

## FavioVazquez / ds-cheatsheets Public

List of Data Science Cheatsheets to rule the world

MIT license

11.2k stars 3.3k forks

[Star](#)

[Watch](#) ▾

[Code](#) [Issues](#) 3 [Pull requests](#) 3 [Actions](#) [Projects](#) [Security](#) [Insights](#)

master ▾

...



FavioVazquez ...

on Oct 31, 2019

[View code](#)

# Data Science Cheatsheets

List of Data Science Cheatsheets to rule the world.

## Table of Contents

- [Business Science](#)
  - [Business Science Problem Framework](#)
  - [Data Science with Python Workflow](#)
  - [Data Science with R Workflow](#)
- [Python](#)
  - [Python Basics](#)
  - [Pandas Basics](#)
  - [Pandas](#)
  - [Importing Data](#)
  - [Jupyter](#)
  - [Numpy Basics](#)
  - [Beginners Python Cheat Sheet](#)

- [Intermediate Python](#)
- [Python REGEX](#)
- [Python 3 Memento](#)
- [R](#)
  - [Tidiverse](#)
  - [data.table](#)
  - [xts](#)
  - [Base R](#)
  - [Data Import with readr](#)
  - [Data Transformation with Dplyr](#)
  - [Apply Functions with purrr](#)
  - [Data transformation with data.table](#)
  - [Dates and Times with lubridate](#)
  - [Randomizr](#)
  - [Regular Expressions](#)
  - [Work with Strings with stringr](#)
  - [Tidy Evaluation with rlang](#)
  - [Xplain](#)
  - [Sintax Comparison](#)
  - [Data and Variable Transformation with sjmisc](#)
  - [R Markdown \(PDF\)](#)
  - [Package Development with devtools](#)
- [Math and Calculus](#)
  - [Refresher Algebra and Calculus](#)
  - [Refresher Probabilities and Statistics](#)
  - [Fundamentals of Probabilities](#)
- [Big Data](#)
  - [Pyspark RDD](#)
  - [Pyspark DF](#)
  - [Dask](#)
  - [Sparklyr](#)
- [Machine Learning](#)
  - [Scitk-Learn \(PDF\)](#)
  - [Machine Learning Modelling in R](#)

- [Caret](#)
- [Estimatr](#)
- [H2O](#)
- [mlr](#)
- [Regression](#)
- [VIP Supervised Learning](#)
- [Segmentation and Clustering](#)
- [VIP Unsupervised Learning](#)
- [VIP Machine Learning Tips and Tricks](#)
- [Choosing the right model](#)
- [Deep Learning - Neural Nets](#)
  - [Keras RStudio - Keras](#)
  - [Deep Learning Basics](#)
  - [Convolutional Neural Networks](#)
  - [Recurrent Neural Networks](#)
  - [Tips and Tricks](#)
- [SQL](#)
  - [SQL cheatsheet by sqltutorial \(PDF\)](#)
  - [SQL cheatsheet by Rebel Labs](#)
- [Data Visualization](#)
  - [Matplotlib](#)
  - [Seaborn](#)
  - [Bokeh](#)
  - [Comprehensive Guide to Data Visualization in Python](#)
  - [Ggplot2](#)
  - [Leaflet](#)
  - [Cartography](#)
  - [Comprehensive Guide to Data Visualization in R](#)
  - [Simple Features sf](#)
  - [survminer](#)

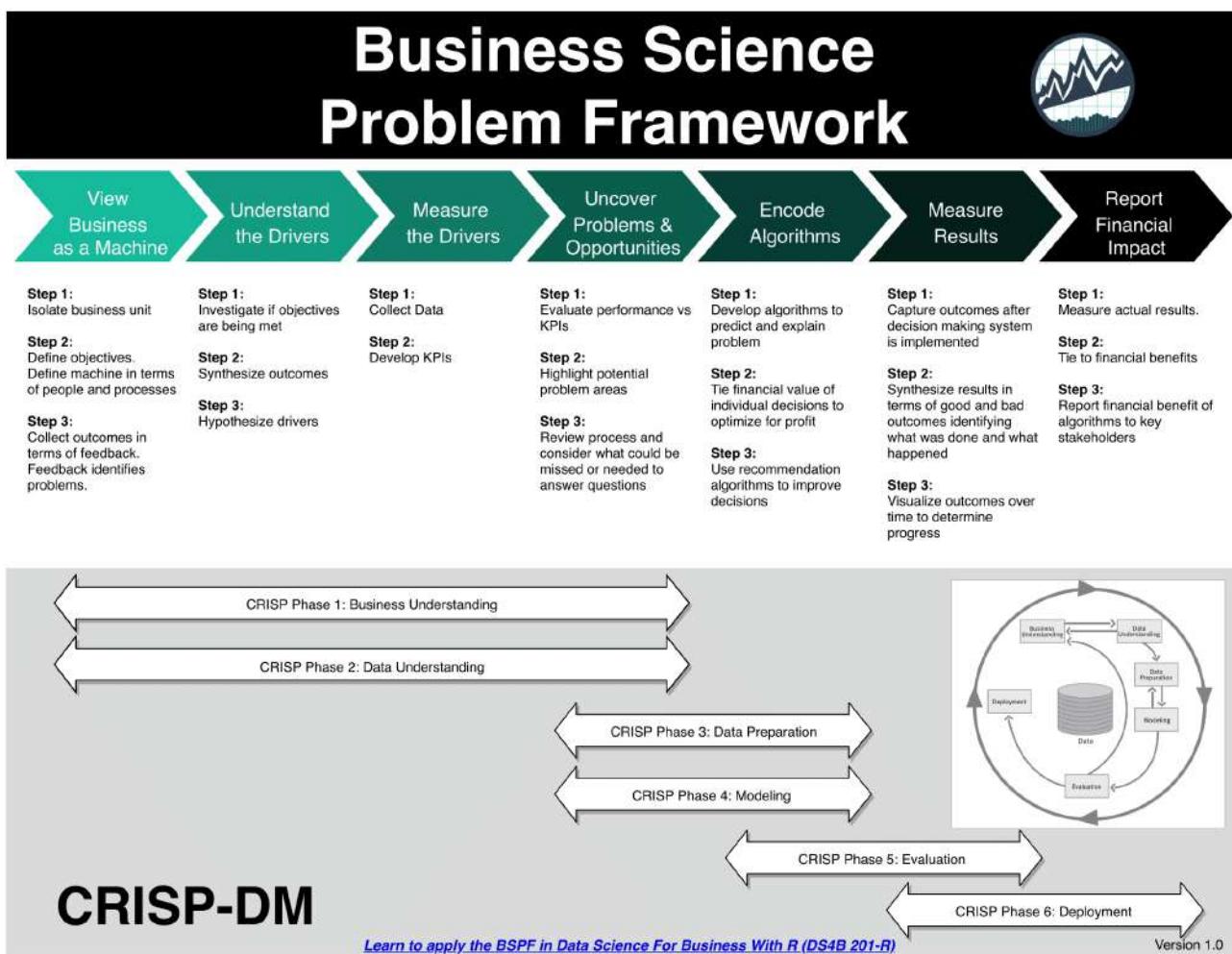
### ☰ README.md

- 
- [Shiny](#)
  - [Vim](#)
  - [Emacs](#)

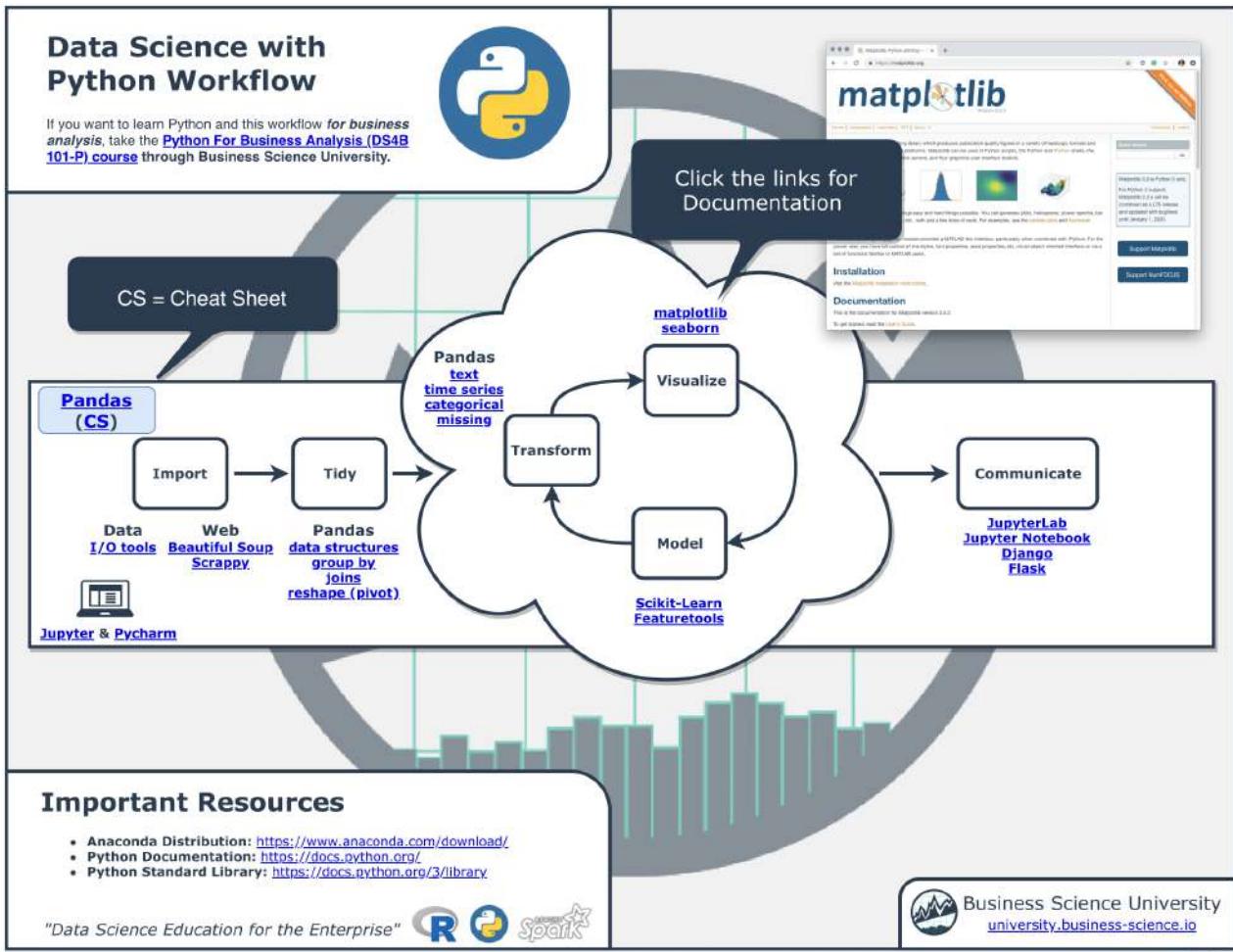
- o Git - Atlassian's Cheatsheet
- o DVC
- o BASH
- Data Science Cheatsheet Takeaway

## Business Science

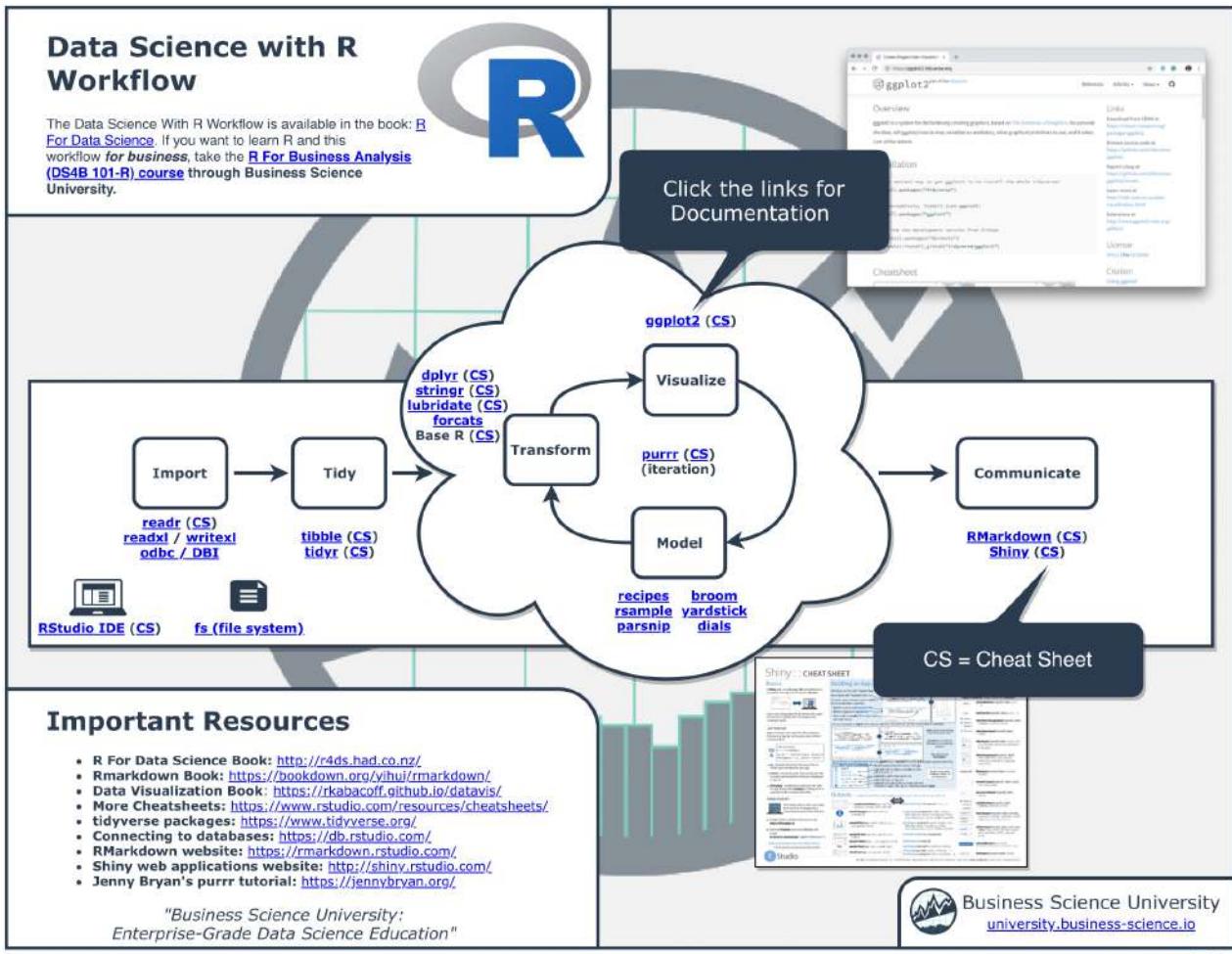
### Business Science Problem Framework (PDF)



### Data Science with Python Workflow (PDF)



## Data Science with R Workflow (PDF)



# Python

## Datacamp

## • Python Basics (PDF)

**Python For Data Science Cheat Sheet**

**Python Basics**

Learn More Python for Data Science interactively at [www.datacamp.com](http://www.datacamp.com)

**Variables and Data Types**

Variable Assignment		
>>> x=5		
>>> x	5	

Calculations With Variables		
>>> x+2	Sum of two variables	
>>> x-2	Subtraction of two variables	
>>> x*2	Multiplication of two variables	
10		
>>> x**2	Exponentiation of a variable	
25		
>>> x%2	Remainder of a variable	
1		
>>> x/float(2)	Division of a variable	
2.5		

Types and Type Conversion		
str()	'5', '3.45!', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

**Asking For Help**

```
>>> help(str)
```

**Strings**

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations		
>>> my_string + 2	"thisStringIsAwesomethisStringIsAwesome"	
>>> my_string + 'Init'	"thisStringIsAwesomeInit"	
>>> 'm' in my_string	True	

**List Operations**

```
>>> my_list + my_list
[my, 'list', 'list', 'nice', 'my', 'list', 'list', 'nice']
>>> my_list * 2
[my, 'list', 'list', 'nice', 'my', 'list', 'list', 'nice']
>>> my_list2 > 4
```

**List Methods**

```
>>> my_list.index('a')
>>> my_list.count('a')
>>> my_list.append('!')
>>> my_list.remove('!')
>>> delmy_list[0::1]
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

**String Operations**

```
>>> my_string[3]
>>> my_string[4:9]
```

**String Methods**

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

**Also see NumPy Arrays**

**Libraries**

Import libraries	
>>> import numpy	pandas
>>> import numpy as np	Data analysis
Selective import	Machine learning
>>> from math import pi	NumPy
	matplotlib

**Install Python**

**ANACONDA**  
Leading open data science platform powered by Python

**jupyter**  
Create and share documents with live code, visualizations, text, ...

**Numpy Arrays**

**Also see Lists**

Selecting Numpy Array Elements	
Subset	Select item at index 1
>>> my_array[1]	
Slice	Select items at index 0 and 1
>>> my_array[0:2]	
Subset 2D Numpy arrays	Select items before index 3
>>> my_2darray[:,0]	
	Copy my_list
	my_list[itemOfList]

**Numpy Array Operations**

```
>>> my_array > 3
array([False, False, False, True, False])
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

**Numpy Array Functions**

Get the dimensions of the array	
>>> my_array.shape	
>>> np.append(other_array)	Append items to an array
>>> np.insert(my_array, 1, 5)	Insert items in an array
>>> np.delete(my_array, [1])	Delete items in an array
>>> np.mean(my_array)	Mean of the array
>>> np.median(my_array)	Median of the array
>>> my_array.corrcoef()	Correlation coefficient
>>> np.std(my_array)	Standard deviation

**DataCamp**  
Learn Python for Data Science interactively

## • Pandas Basics (PDF)

**Python For Data Science Cheat Sheet**

**Pandas Basics**

Learn Python for Data Science interactively at [www.DataCamp.com](http://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.

**Series**

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

Index	1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8

```
>>> s = pd.Series([3, -5, 7, 0], index=[1, 2, 3, 4, 5, 6, 7, 8])
```

**DataFrame**

Index	0	1	2	3	4	5	6	7
0	Belgium	Brussels	11000000					
1	India	New Delhi	1303171038					
2	Brazil	Brasilia	207847528					

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   'Population': [111190846, 1303171038, 207847528]}
>>> df = pd.DataFrame(data,
   columns=['Country', 'Capital', 'Population'])
```

**I/O**

**Read and Write to CSV**

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.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
>>> xlxs = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlxs, '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)
```

**Asking For Help**

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

**Selection**

**Also see NumPy Arrays**

Getting	
>>> df['b']	Get one element
>>> df[1:]	Get subset of a DataFrame

**Selecting, Boolean Indexing & Setting**

By Position	
>>> df.loc[[0], [0]]	Select single value by row & column
>>> df.iat[0, 0]	
'Belgium'	

By Label	
>>> df.loc[0, ['Country']]	Select single value by row & column labels
'Belgium'	
>>> df.iat[0, ['Country']]	
'Belgium'	

By Label/Position	
>>> df.ix[2]	Select single row of subset of rows
Country Brazil Capital Brasilia Population 207847528	
>>> df.ix[[0], ['Capital']]	Select a single column of subset of columns
0 Brussels 1 New Delhi 2 Brasilia	
>>> df.ix[1, 'Capital']	Select rows and columns
'New Delhi'	

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

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

**Dropping**

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

**Sort & Rank**

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

**Retrieving Series/DataFrame Information**

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

Summary	
>>> df.sum()	Sum of values
>>> df.sumsum()	Cumulative sum of values
>>> df.max()	Maximum/Minimum values
>>> df.min()	Minimum/Maximum index value
>>> df.describe()	Summary statistics
>>> df.mean()	Mean of values
>>> df.median()	Median of values

**Applying Functions**

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

**Data Alignment**

**Internal Data Alignment**

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

```
>>> a3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> a3
a    7.0
c   -2.0
d    3.0
```

**Arithmetic Operations with Fill Methods**

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

```
>>> a3.add(b3, fill_value=0)
a    10.0
b   -5.0
c    5.0
d    7.0
>>> a3.sub(b3, fill_value=2)
a    8.0
b   -7.0
c    3.0
d    5.0
>>> a3.div(b3, fill_value=4)
a    2.5
b   -0.5
c    1.25
d    1.75
>>> a3.mul(b3, fill_value=3)
a    30.0
b   -15.0
c    15.0
d    21.0
```

**DataCamp**  
Learn Python for Data Science interactively

## • Pandas (PDF)

### Python For Data Science Cheat Sheet

#### Pandas

Learn Python for Data Science interactively at [www.DataCamp.com](http://www.DataCamp.com)

#### Reshaping Data

##### Pivot

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

Date	Type	Value
2016-01-01	a	11.452
2016-01-01	b	45.051
2016-01-01	c	20.784
2016-01-01	a	11.452
2016-01-01	b	45.908
2016-01-01	c	1.303
2016-01-01	a	26.784

##### Pivot Table

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

Date	Type	a	b	c
2016-01-01	11.452	45.051	20.784	
2016-01-02	11.452	45.908	1.303	
2016-01-03	11.452	45.908	1.303	

##### Stack / Unstack

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

	0	1
0	0.255482	0.255482
1	0.255482	0.369949
2	0.180713	0.273703
3	0.435322	0.429401

Unstacked → Stacked

##### Melt

```
>>> pd.melt(df3,
            id_vars=['Date'],
            value_vars=['Type', 'Value'],
            value_name='Observations')
```

Date	Type	Value	Observations
2016-01-01	a	11.452	11.452
2016-01-01	b	45.051	45.051
2016-01-01	c	20.784	20.784
2016-01-02	a	11.452	11.452
2016-01-02	b	45.908	45.908
2016-01-02	c	1.303	1.303
2016-01-03	a	26.784	26.784

##### Iteration

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

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

#### Advanced Indexing

#### Also see NumPy Arrays

```
>>> df3.loc[:, (df3>1).any()]
>>> df3.loc[:, (df3>1).all()]
>>> df3.loc[:, (df3>1).all(1)]
>>> df3.loc[:, (df3.notnull().any(1)).all(1)]
Indexing With isin
>>> df3.filter(items=df3.Type.isin(['a','b']))
>>> df3.select(lambda x: not x.isin())
Where
>>> s.where(s > 0)
Query
>>> df3.query("second > first")
```

Select cols with any vals > 1  
Select cols with vals > 1  
Select cols with NaN  
Select cols without NaN  
Find same elements  
Filter on values  
Select specific elements

Subset the data  
Query DataFrame

#### Setting/Resetting Index

Set the index  
Reset the index  
Rename DataFrame

#### Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
Forward Filling
>>> df.reindex(range(4), method='ffill')
Backward Filling
>>> s3 = s.reindex(range(5), method='bfill')
```

Country	Capital	Population
0	Bogotá	7.5
1	Buenos Aires	3.3
2	México City	20.3
3	Brasília	2.3
4	Brasília	20.7

#### Multilevel Indexing

```
>>> arrays = [np.array([1,2,3,4]),
             np.array([1,2,3,4,5,6,7,8,9,10])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                       names=['First', 'Second'])
>>> df2.set_index(index, axis=1, inplace=True)
```

#### Duplicate Data

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

Return unique values  
Check duplicates  
Drop duplicates  
Check index duplicates

#### Grouping Data

```
>>> df2.groupby(by=['Date', 'Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x: sum(x)/len(x),
                            'b': np.sum})
Transformation
>>> customsum = lambda x1: (x+x2)
>>> df4.groupby(level=0).transform(customsum)
```

#### Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df.replace("", "0")
DropNaN values
Fill NaN values with a predetermined value
Replace values with others
```

#### Combining Data

#### Merge

data1	data2
x1	x2
a	11.432
b	1.303
c	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.996	20.784

## • Jupyter (PDF)

### Python For Data Science Cheat Sheet

#### Jupyter Notebook

Learn More Python for Data Science interactively at: [www.DataCamp.com](http://www.DataCamp.com)

#### Saving/Loading Notebooks

Create new notebook



Make a copy of the current notebook

Open an existing notebook

Rename notebook

Revert notebook to a previous checkpoint

Download notebook as:

- IPython notebook
- PDF
- HTML
- Markdown
- REST
- LaTeX
- PDF

Save current notebook and record checkpoint

Preview of the printed notebook

Close notebook & stop running any scripts

#### Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

#### Edit Cells

Cut currently selected cells to clipboard

Paste cells from clipboard above current cell

Paste cells from clipboard on top of current cell

Revert "Delete Cells" invocation

Merge current cell with the one above

Move current cell up

Adjust metadata underlying the current notebook

Remove cell attachments

Paste attachments of current cell

#### Insert Cells

Add new cell above the current one

Add new cell below the current one

#### Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:



Installing Jupyter Notebook will automatically install the IPython kernel.

Restart kernel

Restart kernel & run all cells

Restart kernel & run all cells

Interrupt kernel

Interrupt kernel & clear all output

Connect back to a remote notebook

Run other installed kernels

#### Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

Download serialized state of all widget models in use



Save notebook with interactive widgets  
Embed current widgets



#### Command Mode:

Jupyter MyJupyterNotebook Last Checkpoint: a few seconds ago (unsaved changes)



#### Edit Mode:

In | ]

#### Executing Cells

Run selected cells(s)

Run current cells down and create a new one above

Run all cells above the current cell

Change the cell type of current cell

toggle, toggle scrolling and clear all output

#### View Cells

Toggle display of Jupyter logo and filename

Toggle line numbers in cells

Run current cells down and create a new one below

Run all cells below the current cell

toggle, toggle scrolling and clear all output

Toggle display of toolbar

Toggle display of cell actions: None, - Edit metadata, - Raw cell format, - Slideshow, - Attachments, - Tags

1. Save and checkpoint

2. Cell below

3. Cut cell

4. Copy cell

5. Paste cell(s) below

6. Move cell up

7. Move cell down

8. Run current cell

9. Interrupt kernel

10. Restart kernel

11. Display characteristics

12. Open command palette

13. Current kernel

14. Kernel status

15. Log out from notebook server

#### Asking For Help

Walk through a UI tour

Edit the built-in keyboard shortcuts

Description of markdown available in notebook

Python help topics

NumPy help topics

Matplotlib help topics

Pandas help topics

List of built-in keyboard shortcuts

Notebook help topics

Information on unofficial Jupyter Notebook extensions

IPython help topics

SciPy help topics

Sympy help topics

About Jupyter Notebook

DataCamp

Learn Python for Data Science interactively

## • Numpy Basics (PDF)

### Python For Data Science Cheat Sheet

#### NumPy Basics

Learn Python for Data Science interactively at: [www.DataCamp.com](http://www.DataCamp.com)

#### NumPy

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

Use the following import convention:

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

#### NumPy Arrays

##### 1D array

`[1 2 3]`

axis 0

##### 2D array

`[[1, 2, 3], [4, 5, 6]]`

axis 0

axis 1

axis 2



#### Creating Arrays

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

#### Initial Placeholders

```
>>> e = np.zeros((3, 4))
>>> f = np.ones((3, 4), dtype=np.int16)
>>> g = np.linspace(0, 2, 5)
>>> h = np.eye(2)
>>> i = np.random((2, 2))
>>> j = np.empty(3, 2)
```

#### I/O

##### Saving & Loading On Disk

```
>>> np.save("my_array", a)
>>> np.savetxt("array.csv", a, delimiter=',')
>>> np.loadtxt("array.txt")
```

##### Saving & Loading Text Files

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

##### Data Types

<code>&gt;&gt;&gt; np.int64</code>	Signed 64-bit integer type
<code>&gt;&gt;&gt; np.float128</code>	Standard double-precision floating point
<code>&gt;&gt;&gt; np.complex</code>	Complex numbers represented by 128 floats
<code>&gt;&gt;&gt; np.bool</code>	Boolean type storing <code>TRUE</code> and <code>FALSE</code> values
<code>&gt;&gt;&gt; np.object</code>	Fixed-length string type
<code>&gt;&gt;&gt; np.str_(...)</code>	Fixed-length unicode type

#### Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> c.size
>>> d.dtype.name
>>> d.astype(int)
```

Array dimensions  
Length of array  
Number of array dimensions  
Number of array elements  
Name of array elements  
Name of data type  
Convert an array to a different type

#### Asking For Help

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

#### Array Mathematics

```
>>> g = a + b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
>>> a + b
>>> a - b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
>>> a * b
>>> a - b
>>> a / b
>>> a % b
>>> a ** b
>>> a + b
>>> a - b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
>>> a + b
>>> a - b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
>>> a + b
>>> a - b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
>>> a + b
>>> a - b
>>> a * b
>>> a / b
>>> a % b
>>> a ** b
```

Subtraction  
Addition  
Division  
Multiplication  
Exponentiation  
Square root  
Print series of an array  
Element-wise cosine  
Element-wise natural logarithm  
Dot product

Comparison

```
>>> a == b
>>> a != b
>>> a < b
>>> a > b
>>> a <= b
>>> a >= b
```

Element-wise comparison  
Element-wise comparison  
Element-wise comparison

#### Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.cov()
>>> np.std(b)
```

Array-wise sum  
Array-wise minimum value  
Maximum value of an array row  
Cumulative sum of the elements  
Mean  
Median  
Correlation coefficient  
Standard deviation

#### Copying Arrays

```
>>> h = a.view()
>>> h[0] = 5
>>> h = a.copy()
```

Create a view of the array with the same data  
Create a copy of the array  
Create a deep copy of the array

#### Sorting Arrays

```
>>> a.argsort()
>>> c.argsort(axis=0)
```

Sort an array  
Sort the elements of an array's axis

#### Subsetting, Slicing, Indexing

```
>>> a[0]
>>> a[1:2]
>>> a[0:2]
>>> a[1:2:2]
```

Select the element at the 2nd index  
Select the element at row 1 column 2 (equivalent to `a[1][2]`)  
Select items at index 0 and 1  
Select all items at row 0 (equivalent to `a[:,0]`).  
Same as `[0:,0]`

#### Subsetting, Slicing, Indexing

##### Subsetting

`a[0:1:2]`

`a[0:1:2]`

`a[1:2:2]`

- [Beginners Python Cheat Sheet \(Long PDF\)](#)

## Beginner's Python Cheat Sheet

**Variables and Strings**  
Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

**Lists**  
A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

**Lists (cont.)**

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

**Tuples**  
Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements  
*If statements are used to test for particular conditions and respond appropriately.*

equals	x == 42
not equal	x != 42
greater than	x > 42
or equal to	x >= 42
less than	x < 42
or equal to	x <= 42

Conditional tests

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

**Dictionaries**  
Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves a number')
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

**User input**  
Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

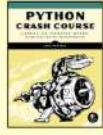
Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

**Python Crash Course**  
Covers Python 3 and Python 2

[nostarchpress.com/pythoncrashcourse](http://nostarchpress.com/pythoncrashcourse)



## Dataquest

- [Intermediate Python \(PDF\)](#)

# Data Science Cheat Sheet

## Python - Intermediate

### KEY BASICS, PRINTING AND GETTING HELP

This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet

**s** - A Python string variable  
**i** - A Python integer variable  
**f** - A Python float variable

**l** - A Python list variable  
**d** - A Python dictionary variable

#### LISTS

**l.pop(3)** - Returns the fourth item from **l** and deletes it from the list  
**l.remove(x)** - Removes the first item in **l** that is equal to **x**  
**l.reverse()** - Reverses the order of the items in **l**  
**l[1::2]** - Returns every second item from **l**, commencing from the 1st item  
**l[-5:]** - Returns the last 5 items from **l** specific axis

#### STRINGS

**s.lower()** - Returns a lowercase version of **s**  
**s.title()** - Returns **s** with the first letter of every word capitalized  
**"23".zfill(4)** - Returns "0023" by left-filling the string with 0's to make it's length 4.  
**s.splitlines()** - Returns a list by splitting the string on any newline characters.  
*Python strings share some common methods with lists*  
**s[:5]** - Returns the first 5 characters of **s**  
**"fri" + "end"** - Returns "friend"  
**"end" in s** - Returns True if the substring "end" is found in **s**

#### RANGE

Range objects are useful for creating sequences of integers for looping.  
**range(5)** - Returns a sequence from 0 to 4  
**range(2000,2018)** - Returns a sequence from 2000 to 2017  
**range(0,11,2)** - Returns a sequence from 0 to 10, with each item incrementing by 2  
**range(0,-10,-1)** - Returns a sequence from 0 to -9  
**list(range(5))** - Returns a list from 0 to 4

#### DICTIONARIES

**max(d, key=d.get)** - Return the key that corresponds to the largest value in **d**  
**min(d, key=d.get)** - Return the key that corresponds to the smallest value in **d**

#### SETS

**my\_set = set(l)** - Return a **set** object containing the unique values from **l**

**len(my\_set)** - Returns the number of objects in **my\_set** (or, the number of unique values from **l**)  
**a in my\_set** - Returns True if the value **a** exists in **my\_set**

#### REGULAR EXPRESSIONS

**import re** - Import the Regular Expressions module  
**re.search("abc",s)** - Returns a **match** object if the regex "abc" is found in **s**, otherwise None  
**re.sub("abc","xyz",s)** - Returns a string where all instances matching regex "abc" are replaced by "xyz"

#### LIST COMPREHENSION

A one-line expression of a for loop  
**[i \*\* 2 for i in range(10)]** - Returns a list of the squares of values from 0 to 9  
**[s.lower() for s in l\_strings]** - Returns the list **l\_strings**, with each item having had the **.lower()** method applied  
**[i for i in l\_floats if i < 0.5]** - Returns the items from **l\_floats** that are less than 0.5

#### FUNCTIONS FOR LOOPING

```
for i, value in enumerate(l):
    print("The value of item {} is {}".
        format(i,value))
- Iterate over the list l, printing the index location of each item and its value

for one, two in zip(l_one,l_two):
    print("one: {}, two: {}".format(one,two))
- Iterate over two lists, l_one and l_two and print each value

while x < 10:
    x += 1
- Run the code in the body of the loop until the value of x is no longer less than 10
```

#### DATETIME

```
import datetime as dt - Import the datetime module
now = dt.datetime.now() - Assign datetime object representing the current time to now
wks4 = dt.datetime.timedelta(weeks=4)
- Assign a timedelta object representing a timespan of 4 weeks to wks4
```

**now - wks4** - Return a **datetime** object representing the time 4 weeks prior to **now**  
**newyear\_2020 = dt.datetime(year=2020, month=12, day=31)** - Assign a **datetime** object representing December 25, 2020 to **newyear\_2020**  
**newyear\_2020.strftime("%A, %b %d, %Y")** - Returns "Thursday, Dec 31, 2020"  
**dt.datetime.strptime('Dec 31, 2020', "%b %d, %Y")** - Return a **datetime** object representing December 31, 2020

#### RANDOM

```
import random - Import the random module
random.random() - Returns a random float between 0.0 and 1.0
random.randint(0,10) - Returns a random integer between 0 and 10
random.choice(l) - Returns a random item from the list l
```

#### COUNTER

```
from collections import Counter - Import the Counter class
c = Counter(l) - Assign a Counter (dict-like) object with the counts of each unique item from l, to c
c.most_common(3) - Return the 3 most common items from l
```

#### TRY/EXCEPT

Catch and deal with Errors  
**l\_ints = [1, 2, 3, "", 5]** - Assign a list of integers with one missing value to **l\_ints**  
**l\_floats = []**  
**for i in l\_ints:**  
 **try:**  
 **l\_floats.append(float(i))**  
 **except:**  
 **l\_floats.append(i)**  
- Convert each value of **l\_ints** to a float, catching and handling ValueError: could not convert string to float: where values are missing.

- [Python REGEX \(PDF\)](#)

# Data Science Cheat Sheet

## Python Regular Expressions

### SPECIAL CHARACTERS

- `\^` | Matches the expression to its right at the start of a string. It matches every such instance before each `\n` in the string.
- `\$` | Matches the expression to its left at the end of a string. It matches every such instance before each `\n` in the string.
- `\.` | Matches any character except line terminators like `\n`.
- `\`` | Escapes special characters or denotes character classes.
- `A|B` | Matches expression `A` or `B`. If `A` is matched first, `B` is left untried.
- `+` | Greedily matches the expression to its left 1 or more times.
- `*` | Greedily matches the expression to its left 0 or more times.
- `?|` | Greedily matches the expression to its left 0 or 1 times. But if `?` is added to qualifiers (`+, *, and ? itself`) it will perform matches in a non-greedy manner.
- `{m}` | Matches the expression to its left `m` times, and not less.
- `{m,n}` | Matches the expression to its left `m` to `n` times, and not less.
- `{m,n}?` | Matches the expression to its left `m` times, and ignores `n`. See `?` above.

### CHARACTER CLASSES

- [A.K.A. SPECIAL SEQUENCES]
- `\w` | Matches alphanumeric characters, which means `a-z`, `A-Z`, and `0-9`. It also matches the underscore, `_`.
  - `\d` | Matches digits, which means `0-9`.
  - `\D` | Matches any non-digits.
  - `\s` | Matches whitespace characters, which include the `\t`, `\n`, `\r`, and space characters.
  - `\S` | Matches non-whitespace characters.
  - `\b` | Matches the boundary (or empty string) at the start and end of a word, that is, between `\w` and `\W`.
  - `\B` | Matches where `\b` does not, that is, the boundary of `\w` characters.

`\A` | Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.

`\Z` | Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.

### SETS

- `[ ]` | Contains a set of characters to match.
- `[amk]` | Matches either `a`, `m`, or `k`. It does not match `amk`.
- `[a-z]` | Matches any alphabet from `a` to `z`.
- `[a\z]` | Matches `a`, `-`, or `z`. It matches `-` because `\` escapes it.
- `[a-]` | Matches `a` or `-`, because `-` is not being used to indicate a series of characters.
- `[~a]` | As above, matches `a` or `-`.
- `[a-zA-Z0-9]` | Matches characters from `a` to `z` and also from `0` to `9`.
- `[(**)]` | Special characters become literal inside a set, so this matches `(`, `*`, `*`, and `)`.
- `[^ab5]` | Adding `^` excludes any character in the set. Here, it matches characters that are not `a`, `b`, or `5`.

### GROUPS

- `( )` | Matches the expression inside the parentheses and groups it.
- `(?)` | Inside parentheses like this, `?` acts as an extension notation. Its meaning depends on the character immediately to its right.
- `(?P=A)` | Matches the expression `AB`, and it can be accessed with the group name.
- `(?aiLmsux)` | Here, `a`, `i`, `l`, `m`, `s`, `u`, and `x` are flags:
  - `a` — Matches ASCII only
  - `i` — Ignore case
  - `l` — Locale dependent
  - `m` — Multi-line
  - `s` — Matches all
  - `u` — Matches unicode
  - `x` — Verbose

`(?:A)` | Matches the expression as represented by `A`, but unlike `(?P=A)`, it cannot be retrieved afterwards.

`(?#...)` | A comment. Contents are for us to read, not for matching.

`A(?=B)` | Lookahead assertion. This matches the expression `A` only if it is followed by `B`.

`A(?!B)` | Negative lookahead assertion. This matches the expression `A` only if it is not followed by `B`.

`(?<=B)A` | Positive lookbehind assertion.

This matches the expression `A` only if `B` is immediately to its left. This can only matched fixed length expressions.

`(?<!B)A` | Negative lookbehind assertion.

This matches the expression `A` only if `B` is not immediately to its left. This can only matched fixed length expressions.

`(?P=name)` | Matches the expression matched by an earlier group named "name".

`(...)1` | The number `1` corresponds to the first group to be matched. If we want to match more instances of the same expression, simply use its number instead of writing out the whole expression again. We can use from `1` up to `99` such groups and their corresponding numbers.

### POPULAR PYTHON RE MODULE FUNCTIONS

- `re.findall(A, B)` | Matches all instances of an expression `A` in a string `B` and returns them in a list.
- `re.search(A, B)` | Matches the first instance of an expression `A` in a string `B`, and returns it as a re match object.
- `re.split(A, B)` | Split a string `B` into a list using the delimiter `A`.
- `re.sub(A, B, C)` | Replace `A` with `B` in the string `C`.

## Others

- [Python 3 Memento \(PDF\)](#)

©2012-2015 - Laurent Pointal Mémento v2.0.6  
License Creative Commons Attribution 4

Base Types							
<code>integer, float, boolean, string, bytes</code>							
<code>int 783 0 -192 0b010 0o642 0xF3</code>	<code>zero</code>	<code>binary</code>	<code>octal</code>	<code>hexa</code>			
<code>float 9.23 0.0 -1.7e-6</code>							
<code>bool True False</code>		<code>x10<sup>-6</sup></code>					
<code>str "One\nTwo"</code>			<code>Multiline string:</code>				

## Python 3 Cheat Sheet

Latest version on :  
<https://perso.limsi.fr/pointal/python:memento>

Container Types	
<code>list [1,5,9] ["x",11,8.9]</code>	<code>["mot"]</code>
<code>tuple (1,5,9) 11,"y",7.4</code>	<code>("mot",)</code>
<code>Non modifiable values (immutables)</code>	<code>expression with only commas → tuple</code>
<code>str bytes</code> (ordered sequences of chars / bytes)	
<code>key containers</code> , no <i>a priori</i> order, fast key access, each key is unique	

<p><b>escapes</b></p> <ul style="list-style-type: none"> <li>'I\'m'</li> <li>escaped '</li> </ul> <p><b>bytes</b> b"toto\xfe\775"</p> <p>hexadecimal octal</p>	<p><b>dictionary</b> dict {"key": "value"}      dict(a=3, b=4, c="v") (key/value associations) {1: "one", 3: "three", 2: "two", 3.14: "pi"}</p> <p><b>collection</b> set {"key1", "key2"}      {1, 9, 3, 0} keys=hashable values (base types, immutables...)      frozenset immutable set</p>	<p><b>set()</b> empty</p>
<p><b>Identifiers</b></p> <p>for variables, functions, modules, classes... names</p> <p>a...zA...Z_ followed by a...zA...Z_0...9</p> <ul style="list-style-type: none"> <li>diacritics allowed but should be avoided</li> <li>language keywords forbidden</li> <li>lower/UPPER case discrimination</li> <li>a toto x7 y_max BigOne</li> <li>By and for</li> </ul> <p><b>= Variables assignment</b></p> <p># assignment <math>\Rightarrow</math> binding of a name with a value 1) evaluation of right side expression value 2) assignment in order with left side names</p> <p>x=1.2+8+sin(y)</p> <p>a=b=c=0 assignment to same value</p> <p>y, z, r=9.2, -7.6, 0 multiple assignments</p> <p>a, b=b, a values swap</p> <p>a, *b=seq unpacking of sequence in *a, b=seq item and list</p> <p>x+=3 increment <math>\Rightarrow</math> x=x+3</p> <p>x-=2 decrement <math>\Rightarrow</math> x=x-2</p> <p>x=None « undefined » constant value</p> <p>del x remove name x</p> <p><b>for lists, tuples, strings, bytes...</b></p> <p>negative index -5 -4 -3 -2 -1</p> <p>positive index 0 1 2 3 4</p> <p><b>lst=[10, 20, 30, 40, 50]</b></p> <p>positive slice 0 1 2 3 4 5</p> <p>negative slice -5 -4 -3 -2 -1</p> <p><b>Items count</b></p> <p><b>len(lst)→5</b></p> <p># index from 0 (here from 0 to 4)</p> <p><b>Access to sub-sequences via lst[start slice:end slice:step]</b></p> <p><b>lst[:-1]→[10, 20, 30, 40]</b>    <b>lst[::-1]→[50, 40, 30, 20, 10]</b>    <b>lst[1:3]→[20, 30]</b>    <b>lst[:3]→[10, 20, 30]</b>  <b>lst[1:-1]→[20, 30, 40]</b>    <b>lst[::-2]→[50, 30, 10]</b>    <b>lst[-3:-1]→[30, 40]</b>    <b>lst[3:]→[40, 50]</b>  <b>lst[::2]→[10, 30, 50]</b>    <b>lst[:]→[10, 20, 30, 40, 50]</b> shallow copy of sequence</p> <p>Missing slice indication <math>\rightarrow</math> from start / up to end.</p> <p>On mutable sequences (list), remove with <b>del lst[3:5]</b> and modify with assignment <b>lst[1:4]=[15, 25]</b></p>		
<p><b>Conversions</b></p> <p>can specify integer number base in 2<sup>nd</sup> parameter</p> <p>truncate decimal part</p> <p>rounding to 1 decimal (0 decimal <math>\rightarrow</math> integer number)</p> <p>bool(x) False for null x, empty container x, None or False x; True for other x</p> <p>str(x)→"..." representation string of x for display (cf. formatting on the back)</p> <p>chr(64)→@ ord('@')→64 code <math>\leftrightarrow</math> char</p> <p>repr(x)→"..." literal representation string of x</p> <p>bytes([72, 9, 64])→b'H\t@\n'</p> <p>list("abc")→['a', 'b', 'c']</p> <p>dict([(3, "three"), (1, "one")])→{1: 'one', 3: 'three'}</p> <p>set(["one", "two"])→{'one', 'two'}</p> <p>separator str and sequence of str → assembled str</p> <p>'.'.join(['toto', '12', 'pswd'])→'toto:12:pswd'</p> <p>str splitted on whitespaces → list of str</p> <p>"words with spaces".split()→['words', 'with', 'spaces']</p> <p>str splitted on separator str → list of str</p> <p>"1,4,8,2".split(",")→['1', '4', '8', '2']</p> <p>sequence of one type <math>\rightarrow</math> list of another type (via list comprehension)</p> <p>[int(x) for x in ('1', '29', '-3')]→[1, 29, -3]</p> <p><b>Sequence Containers Indexing</b></p> <p>Individual access to items via <b>lst[index]</b></p> <p><b>lst[0]→10</b> <math>\Rightarrow</math> first one    <b>lst[1]→20</b>  <b>lst[-1]→50</b> <math>\Rightarrow</math> last one    <b>lst[-2]→40</b></p> <p>On mutable sequences (list), remove with <b>del lst[3]</b> and modify with assignment <b>lst[4]=25</b></p>		
<p><b>Boolean Logic</b></p> <p>Comparisons: &lt; &gt; &lt;= &gt;= == != (boolean results)    <math>\leq \geq \neq</math></p> <p><b>a and b</b> logical and both simultaneously</p> <p><b>a or b</b> logical or one or other or both</p> <p># pitfall : and and or return value of a or of b (under shortcut evaluation). <math>\Rightarrow</math> ensure that a and b are booleans.</p> <p><b>not a</b> logical not</p> <p><b>True</b>    <b>False</b>    True and False constants</p>	<p><b>Statements Blocks</b></p> <p><b>parent statement:</b> statement block 1... ⋮</p> <p><b>parent statement:</b> statement block2... ⋮</p> <p><b>next statement after block 1</b></p> <p># configure editor to insert 4 spaces in place of an indentation tab.</p>	<p><b>Modules/Names Imports</b></p> <p>from monmod import nom1, nom2 as fct <math>\Rightarrow</math> direct access to names, renaming with as</p> <p>import monmod → access via monmod.nom1...</p> <p># modules and packages searched in python path (cf. sys.path)</p> <p><b>Conditional Statement</b></p> <p>statement block executed only if a condition is true</p> <p><b>if logical condition:</b> → statements block</p> <p>Can go with several elif, elif... and only one final else. Only the block of first true condition is executed.</p> <p># with a var x: if bool(x)==True: <math>\Rightarrow</math> if x: if bool(x)==False: <math>\Rightarrow</math> if not x:</p> <p><b>Exceptions on Errors</b></p> <p>Signaling an error: raise ExcClass(...)</p> <p>Errors processing: try: → normal processing block except Exception as e: → error processing block</p> <p># finally block for final processing in all cases.</p>
<p># floating numbers... approximated values</p> <p>Operators: + - * / // % ** Priority (...)    <math>\frac{x}{y}</math>    <math>x^y</math> integer <math>\div</math> remainder @ → matrix <math>\times</math> python3.5+numpy <math>(1+5.3)*2\rightarrow12.6</math> <math>abs(-3.2)\rightarrow3.2</math> <math>round(3.57, 1)\rightarrow3.6</math> <math>pow(4, 3)\rightarrow64.0</math> # usual order of operations</p>	<p>angles in radians</p> <p>from math import sin, pi... <math>\sin(\pi/4)\rightarrow0.707...</math> <math>\cos(2*\pi/3)\rightarrow-0.4999...</math> <math>\sqrt(81)\rightarrow9.0</math> ✓ <math>\log(e^{**2})\rightarrow2.0</math> <math>\text{ceil}(12.5)\rightarrow13</math> <math>\text{floor}(12.5)\rightarrow12</math></p> <p>modules math, statistics, random, decimal, fractions, numpy, etc. (cf. doc)</p>	<p><b>Conditional Loop Statement</b></p> <p>statements block executed as long as condition is true</p> <p><b>while logical condition:</b> → statements block</p> <p>s = 0 # initializations before the loop i = 1 # condition with a least one variable value (here i)</p> <p>while i &lt;= 100:     s = s + i**2     i = i + 1 # make condition variable change!</p> <p><b>Loop Control</b></p> <p>break immediate exit continue next iteration else block for normal loop exit.</p> <p>Algo: <math>i=100</math> <math>s=\sum_{i=1}^{100} i^2</math></p> <p><b>Iterative Loop Statement</b></p> <p>statements block executed for each item of a container or iterator</p> <p><b>for var in sequence:</b> → statements block</p> <p>Go over sequence's values s = "Some text" # initializations before the loop cnt = 0 # loop variable, assignment managed by for statement</p> <p>for c in s:     if c == "e":         cnt = cnt + 1     print("found!", cnt, "'e'")</p> <p>loop on dict/set <math>\leftrightarrow</math> loop on keys sequences use slices to loop on a subset of a sequence</p> <p>Go over sequence's index # modify item at index # access items around index (before / after) lst = [11, 18, 9, 12, 23, 4, 17] lost = [] for i in range(len(lst)):     lost.append(lst[i])</p> <p>habit: don't modify loop variable</p>

```
s = input("Instructions:")
# input always returns a string, convert it to required type
# (cf. boxed Conversions on the other side).
```

**Input**

`len(c) → items count`  
`min(c) max(c) sum(c)` Note: For dictionaries and sets, these operations use keys.  
`sorted(c) → list sorted copy`  
`val in c → boolean, membership operator in (absence not in)`  
`enumerate(c) → iterator on (index, value)`  
`zip(c1, c2...) → iterator on tuples containing ci items at same index`  
`all(c) → True if all c items evaluated to true, else False`  
`any(c) → True if at least one item of c evaluated true, else False`  
Specific to ordered sequences containers (lists, tuples, strings, bytes...)  
`reversed(c) → inverted iterator` c\*5 → duplicate    c+c2 → concatenate  
`c.index(val) → position`    `c.count(val) → events count`  
`import copy`  
`copy.copy(c) → shallow copy of container`  
`copy.deepcopy(c) → deep copy of container`

**Generic Operations on Containers**

# modify original list  
`lst.append(val)` add item at end  
`lst.extend(seq)` add sequence of items at end  
`lst.insert(idx, val)` insert item at index  
`lst.remove(val)` remove first item with value val  
`lst.pop([idx]) → value` remove & return item at index idx (default last)  
`lst.sort() lst.reverse()` sort / reverse liste in place

**Operations on Lists**

**Operations on Dictionaries**  
`d[key]=value`    `d.clear()`  
`d[key] → value`    `del d[key]`  
`d.update(d2)` update/add  
`d.keys()` associations  
`d.values()` → iterable views on  
`d.items()` keys/values/associations  
`d.pop(key[,default]) → value`  
`d.popitem() → (key,value)`  
`d.get(key[,default]) → value`  
`d.setdefault(key[,default]) → value`

**Operations on Sets**  
Operators:  
`|` → union (vertical bar char)  
`&` → intersection  
`-` → difference/symmetric diff.  
`< <= > >=` → inclusion relations  
Operators also exist as methods.  
`s.update(s2)`    `s.copy()`  
`s.add(key)`    `s.remove(key)`  
`s.discard(key)`    `s.clear()`  
`s.pop()`

storing data on disk, and reading it back  
`f = open("file.txt", "w", encoding="utf8")`  
file variable    name of file    opening mode    encoding of  
for operations    on disk       □ 'r' read    chars for text  
                    (+path...)    □ 'w' write    files:  
                                   □ 'a' append    utf8    ascii  
cf. modules os, os.path and pathlib    ... '+' 'x' 'b' 't' latin1 ...  
**writing**    `# read empty string if end of file`    **reading**  
`f.write("coucou")`    `f.read([n])` → next chars  
`f.writelines(list of lines)`    if n not specified, read up to end!  
`f.readline()` → list of next lines  
`f.readlines([n])` → next line  
**text mode t by default (read/write str), possible binary mode b (read/write bytes). Convert from/to required type!**  
`f.close()`    `# dont forget to close the file after use!`  
`f.flush()` write cache    `f.truncate(size)` resize  
reading/writing progress sequentially in the file, modifiable with:  
`f.tell() → position`    `f.seek(position, origin)`  
Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:  
`with open(...) as f:`  
`for line in f:`  
`# processing of line`

```
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:", lst, "-lost:", lost)
```

than 15, memorizing of lost values.

as good

Go simultaneously over sequence's index and values:  
`for idx, val in enumerate(lst):`

**Integer Sequences**

`range([start,] end [,step])`  
# start default 0, end not included in sequence, step signed, default 1  
`range(5) → 0 1 2 3 4`    `range(2, 12, 3) → 2 5 8 11`  
`range(3, 8) → 3 4 5 6 7`    `range(20, 5, -5) → 20 15 10`  
`range(len(seq)) → sequence of index of values in seq`  
# range provides an immutable sequence of int constructed as needed

function name (identifier)

↓ named parameters

`def fct(x, y, z):`

**Function Definition**

`"""documentation"""`

→ # statements block, res computation, etc.

