

[Open in app ↗](#)

Search



Write



Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Master Data Science with This Comprehensive Cheat Sheet

Comprehensive Cheat Sheet for Data Science: Numpy, Pandas, Python, R, ML, DL, NLP, Stats, SQL, PySpark, Plotly, Seaborn, git, Excel, Tableau, and PowerBI

Ritesh Gupta · [Follow](#)

Published in Artificial Intelligence in Plain English · 6 min read · Jan 29, 2023



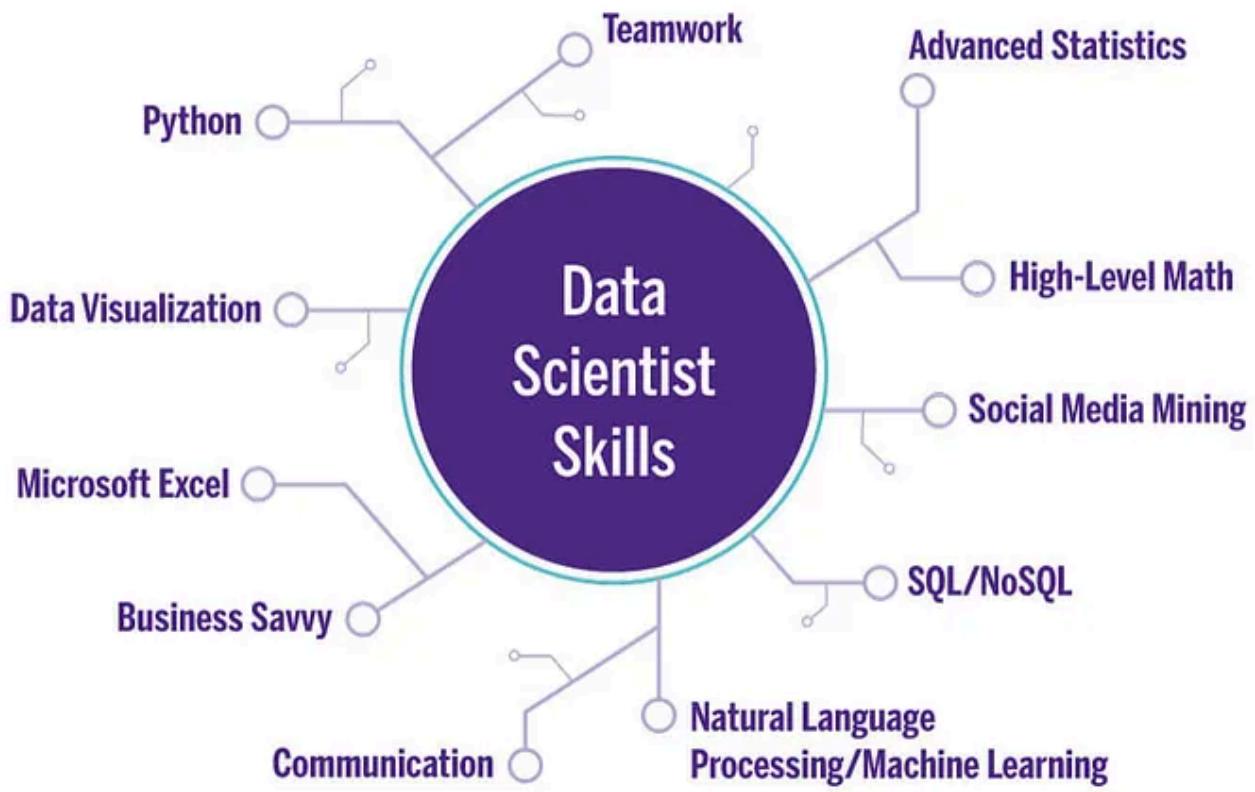
1.5K



24



Data science is a rapidly growing field that combines statistics, mathematics, and computer science to extract insights and knowledge from data. As a data scientist, you need to be proficient in a variety of tools, techniques, and concepts to effectively analyze and visualize data. To help streamline your work, we have created the ultimate data science cheat sheet.



The cheat sheet covers all the essential topics in data science, from the basics of statistics and probability to advanced machine learning algorithms and deep learning techniques. It is designed to be a quick reference guide for data scientists, providing a comprehensive overview of the key concepts and tools used in the field.

Here are some of the topics covered in the data science cheat sheet:

1. **Statistics:** Understanding the basics of statistics is crucial for data science. This section covers key concepts such as mean, median, mode, standard deviation, and correlation.

Key Definitions

- Variable: In statistics, a variable is a quantity that can be measured or counted. In data analysis, a variable is a column in a data frame.
- Descriptive statistics: Numbers that summarize samples. They are also called summary statistics or aggregations.
- Categorical variables: Variables consisting of discrete groups. These categories are called levels (e.g., educational levels).
- Numerical data: Data that consists of numbers (e.g., age).

Categorical Data—Trail Mix

To illustrate statistical concepts on categorical data, we'll be using an unordered categorical variable, consisting of different elements of a trail mix. Our categorical variable contains 10 almonds, 10 cashews, and 15 cranberries.

Food category	Count	Proportion
Almond	10	10 / 45 = 0.222
Cashew	10	10 / 45 = 0.222
Cranberry	25	25 / 45 = 0.444

Visualizing Categorical Variables

- Bar plot
- Stacked bar chart
- Treemap chart

One of the easiest charts to read, which helps in quick comparison of categories. One axis contains categories and the other axis represents values.

Used to compare subcategories within categorical items. Can also be used to compare proportions.

2D rectangles whose size is proportional to the value being measured and can be used to display hierarchically structured data.

Numerical Dataset—Glasses of Water

To illustrate statistical concepts on numerical data, we'll be using a numerical variable, consisting of the volume of water in different glasses.

Glass	Volume (ml)
Small	100 ml
Large	200 ml
Medium	150 ml
Small	100 ml
Large	200 ml
Medium	150 ml
Small	100 ml
Large	200 ml
Medium	150 ml
Small	100 ml

Measures of Center

Measures of center allow you to describe or summarize your data by capturing one value that describes the center of its distribution.

Measure	Definition	How to Find It	Result
Arithmetic mean	The total of the values divided by how many values there are		205.7 ml
Median	The middle value, when values from smallest to largest		150 ml
Mode	The most common value		200 ml

Measures of Spread

Sometimes, rather than caring about the size of values, you care about how different they are.

Measure	Definition	How to Find It	Result
Range	The highest value minus the lowest value		100 ml
Variance	The sum of the squares of the differences between each value and the mean, all divided by one less than the number of data points		940.8 ml ²
IQR (Interquartile Range)	The third quartile minus the first quartile		50 ml

Visualizing Numeric Variables

There are a variety of ways of visualizing numerical data; here's a few of them in action:

- Histogram
- Box plot

Correlation

Correlation is a measure of the linear relationship between two variables. That is, when one variable goes up, does the other variable go up or down? There are several algorithms to calculate correlation, but it is always a score between -1 and +1.

For two variables, X and Y, correlation has the following interpretation:

Correlation score	Interpretation
-1	When X increases, Y decreases. Scatter plot forms a perfect straight line with negative slope.
Between -1 and 0	When X increases, Y decreases.
0	There is no linear relationship between X and Y as the scatter plot looks like a noisy mess.
Between 0 and +1	When X increases, Y increases.
+1	When X increases, Y increases. Scatter plot forms a perfect straight line with positive slope.

Note that correlation does not account for non-linear effects, so if X and Y do not have a straight-line relationship, the correlation score may not be meaningful.

credit: datacamp

For Download [Click here](#)

2. Probability: Probability is a fundamental concept in data science, used to make predictions and draw inferences from data. This section covers basic probability concepts, such as Bayes' theorem and conditional probability.

Definitions

- Event:** A thing that you can observe whether it happens or not.
- Probability:** The chance that an event happens, on a scale from 0 (cannot happen) to 1 (always happens). Denoted $P(\text{event})$.
- Probability universe:** The probability space where all the events you are considering can either happen or not happen.
- Mutually exclusive events:** If one event happens, then the other event cannot happen (e.g., you cannot roll a dice that is both 5 and 6).
- Independent events:** If one event happens, it does not affect the probability that the other event happens (e.g., the weather does not affect the outcome of a dice roll).
- Dependent events:** If one event happens, it changes the probability that the other event happens (e.g., the weather affects traffic outcomes).
- Conjunctive probability (a.k.a. joint probability):** The probability that all events happen.
- Disjunctive probability:** The probability that at least one event happens.
- Conditional probability:** The probability that one event happens, given another event happened.

Multiplication Rules: Probability of two events happening

Mutually exclusive events

Definition: The probability of two mutually exclusive events happening is zero.
Formula: $P(A \cap B) = 0$.
Example: If the probability of it being sunny or raining is 0.3 and the probability of it raining or misty is 0.6, the probability of it being sunny and rainy is 0, since these events are mutually exclusive.

Independent events

Definition: The probability of two independent events happening is the product of the probabilities of each event.
Formula: $P(A \cap B) = P(A) \cdot P(B)$.
Example: If the probability of it being sunny or misty is 0.3 and the probability of your favorite soccer team winning their game today is 0.6, then the probability of it being sunny or misty and your favorite soccer team winning their game today is $0.3 \cdot 0.6 = 0.18$.

The conjunctive fallacy

Definition: The probability of both events happening is always less than or equal to the probability of one event happening. That is $P(A \cap B) \leq P(A)$, and $P(A \cap B) \leq P(B)$. The conjunctive fallacy is when you don't think carefully about probabilities and estimate that the probability of both events happening is greater than the probability of one of the events.

Example: A famous example known as "The Linda problem" comes from a 1983 research experiment. A fictional person was described:

Linda is 31 years old, single, intelligent, and very bright. She is inspired in philosophy. As a student, she was deeply concerned with issues of discrimination and social justice and also participated in anti-nuclear demonstrations.

