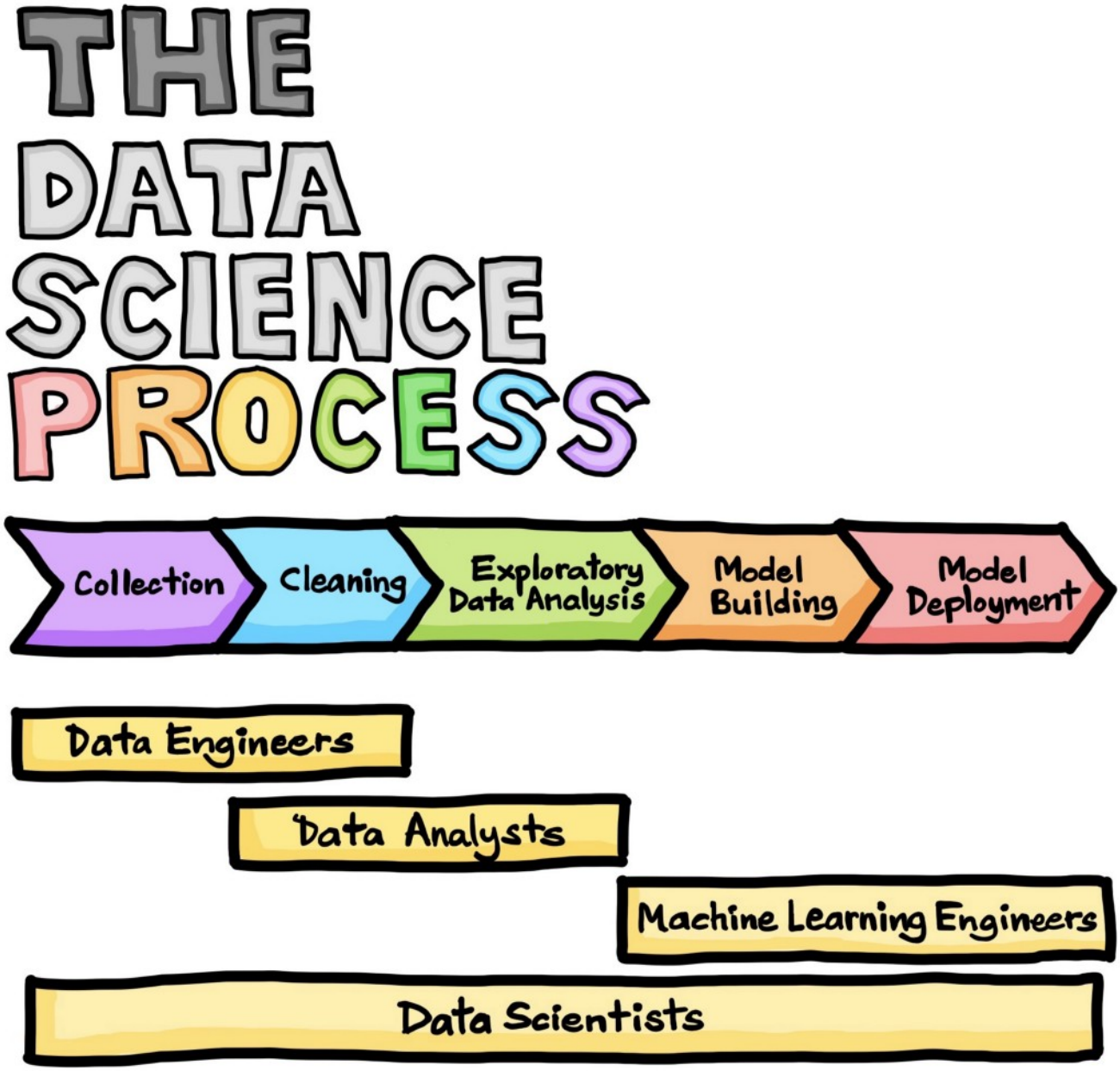


Data science life cycle. (Drawn by Chanin Nantasenamat in collaboration with Ken Jee)

DATA SCIENCE

The Data Science Process

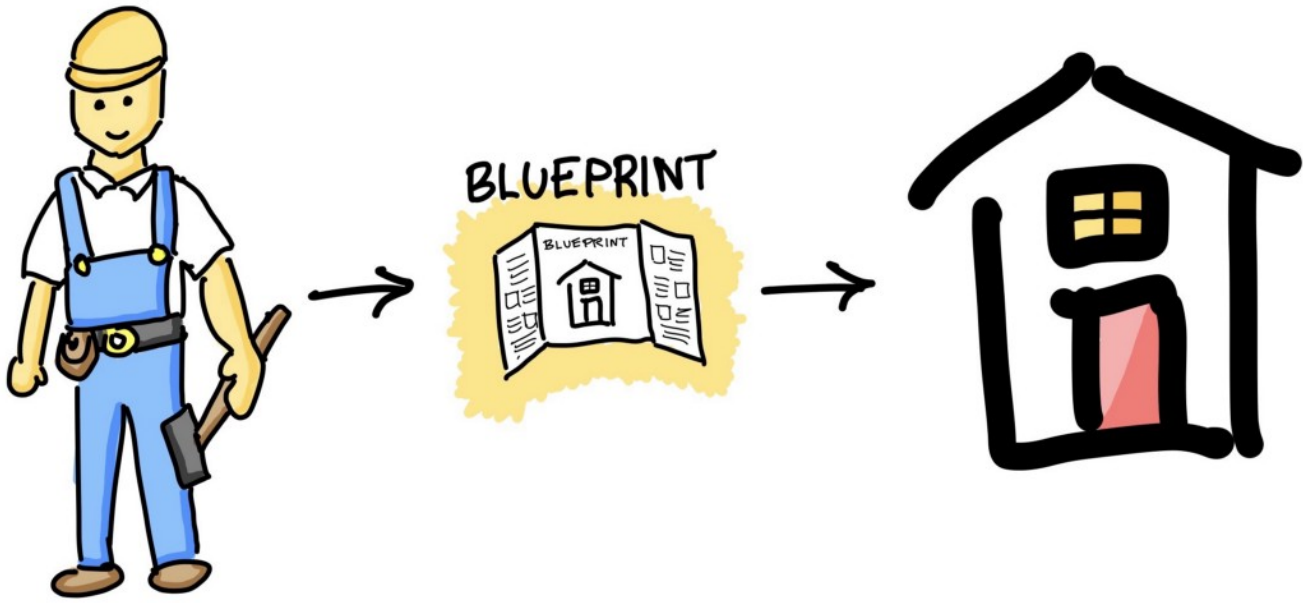
A Visual Guide to Standard Procedures in Data Science

Chanin Nantasenamat

Jul 28 · 7 min read ★

Let's suppose that you've been given a data problem to solve and you're expected to produce unique insights from the data given to you. So the question is, what do you exactly do to transform a data problem through to completion and generate data-driven insights? And most importantly of all, *Where do you start?*

Let's use some analogy here, in the construction of a house or building the guiding piece of information used is the blueprint. So what sorts of information are contained within these blueprints? Information pertaining to the building infrastructure, the layout and exact dimensions of each room, the location of water pipes and electrical wires, etc.

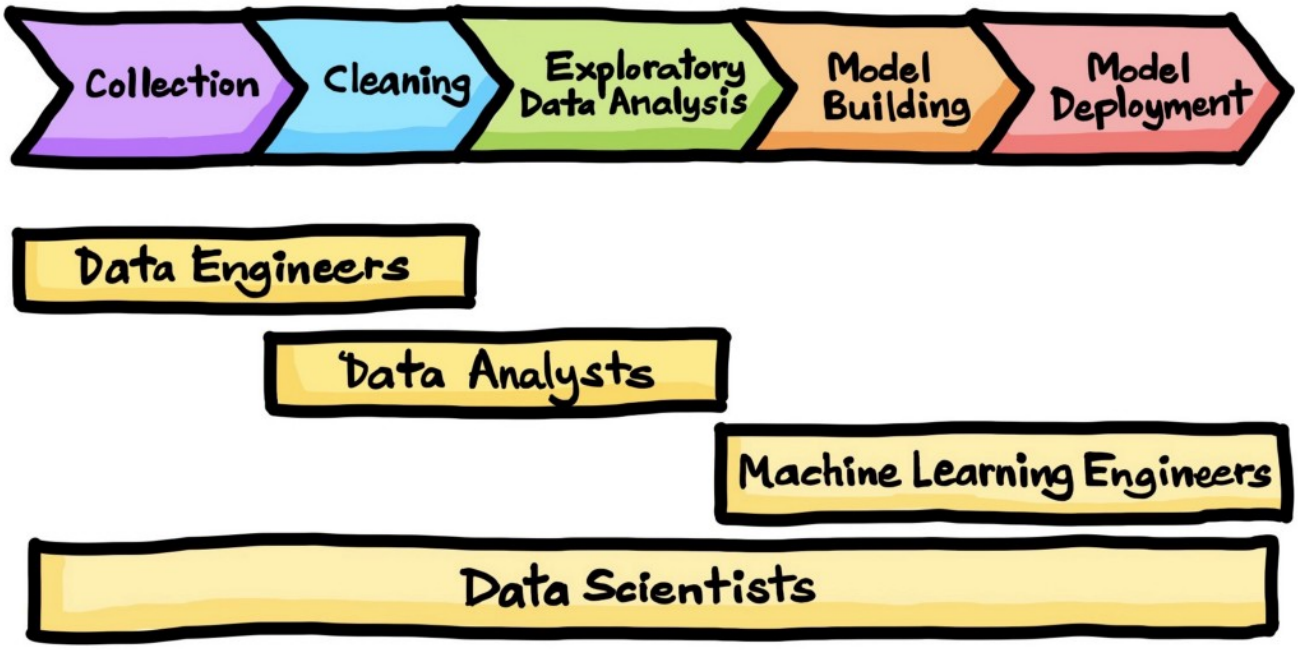


Continuing from where we left off earlier, so where do we start when given a data problem? That is where the *Data Science Process* comes in. As will be discussed in the forthcoming sections of this article, the data science process provides a systematic approach for tackling a data problem. By following through on these recommended guidelines, you will be able to make use of a tried-and-true workflow in approaching data science projects. So without further ado, let's get started!

Data Science Life Cycle

The *data science life cycle* is essentially comprised of data collection, data cleaning, exploratory data analysis, model building and model deployment. For more information, please check out the excellent video by Ken Jee on the *Different Data*

Science Roles Explained (by a Data Scientist). A summary infographic of this life cycle is shown below:



