Hands-on Machine Learning with sklearn, numpy, matplotlib, pandas, keras and TensorFlow 2

Table of contents

[1 Prerequisites 12](#_Toc85998297)

[1.1 Python 12](#_Toc85998298)

[1.1.1 Data Structures 12](#_Toc85998299)

[1.1.2 Flow controls 13](#_Toc85998300)

[1.1.3 Functions and lambdas 14](#_Toc85998301)

[1.1.4 Error handling 14](#_Toc85998302)

[1.1.5 String operations 14](#_Toc85998303)

[1.1.6 IO 14](#_Toc85998304)

[1.1.7 Classes 15](#_Toc85998305)

[1.1.8 Pickle – serialization framework – 16](#_Toc85998306)

[1.1.9 Time 16](#_Toc85998307)

[1.1.10 Regex 16](#_Toc85998308)

[1.1.11 Generator, yield, async-await and coroutines 17](#_Toc85998309)

[1.1.12 Threading 17](#_Toc85998310)

[1.1.13 Python file in .exe format / pyinstaller 17](#_Toc85998311)

[1.2 Numpy 17](#_Toc85998312)

[1.2.1 Arrays (vectors and matrices) 17](#_Toc85998313)

[1.2.2 Shaping 18](#_Toc85998314)

[1.2.3 Stacking 18](#_Toc85998315)

[1.2.4 Array manipulation 19](#_Toc85998316)

[1.2.5 Linalg 19](#_Toc85998317)

[1.2.6 Assert 20](#_Toc85998318)

[1.2.7 Ufuncs 20](#_Toc85998319)

[1.2.8 Miscellaneous (random, linspace, math) 20](#_Toc85998320)

[1.3 Pandas 20](#_Toc85998321)

[1.3.1 Series 20](#_Toc85998322)

[1.3.2 DataFrame 21](#_Toc85998323)

[1.4 Matplotlib 23](#_Toc85998324)

[1.4.1 Lines and 2D functions 23](#_Toc85998325)

[1.4.2 Multiple plots on a figure 24](#_Toc85998326)

[1.4.3 Scatter plots, histograms and images 24](#_Toc85998327)

[1.4.4 Texts and Legends 25](#_Toc85998328)

[1.4.5 Ticks 25](#_Toc85998329)

[1.4.6 ggg 25](#_Toc85998330)

[1.5 Missingno 25](#_Toc85998331)

[1.6 Jupyter notebook 26](#_Toc85998332)

[1.6.1 Shortcuts 27](#_Toc85998333)

[1.6.2 Markdown 27](#_Toc85998334)

[1.7 Databricks notebook 27](#_Toc85998335)

[1.8 R 28](#_Toc85998336)

[1.8.1 Attributes, Basic types, NA and NaN 28](#_Toc85998337)

[1.8.2 Complex types 29](#_Toc85998338)

[1.8.3 Subsetting, indexing 31](#_Toc85998339)

[1.8.4 Date & time 32](#_Toc85998340)

[1.8.5 Reading data, Files 32](#_Toc85998341)

[1.8.6 Control structures 33](#_Toc85998342)

[1.8.7 Functions 34](#_Toc85998343)

[1.8.8 Loop functions (apply-s) 34](#_Toc85998344)

[1.8.9 Statistics 35](#_Toc85998345)

[1.8.10 Misc 35](#_Toc85998346)

[1.9 Database theory 36](#_Toc85998347)

[1.9.1 IMQAV 36](#_Toc85998348)

[1.9.2 Anomalies 37](#_Toc85998349)

[1.9.3 Normal forms 37](#_Toc85998350)

[1.9.4 Data modeling 39](#_Toc85998351)

[1.10 SQL (Microsoft Server SQL) 40](#_Toc85998352)

[1.10.1 Data types 40](#_Toc85998353)

[1.10.2 Create and alter schemes 40](#_Toc85998354)

[1.10.3 Insert rows into table 41](#_Toc85998355)

[1.10.4 Delete rows from table 41](#_Toc85998356)

[1.10.5 Queries 41](#_Toc85998357)

[1.10.6 Joins 42](#_Toc85998358)

[1.10.7 Union, Intersect, Minus(Except) 43](#_Toc85998359)

[1.10.8 Aggregations 44](#_Toc85998360)

[1.10.9 Any/All 45](#_Toc85998361)

[1.10.10 Exists 45](#_Toc85998362)

[1.10.11 Output formatting 45](#_Toc85998363)

[1.10.12 ... 45](#_Toc85998364)

[1.10.13 ... 45](#_Toc85998365)

[1.11 NoSQL 46](#_Toc85998366)

[1.11.1 MongoDB (Document databases) 46](#_Toc85998367)

[1.11.2 Hbase (Column databases) 50](#_Toc85998368)

[1.11.3 Cassandra 52](#_Toc85998369)

[1.11.4 Key-value databases 57](#_Toc85998370)

[1.11.5 Graph databases 57](#_Toc85998371)

[1.12 Elasticsearch 57](#_Toc85998372)

[1.12.1 Use-cases 58](#_Toc85998373)

[1.12.2 Logstash 58](#_Toc85998374)

[1.12.3 Kibana 58](#_Toc85998375)

[1.12.4 X-Pack 58](#_Toc85998376)

[1.12.5 Beats 58](#_Toc85998377)

[1.12.6 Node and cluster 58](#_Toc85998378)

[1.12.7 Document, index, mapping 59](#_Toc85998379)

[1.12.8 Shard 60](#_Toc85998380)

[1.12.9 Replication 60](#_Toc85998381)

[1.12.10 CRUD in ElasticSearch 61](#_Toc85998382)

[1.12.11 Queries 62](#_Toc85998383)

[1.12.12 Analysis & analyzers 63](#_Toc85998384)

[1.12.13 Scoring & boosting 64](#_Toc85998385)

[1.12.14 Cluster health API, Cat API 64](#_Toc85998386)

[1.12.15 Bulk API 65](#_Toc85998387)

[1.12.16 Document versioning 65](#_Toc85998388)

[1.12.17 Combining queries (must, filter, should, must\_not) 65](#_Toc85998389)

[1.13 Linux shell 65](#_Toc85998390)

[1.13.1 General command syntax 65](#_Toc85998391)

[1.13.2 Files, folders, permissions 66](#_Toc85998392)

[1.13.3 Commands for common tasks 67](#_Toc85998393)

[1.13.4 Vim & nano 67](#_Toc85998394)

[1.13.5 Tar (tape archive), zip 68](#_Toc85998395)

[1.13.6 ... 68](#_Toc85998396)

[1.14 Docker 68](#_Toc85998397)

[1.14.1 Container manipulation 69](#_Toc85998398)

[1.14.2 Creating images (Dockerfile) 70](#_Toc85998399)

[1.14.3 Docker volume 71](#_Toc85998400)

[1.14.4 Docker-compose (docker-compose.yml) 71](#_Toc85998401)

[1.14.5 Kubernetes 72](#_Toc85998402)

[1.15 LaTeX 72](#_Toc85998403)

[1.15.1 Symbols 72](#_Toc85998404)

[1.15.2 Formatting 74](#_Toc85998405)

[2 Scikit-Learn and book abstract 75](#_Toc85998406)

[2.1 Types of Machine Learning systems 75](#_Toc85998407)

[2.1.1 Supervised / Unsupervised / Semisupervised / Reinforcement 75](#_Toc85998408)

[2.1.2 Batch / Online 75](#_Toc85998409)

[2.1.3 Instance-based / Model-based 75](#_Toc85998410)

[2.2 Challenges of Machine Learning 75](#_Toc85998411)

[2.3 Testing and Validating 76](#_Toc85998412)

[2.4 Bias and variance 76](#_Toc85998413)

[2.5 Scaling 77](#_Toc85998414)

[2.6 Encoding 77](#_Toc85998415)

[2.6.1 Label encoding 77](#_Toc85998416)

[2.6.2 One hot encoding 77](#_Toc85998417)

[2.6.3 Ordinal encoding 77](#_Toc85998418)

[2.7 Transformation pipelines 78](#_Toc85998419)

[2.8 Hyperparameter tuning 79](#_Toc85998420)

[2.8.1 Grid search 79](#_Toc85998421)

[2.8.2 Randomized search 79](#_Toc85998422)

[2.9 Datasets 80](#_Toc85998423)

[2.10 Test-train split and cross validation 80](#_Toc85998424)

[2.11 Clustering 80](#_Toc85998425)

[2.11.1 Agglomerative Clustering 80](#_Toc85998426)

[2.11.2 K-Means Clustering 81](#_Toc85998427)

[2.11.3 DBSCAN (Density-Based) Clustering 83](#_Toc85998428)

[3 Classification 85](#_Toc85998429)

[3.1 Logistic Regression 85](#_Toc85998430)

[3.1.1 Softmax / Multinomial logistic regression 86](#_Toc85998431)

[3.2 Naive Bayes 87](#_Toc85998432)

[3.2.1 Laplace smoothing 88](#_Toc85998433)

[3.3 Support Vector Machine 89](#_Toc85998434)

[3.3.1 LinearSVC 89](#_Toc85998435)

[3.3.2 SVC for nonlinear data 89](#_Toc85998436)

[3.3.3 Laplace smoothing 90](#_Toc85998437)

[3.4 Decision Tree 91](#_Toc85998438)

[3.5 Random Forest (ensemble method) 91](#_Toc85998439)

[3.6 Gradient Boosting (XGBoost, ensemble method) 91](#_Toc85998440)

[3.7 Confusion Matrix and ROC Curve 91](#_Toc85998441)

[3.7.1 Rand index (=accuracy in binary case) 93](#_Toc85998442)

[3.7.2 Jaccard index 93](#_Toc85998443)

[3.7.3 Precision 93](#_Toc85998444)

[3.7.4 Recall 94](#_Toc85998445)

[3.7.5 F1 Score and Precision-Recall tradeoff 94](#_Toc85998446)

[3.8 Multiclass Classification 96](#_Toc85998447)

[3.8.1 One versus All (One versus Rest) 96](#_Toc85998448)

[3.8.2 One versus One 96](#_Toc85998449)

[3.9 Egyéb 96](#_Toc85998450)

[4 Regression 97](#_Toc85998451)

[4.1 Linear Regression 97](#_Toc85998452)

[4.1.1 Ridge regression (L2 regularization) 98](#_Toc85998453)

[4.1.2 Lasso regression (L1 regularization) 98](#_Toc85998454)

[4.1.3 Elasticnet 99](#_Toc85998455)

[4.2 Locally Weighted Linear Regression 99](#_Toc85998456)

[4.3 Polynomial Regression 100](#_Toc85998457)

[4.4 Support Vector Machine 101](#_Toc85998458)

[4.5 Support Vector Machine 101](#_Toc85998459)

[4.6 Support Vector Machine 101](#_Toc85998460)

[4.7 Support Vector Machine 101](#_Toc85998461)

[5 Statistics 102](#_Toc85998462)

[5.1 Permutation, Variation, Combination 102](#_Toc85998463)

[5.2 Probability theory 102](#_Toc85998464)

[5.3 CDF, PDF, PMF 103](#_Toc85998465)

[5.4 Expected value, Variance, Standard deviation 103](#_Toc85998466)

[5.5 E(X) and V(X) of random length sum of iid variables 105](#_Toc85998467)

[5.6 Moments, Skewness, Kurtosis 106](#_Toc85998468)

[5.7 Median, Mode, Quantile (percentile) 106](#_Toc85998469)

[5.8 Multidimension distributions, Covariance, Correlation 106](#_Toc85998470)

[5.9 Popular discrete distributions 109](#_Toc85998471)

[5.9.1 Indicator function (aka characteristic function, Bernoulli distr.) 109](#_Toc85998472)

[5.9.2 Binomial distribution 109](#_Toc85998473)

[5.9.3 Poisson distribution 110](#_Toc85998474)

[5.9.4 Geometric distribution 110](#_Toc85998475)

[5.9.5 Hypergeometric distribution 110](#_Toc85998476)

[5.10 Popular continous distributions 111](#_Toc85998477)

[5.10.1 Uniform distribution 111](#_Toc85998478)

[5.10.2 Exponential distribution 111](#_Toc85998479)

[5.10.3 Normal distribution 112](#_Toc85998480)

[5.10.4 distribution (Chi-square distribution) 113](#_Toc85998481)

[5.10.5 Student distribution (t distribution) 113](#_Toc85998482)

[5.10.6 Fisher distribution (F distribution) 113](#_Toc85998483)

[5.11 Limit theorems 113](#_Toc85998484)

[5.11.1 Law of large numbers 113](#_Toc85998485)

[5.11.2 de Moivre-Laplace 114](#_Toc85998486)

[5.11.3 Central limit (CLT) 114](#_Toc85998487)

[5.11.4 Hoeffding’s inequality 115](#_Toc85998488)

[5.11.5 Delta method 115](#_Toc85998489)

[5.12 Fundamental concepts of statistics 115](#_Toc85998490)

[5.13 Parameter estimation 117](#_Toc85998491)

[5.13.1 Point estimation 118](#_Toc85998492)

[5.13.2 Interval estimation 120](#_Toc85998493)

[5.13.3 Distance between distributions 122](#_Toc85998494)

[5.14 Hypothesis testing 124](#_Toc85998495)

[5.14.1 Z-test (z-próba, u-próba) 126](#_Toc85998496)

[5.14.2 Student’s t-test 128](#_Toc85998497)

[5.14.3 F-test (Fisher-test) 129](#_Toc85998498)

[5.14.4 Welch’s t-test 130](#_Toc85998499)

[5.14.5 Non-parametric tests introduction, common problems 130](#_Toc85998500)

[5.14.6 Kolmogorov-Smirnov test 131](#_Toc85998501)

[5.14.7 Wilcoxon signed-rank test 131](#_Toc85998502)

[5.14.8 Mann-Whitney U test 133](#_Toc85998503)

[5.14.9 Chi-square test 134](#_Toc85998504)

[5.14.10 ANOVA (Analysis of variance) 136](#_Toc85998505)

[5.14.11 Bartlett-test 137](#_Toc85998506)

[5.14.12 Friedman-test 138](#_Toc85998507)

[5.14.13 Kruskal-Wallis test 139](#_Toc85998508)

[5.14.14 Exact tests introduction 139](#_Toc85998509)

[5.14.15 Fisher’s exact test 140](#_Toc85998510)

[5.14.16 Binomial test 140](#_Toc85998511)

[5.15 Regression Analysis 140](#_Toc85998512)

[5.15.1 Multivariable linear regression 145](#_Toc85998513)

[5.15.2 Model building 147](#_Toc85998514)

[5.15.3 Outlier detection 149](#_Toc85998515)

[5.15.4 Binary logistic regression 149](#_Toc85998516)

[5.16 Data reduction 149](#_Toc85998517)

[5.16.1 Factor analysis 150](#_Toc85998518)

[5.16.2 PCA (principal component analysis) 151](#_Toc85998519)

[5.16.3 MDS (multi dimensional scaling) 151](#_Toc85998520)

[5.16.4 Discrimininant analysis 152](#_Toc85998521)

[5.17 Time series data 152](#_Toc85998522)

[5.17.1 Models for time series 153](#_Toc85998523)

[6 Azure 157](#_Toc85998524)

[6.1 Cloud computing in general 157](#_Toc85998525)

[6.2 IaaS, PaaS, SaaS 157](#_Toc85998526)

[6.3 Cloud models 157](#_Toc85998527)

[6.4 Structured and unstructured data 157](#_Toc85998528)

[6.5 Azure data platform technologies 158](#_Toc85998529)

[6.5.1 Azure Storage 158](#_Toc85998530)

[6.5.2 Azure Data Lake Storage Gen2 158](#_Toc85998531)

[6.5.3 Azure Cosmos DB 158](#_Toc85998532)

[6.5.4 Azure SQL Database 159](#_Toc85998533)

[6.5.5 Azure Synapse Analytics 159](#_Toc85998534)

[6.5.6 Azure Stream Analytics 159](#_Toc85998535)

[6.5.7 Azure HDInsight 159](#_Toc85998536)

[6.5.8 Databricks, Data Factory, Data Catalog 160](#_Toc85998537)

[6.6 Azure for data engineers (ETL, ELT) 160](#_Toc85998538)

[7 Big Data 161](#_Toc85998539)

[7.1 Data engineering workflow 161](#_Toc85998540)

[7.2 Data stores 162](#_Toc85998541)

[7.3 Hadoop 162](#_Toc85998542)

[7.4 Spark 163](#_Toc85998543)

[7.4.1 Spark, MapReduce comparison 163](#_Toc85998544)

[7.4.2 SparkShell & SparkApplication 163](#_Toc85998545)

[7.4.3 Job scheduling 164](#_Toc85998546)

[7.5 RDD, DataFrame, DataSet 170](#_Toc85998547)

[7.6 Spark Streaming 170](#_Toc85998548)

[7.6.1 Data streaming in general 170](#_Toc85998549)

[7.6.2 Spark Streaming 171](#_Toc85998550)

[7.6.3 Structure Streaming 177](#_Toc85998551)

[7.6.4 Databricks Streaming 177](#_Toc85998552)

[7.7 Apache Kafka 180](#_Toc85998553)

[7.7.1 Optimizazion for throughput 185](#_Toc85998554)

[7.7.2 Optimizazion for latency 185](#_Toc85998555)

[7.7.3 Optimizazion for durability 185](#_Toc85998556)

[7.7.4 Optimizazion for availability 185](#_Toc85998557)

[7.7.5 Kafka monitoring & metrics 185](#_Toc85998558)

[7.7.6 Kafka Connect 185](#_Toc85998559)

[7.7.7 Kafka Streams 188](#_Toc85998560)

[7.8 Flow-Based programming 193](#_Toc85998561)

[7.9 NiFi 194](#_Toc85998562)

[7.9.1 MiNiFi 195](#_Toc85998563)

[7.10 StreamSets data collector 195](#_Toc85998564)

[7.11 Orchestration and scheduling 196](#_Toc85998565)

[7.11.1 Apache Airflow 196](#_Toc85998566)

[7.11.2 Jenkins 198](#_Toc85998567)

[8 Neural Networks 200](#_Toc85998568)

[8.1 Forward and Backpropagation 200](#_Toc85998569)

[8.1.1 Math 200](#_Toc85998570)

[8.1.2 Forward propagation on XOR N.N. 200](#_Toc85998571)

[8.1.3 Backpropagation on XOR N.N. 201](#_Toc85998572)

[8.2 Egyéb 202](#_Toc85998573)

[9 NLP (Natural language processing) 203](#_Toc85998574)

[9.1 Stopwords & punctuation 203](#_Toc85998575)

[9.2 Stemming & Lemmatizing 203](#_Toc85998576)

[9.3 Vectorize 203](#_Toc85998577)

[9.3.1 Count vectorization 203](#_Toc85998578)

[9.3.2 N-gram vectorization 204](#_Toc85998579)

[9.3.3 Inverse document frequency weighting (TF-IDF) 204](#_Toc85998580)

[9.4 Egyéb 205](#_Toc85998581)

[9.5 Egyéb 205](#_Toc85998582)

[10 PyQt5 206](#_Toc85998583)

[10.1 Connect 206](#_Toc85998584)

[10.2 MessageBox 206](#_Toc85998585)

[11 Web scraping 207](#_Toc85998586)

[11.1 BeautifulSoup 207](#_Toc85998587)

[11.2 Selenium 207](#_Toc85998588)

[12 DevOps 209](#_Toc85998589)

[12.1 Practices 209](#_Toc85998590)

[13 Math auxiliary 210](#_Toc85998591)

[13.1 Matrix properties 210](#_Toc85998592)

[13.1.1 Matrix properties 210](#_Toc85998593)

[13.2 Eigenvalues and eigenvectors 210](#_Toc85998594)

[13.3 Rank of matrix 211](#_Toc85998595)

[13.4 Kernel and image 211](#_Toc85998596)

[13.5 Norm 211](#_Toc85998597)

[13.6 Function properties 212](#_Toc85998598)

[13.6.1 Jensen’s inequality 212](#_Toc85998599)

[13.6.2 Taylor’s expansion 212](#_Toc85998600)

[13.6.3 Integration by part (partial integral) 213](#_Toc85998601)

[13.7 Inequalities 213](#_Toc85998602)

[13.7.1 Triangle inequality 213](#_Toc85998603)

[13.7.2 Cauchy-Bunyakovskiy-Schwarz inequality 213](#_Toc85998604)

[13.8 Numerical optimalization 214](#_Toc85998605)

[13.8.1 Lagrange multiplier 214](#_Toc85998606)

[13.9 Graph theory 214](#_Toc85998607)

[13.9.1 DAG and topological ordering 214](#_Toc85998608)

[14 Standard normal CDF () chart. 215](#_Toc85998609)

[15 Student-t CDF chart 216](#_Toc85998610)

[16 CDF charts in R 217](#_Toc85998611)

[17 218](#_Toc85998612)

# Prerequisites

The book assumes basic programming knowledge as well familiarity with the most often used scientific libraries such as numpy, pandas and matplotlib.

## Python

Modern programming language, where classes passed as *reference copy* and primitives passed by value.

*Reference copy:* Assigning new objects to this won’t affect the original object’s state. Moreover, modifying the object is only feasible by calling operations (add,pop etc.) on this copy.

### Data Structures

#### List

lst = [1, 2, 3]

lst.append(4) #4 to the end

lst.insert(0, 0) #0 to the beginning

lst.remove(obj)

lst.pop(index) #remove element

lst.copy()

lst.extend(lst\_other)

primes = list(filter(lambda num: num.is\_prime, numbers))

#method called list comprehension:

primes = [num.value **for** num **in** numbers **if** num.is\_prime]

lst[-1] gives you the last element

[\*range(a,b+1,s)] every s-th number starting from a ending with b

text = [**'Python'**, **'is'**, **'a'**, **'fun'**, **'programming'**, **'language'**]

# join elements of text with space

print(' '.join(text))

# Output: Python is a fun programming language

#### Tuple

Similar to list but unchangeable during runtime.

tpl = (1, 2, 3)

zip() function for pairing lists/tuples into a single tuple

#### Set (unordered)

st = {1, 2, 3}

st.add(4)

st.update(list)

st.remove(obj) #will raise Error if not found

st.discard(obj) #no Error thrown

st3 = st.union(st2)

#### Dictionary

dct = { ”a” : 1, ”b”: 2, ”c” : 3 }

dct[”d”] = 4

dct.pop(”c”)

#no error thrown when key doesnt exists, instead it gets #initialized with 0

**from** collections **import** defaultdict  
dct = defaultdict(int)

**for** key **in** dct:  
 **pass  
  
for** value **in** dct.values():  
 **pass  
  
for** (key, value) **in** dct.items():  
 **pass**

### Flow controls

num1 = 20 **if** someBoolValue **else** num1 (one liner ifelse)

There is no switch in python. In python 3.10 we have match-case.

### Functions and lambdas

**def** func(p1, p2, \*args, \*\*kwargs):  
 print(args[0], args[1], kwargs[**"kwparam"**])

#will print 3 4 cc  
  
**def** main():  
 func(1, 2, 3, 4, kwparam=**"cc"**)

lbd = **lambda** x : np.exp(x)

### Error handling

**try**:  
 **raise** NotImplementedError(**"test"**)  
**except** NotImplementedError **as** err:  
 print(err)  
**except**:  
 print(**"wont be printed"**)

### String operations

#### String interpolation

n = **"john"**a = 30  
str1 = **"name: {} age: {:0.2f}"**.format(n, a)  
str2 = **"name: {name} age: {age}"**.format(name=n, age=a)

#### ...

### IO

IO modes:

|  |  |
| --- | --- |
| r | reading  raises error if file’s missing |
| a | append  creates the missing file or writes to the end |
| w | writing  create the missing file or overwrites |
| x | create  error if file exists |

|  |  |
| --- | --- |
| t | text |
| b | binary |

file = open(path, mode) #”rt” ”xb” etc.

for line in file:

**pass**

file.close()

file.write(...)

RAII for IO (will automatically close, even on raised errors):

**with** open(**"path"**, **"mode"**) **as** f:  
 **pass**

### Classes

#### Defining a class and achieve inheritance

**class** BASE:  
 **def** \_\_init\_\_(self, x):  
 self.x = x  
 **def** GetX(self):  
 **return** self.x  
  
**class** DERIVED(BASE):  
 **def** \_\_init\_\_(self, x, y):  
 super().\_\_init\_\_(x) #BASE.\_\_initself.y = y  
 **def** GetY(self):  
 **return** self.y  
  
**def** main():  
 d = DERIVED(10, 20)  
 print(d.GetX())  
 print(d.GetY())  
  
main()

#### Static variables

**class** C:  
 static = **"ABC"  
  
 def** \_\_init\_\_(self):  
 **pass**

#### Class and Static methods

**class** C:  
 static = **"ABC"  
  
 def** \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 @classmethod  
 **def** John(cls):  
 **return** cls(**"John"**, 32)  
  
 @classmethod  
 **def** Eva(cls):  
 **return** cls(**"Eva"**, 22)  
  
 @staticmethod  
 **def** set\_static(val):  
 C.static = val  
  
**def** main():  
 c\_john = C.John()

**@classmethod** decorator is used for returning objects constructed by a set of predefined parameters. Also, it is the most pythonic way to achive multiple constructors as you can create only one \_\_init\_\_ definition.

**@staticmethod** decorator marks the function as a static one.

#### ...

### Pickle – serialization framework –

**import** pickle

**if** os.path.exists(**"ser.data"**):  
 obj = pickle.load(open(**"ser.data"**, **"rb"**))  
**else**:  
 obj = **None** *# some long processing task..* pickle.dump(obj, open(**"ser.data"**, **"wb"**))

### Time

**import** time

time.sleep(t:Float) # makes the thread wait for t seconds

time.time() -> Float # returns the elapsed seconds from 1970.01.01

### Regex

<https://docs.python.org/3/library/re.html>

**import** re

**TOKENIZING:**

re.split(**"\s"**, string) # split on single whitespaces

re.split(**"\s+"**, string) # split on whitespaces

re.split(**"\w+"**, string) # split on words

re.split(**"\W+"**, string) # split on anything but words

re.findall(**"\w+"**, string)# find every words

re.findall(**"\d"**, string)# find every digit

re.findall(**"\d+"**, string)# find every number

re.sub(**"[A-Z]+[0-9]+", "ISO9001"**, string)# replaces every occurence of a regex

### Generator, yield, async-await and coroutines

### Threading

### Python file in .exe format / pyinstaller

The goal here is to make our python file run on machines that either don’t have python installed or are lacking the neccessary packages.

The following command will make a file main.exe with all the code dependencies massaged into the executable. –w means that it will block console popping up.

pyinstaller --onefile -w main.py

The .exe often has to be relocated to be able to reach the non-code resources like pictures and sounds.

To compress the whole folder into a single installer you can use NSIS.

## Numpy

Math library with operations faster than standard library.

**import** numpy **as** np

### Arrays (vectors and matrices)

np.empty(5)

np.zeros((4,4)), np.ones

np.full((4,4), np.pi)

np.empty((0, 2), int) #ideal for loops

np.array([[...], [...], ..., [...]])

np.arange(lower\_bound\_inc, upper\_bound\_exc, step=1) #iota

np.linspace(lower\_bound\_inc, upper\_bound\_inc, split) #cont functs

np.random.rand(4) #4 long vector with rands between 0,1

np.random.randn(4) #4 long vector with N(0,1) distribution

np.fromfunction(**lambda** i, j: i\*j, (4, 4)) #ij can be 0

len(np\_arr) or np\_arr.size

np.insert(arr, where\_index, what, axis)

np.delete(arr, where\_index, axis)

Matrices will keep their 2D formats upon during operations on them and have the asterisk operator overriden by matrix multiplication.

mtx = np.matrix([[1, 0, 0],[0, 1, 0],[0, 0, 1]])

mtx.T #transpose

mtx.I #inverse

### Shaping

arr.shape = (4,4)

arr = np.reshape(arr, (16, 1)) #-1 calculates the other dim, returns a view not a copy!

x = np.atleast\_2d(x) #converts lower dimensions to 2D, useful for reshaping vector type np.arrays to matrix type (for dotprod)

arr.ravel() #to vector

arr.flatten() #copies, and converts copy to vector

**for** elem **in** arr.flat

print(elem) #processing each element in the flattened array

### Stacking

qv1, qv2: [[1 1 1]], [[2 2 2] [2 2 2]]

qv3 = np.vstack((qv1, qv2))

qv3 = [[1 1 1] [2 2 2] [2 2 2]]

qh1, qh2: [[1 1 1] [1 1 1]], [[2 2] [2 2]]

qh3 = np.hstack((qh1, qh2))

qh3 = [[1 1 1 2 2] [1 1 1 2 2]]

### Array manipulation

arr = np.array([[1, 2], [3, 4]])

np.roll(arr, 1) #4 gets shifted outside: [[4, 1], [2, 3]]

np.rot90(arr, k=1) #[[2,4], [1,3]] k=1: ccw (default)

### Linalg

**import** numpy.linalg **as** linalg

**from** numpy.linalg **import** solve

mtx.transpose()

mtx.T

mtx\_mult = mtx1.dot(mtx2)

inverse = linalg.inv(mtx)

pinverse = linalg.pinv(mtx)

identity\_mtx = np.eye(n)

det = linalg.det(mtx)

eigenvalues, eigenvectors = linalg.eig(mtx)

solve a system of equations:

solve(coefficient\_matrix, result\_array)

inner & outer products:

np.inner(a, b) #returns 4

np.outer(a, b) #returns [[2,1], [4,2]]

### Assert

np.testing.assert\_equal(a, b)

if np.equal(..., ...):

### Ufuncs

ufuncs are functions that operate on ndarrays in an element-by-element fashion.

np\_ufunc = np.frompyfunc(python\_func, #i, #o)

np\_ufunc(np.array([1,2,3]))

### Miscellaneous (random, linspace, math)

axis=n we eliminate the nth element in the shape attribute (will be the shape of the output) or we only modify the nth dimension (e.g.: concatenate)

np.linspace(min, max, num): splits the interval (min, max) into num equal parts

r1, r2, r3 = np.random.rand(3, 100): creates a 3x100 random matrix, ri are the ith row

np.random.normal(0, 1, 100): creates a 100 long vector with normally distributed vals

a[i, j] indexing a multidiminensional array

np.sum(), np.max(), np.argmax() and other unary operators

np.squeeze(X, axis=1): removes the axis if it is single dimensioned

np.append(X, y): append element y to array X, returns a new object

arr.flags.writeable = False: makes the ndarray read-only

## Pandas

Library for messing around with data.

**import** pandas **as** pd

### Series

Equivalent of Excel’s column.

|  |  |
| --- | --- |
| s\_indexed | |
| „a” | 30 |
| „b” | 27 |
| „c” | 22 |
| „d” | 41 |

#### Creating Series

|  |  |
| --- | --- |
| s\_base | |
| 0 | 2 |
| 1 | 3 |
| 2 | 4 |
| 3 | 5 |

predefined\_dic = {**"c"** : 22, **"d"** : 40}  
  
s\_base = pd.Series([2, 3, 4, 5])

s = np.exp(s\_base)  
s = s\_base + pd.Series([4, 3, 2, 1])  
s = s\_base + 1  
s = s\_base <= 3

s = pd.Series([30, 27, 22, 41], index=[**"a"**, **"b"**, **"c"**, **"d"**])  
s = pd.Series(predefined\_dic, index=[**"c"**]) #filtering by indices

s = pd.Series([22, 23], index=[**"alice"**, **"bob"**], name=**"ages"**)

s = pd.Series(np.nan, index=[**"def1"**, **"def2"**]) #def1: Nan, def2: NaN

#### Plotting Series

plt.scatter(s.index, s.values)   
#plt.plot would connect the points with linesplt.show()

### DataFrame

Equivalent of Excel’s spreadsheet.

#### Creating DataFrames

predefined\_dic = {  
 **"in letters"** : pd.Series([1, 2, 3]),  
 **"w numbers"** : pd.Series([**"ein"**, **"zwei"**, **"drei"**]),  
 **"rand"**: pd.Series(np.full(3, np.nan))  
 }  
df = pd.DataFrame(predefined\_dic) #if some columns couldn't be matched, those will become NaNsprint(df[[**"in letters"**, **"w numbers"**]])

#filtering  
df = pd.DataFrame(predefined\_dic, columns=[**"w numbers"**], index=[1, 2]) np\_datamatrix = np.array([  
 [**"alice"**, 33, **"mercedes"**],  
 [**"bob"**, 40, **"cadillac"**],  
 [**"chad"**, 25, **"bmw"**],  
])  
df = pd.DataFrame(np\_datamatrix, columns=[**"name"**, **"age"**, **"car"**], index=[**"a"**, **"b"**, **"c"**])

#### Accessing rows

|  |  |  |  |
| --- | --- | --- | --- |
|  | name | age | car |
| a (iloc 0) | alice | 33 | mercedes |
| b (iloc 1) | bob | 40 | cadillac |
| c (iloc 2) | chad | 25 | bmw |

#row of chaddf.loc[**"c"**]  
df.iloc[2]  
#row of alice and chaddf[np.array([**True**, **False**, **True**])]  
#row of alice and chaddf[pd.to\_numeric(df[**"age"**], errors=**"raise"**) <= 35]  
#inserting a new rowdf.loc[**"d"**] = np.array([**"dave"**, 17, np.nan])

df.drop(index=[**"d"**], inplace=**True**) #deleting the new row

#### Accessing columns

df = df.astype({**"age"** : **"int32"**})  
df[**"over 18"**] = [**False**] \* len(df.index)  
df[**"over 18"**] = pd.Series({**"a"** : **True**}) *#other will default to np.nan*df[**"over 18"**] = df[**"age"**] >= 18  
popped\_col = df.pop(**"over 18"**)  
df.drop(columns=[**"over 18"**])  
df.insert(0, **"ID card num"**, pd.Series({**"a"** : **"0x00"**, **"b"** : **"0xcf"**, **"c"** : **"0xfd"**}))  
evdf1 = df.eval(**"age\*\*2 + sin(age)"**)  
age\_limit = 21  
evdf2 = df.eval(**"age > @age\_limit"**)

#### Querying

queried\_df = df.query(**"age > @age\_limit and name.str.contains('a')"**)

#### Sorting

df.sort\_values(**"car"**, ascending=**True**, inplace=**True**)  
df.sort\_values([**"car"**, **"age"**], ascending=[**True**, **True**], inplace=**True**)  
df.sort\_index(ascending=**False**)

#### Handle missing data

df.fillna(**"no data"**, inplace=**True**)  
df.dropna(axis=0, how=**"any"**, inplace=**True**) #drop row if any of its value is np.nan

#### Transform rows

df[**"car\_new"**] = df[**"car"**].apply(**lambda** row: funct(row))

#### Pandas Utilities for reading structured data and misc

axis 0: rows

axis 1: columns

housing = pd.read\_csv(**"housing.dat"**, sep=**"..."**)  
housing[**"age\_category"**] = pd.cut(  
 housing[**"house age"**],  
 bins = (5, 20, 40, 80, np.inf),  
 labels = (**"new"**, **"renovated"**, **"mid"**, **"old"**, **"for demolition"**)  
)

#for printing a subset of the dataframe

df.head(num)

df.tail(num)

len(df) -> number of rows

len(df.columns) -> number of columns

## Matplotlib

Scientific python library for showing graphs.