Participants had to choose which statement had a higher probability of being true:

1. Linda is a bank teller.
2. Linda is a bank teller and is active in the feminist movement.

Most participants chose for the conjunctive fallacy and chose option 2; even though it must be less likely than option 1 using the multiplication rule.

Addition Rules: Probability of at least one event happening

Mutually exclusive events

Definition: The probability of at least one mutually exclusive event happening is the sum of the probabilities of each event happening.
Formula: $P(A \cup B) = P(A) + P(B)$.
Example: If the probability of it being sunny or misty is 0.3 and the probability of it raining or misty is 0.4, the probability of it being sunny or rainy or misty is $0.3 + 0.4 = 0.7$, since these events are mutually exclusive.

Independent events

Definition: The probability of at least one mutually exclusive event happening is the sum of the probabilities of each event happening minus the probability of both events happening.
Formula: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.
Example: If the probability of it being sunny or misty is 0.3 and the probability of your favorite soccer team winning their game today is 0.6, then the probability of it being sunny or misty or your favorite soccer team winning their game today is $0.3 + 0.6 - 0.3 \cdot 0.6 = 0.7$.

The disjunctive fallacy

Definition: The probability of at least one event happening is always greater than or equal to the probability of one event happening. That is $P(A \cup B) \geq P(A)$, and $P(A \cup B) \geq P(B)$. The disjunctive fallacy is when you don't think carefully about probabilities and estimate that the probability of at least one event happening is less than the probability of one of the events.

Example: Returning to the "Linda problem", consider having to rank these five statements in order of probability:

1. Linda is a bank teller.
2. Linda is a bank teller and is active in the feminist movement.

The disjunctive fallacy would be to think that choice 1 had a higher probability of being true, even though that is impossible because of the additive rule of probabilities.

Learn Statistics Online at www.DataCamp.com

datacamp

credit: datacamp

For Download [Click here](#)

3. Data storytelling: Data storytelling is the process of communicating insights and findings from data analysis in a clear and compelling manner. The goal of data storytelling is to engage the audience and convey the key messages in a way that is easy to understand, memorable, and impactful.

Datacamp

Data Storytelling & Communication Cheat Sheet

Learn more online at www.DataCamp.com

> What is data storytelling?

Data storytelling is often called the art of telling stories about data. It's a communication skills where data professionals tell stories that make use of their insights according to Gary Lyman, Author of *Data Storytelling*. According to him, Data Storytelling can be defined as "Data Storytelling is a combination of data, visuals, and narrative."

Date **Visuals** **Narrative**

For more information on Data Storytelling, please refer to [Data Storytelling \(How to Create Effective Data Storytelling\)](#).

> Crafting effective visuals

Choose the best visualization for your story

Each plot type is suited for communicating specific things about specific types of data. Start by choosing an appropriate plot type.

Line plot	Bar plot	Scatter plot	Histogram
Show changes in numeric values over time.	Visualize numeric values in categories to show trends or variations.	Show the relationship between two numeric values.	Show the distribution of numeric values.

To learn about all the types of visualizations you can use, check out our [Data Visualization Cheat Sheet](#).

Keep visualizations minimal and avoid clutter

Minimally, edit your plots to remove unnecessary elements from the messenger of the plot. In particular, reduce data elements (like the grid lines) that aren't directly representing a data value, like the grid lines that distract a great example comes from *Dashboards Analytics*, which demonstrates exactly the value of de-cluttering visualizations.

De-cluttering a visualization in action.
Source: [Dashboards Analytics](#)

Data visualization de-cluttering best practices

- Use just enough white space to keep the visualization from looking busy.
- Remove chart borders when applicable.
- Remove or minimize gridlines or axes when applicable.
- Clean up data labels when applicable.
- Label only the data relevant to the chart (e.g., a legend).
- Remove data series when applicable.
- Use special effects (bold, underline, italic, shadowed) sparingly.

Use text appropriately

While too much text can easily feel俗氣, it can also be an extremely effective tool of highlighting insights within your visualizations. Cole Nussbaumer Knuttila, Author of *Data Storytelling with Data* advises an excellent example with the following visualization:

Power usage vs time
Power usage vs time

Source: [Data Storytelling with Data](#) (Cole Nussbaumer Knuttila)

Using text as a useful visual tool when crafting effective stories

(Source: [Data Storytelling with Data](#) (Cole Nussbaumer Knuttila))

Using text in data visualizations

- When applicable, limit text and focus for clarity.
- Label important data points when necessary.
- Provide visual context around insights within the title or subtitle.
- Adjust font size when highlighting specific insights within your labels.
- When applicable, try to answer common audience questions with labels.

Use colors effectively

The fundamentals of color theory in data visualization

Color is one of the most powerful tools available for emphasizing different aspects of your data visualization; here are different properties to keep in mind when choosing an appropriate color palette for your visualization.

- Value increases the range of possible colors, from red, through orange, green and blue, to purple and back to red.
- Chroma is the intensity of the color, from grey to bright colors.
- Luminance is the brightness of the color, from black to white.

There are three common types of color palettes, which depend on these dimensions.

Type	Purpose	What to vary	Example
Qualitative	Discriminate categories.	Color	A set of 200+ hexagons uses ten different chromatic intensities.
Sequential	Emphasize a range of values.	Chroma or luminance	A large sequencing (value) visualization.
Staggered	Compare between two groups.	Chroma or luminance with hue.	Interior representation (percentage) styling in the USA.

Do not mislead with data stories

The easiest way to lose credibility when presenting stories is to *misrepresent* (or intentionally) mislead with your data insights. Here are four best practices to avoid misleading with data stories.

Same Data, Different Y-Axis

Sharing the y-axis of the smallest value or zero dramatically changes the story told by the chart.

Best practices to avoid misleading with data stories

- If you are visualizing three series data, make sure your time horizons are large enough to truly represent the data.
- If the relative size of each value is important, then ensure that your axes start with zero.
- Ensure that even scales are appropriate given the data you're presenting.
- If you are sampling data for descriptive purposes, make sure the sample is representative of the broader population.
- Use contrasting measures such as mean or median to provide context around your data.

> Crafting effective narratives with data

Know the audience

To communicate effectively, you need to know who your audience is and what their priorities are. There is a range of possible audiences you may encounter when presenting and crafting an audience-specific message will be important. Examples of audiences you may present to are:

Executive
Basic data literacy skills
Prioritizes outcomes & decisions
Cares much more about business impact than risk/return
Wants to see a quick return on investment in a machine learning model occurring or a real technique you're using

Data Leader
Data expert
Prioritizes rigour & insights
Cares much more about your analysis impacts their workday, and what should be their main takeaways from the data story

Business Partner
Advanced data literacy skills
Prioritizes tactical next steps
Cares much more about how your analysis impacts their workday, and what should be their main takeaways from the data story

Considerations when crafting audience specific messaging

Aspect	What do you need to consider?
Message □	<ul style="list-style-type: none"> What content is most interesting? What is the key message?
Platform □	<ul style="list-style-type: none"> What does the audience care about? How does your message relate to that goal? Who is driving decision-making within your audience?
Narrative □	<ul style="list-style-type: none"> What is the audience's perspective? How does this story on audience needs to convince a data story?

Choose the best medium to share your story

There are different ways you can deliver a data story. The importance of each is different depending on the audience of your story and the setting you're delivering your story in.

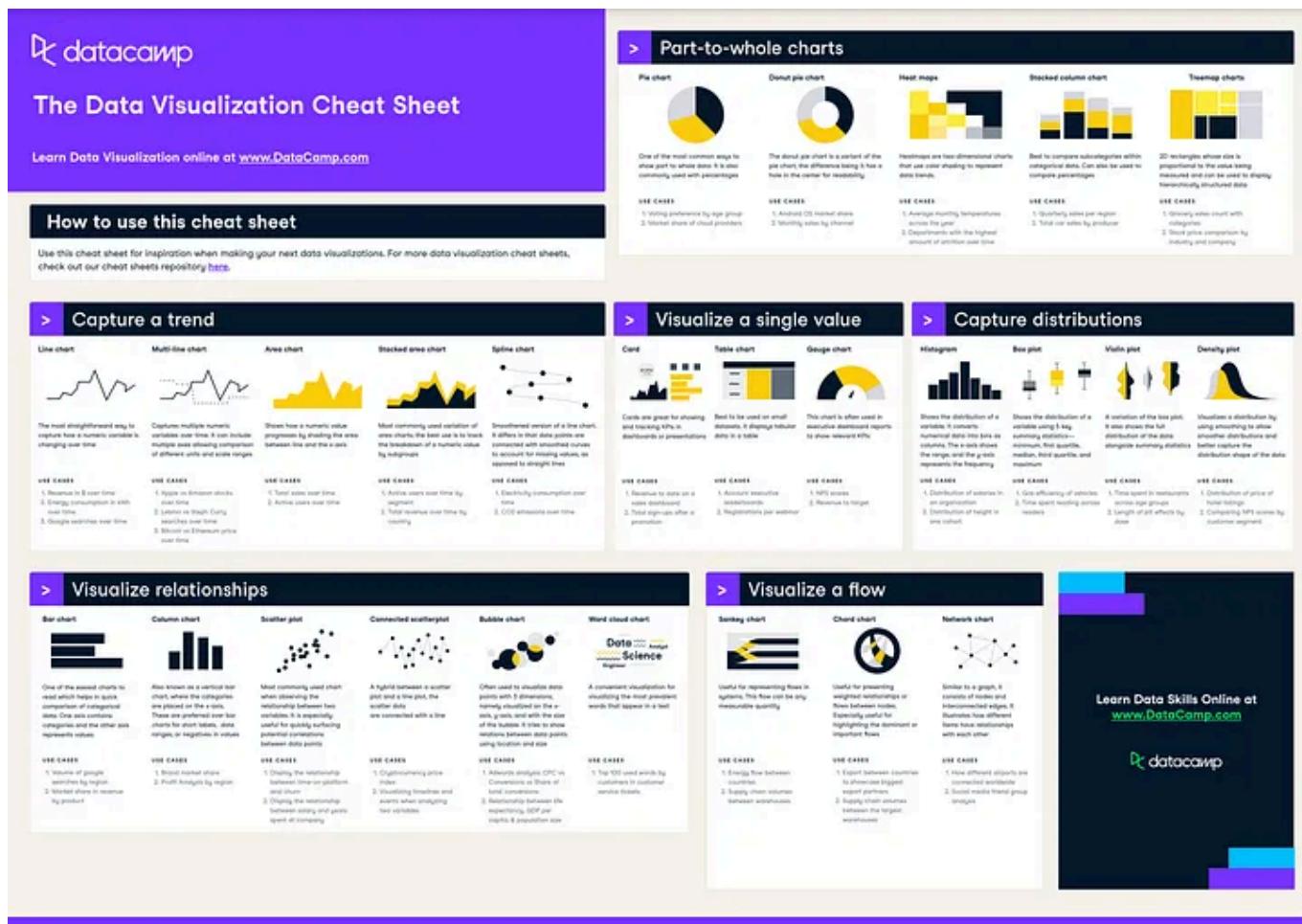
Type	Important considerations
Presentation □	<ul style="list-style-type: none"> Ensures the length of your presentation is appropriate. Ensures any highlighting details fit to the screen. Ensures data is in context so it's understandable.
Long-form report □	<ul style="list-style-type: none"> Be sure to include context around providing useful context around your data insights. Ensures any highlighting details fit to the page.
Newsletter □	<ul style="list-style-type: none"> Ensures first you provide useful context to how you arrived at a certain finding. Mobile use of the audience's perspective. Represents data insights from left to right, top to bottom.
Dashboard □	<ul style="list-style-type: none"> Provides useful context of key visualizations in your dashboard.