`return res` → result value of the call, if no computed result to return: `return None`

# parameters and all variables of this block exist only in the block and during the function call (think of a "black box")

Advanced: `def fct(x, y, z, *args, a=3, b=5, **kwargs):`

\*args variable positional arguments (→tuple), default values.

\*\*kwargs variable named arguments (→dict)

`fct`

`r = fct(3, i+2, 2*i)`

storage/use of one argument per returned value parameter

# this is the use of function Advanced: which does the call

`fct()`    `fct`

**Function Call**

`s.startswith(prefix,start,end))`  
`s.endswith(suffix,start,end))`    `s.strip(chars)`  
`s.count(sub,start,end))`    `s.partition(sep) → (before,sep,after)`  
`s.index(sub,start,end))`    `s.find(sub,start,end))`  
`s.is...()` tests on chars categories (ex. `s.isalpha()`)  
`s.upper()`    `s.lower()`    `s.title()`    `s.swapcase()`  
`s.casefold()`    `s.capitalize()`    `s.center([width,fill])`  
`s.ljust([width,fill])`    `s.rjust([width,fill])`    `s.zfill([width])`  
`s.encode(encoding)`    `s.split([sep])`    `s.join(seq)`

formatting directives    values to format

"modele{} {} {}".format(x, y, r) → str

"{selection : formatting ! conversion}"

□ Selection :  
2    nom  
0.nom  
4[key]  
0[2]  
Example: `"{:+2.3f)".format(45.72793)`  
→ '+45.728'  
`"{1:>10s)".format("toto")`  
→ '        toto'  
`"{x!r)".format(x="I'm")`  
→ "I'm"

□ Formatting :  
fill char alignment sign min width precision-maxwidth type  
<> ^ = + - space 0 at start for filling with 0  
integer: b binary, c char, d decimal (default), o octal, x or X hexa...  
float: e or E exponential, f or F fixed point, g or G appropriate (default),  
string: s ... % percent  
□ Conversion : s (readable text) or x (literal representation)

R

# Datacamp

## • Tidiverse (PDF)

**R For Data Science Cheat Sheet**

Tidyverse for Beginners

Learn More R for Data Science Interactively at: [www.datacamp.com](http://www.datacamp.com)

**Tidverse**

The tidyverse is a powerful collection of R packages that are actually data tools for transforming and visualizing data. All packages of the tidyverse share an underlying philosophy and common APIs.

The core packages are:

- ggplot2, which implements the grammar of graphics. You can use it to visualize your data.
- dplyr is a grammar of data manipulation. You can use it to solve the most common data manipulation challenges.
- tidyverse helps you to create tidy data or data where each variable is in a column, each observation is a row and each value is a cell.
- readr is a fast and friendly way to read rectangular data.
- purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.
- tibble is a modern re-imaging of the data frame.
- string provides a cohesive set of functions designed to make working with strings as easy as possible.
- forcats provide a suite of useful tools that solve common problems with factors.

You can install the complete tidyverse with:

```
> install.packages("tidyverse")
```

Then, load the core tidyverse and make it available in your current R session by running:

```
> library(tidyverse)
```

Note: there are many other tidyverse packages with more specialised usage. They are not loaded automatically with library(tidyverse), so you'll need to load each one with its own call to library().

**Useful Functions**

<code>tidyverse_conflicts()</code>	Conflicts between tidyverse and other packages
<code>tidyverse_deps()</code>	List all tidyverse dependencies
<code>tidyverse_logo()</code>	Get tidyverse logo, using ASCII or unicode characters
<code>tidyverse_packages()</code>	List all tidyverse packages
<code>tidyverse_update()</code>	Update tidyverse packages

**Loading in the data**

<code>&gt; library(datasets)</code>	Load the datasets package
<code>&gt; library(gapminder)</code>	Load the gapminder package
<code>&gt; attach(iris)</code>	Attach iris data to the R search path

**dplyr**

**Filter**

`filter()` allows you to select a subset of rows in a data frame.

```
> iris %>% filter(Species=="virginica")
> iris %>% filter(Species=="virginica", Sepal.Length > 6)
```

Select iris data of species "virginica". Select iris data of species "virginica" and sepal length greater than 6.

**Arrange**

`arrange()` sorts the observations in a dataset in ascending or descending order based on one of its variables.

```
> iris %>% arrange(Sepal.Length)
> iris %>% arrange(desc(Sepal.Length))
```

Sort in ascending order of sepal length  
Sort in descending order of sepal length

**Combine multiple dplyr verbs in a row with the pipe operator %>%:**

```
> iris %>% filter(Species=="virginica") %>% arrange(desc(Sepal.Length))
```

Filter for species "virginica" then arrange in descending order of sepal length

**Mutate**

`mutate()` allows you to update or create new columns of a data frame.

```
> iris %>% mutate(Sepal.Length=Sepal.Length*10)
> iris %>% mutate(SLMM=Sepal.Length*10)
```

Change Sepal.Length to be in millimeters  
Create a new column called SLMM

**Combine the verbs `filter()`, `arrange()`, and `mutate()`:**

```
> iris %>% filter(Species=="virginica") %>% mutate(SLMM=Sepal.Length*10) %>% arrange(desc(SLMM))
```

Filter for species "virginica" then arrange in descending order of sepal length

**Summarize**

`summarise()` allows you to turn many observations into a single data point.

```
> iris %>% summarise(medianSL=median(Sepal.Length))
> iris %>% filter(Species=="virginica") %>% summarise(medianSL=median(Sepal.Length))
```

Summarize to find the median sepal length  
Filter for virginica then summarize the median sepal length

**group\_by()** allows you to summarize within groups instead of summarizing the entire dataset:

```
> iris %>% group_by(Species) %>% summarise(medianSL=median(Sepal.Length),
> maxSL=max(Sepal.Length))
> iris %>% filter(Sepal.Length>6) %>% group_by(Species) %>% summarise(medianSL=median(Sepal.Length),
> maxSL=max(Sepal.Length))
```

Find median and max sepal length of each species  
Find median and max petal length of each species with sepal length > 6

**ggplot2**

**Scatter plot**

Scatter plots allow you to compare two variables within your data. To do this with ggplot2, you use `geom_point()`.

```
> iris_small <- iris %>% filter(Sepal.Length > 5)
> ggplot(iris_small, aes(x=Sepal.Length, y=Sepal.Width)) + geom_point()
```

Compare petal width and length

**Additional Aesthetics**

- Color**
- Size**
- Faceting**
- Line Plots**
- Bar Plots**
- Histograms**
- Box Plots**

**DataCamp**

Learn R for Data Science Interactively

## • data.table (PDF)

**R For Data Science Cheat Sheet**

data.table

Learn R for data science Interactively at: [www.DataCamp.com](http://www.DataCamp.com)

**data.table**

data.table is an R package that provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Load the package:

```
> library(data.table)
```

**Creating A data.table**

<code>&gt; set.seed(453)</code>	Create a data.table and call it DT
<code>&gt; DT &lt;- data.table(V1=c(1L, 2L, V2=LETTERS[1:3], V3=rnorm(3, 0, 1), V4=1:12)</code>	

**Subsetting Rows Using i**

<code>&gt; DT[3:5]</code>	Select 3rd to 5th row
<code>&gt; DT[3:5, ]</code>	Select 3rd to 5th row
<code>&gt; DT[V2=="A"]</code>	Select all rows that have value A in column V2
<code>&gt; DT[V1==1:4]</code>	Select all rows that have value 1 or 4 in column V1

**Manipulating on Columns in j**

<code>&gt; DT[, V2]</code>	Return V2 as a vector
<code>&gt; DT[, V2:V3]</code>	Return V2 and V3 as a data.table
<code>&gt; DT[, sum(V1)]</code>	Return the sum of all elements of V1 in a vector
<code>&gt; DT[, V1:sum(V1), std(V3)]</code>	Return the sum of V1 and the std. dev. of V3 in a data.table
<code>&gt; DT[, .(Aggregate=mean(V1), SD.V3=sd(V3))]</code>	The same as the above, with new names
<code>&gt; DT[, V1, SD.V3=sd(V3)]</code>	Select column V1 and compute std. dev. of V3, which returns a single value and gets recycled
<code>&gt; DT[, .(print(V2), plot(V3), NULL)]</code>	Print column V2 and plot V3

**Doing j by Group**

<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1]</code>	Calculate sum of V4 for every group in V1
<code>&gt; V1</code>	V1
<code>&gt; 1</code>	1
<code>&gt; 2</code>	2
<code>&gt; 3</code>	3
<code>&gt; 4</code>	4
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=(V1, V2)]</code>	Calculate sum of V4 for every group in V1 and V2
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=sign(V1-1)]</code>	Calculate sum of V4 for every group in sign(V1-1)
<code>&gt; V1</code>	V1
<code>&gt; 1</code>	1
<code>&gt; 2</code>	2
<code>&gt; 3</code>	3
<code>&gt; 4</code>	4
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=(V1-1, V2-1)]</code>	The same as the above, with new name
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1]</code>	Calculate sum of V4 for every group in V1 after subsetting on the first 5 rows
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1]</code>	Count number of rows for every group in V1

**General form: DT[i, j, by= ]**

"Take DT, subset rows using i, then calculate j grouped by by"

**Adding/Updating Columns By Reference in j Using :=**

<code>&gt; DT[, V1:=round(exp(V1), 2)]</code>	V1 is updated by what is after := Return the result by calling DT
<code>&gt; DT[, V1:= (V1-round(exp(V1), 2), LETTERS[4:6])]</code>	Columns V1 and V2 are updated by what is after := Alternative to the above one. With [], you print the result to the screen
<code>&gt; DT[, V1:= (V1-round(exp(V1), 2), V2=LETTERS[4:6])][ ]</code>	Remove V1 Remove columns V1 and V2 Delete the column with column name V2 Delete the columns specified in the variable Vcols chosen

**Indexing And Keys**

<code>&gt; set.seed(123)</code>	A key is set on V2; output is returned invisibly
<code>&gt; DT[, V2]</code>	Return all rows where the key column (set to V2) has the value A
<code>&gt; DT[, V2]</code>	Return first row of all rows that match value A in key column V2
<code>&gt; DT[, V2]</code>	Return last row of all rows that match value A in key column V2
<code>&gt; DT[, V2]</code>	Return all rows where key column V2 has value A or B
<code>&gt; DT[, V2]</code>	Return total sum of V4, for rows of key column V2 that have values A or C
<code>&gt; DT[, V2]</code>	Return sum of column V4 for rows of V2 that have value A, and another sum for rows of V2 that have value C
<code>&gt; DT[, V2]</code>	Sort by V1 and then by V2 within each group of V3 (invisible)
<code>&gt; DT[, V2]</code>	Select rows that have value 2 for the first key (V2) and the value C for the second key (V2)
<code>&gt; DT[, V2]</code>	Select rows that have value 2 for the first key (V2) and within those rows the value A or C for the second key (V2)

**Advanced Data Table Operations**

<code>&gt; DT[, -1]</code>	Return the penultimate row of the DT
<code>&gt; DT[, -N]</code>	Return the number of rows
<code>&gt; DT[, -(V2, V3)]</code>	Return V2 and V3 as a data.table
<code>&gt; DT[, 1:10, -1, by=Species]</code>	Return V2 and V3 as a data.table
<code>&gt; DT[, mean(V2), by=, by2=, by3=, by4=, by5=]</code>	Return the results of 5 grouped by all possible combinations of groups specified in by

**.SD & .SDcols**

<code>&gt; DT[, print(.SD), by=V2]</code>	look at what .SD contains
<code>&gt; DT[, (.SD[, 1:10], by=V2)]</code>	Select the first and last row grouped by V2
<code>&gt; DT[, lapply(.SD[, 1:10], sum, by=V2)]</code>	Calculate sum of columns in .SD grouped by V2
<code>&gt; DT[, lapply(.SD[, sum(.SD[, 1:10]), by=V2)]]</code>	Calculate sum of V3 and V4 in .SD grouped by V2
<code>&gt; DT[, lapply(.SD[, sum(.SD[, 1:10]), by=V2, .SDcols=paste0("V", 3:4)])]</code>	Calculate sum of V3 and V4 in .SD grouped by V2

**Chaining**

<code>&gt; DT &lt;- DT[, .(V4, Sum=sum(V4)), by=V1]</code>	Calculate sum of v4, grouped by v1
<code>&gt; V1</code>	V1
<code>&gt; 1</code>	1
<code>&gt; 2</code>	2
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1, V4&gt;40]</code>	Select that group of which the sum is >40
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1, V4&gt;40]</code>	Select that group of which the sum is >=40 (chaining)
<code>&gt; DT[, .(V4, Sum=sum(V4)), by=V1, order=-V1]</code>	Calculate sum of v4, grouped by v1, ordered by v1
<code>&gt; V1</code>	V1
<code>&gt; 1</code>	1
<code>&gt; 2</code>	2

**set () -Family**

**set()**

<code>Syntax: for (i in from:to) set(DT, row, column, new_value)</code>	Sequence along the values of row, and for the values of col, set the values of those elements equal to new (invisible)
<code>&gt; rows &lt;- 1:10(3:4,5:6)</code>	
<code>&gt; cols &lt;- 1:12</code>	
<code>&gt; for(i in seq_along(rows))</code>	
<code>&gt; {set(DT, rows[i], cols, value)}</code>	
<code>&gt; DT</code>	

**setnames()**

<code>Syntax: setnames(DT, "old", "new")</code>	
<code>&gt; setnames(DT, "V2", "Rating")</code>	Set name of v2 to Rating (invisible)
<code>&gt; setnames(DT, "V2", "Rating", c("V2_Rating", "V3_Rating"))</code>	Change 2 column names (invisible)

**setnames()**

<code>Syntax: setnames(DT, "neworder")</code>	
<code>&gt; setorder(DT, c("V2", "V1", "V4", "V3"))</code>	Change column ordering to contents of specified vector (invisible)

**DataCamp**

Learn Python for Data Science Interactively

## -xts (PDF)

### R For Data Science Cheat Sheet

xts

Learn R for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)

#### xts

**extensible Time Series (xts)** is a powerful package that provides an extensible time series class, enabling uniform handling of many R time series classes by extending `zoo`.

Load the package as follows:

```
> library(xts)
```

#### xts Objects

xts objects have three main components:

- `coredata`: always a matrix for xts objects, while it could also be a vector for zoo objects
- `index`: vector of any Date, POSIXct, chron, yearmon, yearqtr, or DateTime classes
- `xtsAttributes`: arbitrary attributes

#### Creating xts Objects

```
> xt1 <- xts(x=1:10, order.by=Sys.Date()-1:10)
> data <- rnorm(5)
> dates <- seq(as.Date("2017-05-01"), length=5, by="days")
> xt2 <- xts(x=data, order.by=dates)
> xt3 <- xts(x=rnorm(10),
+             order.by=as.POSIXct(Sys.Date() + 1:10),
+             born=as.POSIXct("1889-05-08"))
> xt4 <- xts(x=1:10, order.by=Sys.Date() + 1:10)
```

#### Convert To And From xts

```
> data(AirPassengers)
> xt5 <- as.xts(AirPassengers)
```

#### Import/From Files

```
> dat <- read.csv(tmp_file)
> xt6 <- data[, order(by=as.Date(rownames(dat)), "%m/%d/%Y")]
> dat_zoo <- read.zoo(tmp_file,
+                      index.column=0,
+                      sep=",",
+                      format="%m/%d/%Y")
> dat_zoo <- read.zoo(tmp_file, sep=",", FUN=as.yearmon)
> dat_xts <- as.xts(dat_zoo)
```

#### Inspect Your Data

<pre>&gt; core_data &lt;- coredata(xt2)</pre>	Extract core data of objects
<pre>&gt; index(xt1)</pre>	Extract index of objects
<b>Class Attributes</b>	
<pre>&gt; indexClass(xt2)</pre>	Get index class
<pre>&gt; indexClass(indexVectorIndex(xts, "POSIXct"))</pre>	Replacing index class
<pre>&gt; indexTS(xts)</pre>	Get index class
<pre>&gt; indexFormat(xt5) &lt;- "%Y-%m-%d"</pre>	Change format of time display
<b>Time Zones</b>	
<pre>&gt; tzzone(xts1) &lt;- "Asia/Hong_Kong"</pre>	Change the time zone
<pre>&gt; tzzone(xts1)</pre>	Extract the current time zone

#### Export xts Objects

```
> data_xts <- as.xts(matrix)
> tmp <- tempfile()
> write.zoo(data_xts, sep=":", file=tmp)
```

#### Replace & Update

```
> xt2$dates[1] <- 0
> xt5$`061` <- NA
> xt5["2016-05-02"] <- NA
```

Replace values in `xt2` on dates with 0  
Replace dates from 1981 with NA  
Replace the value at 1 specific index with NA

#### Applying Functions

```
> ep1 <- endpoints(xt5, on="weeks", k=2)
> ep2 <- endpoints(xt5, on="years")
[1] 9 32 24 36 48 60 77 84 94 109 120 132 144
> period.apply(xt5, INDEX=ep2, FUN=mean)
> xt5$yearly <- split(xt5,f="yearly")
> lapply(xt5$yearly,FUN=mean)
> do.call(rbind,
+          lapply(xt5$yearly,
+                 function(w) last(w,n=1 month)))
> do.call(rbind,
+          lapply(xt5$yearly,
+                 cumsum))
> rollapply(xt5, 3, sd)
```

Take index values by time  
Calculate the yearly mean  
Split `xt5` by year  
Create a list of yearly means  
Find the last observation in each year in `xt5`  
Calculate cumulative annual passengers  
Apply sd to rolling margins of `xt5`

#### Selecting, Subsetting & Indexing

##### Select

```
> mar55 <- xt5["1955-03"]
```

Get value for March 1955

##### Subset

```
> xt5_1954 <- xt5["1954"]
> xt5_janmarch <- xt5["1954/1954-03"]
> xt5_janmarch <- xt5["1954/1954-02"]
> xt5[4][1]
```

Get all data from 1954  
Extract data from Jan to March '54  
Get all data until March '54  
Subset `xt5` using `ep2`

##### first() and last()

```
> first(xt4,"1 week")
> first(last(xt5,"1 week"), "3 days")
```

Extract first 1 week  
Get first 3 days of the last week of data

##### Indexing

```
> xt2[index(xt3)]
```

Extract rows with the index of `xt3`

```
> days <- c("2017-05-03", "2017-05-23")
```

Extract rows using the vector `days`

```
> xt3[days]
```

Extract rows using `days` as POSIXct

```
> index <- which(index(xt3)-0, indexday(xt3)==0)
```

Index of weekend days

```
> xtall[index]
```

Extract weekend days of `xt1`

#### Missing Values

```
> na.omit(xt5)
> xt5_last <- na.locf(xt5)
> xt5_last <- na.locf(xt5,
+                      fromlast=TRUE)
> na.approx(xt5)
```

Omit NA values in `xt5`  
Fill missing values in `xt5` using last observation  
Fill missing values in `xt5` using next observation  
Interpolate NAs using linear approximation

#### Arithmetic Operations

##### coredata() or as.numeric()

```
> xt53 <- as.numeric(xt5)
> xt53 * 2
> coredata(xt54) - xt53
> coredata(xt54) / xt53
```

Addition  
Multiplication  
Subtraction  
Division

##### Shifting Index Values

```
> xt55 <- lag(xt5)
> diff(xt55, lag=1, differences=1)
```

Period-over-period differences  
Lagged differences

##### Reindexing

```
> xt51 + merge(xt52, index(xt51), all=0)
> xt51 + merge(xt52, index(xt51), all=na.locf)
> xt51 - merge(xt52, index(xt51), all=na.locf)
```

Addition

```
> merge(xt52,xt51,join='inner')
```

Inner join of `xt52` and `xt51`

##### Merging

```
> merge(xt52,xt51,join='left', all=0)
> merge(xt52,xt51,join='left', fill=0)
> merge(xt52,xt51,join='left', fill=0)
> rbind(xt51, xt52)
```

Left join of `xt52` and `xt51`, fill empty spots with 0

Combine `xt51` and `xt52` by rows

DataCamp

Learn R for Data Science interactively



# RStudio

## • R Studio IDE (PDF)

## • Base R (PDF)

## • Data Import with readr (PDF)

## • Data Transformation with Dplyr (PDF)

## • Apply Functions with purrr (PDF)

## • Data transformation with data.table (PDF)

## • Dates and Times with lubridate (PDF)

- [Randomizr \(PDF\)](#)
- [Regular Expressions \(PDF\)](#)
- [Work with Strings with stringr \(PDF\)](#)
- [Tidy Evaluation with rlang \(PDF\)](#)
- [Xplain \(PDF\)](#)
- [Sintax Comparison \(PDF\)](#)
- [Data and Variable Transformation with sjmisc \(PDF\)](#)
- [R Markdown \(PDF\)](#)
- [Package Development with devtools \(PDF\)](#)

## Math and Calculus

---

From @afshinea, @stat110 and @wzchen:

## • Refresher Algebra and Calculus (PDF)

### VIP Refresher: Linear Algebra and Calculus

Afshine AMIDI and Shervine AMIDI

October 6, 2018

#### General notations

□ **Vector** – We note  $x \in \mathbb{R}^n$  a vector with  $n$  entries, where  $x_i \in \mathbb{R}$  is the  $i^{th}$  entry:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

□ **Matrix** – We note  $A \in \mathbb{R}^{m \times n}$  a matrix with  $m$  rows and  $n$  columns, where  $A_{i,j} \in \mathbb{R}$  is the entry located in the  $i^{th}$  row and  $j^{th}$  column:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

*Remark: the vector  $x$  defined above can be viewed as a  $n \times 1$  matrix and is more particularly called a column-vector.*

□ **Identity matrix** – The identity matrix  $I \in \mathbb{R}^{n \times n}$  is a square matrix with ones in its diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

*Remark: for all matrices  $A \in \mathbb{R}^{n \times n}$ , we have  $A \cdot I = I \cdot A = A$ .*

□ **Diagonal matrix** – A diagonal matrix  $D \in \mathbb{R}^{n \times n}$  is a square matrix with nonzero values in its diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

*Remark: we also note  $D$  as  $\text{diag}(d_1, \dots, d_n)$ .*

#### Matrix operations

□ **Vector-vector multiplication** – There are two types of vector-vector products:

- inner product: for  $x, y \in \mathbb{R}^n$ , we have:

$$x^T y = \sum_{i=1}^n x_i y_i \in \mathbb{R}$$

- outer product: for  $x \in \mathbb{R}^m, y \in \mathbb{R}^n$ , we have:

$$xy^T = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & \ddots & \vdots \\ x_m y_1 & \cdots & x_m y_n \end{pmatrix} \in \mathbb{R}^{m \times n}$$

□ **Matrix-vector multiplication** – The product of matrix  $A \in \mathbb{R}^{m \times n}$  and vector  $x \in \mathbb{R}^n$  is a vector of size  $\mathbb{R}^m$ , such that:

$$Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^n a_{r,i} x_i \in \mathbb{R}^m$$

where  $a_{r,i}^T$  are the vector rows and  $a_{r,i}$  are the vector columns of  $A$ , and  $x_i$  are the entries of  $x$ .

□ **Matrix-matrix multiplication** – The product of matrices  $A \in \mathbb{R}^{m \times n}$  and  $B \in \mathbb{R}^{n \times p}$  is a matrix of size  $\mathbb{R}^{m \times p}$ , such that:

$$AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & \ddots & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^n a_{r,i} b_{c,i}^T \in \mathbb{R}^{m \times p}$$

where  $a_{r,i}^T b_{c,i}^T$  are the vector rows and  $a_{r,i}, b_{c,i}$  are the vector columns of  $A$  and  $B$  respectively.

□ **Transpose** – The transpose of a matrix  $A \in \mathbb{R}^{m \times n}$ , noted  $A^T$ , is such that its entries are flipped:

$$\forall i,j, \quad A_{i,j}^T = A_{j,i}$$

*Remark: for matrices  $A, B$ , we have  $(AB)^T = B^T A^T$ .*

□ **Inverse** – The inverse of an invertible square matrix  $A$  is noted  $A^{-1}$  and is the only matrix such that:

$$AA^{-1} = A^{-1}A = I$$

*Remark: not all square matrices are invertible. Also, for matrices  $A, B$ , we have  $(AB)^{-1} = B^{-1}A^{-1}$ .*

□ **Trace** – The trace of a square matrix  $A$ , noted  $\text{tr}(A)$ , is the sum of its diagonal entries:

$$\text{tr}(A) = \sum_{i=1}^n A_{i,i}$$

*Remark: for matrices  $A, B$ , we have  $\text{tr}(AT) = \text{tr}(A)$  and  $\text{tr}(AB) = \text{tr}(BA)$ .*

□ **Determinant** – The determinant of a square matrix  $A \in \mathbb{R}^{n \times n}$ , noted  $|A|$  or  $\det(A)$  is expressed recursively in terms of  $A_{\setminus i, \setminus j}$ , which is the matrix  $A$  without its  $i^{th}$  row and  $j^{th}$  column, as follows:

$$\det(A) = |A| = \sum_{j=1}^n (-1)^{i+j} A_{i,j} |A_{\setminus i, \setminus j}|$$

*Remark:  $A$  is invertible if and only if  $|A| \neq 0$ . Also,  $|AB| = |A||B|$  and  $|A^T| = |A|$ .*

#### Matrix properties

□ **Symmetric decomposition** – A given matrix  $A$  can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}$$

□ **Norm** – A norm is a function  $N : V \rightarrow [0, \infty]$  where  $V$  is a vector space, and such that for all  $x, y \in V$ , we have:

- $N(x + y) \leq N(x) + N(y)$
- $N(ax) = |a|N(x)$  for a scalar
- if  $N(x) = 0$ , then  $x = 0$

For  $x \in V$ , the most commonly used norms are summed up in the table below:

Norm	Notation	Definition	Use case
Manhattan, $L^1$	$\ x\ _1$	$\sum_{i=1}^n  x_i $	LASSO regularization
Euclidean, $L^2$	$\ x\ _2$	$\sqrt{\sum_{i=1}^n x_i^2}$	Ridge regularization
$p$ -norm, $L^p$	$\ x\ _p$	$\left(\sum_{i=1}^n x_i^p\right)^{\frac{1}{p}}$	Hölder inequality
Infinity, $L^\infty$	$\ x\ _\infty$	$\max_i  x_i $	Uniform convergence

□ **Linearly dependence** – A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others.

*Remark: if no vector can be written this way, then the vectors are said to be linearly independent.*

□ **Matrix rank** – The rank of a given matrix  $A$  is noted  $\text{rank}(A)$  and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of  $A$ .

□ **Positive semi-definite matrix** – A matrix  $A \in \mathbb{R}^{n \times n}$  is positive semi-definite (PSD) and is noted  $A \succeq 0$  if we have:

$$A = A^T \quad \text{and} \quad \forall x \in \mathbb{R}^n, \quad x^T Ax \geq 0$$

*Remark: similarly, a matrix  $A$  is said to be positive definite, and is noted  $A \succ 0$ , if it is a PSD matrix which satisfies for all non-zero vector  $x$ ,  $x^T Ax > 0$ .*

□ **Eigenvalue, eigenvector** – Given a matrix  $A \in \mathbb{R}^{n \times n}$ ,  $\lambda$  is said to be an eigenvalue of  $A$  if there exists a vector  $z \in \mathbb{R}^n \setminus \{0\}$ , called eigenvector, such that we have:

$$Az = \lambda z$$

□ **Spectral theorem** – Let  $A \in \mathbb{R}^{n \times n}$ . If  $A$  is symmetric, then  $A$  is diagonalizable by a real orthogonal matrix  $U \in \mathbb{R}^{n \times n}$ . By noting  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ , we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

□ **Singular-value decomposition** – For a given matrix  $A$  of dimensions  $m \times n$ , the singular-value decomposition (SVD) is a factorization technique that guarantees the existence of  $U \in m \times m$  unitary,  $\Sigma \in m \times n$  diagonal and  $V \in n \times n$  unitary matrices, such that:

$$A = U \Sigma V^T$$

#### Matrix calculus

□ **Gradient** – Let  $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$  be a function and  $A \in \mathbb{R}^{m \times n}$  be a matrix. The gradient of  $f$  with respect to  $A$  is a  $m \times n$  matrix, noted  $\nabla_A f(A)$ , such that:

$$\left( \nabla_A f(A) \right)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}$$

*Remark: the gradient of  $f$  is only defined when  $f$  is a function that returns a scalar.*

□ **Hessian** – Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a function and  $x \in \mathbb{R}^n$  be a vector. The hessian of  $f$  with respect to  $x$  is a  $n \times n$  symmetric matrix, noted  $\nabla_x^2 f(x)$ , such that:

$$\left( \nabla_x^2 f(x) \right)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

*Remark: the hessian of  $f$  is only defined when  $f$  is a function that returns a scalar.*

□ **Gradient operations** – For matrices  $A, B, C$ , the following gradient properties are worth having in mind:

$$\nabla_A \text{tr}(AB) = B^T$$

$$\nabla_A \text{tr}(f(A)) = (\nabla_A f(A))^T$$

$$\nabla_A \text{tr}(AB^T C) = CAB + C^T AB^T$$

$$\nabla_A |A| = |A|(A^{-1})^T$$

## • Refresher Probabilities and Statistics (PDF)

### VIP Refresher: Probabilities and Statistics

Afshine AMIDI and Shervine AMIDI

August 6, 2018

#### Introduction to Probability and Combinatorics

**Sample space** – The set of all possible outcomes of an experiment is known as the sample space of the experiment and is denoted by  $S$ .

**Event** – Any subset  $E$  of the sample space is known as an event. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in  $E$ , then we say that  $E$  has occurred.

**Axioms of probability** – For each event  $E$ , we denote  $P(E)$  as the probability of event  $E$  occurring. By noting  $E_1, \dots, E_n$  – mutually exclusive events, we have the 3 following axioms:

$$(1) \quad 0 \leq P(E) \leq 1 \quad (2) \quad P(S) = 1 \quad (3) \quad P\left(\bigcup_{i=1}^n E_i\right) = \sum_{i=1}^n P(E_i)$$

**Permutation** – A permutation is an arrangement of  $r$  objects from a pool of  $n$  objects, in a given order. The number of such arrangements is given by  $P(n, r)$ , defined as:

$$P(n, r) = \frac{n!}{(n-r)!}$$

**Combination** – A combination is an arrangement of  $r$  objects from a pool of  $n$  objects, where the order does not matter. The number of such arrangements is given by  $C(n, r)$ , defined as:

$$C(n, r) = \frac{P(n, r)}{r!} = \frac{n!}{r!(n-r)!}$$

Remark: we note that for  $0 \leq r \leq n$ , we have  $P(n, r) \geq C(n, r)$ .

#### Conditional Probability

**Bayes' rule** – For events  $A$  and  $B$  such that  $P(B) > 0$ , we have:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Remark: we have  $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$ .

**Partition** – Let  $\{A_i, i \in [1, n]\}$  be such that for all  $i$ ,  $A_i \neq \emptyset$ . We say that  $\{A_i\}$  is a partition if we have:

$$\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^n A_i = S$$

Remark: for any event  $B$  in the sample space, we have  $P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$ .

**Extended form of Bayes' rule** – Let  $\{A_i, i \in [1, n]\}$  be a partition of the sample space. We have:

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^n P(B|A_i)P(A_i)}$$

**Independence** – Two events  $A$  and  $B$  are independent if and only if we have:

$$P(A \cap B) = P(A)P(B)$$

#### Random Variables

**Random variable** – A random variable, often noted  $X$ , is a function that maps every element in a sample space to a real line.

**Cumulative distribution function (CDF)** – The cumulative distribution function  $F$ , which is monotonically non-decreasing and is such that  $\lim_{x \rightarrow -\infty} F(x) = 0$  and  $\lim_{x \rightarrow +\infty} F(x) = 1$ , is defined as:

$$F(x) = P(X \leq x)$$

Remark: we have  $P(a < X \leq b) = F(b) - F(a)$ .

**Probability density function (PDF)** – The probability density function  $f$  is the probability that  $X$  takes on values between two adjacent realizations of the random variable.

**Relationships involving the PDF and CDF** – Here are the important properties to know in the discrete (D) and the continuous (C) cases:

Case	CDF $F$	PDF $f$	Properties of PDF
(D)	$F(x) = \sum_{x_i \leq x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leq f(x_j) \leq 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^x f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geq 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

**Variance** – The variance of a random variable, often noted  $\text{Var}(X)$  or  $\sigma^2$ , is a measure of the spread of its distribution function. It is determined as follows:

$$\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

**Standard deviation** – The standard deviation of a random variable, often noted  $\sigma_x$ , is a measure of the spread of its distribution function which is compatible with the units of the actual random variable. It is determined as follows:

$$\sigma = \sqrt{\text{Var}(X)}$$

**Expectation and Moments of the Distribution** – Here are the expressions of the expected value  $E[X]$ , generalized expected value  $E[g(X)]$ ,  $k^{th}$  moment  $E[X^k]$  and characteristic function  $\psi(\omega)$  for the discrete and continuous cases:

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	$\psi(\omega)$
(D)	$\sum_{i=1}^n x_i f(x_i)$	$\sum_{i=1}^n g(x_i) f(x_i)$	$\sum_{i=1}^n x_i^k f(x_i)$	$\sum_{i=1}^n f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} xf(x)dx$	$\int_{-\infty}^{+\infty} g(x)f(x)dx$	$\int_{-\infty}^{+\infty} x^k f(x)dx$	$\int_{-\infty}^{+\infty} f(x)e^{i\omega x} dx$

Remark: we have  $e^{i\omega x} = \cos(\omega x) + i\sin(\omega x)$ .

**Revisiting the  $k^{th}$  moment** – The  $k^{th}$  moment can also be computed with the characteristic function as follows:

$$E[X^k] = \frac{1}{i^k} \left[ \frac{\partial^k \psi}{\partial \omega^k} \right]_{\omega=0}$$

**Transformation of random variables** – Let the variables  $X$  and  $Y$  be linked by some function. By noting  $f_X$  and  $f_Y$  the distribution function of  $X$  and  $Y$  respectively, we have:

$$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|$$

**Leibniz integral rule** – Let  $g$  be a function of  $x$  and potentially  $c$ , and  $a, b$  boundaries that may depend on  $c$ . We have:

$$\frac{\partial}{\partial c} \left( \int_a^b g(x)dx \right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x)dx$$

**Chebyshev's inequality** – Let  $X$  be a random variable with expected value  $\mu$  and standard deviation  $\sigma$ . For  $k, \sigma > 0$ , we have the following inequality:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

### Jointly Distributed Random Variables

**Conditional density** – The conditional density of  $X$  with respect to  $Y$ , often noted  $f_{X|Y}$ , is defined as follows:

$$f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}$$

**Independence** – Two random variables  $X$  and  $Y$  are said to be independent if we have:

$$f_{XY}(x,y) = f_X(x)f_Y(y)$$

**Marginal density and cumulative distribution** – From the joint density probability function  $f_{XY}$ , we have:

Case	Marginal density	Cumulative function
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i, y_j)$	$F_{XY}(x,y) = \sum_{x_i \leq x} \sum_{y_j \leq y} f_{XY}(x_i, y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y)dy$	$F_{XY}(x,y) = \int_{-\infty}^x \int_{-\infty}^y f_{XY}(x',y')dx'dy'$

**Distribution of a sum of independent random variables** – Let  $Y = X_1 + \dots + X_n$  with  $X_1, \dots, X_n$  independent. We have:

$$\psi_Y(\omega) = \prod_{k=1}^n \psi_{X_k}(\omega)$$

**Covariance** – We define the covariance of two random variables  $X$  and  $Y$ , that we note  $\sigma_{XY}^2$  or more commonly  $\text{Cov}(X,Y)$ , as follows:

$$\text{Cov}(X,Y) \triangleq \sigma_{XY}^2 = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y$$

**Correlation** – By noting  $\sigma_X, \sigma_Y$  the standard deviations of  $X$  and  $Y$ , we define the correlation between the random variables  $X$  and  $Y$ , noted  $\rho_{XY}$ , as follows:

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y}$$

Remarks: For any  $X, Y$ , we have  $\rho_{XY} \in [-1,1]$ . If  $X$  and  $Y$  are independent, then  $\rho_{XY} = 0$ .

**Main distributions** – Here are the main distributions to have in mind:

Type	Distribution	PDF	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n, p)$ Binomial	$P(X=x) = \binom{n}{x} p^x q^{n-x}$ $x \in [0, n]$	$(pe^{i\omega} + q)^n$	$np$	$npq$
	$X \sim \text{Po}(\mu)$ Poisson	$P(X=x) = \frac{\mu^x}{x!} e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	$\mu$	$\mu$
(C)	$X \sim \mathcal{U}(a, b)$ Uniform	$f(x) = \frac{1}{b-a}$ $x \in [a, b]$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu, \sigma)$ Gaussian	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \left( \frac{x-\mu}{\sigma} \right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	$\mu$	$\sigma^2$
	$X \sim \text{Exp}(\lambda)$ Exponential	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

### Parameter estimation

**Random sample** – A random sample is a collection of  $n$  random variables  $X_1, \dots, X_n$  that are independent and identically distributed with  $X$ .

**Estimator** – An estimator  $\hat{\theta}$  is a function of the data that is used to infer the value of an unknown parameter  $\theta$  in a statistical model.

**Bias** – The bias of an estimator  $\hat{\theta}$  is defined as being the difference between the expected value of the distribution of  $\hat{\theta}$  and the true value, i.e.:

$$\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta$$

Remark: an estimator is said to be unbiased when we have  $E[\hat{\theta}] = \theta$ .

**Sample mean and variance** – The sample mean and the sample variance of a random sample are used to estimate the true mean  $\mu$  and the true variance  $\sigma^2$  of a distribution, are noted  $\bar{X}$  and  $s^2$  respectively, and are such that:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad \text{and} \quad s^2 = \hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

**Central Limit Theorem** – Let us have a random sample  $X_1, \dots, X_n$  following a given distribution with mean  $\mu$  and variance  $\sigma^2$ , then we have:

$$\bar{X}_{n \rightarrow +\infty} \sim \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

## Fundamentals of Probabilities (PDF)

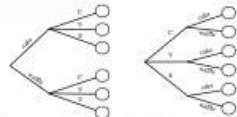
### Probability Cheatsheet v2.0

Compiled by William Chen (<http://wzchen.com>) and Joe Blitzstein, with contributions from Sebastian Chan, Yuan Jiang, Ying Han, and others. Material based on Joe Blitzstein's [Stat110](http://stat110.net) lectures (<http://stat110.net>) and Blitzstein's Introduction to Probability textbook (<http://bit.ly/introprobability>). Licensed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nd/4.0/). Please share comments, suggestions, and errors at [http://github.com/wzchen/probability\\_cheatsheet](http://github.com/wzchen/probability_cheatsheet).

Last Updated September 4, 2015

### Counting

#### Multiplication Rule



Let's say we have a compound experiment (an experiment with multiple components). If the 1st component has  $n_1$  possible outcomes, the 2nd component has  $n_2$  possible outcomes, ..., and the  $r$ th component has  $n_r$  possible outcomes, then overall there are  $n_1 n_2 \dots n_r$  possibilities for the whole experiment.

#### Sampling Table



The sampling table gives the number of possible samples of size  $k$  out of a population of size  $n$ , under various assumptions about how the sample is collected.

	Order Matters	Not Matter
With Replacement	$n^k$	$\binom{n+k-1}{k}$
Without Replacement	$n!$	$\binom{n}{k}$

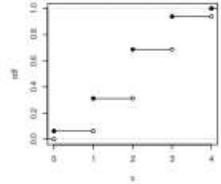
#### Naive Definition of Probability

If all outcomes are equally likely, the probability of an event  $A$  happening is:

$$P_{\text{naive}}(A) = \frac{\text{number of outcomes favorable to } A}{\text{number of outcomes}}$$

**Cumulative Distribution Function (CDF)** Gives the probability that a random variable is less than or equal to  $x$ :

$$F_X(x) = P(X \leq x)$$



The CDF is an increasing, right-continuous function with  $F_X(x) \rightarrow 0$  as  $x \rightarrow -\infty$  and  $F_X(x) \rightarrow 1$  as  $x \rightarrow \infty$ .

**Independence** Intuitively, two random variables are independent if knowing the value of one gives no information about the other.

Discrete r.v.s  $X$  and  $Y$  are independent if for all values of  $x$  and  $y$

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

#### Expected Value and Indicators

##### Expected Value and Linearity

**Expected Value** (a.k.a. mean, expectation, or average) is a weighted average of the possible outcomes of our random variable. Mathematically, if  $x_1, x_2, x_3, \dots$  are all of the distinct possible values that  $X$  can take, the expected value of  $X$  is

$$E(X) = \sum_i x_i P(X = x_i)$$

X	F	X + F	X + F
1	4	7	7
2	2	6	6
3	8	10	10
4	0	10	10
5	-1	-2	-2
6	-1	-1	-1
7	0	1	1
8	1	5	5

**Linearity** For any r.v.s  $X$  and  $Y$ , and constants  $a, b, c$ ,

$$E(aX + bY + c) = aE(X) + bE(Y) + c$$

**Same distribution implies same mean** If  $X$  and  $Y$  have the same distribution, then  $E(X) = E(Y)$  and, more generally,

$$E(g(X)) = E(g(Y))$$

**Conditional Expected Value** is defined like expectation, only conditioned on any event  $A$ .

$$E(X|A) = \sum_x x P(X = x|A)$$

### Fundamentals of Probabilities (PDF)

### Thinking Conditionally

#### Independence

**Independent Events**  $A$  and  $B$  are independent if knowing whether  $A$  occurred gives no information about whether  $B$  occurred. More formally,  $A$  and  $B$  (which have nonzero probability) are independent if and only if one of the following equivalent statements holds:

$$\begin{aligned} P(A \cap B) &= P(A)P(B) \\ P(A|B) &= P(A) \\ P(B|A) &= P(B) \end{aligned}$$

**Conditional Independence**  $A$  and  $B$  are conditionally independent given  $C$  if  $P(A \cap B|C) = P(A|C)P(B|C)$ . Conditional independence does not imply independence, and independence does not imply conditional independence.

#### Unions, Intersections, and Complements

**De Morgan's Laws** A useful identity that can make calculating probabilities of unions easier by relating them to intersections, and vice versa. Analogous results hold with more than two sets.

$$\begin{aligned} (A \cup B)^c &= A^c \cap B^c \\ (A \cap B)^c &= A^c \cup B^c \end{aligned}$$

#### Joint, Marginal, and Conditional

**Joint Probability**  $P(A \cap B)$  or  $P(A, B)$  – Probability of  $A$  and  $B$ .

**Marginal (Unconditional) Probability**  $P(A)$  – Probability of  $A$ .

**Conditional Probability**  $P(A|B) = P(A, B)/P(B)$  – Probability of  $A$ , given that  $B$  occurred.

**Conditional Probability vs Probability**  $P(A|B)$  is a probability function for any fixed  $B$ . Any theorem that holds for probability also holds for conditional probability.