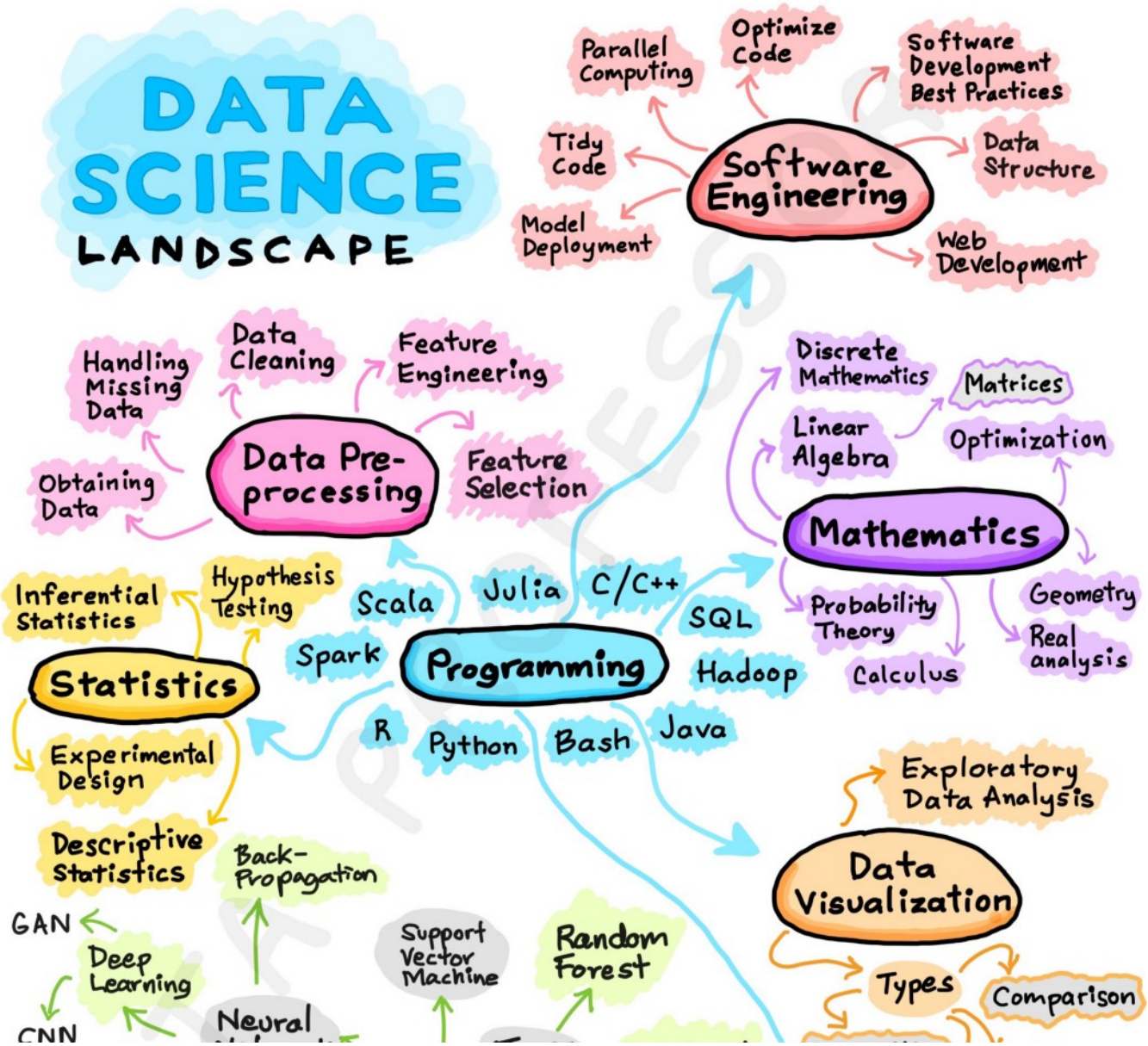
Data science life cycle. (Drawn by Chanin Nantasenamat in collaboration with Ken Jee)

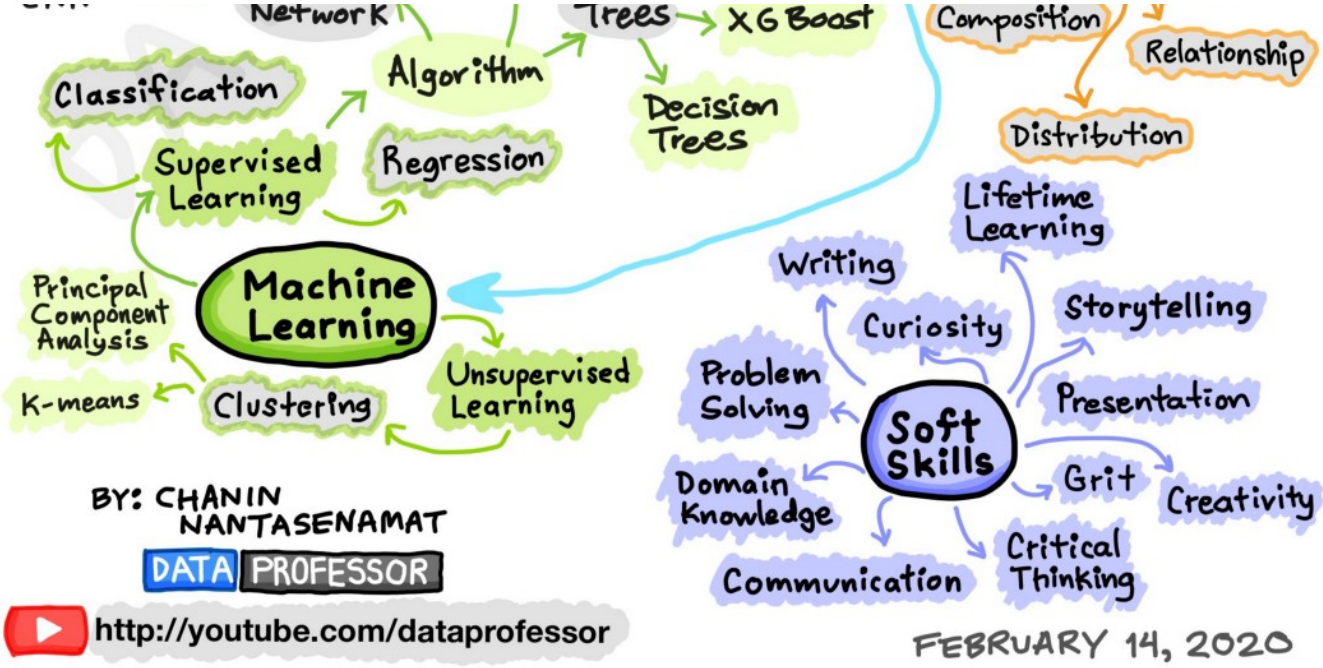
Such a process or workflow of drawing insights from data is best described by CRISP-DM and OSEMN. It should be noted that both are comprised of essentially the same core concepts while each framework was released at different time. Particularly, CRISP-DM was released at a time (1996) when data mining has started to gain traction

and was missing a standard protocol for carrying out data mining tasks in a robust manner. Fourteen years later (2010), the OSEMN framework was introduced and it summarizes the key tasks of a data scientist.

Personally, having started my own journey into the world of data in 2004 and the field was known back then as *Data Mining*. Much of the emphasis at the time was placed in translating data to knowledge where another common term that is also used to refer to data mining is *Knowledge Discovery in Data*.

Over the years, the field has matured and evolved to encompass other skillsets that led to the eventual coining of the term *Data Science* that goes beyond merely building models but also encompasses other skillsets both technical and soft skills. Previously, I have drawn an infographic that summarizes these 8 essential skillsets of data science as shown below. Also check out the accompanying YouTube video on *How to Become a Data Scientist (Learning Path and Skill Sets Needed)*.





8 Skillsets of Data Science. (Drawn by Chanin Nantasenamat)

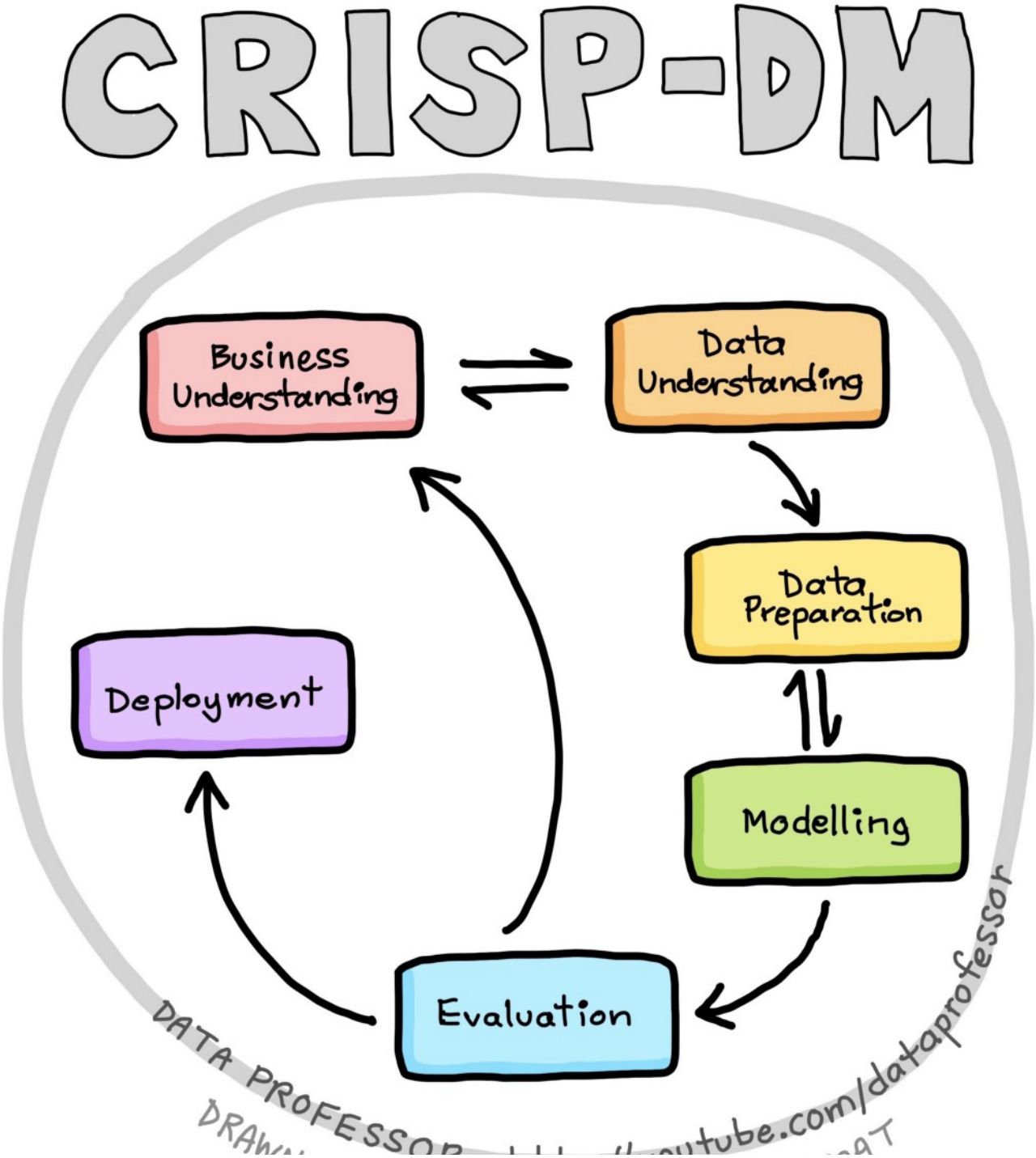
CRISP-DM

The acronym CRISP-DM stands for Cross Industry Standard Process for Data Mining and CRISP-DM was introduced in 1996 in efforts to standardize the process of data mining (also referred to as knowledge discovery in data) such that it can serve as a

standard and reliable workflow that can be adopted and applied in various industry. Such standard process would serve as a *"best practice"* that boasts several benefits.

Aside from providing a reliable and consistent of process by which to follow in carrying out data mining projects but it would also instill confidence to customers and stakeholders who are looking to adopt data mining in their organizations.

It should be noted that back in 1996, data mining had just started to gain mainstream attention and was at the early phases and the formulation of a standard process would help to lay the solid foundation and groundwork for early adopters. A more in-depth historical look of CRISP-DM is provided in the article by Wirth and Hipp (2000).



Standard process for performing data mining according to the CRISP-DM framework. (Drawn by Chanin Nantasenamat)

The **CRISP-DM framework** is comprised of 6 major steps:

1. ***Business understanding*** — This entails the understanding of a project's objectives and requirements from the business viewpoint. Such business perspectives are used to figure out what business problems to solve via the use of data mining.

2. ***Data understanding*** — This phase allows us to become familiarize with the data and this involves performing exploratory data analysis. Such initial data exploration may allow us to figure out which subsets of data to use for further modeling as well as aid in the generation of hypothesis to explore.

3. ***Data preparation*** — This can be considered to be the most time-consuming phase of the data mining process as it involves rigorous data cleaning and pre-processing as well as the handling of missing data.

