import json`
	`records = b.map(json.loads)`
	`  .filter(lambda d: d["name"] == "Alice")`

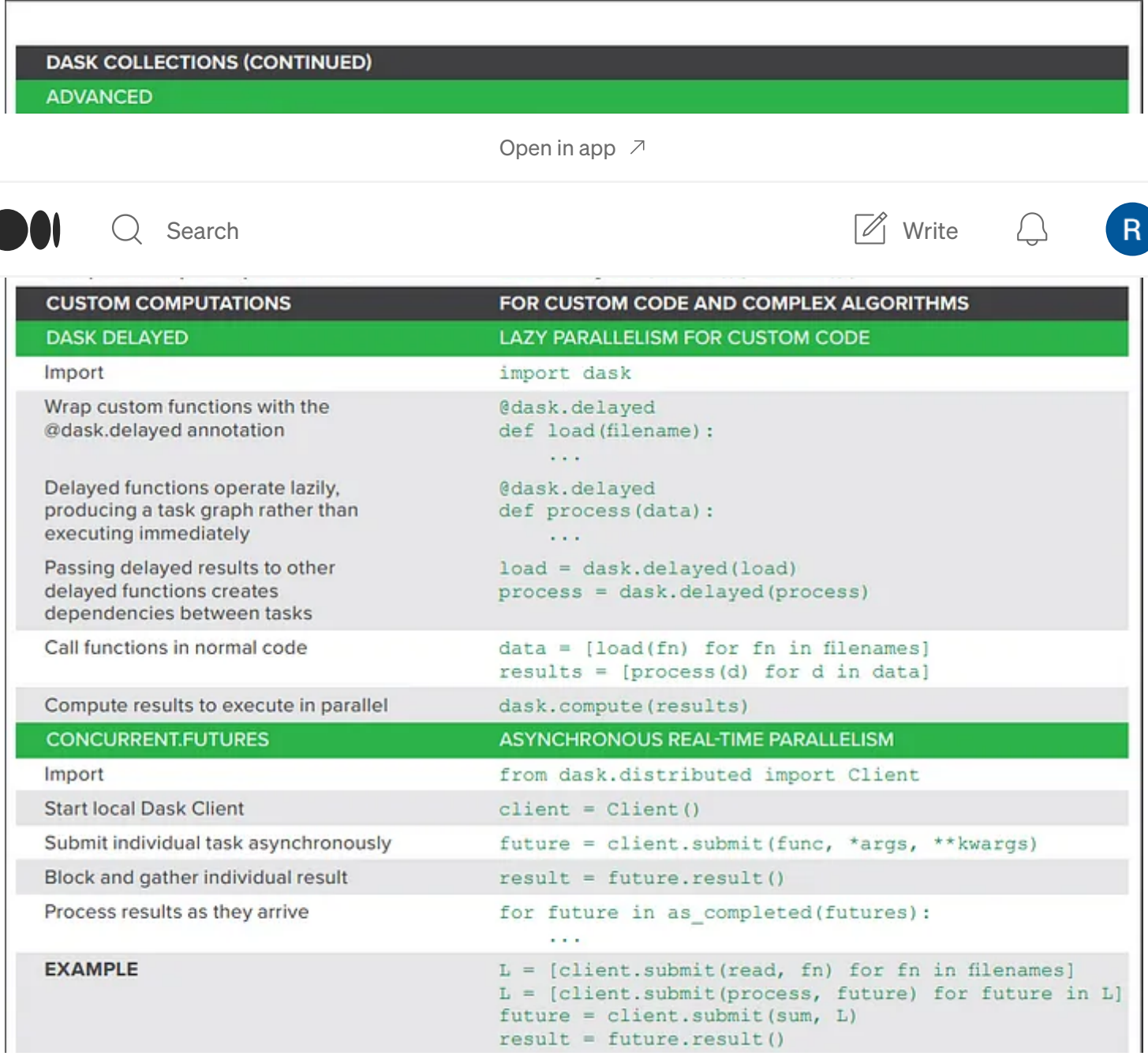
Compute aggregations like mean, count, sum	`records.pluck('key-name').mean().compute()`
Or store results back to text formats	`records.to_textfiles('output.*.json')`

EXAMPLE	`db.read_text('s3://bucket/my-data.*.json')`
	`  .map(json.loads)`
	`  .filter(lambda d: d["name"] == "Alice")`
	`  .to_textfiles('s3://bucket/output.*.json')`

ANACONDA.