#### Probability of an Intersection or Union

##### Intersections via Conditioning

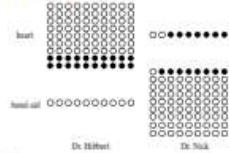
$$P(A, B) = P(A)P(B|A)$$

$$P(A, B, C) = P(A)P(B|A)P(C|A, B)$$

##### Unions via Inclusion-Exclusion

$$\begin{aligned} P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\ P(A \cup B \cup C) &= P(A) + P(B) + P(C) \\ &\quad - P(A \cap B) - P(A \cap C) - P(B \cap C) \\ &\quad + P(A \cap B \cap C). \end{aligned}$$

#### Simpson's Paradox



It is possible to have

$$P(A \mid B, C) < P(A \mid B^c, C) \text{ and } P(A \mid B, C^c) < P(A \mid B^c, C^c)$$

yet also  $P(A \mid B) > P(A \mid B^c)$ .

#### Indicator Random Variables

**Indicator Random Variable** is a random variable that takes on the value 1 or 0. It is always an indicator of some event: if the event occurs, the indicator is 1; otherwise it is 0. They are useful for many problems about counting how many events of some kind occur. Write

$$I_A = \begin{cases} 1 & \text{if } A \text{ occurs}, \\ 0 & \text{if } A \text{ does not occur}. \end{cases}$$

Note that  $I_A^2 = I_A$ ,  $I_A I_B = I_{A \cap B}$ , and  $I_{A \cup B} = I_A + I_B - I_A I_B$ .

**Distribution**  $I_A \sim \text{Bern}(p)$  where  $p = P(A)$ .

**Conditional Bridge** The expectation of the indicator for event  $A$  is the probability of event  $A$ :  $E(I_A) = P(A)$ .

#### Variance and Standard Deviation

$$\text{Var}(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$$

$$\text{SD}(X) = \sqrt{\text{Var}(X)}$$

#### Continuous RVs, LOTUS, UoU

#### Continuous Random Variables (CRVs)

What's the probability that a CRV is in an interval? Take the difference in CDF values (or use the PDF as described later).

$$P(a \leq X \leq b) = P(X \leq b) - P(X \leq a) = F_X(b) - F_X(a)$$

For  $X \sim \mathcal{N}(\mu, \sigma^2)$ , this becomes

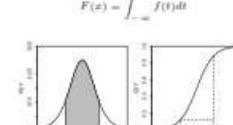
$$P(a \leq X \leq b) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)$$

What is the Probability Density Function (PDF)? The PDF  $f$  is the derivative of the CDF  $F$ :

$$F'(x) = f(x)$$

A PDF is nonnegative and integrates to 1. By the fundamental theorem of calculus, to get from PDF back to CDF we can integrate:

$$F(x) = \int_{-\infty}^x f(t)dt$$



To find the probability that a CRV takes on a value in an interval, integrate the PDF over that interval:

$$F(b) - F(a) = \int_a^b f(x)dx$$

How do I find the expected value of a CRV? Analogous to the discrete case, where you sum  $x$  times the PMF, for CRVs you integrate  $x$  times the PDF.

$$E(X) = \int_{-\infty}^{\infty} x f(x)dx$$

**Some distribution implies same mean** If  $X$  and  $Y$  have the same distribution, then  $E(X) = E(Y)$  and, more generally,

$$E(g(X)) = E(g(Y))$$

**Conditional Expected Value** is defined like expectation, only conditioned on any event  $A$ .

$$E(X|A) = \sum_x x P(X = x|A)$$

#### Law of Total Probability (LOTTP)

Let  $B_1, B_2, \dots, B_n$  be a partition of the sample space (i.e., they are disjoint and their union is the entire sample space).

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \dots + P(A|B_n)P(B_n)$$

For **LOTTP with extra conditioning**, just add in another event  $C$ :

$$P(A|C) = P(A|B_1, C)P(B_1|C) + \dots + P(A|B_n, C)P(B_n|C)$$

$$P(A|C) = P(A \cap B_1|C) + P(A \cap B_2|C) + \dots + P(A \cap B_n|C)$$

Special case of LOTTP with  $B$  and  $B^c$  as partition:

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$

$$P(A) = P(A \cap B) + P(A \cap B^c)$$

#### Bayes' Rule

**Bayes' Rule, and with extra conditioning (just add in  $C$ )**

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(A|B, C) = \frac{P(B|A, C)P(A|C)}{P(B|C)}$$

We can also write

$$P(A|B, C) = \frac{P(A, B, C)}{P(B, C)} = \frac{P(B|A, C)P(A|C)}{P(B|C)}$$

Odds Form of Bayes' Rule

$$\frac{P(A|B)}{P(A^c|B)} = \frac{P(B|A)}{P(B|A^c)}$$

The posterior odds of  $A$  are the likelihood ratio times the prior odds.

#### Random Variables and their Distributions

##### PMF, CDF, and Independence

**Probability Mass Function (PMF)** Gives the probability that a discrete random variable takes on the value  $x$ .

$$p_X(x) = P(X = x)$$

The PMF satisfies

$$p_X(x) \geq 0 \text{ and } \sum_x p_X(x) = 1$$

#### LOTUS

**Expected value of a function of an r.v.** The expected value of  $X$  is defined this way:

$$E(X) = \sum_x xP(X = x) \text{ (for discrete } X)$$

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx \text{ (for continuous } X)$$

**The Law of the Unconscious Statistician (LOTUS)** states that you can find the expected value of a function of a random variable,  $g(X)$ , in a similar way, by replacing the  $x$  in front of the PMF/PDF by  $g(X)$  but still working with the PMF/PDF of  $X$ :

$$E(g(X)) = \sum_x g(x)p(X = x) \text{ (for discrete } X)$$

$$E(g(X)) = \int_{-\infty}^{\infty} g(x)f(x)dx \text{ (for continuous } X)$$

**What's a function of a random variable?** A function of a random variable is also a random variable. For example, if  $X$  is the number of bikes you see in an hour, then  $g(X) = 2X$  is the number of bike wheels you see in that hour and  $h(X) = \binom{X}{2}$  is the number of pairs of bikes such that you see both of those bikes in that hour:

**What's the point?** You don't need to know the PMF/PDF of  $g(X)$  to find its expected value. All you need is the PMF/PDF of  $X$ .

#### Universality of Uniform (UoU)

When you plug any CRV into its own CDF, you get a Uniform(0,1) random variable. When you plug a Uniform(0,1) r.v. into an inverse CDF, you get an r.v. with that CDF. For example, let's say that a random variable  $X$  has CDF

$$F(x) = 1 - e^{-x}, \text{ for } x > 0$$

By UoU, if we plug  $X$  into this function then we get a uniformly distributed random variable.

$$F(X) = 1 - e^{-X} \sim \text{Unif}(0, 1)$$

Similarly, if  $U \sim \text{Unif}(0, 1)$  then  $F^{-1}(U)$  has CDF  $F$ . The key point is that for any continuous random variable  $X$ , we can transform it into a Uniform random variable and back by using its CDF.

#### Moments and MGFs

##### Moments

Moments describe the shape of a distribution. Let  $X$  have mean  $\mu$  and standard deviation  $\sigma$ , and  $Z = (X - \mu)/\sigma$  be the *standardized version* of  $X$ . The  $k$ th moment of  $X$  is  $\mu_k = E(X^k)$  and the  $k$ th standardized moment of  $X$  is  $\mu_k = E(Z^k)$ . The mean, variance, skewness, and kurtosis are important summaries of the shape of a distribution.

**Mean**  $E(X) = \mu_1$

**Variance**  $\text{Var}(X) = \mu_2 - \mu_1^2$

**Skewness**  $\text{Skew}(X) = \mu_3$

**Kurtosis**  $\text{Kurt}(X) = \mu_4 - 3$

### Moment Generating Functions

**MGF** For any random variable  $X$ , the function

$$M_X(t) = E(e^{tX})$$

is the **moment generating function (MGF)** of  $X$ , if it exists for all  $t$  in some open interval containing  $0$ . The variable  $t$  could just as well have been called  $a$  or  $v$ . It's a bookkeeping device that lets us work with the function  $M_X$  rather than the **moment of summation**.

**Why is it called the Moment Generating Function?** Because the  $k$ th derivative of the moment generating function, evaluated at 0, is the  $k$ th moment of  $X$ .

$$\mu_k = E(X^k) = M_X^{(k)}(0)$$

This is true by Taylor expansion of  $e^{tX}$  since

$$M_X(t) = E(e^{tX}) = \sum_{k=0}^{\infty} \frac{E(X^k)t^k}{k!} = \sum_{k=0}^{\infty} \frac{\mu_k t^k}{k!}$$

**MGF of linear functions** If we have  $Y = aX + b$ , then

$$M_Y(t) = E(e^{t(Y+aX+b)}) = e^{tb} E(e^{taX}) = e^{tb} M_X(at)$$

**Uniqueness** If it exists, the MGF uniquely determines the distribution. This means that for any two random variables  $X$  and  $Y$ , they are distributed the same (their PMFs/PDFs are equal) if and only if their MGFs are equal.

**Summing Independent RVs by Multiplying MGFs.** If  $X$  and  $Y$  are independent, then

$$M_{X+Y}(t) = E(e^{t(X+Y)}) = E(e^{tX})E(e^{tY}) = M_X(t) \cdot M_Y(t)$$

The MGF of the sum of two random variables is the product of the MGFs of those two random variables.

### Joint PDFs and CDFs

#### Joint Distributions

The joint CDF of  $X$  and  $Y$  is

$$F(x, y) = P(X \leq x, Y \leq y)$$

In the discrete case,  $X$  and  $Y$  have a **joint PMF**

$$p_{X,Y}(x, y) = P(X=x, Y=y).$$

In the continuous case, they have a **joint PDF**

$$f_{X,Y}(x, y) = \frac{\partial^2}{\partial x \partial y} F_{X,Y}(x, y).$$

The joint PMF/PDF must be nonnegative and sum/integrate to 1.



#### Conditional Distributions

**Conditioning and Bayes' rule for discrete r.v.s**

$$P(Y=y|X=x) = \frac{P(X=x, Y=y)}{P(X=x)} = \frac{P(X=x|Y=y)P(Y=y)}{P(X=x)}$$

**Conditioning and Bayes' rule for continuous r.v.s**

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x,y)}{f_X(x)} = \frac{f_{X,Y}(x|y)f_Y(y)}{f_X(x)}$$

**Hybrid Bayes' rule**

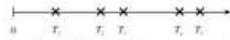
$$f_{X|A}(x|A) = \frac{P(A|X=x)f_X(x)}{P(A)}$$

#### Poisson Process

**Definition** We have a **Poisson process** of rate  $\lambda$  arrivals per unit time if the following conditions hold:

1. The number of arrivals in a time interval of length  $t$  is  $\text{Pois}(\lambda t)$ .
2. Numbers of arrivals in disjoint time intervals are independent.

For example, the numbers of arrivals in the time intervals  $[0, 5]$ ,  $(5, 12]$ , and  $(13, 23]$  are independent with  $\text{Pois}(5\lambda)$ ,  $\text{Pois}(7\lambda)$ ,  $\text{Pois}(10\lambda)$  distributions, respectively.



**Count-Time Duality** Consider a Poisson process of emails arriving in an inbox at rate  $\lambda$  emails per hour. Let  $T_n$  be the time of arrival of the  $n$ th email (relative to some starting time 0) and  $N_t$  be the number of emails that arrive in  $[0, t]$ . Let's find the distribution of  $T_1$ . The event  $T_1 > t$ , the event that you have waited more than  $t$  hours to get the first email, is the same as the event  $N_t = 0$ , which is the event that there are no emails in the first  $t$  hours. So,

$$P(T_1 > t) = P(N_t = 0) = e^{-\lambda t} \longrightarrow P(T_1 \leq t) = 1 - e^{-\lambda t}$$

Thus we have  $T_1 \sim \text{Expo}(\lambda)$ . By the memoryless property and similar reasoning, the interarrival times between emails are i.i.d.  $\text{Expo}(\lambda)$ , i.e., the differences  $T_n - T_{n-1}$  are i.i.d.  $\text{Expo}(\lambda)$ .

#### Order Statistics

**Definition** Let's say you have  $n$  i.i.d. r.v.s  $X_1, X_2, \dots, X_n$ . If you arrange them from smallest to largest, the  $i$ th element in that list is the  $i$ th order statistic, denoted  $X_{(i)}$ . So  $X_{(1)}$  is the smallest in the list and  $X_{(n)}$  is the largest in the list.

Note that the order statistics are **dependent**, e.g., learning  $X_{(1)} = 42$  gives us the information that  $X_{(1)}, X_{(2)}, X_{(3)}$  are  $\geq 42$  and  $X_{(5)}, X_{(6)}, \dots, X_{(n)}$  are  $\geq 42$ .

**Distribution** Taking  $n$  i.i.d. random variables  $X_1, X_2, \dots, X_n$  with CDF  $F(x)$  and PDF  $f(x)$ , the CDF and PDF of  $X_{(i)}$  are:

$$F_{X_{(i)}}(x) = P(X_{(i)} \leq x) = \sum_{k=i}^n \binom{n}{k} F(x)^k (1 - F(x))^{n-k}$$

$$f_{X_{(i)}}(x) = n \binom{n-1}{i-1} F(x)^{i-1} (1 - F(x))^{n-i} f(x)$$

**Uniform Order Statistics** The  $j$ th order statistic of  $i.i.d.$   $U_1, \dots, U_n \sim \text{Unif}(0, 1)$  is  $U_{(j)} \sim \text{Beta}(j, n-j+1)$ .

#### Conditional Expectation

**Conditioning on an Event** We can find  $E(Y|A)$ , the expected value of  $Y$  given that event  $A$  occurred. A very important case is when  $A$  is the event  $X = x$ . Note that  $E(Y|A)$  is a **number**. For example:

- The expected value of a fair die roll, given that it is prime, is  $\frac{1}{2} \cdot 2 + \frac{1}{3} \cdot 3 + \frac{1}{6} \cdot 5 = \frac{43}{12}$ .
- Let  $V$  be the number of successes in 10 independent Bernoulli trials with probability  $p$  of success. Let  $A$  be the event that the first 3 trials are all successes. Then

$$E(Y|A) = 3 + 7p$$

since the number of successes among the last 7 trials is  $\text{Bin}(7, p)$ .

### Marginal Distributions

To find the distribution of one (or more) random variables from a joint PMF/PDF, sum/integrate over the unwanted random variables.

#### Marginal PMF from joint PMF

$$P(X=x) = \sum_y P(X=x, Y=y)$$

#### Marginal PDF from joint PDF

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y) dy$$

### Independence of Random Variables

Random variables  $X$  and  $Y$  are independent if and only if any of the following conditions holds:

- Joint CDF is the product of the marginal CDFs
- Joint PMF/PDF is the product of the marginal PMFs/PDFs
- Conditional distribution of  $Y$  given  $X$  is the marginal distribution of  $Y$

Write  $X \perp\!\!\!\perp Y$  to denote that  $X$  and  $Y$  are independent.

### Multivariate LOTUS

LOTUS in more than one dimension is analogous to the 1D LOTUS.

For discrete random variables:

$$E(g(X, Y)) = \sum_x \sum_y g(x, y) P(X=x, Y=y)$$

For continuous random variables:

$$E(g(X, Y)) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) f_{X,Y}(x, y) dx dy$$

### Covariance and Transformations

#### Covariance and Correlation

**Covariance** is the analog of variance for two random variables.

$$\text{Cov}(X, Y) = E((X - E(X))(Y - E(Y))) = E(XY) - E(X)E(Y)$$

Note that

$$\text{Cov}(X, X) = E(X^2) - (E(X))^2 = \text{Var}(X)$$

**Correlation** is a standardized version of covariance that is always between  $-1$  and  $1$ .

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}}$$

**Covariance and Independence** If two random variables are independent, then they are uncorrelated. The converse is not necessarily true (e.g., consider  $X \sim \mathcal{N}(0, 1)$  and  $Y = X^2$ ).

$$X \perp\!\!\!\perp Y \implies \text{Cov}(X, Y) = 0 \implies E(XY) = E(X)E(Y)$$

**Covariance and Variance** The variance of a sum can be found by

$$\text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$$

$$\text{Var}(X_1 + X_2 + \dots + X_n) = \sum_{i=1}^n \text{Var}(X_i) + 2 \sum_{i < j} \text{Cov}(X_i, X_j)$$

If  $X$  and  $Y$  are independent then they have covariance 0, so

$$X \perp\!\!\!\perp Y \implies \text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y)$$

If  $X_1, X_2, \dots, X_n$  are identically distributed and have the same covariance relationships (often by **symmetry**), then

$$\text{Var}(X_1 + X_2 + \dots + X_n) = n\text{Var}(X_1) + 2 \binom{n}{2} \text{Cov}(X_1, X_2)$$

- Let  $T \sim \text{Exp}(1/10)$  be how long you have to wait until the shuttle comes. Given that you have already waited  $t$  minutes, the expected additional waiting time is 10 more minutes, by the memoryless property. That is,  $E(T|T > t) = t + 10$ .

#### Discrete Y

$$E(Y) = \sum_y y P(Y=y)$$

$$E[VA] = \sum_y y P(Y=y|A)$$

#### Continuous Y

$$E(Y) = \int_{-\infty}^{\infty} y f_Y(y) dy$$

$$E[VA] = \int_{-\infty}^{\infty} y f_Y(y|A) dy$$

**Conditioning on a Random Variable** We can also find  $E(Y|X)$ , the expected value of  $Y$  given the random variable  $X$ . This is a **function** of the random variable  $X$ . It is not a number except in certain special cases such as if  $X \perp\!\!\!\perp Y$ . To find  $E(Y|X)$ , find  $E(Y|X=x)$  and then plug in  $X$  for  $x$ . For example:

- If  $E(Y|X=x) = x^2 + 5x$ , then  $E(Y|X) = X^2 + 5X$ .
- Let  $Y$  be the number of successes in 10 independent Bernoulli trials with probability  $p$  of success and  $X$  be the number of successes among the first 3 trials. Then  $E(Y|X=x) = x^2$  since if we know  $X=x$  then we know  $Y=x^2$ . And  $E(X|Y=y)=0$  since if we know  $Y=y$  then we know  $X=\pm\sqrt{y}$ , with equal probability (by symmetry). So  $E(Y|X)=X^2$ ,  $E(X|Y)=0$ .

#### Properties of Conditional Expectation

1.  $E(Y|X) = E(Y)$  if  $X \perp\!\!\!\perp Y$
2.  $E(h(X)W|X) = h(X)E(W|X)$  (taking out what's known) In particular,  $E(h(X)|X) = h(X)$ .
3.  $E(E(Y|X)) = E(Y)$  (Adam's Law, a.k.a. Law of Total Expectation)

**Adam's Law (a.k.a. Law of Total Expectation)** can also be written in a way that looks analogous to LOTP. For any events  $A_1, A_2, \dots, A_n$  that partition the sample space,

$$E(Y) = E(Y|A_1)P(A_1) + \dots + E(Y|A_n)P(A_n)$$

For the special case where the partition is  $A, A^c$ , this says

$$E(Y) = E(Y|A)P(A) + E(Y|A^c)P(A^c)$$

#### Eve's Law (a.k.a. Law of Total Variance)

$$\text{Var}(Y) = E(\text{Var}(Y|X)) + \text{Var}(E(Y|X))$$

### MVN, LLN, CLT

#### Law of Large Numbers (LLN)

Let  $X_1, X_2, \dots$  be i.i.d. with mean  $\mu$ . The **sample mean** is

$$\bar{X}_n = \frac{X_1 + X_2 + \dots + X_n}{n}$$

The **Law of Large Numbers** states that as  $n \rightarrow \infty$ ,  $\bar{X}_n \rightarrow \mu$  with probability 1. For example, in flips of a coin with probability  $p$  of Heads, let  $X_i$  be the indicator of the  $i$ th flip being Heads. Then LLN says the proportion of Heads converges to  $p$  (with probability 1).

#### Markov Chains

##### Definition

A **Markov chain** is a random walk in a **state space**, which we will assume is finite, say  $\{1, 2, \dots, M\}$ . We let  $X_t$  denote which element of the state space the walk is visiting at time  $t$ . The **Markov chain** is the sequence of random variables tracking where the walk is at all points in time,  $X_0, X_1, X_2, \dots$ . By definition, a Markov chain must satisfy the **Markov property**, which says that if you want to predict where the chain will be at a future time, if we know the present state then the entire past history is irrelevant. *Given the present, the past and future are conditionally independent.* In symbols,

$$P(X_{n+1} = j | X_0 = i_0, X_1 = i_1, \dots, X_n = i) = P(X_{n+1} = j | X_n = i)$$

**State Properties**

A state is either **recurrent** or **transient**.

- If you start at a **recurrent state**, then you will always return back to that state at some point in the future. You can check-out any time you like, but you can never leave.

- Otherwise you are at a **transient state**. There is some positive probability that once you leave you will never return. You don't have to go home, but you can't stop here.

A state is either **periodic** or **aperiodic**.

- If you start at a **periodic state** of period  $k$ , then the GCD of the possible numbers of steps it would take to return back is  $k > 1$ .

- Otherwise you are at an **aperiodic state**. The GCD of the possible numbers of steps it would take to return back is 1.

### Transition Matrix

Let the state space be  $\{1, 2, \dots, M\}$ . The transition matrix  $Q$  is the  $M \times M$  matrix where element  $q_{ij}$  is the probability that the chain goes from state  $i$  to state  $j$  in one step:

$$q_{ij} = P(X_{n+1} = j | X_n = i)$$

To find the probability that the chain goes from state  $i$  to state  $j$  in exactly  $m$  steps, take the  $(i, j)$  element of  $Q^m$ :

$$q_{ij}^{(m)} = P(X_{n+m} = j | X_n = i)$$

If  $X_0$  is distributed according to the row vector PMF  $p$ , i.e.,  $p_j = P(X_0 = j)$ , then the PMF of  $X_m$  is  $pQ^m$ .

### Chain Properties

A chain is **irreducible** if you can get from anywhere to anywhere. If a chain (on a finite state space) is irreducible, then all of its states are recurrent. A chain is **periodic** if any of its states are periodic, and is **aperiodic** if none of its states are periodic. In an irreducible chain, all states have the same period.

A chain is **reversible** with respect to  $\vec{s}$  if  $s_i q_{ij} = s_j q_{ji}$  for all  $i, j$ . Examples of reversible chains include any chain with  $q_{ii} = q_{jj}$ , with  $\vec{s} = (\frac{1}{M}, \frac{1}{M}, \dots, \frac{1}{M})$ , and random walks on an undirected network.

### Stationary Distribution

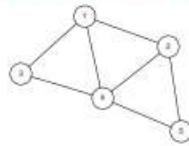
Let us say that the vector  $\vec{s} = (s_1, s_2, \dots, s_M)$  be a PMF (written as a row vector). We will call it the **stationary distribution** for the chain if  $\vec{s}Q = \vec{s}$ . As a consequence, if  $X_0$  has the stationary distribution, then all future  $X_{t+1}, X_{t+2}, \dots$  also have the stationary distribution.

For irreducible, aperiodic chains, the stationary distribution exists, is unique, and  $s_i$  is the long-run probability of a chain being at state  $i$ . The expected number of steps to return to  $i$  starting from  $i$  is  $1/s_i$ .

To find the stationary distribution, you can solve the matrix equation  $(Q^T - I)\vec{s}' = 0$ . The stationary distribution is uniform if the columns of  $Q$  sum to 1.

**Reversibility Condition Implies Stationarity** If you have a PMF  $\vec{s}$  and a Markov chain with transition matrix  $Q$ , then  $s_i q_{ij} = s_j q_{ji}$  for all states  $i, j$  implies that  $\vec{s}$  is stationary.

### Random Walk on an Undirected Network



If you have a collection of **nodes**, pairs of which can be connected by undirected **edges**, and a Markov chain is run by going from the current node to a uniformly random node that is connected to it by an edge, then this is a random walk on an undirected network. The stationary distribution of this chain is proportional to the **degree sequence** (this is the sequence of degrees, where the degree of a node is how many edges are attached to it). For example, the stationary distribution of random walks on the network shown above is proportional to  $(3, 3, 2, 4, 2)$ , as it's  $(\frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12})$ .

- $\text{Bin}(\frac{n}{M}, p) \sim \text{Beta}(a, b)$
- $X + Y \stackrel{\text{iid}}{\sim} \text{Bin}(\frac{n}{M}, p)$

This is known as the **bank-post office result**.

### $\chi^2$ (Chi-Square) Distribution

Let us say that  $X$  is distributed  $\chi_n^2$ . We know the following:

**Story** A Chi-Square( $n$ ) is the sum of the squares of  $n$  independent standard Normal r.v.s.

#### Properties and Representations

- $X$  is distributed as  $Z_1^2 + Z_2^2 + \dots + Z_n^2$  for i.i.d.  $Z_i \sim \mathcal{N}(0, 1)$   
 $X \sim \text{Gamma}(n/2, 1/2)$

### Discrete Distributions

#### Distributions for four sampling schemes

	Replace	No Replace
Fixed # trials ( $n$ )	Binomial (Born if $n = 1$ )	HGeom
Draw until $r$ success	NBin (Geom if $r = 1$ )	NHGeom

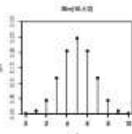
### Bernoulli Distribution

The Bernoulli distribution is the simplest case of the Binomial distribution, where we only have one trial ( $n = 1$ ). Let us say that  $X$  is distributed  $\text{Bern}(p)$ . We know the following:

**Story** A trial is performed with probability  $p$  of "success", and  $X$  is the indicator of success: 1 means success, 0 means failure.

**Example** Let  $X$  be the indicator of Heads for a fair coin toss. Then  $X \sim \text{Bern}(\frac{1}{2})$ . Also,  $1 - X \sim \text{Bern}(\frac{1}{2})$  is the indicator of Tails.

### Binomial Distribution



Let us say that  $X$  is distributed  $\text{Bin}(n, p)$ . We know the following:

**Story**  $X$  is the number of "successes" that we will achieve in  $n$  independent trials, where each trial is either a success or a failure, each with the same probability  $p$  of success. We can also write  $X$  as a sum of multiple independent  $\text{Bern}(p)$  random variables. Let  $X \sim \text{Bin}(n, p)$  and  $X_i \sim \text{Bern}(p)$ , where all of the Bernoullis are independent. Then

$$X = X_1 + X_2 + X_3 + \dots + X_n$$

**Example** If Jeremy Lin makes 10 free throws and each one independently has a  $\frac{1}{2}$  chance of getting in, then the number of free throws he makes is distributed  $\text{Bin}(10, \frac{1}{2})$ .

**Properties** Let  $X \sim \text{Bin}(n, p)$ ,  $Y \sim \text{Bin}(m, p)$  with  $X \perp\!\!\!\perp Y$ .

- Redefine success  $n - X \sim \text{Bin}(n, 1 - p)$
- Sum  $X + Y \sim \text{Bin}(n + m, p)$

### Continuous Distributions

#### Uniform Distribution

Let us say that  $U$  is distributed  $\text{Unif}(a, b)$ . We know the following:

**Properties of the Uniform** For a Uniform distribution, the probability of a draw from any interval within the support is proportional to the length of the interval. See *Universality of Uniform and Order Statistics* for other properties.

**Example** William throws darts really badly, so his darts are uniform over the whole room because they're equally likely to appear anywhere. William's darts have a Uniform distribution on the surface of the room. The Uniform is the only distribution where the probability of hitting in any specific region is proportional to the length/area/volume of that region, and where the density of occurrence in any one specific spot is constant throughout the whole support.

#### Normal Distribution

Let us say that  $X$  is distributed  $\mathcal{N}(\mu, \sigma^2)$ . We know the following:

**Central Limit Theorem** The Normal distribution is ubiquitous because of the Central Limit Theorem, which states that the sample mean of i.i.d. r.v.s will approach a Normal distribution as the sample size grows, regardless of the initial distribution.

**Location-Scale Transformation** Every time we shift a Normal r.v. (by adding a constant) or rescale a Normal (by multiplying by a constant), we change it to another Normal r.v. For any Normal  $X \sim \mathcal{N}(\mu, \sigma^2)$ , we can transform it to the standard  $\mathcal{N}(0, 1)$  by the following transformation:

$$Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1)$$

**Standard Normal** The Standard Normal,  $Z \sim \mathcal{N}(0, 1)$ , has mean 0 and variance 1. Its CDF is denoted by  $\Phi$ .

#### Exponential Distribution

Let us say that  $X$  is distributed  $\text{Exp}(\lambda)$ . We know the following:

**Story** You're sitting in an armchair reading right before the break of dawn, wishing that airplanes in the night sky were shooting stars, because you could really use a wish right now. You know that shooting stars come on average every 15 minutes, but a shooting star is not "due" to come just because you've waited so long. Your waiting time is memoryless; the additional time until the next shooting star comes does not depend on how long you've waited already.

**Example** The waiting time until the next shooting star is distributed  $\text{Exp}(1)$  hours. Here  $\lambda = 1$  is the **rate parameter**, since shooting stars arrive at a rate of 1 per 1/4 hour on average. The expected time until the next shooting star is  $1/\lambda = 1/4$  hour.

#### Expo as a rescaled Exp(1)

$$Y \sim \text{Exp}(\lambda) \rightarrow X = \lambda Y \sim \text{Exp}(1)$$

**Memorylessness** The Exponential Distribution is the only continuous memoryless distribution. The memoryless property says that for  $X \sim \text{Exp}(\lambda)$  and any positive numbers  $s$  and  $t$ ,

$$P(X > s + t | X > s) = P(X > t)$$

Equivalently,

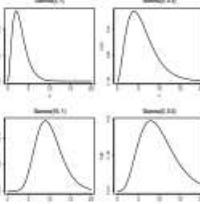
$$X - s | (X > s) \sim \text{Exp}(\lambda)$$

For example, a product with an  $\text{Exp}(\lambda)$  lifetime is always "as good as new" (it doesn't experience wear and tear). Given that the product has survived  $s$  years, the additional time that it will last is still  $\text{Exp}(\lambda)$ .

**Min of Expo** If we have independent  $X_1 \sim \text{Exp}(\lambda_1)$ , then  $\min(X_1, \dots, X_k) \sim \text{Exp}(\lambda_1 + \lambda_2 + \dots + \lambda_k)$ .

**Max of Expo** If we have i.i.d.  $X_i \sim \text{Exp}(\lambda)$ , then  $\max(X_1, \dots, X_n)$  has the same distribution as  $Y_1 + Y_2 + \dots + Y_n$ , where  $Y_j \sim \text{Exp}(\lambda)$  and the  $Y_j$  are independent.

#### Gamma Distribution

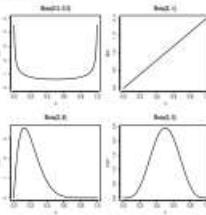


Let us say that  $X$  is distributed  $\text{Gamma}(a, \lambda)$ . We know the following:

**Story** You sit waiting for shooting stars, where the waiting time for a star is distributed  $\text{Exp}(\lambda)$ . You want to see  $n$  shooting stars before you go home. The total waiting time for the  $n$ th shooting star is  $\text{Gamma}(n, \lambda)$ .

**Example** You are at a bank, and there are 3 people ahead of you. The serving time for each person is Exponential with mean 2 minutes. Only one person at a time can be served. The distribution of your waiting time until it's your turn to be served is  $\text{Gamma}(\frac{3}{2}, \frac{1}{2})$ .

#### Beta Distribution



Let us say that  $X$  is distributed  $\text{Beta}(a, b)$ . We know the following:

**Properties of the Binomial** In the Bayesian approach to statistics, parameters are viewed as random variables, to reflect our uncertainty. The prior for a parameter is its distribution before observing data. The posterior is the distribution for the parameter after observing data. Beta is the conjugate prior of the Binomial, because if you have a Beta-distributed prior on  $p$  in a Binomial, then the posterior distribution on  $p$  given the Binomial data is also Beta-distributed. Consider the following two-level model:

$$X | p \sim \text{Bin}(n, p)$$

$$p \sim \text{Beta}(a, b)$$

Then after observing  $X = x$ , we get the posterior distribution

$$p | (X = x) \sim \text{Beta}(x + a, n - x + b)$$

**Order statistics of the Uniform** See *Order Statistics*.

**Beta-Gamma relationship** If  $X \sim \text{Gamma}(a, \lambda)$ ,  $Y \sim \text{Gamma}(b, \lambda)$ , with  $X \perp\!\!\!\perp Y$  then

**Properties** Let  $X \sim \text{Pois}(\lambda_1)$  and  $Y \sim \text{Pois}(\lambda_2)$ , with  $X \perp\!\!\!\perp Y$ .

- Sum  $X + Y \sim \text{Pois}(\lambda_1 + \lambda_2)$

- Conditional**  $X | (X + Y = n) \sim \text{Bin}\left(n, \frac{\lambda_1}{\lambda_1 + \lambda_2}\right)$

- Chicken-egg** If there are  $Z \sim \text{Pois}(\lambda)$  items and we randomly and independently "accept" each item with probability  $p$ , then the number of accepted items  $Z_1 \sim \text{Pois}(\lambda p)$ , and the number of rejected items  $Z_2 \sim \text{Pois}((1-p)\lambda)$ , and  $Z_1 \perp\!\!\!\perp Z_2$ .

### Multivariate Distributions

#### Multinomial Distribution

Let us say that the vector  $\vec{X} = (X_1, X_2, X_3, \dots, X_k) \sim \text{Mult}_k(n, \vec{p})$ , where  $\vec{p} = (p_1, p_2, \dots, p_k)$ .

**Story** We have  $n$  items, which can fall into any one of the  $k$  buckets independently with the probabilities  $\vec{p} = (p_1, p_2, \dots, p_k)$ .

**Example** Let us assume that every year, 100 students in the Harry Potter Universe are randomly and independently sorted into one of four houses with equal probability. The number of people in each of the houses is distributed  $\text{Mult}_4(100, \vec{p})$ , where  $\vec{p} = (0.25, 0.25, 0.25, 0.25)$ . Note that  $X_1 + X_2 + \dots + X_4 = 100$ , and they are dependent.

**Joint PMF** For  $n = n_1 + n_2 + \dots + n_k$ ,

$$P(\vec{X} = \vec{n}) = \frac{n!}{n_1! n_2! \dots n_k!} p_1^{n_1} p_2^{n_2} \dots p_k^{n_k}$$

**Marginal PMF, Lumping, and Conditionals Marginally**,  $X_i \sim \text{Bin}(n_i, p_i)$ , since we can define "success" to mean category  $i$ . If you lump together multiple categories in a Multinomial, then it is still Multinomial. For example,  $X_i + X_j \sim \text{Bin}(n_i + n_j, p_i + p_j)$  for  $i \neq j$ . Similarly, if  $k = 6$  and we lump categories 1-2 and lump categories 3-5, then  $(X_1 + X_2, X_3 + X_4 + X_5, X_6) \sim \text{Mult}_3(n, (p_1 + p_2, p_3 + p_4 + p_5, p_6))$ . Conditioning on some  $X_j$  also still gives a Multinomial.

**Variance and Covariances** We have  $X_i \sim \text{Bin}(n, p_i)$  marginally, so  $\text{Var}(X_i) = np_i(1 - p_i)$ . Also,  $\text{Cov}(X_i, X_j) = -np_i p_j$  for  $i \neq j$ .

#### Multivariate Uniform Distribution

See the univariate Uniform for stories and examples. For the 2D Uniform on some region, probability is proportional to area. Every point in the support has equal density, of value  $\frac{1}{\text{area of region}}$ . For the 3D Uniform, probability is proportional to volume.

#### Multivariate Normal (MVN) Distribution

A vector  $\vec{X} = (X_1, X_2, \dots, X_k)$  is Multivariate Normal if every linear combination is Normally distributed, i.e.,  $t_1 X_1 + t_2 X_2 + \dots + t_k X_k$  is Normal for any constants  $t_1, t_2, \dots, t_k$ . The parameters of the Multivariate Normal are the **mean vector**  $\vec{\mu} = (\mu_1, \mu_2, \dots, \mu_k)$ , and the **covariance matrix** where the  $(i, j)$  entry is  $\text{Cov}(X_i, X_j)$ .

**Properties** The Multivariate Normal has the following properties.

- Any subvector is also MVN.
- If any two elements within an MVN are uncorrelated, then they are independent.
- The joint PDF of a Bivariate Normal  $(X, Y)$  with  $\mathcal{N}(0, 1)$  marginal distributions and correlation  $\rho \in (-1, 1)$  is

$$f_{X,Y}(x, y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}(x^2 + y^2 - 2\rho xy)\right).$$

with  $\tau = \sqrt{1 - \rho^2}$ .

## Distribution Properties

### Important CDFs

Standard Normal  $\Phi$

Exponential( $\lambda$ )  $F(x) = 1 - e^{-\lambda x}$ , for  $x \in (0, \infty)$

Uniform(0,1)  $F(x) = x$ , for  $x \in [0, 1]$

### Convolutions of Random Variables

A convolution of  $n$  random variables is simply their sum. For the following results, let  $X$  and  $Y$  be independent.

1.  $X \sim \text{Pois}(\lambda_1)$ ,  $Y \sim \text{Pois}(\lambda_2) \rightarrow X + Y \sim \text{Pois}(\lambda_1 + \lambda_2)$
2.  $X \sim \text{Bin}(n_1, p)$ ,  $Y \sim \text{Bin}(n_2, p) \rightarrow X + Y \sim \text{Bin}(n_1 + n_2, p)$ .  $\text{Bin}(n, p)$  can be thought of as a sum of  $n$  i.i.d.  $\text{Bern}(p)$  r.v.s.
3.  $X \sim \text{Gamma}(a_1, \lambda)$ ,  $Y \sim \text{Gamma}(a_2, \lambda) \rightarrow X + Y \sim \text{Gamma}(a_1 + a_2, \lambda)$ .  $\text{Gamma}(n, \lambda)$  with  $n$  an integer can be thought of as a sum of  $n$  i.i.d.  $\text{Exp}(\lambda)$  r.v.s.
4.  $X \sim \text{NBin}(r_1, p)$ ,  $Y \sim \text{NBin}(r_2, p) \rightarrow X + Y \sim \text{NBin}(r_1 + r_2, p)$ .  $\text{NBin}(r, p)$  can be thought of as a sum of  $r$  i.i.d.  $\text{Geom}(p)$  r.v.s.
5.  $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$ ,  $Y \sim \mathcal{N}(\mu_2, \sigma_2^2) \rightarrow X + Y \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$ .

### Special Cases of Distributions

1.  $\text{Bin}(1, p) \sim \text{Bern}(p)$
2.  $\text{Beta}(1, 1) \sim \text{Unif}(0, 1)$
3.  $\text{Gamma}(1, \lambda) \sim \text{Exp}(\lambda)$
4.  $\chi_n^2 \sim \text{Gamma}\left(\frac{n}{2}, \frac{1}{2}\right)$
5.  $\text{NBin}(1, p) \sim \text{Geom}(p)$

### Inequalities

1. Cauchy-Schwarz  $E(XY) \leq \sqrt{E(X^2)E(Y^2)}$
2. Markov  $P(X \geq a) \leq \frac{E(X)}{a}$  for  $a > 0$
3. Chebyshev  $P(|X - \mu| \geq a) \leq \frac{\sigma^2}{a^2}$  for  $E(X) = \mu$ ,  $\text{Var}(X) = \sigma^2$
4. Jensen  $E(g(X)) \geq g(E(X))$  for  $g$  convex; reverse if  $g$  is concave

### Formulas

#### Geometric Series

$$1 + r + r^2 + \dots + r^{n-1} = \sum_{k=0}^{n-1} r^k = \frac{1 - r^n}{1 - r}$$

$$1 + r + r^2 + \dots = \frac{1}{1 - r} \quad \text{If } |r| < 1$$

#### Exponential Function ( $e^x$ )

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n$$

#### Gamma and Beta Integrals

You can sometimes solve complicated-looking integrals by pattern-matching to a gamma or beta integral:

$$\int_0^{\infty} x^{a-1} e^{-bx} dx = \Gamma(a) \quad \int_0^{\infty} x^{a-1} (1-x)^{b-1} dx = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Also,  $\Gamma(a+1) = a\Gamma(a)$ , and  $\Gamma(n) = (n-1)!$  if  $n$  is a positive integer.

#### Adam's Law and Eve's Law

William really likes speed-solving Rubik's Cubes. But he's pretty bad at it, so sometimes he fails. On any given day, William will attempt  $N$   $\text{Geom}(s)$  Rubik's Cubes. Suppose each time, he has probability  $p$  of solving the cube successfully. Let  $T$  be the total number of Rubik's Cubes he solves during a day. Find the mean and variance of  $T$ .

**Answer:** Note that  $T \sim \text{Bin}(N, p)$ . So by Adam's Law,

$$E(T) = E(E(T|N)) = E(Np) = \frac{p(1-s)}{s}$$

Similarly, by Eve's Law, we have that

$$\text{Var}(T) = E(\text{Var}(T|N)) + \text{Var}(E(T|N)) = E(Np(1-p)) + \text{Var}(Np)$$

$$= \frac{p(1-p)(1-s)}{s} + \frac{p^2(1-s)(p+s(1-p))}{s^2}$$

#### MGF – Finding Moments

Find  $E(X^n)$  for  $X \sim \text{Exp}(\lambda)$  using the MGF of  $X$ . **Answer:** The MGF of an  $\text{Exp}(\lambda)$  is  $M(t) = \frac{1}{1-\lambda t}$ . To get the third moment, we can take the third derivative of the MGF and evaluate at  $t = 0$ :

$$E(X^3) = \frac{6}{\lambda^3}$$

But a much nicer way to use the MGF here is via pattern recognition: note that  $M(t)$  looks like it came from a geometric series:

$$\frac{1}{1-\frac{x}{\lambda}} = \sum_{n=0}^{\infty} \left(\frac{x}{\lambda}\right)^n = \sum_{n=0}^{\infty} \frac{n!}{\lambda^n} \frac{x^n}{n!}$$

The coefficient of  $\frac{x^3}{3!}$  here is the 3rd moment of  $X$ , so we have  $E(X^3) = \frac{6}{\lambda^3}$  for all nonnegative integers  $n$ .

#### Markov chains (1)

Suppose  $X_n$  is a two-state Markov chain with transition matrix

$$Q = \begin{pmatrix} 0 & 1 \\ 1 - \alpha & \alpha \end{pmatrix}$$

Find the stationary distribution  $\vec{s} = (s_0, s_1)$  of  $X_n$  by solving  $\vec{s}Q = \vec{s}$ , and show that the chain is reversible with respect to  $\vec{s}$ . **Answer:** The equation  $s_0 = s_0(1-\alpha) + s_1\beta$  and  $s_1 = s_0(\alpha) + s_1(1-\beta)$ .

By solving this system of linear equations, we have

$$\vec{s} = \begin{pmatrix} \beta & \alpha \\ \alpha + \beta & \alpha + \beta \end{pmatrix}$$

To show that the chain is reversible with respect to  $\vec{s}$ , we must show  $s_i q_{ij} = s_j q_{ji}$  for all  $i, j$ . This is done if we can show  $s_0 q_{01} = s_1 q_{10}$ . And indeed,

$$s_0 q_{01} = \frac{\alpha\beta}{\alpha + \beta} = s_1 q_{10}$$

#### Markov chains (2)

William and Sebastian play a modified game of Settlers of Catan, where every turn they randomly move the robber (which starts on the center tile) to one of the adjacent hexagons.

## Euler's Approximation for Harmonic Sums

$$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \approx \log n + 0.577 \dots$$

## Stirling's Approximation for Factorials

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

## Miscellaneous Definitions

**Medians and Quantiles** Let  $X$  have CDF  $F$ . Then  $X$  has median  $m$  if  $F(m) \geq 0.5$  and  $F(x) \leq 0.5$ . For  $X$  continuous,  $m$  satisfies  $F(m) = 0.5$ . In general, the  $q$ th quantile of  $X$  is  $\min\{x : F(x) \geq q\}$ ; the median is the case  $q = 0.5$ .

**Orderings of i.i.d. random variables** **Example:** Since the arrival times of the five cars are i.i.d., all 120 orderings of the arrivals are equally likely. There are  $120!$  orderings that involve the Lyfts arriving first, so the probability that the Lyfts arrive first is  $\frac{120!}{120!} = 1/120$ . Alternatively, there are  $\binom{5}{3}$  ways to choose 3 of the 5 slots for the Lyfts to occupy, where each of the choices are equally likely. One of these choices has all 3 of the Lyfts arriving first, so the probability is  $\frac{1}{\binom{5}{3}} = 1/10$ .

## Linearity and First Success

This problem is commonly known as the coupon collector problem. There are  $n$  coupon types. At each draw, you get a uniformly random coupon type. What is the expected number of coupons needed until you have a complete set? **Answer:** Let  $N$  be the number of coupons needed; we want  $E(N)$ . Let  $N = N_1 + \dots + N_r$ , where  $N_i$  is the draws to get our first new coupon,  $N_2$  is the additional draws needed to draw our second new coupon and so on. By the story of the First Success,  $N_2 \sim \text{FS}(n-1/n)$  (after collecting first coupon type),  $N_3 \sim \text{FS}(n-2/n)$ , and  $N_r \sim \text{FS}(n-(r-1)/n)$ . By linearity,  $N \sim \text{FS}(n)$ .

This is approximately  $n(\log n + 0.577)$  by Euler's approximation.

## Example Problems

### Contributions from Sebastian Chiu

#### Calculating Probability

A textbook has  $n$  typos, which are randomly scattered amongst its  $n$  pages, independently. You pick a random page. What is the probability that it has no typos? **Answer:** There is a  $(1 - \frac{1}{n})^n$  probability that any specific typo isn't on your page, and thus a

$$\left(1 - \frac{1}{n}\right)^n$$

probability that there are no typos on your page. For  $n$  large, this is approximately  $e^{-1} \approx 1/e$ .

#### Linearity and Indicators (1)

In a group of  $n$  people, what is the expected number of distinct birthdays (month and day)? What is the expected number of birthday matches? **Answer:** Let  $I_j$  be the indicator for the  $j$ th day being represented.

$$E(I_j) = 1 - P(\text{no one born on day } j) = 1 - (364/365)^n$$

By linearity,  $E(X) = 365(1 - (364/365)^n)$ . Now let  $Y$  be the number of birthday matches and  $I_j$  be the indicator that the  $j$ th pair of people have the same birthday. The probability that any two specific people share a birthday is  $1/365$ , so  $E(Y) = \binom{n}{2}/365$ .

#### Linearity and Indicators (2)

This problem is commonly known as the birthday problem. There are  $n$  people at a party, each with a hat. At the end of the party, they each leave with a random hat. What is the expected number of people who leave with the right hat? **Answer:** Each hat has a  $1/n$  chance of going to the right person. By linearity, the average number of hats that go to their owners is  $n(1/n) = 1$ .

## Expectation of Negative Hypergeometric

What is the expected number of cards that you draw before you pick your first Ace in a shuffled deck (not counting the Ace)? **Answer:** Consider drawing cards one by one. Let  $I_j$  be the indicator that card  $j$  will be drawn before the first Ace. Note that  $I_j = 1$  means that  $j$  is before all 4 of the Aces in the deck. The probability that this occurs is  $1/5!$  by symmetry. Let  $X$  be the number of cards drawn before the first Ace. Then  $X = I_1 + I_2 + \dots + I_{48}$ , where each indicator corresponds to one of the 48 non-Aces. Thus,

$$E(X) = E(I_1) + E(I_2) + \dots + E(I_{48}) = 48/5 = 9.6.$$

## Minimum and Maximum of RVs

What is the CDF of the maximum of  $n$  independent  $\text{Unif}(0,1)$  random variables? **Answer:** Note that for r.v.'s  $X_1, X_2, \dots, X_n$ ,

$$P(\min(X_1, X_2, \dots, X_n) \geq a) = P(X_1 \geq a, X_2 \geq a, \dots, X_n \geq a)$$

Similarly,

$$P(\max(X_1, X_2, \dots, X_n) \leq a) = P(X_1 \leq a, X_2 \leq a, \dots, X_n \leq a).$$

We will use this principle to find the CDF of  $U_{(n)}$ , where  $U_{(n)} = \max(U_1, U_2, \dots, U_n)$  and  $U_i \sim \text{Unif}(0,1)$  are i.i.d.

$$P(\max(U_1, U_2, \dots, U_n) \leq a) = P(U_1 \leq a, U_2 \leq a, \dots, U_n \leq a) = P(U_1 \leq a)P(U_2 \leq a) \dots P(U_n \leq a) = a^n$$

for  $0 < a < 1$  (and the CDF is 0 for  $a \leq 0$  and 1 for  $a \geq 1$ ).