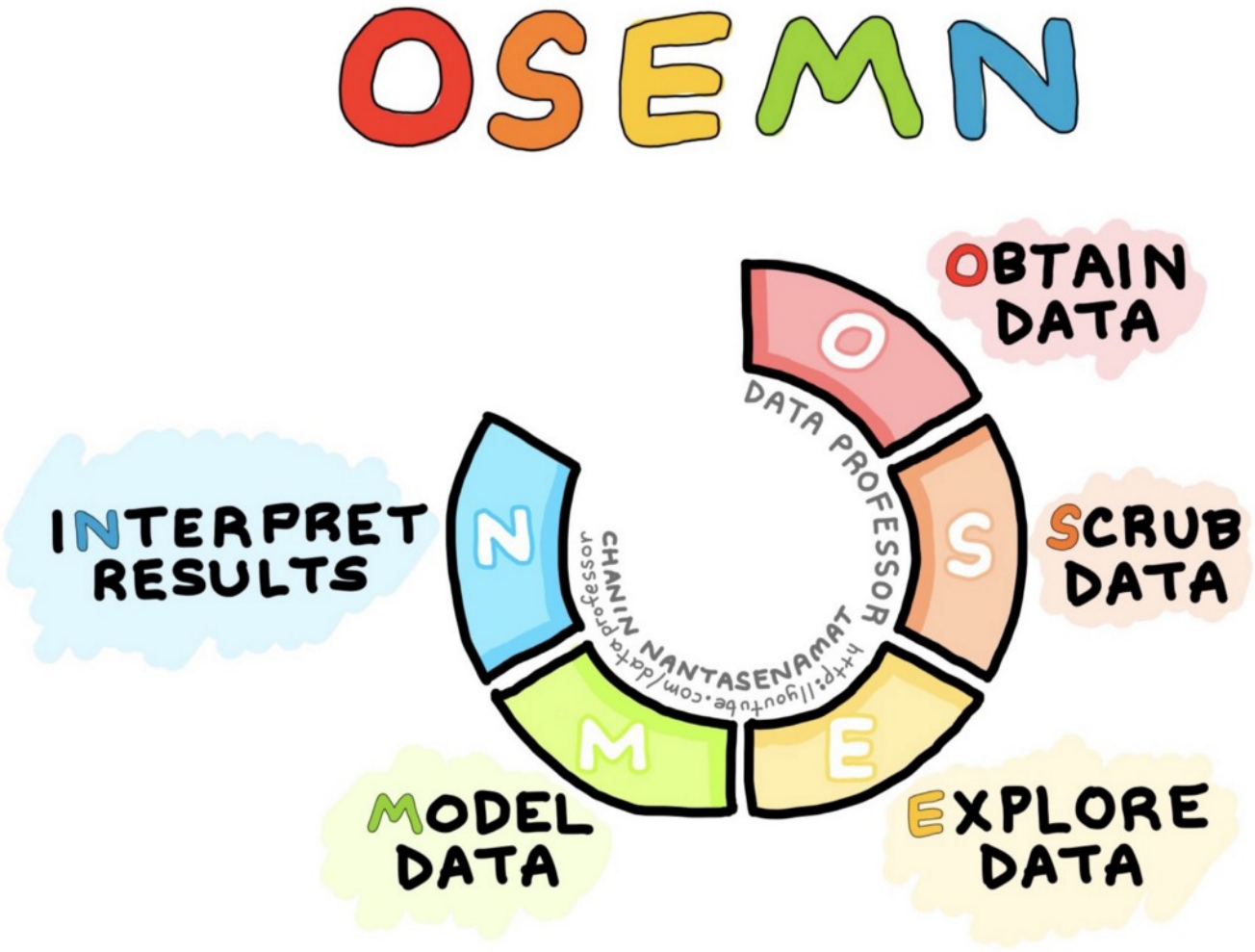
4. ***Modelling*** — The pre-processed data are used for model building in which learning algorithms are used to perform multivariate analysis.

5. ***Evaluation*** — In performing the 4 aforementioned steps, it is important to evaluate the accrued results and review the process performed thusfar to determine whether the originally set business objectives are met or not. If deemed appropriate, some steps may need to be performed again. Rinse and repeat. Once it is deemed that the results and process are satisfactory then we are ready to move to deployment. Additionally, in this evaluation phase, some findings may ignite new project ideas for which to explore.

6. ***Deployment*** — Once the model is of satisfactory quality, the model is then deployed, which may range from being a simple report, an API that can be accessed via programmatic calls, a web application, etc.

OSEMN

In a 2010 post ***"A Taxonomy of Data Science"*** on dataists blog, Hilary Mason and Chris Wiggins introduced the OSEMN framework that essentially constitutes a taxonomy of the general workflow that data scientists typically perform as shown in the diagram below. Shortly after in 2012, Davenport and Patil published their landmark article *"Data Scientist: The Sexiest Job of the 21st Century"* in the Harvard Business Review that has attracted even more attention to the burgeoning field of data science.



Data science process described in 5 steps by Mason and Higgins (2000) known as the OSEMN framework. (Drawn by Chanin Nantasenamat)

The **OSEMN framework** is comprised of 5 major steps and can be summarized as follows:

1. ***Obtain Data*** — Data forms the requisite of the data science process and data can come from pre-existing ones or from newly acquired data (from surveys), from newly queried data (from databases or APIs), downloaded from the internet (e.g. from repositories available on the cloud such as GitHub) or extracted

2. ***Scrub Data*** — Scrubbing the data is essentially data cleaning and this phase is considered to be the most time-consuming as it involves handling missing data as well as pre-processing it to be as error-free and uniform as possible.

3. ***Explore Data*** — This is essentially exploratory data analysis and this phase allows us to gain an understanding of the data such that we can figure out the course of actions and areas that we can to explore in the modeling phase. This entails the use of descriptive statistics and data visualizations.

4. ***Model Data*** — Here, we make use of machine learning algorithms in efforts to make sense of data and gain useful insights that are essential for data-driven decision-making.

5. ***Interpret Results*** — This is perhaps one of the most important phase and yet the least technical as it pertains to actually making sense of the data by figuring out how to simplify and summarize results from all the models built. This entails drawing meaningful conclusion and rationalizing actionable insights that would essentially allow us to figure out what the next course of actions are. For example, what are the most important features that influences the class labels (**Y** variables).

Conclusion

In summary, we have gone covered the data science process by showing you the highly simplified data science life cycle along with the widely popular CRISP-DM and OSEMN frameworks. These frameworks provides a high-level guidance on handling a data science project from end to end where all encompasses the same core concepts of data compilation, pre-processing, exploration, modeling, evaluation, interpretation and deployment. It should be noted that the flow amongst these processes is not linear and that in practice the flow can be non-linear and can re-iterate until satisfactory condition is met.

About Me

I work full-time as an Associate Professor of Bioinformatics and Head of Data Mining and Biomedical Informatics at a Research University in Thailand. In my after work

hours, I'm a YouTuber (AKA the Data Professor) making online videos about data science. In all tutorial videos that I make, I also share Jupyter notebooks on GitHub (Data Professor GitHub page).

Data Professor

Data Science, Machine Learning, Bioinformatics, Research and Teaching are my passion. The Data Professor YouTube...

www.youtube.com

Connect with Me on Social Network

- ☑ YouTube: http://youtube.com/dataprofessor/
- ☑ Website: http://dataprofessor.org/ (Under construction)
- ☑ LinkedIn: https://www.linkedin.com/company/dataprofessor/
- ☑ Twitter: https://twitter.com/thedataprof
- ☑ FaceBook: http://facebook.com/dataprofessor/
- ☑ GitHub: https://github.com/dataprofessor/
- ☑ Instagram: https://www.instagram.com/data.professor/

Task Cheatsheet for Almost Every Machine Learning Project

A checklist of tasks for building End-to-End ML projects

Harshit Tyagi
Jul 4 · 5 min read ★

Task Cheatsheet for Machine Learning Projects

To-do Checklist + Add a view Search New

#	Steps	Desc
1	Define the problem from a high-level view	Problem Statement
2	Get the data	Problem Statement
3	Initial Exploration of Data	Data Analysis
4	Exploratory Data Analysis to uncover hidden patterns and data transformation for ETL pipelines	Data Analysis Data Engineering
5	Analyse results of different models and shortlist the ones with good performance measure	Model Engineering Data Analysis
6	Fine-tune the shortlisted models check for ensemble methods	Model Engineering
7	Document code and communicate your solution	Data Analysis
8	Deploy and Monitor	Data Engineering

As I am working on creating a range of portfolio-worthy projects for all of you, I thought of documenting practices that I've either learned from someone or developed while working.

In this blog, I've captured the checklist of tasks that I keep referring to while working on an end-to-end ML project.

> *Why do I even need a checklist?*

Since you are required to deal with numerous elements in a project (wrangling, preparation, questions, models, fine-tuning, etc.), it's easy to lose track of things.

It guides you through the next steps and pushes you to check if every task has been executed successfully or not.

Sometimes, we struggle to find the starting point, the checklist helps you elicit the right information(data) from the right sources in order to establish relationships and uncover correlational insights.

It's a best practice to have every part of the project undergo a paradigm of checks.

As **Atul Gawande** says in his book — The Checklist Manifesto,

> the volume and complexity of what we know has exceeded our individual ability to deliver its benefits correctly, safely, or reliably.

So, let me walk you through this crisp and concise list of action items that will reduce your workload and enhance your output…

Machine Learning Project Checklist

These are 8–10 steps that you have to perform in almost every ML project. A few of the steps can be executed interchangeably in order.

1. Define the problem from a high-level view

This is to understand and articulate the business logic of the problem. It should tell you:

- the nature of the problem(supervised/unsupervised, classification/regression),
- type of solutions you can develop
- what metrics you should use to measure performance?

- is machine learning the right approach to solve this problem?
- manual approach to solving the problem.
- the inherent assumptions of the problem

2. Identify the data sources and acquire the data

In most cases, this step can be executed before the first step if you have the data with you and you want to define the questions(problem) around it to make better use of the incoming data.

Based on the definition of your problem, you'd need to identify the sources of data which can be a database, a data repository, sensors, etc. For an application to be deployed in production, this step should be automated by developing data pipelines to keep the incoming data flowing into the system.

- list the sources and amount of data you need.
- check if space is going to be an issue.
- check if you're authorized to use the data for your purpose or not.
- acquire the data and convert it into a workable format.
- check the type of data(textual, categorical, numerical, time series, images)
- take aside a sample of it for final testing purposes.

3. Initial Exploratoration of Data

This is the step where you study all the features that impact your outcome/prediction/target. If you have a huge chunk of data, sample it down for this step to make the analysis more manageable.
Steps to follow:

- use jupyter notebooks as they provide an easy and intuitive interface to study the data.
- identify the target variable
- identify the types of features(categorical, numerical, textual, etc.)
- analyze the correlation between features.

- add a few data visualizations for easy interpretation of the impact of each feature on the target variable.
- document your findings.

4. Exploratory Data Analysis to Prepare the Data

It's time to execute the findings of the previous step by defining functions for data transformations, cleaning, feature selection/engineering, and scaling.

- Write functions to transform the data and automate the process for the forthcoming batches of data.
- Write functions to clean the data(imputing missing values and handling outliers)
- Write functions to select and engineer features — drop redundant features, format conversion of features, and other mathematical transformations.
- Feature scaling — standardize the features.

5. Develop a baseline model and then explore other models to shortlist the best ones

Create a very basic model which should serve as the baseline for all the other complex machine learning model. Checklist of steps:

- Train a few commonly used ML models like naive bayes, linear regression, SVM, etc using default parameters.
- Measure and compare the performance of each model with the baseline and with all the others.
- Employ *N*-fold cross-validation for each model and compute the mean and standard deviation of the performance metrics on the *N* folds.
- Study the features that have the most impact on the target.
- Analyze the types of errors the models make while predicting.
- Engineer the features in a different manner.
- Repeat the above steps a few times(trial and error) to be sure that we have used the right features in the right format.
- Shortlist the top models based on their performance measures.

6. Fine-tune your shortlisted models and check for ensemble methods

This needs to be one of the crucial steps where you would be moving closer to your final solution. Major steps should include:

- Hyperparameter tuning using cross-validation.
- Use automated tuning methods like random search or grid search to find out the best configuration for your best models.
- Test ensemble methods like voting classifiers etc.
- Test the models with as much data as possible.
- Once finalized, use the unseen test sample that we set aside, in the beginning, to check for overfitting or underfitting.

7. Document Code and Communicate your solution

The process of communication is manifold. You need to keep in mind all the existing and potential stakeholders. Therefore the major steps include:

- Document the code as well as your approach and journey throughout your project.
- Create a dashboard like voila or an insightful presentation with close to self-explanatory visualizations.
- Write a blog/report capturing how you analyzed the features, tested different transformations, etc. Capture your learning(failures and techniques that worked)
- Conclude with the main outcome and future scope(if any)

8. Deploy your model in production, Monitor!

If your project requires deployment to be tested on live data, you should create a web application or a REST API to be used across all platforms(web, android, iOS). Major steps(would vary depending on the project) include:

- Save your final trained model into an h5 or pickle file.
- Serve your model using web services, you can use Flask to develop these web services.
- Connect the input data sources and set up the ETL pipelines.