**import** matplotlib.pyplot **as** plt

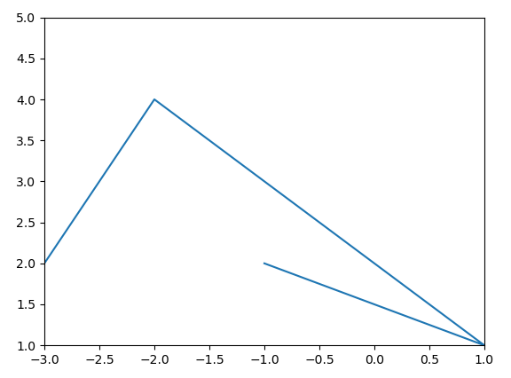
### Lines and 2D functions

plt.figure(**"figure\_title"**, figsize=(10, 12))

plt.axis(**"off"**)



plt.plot([2, 1, 2, 4, 1])  
plt.show()



plt.plot([-3,-2, 1, -1], [2, 4, 1, 2])  
plt.axis([-3, 1, 1, 5]) *# xe[-3, 1], ye[1, 5]*plt.show()



x = np.linspace(-1.4, 1.4, 30)  
plt.plot(x, x, **"g--"**, linewidth=3)  
plt.plot(x, x\*\*2, **"r:"**)  
plt.plot(x, x\*\*3, **"b^"**, alpha=0.1)  
plt.show()



x = np.linspace(-2, 2, 500)  
y = x\*\*2  
plt.plot(x, y)  
plt.title(**"x squared"**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"y"**)  
plt.grid(**True**)  
plt.show()

### Multiple plots on a figure



x = np.linspace(-1.4, 1.4, 30)  
plt.subplot(2,2,1)  
plt.plot(x,x)  
plt.subplot(2,2,2)  
plt.plot(x, x\*\*2)  
plt.subplot(2,1,2)  
plt.plot(x, x\*\*3)  
plt.show()

*subplot(2,2,2):*

*A table made up by two rows and two columns. We put the figure in the 2nd „cell”.*

### Scatter plots, histograms and images



mu, sigma = 100, 15  
x = mu + sigma \* np.random.randn(10000)  
plt.grid(**True**)  
plt.hist(x, bins=50, facecolor=**'green'**, edgecolor=**"black"**, alpha=0.75)  
plt.show()



x = np.linspace(-1.4, 1.4, 30)  
y = x\*\*2  
plt.scatter(x, y, s=200.0, c=**"red"**, alpha=0.9, edgecolors=**"green"**)  
plt.show()

plt.imshow(pic, cmap=**"binary"**)

plt.imshow(pic, cmap=plt.cm.binary)

plt.axis(**"off"**)  
plt.show() *# pic: NxM bitmap*

### Texts and Legends



x = np.linspace(-1.4, 1.4, 30)  
y = x\*\*2  
px = 0  
py = px\*\*2  
  
plt.plot(x, y)  
plt.plot(px, py, **"ro"**)  
plt.text(px, py + 0.05, **"minima"**, fontsize=10, color=**"red"**, horizontalalignment=**"center"**)  
plt.show()



plt.scatter([0.1, 0.15], [0.1, 0.08], color=**"red"**, marker=**"x"**, label=**"safe"**)  
plt.scatter([1, 0.89], [0.95, 0.99], color=**"green"**, marker=**"s"**, label=**"dangerous"**)  
plt.legend(loc=**"upper left"**)  
plt.show()

### Ticks

### ggg

## Missingno

Visualization library for missing values in pandas dataframe.

**import** matplotlib.pyplot **as** plt

**import** missingno **as** msno



#displays the dataframe, white lines represent missing values

msno.matrix(df)

plt.show()



#displays the # of missing

values per column

msno.bar(df)

plt.show()

#displays correlations

msno.heatmap(df)

plt.show()

If corr is close to 1 then a value present in the column will imply that a value is present in the other column.

If corr is 0 we cannot say anything, missing values appear randomly.

If corr is close to -1 then a value present in the column will imply that a value is missing in the other column.

## Jupyter notebook

*pip install notebook*

*jupyter notebook (localhost:8888)*



### Shortcuts

Create cell above: ESC + a

Create cell below: ESC + b

Delete cell: ESC + d + d / ESC + x

Make cell MD: ESC + m

Make cell code: ESC + y

Undo: CTRL + z

Multiselection: ESC + SHIFT + arrow

Merge: ESC + SHIFT + m

Execute cell: CTRL + ENTER

Execute cell and jump tp next: SHIFT + ENTER

Execute cell and make a new below: ALT + ENTER

### Markdown

# :<h1> header

## : <h2> header

[Daring Fireball](<https://daringfireball.net/projects/markdown/>) : link

- : unordered listing

1., 2.,... : ordered listing

> : block quote

\* ... \*: bold

\_ ... \_: italic

double space: carriage return (new line)

## Databricks notebook

https://accounts.cloud.databricks.com/workspaces

Edit mode: When a cursor is blinking in the textbox.

Command mode: When a cell is selected without a blinking cursor.

<Option> --- ALT

<CMD> --- CTRL/WIN

Table

Description automatically generated

To create a Markdown (comment) cell use the %md magin command.

Temporary views:

CREATE OR REPLACE TEMPORARY VIEW <QUERY\_NAME> AS <SELECT QUERY>

SELECT \* FROM <QUERY\_NAME>

## R

In R console we can include libraries (e.g.: library(datasets), library(**"BSDA"**)) and can install new ones:

install.packages("BoutrosLab.plotting.general")

### Attributes, Basic types, NA and NaN

atomic types: character, numeric, integer, complex, logical

x <- 1 #numeric

x <- 1L #integer

x <- 1/0 #numeric, Inf

x <- 0/0 #numeric, NaN

types gets printed on using class(object) and can be checked with is.numeric(obj)..

type casts: as.numeric(x), as.character(x)

operations with non-ordinary notations: %% (mod), %/% (integer division), ^

logical operations (and, or) are single digit: & |

check if two objects are identical: identical(x, y)

format string (3 to 003): sprintf("%03d", 3)

get size of an object: object.size(o)

NA: not available, missing value

NaN: not a number, counts as an NA as well

|  |  |  |
| --- | --- | --- |
|  | **NA** | **NaN** |
| is.na(x) | TRUE | TRUE |
| is.nan(x) | FALSE | TRUE |

### Complex types

vector (for storing elements with the same class):

x <- rep(NA, times=5) #NA NA NA NA NA

x <- rep(c(1, 2), each=2) #1 1 2 2

x <- c("numeric", length=5) #0 0 0 0 0

x <- c(1 + 0i, 2 + 3i) #explicitly telling the elements

x <- c(x, 3 + 2i) #appending to vector

x <- c(col1 = 1, col2 = 2) #named vector (names(object))

x <- 1:5 #int sequence: 1 2 3 4 5

x <- seq(0, 2, by=0.5) #0.0 0.5 1.0 1.5 2.0

x <- seq(0, 1, length=5) #0.0 0.25 0.5 0.75 1

x <- seq\_along(x) #1 2 3 ... length(x)

x <- rnorm(1000) #1000 normally distributed value

x <- sample(x, 100) #selects 100 elements randomly

length of vector: length(x)

list (for storing elements even with different types):

x <- list(1, TRUE, 1+2i)

x <- list(c1=1, c2=TRUE, c3=1+2i)

matrix (vector with set dimension attribute):

x <- matrix(nrow=2, ncol=3) # [NA NA NA]

# [NA NA NA]

x <- matrix(1:6, nrow=2, ncol=3) #column major filling

# [1 3 5]

# [2 4 6]

x <- 1:3 #1 2 3

y <- 6:8 #6 7 8 [1 6]

m1 <- cbind(x, y) #column-wise merge [2 7]

[3 8]

m2 <- rbind(x, yÖ #row-wise merge [1 2 3]

[6 7 8]

x <- 1:10 #redefining the dimension of a vector

dim(x) <- c(2, 5) [1 2 3 4 5]

[6 7 8 9 10]

inverse matrix: solve(m)

transpose matrix: t(m)

true matrix multiplication: m1 %\*% m2

factor (for categorizing data):

dirs <- c("left", "up", "up", "down", "up", "right", "left")

fdirs <- factor(dirs) # left up up down up right left

# Levels: down left right up

# levels are automatically ordered

fdirs2 <- factor(dirs, levels=c("up", "down", "left", "right")

# levels are manually ordered

Factor level generation, gl function (num\_leves, replications, seq):

f <- gl(3, 2, c("NY", "BP", "LA"))

# NY NY BP BP LA LA

# Levels: NY BP LA

On loading tabular data, columns of strings will be interpreted as factors.

data frame (handling tabular data with different types):

empty\_df <- data.frame(matrix(nrow=0, col=3))

colnames(empty\_df) <- c("col1", "col2", "col3")

df <- data.frame(col1 = 1:4, col2 = c(T,F,F,F))

add rows by calling: df <- rbind(df, list(col1 = 1, col2 = F))

add columns by calling: df <- cbind(df, col3 = c("a", "b"))

printing dataframe: head(df, n) tail(df, n)

dimensions of dataframe: nrow(df) ncol(df)

### Subsetting, indexing

[] : return an object of the same class, can be used to select multiple elements

[[]] : can only extract a single element from a list or dataframe, might return another type of object than is was called on

$: same as [[]], can helps us extract a single element by name

vector examples:

x <- c("a", "b", "c", "b", "d", "a")

x[1] #a, indexing starts at 1 in R!

x[1:3] #a b c

y[x %in% 1:3] #every element in y in [1, 3]

x[x < "c"] #a b b a

list examples:

x <- list(foo=1:4, bar=0.6)

x[1] #returns 1:4 sequence as LIST!

x[[1]] #returns 1:4 sequence as raw vector

x$foo #returns 1:4 sequence as raw vector

x[c(1,2)] #returns foo and bar as list

x[c("foo", "bar")] #returns foo and bar as list

x$f #partial matching, autocompleted to foo

matrix examples:

x <- matrix(rep(1, 6), 3, 2)

x[1,1] #returns 1 as numeric

x[1, ] #returns the first row as vector of numerics

x[1, , drop=FALSE] #returns the first row as a matrix

x[c(1, 2), ] #returns the 1st and 2nd row as a matrix

data frame examples:

df <- df[!is.na(df), c1] #returns every non-NA elem under c1

df <- df[complete.cases(df),] #returns NA-free rows

### Date & time

time:

t <- Sys.time() #class(t) POSIXct ("2021-03-08 22:43:50 CET") #unclass(t) integer (seconds since 1970)

t <- as.POSIXlt(t) #class(t) POSIXlt ("2021-03-08 22:43:50 CET")

#unclass(t) list with names: sec, min, ...

t <- strptime("October 17, 1999 08:00", "%B %d, %y %H:%M")

#will create a POSIXlt object

date:

d <- Sys.Date() #class(d) Date ("2021-03-08")

#unclass(d) integer (days since 1970)

d <- as.Date("1969-01-01) #unclass(d) integer (-365)

weekdays(d/t): "Monday"

months(d/t) "March"

quarters(d/t) "Q1"

### Reading data, Files

file.path(path1, path2) #will join path1 and path2

paste(str1, str2, ...) #concatenates strings

read.table ( filename, header=FALSE, sep="\tab", colClasses, nrows,

commentChar="#", skipBegin=0, stringAsFactors=TRUE)

read.csv is the same as read.table but sep is ",".

getwd() return the currect working directory.

source – dump:

y <- data.frame(a=1, b="str")

dput(y, "ser.R")

y <- dget("ser.R")

dget – dput:

Can handle multiple data.

x <- "foo"

y <- c(1, 2, 3)

dump(c(x, y), file="ser.R") #serialization

rm(x, y) #clears variable from enviroment

source("ser.R") #deserialization

### Control structures

if() {} else if {} else {}

y <- if (x < 3) { 10 } else { -10 }

for (i in 1:10)

for (elem in x)

for (i in seq\_along(x)) { next }

while(cnt < 10) { cnt <- cnt + 1}

repeat { ... break ... } #creates infinite loop

### Functions

Usually functions are written in a .R file which is then included to the current environment with sourcing (source("XYZ.R")). The last row of function is the return statement.

Complex examples:

f <- function(x=NA, fun, ...) {

args <- list(...)

y <- if (args[[using.fun]] == TRUE) { fun(args[[data]]) }

else { 0 }

x + y

}

f(c(1, 2), mean, using.func=TRUE, data=c(1, 1))

% bin % <- function(lhs, rhs) {

lhs %% rhs

}

x %bin% y

### Loop functions (apply-s)

lapply (x, fun, ...): Evaluates function on a list’s/vector’s each element.

Always returns a list.

sapply (x, fun, ...): Same as lapply but return the simplest possible class. By simplest class we mean either a single number or a vector.

vapply (x, fun, template, ...): Same as sapply, but checks whether the return type matches the template. If not it halts the program with an error. Template can be numeric(5) for example.

apply (x, margin, fun, ...): Allows us to call functions other than on arrays. The margin parameter’s counterpart in pandas is axis.

apply(m, 1, mean): mean of every row in matrix m (use rowMeans!)

mapply (fun, ..., more.args=NULL, simplitfy=TRUE, use.names=TRUE):

With mapply we can use a function with multiple sets of arguments.

mapply(rep, 1:3, 3:1) #list(rep(1, 3), rep(2, 2), rep(3, 1))

tapply (x, index, fun, ..., simplify=TRUE):

Useful when we need to break up the vector to groups defined by some classifying factor.

x <- c(rnorm(10), runif(10), rnorm(10, 1))

f <- gl(3, 10)

tapply(x, f, mean) #mean of rnorm, runif, rnorm1 in a list

split (x, f, drop=FALSE):

Will break up the vector to grups defined by the classifying factor f.

Will return a list of (# of levels in f) vectors. Can be used to simulate the group by statement.

table (x,f):

Maps the factor levels to their number of appearance in a tabular form.

### Statistics

empirical corrected variance: var(x)

### Misc

str: Compatly displays the internal structure of an object, alternative to summary

set.seed: important to set beforehand for reproducibility

sample(1:10, replace=FALSE): will make a permutation of 1:10

Random number generation: (with norm, gamma, poiss, etc. postfixes)

d: density (evaluates f(x), Probability DF)

r: random

p: cumulative distribution (returns F(x), CDF)

q: quantily function (returns F-1(x))

rbinom(100, size=1, prob = 0.7): gives the result of 100 runs of 1 coin toss each.

Profiling: system.type(expr)

elapsed time (”wall clock”): time you experience

user time: time charged to the CPU(s)

often: elapsed time = user time

if (elapsed time > user time): CPU waits for resources without running code

if (elapsed time < user time): In case of multiple processors

Rprof(), summaryRprof(): Prints out function call stack every 0.02s

#Rprof’s output will be just a bunch of function names

#summaryRprof will tabulate this output and will calculate how much

#time is spend in which function

by.total: Divides the spent time in each function by the total runtime.

by.self: Doest the same but subtracts the lower level function calls.

LETTERS: predefined vector of every english letter

plot(x, y, xlab, ylab, xlim, ylim, sub, main, pch ...)

(pch: point shape: triangle, filled circle, etc.)

## Database theory

### IMQAV

#### Ingestion

Is a set of software engineering techniques to make high volumes of data arrive rapidly. It is often achieved by using streaming.

#### Modeling

The process of creating data storage that is appropriate for that specific domain. (Relational database, Document, Key-Value, Graph based databases)

#### Query

The process consists of extracting data and modifying data (such as handling missing data)

#### Analyze

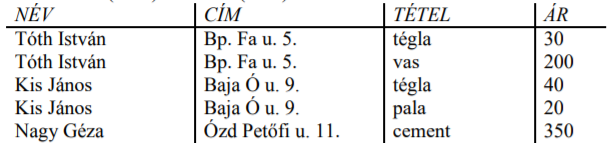
Multiple techniques to analize the data: computer science, stats, ML/AI.

#### Visualize

Transforming data into visually attractive and informative formats.

### Anomalies

Anomalies are unpleasant side effects we get when we have redundancy in our database.

In the database below (assuming each person has only one address) we have the address stored multiple times redundantly. (**redundancy**: the value of the attribute can be deduced from another attributes)

#### Update anomaly

Imagine a change in Kis János’ address. The first problem is that we have to update multiple rows and when doing that we might get a system failure leading to loss of information as we might get contradictory data (we manage to update X rows but fail to update Y)

#### Insertion anomaly

We cannot add a new person without specifying ’TÉTEL’. Also, adding a new row with an existing name but different address we lose information as from then on we have no idea which is the real address.

#### Deletion anomaly

Often an attribute is a part of a key. If we’d like to delete ’cement’, we’ll lose all the information of Nagy Géza.

### Normal forms

**functional dependency (X -> Y):** X determines the value of Y. (e.g.: country, city, district -> zip number).

**key:** we call X a key if it determines the whole row (X -> R), and there are no real subsets of X that does the same. If there is at least on real subset, then we call X a superkey.

**foreign key:** we call attribute A a foreign key, if it is a key in another table.

#### 1NF

A relational schema is 1NF if every attribute has atomic value. (there are no lists in cells) and each row is unique. Additionally we do not expect any sorting neiher in rows nor in attribues.

#### 2NF

We call A attribute a **primary** one if it is part of any K keys, otherwise a **secondary** one.

A relational schema is 2NF if all the secondary attributes fully depends on a key, so a part of a key is not enough to determine the value of a secondary attribute.

Imagine having a table of **T**(NAME, PROJECT, PROJECT\_START\_DATE). Even though PROJECT\_START\_DATE is not the part of the key it is clearly defined by the PROJECT which is the key’s real subset. For this reason **T** is only 1NF.

Every 1NF schema can be decomposed into 2NF schemas without loss of information. In this case we would make a **T1**(NAME, PROJECT\*) and **T2**(PROJECT, PROJECT\_START) schemas. PROJECT in **T1** is both a primary and a foreign key (this can be done for 1 to 1 relationships).

#### 3NF

**transitive dependency:** Let X be a subset of the schema R. A is an attribute in R. A is transitively dependent on X if there is a Y subset in R for

A relational schema is 3NF if none of the secondary (non-prime) attributes A depends transitively on any key, so we cannot find a proper Y for the definition to fulfill.

#### BCNF (3.5NF)

A relational schema is BCNF if none of the attributes A depends transitively on any key, so we cannot find a proper Y for the transitive dependency’s definition to fulfill.

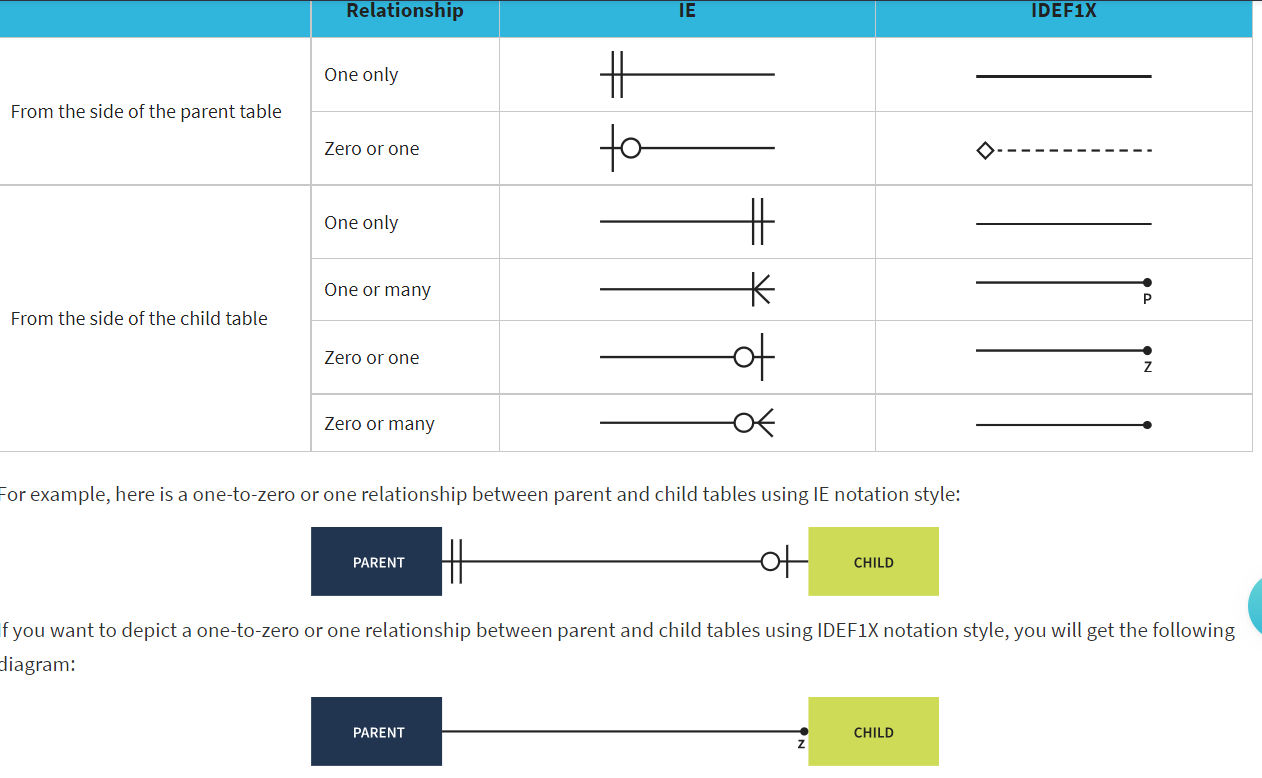
BCNF is redundancy-free. Every schema with at most two columns becomes BCNF inevitably.

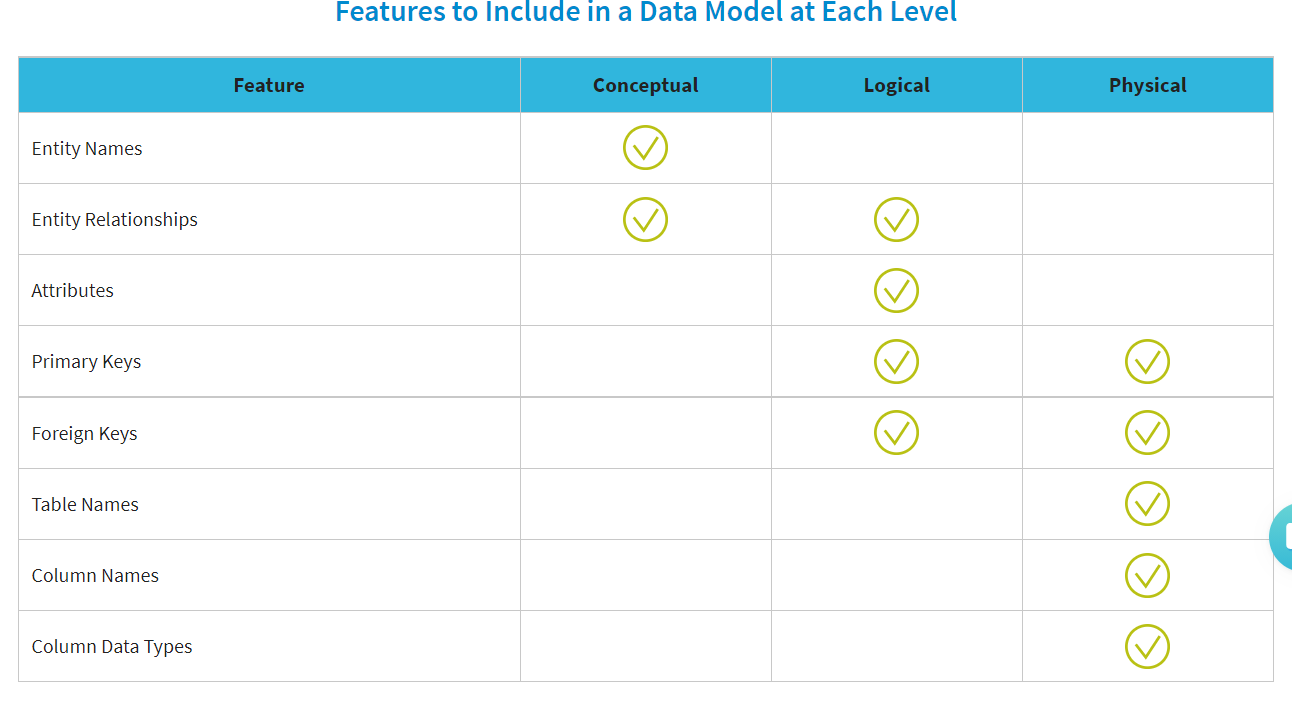
#### 4NF

#### 5NF

#### 6NF

### Data modeling





## SQL (Microsoft Server SQL)

SQL is a Data Query/Definition/Control/Manipulation language which allows us to issue Create, Read, Update, Delete (CRUD) commands to a relational database system.

### Data types

INT(size=255)

DEC(size=10, precision=0)

BOOL

VARCHAR(length)

BLOB # Binary Large OBject

DATA # YYYY-MM-DD

TIMESTAMP # YYYY-MM-DD HH:MM:SS

### Create and alter schemes

CREATE TABLE student (

id INT, -- IDENTITY(1,1) for autoincrement

-- teacher\_id INT,

name VARCHAR(20), -- NOT NULL, UNIQUE

major VARCHAR(20), -- DEFAULT ’unknown’

PRIMARY KEY(student\_id),

-- FOREIGN KEY (teacher\_id) REFERENCES teacher(id)

-- ON DELETE SET NULL

-- ON DELETE CASCADE

);

ALTER TABLE ADD gpa DEC(3,2);

ALTER TABLE DROP COLUMN gpa;

DROP TABLE student;

primary key: A minimal set of attributes that uniquely specifies a row.

foreign key: A primary key of another scheme.

### Insert rows into table

INSERT INTO student VALUES(1, ’Jack’, ’CS’);

INSERT INTO student(id, name) VALUES(2, ’Kate’);

INSERT INTO student(id) VALUES (3), (4), (5), ... ; --multiple rows

### Delete rows from table

DELETE FROM student WHERE major=’Unknown’

### Queries

SELECT DISTINCT \* FROM nobel\_win

WHERE SUBJECT NOT LIKE 'P%' –underscore(\_) for single char not %

ORDER BY YEAR DESC, WINNER ASC –first sort by year desc then name asc

FETCH FIRST 3 ROWS ONLY; –ORACLE

–LIMIT 3 MySQL

–SELECT TOP 3 MSSQL

SELECT *c.name*  
FROM sellers s  
INNER JOIN customers c –INNER/LEFT/RIGHT/FULLON s.sold\_to = c.customer\_id;

SELECT \*

FROM nobel\_win

WHERE subject NOT IN('Economics', 'Chemistry') --SINGLE QUOTE

SELECT \* FROM item\_mast

WHERE PRO\_PRICE BETWEEN 200 AND 600

SELECT

PRO\_NAME AS "ItemName", --NOTE DOUBLE QUOTES!!

PRO\_PRICE AS "ItemPrice"

FROM item\_mast

WHERE PRO\_PRICE >= 250

ORDER BY PRO\_PRICE DESC, PRO\_NAME ASC;

SELECT \* FROM testtable

WHERE col1 LIKE '%/\_%' ESCAPE '/' --Escape underscore

SELECT commission FROM salesman

WHERE salesman\_id IN (SELECT salesman\_id FROM customer WHERE city='London')

-- IN: subqueries with multiple rows of output

-- =/<>/>/<: subqueries with single output

### Joins

|  |  |
| --- | --- |
| **candidates** | |
| id | name |
| 1 | John Doe |
| 2 | Lily Bush |
| 3 | Peter Drucker |
| 4 | Jane Doe |

|  |  |
| --- | --- |
| **employees** | |
| id | name |
| 1 | John Doe |
| 2 | Jane Doe |
| 3 | Michael Scott |
| 4 | Jack Sparrow |

SELECT \* (AS..AS..AS) FROM

candidates c \_??\_ JOIN employees e

ON c.name = e.name

#### Inner (INNER JOIN / JOIN)

|  |  |  |  |
| --- | --- | --- | --- |
| **inner joined table** | | | |
| candidates.id | candidates.name | employees.id | employees.name |
| 1 | John Doe | 1 | John Doe |
| 4 | Jane Doe | 2 | Jane Doe |



#### Left

In case of left join we don’t filter the left table so on the right side NULLs may appear.

|  |  |  |  |
| --- | --- | --- | --- |
| **left joined table** | | | |
| candidates.id | candidates.name | employees.id | employees.name |
| 1 | John Doe | 1 | John Doe |
| 2 | Lily Bush | NULL | NULL |
| 3 | Peter Drucker | NULL | NULL |
| 4 | Jane Doe | 2 | Jane Doe |



#### Right

In case of right join we don’t filter the right table so on the left side NULLs may appear.

|  |  |  |  |
| --- | --- | --- | --- |
| **right joined table** | | | |
| candidates.id | candidates.name | employees.id | employees.name |
| 1 | John Doe | 1 | John Doe |
| 4 | Jane Doe | 2 | Jane Doe |
| NULL | NULL | 3 | Michael Scott |
| NULL | NULL | 4 | Jack Sparrow |



#### Outer (Full)

|  |  |  |  |
| --- | --- | --- | --- |
| **inner joined table** | | | |
| candidates.id | candidates.name | employees.id | employees.name |
| 1 | John Doe | 1 | John Doe |
| 2 | Lily Bush | NULL | NULL |
| 3 | Peter Drucker | NULL | NULL |
| 4 | Jane Doe | 2 | Jane Doe |
| NULL | NULL | 3 | Michael Scott |
| NULL | NULL | 4 | Jack Sparrow |



### Union, Intersect, Minus(Except)

#### Union

Can concatenate the result of two select statements if they have the same amount of columns. UNION ALL would allow duplicate elements (double length table in the example):

SELECT \* FROM employee

UNION

SELECT \* FROM employee

#### Intersect

Will return the intersection of two select statements that has the same number of columns:

SELECT \* FROM employee

INTERSECT

SELECT \* FROM employee

#### Minus/Except

MINUS and EXCEPT is essentially the same. The former is the syntax of Oracle and MySQL the latter is used in Microsoft SQL server. It will return the difference of two queries:

SELECT \* FROM employee

EXCEPT

SELECT TOP 2 \* FROM employee

### Aggregations

Aggregations are special functions that derive a value from a data subset.

**SUM, AVG, COUNT** (used with DISTINCT often), **MAX, MIN**

WHERE clause doesn’t work with values returned by aggregate function. Instead we use the HAVING clause.

SELECT edp.DPT\_NAME FROM

emp\_department edp INNER JOIN emp\_details edt

ON edp.DPT\_CODE = edt.EMP\_DEPT

GROUP BY edp.DPT\_NAME

HAVING COUNT(\*) > 2

The result of the GROUP BY statement has to be unambigous.

|  |  |  |
| --- | --- | --- |
| c1 | c2 | c3 |
| a | x | u |
| a | y | u |
| b | x | u |

|  |  |  |
| --- | --- | --- |
| c1 | c2 | c3 |
| a | x | u |
| a | y | u |
| b | x | u |
| b | x | u |

|  |  |  |
| --- | --- | --- |
| c1 | c2 | c3 |
| a | x/y ?? | u |
| b | x | u |

**original GROUP BY c1 GROUP BY c1, c2**

### Any/All

**Any:** Will return true of any row of a query fulfills a condition.

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ANY  
  (SELECT column\_name  FROM table\_name  WHERE condition);

**All:** Will return true if every row of a query fulfills a condition.

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ALL  
  (SELECT column\_name  FROM table\_name  WHERE condition);

### Exists

EXISTS(query) checks whether the return set is empty or not. Can be used with if-else (MSSQL) or with WHERE clause.

### Output formatting

SELECT CONCAT(probabilty\*100, '%') FROM client

**LEN(str)** lenght of a string column

**LEFT(str, n)** first n character of a string

**LOWER/UPPER(str)** converts strong to lower/uppercase

### ...

### ...

## NoSQL

Ideal for storing massive amount of data that has no fixed structure (rigid schema) and for dynamic data. NoSQL databases can be scaled horizontally (by increasing the data servers to reduce the load) and offers quick queries as no joining needed. On the other hand it requires more storage and is not ACID-compliant.

CAP (consistency, availability, partition tolearance) theorem helps to choose whether to use SQL or NoSQL. **A system can only support two of these three requirements.**

* Consistency: All clients will read the same on the same query.
* Availability: Ensures that all the clients can both read and write data.
* Partition tolerance: The database is partitioned to multiple machines so the system can carry on doing its work even during network outage.
* Examples: CA: RDBMs, AP: Cassandra, CP: MongoDB, HBase

### MongoDB (Document databases)

*Document databases store data as a key-value pair, where the value part is a document (XML, YAML, JSON, plaintext).*

Every MongoDB document requires a primary key. The programmer can provide his own id, but when not provided MongoDB’s ObjectId will be used. MongoDB stores its documents in BSON format which has a size limit of 16MB.