## Pattern-matching with $e^x$ Taylor series

For  $X \sim \text{Pois}(\lambda)$ , find  $E\left(\frac{1}{X+1}\right)$ . **Answer:** By LOTUS,

$$E\left(\frac{1}{X+1}\right) = \sum_{k=0}^{\infty} \frac{1}{k+1} \frac{e^{-\lambda} \lambda^k}{k!} = \frac{e^{-\lambda}}{\lambda} \sum_{k=0}^{\infty} \frac{\lambda^{k+1}}{(k+1)!} = \frac{e^{-\lambda}}{\lambda} (e^{\lambda} - 1)$$

**4. Calculating expectation.** If it has a named distribution, check out the table of distributions. If it's a function of an r.v. with a named distribution, try LOTUS. If it's a count of something, try breaking it up into indicator r.v.s. If you can condition on something natural, consider using Adam's Law.

**5. Calculating variance.** Consider independence, named distributions, and LOTUS. If it's a count of something, break it up into a sum of indicator r.v.s. If it's a sum, use properties of covariance. If you can condition on something unnatural, consider using Eve's Law.

**6. Calculating  $E(X^2)$ .** Do you already know  $E(X)$  or  $\text{Var}(X)$ ? Recall that  $\text{Var}(X) = E(X^2) - (E(X))^2$ . Otherwise, try LOTUS.

**7. Calculating covariance.** Use the properties of covariance. If you're trying to find the covariance between two components of a Multinomial distribution,  $X_1, X_2$ , then the covariance is  $-np_1p_2$ , for  $i \neq j$ .

**8. Symmetry.** If  $X_1, X_2, \dots, X_n$  are i.i.d., consider using symmetry.

**9. Calculating probabilities of orderings.** Remember that all  $n!$  ordering of i.i.d. continuous random variables  $X_1, X_2, \dots, X_n$  are equally likely.

**10. Determining independence.** There are several equivalent definitions. Think about simple and extreme cases to see if you can find a counterexample.

**11. Do a painful integral.** If your integral looks painful, see if you can write your integral in terms of a known PDF (like Gamma or Beta), and use the fact that PDFs integrate to 1.

**12. Before moving on.** Check some simple and extreme cases; check whether the answer seems plausible, check for bugs/zeros.

## Biohazards

Contributions from Jessie Hwang

**1. Don't misuse the naive definition of probability.** When asking "What is the probability that in a group of 3 people, no two have the same birth month?", it is not correct to treat the people as indistinguishable bulls being placed into 12 boxes, since that assumes the list of birth months (January, February, March) is just as likely as the list (January, April, June), even though the latter is six times more likely.

**2. Don't confuse unconditional, conditional, and joint probabilities.** In applying  $P(A|B) = \frac{P(B \cap A)}{P(B)}$ , it is not correct to say " $P(B) = 1$  because we know  $B$  happened";  $P(B)$  is the prior probability of  $B$ . Don't confuse  $P(A|B)$  with  $P(A, B)$ .

**3. Don't assume independence without justification.** In the matching problem, the probability that card 1 is a match and card 2 is a match is not  $1/n^2$ . Binomial and Hypergeometric are often confused; the trials are independent in the Binomial story and dependent in the Hypergeometric story.

**4. Don't forget to do sanity checks.** Probabilities must be between 0 and 1. Variances must be  $\geq 0$ . Supports must make sense. PMFs must sum to 1. PDFs must integrate to 1.

**5. Don't confuse random variables, numbers, and events.** Let  $X$  be an r.v. Then  $g(X)$  is an r.v. for any function  $g$ . In particular,  $X^2$ ,  $|X|$ ,  $F(X)$ , and  $I_{X>a}$  are r.v.s.  $P(X^2 < X | X > 0)$ ,  $E(X)$ ,  $\text{Var}(X)$ , and  $P(E(X))$  are numbers.  $X = 2$  and  $F(X) \geq -1$  are events. It does not make sense to write  $\int_{-\infty}^{\infty} F(X)dx$ , because  $F(X)$  is a random variable. It does not make sense to write  $P(X)$ , because  $X$  is not an event.

6. Don't confuse a random variable with its distribution. To get the PDF of  $X^2$ , you can't just square the PDF of  $X$ . The right way is to use transformation. To get the PDF of  $X + Y$ , you can't just add the PDF of  $X$  and the PDF of  $Y$ . The right way is to compute the convolution.

7. Don't pull non-linear functions out of expectations.  $E(g(X))$  does not equal  $g(E(X))$  in general. The St. Petersburg paradox is an extreme example. See also Jensen's inequality. The right way to find  $E(g(X))$  is with [Convolution](#).

## Recommended Resources

- Introduction to Probability Book (<http://bit.ly/introprobability>)
- Stat 110 Online (<http://stat110.net>)
- Stat 110 Online Blog (<https://stat110.quora.com/>)
- Quora Probability FAQ (<https://bit.ly/probabilityfaq>)
- R Studio (<https://www.rstudio.com>)
- LaTeX File ([https://github.com/wzchen/probability\\_cheatSheet](https://github.com/wzchen/probability_cheatSheet))

Please share this cheatsheet with friends!  
[http://wzchen.com/probability\\_cheatSheet](http://wzchen.com/probability_cheatSheet)

## Distributions in R

Command	What it does
<code>help(distributions)</code>	shows documentation on distributions
<code>dbinom(x,n,p)</code>	PMF $P(X = k)$ for $X \sim \text{Bin}(n, p)$
<code>pbinom(x,n,p)</code>	CDF $P(X \leq x)$ for $X \sim \text{Bin}(n, p)$
<code>qbinom(x,n,p)</code>	nth quantile for $X \sim \text{Bin}(n, p)$
<code>rbinom(n, size, prob)</code>	random sample from $\text{Bin}(n, p)$
<code>dgeom(x,p)</code>	PMF $P(X = k)$ for $X \sim \text{Geom}(p)$
<code>dhyper(x,w,b,n)</code>	PMF $P(X = k)$ for $X \sim \text{Hypergeom}(w, b, n)$
<code>dbinom(x,r,p)</code>	PMF $P(X = k)$ for $X \sim \text{NBin}(r, p)$
<code>dpois(x,r)</code>	PMF $P(X = k)$ for $X \sim \text{Pois}(r)$
<code>dbeta(x,a,b)</code>	PDF $f(x)$ for $X \sim \text{Beta}(a, b)$
<code>dcnt(x,n)</code>	PDF $f(x)$ for $X \sim \chi_n^2$
<code>ddexp(x,b)</code>	PDF $f(x)$ for $X \sim \text{Exp}(b)$
<code>dgamma(x,a,r)</code>	PDF $f(x)$ for $X \sim \text{Gamma}(a, r)$
<code>dlnorm(x,s,x)</code>	PDF $f(x)$ for $X \sim \mathcal{LN}(m, s^2)$
<code>dmult(x,n,s)</code>	PDF $f(x)$ for $X \sim \mathcal{N}(m, s^2)$
<code>dt(n,a)</code>	PDF $f(x)$ for $X \sim t_n$
<code>dnif(x,a,b)</code>	PDF $f(x)$ for $X \sim \text{Unif}(a, b)$

The table above gives R commands for working with various named distributions. Commands analogous to `pnorm`, `qnorm`, and `rnorm` work for the other distributions in the table. For example, `pnorm`, `qnorm`, and `rnorm` can be used to get the CDF, quantiles, and random generation for the Normal. For the Multinomial, `rmultinom` can be used for calculating the joint PMF and `rmvtnorm` can be used for generating random vectors. For the Multivariate Normal, after installing and loading the `mvtnorm` package `dmvnorm` can be used for calculating the joint PDF and `rmvnorm` can be used for generating random vectors.

## Table of Distributions

Distribution	PMF/PDF and Support	Expected Value	Variance	MGF
Bernoulli Bern( $p$ )	$P(X = 1) = p$ $P(X = 0) = q = 1 - p$	$p$	$pq$	$q + pe^t$
Binomial Bin( $n, p$ )	$P(X = k) = \binom{n}{k} p^k q^{n-k}$ $k \in \{0, 1, 2, \dots, n\}$	$np$	$npq$	$(q + pe^t)^n$
Geometric Geom( $p$ )	$P(X = k) = q^k p$ $k \in \{0, 1, 2, \dots\}$	$q/p$	$q/p^2$	$\frac{p}{1-qe^t}, qe^t < 1$
Negative Binomial NBin( $r, p$ )	$P(X = n) = \binom{r+n-1}{n-1} p^r q^n$ $n \in \{0, 1, 2, \dots\}$	$rq/p$	$rq/p^2$	$(\frac{p}{1-qe^t})^r, qe^t < 1$
Hypergeometric HGeom( $w, b, n$ )	$P(X = k) = \binom{w}{k} \binom{b}{n-k} / \binom{n}{w}$ $k \in \{0, 1, 2, \dots, n\}$	$\mu = \frac{nw}{b+w}$	$\left(\frac{w+b-n}{w+b-1}\right) n \frac{\mu}{n} (1 - \frac{\mu}{n})$	messy
Poisson Pois( $\lambda$ )	$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$ $k \in \{0, 1, 2, \dots\}$	$\lambda$	$\lambda$	$e^{\lambda(e^t - 1)}$
Uniform Unif( $a, b$ )	$f(x) = \frac{1}{b-a}$ $x \in (a, b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$	$\frac{e^{tb}-e^{ta}}{b-a}$
Normal $N(\mu, \sigma^2)$	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x - \mu)^2/(2\sigma^2)}$ $x \in (-\infty, \infty)$	$\mu$	$\sigma^2$	$e^{t\mu + \frac{\sigma^2 t^2}{2}}$
Exponential Expo( $\lambda$ )	$f(x) = \lambda e^{-\lambda x}$ $x \in (0, \infty)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$	$\frac{\lambda}{\lambda-t}, t < \lambda$
Gamma Gamma( $n, \lambda$ )	$f(x) = \frac{1}{\Gamma(n)} (\lambda x)^n e^{-\lambda x} \frac{1}{x^n}$ $x \in (0, \infty)$	$\frac{n}{\lambda}$	$\frac{n}{\lambda^2}$	$\left(\frac{\lambda}{\lambda-t}\right)^n, t < \lambda$
Beta Beta( $a, b$ )	$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}$ $x \in (0, 1)$	$\mu = \frac{a}{a+b}$	$\frac{\mu(1-\mu)}{(a+b+1)}$	messy
Log-Normal $\mathcal{LN}(\mu, \sigma^2)$	$\frac{1}{x\sigma\sqrt{2\pi}} e^{-(\log x - \mu)^2/(2\sigma^2)}$ $x \in (0, \infty)$	$\theta = e^{\mu + \sigma^2/2}$	$\theta^2(e^{\sigma^2} - 1)$	doesn't exist
Chi-Square $\chi_n^2$	$\frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$ $x \in (0, \infty)$	$n$	$2n$	$(1 - 2t)^{-n/2}, t < 1/2$
Student- $t$ $t_n$	$\frac{\Gamma((n+1)/2)}{\sqrt{\pi n/2}} (1 + x^2/n)^{-(n+1)/2}$ $x \in (-\infty, \infty)$	0 if $n > 1$	$\frac{n}{n-2}$ if $n > 2$	doesn't exist

# Big Data

# Python

- Pyspark RDD (PDF)

## Python For Data Science Cheat Sheet

### PySpark - RDD Basics

Learn Python for data science interactively at [www.DataCamp.com](http://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
>>> sc.pythonVer
>>> sc.master
>>> str(sc.sparkHome)
>>> str(sc.sparkUser())
>>> sc.appName
>>> sc.applicationId
>>> sc.defaultParallelism
>>> sc.defaultMinPartitions
```

Retrieve SparkContext version  
Retrieve Python version  
Master URL to connect to  
Path where Spark is installed on worker nodes  
Retrieve name of the Spark User running  
Spark Context  
Return application name  
Retrieve application ID  
Return default level of parallelism  
Default minimum number of partitions for RDDs

##### Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
...     .setMaster("local[2]")
...     .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

```
>>> rdd1 = sc.parallelize([('a', 1), ('a', 2), ('b', 1)])
>>> rdd2 = sc.parallelize([('a', 1), ('d', 1), ('b', 1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([('a', 'a'), ('v', 'v'), ('b', 'b'), ('p', 'p')])
```

###### External Data

Read either one text file from HDFS, a local file system 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.count()
List the number of partitions
>>> rdd.countByKey()
Count RDD instances
>>> rdd.countByValue()
Count RDD instances by value
>>> rdd.collectAsMap()
Return (key,value) pairs as a dictionary
>>> rdd.collect()
Sum of RDD elements
>>> sc.parallelize([]).isEmpty()
Check whether RDD is empty
```

### Summary

```
>>> rdd3.max()
Maximum value of RDD elements
>>> rdd3.min()
Minimum value of RDD elements
>>> rdd3.mean()
Mean value of RDD elements
>>> rdd3.stdev()
Standard deviation of RDD elements
>>> rdd3.variance()
Compute variance of RDD elements
>>> rdd3.histogram(3)
Compute histogram by bins
>>> rdd3.stats()
Summary statistics (count, mean, stdev, max & min)
```

### Applying Functions

```
>>> rdd4.map(lambda x: x*(x+1)).collect()
Apply a function to each RDD element
>>> rdd5 = rdd1.flatMap(lambda x: x*x[1], x[0])
...
>>> rdd5.collect()
Apply a function to each RDD element and flatten the result
>>> rdd6.flatMapValues(lambda x: x*x).collect()
Apply a flatMap function to each (key,value) pair of RDD without changing the keys
```

### Selecting Data

```
>>> rdd2.collect()
Return a list with all RDD elements
>>> rdd2.take(2)
Take first 2 RDD elements
>>> rdd2.first()
Take first RDD element.
>>> rdd2.top(2)
Take top 2 RDD elements
>>> rdd3.sample(False, 0.15, 81).collect()
Return sampled subset of RDD
>>> rdd3.filter(lambda x: "a" in x).collect()
Filter the RDD
>>> rdd3.distinct().collect()
Return distinct RDD values
>>> rdd3.keys().collect()
Return (key,value) RDD's keys
```

### Iterating

```
>>> def g(x): print(x)
>>> rdd4.foreach(g)
('a', 1)
('b', 1)
('a', 2)
('a', 2)
```

Apply a function to all RDD elements.

## Reshaping Data

### Reducing

```
>>> rdd3.reduceByKey(lambda x,y: x+y).collect()
[('a', 9), ('b', 2)]
>>> rdd3.reduce(lambda a, b: a + b)
('a', 10), ('b', 2)
```

Merge the min. values for each key  
Merge the additive values

Return RDD of grouped values

Group RDD by key

### Grouping by

```
>>> rdd3.groupByKey(lambda x: x % 2).mapValues(list).collect()
```

Aggregate RDD elements of each partition and then the results.

Aggregate values of each RDD key

```
>>> rdd3.aggregateByKey(0, 0, lambda a, v: a+v).collect()
```

Aggregate the elements of each partition, and then the results.

Merge the value for each key

```
>>> rdd3.keyBy(lambda x: x*x).collect()
```

Create tuples of RDD elements by applying a function

### Mathematical Operations

```
>>> rdd2.subtract(rdd2).collect()
Return each rdd2 value not contained in rdd2
>>> rdd2.subtractByKey(rdd2).collect()
Return each (key,value) pair of rdd2 with no matching key in rdd2
>>> rdd2.cartesian(rdd2).collect()
Return the Cartesian product of rdd and rdd2
```

### Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect()
[('a', 1), ('b', 1), ('a', 2)]
>>> rdd2.sortByKey().collect()
[('a', 1), ('b', 1), ('a', 2)]
```

Sort RDD by given function

Sort (key,value) RDD by key

### Repartitioning

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

### Saving

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

### Stopping SparkContext

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

### Execution

```
>>> !/bin/spark-submit examples/src/main/python/pl.py
```

DataCamp



- Pyspark DF (PDF)

## Python For Data Science Cheat Sheet

### PySpark - SQL Basics

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



### PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.



### Initializing SparkSession

A SparkSession can be used to create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

### Creating DataFrames

#### From RDDs

```
>>> from pyspark.sql.types import *
Infer Schema
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
>>> peopleDF = spark.createDataFrame(people)
Specify Schema
>>> schemaString = "name age"
>>> fields = [StructField("name", StringType(), True) for field_name in schemaString.split(",")]
>>> Schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
+-----+
| name | age |
+-----+
| Paul | 32 |
| Tim | 41 |
| Jordan | 39 |
+-----+
```

#### From Spark Data Sources

```
JSON
>>> df = spark.read.json("customer.json")
>>> df.show()
+-----+
| address | city | firstName | lastName | phoneNumber |
+-----+
| [New York, 10001, John, Doe] | 100-123-4567 |
+-----+
>>> df2 = spark.read.load("people.json", format="json")
Parquet
>>> df3 = spark.read.load("users.parquet")
TXT files
>>> df4 = spark.read.text("people.txt")
```

#### Inspect Data

>>> df.dtypes	Return all column names and data types
>>> df.show()	Display the content of df
>>> df.head(0)	Return first n rows
>>> df.first()	Return first row
>>> df.take(2)	Return the first n rows
>>> df.schema	Return the schema of df

### Duplicate Values

```
>>> df = df.dropDuplicates()
```

### Queries

```
>>> from pyspark.sql import functions as F
Select
>>> df.select("firstName").show()
>>> df.select("firstName", "lastName") \
    .show()
>>> df.select("firstName", \
    F.col("age").\
    explode("phoneNumbers") \
    .alias("contactInfo")) \
    .select("contactInfo.type", \
    "firstName", \
    "age") \
    .show()
>>> df.select("age").filter("age > 24).show()
When
>>> df.select("firstName", \
    F.when(df.age > 30, 1) \
    .otherwise(0)) \
    .show()
>>> df.filter(firstName.isin("Jane", "Boris")).collect()
Like
>>> df.select("firstName", \
    df.lastName.like("Smith")) \
    .show()
StartsWith - EndsWith
>>> df.select("firstName", \
    df.lastName) \
    .startsWith("Sm") \
    .show()
>>> df.select(df.lastName.endsWith("th")) \
    .show()
Substring
>>> df.select(df.firstName.substring(0, 3)) \
    .alias("name") \
    .collect()
Between
>>> df.select(df.age.between(22, 24)) \
    .show()
```

### Add, Update & Remove Columns

#### Adding Columns

```
>>> df = df.withColumn("city", df.address.city) \
    .withColumn("postalCode", df.address.postalCode) \
    .withColumn("state", df.address.state) \
    .withColumn("address", df.address.streetAddress) \
    .withColumn("telePhoneNumbers", \
        explode(df.phoneNumber.number)) \
    .withColumn("telephoneType", \
        explode(df.phoneNumber.type))
```

#### Updating Columns

```
>>> df = df.withColumnRenamed('telephoneNumber', 'phoneNumber')
```

#### Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.addresses).drop(df.phoneNumber)
```

Return substrings of FirstName

Show ages values are true if between 22 and 24

Show FirstName and LastName if LastName starts with Sm

Show last names ending in th

Show all entries in firstName column

Show all entries in FirstName, age and type

Show all entries in firstName and age, add 1 to the entries of age

Show all entries where age > 24

Show FirstName or 0 depending on age >= 30

Show FirstName if in the given options

Show FirstName and LastName if FirstName is like Smith

Show all entries in the entries of age, add 1 to the entries of age

Show all entries in the entries of age, add 1 to the entries of age

### GroupBy

```
>>> df.groupby("age") \
    .count() \
    .show()
```

Group by age, count the members in the groups

### Filter

```
>>> df.filter(df["age"] > 24).show()
```

Filter entries of age, only keep those records of which the values are > 24

### Sort

```
>>> peopleDF.sort(peopleDF.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age", "city"], ascending=[0, 1]).collect()
```

### Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na.replace(10, 20).show()
```

Replace null values

Return new df omitting rows with null values

Return new df replacing one value with another

### Repartitioning

```
>>> df.repartiton(10).rdd \
    .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

df with no partitions

df with 1 partition

### Running SQL Queries Programmatically

#### Registering DataFrames as Views

```
>>> peopleDF.createOrReplaceTempView("people")
>>> df.createTempView("customers")
>>> df.createOrReplaceTempView("customer")
```

#### Query Views

```
>>> df = spark.sql("SELECT * FROM customer").show()
>>> peopleDF = spark.sql("SELECT * FROM global_temp.people").show()
```

### Output

#### Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert df into an RDD

Convert df into a RDD of string

Return the contents of df as Pandas Dataframe

#### Write & Save to Files

```
>>> df.select("firstName", "city") \
    .write \
    .json("nameAndCity.json")
>>> df.select("firstName", "age") \
    .write \
    .parquet("namesAndAges.parquet")
```

#### Stopping SparkSession

```
>>> spark.stop()
```

DataCamp

Learn Python for Data Science interactively



## Dask (PDF)



## DASK FOR PARALLEL COMPUTING CHEAT SHEET

See full Dask documentation at: <http://dask.pydata.org/>

These instructions use the conda environment manager. Get yours at <http://bit.ly/getconda>

### DASK QUICK INSTALL

Install Dask with conda

```
conda install dask
```

Install Dask with pip

```
pip install dask[complete]
```

### DASK COLLECTIONS

#### EASY TO USE BIG DATA COLLECTIONS

### DASK DATAFRAMES

#### PARALLEL PANDAS DATAFRAMES FOR LARGE DATA

Import

```
import dask.dataframe as dd
```

Read CSV data

```
df = dd.read_csv('my-data.*.csv')
```

Read Parquet data

```
df = dd.read_parquet('my-data.parquet')
```

Filter and manipulate data with Pandas syntax

```
df['z'] = df.x + df.y
```

Standard groupby aggregations, joins, etc.

```
result = df.groupby(df.z).y.mean()
```

Compute result as a Pandas dataframe

```
out = result.compute()
```

Or store to CSV, Parquet, or other formats

```
result.to_parquet('my-output.parquet')
```

### EXAMPLE

```
df = dd.read_csv('filenames.*.csv')
df.groupby(df.timestamp.day) \
    .value.mean().compute()
```

### DASK ARRAYS

#### PARALLEL NUMPY ARRAYS FOR LARGE DATA

Import

```
import dask.array as da
```

Create from any array-like object

```
import h5py
dataset = h5py.File('my-data.hdf5') ['/group/dataset']
x = da.from_array(dataset, chunks=(1000, 1000))
```

Including HDFS, NetCDF, or other on-disk formats.

Alternatively generate an array from a random distribution:

EXAMPLE	<pre>import numpy as np # Manually generate an array from a random Gaussian distribution. x = np.random.uniform(shape=(100, 100), chunks=(100, 100))</pre>
Perform operations with NumPy syntax	<pre>y = x.dot(x.T - 1) - x.mean(axis=0)</pre>
Compute result as a NumPy array	<pre>result = y.compute()</pre>
Or store to HDF5, NetCDF or other on-disk format	<pre>out = f.create_dataset(...) x.store(out)</pre>
DASK BAGS	PARELLEL LISTS FOR UNSTRUCTURED DATA
Import	<pre>import dask.bag as db</pre>
Create Dask Bag from a sequence	<pre>b = db.from_sequence(seq, npartitions)</pre>
Or read from text formats	<pre>b = db.read_text('my-data.*.json')</pre>
Map and filter results	<pre>import json records = b.map(json.loads)     .filter(lambda d: d["name"] == "Alice")</pre>
Compute aggregations like mean, count, sum	<pre>records.pluck('key-name').mean().compute()</pre>
Or store results back to text formats	<pre>records.to_textfiles('output.*.json')</pre>
EXAMPLE	<pre>db.read_text('s3://bucket/my-data.*.json')     .map(json.loads)     .filter(lambda d: d["name"] == "Alice")     .to_textfiles('s3://bucket/output.*.json')</pre>



CONTINUED ON BACK →

DASK COLLECTIONS (CONTINUED)	
ADVANCED	
Read from distributed file systems or cloud storage	<pre>df = dd.read_parquet('s3://bucket/myfile.parquet')</pre>
Prepend prefixes like hdfs://, s3://, or gcs:// to paths	<pre>b = db.read_text('hdfs:///path/to/my-data.*.json')</pre>
Persist lazy computations in memory	<pre>df = df.persist()</pre>
Compute multiple outputs at once	<pre>dask.compute(x.min(), x.max())</pre>
CUSTOM COMPUTATIONS	FOR CUSTOM CODE AND COMPLEX ALGORITHMS
DASK DELAYED	LAZY PARALLELISM FOR CUSTOM CODE
Import	<pre>import dask</pre>
Wrap custom functions with the @dask.delayed annotation	<pre>@dask.delayed def load(filename):     ... @dask.delayed def process(data):     ... load = dask.delayed(load) process = dask.delayed(process)</pre>
Delayed functions operate lazily, producing a task graph rather than executing immediately	
Passing delayed results to other delayed functions creates dependencies between tasks	
Call functions in normal code	<pre>data = [load(fn) for fn in filenames] results = [process(d) for d in data]</pre>
Compute results to execute in parallel	<pre>dask.compute(results)</pre>
CONCURRENT.FUTURES	ASYNCHRONOUS REAL-TIME PARALLELISM
Import	<pre>from dask.distributed import Client</pre>
Start local Dask Client	<pre>client = Client()</pre>
Submit individual task asynchronously	<pre>future = client.submit(func, *args, **kwargs)</pre>
Block and gather individual result	<pre>result = future.result()</pre>
Process results as they arrive	<pre>for future in as_completed(futures):     ... T = [client.submit(read, fn) for fn in filenames]</pre>
EXAMPLE	

		L = [client.submit(process, future) for future in L] future = client.submit(sum, L) result = future.result()
SET UP CLUSTER	HOW TO LAUNCH ON A CLUSTER	
<b>MANUALLY</b>		
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')	
<b>ON A SINGLE MACHINE</b>		
Call Client() with no arguments for easy setup on a single host	client = Client()	
<b>CLOUD DEPLOYMENT</b>		
See dask-kubernetes project for Google Cloud	pip install dask-kubernetes	
See dask-ec2 project for Amazon EC2	pip install dask-ec2	
<b>MORE RESOURCES</b>		
User Documentation	<a href="#">dask.pydata.org</a>	
Technical documentation for distributed scheduler	<a href="#">distributed.readthedocs.org</a>	
Report a bug	<a href="#">github.com/dask/dask/issues</a>	
 <b>ANACONDA.</b>		anaconda.com · info@anaconda.com · 512-776-1066 8/20/2017

## R

---

- [Sparklyr \(PDF\)](#)

# Machine Learning

---

## Python

---

- [Scikit-Learn \(PDF\)](#)

## Python For Data Science Cheat Sheet

### Scikit-Learn

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



#### Scikit-learn

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



#### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> from sklearn.datasets import load_iris
>>> X, y = iris.data[, :-1], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
>>> minmax = preprocessing.StandardScaler().fit(X_train)
>>> X_train = minmax.transform(X_train)
>>> X_test = minmax.transform(X_test)
>>> knc = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knc.fit(X_train, y_train)
>>> y_pred = knc.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

#### Loading The Data

[Also see NumPy & Pandas](#)

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

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

#### Training And Test Data

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

#### Preprocessing The Data

##### Standardization

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

##### Normalization

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

##### Binarization

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

## Create Your Model

### Supervised Learning Estimators

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

Support Vector Machines (SVM)
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')

Naive Bayes
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()

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

### Unsupervised Learning Estimators

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

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

## Model Fitting

### Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
```

### Unsupervised Learning

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

Fit the model to the data

Fit the model to the data

Fit to data, then transform it

## Prediction

### Supervised Estimators

```
>>> y_pred = lr.predict(np.random.rand(2,5))
>>> y_pred = lr.predict(X_test)
```

### Unsupervised Estimators

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

Predict labels

Predict labels

Estimate probability of a label

Predict labels in clustering algs

## Evaluate Your Model's Performance

### Classification Metrics

Accuracy Score >>> knn.score(X_test, y_test) >>> from sklearn.metrics import accuracy_score >>> accuracy_score(y_test, y_pred)	Estimator score method Metric scoring functions
Classification Report >>> from sklearn.metrics import classification_report >>> print(classification_report(y_test, y_pred))	Precision, recall, f1-score and support
Confusion Matrix >>> from sklearn.metrics import confusion_matrix >>> print(confusion_matrix(y_test, y_pred))	

### Regression Metrics

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

Mean Squared Error >>> from sklearn.metrics import mean_squared_error >>> mean_squared_error(y_test, y_pred)	
--	--

R <sup>2</sup> Score >>> from sklearn.metrics import r2_score >>> r2_score(y_true, y_pred)	
--	--

### Clustering Metrics

Adjusted Rand Index >>> from sklearn.metrics import adjusted_rand_score >>> adjusted_rand_score(y_true, y_pred)	
---	--

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

V-measure >>> from sklearn.metrics import v_measure_score >>> metrics.v_measure_score(y_true, y_pred)	
---	--

### Cross-Validation

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

### Tune Your Model

#### Grid Search

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

#### Randomized Parameter Optimization

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

DataCamp  
Learn Python for Data Science interactively



## R

- Machine Learning Modelling in R (PDF)

- Caret (PDF)

- Estimatr (PDF)

- H2O (PDF)

- mlr (PDF)

## Supervised Learning

- Regression (PDF)

## Regression (Machine Learning)

**Summary:**

- Common Applications in Business: Used to predict a numeric value (e.g. forecasting sales, estimating prices, etc).
- Key Concept: Data is usually in a rectangular format (like a spreadsheet) with one column that is a **target** (e.g. price) and other columns that are **predictors** (e.g. product category)
- Gotchas:
  - Preprocessing: Knowing when to preprocess data (normalize) prior to machine learning step
  - Feature Engineering: Getting good features is more important than applying complex models.
- Parameter Tuning: Higher complexity models have many parameters that can be tuned.
- Interpretability: Some models are more explainable than others, meaning the estimates for each feature means something in relation to the target. Other models are not interpretable and require additional tools (e.g. LIME) to explain.

**Terminology:**

- Supervised vs Unsupervised: Regression is a supervised technique that requires training with a "target" (e.g. price of product or sales by month). The algorithm learns by identifying relationships between the target & the **predictors** (attributes related to the target like category of product or month of sales).
- Classification vs Regression: Classification aims to predict classes (either binary yes/no or multi-class categorical). Regression aims to predict a numeric value (e.g. product price = \$4,233).
- Preprocessing: Many algorithms require preprocessing, which transforms the data into a format more suitable for the machine learning algorithm. A common example is "standardization" or scaling the feature to be in a range of [0,1] (or close to it).
- Hyper Parameter & Tuning: Machine learning algorithms have many parameters that can be adjusted (e.g. learning rate in GBM). Tuning is the process of systematically finding the optimum parameter values.
- Cross Validation: Machine learning algorithms should be tuned on a validation set as opposed to a test set. Cross-validation is the process of splitting the training set into multiple sets using a portion of the training set for tuning.
- Performance Metrics (Regression): Common performance metrics are Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). These measures provide an estimate of model performance to compare models to each other.

**R Cheat Sheet**

**Parsnip (Machine Learning):**

- Model List (start here first)
- Linear Regression & GLM
- Decision Tree
- Random Forest
- Boosted Trees (XGBoost)
- SVM: Poly & Radial

**Keras (Deep Learning)**

**H2O (ML & DL Framework)**

**MLR (ML Framework)**

**Python Cheat Sheet**

**Scikit-Learn (Machine Learning):**

- Linear Regression
- GLM (Elastic Net)
- Decision Tree (Regressor)
- Random Forest (Regressor)
- AdaBoost (Regressor)
- XGBoost
- SVM (Regressor)

**Keras (Deep Learning)**

**H2O (ML & DL Framework)**

**Resources**

- [Business Analysis With R Course \(DS4B 101-R\) - Modeling - Week 6](#)
- [Business Science Problem Framework](#)
- [Ultimate R Cheat Sheet | Ultimate Python Cheat Sheet](#)

version: 1.1

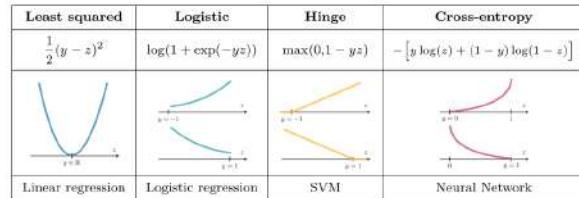
From @afshinea:

- [VIP Supervised Learning \(PDF\)](#)

## VIP Cheatsheet: Supervised Learning

Afshine AMIDI and Shervine AMIDI

September 9, 2018



### Introduction to Supervised Learning

Given a set of data points  $\{x^{(1)}, \dots, x^{(m)}\}$  associated to a set of outcomes  $\{y^{(1)}, \dots, y^{(m)}\}$ , we want to build a classifier that learns how to predict  $y$  from  $x$ .

□ **Type of prediction** – The different types of predictive models are summed up in the table below:

	Regression	Classifier
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

□ **Type of model** – The different models are summed up in the table below:

	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

### Notations and general concepts

□ **Hypothesis** – The hypothesis is noted  $h_{\theta}$  and is the model that we choose. For a given input data  $x^{(i)}$ , the model prediction output is  $h_{\theta}(x^{(i)})$ .

□ **Loss function** – A loss function is a function  $L : (z, y) \in \mathbb{R} \times \mathbb{Y} \mapsto L(z, y) \in \mathbb{R}$  that takes as inputs the predicted value  $z$  corresponding to the real data value  $y$  and outputs how different they are. The common loss functions are summed up in the table below:

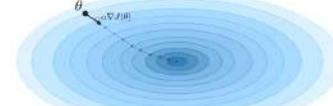


□ **Cost function** – The cost function  $J$  is commonly used to assess the performance of a model, and is defined with the loss function  $L$  as follows:

$$J(\theta) = \sum_{i=1}^m L(h_{\theta}(x^{(i)}), y^{(i)})$$

□ **Gradient descent** – By noting  $\alpha \in \mathbb{R}$  the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function  $J$  as follows:

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$



*Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.*

□ **Likelihood** – The likelihood of a model  $L(\theta)$  given parameters  $\theta$  is used to find the optimal parameters  $\theta$  through maximizing the likelihood. In practice, we use the log-likelihood  $\ell(\theta) = \log(L(\theta))$  which is easier to optimize. We have:

$$\theta^{\text{opt}} = \arg \max_{\theta} L(\theta)$$

□ **Newton's algorithm** – The Newton's algorithm is a numerical method that finds  $\theta$  such that  $\ell'(\theta) = 0$ . Its update rule is as follows:

$$\theta \leftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

*Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:*

$$\theta \leftarrow \theta - (\nabla_{\theta}^2 \ell(\theta))^{-1} \nabla_{\theta} \ell(\theta)$$

### Linear regression

We assume here that  $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$

□ **Normal equations** – By noting  $X$  the matrix design, the value of  $\theta$  that minimizes the cost function is a closed-form solution such that:

$$\theta = (X^T X)^{-1} X^T y$$

□ **LMS algorithm** – By noting  $\alpha$  the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of  $m$  data points, which is also known as the Widrow-Hoff learning rule, is as follows:

$$\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m [y^{(i)} - h_{\theta}(x^{(i)})] x_j^{(i)}$$

*Remark: the update rule is a particular case of the gradient ascent.*

□ **LWR** – Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by  $w^{(i)}(x)$ , which is defined with parameter  $\tau \in \mathbb{R}$  as:

$$w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

### Classification and logistic regression

□ **Sigmoid function** – The sigmoid function  $g$ , also known as the logistic function, is defined as follows:

$$\forall z \in \mathbb{R}, \quad g(z) = \frac{1}{1 + e^{-z}} \in [0, 1]$$

□ **Logistic regression** – We assume here that  $y|x; \theta \sim \text{Bernoulli}(\phi)$ . We have the following form:

$$\phi = p(y=1|x; \theta) = \frac{1}{1 + \exp(-\theta^T x)} = g(\theta^T x)$$

*Remark: there is no closed form solution for the case of logistic regressions.*

□ **Softmax regression** – A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, we set  $\theta_K = 0$ , which makes the Bernoulli parameter  $\phi_i$  of each class  $i$  equal to:

$$\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^K \exp(\theta_j^T x)}$$

### Generalized Linear Models

□ **Exponential family** – A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter, also called the canonical parameter or link function,  $\eta$ , a sufficient statistic  $T(y)$  and a log-partition function  $a(\eta)$  as follows:

$$p(y; \eta) = b(y) \exp(\eta T(y) - a(\eta))$$

*Remark: we will often have  $T(y) = y$ . Also,  $\exp(-a(\eta))$  can be seen as a normalization parameter that will make sure that the probabilities sum to one.*

Here are the most common exponential distributions summed up in the following table:

Distribution	$\eta$	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	$y$	$\log(1 + \exp(\eta))$	1
Gaussian	$\mu$	$y$	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\eta^2}{2}\right)$
Poisson	$\log(\lambda)$	$y$	$e^{\eta}$	$\frac{1}{y!}$
Geometric	$\log(1 - \phi)$	$y$	$\log\left(\frac{\phi}{1-\phi}\right)$	1

□ **Assumptions of GLMs** – Generalized Linear Models (GLM) aim at predicting a random variable  $y$  as a function to  $x \in \mathbb{R}^{n+1}$  and rely on the following 3 assumptions:

$$(1) \quad y|x; \theta \sim \text{ExpFamily}(\eta) \quad (2) \quad h_{\theta}(x) = E[y|x; \theta] \quad (3) \quad \eta = \theta^T x$$

*Remark: ordinary least squares and logistic regression are special cases of generalized linear models.*

### Support Vector Machines

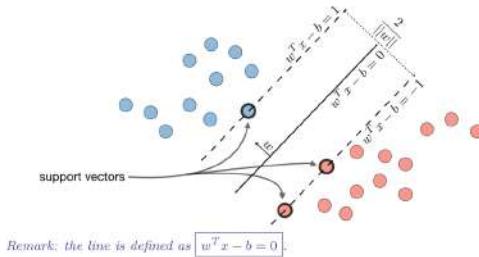
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

□ **Optimal margin classifier** – The optimal margin classifier  $h$  is such that:

$$h(x) = \text{sign}(\theta^T x - b)$$

where  $(w, b) \in \mathbb{R}^n \times \mathbb{R}$  is the solution of the following optimization problem:

$$\min \frac{1}{2} \|w\|^2 \quad \text{such that} \quad y^{(i)}(w^T x^{(i)} - b) \geq 1$$



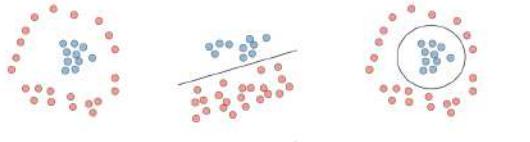
**Hinge loss** – The hinge loss is used in the setting of SVMs and is defined as follows:

$$L(z,y) = [1 - yz]_+ = \max(0, 1 - yz)$$

**Kernel** – Given a feature mapping  $\phi$ , we define the kernel  $K$  to be defined as:

$$K(x,z) = \phi(x)^T \phi(z)$$

In practice, the kernel  $K$  defined by  $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$  is called the Gaussian kernel and is commonly used.



Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping  $\phi$ , which is often very complicated. Instead, only the values  $K(x,z)$  are needed.

**Lagrangian** – We define the Lagrangian  $\mathcal{L}(w,b)$  as follows:

$$\mathcal{L}(w,b) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

Remark: the coefficients  $\beta_i$  are called the Lagrange multipliers.

### Generative Learning

A generative model first tries to learn how the data is generated by estimating  $P(x|y)$ , which we can then use to estimate  $P(y|x)$  by using Bayes' rule.

### Gaussian Discriminant Analysis

**Setting** – The Gaussian Discriminant Analysis assumes that  $y$  and  $x|y = 0$  and  $x|y = 1$  are such that:

$$y \sim \text{Bernoulli}(\phi)$$

$$x|y = 0 \sim \mathcal{N}(\mu_0, \Sigma) \quad \text{and} \quad x|y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

**Estimation** – The following table sums up the estimates that we find when maximizing the likelihood:

$\hat{\phi}$	$\hat{\mu}_j \quad (j = 0, 1)$	$\hat{\Sigma}$
$\frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=j\}}}$	$\frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$

### Naive Bayes

**Assumption** – The Naive Bayes model supposes that the features of each data point are all independent:

$$P(x|y) = P(x_1, x_2, \dots | y) = P(x_1|y)P(x_2|y) \dots = \prod_{i=1}^n P(x_i|y)$$

**Solutions** – Maximizing the log-likelihood gives the following solutions, with  $k \in \{0, 1\}$ ,  $j \in [1, L]$

$$P(y=k) = \frac{1}{m} \times \#\{j|y^{(j)}=k\} \quad \text{and} \quad P(x_i=l|y=k) = \frac{\#\{j|y^{(j)}=k \text{ and } x_i^{(j)}=l\}}{\#\{j|y^{(j)}=k\}}$$

Remark: Naive Bayes is widely used for text classification and spam detection.

### Tree-based and ensemble methods

These methods can be used for both regression and classification problems.

**CART** – Classification and Regression Trees (CART), commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

**Random forest** – It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

Remark: random forests are a type of ensemble methods.

**Boosting** – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

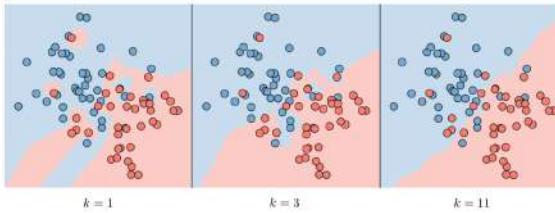
Adaptive boosting	Gradient boosting
- High weights are put on errors to improve at the next boosting step - Known as AdaBoost	- Weak learners trained on remaining errors

$$\hat{e}(h) = \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\{h(x^{(i)}) \neq y^{(i)}\}}$$

### Other non-parametric approaches

**k-nearest neighbors** – The k-nearest neighbors algorithm, commonly known as k-NN, is a non-parametric approach where the response of a data point is determined by the nature of its  $k$  neighbors from the training set. It can be used in both classification and regression settings.

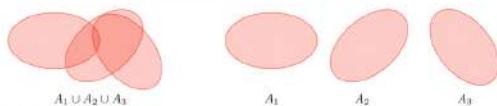
Remark: The higher the parameter  $k$ , the higher the bias, and the lower the parameter  $k$ , the higher the variance.



### Learning Theory

**Union bound** – Let  $A_1, \dots, A_k$  be  $k$  events. We have:

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k)$$



**Hoeffding inequality** – Let  $Z_1, \dots, Z_m$  be  $m$  iid variables drawn from a Bernoulli distribution of parameter  $\phi$ . Let  $\hat{\phi}$  be their sample mean and  $\gamma > 0$  fixed. We have:

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 m)$$

Remark: this inequality is also known as the Chernoff bound.

**Training error** – For a given classifier  $h$ , we define the training error  $\hat{e}(h)$ , also known as the empirical risk or empirical error, to be as follows:

**Probably Approximately Correct (PAC)** – PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:

- the training and testing sets follow the same distribution
- the training examples are drawn independently

**Shattering** – Given a set  $S = \{x^{(1)}, \dots, x^{(d)}\}$ , and a set of classifiers  $\mathcal{H}$ , we say that  $\mathcal{H}$  shatters  $S$  if for any set of labels  $\{y^{(1)}, \dots, y^{(d)}\}$ , we have:

$$\exists h \in \mathcal{H}, \quad \forall i \in [1, d], \quad h(x^{(i)}) = y^{(i)}$$

**Upper bound theorem** – Let  $\mathcal{H}$  be a finite hypothesis class such that  $|\mathcal{H}| = k$  and let  $\delta$  and the sample size  $m$  be fixed. Then, with probability of at least  $1 - \delta$ , we have:

$$\hat{e}(h) \leq \left( \min_{h \in \mathcal{H}} e(h) \right) + 2 \sqrt{\frac{1}{2m} \log \left( \frac{2k}{\delta} \right)}$$

**VC dimension** – The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class  $\mathcal{H}$ , noted  $\text{VC}(\mathcal{H})$ , is the size of the largest set that is shattered by  $\mathcal{H}$ .

Remark: the VC dimension of  $\mathcal{H}$  = [set of linear classifiers in 2 dimensions] is 3.



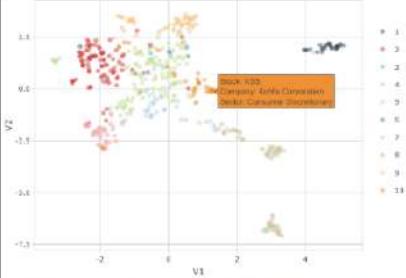
**Theorem (Vapnik)** – Let  $\mathcal{H}$  be given, with  $\text{VC}(\mathcal{H}) = d$  and  $m$  the number of training examples. With probability at least  $1 - \delta$ , we have:

$$\hat{e}(h) \leq \left( \min_{h \in \mathcal{H}} e(h) \right) + O \left( \sqrt{\frac{d}{m} \log \left( \frac{m}{d} \right)} + \frac{1}{m} \log \left( \frac{1}{\delta} \right) \right)$$

# Unsupervised Learning

- Segmentation and Clustering (PDF)

## Segmentation & Clustering



Combining K-Means & UMAP to visualize clusters by Stock Price Movements

Type	Popular Methods	Uses	Data Treatment
Clustering	K-Means Hierarchical Clustering	<b>Group Detection:</b> Methods use a measure of similarity (e.g. Euclidean distance) to detect groups within data set	Standardized or normalized
Dimensionality Reduction	PCA UMAP tSNE	<b>Reduce Width of Data:</b> Performing Machine Learning on wide data can drastically increase the time for algorithms to converge. Dimensionality reduction can be applied as a preprocessing step to reduce the width (number of columns) of the data but still maintain a high proportion of the overall structure.  <b>Visualization:</b> Visualizing the first two components as X and Y often can enable cluster visualization. Combining with clustering techniques can provide a useful method of visualization.	Standardized or normalized

**Resources**

- [Business Analysis With R Course \(DS4B 101-R\) - Modeling - Week 6](#)
- [Business Science Problem Framework](#)
- [Ultimate R Cheat Sheet](#) | [Ultimate Python Cheat Sheet](#)

### Summary:

- **Common Applications in Business:** Can be used for finding segments within Customers, Companies, etc.
- **Key Concept:** Transform data into a matrix enabling trends to be compared across units of measure (e.g. user-item matrix)
- **Gotchas:** Data must be normalized or standardized to enable comparison. This often requires calculation proportions of values by customer, company, etc to ensure the larger values do not dominate the trend mining operation.
- **How Many Components/Clusters?** Use a *Scree Plot* to determine the proportion of variance explained or total within sum of squares

**R Cheat Sheet**

**K-Means**

```
set.seed(0)
kmeans_obj <- kmeans(X, centers = 4)
```

**UMAP**

```
library(umap)

umap_obj <- umap(X)
```

**Python Cheat Sheet**

**K-Means**

```
from sklearn.cluster import KMeans
kmeans = KMeans(
    n_clusters=4,
    random_state=0).fit(X)
```

**UMAP**