Learn more about data storytelling at www.DataCamp.com

credit: datacamp

For Download [Click here](#)

4. Data Visualization: Data visualization is an essential part of data science, allowing you to explore and understand the relationships and patterns in your data. This section covers popular visualization tools such as ggplot2, Matplotlib, and Seaborn.



credit: datacamp

For Download [Click here](#)

5. Machine Learning: Machine learning is the process of training algorithms to automatically learn from data and make predictions. This section covers popular machine learning algorithms such as linear regression, decision trees, and k-nearest neighbors.

ALGORITHM	DESCRIPTION	APPLICATIONS	ADVANTAGES	DISADVANTAGES
Linear Models	Linear Regression	A simple algorithm that models a linear relationship between inputs and a continuous numerical output variable	USE CASES 1. Stock price prediction 2. Predicting housing prices 3. Predicting customer lifetime value	1. Explainable method 2. Interpretable results by its output coefficients 3. Faster to train than other machine learning models
	Logistic Regression	A simple algorithm that models a linear relationship between inputs and a categorical output (1 or 0)	USE CASES 1. Credit risk score 2. Customer churn prediction	1. Assumes linearity between inputs and outputs 2. Sensitive to outliers 3. Can suffer with small, high-dimensional data
	Ridge Regression	Part of the regression family — it penalizes features that have low predictive outcomes by shrinking their coefficients closer to zero. Can be used for classification or regression	USE CASES 1. Predictive maintenance for automobiles 2. Sales revenue prediction	1. All the predictors are kept in the final model 2. Doesn't perform feature selection
	Lasso Regression	Part of the regression family — it penalizes features that have low predictive outcomes by shrinking their coefficients to zero. Can be used for classification or regression	USE CASES 1. Predicting housing prices 2. Predicting clinical outcome based on health data	1. Less prone to overfitting 2. Can handle high-dimensional data 3. No need for feature selection
Tree-Based Models	Decision Tree	Decision Tree models make decision rules on the features to produce predictions. It can be used for classification or regression	USE CASES 1. Customer churn prediction 2. Credit score modeling 3. Disease prediction	1. Explainable and interpretable 2. Can handle missing values
	Random Forests	An ensemble learning method that combines the output of multiple decision trees	USE CASES 1. Credit score modeling 2. Predicting housing prices	1. Reduces overfitting 2. Higher accuracy compared to other models
	Gradient Boosting Regression	Gradient Boosting Regression employs boosting to make predictive models from an ensemble of weak predictive learners	USE CASES 1. Predicting car emissions 2. Predicting rate holding time amount	1. Better accuracy compared to other regression models 2. It can handle multicollinearity 3. It can handle non-linear relationships
	XGBoost	Gradient Boosting algorithm that is efficient & flexible. Can be used for both classification and regression tasks	USE CASES 1. Churn prediction 2. Claims processing in insurance	1. Provides accurate results 2. Captures non-linear relationships
	LightGBM Regressor	A gradient boosting framework that uses a tree-based learning algorithm. LightGBM is designed to be more efficient than other implementations	USE CASES 1. Predicting flight time for airlines 2. Predicting cholesterol levels based on health data	1. Can handle large amounts of data 2. Computational efficient & fast training speed 3. Low memory usage
Unsupervised Learning	K-Means	K-Means is the most widely used clustering approach—it determines K clusters based on euclidean distances	USE CASES 1. Customer segmentation 2. Recommendation systems	1. Scales to large datasets 2. Simple to implement and interpret 3. Results in tight clusters
	Hierarchical Clustering	A "bottom-up" approach where each data point is treated as its own cluster—and then the closest two clusters are merged together iteratively	USE CASES 1. Fraud detection 2. Document clustering based on similarity	1. There is no need to specify the number of clusters 2. The resulting dendrogram is informative
	Gaussian Mixture Models	A probabilistic model for modeling normally distributed clusters within a dataset	USE CASES 1. Customer segmentation 2. Recommendation systems	1. Computes a probability for an observation belonging to a cluster 2. Can identify overlapping clusters 3. More accurate results compared to K-means
Association	Apriori algorithm	Rule-based approach that identifies the most frequent items in a given dataset where prior knowledge of frequent itemset properties is used	USE CASES 1. Product placements 2. Recommendation engines 3. Promotion optimization	1. Results are intuitive and interpretable 2. Exhaustive approach so it finds all rules based on the confidence and support 3. Results in many overlapping item sets

credit: datacamp

| **For Download [Click here](#)**

6. Deep Learning: Deep learning is a subfield of machine learning that uses artificial neural networks to model complex relationships in data. This section covers popular deep learning techniques such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

VIP Cheatsheet: Deep Learning

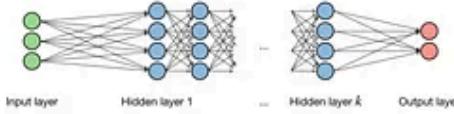
Afshine AMIDI and Shervine AMIDI

September 15, 2018

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

■ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

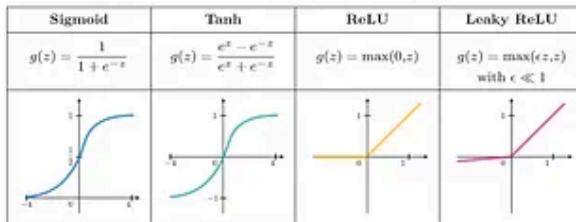


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} z + b_j^{[i]}$$

where we note w , b , z the weight, bias and output respectively.

■ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:



■ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$L(z,y) = -[y \log(z) + (1-y) \log(1-z)]$$

■ **Learning rate** – The learning rate, often noted η , indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

■ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

As a result, the weight is updated as follows:

$$w \leftarrow w - \eta \frac{\partial L(z,y)}{\partial w}$$

■ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

■ **Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1 - p$.

Convolutional Neural Networks

■ **Convolutional layer requirement** – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$N = \frac{W - F + 2P}{S} + 1$$

■ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Credit: stanford

| **For Download [Click here](#)**

7. Big Data: Big data refers to large and complex data sets that can't be effectively processed by traditional data processing techniques. This section covers popular big data tools such as Hadoop, Spark, and NoSQL databases.

HADOOP AND MAPREDUCE CHEAT SHEET

Hadoop & MapReduce Basics

HDFS

HDFS is a framework designed to handle large volume of structured and unstructured data.

- Security:**
 - Define users
 - Enable Kerberos in Hadoop
 - Set up X.509 gateway to control access and authentication to the HDFS cluster
- Groups:**
 - Define groups
 - Define HD's permissions
 - Define HDFS ACL's
- Audit:**
 - Enable process execution audit trail
 - Enable file encryption with Hadoop

MapReduce

MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of systems referred as clusters. Basically, it is a processing technique and program model for distributed computing based on Java.

Components of MapReduce

Mapper: The applications implement Map and Reduce functions and form the core of the job.

Reducer: Unit test framework for MapReduce

Parameters:

Parameter	Task
input directory/file name	Shows input location for the mapper
output directory-name	Shows output location for the mapper
mapper executable or script or .mrcfile-name	Used for Mapper executable
reducer executable or script or .mrclfile-name	Used for Reducer executable
file file name	Makes the mapper, reducer, combiner executable available locally on the computing nodes
numreducetasks	This is used to specify number of reducers
mapredlog	Script to call when the map task fails
reducereolog	Script to call when the reduce task fails

Commands used to interact with MapReduce

Command	Description
hadoop job -list -q [jobid]	Shows the details of the job
hadoop job -status -q [jobid]	Shows the status of the job
hadoop job -cancel -q [jobid]	Cancels the job
hadoop job -kill -q [jobid]	Kills the job
hadoop job -kill-task -taskid	Kills the task
hadoop job -kill-task -taskid	Kills the task
hadoop job -set-priority [jobid]	Changes and sets the priority of the job
--DUMP --kill/hadoop job -kill	This command kills the job created
--DUMP --history [jobid] --history	This is used to show the history of the jobs

IntelliPaat FURTHERMORE: Big Data Hadoop Certification Training Course

Credit: Intellipaat

For Download [Click here](#)

8. NLP: NLP (Natural Language Processing) is a subfield of artificial intelligence that focuses on the interaction between computers and humans using natural language. NLP involves the development of algorithms and models that enable computers to process, analyze, and understand human language. The goal of NLP is to make it possible for computers to understand and generate human language in a way that is both accurate and natural.