<https://cloud.mongodb.com/> for further instructions (whitelists, default db, creating collections, monitoring data CRUD)

**import** pymongo

cluster = pymongo.MongoClient(**"mongodb+srv://admin:admin@cluster0.erpp4.mongodb.net/db?retryWrites=true&w=majority"**)  
database = cluster[**"db"**]  
collection = database[**"c"**]

collection.insert\_one

collection.insert\_many

collection.delete\_many

collection.find

collection.update\_one{{**"\_id"**: 5}, {**"$set"**: {**"name"**:**"john doe"**}}

Find every document (cursor to the docs) with an existing **name** field:

results = collection.find({**"name":** {**"$exists"**: **False**}})

#### MongoDB Query

results = collection.find({**"age":** {$gt: 18}, {name: 1, address: 1} ).limit(5) // filter, projection, limit

results = collection.find( $and : [

{$or : [{price : 0.99}, {price : {$in: [1, 2, 3, 4, 5]}}]},

{$or : [{sale: true}, {quality : {$gte: 100}}]}

]

results = collection.find({**"name":** {**"$exists"**: **False**}})

results = collection.find({**"zipCode":** {**"$type"**: **"double"**}})

results = collection.find({**"carsArray":** {$elemMatch: {

{price: {$gte: 9999}}

}}})

#### MongoDB Validation

MongoDB provides the capability to perform

database.runCommand({

collMod: "c", // add validation to existing collection

validator: {

{**"name":** {**"$exists"**: **False**}}

}

validationLevel: "moderate" // strictness of validation rules

})

Database.createCollection("c2",

validator: {

{**"name":** {**"$exists"**: **True**}}

}

)

#### Embedded documents & document references

OneToOne relationship: (name and address)

{\_id: ”jjd”, name: ”John Jay Doe”}

{person\_id=”jjd”, zip:1111, ...}

{\_id:”jjd”, name:”John Jay Doe”, address:{

zip:1111, ... //embedded

}}

OneToMany relationship: (name and address1, address2)

{\_id: ”jjd”, name: ”John Jay Doe”}

{person\_id=”jjd”, zip:1111, ...}

{person\_id=”jjd”, zip:2222, ...}

{\_id:”jjd”, name:”John Jay Doe”,

//embedding

address1:{

zip:1111, ...

}

address2:{

zip:2222, ...

}

}

{\_id:”jjd”, name:”John Jay Doe”}

{\_id:”jd”, name:”Jane Doe”}

//data reference

{

tenants: [jd, jjd],

zip:1111, ...

}

{

Tenants:[jd],

zip:2222, ...

}

ManyToMany relationship: (name and address)

???

#### Indices

By default mongoDB keeps an index structure on the \_id field. Basic index structures include every document of a collection even if the indexed field is missing, in those cases the index field will be replaced by null.

You can add additional ascending indexes on fields (-1 descending):

database.records.createIndex( {score: 1} ) // single

database.records.createIndex( {score: 1, prevScore: 1} ) // compound

database.records.createIndex( {scoreArr: 1} ) // multikey

/\* note that you cannot use compound keys with indices on arrays \*/

database.records.createIndex( {scoreString: **"text"**} ) // string

database.records.createIndex( {\_id: **"hashed"**} ) // hash id

// supports single and compound key

database.addresses.createIndex( {storey: 1}, {sparse: true} )

// these will skip every document where the field storey does not

// exist, without that the storey field would be filled with null

database.restaurants.createIndex(

{name: 1},

{partialFilterExpression: {

rating: {$gt: 4.75}

}}

) // partial index

Partial indices only index documents where the field fulfills some condition that is present in the partialFilterExpression.

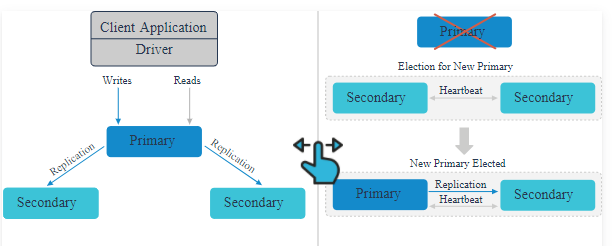
#### GridFS

To overcome the problem of BSON storing at most 16MB we can use GridFS which will divide documents into chunks. Do not use GridFS if it is neccessary to atomically update the file.

#### Replication

Replication provides redundancy (duplicate storing) and increases data availability. If a database server is lost, replication offers fault tolerance capabilites to out system. A replica set is a group of mongod instances (daemon process for mongoDB system) which contains the primary node and other nodes that are called secondary nodes. All the member of the replicate set can accept read operation but typically read operations are directed to the primary node. Also, primary node is the only node that can receive write operations.

Secondary nodes will replicate the primary’s operation log. In case of unavailability a secondary node will elect itself the new primary.



There are special members in the replica set:

* **Priority 0 member:** can accept read, but will not ever be a primary node, it can elect primaries though. Actually stores the replicated data.
* **Arbiter:** Does not store data, but takes part in the elections (ideal to resolve tie results).
* **Hidden member:** Hidden members are forced to be priority zeros. Ideal for special usage pattern.
* **Delayed member:** Delayed members are compulsorily hidden members, and they reflect an earlier state of the replica set. A delayed member can make it possible to recover from fault like dropperd databases or unsuccessful upgrades.

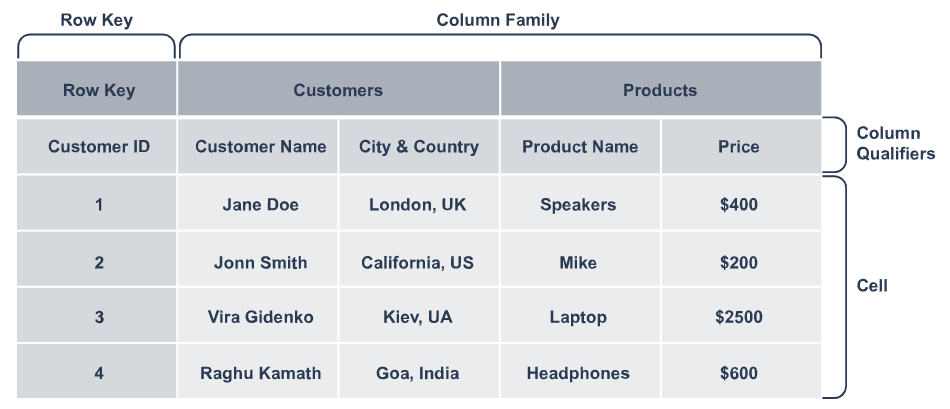
### Hbase (Column databases)

*In case of column databases the data is stored in a tabular format (rows and columns) but the columns are organized into families as some of the attributes are highly related and will be manipulated as a unit. Column families can be sparse! ([Phone number, Email, Mobile number] is the same as [Email, Mobile number].)*

*Ideally you would use column databases when you have a lot of data and more or less can predict the query pattern that will need to be executed.*

HBase is a column oriented distributed database in Hadoop using HDFS (implicit faul tolerance). A table here has multiple column families and each cell value has a timestamp.

A cell is identified by the row key and the column qualifier (column name).



#### HBase architecture

**region**: HBase tables are divided into regions in a manner that columns of the same column families are stored in one region. Region is by default 256MB. A group of regions is managed by the **Region Server** (responsible for read and write operations). Region server has a variety of components.

* **Write ahead log:** stores new data that is not persisted yet. The data then gets copied to the MemStore.
* **Block cache:** stores the frequently read data in memory
* **MemStore:** saves the newly incoming data before commiting to disk. There is one MemStore for each column family in a region.
* **HFile:** is the pack of actual cells. It is stored on HDFS. MemStore commits to HFile, when the size exceeds.

**HMaster:**  handles a collection of RegionServers (assigns regions and monitors) that resides on the same data node. Gives an interface for creating, deleting and updating tables.

**Zookeeper**: distributed coordination service among Data nodes. Region servers and the HMaster service sends heartbeats to Zookeper overtime, so it can check which server is available and could also provide server failure notifications. Zookeper also maintains an inactive server for rescuing.

**Meta table**: is a special catalog table that maintains a list of region servers. It does a mapping from row key to region server.

#### HBase bloom filter

A technique that can reduce the number of disk reads by predicting whether a given element is a member of a set of data. If it says no, then the set of data will not be read.

#### HBase compaction

HBase combines HFiles to reduce the number of storage and disks to read from.

Minor compaction will move small HFiles to bigger ones by commiting.

Major compaction will merge small HFiles of the same region together and drops the expired cells. It is a disk heavy tasks so it requires proper scheduling (low peak hours).

#### HBase operations & commands

PUT – SCAN (filter) – GET – DELETE

**status:** returns the status of the HBase

**version:** gives the version of the HBase being used

**whoami:** information of user

**create ’table’, ’column families’:** used to create tables

**disable ’table’:** disables table

**is\_disabled ’table’:** verifies whether a table is disabled

**list:** lists all the tables

**enable ’table’:** enables table

**is\_enabled ’table’:** verifies whether a table is enabled

**exit:** exits the shell

**alter ’table’, ’conditions’:** used to alter table

**exists ’table’:** verifies if the table exists or not

**describe ’table’:** prints information about the table

**drop ’table’:** deletes table (make it disabled in before)

**drop\_all <regex>:** drops all the tables that match the regex

**put ’table’, ’row(int)’, ’colfam:colname’, ’value’:** puts a cell value in a specified column and row in a given table.

**delete ’table’, ’row(int)’, ’colfam:colname’, ’ts’:** used to delete a cell value

**deleteall ’table’, ’row(int)’:** deletes every column in the row

**get ’table’, ’row(int)** **{COLUMN ⇒ 'personal:name'}’:** read the content of a row or a cell

**count ’table’:** returns the number of rows in a table

**scan ’table’:** returns the whole table data

**truncate ’table’:** recreates (disable, drop, create) a table

**grant ...:** can give specific rights to a user

**user\_permission ’table’:** gives total permission for a table

**revoke:** used to revoke user’s access of a table

### Cassandra

A column type database made by Facebook adopted by Apache. Good for heavy loads of writing operations as it has quick inter-node distribution. Cassandra is always on because its clusters are always available.

On the other hand Cassandra requires denormalization in order to make efficient quieries and aggregations as there is no support for the join operation. Ordering is only possible at creation time per partition basis. A column can be at most 2GB at the number of rows in a partition should not excess 100 000 items.

#### Cassandra storage mechanism

In cassandra data is stored as a set of rows organized into tables (=column families). Each row is identified by a primary key value. Data is partitioned on primary key.

**Keyspace:** a container for replication purposes that needs to be created before the table. It can contain many tables, but a table can only belong to a single keyspace. A replication level could be set when making the keyspace and also later on.

**Table (column family):** tables contain a set of columns and a primary key.

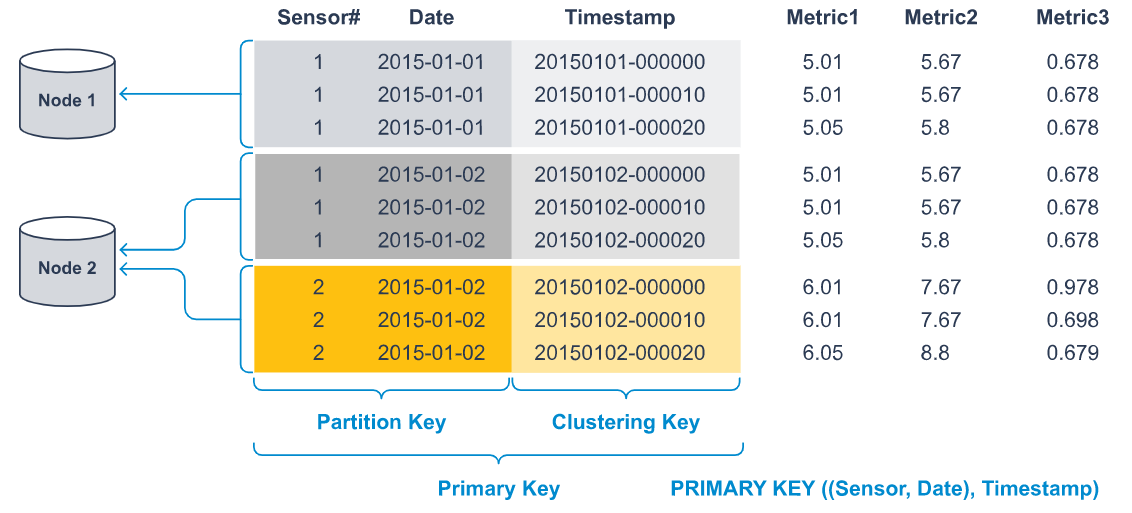
**Column:** Each column has to have an associated type which can be complex types like set, list and map. Column values have an associated timestamp as well.

--------------------------------------------------------------------------------------------------------

**Partition:** is a collection of rows that share the same partition keys. The rows are ordered by their cluster keys within the partition.

**Partition header:** contains the partition key and deletion information (tombstone)

**Clustering:** It represent a slice of rows.



Primary key = Partition Key + Clustering Key

# of cells in a partition = #rows · (#cols – #cols\_of\_pk - #static\_cols) + #static\_cols

#### Cassandra topology & storage

**Ring:** group of nodes serving the same dataset. Data is distributed around the nodes depending on their hashed partition key (aka **token**).

**Snitch:** Determines to which datacenter and rack nodes belongs to.

--------------------------------------------------------------------------------------------------------

**Memtable:** in memory structures where Cassandra buffers writes. There is one active memtable per table. Memtables are flushed onto the disk and become immutable SSTables. This can happen due to size in memory being excess, or the commitlog is full.

**SSTable:** used for persisting data on disk. Cassandra often compact small SSTable into a bigger one.

**Commitlog:** Written data will be first displayed on the commit log and then on the memtable. This can offer recovery capabilities. When the data from memlog is commited to SSTable, the commitlog entry can be archived or deleted.

#### Consistency levels

**Read:** specifies how many replicas must respond to read request. If an inconsistency is revealed then Cassandra initiates a read repair.

**Write:** specifies how many replicas must respond to the write request. The level specifies when we can consider a write successful.

The consistency level is defaulted to ONE (TWO, THREE).

* ALL: data needs to written to the memory log on every replica nodes. On reads every replica is checked and the record with the most recent timestamp is returned.
* EACH\_QUORUM: is only supported for writes. The write must be entried to the majority of the replica nodes’ commitlog and memtable.
* LOCAL\_ONE: every write has to be sent to at least one replica node in the local datacenter. In case of reads the item from the closest replica node is returned.
* LOCAL\_QUORUM: The write must be entried to the majority of the replica nodes’ commitlog and memtable in the same datacenter as the coordinator node. In case of readings a record is returned if the majority of replicas in the datacenter has reported.
* SERIAL: achieves consistency for lightweight writes implicitly. Allows reading the current state of data.
* LOCAL\_SERIAL: ???

Lower consistency improves throughput at the expense of data correctness. A strong consistency will fulfill the following formula:

*ReadC + WriteC > ReplicationFactor*

So if the replication factor is 3, the quorum (majority) will be 2, both in Read and Write. 2+2 > 3.

#### Repair

Data in replica can easily be inconsistent due to the distributed nature of the data. As a part of a regular maintenance every cluster is repairing its inconsistencies. Cassandra offers the following repair processes: hinted handoff, read repair, antientropy repair.

* Hinted handoff: if a node is unresponsive the missed write will be stored by the coordinator node. Hinted handoff has to be set in the cassandra.yaml. When the gossip mechanism finds out that the node is back online it will replay the hint file and then delete.
* Read repair: Data replicas in this case are repaired while doing read. On reading with consistency level TWO, two nodes are read at least, if they are inconsistent, then the records with the latest timestamps will be read and the inconsistent datasets read will be merged and written back.
* Antientropy repair: Is a process that updates all replicas to their newest version with a help of a Merkle tree.

**Repair:** parallel repair (repairs all data nodes with the same replica data at the same time, default behaviour), sequential repair, datacenter parallel repair (many datacenter is parellelly repaired with each node being sequentially fixed)

#### Data maintenance

Data is stored in SSTables. Inserts and updates will not overwrite existing data but rather insert a new timestamped version. The database does not really perform deletes on data removal instead it will flag the records with a tombstone mark. An item with the tombstona mark will be evicted from storage on the next compaction.

Compaction strategies:

* SizeTieredCompactionStrategy (STCS): will initiate compaction when the database accumulated (default 4) of similar sized SSTables.
* LeveledCompactionStrategy (LCS): Memtables are commited into Level 0 SSTables. Level 0 SSTables are merged into Level 1s and it goes on..
* TimeWindowCompactionStrategy (TWCS): SSTables are grouped according to some time window. Grouped SSTables are merged together by the end of the time window.
* DateTieredCompactionStrategy (DTCS): Similar to STCS but the compaction will be induced by the age and not the size. So SSTables with similar age will be compacted.

#### Nodetool

Nodetool is a Command Line interface for managing Cassandra noderings.

**status:** info about cluster

**repair:** repairs cluster

**describecluster:** provides name, snitch, ... of a cluster

**failuredetector:** failure detector information

**getendpoints:** ip addresses of the replica noded that own the partition key

**info:** information about the node

**cleanup:** cleans up keyspaces and partition keys no longer belonging to a node.

**garbagecollect:** removes deleted data from one or more tables.

**flush:** flusher one or more table from the memtable

**compact:** forces a major compaction on one or more table using STCS or DTCS

**compactionstats:** gives statistics about compaction

**compactionhistory:** history of compactions

**snapshot:** backs up data

**clearsnapshot:** removes one or more snapshots

#### Data modelling

Modelling is quite different than in case of a relational database as there are no joins and the usage of foreign keys are not enforced. Also, it is preferred to keep the data denormalized for faster writes and reads. Data modelling in cassandra is based around queries.

Every entities that are related to each other should be in the same table as most of the queries will target all of them. In Cassandra you have the standard primitive types, maps, lists, sets, and UDT (user defined types) as possibilities to set to columns.

**Compound** key: Primary key is made up from one column making the Partition key and at least one other column. (Person: [country] – [id\_card\_number])

**Composite** key: Primary key is made up from at least two columns that comprises the Partition key and at least one additional column. With this we can distribute the database depending on more than one field.

Regarding indexing, do not use indexes on high cardinality (lot of unique elements) columns, on counter columns and on columns that are update constantly.

Batches are atomic transactions (a set of inserts for instance) that can happen on a single partition and on multiple partitions. The latter is not recommended due to performance issues unless strict atomicity must be ensured.

### Key-value databases

*Are the simplest form of NoSQL databases, each item contains a key and a value. Ideal for use cases where large data needs to be stored without needing to apply complex queries on them. (e.g.: Redis for cache, Dynamo Amazon)*

### Graph databases

*Graph databases store data in nodes and edges which can have attributes too. Edges can be directed and represent some kind of relationship. Ideal for traversing and find fraud detections and recommendations. (e.g: Neo4J)*

## Elasticsearch

A tool for storing, searching and analyzing big data in nearly realtime. Elasticsearch has built in parallelization and operates on several nodes.

Shard is a partition of data that runs on a node. Replica shard is the copy of the primary shard that runs on a node different from the a primary shard. There is no single point of failure. A defected node can be replaced anytime. Fault tolerance is ensured by stroing replicas of the data and the indices withing the same cluster.

On the other hand, ElasticSearch’s requests and responses will be JSON exclusively. We don’t have a freedom of selecting the format. Also, it requires heavy orginizing and massive amount of available RAM (64GB).

### Use-cases

Logging & log analysis in real time (finding trends and security flaws, etc.), implementing complex search engines, lack of strict schema enables combining data from various data sources. Also good for autotagging and face recognititon.

### Logstash

Logstash offers connection to elasticstack. Logstash can filter data. Logstash has 3 stages when we talk about handling a data event. **inputs – filters – outputs**

### Kibana

Kibana is an analytics and visualization tool that can be used as a UI for managing ElasticStack (assign permissions, take snapshots, etc).

### X-Pack

An extension that provides security, monitoring, reporting and machine learning. Implicitly installed with elasticsearch.

### Beats

It helps the system to ingest data in real-time through data shippers.

### Node and cluster

When launching Elasticsearch we start up a node. A node belongs to a cluster. A node is essentially an instance of Elasticseatch that stores data. A node refers to an elasticsearch instance and not a machine. On a node startup it will either join to an existing cluster or make it own. The nodes in the same cluster know about each other.

Cluster is a collection of nodes that have the same **cluster.name** property. Clusters are independent from each each other, and a cross cluster searching is an antipattern. The practice is that we logically group together nodes to clusters.

Node roles (node.roles: [data, ingest, ...]):

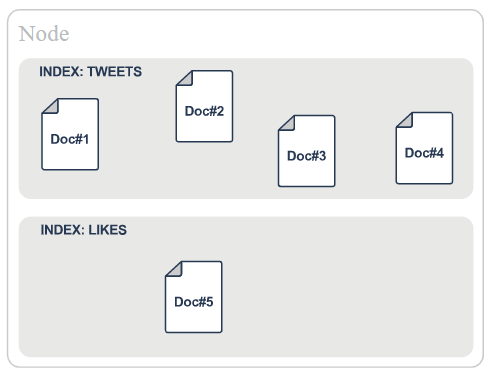
* **master-eligible:** a master node can create an index and tracks the other nodes in the cluster and decide which shards to allocate to which nodes. Master-eligible nodes become master nodes by winning the elections on the master election process. Master-eligible nodes should not care about resource-intensive tasks in order to lead the cluster well.
* **data:** hold shards that contains the document you have indexed. Data nodes does CRUD, search and aggregations. Data nodes ha IO and CPU heavy tasks and thus monitoring needed to decide whether we need to add any additional nodes.
* **ingest:** can apply a transformation pipeline to a document before indexing
* **machine learning:** enables machine learning features on the node, needs X-Pack
* **remote:** allows the node to act like a cross-cluster client
* **transform:** run transforms and handle transform API requests
* **coordinating:** no roles added to the node so it works as a load balancer (because it can only handler request routing and searchreduce?)

### Document, index, mapping

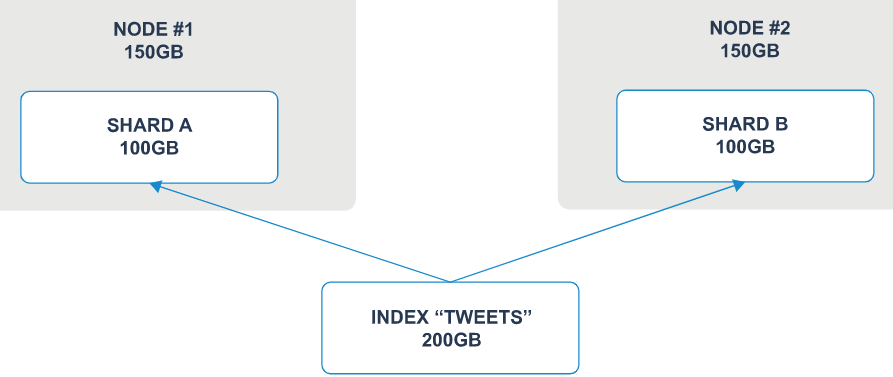
**Document:** Document is a set of key-value pairs in JSON. When a document is stored it is indexed and searchable in nearly realtime. Elasticsearch can be schemaless, that means that the indexing can be done without caring about different fields.

**Index:** Optimized collection of documents. Maps to one ore more primary shards.

**Mapping:** Is the process of defining how a document and the fields inside are indexed and stored. (which fields are geolocations and the format of data values, etc.). Due to dynamic mapping, new fields will be automatically indexed.



### Shard

Sharding is a way to divide an index into separate shards. A shard can store at most two billion documents. By default an index contains a single shard that can be increased with the SplitAPI and reduce with the ShrinkAPI.

### Replication

What happends when a node breaks down? Elasticseatch natively support shard replication and it is actually enabled by default. A shard that is replicated is called a primary shard. Replication can only happen in clusters with multiple nodes.

### CRUD in ElasticSearch

A document is a JSON serialized object that has multiple metadata fields assoicted with it.

**\_index:** doc belongs to this index

**\_type (pre ES7.0):** doc’s type, not used anymore

**\_id:** doc’s id

**\_source:** the JSON representing the doc’s body

**\_size:** the size of the \_source field in Bs.

**\_field\_names:** all fields in the doc that contain non-null values

**\_ignored:** every ignored field (due to malformation)

**\_routing:** maps document to a specific shard

Before making and querying documents one has to create an index first. (Index API, Delete API).

Create document:

POST (PUT) /courses/\_doc/(optional doc ID) // create or update

POST (PUT) /courses/\_create/(optional doc ID) // only create

POST ...

{

”name”: ”learn A S D now!”,

”publish\_date”: ”2021-09.27”

}

// \_index: ”courses”, \_type: ”\_doc”, id: ”autogen or specified”

Read document:

GET (HEAD) /courses/\_doc/id // returns the full document

GET (HEAD) /courses/\_source/id // returns just the doc’s source field

GET: actually returns fields

HEAD: verifies if a document exists (empty response with 200OK)

Update document:

POST /courses/\_update/id // gets the doc, applies script, reindexes the result

// update API allows upserting, so missing id-s that

// we want to update will be inserted instead

POST ...

{

”name”: ”Learn ASD now!”,

”description”: ”A course for technology enthusiasts.”,

}

// will correct the name field, and add an extra field description

POST ...

{

”script”: {},

”upsert”: {}

}

// unless the document exists upsert will be executed

Delete document:

DELETE /courses(index)/\_doc/id

### Queries

In ElasticSearch text fields are tokenized (see NLP/nltk) for analysis, so it cant be matched with string literals.

**Term level queries**: queries that retrieve data on precise values in the structured data.

GET /course/\_search

{

”query”:

{

”range”: {...},

”term”: {...}

}

}

**Full text queries**: allows you to analyze text fields such as an email

Aggregations

POST / courses / \_search

{

"aggregations/aggs":

{

"avg\_rating":

{

"avg": {"field": "rating"}

}

}

} // returns the avg of ratings

POST / courses / \_search

{

"aggregations/aggs":

{

"stats": {"field": "rating"}

}

} // returns the min,max,sum,count and avg of the field ”rating”.

Similar way we can retrieve the percentiles and the cardinalities of fields.

### Analysis & analyzers

Types of search:

* **Structured search**: examining data that has fixed structure (date, number). A document will be either be in the answer or not, there is no scoring.
* **Multifield search**: multiple query clauses are combined and their relevance scores has to be compared.
* **Fulltext search**: search withing fulltext fields finding the most relevant docs depending on some search condition which has to be matched.

Full-text search:

Fulltext search requires inverted indexing. Inverted indexing maps terms to documents. Upon searching each document gets a relevance score depending on the term being searched. The documents with the highest relevance score will be shown to the user.

When a new document is added an analysis process is getting run before actually storing the document on hard disk. The analysis process make text easier to search and can be customized. The analysis process in case of texts removes the punctuation marks, the unknown an seldom used characters and then split the text to tokens, while making text lowercase.

INPUT >> CHAR FILTER >> TOKENIZER >> TOKEN FILTER >> OUTPUT

**Char filter**:

html\_strip: removes html tags, and replaces html entities with their decoded counterpart (&amp -> &).

mapping: replaces the key with the value in a text ( 😊 -> happy )

**Tokenizer**:

word oriented tokenizer: usually used to split text into individual words

partial word tokenizer: breaks up texts or words into fragments

structured text tokenizer: used for structured texts (ZIP codes, ID-s, addresses)

”tokenizer”: ”standard”

”tokenizer”: ”uax url email” can handle urls

”tokenizer”: ”ngram” makes tokens by applying a 1 and 2 long sliding window

”tokenizer”: ”pattern” splits text by \W+ (non word strings)

**Token filters**:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html>

----------------------------------------------------------------------------------------------------------

Analyzers are combinations of a tokenizer and filters (char filter + token filter).

### Scoring & boosting

Scoring works by applying the TF/IDF method. Boost increases the score of a specific field.

Boosting at index time: boost needs to be specified in the mapping

Boosting at query time: boost is implemented at the query post

### Cluster health API, Cat API

Cluster status is set by the worst index status. Cluster status can be red, yellow and green.

**Red:** indicates that some of the data is missing, but the cluster can still operate. The given shard is not allocated.

**Yellow**: indicates that all the data is available and the primary shard is allocated but replicas aren’t.

**Green**: indicates that the cluster is fully functional and all the shards are allocated

GET \_cluster/health

----------------------------------------------------------------------------------------------------------

The CAT (compact and aligned text) API provides statistics about your cluster in a user-friendy/human-readable interface. The query string for the statistics has the following format: GET /\_cat/master?help <params>

### Bulk API

Let’s the user perform multiple types of request simultaneously (multiple indexing, create, update, delete)

POST /\_bulk

{ ... }

{ ”create”: {”\_index”: ”a”, ”id”: ”2”} }

{ ... }

### Document versioning

**External versioning**: allows you to specify a \_version parameter for the document which can be used as a version indicator in other systems (RDBMS).

**Internal versioning**: can be done with the \_version parameter or with the \_seq\_no and \_primary\_term params. (for every operations there is an assigned sequence number and a primary term).

Versioning helps to avoid the problem of updating a document with an older version and thus supporting concurrency.

### Combining queries (must, filter, should, must\_not)

**must:** appears in matching documents and take scoring into consideration, an AND

**filter:** same as must but scoring is not taken into consideration

**should:** can influence the score, can be considered a logical OR when matching

**must\_not:** the close must not appear in the documents, scoring is ignored all will be zero

## Linux shell

### General command syntax

Command + Option(s) + Argument(s) e.g.: ls -lh /usr/bin

Options with single dash can be concatenated if the option itself is shortened into a single letter, while the ones with double dash have to be separated by a whitespace.



### Files, folders, permissions

**file** path: determines file type

**stat** path: displays extended information about the file/directory

**pwd**: print working directory

**ls (-R, -a)**: lists every file and folder inside the wd (with R to infinite width), -a will even show hidden files.

**. .. ~ /**: current directory / parent directory / user’s home directory / root dir

**mkdir (-p)** path: creates a directory (with p it will create the needed parent dirs as well)

**rmdir** path: removes an empty directory

**cp** cpsrc, cpdest: copies cpsrc to cpdest (cpdest can be a directory)

**mv** mvsrc, mvdest: moves mvsrc to mvdest (mvdest can be a directory) it is the suggested way of renaming files

**\* ?**: regex for cp and move: **\*** can be any number of chars while **?** can refer to one

**rm (-r)** path: removes file (with r flag it can delete total directories)

**find** path-name ”\*.txt”: find every file/directory recursively in path matching the regex in the double quotes

**sudo**: gives superuser permissions for a limited period of time (sudo -k is for exiting superuser mode)



Read: one can see the file’s content but cannot modify it (**4**)

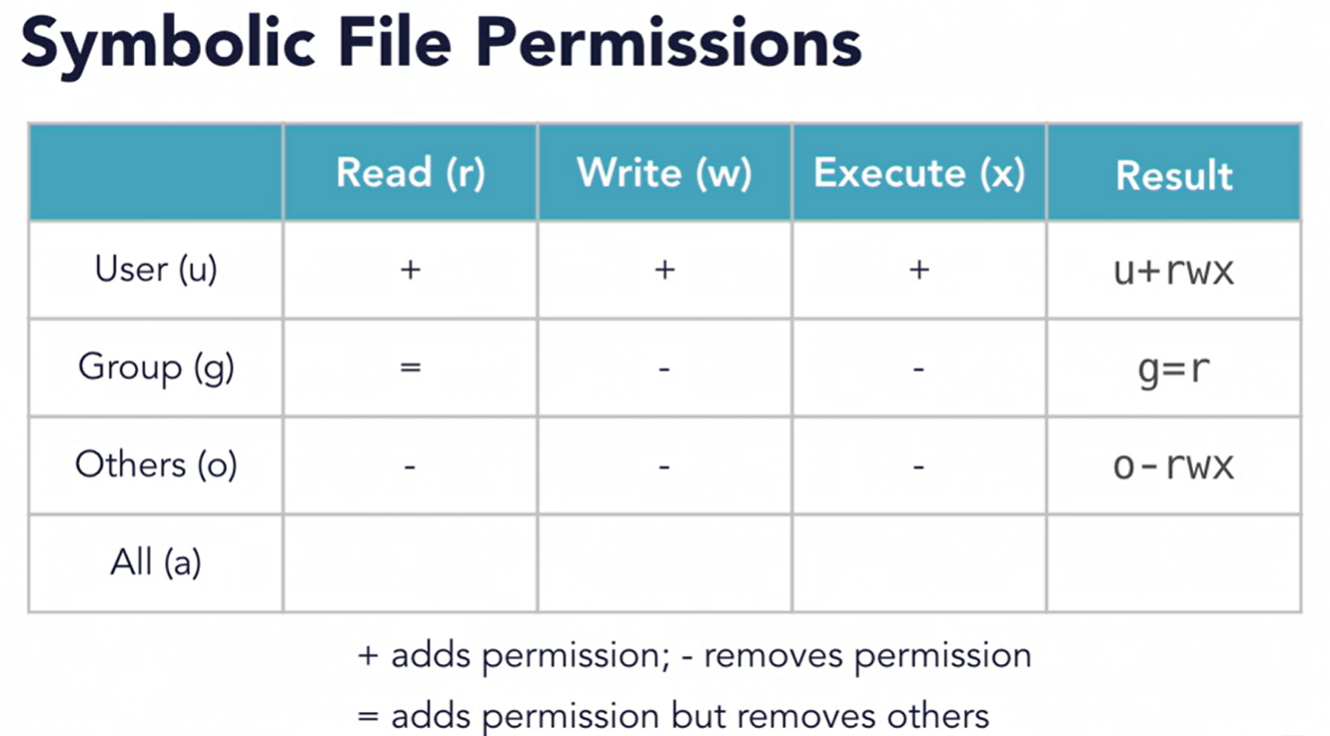
Write: one can make changes to the file but cannot see it’s contents (**2**)

Execute: one can run the file (.sh for example) (**1**)

**sudo chmod 777** file: makes ’file’ readable, writeable and executable for every groups







**ln (-s)** linked\_file link: makes a hardlink (soft/symbolic link with -s) to the file given in the last parameter. Softlink can lose reference upon moving the linked\_file or the link itself, while hardlink cannot as it points to the file’s data and not the file itself.