```
import umap

reducer = umap.UMAP()
embedding = reducer.fit_transform(X)
```

version: 1.0

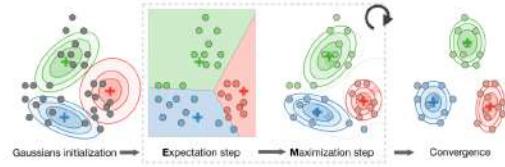
From @afshinea:

- [VIP Unsupervised Learning \(PDF\)](#)

## VIP Cheatsheet: Unsupervised Learning

Afhine AMIDI and Shervine AMIDI

September 9, 2018



### Introduction to Unsupervised Learning

**Motivation** – The goal of unsupervised learning is to find hidden patterns in unlabeled data  $\{x^{(1)}, \dots, x^{(m)}\}$ .

**Jensen's inequality** – Let  $f$  be a convex function and  $X$  a random variable. We have the following inequality:

$$E[f(X)] \geq f(E[X])$$

### Expectation-Maximization

**Latent variables** – Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted  $z$ . Here are the most common settings where there are latent variables:

Setting	Latent variable $z$	$x z$	Comments
Mixture of $k$ Gaussians	Multinomial( $\phi$ )	$\mathcal{N}(\mu_j, \Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0, I)$	$\mathcal{N}(\mu + \Lambda z, \psi)$	$\mu_j \in \mathbb{R}^n$

**Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter  $\theta$  through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step:** Evaluate the posterior probability  $Q_i(z^{(i)})$  that each data point  $x^{(i)}$  came from a particular cluster  $z^{(i)}$  as follows:

$$Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)}; \theta)$$

- M-step:** Use the posterior probabilities  $Q_i(z^{(i)})$  as cluster specific weights on data points  $x^{(i)}$  to separately re-estimate each cluster model as follows:

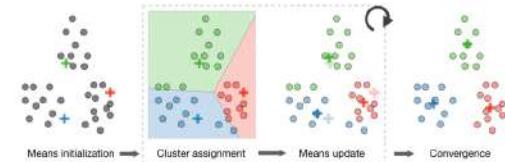
$$\theta_i = \underset{\theta}{\operatorname{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log \left( \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right) dz^{(i)}$$

### k-means clustering

We note  $c^{(i)}$  the cluster of data point  $i$  and  $\mu_j$  the center of cluster  $j$ .

**Algorithm** – After randomly initializing the cluster centroids  $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ , the  $k$ -means algorithm repeats the following step until convergence:

$$c^{(i)} = \arg \min_j \|x^{(i)} - \mu_j\|^2 \quad \text{and} \quad \mu_j = \frac{\sum_{i=1}^m \mathbb{1}_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m \mathbb{1}_{\{c^{(i)}=j\}}}$$



**Distortion function** – In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c(i)}\|^2$$

### Hierarchical clustering

**Algorithm** – It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

**Types** – There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

### Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

**Silhouette coefficient** – By noting  $a$  and  $b$  the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient  $s$  for a single sample is defined as follows:

$$s = \frac{b - a}{\max(a, b)}$$

**Calinski-Harabaz Index** – By noting  $k$  the number of clusters,  $B_k$  and  $W_k$  the between and within-clustering dispersion matrices respectively defined as

$$B_k = \sum_{j=1}^k n_{c(j)} (\mu_{c(j)} - \mu)(\mu_{c(j)} - \mu)^T, \quad W_k = \sum_{i=1}^m (x^{(i)} - \mu_{c(i)}) (x^{(i)} - \mu_{c(i)})^T$$

The Calinski-Harabaz index  $s(k)$  indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N-k}{k-1}$$

### Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

**Eigenvalue, eigenvector** – Given a matrix  $A \in \mathbb{R}^{n \times n}$ ,  $\lambda$  is said to be an eigenvalue of  $A$  if there exists a vector  $z \in \mathbb{R}^n \setminus \{0\}$ , called eigenvector, such that we have:

$$Az = \lambda z$$

**Spectral theorem** – Let  $A \in \mathbb{R}^{n \times n}$ . If  $A$  is symmetric, then  $A$  is diagonalizable by a real orthogonal matrix  $U \in \mathbb{R}^{n \times n}$ . By noting  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ , we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

*Remark:* the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix  $A$ .

**Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on  $k$  dimensions by maximizing the variance of the data as follows:

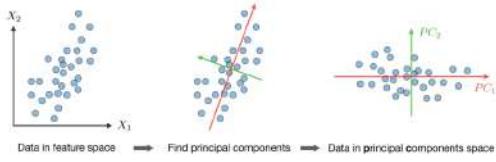
- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad \text{where} \quad \mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \text{and} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

- Step 2: Compute  $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \in \mathbb{R}^{n \times n}$ , which is symmetric with real eigenvalues.

- Step 3: Compute  $u_1, \dots, u_k \in \mathbb{R}^n$  the  $k$  orthogonal principal eigenvectors of  $\Sigma$ , i.e. the orthogonal eigenvectors of the  $k$  largest eigenvalues.

- Step 4: Project the data on  $\text{span}_{\mathbb{R}}(u_1, \dots, u_k)$ . This procedure maximizes the variance among all  $k$ -dimensional spaces.



### Independent component analysis

It is a technique meant to find the underlying generating sources.

**Assumptions** – We assume that our data  $x$  has been generated by the  $n$ -dimensional source vector  $s = (s_1, \dots, s_n)$ , where  $s_i$  are independent random variables, via a mixing and non-singular matrix  $A$  as follows:

$$x = As$$

The goal is to find the unmixing matrix  $W = A^{-1}$  by an update rule.

**Bell and Sejnowski ICA algorithm** – This algorithm finds the unmixing matrix  $W$  by following the steps below:

- Write the probability of  $x = As = W^{-1}s$  as:

$$p(x) = \prod_{i=1}^n p_s(w_i^T x) \cdot |W|$$

- Write the log likelihood given our training data  $\{x^{(i)}, i \in [1, m]\}$  and by noting  $g$  the sigmoid function as:

$$l(W) = \sum_{i=1}^m \left( \sum_{j=1}^n \log(g'(w_j^T x^{(i)})) + \log |W| \right)$$

Therefore, the stochastic gradient ascent learning rule is such that for each training example  $x^{(i)}$ , we update  $W$  as follows:

$$W \leftarrow W + \alpha \begin{pmatrix} \begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)T} + (W^T)^{-1} \end{pmatrix}$$

## Hacks, tricks and tips

---

From @afshinea:

- [VIP Machine Learning Tips and Tricks \(PDF\)](#)

## VIP Cheatsheet: Machine Learning Tips

Afshine AMIDI and Shervine AMIDI

September 9, 2018

### Metrics

Given a set of data points  $\{x^{(1)}, \dots, x^{(m)}\}$ , where each  $x^{(i)}$  has  $n$  features, associated to a set of outcomes  $\{y^{(1)}, \dots, y^{(m)}\}$ , we want to assess a given classifier that learns how to predict  $y$  from  $x$ .

### Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

**Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:

		Predicted class	
		+	-
Actual class	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

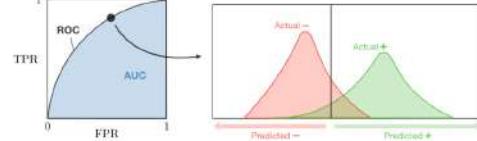
**Main metrics** – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall / Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

**ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

**AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



### Regression

**Basic metrics** – Given a regression model  $f$ , the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$SS_{tot} = \sum_{i=1}^m (y_i - \bar{y})^2$	$SS_{reg} = \sum_{i=1}^m (f(x_i) - \bar{y})^2$	$SS_{res} = \sum_{i=1}^m (y_i - f(x_i))^2$

**Coefficient of determination** – The coefficient of determination, often noted  $R^2$  or  $r^2$ , provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

**Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables  $n$  that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R <sup>2</sup>
$\frac{SS_{res} + 2(n+1)\sigma^2}{m}$	$2[(n+2) - \log(L)]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

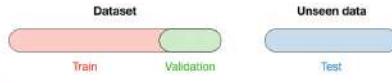
where  $L$  is the likelihood and  $\hat{\sigma}^2$  is an estimate of the variance associated with each response.

### Model selection

**Vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

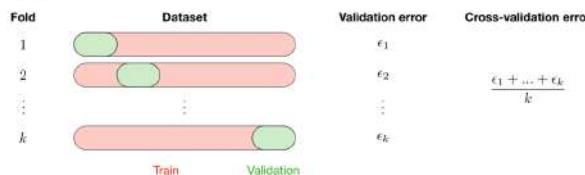
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



**Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
- Training on $k-1$ folds and assessment on the remaining one - Generally $k = 5$ or 10	- Training on $n-p$ observations and assessment on the $p$ remaining ones - Case $p = 1$ is called leave-one-out

The most commonly used method is called  $k$ -fold cross-validation and splits the training data into  $k$  folds to validate the model on one fold while training the model on the  $k-1$  other folds, all of this  $k$  times. The error is then averaged over the  $k$  folds and is named cross-validation error.



**Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda [(1-\alpha)\ \theta\ _1 + \alpha\ \theta\ _2^2]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

**Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

### Diagnostics

**Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

**Variance** – The variance of a model is the variability of the model prediction for given data points.

**Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

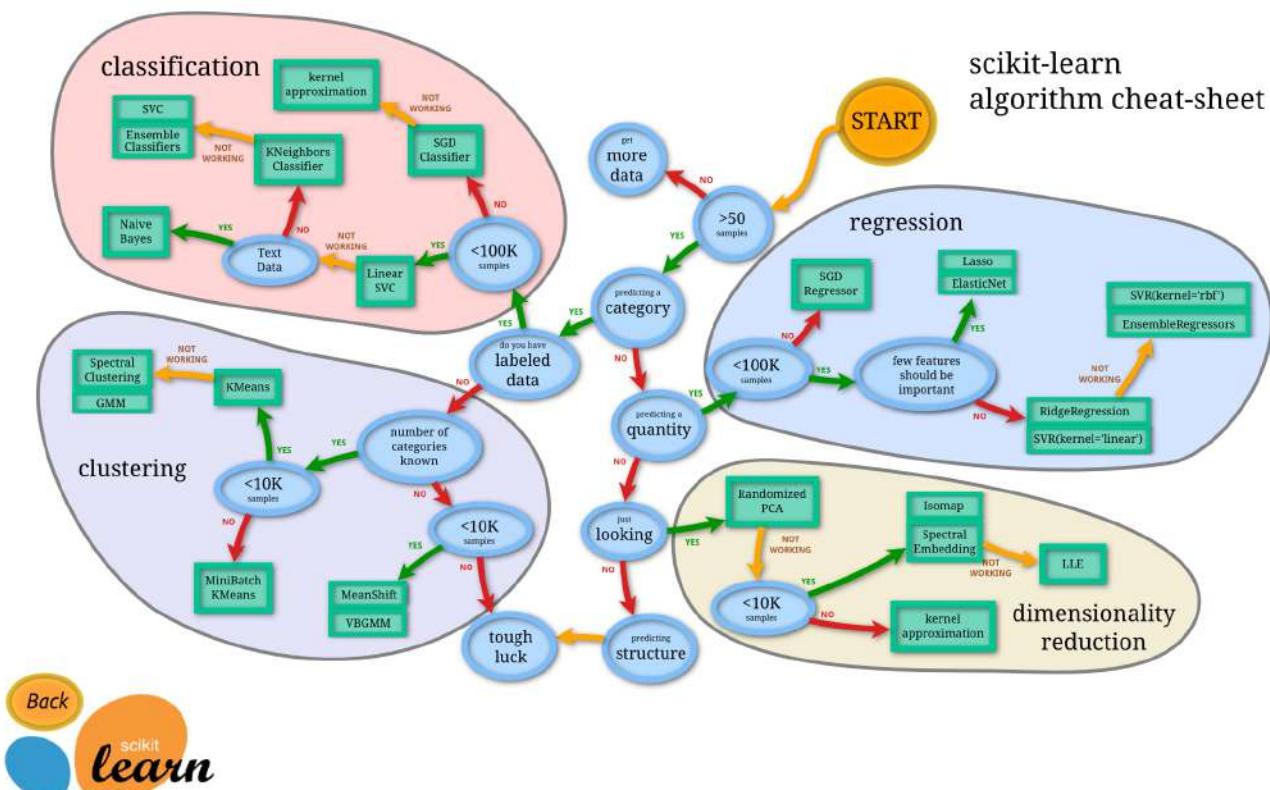
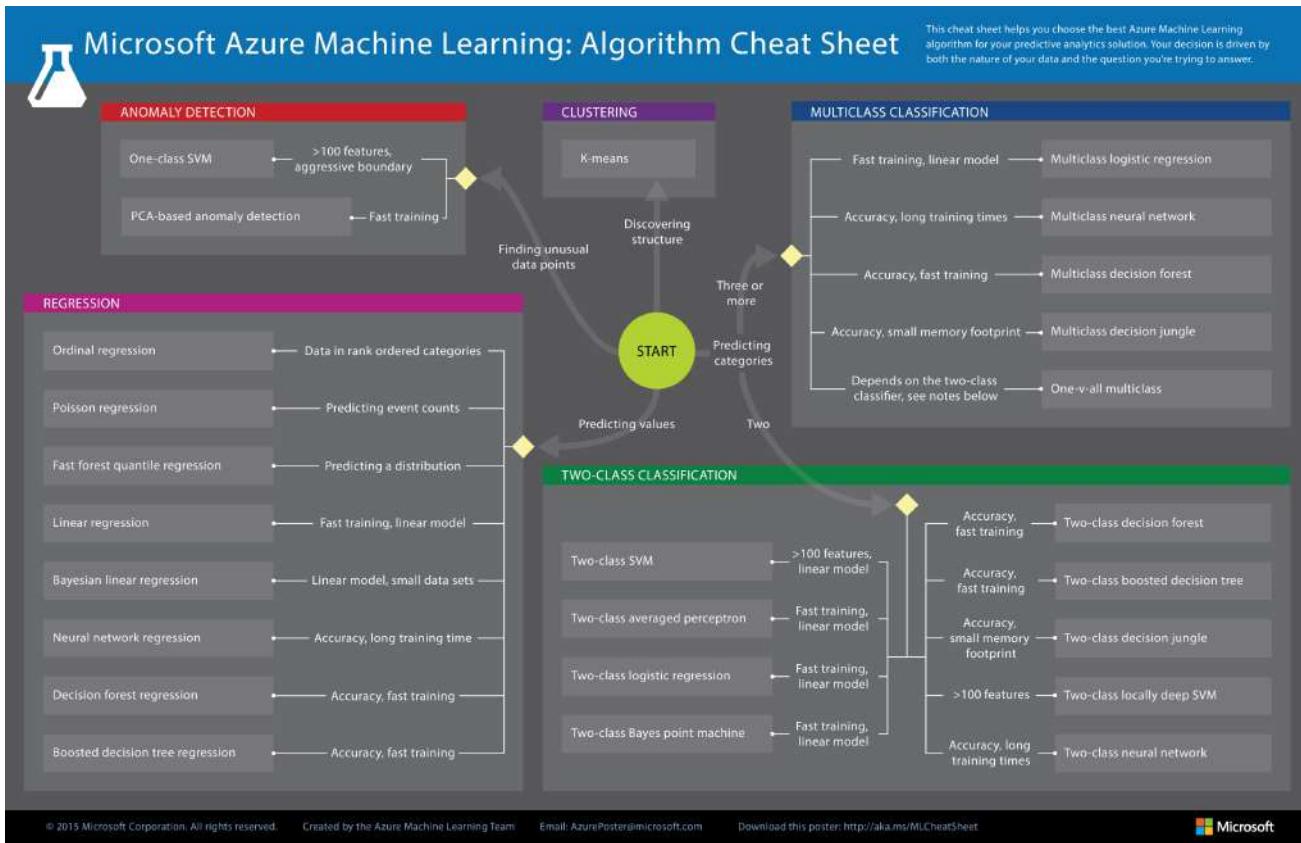
Symptoms	Underfitting	Just right	Overfitting
	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

Classification		
Deep learning		
Remedies	<ul style="list-style-type: none"> <li>- Complexify model</li> <li>- Add more features</li> <li>- Train longer</li> </ul>	<ul style="list-style-type: none"> <li>- Regularize</li> <li>- Get more data</li> </ul>

❑ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❑ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

## Choosing the right model

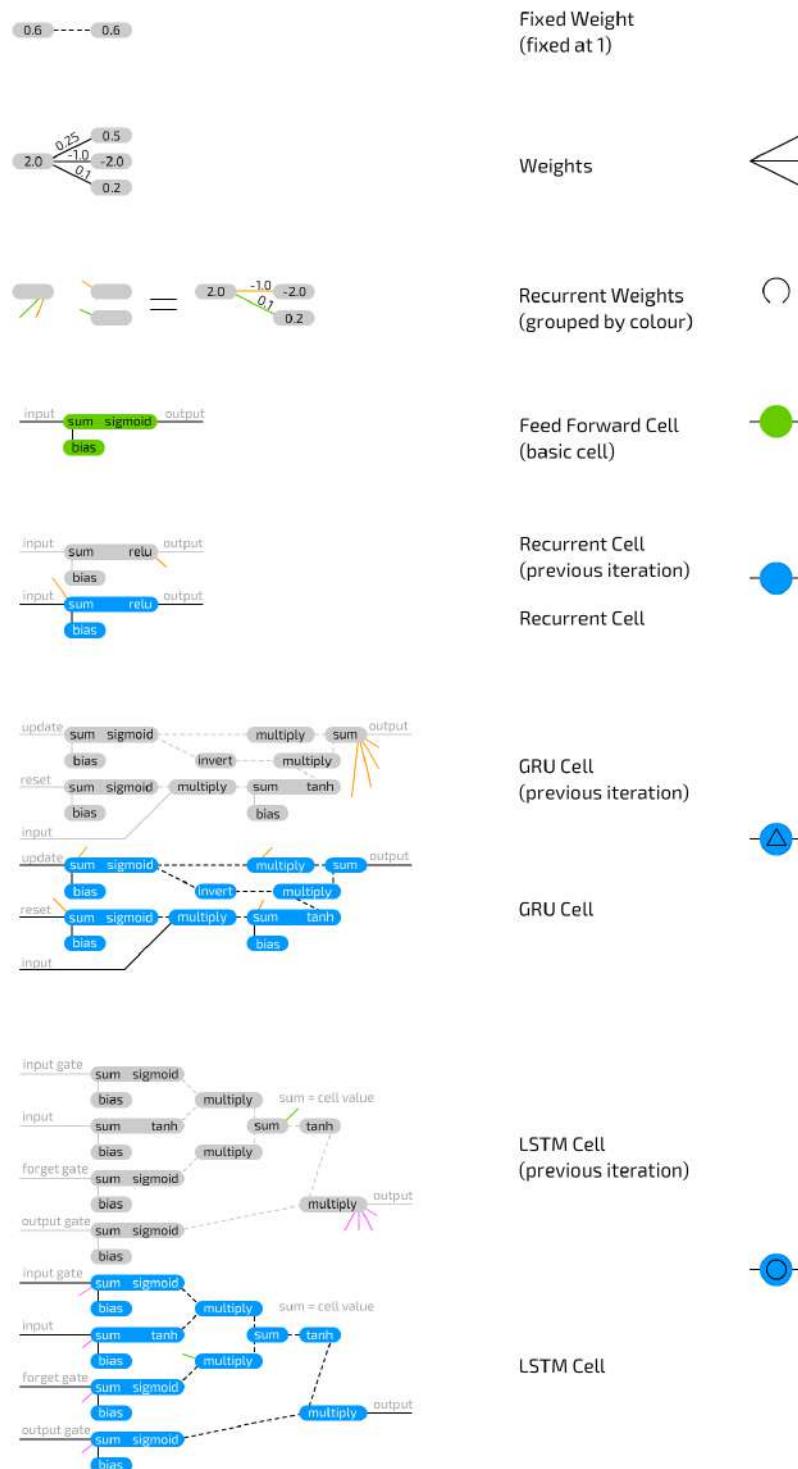


# Deep Learning

## Neural Nets

An informative chart to build  
**Neural Network Cells**

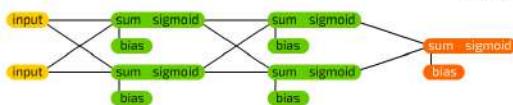
©2016 Fjodor van Veen - asimovinstitute.org



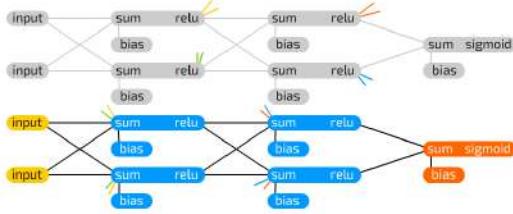
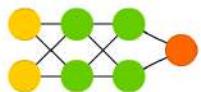
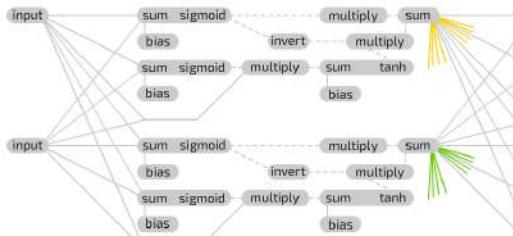
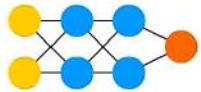
An informative chart to build

# Neural Network Graphs

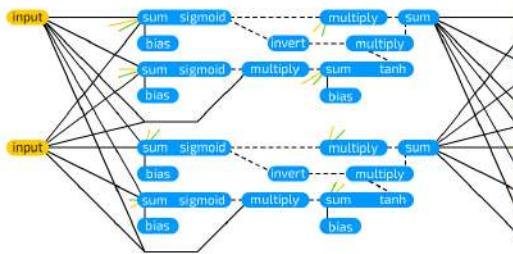
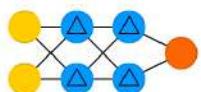
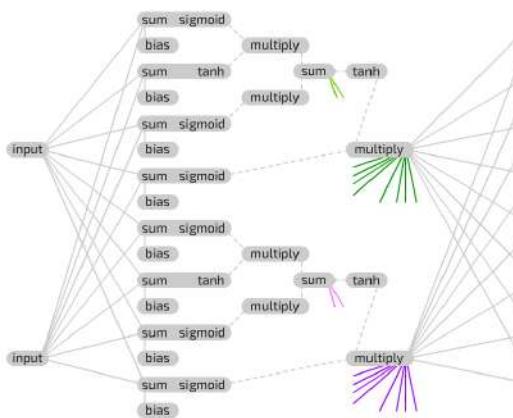
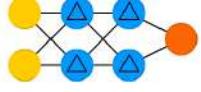
©2016 Fjodor van Veen - asimovinstitute.org



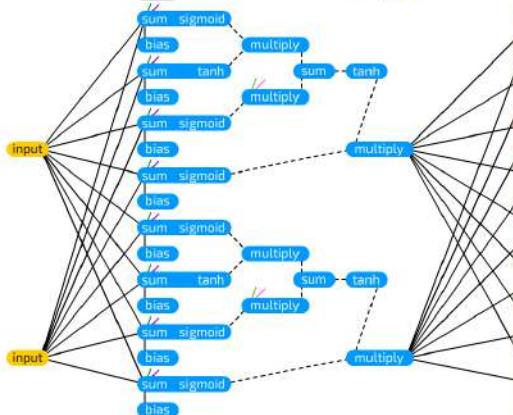
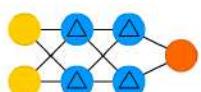
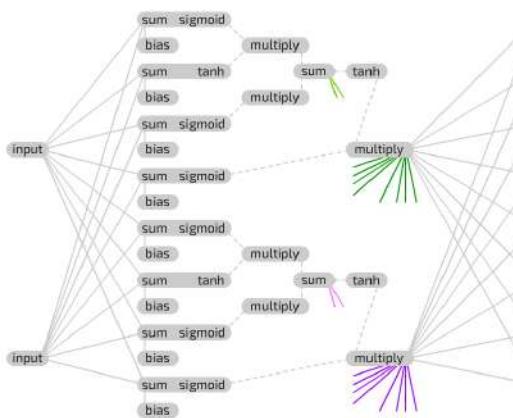
Deep Feed Forward Example

Deep Recurrent Example  
(previous iteration)

Deep Recurrent Example

Deep GRU Example  
(previous iteration)

Deep GRU Example

Deep LSTM Example  
(previous iteration)

Deep LSTM Example

- Keras RStudio (PDF)

# Python

- Keras (PDF)

## Python For Data Science Cheat Sheet

Keras

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



### Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

#### A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.randint(1000,100)
>>> labels = np.random.randint(2, size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(12,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1,activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

#### Data

[Also see NumPy, Pandas & Scikit-Learn](#)

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

#### Keras Data Sets

```
>>> from keras.datasets import boston_housing,
      mnist,
      cifar10,
      cifar100
>>> (x_train,y_train) = mnist.load_data()
>>> (x_train2,y_train2),(x_test,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = cifar100.load_data()
>>> num_classes = 10
```

#### Other

```
>>> from urllib.request import urlopen
>>> url = "http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data", delimiter=",")
>>> X = data[1:,0:8]
>>> y = data [1,8]
```

#### Preprocessing

##### Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

##### One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> y_train3 = to_categorical(y_train3, num_classes)
>>> y_test3 = to_categorical(y_test3, num_classes)
```

## Model Architecture

### Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

### Multilayer Perceptron (MLP)

#### Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

#### Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

#### Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

### Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model1.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model1.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

### Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(10000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

[Also see NumPy & Scikit-Learn](#)

## Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> x_train,x_test,y_train,y_test = train_test_split(x,
                                                y,
                                                test_size=0.33,
                                                random_state=42)
```

## Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(xtrain)
>>> standardised_X = scaler.transform(x_train2)
>>> standardised_X_test = scaler.transform(x_test2)
```

## Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape  
Model summary representation  
Model configuration  
List all weight tensors in the model

## Compile Model

<b>MLP: Binary Classification</b>	<code>&gt;&gt;&gt; model.compile(optimizer='adam',                   loss='binary_crossentropy',                   metrics=['accuracy'])</code>
<b>MLP: Multi-Class Classification</b>	<code>&gt;&gt;&gt; model.compile(optimizer='rmsprop',                   loss='categorical_crossentropy',                   metrics=['accuracy'])</code>
<b>MLP: Regression</b>	<code>&gt;&gt;&gt; model.compile(optimizer='rmsprop',                   loss='mse',                   metrics=['mse'])</code>

## Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

## Model Training

```
>>> model3.fit(x_train4,
              y_train4,
              batch_size=32,
              epochs=15,
              verbose=1,
              validation_data=(x_test4,y_test4))
```

## Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

## Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4, batch_size=32)
```

## Save/Reload Models

```
>>> from keras.models import load_model
>>> model3.save("model.h5")
>>> my_model = load_model("my_model.h5")
```

## Model Fine-tuning

### Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
```

## Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model1.fit(x_train4,
              y_train4,
              batch_size=32,
              epochs=15,
              validation_data=(x_test4,y_test4),
              callbacks=[early_stopping_monitor])
```

**DataCamp**

Learn Python for Data Science interactively

From @afshinea:

- Deep Learning Basics (PDF)

## VIP Cheatsheet: Deep Learning

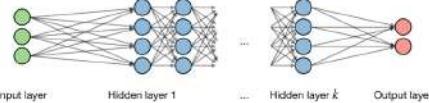
Afshine AMIDI and Shervine AMIDI

September 15, 2018

### Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

**Architecture** – The vocabulary around neural networks architectures is described in the figure below:

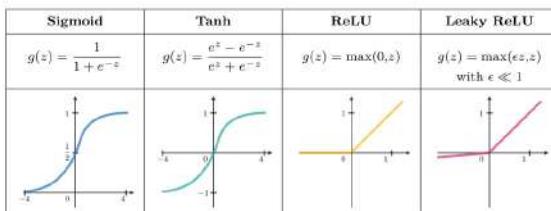


By noting  $i$  the  $i^{th}$  layer of the network and  $j$  the  $j^{th}$  hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

where we note  $w$ ,  $b$ ,  $z$  the weight, bias and output respectively.

**Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:



**Cross-entropy loss** – In the context of neural networks, the cross-entropy loss  $L(z, y)$  is commonly used and is defined as follows:

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)]$$

As a result, the weight is updated as follows:

$$w \leftarrow w - \eta \frac{\partial L(z, y)}{\partial w}$$

**Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

**Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability  $p$  or kept with probability  $1 - p$ .

### Convolutional Neural Networks

**Convolutional layer requirement** – By noting  $W$  the input volume size,  $F$  the size of the convolutional layer neurons,  $P$  the amount of zero padding, then the number of neurons  $N$  that fit in a given volume is such that:

$$N = \frac{W - F + 2P}{S} + 1$$

**Batch normalization** – It is a step of hyperparameter  $\gamma, \beta$  that normalizes the batch  $\{x_i\}$ . By noting  $\mu_B, \sigma_B^2$  the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

### Recurrent Neural Networks

**Types of gates** – Here are the different types of gates that we encounter in a typical recurrent neural network:

Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

**LSTM** – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding ‘forget’ gates.

### Reinforcement Learning and Control

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

**Markov decision processes** – A Markov decision process (MDP) is a 5-tuple  $(S, A, \{P_{sa}\}, \gamma, R)$  where:

- $S$  is the set of states
- $A$  is the set of actions
- $\{P_{sa}\}$  are the state transition probabilities for  $s \in S$  and  $a \in A$
- $\gamma \in [0, 1]$  is the discount factor
- $R : S \times A \rightarrow \mathbb{R}$  or  $R : S \rightarrow \mathbb{R}$  is the reward function that the algorithm wants to maximize

**Policy** – A policy  $\pi$  is a function  $\pi : S \rightarrow A$  that maps states to actions.

Remark: we say that we execute a given policy  $\pi$  if given a state  $s$  we take the action  $a = \pi(s)$ .

**Value function** – For a given policy  $\pi$  and a given state  $s$ , we define the value function  $V^\pi$  as follows:

$$V^\pi(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi]$$

**Bellman equation** – The optimal Bellman equations characterizes the value function  $V^\pi$  of the optimal policy  $\pi^*$ :

$$V^{\pi^*}(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

Remark: we note that the optimal policy  $\pi^*$  for a given state  $s$  is such that:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

**Value iteration algorithm** – The value iteration algorithm is in two steps:

- [Convolutional Neural Networks \(PDF\)](#)

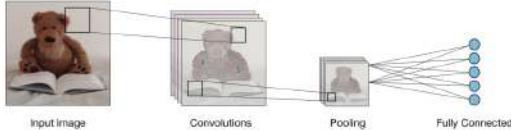
## VIP Cheatsheet: Convolutional Neural Networks

Afshine AMIDI and Shervine AMIDI

November 26, 2018

### Overview

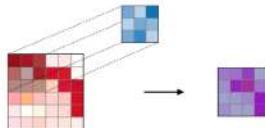
**Architecture of a traditional CNN** – Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters that are described in the next sections.

### Types of layer

**Convolution layer (CONV)** – The convolution layer (CONV) uses filters that perform convolution operations as it scanning the input  $I$  with respect to its dimensions. Its hyperparameters include the filter size  $F$  and stride  $S$ . The resulting output  $O$  is called *feature map* or *activation map*.

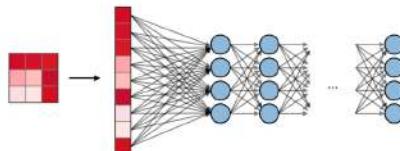


Remark: the convolution step can be generalized to the 1D and 3D cases as well.

**Pooling (POOL)** – The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Purpose	Max pooling	Average pooling
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

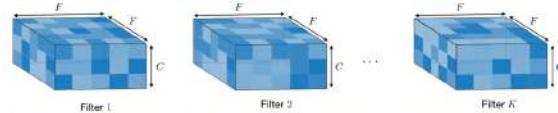
**Fully Connected (FC)** – The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



### Filter hyperparameters

The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

**Dimensions of a filter** – A filter of size  $F \times F$  applied to an input containing  $C$  channels is a  $F \times F \times C$  volume that performs convolutions on an input of size  $I \times I \times C$  and produces an output feature map (also called activation map) of size  $O \times O \times 1$ .



Remark: the application of  $K$  filters of size  $F \times F$  results in an output feature map of size  $O \times O \times K$ .

**Stride** – For a convolutional or a pooling operation, the stride  $S$  denotes the number of pixels by which the window moves after each operation.



**Zero-padding** – Zero-padding denotes the process of adding  $P$  zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

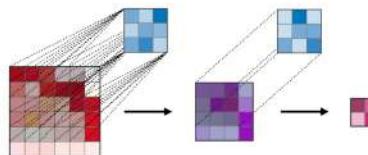
	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lceil \frac{S(\frac{I}{S}-1)-I+F-S}{2} \right\rceil$ $P_{\text{end}} = \left\lceil \frac{S(\frac{I}{S})-I+F-S}{2} \right\rceil$	$P_{\text{start}} \in [0, F-1]$ $P_{\text{end}} = F-1$
Illustration			
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ - Output size is mathematically convenient - Also called 'half' padding	- Maximum padding such that end convolutions are applied on the limits of the input - Filter 'sees' the input end-to-end

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	$N_{\text{in}}$
Output size	$O \times O \times K$	$O \times O \times C$	$N_{\text{out}}$
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	- One bias parameter per filter - In most cases, $S < F$ - A common choice for $K$ is $2C$	- Pooling operation done channel-wise - In most cases, $S = F$	- Input is flattened - One bias parameter per neuron - The number of FC neurons is free of structural constraints

**Receptive field** – The receptive field at layer  $k$  is the area denoted  $R_k \times R_k$  of the input that each pixel of the  $k$ -th activation map can 'see'. By calling  $F_j$  the filter size of layer  $j$  and  $S_i$  the stride value of layer  $i$  and with the convention  $S_0 = 1$ , the receptive field at layer  $k$  can be computed with the formula:

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

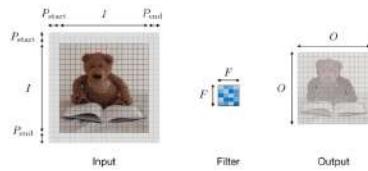
In the example below, we have  $F_1 = F_2 = 3$  and  $S_1 = S_2 = 1$ , which gives  $R_2 = 1 + 2 \cdot 1 + 2 \cdot 1 = 5$ .



### Tuning hyperparameters

**Parameter compatibility in convolution layer** – By noting  $I$  the length of the input volume size,  $F$  the length of the filter,  $P$  the amount of zero padding,  $S$  the stride, then the output size  $O$  of the feature map along that dimension is given by:

$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$

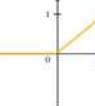
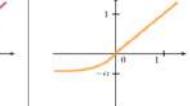


Remark: often times,  $P_{\text{start}} = P_{\text{end}} \triangleq P$ , in which case we can replace  $P_{\text{start}} + P_{\text{end}}$  by  $2P$  in the formula above.

**Understanding the complexity of the model** – In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

### Commonly used activation functions

**Rectified Linear Unit** – The rectified linear unit layer (ReLU) is an activation function  $g$  that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

☐ **Softmax** – The softmax step can be seen as a generalized logistic function that takes as input a vector of scores  $x \in \mathbb{R}^n$  and outputs a vector of output probability  $p \in \mathbb{R}^n$  through a softmax function at the end of the architecture. It is defined as follows:

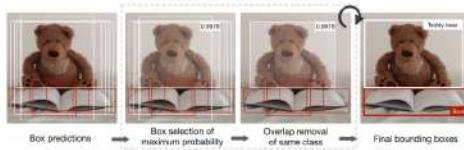
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

### Object detection

☐ **Types of models** – There are 3 main types of object recognition algorithms, for which the nature of what is predicted is different. They are described in the table below:

Image classification	Classification w. localization	Detection
		
- Classifies a picture - Predicts probability of object	- Detects object in a picture - Predicts probability of object and where it is located	- Detects up to several objects in a picture - Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

☐ **Detection** – In the context of object detection, different methods are used depending on whether we just want to locate the object or detect a more complex shape in the image. The two main ones are summed up in the table below:



☐ **YOLO** – You Only Look Once (YOLO) is an object detection algorithm that performs the following steps:

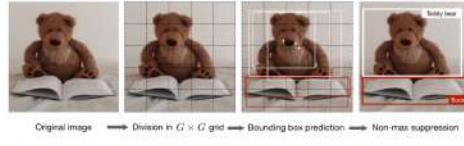
- Step 1: Divide the input image into a  $G \times G$  grid.
- Step 2: For each grid cell, run a CNN that predicts  $y$  of the following form:

$$\hat{y} = \left[ p_c, b_x, b_y, b_w, b_h, c_1, c_2, \dots, c_p, \dots \right]^T \in \mathbb{R}^{G \times G \times k \times (5 + p)}$$

repeated  $k$  times

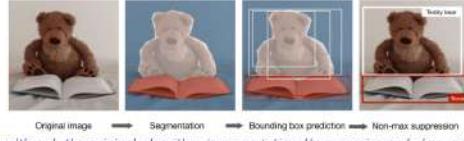
where  $p_c$  is the probability of detecting an object,  $b_x, b_y, b_w, b_h$  are the properties of the detected bounding box,  $c_1, \dots, c_p$  is a one-hot representation of which of the  $p$  classes were detected, and  $k$  is the number of anchor boxes.

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



Remark: when  $p_c = 0$ , then the network does not detect any object. In that case, the corresponding predictions  $b_x, \dots, c_p$  have to be ignored.

☐ **R-CNN** – Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.

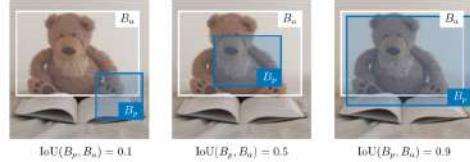


Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

Bounding box detection	Landmark detection
Detects the part of the image where the object is located	<ul style="list-style-type: none"> <li>- Detects a shape or characteristics of an object (e.g. eyes)</li> <li>- More granular</li> </ul>

☐ **Intersection over Union** – Intersection over Union, also known as IoU, is a function that quantifies how correctly positioned a predicted bounding box  $B_p$  is over the actual bounding box  $B_a$ . It is defined as:

$$\text{IoU}(B_p, B_a) = \frac{|B_p \cap B_a|}{|B_p \cup B_a|}$$



Remark: we always have  $\text{IoU} \in [0, 1]$ . By convention, a predicted bounding box  $B_p$  is considered as being reasonably good if  $\text{IoU}(B_p, B_a) \geq 0.5$ .

☐ **Anchor boxes** – Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometric form.

☐ **Non-max suppression** – The non-max suppression technique aims at removing duplicate overlapping bounding boxes of a same object by selecting the most representative ones. After having removed all boxes having a probability prediction lower than 0.6, the following steps are repeated while there are boxes remaining:

- Step 1: Pick the box with the largest prediction probability.
- Step 2: Discard any box having an  $\text{IoU} \geq 0.5$  with the previous box.

### Face verification and recognition

☐ **Types of models** – Two main types of model are summed up in table below:

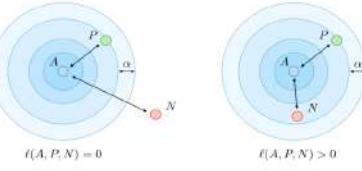
Face verification	Face recognition
<ul style="list-style-type: none"> <li>- Is this the correct person?</li> <li>- One-to-one lookup</li> </ul> <p>Query  Reference </p>	<ul style="list-style-type: none"> <li>- Is this one of the <math>K</math> persons in the database?</li> <li>- One-to-many lookup</li> </ul> <p>Query  Database </p>

☐ **One Shot Learning** – One Shot Learning is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are. The similarity function applied to two images is often noted  $d(\text{image 1}, \text{image 2})$ .

☐ **Siamese Network** – Siamese Networks aim at learning how to encode images to then quantify how different two images are. For a given input image  $x^{(t)}$ , the encoded output is often noted as  $f(x^{(t)})$ .

☐ **Triplet loss** – The triplet loss  $\ell$  is a loss function computed on the embedding representation of a triplet of images  $A$  (anchor),  $P$  (positive) and  $N$  (negative). The anchor and the positive example belong to the same class, while the negative example to another one. By calling  $\alpha \in \mathbb{R}^+$  the margin parameter, this loss is defined as follows:

$$\ell(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$



### Neural style transfer

☐ **Motivation** – The goal of neural style transfer is to generate an image  $G$  based on a given content  $C$  and a given style  $S$ .



**Activation** – In a given layer  $l$ , the activation is noted  $a^{[l]}$  and is of dimensions  $n_H \times n_w \times n_c$

**Content cost function** – The content cost function  $J_{\text{content}}(C, G)$  is used to determine how the generated image  $G$  differs from the original content image  $C$ . It is defined as follows:

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l]}(C) - a^{[l]}(G)\|^2$$

**Style matrix** – The style matrix  $G^{[l]}$  of a given layer  $l$  is a Gram matrix where each of its elements  $G_{kk'}^{[l]}$  quantifies how correlated the channels  $k$  and  $k'$  are. It is defined with respect to activations  $a^{[l]}$  as follows:

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

*Remark: the style matrix for the style image and the generated image are noted  $G^{[l](S)}$  and  $G^{[l](G)}$  respectively.*

**Style cost function** – The style cost function  $J_{\text{style}}(S, G)$  is used to determine how the generated image  $G$  differs from the style  $S$ . It is defined as follows:

$$J_{\text{style}}(S, G) = \frac{1}{(2n_H n_w n_c)^2} \|G^{[l](S)} - G^{[l](G)}\|_F^2 = \frac{1}{(2n_H n_w n_c)^2} \sum_{k, k'=1}^{n_c} \left( G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \right)^2$$

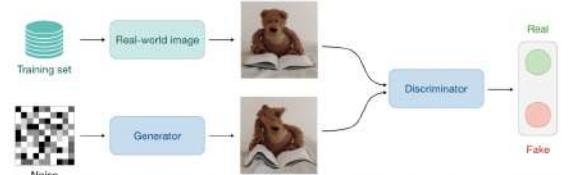
**Overall cost function** – The overall cost function is defined as being a combination of the content and style cost functions, weighted by parameters  $\alpha, \beta$ , as follows:

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

*Remark: a higher value of  $\alpha$  will make the model care more about the content while a higher value of  $\beta$  will make it care more about the style.*

#### Architectures using computational tricks

**Generative Adversarial Network** – Generative adversarial networks, also known as GANs, are composed of a generative and a discriminative model, where the generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true image.



*Remark: use cases using variants of GANs include text to image, music generation and synthesis.*

**ResNet** – The Residual Network architecture (also called ResNet) uses residual blocks with a high number of layers meant to decrease the training error. The residual block has the following characterizing equation:

$$a^{[l+2]} = g(a^{[l]} + s^{[l+2]})$$

**Inception Network** – This architecture uses inception modules and aims at giving a try at different convolutions in order to increase its performance. In particular, it uses the  $1 \times 1$  convolution trick to lower the burden of computation.

\* \* \*

- Recurrent Neural Networks (PDF)

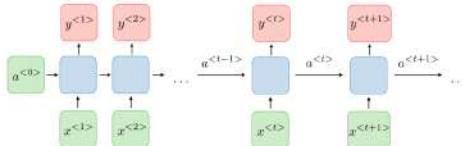
## VIP Cheatsheet: Recurrent Neural Networks

Afshine AMIDI and Shervine AMIDI

November 26, 2018

### Overview

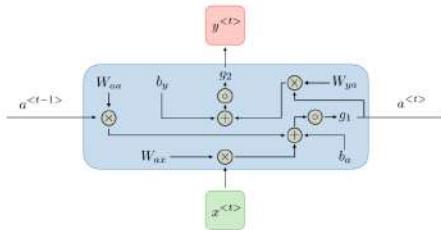
**Architecture of a traditional RNN** – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep  $t$ , the activation  $a^{<t>}$  and the output  $y^{<t>}$  are expressed as follows:

$$a^{<t>} = g_1(W_{ax}a^{<t-1>} + W_{aa}a^{<t-1>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where  $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$  are coefficients that are shared temporally and  $g_1, g_2$  activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
<ul style="list-style-type: none"> <li>Possibility of processing input of any length</li> <li>Model size not increasing with size of input</li> <li>Computation takes into account historical information</li> <li>Weights are shared across time</li> </ul>	<ul style="list-style-type: none"> <li>Computation being slow</li> <li>Difficulty of accessing information from a long time ago</li> <li>Cannot consider any future input for the current state</li> </ul>

**Applications of RNNs** – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

**Loss function** – In the case of a recurrent neural network, the loss function  $\mathcal{L}$  of all time

steps is defined based on the loss at every time step as follows:

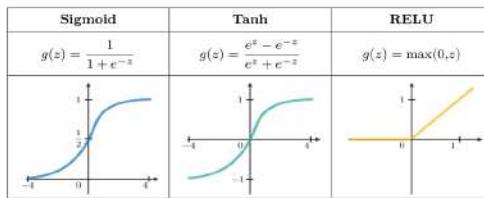
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_x} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

**Backpropagation through time** – Backpropagation is done at each point in time. At timestep  $T$ , the derivative of the loss  $\mathcal{L}$  with respect to weight matrix  $W$  is expressed as follows:

$$\frac{\partial \mathcal{L}(T)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}(T)}{\partial W} \Big|_{(t)}$$

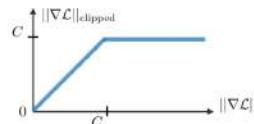
### Handling long term dependencies

**Commonly used activation functions** – The most common activation functions used in RNN modules are described below:



**Vanishing/exploding gradient** – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

**Gradient clipping** – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



**Types of gates** – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted  $\Gamma$  and are equal to:

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

where  $W, U, b$  are coefficients specific to the gate and  $\sigma$  is the sigmoid function. The main ones are summed up in the table below:

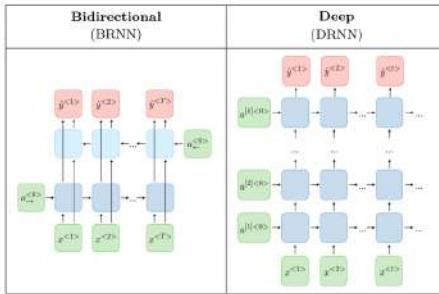
Type of gate	Role	Used in
Update gate $\Gamma_u$	How much past should matter now?	GRU, LSTM
Relevance gate $\Gamma_r$	Drop previous information?	GRU, LSTM
Forget gate $\Gamma_f$	Erase a cell or not?	LSTM
Output gate $\Gamma_o$	How much to reveal of a cell?	LSTM

**GRU/LSTM** – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$c^{<t>}$	$\tanh(W_c[\Gamma_r * \tilde{c}^{<t-1>} + (1 - \Gamma_u) * c^{<t-1>}] + b_c)$	$\tanh(W_c[\Gamma_r * a^{<t-1>} + x^{<t>}] + b_c)$
$c^{<t-1>}$	$\Gamma_u * \tilde{c}^{<t-1>} + (1 - \Gamma_u) * c^{<t-1>}$	$\Gamma_u * \tilde{c}^{<t-1>} + \Gamma_f * c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o * c^{<t>}$
Dependencies		

Remark: the sign  $*$  denotes the element-wise multiplication between two vectors.

**Variants of RNNs** – The table below sums up the other commonly used RNN architectures:



### Learning word representation

In this section, we note  $V$  the vocabulary and  $|V|$  its size.

**Representation techniques** – The two main ways of representing words are summed up in the table below:

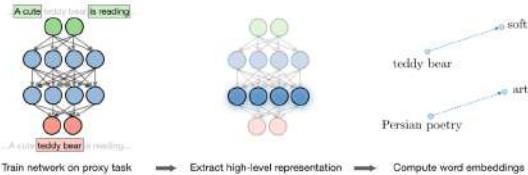
1-hot representation	Word embedding
<ul style="list-style-type: none"> <li>- Noted <math>\phi_w</math></li> <li>- Naive approach, no similarity information</li> </ul>	<ul style="list-style-type: none"> <li>- Noted <math>e_w</math></li> <li>- Takes into account words similarity</li> </ul>

**Embedding matrix** – For a given word  $w$ , the embedding matrix  $E$  is a matrix that maps its 1-hot representation  $\phi_w$  to its embedding  $e_w$  as follows:

$$e_w = E\phi_w$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

**Word2vec** – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



**Skip-gram** – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word  $t$  happening with a context word  $c$ . By noting  $\theta_t$  a parameter associated with  $t$ , the probability  $P(t|c)$  is given by:

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

**Negative sampling** – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of  $k$  negative examples and 1 positive example. Given a context word  $c$  and a target word  $t$ , the prediction is expressed by:

$$P(y=1|c,t) = \sigma(\theta_t^T e_c)$$

Remark: this method is less computationally expensive than the skip-gram model.

**GloVe** – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurrence matrix  $X$  where each  $X_{i,j}$  denotes the number of times that a target  $i$  occurred with a context  $j$ . Its cost function  $J$  is as follows:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{i,j})(\theta_i^T e_j + b_i + b'_j - \log(X_{i,j}))^2$$

here  $f$  is a weighting function such that  $X_{i,j} = 0 \implies f(X_{i,j}) = 0$ .

Given the symmetry that  $e$  and  $\theta$  play in this model, the final word embedding  $e_w^{(\text{final})}$  is given by:

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

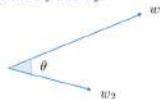
Remark: the individual components of the learned word embeddings are not necessarily interpretable.

### Comparing words

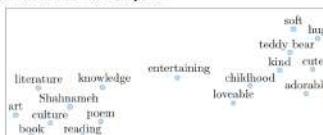
**Cosine similarity** – The cosine similarity between words  $w_1$  and  $w_2$  is expressed as follows:

$$\text{similarity} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta)$$

Remark:  $\theta$  is the angle between words  $w_1$  and  $w_2$ .



**t-SNE** – t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



### Language model

**Overview** – A language model aims at estimating the probability of a sentence  $P(y)$ .

**n-gram model** – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

**Perplexity** – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words  $T$ . The perplexity is such that the lower, the better and is defined as follows:

$$PP = \prod_{t=1}^T \left( \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Remark: PP is commonly used in t-SNE.

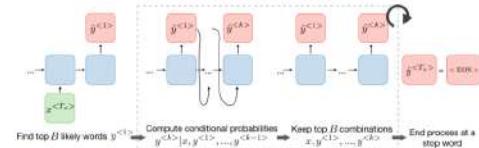
### Machine translation

**Overview** – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence  $y$  such that:

$$y = \arg \max_{y^{<1>} \dots y^{<T_y>}} P(y^{<1>} \dots y^{<T_y>} | x)$$

**Beam search** – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence  $y$  given an input  $x$ .

- Step 1: Find top  $B$  likely words  $y^{<1>}$
- Step 2: Compute conditional probabilities  $y^{<2>} | x, y^{<1>} \dots, y^{<k-1>}$
- Step 3: Keep top  $B$  combinations  $x, y^{<1>} \dots, y^{<k>}$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

**Beam width** – The beam width  $B$  is a parameter for beam search. Large values of  $B$  yield better result but with slower performance and increased memory. Small values of  $B$  lead to worse results but is less computationally intensive. A standard value for  $B$  is around 10.

**Length normalization** – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective} = \frac{1}{T_y} \sum_{t=1}^{T_y} \log \left[ p(y^{<t>} | x, y^{<1>} \dots, y^{<t-1>}) \right]$$

Remark: the parameter  $\alpha$  can be seen as a softener, and its value is usually between 0.5 and 1.

**Error analysis** – When obtaining a predicted translation  $\hat{y}$  that is bad, one can wonder why we did not get a good translation  $y^*$  by performing the following error analysis:

Case	$P(y^* x) > P(\hat{y} x)$	$P(y^* x) \leq P(\hat{y} x)$
Root cause	Beam search faulty	RNN faulty
Remedies	Increase beam width	<ul style="list-style-type: none"> <li>- Try different architecture</li> <li>- Regularize</li> <li>- Get more data</li> </ul>

**Bleu score** – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp \left( \frac{1}{n} \sum_{k=1}^n p_k \right)$$

where  $p_n$  is the bleu score on n-gram only defined as follows:

$$p_n = \frac{\sum_{\substack{n\text{-gram} \in \hat{y}}} \text{count}_{clip}(n\text{-gram})}{\sum_{n\text{-gram} \in \hat{y}} \text{count}(n\text{-gram})}$$

*Remark:* a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.

### Attention

□ **Attention model** – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting  $\alpha^{<t,t'>}$  the amount of attention that the output  $y^{<t>}$  should pay to the activation  $a^{<t'>}$  and  $c^{<t'>}$  the context at time  $t$ , we have:

$$e^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \quad \text{with} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

*Remark:* the attention scores are commonly used in image captioning and machine translation.



A cute teddy bear is reading Persian literature



A cute teddy bear is reading Persian literature

□ **Attention weight** – The amount of attention that the output  $y^{<i>}$  should pay to the activation  $a^{<t'>}$  is given by  $\alpha^{<t,t'>}$  computed as follows:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

*Remark:* computation complexity is quadratic with respect to  $T_x$ .

\* \* \*

- [Tips and Tricks \(PDF\)](#)

## VIP Cheatsheet: Tips and Tricks

Afshine AMIDI and Shervine AMIDI

November 26, 2018

### Data processing

**Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Random crop
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row
Color shift	Noise addition	Information loss	Contrast change
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

**Batch normalization** – It is a step of hyperparameter  $\gamma, \beta$  that normalizes the batch  $\{x_i\}$ . By noting  $\mu_B, \sigma_B^2$  the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

### Training a neural network

**Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

**Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

**Loss function** – In order to quantify how a given model performs, the loss function  $L$  is usually used to evaluate to what extent the actual outputs  $y$  are correctly predicted by the model outputs  $z$ .

**Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss  $L(z,y)$  is commonly used and is defined as follows:

$$L(z,y) = -[y \log(z) + (1-y) \log(1-z)]$$

**Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight  $w$  is computed using the chain rule.

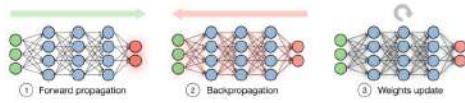
$$\frac{\partial L}{\partial f(x)} \cdot \frac{\partial f(x)}{\partial x} \quad \begin{matrix} f \\ \longleftarrow x \quad f(x) \end{matrix}$$

Using this method, each weight is updated with the rule:

$$w \leftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}$$

**Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.



### Parameter tuning

**Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

**Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

**Learning rate** – The learning rate, often noted  $\alpha$  or sometimes  $\eta$ , indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

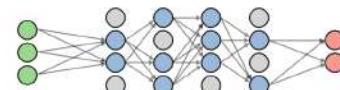
**Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of $w$	Update of $b$
Momentum	- Dampens oscillations - Improvement to SGD - 2 parameters to tune	$w \leftarrow w - \alpha v_{dw}$	$b \leftarrow b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Speeds up learning algorithm by controlling oscillations	$w \leftarrow w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \leftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Most popular method - 4 parameters to tune	$w \leftarrow w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \leftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

### Regularization

**Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability  $p > 0$ . It forces the model to avoid relying too much on particular sets of features.

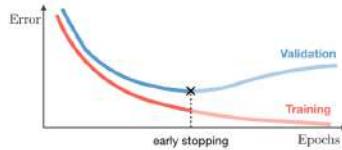


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter  $1-p$ .

**Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
$\dots + \lambda   \theta  _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda   \theta  _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda [(1 - \alpha)  \theta  _1 + \alpha  \theta  _2^2]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

**Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



#### Good practices

**Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

**Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

	Numerical gradient	Analytical gradient
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	<ul style="list-style-type: none"> <li>- Expensive; loss has to be computed two times per dimension</li> <li>- Used to verify correctness of analytical implementation</li> <li>- Trade-off in choosing <math>h</math>: not too small (numerical instability) nor too large (poor gradient approx.)</li> </ul>	<ul style="list-style-type: none"> <li>- 'Exact' result</li> <li>- Direct computation</li> <li>- Used in the final implementation</li> </ul>

\* \* \*

# SQL

- SQL cheatsheet by sqltutorial (PDF)



## SQL CHEAT SHEET <http://www.sqltutorial.org>

### QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t ORDER BY c1 LIMIT n OFFSET offset;
Skip offset of rows and return the next n rows

SELECT c1, aggregate(c2) FROM t GROUP BY c1;
Group rows using an aggregate function

SELECT c1, aggregate(c2) FROM t GROUP BY c1 HAVING condition;
Filter groups using HAVING clause
```

### QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2 FROM t1 INNER JOIN t2 ON condition;
Inner join t1 and t2

SELECT c1, c2 FROM t1 LEFT JOIN t2 ON condition;
Left join t1 and t2

SELECT c1, c2 FROM t1 RIGHT JOIN t2 ON condition;
Right join t1 and t2

SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 ON condition;
Perform full outer join

SELECT c1, c2 FROM t1 CROSS JOIN t2;
Produce a Cartesian product of rows in tables

SELECT c1, c2 FROM t1, t2;
Another way to perform cross join

SELECT c1, c2 FROM t1 A INNER JOIN t2 B ON condition;
Join t1 to itself using INNER JOIN clause
```

### USING SQL OPERATORS

```
SELECT c1, c2 FROM t1 UNION [ALL] SELECT c1, c2 FROM t2;
Combine rows from two queries

SELECT c1, c2 FROM t1 INTERSECT SELECT c1, c2 FROM t2;
Return the intersection of two queries

SELECT c1, c2 FROM t1 MINUS SELECT c1, c2 FROM t2;
Subtract a result set from another result set

SELECT c1, c2 FROM t1 WHERE c1 [NOT] LIKE pattern;
Query rows using pattern matching %, _

SELECT c1, c2 FROM t1 WHERE c1 [NOT] IN value_list;
Query rows in a list

SELECT c1, c2 FROM t1 WHERE c1 BETWEEN low AND high;
Query rows between two values

SELECT c1, c2 FROM t1 WHERE c1 IS [NOT] NULL;
Check if values in a table is NULL or not
```

## SQL CHEAT SHEET <http://www.sqltutorial.org>



### MANAGING TABLES

```
CREATE TABLE t (id INT PRIMARY KEY, name VARCHAR NOT NULL, price INT DEFAULT 0);
Create a new table with three columns

DROP TABLE t;
Delete the table from the database

ALTER TABLE t ADD column;
Add a new column to the table

ALTER TABLE t DROP COLUMN c;
Drop column c from the table

ALTER TABLE t ADD constraint;
Add a constraint

ALTER TABLE t DROP constraint;
Drop a constraint

ALTER TABLE t1 RENAME TO t2;
Rename a table from t1 to t2

ALTER TABLE t1 RENAME c1 TO c2;
Rename column c1 to c2

TRUNCATE TABLE t;
Remove all data in a table
```

### USING SQL CONSTRAINTS

```
CREATE TABLE t (c1 INT, c2 INT, c3 VARCHAR, PRIMARY KEY (c1,c2));
Set c1 and c2 as a primary key

CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 INT, FOREIGN KEY (c2) REFERENCES t2(c2));
Set c2 column as a foreign key

CREATE TABLE t (c1 INT, c1 INT, UNIQUE(c2,c3));
Make the values in c1 and c2 unique

CREATE TABLE t (c1 INT, c2 INT, CHECK(c1> 0 AND c1 >= c2));
Ensure c1 > 0 and values in c1 >= c2

CREATE TABLE t (c1 INT PRIMARY KEY, c2 VARCHAR NOT NULL);
Set values in c2 column not NULL
```

### MODIFYING DATA

```
INSERT INTO t(column_list) VALUES(value_list);
Insert one row into a table

INSERT INTO t(column_list) VALUES (value_list), (value_list), ...;
Insert multiple rows into a table

INSERT INTO t1(column_list) SELECT column_list FROM t2;
Insert rows from t2 into t1

UPDATE t SET c1 = new_value;
Update new value in the column c1 for all rows

UPDATE t SET c1 = new_value, c2 = new_value WHERE condition;
Update values in the column c1, c2 that match the condition

DELETE FROM t;
Delete all data in a table

DELETE FROM t WHERE condition;
Delete subset of rows in a table
```

## SQL CHEAT SHEET <http://www.sqltutorial.org>

**MANAGING VIEWS**

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
Create a new view that consists of c1 and c2
```

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
Create a new view with check option
```

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
Create a recursive view
```

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
Create a temporary view
```

```
DROP VIEW view_name;
Delete a view
```

**MANAGING INDEXES**

```
CREATE INDEX idx_name
ON t(c1,c2);
Create an index on c1 and c2 of the table t
```

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
Create a unique index on c3, c4 of the table t
```

```
DROP INDEX idx_name;
Drop an index
```

**MANAGING TRIGGERS**

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE;
EXECUTE stored_procedure;
Create or modify a trigger
```

**WHEN**

- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

**EVENT**

- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

**TRIGGER TYPE**

- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
Create a trigger invoked before a new row is inserted into the person table
```

```
DROP TRIGGER trigger_name;
Delete a specific trigger
```

- SQL cheatsheet by Rebel Labs

### SQL cheat sheet

For more awesome cheat sheets visit [rebellabs.org/](http://rebellabs.org/)



**Basic Queries**

- filter your columns
- **SELECT** col1, col2, col3, ... **FROM** table1
- filter the rows
- **WHERE** col4 = 1 **AND** col5 = 2
- aggregate the data
- **GROUP BY** ...
- limit aggregated data
- **HAVING** count(\*) > 1
- order of the results
- **ORDER BY** col2

Useful keywords for SELECTS:

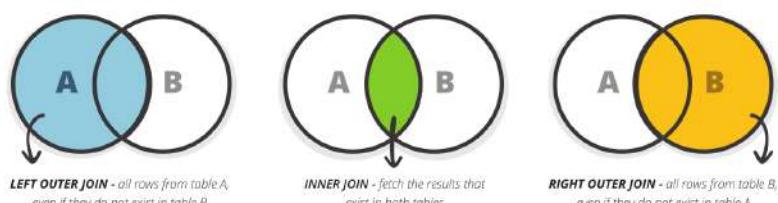
**DISTINCT** - return unique results

**BETWEEN** a **AND** b - limit the range, the values can be numbers, text, or dates

**LIKE** - pattern search within the column text

**IN** (a, b, c) - check if the value is contained among given.

**The Joy of JOINS**



**LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B

**INNER JOIN** - fetch the results that exist in both tables

**RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A

**Updates on JOINed Queries**

You can use **JOINS** in your **UPDATES**

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

**Semi JOINS**

You can use subqueries instead of **JOINS**:

```
SELECT col1, col2 FROM table1 WHERE id IN
(SELECT t1.id FROM table2 WHERE date >
CURRENT_TIMESTAMP)
```

**Indexes**

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

**Don't forget:**

- Avoid overlapping indexes
- Avoid indexing on too many columns

Indexes can speed up **DELETE** and **UPDATE** operations

**Useful Utility Functions**

- convert strings to dates:  
**TO\_DATE** (Oracle, PostgreSQL), **STR\_TO\_DATE** (MySQL)
- return the first non-NULL argument:  
**COALESCE** (col1, col2, "default value")
- return current time:  
**CURRENT\_TIMESTAMP**
- compute set operations on two result sets  
**SELECT** col1, col2 **FROM** table1
   
**UNION / EXCEPT / INTERSECT**
  
**SELECT** col3, col4 **FROM** table2,

**Union** - returns data from both queries

**Except** - rows from the first query that are not present in the second query

**Intersect** - rows that are returned from both queries

**Reporting**

Use aggregation functions

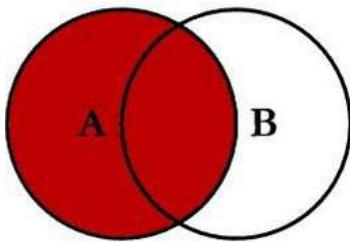
**COUNT** - return the number of rows

**SUM** - cumulate the values

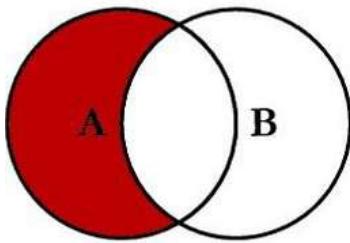
**AVG** - return the average for the group

**MIN / MAX** - smallest / largest value

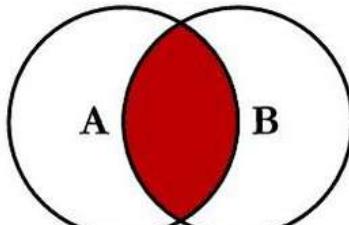
# SQL JOINS



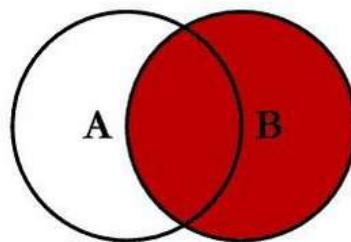
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



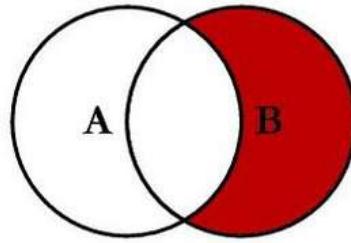
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



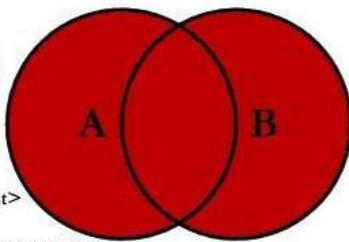
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



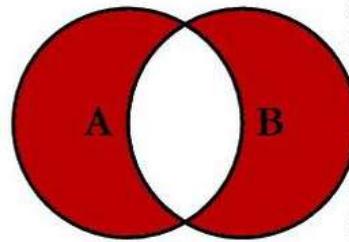
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

## Data Visualization

### Python

- [Matplotlib \(PDF\)](#)

## Python For Data Science Cheat Sheet

### Matplotlib

Learn Python interactively at [www.DataCamp.com](http://www.DataCamp.com)



#### Matplotlib

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



#### 1) Prepare The Data

Also see [Lists & NumPy](#)

##### 1D Data

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

##### 2D Data or Images

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

#### 2) Create Plot

```
>>> import matplotlib.pyplot as plt
Figure
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

##### Axes

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

```
>>> fig.add_axes()
```

```
>>> ax1 = fig.add_subplot(221) # row-col-num
```

```
>>> ax2 = fig.add_subplot(212)
```

```
>>> ngl, axes = plt.subplots(nrows=2, ncols=2)
```

```
>>> fig, axes2 = plt.subplots(ncols=3)
```

#### 3) Plotting Routines

##### 1D Data

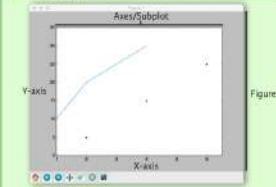
<pre>&gt;&gt;&gt; fig, ax = plt.subplots() &gt;&gt;&gt; lines = ax.plot(x,y) &gt;&gt;&gt; points = ax.scatter(x,y) &gt;&gt;&gt; bars = ax.bar([1, 2, 3], [3, 4, 5]) &gt;&gt;&gt; axes1[0,0].bar([1, 2, 3], [3, 4, 5]) &gt;&gt;&gt; axes1[1,0].bar([0, 1, 2], [2, 3, 4]) &gt;&gt;&gt; axes1[1,1].axhline(0.45) &gt;&gt;&gt; axes1[1,1].axvline(0.45) &gt;&gt;&gt; ax.HLL([x,y,color='blue']) &gt;&gt;&gt; ax.HLL([x,y,color='yellow'])</pre>	Draw points with lines or markers connecting them Draw unconnected points, scaled or colored Plot vertical rectangles (constant width) Plot horizontal rectangles (constant height) Draw a horizontal line across axes Draw a vertical line across axes Draw filled polygons Fill between y-values and o
---	---

##### 2D Data or Images

<pre>&gt;&gt;&gt; fig, ax = plt.subplots() &gt;&gt;&gt; im = ax.imshow(img,                   cmap='gist_earth',                   interpolation='nearest',                   vmin=-2,                   vmax=2)</pre>	Colormapped or RGB arrays
--	---------------------------

## Plot Anatomy & Workflow

### Plot Anatomy



### Workflow

The basic steps to creating plots with matplotlib are:

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

#### 4) Customize Plot

##### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha=0.4)
>>> ax.set_color_cycle(['red','green'])
>>> ax.set_colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

##### Markers

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

##### Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,lw=4)
>>> plt.plot(x,y,**{'ls':"-", "c": "r"})
>>> plt.set_lines(color="c", linewidth=4.0)
```

##### Text & Annotations

```
>>> ax.text(1, 2,
           'Example Graph',
           style='italic')
>>> ax.annotate("Size",
               xy=(8, 0),
               xycoords='data',
               xytext=(10.5, 0),
               textcoords='data',
               arrowprops=dict(arrowstyle=">",
                               connectionstyle="arc3"))
```

##### Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[0,1].streamplot(X,Y,U,V)
```

##### Mathtext

```
>>> plt.title(r'$\Sigma_{i=1}^{100}$', fontsize=20)
```

##### Limits, Legends & Layouts

<pre>&gt;&gt;&gt; ax.margins(0.0, y=0.1) &gt;&gt;&gt; ax.axis('equal') &gt;&gt;&gt; ax.set_xlim(0,10.5), ax.set_ylim(-1.5,1.5) &gt;&gt;&gt; ax.set_xlim(0,10.5)</pre>	Add padding to the plot Set the aspect ratio of the plot to 1 Set limits for x and y-axis Set limits for x-axis
---	--

##### Legends

```
>>> ax.set_title("Example Axes",
                 xlabel="X-Axis",
                 ylabel="Y-Axis",
                 title="Example Axes")
```

##### Ticks

```
>>> ax.xaxis.set_ticks(range(1,5),
                      ticklabelsize=3, loc="left")
```

```
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

##### Subplot Spacing

<pre>&gt;&gt;&gt; fig3.subplots_adjust(hspace=0.5,                         hspace=0.3,                         left=0.125,                         right=0.9,                         top=0.9,                         bottom=0.1)</pre>	Adjust the spacing between subplots
--	-------------------------------------

##### Axes Spines

<pre>&gt;&gt;&gt; ax1.spines['top'].set_visible(False) &gt;&gt;&gt; ax1.spines['bottom'].set_position({'outward', 10})</pre>	Fit subplot(s) to the figure area Make the top axis line for a plot invisible Move the bottom axis line outward
--	---

#### 5) Save Plot

Save Figures	<pre>&gt;&gt;&gt; plt.savefig('foo.png')</pre>	Save transparent figures	<pre>&gt;&gt;&gt; plt.savefig('foo.png', transparent=True)</pre>
--------------	--	--------------------------	--

#### 6) Show Plot

<pre>&gt;&gt;&gt; plt.show()</pre>
------------------------------------

#### Close & Clear

Clear an axis	Clear the entire figure	Close a window
---------------	-------------------------	----------------

### DataCamp

Learn Python for Data Science interactively

## Seaborn (PDF)

## Python For Data Science Cheat Sheet

### Seaborn

Learn Data Science interactively at [www.DataCamp.com](http://www.DataCamp.com)



#### Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> g = sns.FacetGrid(tips,
                     col="survived",
                     row="sex",
                     margin_titles=True)
>>> g.map(plt.hist, "tip",
        "total_bill",
        data=tips,
        aspect=2)
>>> g = g.set_axis_labels("Tip","Total Bill (USD)"). \
        set(xlim=0,10), ylim(0,100))
>>> plt.title("Title")
>>> plt.show(g)
```

#### 1) Data

Also see [Lists, NumPy & Pandas](#)

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame((x:np.arange(1,10),
                        "y":np.random.normal(0,4,100)))
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

#### 2) Figure Aesthetics

Also see [Matplotlib](#)

```
>>> f, ax = plt.subplots(figsize=(5, 5)) # Create a figure and one subplot
Seaborn styles
>>> sns.set()
>>> sns.set_style("whitegrid") # (Reset the seaborn default)
>>> sns.set_style("ticks", # Set the matplotlib parameters
                {"xtick.major.size":8,
                 "ytick.major.size":8})
>>> sns.axes_style("whitegrid") # Set the matplotlib parameters
Return a dict of params or use with
with to temporarily set the style
```

#### 3) Plotting With Seaborn

##### Axis Grids

<pre>&gt;&gt;&gt; g = sns.FacetGrid(titanic,                      col="survived",                      row="sex") &gt;&gt;&gt; g.map(plt.hist, "age") &gt;&gt;&gt; g = g.map(plt.hist, "age",               "survived",               hue="sex",               data=titanic) &gt;&gt;&gt; sns.lmplot(x="sex",                y="total_bill",                data=titanic,                aspect=2)</pre>	Subplot grid for plotting conditional relationships Draw a categorical plot onto a FacetGrid Plot data and regression model fits across a FacetGrid
--	---

##### Categorical Plots

<pre>&gt;&gt;&gt; sns.stripplot(x="species",                   y="petal_length",                   data=iris) &gt;&gt;&gt; sns.swarmplot(x="species",                   y="petal_length",                   data=iris) Bar Chart &gt;&gt;&gt; sns.barplot(x="sex",                 y="survived",                 hue="class",                 data=titanic) Count Plot &gt;&gt;&gt; sns.countplot(x="deck",                   data=titanic,                   palette="Greens_d")</pre>	Scatterplot with one categorical variable Categorical scatterplot with non-overlapping points Show point estimates and confidence intervals with scatterplot glyphs Show count of observations
---	---

##### Point Plot

<pre>&gt;&gt;&gt; sns.pointplot(x="class",                   y="survived",                   hue="sex",                   data=iris,                   palette="Greens_d")</pre>	Show point estimates and confidence intervals as rectangular bars
--	---

##### Boxplot

<pre>&gt;&gt;&gt; sns.boxplot(x="alive",                 y="age",                 hue="split_male",                 data=iris) &gt;&gt;&gt; sns.boxplot(data=iris,orient="h")</pre>	Boxplot with wide-form data
---	-----------------------------

##### Violinplot

<pre>&gt;&gt;&gt; sns.violinplot(x="sex",                     y="survived",                     hue="survived",                     data=titanic)</pre>	Violin plot
---	-------------

##### Regression Plots

<pre>&gt;&gt;&gt; sns.regplot(x="sepal_width",                  y="sepal_length",                  data=iris,                  ax=ax)</pre>	Plot data and a linear regression model fit
---	---

##### Distribution Plots

<pre>&gt;&gt;&gt; plot = sns.distplot(data.y,                          kde=False,                          color="b")</pre>	Plot univariate distribution
---	------------------------------

##### Matrix Plots

<pre>&gt;&gt;&gt; sns.heatmap(uniform_data,vmin=0,vmax=1)</pre>	Heatmap
---	---------

#### 4) Further Customizations

Also see [Matplotlib](#)

##### Axisgrid Objects

<pre>&gt;&gt;&gt; g = sns.FacetGrid(titanic,                      col="survived",                      row="sex",                      margin_titles=True) &gt;&gt;&gt; g.set(ylim=(0,100)) &gt;&gt;&gt; g.set(xticklabels(rotation=45)) &gt;&gt;&gt; i = i.plot_marginals(sns.JointGrid,                          "sex",                          data=titanic) &gt;&gt;&gt; i = i.plot_marginals(sns.JointGrid,                          "sex",                          data=titanic,                          kind="kde")</pre>	Remove left spine Set the labels of the y-axis Set the tick labels for x Set the axis labels Set the limit and ticks of the x- and y-axis
---	---

##### Plot

<pre>&gt;&gt;&gt; plt.title("A Title") &gt;&gt;&gt; plt.xlabel("Survived") &gt;&gt;&gt; plt.ylabel("Sex") &gt;&gt;&gt; plt.ylim(0,100) &gt;&gt;&gt; plt.setp(ax,xticks=[0,5]) &gt;&gt;&gt; plt.tight_layout()</pre>	Add plot title Adjust the label of the y-axis Adjust the label of the x-axis Add the ticks of the y-axis Adjust the ticks of the x-axis Adjust a plot property Adjust subplot params
---	--

#### 5) Show or Save Plot

Also see [Matplotlib](#)

##### Close & Clear

<pre>&gt;&gt;&gt; plt.show() &gt;&gt;&gt; plt.savefig("foo.png") &gt;&gt;&gt; plt.savefig("foo.png",                 transparent=True)</pre>	Show the plot Save the plot as a figure Save transparent figure
--	---

##### Close & Clear

<pre>&gt;&gt;&gt; plt.clf() &gt;&gt;&gt; plt.cla() &gt;&gt;&gt; plt.close()</pre>	Clear an axis Clear an entire figure Close a window
---	---

### DataCamp

Learn Python for Data Science interactively

## Bokeh (PDF)

## Python For Data Science Cheat Sheet

### Bokeh

Learn Bokeh [Interactively](#) at [www.DataCamp.com](#), taught by Bryan Van de Ven, core contributor



#### Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data: Python lists, NumPy arrays, Pandas DataFrames and other sequences of values.
2. Create a new plot.
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import show, output_file, save
>>> p = figure(title="simple line example", x_axis_label="x", y_axis_label="y")
>>> p.line(x, y, legend="Temp.", line_width=2)
>>> output_file("lines.html") # Step 4
>>> show(p) # Step 5
```

#### 1 Data

[Also see Lists, NumPy & Pandas](#)

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9, 4, 65, "USA"], [33.4, 4, 65, "USA"], [21.4, 4, 109, "Europe"]]), columns=['mpg', 'cyl', 'hp', 'origin'], index=['Toyota', 'Pint', 'Volvo'])
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

#### 2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=100,
>>>     x_range=[0, 8], y_range=[0, 8])
>>> p3 = figure()
```

### 3 Renderers & Visual Customizations

#### Glyphs

**Scatter Markers**  
  

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
>>>             fill_color="white")
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
>>>             color='blue', size=1)
```

**Line Glyphs**  
  

```
>>> p2.multi_line(pd.DataFrame([(1,2,3), (5,6,7)]),
>>>                 pd.DataFrame([(3,4,5), (3,2,1)]),
>>>                 color="blue")
```

#### Customized Glyphs

[Also see Data](#)  
**Selection and Non-Selection Glyphs**  
  

```
>>> p = Figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
>>>          selection_color='red',
>>>          nonselection_alpha=0.1)
```

**Hover Glyphs**  
  

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

**Colormapping**  
  

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
>>>     factors=['US', 'Asia', 'Europe'],
>>>     palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
>>>             color=dict(field='origin',
>>>                         transform=color_mapper),
>>>             legend='Origin')
```

#### Legend Location

**Inside Plot Area**  
  

```
>>> p1.legend.location = 'bottom_left'
```

**Outside Plot Area**  
  

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1]),
>>>                   fill_color="blue", line_color="black",
>>>                   legend=Legend(items=[("One", [p1, r1]), ("Two", [r2])]),
>>>                   location=(10, -30))
>>> p1.add_layout(legend, "right")
```

#### Legend Orientation

**Horizontal**  
  

```
>>> p1.legend.orientation = "horizontal"
```

**Vertical**  
  

```
>>> p1.legend.orientation = "vertical"
```

#### Legend Background & Border

**Border**  
  

```
>>> p1.legend.border_line_color = "navy"
>>> p1.legend.background_fill_color = "white"
```

#### Rows & Columns Layout

**Rows**  
  

```
>>> from bokeh.layouts import row
>>> layout = row(p1, p2, p3)
```

**Columns**  
  

```
>>> from bokeh.layouts import column
>>> layout = column(p1, p2, p3)
```

**Nesting Rows & Columns**  
  

```
>>> layout = row(column(p1, p2), p3)
```

#### Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1, p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1, p2], [p3]])
```

#### Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

#### Linked Plots

**Linked Axes**  
  

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

**Linked Brushing**  
  

```
>>> p4 = Figure(plot_width = 100,
>>>               tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = Figure(plot_width = 200,
>>>               tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

### 4 Output & Export

#### Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

#### HTML

**Standalone HTML**  
  

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
```

**Components**  
  

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

#### PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

#### SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

### 5 Show or Save Your Plots

**Show**  
  

```
>>> show(p1)
```

**Save**  
  

```
>>> save(p1)
```

**DataCamp**  
[Learn Python for Data Science Interactively!](#)



# DATA VISUALISATION IN PYTHON

## CHEATSHEET

### Why Is Data Visualisation an Important Concept ?

- Because it help us understand distribution, trend, relationship, comparison and composition of data values
- It helps decision makers to quickly examine large piles of data and discover the hidden pattern/insights

**" BEAUTY OF AN ART LIES IN THE MESSAGE IT CONVEYS "**

### WHAT IS REQUIRED TO MAKE VISUALISATION IN PYTHON ?

#### MATPLOTLIB

Python based plotting library offers matplotlib with a complete 2D support along with limited 3D graphic support. It is useful in producing publication quality figures in interactive environment across platforms.

#### SEABORN

Being based on matplotlib, seaborn offers various features such as built in themes, color palettes, functions and tools to visualize univariate, bivariate, linear regression, matrices of data, statistical time series etc which lets us to build complex visualizations.

**Sample Data Set Used For The VISUALISATION Show Below**

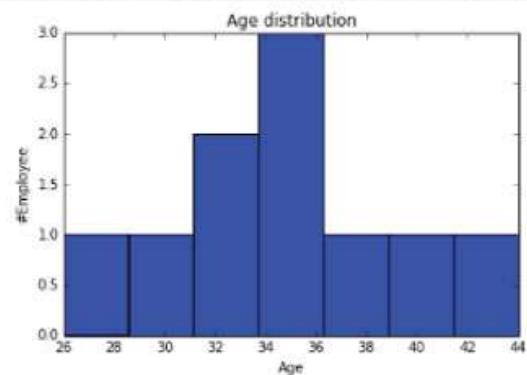
EMPID	Gender	Age	Sales	BMI	Income
E001	M	34	123	Normal	350
E002	F	40	114	Overweight	450
E003	F	37	135	Obesity	169
E004	M	30	139	Underweight	189
E005	F	44	117	Underweight	183
E006	M	36	121	Normal	80
E007	M	32	133	Obesity	166
E008	F	26	140	Normal	120
E009	M	32	133	Normal	75
E010	M	36	133	Underweight	40

## Import Data Set:

```
import matplotlib.pyplot as plt
import pandas as pd
df=pd.read_excel("E:/First.xlsx", "Sheet1")
```

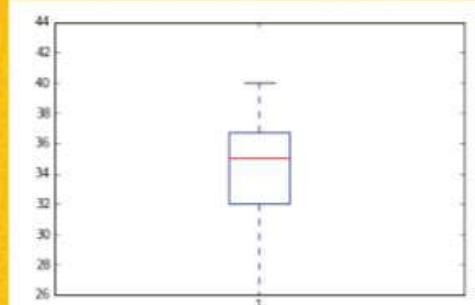
## Histogram

```
fig=plt.figure()
ax = fig.add_subplot(1,1)
ax.hist(df['Age'],bins = 7) # Here you can
play with number of bins Labels and Tit
plt.title('Age distribution')
plt.xlabel('Age')
plt.ylabel('#Employee')
plt.show()
```



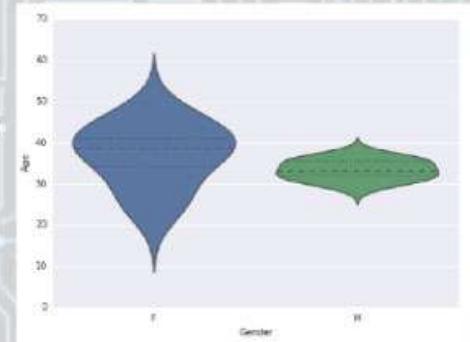
## Box Plot

```
import matplotlib.pyplot as plt
import pandas as pd
fig=plt.figure()
ax = fig.add_subplot(1,1)
x.boxplot(df['Age'])
plt.show()
```

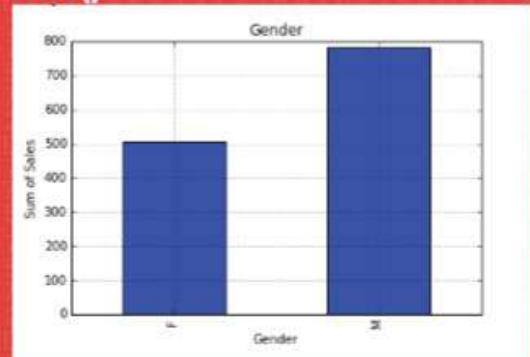


**VIOLIN PLOT**

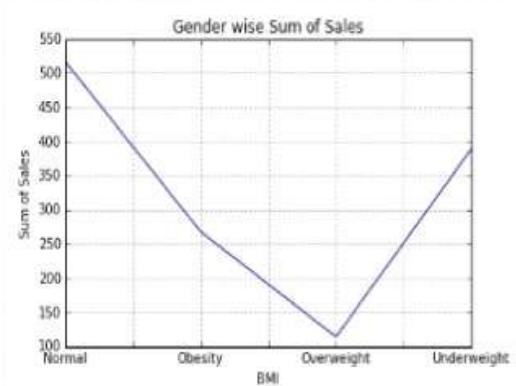
```
import seaborn as sns
sns.violinplot(df['Age'], df['Gender'])
#Variable Plot
sns.despine()
```

**Bar Chart**

```
var = df.groupby('Gender').Sales.sum()
#grouped sum of sales at
Gender level
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('Gender')
ax1.set_ylabel('Sum of Sales')
ax1.set_title("Gender wise Sum of Sales")
var.plot(kind='bar')
```

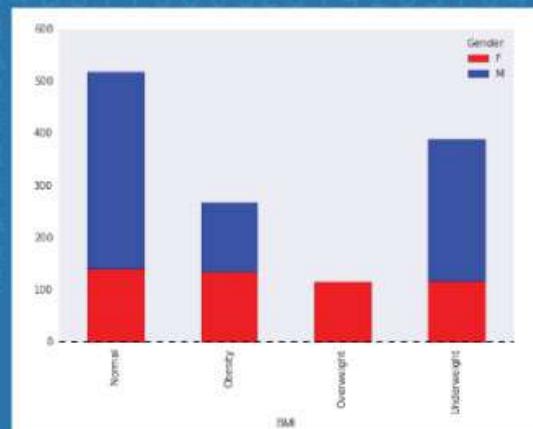
**Line Chart**

```
var = df.groupby('BMI').Sales.sum()
fig = plt.figure()
ax1 = fig.add_subplot(1,1,1)
ax1.set_xlabel('BMI')
ax1.set_ylabel('Sum of Sales')
ax1.set_title("BMI wise Sum
of Sales")
var.plot(kind='line')
```

**Stacked Column Chart**

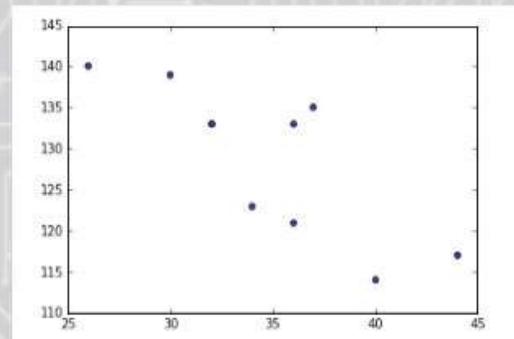
```
var = df.groupby(['BMI','Gender']).Sales.sum()
```

```
var.unstack().plot(kind='bar',stacked=True, color=['red','blue'], grid=False)
```



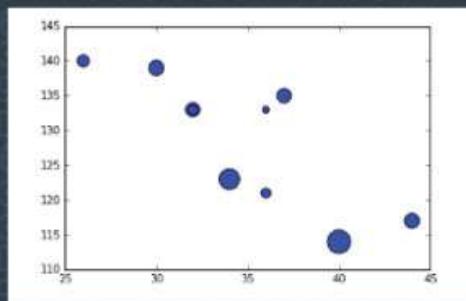
## Scatter Plot

```
fig = plt.figure()
ax = fig.add_subplot(1,1)
ax.scatter(df['Age'],df['Sales'])
plt.show()
```



## Bubble Plot

```
fig = plt.figure()
ax = fig.add_subplot(1,1)
ax.scatter(df['Age'],df['Sales'], s=df['Income'])
plt.show()
```

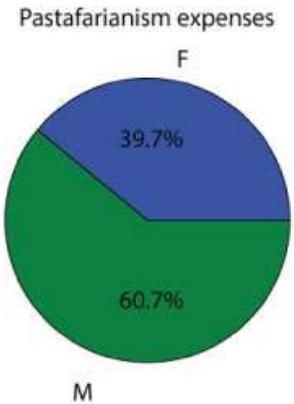


## Pie Chart

```

var=df.groupby(['Gender']).sum().stack()
temp=var.unstack()
type(temp)
x_list = temp['Sales']
label_list = temp.index
pyplot.axis("equal") #The pie chart
is oval by default. To make it a
circle use pyplot.axis("equal")
plt.pie(x_list,labels=label_list,autopct="%1.1f%%")
plt.title("Pastafarianism expenses")
plt.show()

```

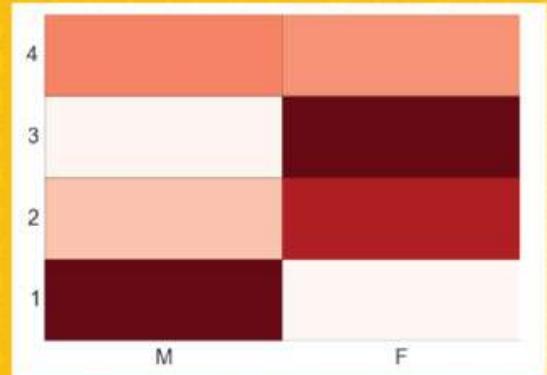


## Heat Map

```

import numpy as np
data = np.random.rand(4,2)
rows = list('1234') #rows
categories columns =
list('MF') #column categories
fig,ax=plt.subplots()
ax.pcolor(data,cmap=plt.cm.Reds,edgecolors='k')
ax.set_xticks(np.arange(0,2)+0.5)
ax.set_yticks(np.arange(0,4)+0.5)
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()
ax.set_xticklabels(categories,minor=False,fontsize=20)
ax.set_yticklabels(rows,minor=False,fontsize=20)
plt.show()

```



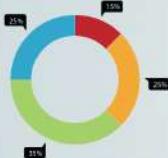
To view the complete guide on data visualisation in python  
**visit here : <http://bit.ly/1FjTkRF>**

## R

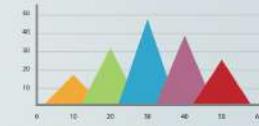
---

- [Ggplot2 \(PDF\)](#)
- [Leaflet \(PDF\)](#)
- [Cartography \(PDF\)](#)
- [Comprehensive Guide to Data Visualization in R](#)

# Data Visualization in R CHEATSHEET



in R

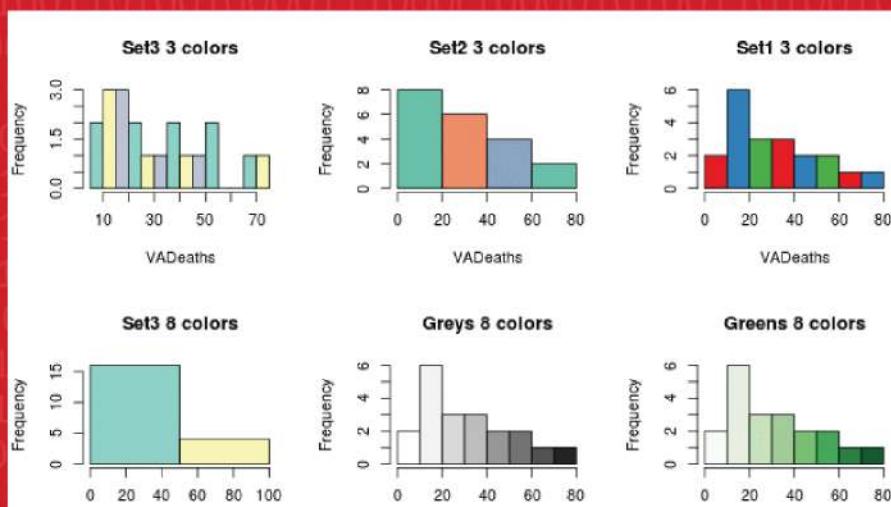


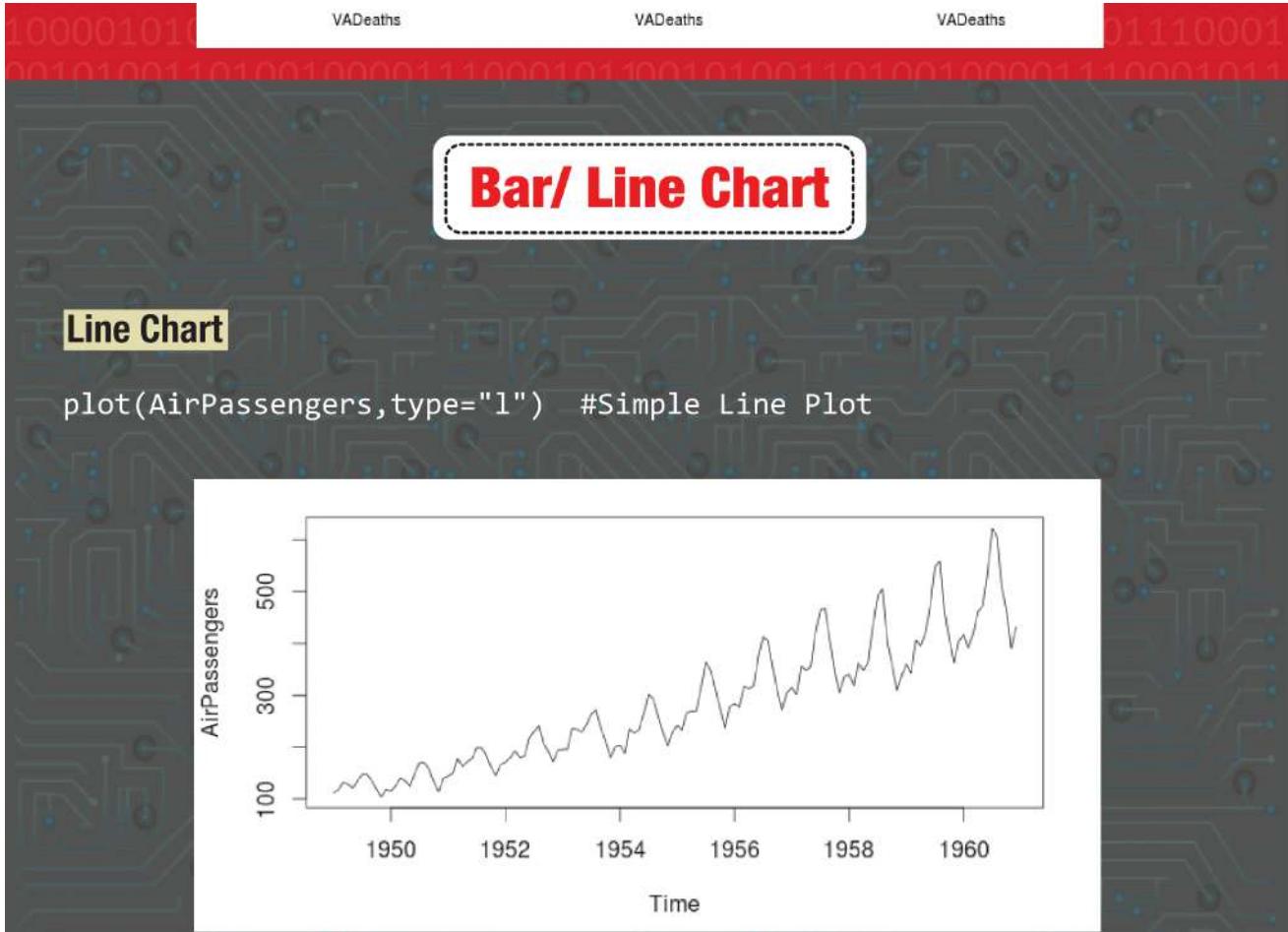
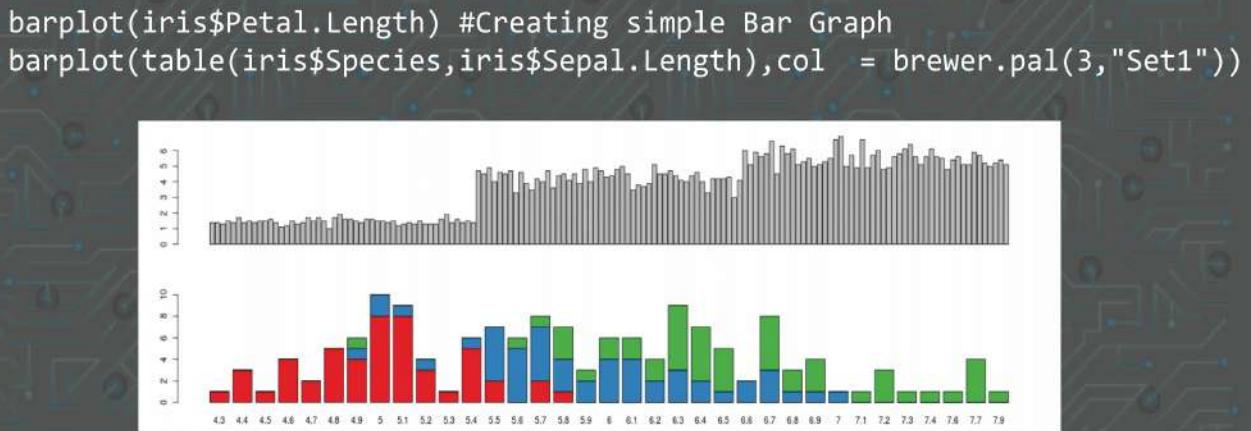
R Programming offers a set of inbuilt functions & libraries (such as `ggplot2`, `leaflet`, `lattice`) to create visualizations & present data stories. Below is a guide to create basic and advanced visualizations in R

## BASIC VISUALIZATIONS

### Histogram

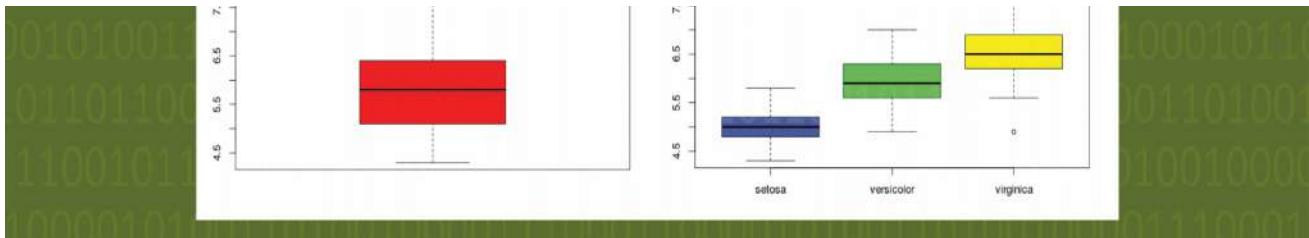
```
library(RColorBrewer)
data(VADeaths)
par(mfrow=c(2,3))
hist(VADeaths, breaks=10, col=brewer.pal(3,"Set3"), main="Set3 3 colors")
```



**Bar Chart****Box Plot ( including group-by option )**

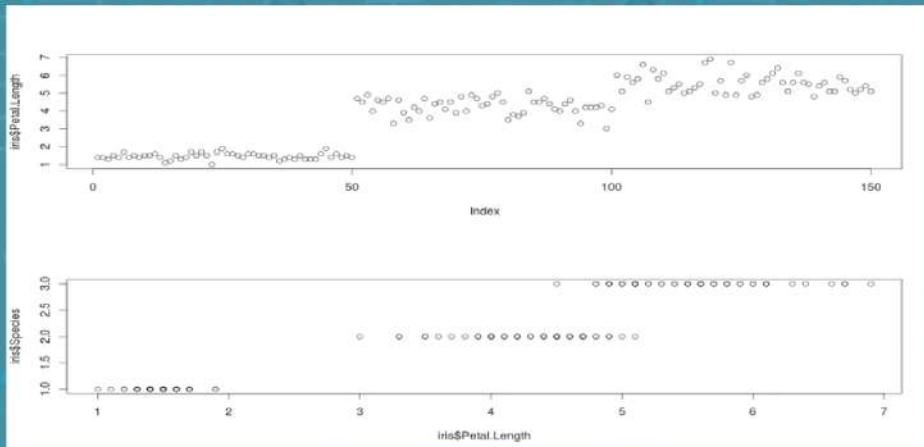
```
data(iris)
par(mfrow=c(2,2))
boxplot(iris$Sepal.Length,col="red")

boxplot(iris$Sepal.Length~iris$Species,col=topo.colors(3))
```

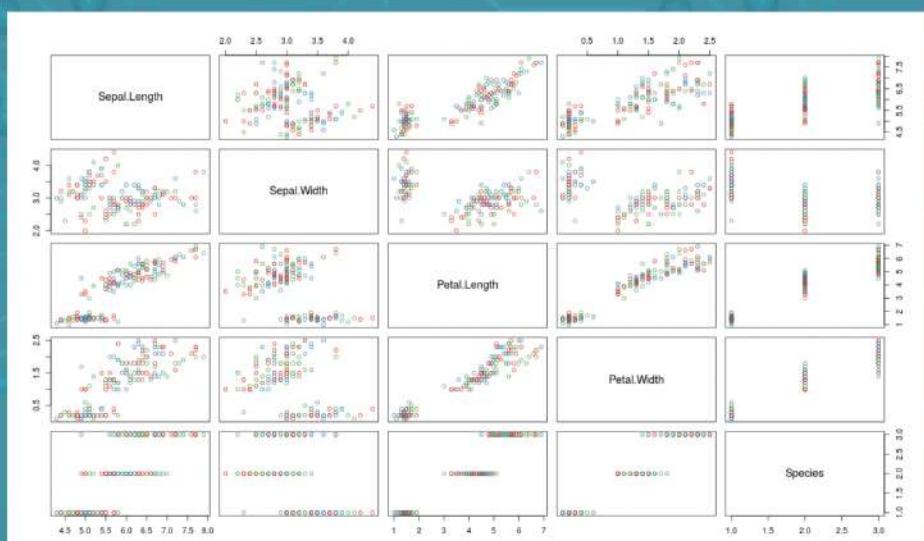


## Scatter Plot (including 3D and other features)

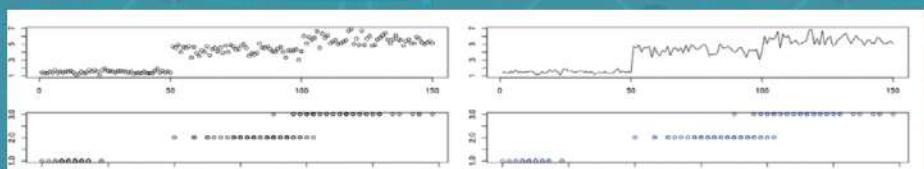
```
#Simple Scatter Plot
plot(x=iris$Petal.Length)
#Multivariate Scatter Plot
plot(x=iris$Petal.Length,y=iris$Species)
```

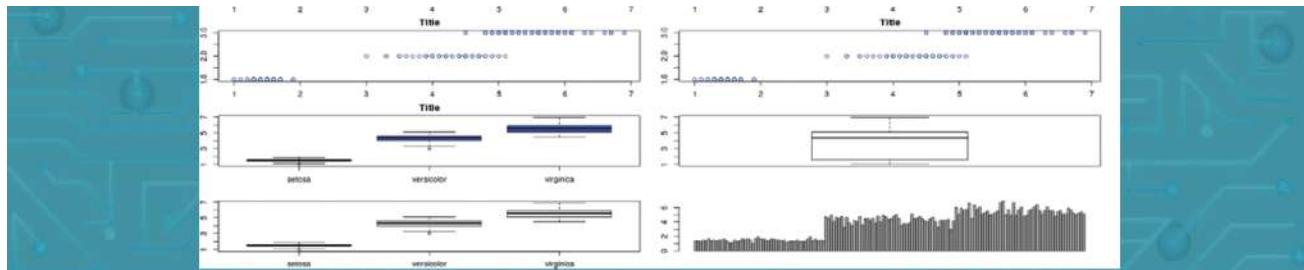