- Manage dependencies using pipenv, docker/Kubernetes(based on scaling requirements)
- You can use AWS, Azure, or Google Cloud Platform to deploy your service.
- Monitor the performance on live data or simply for people to use your model with their data.

Note: The checklist can be adapted depending on the complexity of your project.

Data Science with Harshit

With this channel, I am planning to roll out a couple of series covering the entire data science space. Here is why you should be subscribing to the channel:

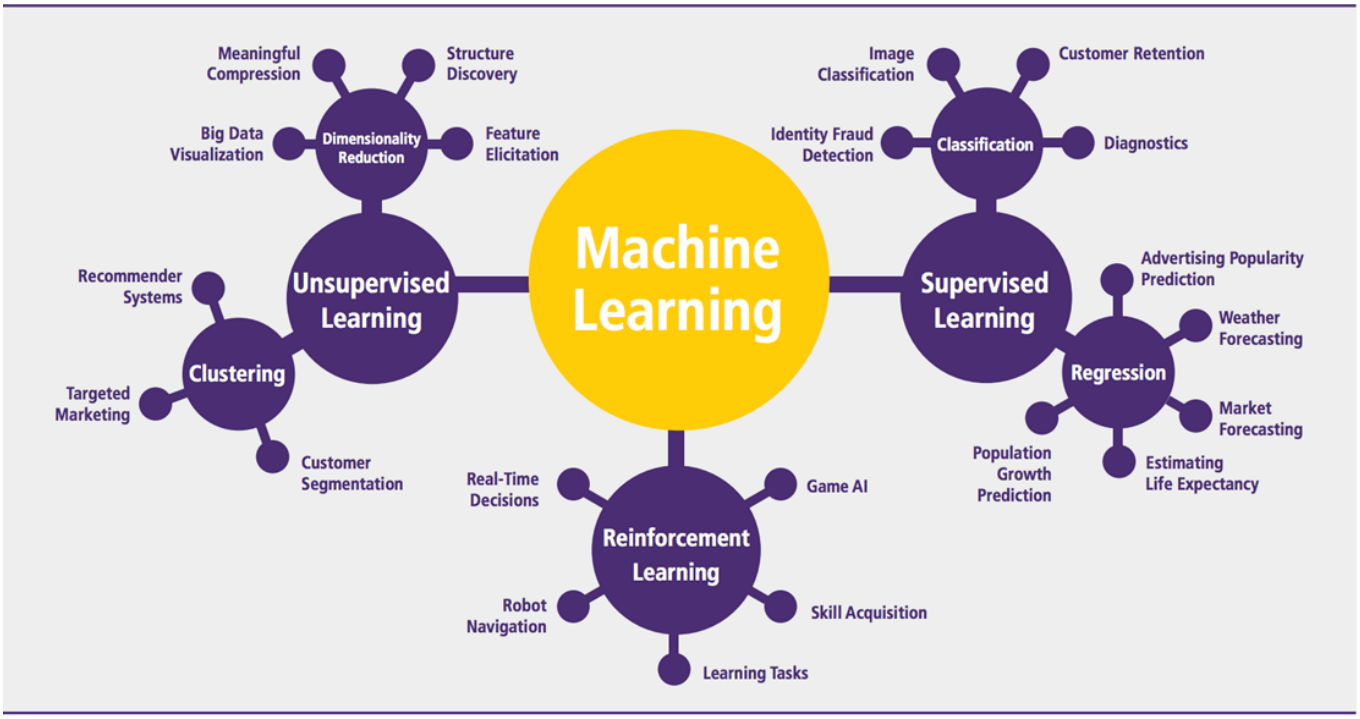
- These series would cover all the required/demanded quality tutorials on each of the topics and subtopics like Python fundamentals for Data Science.
- Explained Mathematics and derivations of why we do what we do in ML and Deep Learning.

- Podcasts with Data Scientists and Engineers at Google, Microsoft, Amazon, etc, and CEOs of big data-driven companies.
- Projects and instructions to implement the topics learned so far. Learn about new certifications, Bootcamp, and resources to crack those certifications like this **TensorFlow Developer Certificate Exam by Google.**

Analytics Vidhya

Which Machine Learning Algorithm Should You Use By Problem Type?

Sukanya Bag Follow
Oct 7 · 5 min read



When I was beginning my way in data science, I **often faced the problem** of **choosing the most appropriate algorithm** for my specific problem. If you're like me, when you open some article about machine learning algorithms, you see dozens of detailed descriptions. The **paradox** is that they **don't ease the choice.**

Well, to not let you feel out of the track, I would suggest you to have a good understanding of the implementation and mathematical intuition behind several supervised and unsupervised Machine Learning Algorithms like -

- **Linear regression**

- **Logistic regression**
- **Decision tree**
- **Naive Bayes**
- **Support vector machine**
- **Random forest**
- **AdaBoost**
- **Gradient-boosting trees**
- **Simple neural network**
- **Hierarchical clustering**
- **Gaussian mixture model**
- **Convolutional neural network**
- **Recurrent neural network**
- **Recommender system**

Remember, the list of Machine Learning Algorithms I mentioned are the ones that are mandatory to have a good knowledge of , while you are a beginner in Machine/Deep Learning !

Now that we have some intuition about types of machine learning tasks, let's explore the most popular algorithms with their applications in real life, based on their problem statements !

Try to work on each of these problem statements after getting to the end of this blog ! I can assure you would learn a lot, a hell lot!

Problem Statement 1 -

To Predict the Housing Prices

Machine Learning Algorithm(s) to solve the problem —

- **Advanced regression techniques like random forest and gradient boosting**

Problem Statement 2 -

Explore customer demographic data to identify patterns

Machine Learning Algorithm(s) to solve the problem —

- **Clustering (elbow method)**

Problem Statement 3 -

Predicitng Loan Repayment

Machine Learning Algorithm(s) to solve the problem —

- **Classification Algorithms for imbalanced dataset**

Problem Statement 4 -

Predict if a skin lesion is benign or malignant based on its characteristics (size, shape, color, etc)

Machine Learning Algorithm(s) to solve the problem —

- **Convolutional Neural Network (U-Net being the best for segmentation stuffs)**

Problem Statement 5 -

Predict client churn

Machine Learning Algorithm(s) to solve the problem —

- **Linear discriminant analysis** (LDA) or **Quadratic discriminant analysis** (QDA)

(particularly popular because it is both a classifier and a dimensionality reduction technique)

Problem Statement 6 -

Provide a decision framework for hiring new employees

Machine Learning Algorithm(s) to solve the problem —

- **Decision Tree** is a pro gamer here

Problem Statement 7 -

Understand and predict product attributes that make a product most likely to be purchased

Machine Learning Algorithm(s) to solve the problem **—**

- **Logistic Regression**
- **Decision Tree**

Problem Statement 8 -

Analyze sentiment to assess product perception in the market.

Machine Learning Algorithm(s) to solve the problem **—**

- **Naive Bayes — Support Vector Machines** (NBSVM)

Problem Statement 9 -

Create classification system to filter out spam emails

Machine Learning Algorithm(s) to solve the problem **—**

- **Classification Algorithms —**

Naive Bayes, SVM , Multilayer Perceptron Neural Networks (MLPNNs) and **Radial Base Function Neural Networks (RBFNN) suggested.**

Problem Statement 10 -

Predict how likely someone is to click on an online ad

Machine Learning Algorithm(s) to solve the problem **—**

- **Logistic Regression**
- **Support Vector Machines**

Problem Statement 11 -

Detect fraudulent activity in credit-card transactions.

Machine Learning Algorithm(s) to solve the problem —

- **Adaboost**
- **Isolation Forest**
- **Random Forest**

Problem Statement 12 -

Predict the price of cars based on their characteristics

Machine Learning Algorithm(s) to solve the problem —

- **Gradient-boosting trees are best at this.**

Problem Statement 13 -

Predict the probability that a patient joins a healthcare program

Machine Learning Algorithm(s) to solve the problem —

- **Simple neural networks**

Problem Statement 14 -

Predict whether registered users will be willing or not to pay a particular price for a product.

Machine Learning Algorithm(s) to solve the problem —

- **Neural Networks**

Problem Statement 15 -

Segment customers into groups by distinct charateristics (eg, age group)

Machine Learning Algorithm(s) to solve the problem —

- **K-means clustering**

Problem Statement 16 -

Feature extraction from speech data for use in speech recognition systems

Machine Learning Algorithm(s) to solve the problem —

- **Gaussian mixture model**

Problem Statement 17 -

Object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence.

Machine Learning Algorithm(s) to solve the problem —

- **Gaussian mixture model**

Problem Statement 18 -

Organizing the genes and samples from a set of microarray experiments so as to reveal biologically interesting patterns.

Machine Learning Algorithm(s) to solve the problem —

- **Hierarchical clustering algorithms**

Problem Statement 19 -

Recommend what movies consumers should view based on preferences of other customers with similar attributes.

Machine Learning Algorithm(s) to solve the problem —

- **Recommender system**

Problem Statement 20 -

Recommend news articles a reader might want to read based on the article she or he is reading.

Machine Learning Algorithm(s) to solve the problem —

- **Recommender system**

Problem Statement 21 -

Recommend news articles a reader might want to read based on the article she or he is reading.

Machine Learning Algorithm(s) to solve the problem —

- **Recommender system**

Problem Statement 22 -

Optimize the driving behavior of self-driving cars

Machine Learning Algorithm(s) to solve the problem —

- **Reinforcement Learning**

Problem Statement 23 -

Diagnose health diseases from medical scans.

Machine Learning Algorithm(s) to solve the problem —

- **Convolutional Neural Networks**

Problem Statement 24 -

Balance the load of electricity grids in varying demand cycles

Machine Learning Algorithm(s) to solve the problem —

- **Reinforcement Learning**

Problem Statement 25 -

When you are working with time-series data or sequences (eg, audio recordings or text)

Machine Learning Algorithm(s) to solve the problem —

- **Recurrent neural network**
- LSTM

Problem Statement 26 -

Provide language translation

Machine Learning Algorithm(s) to solve the problem —

- **Recurrent neural network**

Problem Statement 27 -

Generate captions for images

Machine Learning Algorithm(s) to solve the problem —

- **Recurrent neural network**

Problem Statement 28 -

Power chatbots that can address more nuanced customer needs and inquiries

Machine Learning Algorithm(s) to solve the problem —

- **Recurrent neural network**

I hope that I could explain to you common perceptions of the most used machine learning algorithms and give intuition on how to choose one for your specific problem.

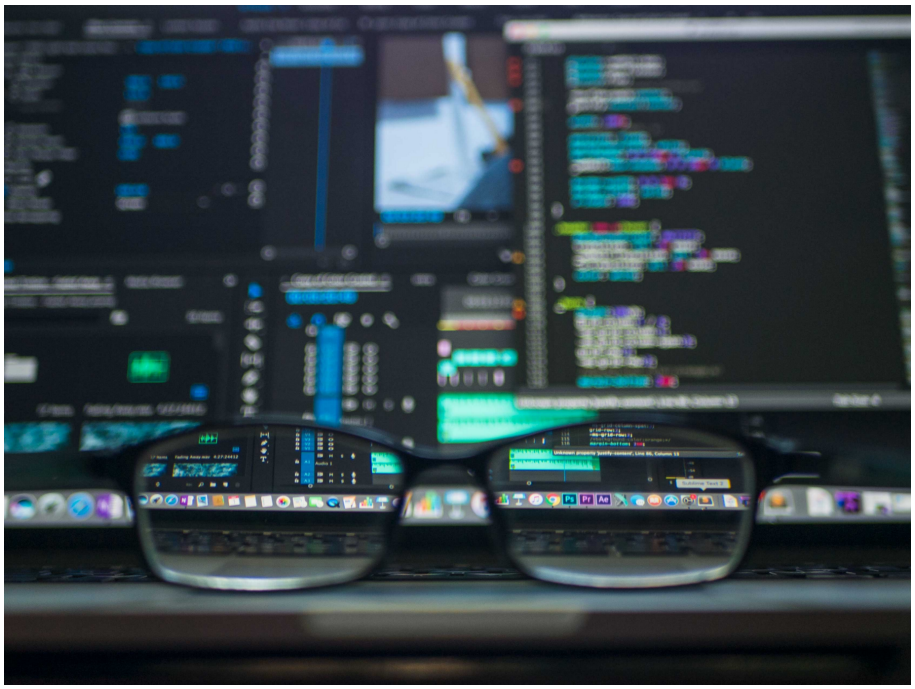
Happy Machine Learning ! :)

Until next time..!