### Commands for common tasks

**wc (-l, -w, -c)** file: returns the number of lines/words/characters in the document

**cat, head, tail** file: displays the full text/first 10 lines/last 10 lines of the file

**less**: similar to cat but gives slightly more control

**grep (-n)** text file: finds (highlights) every occurence of the text/regex(-E) in the file. Grep is case sensitive unless we use -i. -n will display line numbers, -v will return those rows where there were no regex matching.

**sort (-n)** file: sort the lines of a file (-n: numerically, -r: reversed, -k2: depending on the 2nd column, -u: keep only unique lines)

**awk ’{program}’** file: modifying information from a file or stream. Helpful for retrieving data.

**sed ”program”** file: modifying information from a file or stream. Helpful for modifying data.

**rev** file: will print every line in a reversed order.

**tac** file: print lines reversed. **cat** reversed.

redirections:

cmd > file: stdout redirected to file (file gets emptied before)

cmd 2> file: stderr redirected to file (file gets emptied before)

cmd >> file: stdout redirected to file (append mode)

cmd 2>> file: stderr redirected to file (append mode)

### Vim & nano

Vim:

* Quit: **:q** (**:q!** Exit without saving)
* Insert mode: **i**
* Command mode (exit insert mode): ESC
* Save file: **:w** filename
* Save and quit **:wq**
* Open: **vim** filename

Nano (has a help toolbar for users):

* Find & select: CTRL W

### Tar (tape archive), zip

-**c**: create a tar

**-v**: verbose

**-f**: output a file and do not display

-**a**: use the proper archived method deduced from the extension of the output\_filename (e.g.: tar.gz, tar.bz2)

**-x**: extract

**-C**: defines the directory for extraction

Example archiving and unpacking:

**tar cvfa** output\_file.tar.gz file(s) #archiving

**tar xf** archive.tar.gz -C path\_to\_unpack #extracting

**zip -r** archive.zip file(s) #archiving

**unzip** archive.zip #extracting

### ...

## Docker

Docker is an ecosystem around creating and running containers. In a docker image we can have dependencies and other configs (FileSystem Snapshot) from which we can launch running containers.

Docker is a virtual machine with linux OS and by that it allows us to use Namespacing (define the resources that the container can use: Memory/Networking, ...) and Control Groups which helps limiting the usage of these resources. The docker image can contain a default (startup) command (e.g: run hello-world).

To list every available images on the computer use: docker images

An image become a container when we issue the command docker run.

A stopped container is converted into an image on using docker commit.

### Container manipulation

*Note: unique\_id can be replaced with container’s name.*

docker run <image name> // docker create + docker start

// (-ti terminal interactive)

// (--rm destroy after exiting)

// (-d detach: run in background)

// (--name <name>, sets a name for the container)

docker run <image name> <overriden default command>

// e.g.: docker run ubuntu bash -c ”sleep 5; echo asd”

docker create <image name> <odc> // return a unique id

docker start –a <unique\_id> // runs the image, -a: output to console

docker logs <unique\_id> // gets every log that was emitted from the run

docker attach <unique\_id> // connects to a detached container

// detach can happen by CTRL p + CTRL q

docker ps // lists every running container

docker ps –all // every container that used to run (exited can be restarted)

docker ps -l // latest exiting container

docker rm <unique\_id> // removes a stopped container completely

docker system prune // deletes every stopped containers and caches (-f force)

docker stop <unique\_id> // SIGTERM (give time for cleanup and save)

docker kill <unique\_id> // SIGKILL (instantenaous stop of the process)

// gets called automatically after 10s of docker stop

// use both kill and rm for removing a container

docker exec –it <unique\_id> <command> // input text, additional command

docker exec –it <unique\_id> sh // open shell in the context of container

docker run –it busybox sh // alternative

docker commit <unique\_id> // will make an image of a stopped container

// and will return an image id

docker tag <image\_id> <tag> // names the image so we can refer to by its

// name and not the crazy long image id

docker commit <unique\_id> <tag> // shorthand for the two

docker logs <unique\_id> // prints the logs of the container (optimal for post-// mortem debugging)

### Creating images (Dockerfile)

Put parts that change often to the end of the Dockerfile, as Docker can only use build caching until it meets with the first changed changed line. Dockerfile is evaluated line by line creating images each time. A process started on one line will not be running on the next. Environment variables can be used globally in the Dockerfile if used with the ENV command (e.g.: ENV DB\_PORT=8080, RUN ”echo ${DB\_PORT}”).

**FROM --- COPY --- RUN --- CMD**

FROM node:alpine (hub.docker.com/explore to see the popular ones)

*MAINTAINER fname lastname <email>*

WORKDIR /usr/app (against overwriting the root directory)

COPY ./ ./ (copy everything in the wd to the container’s filesystem, SPLIT!)

RUN npm install (will look for package.json to install)

CMD [”npm”, ”start”] (EXEC format instead shell form as it is preferred)

**RUN** will be executed during the build stage and **CMD** is executed upon running the container. ADD is similar to copy but allows using URLs (download links) and automatically does tarball extraction on the go. ENTRYPOINT is similar to cmd, while cmd defines the entire first command (with params) entrypoint defines only the start of it and the params can be passes as arguments when using docker run. Use EXPOSE for port mapping, Use VOLUME for defining shared volumes (do not use local paths ever). Use USER to change user for run commands.

Dockerfile // docker build . in its folder

docker build <unique\_id> // if we changed anything besides Dockerfile!

docker build –f Dockerfile.dev . // allows custom dockerfile name

docker build –t mkis98/test:latest . // this way we don’t need id-s, img tagging

docker run –p 8080:8080 <unique\_id> // every incoming request on localhost 8080 will be forwarded to the container’s 8080 port

docker run -p outside\_port:inside\_port/<tcp/udp> // inside\_port can be

// omitted for dynamic port allocation

### Docker volume

Virtual disks for storing and sharing data between containers. The volume can be **persistent** so that it keeps living after the host stops, or can be **ephemeral** which means it will evaporate on container stop. Volumes are not the part on an image.

docker run -ti -v <local/host full path>:/<path in the container> <image>

# docker run -ti -v ~Documents/LLD/share:/share ubuntu bash

docker run -ti -v <path in container> <image> #volume not shared with host

docker run -ti --volumes-from <container\_name which has a volume> <image>

# volume sharing between containers

### Docker-compose (docker-compose.yml)

Used for single machine container orchestration. Brings up all containers, volumes, networks, etc. in one command.

version: ’3’

services:

s1:

image: ’redis’

s2:

restart: always # on crash policy

build: . # Built from dockerfile, image defined there

ports:

- ”4001:8080” # array

docker-compose up (= docker run <image>)

docker-compose up –build (= docker build ., docker run <image>)

docker-compose up –d (open service in the background)

docker-compose down (stop services facilitating using ’docker stop’s)

On crash restart policies: **”no”**, **always**, **unless-stopped** (always restart except on forced close, **on-failure** (exit code is not 0)

docker-compose ps (will look for docker-compose.yml and finds the running containters defined there)

### Kubernetes

In Kubernetes containers run programs. Containers that intended to be run together are grouped into **pods**. Kubernetes allow to connect pods regardless of how things move around in the infrastructure. Can be used on-premise and in the cloud.

## LaTeX

Editor can be found here: <https://www.overleaf.com/>

useful packages to include:

\usepackage{amsmath} % advanced math symbols, spaces

\usepackage{physics} % for prettier vector symbols

### Symbols

Inline and complex formulas: $ e = 2.71... $ $$ complex formula $$

Greek alphabet: \alpha \gamma \Alpha \Gamma \pi

Partial derivative letter: \partial

Basic arithmetic: \cdot (mult dot) \dots (...) ^{x + 1} v\_{index} \vdots colv \cross

Fractions: \frac{a}{b} \xfrac{a}{b} (tilted line)

Sum and product: \sum\_{}^{} \prod\_{}^{} % bounds are omittable

Integral: \int\_{-\infty}^{\infty}{f(x) \text{ } dx}

Root: \sqrt[n]{...}

Limit: \lim\_{n \to \infty}

Autoscaling bracket: \left(1 + \frac{1}{n}\right)^n

Sets: \in \notin \subset \subseteq \cap (intersection) \cup \setminus

Vectors: \vb{A\_{n\cross n}} % physics library for bold vectors

Matrix:

$$

\begin{pmatrix/bmatrix}

1 & 2 & 3 \\

4 & 5 & a+1 \\

\end{pmatrix/bmatrix}^T

$$

\def\A{

\begin{bmatrix}

x\_1 & x\_2 & \cdots & x\_N

\end{bmatrix}}

\def\B{

\begin{bmatrix}

ax\_0 + bx\_1 \\

ax\_1 + bx\_2 \\

\vdots \\

x\_{N-1} + x\_N

\end{bmatrix}}

\def\C{

\begin{bmatrix}

z\_1 \\

z\_2 \\

\vdots \\

z\_N

\end{bmatrix}}

$$ y =\A \left(\B - \C\right) $$



### Formatting

**bold:** \textbf{...}

**underline:** \underline{}

**italic:** \textit{...}

**emphasize:** \emph{...} % effect depends on the context

\chapter{Intro} \section{...} \subsection{...} \section\*{no #}

\begin{itemize/enumerate}

\item ... % will insert bullett/number

\item ... % will insert bullett/number

\end{itemize/enumerate}

# Scikit-Learn and book abstract

## Types of Machine Learning systems

### Supervised / Unsupervised / Semisupervised / Reinforcement

Supervised: Training set includes the solutions (aka the labels) – Classification –

Unsupervised: Training set is unlabeled – Clustering & Anomaly Detection –

Semisupervised: Combined version of the previous types

Reinforcement learning: Reward based training of an agent

### Batch / Online

Batch: The system must be retrained from scratch with the full dataset on every new version

Online: The system can be trained incrementally by feeding it data instances either individually or in mini-batches

### Instance-based / Model-based

Instance-based: Similarity to the already learnt examples is measured on new cases

Model-based: Build a model from the learnt examples and make predictions according to the model.

## Challenges of Machine Learning

Insufficient quantity of training data

Nonrepresentative training data: training data must be representative of the new cases

Poor-quality: outliers, noise, errors, n/a-s

Redundant features

Overfitting training data (regularization helps)

Underfitting training data (model with more params helps)

## Testing and Validating

Testing how well the model will generalize to new cases is done by splitting the data set into a *training set* and a *test set*. We then train our model with the training set and test its error rate on never previously seen cases from the testing set. This error rate is called the *generalization error*. If we evaluate our model on the training set and get a much lower training error than the generalization error of the corresponding test set then we have the problem of overfitting, meaning that our model only performs well on the training data and the handling of new instances is quite flawed.

Often we hold out another set called the *validation set* to avoid refining (experimenting with regularization hyperparameters) a model to perform well solely on a particular training set. We train the model with the reduced training set (full traning set minus the validation set) and choose the model that produces the lowest generalization error on the validation set. After this process we retrain this model with the full training set and lastly evaluate the final model on the test set to get an estimate of the generalization error.

Cross-validation: The idea is to use multiple validation sets. Each model candidate (SVM, logistic regression etc) is evaluated once per validation set and trained on the rest of the training set. By averaging the evaulation results we get a realistic picture of the performance of out model at the cost of training time.

## Bias and variance

**bias:** Measures how much the model differs from the true labelling function.

**variance:** Measures how sensitive the model is to small deviations in the training data.

**Underfitting:** The model has high bias and low variance which means that it makes too general predictions (good instead of great, but can be worse) but at least it does it consistently. Solution: make the model more complex.

**Overfitting**: The model has low bias but high variance which means that it makes too specific predictions so it will generalize bad and will make extremely large errors on new instances. Solution: regularization, simplified model.

## Scaling

Machine learning models work better when features are on a similar scale (gradient descent can use larger learning rate, svm can draw a larger margin boundary). The most often used scaling method are MinMax scaling [0, 1] and Standard scaling N(0, 1).

**from** sklearn.preprocessing **import** MinMaxScaler, StandardScaler  
  
X = np.array([[1], [2], [3], [4]])

scaler = MinMaxScaler()  
X = scaler.fit\_transform(X)

## Encoding

### Label encoding

Converts a 1 dimensional vector of strings/numbers into categorical values (ints). Ideal to transform **y**-s that are in string format and **has ranking**. (bad/average/good)

**from** sklearn.preprocessing **import** LabelEncoder

labels = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**])

label\_encoder = LabelEncoder()

cat\_labels = label\_encoder.fit\_transform(labels)

print(label\_encoder.inverse\_transform(cat\_labels)) #prints labels

### One hot encoding

Converts a 2 dimensional array of strings/numbers into one-hot encoded format. Ideal to transform **y**-s for softmax regression and **y-**s that are unranked (nominal).

**from** sklearn.preprocessing **import** OneHotEncoder

labels = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**])

labels = np.reshape(labels, (-1, 1))

oh\_encoder = OneHotEncoder(dtype=np.int32)

oh\_labels = oh\_encoder.fit\_transform(labels).toarray()

print(oh\_encoder.inverse\_transform(oh\_labels)) #prints labels

### Ordinal encoding

Converts a 2 dimensional array of strings/number into categorical values (floats). Ideal to transform features that are strings (city/car brand).

**from** sklearn.preprocessing **import** OrdinalEncoder

#two column of cities

features = np.array([**"Tokyo"**, **"London"**, **"London"**, **"Frankfurt"**],

[**"London"**, **"Los Angeles"**, **"Frankfurt"**, **"Frankfurt"**])

features = features.T

ordinal\_encoder = OrdinalEncoder(dtype=np.int32)

cat\_features = ordinal\_encoder.fit\_transform(features)

print(ordinal\_encoder.inverse\_transform(cat\_labels)) #prints features

## Transformation pipelines

With pipelines we can define a series of data transformations to be executed consecutively (in the example: PolynomialFeatures with LinearRegression).

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.preprocessing **import** PolynomialFeatures

**from** sklearn.linear\_model **import** LinearRegession

polynomial\_regression = Pipeline([

(**"poly\_features"**, PolynomialFeatures(degree=4, include\_bias=**False**)),

(**"lin\_regression"**,LinearRegression())

])

polynomial\_regression.fit(x, y)

Fit will call each estimator’s (=member) *fit\_transform(...)* function except on the last one where it calls the regular *fit*function.

**from** sklearn.compose **import** ColumnTransformer

**from** sklearn.impute **import** SimpleImputer

missing\_embark = Pipeline([  
 (**"fill\_na"**, SimpleImputer(strategy=**"constant"**, fill\_value=**"other"**)),  
 (**"one\_hot"**, OneHotEncoder(dtype=np.int32))  
])  
  
missing\_agefare = Pipeline([  
 (**"fill\_na"**, SimpleImputer(strategy=**"median"**)),  
 (**"one\_hot"**, StandardScaler())  
])  
  
column\_preprocessing = ColumnTransformer([  
 (**"sex\_encoder"**, OrdinalEncoder(dtype=np.int32), [**"Sex"**]),  
 (**"pclass\_encoder"**, OrdinalEncoder(dtype=np.int32), [**"Pclass"**]),  
 (**"missing\_embark"**, missing\_embark, [**"Embarked"**]),  
 (**"missing\_agefare"**, missing\_agefare, [**"Age"**, **"Fare"**]),  
 (**"min\_max\_scaler"**, MinMaxScaler(), [**"SibSp"**, **"Parch"**])  
])

Column transformers are similar to pipelines. They are responsible for doing the standard column modifications (scaling/handling missings/encoding). The input should be a pandas dataframe. The output will be a numpy matrix.

## Hyperparameter tuning

### Grid search

Grid search helps with finding the best performing hyperparameters. As grid search is slow we should first start with coarse-grained searching and fine-grain it overtime.

**from** sklearn.model\_selection **import** GridSearchCV

param\_grid = [

{**"n\_estimators"**:[3, 10, 30], **"max\_features"**:[3, 4, 5] },

{**"n\_estimators"**:[3, 10], **"max\_features"**:[2, 3], **"bootstrap"**=[**False**] }

]

forest\_reg = RandomForestRegressor()

# cv: cv-fold cross validation

grid\_search = GridSearchCV(forest\_reg, param\_grid, cv=5)

grid\_search.fit(X, y)

grid\_search.best\_params\_ # optimal params in a dict

grid\_search.best\_estimator\_ # matching RandomForestRegressor object

pd.DataFrame(gs\_fit.cv\_results\_).sort\_values('mean\_test\_score',ascending=**False**)[0:5] # results of every run in a clear format

### Randomized search

Random search will not try out every combination of hyperparameters but instead will choose randomly from the intervals we’re trying to pick our hyperparamters from.

**from** sklearn.model\_selection **import** RandomizedSearchCV

**import** scipy.stats **as** stats

# np.linspace, np.arange should work too

param\_dist = [{  
 **"n\_estimators"**: [\*range(15, 150, 5)],  
 **"max\_depth"**: [5,6,7,8,9,10,11,12],  
 **"max\_features"**: [3,4,5,6,7,8,9],  
 **"min\_samples\_split"**: [2,5,10,15,25],  
 **"ccp\_alpha"**: stats.uniform(0, 3)  
}]

forest\_reg = RandomForestRegressor()

# cv: cv-fold cross validation

# n\_iter: how many combinations of params we want to sample

rnd\_search = RandomizedSearchCV(forest\_reg, param\_dist, cv=5, n\_iter=20)

rnd\_search.fit(X, y)

rnd\_search.best\_params\_ # optimal params in a dict

rnd\_search.best\_estimator\_ # matching RandomForestRegressor object

## Datasets

**from** sklearn.datasets **import** fetch\_xxx / load\_xxx

After fetching use the suitable keys (keys()) to extract the data and the labels.

## Test-train split and cross validation

**from** sklearn.model\_selection **import** cross\_val\_score, KFold  
**from** sklearn.model\_selection **import** train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

k\_fold = KFold(n\_splits=5)  
val\_scores = cross\_val\_score(sgd\_classifier, X\_train, y\_train, cv=3 or k\_fold, scoring=**"accuracy"**)

## Clustering

### Agglomerative Clustering

The method:

As initialization we put our **k** datapoint into separate clusters and in every step we will choose the two clusters that are closest (up to a point defined by the *distance\_threshold*) defined by a linkage criterion and merge them. The method will not solve the problem of choosing the number of wanted clusters but we can utilize dendrograms to tackle this problem.

The linkage criterion between two clusters has various forms:

**single linkage:**

**complete linkage:**

**average linkage:**

**ward linkage:** complex method for minimum variance within clusters

**from** sklearn.cluster **import** AgglomerativeClustering

clustering = AgglomerativeClustering (n\_clusters=5, ...)

clustering.fit(x)

clustering.labels\_

### K-Means Clustering

An algorithm that will split the datapoints into **k** (predefined number) clusters. The position of the cluster-means are updated dynamically until a stable state. The default positions of the cluster-means can be predefined or chosen randomly. A point is belonging to a specific cluster if the distance between them is the less than the distance between the point and any other cluster mean. The distance metrics here is the eucledian distance.

Algorithm:

* We have the defult positions of cluster means
* We go through every data point and append to the cluster for which the cluster-mean-datapoint distance is minimal.
* We recalculate the claster-means by averaging the data points belonging to the same cluster
* We keep on doing this until convergence

Note that the cluster centers are not necessarily real data points. If we want our cluster centers to represent real data then we should just overwrite the average with the closest real data point. In this case the method is called **K-Medoid**.

K-Means algorithm is not ideal for density based clustering or when the expected clusters are concave. sklearn’s KMeans implementation uses an updated version of the method mentioned above outputting suboptimal solutions less likely.

**from** sklearn.cluster **import** KMeans

**from** sklearn.cluster **import** MiniBatchKMeans

kmeans = KMeans(n\_clusters=5, init, n\_init, ...)

kmeans.fit(x)

kmeans.labels\_

Evaluation of clusters:

**Silhouette**

is x’s average distance from every other point in its cluster.

is the minimum of x’s average distance from every other point in foreign clusters.

refers to the class of x, return the numeric class label of x.

Silhoutte is a number between -1 and 1. Silhoutte of 1 means that x is far from the neighbouring clusters. Silhoutte of 0 means that the data point is on the border of two clusters. Silhoutte -1 is an indicator that the point is in the wrong cluster.

**Elbow curve**

Elbow curve can help us finding the ideal number of clusters for the K-Means algorithm. The elbow curve is defined the following way:



We choose the highest k value for which the error significantly dropped. Note, that if we choose the number of cluster equal to the number of datapoints the error will be zero. The W(k) value is stored in the *inertia\_*member of the KMeans class.

**from** sklearn.cluster **import** KMeans

**from** matplotlib **import** pyplot **as** plt

X = # data

distorsions = []

for k in range(2, 20):

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X)

distorsions.append(kmeans.inertia\_)

fig = plt.figure(figsize=(15, 5))

plt.plot(range(2, 20), distorsions)

plt.grid(**True**)

plt.title('Elbow curve')

### DBSCAN (Density-Based) Clustering

DBSCAN is a computationally intensive clustering technique that can handle concave (non-convex) groups by using a density-based approach.

The *eps* neighbourhood of a data point **p** are the points that are at most *eps* far from **p**:

We say that **p** is directly density reachable from **q** with respect to *eps, min\_samples* if:

**q** in this case is called a core point.

We say that **p** is density reachable from **q** with respect to *eps, min\_samples* if there is a series of where each is directly density reachable from .

We say that **p** is density connected to **q** with respect to *eps, min\_samples* if there is an **o** point such that both **p** and **q** are density reachable from **o**.

If a point **p** is in cluster **C** and **q** is density reachable from **p** then **q** will be a point of the cluster too. In the cluster every point is density connected. The DBSCAN method might not classify points. Those unclassified points are considered noise.

**from** sklearn.cluster **import** DBSCAN

dbscan = DBSCAN(eps=3, min\_samples=5)

dbscan.fit(x)

dbscan.labels\_

# Classification

During classification we are predicting classes.

## Logistic Regression

Logistic regression is used to estimate the probability of an instance failling into a particular class. Logistic regression is a binary classifier: we decide whether the instance belongs to a certain class or not.

is the **sigmoid**/logistic function that clamps its argument to the (0, 1) interval. The derivative of sigmoid is :



This mixed results in the following formula:

With maximum likelihood estimation we can estimate for which the given classification outputs (talking about the training set) appeared with the highest probablity:

The optimal can be found with the gradient ascent algorithm:

**from** sklearn.linear\_model **import** LogisticRegression

clf = LogisticRegression(...)

clf.fit(X\_train, Y\_train)

Y\_preds = clf.predict(X\_test)

By specifying *multi\_class=’multinomial’* we got to do multiclass classification with the so-called Softmax regression method. C parameter will be the inverse of the regularization constant.

### Softmax / Multinomial logistic regression

Let **K** denote the number of classes we are trying to classify our datapoints to, the outputs become one hot encoded.

For each instance we give several softmax scores corresponding to a class k. Note that each class will have its own parameters.

Now, we can predict the probability of **x** belonging to the class k the following way:

To determine the parameters we will use the categorical cross-entropy loss function:

Here is the probability of the ith instance falling into the class k. It is either 0 or 1 defined by the digit on the kth index of the one hot encoded vector. on the other side is a matrix containing each class’ parameters in the following structure:

The loss function can be minimized knowing:

## Naive Bayes

Naive Bayes has a strong assumption on the datapoint’s features which is that they are conditionally independent given y (the class they are in).

Since P() is a constant we want to classify the example **x** to the y for which the is maximal.

P(y) can be calculated from the training set’s labels:

The naive Bayes classifiers differ in calculating . It depends on our assumption regarding the distribution of .

Note that this method might not work in case of Bernoulli features (a word was present in an email or not). Suppose we have a spam email (y=1) and the word „Bye” in its text.

will return 0 if there were no training examples that were spams and included the word „Bye” which messes up the classification, even if we had several evidence that this mail was indeed spam. This is resolved by the smoothing techniques.

**from** sklearn.naive\_bayes **import** GaussianNB, MultinomialNB, BernoulliNB

clf = BernoulliNB(alpha=1.0) #1.0 for laplace smoothing

clf.fit(X\_train, Y\_train)

Y\_preds = clf.predict(X\_test)

Use naive Bayes as a baseline as it is fast and sometimes works well.

### Laplace smoothing

Let denote the probability of the jth word being present in a spam email (calculated from the training set):

If we haven’t seen a word appearing in either spam or non-spam emails during training time it’s a bad assumption that it won’t ever show up (0 probability). The correction:

+2 refers to the number of possible classes (spam, non-spam).

## Support Vector Machine

Support vector machines tend to solve the issue of finding the largest margin classifier. Support vector machines require proper scaling to use them efficiently.

### LinearSVC

**from** sklearn.svm **import** LinearSVC

# ... pipeline for standard scaling

clf = LinearSVC(C=1, loss=**"hinge"**)

clf.fit(X, y) # y does not require any encoding

clf.predict() # will return the class, not probabilities

Sklearn implements a subtype of SVMs called ’soft margin’ classifiers. It solves the problem of having outliers and can handle non-separable cases. The softness of the margin can be controlled with the C parameter. A low C (~1) parameter allows a lot of margin violations, a high one (~100) will allow only a few. Decreasing C will strengthen the regularizing effect.

### SVC for nonlinear data

**from** sklearn.svm **import** SVC

# ... pipeline for standard scaling

clf = LinearSVC(C=1, loss=**"hinge"**)

clf.fit(X, y) # y does not require any encoding

clf.predict() # will return the class, not probabilities

linear kernel:

Will result the exact same classifier as LinearSVC did but much-much faster.

**from** sklearn.svm **import** SVC

# ... pipeline for standard scaling

clf = SVC(C=1, loss=**"hinge"**, kernel=**"linear"**)

polynomial kernel:

Gives the same result if we just did polynomial feature preprocessing in before fitting with enhanced efficiency.

The polynomial kernel is defined as the following:

Here x and y are two datapoints, is gamma, is coeff0, d is degree.

The transformed data points (the ones that are transformed by a yet unknown , called the feature map) should fulfill the following property:

From that the feature map can be computed [Wikipedia](https://en.wikipedia.org/wiki/Polynomial_kernel).

**from** sklearn.svm **import** SVC

# ... pipeline for standard scaling

clf = SVC(C=1, degree=3, coeff0=1, gamma=1, kernel=**"poly"**)

rbf kernel (Gaussian radial basis function):

The theoretical part is similar as it was in the polynomial kernel. The RBF kernel is:

is the gamma hyperparameter.

**from** sklearn.svm **import** SVC

# ... pipeline for standard scaling

clf = SVC(C=1, gamma=1, kernel=**"rbf"**)

### Laplace smoothing

## Decision Tree

## Random Forest (ensemble method)

Ensemble method is technique that creates multiple models and them combines them to produce better results than any of the single models would individually. Many models get combined into a metamodel.

In case of random forest an aggregated prediction of many decision trees will determine the final prediction. Implicitly handles outliers, missing values and less likely to overfit. Random forest without a single data cleaning could give surprisingly good results and can return the importances of each feature. (feature\_importances\_)

**from** sklearn.ensemble **import** RandomForestClassifier

Bagging: Random forest makes a number of decision trees on various subsets of the given dataset and takes an average to improve the prediction accuracy. This task can be done parallelly.

## Gradient Boosting (XGBoost, ensemble method)

Gradient boosting is a similar ensemble method to random forest. Worth to use on unbalanced datasets, where one label has larger number of instances. On the other hand, it can overfit.

**from** sklearn.ensemble **import** GradientBoostingClassifier

Boosting: Boosting is a sequential process. (TODO)

## Confusion Matrix and ROC Curve

**from** sklearn.metrics **import** confusion\_matrix  
**from** sklearn.metrics **import** precision\_score  
**from** sklearn.metrics **import** recall\_score  
**from** sklearn.metrics **import** f1\_score  
  
cfm = confusion\_matrix(y\_train, y\_preds)  
ps = precision\_score(y\_train, y\_preds)  
rs = recall\_score(y\_train, y\_preds)  
fs = f1\_score(y\_train, y\_preds)

Given the task of deciding whether a handwritten digit is 5 or not:

**Precision (3 / 4)**



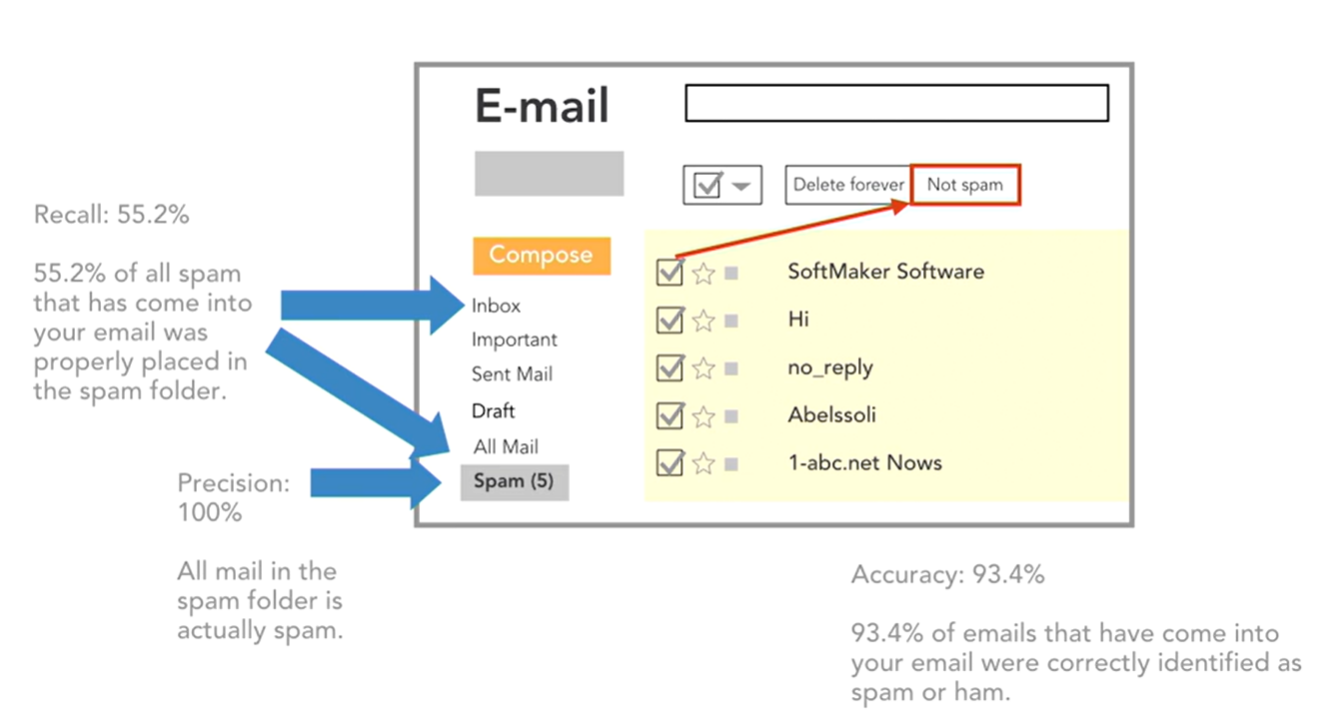
**Recall (3 / 5)**

\*TN: The letter is actually other than 5 and fortunately the model says the same.

\*\*FP: The letter is actually other 5 but we accidentally classify it as 5.

\*\*\*FN: The letter is actually 5, but we accidentally classify it as other than 5.

\*\*\*\*TP: The letter is actually 5 and luckily the model says so as well.



### Rand index (=accuracy in binary case)

We divide the correctly clustered/classified instances with the number of every index.

### Jaccard index

Jaccard index is mostly used in computer vision when using bounding boxes to identify objects. Jaccard index in computer vision will be the area of overlap divided by the area of union often called the IoU metrics (Intersection over Union).



### Precision

Amongst all the cases the model predicted positive, how many of them is actually positive. (Collecting games that can be played by children. Gather less, but guarantee that those games wont show any gore / adult content)

### Recall

How many amongst all the positive cases in the real world got found by our model. (For example we’d like to get all the patients with viral infections from a group of people. It’s definitely better if someone has to go to the hospital redundatly rather than missing someone with dangerous disease.)

### F1 Score and Precision-Recall tradeoff

Usually, the lower the precision is, the higher the recall is. Thus, we combined these two measures into ’F1 Score’ by taking the harmonic mean of them.

Ideally, we’d like to maximize this.

**from** sklearn.metrics **import** precision\_recall\_curve  
  
y\_scores = sgd\_classifier.decision\_function(X\_test)  
precisions, recalls, thresholds = precision\_recall\_curve(Y\_test\_5, y\_scores)  
plt.title(**"PR curve"**)  
plt.plot(thresholds, recalls[:-1], **"g-"**, label=**"Recall"**)  
plt.plot(thresholds, precisions[:-1], **"b--"**, label=**"Precision"**)  
plt.legend(loc=**"center right"**)  
plt.ylim([0, 1])  
plt.xlabel(**"Threshold"**)  
plt.grid(**"on"**)  
plt.show()

*Receiver operation characteristic curve* (**ROC**) is another tool used with binary classifiers.

ROC curve plots the True Positive Rate (TPR a.k.a. recall or sensitivity) against the False Positive Rate (FPR). FPR is equal to (1 – True Negative Rate). True Negative Rate (specificity) is the ratio of the actually negatively marked instances over all the negative instances of the real world.



We can see the tradeoff here as well. As we find more and more actually positive

cases the false positive rate increases simultaneously. One way to compare classifiers is to measure the AUC (area under the curve). An ideal classifier’s AUC would be close to 1 because we want a classifier that almost touches the upper left corner.