Cheatography

NLP Cheat Sheet

by sree017 via [cheatography.com/126402/cs/24446/](https://cheatography.com/sree017/cs/24446/)

Tokenization

```
Tokenization breaks the
raw text into words,
sentences called tokens.
These tokens help in
understanding the
context or developing
the model for the NLP.
... If the text is split
into words using some
separation technique it
is called word
tokenization and same
separation done for
sentences is called
sentence tokenization.
# NLTK
import nltk
nltk.download('punkt')
paragraph = "write
paragrapgh here to
convert into tokens."
sentences = nltk.sent-
_tokenize(paragraph)
words = nltk.word_token-
ize(paragraph)
# Spacy
from spacy.lang.en
import English
nlp = English()
sbd = nlp.create_pipe-
('sentencizer')
nlp.add_pipe(sbd)
doc = nlp(paragraph)
[sent for sent in
doc.sents]
nlp = English()
doc = nlp(paragraph)
```

Tokenization (cont)

```
[word for word in doc]
# Keras
from keras.preprocessing.text import text_to_w-
ord_sequence
text_to_word_sequence-
(paragraph)
# gensis
from gensim.summarizati-
on.textcleaner import
split_sentences
split_sentences(parag-
raph)
from gensim.utils import
tokenize
list(tokenize(para-
graph))
```

Bag Of Words & TF-IDF (cont)

```
X = cv.fit_transform(c-
ounters).toarray()
Term Frequency-Inverse
Document Frequency (TF-
IDF):
Term frequency-in-
verse document
frequency, is a
numerical statistic that
is intended to reflect
how important a word is
to a document in a
collection or corpus.
```

```
T.F = No of rep of
words in sentence/No of
words in sentence
```

```
IDF = No of sentences
/ No of sentences
containing words
from sklearn.feature_ex-
traction.text import
TfidfVectorizer
cv = TfidfVectorizer()
X = cv.fit_transform(c-
ounters).toarray()
N-gram Language Model:
An N-gram is a sequence
of N tokens (or words).
A 1-gram (or unigram) is
a one-word sequence.the
unigrams would simply
be: "I", "love",
"reading", "blogs",
"about", "data",
"science", "on", "Analy-
tics", "Vidhya".
```

Bag Of Words & TF-IDF (cont)

```
A 2-gram (or bigram) is
a two-word sequence of
words, like "I love",
"love reading", or
"Analytics Vidhya".
And a 3-gram (or
trigram) is a three-word
sequence of words like
"I love reading", "about
data science" or "on
Analytics Vidhya".
```

Stemming & Lemmatization

```
From Stemming we will
process of getting the
root form of a word. We
would create the stem
words by removing the
prefix or suffix of a
word. So, stemming a
word may not result in
actual words.
paragraph = ""
# NLTK
from nltk.stem import
PorterStemmer
from nltk import sent_t-
okenize
from nltk import word_t-
okenize
stem = PorterStemmer()
sentence = sent_tokeniz-
e(paragraph)[1]
words = word_tokenize(s-
entence)
[stem.stem(word) for
word in words]
# Spacy
```



By sree017
cheatography.com/sree017/

Published 26th September, 2020.
Last updated 26th September, 2020.
Page 1 of 3.

Sponsored by [CrosswordCheats.com](http://crosswordcheats.com)
Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

For Download [Click here](#)

9. SQL: SQL (Structured Query Language) is a programming language used to manage and manipulate data stored in relational databases. SQL is used to insert, update, and retrieve data in a database, as well as to create and modify database structures, such as tables and indexes.

What is SQL?

SQL stands for "structured query language". It is a language used to query, analyze, and manipulate data from databases. Today, SQL is one of the most widely used tools in data.

The different dialects of SQL

Although SQL languages all share a basic structure, some of the specific commands and syntax can differ slightly. Popular dialects include MySQL, SQLite, SQL Server, Oracle SQL, and more. PostgreSQL is a good place to start – since it's close to standard SQL syntax and is easily adopted to other dialects.

Sample Data

Throughout this cheat sheet, we'll use the columns listed in this sample table of `airbnb_listings`.

airbnb_listings				
ID	city	country	number_of_reviews	year_listed
1	Paris	France	5	2018
2	Tokyo	Japan	2	2019
3	New York	USA	2	2020

Querying tables

- Get all the columns from a table


```
SELECT *
        FROM airbnb_listings;
```
- Return the `city` column from the table


```
SELECT city
        FROM airbnb_listings;
```
- Get the `city` and `year_listed` columns from the table


```
SELECT city, year_listed
        FROM airbnb_listings;
```
- Get the listing (`id`, `city`), ordered by the `number_of_reviews` in ascending order


```
SELECT id, city
        FROM airbnb_listings
        ORDER BY number_of_reviews ASC;
```

Filtering Data

Filtering on numeric columns

- Get all the listings where `number_of_reviews` is more or equal to 3


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews >= 3;
```
- Get all the listings where `number_of_reviews` is more than 3


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews > 3;
```
- Get all the listings where `number_of_reviews` is exactly equal to 3


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews = 3;
```
- Get all the listings where `number_of_reviews` is lesser or equal to 3


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews <= 3;
```
- Get all the listings where `number_of_reviews` is lesser than 3


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews < 3;
```
- Get all the listings where `number_of_reviews` is between 3 and 5


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews BETWEEN 3 AND 5;
```

Filtering on text columns

- Get all the listings that are based in "Paris"


```
SELECT *
        FROM airbnb_listings
        WHERE city like '%Paris%';
```
- Get the listings based in the "USA" and is "France"


```
SELECT *
        FROM airbnb_listings
        WHERE country IN ('USA', 'France');
```
- Get all the listings where the city starts with "T" and where the city does not end in "T"


```
SELECT *
        FROM airbnb_listings
        WHERE city like 'T%' AND city NOT like '%T';
```

Filtering on multiple columns

- Get all the listings in `Paris` where `number_of_reviews` is bigger than 3


```
SELECT *
        FROM airbnb_listings
        WHERE city = 'Paris' AND number_of_reviews > 3;
```
- Get all the listings in `Paris` OR the ones that were listed after 2010


```
SELECT *
        FROM airbnb_listings
        WHERE city = 'Paris' OR year_listed > 2010;
```

Filtering on missing data

- Return the listings where `number_of_reviews` is missing


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews IS NULL;
```
- Return the listings where `number_of_reviews` is not missing


```
SELECT *
        FROM airbnb_listings
        WHERE number_of_reviews IS NOT NULL;
```

Aggregating Data

Simple aggregations

- Get the total number of rooms available across all listings


```
SELECT sum(number_of_reviews)
        FROM airbnb_listings;
```
- Get the average number of rooms per listing across all listings


```
SELECT avg(number_of_reviews)
        FROM airbnb_listings;
```
- Get the listing with the highest number of rooms across all listings


```
SELECT max(number_of_reviews)
        FROM airbnb_listings;
```
- Get the listing with the lowest number of rooms across all listings


```
SELECT min(number_of_reviews)
        FROM airbnb_listings;
```

Grouping, filtering, and sorting

- Get the total number of rooms for each country


```
SELECT country, sum(number_of_reviews)
        FROM airbnb_listings
        GROUP BY country;
```
- Get the average number of rooms for each country


```
SELECT country, avg(number_of_reviews)
        FROM airbnb_listings
        GROUP BY country;
```
- Get the listing with the maximum number of rooms per country


```
SELECT country, max(number_of_reviews)
        FROM airbnb_listings
        GROUP BY country;
```
- Get the listing with the lowest amount of rooms per country


```
SELECT country, min(number_of_reviews)
        FROM airbnb_listings
        GROUP BY country;
```
- Get each country, get the average number of rooms per listing, sorted by ascending order


```
SELECT country, avg(number_of_reviews) AS avg_rooms
        FROM airbnb_listings
        GROUP BY country;
```
- For Japan and the USA, get the average number of rooms per listing. In each country


```
SELECT country, avg(number_of_reviews)
        FROM airbnb_listings
        WHERE country IN ('USA', 'Japan')
        GROUP BY country;
```
- Get the number of cities per country, where there are listings


```
SELECT country, count(city) AS number_of_cities
        FROM airbnb_listings
        GROUP BY country;
```
- Get the sum of the rooms where there were more than 100 listings per year