All Machine Learning Models Explained in 6 Minutes

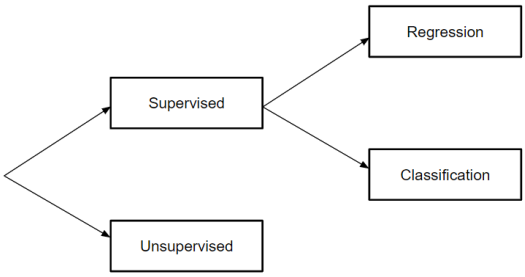
Intuitive explanations of the most popular machine learning models.

Terence S Follow
Jan 6 · 7 min read ★



If you like this, you should check out my free data science resource with new material every week!

In my previous article, I explained what **regression** was and showed how it could be used in application. This week, I'm going to go over the majority of common machine learning models used in practice, so that I can spend more time building and improving models rather than explaining the theory behind it. Let's dive into it.



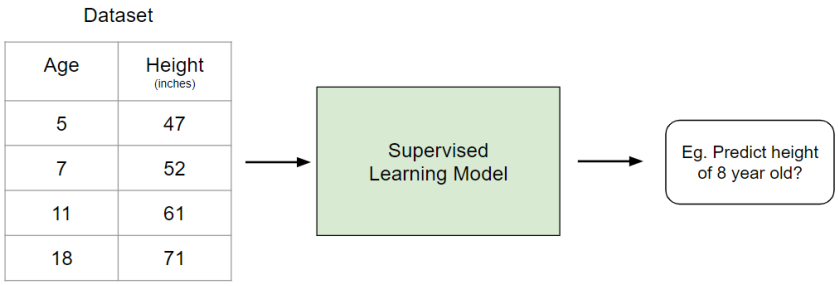
Fundamental Segmentation of Machine Learning Models

All machine learning models are categorized as either **supervised** or **unsupervised**. If the model is a supervised model, it's then sub-categorized as either a **regression** or **classification** model. We'll go over what these terms mean and the corresponding models that fall into each category below.

Supervised Learning

Supervised learning involves learning a function that maps an input to an output based on example input-output pairs [1].

For example, if I had a dataset with two variables, age (input) and height (output), I could implement a supervised learning model to predict the height of a person based on their age.



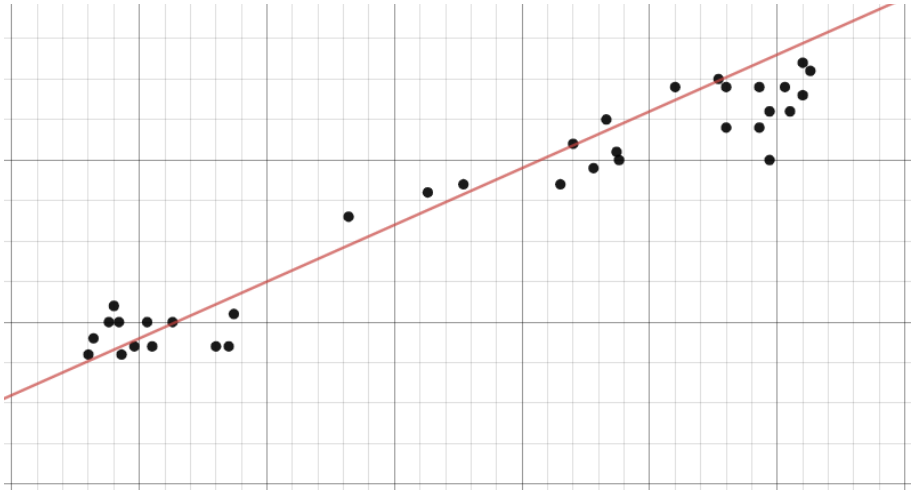
Example of Supervised Learning

To re-iterate, within supervised learning, there are two sub-categories: regression and classification.

Regression

In **regression** models, the output is continuous. Below are some of the most common types of regression models.

Linear Regression



Example of Linear Regression

The idea of linear regression is simply finding a line that best fits the data. Extensions of linear regression include multiple linear regression (eg. finding a plane of best fit) and polynomial regression (eg. finding a curve of best fit). You can learn more about linear regression in my previous article.

Decision Tree

Image taken from Kaggle

Decision trees are a popular model, used in operations research, strategic planning, and machine learning. Each square above is called a **node**, and the more nodes you have, the more accurate your decision tree will be (generally). The last nodes of the decision tree, where a decision is made, are called the **leaves** of the tree. Decision trees are intuitive and easy to build but fall short when it comes to accuracy.

Random Forest

Random forests are an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees

using bootstrapped datasets of the original data and randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all of the predictions of each decision tree. What's the point of this? By relying on a "majority wins" model, it reduces the risk of error from an individual tree.

For example, if we created one decision tree, the third one, it would predict 0. But if we relied on the mode of all 4 decision trees, the predicted value would be 1. This is the power of random forests.

StatQuest does an amazing job walking through this in greater detail. See here.

Neural Network

Visual Representation of a Neural Network

A Neural Network is essentially a network of mathematical equations. It takes one or more input variables, and by going through a network of equations, results in one or more output variables. You can also say that a neural network takes in a vector of inputs and returns a vector of outputs, but I won't get into matrices in this article.

The blue circles represent the **input layer,** the black circles represent the **hidden layers,** and the green circles represent the **output layer.** Each node in the hidden layers represents both a linear function and an activation function that the nodes in the previous layer go through, ultimately leading to an output in the green circles.

- If you would like to learn more about it, check out my beginner-friendly explanation on neural networks.

Classification

In classification models, the output is discrete. Below are some of the most common types of classification models.

Logistic Regression

Logistic regression is similar to linear regression but is used to model the probability of a finite number of outcomes, typically two. There are a number of reasons why logistic regression is used over linear regression when modeling probabilities of outcomes (see here). In essence, a logistic equation is created in such a way that the output values can only be between 0 and 1 (see below).

Support Vector Machine

A **Support Vector Machine** is a supervised classification technique that can actually get pretty complicated but is pretty intuitive at the most fundamental level.

Let's assume that there are two classes of data. A support vector machine will find a **hyperplane** or a boundary between the two classes of data that maximizes the margin between the two classes (see below). There are many planes that can separate the two classes, but only one plane can maximize the margin or distance between the classes.

If you want to get into greater detail, Savan wrote a great article on Support Vector Machines here.

Naive Bayes

Naive Bayes is another popular classifier used in Data Science. The idea behind it is driven by Bayes Theorem:

In plain English, this equation is used to answer the following question. "What is the probability of y (my output variable) given X? And because of the naive assumption that variables are independent given the class, you can say that:

As well, by removing the denominator, we can then say that P(y|X) is proportional to the right-hand side.

Therefore, the goal is to find the class y with the maximum proportional probability.

Check out my article "A Mathematical Explanation of Naive Bayes" if you want a more in-depth explanation!

Decision Tree, Random Forest, Neural Network

These models follow the same logic as previously explained. The only difference is that that output is discrete rather than continuous.

Unsupervised Learning

Unlike supervised learning, **unsupervised learning** is used to draw inferences and find patterns from input data without references to labeled outcomes. Two main methods used in unsupervised learning include clustering and dimensionality reduction.

Clustering

Taken from GeeksforGeeks

Clustering is an unsupervised technique that involves the grouping, or **clustering**, of data points. It's frequently used for customer segmentation, fraud detection, and document classification.

Common clustering techniques include **k-means** clustering, **hierarchical** clustering, **mean shift** clustering, and **density-based** clustering. While each technique has a different method in finding clusters, they all aim to achieve the same thing.

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables [2]. In simpler terms, its the process of reducing the dimension of your feature set (in even simpler terms, reducing the number of features). Most dimensionality reduction techniques can be categorized as either **feature elimination** or **feature extraction.**

A popular method of dimensionality reduction is called **principal component analysis.**

Principal Component Analysis (PCA)

In the simplest sense, **PCA** involves project higher dimensional data (eg. 3 dimensions) to a smaller space (eg. 2 dimensions). This results in a lower dimension of data, (2 dimensions instead of 3 dimensions) while keeping all original variables in the model.

There is quite a bit of math involved with this. If you want to learn more about it…

Check out this awesome article on PCA here.

If you'd rather watch a video, StatQuest explains PCA in 5 minutes here.

Conclusion

Obviously, there is a ton of complexity if you dive into any particular model, but this should give you a fundamental understanding of how each machine learning algorithm works!

> *For more articles like this one, check out https://blog.datatron.com/*

References

[1] Stuart J. Russell, Peter Norvig, Artificial Intelligence: A Modern Approach (2010), Prentice Hall

[2] Roweis, S. T., Saul, L. K., Nonlinear Dimensionality Reduction by Locally Linear Embedding (2000), *Science*

Thanks for Reading!

If you like my work and want to support me…

1. The BEST way to support me is by following me on **Medium** here.
2. Follow me on **LinkedIn** here.
3. Sign up on my **email list** here.
4. Want to collaborate? Check out my website, **shintwin.com**

My top 4 functions to style the Pandas Dataframe

Why you need to stylish your pandas Dataframe and how to do it

Cornellius Yudha Wijaya

May 21 · 6 min read ★

Photo by Paweł Czerwiński on Unsplash

Pandas Dataframe is the most used object for Data scientists to analyze their data. While the main function is to just place your data and get on with the analysis, we could still style our data frame for many purposes; namely, for **presenting data** or **better aesthetic**.

Let's take an example with a dataset. I would use the 'planets' data available from seaborn for learning purposes.

```
#Importing the modules
import pandas as pd
import seaborn as sns

#Loading the dataset
planets = sns.load_dataset('planets')

#Showing 10 first row in the planets data
planets.head()
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009
5	Radial Velocity	1	185.840	4.80	76.39	2008
6	Radial Velocity	1	1773.400	4.64	18.15	2002
7	Radial Velocity	1	798.500	NaN	21.41	1996
8	Radial Velocity	1	993.300	10.30	73.10	2008
9	Radial Velocity	2	452.800	1.99	74.79	2010

We could style our previous Pandas Data Frame just like below.

10 top data of Planets Dataset

method	number	orbital_period	mass	distance	year
Radial Velocity	1	269.3	7.1	77.4	2006
Radial Velocity	1	874.8	2.2	57.0	2008
Radial Velocity	1	763.0	2.6	19.8	2011
Radial Velocity	1	326.0	19.4	110.6	2007
Radial Velocity	1	516.2	10.5	119.5	2009
Radial Velocity	1	185.8	4.8	76.4	2008
Radial Velocity	1	1773.4	4.6	18.1	2002
Radial Velocity	1	798.5	nan	21.4	1996
Radial Velocity	1	993.3	10.3	73.1	2008
Radial Velocity	2	452.8	2.0	74.8	2010

It looks colorful right now because I put everything inside but I would break it down some of my favorite. Here are 4 functions to style our Pandas Data Frame object that I often use in everyday work.

> "While the main function is to just place your data and get on with the analysis, we could still style our data frame for many purposes; namely, for presenting data or better aesthetic."

. . .

1. Hiding Function

Sometimes when you do analysis and presenting the result to the other, you only want to show the most important aspect. I know when I present my Data Frame to the non-technical person, the question is often about the Index in their default number such as "what is this number?". For that reason, we could try to **hide the index** with the following code.