```
plot(iris,col=brewer.pal(3,"Set1"))
```



```
pie(table(iris$Species))
```

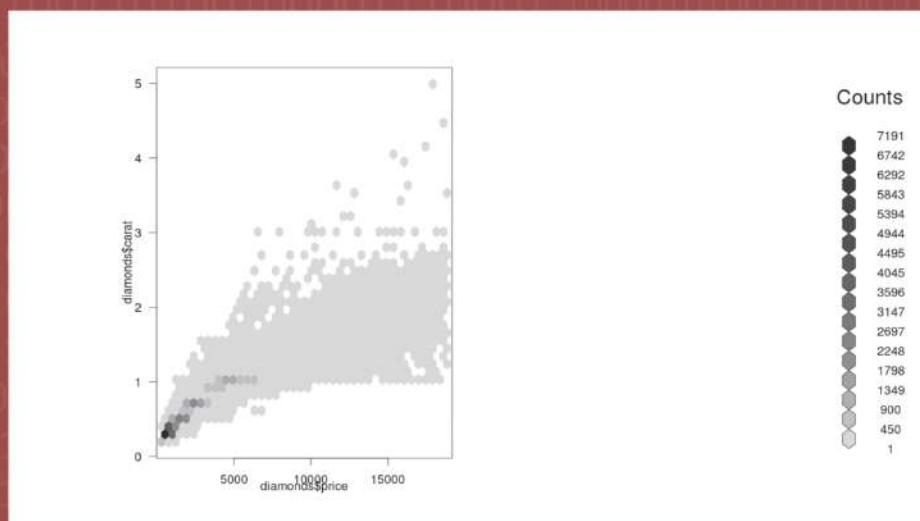




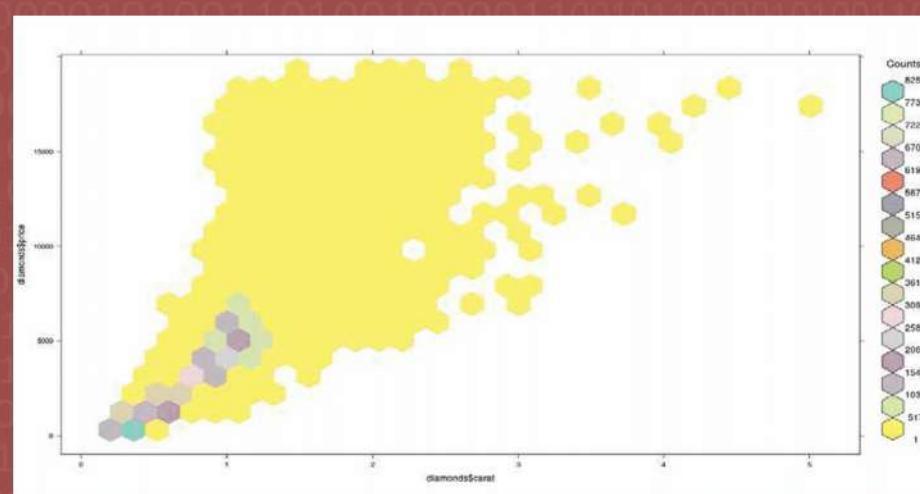
# Advanced Visualizations

## Hexbin Binning

```
>library(hexbin)
>a=hexbin(diamonds$price,diamonds$carat,xbins=40)
>library(RColorBrewer)
>plot(a)
```

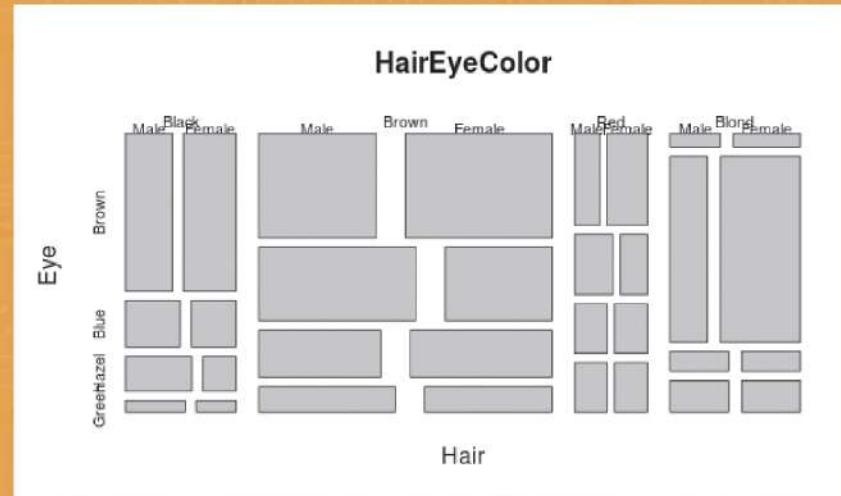


```
>library(RColorBrewer)
>rf <- colorRampPalette(rev(brewer.pal(40, 'Set3')))
>hexbinplot(diamonds$price~diamonds$carat, data=diamonds, colramp=rf)
```



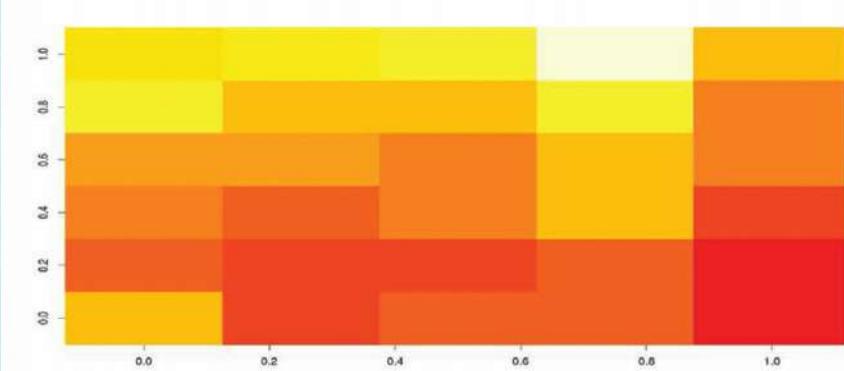
## Mosaic Plot

```
> data(HairEyeColor)
> mosaicplot(HairEyeColor)
```



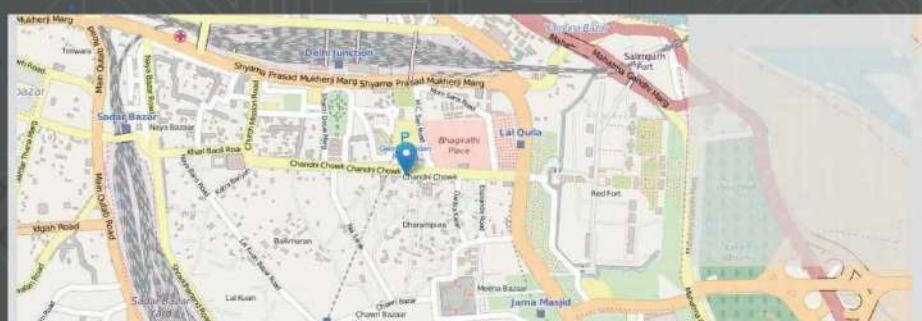
## Heat Map

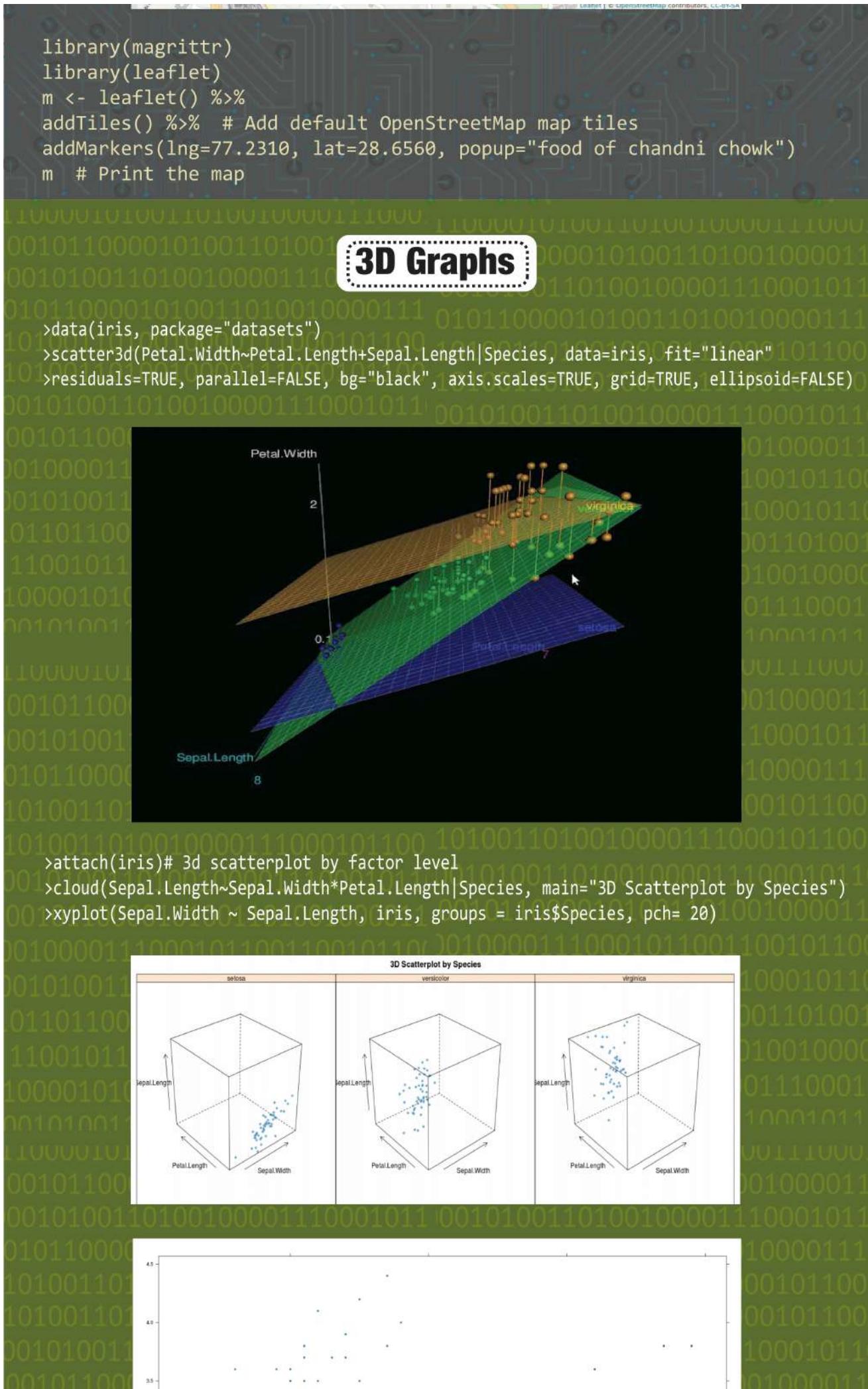
```
> heatmap(as.matrix(mtcars))
> image(as.matrix(b[2:7]))
```

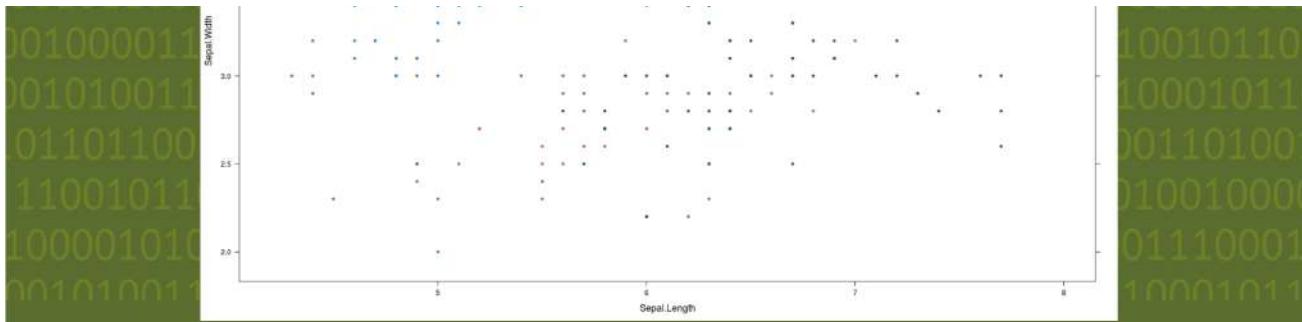


## Map Visualization

```
devtools::install_github("rstudio/leaflet")
```

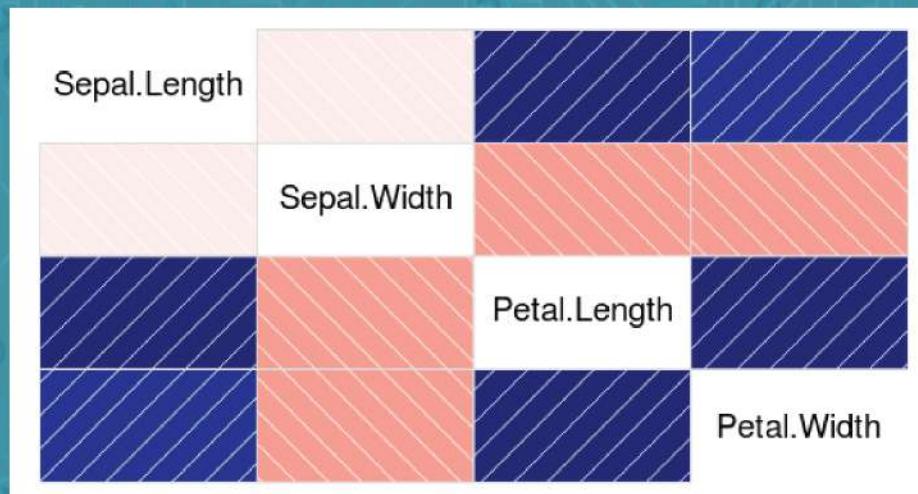






## Correlogram (GUIs)

```
> cor(iris[1:4])
      Sepal.Length   Sepal.Width   Petal.Length   Petal.Width
Sepal.Length  1.0000000 -0.1175698  0.8717538  0.8179411
Sepal.Width   -0.1175698  1.0000000 -0.4284401 -0.3661259
Petal.Length  0.8717538 -0.4284401  1.0000000  0.9628654
Petal.Width   0.8179411 -0.3661259  0.9628654  1.0000000
> corrgram(iris)
```



To view the complete guide on Data Visualization in R  
visit here : <http://bit.ly/1DhD1Sk>

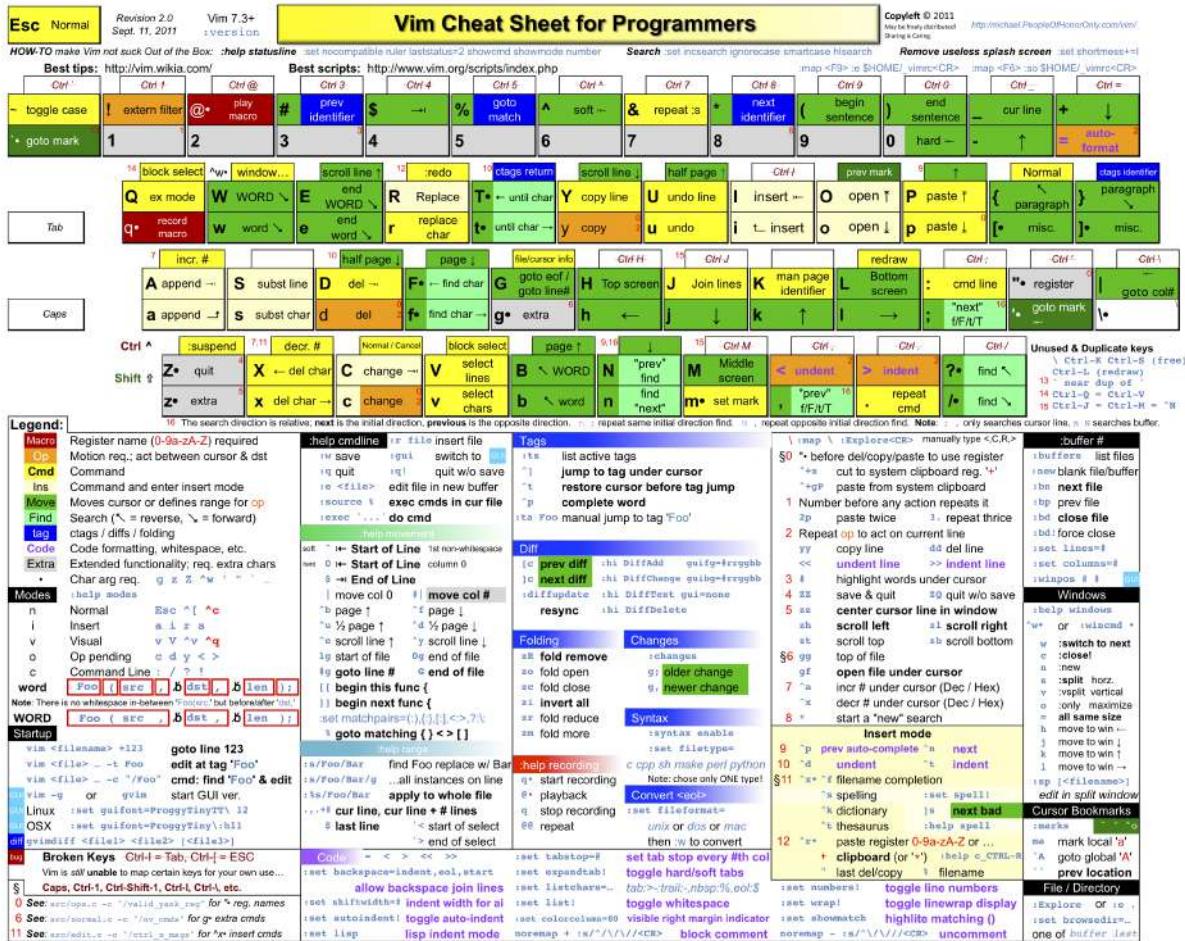


- Simple Features sf (PDF)
- survminer (PDF)

## Data Science in General and Others

- Shiny (PDF)

- Vim (PDF)



- Emacs (PDF)

# HOW TO LEARN EMACS

A beginner's guide to Emacs 24 or later  
Sacha Chua (@sachac) · sachachua.com/begin-emacs

STEP 1 If you're a developer or sysadmin  
**Learn Vim** → the other text editor  
It's okay. Learn the basics so that you can easily work on other people's computers. If you know your way around Vim, people won't give you as much grief over Emacs.

Bonus mini-cheatsheet!  
Here's what you need to know in Vim:  
- insert mode → Esc command mode  
- long-time VI user trying out Emacs  
- evil mode Emacs ↴

You can actually edit remote files in Emacs, but it's an intermediate topic. (C-x (C-x))? See TRAMP!

Okay. Once you know the basics of Vim, you can get on with learning Emacs. ↴

Learn how to learn  
There are some old books on Emacs, but the version differences can be rather confusing.  
Start with the built-in tutorial instead. Help → Emacs Tutorial ↴ repeat as many times as you need to.

Can't use the menu? Press **Control-h** ↴ to start.  
NOTE: The Emacs tutorial has lots of weird terms: "Meta key", "frame", "buffer". This is because Emacs started a long time ago. Don't worry, you'll get the hang of it with practice.

Some things that might help:  
Ex: C-s This is how keyboard shortcuts are written. Press Control & s at the same time, then let go of x and press Control & s again. Tip: Since you're using Control for both keys, you can hold Control down instead of letting go between x and s.

M-x lets you call commands by name, which is great if you can't remember the keyboard shortcut. Ex: M-x help-with-tutorial RET Then press Enter/Return.

Other resources:  
emacswiki.org Lots of resources planet.emacswiki.org Emacs-related blogs (IRC, freenode.net) Emacs Great for help and hanging out

STEP 2 Why Emacs?  
- Amazingly customizable  
- Endless room for growth

STEP 3 Extend & customize  
M-x load-theme RET Try out color themes M-x customize-group RET Set common options M-x customize-face RET Change background, foreground, etc. M-x list-packages RET Install lots of modules and then... editing your ~/.emacs.d/init.el file!  
C-x C-1 initializes your Emacs, adds new functionality, and so on. Just use C-x C-f to create it! See emacswiki.org for lots of examples Use M-x eval-buffer or restart Emacs to see the changes! Brkfst your .emacs.d/q skips your customizations

STEP 4 Learn Emacs basics  
C-x C-f open (find-file)  
C-x C-s save  
C-x C-c quit  
NOTE: You don't need to quit Emacs after each file. Just leave it running and use C-x C-f to open the next.

How to select text  
Go to the start of your selection and press C-SPC (Control+space). Go to the end of your selection and run your command. C-w cut (kill) C-y paste (yank) M-u paste older things C- undo

\* Learn how to use Keyboard macros. They're awesome. C-x ( start macro C-x ) end macro C-x e execute macro C-... again

STEP 5 Explore!  
Org-mode org organize your life in plain text  
Narrowing/ Widening  
TRAMP remote access Eshell / Term command-line in Emacs  
Calc powerful calculator and converter  
Writing & debugging Emacs Lisp (it sounds scary, but it's powerful!)  
There's so much more! Ask away, and discover more by exploring!

Questions? I'd love to hear 2013(2)

Sacha Chua

## Git - Atlassian's Cheatsheet (PDF)

### Git Cheat Sheet

Git Basics	
<code>git init &lt;directory&gt;</code>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
<code>git clone &lt;repo&gt;</code>	Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
<code>git config user.name &lt;name&gt;</code>	Define author name to be used for all commits in current repo. Devs commonly use --global flag to set config options for current user.
<code>git add &lt;directory&gt;</code>	Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
<code>git commit -m "&lt;message&gt;"</code>	Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
<code>git status</code>	List which files are staged, unstaged, and untracked.
<code>git log</code>	Display the entire commit history using the default format. For customization see additional options.
<code>git diff</code>	Show unstaged changes between your index and working directory.

Rewriting Git History	
<code>git commit --amend</code>	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
<code>git rebase &lt;base&gt;</code>	Rebase the current branch onto <base>. <base> can be a commit ID, a branch name, a tag, or a relative reference to HEAD.
<code>git reflog</code>	Show a log of changes to the local repository's HEAD. Add --relative-date flag to show date info or --all to show all refs.

Git Branches	
<code>git branch</code>	List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
<code>git checkout -b &lt;branch&gt;</code>	Create and checkout a new branch named <branch>. Drop the -b flag to checkout an existing branch.
<code>git merge &lt;branch&gt;</code>	Merge <branch> into the current branch.

Remote Repositories	
<code>git remote add &lt;name&gt; &lt;url&gt;</code>	Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands.
<code>git fetch &lt;remote&gt; &lt;branch&gt;</code>	Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs.
<code>git pull &lt;remote&gt;</code>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
<code>git push &lt;remote&gt; &lt;branch&gt;</code>	Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

## Additional Options +

git config		git diff	
<code>git config --global user.name &lt;name&gt;</code>	Define the author name to be used for all commits by the current user.	<code>git diff HEAD</code>	Show difference between working directory and last commit.
<code>git config --global user.email &lt;email&gt;</code>	Define the author email to be used for all commits by the current user.	<code>git diff --cached</code>	Show difference between staged changes and last commit
<code>git config --global alias. &lt;alias-name&gt; &lt;git-command&gt;</code>	Create shortcut for a Git command. E.g. <code>alias.glog log --graph --oneline</code> will set <code>git glog</code> equivalent to <code>git log --graph --oneline</code> .	git reset	
<code>git config --system core.editor &lt;editor&gt;</code>	Set text editor used by commands for all users on the machine. <code>&lt;editor&gt;</code> arg should be the command that launches the desired editor (e.g., vi).	<code>git reset</code>	Reset staging area to match most recent commit, but leave the working directory unchanged.
<code>git config --global --edit</code>	Open the global configuration file in a text editor for manual editing.	<code>git reset --hard</code>	Reset staging area and working directory to match most recent commit and overwrites all changes in the working directory.
git log		<code>git reset &lt;commit&gt;</code>	Move the current branch tip backward to <code>&lt;commit&gt;</code> , reset the staging area to match, but leave the working directory alone.
<code>git log --limit=</code>	Limit number of commits by <code>&lt;limit&gt;</code> . E.g. <code>git log -5</code> will limit to 5 commits.	<code>git reset --hard &lt;commit&gt;</code>	Same as previous, but resets both the staging area & working directory to match. Deletes uncommitted changes, and all commits after <code>&lt;commit&gt;</code> .
<code>git log --oneline</code>	Condense each commit to a single line.	git rebase	
<code>git log -p</code>	Display the full diff of each commit.	<code>git rebase -i &lt;base&gt;</code>	Interactively rebase current branch onto <code>&lt;base&gt;</code> . Launches editor to enter commands for how each commit will be transferred to the new base.
<code>git log --stat</code>	Include which files were altered and the relative number of lines that were added or deleted from each of them.	git pull	
<code>git log --author= "&lt;pattern&gt;"</code>	Search for commits by a particular author.	<code>git pull --rebase &lt;remote&gt;</code>	Fetch the remote's copy of current branch and rebases it into the local copy. Uses <code>git rebase</code> instead of <code>merge</code> to integrate the branches.
<code>git log --grep="&lt;pattern&gt;"</code>	Search for commits with a commit message that matches <code>&lt;pattern&gt;</code> .	git push	
<code>git log &lt;since&gt;..&lt;until&gt;</code>	Show commits that occur between <code>&lt;since&gt;</code> and <code>&lt;until&gt;</code> . Args can be a commit ID, branch name, HEAD, or any other kind of revision reference.	<code>git push &lt;remote&gt; --force</code>	Forces the <code>git push</code> even if it results in a non-fast-forward merge. Do not use the <code>--force</code> flag unless you're absolutely sure you know what you're doing.
<code>git log -- &lt;file&gt;</code>	Only display commits that have the specified file.	<code>git push &lt;remote&gt; --all</code>	Push all of your local branches to the specified remote.
<code>git log --graph --decorate</code>	<code>--graph</code> flag draws a text based graph of commits on left side of commit msgs. <code>--decorate</code> adds names of branches or tags of commits shown.	<code>git push &lt;remote&gt; --tags</code>	Tags aren't automatically pushed when you push a branch or use the <code>--all</code> flag. The <code>--tags</code> flag sends all of your local tags to the remote repo.

- DVC (PDF)



## Basics

**Initializing**  
Initialize a DVC environment  
`$ dvc init`

**Remote**  
Set up a remote to keep and share data files  
`$ dvc remote add -d myremote /path`  
\*Possible remotes include local, s3, gs, azure, ssh, hdfs and http.  
Show all available remotes  
`$ dvc remote list`

Modify remote settings  
`$ dvc remote modify myremote`  
\*Use if remote requires extra configuration

**Adding Files**  
Add files under DVC control  
`$ dvc add filename`  
\*Use --no-commit to stop adding the file to the cache.

**Share Data**  
Push all data files to the remote storage  
`$ dvc push`

Push outputs of a specific .dvc file  
`$ dvc push filename.dvc`

### Retrieve Data

Download files from the remote storage  
`$ dvc pull`

Download files from a specific .dvc file  
`$ dvc pull filename.dvc`

Checkout files from cache into working space  
`$ dvc checkout`

### The Pipeline

Add transformations and generate a stage file from a given command  
`$ dvc run -d dependencyfile \ -o outputfile python command.py`  
\*Use --file to specify the name of the generated .dvc file.  
\*Use --metrics to output a file containing the metric.

**Metrics**  
Collect and display project metrics  
`$ dvc metrics show`  
\*Use --all to show the metrics in all branches.

**Visualizing**  
Show stages in a pipeline  
`$ dvc pipeline show --ascii file.dvc`  
\*Add --commands or --cuts to show more detail.  
Show connected pipelines of DVC stages  
`$ dvc pipeline list`

**Reproducing**  
Reproduce outputs defined in .dvc file  
`$ dvc repro filename.dvc`  
\*Name a .dvc file "Dvcfile" to be used by `dvc repro` by default

<https://github.com/iterative/dvc> 

<https://dvc.org/chat> 

## Other Commands

Set/unset cache directory location  
`$ dvc cache dir /path`

Commit outputs to cache  
`$ dvc commit`  
\*Use if you specified --no-commit in `dvc add/run/repo`

Config repository or global options  
`$ dvc config`  
\*Config the default remote using `core.remote myremote`  
\*Config `core (loglevel, remote), cache and state` settings

Fetch files from the remote to the local cache  
`$ dvc fetch file.dvc`

Remove unused objects from cache  
`$ dvc gc`

Import file from URL to local directory  
`$ dvc import url /path`  
\*Supported schemes include local, s3, gs, azure, ssh, hdfs and http.

Remove data files tracked by dvc  
`$ dvc remove filename.dvc`

Show changed stages in the pipeline  
`$ dvc status`

Made by Carl Handlin based on the documentation for DVC at <https://dvc.org/doc>.

- [BASH \(PDF\)](#)

# Linux Bash Shell Cheat Sheet

## Basic Commands

### Basic Terminal Shortcuts

```
CTRL L = Clear the terminal
CTRL D = Logout
SHIFT Page Up/Down = Go up/down the terminal
CTRL A = Cursor to start of line
CTRL E = Cursor to the end of line
CTRL U = Delete left of the cursor
CTRL K = Delete right of the cursor
CTRL W = Delete word on the left
CTRL Y = Paste (after CTRL U,K or W)
TAB = auto completion of file or command
CTRL R = reverse search history
!! = repeat last command
CTRL Z = stops the current command (resume with fg in foreground or bg in background)
```

### Basic Terminal Navigation

```
ls -a = list all files and folders
ls <folderName> = list files in folder
ls -lh = Detailed list, Human readable
ls -l *.jpg = list jpeg files only
ls -lh <fileName> = Result for file only

cd <folderName> = change directory
    if folder name has spaces use ""
cd / = go to root
cd .. = go up one folder, tip: ../../...

du -h: Disk usage of folders, human readable
du -ah: " " " files & folders, Human readable
du -sh: only show disc usage of folders

pwd = print working directory
man <command> = shows manual (RTFM)
```

### Basic file manipulation

```
cat <fileName> = show content of file
                (less, more)
head = from the top
    -n <#oflines> <fileName>
tail = from the bottom
    -n <#oflines> <fileName>

mkdir = create new folder
mkdir myStuff ..
mkdir myStuff/pictures/ ..

cp image.jpg newimage.jpg = copy and rename a file
cp image.jpg <folderName>/ = copy to folder
cp image.jpg folder/sameImageNewName.jpg
cp -R stuff otherStuff = copy and rename a folder
cp *.txt stuff/ = copy all of *<file type> to folder

mv file.txt Documents/ = move file to a folder
mv <folderName> <folderName2> = move folder in folder
mv filename.txt filename2.txt = rename file
mv <fileName> stuff/newfileName
mv <folderName> .. = move folder up in hierarchy

rm <fileName> .. = delete file (s)
rm -i <fileName> .. = ask for confirmation each file
rm -f <fileName> = force deletion of a file
rm -r <foldername>/ = delete folder

touch <fileName> = create or update a file

ln file1 file2 = physical link
ln -s file1 file2 = symbolic link
```

# Linux Bash Shell Cheat Sheet

## Basic Commands

### Researching Files

The slow method (sometimes very slow):

```
locate <text> = search the content of all the files
locate <fileName> = search for a file
sudo updatedb = update database of files
```

```
find = the best file search tool(fast)
find -name "<fileName>" = search for files who start with the word text
find -name "*text*" = " " " " " " " "
```

### Advanced Search:

```
Search from file Size (in ~)
    find ~ -size +10M = search files bigger than.. (M,K,G)

Search from last access
    find -name "<filetype>" -atime -5
        ('-' = less than, '+' = more than and nothing = exactly)

Search only files or directory's
    find -type d --> ex: find /var/log -name "syslog" -type d
    find -type f = files

More info: man find, man locate
```

### Extract, sort and filter data

```
grep <someText> <fileName> = search for text in file
    -i = Doesn't consider uppercase words
    -I = exclude binary files
grep -r <text> <folderName>/ = search for file names
    with occurrence of the text
```

### With regular expressions:

```
grep -E '^<text>' <fileName> = search start of lines
    with the word text
grep -E <0-4> <fileName> = shows lines containing numbers 0-4
grep -E <a-zA-Z> <fileName> = retrieve all lines
    with alphabetical letters
```

```
sort = sort the content of files
sort <fileName> = sort alphabetically
sort -o <file> <outputFile> = write result to a file
sort -r <fileName> = sort in reverse
sort -R <fileName> = sort randomly
sort -n <fileName> = sort numbers
```

```
wc = word count
wc <fileName> = nbr of line, nbr of words, byte size
    -l (lines), -w (words), -c (byte size), -m
    (number of characters)
```

```
cut = cut a part of a file
    -c --> ex: cut -c 2-5 names.txt
            (cut the characters 2 to 5 of each line)
    -d (delimiter)           (-d & -f good for .csv files)
    -f (# of field to cut)
```

more info: man cut, man sort, man grep

# Linux Bash Shell Cheat Sheet

## Basic Commands

### Time settings

```
date = view & modify time (on your computer)

View:
    date "+%H" --> If it's 9 am, then it will show 09
    date "+%H:%M:%S" = (hours, minutes, seconds)
    %Y = years

Modify:
    MMDDhhmmYYYY
    Month | Day | Hours | Minutes | Year
    sudo date 031423421997 = March 14th 1997, 23:42

Execute programs at another time
use 'at' to execute programs in the future

Step 1, write in the terminal: at <timeOfExecution> ENTER
ex --> at 16:45 or at 13:43 7/23/11 (to be more precise)
or after a certain delay:
    at now +5 minutes (hours, days, weeks, months, years)

Step 2: <ENTER COMMAND> ENTER
    repeat step 2 as many times you need
Step 3: CTRL D to close input

atq = show a list of jobs waiting to be executed

atrm = delete a job n°<x>
ex (delete job #42) --> atrm 42

sleep = pause between commands
    with ';' you can chain commands, ex: touch file; rm file
    you can make a pause between commands (minutes, hours, days)
    ex --> touch file; sleep 10; rm file <-- 10 seconds
```

### (continued)

```
crontab = execute a command regularly
    -e = modify the crontab
    -l = view current crontab
    -r = delete your crontab
In crontab the syntax is
<Minutes> <Hours> <Day of month> <Day of week (0-6,
0 = Sunday)> <COMMAND>

ex, create the file movies.txt every day at 15:47:
47 15 * * * touch /home/bob/movies.txt
* * * * --> every minute
at 5:30 in the morning, from the 1st to 15th each month:
30 5 1-15 * *
at midnight on Mondays, Wednesdays and Thursdays:
0 0 * * 1,3,4
every two hours:
0 */2 * * *
every 10 minutes Monday to Friday:
*/10 * * * 1-5
```

### Execute programs in the background

```
Add a '&' at the end of a command
    ex --> cp bigMovieFile.mp4 &

nohup: ignores the HUP signal when closing the console
    (process will still run if the terminal is closed)
    ex --> nohup cp bigMovieFile.mp4

jobs = know what is running in the background

fg = put a background process to foreground
    ex: fg (process 1), f%2 (process 2), f%3, ...
```

# Linux Bash Shell Cheat Sheet

## Basic Commands

### Process Management

```
w = who is logged on and what they are doing

tload = graphic representation of system load average
    (quit with CTRL C)

ps = Static process list
    -ef --> ex: ps -ef | less
    -ejH --> show process hierarchy
    -u --> process's from current user

top = Dynamic process list
While in top:
    • q to close top
    • h to show the help
    • k to kill a process

CTRL C to stop a current terminal process

kill = kill a process
    You need the PID # of the process
        ps -u <AccountName> | grep <Application>
    Then
        kill <PID> . . .
kill -9 <PID> = violent kill

killall = kill multiple process's
    ex --> killall locate

extras:
    sudo halt <-- to close computer
    sudo reboot <-- to reboot
```

### Create and modify user accounts

```
sudo adduser bob = root creates new user
sudo passwd <AccountName> = change a user's password
sudo deluser <AccountName> = delete an account

addgroup friends = create a new user group
delgroup friends = delete a user group

usermod -g friends <Account> = add user to a group
usermod -g bob boby = change account name
usermod -aG friends bob = add groups to a user without loosing the ones he's already in
```

### File Permissions

```
chown = change the owner of a file
    ex --> chown bob hello.txt
chown user:bob report.txt = changes the user owning report.txt to 'user' and the group owning it to 'bob'
-R = recursively affect all the sub folders
    ex --> chown -R bob:bob /home/Daniel

chmod = modify user access/permission - simple way
    u = user
    g = group
    o = other

    d = directory (if element is a directory)
    l = link (if element is a file link)
    r = read (read permissions)
    w = write (write permissions)
    x = execute (only useful for scripts and programs)
```

# Linux Bash Shell Cheat Sheet

## Basic Commands

### File Permissions (continued)

```
'+' means add a right
'-' means delete a right
'=' means affect a right

ex --> chmod g+w someFile.txt
      (add to current group the right to modify someFile.txt)

more info: man chmod
```

### Flow redirection

Redirect results of commands:

'>' at the end of a command to redirect the result to a file  
 ex --> ps -ejH > process.txt  
 '>>' to redirect the result to the end of a file

### Redirect errors:

'2>' at the end of the command to redirect the result to a file  
 ex --> cut -d , -f 1 file.csv > file 2> errors.log  
 '2>&1' to redirect the errors the same way as the standard output

Read progressively from the keyboard

```
<Command> << <wordToTerminateInput>
      ex --> sort << END --> This can be anything you want
          > Hello
          > Alex
          > Cinema
          > Game
          > Code
          > Ubuntu
          > END
```

### Flow Redirection (continued)

```
terminal output:
Alex
Cinema
Code
Game
Ubuntu
```

Another example --> wc -m << END

### Chain commands

'|' at the end of a command to enter another one  
 ex --> du | sort -nr | less

### Archive and compress data

Archive and compress data the long way:

Step 1, put all the files you want to compress in the same folder: ex --> mv \*.txt folder/

Step 2, Create the tar file:  
 tar -cvf my\_archive.tar folder/  
 -c : creates a .tar archive  
 -v : tells you what is happening (verbose)  
 -f : assembles the archive into one file

Step 3.1, create gzip file (most current):  
 gzip my\_archive.tar  
 to decompress: gunzip my\_archive.tar.gz

Step 3.2, or create a bzip2 file (more powerful but slow):  
 bzip2 my\_archive.tar  
 to decompress: bunzip2 my\_archive.tar.bz2

# Linux Bash Shell Cheat Sheet

## Basic Commands

### Archive and compress data (continued)

step 4, to decompress the .tar file:  
 tar -xvf archive.tar archive.tar

Archive and compress data the fast way:

```
gzip: tar -zcvf my_archive.tar.gz folder/
      decompress: tar -zcvf my_archive.tar.gz Documents/
bzip2: tar -jcvf my_archive.tar.gz folder/
      decompress: tar -jxvf archive.tar.bz2 Documents/
```

Show the content of .tar, .gz or .bz2 without decompressing it:

```
gzip:
      gzip -zt archive.tar.gz
bzip2:
      bzip2 -jtf archive.tar.bz2
tar:
      tar -tf archive.tar
```

tar extra:  
 tar -rvf archive.tar file.txt = add a file to the .tar

You can also directly compress a single file and view the file without decompressing:

Step 1, use gzip or bzip2 to compress the file:  
 gzip numbers.txt

Step 2, view the file without decompressing it:  
 zcat = view the entire file in the console (same as cat)
 zmore = view one screen at a time the content of the file (same as more)
 zless = view one line of the file at a time (same as less)

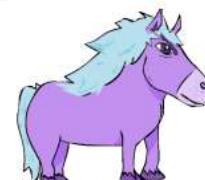
### Installing software

When software is available in the repositories:  
 sudo apt-get install <nameOfSoftware>  
 ex--> sudo apt-get install aptitude

If you download it from the Internets in .gz format (or bz2) - "Compiling from source"  
 Step 1, create a folder to place the file:  
 mkdir /home/username/src << then cd to it

Step 2, with 'ls' verify that the file is there (if not, mv ..file.tar.gz /home/username/src/)

Step 3, decompress the file (if .zip: unzip <file>)
 <--  
 Step 4, use 'ls', you should see a new directory  
 Step 5, cd to the new directory  
 Step 6.1, use ls to verify you have an INSTALL file, then: more INSTALL  
 If you don't have an INSTALL file:  
 Step 6.2, execute ./configure << creates a makefile  
 Step 6.2.1, run make << builds application binaries  
 Step 6.2.2 : switch to root --> su  
 Step 6.2.3 : make install << installs the software  
 Step 7, read the readme file



By @ml874

- [Data Science Cheatsheet \(PDF\)](#)

## Data Science Cheatsheet

Compiled by Maverick Lin (<http://mavericklin.com>)

Last Updated August 13, 2018

### What is Data Science?

Multi-disciplinary field that brings together concepts from computer science, statistics/machine learning, and data analysis to understand and extract insights from the ever-increasing amounts of data.

Two paradigms of data research.

1. **Hypothesis-Driven:** Given a problem, what kind of data do we need to help solve it?
2. **Data-Driven:** Given some data, what interesting problems can be solved with it?

The heart of data science is to always ask questions. Always be curious about the world.

1. What can we learn from this data?
2. What actions can we take once we find whatever it is we are looking for?

### Types of Data

**Structured:** Data that has predefined structures. e.g. tables, spreadsheets, or relational databases.

**Unstructured Data:** Data with no predefined structure, comes in any size or form, cannot be easily stored in tables. e.g. blobs of text, images, audio

**Quantitative Data:** Numerical. e.g. height, weight

**Categorical Data:** Data that can be labeled or divided into groups. e.g. race, sex, hair color.

**Big Data:** Massive datasets, or data that contains greater *variety* arriving in increasing *volumes* and with even-higher *velocity* (3 Vs). Cannot fit in the memory of a single machine.

### Data Sources/Fomats

**Most Common Data Formats** CSV, XML, SQL, JSON, Protocol Buffers

**Data Sources** Companies/Proprietary Data, APIs, Government, Academic, Web Scraping/Crawling

### Main Types of Problems

Two problems arise repeatedly in data science.

**Classification:** Assigning something to a discrete set of possibilities. e.g. spam or non-spam, Democrat or Republican, blood type (A, B, AB, O)

**Regression:** Predicting a numerical value. e.g. someone's income, next year GDP, stock price

### Probability Overview

Probability theory provides a framework for reasoning about likelihood of events.

#### Terminology

**Experiment:** procedure that yields one of a possible set of outcomes e.g. repeatedly tossing a die or coin

**Sample Space S:** set of possible outcomes of an experiment e.g. if tossing a die,  $S = \{1,2,3,4,5,6\}$

**Event E:** set of outcomes of an experiment e.g. event that a roll is 5, or the event that sum of 2 rolls is 7

**Probability of an Outcome s or P(s):** number that satisfies 2 properties

1. for each outcome  $s$ ,  $0 \leq P(s) \leq 1$

2.  $\sum p(s) = 1$

**Probability of Event E:** sum of the probabilities of the outcomes of the experiment:  $p(E) = \sum_{s \in E} p(s)$

**Random Variable V:** numerical function on the outcomes of a probability space

**Expected Value of Random Variable V:**  $E(V) = \sum_{s \in S} p(s) * V(s)$

#### Independence, Conditional, Compound

**Independent Events:** A and B are independent iff:

$$P(A \cap B) = P(A)P(B)$$

$$P(A|B) = P(A)$$

$$P(B|A) = P(B)$$

**Conditional Probability:**  $P(A|B) = P(A,B)/P(B)$

**Bayes Theorem:**  $P(A|B) = P(B|A)P(A)/P(B)$

**Joint Probability:**  $P(A,B) = P(B|A)P(A)$

**Marginal Probability:**  $P(A)$

### Probability Distributions

**Probability Density Function (PDF):** Gives the probability that a rv takes on the value  $x$ :  $p_X(x) = P(X = x)$

**Cumulative Density Function (CDF):** Gives the probability that a random variable is less than or equal to  $x$ :  $F_X(x) = P(X \leq x)$

**Note:** The PDF and the CDF of a given random variable contain exactly the same information.

### Descriptive Statistics

Provides a way of capturing a given data set or sample. There are two main types: **centrality** and **variability** measures.

#### Centrality

**Arithmetic Mean** Useful to characterize symmetric distributions without outliers  $\mu_X = \frac{1}{n} \sum x$

**Geometric Mean** Useful for averaging ratios. Always less than arithmetic mean =  $\sqrt[n]{a_1 a_2 \dots a_n}$

**Median** Exact middle value among a dataset. Useful for skewed distribution or data with outliers.

**Mode** Most frequent element in a dataset.

#### Variability

**Standard Deviation** Measures the squares differences between the individual elements and the mean

$$\sigma = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N-1}}$$

**Variance**  $V = \sigma^2$

#### Interpreting Variance

Variance is an inherent part of the universe. It is impossible to obtain the same results after repeated observations of the same event due to random noise/error. Variance can be explained away by attributing to sampling or measurement errors. Other times, the variance is due to the random fluctuations of the universe.

#### Correlation Analysis

Correlation coefficients  $r(X,Y)$  is a statistic that measures the degree that  $Y$  is a function of  $X$  and vice versa. Correlation values range from -1 to 1, where 1 means fully correlated, -1 means negatively-correlated, and 0 means no correlation.

**Pearson Coefficient.** Measures the degree of the relationship between linearly related variables

$$r = \frac{Cov(X,Y)}{\sigma(X)\sigma(Y)}$$

**Spearman Rank Coefficient** Computed on ranks and depicts monotonic relationships

**Note:** Correlation does not imply causation!

### Data Cleaning

Data Cleaning is the process of turning raw data into a clean and analyzable data set. "Garbage in, garbage out." Make sure garbage doesn't get put in.

#### Errors vs. Artifacts

1. **Errors:** information that is lost during acquisition and can never be recovered e.g. power outage, crashed servers
2. **Artifacts:** systematic problems that arise from the data cleaning process. these problems can be corrected but we must first discover them

#### Data Compatibility

Data compatibility problems arise when merging datasets. Make sure you are comparing "apples to apples" and not "apples to oranges". Main types of conversions/unifications:

- units (metric vs. imperial)
- numbers (decimals vs. integers),
- names (John Smith vs. Smith, John),
- time/dates (UNIX vs. UTC vs. GMT),
- currency (currency type, inflation-adjusted, dividends)

#### Data Imputation

Process of dealing with missing values. The proper methods depend on the type of data we are working with. General methods include:

- Drop all records containing missing data
- Heuristic-Based: make a reasonable guess based on knowledge of the underlying domain
- Mean Value: fill in missing data with the mean
- Random Value
- Nearest Neighbor: fill in missing data using similar data points
- Interpolation: use a method like linear regression to predict the value of the missing data

#### Outlier Detection

Outliers can interfere with analysis and often arise from mistakes during data collection. It makes sense to run a "sanity check".

#### Miscellaneous

Lowercasing, removing non-alphanumeric, repairing, unidecode, removing unknown characters

**Note:** When cleaning data, always maintain both the raw data and the cleaned version(s). The raw data should be kept intact and preserved for future use. Any type of data cleaning/analysis should be done on a copy of the raw data.

### Feature Engineering

Feature engineering is the process of using domain knowledge to create features or input variables that help machine learning algorithms perform better. Done correctly, it can help increase the predictive power of your models. Feature engineering is more of an art than science. FE is one of the most important steps in creating a good model. As Andrew Ng puts it:

*"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."*

#### Continuous Data

**Raw Measures:** data that hasn't been transformed yet

**Rounding:** sometimes precision is noise; round to nearest integer, decimal etc..

**Scaling:** log, z-score, minmax scale

Imputation: fill in missing values using mean, median, model output, etc..

**Binning:** transforming numeric features into categorical ones (or binned) e.g. values between 1-10 belong to A, between 10-20 belong to B, etc.

**Interactions:** interactions between features: e.g. subtraction, addition, multiplication, statistical test

**Statistical:** log/power transform (helps turn skewed distributions more normal), Box-Cox

**Row Statistics:** number of NaN's, 0's, negative values, max, min, etc

**Dimensionality Reduction:** using PCA, clustering, factor analysis etc

#### Discrete Data

**Encoding:** since some ML algorithms cannot work on categorical data, we need to turn categorical data into numerical data or vectors

**Ordinal Values:** convert each distinct feature into a random number (e.g. [r,g,b] becomes [1,2,3])

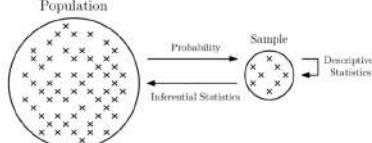
**One-Hot Encoding:** each of the m features becomes a vector of length m with containing only one 1 (e.g. [r, g, b] becomes [[1,0,0],[0,1,0],[0,0,1]])

**Feature Hashing Scheme:** turns arbitrary features into indices in a vector or matrix

**Embeddings:** if using words, convert words to vectors (word embeddings)

### Statistical Analysis

Process of statistical reasoning: there is an underlying population of possible things we can potentially observe and only a small subset of them are actually sampled (ideally at random). Probability theory describes what properties our sample should have given the properties of the population, but **statistical inference** allows us to deduce what the full population is like after analyzing the sample.



#### Sampling From Distributions

**Inverse Transform Sampling** Sampling points from a given probability distribution is sometimes necessary to run simulations or whether your data fits a particular distribution. The general technique is called *inverse transform sampling* or Smirnov transform. First draw a random number  $p$  between [0,1]. Compute value  $x$  such that the CDF equals  $p$ :  $F_X(x) = p$ . Use  $x$  as the value to be the random value drawn from the distribution described by  $F_X(x)$ .

**Monte Carlo Sampling** In higher dimensions, correctly sampling from a given distribution becomes more tricky. Generally want to use Monte Carlo methods, which typically follow these rules: define a domain of possible inputs, generate random inputs from a probability distribution over the domain, perform a deterministic calculation, and analyze the results.

### Classic Statistical Distributions

#### Binomial Distribution (Discrete)

Assume  $X$  is distributed  $\text{Bin}(n,p)$ .  $X$  is the number of "successes" that we will achieve in  $n$  independent trials, where each trial is either a success or failure and each success occurs with the same probability  $p$  and each failure occurs with probability  $q=1-p$ .  
 PDF:  $P(X=x) = \binom{n}{x} p^x (1-p)^{n-x}$   
 EV:  $\mu = np$  Variance =  $npq$

#### Normal/Gaussian Distribution (Continuous)

Assume  $X$  is distributed  $\mathcal{N}(\mu, \sigma^2)$ . It is a bell-shaped and symmetric distribution. Bulk of the values lie close to the mean and no value is too extreme. Generalization of the binomial distribution as  $\mu \rightarrow \infty$ .

PDF:  $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$

EV:  $\mu$  Variance:  $\sigma^2$

**Implications:** 68%-95%-99% rule. 68% of probability mass fall within  $1\sigma$  of the mean, 95% within  $2\sigma$ , and 99.7% within  $3\sigma$ .

#### Poisson Distribution (Discrete)

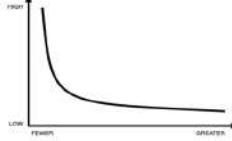
Assume  $X$  is distributed  $\text{Pois}(\lambda)$ . Poisson expresses the probability of a given number of events occurring in a fixed interval of time/space if these events occur independently and with a known constant rate  $\lambda$ .

PDF:  $P(x) = \frac{e^{-\lambda} \lambda^x}{x!}$  EV:  $\lambda$  Variance =  $\lambda$

#### Power Law Distributions (Discrete)

Many data distributions have much longer tails than the normal or Poisson distributions. In other words, the change in one quantity varies as a *power* of another quantity. It helps measure the inequality in the world, e.g. wealth, word frequency and Pareto Principle (80/20 Rule).

PDF:  $P(X=x) = cx^{-\alpha}$ , where  $\alpha$  is the law's exponent and  $c$  is the normalizing constant



### Modeling- Overview

Modeling is the process of incorporating information into a tool which can forecast and make predictions. Usually, we are dealing with statistical modeling where we want to analyze relationships between variables. Formally, we want to estimate a function  $f(X)$  such that:

$$Y = f(X) + \epsilon$$

where  $X = (X_1, X_2, \dots, X_p)$  represents the input variables,  $Y$  represents the output variable, and  $\epsilon$  represents random error.

**Statistical learning** is set of approaches for estimating this  $f(X)$ .

#### Why Estimate $f(X)$ ?

**Prediction:** once we have a good estimate  $\hat{f}(X)$ , we can use it to make predictions on new data. We treat  $\hat{f}$  as a black box, since we only care about the accuracy of the predictions, not why or how it works.

**Inference:** we want to understand the relationship between  $X$  and  $Y$ . We can no longer treat  $\hat{f}$  as a black box since we want to understand how  $Y$  changes with respect to  $X = (X_1, X_2, \dots, X_p)$

#### More About $\epsilon$

The error term  $\epsilon$  is composed of the reducible and irreducible error, which will prevent us from ever obtaining a perfect  $\hat{f}$  estimate.

- **Reducible:** error that can potentially be reduced by using the most appropriate statistical learning technique to estimate  $f$ . The goal is to minimize the reducible error.
- **Irreducible:** error that cannot be reduced no matter how well we estimate  $f$ . Irreducible error is unknown and unmeasurable and will always be an upper bound for  $\epsilon$ .

**Note:** There will always be trade-offs between model flexibility (prediction) and model interpretability (inference). This is just another case of the bias-variance trade-off. Typically, as flexibility increases, interpretability decreases. Much of statistical learning/modeling is finding a way to balance the two.

### Modeling- Philosophies

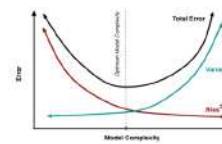
Modeling is the process of incorporating information into a tool which can forecast and make predictions. Designing and validating models is important, as well as evaluating the performance of models. Note that the best forecasting model may not be the most accurate one.

#### Philosophies of Modeling

**Occam's Razor** Philosophical principle that the simplest explanation is the best explanation. In modeling, if we are given two models that predict equally well, we should choose the simpler one. Choosing the more complex one can often result in overfitting.

**Bias Variance Trade-Off** Inherent part of predictive modeling, where models with lower bias will have higher variance and vice versa. Goal is to achieve low bias and low variance.

- **Bias:** error from incorrect assumptions to make target function easier to learn (high bias  $\rightarrow$  missing relevant relations or underfitting)
- **Variance:** error from sensitivity to fluctuations in the dataset, or how much the target estimate would differ if different training data was used (high variance  $\rightarrow$  modeling noise or overfitting)



**No Free Lunch Theorem** No single machine learning algorithm is better than all the others on all problems. It is common to try multiple models and find one that works best for a particular problem.

#### Thinking Like Nate Silver

1. **Think Probabilistically** Probabilistic forecasts are more meaningful than concrete statements and should be reported as probability distributions (including  $\sigma$  along with mean prediction  $\mu$ ).

2. **Incorporate New Information** Use live models, which continually updates using new information. To update, use Bayesian reasoning to calculate how probabilities change in response to new evidence.

3. **Look For Consensus Forecasts** Use multiple distinct sources of evidence. Some models operate this way, such as boosting and bagging, which uses large number of weak classifiers to produce a strong one.

### Modeling- Taxonomy

There are many different types of models. It is important to understand the trade-offs and when to use a certain type of model.

#### Parametric vs. Nonparametric

- **Parametric:** models that first make an assumption about a function form, or shape, of  $f$  (linear). Then fits the model. This reduces estimating  $f$  to just estimating set of parameters, but if our assumption was wrong, will lead to bad results.
- **Non-Parametric:** models that don't make any assumptions about  $f$ , which allows them to fit a wider range of shapes; but may lead to overfitting

#### Supervised vs. Unsupervised

- **Supervised:** models that fit input variables  $x_i = (x_1, x_2, \dots, x_n)$  to a known output variables  $y_i = (y_1, y_2, \dots, y_n)$
- **Unsupervised:** models that take in input variables  $x_i = (x_1, x_2, \dots, x_n)$ , but they do not have an associated output to supervise the training. The goal is to understand relationships between the variables or observations.

#### Blackbox vs. Descriptive

- **Blackbox:** models that make decisions, but we do not know what happens "under the hood" e.g. deep learning, neural networks
- **Descriptive:** models that provide insight into *why* they make their decisions e.g. linear regression, decision trees

#### First-Principle vs. Data-Driven

- **First-Principle:** models based on a prior belief of how the system under investigation works, incorporates domain knowledge (ad-hoc)
- **Data-Driven:** models based on observed correlations between input and output variables

#### Deterministic vs. Stochastic

- **Deterministic:** models that produce a single "prediction" e.g. yes or no, true or false
- **Stochastic:** models that produce probability distributions over possible events

#### Flat vs. Hierarchical

- **Flat:** models that solve problems on a single level, no notion of subproblems
- **Hierarchical:** models that solve several different nested subproblems

### Modeling- Evaluation Metrics

Need to determine how good our model is. Best way to assess models is out-of-sample predictions (data points your model has never seen).

#### Classification

	Predicted Yes	Predicted No
Actual Yes	True Positives (TP)	False Negatives (FN)
Actual No	False Positives (FP)	True Negatives (TN)

**Accuracy:** ratio of correct predictions over total predictions. Misleading when class sizes are substantially different.  $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

**Precision:** how often the classifier is correct when it predicts positive:  $precision = \frac{TP}{TP+FP}$

**Recall:** how often the classifier is correct for all positive instances:  $recall = \frac{TP}{TP+FN}$

**F-Score:** single measurement to describe performance:  $F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

**ROC Curves:** plots true positive rates and false positive rates for various thresholds, or where the model determines if a data point is positive or negative (e.g. if  $>0.8$ , classify as positive). Best possible area under the ROC curve (AUC) is 1, while random is 0.5, or the main diagonal line.

#### Regression

Errors are defined as the difference between a prediction  $\hat{y}$  and the actual result  $y$ .

**Absolute Error:**  $\Delta = \hat{y} - y$

**Squared Error:**  $\Delta^2 = (\hat{y} - y)^2$

**Mean-Squared Error:**  $MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$

**Root Mean-Squared Error:**  $RMSD = \sqrt{MSE}$

**Absolute Error Distribution:** Plot absolute error distribution: should be symmetric, centered around 0, bell-shaped, and contain rare extreme outliers.

### Modeling- Evaluation Environment

Evaluation metrics provides use with the tools to estimate errors, but what should be the process to obtain the best estimate? Resampling involves repeatedly drawing samples from a training set and refitting a model to each sample, which provides us with additional information compared to fitting the model once, such as obtaining a better estimate for the test error.

#### Key Concepts

**Training Data:** data used to fit your models or the set used for learning

**Validation Data:** data used to tune the parameters of a model

**Test Data:** data used to evaluate how good your model is. Ideally your model should never touch this data until final testing/evaluation

#### Cross Validation

Class of methods that estimate test error by holding out a subset of training data from the fitting process.

**Validation Set:** split data into training set and validation set. Train model on training and estimate test error using validation. e.g. 80-20 split

**Leave-One-Out CV (LOOCV):** split data into training set and validation set, but the validation set consists of 1 observation. Then repeat n-1 times until all observations have been used as validation. Test error is the average of these n test error estimates.

**k-Fold CV:** randomly divide data into k groups (folds) of approximately equal size. First fold is used as validation and the rest as training. Then repeat k times and find average of the k estimates.

#### Bootstrapping

Methods that rely on random sampling with replacement. Bootstrapping helps with quantifying uncertainty associated with a given estimate or model.

#### Amplifying Small Data Sets

What can we do if we don't have enough data?

- **Create Negative Examples-** e.g. classifying presidential candidates, most people would be unqualified so label most as unqualified
- **Synthetic Data-** create additional data by adding noise to the real data

### Linear Regression

Linear regression is a simple and useful tool for predicting a quantitative response. The relationship between input variables  $X = (X_1, X_2, \dots, X_p)$  and output variable  $Y$  takes the form:

$$Y \approx \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon$$

$\beta_0, \dots, \beta_p$  are the unknown coefficients (parameters) which we are trying to determine. The best coefficients will lead us to the best "fit", which can be found by minimizing the *residual sum squares* (RSS), or the sum of the differences between the actual  $i$ th value and the predicted  $i$ th value.  $RSS = \sum_{i=1}^n e_i^2$ , where  $e_i = y_i - \hat{y}_i$ .

#### How to find best fit?

**Matrix Form:** We can solve the closed-form equation for coefficient vector  $w$ :  $w = (X^T X)^{-1} X^T Y$ .  $X$  represents the input data and  $Y$  represents the output data. This method is used for smaller matrices, since inverting a matrix is computationally expensive.

**Gradient Descent:** First-order optimization algorithm. We can find the minimum of a *convex* function by starting at an arbitrary point and repeatedly take steps in the downward direction, which can be found by taking the negative direction of the gradient. After several iterations, we will eventually converge to the minimum. In our case, the minimum corresponds to the coefficients with the minimum error, or the best line of fit. The learning rate  $\alpha$  determines the size of the steps we take in the downward direction.

Gradient descent algorithm in two dimensions. Repeat until convergence.

1.  $w_0^{t+1} := w_0^t - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1)$
2.  $w_1^{t+1} := w_1^t - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1)$

For non-convex functions, gradient descent no longer guarantees an optimal solution since there may be local minimas. Instead, we should run the algorithm from different starting points and use the best local minima we find for the solution.

**Stochastic Gradient Descent:** instead of taking a step after sampling the *entire* training set, we take a small batch of training data at random to determine our next step. Computationally more efficient and may lead to faster convergence.

### Linear Regression II

#### Improving Linear Regression

**Subset/Feature Selection:** approach involves identifying a subset of the  $p$  predictors that we believe to be best related to the response. Then we fit model using the reduced set of variables.

#### • Best, Forward, and Backward Subset Selection

**Shrinkage/Regularization:** all variables are used, but estimated coefficients are shrunken towards zero relative to the least squares estimate.  $\lambda$  represents the tuning parameter- as  $\lambda$  increases, flexibility decreases  $\rightarrow$  decreased variance but increased bias. The tuning parameter is key in determining the sweet spot between under and over-fitting. In addition, while Ridge will always produce a model with  $p$  variables, Lasso can force coefficients to be equal to zero.

#### • Lasso (L1): $\min \text{RSS} + \lambda \sum_{j=1}^p |\beta_j|$

#### • Ridge (L2): $\min \text{RSS} + \lambda \sum_{j=1}^p \beta_j^2$

**Dimension Reduction:** projecting  $p$  predictors into a  $M$ -dimensional subspace, where  $M < p$ . This is achieved by computing  $M$  different linear combinations of the variables. Can use PCA.

**Miscellaneous:** Removing outliers, feature scaling, removing multicollinearity (correlated variables)

### Evaluating Model Accuracy

Residual Standard Error (RSE):  $\text{RSE} = \sqrt{\frac{1}{n-2} \text{RSS}}$ . Generally, the smaller the better.

$R^2$ : Measure of fit that represents the proportion of variance explained, or the *variability in Y that can be explained using X*. It takes on a value between 0 and 1. Generally the higher the better.  $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$ , where Total Sum of Squares (TSS) =  $\sum (y_i - \bar{y})^2$

### Evaluating Coefficient Estimates

Standard Error (SE) of the coefficients can be used to perform hypothesis tests on the coefficients:

$H_0$ : No relationship between  $X$  and  $Y$ ,  $H_0$ : Some relationship exists. A p-value can be obtained and can be interpreted as follows: a small p-value indicates that a relationship between the predictor ( $X$ ) and the response ( $Y$ ) exists. Typical p-value cutoffs are around 5 or 1 %.

### Logistic Regression

Logistic regression is used for classification, where the response variable is categorical rather than numerical.

The model works by predicting the probability that  $Y$  belongs to a particular category by first fitting the data to a linear regression model, which is then passed to the logistic function (below). The logistic function will always produce a S-shaped curve, so regardless of  $X$ , we can always obtain a sensible answer (between 0 and 1). If the probability is above a certain predetermined threshold (e.g.  $P(\text{Yes}) > 0.5$ ), then the model will predict Yes.

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

#### How to find best coefficients?

**Maximum Likelihood:** The coefficients  $\beta_0, \dots, \beta_p$  are unknown and must be estimated from the training data. We seek estimates for  $\beta_0, \dots, \beta_p$  such that the predicted probability  $p(x_i)$  of each observation is a number close to one if it's observed in a certain class and close to zero otherwise. This is done by maximizing the likelihood function:

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_i))$$

### Potential Issues

**Imbalanced Classes:** imbalance in classes in training data lead to poor classifiers. It can result in a lot of false positives and also lead to few training data. Solutions include forcing balanced data by removing observations from the larger class, replicate data from the smaller class, or heavily weight the training examples toward instances of the larger class.

**Multi-Class Classification:** the more classes you try to predict, the harder it will be for the classifier to be effective. It is possible with logistic regression, but another approach, such as Linear Discriminant Analysis (LDA), may prove better.

### Distance/Network Methods

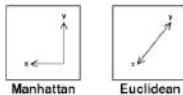
Interpreting examples as points in space provides a way to find natural groupings or clusters among data e.g. which stars are the closest to our sun? Networks can also be built from point sets (vertices) by connecting related points.

#### Measuring Distances/Similarity Measure

There are several ways of measuring distances between points  $a$  and  $b$  in  $d$  dimensions- with closer distances implying similarity.

**Minkowski Distance Metric:**  $d_k(a, b) = \sqrt[k]{\sum_{i=1}^d |a_i - b_i|^k}$ . The parameter  $k$  provides a way to tradeoff between the largest and the total dimensional difference. In other words, larger values of  $k$  place more emphasis on large differences between feature values than smaller values. Selecting the right  $k$  can significantly impact the the meaningfulness of your distance function. The most popular values are 1 and 2.

- Manhattan ( $k=1$ ): city block distance, or the sum of the absolute difference between two points
- Euclidean ( $k=2$ ): straight line distance



**Weighted Minkowski:**  $d_k(a, b) = \sqrt[k]{\sum_{i=1}^d w_i |a_i - b_i|^k}$ , in some scenarios, not all dimensions are equal. Can convey this idea using  $w_i$ . Generally not a good idea- should normalize data by Z-scores before computing distances.

**Cosine Similarity:**  $\cos(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$ , calculates the similarity between 2 non-zero vectors, where  $a \cdot b$  is the dot product (normalized between 0 and 1), higher values imply more similar vectors

**Kullback-Leibler Divergence:**  $KL(A||B) = \sum_{i=1}^d a_i \log_2 \frac{a_i}{b_i}$ . KL divergence measures the distances between probability distributions by measuring the uncertainty gained or uncertainty lost when replacing distribution A with distribution B. However, this is not a metric but forms the basis for the Jensen-Shannon Divergence Metric.

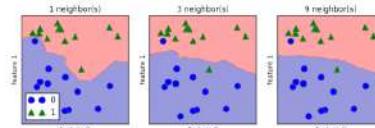
**Jensen-Shannon:**  $JS(A, B) = \frac{1}{2} KL(A||M) + \frac{1}{2} KL(M||B)$ , where  $M$  is the average of  $A$  and  $B$ . The JS function is the right metric for calculating distances between probability distributions

### Nearest Neighbor Classification

Distance functions allow us to identify the points closest to a given target, or the *nearest neighbors (NN)* to a given point. The advantages of NN include simplicity, interpretability and non-linearity.

#### k-Nearest Neighbors

Given a positive integer  $k$  and a point  $x_0$ , the KNN classifier first identifies  $k$  points in the training data most similar to  $x_0$ , then estimates the conditional probability of  $x_0$  being in class  $j$  as the fraction of the  $k$  points whose values belong to  $j$ . The optimal value for  $k$  can be found using cross validation.



#### KNN Algorithm

1. Compute distance  $D(a, b)$  from point  $b$  to all points
2. Select  $k$  closest points and their labels
3. Output class with most frequent labels in  $k$  points

#### Optimizing KNN

Comparing a query point  $a$  in  $d$  dimensions against  $n$  training examples computes with a runtime of  $O(nd)$ , which can cause lag as points reach millions or billions. Popular choices to speed up KNN include:

- **Voronoi Diagrams:** partitioning plane into regions based on distance to points in a specific subset of the plane
- **Grid Indexes:** carve up space into  $d$ -dimensional boxes or grids and calculate the NN in the same cell as the point
- **Locality Sensitive Hashing (LSH):** abandons the idea of finding the exact nearest neighbors. Instead, batch up nearby points to quickly find the most appropriate bucket  $B$  for our query point. LSH is defined by a hash function  $h(p)$  that takes a point/vector as input and produces a number/code as output, such that it is likely that  $h(a) = h(b)$  if  $a$  and  $b$  are close to each other, and  $h(a) \neq h(b)$  if they are far apart.

### Clustering

**Clustering** is the problem of grouping points by similarity using distance metrics, which ideally reflect the similarities you are looking for. Often items come from logical "sources" and clustering is a good way to reveal those origins. Perhaps the first thing to do with any data set. Possible applications include: hypothesis development, modeling over smaller subsets of data, data reduction, outlier detection.

#### K-Means Clustering

Simple and elegant algorithm to partition a dataset into  $K$  distinct, non-overlapping clusters.

1. Choose a  $K$ . Randomly assign a number between 1 and  $K$  to each observation. These serve as initial cluster assignments
2. Iterate until cluster assignments stop changing
  - (a) For each of the  $K$  clusters, compute the cluster centroid. The  $k$ th cluster centroid is the vector of the  $p$  feature means for the observations in the  $k$ th cluster.
  - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using distance metric).

Since the results of the algorithm depends on the initial random assignments, it is a good idea to repeat the algorithm from different random initializations to obtain the best overall results. Can use MSE to determine which cluster assignment is better.

#### Hierarchical Clustering

Alternative clustering algorithm that does not require us to commit to a particular  $K$ . Another advantage is that it results in a nice visualization called a **dendrogram**. Observations that fuse at bottom are similar, where those at the top are quite different- we draw conclusions based on the location on the vertical rather than horizontal axis.

1. Begin with  $n$  observations and a measure of all the  $\binom{n(n-1)}{2}$  pairwise dissimilarities. Treat each observation as its own cluster.

2. For  $i = n, n-1, \dots, 2$ 
  - (a) Examining all pairwise inter-cluster dissimilarities among the  $i$  clusters and identify the pair of clusters that are least dissimilar (most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates height in dendrogram where fusion should be placed.
  - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using distance metric).

**Linkage:** Complete (max dissimilarity), Single (min), Average, Centroid (between centroids of cluster A and B)

**Machine Learning Part I****Comparing ML Algorithms**

**Power and Expressibility:** ML methods differ in terms of complexity. Linear regression fits linear functions while NN defend piecewise-linear separation boundaries. More complex models can provide more accurate models, but at the risk of overfitting.

**Interpretability:** some models are more transparent and understandable than others (white box vs. black box models)

**Ease of Use:** some models feature few parameters/decisions (linear regression/NN), while others require more decision making to optimize (SVMs)

**Training Speed:** models differ in how fast they fit the necessary parameters

**Prediction Speed:** models differ in how fast they make predictions given a query

Method	Power of Expression	Ease of Interpretation	Ease of Use	Training Speed	Prediction Speed
Linear Regression	low	high	high	fast	fast
Nearest Neighbor	high	low	medium	slow	fast
Naive Bayes	medium	high	high	fast	fast
Decision Trees	high	medium	medium	fast	fast
Support Vector Machines	high	low	medium	slow	fast
Boosting	high	low	medium	slow	fast
Graphical Models	high	medium	medium	fast	fast
Deep Learning	high	low	high	slow	fast

**Naïve Bayes**

Naïve Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naïve" assumption of independence between every pair of features.

**Problem:** Suppose we need to classify vector  $X = x_1 \dots x_n$  into  $m$  classes,  $C_1 \dots C_m$ . We need to compute the probability of each possible class given  $X$ , so we can assign  $X$  the label of the class with highest probability. We can calculate a probability using the Bayes' Theorem:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

Where:

1.  $P(C_i)$ : the prior probability of belonging to class  $i$
2.  $P(X)$ : normalizing constant, or probability of seeing the given input vector over all possible input vectors
3.  $P(X|C_i)$ : the conditional probability of seeing input vector  $X$  given we know the class is  $C_i$

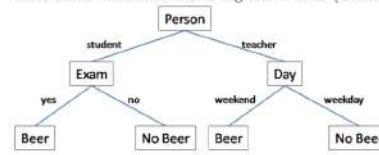
The prediction model will formally look like:

$$C(X) = \operatorname{argmax}_{i \in \text{classes}(t)} \frac{P(X|C_i)P(C_i)}{P(X)}$$

where  $C(X)$  is the prediction returned for input  $X$ .

**Machine Learning Part II****Decision Trees**

Binary branching structure used to classify an arbitrary input vector  $X$ . Each node in the tree contains a simple feature comparison against some field ( $x_i > 427$ ). Result of each comparison is either true or false, which determines if we should proceed along to the left or right child of the given node. Also known as sometimes called classification and regression trees (CART).



**Advantages:** Non-linearity, support for categorical variables, easy to interpret, application to regression.

**Disadvantages:** Prone to overfitting, instable (not robust to noise), high variance, low bias

**Note:** rarely do models just use one decision tree. Instead, we aggregate many decision trees using methods like ensembling, bagging, and boosting.

**Ensembles, Bagging, Random Forests, Boosting**

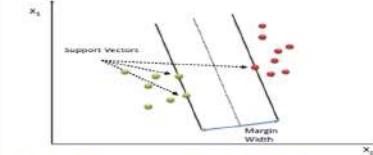
**Ensemble learning** is the strategy of combining many different classifiers/models into one predictive model. It revolves around the idea of voting: a so-called "wisdom of crowds" approach. The most predicted class will be the final prediction.

**Bagging:** ensemble method that works by taking  $B$  bootstrapped subsamples of the training data and constructing  $B$  trees, each tree training on a distinct subsample as **Random Forests**: builds on bagging by decorrelating the trees. We do everything the same like in bagging, but when we build the trees, everytime we consider a split, a random sample of the  $p$  predictors is chosen as split candidates, not the full set (typically  $m \approx \sqrt{p}$ ). When  $m = p$ , then we are just doing bagging.

**Boosting:** the main idea is to improve our model where it is not performing well by using information from previously constructed classifiers. Slow learner. Has 3 tuning parameters: number of classifiers  $B$ , learning parameter  $\lambda$ , interaction depth  $d$  (controls interaction order of model).

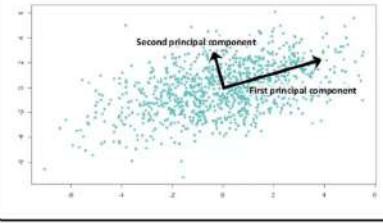
**Machine Learning Part III****Support Vector Machines**

Work by constructing a hyperplane that separates points between two classes. The hyperplane is determined using the maximal margin hyperplane, which is the hyperplane that is the maximum distance from the training observations. This distance is called the margin. Points that fall on one side of the hyperplane are classified as -1 and the other +1.

**Principal Component Analysis (PCA)**

**Principal components** allow us to summarize a set of correlated variables with a smaller set of variables that collectively explain most of the variability in the original set. Essentially, we are "dropping" the least important feature variables.

**Principal Component Analysis** is the process by which principal components are calculated and the use of them to analyzing and understanding the data. PCA is an unsupervised approach and is used for dimensionality reduction, feature extraction, and data visualization. Variables after performing PCA are independent. Scaling variables is also important while performing PCA.

**Machine Learning Part IV****ML Terminology and Concepts**

**Features:** input data/variables used by the ML model

**Feature Engineering:** transforming input features to be more useful for the models. e.g. mapping categories to buckets, normalizing between -1 and 1, removing null

**Train/Eval/Test:** training is data used to optimize the model, evaluation is used to assess the model on new data during training, test is used to provide the final result

**Classification/Regression:** regression is prediction a number (e.g. housing price), classification is prediction from a set of categories(e.g. predicting red/blue/green)

**Linear Regression:** predicts an output by multiplying and summing input features with weights and biases

**Logistic Regression:** similar to linear regression but predicts a probability

**Overfitting:** model performs great on the input data but poorly on the test data (combat by dropout, early stopping, or reduce # of nodes or layers)

**Bias/Variance:** how much output is determined by the features. more variance often can mean overfitting, more bias can mean a bad model

**Regularization:** variety of approaches to reduce overfitting, including adding the weights to the loss function, randomly dropping layers (dropout)

**Ensemble Learning:** training multiple models with different parameters to solve the same problem

**A/B testing:** statistical way of comparing 2+ techniques to determine which technique performs better and also if difference is statistically significant

**Baseline Model:** simple model/heuristic used as reference point for comparing how well a model is performing

**Bias:** prejudice or favoritism towards some things, people, or groups over others that can affect collection/sampling and interpretation of data, the design of a system, and how users interact with a system

**Dynamic Model:** model that is trained online in a continuously updating fashion

**Static Model:** model that is trained offline

**Normalization:** process of converting an actual range of values into a standard range of values, typically -1 to +1

**Independently and Identically Distributed (i.i.d.):** data drawn from a distribution that doesn't change, and where each value drawn doesn't depend on previously drawn values; ideal but rarely found in real life

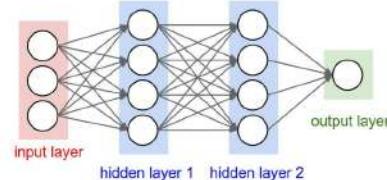
**Hyperparameters:** the "knobs" that you tweak during successive runs of training a model

**Generalization:** refers to a model's ability to make correct predictions on new, previously unseen data as opposed to the data used to train the model

**Cross-Entropy:** quantifies the difference between two probability distributions

**Deep Learning Part I****What is Deep Learning?**

Deep learning is a subset of machine learning. One popular DL technique is based on Neural Networks (NN), which loosely mimic the human brain and the code structures are arranged in layers. Each layer's input is the previous layer's output, which yields progressively higher-level features and defines a hierarchy. A Deep Neural Network is just a NN that has more than 1 hidden layer.



input layer      hidden layer 1      hidden layer 2      output layer

Recall that statistical learning is all about approximating  $f(X)$ . Neural networks are known as **universal approximators**, meaning no matter how complex a function is, there exists a NN that can (approximately) do the job. We can increase the approximation (or complexity) by adding more hidden layers and neurons.

**Popular Architectures**

There are different kinds of NNs that are suitable for certain problems, which depend on the NN's architecture.

**Linear Classifier:** takes input features and combines them with weights and biases to predict output value

**DNN:** deep neural net, contains intermediate layers of nodes that represent "hidden features" and activation functions to represent non-linearity

**CNN:** convolutional NN, has a combination of convolutional, pooling, dense layers, popular for image classification.

**Transfer Learning:** use existing trained models as starting points and add additional layers for the specific use case. idea is that highly trained existing models know general features that serve as a good starting point for training a small network on specific examples

**RNN:** recurrent NN, designed for handling a sequence of inputs that have "memory" of the sequence. LSTMs are a fancy version of RNNs, popular for NLP

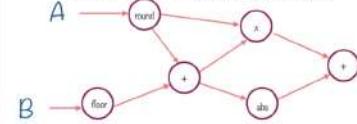
**GAN:** general adversarial NN, one model creates fake examples, and another model is served both fake example and real examples and is asked to distinguish

**Wide and Deep:** combines linear classifiers with deep neural net classifiers, "wide" linear parts represent memorizing specific examples and "deep" parts represent understanding high level features

**Deep Learning Part II****Tensorflow**

Tensorflow is an open source software library for numerical computation using data flow graphs. Everything in TF is a graph, where nodes represent operations on data and edges represent the data. Phase 1 of TF is building up a computation graph and phase 2 is executing it. It is also distributed, meaning it can run on either a cluster of machines or just a single machine.

TF is extremely popular/suitable for working with Neural Networks, since the way TF sets up the computational graph pretty much resembles a NN.

**Tensors Flow Through the Graph**

$$Y = \operatorname{round}(A) + \operatorname{floor}(B) * \operatorname{round}(A) + \operatorname{abs}(\operatorname{round}(A)) + \operatorname{floor}(B)$$

**Tensors**

In a graph, tensors are the edges and are multidimensional data arrays that flow through the graph. Central unit of data in TF and consists of a set of primitive values shaped into an array of any number of dimensions.

A tensor is characterized by its rank (# dimensions in tensor), shape (# of dimensions and size of each dimension), data type (data type of each element in tensor).

**Placeholders and Variables**

**Variables:** best way to represent shared, persistent state manipulated by your program. These are the parameters of the ML model are altered/trained during the training process. Training variables.

**Placeholders:** way to specify inputs into a graph that hold the place for a Tensor that will be fed at runtime. They are assigned once, do not change after. Input nodes

**Deep Learning Part III****Deep Learning Terminology and Concepts**

**Neuron:** node in a NN, typically taking in multiple input values and generating one output value, calculates the output value by applying an activation function (nonlinear transformation) to a weighted sum of input values  
**Weights:** edges in a NN, the goal of training is to determine the optimal weight for each feature; if weight = 0, corresponding feature does not contribute

**Neural Network:** composed of neurons (simple building blocks that actually "learn"), contains activation functions that makes it possible to predict non-linear outputs

**Activation Functions:** mathematical functions that introduce non-linearity to a network e.g. RELU, tanh

**Sigmoid Function:** function that maps very negative numbers to a number very close to 0, huge numbers close to 1, and 0 to .5. Useful for predicting probabilities

**Gradient Descent/Backpropagation:** fundamental loss optimizer algorithms, of which the other optimizers are usually based. Backpropagation is similar to gradient descent but for neural nets

**Optimizer:** operation that changes the weights and biases to reduce loss e.g. Adagrad or Adam

**Weights / Biases:** weights are values that the input features are multiplied by to predict an output value. Biases are the value of the output given a weight of 0.

**Converge:** algorithm that converges will eventually reach an optimal answer, even if very slowly. An algorithm that doesn't converge may never reach an optimal answer.

**Learning Rate:** rate at which optimizers change weights and biases. High learning rate generally trains faster but risks not converging, whereas a lower rate trains slower

**Numerical Instability:** issues with very large/small values due to limits of floating point numbers in computers

**Embeddings:** mapping from discrete objects, such as words, to vectors of real numbers. useful because classifiers/neural networks work well on vectors of real numbers

**Convolutional Layer:** series of convolutional operations, each acting on a different slice of the input matrix

**Dropout:** method for regularization in training NNs, works by removing a random selection of some units in a network layer for a single gradient step

**Early Stopping:** method for regularization that involves ending model training early

**Gradient Descent:** technique to minimize loss by computing the gradients of loss with respect to the model's parameters, conditioned on training data

**Pooling:** Reducing a matrix (or matrices) created by an earlier convolutional layer to a smaller matrix. Pooling usually involves taking either the maximum or average value across the pooled area

**Big Data- Hadoop Overview**

Data can no longer fit in memory on one machine (monolithic), so a new way of computing was devised using a group of computers to process this "big data" (distributed). Such a group is called a cluster, which makes up server farms. All of these servers have to be coordinated in the following ways: partition data, coordinate computing tasks, handle fault tolerance/recovery, and allocate capacity to process.

**Hadoop**

Hadoop is an open source *distributed* processing framework that manages data processing and storage for big data applications running in clustered systems. It is comprised of 3 main components:

- **Hadoop Distributed File System (HDFS):** a distributed file system that provides high-throughput access to application data by partitioning data across many machines
- **YARN:** framework for job scheduling and cluster resource management (task coordination)
- **MapReduce:** YARN-based system for parallel processing of large data sets on multiple machines

**HDFS**

Each disk on a different machine in a cluster is comprised of 1 master node and the rest are workers/data nodes. The **master node** manages the overall file system by storing the directory structure and the metadata of the files. The **data nodes** physically store the data. Large files are broken up and distributed across multiple machines, which are also replicated across multiple machines to provide fault tolerance.

**MapReduce**

Parallel programming paradigm which allows for processing of huge amounts of data by running processes on multiple machines. Defining a MapReduce job requires two stages: map and reduce.

- **Map:** operation to be performed in parallel on small portions of the dataset. The output is a key-value pair  $\langle K, V \rangle$
- **Reduce:** operation to combine the results of Map

**YARN- Yet Another Resource Negotiator**

Coordinates tasks running on the cluster and assigns new nodes in case of failure. Comprised of 2 subcomponents: the resource manager and the node manager. The **resource manager** runs on a single master node and schedules tasks across nodes. The **node manager** runs on all other nodes and manages tasks on the individual node.

**Big Data- Hadoop Ecosystem**

An entire ecosystem of tools have emerged around Hadoop, which are based on interacting with HDFS. Below are some popular ones:

**Hive:** data warehouse software built on top of Hadoop that facilitates reading, writing, and managing large datasets residing in distributed storage using SQL-like queries (HiveQL). Hive abstracts away underlying MapReduce jobs and returns HDFS in the form of tables (not HDFS).

**Pig:** high level scripting language (Fig Latin) that enables writing complex data transformations. It pulls unstructured/incomplete data from sources, cleans it, and places it in a database/data warehouses. Pig performs ETL into data warehouse while Hive queries from data warehouse to perform analysis (GCP: DataFlow).

**Spark:** framework for writing fast, distributed programs for data processing and analysis. Spark solves similar problems as Hadoop MapReduce but with a fast in-memory approach. It is an unified engine that supports SQL queries, streaming data, machine learning and graph processing. Can operate separately from Hadoop but integrates well with Hadoop. Data is processed using Resilient Distributed Datasets (RDDs), which are immutable, lazily evaluated, and tracks lineage.

**Hbase:** non-relational, NoSQL, column-oriented database management system that runs on top of HDFS. Well suited for sparse data sets (GCP: BigTable)

**Flink/Kafka:** stream processing framework. Batch streaming is for bounded, finite datasets, with periodic updates, and delayed processing. Stream data and stream processing must be decoupled via a message queue. Can group streaming data (windows) using tumbling (non-overlapping time), sliding (overlapping time), or session (session gap) windows.

**Beam:** programming model to define and execute data processing pipelines, including ETL, batch and stream (continuous) processing. After building the pipeline, it is executed by one of Beam's distributed processing back-ends (Apache Apex, Apache Flink, Apache Spark, and Google Cloud Dataflow). Modeled as a Directed Acyclic Graph (DAG).

**Oozie:** workflow scheduler system to manage Hadoop jobs

**Sqoop:** transferring framework to transfer large amounts of data into HDFS from relational databases (MySQL)

**SQL Part I**

Structured Query Language (SQL) is a declarative language used to access & manipulate data in databases. Usually the database is a Relational Database Management System (RDBMS), which stores data arranged in relational database tables. A table is arranged in columns and rows, where columns represent characteristics of stored data and rows represent actual data entries.

**Basic Queries**

- filter columns: **SELECT col1, col3... FROM table1**
- filter the rows: **WHERE col4 = 1 AND col5 = 2**
- aggregate the data: **GROUP BY...**
- limit aggregated data: **HAVING count(\*) > 1**
- order of the results: **ORDER BY col2**

Useful Keywords for **SELECT**

**DISTINCT:** return unique results

**BETWEEN**: a AND b- limit the range, the values can be numbers, text, or dates

**LIKE:** pattern search within the column text

**IN (a, b, c)** - check if the value is contained among given

**Data Modification**

- update specific data with the **WHERE** clause:
- **UPDATE table1 SET col1 = 1 WHERE col2 = 2**
- insert values manually

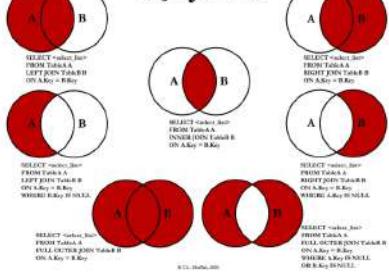
**INSERT INTO table1 (col1,col3) VALUES (val1,val3);**

- by using the results of a query

**INSERT INTO table1 (col1,col3) SELECT col1,col2 FROM table2;**

**Joins**

The JOIN clause is used to combine rows from two or more tables, based on a related column between them.

**SQL JOINS****Python- Data Structures**

Data structures are a way of storing and manipulating data and each data structure has its own strengths and weaknesses. Combined with algorithms, data structures allow us to efficiently solve problems. It is important to know the main types of data structures that you will need to efficiently solve problems.

**Lists:** or arrays, ordered sequences of objects, mutable

```
>>> l = [42, 3.14, "hello", "world"]
```

**Tuples:** like lists, but immutable

```
>>> t = (42, 3.14, "hello", "world")
```

**Dictionaries:** hash tables, key-value pairs, unsorted

```
>>> d = {"life": 42, "pi": 3.14}
```

**Sets:** mutable, unordered sequence of unique elements. frozensets are just immutable sets

```
>>> s = set([42, 3.14, "hello", "world"])
```

**Collections Module**

**deque:** double-ended queue, generalization of stacks and queues; supports append, appendLeft, pop, rotate, etc

```
>>> s = deque([42, 3.14, "hello", "world"])
```

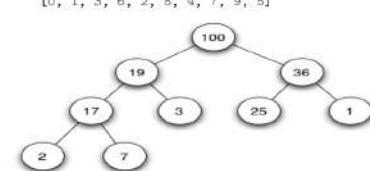
**Counter:** dict subclass, unordered collection where elements are stored as keys and counts stored as values

```
>>> c = Counter('apple')
>>> print(c)
Counter({'p': 2, 'a': 1, 'l': 1, 'e': 1})
```

**heapq Module**

**Heap Queue:** priority queue, heaps are binary trees for which every parent node has a value greater than or equal to any of its children (max-heap), order is important; supports push, pop, pushpop, heappop, replace functionality

```
>>> heap = []
>>> for n in data:
...     heappush(heap, n)
...
>>> heap
[0, 1, 3, 6, 2, 8, 4, 7, 9, 5]
```

**Recommended Resources**

- Data Science Design Manual ([www.springer.com/us/book/9783319554433](http://www.springer.com/us/book/9783319554433))
- Introduction to Statistical Learning ([www-bcf.usc.edu/~gareth/ISL/](http://www-bcf.usc.edu/~gareth/ISL/))
- Probability Cheatsheet ([www.wzchen.com/probability-cheatsheet/](http://www.wzchen.com/probability-cheatsheet/))
- Google's Machine Learning Crash Course ([developers.google.com/machine-learning/crash-course/](https://developers.google.com/machine-learning/crash-course/))

# Contributors:

---

## Favio Vázquez

---

### Releases

No releases published

---

### Packages

No packages published

---

### Contributors 10