```
SELECT year, sum(number_of_reviews)
        FROM airbnb_listings
        GROUP BY year;
```

Credit: datacamp

For Download [Click here](#)

10. Python: Python is a high-level programming language that is widely used for a variety of tasks, including web development, data analysis, artificial intelligence, and scientific computing. Python is known for its simple and expressive syntax, making it a popular choice for both beginners and experienced developers.

The cheat sheet is organized into several sections:

- Getting started with lists:** Includes creating lists, list functions and methods, and selecting list elements.
- Getting started with characters and strings:** Includes creating strings with double or single quotes, string methods, and combining and splitting strings.
- Getting started with dictionaries:** Includes creating dictionaries, dictionary functions and methods, and selecting dictionary elements.
- Getting started with DataFrames:** Includes creating DataFrames, creating DataFrame elements, and manipulating DataFrames.
- Other sections:** Accessing help and getting object types, importing packages, the working directory, operators, numeric comparison operators, and logical operators.

Each section provides examples and descriptions of Python syntax and functionality.

credit: datacamp

For Download [Click here](#)

11. R: R is a high-level programming language and software environment for statistical computing and graphics. R is widely used by statisticians, data scientists, and researchers for data analysis and visualization.

Getting started with vectors

Vectors are one-dimensional arrays that can hold numeric data, character data, or logical data. In other words, a vector is a simple tool to store data.

Creating vectors

Input	Output	Description
<code>c(1, 2, 3)</code>	<code>[1] 1 2 3</code>	Creates a vector using elements separated by commas
<code>1:7</code>	<code>[1] 1 2 3 4 5 6 7</code>	Creates a vector of integers between two numbers
<code>seq(2, 8, by = 2)</code>	<code>[1] 2 4 6 8</code>	Creates a vector between two numbers, with a step interval between each element
<code>rep(2, 8, times = 4)</code>	<code>[1] 2 2 2 2 2 2 2 2</code>	Creates a vector of given elements repeated a number of times
<code>rep(2, 8, each = 3)</code>	<code>[1] 2 2 2 2 2 2 2 2</code>	Creates a vector of given elements repeating each element a number of times.

Vector Functions

These functions perform operations over a whole vector.

- `length(x)`: Returns the length of a vector
- `rev(x)`: Reverses the elements of a vector
- `table(x)`: Returns a vector of the counts of values in a vector
- `unique(x)`: Returns unique elements in a vector

Selecting vector elements

These functions allow us to refer to particular parts of a vector.

- `x[1]`: Returns the first element of a vector
- `x[1:3]`: Returns the first three elements of a vector
- `x[-1]`: Returns all elements except the first
- `x[1:3] <- 10`: Changes the first three elements of a vector to 10
- `x[1:3] <- 10`: Changes the second and fourth elements of a vector to 10
- `x[x > 5] <- 10`: Changes elements less than 5 to 10
- `x[x <= 5] <- 10`: Changes elements less than or equal to 5 to 10
- `x[x < 5] <- 10`: Changes elements less than 5 to 10

Getting started with Data Frames in R

A data frame holds the variables of a data set as columns and the observations as rows.

Accessing rows

- `df[1]`: Returns the first row of `df`, even on the column
- `df[1:3]`: Returns rows 1 to 3 of `df`
- `df[-1]`: Returns all rows of the data frame except the first
- `df[1, 2]`: Returns the second value of the second column
- `df[1, -2]`: Returns all values of the second column except the second value
- `df[, 2]`: Returns the third column of the data frame
- `df[1,]`: Returns the first row of the data frame

Manipulating Data Frames in R

`dplyr` allows us to easily and precisely manipulate data frames. To use the following functions, you should install and load `dplyr` using `install.packages("dplyr")`.

- `filter(df, x == 10)`: Filters a row where `x` is equal to 10
- `select(df, -x)`: Selects all columns except `x`
- `arrange(df, -x)`: Arranges rows in ascending order based on the value of `x`
- `mutate(df, x = 10)`: Adds a new column `x` with value 10
- `summarise(df, total = sum(x))`: Calculates the sum of all values in `x`
- `group_by(df, category)`: Groups data by `category`
- `summarise(df, total = sum(x), n = n())`: Groups data by `category` and calculates the sum and count of `x` for each group
- `pull(df, x)`: Converts a data frame column into a vector
- `unnest(df)`: Converts a data frame column into multiple rows
- `nest(df)`: Converts multiple rows into a single row
- `do(df, mean(x))`: Creates a "summary" row of a total grouped by `category`. Other functions will then aggregate each "group" according to the results

Try this Cheat Sheet on DataCamp Workspace

Get Started →

credit: datacamp

For Download [Click here](#)

12. Numpy: NumPy is a Python library for numerical computing, specifically for arrays and matrices. It is a fundamental package for scientific computing with Python, and is widely used in data science, machine learning, and other technical fields.

Numpy

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

Use the following import convention:

```
%% Import numpy as np
```

NumPy Arrays

1D array 2D array 3D array

Creating Arrays

```
np.array([1,2,3])          # 1D array
np.array([[1,2,3],[4,5,6]]) # 2D array
np.array([[[1,2,3],[4,5,6]],[[7,8,9],[0,1,2]]]) # 3D array
```

I/O

Saving & Loading On Disk

```
np.savetxt('filename', a)    # Save array to disk
np.loadtxt('filename', a)    # Load array from disk
```

Saving & Loading Text Files

```
np.savetxt('textfile.txt', a, delimiter=',') # Save array to disk
np.loadtxt('textfile.txt', a, delimiter=',') # Load array from disk
```

Asking For Help

```
np.info(np.ndarray)
```

Inspecting Your Array

```
a.shape # array dimensions
a.dtype # array's data type
a.ndim # number of array dimensions
a.size # total number of array elements
a.itemsize # size of each array element
a.dtype.itemsize # itemsize in bytes
a.astype(DTYPE) # convert an array to a different type
```

Data Types

```
np.int_ # integer data type
np.float_ # floating-point data type
np.bool_ # boolean (single precision floating point)
np.str_ # string type (represented by 100 'floats')
np.bytes_ # string type (represented by 100 'bytes')
np.complex_ # complex floating-point type
np.datetime64 # datetime type
```

Array Mathematics

Arithmetic Operations

```
g = a + b # addition
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a + b # addition
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a - b # subtraction
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a * b # multiplication
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a * b # multiplication
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a / b # division
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a // b # floor division
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a % b # remainder
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a ** b # power
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
```

Comparison

```
a == b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a == b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a != b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a < b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a > b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a <= b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a >= b # element-wise comparison
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
```

Aggregate Functions

```
a.sum() # sum of array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.sum() # sum of array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.max() # maximum value of an array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.argmax() # index of max value
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.mean() # mean of array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.var() # variance of array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.std() # standard deviation
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
```

Copying Arrays

```
b = a.copy() # Create a copy of the array with the same data
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
b = a.copy() # Create a copy of the array
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
```

Sorting Arrays

```
a.argsort() # sort the elements of an array's axis
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
a.argsort() # sort the elements of an array's axis
array([[1, 2, 3], [4, 5, 6]])
array([[1, 2, 3], [4, 5, 6]])
```

Subsetting, Slicing, Indexing

Selecting

```
a[0] # extract the element at the 0th index
b # extract the element at index 0 column 0 (equivalent to a[0][0])
a[0,0]
```

Using

```
a[0,1] # extract slice at index 0 and 2
array([1, 2, 3])
array([1, 2, 3])
array[0,1,2] # extract slice at row 0 and 1, 0 columns
array([1, 2, 3])
array[0,1,2]
array[0,1,2]
array[0,1,2]
array[0,1,2]
array[0,1,2]
```

Boolean Indexing

```
a[a>1] # extract elements from a base mask
array([1, 2, 3])
array([1, 2, 3])
```

Parsing Indexing

```
arr[0, 1, 0, 2, 1, 0, 1, 0] # extract elements 0, 4, 8, 12, 16, 20, 24, 28
array([1, 2, 3])
array([1, 2, 3])
array[0, 1, 0, 2, 1, 0, 1, 0]
```

Reshaping

```
a.reshape(3, 2) # reshape array
array([1, 2, 3])
array([1, 2, 3])
array[3, 2]
array[3, 2]
```

Changing Shape

```
a.ravel() # flatten the array
array([1, 2, 3])
array[1, 2, 3]
```

Adding Newaxis

```
np.newaxis[None] # add a new axis with shape (1,1)
array([1, 2, 3])
array([1, 2, 3])
np.newaxis[None] # repeat along an axis
array([1, 2, 3])
array([1, 2, 3])
np.newaxis[None] # flatten array
array([1, 2, 3])
array([1, 2, 3])
array[None]
```

Copying Array

```
np.array([1, 2, 3]).copy() # create a copy of array
array([1, 2, 3])
array([1, 2, 3])
np.array([1, 2, 3]).copy() # make array vertically (row-wise)
array([1, 2, 3])
array([1, 2, 3])
array[1, 2, 3]
```

Flattening

```
arr.reshape(3, 2).ravel() # flatten the array horizontally on the first axis
array([1, 2, 3])
array[1, 2, 3]
arr.reshape(3, 2).ravel() # flatten the array vertically on the first axis
array([1, 2, 3])
array[1, 2, 3]
```

credit: datacamp

For Download [Click here](#)

13. Pandas: Pandas is a Python library for data manipulation and analysis. It provides data structures for efficiently storing large datasets and tools for working with them. Pandas is widely used in data science, machine learning, and other technical fields for tasks such as data cleaning, aggregation, and transformation.

datacamp

Python For Data Science

Data Wrangling in Pandas Cheat Sheet

Learn Data Wrangling online at www.DataCamp.com

Reshaping Data

Pivot

```
pd.melt(df, id_vars='Country', value_vars=['Alpha', 'Beta', 'Gamma'])
# Pivot more into columns
pd.pivot('Country', 'Alpha', 'Beta')
```

Pivot Table

```
pd.pivot_table(df, index='Country', values='Value', columns='Type')
```

Stack / Unstack

```
df.stack() # Stack a level of column labels
df.unstack() # Unstack a level of index labels
```

Melt

```
pd.melt(df, id_vars='Country', value_vars=['Alpha', 'Beta', 'Gamma'])
```

Iteration

```
for i, row in df.iterrows():
    print(i, row)
for i, row in df.iterrows():
    print(i, row['Alpha'])
```

Missing Data

```
df.dropna() # Drop all N/A values
df.fillna(0) # Fill N/A values with 0
df.replace('N/A', 0) # Replace values with others
```

Advanced Indexing

Also see NumPy Arrays

Selecting

```
df[0] # First row
df[0:2] # First two rows
df[-1] # Last row
df[0:-1] # All rows except last
df[::2] # Every second row
df[1::2] # Every second row starting at index 1
df[1:-1] # Every second row starting at index 1 and ending at index -1
df[[1, 3, 5]] # Specific rows
```

Indexing With Slices