**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** roc\_auc\_score  
  
fpr, tpr, thresholds = roc\_curve(Y\_test\_5, Y\_preds\_5)  
auc\_score = roc\_auc\_score(Y\_test\_5, Y\_preds\_5)  
plt.title(**"Receiver Operating Characteristic"**)  
plt.plot(fpr, tpr, **"b"**, label=**"AUC = {:0.2f}"**.format(auc\_score))  
plt.legend(loc=**"lower right"**)  
plt.plot([0, 1], [0, 1], **"r--"**)  
plt.xlim([0, 1])  
plt.ylim([0, 1])  
plt.ylabel(**"True Positive Rate"**)  
plt.xlabel(**"False Positive Rate"**)  
plt.grid(**"on"**)  
plt.show()

## Multiclass Classification

Doing multiclass classification is not always supported natively by the most frequent classification algorithms. Thus, the problem of multiclass classificiation is often reduced to multiple binary classification problems.

The examples below will show two different approach to solve the 10-digit handwriting recognition problem.

### One versus All (One versus Rest)

Make 10 binary classifier that tells whether the given digit is 0, 1, ..., 9 or not.

On a newly introduced digit we run all these 10 classifiers and select the class whose classifier outputs the highest score.

### One versus One

This method trains a binary classifier for all possible pairs (45 for the current problem)

Run all 45 classifiers on a digit never seen before and choose the class that wins the most.

## Egyéb

# Regression

While doing regression we try to predict a continous outcome variable (y) based on some input data **x**.

## Linear Regression

Linear regression assumes that y is a linear function of **x**’s attributes.

We can rearrange this into a matrix form (we prepend **x** with the intercept term 1):

The task from now on is to find the parameters that minimize the loss function (here m is the number of training examples):

We can solve the minimization problem by gradient descent (iterative) or normal equation, for which we define the desing matrix **X** where the rows are the training examples without the values (**y** column vector):

The normal equation:

**from** sklearn.linear\_model **import** LinearRegression

regression = SGDClassifier()

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

regression.intercept\_

regression.coeff\_

### Ridge regression (L2 regularization)

The extra term in the cost function forces the learning algorithm to not only fit the data but also keep the model’s weight small.

adjusts the power of the regularization. An close to zero will result in the classic linear regression, on the other hand a significantly large will result a flat line passing through the data’s mean. Note that the bias term is not regularized.

For ridge linear regression we have a closed form (non-iterative) solution as well:

**from** sklearn.linear\_model **import** Ridge

regression = Ridge(alpha=1.0)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

L2 (2nd norm) regularization works well with polynomial regression.

### Lasso regression (L1 regularization)

Least absolute shrinkage and selection operator.

It works similarly as the L2 regularization but here we are using the first norm. This type of regularization excels at completely eliminating weights connected with the less important features.

**from** sklearn.linear\_model **import** Lasso

regression = Lasso(alpha=1.0)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

### Elasticnet

Elasticnet is a middle ground between Ridge and Lasso regression (or L1 and L2 regularization). The mix of the two methods can be controlled with a hyperparamter r as follows:

Elasticnet is slightly more preferred over Lasso, even if you suspect that there will be some useless features.

**from** sklearn.linear\_model **import** ElasticNet

regression = ElasticNet(alpha=1.0, l1\_ratio=0.5)

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

## Locally Weighted Linear Regression

LWLR is a non-parametric learning algorithm which means there is no fixed to the data, but every prediction will require a completely new parametrization. This essentially means that we have to keep our data in the memory. Locally weighted linear regression can help in problems where classic linear regression fails.



When doing LWLR the loss function what we’d like to minimalize with the proper looks like the following:

w is the weighting function, a common choice for w is:

is the bandwidth parameter which controls the wideness of the significant neighbourhood (larger tau implies larger non-zero impact neighbourhood).

If is close to **x** then the weight is large (close to 1) and has large impact on the output. On the other hand when is further away then the weight will be close to 0.

LWLR has no sklearn implementation.

Use LWLR for low dimensional dataset (at most 10 features) where we have a lot of data and we don’t want to do feature selection.

## Polynomial Regression

Quite often a straigth line won’t be able to fit the data properly as the data is clearly nonlinear. When doing polynomial regression we add certain powers of the original features as new features and with that we use the classic linear regression algorithm.

**from** sklearn.preprocessing **import** PolynomialFeatures

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_train = poly.fit\_transform(X\_train)

Here if X\_train was a single training example with features [a, b] then the fit\_transform will do the following expansion of input features (with include\_bias turned off, thus missing a 1 from the beginning):

With the new training data we can do linear regression to determine :

**from** sklearn.linear\_model **import** LinearRegression

regression = SGDClassifier()

regression.fit(X\_train, Y\_train)

Y\_preds = regression.predict(X\_test)

regression.intercept\_

regression.coeff\_

## Support Vector Machine

## Support Vector Machine

## Support Vector Machine

## Support Vector Machine

# Statistics

## Permutation, Variation, Combination

P = 1·2·...·n = n! (number of orderings of n different elements)

P = n! / (k1!·k2!·...·kn!) (number of orderings if we have k1, k2, ..., kn pieces of the same instance)

V = n(n-1)...(n-k+1) (numbers of ordering of k diff. elements chosen from n diff. elements)

V = n·n·...·n = nk (if we allow repetition during the selection of the k elements)

C = n! / k!(n-k)! (same as the first V but we dont order, that’s why the k! division)

C = (n-k-1 k) (same as C but we allow repetition)

## Probability theory

P(A+B) = P(A) + P(B) – P(AB)

P(AB) = P(A|B)P(B) = P(B|A)P(A)

if A and B are independent events then P(AB) = P(A)P(B), P(A|B)=P(A)

if A and B are mutually exclusive (aka disjoint) then P(AB) = 0

Mutually exclusive events are those events that cannot occur at the same time. In case of independent events one event remains unaffected by the occurrence of the other event. Note, that two mutually exclusive event are never independent: one occuring anticipates the occurrence of the other (if A happened B will not).

Events A1, A2, ..., An are fully indenpendent if they are all pairwise independent.

We examine the probability of A happening if we know that B already did. P(B) cannot be 0 for this reason.

**Chain rule of probality:**

**Law of total probability:**

A1, A2, ..., AN are pairwise disjoint and their union is the whole sample space (their probabilities summed up is 1). P(Ai) cannot be zero.

**Bayes’ rule:**

We have the same assumptions as before. We substitute P(B) with the formula given above.

## CDF, PDF, PMF

**Cumulative distribution function (CDF):**

**Probability density function (PDF):**

**Probability mass function (PMF):**

It is the same as PDF, but in discrete case. P(X = x) for example.

## Expected value, Variance, Standard deviation

**Expected value:**

Discrete:

Continous:

**Variance (the average squared discrepancy from the mean):**

**Standard deviation:**

Identities:

E(aX + b) = aE(X) + b

V(aX + b) = a2V(X)

By substituting with a, we got an equivalent form:

z-score of X:

**Conditional expectation:**

Discrete:

Continous:

**Law of total expectation:**

The expected value of the X|Y conditional expectation is the expected value of X.

**Law of unconscious statistician (LOTUS):**

Discrete:

Continous:

## E(X) and V(X) of random length sum of iid variables

iid: Identically and independently distributed.

are iid. N is a random variable coming from a specific distribution.

## Moments, Skewness, Kurtosis

kth moment: E(Xk)

kth central moment: E[(X-E(X))k]

kth abs. moment: E(|X|k)

kth abs. central moment: E[|X-E(X)|k]

The 1st moment is the expected value. The 2nd central moment is the variance. With little modification the 3rd c. moment is the skewness (assimetry of the random variable about its mean) and the 4th is the kurtosis.

## Median, Mode, Quantile (percentile)

The median is a single number that separates the data sample to a lower half and a higher half. In discrete case median is the middle element of the ordered dataset or the average of the 2 middle elements.

The modes are the most frequent elements, so the ones that have the highest probabilities. For continous distributions the modes are the x values for which the PDF has a local maximum.

The p-quantile of the random variable X is the number for which the X is below with p chance and above with 1-p chance. The percentile is basically the same concept except we grant the p chance in percents.

Quantile () for the probablity.

## Multidimension distributions, Covariance, Correlation

E(X + Y) = E(X) + E(Y)

V(X + Y) = V(X) + V(Y) + 2cov(X, Y)

V(X - Y) = V(X) + V(Y) - 2cov(X, Y)

E(XY) = E(X)E(Y) if X and Y are independent

V(X+Y) = V(X-Y) = V(X) + V(Y) if X and Y are independent

**Joint CDF**:

Discrete: (xi, yj) are the pairs that can be assigned to (X, Y)

Continous:

**Marginal CDFs:**

Events X and Y are independent if (discrete) or (continous).

Discrete:

marginal CDF of X:

marginal CDF of Y:

Continous:

marginal CDF of X:

marginal CDF of Y:

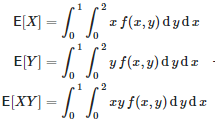
**Marginal PDFs:**

marginal PDF of X:

marginal PDF of Y:

Expected values of individual variables:

*In the example the joint pdf returns nonzero having x between 0 and 1 and y between 0 and 2.*



**Law of total probability and Bayes’ rule in continous case:**

**Covariance:**

Covariance is the unscaled version of correlation. Covariance indicates the direction of the linear relationship between variables.

Note that: , which means that we don’t need to center X and Y both, but only one.

**Correlation:**

Correlation on the other hand both measures the strength and the direction of the linear relationship between two variables. R is in [-1, 1]. If X and Y are independent, then R is 0.

The statement ”If the covariance/correlation is zero then X and Y are independent” is false generally. A well known exception is the **multivariate Gaussian**.

## Popular discrete distributions

### Indicator function (aka characteristic function, Bernoulli distr.)

PMF short form (X follows a Ber(p)), without braces:

### Binomial distribution

We make n independent experiments and we know that the event will happen with probability p. The goal is to find out the probability of the event happening k times during the n trials. X will be the number of successful trials when the event went off. It can be considered as the sum on n indicators.

Expected value and variance:

### Poisson distribution

If n is large and p is really low the binomial distribution transforms into the so-called Poisson distribution. is the expected value (= np).

Expected value and variance:

### Geometric distribution

We keep on experimenting until we finally make the event A (p = P(A)) go off. X is the number of experiments we need to do until the first appearance of A.

Expected value and variance:

### Hypergeometric distribution

We have N product in which M is faulty. The goal is to find out the probability of us choosing n product (without replacement) we got k faulty in our hands.

Expected value:

## Popular continous distributions

### Uniform distribution

X is uniformly distributed in I=[a,b] if the probability of X being in any subset [a0, b0] is equal with the ratio between the width of the subset and I.

Expected value and variance:

### Exponential distribution

X follows an exponential distribution with parameter if the following holds:

Expected value and variance:

Expected value equation can be proven using partial integration and L’Hopital rule.

Exponential distribution is a memoryless kind of distribution:

On a real example (tram arrival times) this means that the probability of a tram arriving in the next **s** minutes given that I had been waiting for **t** minutes already is the same as if I would start the waiting again from scratch.

### Normal distribution

X is normally distributed if there are parameters ) for which the PDF is:

X is standard normally distributed if . In this case the PDF and the CDF are:

Relation between and :

values can be found in tables. Also = 1 - because is symmetric.

Expected value and variance:

The sum on normally distributed random variables will be itself a normally distributed variable:

### distribution (Chi-square distribution)

If X1, X2, ..., Xn are all standard normally distributed fully independent random variables then we say that the random variable

is following a distribution with n degrees of freedom.

### Student distribution (t distribution)

If X1, X2, ..., Xn, Y are all standard normally distributed fully independent random variables then we say that the random variable

is following a Student (or tn) distribution with n degrees of freedom.

### Fisher distribution (F distribution)

If X is a distributed random variable with n degrees of freedom and Y is a

distributed random variable with k degrees of freedom we say that the random variable

is following a Fisher distribution with k and n parameters.

## Limit theorems

### Law of large numbers

The relative frequency (empirical probability) will converge to the actual probablity as we increase the number of trials.

If X1, X2, ..., Xn are independent random variables from the same distribution (samples) then we can say that the mean of those variables will converge to the expected value.

### de Moivre-Laplace

We make a lot of independent experiments. Let Xi the indicator that tells whether the ith trial was successful. (X is 1 if head, 0 if tail for example in case of a coin tossing game). For a successful trial (A goes off) we have p = P(A) chance.

or if we consider X as a binomially distributed random variable :

### Central limit (CLT)

X1, X2, ..., Xn are independent random variables that are following the same distribution. Assuming expected value E(X)and standard deviation we got the following formula:

Alternative formulation:

For multivariate case (with expected value vector and covariance matrix) we have:

As in 1 dimensional case we assume that the number under the square root is larger than or equal to zero we need to assure that the matrix is positive semidefinite. Luckily, every covariance matrix has this property.

Note: *Square rooting a matrix is hard. It requires Cholesky decomposition or eigenvalue decomposition.*

### Hoeffding’s inequality

X1, X2, ..., Xn are independent random variables that are following the same distribution and can take values in between [a, b].

The Hoeffding inequality holds for small n-s (< 30) unline CLT:

### Delta method

Zn is a sequence of random variables that satisfies:

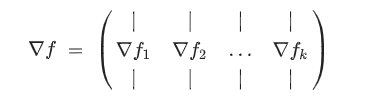
If g is a continous function, then the term (standard variance\*):

The statement can be proven by Taylor expanding around .

Delta method in multivariate case:

Let g be a continiously differentiable method mapping from . (variance\*)

is often the sample average.



Gradient of the g function.

\**The second parameter is sometimes the std other time the variance.*

## Fundamental concepts of statistics

**Population**: Set of similar items on which we’d like to make experiments. (Every FB user)

**Sample**: Set of individual items selected from the population.

**Types of statistical variables:**

Quantitative/Numerical: the number represents real amounts

Qualitative/Categorical data: the number represents grouping

Nominal var: The groups has no order (Man/Woman, Brand)

Ordinal var: The groups are ranked (rating)

We model the sample with n random independent variables with the same distribution. We can image that we are given n random instances of the population and from that, we’d like to know more about the population. The sampling should be representative.

We call a function of samples a statistic (e.g.: average). Estimator is a type of statistic that does not depend on the parameter that it wants to estimate.

Sample mean:

kth smallest:

Sample standard deviation:

Uncorrected:

Corrected (prefer this as it gives precise estimate!):

Sample range:

Empirical CDF:

**Glivenko-Cantelli’s theorem:**

As we increase the number of samples we can reconstruct the CDF of the population fully.

**sup** of a set is the lowest value for which every set-element is lower. The max must the be element of the set, the sup doen’t have to.

## Parameter estimation

We have an assumption on the distribution family that the population follows. From the samples we’d like to get the parameters of that distribution. (e.g.: mean and std for a distribution assumed Gaussian/normal)

Statistics (Tn) is a function of the samples X1, X2, ..., Xn. We say that the parameter is unbiased if

An estimation is consistent if .

An estimation is strongly consistent if the variance converges to zero:

An estimation T is unbiased (torzítatlan) if

An estimation T is asymptotically unbiased if

The bias of a biased estimator is: .

Between two unbiased estimators the more performant is the one with the faster convergence to zero variance . (namely, for fixed semi-large n the variance is lower).

For biased estimaters we define performance a bit different way. In this case the metric is called the quadratic risk, which we want to minimize in limit:

**Cramer-Rao inequality:**

We saw that in case of a strongly consistent estimator we have 0 as variance in limit. The Cramer-Rao bound defines a boundary for fixed n for which the variance cannot fall below.

I is the Fisher information defined as following:

is the log likelihood function.

In multidimensional case we can introduce the concept of the Fisher information matrix:

If theta is in then I is a d by d matrix.

### Point estimation

We exactly determine the parameter that best fits our sample.

Maximum likelihood estimation:

We look for the parameters for which we have the highest chance to get our n samples.

is the parametrized PMF or PDF. We choose such a way that the (log)likelihood becomes maximal:

To determine we derivate the log(likelihood) function with respect to and make it equal with 0. This may give the stationary point for the maximum.

MLE can be used for determining the parameters of multiparametered distributions (Gaussians for example). In those cases gradients and Hessians are used replacing the standard first and second derivatives.

Method of moments:

It is a more general method than MLE. is the definition of the jth moment. A requirement to use MoM is that the parameters we’re looking for should be constructed from the moments in some way.

empirical moment (to model the expected value of Xj):

With empirical moments we can give estimates to the parameters (remember we have the assumption that each parameter is a function of moments )

**Example for MoM:**

Let’s make an estimation on the expected value () and the standard deviation () of a normal distribution with the method of moments.

First check out whether those parameters can be constructed from moments:

Yes they can, so we calculate the empirical moments:

From this we can have our estimations ():

### Interval estimation

In case of continous distributions we cannot assure that the point estimation gives us the real parameters (actually we have 0 chance for that), so we’d like an interval in which we will likely find the real parameters of the underlying distribution of the population.

In case of interval estimation, we use two sample statistics (Ln and Un) for which

In this case we call the interval (ln, un) the 100% confidence interval for .

**Example for interval estimation**

Let’s give a confidence interval for the expected value () of the normal distribution if the standard deviation () is known.

We know that follows a standard normal distribution.

Find the critical y for which:

From this we have the confidence interval for with rearranging:

To determine y we need to do the following ( is the std. normal CDF):

Often the bounds of the confidence interval contains the estimated parameter itself. To solve this we have three methods.

**Conservative:** Upper estimate the parameter in the bounds.

Example, the variance of Bernoulli can be upper estimated:

**Solve:** Two tail symmetric problems can be solved by squaring.

**Plug in:** We can just replace the parameter with the proper sample statistics. For example we could replace with . Note in this case having a large n is more important as we depend on limits twice as much (CLT + ). When doing plug-in the confidence interval will have an asymptotic confidence level.

### Distance between distributions

#### Total variance (TV)

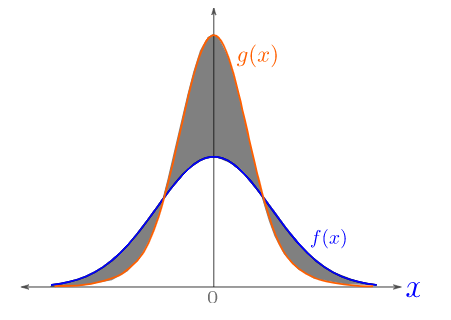
The **total variance (TV)** is defined similarly for both discrete and continous variables. Define **P** and **Q** as two distibutions, with PMFs or PDFs f and g.

This can be computed in both cases:

Discrete case:

Continous case:

The TV distance can be used to measure the difference between the real parameter and the estimated parameter of a Gaussian. Zero distance means perfect fitting. Also, worth noting the TV is at most 1 by triangle inequality.



For comparing multiple distributions not akin to each other we can utilize the triangle equality as well (e.g.: **P** and **V** are Bernoulli random variables while **Q** is Poisson distributed.):

#### Kullback-Leibler divergence (KL)

Let **P** and **Q** bet two distributions with PMFs or PDSs of **p** and **q**. Like before **E** will be the sample space.

Discrete case:

Continous case:

KL-divergence is not symmetric. KL-divergence is not a distance, but it has the *definite*propery meaning that if KL is zero, then the two distributions are exactly the same.

## Hypothesis testing

We’d like to review the truthfulness of statements. Usually these statement are referring to the overall population (null hypothesis) but we only have data samples. Also, often these statement are not completely true. We’d like to discover what is the contradiction level between the statement and our data when we don’t believe what the statement claims. Where’s the point where we sense significant level of discrepancy between the data we see and the statement? How can we measure significance level?

By default we accept every statement unless the data clearly says the opposite.

During hypothesis testing the goal is not to find the adequate mean and other parameters and compare with the one the null hypothesis says (e.g.: all car have 5,4 +/- 0.1 liter consumption) but rather to measure how much our data sample differs from the null hypothesis and what is the difference we can still tolerate.

We accept/reject a statement on a given significance level. (Significance level is the probability that that the statement is true but our data says the opposite)

H0 (null hypothesis) will say the distribution is in the set of .

H1 (alternative hypothesis) will say the distribution is the set of .

and are non-overlapping sets.

|  |  |  |
| --- | --- | --- |
|  | H0 accepted | H0 rejected |
| H0 is true (reality) | TN – correct decision | Type 1 error: False positive |
| H0 is false (reality) | Type 2 error: False negative | TP – correct decision |

Number of false positives can be measure with the significance level (usually 0.05 - 0.1). Number of false negatives cannot be controlled and overall we have hard time estimating it. As a rule of thumb we can keep the presence of type 2 errors low by not choosing a confidence interval that is excessively large.

**power of a test:**

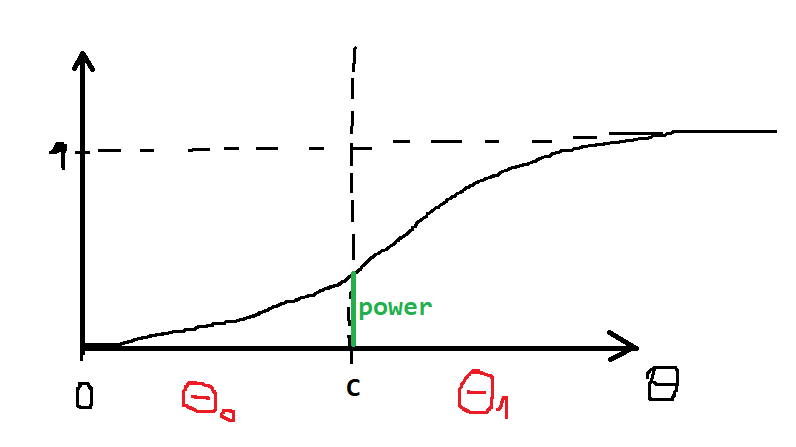
Let be the type 1 error and be the type 2 error. is the parameter we’re testing on.

and are disjoint sets.

The power of a test is defined as follows:

The power function here is the probability of not commiting a type 2 error:

The graph plotted with the actual power highligted:



We prefer tests that have low power.

Note that the significance level (type 1 error) can be defined similarly as the power:

**p-value:** p-value is the probability of getting a data sample at least as extreme as the sample statistics assuming that the null hypothesis is true. If the p value is lower than the significance level we have to reject the claims of the null hypothesis. Imagine having a data sample with mean of 25 whereas the null hypothesis stated that the mean is 20. Then the p value is:

p-value is the significance level of the test statistics where we just reject the null hypothesis. If this p-value is lower than the predefined significance level we reject the null hypothesis. It is another method besides critical values to test hypotheses.

If we reject the null hypothesis on a significance level (0.1 for example -> 90%) then we reject it on **every larger** significance level as well (0.15 -> 85%). So if our predefined significance level was 0.15 and we get that the p-value is 0.1, then we have to reject the null hypothesis.

Low p-value means real scientific discovery (high chance of rejecting!).

### Z-test (z-próba, u-próba)

We have X1, X2, ..., Xn samples and we know that each of them is coming from a normal distribution with known standard deviation () but unknown mean ().

The null hypothesis says that the mean is . is the chosen significance level.

If H0 is true than u follows a standard normal distribution,

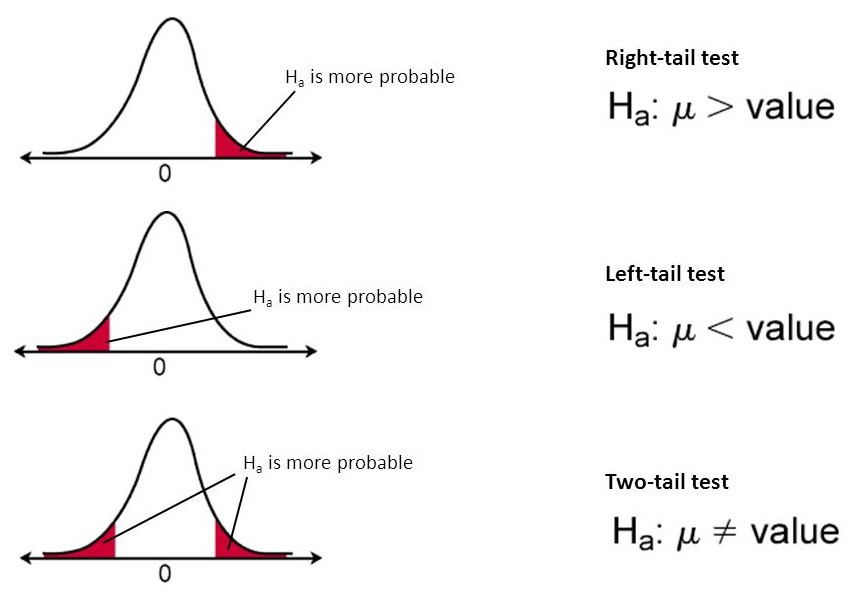
We substitute the Xi-s with our real numeric data and if we get that the then we accept the null hypothesis on the significance level . Once again if we accept the null hypothesis then that means that between the sample mean and the proposed mean there were no significant differences to reject the claim.

|  |  |
| --- | --- |
|  | 0.9 |
|  | 0.95 |
|  | 0.975 |

We use Z-tests on-non normally distributed data too if we have a lot of samples. You can justify the legitimacy of this method with the central limit theorem (CLT).

**Meaning of two-tailed (two-sided) test:**

A two-tailed z-test will check both if the mean is significantly greater than the one the null hypothesis claims or significantly lower. On the other side a **one-tail test** will only check if the sample mean is lower or higher than the claimed one.



*Figure on how to compose a test in the two-tail and one-tail case. Ha is the* ***alternative*** *hypothesis.*

*(right one-tail)*

*(left one-tail)*

*(two-tail)*

two sample case:

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We know the distributions’ standard deviation.

The null hypothesis in this case claims that the expected values are the same. ()

By using the null hypothesis () and standardizing we get:

And from this point we can follow the previously introduced method:

### Student’s t-test

We are given X1, X2, ..., Xn  independent and normally distributed samples, but we don’t know the distributions’ standard deviation.

The null hypothesis says that the expected value is m0.

t follows a Student (t) distribution with n-1 degrees of freedom.

can be found in one of the charts on the last page.

two sample case:

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know the distributions’ standard deviation but we know that they can be considered equal (this should be reviewed with F-test).

The null hypothesis in this case claims that the expected values are the same. ()

This time t follows a Student (t) distribution with n+m-2 degrees of freedom.

paired sample t-test:

We have (X1, Y1), ... (Xn, Yn) samples (we take samples from the same objects twice for like comparing „before-after” or similar purposes). The assumption here is that is normally distributed:

The standard deviation () is unknown.

The null hypothesis in this case is that the expected value is zero ( or .

t follows a Student (t) distribution with n-1 degrees of freedom.

### F-test (Fisher-test)

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know neither the underlying distributions’ mean nor their standard deviation.

The null hypothesis questions whether we can consider .

We take the corrected variances , and our test statistics will be:

f follows a Fisher distribution with parameters n-1 and m-1.

can be found in one of the charts on the last page.

### Welch’s t-test

If F-test is rejected then we cannot use the t-test for two samples checking whether their expected value is the same.

We are given X1, X2, ..., Xn and Y1, Y2, ..., Ym independent and normally distributed samples. We don’t know neither the underlying distributions’ mean nor their standard deviation.

The null hypothesis in this case claims that the expected values are the same. ()

is Student distributed with f degrees of freedom where for f the following holds:

can be found in one of the charts on the last page.

### Non-parametric tests introduction, common problems

non-parametric tests:We consider the distribution completely unknown.

Due to the lack of information non-parametric tests only works on large data. Our information is really general in this case (standard deviation is finite, the distribution is continous, etc.)

* **Goodness of fit (illeszkedésvizsgálat):** When measuring goodness of fit, we’d like to assure that our sample is following some hypothetical distribution. (null hypothesis). For this we can use Chi-Square test or the single-sample Kolmogorov-Smirnov test.
* **Test of independence (függetlenségvizsgálat):** We check whether the examined variables are independent. For this we can use Chi-Square test.
* **Test of homogenity (homogenitásvizsgálat):** We have two samples and we’d like to check whether they come from the same distribution. For this we can use Chi-Square test, two sample Kolmogorov-Smirnov test, Mann-Whitney test, Wilcoxon test, Kruskal-Wallis test, Friedman test

### Kolmogorov-Smirnov test

One sample for checking goodness of fit:

We have a sample of X1, X2, ..., Xn  random variables which come from a continous distribution. The null hypothesis say that the sample follows the hypothetical CDF .

We can omit the term, so we must only compare the supremum values.

The critical values can be found with the linked code snippet in RStudio.

Two sample for test of homogenity:

We have a sample of X1, X2, ..., Xn and Y1, Y2, ..., Ym random variables which come from a continous distribution. We’d like to know whether they have the same distribution.

Where and are the empirical CDFs of X and Y.

### Wilcoxon signed-rank test

Homogenity test that can be used for ordinal variables (aka variables that are ranked, and ranking contains additional information).

We have a sample of X1, X2, ..., Xn  random ordinal variables which come from a continous distribution. We need to decide whether we can accept the proposed median of:

H0 says that the median should be med0:

(**remove the zero differences!**)

is the ranking of in the ordered samples of a-s ().

*In case of identical values (1 1 3 5) we will substitute the rankings of identical values with the ranking average so instead of (1 2 3 4) the result will be (1.5 1.5 3 4). In R programming this effect can be achieved by using rank(x, ties.method=”average”).*

ad

When the amount of samples is low (at most 20 elements):

We check the critical values for from a table (qsignrank). The critical value depends on the number of samples and the level of significance. We accept the null hypothesis if:

When we have sufficiently large amount of samples:

Paired sample Wilcoxon test:

We have (X1, Y1), ... (Xn, Yn) samples of ordinals. We’d like to decide whether their distributions are the same. In other words the null hypothesis questions if the difference of the pairwise samples has the median of 0 or not on some significance level.

This problem can be transformed into the problem we saw during the one sample case by redefining the following way:

### Mann-Whitney U test

We have a sample of X1, X2, ..., Xn and Y1, Y2, ..., Ym random ordinal variables. We’d like to know whether they have the same distribution (or the same median).

First we create the sample Z: by forming a union between X and Y. Then we rank the samples by their absolute rank in the ordered Z sample.

Let C be the critical value for the n and m on the given significance level.

If then we accept that their distribution is the same.

For large samples (over 20) the following statistics could be considered standard normal:

The uc values can be found with the help of the proper table or qnorm(1-eps/2)

### Chi-square test

goodness of fit:

We have X1, X2, ..., Xn samples and we’d like to determine whether they follow the hypotethical CDF (null hypothesis says that they do!)

As a preparation we split the real numbers into **r** disjoint subset:

Then we calculate the probability of an X falling into the interval in case of the hypothetical CDF .

will be the number of data samples falling into the interval . We would expect it to be approximately .

The test statistics will be:

T follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom. Approximated or calculated parameters decrease the degrees of freedom (empirical mean, empirical variance from the samples).

If then we accept the null hypothesis.

test of indepence:

We have (X1, Y1), ... (Xn, Yn) paired samples. We’d like to check whether the components of the samples are independent or there is a significant bound.

claims that

We split the space of real numbers into k and l partitions:

Define to be the number of (X,Y) pairs for which .

Define the empirical probabilities the following way:

Our test statistics will be:

Tn follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom.

If then we accept the null hypothesis and by that, we consider X and Y independent.

test of homogenity:

We have independent samples of X1, X2, ..., Xn and Y1, Y2, ..., Ym random variables. We’d like to check if their distributions are the same (null hypothesis).

We split the space of real numbers into r partitions:

Define as the number of samples from X and Y (respectively) falling into the partition .

The constructed test statistics (T) will be:

T follows a chi-square distribution with degrees of freedom. We can find the critical value by plugging in the significance level and degrees of freedom.

If then we accept the null hypothesis and by that, we consider X identically distributed as Y.

### ANOVA (Analysis of variance)

ANOVA is a method for comparing several normally distributed populations’ mean. An important criteria here is that the variance should be equal for every distribution.

two sample Student (t) test is a special case of the ANOVA method (two groups).

ANOVA checks whether the deviation is caused by pure randomness or there is some connection between the groups (eg.: difference between means). Doing t-tests pairwise is feasible to do on low group count, but the computational requirement increases quadratically as we try to use it on more and more groups. Also, we have high chance to get Type 1 errors while doing independent t-tests rendering the significance level useless.

We have the samples coming from the distributions respectively. Also, we know that .

Define m as:

In this case (model of ANOVA):

here is the deviation of the group (group effect)

is the error term.

One-way ANOVA (egyszempontos varianciaanalízis):

H0: The null hypothesis claims that .

**Can be proved that**

Q sums up n normal variables but computes one: with n-1 degrees of freedom.

sums up p normal variable but computes one: with p-1 degrees of freedom.

sums up n normal variable but computes p: with n-p degrees of freedom.

Our test statistics will be:

T follow a Fisher distribution with (p-1), (n-p) parameters.

If this is fulfilled for T then we accept that .

### Bartlett-test

To do ANOVA we have to ensure that the variances are identical.

We are given the samples coming from the distributions respectively.

H0: The null hypothesis claims that the variances are the same: .

Define the in-group empirical variance and total empirical variance the following way:

The test statistics will be:

This T approximately distributed with (p-1) degrees of freedom.

If this is fulfilled for T then we accept that .

### Friedman-test

The Friedman-test is the generalization of the paired sample Wilcoxon sign-rank test. For various **ordinal** groups (samples) we’d like to determine if they are following the same distribution (basically means that they have identical medians) or not.

We are given the samples . (n groups, each containing p random variables (threatments))

We order each sample in increasing order and make a ranking. Let be the ranking of the ith variable in the jth sample group.

For each ’column’ we sum up the ranks:

Our test statistics will be:

In case of low amount of samples we look for the critical values in a table.

Otherwise we know that T is distributed with (p-1) degrees of freedom.

### Kruskal-Wallis test

Kruskal-Wallist test is the generalization of the two sample Wilcoxon (Mann-Whitney) test.We’re given the samples . (n groups, each containing p random variables (threatments)). In this case the **ordinal** samples are completely **independent** from each other. And the null hypothesis claims that they are all coming from the same distribution (aka they have the same median).

H0:

First order the concatenated samples and define the rank numbers. The concatenated sample will contain N element, where:

is the sum of the ranks of the random variables in the ith sample. Our final test statistics will be:

~~If the amount of samples is low, we use the Kruskal-Wallis table to find the critical values~~, otherwise we know that T follows a distribution with p-1 degrees of freedom.