CONTINUED ON BACK →

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf



DASK COLLECTIONS (CONTINUED)	
ADVANCED	
CUSTOM COMPUTATIONS	**FOR CUSTOM CODE AND COMPLEX ALGORITHMS**
DASK DELAYED	LAZY PARALLELISM FOR CUSTOM CODE
Import	`import dask`

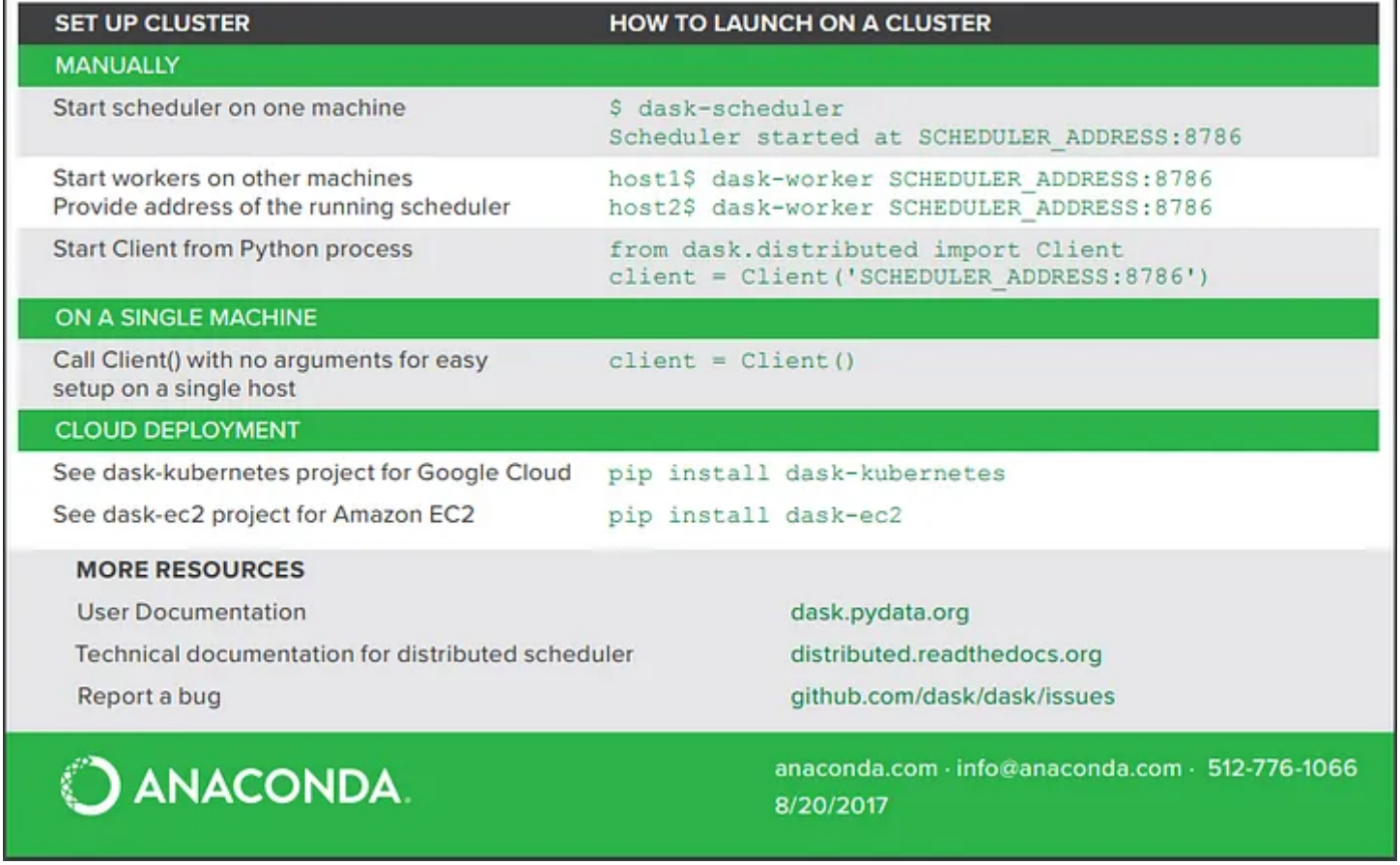
Wrap custom functions with the @dask.delayed annotation	`@dask.delayed`
	`def load(filename):`
	`  ...`

Delayed functions operate lazily, producing a task graph rather than executing immediately	`@dask.delayed`
	`def process(data):`
	`  ...`