```
#Using hide_index() from the style function
```

```
planets.head(10).style.hide_index()
```

method	number	orbital_period	mass	distance	year
Radial Velocity	1	269.300000	7.100000	77.400000	2006
Radial Velocity	1	874.774000	2.210000	56.950000	2008
Radial Velocity	1	763.000000	2.600000	19.840000	2011
Radial Velocity	1	326.030000	19.400000	110.620000	2007
Radial Velocity	1	516.220000	10.500000	119.470000	2009
Radial Velocity	1	185.840000	4.800000	76.390000	2008
Radial Velocity	1	1773.400000	4.640000	18.150000	2002
Radial Velocity	1	798.500000	nan	21.410000	1996
Radial Velocity	1	993.300000	10.300000	73.100000	2008
Radial Velocity	2	452.800000	1.990000	74.790000	2010

Just like that, we hide our index. It is a simple thing but in the working environment I know sometimes it would become a problem. Just one extra column that becomes a question.

In addition, we could try to **hide unnecessary columns** with the chaining method. Let's say I don't want to show the 'method' and 'year' columns then we could write it with the following code.

```
#Using hide_columns to hide the unnecesary columns

planets.head(10).style.hide_index().hide_columns(['method','year'])
```

number	orbital_period	mass	distance
1	269.300000	7.100000	77.400000
1	874.774000	2.210000	56.950000
1	763.000000	2.600000	19.840000
1	326.030000	19.400000	110.620000
1	516.220000	10.500000	119.470000
1	185.840000	4.800000	76.390000

1	185.840000	4.800000	76.390000
1	1773.400000	4.640000	18.150000
1	798.500000	nan	21.410000
1	993.300000	10.300000	73.100000
2	452.800000	1.990000	74.790000

. . .

2. Highlight Function

There is a time when we want to present our data frame and only highlight the important number, for example the **highest number**. In this case, we could use the built-in method to highlight it with the following code.

```
#Highlight the maximum number for each column

planets.head(10).style.highlight_max(color = 'yellow')
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

In the data frame above, we highlight the maximum number in each column with the color yellow. If you want to highlight the **minimum number** instead, we could do it with the following code.

```
planets.head(10).style.highlight_min(color = 'lightblue')
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

and if you want to chain it, we could also do that.

```
#Highlight the minimum number with lightblue color and the maximum
number with yellow color

planets.head(10).style.highlight_max(color='yellow').highlight_min(c
olor = 'lightblue')
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996

8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

Instead of each column, you could actually highlight the minimum or maximum number for **each row**. I show it in the following code.

```
#Adding Axis = 1 to change the direction from column to row

planets.head(10).style.highlight_max(color = 'yellow', axis =1)
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

As we can see, it is useless right now to change the axis as it did not highlight any important information. It would be more useful in the case when each column is not that different from each other.

As an addition, we could highlight the **null value** with the following code.

```
#Higlight the null value

planets.head(10).style.highlight_null(null_color = 'red')
```

	method	number	orbital_period	mass	distance	year

0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

3. Gradient Function

While presenting your data, we could also use all the information as the main way to present the data. I often present the data with a background color to highlight which number is in the lower area and where is the one in the higher area. Let's use the example by the following code.

```
#Gradient background color for the numerical columns

planets.head(10).style.background_gradient(cmap = 'Blues')
```

With the background_gradient function, we could color the data frame as a gradient. The color would depend on the cmap parameter where the parameter is accepting colormaps from the matplotlib library.

We could also use a **bar chart** as our gradient background color. Let me show it in the example below.

```
#Sort the values by the year column then creating a bar chart as the background

planets.head(10).sort_values(by = 'year').style.bar(color= 'lightblue')
```

	method	number	orbital_period	mass	distance	year
7	Radial Velocity	1	798.500000	nan	21.410000	1996
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011

As we could see above, we now highlight the number from the lowest to the highest number in a different way than the background_gradient function is. We could also see the index is not in order because of the sort function; it is better to hide the index as I told you in the passage above.

4. Custom Function

If you prefer to have a more specific requirement to style your data frame, you could actually do it. We could pass our style functions into one of the following methods:

- `Styler.applymap`: element-wise
- `Styler.apply`: column-/row-/table-wise

Both of those methods take a function (and some other keyword arguments) and apply our function to the DataFrame in a certain way. Let's say that I have a threshold that any number below 20 should be colored red. We could do that by using the following code.

```
#Create the function to color the numerical value into red color

def color_below_20_red(value):
    if type(value) == type(''):
        return 'color:black'
    else:
        color = 'red' if value <= 20 else 'black'
        return 'color: {}'.format(color)

#We apply the function to all the element in the data frame by using
the applymap function

planets.head(10).style.applymap(color_below_20_red)
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.100000	77.400000	2006
1	Radial Velocity	1	874.774000	2.210000	56.950000	2008
2	Radial Velocity	1	763.000000	2.600000	19.840000	2011
3	Radial Velocity	1	326.030000	19.400000	110.620000	2007
4	Radial Velocity	1	516.220000	10.500000	119.470000	2009
5	Radial Velocity	1	185.840000	4.800000	76.390000	2008
6	Radial Velocity	1	1773.400000	4.640000	18.150000	2002
7	Radial Velocity	1	798.500000	nan	21.410000	1996
8	Radial Velocity	1	993.300000	10.300000	73.100000	2008
9	Radial Velocity	2	452.800000	1.990000	74.790000	2010

Just like that, every single number below or equal to 20 would be colored red and the rest is in black.

. . .

Conclusion

I have shown my top 4 functions to use when styling our data frame; hiding functions. We could style our data frame for presenting our data or just a better aesthetic. If you want to read more about what we can do to style our data frame, you could read it here.

Data Science | Python | Machine Learning | Programming | Education

9 pandas visualizations techniques for effective data analysis

Learn how to use line graphs, scatter plots, histograms, boxplots, and a few other visualization techniques using pandas library only

Magdalena Konkiewicz
May 14 · 7 min read ★

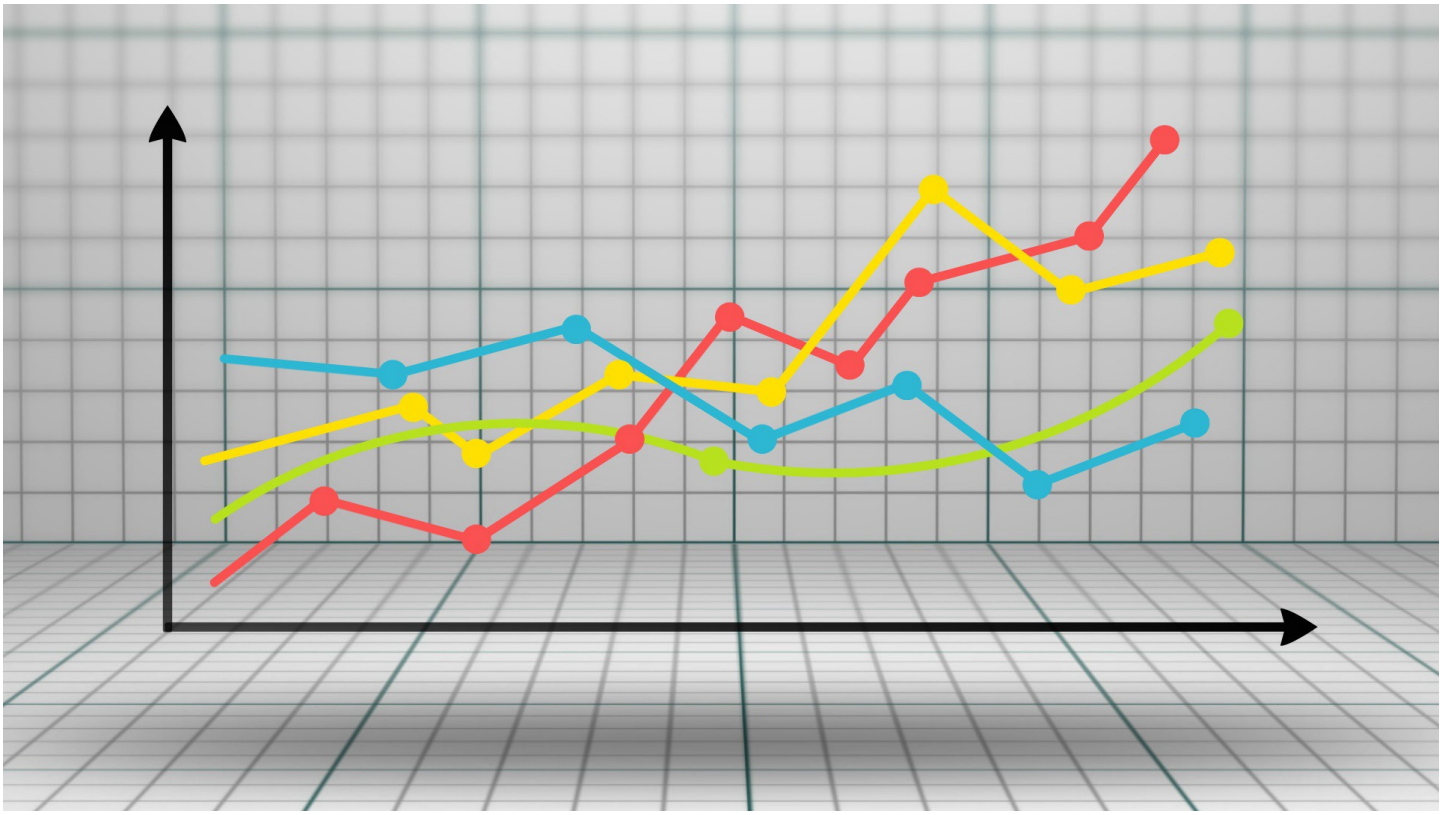


Image by Mediamodifier from Pixabay

Introduction

In this article, I would like to present you with nine different visualization techniques that will help you analyze any data set. Most of these techniques require just one line of code. We all love simplicity, don't we?

You will learn how to use:

line plot,

scatter plot,

area plot,

bar chart,

piechart,

histogram,

kernel density function,

boxplot,

and scatter matrix plot.

We will discuss all of the above visualization techniques, explore different ways of using them, and learn how to customize them to suit a data set.

Let's get started.

. . .

Load data set and quick data exploration

For simplicity purposes, we will use the Iris data set that can be loaded from a scikit-learn library using the following code:

```
from sklearn.datasets import load_iris
import pandas as pd
```

```
data = load_iris()
df = pd.DataFrame(data['data'], columns=data['feature_names'])
df['species'] = data['target']
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

As you can see we have a data set with five columns only. Let's call info() function on the data frame for quick analysis:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
species              150 non-null int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

As you can see there are only 150 entries, there are no missing values in any of the columns.

Additionally, we have learnt that the first four columns have float values whereas the last column allows integers only. In fact from the data set description we know that *species* column will take only three values each one representing one type of flower.

To confirm this you can call the unique() function on that column:

```
df.species.unique()
array([0, 1, 2])
```

Indeed *species* column take only three values: 0, 1, and 2.

Knowing this basic information about our data set we can proceed to visualizations. Note that if there were some missing values in the columns you should either drop them or fill them in. This is because some of the techniques we will discuss later on will not allow for missing values.

. . .

Line plot

We are going to start our visualizations with a simple line plot. Let's just call it on the whole data frame.

```
df.plot()
```


As you can see here it has plotted all the column values in different colours against the index value (x-axis). This is the default behaviour when we do not supply the x-axis parameter for the function.

As you can see this plot is not very useful. The line plot would be a good choice if the x-axis was a time series. Then we could probably see some trends in time within data.

In our case, we can only see that the data is ordered by *species* column (purple steps in the graph) and that some other columns have a moving average that follows that

pattern (petal length especially that is marked in red).

. . .

Scatter plot

The next type of visualisation we are going to discover is a scatter plot. This a perfect type of plot to visualise correlations between two continuous variables. Let's demonstrate it by plotting sepal length against sepal width.

```
df.plot.scatter(x='sepal length (cm)', y='sepal width (cm)')
```


As you can see in order to produce this graph you need to specify x and y-axis for the plot by supplying its column names. This graph reveals that there is no strong correlation between the two variables. Let's examine a different pair, *sepal length* and *petal length*:

```
df.plot.scatter(x='sepal length (cm)', y='petal length (cm)')
```

In this case, we can see that when sepal length increases petal length increases as well (it is stronger for values of sepal length larger than 6 cm).

. . .

Area plot

Let's create an area plot for the data frame. I will include all dimensions with centimetres in my plot but remove the *species* column as it will not make sense to include it in our case.