### Exact tests introduction

In case of non-parametric tests we only know the asymptotical CDF and by that we can make huge errors on low sample count. Exact tests try to tackle this problem.

In case of exact test we try to calculate the chance of Type 1 errors with combinatorical methods ().

### Fisher’s exact test

If we have too little samples to do the test of independence, we prefer proving independence with manual probability calculation, mostly with hypergeometric distribution.

Gives lower (thus better) significance level than test and helps us avoiding Type 1 errors.

### Binomial test

We have X1, X2, ..., Xn samples all of them having the indicator distribution with p probability (parameter). This type of exact test is useful when it is costy to make samples (check the quality of ammunition for example).

The null hypothesis (H0) says that:

We sum up how many times our events happened.

We check the binomial table for the critical value corresponding to the significance level . On large samples (30+) we can use z-test (u-test):

T is approximately a standard normal distribution. We can find its critical values by taking a look at the proper CDF. Also, as an alternative solution we can calculate the p-value (getting a result this extreme) and compare it with the significance level. If the p-value is less then the significance level then we reject the null hypothesis.

## Regression Analysis

If we find out that two variables are dependent from each other then we’d like to measure the bound between them. First time with a number, then describing it with a function.

By applying Cauchy-Bunyakovszky-Schwarz\* inequality ():

\*

If X and Y are independent the correlation is 0 but the reversed case doesn’t hold.

If corr(X, Y) is 1 or -1, then the connection between X and Y is linear:

If we have the (X1, Y1), (X2, Y2), ..., (Xn, Yn) statistical samples from the X, Y random variable we can define the empirical covariance and correlation:

If we get a non-zero correlation from the data first we need to guarantee that is not by randomness. So we make up a null hypothesis claiming that the correlation was 0. And test the following statistics:

Which is Student (t) distributed with n-2 degrees of freedom.

The goal of regression: We have a random variable Y and we would like to approximate it by a function of the X1, X2, ..., Xp variables. By approximation we mean an f function for which the following expression is minimal:

f is coming from a family of functions (linear, quadratic, logarithmic, exponential).

linear regression between X and Y:

The minimum of the h function can be at that point where the gradient is the zero vector:

Solving this we get that the regressor line is:

The task can be explained the following way too:

We are looking for a and b such that the expected value of the squared error term () is minimal:

In practice we have the samples (X1, Y1), (X2, Y2), ..., (Xn, Yn) from the X and Y random variables. Let:

is coming from an specific distribution what is independent from every has 0 mean and standard deviation. By constructing a summed error function (loss function) we can describe our new goal, which is to minimize:

Quality measures:

Coefficient of determination:

Coefficient of determination will qualify how good the linear regression explains data. It measures how good we can guess the value of knowing compared to our naive guess which would be the average of the Y-s we got so far. If this coefficient is 0.95 then the explains 95% of the ’s value.

SST (total sum of squares)

SSE (sum of squares of errors / residuals)

SSR (sum of squares of regression)

**SST** = **SSR** + **SSE**

**Theorem**: The parameters a, b are approximated by and in an unbiased manner.

**Theorem** (Gauss-Markov): The approximations for a,b we get from the least squares method are the most efficient ones (aka they have to lowest standard deviation)

**Theorem**: An approximation for the error’s variance:

If we assume that the error is normally distributed then we can estimate the standard errors of parameters:

By knowing the empirical variances we can define a confidence interval for estimating a,b:

The same way we can define a confidence interval around using :

We’d like to ensure that between X and Y there’s an actual bound so we hypothesis test and on a proper significance level:

H0 claims that .

With the same approach we can accept or reject the null hypothesis for .

A special case when we test the hypothesis claiming that . If we fail to reject this one that would mean that there are no linear connection between X and Y.

H0 claims that .

T is Fisher distributed with parameters 1 and (n-2).

Quite often our data does not follow a linear trend. So the function f is not neccessarily an aX+b type function but can be:

* polynomial:
* exponential I:
* exponential II:

Some of these function families can be converted into the previously introduced linear kind of function:

So if we name the random variable ln(Y) as Z we got a classic linear regression task between Z~X and a, ln(b) can be found.

### Multivariable linear regression

Y is the dependent variable (eredményváltozó).

**Dependent variable:** Upon manipulating the X independent variables we expect change in the value of the dependent variable. (effect)

**Independent variable:** The variable is stable and unaffected by other independent variables. (cause, predictors)

are the independent (**uncorrelated**) variables (magyarázó változók).

The model:

Given the data

Here is the ith attribute in the jth row.

Where **X** is a sized matrix where the first column is filled with ones.

We’d like to define -s such that the sum of -s are minimal. The errors are coming from various sources: missed independent variable, redundant independent variable, wrong function family, measurement errors, random unexplainable effects. The error can be considered normally distributed.

The loss function is:

Can be computed that it will have a minimum at:

We can determines which independent variable had the greatest contribution (needs abs) to the value of the dependent variable by normalizing :

Where is the uncorrected sd of Y-s and is the uncorrected sd of -s (standard deviation of the ith attribute/column).

Coefficient of determination():

SST (total sum of squares)

SSE (sum of squares of errors / residuals)

SSR (sum of squares of regression)

**SST** = **SSR** + **SSE**

Regular coefficient of determination would be close to one when adding a lot of extra unneeded independent variables. By punishing this kind of addition we get a more correct measure which is called the adjusted coefficient of determination:

This is less then or equal with 1, but might go below 0.

Before utilizing the result, we’d like to check whether the a factors can be considered different than 0:

We use a global ANOVA to accept or reject the null hypothesis.

T follows a Fisher distribution with parameters k, (n-k-1).

If it turns out that our model is not a null model (so we previously rejected ), then we can one by one check the relevance of the variables:

This time T follows a Student (t) distribution with n-k degrees of freedom.

The variance of errors:

The error of parameters:

### Model building

How do we decide which independent variables should we add to the mode?

SPSS and statistical softwares use partial F-test in which we check whether introducing a new (pth) independent variable increases the coefficient of determination or not.

Out test statistics:

T is Fisher distributed with parameters 1, n-p-1.

Automatic model building techniques:

* ALL VAR. ENTERED: We enter all variables in a block.
* FORWARD SELECTION: The variables added sequentially depending on a criterion (the one with the lowest rejected significance level under PIN).
* BACKWARD ELIMINATION: We enter all variables and then sequentially erase variables (the one with the highest rejected significance level over POUT)
* STEPWISE SELECTION: A method that both use PIN and POUT.

Multicollinearity:

Unwanted linear connection between independent variables.

Quality indicators of multicollinearity:

**Tolerance**: Measures how tightly the ith independent variable is defined by others. A tolerance close to zero is bad, it would mean that there is an association between the variables in a form of a function. For each independent variable we construct this function like we previously did by finding the parameters that minimizes the loss funcion. For every regression made we calculate the coefficient of determination (denoted as for the regression of ). The value of tolerance is 1-. Tolerance above 0.1 is acceptable.

**Variance inflation factor (VIF)**: This measure is the reciprocal of Tolerance. The ideal value of VIF is 1 but until 10 it can be accepted.

**Condition index (CI)**: A statistics derived from the correlation matrix of the s. CI is the square root of the ratio of the largest and the smallest eigenvalue. CI > 15 means a strong correlation so C < 15 is acceptable only.

### Outlier detection

Removing outliers are as important as removing unneeded s. To detect outliers we computer the so-called levarage matrix the following way:

The diagonal elements () defines the strongness of the ith data point’s impact on the regressor’s estimation. The ith data point is casual if and significant if .

The state of the data point depending on :

* > 0.5: Outlier
* > 0.2: Suspicious and risky to involve in the loss function
* < 0.2: Can be added to our analysis with no worried

### Binary logistic regression

We use logistic regression to anticipate an event that happend with p probablity. Binary regression assigns 0 to probablity values and 1 to the ones which are . This is equalent if we examine the sign of the expression (negative 0, positive 1).

## Data reduction

Helps reducting the size of the data matrix, thus increasing the computational efficiency. To achieve this we might introduce new hidden variables assembled from various, somehow dependent attributes. We do this a way that our statistical explorations will be still valid for the underlying population. Data reduction can help eliminating columns (attributes/variables) as well as rows (cases).

### Factor analysis

We have large amount of dependent variables containing superflous (and redundant) information. We’d like to represent the connection with low number of uncorrelated factor variables.

Correlation matrix:

Anti image matrix:

The element other than the diagonal show how independent two variables’ variance are. The lower, the better. The diagonal elements (MSA) show how much the variance of a random variable is determined by the others. The higher, the better as we couldn’t compress independent enough variables.

Kaiser-Meyer-Olkin (KMO) value:

A complex measure made up from empirical correlations and empirical partial correlations which tells how much the variables depend from each other. This has to be over a limit (0.5, but rather 0.8) to even consider using factor analysis or PCA.

Bartlett test of sphericity:

The null hypothesis will claim that the correlation matrix can be considered the identity matrix. We reject this hypothesis if the value of the test statistics is large so the significance level for approval is super low. If we cannot reject H0, then we should forget about using factor analysis or PCA once again.

model of factor analysis:

We have X1, X2, ..., Xn variables put into the observation (X) matrix.

A: Loading matrix (p, k)

F: Factor matrix (k, n)

U: Unique factor vector (error term) (p, n)

### PCA (principal component analysis)

An algorith that will determine the component of the factor analysis model.

Let be the covariance matrix of X1, X2, ..., Xn. By knowing that is symmetric we can spectrally decompose it.

are the eigenvectors and is an orthonormal basis. In this case will be the X’s principal component where U is a matrix of column vectors. Y[1] is the direction with the largest variance, then comes Y[2], etc.

**Theorem:** Dimensionality reduction done with PCA algorithm minimizes the information loss.

Rotation of the load matrix:

QUARTIMAX: The number of variables will the lowest possible.

VARIMAX: The number of variables for which we have heavily weighted factors will be the low.

### MDS (multi dimensional scaling)

When doing PCA we assumed that our variables are not categorical or ordinal.

In case of MDS the goal is to make a geometrical representation of similarities that would display the hierarchy between the variables the most obvious and clear way.

So we have data points in the n-dimensional space and we’d like to find point in 2D or 3D that has matching distance metrics than the given data points. In short: we’d like to represent out data in low dimensional space (possibly 3D) with minimal distortion in the representation.

We have **n** data points: each of the has **p** attributes (can be said that we have n point in the p dimensional space). We’d like a lower dimension representation.

Distance matrix:

d is the distance function in the p-dimensional space.

We want points in a lower dimensional space for which:

is low. Here, is the distance function in the lower dimensional space.

To measure the quality of the MDS procedure we have 3 metrics.

**s-stress:**

**stess:**

Stresses must be low for a good representation.

**RSQ:**

The squared correlation of D matrix and the representation’s distance matrix. If large then all the variances can be respresented pretty well.

### Discrimininant analysis

We have the k datapoint already clustered (each datapoint has p attribute). We’d like to linearly separate (linear combination of the attributes) those cluesters with functions. Doing this can help classify new, not yet seen, datapoints.

## Time series data

The random variables are indexed by timesteps.

Can be used for prediction or to complement missing timesteps.

The CDF becomes 2 parametered:

Autocovariance & autocorrelation:

Autocovariance & autocorrelation function (ACF):

A time series is autouncorrelated (has no memory) if:

Partial autocorrelation function (PACF): Correlation between without knowing the variables of the timesteps in between.

First task what we should do when analysing time series is to prove that the time series has actually autocorrelating elements, so there is explainable value difference depending on the timestep.

Define as the indicator indicating 1 if , 0 otherwise. If -s are independent then the measure standardized is standard normally distributed:

The null hypothesis says that the is indeed , so we all the X-es are indepenedent, so there are no time series to analyze. It would be preferred if we could just reject this null hypothesis. (**Váltakozásmódszer,** csúcsmódszer, előjelmódszer /\* TODO: need english translations! \*/)

### Models for time series

**Deterministic model**:

The time series can be fully described by a function of t, the randomness is only present at the error term.

Decomposition model: The time series is decomposed to different components which are either summed up or multiplied together to reconstruct the time series.

* Additive:
* Multiplicative:

T, C, S and I are the trend, cyclical, seasonal and irregular (noise) component.

trend: A long-term impact on the whole timeline.

seasonality: Short-term periodical effect (increasing beer sales during summer).

cyclicity: Long-term periodical effect that happens irregularly (economical crisis).

irregularity (noise): has 0/1 expected value and low standard deviation.

**trend analysis**:

The goal is to find the trend an subtract it from the time series.

Moving average:

We define the trend with the help of the moving average of the original time series. Suppose that our time series it T long and our sampling window is k wide. The trend will be:

By doing averaging we get rid of the seasonal fluctuation as well as the irregularity.

Linear trend:

We look for a trend in the following format:

With the method of least square we are trying to find the value of a and b for which the:

expression is minimal (similarly to regression).

**seasonality:**

Assume we already found the trend and remove from the time series, so we have a simplified dataset where there are no trend influence. The time series can be written formally as follows:

Where d is the seasonal component an X is a stationary random influence in the given year and month and is:

Here n is the number of years and m is the number of months or any other hierarchical metric. To calculate the seasonal component d (seasonal index) we first define the seasonal average (for every month we compute the averages of yearly data):

**ciclycity:**

Advanced! TODO

**Stochastic model:**

The time series is not only the function of time but the previous timesteps has major impact on the wheregoings. Randomness has reponsibility forming the time series, randomness is not only noise. So while the deterministic model can be written as:

The stochastic model has recursion in its formal definition and the variance is not fixed but will increase along with the time:

// TODO ARMA ARIMA

# Azure

## Cloud computing in general

Let the user borrow a PC from the provider’s data center. The user can customize the services he or she wants to use and don’t have to worry about upkeeping (maintaining) that PC like we should if it was our own.

Services: computing power, storage, making VMs, AI/ML

Azure uses pay-as-you-go type of billing, you only pay for the computing power you use.

## IaaS, PaaS, SaaS

Infrastructure as a service: cloud provider will keep the hw up do take, but OS and network maintenance is the tenant’s responsibility. (Azure VMs)

Platform as a service: cloud provider manages the VMs and networking, the tenant daploys their application in this manages hosting environment

Software as a service: cloud provider manages all the aspects of the application environment (VM, networking, storage, applications). The tenants only have to provide their data. (Office 365)

## Cloud models

Public: services in public clouds are offered over the public internet for anyone who purchases them. (servers & storage)

Private: services that can be only exclusively used by a specific business or organization. A private cloud is usually located at the organization’s on-site datacenter.

Hybrid: allows data to be shared in public and private cloud

## Structured and unstructured data

Structured: Structure is defined at design time, mainly tables. Structured systems react slowly to changes due to the change needed on a data requirement update (inserting new column can result bothersome bulk updates. (Microsoft SQL Server, Azure SQL Database and Azure SQL Data Warehouse)

Unstructured: Nonrelational/semistructural systems (NoSQL). Data is typically loaded in raw format and the structure is defined only when the data is read. Unstructured systems have various types:

1. Key-value: KV pairs stored in a tabular structure.
2. Document: Stores documents that are tagged with metadata to help queries.
3. Graph: Builds relationships by using a structure of nodes and edges.
4. Column: Stores data based on columns rather than rows.

## Azure data platform technologies

### Azure Storage

Azure storage is the base storage type in Azure. It can operate as a NoSQL store. Azure storage has 4 configurations all of them are encrypted and utilize the role based access control (RBAC) that can be set up in Azure Resource Manager.

Azure Blob: Cheap but not queriable store for text, and binary data.

Azure Files: Managed file shares for cloud or local deployments.

Azure Queue: A messaging store between app components.

Azure Table: A NoSQL store for schemaless storage of unstructured data.

To ingest data into your system use Azure Data Factory, Storage Explorer, VS or PowerShell.

### Azure Data Lake Storage Gen2

Data Lake is designed to store massive amounts of data for big data analytics.

### Azure Cosmos DB

Use Cosmos DB when you need a NoSQL db of one of the supported API model (MongoDB, Cassandra, Gremlin, Table) at a global scale with low latency.

Ingestion can be achieved via Azure Data Factory.

### Azure SQL Database

Supports structured (relational) and unstructured formats (spatial data, XML). Can be scaled to have global coverage.

### Azure Synapse Analytics

Cloud based data platform that brings together enterprise data warehousing and big data analytics. It excels at processiong large volume of data. Uses PolyBase and TSQL.

### Azure Stream Analytics

Used by data engineers to process streaming data (data arriving at realtime like IoT sensor information, web logs) and do real-time analysis on it. Azure Event Hubs provides big-data streaming services and is integrated into Databricks, Stream Analytics, Azure Data Lake Storage and HDInsight.

### Azure HDInsight

Provides technologies to help ingest, process and analyze big data. It includes Apache Hadoop/Spark/Kafka and Hbase.

**Kafka**: Used to compse data pipelines. Allows users to publish or subscribe to real-time data streams.

**Storm**: Used for streaming analytics distributedly.

**HBase**: A NoSQL database built on Hadoop used for search engines.

**Hadoop**: stores data in a file system, whereas Spark stores data in memory making the latter at least 100 times faster.

In Hadoop use Java and Python to process big data. Mapper consumes and analyzes input data. It then emits tuples that Reducer can analyze. Reducer runs summaries to create smaller result sets. (MapReduce)

Spark processes streams by using Spark Streaming.

In Hadoop query with Pig and HiveQL, in Spark use Spark SQL.

### Databricks, Data Factory, Data Catalog

Databricks is a serverless platform. It provides an interactive workspace for Spark-based applications. In Databricks notebooks you can use R, Python, Scale and SQL.

Data Factory is a cloud-integration service. It orchestrates the movement of data between various data stores. Use Data Factory to create and schedule data-driven workflows (pipelines).

Data Catalog can help managing metadata and annotations. It excels at storing documentation.

## Azure for data engineers (ETL, ELT)

E: Extract

T: Transform

L: Load

In Azure the standard is ELT. For standard ETL the most popular tool to be used is Azure Data Factory. In ETL the data’s structure is defined before the loading. On the contrary ELT allows us to define the structure right before transformation so it is possible to use the same data in multiple streams and the data can be stored (loaded) in its original format.

**Extraction** can happen from various sources which include Cosmos DB, Data Lake, Azure Files, Azure Blob storage. During extraction data engineers defines data (by query or file selection) and its source (Resource group/subscription).

**Transformation** operations can include splitting, combining, deriving, adding, removing or pivoting columns.

**Load** means defining a destination (including not only the place but the format)

# Big Data

Large scale of data. Relational databases are not intended to store Big Data as a single inserted column can cause the need of bulk updates, which would cost millions of dollars.

Big Data is usually analysed/processed through computer clusters as it can offer higher availability, more resources and easy scaleability. Big data can be compressed too to save storage but most often we are focusing on implementing real time solutions.

Big data problem (5V): Volume (large amounts of data), Velocity (the data is generated rapidly), Variety (many structures of data, structured/unstructured/semi-structured), Veracity (can be inconsistencies in the data), Value (raw data has no value as it has to be processed first to give business insights).

Data collection: streaming/ingesting data from enteprise apps/iot/socal media

Data storage: real time access to data streams, can keep data locally/on-cloud possibly in a geographically distributed manner.

Data exploration: designing the data product with access to the original data with its original format

Data governance: data catalogization, data access controll, identifying lineage

Data product: data product delivery (Tableu viz, etc.), enterprise integration

## Data engineering workflow

Ingest: Taking raw data from various sources and adding it to a data platform. (Azure Data Factory, AWS Glue, Google Transfer, Nifi, Streamset or Apache Sqoop)

Persist: the ingestion process hands over the data to a component that manages storage (Apache Hadoop’s HDFS, Amazon S3, Azure Blob / Data Lake, Google Storage). These components allow to store large volumes across multiple nodes in the cluster.

Prepare: The system can begin data processing bringing the information to the surface depending on what type of insight is in desire. Batch processing is a widespread technique for processing large volume of data.

## Data stores

Data lake: large collections of data (raw and curated) in a scalable storage good for analytics, but not ideal for ETL/ELT.

Data warehouse: in the data warehouse data is cleansed, transformed, categorized and tagged.

Data mart: dm is a subset of a data warehouse created for one specific subject area that need to be accessed in short amount of time.

Data vault: dv is a model to solve a complete business problem with changing requirements

## Hadoop

A designated tool for storing and processing big data effectively. Can handle multiple types of structured and unstructured data.

Hadoop includes 2 main systems:

* HDFS (Hadoop distributed file system), which is a storage system spread over multiple machines. The smallest volume HDFS can write or read is named a block which is -by default- 128MB.
* MapReduce engine, which filters/sorts/uses the database in some way. Data from HDFS is transferred to MapReduce for further processing (this can be parallelized).

Hadoop architecture:

**Name node:** Master server that exists in the HDFS cluster. It manages the file system namespace by executing operations like opening/closing/renaming. NameNode regulates the access to the files by clients. Client apps can communicate to NameNode to manage a file and NameNode will respond back with a successful request returning the list of relevant DataNode servers where the data lives (according to FSImage). System metadata changes are stored in a transaction log in the NameNode’s local FS (editlog). NameNode is a single point of failure thus secondary NameNode was interoduced that stores a copy of the system metadata and editlog.

**Data node:** Manages the state of an HDFS node and interacts with block. Can perform jobs like semantic/language analysis, statistical or ML tasks and also jobs like clustering, indexing, data importing/exporting. On start every DataNode connects to the NameNode. The DataNode verifies the block replicas in its ownership by sending a block report, and after that the DataNode will send a heartbeat to the NameNode every 3 seconds to confirm that the DataNode is operating and the block replicas it hosts are available.

**YARN:** Introduced in 2.0 YARN is a large-scale OS used for Big Data processing. Its reponsibility is to manage resources.

Hive data warehouse software enables reading, writing, and managing large datasets in distributed storage. Using the Hive query language (HiveQL), which is very similar to SQL, queries are converted into a series of jobs that execute on a Hadoop cluster through MapReduce or Apache Spark.

## Spark

Apache Spark is a general engine for large-scale data processing (cluster-computing framework). Spark has significant increase in performance and speed compared to its predecessor, MapReduce.

Comparing Spark and Hadoop is not really possible as Hadoop is an ecosystem. Spark has no distributed file system but can use Hadoop’s HDFS.

### Spark, MapReduce comparison

MapReduce workes sequentially (batch processing exclusively), so it read from a cluster, performed an operation and wrote back to a cluster (hard disk) Spark works in the memory unless the data cannot fit in. In this case it can just use the hard disk.

Spark is made in Scala whereas MapReduce is programmed in Java.

Spark jobs takes less time to code, with lower complexity.

### SparkShell & SparkApplication

SparkShell allows experimentally manipulate data. SparkApplication on the other hand is an independent program (like a job) running.

Spark application:

**Spark driver**: (driver process) SD is JVM that hosts SparkContext for a Spark application. Spark driver splits an application in tasks and schedules them to run on executors. The Spark driver also hosts a WebUI.

**Executor**: executes the code assigned to it by the driver and reports the computation state back to the driver node. Executors can be identified by their id and a hostname they run on.

**RDD (Resilient distributed dataset)**: It is the Spark representation of data. RDD is an immutable, partitioned collection of records.

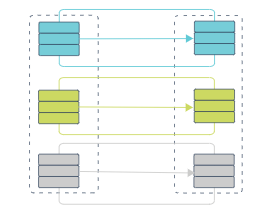
**Stage**: physical unit of execution (set of parallel tasks, one task per RDD partition).

**Job**: consists of a set of tasks arranged in stages

**Task**: (command) is the smallest unit of execution launched to compute an RDD partition.

### Job scheduling

RDD operations have two types: transformations and actions. A transformation produces new RDDs from the existing ones. When RDDs need to be converted into an actual dataset actions are used. Transformations are lazy as it does not execute immediately but rather arranged in a DAG.

Narrow transformation: I and O stays on the same partition without needing to move data. (examples are: map(func); flatMap(func); filter(func); mapPartition(func); mapPartitionWithIndex(func); union(dataset); zip(dataset); zipWithIndex(); zipWithUniqueId().)

**map(func):** returns a new distributed dataset by passing each element of the original dataset through a function.

val z = sc.parallelize(1 to 3).map(i => i\*i) // 1, 4, 9

**filter(func):** returns a new distributed dataset by selecting the element on which a function returns true.

val z = sc.parallelize(1 to 3).filter(i => i % 2 == 1) // 1, 3

**union(func):** returns a new distributed dataset with the element of x and y in the same partition.

val x = sc.parallelize(1 to 3)

val y = sc.parallelize(3 to 5)

val z = x.union(y) // 1, 2, 3, 3, 4, 5

**flatMap(func):** similar to map but you’ll get a flattened version of the result located in the same partition

val x = sc.parallelize(1 to 3, 2) // 1, 2, 3 elements in 2 partitions

val z = x.flatMap(x => List(x,x)) // map would result [1,1] [2,2] [3,3]

// flatMap will result 1, 1, 2, 2, 3, 3

**mapPartitions(func):** similar to map but it runs on each partition of an RDD instead of every element.

val x = sc.parallelize(1 to 3, 2) // 1, 2, 3 partitioned as (1, 2) (3)

val z = x.mapPartitions(p => List(p.next).iterator)

// 1, 3 as these are the first elements of the partitions

**mapPartitionsWithIndex(func):** similar to mapPartitions but you can use the index of the partition for the mapping.

val x = sc.parallelize(1 to 3, 2)

val z = x.mapPartitionsWithIndex( (index, iter) => iter.toList.map(x => index + "," + x).iterator)

// 0, 1, 0, 2, 1, 3 P0 -> 1, P0 -> 2, P1 -> 3

**zip(dataset):** Creates a hashmap where keys will be **x** and values will be **y**.

val x = sc.parallelize('A' to 'C',2)

val y = sc.parallelize(1 to 3, 2)

val z = x.zip(y) // (A,1), (B,2), (C,3)

**zipWithIndex():** Zips each RDD items with its index. Index is independent on the partition.

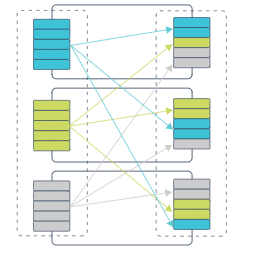
val x = sc.parallelize('A' to 'C',2)

val z = x.zipWithIndex() // (A,0), (B,1), (C,2)

**zipWithUniqueId():** Zips the RDD items with a unique index. Unique index is computed with the *i = k + (0, 1, 2, …)n* formula where *k* is the index of the partition the element resides and *n*is the total numbers of partitions. The integer is depending on the order of the item in its partition.

val x = sc.parallelize('A' to 'C',2)

val z = x.zipWithUniqueId() // (A,0), (B,1), (C,3)

Wide transformation: Data shuffling is needed as data from other partitions is required. (examples are: intersection(dataset) distinct([numTasks]) groupByKey([numTasks]) reduceByKey(func, [numTasks]) sortByKey([ascending], [numTasks]) join(dataset, [num) coalesce(numTasks) cartesian(dataset) repartition(numTasks))

**intersection(dataset):** returns a new RDD with the intersecting elements. Intersection causes data shuffling (repartitioning between nodes) so the order of the output is not always clear.

val x = sc.parallelize(1 to 5) // 1, 2, 3, 4, 5

val y = sc.parallelize(3 to 6, 2) // (3, 4), (5, 6)

val z = x.intersection(y) // 4, 3, 5

**distinct([numTasks]):** returns a new RDD with only te disting elements of an RDD. The ordering can change in this case too.

val x = sc.parallelize(Array(1, 2, 1, 3, 2), 2) // (1, 2), (1, 3, 2)

val z = x.distinct() // 1, 3, 2

**coalesce(numPartitions):** reduces the number of partitions in the RDD to the given one.

val x = sc.parallelize(1 to 6, 4) // [1], [2, 3], [4], [5, 6]

val z = x.coalesce(2) // [1, 2, 3], [4, 5, 6]

**repartition(numPartitions):** changes the number of partition in the RDD to the given one.

val x = sc.parallelize(1 to 6, 4) // [1], [2, 3], [4], [5, 6]

val z = x.repartition(2) // [1, 2, 4, 5], [3, 6]

Coalesce will do nothing when trying to increase the number of partitions. The number of partitions can be queried with .getNumPartitions().

Actions (count(); collect(); take(n); top(n); countByValue(); reduce(func); fold(zeroValue, func); aggregate(zeroValue, seqOp, combOp); foreach(func); saveAsTextFile(path); saveAsSequenceFile(path); saveAsObjectFile(path)):

**take(n):** take will return the first n elements of an RDD starting with its zeroth partition.

val x = sc.parallelize(1 to 5).take(3) // 1, 2, 3

**top(n):** top will return the top n largest elements of an RDD.

val x = sc.parallelize(1 to 5).top(3) // 5, 4, 3

**reduce(func):** aggregates the element of the RDD by the passed in function. The function keeps track of an accumulator value which is the result of the latest function call.

val x = sc.parallelize(1 to 4).reduce((acc, item) => acc + item) // 1-3-6-10 -> x=10

**fold(zeroValue, func):** similar to reduce but you can predefine a default value for the accumulator. An empty RDD thus won’t make the action crash.

val x = sc.parallelize(1 to 4).filter(\_>10).fold(0)((acc, item) => acc + item)

// [1, 2, 3, 4] filtered to [ ], and then each element gets added

// as there are no element the zeroValue will be returned instead

**aggregate(zeroValue, seqOp, combOp):** We assign zeroValue to the accumulator then we run the sequence operator (seqOp) a number of times. combOp will define the way of combining result on different partitions after finishing seqOp-s on them.

val x = sc.parallelize(1 to 4).aggregate((0, 0)) (

(acc, item) => (acc.\_1 + item, acc.\_2 + 1),

(acc1, acc2) => (acc1.\_1 + acc2.\_1, acc1.\_2 + acc2.\_2)

)

// 1, 2, 3, 4

// (0 + 1, 0 + 1) >> (1 + 2, 1 + 1) >> (3 + 3, 2 + 1) >> (6 + 4, 3 + 1)

// (10,4)

**foreach(func):** runs a specific function on each row of the dataset.

val x = sc.parallelize(1 to 4, 2)

x.foreach(println)

**saveAsTextFile(path):** writes the dataset to a set of text file (as many partitions we have).

val x = sc.parallelize(1 to 4, 2)

x.saveAsTextFile("C:\\spark\\numbers.txt")

**saveAsSequenceFile(path):** writes the dataset as a Hadoop sequence file in a given path on the local computer or in the HDFS.

val x = sc.parallelize(1 to 4, 2)

x.map(i => (i, i\*i)).saveAsSequenceFile("C:\\spark\\numbers.sq")

**saveAsObjectFile(path):** serializes the dataset using Java serialization.

val x = sc.parallelize(1 to 4, 2)

x.saveAsObjectFile("C:\\spark\\numbers.obj")

Transformation on paired RDDs: (Paired RDD: key-value pairs like (word) => (word, count))

**mapValues(func):** applies a function of every value of the paired RDD items without changing their keys.

val x = sc.parallelize(Array(('A', 1), ('B', 2), ('C', 3)))

val z = x.mapValues(i => i\*i) // (A,1), (B,4), (C,9)

**flatMapValues(func):** applies the flatMap function to the values of the paired RDD and assigns the old key to every returned element.

val x = sc.parallelize(Array(('A', 1), ('B', 2), ('C', 3)))

val z = x.flatMapValues(i => List(i,i)) // 1, 1, 2, 2, 3, 3

// (A,1), (A,1), (B,2), (B,2), (C,3), (C,3)

**keys():** returns an RDD of just the keys.

val keys = x.keys

// A, B, C

**values():** returns an RDD of just the values.

val values = x.values

// 1, 2, 3

**groupByKey([numTasks]):** groups values with the same key.

val x = sc.parallelize(Array((‘B', 1), ('B', 2), (‘A', 3) , (‘A’, 4) , (‘B’, 5)))

val z = x.groupByKey() // (B, [1, 2, 5]), (A, [3, 4]))

**reduceByKey(func, [numTasks]):** combines the values (by the reduce function) of the same key.

val x = sc.parallelize(Array((‘B', 1), ('B', 2), (‘A', 3) , (‘A’, 4) , (‘B’, 5)))

val z = x.reduceByKey((acc, item) => acc+item) // (B,8), (A,7)

**aggregateByKey(zeroValue, seqOp, combOp, [numTasks]):** same as aggregate but the aggregation happens for each key individually.

val x = sc.parallelize(Array((‘B', 1), ('B', 2), (‘A', 3) , (‘A’, 4) , (‘B’, 5)))

val z = x.aggregateByKey((0, 0))(

(acc, item) => (acc.\_1 + item, acc.\_2 + 1),

(acc1,acc2) => (acc1.\_1 + acc2.\_1, acc1.\_2 + acc2.\_2)

)

// (B,(8,3)), (A,(7,2))

**sortByKey([ascending], [numTasks]):** return an RDD sorted by its keys.

val x = sc.parallelize(Array((‘B', 1), ('B', 2), (‘A', 3) , (‘A’, 4) , (‘B’, 5)))

val z = x.sortByKey() // (A,3), (A,4), (B,1), (B,2), (B,5)

**subtractByKey(otherDataset, [numTasks]):** removes element with a key present in the other RDD.

val x = sc.parallelize(Array(('C', 1), ('B', 2), ('A', 3), ('A', 4)))

val y = sc.parallelize(Array(('A', 8), ('B', 7), ('A', 6), ('D', 5)))

val z = x.subtractByKey(y) // (C,1)

**cogroup(otherDataset, [numTasks]):** groups data along RDDs sharing the same keys.

val x = sc.parallelize(Array(('C', 1), ('B', 2), ('A', 3), ('A', 4)))

val y = sc.parallelize(Array(('A', 8), ('B', 7), ('A', 6), ('D', 5)))

val z = x.cogroup(y) // (B,([2],[7])), (A,([3, 4],[8, 6])), (C,([1],[])), (D,([],[5]))

**join(otherDataset, [numTasks]):** join performs and inner join between two RDDs. A: (K, V) B: (K, W) A join B will return a dataset of every (K, (V, W)) pairs.

val x = sc.parallelize(Array(('C', 1), ('B', 2), ('A', 3), ('A', 4)))

val y = sc.parallelize(Array(('A', 8), ('B', 7), ('A', 6), ('D', 5)))

val z1 = x.join(y) // (A, (3, 8)), ..., (B, (2, 7))

val z2 = x.leftOuterJoin(y)

val z3 = x.rightOuterJoin(y)

Actions on paired RDDs:

**countByKey():** counts the number of each key.

val x = sc.parallelize(Array(('B', 1), ('B', 2), ('A', 3), ('A', 4), ('B', 5)))

val z = x.countByKey() // B -> 3, A -> 2

**collectAsMap():** converts the dataset into a map. For key A 4 is defined last, while for B it is the item 5.

val x = sc.parallelize(Array(('B', 1), ('B', 2), ('A', 3), ('A', 4), ('B', 5)))

val z = x.collectAsMap() // A -> 4, B -> 5

**lookup(key):** return all values associated with the provided key.

val x = sc.parallelize(Array(('B', 1), ('B', 2), ('A', 3), ('A', 4), ('B', 5)))

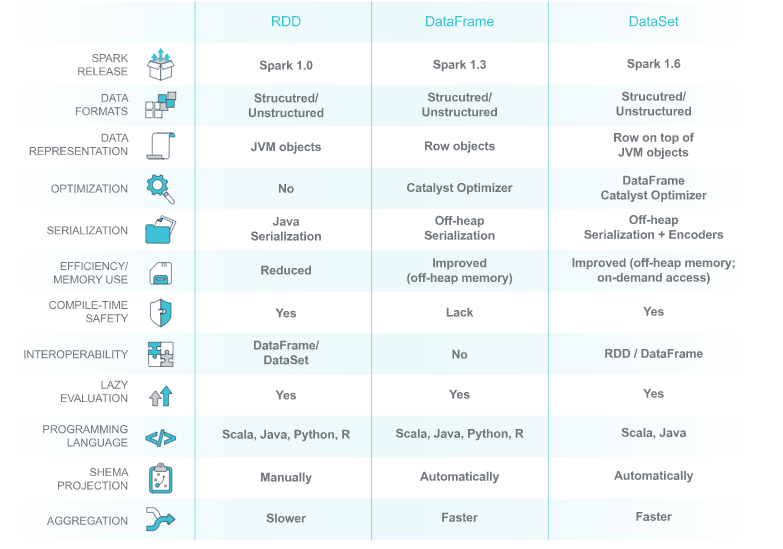
val z = x.lookup('A') // 3, 4

## RDD, DataFrame, DataSet

RDD is a distributed collection of data items spread across clusters.