Passing delayed results to other delayed functions creates dependencies between tasks	`load = dask.delayed(load)` `process = dask.delayed(process)`
Call functions in normal code	`data = [load(fn) for fn in filenames]` `results = [process(d) for d in data]`
Compute results to execute in parallel	`dask.compute(results)`
CONCURRENT.FUTURES	ASYNCHRONOUS REAL-TIME PARALLELISM
Import	`from dask.distributed import Client`
Start local Dask Client	`client = Client()`
Submit individual task asynchronously	`future = client.submit(func, *args, **kwargs)`
Block and gather individual result	`result = future.result()`
Process results as they arrive	`for future in as_completed(futures):` `  ...`

EXAMPLE	`L = [client.submit(read, fn) for fn in filenames]`
	`L = [client.submit(process, future) for future in L]`
	`future = client.submit(sum, L)`
	`result = future.result()`

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf



SET UP CLUSTER	HOW TO LAUNCH ON A CLUSTER
MANUALLY	
Start scheduler on one machine	`$ dask-scheduler` `Scheduler started at SCHEDULER_ADDRESS:8786`
Start workers on other machines Provide address of the running scheduler	`host1$ dask-worker SCHEDULER_ADDRESS:8786` `host2$ dask-worker SCHEDULER_ADDRESS:8786`
Start Client from Python process	`from dask.distributed import Client` `client = Client('SCHEDULER_ADDRESS:8786')`
ON A SINGLE MACHINE	
Call Client() with no arguments for easy setup on a single host	`client = Client()`
CLOUD DEPLOYMENT	
See dask-kubernetes project for Google Cloud	`pip install dask-kubernetes`
See dask-ec2 project for Amazon EC2	`pip install dask-ec2`

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

> *Thank you for reading.*
>
> *If you want to get into contact, you can reach out to me at* ***ahikailash1@gmail.com***

About Me:

I am a Co-Founder of MateLabs, where we have built Mateverse, an ML Platform which enables everyone to easily build and train Machine Learning Models, without writing a single line of code.

> ***Note****: Recently, I published a book on GANs titled "Generative Adversarial Networks Projects", in which I covered most of the widely popular GAN architectures and their implementations. DCGAN, StackGAN, CycleGAN, Pix2pix, Age-cGAN, and 3D-GAN have been covered in details at the implementation level. Each architecture has a chapter dedicated to it. I have explained these networks in a very simple and descriptive language using Keras framework with Tensorflow*

backend. If you are working on GANs or planning to use GANs, give it a read and share your valuable feedback with me at ***ahikailash1@gmail.com***

Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow...

Explore various Generative Adversarial Network architectures using the Python ecosystem Key Features Use different...

www.amazon.com

You can grab a copy of the book from http://www.amazon.com/Generative-Adversarial-Networks-Projects-next-generation/dp/1789136679https://www.amazon.in/Generative-Adversarial-Networks-Projects-next-generation/dp/1789136679?fbclid=IwAR0X2pDk4CTxn5GqWmBbKIgiB38WmFX-sqCpBNI8k9Z8I-KCQ7VWRpJXm7I https://www.packtpub.com/big-data-and-business-intelligence/generative-adversarial-networks-projects?fbclid=IwAR2OtU21faMFPM4suH_HJmy_DRQxOVwJZB0kz3ZiSbFb_MW7INYCqqV7U0c

Machine Learning | Artificial Intelligence | Deep Learning | Technology | Computer Science

Written by Kailash Ahirwar

Follow

5.8K Followers · Writer for Startups & Venture Capital

Decentralizing Artificial Intelligence | Co-founder - Raven Protocol | Mate Labs | Author - Generative Adversarial Networks Projects

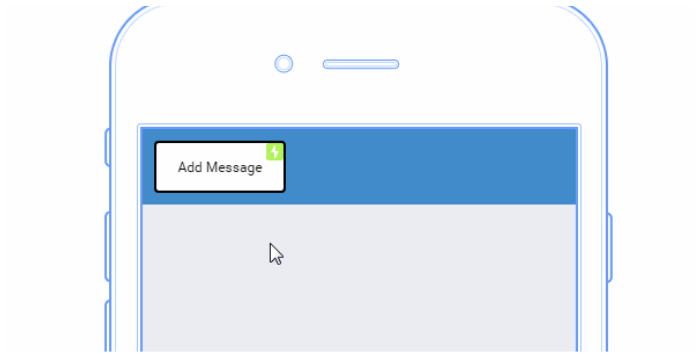
More from Kailash Ahirwar and Startups & Venture Capital

Kailash Ahirwar

A Very Short Introduction to Diffusion Models

Artificial Intelligence is constantly evolving to solve hard and complex problems. Image...

4 min read · Sep 26



Sophie Paxton in Startups & Venture Capital

Your UI isn't a Disney Movie

My previous short article about gratuitous animation really struck a chord with people....