```
columns = ['sepal length (cm)', 'petal length (cm)', 'petal width
(cm)', 'sepal width (cm)']
df[columns].plot.area()
```

The measurements on this graph are stuck one on top of each other. Looking at this chart you can visually examine the ratio between each measurement that is included in the graph. You can see that all sizes have a growing trend towards the later entries.

. . .

Bar chart

This is the good type of a graph to include when showing averages or counts of entries. Let's use it to compute averages for each dimension for each species in our data set.

In order to do it, you will need to use a groupby() and mean() function. I am not going to explain how they work in detail here but you can check this article that explains these concepts.

```
df.groupby('species').mean().plot.bar()
```

As you can see this is very straight forward to read. I can see that there are differences in average measurements for different species and different columns.

. . .

Piechart

You can use a pie chart in order to visualize the class count for your target variable. We will do it here for the Iris data set we are working on. Again we will need some helper functions. This time it is groupby() and count():

```
df.groupby('species').count().plot.pie(y='sepal length (cm)')
```

As you can see we have perfect proportions for our classes as our data set consist of 50 entries for each class.

Note that we had to use *y* parameter here and set it to some column name. We have used *sepal length* column here but it could be any column as the counts are the same for all of them.

. . .

Histogram

This is a perfect visualization for any continuous variable. Let's start with simple hist() function.

```
import matplotlib.pyplot as plt
df.hist()
plt.tight_layout()
```

As you can see this produces a histogram for each numeric variable in the data set.

I had to add some extra lines of code in order to customize the graph. This is the first import line and the last line where I call tight_layout() function. If this is not added the labels and subgraph names may overlap and not be visible.

. . .

Kernel density function

In a similar way as a histogram you can use kernel density function:

```
df.plot.kde(subplots=True, figsize=(5,9))
```

You can see that it gives similar results to the histogram.

We had to specify a figure size here as without it the graphs were squashed vertically too much. Also, we had set subplots parameter to *True* as by default all columns would be displayed on the same graph.

. . .

Boxplot

Another visualisation that should be used for numerical variables. Let's create boxplots for all measurement columns (we are excluding *species* column as the box plot does not make sense for this categorical variable):

```
columns = ['sepal length (cm)', 'petal length (cm)', 'petal width
(cm)', 'sepal width (cm)']
df[columns].plot.box()
plt.xticks(rotation='vertical')
```

As you can see all boxplots are drawn on the same plot. This is fine in our case as we do not have too many variables to visualise.

Note that we had to rotate the x labels as without it the names of the labels were overlapping with each other.

. . .

Scatter matrix plot

This is one of my favourites visualisation technique from pandas as it allows you to do a quick analysis of all numerical values in the dataset and their correlations.

By default, it will produce scatterplots for all numeric pairs of variables and histograms for all numeric variables in the data frame:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, figsize=(10, 10))
```

As you can the results is this beautiful set of plots that indeed can tell you a lot about the data set using only one line of code. I can spot some correlations between variables in this data set just by glancing at it.

The only additional parameter I had to set was figure size and this is because the plots were very small with a default size of the chart.

. . .

Summary

I hope you have enjoyed this short tutorial about different pandas visualisations techniques and that you will be able to apply this knowledge to a data set of your choice.

Happy graphing!

. . .

PS: I am writing articles that explain basic Data Science concepts in a simple and comprehensible manner. If you liked this article there are some other ones you may enjoy:

Sorting data frames in pandas

How to sort data frames quickly and efficiently

towardsdatascience.com

7 practical pandas tips when you start working with the library

Explaining some things that are not so obvious at first glance...

towardsdatascience.com

Jupyter notebook autocompletion

The best productivity tool for Data Scientist you should be using if you are not doing it yet...

towardsdatascience.com

Artificial Intelligence | Data Science | Programming | Data Visualization | Machine Learning

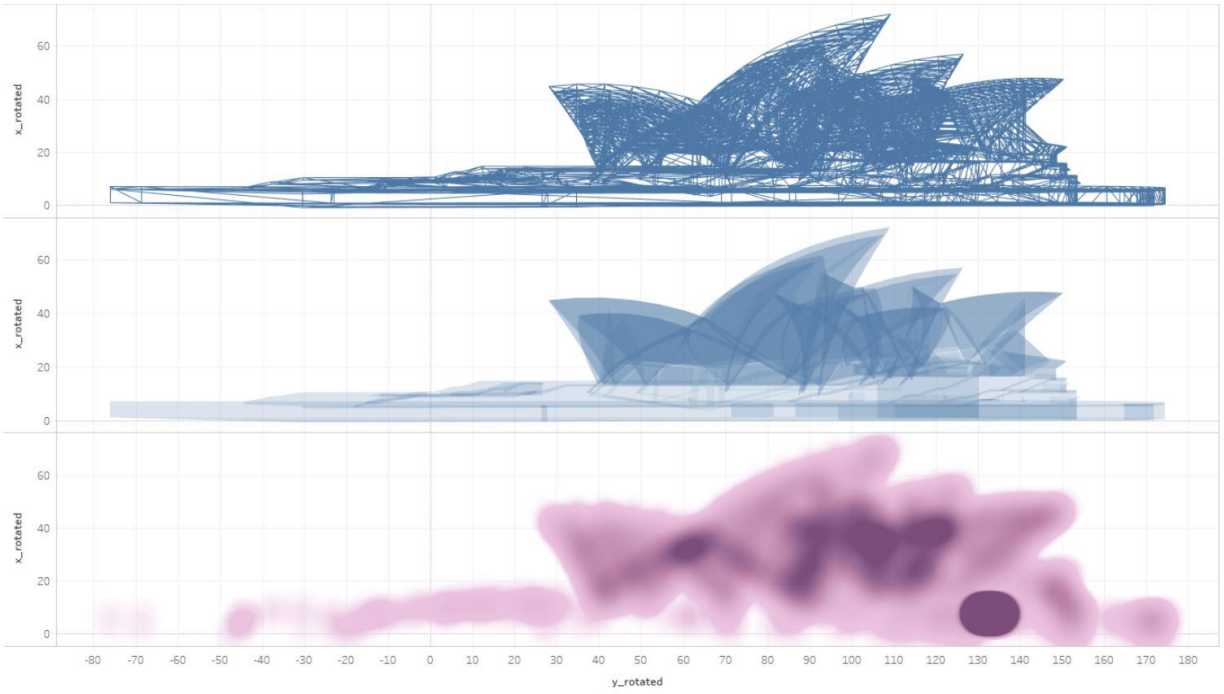
Wrestling with Tableau's Polygon Mapping

Darcy Vance Oct 18, 2019 · 8 min read ★

I could not, for the life of me, find a simple tutorial regarding this incredibly powerful feature of Tableau.

So I thought I'd try my hand at writing so that the next person wanting to learn this (or heaven forbid, the next time-constrained data viz consultant needing this information urgently), doesn't have to trawl through message boards and answer forums, and can get building sooner.

Tableau's Polygon chart type is an incredibly powerful tool for any data viz professional. Most of the insane dashboards for Makeover Monday and crazy, "Tableau-breaking" feature pieces (like this one by Frances Jones), use Polygon Mapping in some way.



3D “Wireframe” Path Model, 3D Polygon Model and Density Map of the Sydney Opera House — Credit: Frances Jones

But beyond fancy shapes and 3D monuments, Polygon mapping can also make the difference in professional visualisation development. Let’s have a look at a demo, and then we can look at some professional use cases.

The Demo

A bit of context:

I’ve always been a fan of professional wrestling. If Game of Thrones was a soap opera, told entirely through interpretive dance, and played out weekly over the span of decades, it would be closing in on WWE’s market share of the sport.

"The Heartbreak Kid" Shawn Michaels performing a high-risk manoeuvre on "Mankind" Mick Foley.

And I'm not using the word "sport" lightly. The performers that involve themselves in this amalgamation of sport and entertainment consistently place themselves in dangerous situations for the benefit of the live, and television, audience. The injuries that can result from this are as real as in any other sport.

I wanted to try and visualise the injuries sustained by some of these performers, as both a training exercise in Tableau, but also due to my own curiosity. This is how I did it:

Step 0: Understanding the data

First, we must understand what is required for polygons to work in Tableau.

Tableau's Polygon mapping requires “an ordered, enclosed sets of X, Y coordinates”. This is a fancy way of saying that, in order to draw a shape on a plane, we require:

- A number of X, Y coordinates
- An order that these X, Y coordinates occur
- And that the set needs to be completed (i.e. the last X, Y value should be the same as the first)

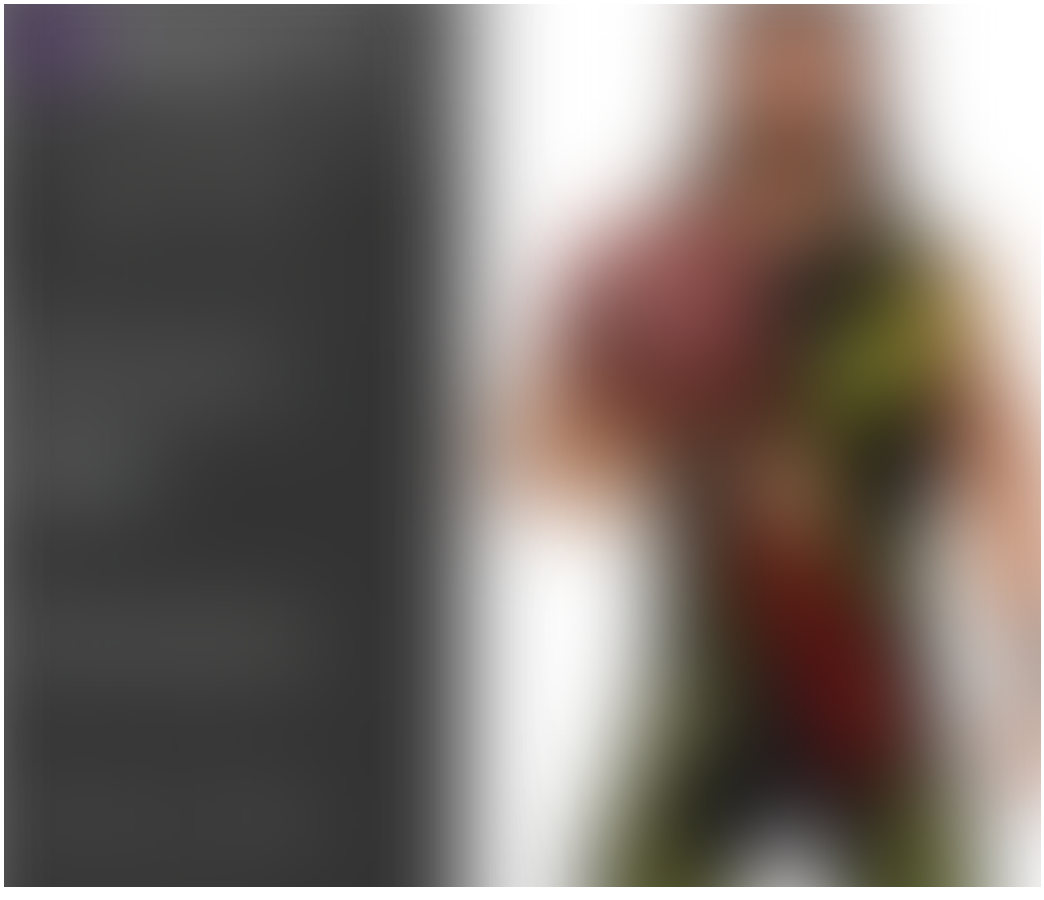
Example: Required data to create the above quadrilateral. Notice there are five values, as we have “completed” the shape by returning to the first X, Y position.

Now, this can be done in any way you like, and it’s relatively easy when creating basic shapes. But as soon as we want to replicate complex shapes, we require some help. This is where the *InterWorks Drawing Utility for Tableau* comes in.

This is a free, browser-based piece of software that allows you to plot points, paths and polygons on top of any image you like. For our example, to better understand the patterns of injuries for professional wrestlers, we are going to represent our data in the form of the human body.