```
df[0:2, 0:2] # Extract rows with index 0 and 1, columns with index 0 and 1
df[0, 0:2] # Extract row with index 0, columns with index 0 and 1
df[0:2, 1:2] # Extract rows with index 0 and 1, columns with index 1 and 2
df[0:2, -1] # Extract rows with index 0 and 1, last column
```

Where

```
df[df['Country'] == 'USA'] # Filter the data
```

Querying

```
df.query('second > first') # Using Boolean
```

Setting/Resetting Index

```
df.set_index('Country') # Set the index
df.reset_index(inplace=True) # Reset the index
df.set_index(['Country', 'Capital']) # Set multiple columns as index
df.reset_index(inplace=True)
```

Reindexing

```
df.reindex([1, 2, 3, 4]) # Forward Filling
df.reindex([1, 2, 3, 4], method='ffill')
df.reindex([1, 2, 3, 4], method='bfill') # Backward Filling
```

MultIndexing

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]], index=[1, 2, 3], columns=[1, 2, 3])
df.index.name = 'Country'
df.columns.name = 'Capital'
df['Population'] = [100000000, 200000000, 300000000]
df['Continent'] = ['Europe', 'Asia', 'Africa']
df.set_index(['Continent', 'Country'], inplace=True)
df['Population'] = df['Population'].apply(lambda x: x * 1000000000)
```

Combining Data

Merge

```
pd.merge(left=df1, right=df2, on='id', how='left')
pd.merge(left=df1, right=df2, on='id', how='right')
pd.merge(left=df1, right=df2, on='id', how='inner')
pd.merge(left=df1, right=df2, on='id', how='outer')
```

Join

```
pd.concat([df1, df2], axis=0)
```

Concatenate

```
pd.concat([df1, df2], axis=1)
```

Dates

```
pd.date_range('2020-01-01', '2020-01-03')
pd.date_range('2020-01-01', periods=3)
```

Also see Datetime

```
df['Date'] = pd.date_range('2020-01-01', periods=len(df))
df['Date'] = pd.date_range('2020-01-01', periods=3)
```

Also see Datetime

```
df['Date'] = pd.date_range('2020-01-01', periods=3)
df['Date'] = pd.date_range('2020-01-01', periods=3, freq='H')
```

Also see Datetime

```
df['Date'] = pd.date_range('2020-01-01', periods=3, freq='W')
```

Visualization

Also see Matplotlib

```
import matplotlib.pyplot as plt
plt.plot(df)
plt.show()
```

Credit: datacamp

For Download [Click here](#)

14. Seaborn: Seaborn is a data visualization library in Python, built on top of Matplotlib, that provides a high-level interface for creating statistical graphics. It is focused on the use of visualizations for exploring and understanding the structure of complex datasets.

1 Data

```
```python
Import pandas as pd
Import Seaborn as sns
Import matplotlib.pyplot as plt
sns.set_theme(style="white")
sns.set_color_codes("magma_r")
sns.set_context("paper", font_scale=1.5)
sns.set_style("whitegrid", {"font.family": "serif", "font.size": 16})
sns.set(rc={"figure.figsize": (10, 6)})

Create a random dataset
df = sns.load_dataset("tips")
```


Seaborn uses seaborn.set() to define global styling parameters.



2 Figure Aesthetics



```
```python
# Import Seaborn as sns
# Import Matplotlib's pyplot as plt
sns.set_theme(style="white")
sns.set_color_codes("magma_r")
sns.set_context("paper", font_scale=1.5)
sns.set_style("whitegrid", {"font.family": "serif", "font.size": 16})
sns.set(rc={"figure.figsize": (10, 6)})

# Create a random dataset
df = sns.load_dataset("tips")
```


3 Plotting With Seaborn

Axis Grids


```
```python
Import Seaborn as sns
Import Matplotlib's pyplot as plt
sns.set_theme(style="white")
sns.set_color_codes("magma_r")
sns.set_context("paper", font_scale=1.5)
sns.set_style("whitegrid", {"font.family": "serif", "font.size": 16})
sns.set(rc={"figure.figsize": (10, 6)})

Create a random dataset
df = sns.load_dataset("tips")
```


4 Further Customizations



Also see Matplotlib



### Axisgrid Objects



```
```python
# Import Seaborn as sns
# Import Matplotlib's pyplot as plt
sns.set_theme(style="white")
sns.set_color_codes("magma_r")
sns.set_context("paper", font_scale=1.5)
sns.set_style("whitegrid", {"font.family": "serif", "font.size": 16})
sns.set(rc={"figure.figsize": (10, 6)})

# Create a random dataset
df = sns.load_dataset("tips")
```


5 Show or Save Plot

Also see Matplotlib


```
```python
Import Seaborn as sns
Import Matplotlib's pyplot as plt
sns.set_theme(style="white")
sns.set_color_codes("magma_r")
sns.set_context("paper", font_scale=1.5)
sns.set_style("whitegrid", {"font.family": "serif", "font.size": 16})
sns.set(rc={"figure.figsize": (10, 6)})

Create a random dataset
df = sns.load_dataset("tips")
```

```


```


```


```


```

credit:datacamp

For Download [Click here](#)

15. Plotly Express: Plotly Express is a high-level data visualization library in Python, built on top of Plotly, that provides a simple and expressive way to create interactive, animated and publication-quality visualizations. It is designed to help users quickly create visualizations without writing too much code and focuses on providing a wide range of charts and options with sensible defaults.

What is plotly?

Plotly Express is a high-level data visualization package that allows you to create interactive plots with very little code. It is built on top of Plotly Graph Objects, which provides a lower-level interface for developing custom visualizations.

Interactive controls in Plotly

Plotly plots have interactive controls shown in the top-right of the plot. The controls allow you to do the following:

- Download plot as .png: Save your interactive plot as a static PNG.
- Zoom: Zoom in on a region of interest in the plot.
- Pan: Move around in the plot.
- Box Select: Select a rectangular region of the plot to be highlighted.
- Lasso Select: Draw a region of the plot to be highlighted.
- Autoscale: Zoom to a "best" scale.
- Reset axes: Return the plot to its original state.
- Toggle Spike Lines: Show or hide lines to the axes whenever you hover over data.
- Show closest data on hover: Show details for the nearest data point to the cursor.
- Compare data on hover: Show the nearest data point to the x coordinate of the cursor.

Plotly Express code pattern

The code pattern for creating plots is to call the plotting function, passing a data frame as the first argument. The x argument is a string naming the column to be used on the x-axis. The y argument can either be a string or a list of strings naming column(s) to be used on the y-axis.

```
px.plotting(pd.DataFrame, # DataFrame being visualized
           x["column-for-x-axis"], # Accepts a string or a list of strings
           y["columns-for-y-axis"], # Accepts a string or a list of strings
           title="Title", # Accepts a string
           width=1000, height=500, # Accepts an integer
           widthWidthInPixels, # Accepts an integer
           heightHeightInPixels) # Accepts an integer
```

Common plot types

Scatter plots

```
# Create a scatterplot on a DataFrame named clinical_data
px.scatter(clinical_data, x="experiment_1", y="experiment_2")
```

Set the size argument to the name of a numeric column to control the size of the points and create a bubble plot.

Line plots

```
# Create a lineplot on a DataFrame named stock_data
px.line(stock_data, x="date", y="price", width=2)
```

Set the line_color argument to the name of a categorical column to have dashes or dots for different lines.

Bar plots

```
# Create a barplot on a DataFrame named commodity_data
px.bar(commodity_data, x="category", y="value", color_discrete_map={"apple": "#f9a86a", "orange": "#ff7f0e", "banana": "#ffbb78", "grapes": "#98df8a"})
(px.bar).update_x_labels("Category")
(px.bar).update_y_labels("Value")
```

Swap the x and y arguments to draw horizontal bars.

Histograms

```
# Create a histogram on a DataFrame named billt_bill
px.histogram(billt_bill, x="bill", bins=5)
```

Set the nbins argument to control the number of bins shown in the histogram.

Heatmaps

```
# Create a heatmap on a DataFrame named iris_data
px.imshow(iris_data.corr(), xaxis="x", yaxis="y",
          zmin=-1, zmax=1, color_continuous_scale="rdvbu")
```

Set the text_auto argument to True to display text values for each cell.

Customizing markers in Plotly

When working with visualizations like scatter plots, lineplots, and more, you can customize markers according to certain properties. These include:

- size: set the marker size
- color: set the marker color
- opacity: set the marker transparency
- line: set the width and color of a border
- shape: set the shape of the marker

```
# In this example, we're updating a scatter plot named fig_scatter
fig_scatter.update_traces(marker={"size": 20,
                                    "color": "#ff7f0e",
                                    "opacity": 0.5,
                                    "line": {"dash": [2, 2], "color": "#ff7f0e"}, "shape": "square"})
```

Customizing lines in Plotly

When working with visualizations that contain lines, you can customize them according to certain properties. These include:

- color: set the line color
- dash: set the dash style ("solid", "dash", "longdash", "dashdot", "longdashdot")
- width: set the line width
- shape: set how values are connected ("linear", "spline", "vh", "vhv")

```
# In this example, we're updating a scatter plot named fig_line
fig_line.update_traces(line={"color": "#ff7f0e",
                             "shape": "spline",
                             "width": 4})
```

Customizing bars in Plotly

When working with barplots and histograms, you can update the bars themselves according to the following properties:

- size: set the marker size
- color: set the marker color
- opacity: set the marker transparency
- line: set the width and color of a border
- shape: set the shape of the marker

```
# In this example, we're updating a scatter plot named fig_bar
fig_bar.update_traces(marker={"color": "#ff7f0e",
                               "size": 100,
                               "line": {"dash": [2, 2], "color": "#ff7f0e"}, "shape": "square"})
```

In this example, we're updating a histogram named fig_hist
fig_hist.update_traces(marker={"color": "#ff7f0e",
 "size": 100,
 "line": {"dash": [2, 2], "color": "#ff7f0e"}, "shape": "square"})

Learn Data Skills Online at www.DataCamp.com

credit:datacamp

For Download [Click here](#)

16. Git: Git is a version control system for software development and code management. It allows developers to track changes made to code over time, collaborate on projects with other developers, and maintain different versions of the codebase. Git operates on a distributed model, meaning that multiple copies of a repository can exist on different machines, making it easy to work offline and share changes with others.

datacamp

Git Cheat Sheet

Learn Git online at www.DataCamp.com

What is Version Control?