DataFrame is a distributed collection of data items organized into columns. It is conceptually the same as a table in SQL. Or pandas dataframe in python.

DataSet is a strongly typed immutable collection, which benefits of the Catalyst query optimizer. Code is building a query plan. DataSets has a DataFrame view.



## Spark Streaming

### Data streaming in general

MapReduce excels at batch processing and offers high throughput but it isn’t suitable for non-batched processes, like realtime queries and big data streams. Data streaming is the phenomena of data generated continously from thousand of sources and sent simultaneously in small sizes.

* **Batch processing**: we have to wait until a certain amount of data is accumulared, which means high latency. The incoming data is stored in some storage and periodically some computation is commited on them.
* **Stream processing**: A typical stream application consists of some producers that generate data and some consumers that process these piled up data. Consumers can aggregate data or produce new streams in realtime. There has to be a storage and a processing layer in every streaming system. The **storage layer** is responsible for maintaining data consistency and record ordering in a way that enables real time processing to work effectively. The **processing layer** is responsible for applying computation to the data and then notifying the storage layer to delete the unneeded data (the data we no longer need).

Stream processing advantages: The system can perform computations as soon as new data arrives, we don’t have to wait for data accumulation (uniform processing). Stream processing systems are a good fit for tasks that involves never ending data streams.

### Spark Streaming

Spark Streaming is an extension to the Spark API which enables the developer to create scalable, fault-tolerant, high-throughput live data stream processing system. Data is coming from many sources and can be processed with complex algorithms expressed with map, reduce, join and window. Processed data is output to filesystems, dbs and possibly dashboards.

Text

Description automatically generated

Spark streaming receives live input data and divides it into batches which are then processed by the Spark engine. Spark streaming provides a high-level abstraction of streams called DStream (discretized stream), which is essentially a sequence of RDDs.

Spark streaming is designed to process microbatches (data piled up in a sub-second), so instead of processing records one by one, the data stream gets discretized. The discretized data is buffered to Spark’s worker nodes. Spark tasks are assigned dynamically to nodes unlike in a continous operation model.

* **Block interval**: The incoming data is broken into blocks (by a time measure called block interval, which is 200ms by def. declared by Spark). A partition is allotted to each block.

Diagram

Description automatically generated

* **Batch interval**: Batches are created in a regular time interval. Any data that arrives in that time interval is appended to the batch. After the end of each interval a new RDD is created. Batch interval can be configured by the developer and is typically between 500ms and several seconds.

Graphical user interface, table

Description automatically generated

Receivers: Every input DStream is associated with a receiver object that receives data from the source and stores it in Spark’s memory for later processing. A receiver first splits the data into data blocks which are then replicated on another executors.

Sources:

* Basic: provided in the spark library like: Socket, File and Actor streams.
* Advanced: dependencies needs to be specified for them like: Kafka, Flume, Kinesis, Twitter.
* Custom: customized sources can be developed too.

Spark streaming API:

* Defining input:

val ssc = new StreamContext(sparkContext, Second(1)) #1 s as batch interval

val stream1 = ssc.socketTextStream(”localhost”, portnum)

val words = stream1.flatMap(\_.split(”[\\s](file:///\\s)”))

words.saveAsTextFiles(”...”)

Diagram

Description automatically generated

* Transformations: transformations of DStream leads to another DStream. Transformations are mostly from the RDD API, but there are Streaming specific transformations like lazy evaluation.
* Joining: Can be done between DStreams and RDDs and between two DStreams.
* Windowing: special transformations performed with windows this way we can do computation on multiple RDDs.

val windowedWordsCnt = pairs.reduceByKeyAndWindow( (a,b) => (a + b), windows\_length, sliding\_interval) #Second(...)

Graphical user interface, application

Description automatically generated

* Stateful operations: use in scenarios where we need a state available in each micro batch. In case of calculating the length of a window we must use states. A state has to be provided a certain ID. (???) updateStateByKey, mapWithState.
* Caching: for data that will be accessed various time. Use persist() to force caching.

Delivery semantics:

*The approach of handling message reprocessing upon failure. From weak to strong. Can be adjusted by setting the checkpoint directory and the SparkConf property and thus enable WAL (write-ahead-log).*

* **At-most once**: Each record may or may not be processed. If a record is processed it is only processed once. Data loss can occur. (**Unrealiable receiver**)
* **At-least once**: Each record is guaranteed to be processed but it may get processed multiple times, so data duplications can take place.
* **Exactly-once**: Each record is guaranteed to be processed once and only once. There is neither any data loss nor any data duplication. (**Reliable receiver**)

Kafka integration:

* **Receive based approach:** Receiver recieves the data. The receiver itself is implemented using Kafka Consumer API. The data receiver from Kafka is stored in Spark execuros. WAL has to be enabled in Spark Streaming manually. The approach offers at-least once semantics.

Diagram

Description automatically generated

* **Direct approach (newer):** There are no receivers but instead Kafka is periodically queried form the latest offsets in each (topic, partition) pair. When the jobs to process the data are launched Kafka consumer API is used to read the defined ranges of offsets. WALs are not used so it offers faster processing and exactly-once semantics.

Diagram

Description automatically generated

Checkpointing:

A streaming app must operate 24/7 and thus must be resilient to failures. Hence, Spark Streaming needs to checkpoint just enough intermediate results to restore a job. Checkpoints also allows to save configurations of the StreamingContext, DStreams operations and batch states (metadata). Checkpointing also saved the generated RDDs to a reliable storage.

Spark UI:

* **Input rate**: The frequency of the streamping app receiving data.
* **Scheduling delay**: The time waits for previous batches to complete. Zero delay would mean that the batches are processed as soon as they are created.
* **Processing time**: The time required to complete all streaming jobs of a batch.
* **Total delay**: The time elapsed from submitting to completing all jobs of a batch.

Performance tuning:

The batch interval defines the size of the batch in seconds. The application is **stable** if the processing time is less than the batch interval. In case of processing time consistently exceeding the batch interval, then the processing time needs to be reduced or the batch interval rised.

* **Parallelism in data receiving:** by default each input DStream creates a single receiver. If it is justified we can split a single input data stream into multiple streams. (e.g.: the input data streams receiving two topics of data)
* **Parallelism in data processing:** The concurrency level of distributed reduce operations can be controlled through the sparks.default.parallelism property. Not using enough parallelization will cause under-utilization.
* **Data serialization:** The type of serialization can lift the load from the CPU and memory.
* **Task launching overheads:** If the number of task launched per second is high then the overhead of sending out tasks to slaves may be significant. Running Spark in standalone mode or coarse-grained Mesos mode leads to better task launch times that the fine-grained Mesos mode.
* **Right batch interval**

### Structure Streaming

An API which was introduced in Spark 2.2. It allows the developer to take the same operations that one would perform in classic batch mode Spark API (merely and run them for streaming. Structured Streaming guarantees consistency for both within the engine and in interactions with external systems.

* Consistency: A sequential stream is processed in order everytime.
* Fault tolerance: Is automatically handled (including output sinks).

In Structured Streaming there is no batch concept. Streaming with Structured Streaming hence is more likely towards real streaming. The **received data** is appended to a continously flowing data stream, where each row is processed and the result is updated into an unbounded result table.

A picture containing table

Description automatically generated

Structured Streaming allows to process data based on the event-time (when the data was generated) instead of ingestion time (the time Spark receives the data) solely like in Spark Streaming.

Structure Streaming offers total end-to-end exactly-once semantics.

### Databricks Streaming

Delta Lake’s multihop architecture allows data engineers to build a pipeline that begins with raw data as a SSOT (single source of truth).

**Delta table**: Data in parquet + Delta log + Metastore (optional)

To enable querying tables using SQL we must register the table in the Metastore.

Chart

Description automatically generated

* Bronze table:

*In the bronze table we’d like to record metadata of the ingestion. In bronze table there are no data parsing (raw data stays, just extended with metadata) as that process is very error-prone.*

from pyspark.sql.functions import current\_timestamp, lit

raw\_health\_tracker\_data\_df = raw\_health\_tracker\_data\_df.select(

lit("files.training.databricks.com").alias("datasource"),

current\_timestamp().alias("ingesttime"),

"value",

current\_timestamp().cast("date").alias("ingestdate"))

)

raw\_health\_tracker\_data\_df.select(

"datasource", "ingesttime", "value", col("ingestdate").alias("p\_ingestdate")

)

# the partitioning column is generally prefixed with p\_

.writeStream.format("delta")

.outputMode("append")

.option("checkpointLocation", bronzeCheckpoint)

.partitionBy("p\_ingestdate")

.queryName("write\_raw\_to\_bronze")

.start(bronzePath)

# will start saving the datastream in a DBFS (S3 or ADLS) in parquet format

# creates an additional **delta log that keeps track of all transactions**

# at the end of raw\_to\_bronze notebook it is worth checking the bronze table’s

# schema

assert bronze\_health\_tracker.schema == \_parse\_datatype\_string("""

datasource STRING, // meta

ingesttime TIMESTAMP, // meta

value STRING, // actual data

p\_ingestdate DATE // meta

"""), "Schemas do not match"

print("Assertion passed.")

bronzeDF.withColumn("nested\_json", from\_json("value", json\_schema))

bronzeDF.select(”nested\_json.\*”, ...)

assert silver\_health\_tracker.schema == \_parse\_datatype\_string(

"""

value STRING,

time TIMESTAMP,

name STRING,

device\_id STRING,

steps INTEGER,

day INTEGER,

month INTEGER,

hour INTEGER

""")

silver\_health\_tracker = silver\_health\_tracker.select(

"value",

col("device\_id").cast("integer"),

"steps",

col("time").alias("eventtime"),

"name",

col("time").cast("date").alias("p\_eventdate")

)

# adding parquet to the metastore in an idempotent way (multiple runs will have # the same result)

spark.sql(

"""

DROP TABLE IF EXISTS health\_tracker\_plus\_bronze

"""

)

spark.sql(

f"""

CREATE TABLE health\_tracker\_plus\_bronze

USING DELTA

LOCATION "{bronzePath}"

"""

)

* Silver table

*Silver table has the inner data parsed and cleansed.*

from pyspark.sql.functions import from\_json

json\_schema = "device\_id INTEGER, heartrate DOUBLE, name STRING, time FLOAT"

silver\_health\_tracker = bronzeDF.select(

from\_json(col("value"), json\_schema).alias("nested\_json")

).select("nested\_json.\*")

silver\_health\_tracker = silver\_health\_tracker.select(

"device\_id",

"heartrate",

from\_unixtime("time").cast("timestamp").alias("eventtime"),

"name",

from\_unixtime("time").cast("date").alias("p\_eventdate"),

)

// missing data handling, wrong data cleansing

* Gold table

*Typically aggregations.*

gold\_health\_tracker\_data\_df =(

silverTableReadStream.groupBy("device\_id")

.agg(

mean("heartrate").alias("mean\_heartrate"),

stddev("heartrate").alias("std\_heartrate"),

max("heartrate").alias("max\_heartrate"),

)

)

gold\_health\_tracker\_data\_df.writeStream

.outputMode("complete") #complete instead of append as aggregations are #supposed to be overwritten

.option("checkpointLocation", tableCheckpoint)

.format("delta")

.queryName(tableName)

.start(tablePath)

## Apache Kafka

For effective business multiple systems and application must be able to communicate with each other. Over the years messaging systems became a critical thing odf the big data stack.

The messaging is decoupled. The source does not have to wait until the target receives the message, and the target can do something in before receiving the message. Also, the two end needn’t know each other’s URL.

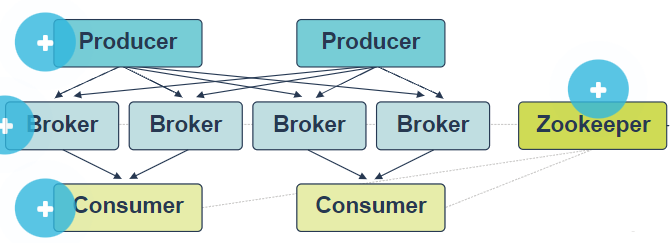
Publish/Subscribe model: a client establishes connection with a broker and send the message there. The broker filters the messages and distributes them to the subscribed clients (the clients that are interested in the content of the message, ALL OF THEM). Subscribed clients also have established connection with the broker. This is the model used by Apache Kafka.

Message querying model: Each message that is produced by the producer is consumed by only one consumer.

Apache Kafka is a high throughput, low-latency platform for handling real time data feeds (events) either in realtime or retrospectively. Also, based on the current architectures it provides easily scalable solutions and has the ability to store and reprocess data in a fault tolerant way. Apache Kafka publishes messages to a topic and those messages get persisted there, so consumer intervention will not remove the message itself offering the possibility to replay messages.

Apache Kafka is client-centric unline traditional message systems as it prefers more complexity and responsibility (fair distribution, replaying messages that had delivery failure) in the client instead of the broker which result an immensely fast broker. Horizontal scaling (adding more partitions) is also allowed, new consumers won’t slow down the system

**Kafka architecture:**

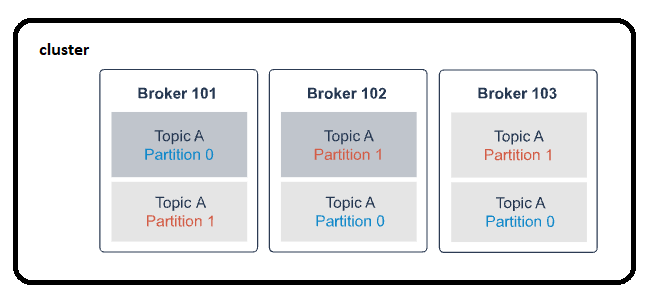


A Kafka **cluster** is made up from one or more brokers (Kafka servers) running Kafka. A consumer push records into a broker, while a consumer pulls out records. It is benefitial to run multiple brokers as it can offer data replication. For each topic the Kafka cluster maintains a commit log with one or more partitions, meaning that **topics are split into partitions** for speed an scalability.

Zookeeper is responsible for updating the topology. It knows when a topic is added or removed or when a broker died or a new got introduced.

Each message appended to a partition by Kafka in an ordered and immutable manner. Each message in a topic has an assigned sequential number that uniquely identifies the message within a partition. The sequence number is called the offset. Topic logs are split into different partitions randomly (round-robin) unless a partition key is specified.

Replications are backup of partitions. If the replication factor is x then Kafka will create x identical replicas of each partitions and place them in the cluster in order to make them accessible for all operations. The replication factor can be at most the number of brokers. In a cluster there’s an elected cluster controller broker which has to assign partitions to brokers and monitor broker failures. A partition is owned by only one broker in the cluster, called the *leader of that partition*.



In the example we can see a system with replication factor of 3. This system can handle losing one or two brokers as the third will still have all the data. The user will experience sligth performance degradation.

Partition ids are the same for every replicas of the same partition. Every partition has one leader replica and multiple follower replicas.

Leader replica: handles every read and write requests of that partition.

Follower replica: their job is to stay up to date with the leader replica and substitute the leader if it dies.

When choosing the new leader we have to select a follower which did not fall behind or died (aka an InSync replica). The follower has a list of the InSync replicas, as a message is only commited if it has been successully copied to all InSync replicas. As the bottleneck here is the slowly syncing replicas, we must get rid of those.

The attribute InSync can be controlled through the **replica.lag.max.messages** (how many messages can an InSync replica can fall behind) and **replica.lag.time.max.ms** attributes.

A replica that loses connection to zookeeper, stops fetching messages of falls behind tremendously is considered an OutOfSync replica. OOS replicas gets back into the InSync status when it reconnects to Zookeeper and catches up with the most recent messages written to the leader.

The **load balancing** can be done by the **producers** as they are the ones who select the partition of a topic to which they publish their messages. It can be done with a message key (all messages with the same key go to the same partition) or keyless, which will induce a round robin distribution.

**Consumers** can be grouped into consumer groups. Each message published to a topic is delivered to one customer instance within each subscribing consumer group. If a consumer joins or leaves the partitions will be rebalanced between the group members.

**Log compaction**: retains the last known value of a record

**Message delivery guarantees:**

* **At most once**: Messages may be lost and never redelivered.A message is pulled one or more times and processed each time. There can be duplicates, but no missing data.
* **At least once:** Messages are never lost and can be redelivered. The message is pulled only once, so it may result missing data on failure, but on the other hand, no duplicates.
* **Exactly once:** Each message is delivered only once. The message is pulled on or more times but processes ones. No duplicates, no missing data.

**Batching & compression:** Messages are piled up into batches which are then compressed. This will speed up the network transfer and allows better disk utilization, however it requires extra CPU cycles from the producer (compression) and the consumer (decompression).

**MirrorMaker:** used for copying clusters (brokers with topics). Not intended to use as a fault-tolerance mechanism.

**Service goals:**

* **Throughput**: The rate at which data is moved to producers to brokers or brokers to consumers.
* **Latency**: is the elapsed time moving messages from end to end (producer to broker to consumer)
* **Durability**: guarantees that messages that have been committed will not be lost. (Kafka and S3 integration)
* **Availability**: refers to the minimization of downtime in case of unexpected/sudden failures.

Service goals can be optimized on a broker level, but there’s a possibility to override topic configurations to give specialized optimization for certain topics. Without overriding the broker configuration applies.

### Optimizazion for throughput

### Optimizazion for latency

### Optimizazion for durability

### Optimizazion for availability

### Kafka monitoring & metrics

Broker metrics:

**Under replicated partitions**: ideally in a cluster the number of InSync replicas is equal to the total number of replicas. If a replica falls behind, we will experience a rise in the number of under replicated partitions.

**Offline partition count**: The number of partitions without a leader. As the leader partition is the only one that can handle read and write requests this metrics being non zero is a serious issue that needs urgent resolution.

**Active controller count**: In a cluster there should be only one controller which is responsible for maintaining the list of partition leaders.

Consumer and producer metrics:

Consumer and producer throughput and lag in producer’s offset (higher) and consumer’s offset (lower).

### Kafka Connect

**Kafka ecosystem**: Kafka Core, Kafka Streams, Kafka Connect, Kafka REST Proxy, Schema registry.

Kafka Core: brokers, topics, logs, partitions, MirrorMaker

Kafka Connect is the connector API for creating reusable producers and consumers. It includes prepackaged Connectors for S3, Hadoop and Cassandra source connectors. Sink connectors are the destination of records.

Kafka Streams: A client library for building apps and microservices, where the input and output data are stored in Kafka clusters. Kafka Streams can transform, aggregate, create & process records from a stream and produces derivative streams (possibly in realtime).

Schema registry: It is a layer for schemas that are used by Kafka as its underlying storage mechanism. (uses AVRO, validates producer and consumer compatibility)

**Kafka connect:** a framework for connection Kafka with external systems such as databases, key-value stores, search indexes and file systems. Connect is used for streaming data between kafka and other systems in a scalable and reliable way. Kafka Connect collects metrics or takes an entire database and puts it into a Kafka Topic.

**Concepts:** *Connectors, Tasks, Workers, Converters, Transforms, Dead Letter Queue.*

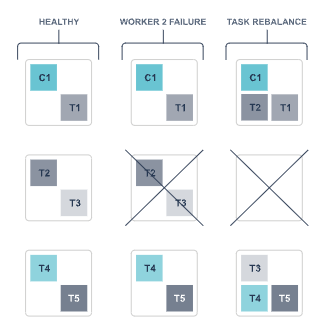
#### Connectors

Define where data should be copied to and from. A connector instance is a logical job that is responsible for managing the copying of data between Kafka and other systems. All the classes that are used by a connector are defined in a connector plugin (preferably use the prewritten ones).

* Source connectors: ingests entire databases and streams table updates to Kafka topics. Can also collect metrics and store those in Kafka topics.
* Sink connectors: delivers data from Kafka topics into services like Elasticsearch and Hadoop.

#### Tasks

Each connector instance coordinates a set of tasks that copy the data. Kafka connect provides built-in parallelism by breaking up a single job into many tasks. Tasks can be started/stopped/restarted at anytime. Workers are the units running those tasks, and on failure the tasks are rebalanced between workers. On failing tasks, however, no rebalancing happens, an as such these tasks need to be manually restarted via REST API.



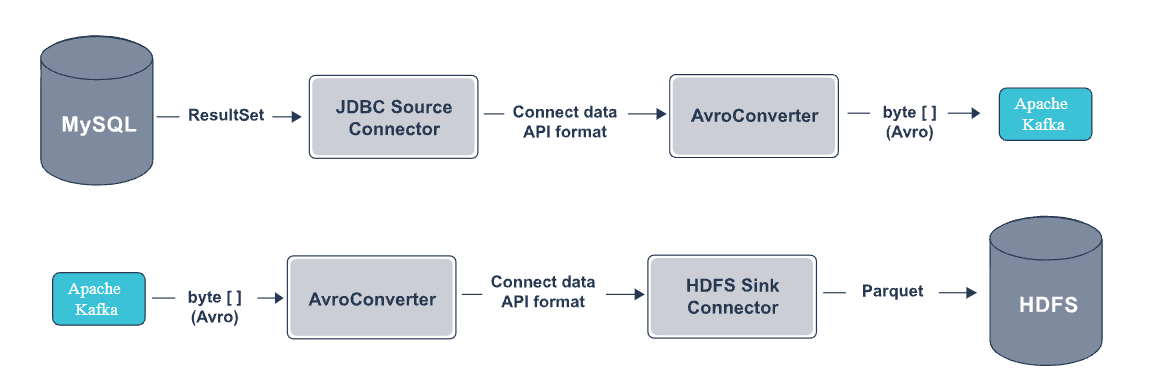
#### Workers

Connectors and task are scheduled to execute in a process. These processes are called workers.

* **Standalone** mode: a single worker is responsible for executing all connectors and tasks. It requires minimal configuration so it is used mostly for trying out new plans or doing smaller jobs. As it is a single process, there is no fault tolerance and no horizontal scalability.
* **Distributed** mode: in this mode you can start multiple workers and they will coordinate the schedule of executing connectors and tasks. Upon adding or removing a worker, the tasks are redistributed between the still living workers.

#### Converters

Converters help tasks to change the format of data from byte to a Connect internal data format (AvroFormatter, etc. or bytearrayconverter for passthrough) and vica versa.



#### Transformers

A function that takes a row as an input and outputs a modified row. Can be chained.

#### Dead letter queue

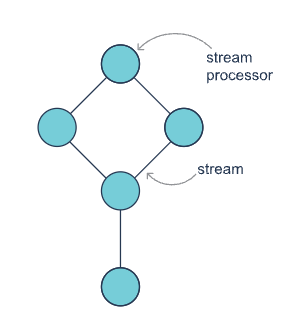
A faulty record cannot be handled by a connector (e.g.: a sink connector expects Avro, but gets Json). An error is handled a way that depends on the ***errors.tolerance*** parameter.

* None: The connector immediately fails on an invalid record, and goes to a failed state. You need to manually review the error via the Worker Log, correct it and restart the connector.
* All: invalid records are ignored and processing continues. Invalid record errors are not written to the Worker Log, hence the presence of errors can be detected by monitoring internal metrics (# of source processed records vs # of sink processed records).

### Kafka Streams

Is a client library for processing and analyzing data stored in Kafka which is either written back to Kafka or sent to an external system. Kafka has high-througput and can work real-time in a record-at-a-time processing format, secure, fault-tolerant, scalable, easily integratable.

**Stream**: unbounded (unknown or infinite size), continously updating dataset. Streams consists of one or more partitions just like Kafka Topics.

**Stream processing app**: any program that makes use of the Streams library. The computational logic is defined by one or more **processor topologies (graph of stream processors(N) and streams(E))**. The stream processing app doesn’t run inside a broker but rather in a separate JVM/Cluster. Launching multiple instances of the same app will cause them to collaborate even through failures. That feature ensures easy scalability and fault-tolerance and no data loss. Developers can define topologies with the low level Processor API or with the high level Kafka Streams DSL.

**Stream processor**: A processing step in the topology that is a step which does transformations on the data (map, filter, aggregation). It receives one record as an input at a time from an upstream processor, applies a transformation on it, and returns one or more output records for the downstream processors.

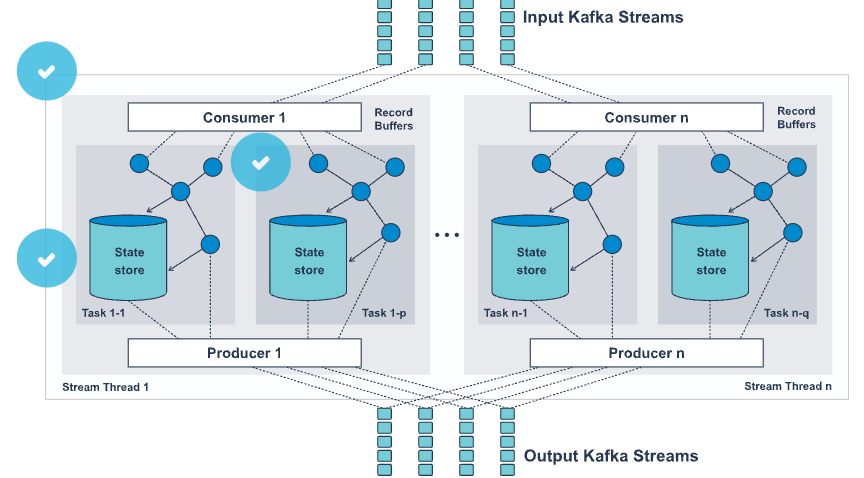
* **Source processor**: does not have an upstream processor, it produces an input stream by consuming records from one or more Kafka topics.
* **Sink processor**: does not have a downstream processor, it sends the processed records to a specified Kafka topic.

Kafka Stream DSL has prewritten map, filter, join and aggregate operations, whereas the Processor API allows the developer to make custom transformations.

Kafka streams assign a timestamp to every record.

* **Event time:** The point in time when the data record was born.
* **Processing time:** The point in time when the data record happens to be processed by a stream processing app (when the record is consumed). It may be some time later than the event time.
* **Ingestion time:** The point in time when a data record is stored in a Kafka broker.

**State**: ability to recall information previously seen and connect it to current information. Often, stream processing apps do not require a state and every message can be processed independently. State, however, neccessary for many operations offered in Kafka Streams DSL like aggregation, grouping or joining. To save states Kafka Streams maintains **state stores**. State stores can be a hash map, KV store or any different convenient data structure. Kafka Streams allow read-only queries (method, external process, etc.) targeting those state stores.



Each **stream partition** is mapped to a **topic partition.**

So the **records in the stream** are mapped to **messages from that topic.**

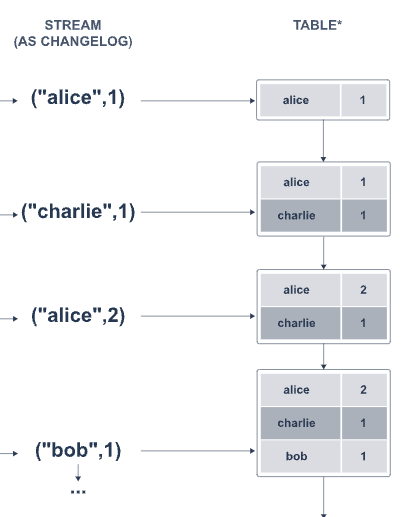
The **keys of the records** determine the partitioning (both in Kafka Streams and Kafka), so it specifies how records routed to specific partitions within topics.

Fault tolerance:

* **Tasks**: Kafka partitions are replicated so stream data is always available. Failing tasks (hosting machine failure) will be restarted automatically on a remaining active instance of the Stream processing application.
* **Local state stores**: for each state store a replicated changelog Kafka topic is saved. The changelog topic is partitioned too, each tasks has its dedicated changelog topic partition.
* **Log compaction**: Old changelog topics are purged to prevent topics to geow indefinitely.

**Interactive queries** helps external apps to directly query the latest results (without first exporting to a DB). It allows real-time monitoring (fraud and trend detection).

#### Kafka Streams DSL (Domain Specific Language)

A stream can be viewed as a table and a table can be viewed as a streams, that the whole point of the duality. Duality allows Interactive Queries.

Stream to table

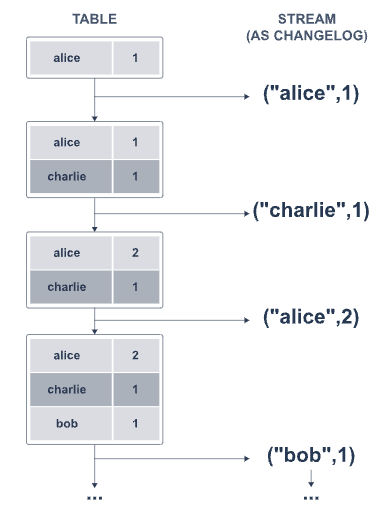


Table to stream

Apache Kafka Streams models duality of stream & tables with the KStream and KTable abstractions.

* **KStream**: abstraction of a record stream. A record is always an insert in case of streams as there is no record-replacing. („alice”, 1), („alice”, 3) the summed number of clicks would be 4, as the second entry for alice is not a replacement just an appended record. In case of KStreams do not enable log compaction as it will lead to losses in the history.
* **KTable**: abstaction of a changelog stream, where each value is an update. A data record is always an upsert (delete is indicated with a null tombstone). („alice”, 1), („alice”, 3) the summed number of clicks would be 3 correctly, as the latter record updates the former. Log compaction needs to be enabled for flawless using of KTables.

GlobalKTable differs from KTable in the data it is being populated with. In case the programmer wants to read a topic in a table then use GlobalKTable to read every message from a topic. Using the common KTable would cause to read in only the local partition.

Stream transformations:

* **Stateless transformation**: does not require the state store to be used.
* **Stateful transformation**: requires the utilization of the state store. (aggregation, joining, windowing, any stateful transformers made with Processor API)

Stateful transformations:

* **Aggregation**: takes on input stream or table and return a single output record. Common examples are sums and counts. The input can be both KTable and KStream but the output stream will always be a KTable so a late arriving record will still be visible on the output because KTable exclusively supports UPSERTing.
  + **Rolling aggregation**: aggregates the values of records by the grouped key.
  + **Windowed aggregation**: aggregates the values of records per window by the grouped key.
* **Joins**: streams and tables can also be joined. Join mergers two input streams and/or tables based on some keys and yields a new stream/table depending the on which kind of models are being joined.
  + **Inner join**: KS-KS, KT-KT, KS-KT, KT-GKT
  + **Left join**: KS-KS, KT-KT, KS-KT, KT-GKT
  + **Outer join**: KS-KS, KT-KT
* **Windowing**: Lets you control how to group records that have the same key for stateful operations such as aggregations or joins into windows.

#### KSQLDB

Greatly reduces the operational complexity and expertise required to build stream processing applications.

Categories:

* **Collections**: durable storage for sequences of events. Streams, Tables like KStream, KTable.
* **Materialized view**: derived representation of streams or tables.
* **Queries**: allows the user to reach the materializations from applications and microservices. **Pull queries** enable you to fetch the current state of the materialized view with low latency. **Push queries** enable you to subscribe to updates of the materialized view. This way the application can react to new information in real time.

Architecture:

* **ksqlDB server**: ksqlDB engine + REST API. Ksql server instances communicate with a kafka cluster. ksqlDB servers can be added dynamically without restarting the application.
* **ksqlDB engine**: executes SQL statements and queries. Each server instance runs one. The engine parses the SQL code and builds the corresponding Kafka Streams topology.
* **REST**: enables communicating with the ksqlDB from the CLI and the confluent control center.
* **ksqlDB CLI**: comman line interface to interact with ksqlDB server instances. Designed to be similar to MySQL and closely related apps.
* **ksqlDB UI (CCCUI)**: Visualized control panel.