5 min read · Oct 6, 2015

129 

Charles R Poliquin in Startups & Venture Capital

10 Household Items That Are Dropping Your Testosterone Level...

We live in a toxic world, and this toxicity affects androgen levels at an epidemic level

7 min read · Jan 19, 2018

879 1

4.5K 113

Kailash Ahirwar

A Very Short Introduction to Inception Score(IS)

Generative Adversarial Networks or GANs for short were successful in generating high-...

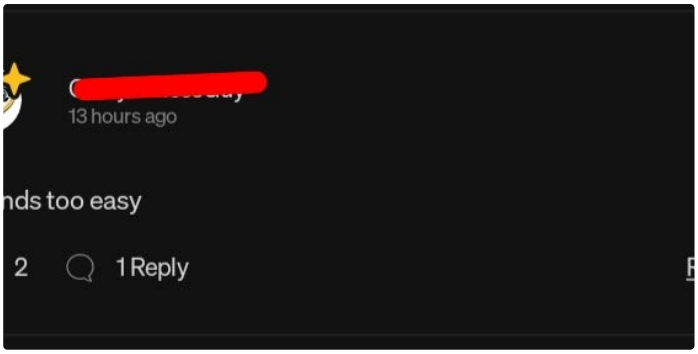
4 min read · Feb 24, 2021

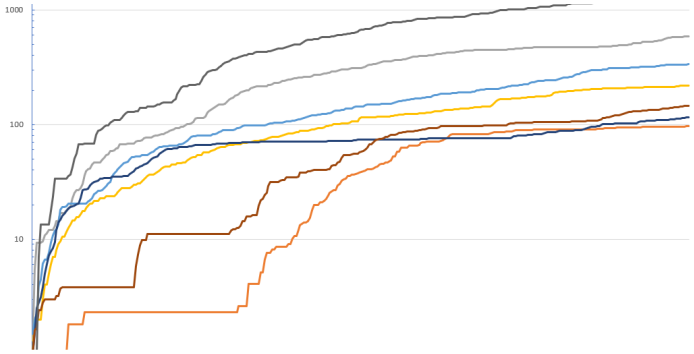
85

See all from Kailash Ahirwar

See all from Startups & Venture Capital

Recommended from Medium





Paul Rose

I Found A Very Profitable AI Side Hustle

And it's perfect for beginners

6 min read · Oct 19

12.7K 222

Pau Blasco i Roca in Towards Data Science

My Life Stats: I Tracked My Habits for a Year, and This Is What I...

I measured the time I spent on my daily activities (studying, doing sports, socializing...

12 min read · Nov 21

3.1K 57

Lists

Predictive Modeling w/ Python

20 stories · 659 saves

AI Regulation

6 stories · 208 saves

Natural Language Processing

924 stories · 441 saves

Practical Guides to Machine Learning

10 stories · 739 saves

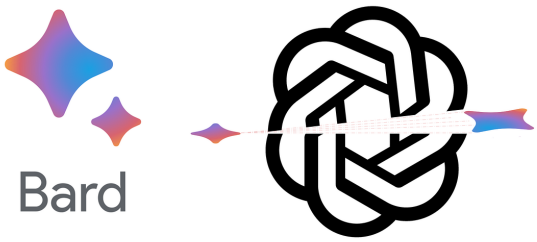
Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21

12.4K 238



AL Anany

The ChatGPT Hype Is Over—Now Watch How Google Will Kill...

It never happens instantly. The business game is longer than you know.

· 6 min read · Sep 1

20K 615

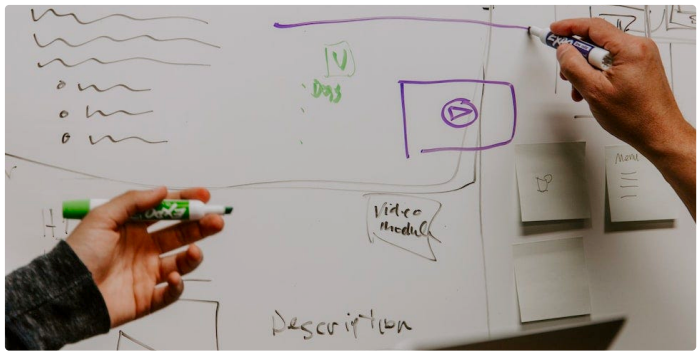
Data Scian by Imad Adrees

Best Portfolio Projects for Data Science

“How can I showcase my data skills to the world?” you may be asking. Fear not, for the...

5 min read · Sep 19

611 5



Carlos Arguelles

My favorite coding question to give candidates

A coding question, from the viewpoint of an Google/Amazon/Microsoft interviewer

11 min read · Nov 12

4.5K 49

See more recommendations