Version control systems are tools that manage changes made to files and directories in a project. They allow you to keep track of what you did when, undo any changes you decide you don't want, and collaborate at scale with others. This cheat sheet focuses on one of the most popular ones, Git.

> Key Definitions

Throughout this cheat sheet, you'll find git-specific terms and jargon being used. Here's a run-down of the terms you might encounter:

- Local repo or repository:** A local directory containing code and files for the project.
- Remote repository:** An online version of the local repository located on services like GitHub, GitLab, and Bitbucket.
- Cleaning:** The act of moving a clone or copy of a repository in a new directory.
- Commit:** A snapshot of the project you can come back to.
- Branch:** A feature of Git that allows for working on an isolated environment without affecting the main project.
- HEAD:** The process of committing has become regular.

More advanced definitions:

- git commit -f**: If the file that you other files you want git not to track (e.g. large delta files, private info) and any binary files that should be seen by the public.
- git rebase**: Replaces the history of a branch with new commits.
- git stash**: Stores another type of cache that stores uncommitted changes you may want to come back later.
- Commit ID**: An hex or unique identifier for each commit, used for switching to different solve points.
- HEAD (tagged)**: Reference name for the latest commit, to save you having to type Commit ID. HEAD@{n} is used to refer to older commits (e.g. HEAD~2 refers to the second to last commit).

> Installing Git

On Mac & Linux: Using an installer
 1. Download the installer for Mac
 2. Follow the prompts
 On Linux:
 \$ sudo apt-get install git
 On Windows:
 1. Download the Visual Studio Windows Installer
 2. Follow the prompts

Check if installation successful (On any platform):
 \$ git --version

> Setting Up Git

If you are working in a team on a single repo, it is important for others to know who made certain changes to the code. So, Git allows you to set user credentials such as name, email, etc..

Set your basic information:

- Configure your email
 \$ git config user.name [your_email@example.com]
- Configure your name
 \$ git config user.name [your_name]

Important tags to determine the scope of configurations:

- Local directory, single project (this is the default tag):
 \$ git config --local user.name "key_email@example.com"
- All projects on the machine:
 \$ git config --global user.name "key_email@example.com"
- For all users on the current machine:
 \$ git config --system user.name "key_email@example.com"

Other useful configuration commands

- List all configurations
 \$ git config --list
- Show the value of a single tag
 \$ git config --get key

Setting aliases for common commands

If you find yourself using a command frequently, git lets you set an alias for that command to surface it more quickly.

- Create an alias named go for the "git checkout" command
 \$ git config --global alias.go checkout
- Go to a specific tag
 \$ git config --global alias.go checkout
- Create an alias named go for the "git add" command
 \$ git config --global alias.go add

> What is a Branch?

Branches are special types of the code base which allow you to work on different parts of a project and new features in an isolated environment. Changes made to the files in a branch will affect the "main branch" which is the main project development channel.

Commits working on a new branch
 Main branch
 New branch
 Experimental branch
 Work on experimental
 Merge the experiment

> Git Basics

What is a repository?

A repository is a storage location that stores code and the necessary files that allow it to run without errors. A repository can be local or remote. A local repo is typically a directory on your machine while a remote repo is hosted on servers like GitHub.

Creating local repositories

- Clone a repository from remote hosts (GitHub, GitLab, DeployHub, etc.)
 \$ git clone https://github.com/username/repo_name.git
- Initialize git tracking inside the current directory
 \$ git init
- Create a git-repo inside repository inside a new directory
 \$ git init [dir_name]
- Create a new branch
 \$ git branch [branch_name] & cd [branch_name]
- Changing into a specified directory
 \$ git clone [repo_url] & cd [dir_name]

What is a remote?

There are two primary methods of cloning a repository - HTTPS and SSH. While HTTPS cloning is generally considered to be more secure because git has to use an SSH key for authentication, HTTPS cloning is much simpler and the recommended cloning option for GitHub.

HTTPS

```
$ git clone https://github.com/username/repo_name.git
$ cd
$ git clone https://github.com/username/repo_name.git
$ cd
```

SSH

```
$ git clone git@github.com:username/repo_name.git
$ cd
```

Moving remote repositories

- Use git remote -v to see the URL
 \$ git remote -v
- Create a new connection called `remote2` to a remote repository on servers like GitHub, GitLab, DeployHub.
 \$ git remote add remote2 https://github.com/username/repo_name.git
- Remove a connection to a remote host called `remote1`
 \$ git remote rm remote1
- Remove a remote connection
 \$ git remote remove [remote_name]

Getting help

- Get help on a command
 \$ git help [command]
- Get help on a specific command
 \$ git help [command_name]

Setting up a local repository

- Initialize git tracking inside the current directory
 \$ git init
- Commit the first commit (including untracked changes)
 \$ git commit -m "Initial commit"
- Push the first commit to the remote repository
 \$ git push origin master
- Push the first commit to a specific branch
 \$ git push origin [branch_name]
- Push the first commit to a specific branch with a message
 \$ git push origin [branch_name] --force-with-lease
- Push the message of the most recent
 \$ git commit --amend --[commit message]
- Push the message of the most recent to a specific branch
 \$ git push origin [branch_name] --force-with-lease

Working With Files

Adding and removing files

- Add a file or directory to git tracking
 \$ git add [file_or_directory]
- Add all untracked and tracked files inside the current directory to git
 \$ git add .
- Remove the file from the current directory or skipping over
 \$ git rm [file_or_directory]
- Remove all files from the current directory or skipping over
 \$ git rm -r [directory]

Solving problems with changes

- See changes in the local repository
 \$ git status
- Seeing a snapshot of the staged changes with a custom message
 \$ git diff --staged --name-only "My changes are committed!"
- Showing changes in a remote file and committing with a message
 \$ git add . & git commit --message "My changes are committed!"
- Editing the message of the most recent
 \$ git commit --amend --[commit message]
- Push the message of the most recent to a specific branch (provide the URL of the branch)
 \$ git push origin [branch_name]

A note on stashes

Git stash allows you to temporarily save settings you made to your working copy so you can return to your work later. Throwing is especially useful when you are not yet ready to commit changes you've done, but would like to record them at a later time.

Branches

- Get all branches
 \$ git branch
- Get all branches --long
 \$ git branch --long
- Get all branches --all
 \$ git branch --all
- Create a temporary branch named `new_branch` without checking out that branch
 \$ git branch --new [branch_name]
- Switch to a existing branch named `new_branch`
 \$ git checkout [branch_name]
- Create a new branch and switch to it
 \$ git branch [branch_name] & git checkout [branch_name]
- Push adds a local branch (pushing during untracked changes)
 \$ git branch --track [branch_name]
- Push adds a local branch (pushing merged or unmerged)
 \$ git branch --no-track [branch_name]
- Compare the differences between two branches
 \$ git diff [branch_name1]..[branch_name2]
- Compare a single file between two branches
 \$ git diff [branch_name1]..[branch_name2]

Pulling changes

- Download all existing and new branches from the repository without switching them on the local repo
 \$ git pull
- Only download the specified branches from the remote
 \$ git fetch [branch_name]

Logging and reviewing work

- List all commits with their author, commit ID, date and message
 \$ git log
- List all commits like the log command, but also show the number of commits since the last commit
 \$ git log --count=10
- Log of commits with git statistics
 \$ git log --stat

Learn Data Skills Online at www.DataCamp.com

credit: datacamp

For Download [Click here](#)

17. PySpark: PySpark is a Python API for Apache Spark, an open-source, distributed computing system for big data processing and analysis. PySpark provides a way for Python developers to use Spark's powerful processing engine to process and analyze large datasets in parallel. It enables developers to scale out their computations and perform complex data transformations and aggregations, while leveraging the simplicity and expressiveness of Python programming.

datacamp

Python For Data Science

PySpark SQL Basics Cheat Sheet

Learn PySpark SQL online at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

```
from pyspark.sql import SparkSession
# SparkSession can be used to DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read/persist files.
# See https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql
spark = SparkSession.builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some value") \
    .getOrCreate()
```

Creating DataFrames

From RDDs

```
from pyspark.sql import SparkSession
# Infer Schema
# df = spark.read.csv('file.csv')
# df = spark.read.json('file.json')
# df = spark.read.parquet('file.parquet')
# df = spark.read.format('avro').load('file.avro')
# df = spark.createDataFrame(dataframe)
```

From Spark Data Sources

```
JSON
# df = spark.read.json('customer.json')
# df = spark.read.json('customer.json', schema=schema)

# df = spark.read.load('customer.json', format='json')

Parquet Files
# df = spark.read.load('customer.parquet')

TXT files
# df = spark.read.text('customer.txt')
```

Filter

```
# filter entries of age, only keep those records of which the values are > 20
df.filter(df['age'] > 20).show()
```