## Flow-Based programming

**Flow-Based programming:** A network of blackbox processes which exchange data through externally specified connections.

**Pipeline:** copying from on-premise to cloud, reformatting, joining with other datasets. A pipeline is the sum of all these steps. Each step requires its own tools and softwares.

Pipelines are sensitive to changes in data or schema as changes might require adjustments in the pipeline script. Also, the data engineer must be cautious about failures and data losses.

## NiFi

Apache NiFi is a tool for automating and managing dataflow between systems (DBs, Data Lakes, Data Platform). NiFi helps processing and distributing data and can propagate any content from any source to any destination.

NiFi is ideal for data routing and simple transformations. It can quickly create data flows without coding a single line while providing high throughput and resilience.

NiFi solves the issue of us having to set up multiple ingestion pipelines. NiFi offers out-of-the-box tools to ingest data from several data sources in a secured way kickstarting our Big Data projects. NiFi also helps to set up a geographically distributed architecture and high levels of SLA (security, performance, availability)

**FlowFile (information packet):** an object moving in the system, can contain any type of information in any format. (attribute + content)

**FlowFile processor (black box):** processors can operate on one or more flowfiles in a given unit of work and either commit or rollback. FFPs are doing data routing, transformation and mediation.

**Connection (bounded buffer):** Actual linkage between processor. Connections are queues that can interact at different rates has priorities and can have upper bounds on load.

**Flow controller (scheduler):** Saved the knowledge of how processes connect.

**Process group (subnet):** specific set of processes and connections which can receive data and send data (I and O ports)

Repositories:

* **FlowFile repo**: The metadata of the FlowFiles being processed are held in a HashMap (memory). The metadata holds all the attributes and the pointer to the actual content which is stored in the content repo. FlowFile repo is a write ahead log, so operations that are cancelled by failures can be retried.
* **Content repo**: Holds FlowFile’s content on the disk and reads into the memory only when it is needed.
* **Provenance repo**: The storage of each FlowFile’s history. It can provide data lineage (provenance).

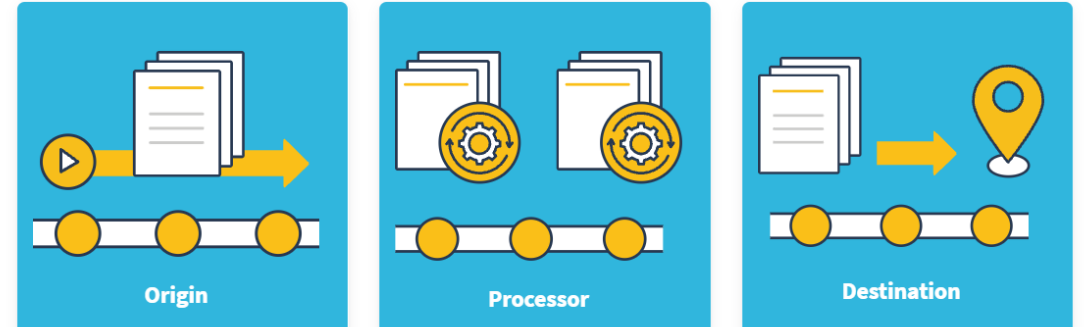
### MiNiFi

Designed to solve the hardships of transmitting data feeds to and from the source of origin. Allows data to be collected from various protocols and offers encryption and data provenance.

## StreamSets data collector

Data collector is an application to create data pipelines that offer continous data ingestion and no latency. SS enables users to migrate data from NoSQL, RDMS to another destination like Amazon S3, a data lake or a warehouse. SS can also help cleaning and transforming data to the ideal format.

In order to define the flow of data we have to design a pipeline in data collector. A pipeline consists of two+ stages. One representing the origin, one the destination and there could be any number of addtional stages responsible for performing data processing.



The origin creates a batch of data (reads into batch till the batch is full or there’s a timeout). The origin then send this batch forward. There can be only one origin.

The batch then moves from processor to processor until it reached the destination. On successful delivery DataCollector commits the success internally and notifies the origin to create the next (new) batch. There can be multiple destionations.

SS Data collector edge: runs pipelines (edge pipeline) on edge devices

SS Control hub: central point of controlling all of the dataflow pipelines

## Orchestration and scheduling

Orchestration tends to solve the problem of scalability as most of the Big Data projects fail when they are scaled to enterprise level.

### Apache Airflow

Airflow is a platform for programatically scheduling and monitoring Big Data workflows. It excels at architecting complex data pipelines and ETL. In Airflow workflows are designed and expressed as DAGs where each node is a task. Airflow is a code first platform (Python only).

Airflow has easy to use UI, but making test cases is hard there’s no windows support and tricky to attach airflow usage into the CI/CD pipeline as CI/CD that restart the container will kill any running processes.

**Operator:** DAG defines how to run a workflow, whereas operators determines what actually gets done by a task. Operators can be parametrized by a dictionary of *default\_args* passed to the DAG.

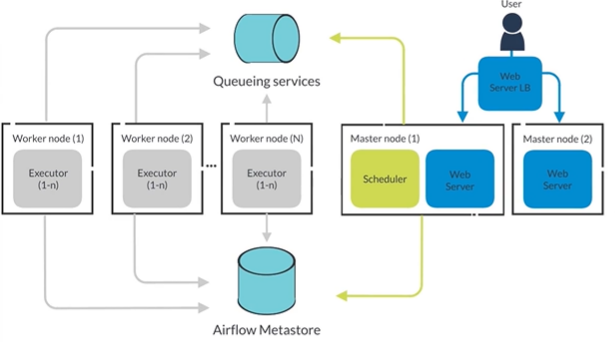
**DAG run:** The tasks in the DAG are run for a specific *execution\_date*. Each task is an implementation of an operator (task’s providing specific values).

Airflow daemons:

* **WebServer**: a daemon that accepts http requests allowing the user to interact with Airflow via Python flaskweb. Through the webserver DAGs can be paused, restarted or the user can view running DAGs and manage user roles or try out multiple configurations.
* **Scheduler**: Periodically polls to determine if any DAG needs to be triggered based on its schedule. This daemon is really important as it is the one that orchestrates DAGs (and thus tasks).
* **Executors**: The daemon that gets tasks executed. The executor is the node which runs the actual tasks.
  + **Sequential**: Each task is run on the same machine as the scheduler. This can only be done sequentially.
  + **Local**: Same as sequential but multiple tasks can run parallelly. An additional database is needed for DAG metadata (statuses).
  + **Celery**: The workload is distributed to multiple workers that can run on different devices. The best option for scalability.
* **Metastore**: Stores configurations (variables and connections) and holds user information, roles & policies. The scheduler puts the metadata created of the DAG here.

**Single Node architecture:** Airflow has to be configured with the LocalExecutor mode. Every service (daemon) is pushed into a single node.

**Multi Node architecture:** There’s a master node (scheduler & webserver) and multiple worker nodes (executors). Executors can be scaled horizontally by adding more executors in order to take load off the existing nodes. Executors can be scaled vertically by allowing more daemons to run on the specific node (*celeryd\_concurrency*). It is possible to horizontally scale Master Node’s mainly to solve the issue of having too much http requests (but only a single scheduler can run).



**Operators:** BashOperator (used to execute bash cmds), PythonOperator (calls a function with a specific signature), EmailOperator (sends emails using SMTP), SimpleHTTPOperator (sends an HTTP request that can trigger actions on a remote systems), MySqlOperator, SqliteOperator, OracleOperator (used to run SQL commands)

**Sensors:** Sensors are special operators that waits until an external or internal trigger (notification). ExternalTaskSensor (waits for another task in a different DAG to complete execution), HivePartitionSensor (waits for a specific value of partition of hive table to get created), S3KeySensor (waits for a specific file or dir to be available on an S3 bucket)

**Hooks:** are interfaces to services that are external to the Airflow Cluster. Hooks provide a unified interface to reach external services like Amazon S3, MySQL, Hive, Qubole etc. Hooks are supporting operators to interact with external services.

**XCom**: Lets tasks exchange messages. XComs are described by a key, value, timestamp and track attributes. XComs can be pushed or pulled. When pushed an XCom gets available to other tasks. On pulling our goal is to retrieve XComs optionally applying filter (key, task\_id) in before.

### Jenkins

Jenkins is an automation server that support practically every tool of CI/CD (build, static analysis, deployment, staging...). The pipeline is described in a ***jenkinsfile***. The jenkinsfile is often version controlled along with the project.

**Freestyle job:** define sequential tasks in an application lifecycle

**Pipeline:** enable you to define the whole application lifecycle, pipelines can be stopped for human input or approval before completing the jobs. A pipeline is a user-defined model of a CD pipeline which consists a code of the entire build process.

Pipeline concepts

**Node:** a machine that is capable of executing a pipeline

**Stage:** conceptually distinc set of tasks (build, test, deploy)

**Step:** a single task

**Controller:** basic installation of Jenkins, handles tasks

**Agent:** a computer for offloading particular projects

**Executor:** computational resource for compiling code, can run both on the controller and agent machine either by itself or with other executors parallelly.

The ***jenkinsfile*** stores the entire workflow in code. A jenkinsfile can be written declaratively or scripted. The declarative syntax is preferred as it makes reading and writing pipeline codes easier.

With the Blue Ocean UI automatically creates and writes your pipeline through the graphical pipeline editor.

# Neural Networks

## Forward and Backpropagation

### Math

chain rule:

### Forward propagation on XOR N.N.



### Backpropagation on XOR N.N.

## Egyéb

# NLP (Natural language processing)

## Stopwords & punctuation

The stopwords are the words that do not hold any meaning and are so-called sentiment neutral

**import** nltk

nltk.corpus.stopwords.words(“english”)

#[I, me, my, …]

**import** string

string.punctuation

#'!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~'

## Stemming & Lemmatizing

Converts words to their base (stem) form, because it is expected that the base form and the conjugated form has similar meanings. Stemming chops off the end of the word according to some heuristics and that’s why it can result words that cannot be found in dictionaries (entry -> entri) but on the other hand it is a fast algorithm. On the opposite side lemmatization will always result a dictonary word in trade of speed.

connection/connected/connective -> connect

**import** nltk

ps = nltk.PorterStemmer()

wn = nltk.WordNetLemmatizer()

ps.stem(“connection”) #connect

ps.stem(“uselessness”) #useless

wn.lemmatize(“entry”) #entry (instead of stem’s entri!)

## Vectorize

### Count vectorization

We’d like to convert our dataset into a frequency matrix, where each row refers to a token array and columns are the frequencies of the corresponding words.

**from** sklearn.feature\_extraction.text **import** CountVectorizer

cv = CountVectorizer(analyzer=the\_function\_that\_converts\_text\_to\_token\_array)

X\_counts = cv.fit\_transform(data["body\_text"]) #sparse matrix

X\_counts\_df = pd.DataFrame(X\_counts.toarray()) #dense matrix in df

X\_counts\_df.columns = cv.get\_feature\_names() #attribute names are no longer numbers, but rather the words themselves

### N-gram vectorization

We count the frequencies of every n long combination of adjacent words. n is a hyperparameter that has to be tuned. Autocomplete uses N-gram utilizing the most frequent trigrams.

*NLP is an interesting topic:*

* bigram: [nlp is] [is an] [an interesting] [interesting topic]
* trigram: [nlp is an] [is an interesting] [an interesting topic]
* 4-gram: [nlp is an interesting] [is an interesting topic]

**from** sklearn.feature\_extraction.text **import** CountVectorizer

cv = CountVectorizer(ngram\_range=(2, 3)) #interested in bi and trigrams, analyzer is default as in this case we don’t need tokenz but rather the tokens joined into a **full string**

X\_counts = cv.fit\_transform(data["body\_text"])

#the same as unigram count vectorization

### Inverse document frequency weighting (TF-IDF)

The cell values are not frequencies, but weights measuring the importance of the given word in the given document (SMS/email etc).

If a words is frequent in a single document, but not typical in others then this word will have a huge weight indicating that word has a meaning that makes us able to clearly differentiate the document from others. It highlights important but seldom used words.

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

tfidf=TfidfVectorizer(analyzer = the\_function\_that\_converts\_text\_to\_token\_array)

X\_tfidf = tfidf.fit\_transform(data['body\_text'])

## Egyéb

## Egyéb

# PyQt5

Allows us to make cross platform UI-s. The UI is made by using Qt Designer (.ui file). We can convert the created UI to python code with the following command:

*pyuic5 -x gui.ui -o tst.py*

## Connect

We’d like to connect callback function to our ui element (Button, Menu item, etc.) to handle actions:

**def** click\_cb(self):

self.label.setText(**"button clicked"**)

self.label.adjustSize()

**def** trig\_cb(self, param1):

self.label.setText(**"menuitem clicked "** + param1)

self.label.adjustSize()

self.button1.clicked.connect(self.click\_cb)  
self.menuItem1.triggered.connect(lambda: self.trig\_cb(**"extra params"**))

## MessageBox

**from** PyQt5.QtWidgets **import** QMessageBox

msg = QMessageBox()  
msg.setWindowTitle(**"WARNING!"**)  
msg.setText(**"You are about to end this man's whole career"**)  
msg.setIcon(QMessageBox.Warning)

msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)

msg.setDefaultButton(QMessageBox.Cancel)

msg.setInformativeText(**"Stop. Get some help."**)

msg.buttonClicked.connect(self.msg\_box\_btn\_handler) *#additional param*msg.exec\_()

*#alternative for simple use-cases*

QMessageBox.warning(**None**, **"Warning"**, **"Ur gon' format C."**)

# Web scraping

Web scraping is used for extracting data from websites directly from the raw HTML code.

Online JS beautifier (<https://beautifier.io/>) can help to go over unformatted HTML codes.

## BeautifulSoup

pip install beautifulsoup4

pip install lxml # to parse non-perfect/broken HTML codes as well

**from** bs4 **import** BeautifulSoup

**from** urllib.request **import** urlopen

**import** re

connection = urlopen(URL\_PAGE1)  
raw\_html = connection.read()  
connection.close()

soup = BeautifulSoup(raw\_html, **"lxml"**)  
cars = soup.find\_all(**"div"**, {**"class"**: re.compile(**"row talalati-sor\*"**)})x = cars[0].find(**"div"**, {**"class"** : **"vetelar"**})  
x.text **#will return the text from the div**

x.div.div.a.img[**"title"**] **#will return the title attr of the image tag**

## Selenium

pip install selenium

**from** selenium **import** webdriver  
**from** selenium.webdriver.common.keys **import** Keys  
**from** selenium.webdriver.common.by **import** By  
**from** selenium.webdriver.common.action\_chains **import** ActionChains  
**from** selenium.webdriver.support.ui **import** WebDriverWait  
**from** selenium.webdriver.support **import** expected\_conditions **as** EC  
**from** selenium.webdriver.chrome.options **import** Options

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 chrome\_options = Options()  
 chrome\_options.add\_argument(**"--headless"**)  
 chrome\_options.add\_argument(**"screenshot"**)

driver = webdriver.Chrome(**"C:\Chromedriver\chromedriver.exe"**, options=...)  
driver.get(**"https://play.typeracer.com/"**)

Explicit wait (wait for a webelement to appear, being enabled, etc.):

WebDriverWait(driver, 20).until(  
 EC.presence\_of\_element\_located((By.CLASS\_NAME, **"txtInput"**)) **and** EC.element\_to\_be\_clickable((By.CLASS\_NAME, **"txtInput"**))  
)

Implicit wait, which is similar to time.sleep(..):

driver.implicitly\_wait(3.5)

Find first webelement or every webelements with id,classname,css\_selector,etc.:

agree\_button = driver.find\_element\_by\_css\_selector(  
 **'button.VfPpkd-LgbsSe.VfPpkd-LgbsSe-OWXEXe-k8QpJ.VfPpkd-LgbsSe-OWXEXe-dgl2Hf.nCP5yc.AjY5Oe.DuMIQc.IIdkle'**)

#note that space is not allowed, replace with dot (.)!  
agree\_button.click()

search = driver.find\_element\_by\_id(**"search"**)  
search.send\_keys(**"Severina - Italiana"**)search.send\_keys(Keys.CONTROL + **"a"**)  
search.send\_keys(Keys.DELETE)  
search.send\_keys(**"Maja Suput - Neprilika"**)  
search.send\_keys(Keys.RETURN)

We can define ActionChain-s for frequent actions we want to happen together. Also, ActionChain can help with unclickable webelements. To fire an ActionChain call perform().

action\_chain = ActionChains(driver)

action\_chain.move\_to\_element(spans[i]).click()

action.perform()

Auxiliary functions for browsing:

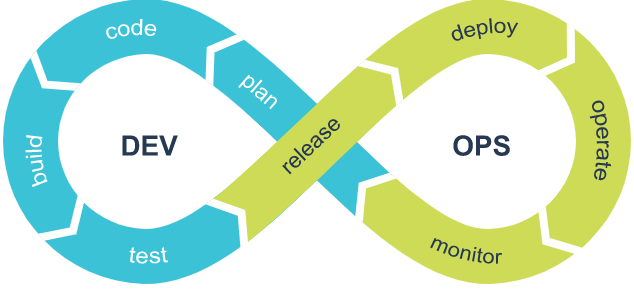
driver.back() # goes back by one website  
driver.forward() # goes forward by one website

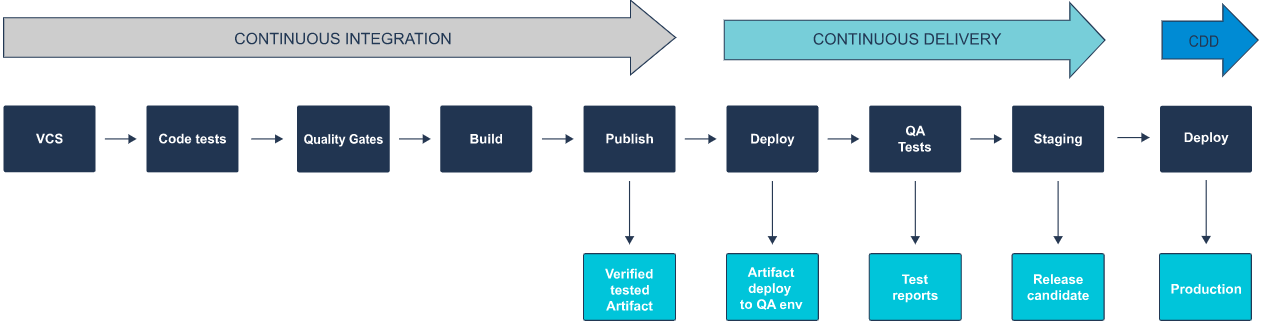
driver.close() # closes tab

driver.quit() # closes browser

# DevOps

Development and operations (in short: DevOps) provide a variety of principles and practices for effective collaboration between development and operations teams. DevOps helps organizations to release product more frequently and to identify a bug or problem rapidly.





## Practices

* **Continous Integration**: Development teams frequently merge smaller code updates into the central repository where automated testing is performed. This help finding major issues like bugs much-much faster.
* **Continous Delivery**: Involves building, testing and preparing code changes for delivery. After the build stage the product is deployed into a testing/production environment.
* **Automation**: DevOps is achieved if the building/testing/etc. is fully automated.
* **Configuration Management**: Automated OS and host configuration which liberates the developer from the need to manually configure OS-es.
* **Monitoring and Logging**: The process should be monitored along with the user responses. With additional analysis organizations can track their services more effectively.

# Math auxiliary

## Matrix properties

### Matrix properties

Positive semidefinite:

Positive definite:

Negative semidefinite:

Negative definite:

Alternative definition:

H is negative semidefinite if:

A square matrix is **singular** if it is not invertible (or equivalently, has zero determinant).

## Eigenvalues and eigenvectors

**A** is a square matrix, **x** is a non null vector. is a scalar called eigenvalue. The is the eigenvector associated with the eigenvalue .

Trace are the sum of the diagonal entries of a square matrix.

**Theorem:** The sum of eigenvalues is exactly the trace.

**Theorem:** The product of eigenvalues will result in the matrix’ determinant.

## Rank of matrix

Row rank: number of linearly independent rows. ().

Column rank: number of linearly independent columns. ().

Often the row rank is equal to the column rank, so we simply refer to this common value as the rank of the matrix.

A matrix full of zeros has a rank of 0. Zero vectors do not increase the rank.

An matrix is invertible if and only if the rank of the matrix is .

## Kernel and image

Kernel (nullspace) is the set of vectors for which a given linear mapping outputs the zero vector.

// TODO

## Norm

Special cases:

* 1-norm:
* 2-norm:
* max-norm:

Proof for max-norm:

*Raising absolute values to large power will render elements other than the maximum negligible in amplitude.*

## Function properties

A function is **injective** if for any x and y, f(x)=f(y) is fulfilled if and only if x=y.

### Jensen’s inequality

For any **convex function** f and random variable X the following holds:

For any **concave function** f and random variable X the following holds:

Convex function: a chord between two points will be above the function (x2)

Concave function: a chord between two point will be below the function (-x2)

Jensen’s inequality becomes a strict inequality if there are no flat sections on the function (aka. the function is strictly convex/concave).

|  |  |  |
| --- | --- | --- |
| The function is ... in an interval X if the following is fulfilled for every x in X: | - | strictly |
| Convex |  |  |
| Concave |  |  |

|  |  |  |
| --- | --- | --- |
| The multidimensional function is ... in the set X if the following is fulfilled for every **x** in X: | - | strictly |
| Convex |  |  |
| Concave |  |  |

**x** is an n long row vector, H is the Hessian with n rows and n cols.

### Taylor’s expansion

a is the arbitrarily used center of the Taylor expansion, which is chosen 0 frequently.

### Integration by part (partial integral)

We know from the product rule of derivatives that:

Here u and v are shorthands for functions u(x) and v(x).

Indefinite case:

Integrating both sides will result in:

Definite case:

## Inequalities

### Triangle inequality

### Cauchy-Bunyakovskiy-Schwarz inequality

For vectors **x** and **y** the following always holds:

## Numerical optimalization

### Lagrange multiplier

Lagrange multiplier tackles the problem of having an f(x) function to minimize with respect to some x but simultaneously having a single equality constraint on x denoted with g(x).

The original optimalization problem

is being converted into an extended form:

is called the Lagrange multiplier.

## Graph theory

### DAG and topological ordering

A graph is directed acyclic graph if it has a topological ordering. A graph is a DAG if there is no series of directed edges from any vertex **v** that we follow we get back to **v**.

Topological ordering is an order of vertices in which every edge in the graph is directed from an earlier appearing to a later appearing.

# Standard normal CDF () chart.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***z*** | **+ 0.00** | **+ 0.01** | **+ 0.02** | **+ 0.03** | **+ 0.04** | **+ 0.05** | **+ 0.06** | **+ 0.07** | **+ 0.08** | **+ 0.09** |
| **0.0** | 0.50000 | 0.50399 | 0.50798 | 0.51197 | 0.51595 | 0.51994 | 0.52392 | 0.52790 | 0.53188 | 0.53586 |
| **0.1** | 0.53983 | 0.54380 | 0.54776 | 0.55172 | 0.55567 | 0.55962 | 0.56360 | 0.56749 | 0.57142 | 0.57535 |
| **0.2** | 0.57926 | 0.58317 | 0.58706 | 0.59095 | 0.59483 | 0.59871 | 0.60257 | 0.60642 | 0.61026 | 0.61409 |
| **0.3** | 0.61791 | 0.62172 | 0.62552 | 0.62930 | 0.63307 | 0.63683 | 0.64058 | 0.64431 | 0.64803 | 0.65173 |
| **0.4** | 0.65542 | 0.65910 | 0.66276 | 0.66640 | 0.67003 | 0.67364 | 0.67724 | 0.68082 | 0.68439 | 0.68793 |
|  |  |  |  |  |  |  |  |  |  |  |
| **0.5** | 0.69146 | 0.69497 | 0.69847 | 0.70194 | 0.70540 | 0.70884 | 0.71226 | 0.71566 | 0.71904 | 0.72240 |
| **0.6** | 0.72575 | 0.72907 | 0.73237 | 0.73565 | 0.73891 | 0.74215 | 0.74537 | 0.74857 | 0.75175 | 0.75490 |
| **0.7** | 0.75804 | 0.76115 | 0.76424 | 0.76730 | 0.77035 | 0.77337 | 0.77637 | 0.77935 | 0.78230 | 0.78524 |
| **0.8** | 0.78814 | 0.79103 | 0.79389 | 0.79673 | 0.79955 | 0.80234 | 0.80511 | 0.80785 | 0.81057 | 0.81327 |
| **0.9** | 0.81594 | 0.81859 | 0.82121 | 0.82381 | 0.82639 | 0.82894 | 0.83147 | 0.83398 | 0.83646 | 0.83891 |
|  |  |  |  |  |  |  |  |  |  |  |
| **1.0** | 0.84134 | 0.84375 | 0.84614 | 0.84849 | 0.85083 | 0.85314 | 0.85543 | 0.85769 | 0.85993 | 0.86214 |
| **1.1** | 0.86433 | 0.86650 | 0.86864 | 0.87076 | 0.87286 | 0.87493 | 0.87698 | 0.87900 | 0.88100 | 0.88298 |
| **1.2** | 0.88493 | 0.88686 | 0.88877 | 0.89065 | 0.89251 | 0.89435 | 0.89617 | 0.89796 | 0.89973 | 0.90147 |
| **1.3** | 0.90320 | 0.90490 | 0.90658 | 0.90824 | 0.90988 | 0.91149 | 0.91308 | 0.91466 | 0.91621 | 0.91774 |
| **1.4** | 0.91924 | 0.92073 | 0.92220 | 0.92364 | 0.92507 | 0.92647 | 0.92785 | 0.92922 | 0.93056 | 0.93189 |
|  |  |  |  |  |  |  |  |  |  |  |
| **1.5** | 0.93319 | 0.93448 | 0.93574 | 0.93699 | 0.93822 | 0.93943 | 0.94062 | 0.94179 | 0.94295 | 0.94408 |
| **1.6** | 0.94520 | 0.94630 | 0.94738 | 0.94845 | 0.94950 | 0.95053 | 0.95154 | 0.95254 | 0.95352 | 0.95449 |
| **1.7** | 0.95543 | 0.95637 | 0.95728 | 0.95818 | 0.95907 | 0.95994 | 0.96080 | 0.96164 | 0.96246 | 0.96327 |
| **1.8** | 0.96407 | 0.96485 | 0.96562 | 0.96638 | 0.96712 | 0.96784 | 0.96856 | 0.96926 | 0.96995 | 0.97062 |
| **1.9** | 0.97128 | 0.97193 | 0.97257 | 0.97320 | 0.97381 | 0.97441 | 0.97500 | 0.97558 | 0.97615 | 0.97670 |
|  |  |  |  |  |  |  |  |  |  |  |
| **2.0** | 0.97725 | 0.97778 | 0.97831 | 0.97882 | 0.97932 | 0.97982 | 0.98030 | 0.98077 | 0.98124 | 0.98169 |
| **2.1** | 0.98214 | 0.98257 | 0.98300 | 0.98341 | 0.98382 | 0.98422 | 0.98461 | 0.98500 | 0.98537 | 0.98574 |
| **2.2** | 0.98610 | 0.98645 | 0.98679 | 0.98713 | 0.98745 | 0.98778 | 0.98809 | 0.98840 | 0.98870 | 0.98899 |
| **2.3** | 0.98928 | 0.98956 | 0.98983 | 0.99010 | 0.99036 | 0.99061 | 0.99086 | 0.99111 | 0.99134 | 0.99158 |
| **2.4** | 0.99180 | 0.99202 | 0.99224 | 0.99245 | 0.99266 | 0.99286 | 0.99305 | 0.99324 | 0.99343 | 0.99361 |
|  |  |  |  |  |  |  |  |  |  |  |
| **2.5** | 0.99379 | 0.99396 | 0.99413 | 0.99430 | 0.99446 | 0.99461 | 0.99477 | 0.99492 | 0.99506 | 0.99520 |
| **2.6** | 0.99534 | 0.99547 | 0.99560 | 0.99573 | 0.99585 | 0.99598 | 0.99609 | 0.99621 | 0.99632 | 0.99643 |
| **2.7** | 0.99653 | 0.99664 | 0.99674 | 0.99683 | 0.99693 | 0.99702 | 0.99711 | 0.99720 | 0.99728 | 0.99736 |
| **2.8** | 0.99744 | 0.99752 | 0.99760 | 0.99767 | 0.99774 | 0.99781 | 0.99788 | 0.99795 | 0.99801 | 0.99807 |
| **2.9** | 0.99813 | 0.99819 | 0.99825 | 0.99831 | 0.99836 | 0.99841 | 0.99846 | 0.99851 | 0.99856 | 0.99861 |
|  |  |  |  |  |  |  |  |  |  |  |
| **3.0** | 0.99865 | 0.99869 | 0.99874 | 0.99878 | 0.99882 | 0.99886 | 0.99889 | 0.99893 | 0.99896 | 0.99900 |
| **3.1** | 0.99903 | 0.99906 | 0.99910 | 0.99913 | 0.99916 | 0.99918 | 0.99921 | 0.99924 | 0.99926 | 0.99929 |
| **3.2** | 0.99931 | 0.99934 | 0.99936 | 0.99938 | 0.99940 | 0.99942 | 0.99944 | 0.99946 | 0.99948 | 0.99950 |
| **3.3** | 0.99952 | 0.99953 | 0.99955 | 0.99957 | 0.99958 | 0.99960 | 0.99961 | 0.99962 | 0.99964 | 0.99965 |
| **3.4** | 0.99966 | 0.99968 | 0.99969 | 0.99970 | 0.99971 | 0.99972 | 0.99973 | 0.99974 | 0.99975 | 0.99976 |
|  |  |  |  |  |  |  |  |  |  |  |
| **3.5** | 0.99977 | 0.99978 | 0.99978 | 0.99979 | 0.99980 | 0.99981 | 0.99981 | 0.99982 | 0.99983 | 0.99983 |
| **3.6** | 0.99984 | 0.99985 | 0.99985 | 0.99986 | 0.99986 | 0.99987 | 0.99987 | 0.99988 | 0.99988 | 0.99989 |
| **3.7** | 0.99989 | 0.99990 | 0.99990 | 0.99990 | 0.99991 | 0.99991 | 0.99992 | 0.99992 | 0.99992 | 0.99992 |
| **3.8** | 0.99993 | 0.99993 | 0.99993 | 0.99994 | 0.99994 | 0.99994 | 0.99994 | 0.99995 | 0.99995 | 0.99995 |
| **3.9** | 0.99995 | 0.99995 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99996 | 0.99997 | 0.99997 |
|  |  |  |  |  |  |  |  |  |  |  |
| **4.0** | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99997 | 0.99998 | 0.99998 | 0.99998 | 0.9999 |

# Student-t CDF chart

| **Conf. Level** | **90%** | **95%** | **98%** | **99%** |
| --- | --- | --- | --- | --- |
| **One Tail** | **0.050** | **0.025** | **0.010** | **0.005** |
| **Two Tail** | **0.100** | **0.050** | **0.020** | **0.010** |
| **df = 1** | 6.314 | 12.706 | 31.821 | 63.657 |
| **2** | 2.920 | 4.303 | 6.965 | 9.925 |
| **3** | 2.353 | 3.182 | 4.541 | 5.841 |
| **4** | 2.132 | 2.776 | 3.747 | 4.604 |
| **5** | 2.015 | 2.571 | 3.365 | 4.032 |
| **6** | 1.943 | 2.447 | 3.143 | 3.707 |
| **7** | 1.895 | 2.365 | 2.998 | 3.499 |
| **8** | 1.860 | 2.306 | 2.896 | 3.355 |
| **9** | 1.833 | 2.262 | 2.821 | 3.250 |
| **10** | 1.812 | 2.228 | 2.764 | 3.169 |
| **11** | 1.796 | 2.201 | 2.718 | 3.106 |
| **12** | 1.782 | 2.179 | 2.681 | 3.055 |
| **13** | 1.771 | 2.160 | 2.650 | 3.012 |
| **14** | 1.761 | 2.145 | 2.624 | 2.977 |
| **15** | 1.753 | 2.131 | 2.602 | 2.947 |
| **16** | 1.746 | 2.120 | 2.583 | 2.921 |
| **17** | 1.740 | 2.110 | 2.567 | 2.898 |
| **18** | 1.734 | 2.101 | 2.552 | 2.878 |
| **19** | 1.729 | 2.093 | 2.539 | 2.861 |
| **20** | 1.725 | 2.086 | 2.528 | 2.845 |
| **21** | 1.721 | 2.080 | 2.518 | 2.831 |
| **22** | 1.717 | 2.074 | 2.508 | 2.819 |
| **23** | 1.714 | 2.069 | 2.500 | 2.807 |
| **24** | 1.711 | 2.064 | 2.492 | 2.797 |
| **25** | 1.708 | 2.060 | 2.485 | 2.787 |
| **26** | 1.706 | 2.056 | 2.479 | 2.779 |
| **27** | 1.703 | 2.052 | 2.473 | 2.771 |
| **28** | 1.701 | 2.048 | 2.467 | 2.763 |
| **29** | 1.699 | 2.045 | 2.462 | 2.756 |
| **30** | 1.697 | 2.042 | 2.457 | 2.750 |
| **40** | 1.684 | 2.021 | 2.423 | 2.704 |
| **50** | 1.676 | 2.009 | 2.403 | 2.678 |
| **60** | 1.671 | 2.000 | 2.390 | 2.660 |
| **70** | 1.667 | 1.994 | 2.381 | 2.648 |
| **80** | 1.664 | 1.990 | 2.374 | 2.639 |
| **90** | 1.662 | 1.987 | 2.368 | 2.632 |
| **100** | 1.660 | 1.984 | 2.364 | 2.626 |
| **+inf** | 1.645 | 1.960 | 2.326 | 2.576 |

# CDF charts in R

Critical U values:

Critical T values:

Critical F values:

Critical KS values: =

(needs: "BoutrosLab.plotting.general")

Critical values for low sample Wilocoxon test:

=

Critical values for low sample Mann-Whitney test:

=

Critical CSq values:

Critical Friedman value:

(needs: 'SuppDists')

Critical binom value:

# 