Duplicate Values

```
## df = df.dropDuplicates()
```

Queries

```
# from pyspark.sql import functions as F
# Selects
# df.select('name').alias('name') # Show all entries in firstName column
# df.select(['name','lastName']).alias('name') # Show all entries in firstName, lastName columns
# df.select(['name',F.sum('age')]).alias('name') # Show all entries in firstName, age and sum of age
# df.select(['name',F.sum('age')].alias('sum')) # Show all entries in firstName and sum of age
# df.select(['name',F.sum('age') + 10].alias('sum')) # Show all entries where age > 10
# When
# df.select(['name',F.when(F.col('age') < 10, F.col('age')).otherwise(F.col('age') + 10), F.col('age') + 10]).alias('name')
# df.select(['name',F.when(F.col('age') < 10, F.col('age')).otherwise(F.col('age') + 10), F.col('age') + 10]).alias('name')
# df.select(['name',F.when(F.col('age') < 10, F.col('age')).otherwise(F.col('age') + 10), F.col('age') + 10]).alias('name')
# Like
# df.select(['name']).where(firstName.like('J%')).alias('name')
# FirstName - EndWith
# df.select(['name']).where(firstName.endswith('n')).alias('name')
# df.select(['name']).where(firstName.endsWith('n')).alias('name')
# SubString
# df.select([F.substring(firstName, 1, 10).alias('name')]).alias('name')
# Between
# df.select([F.between(age, 20, 30).alias('age')]).alias('age')
```

Add, Update & Remove Columns

Adding Columns

```
# df = df.withColumn('city', F.when(address.state != 'IL', F.concat(F.col('city'), F.lit(',') , F.col('state'))))
# df = df.withColumn('phoneNumber', F.explode(F.col('phones').cast('array')).alias('phoneNumber'))
# df = df.withColumn('phoneType', F.explode(F.col('phones').cast('array')).alias('phoneType')))
```

Updating Columns

```
# df = df.withColumn('phoneNumber', F.col('phoneNumber').alias('phoneNumber'))
```

Removing Columns

```
# df = df.drop(F.col('phoneNumber'))
# df = df.drop(F.col('phoneNumber')).drop(F.col('phoneNumber'))
```

Missing & Replacing Values

```
# df = df.fillna(0).alias('df')
# df = df.dropna().alias('df')
# df = df.fillna(0, subset=[col1, col2]).alias('df')
```

GroupBy

```
# df.groupby('age').sum().alias('df')
```

Sort

```
# df.orderBy('age').alias('df')
# df.orderBy('age', ascending=False).alias('df')
# df.orderBy('age', 'name', ascending=False).alias('df')
```

Repartitioning

```
# df.repartition(100).alias('df')
```

Running Queries Programmatically

Registering DataFrames as Views

```
# df.registerTempTable('customer')
# df.createOrReplaceTempView('customer')
# df.createOrReplaceGlobalTempView('customer')
```

Query Views

```
# spark.sql('SELECT * FROM customer').alias('df')
# spark.sql('SELECT * FROM glect WHERE temp.emp_id = 1').alias('df')
```

Inspect Data

Data Structures

```
# df.printSchema() # prints schema of column names and data types
# df.show() # displays the content of df
# df.show(1) # displays the first row
# df.show(10) # displays the first n rows in df, where n is the number of columns before the column name of df
# df.count() # Count the number of rows in df
# df.distinctCount() # Count the number of distinct rows in df
# df.explain() # Explain the logical and physical plan
```

Output

Write & Save to File

```
# df.write.csv('output.csv') # write DataFrame as CSV file
# df.write.parquet('output.parquet') # write DataFrame as Parquet file
# df.write.json('output.json') # write DataFrame as JSON file
# df.write.orc('output.orc') # write DataFrame as ORC file
# df.write.orc('output.orc', compression='snappy') # write DataFrame as ORC file with snappy compression
```

Stopping SparkSession

```
# spark.stop()
```

Learn Data Skills Online at www.DataCamp.com

credit: datacamp

For Download [Click here](#)

18. Excel Cheat Sheet For Download [Click here](#)

19. Tableau Cheat Sheet For Download [Click here](#)

20. Power BI Cheat Sheet For Download [Click here](#)

Conclusion

In conclusion, the data science cheat sheet is a valuable resource for anyone looking to expand their knowledge in the field of data science. Whether you're a beginner or an experienced data scientist, the cheat sheet provides a

quick reference for all the essential concepts and tools used in the field.

Bookmark this cheat sheet and keep it handy as you work on your next data science project.

Thanks for Reading!

If you enjoyed this, [follow me](#) to never miss another article on data science guides, tricks and tips, life lessons, and more!

More content at [PlainEnglish.io](#). Sign up for our [free weekly newsletter](#). Follow us on [Twitter](#), [LinkedIn](#), [YouTube](#), and [Discord](#).

Interested in scaling your software startup? Check out [Circuit](#).

[Data Visualization](#)[Data Science](#)[Machine Learning](#)[Deep Learning](#)[Python](#)

Written by Ritesh Gupta

3.4K Followers · Writer for Artificial Intelligence in Plain English

[Follow](#)

Data Scientist, I write Article on Machine Learning| Deep Learning| NLP | Open CV |

AI Lover ❤️

More from Ritesh Gupta and Artificial Intelligence in Plain English



 Ritesh Gupta

10 Automated EDA Tools That Will Save You Hours Of Work

Exploratory Data Analysis (EDA) is the process of analyzing and summarizing the...

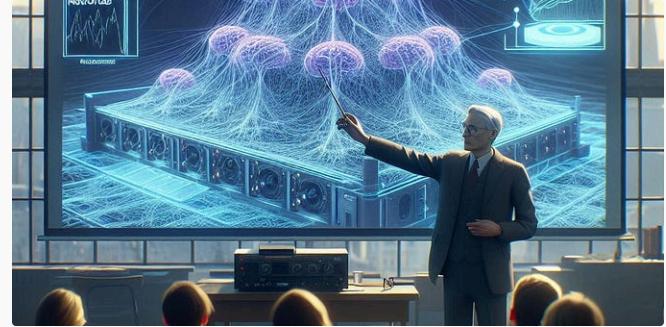
7 min read · Jan 29, 2023

 386

 6



•••



 Austin Star... in Artificial Intelligence in Plain Engli...

Reinforcement Learning is Dead. Long Live the Transformer!

Large Language Models are more powerful than you imagine

 · 8 min read · Jan 13, 2024

 2.1K

 49



•••



Kane Hoop... in Artificial Intelligence in Plain Engli...

Understanding AI Similarity Search

A guide to understanding similarity search (also known as semantic search), one of the...

12 min read · Apr 16, 2024

169

3



...



Ritesh Gupta

14 Life Changing Lessons From Chanakya Niti everyone should...

Who is Chanakya?

4 min read · Jan 29, 2023

108

1

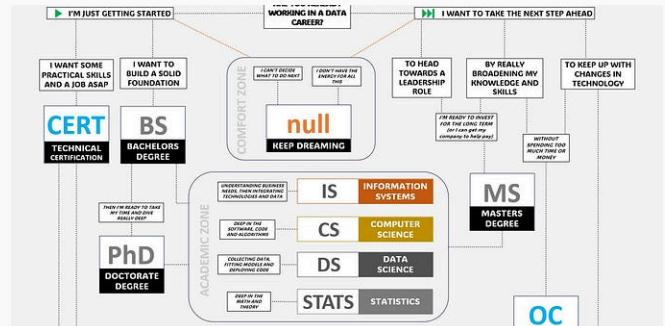
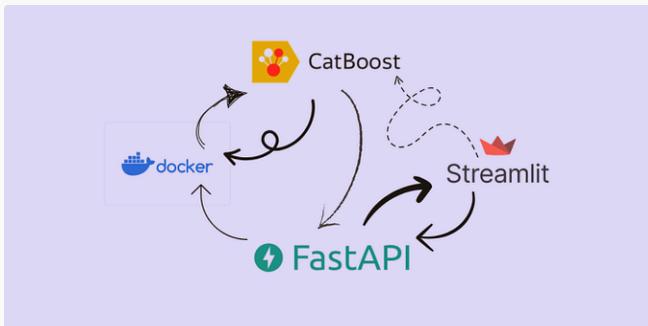


...

[See all from Ritesh Gupta](#)

[See all from Artificial Intelligence in Plain English](#)

Recommended from Medium





Ramazan Olmez



Stan Pugsley

End-to-End Machine Learning Project: Churn Prediction

The main objective of this article is to develop an end-to-end machine learning project. For...

18 min read · Feb 22, 2024



131



...

5 min read · Jan 31, 2024

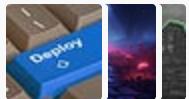


90



...

Lists



Predictive Modeling w/ Python

20 stories · 1134 saves



Coding & Development

11 stories · 581 saves



Practical Guides to Machine Learning

10 stories · 1360 saves



Natural Language Processing

1410 stories · 906 saves



 Nathan Rosidi

Data Science in 2024—What Has Changed

What has changed in the data science landscape, and what are the challenges of th...

4 min read · Jan 29, 2024

 1.3K  24

+ 

 Nilimesh Halder, PhD in Level Up Coding

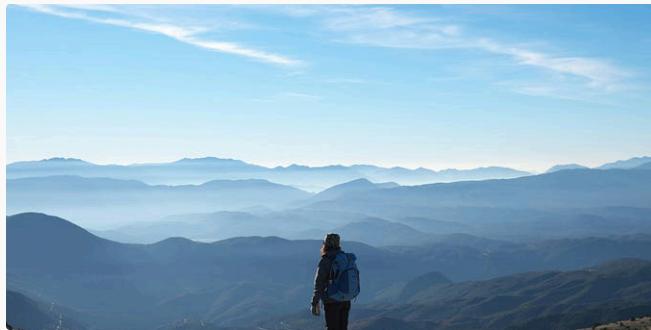
Unlocking Economic Insights with Exploratory Data Analysis

A Comprehensive Guide with Python and R

 24 min read · 6 days ago

 25 

+ 



 Sze Zhong LIM in Data And Beyond

Mastering Exploratory Data Analysis (EDA): Everything You...

A systematic approach to EDA your data and prep it for machine learning.

18 min read · Apr 6, 2024

 453  4

+ 

 Miriam Santos in Towards Data Science

A Data Scientist's Essential Guide to Exploratory Data Analysis

Best Practices, Techniques, and Tools to Fully Understand Your Data

11 min read · May 30, 2023

 1.3K  13

+ 

[See more recommendations](#)