> *Step 1: Creating our polygons*

1. Load your selected image into the InterWorks Drawing Utility for Tableau. (I am using current Universal Champion Seth Rollins as my polygon model, but you can use any image you like.)
2. Under Drawing Options, select Polygons and check “Snap to Existing Points” and “Show Guide Line”. (If mapping mainly straight, rigid objects, “Align with Previous Points” will allow perfect 90° angles.)



3. Start drawing polygons. This is as simple as clicking around the areas of the image that you wish to map. One of the body parts listed in my injury statistics was “Elbow and Forearm”. As you begin your path, you will see the data being generated in the table (or as text, if you’ve selected “CSV”) below Point Data.

Once you finish a polygon, you will notice that the tool fills in the space to make it clear what has been mapped so far. If you have selected “Snap to Existing Points”, you can also use the existing points for new polygons to ensure all edges match correctly.

Once a polygon is completed, you can also click and drag points to change their location. This massively reduces time in fixing up intricate details later on.

After we've mapped all of our polygons, we can "Copy" the table data and paste this into Excel or some other tool. (I used Excel over a text editor to consolidate all of my data — more on this to come).

Step 2: Displaying our polygons

Now that we have our Shape ID, Point ID, X and Y coordinate data. Constructing the polygons in Tableau is easy!

Simply change the Graph Type to "Polygon" and you will see the new Mark Type "Path". The path to which this is referring is looking for the order of the points provided, to be able to plot them accurately on the X, Y plane.

Drag X and Y to Rows and Columns, Point ID to Path and Shape ID to Detail.

And there you have it! Polygon mapping in Tableau!

But if all we wanted to do was draw and image on an X, Y plane, we wouldn't need Tableau. And we aren't exactly generating any insights from this recreation of an image. We need data!

Step 3: Constructing our data

I retrieved some data on injuries in professional wrestling and it looked like this.

We have a two-value hierarchy of Body Group (Upper Extremity, Lower Extremity, Trunk and Head & Neck), and then this group split into Body Part, followed by a raw count of injuries over the study's life cycle.

Seeing as we already have Interworks data with a value representing each polygon (ShapeID), I saw creating an intermediate table to map Body Part to ShapeID as the most simple solution.

So once we jump into Tableau, we map "Interworks.ShapeID" to "PolygonMapping.ShapeID" and "PolygonMapping.BodyPart" to "ResearchData"."BodyPart".

We now have a connection between our polygons and the data we want to display!

> *Step 3: Adding context through background images*

We do still have one problem though. Looking back at what we have in Tableau so far. We can see that Seth's title belt is being treated as a polygon (technically another body part).

For people without context, they may not understand what this shape is or they may mistake it for a part of the body. Additionally, simply removing the polygon through filters doesn't particularly help.

Now we have a big gash through the centre of our figure, and while this would be one hell of an injury, it's not accurate to what we are trying to visualise.

Luckily, we can add a Background Image to the sheet to give context.

Simply go to Maps > Background Images > *Your Data Source*, and select the image you want to underlay. For our example, I have simply painted out most of Seth, from the image, leaving the Universal Championship belt.

Once we upload our image, give the X and Y margins to denote scale, and change the opacity, the output we are left with provides context without sacrificing any of the data points we wish to display.

> *Step 4: Visualising the data*

Now that we have generated and displayed our polygons, constructed and connected our data sources, and added context to the image, the only part left is to visualise our results.

Drag your measure to the Colour mark type and viola!

Not only do we have a cool viz, but we can immediately make connections and gain insight.

While before we were focusing on “Body Group” in our data, by visualising it in a natural way, we can see that injuries are not localised in that way. Rather, they are most serious over the performers’ joints (knee, wrist, shoulder, elbow/forearm and ankles). This is something that we may not have immediately understood by looking at raw data or conventional visualisations, like bar and column graphs.

Professional Use Cases

I’ll admit, I highly doubt I, or anyone else, will ever make a living by visualising professional wrestling (if you do, let me know), but this feature has a wide variety of use cases.

The ability to visualise data in it’s natural, physical form, takes advantage of neural pathways and connections that are already available in humans. Mapping the human body is an example of this. We are able to immediately and accurately identify patterns.

correlations and potential causes to issues involving the human body.

Some other use cases include:

Custom Floorplan Visualisations

By using a customer's own floorplan as a template, future climate control systems could be customised to an individual dwelling (with the above example displaying temperature data captured by IoT-enabled devices inside a residential dwelling).

Additionally, similar implementations could use floorplans of shopping centres or airports to display spending trends and foot-traffic patterns.

Water Quality Visualisation Using Spatial Polygons

Mockup of water quality across the Lane Cove river and inner harbour region of Sydney Harbor.

As Tableau allows for the addition of custom spatial regions, the Interworks “Maps” option is very powerful. This makes the creation of custom, intricate polygons easy.

In the above example, part of the Lane Cove River and inner Sydney Habour have been given custom regions that local councils or organisations may want to keep an eye on.

By visualising the data in a physical representation, rather than by bar charts and dot plots, we would more easily point out sources of poor water quality and identify the flow patterns in these systems.

And there you have it. If you have any cool use cases to share, let me know and I can add them to the post.

If you have any further questions, don't hesitate to reach out.

Darcy Vance - Graduate - Deloitte Australia | LinkedIn

View Darcy Vance's profile on LinkedIn, the world's largest professional community. Darcy has 1 job listed on their...

www.linkedin.com

Polygon Mapping Injury Tableau Data

Twitter Sentiment Analysis using fastText

Sanket Doshi Mar 6, 2019 · 9 min read

In this blog, we'll be analyzing the sentiments of various tweets using a fastText library which is easy to use and fast to train.

Twitter sentiment analysis

What is fastText?

FastText is an NLP library developed by the Facebook AI. It is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices.

Why fastText?

The main disadvantage of deep neural network models is that they took a large amount of time to train and test. Here, fastText have an advantage as it takes very less amount of time to train and can be trained on our home computers at high speed.

As per the Facebook AI blog on fastText, the accuracy of this library is on par of deep neural networks and requires very less amount of time to train.

	Yahoo		Amazon full		Amazon polarity	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
char-CNN	71.2	1 day	59.5	5 days	94.5	5 days
VDCNN	73.4	2h	63	7h	95.7	7h
fastText	72.3	5s	60.2	9s	94.6	10s

comparison between fastText and other deep learning based models

Now, we know about fastText and why we're using it we'll see how to use this library for sentiment analysis.

Get Dataset

We'll be using the dataset available on betsentiment.com. Tweets have four labels with values positive, negative, neutral and mixed. We'll ignore all the tweets with the mixed label.

We'll use teams tweet dataset as a training set while player dataset as a validation set.

Cleaning dataset

As we know, before training any model we need to clean data and it's true here also.

We'll clean tweets based on these rules:

1. Remove all the hashtags as hashtags do not affect sentiments.
2. Remove mentions as they also do not weigh in sentiment analyzing.
3. Replace any emojis with the text they represent as emojis or emoticons plays an important role in representing a sentiment.
4. Replace contractions with their full forms.
5. Remove any URLs present in tweets as they are not significant in sentiment analysis.
6. Remove punctuations.
7. Fix misspelled words (very basic as this is a very time-consuming step).
8. Convert everything to lowercase.
9. Remove HTML tags if present.

Rules to clean tweets:

We'll clean this tweet

```
tweet = '<html> bayer leverkusen goalkeeeeper bernd leno will
not be #going to napoli. his agent uli ferber to bild: "I can
confirm that there were negotiations with napoli, which we
have broken off. napoli is not an option." Atletico madrid
and Arsenal are the other strong rumours. #b04 #afc </html>'
```

Remove HTML tags

Sometimes twitter response contains HTML tags and we need to remove this.

We'll be using `Beautifulsoup` package for this purpose.

If there are not HTML tags present than it will return the same text.

```
tweet = BeautifulSoup(tweet).get_text()

#output
'bayer leverkusen goalkeeeeper bernd leno will not be #going
to napoli. his agent uli ferber to bild: "I can confirm that
there were negotiations with napoli, which we have broken
off. napoli is not an option." Atletico madrid and Arsenal
are the other strong rumours. #b04 #afc'
```

We'll be using regex to match expressions to removed or to be replaced. For this, `re` package will be used.

Remove hashtags

Regex `@[A-Za-z0-9]+` represents mentions and `#[A-Za-z0-9]+` represents hashtags. We'll we replacing every word matching this regex with spaces.

```
tweet = ' '.join(re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)", "
", tweet).split())

#output
'bayer leverkusen goalkeeeeper bernd leno will not be to
napoli. his agent uli ferber to bild: "I can confirm that
there were negotiations with napoli, which we have broken
```

```
off. napoli is not an option." Atletico madrid and Arsenal
are the other strong rumours.'
```

Remove URLs

Regex `\w+:\/\/\S+` matches all the URLs starting with http:// or https:// and replacing it with space.

```
tweet = ' '.join(re.sub("(\w+:\/\/\S+)", " ", tweet).split())

#output
'bayer leverkusen goalkeeeeper bernd leno will not be to
napoli. his agent uli ferber to bild: "I can confirm that
there were negotiations with napoli, which we have broken
off. napoli is not an option." Atletico madrid and Arsenal
are the other strong rumours.'
```

Remove punctuations

Replacing all the punctuations such as `.,!?:;-=` with space.

```
tweet = ' '.join(re.sub("[\.\,\!\?\:\;\-\=]", " ",
tweet).split())

#output
'bayer leverkusen goalkeeeeper bernd leno will not be napoli
his agent uli ferber to bild "I can confirm that there were
negotiations with napoli which we have broken off napoli is
not an option " Atletico madrid and Arsenal are the other
strong rumours'
```

Lower case

To avoid case sensitive issue

```
tweet = tweet.lower()


#output
'bayer leverkusen goalkeeeeper bernd leno will not be napoli
his agent uli ferber to bild "i can confirm that there were
negotiations with napoli which we have broken off napoli is
not an option " atletico madrid and arsenal are the other
strong rumours'
```

Replace contractions

Remove contractions and translate into appropriate slang. There is no universal list to replace contractions so we have made it for our purpose.

```
CONTRACTIONS = {"mayn't":"may not", "may've":"may
have",......}


tweet = tweet.replace("’","'")
words = tweet.split()
reformed = [CONTRACTIONS[word] if word in CONTRACTIONS else
word for word in words]
tweet = " ".join(reformed)


#input
'I mayn’t like you.'


#output
'I may not like you.'
```

Fix misspelled words

Here we are not actually building any complex function to correct the misspelled words but just checking that each character should occur not more than 2 times in every word. It’s a very basic misspelling check.

```
tweet = ''.join(''.join(s)[:2] for _, s in
itertools.groupby(tweet))


#output
'bayer leverkusen goalkeeper bernd leno will not be napoli
his agent uli ferber to bild "i can confirm that there were
negotiations with napoli which we have broken off napoli is
not an option " atletico madrid and arsenal are the other
strong rumours'
```

Replace emojis or emoticons

As emojis and emoticons play a significant role in expressing the sentiments we need to replace them with the expression they represent in plain English.

For emojis, we'll be using `emoji` package and for emoticons, we'll be building our own dictionary.

```
SMILEYS = {":-(":"sad", ":-)":"smiley", ....}

words = tweet.split()
reformed = [SMILEY[word] if word in SMILEY else word for word
in words]
tweet = " ".join(reformed)


#input
'I am :-('


#output
'I am sad'
```

For emojis

Emoji package return values for given emoji as `:flushed_face:` so we need to remove `:` from a given output.

```
tweet = emoji.demojize(tweet)
tweet = tweet.replace(":"," ")
tweet = ' '.join(tweet.split())

#input
'He is 😳'

#output
'He is flushed_face'
```

So, we've cleaned our data.

Why not use NLTK stop words?

Removing stop words is an efficient way while cleaning data. It removes all the insignificant words and usually is the most common words used in each sentence. To get all the stop words present in the NLTK library

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "i
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havi
ng', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'o
f', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'onc
e', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'so
me', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'could
n', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

NLTK stop words

We can see that if NLTK stopwords are used than all the negative contractions will be removed which plays a significant role in sentiment analysis.

Formatting the Dataset

Need to format the data in which fastText requires for supervised learning.

FastText assumes the labels are words that are prefixed by the string `__label__`.

The input to the fastText model should look like

```
__label__NEUTRAL _d i 'm just fine i have your fanbase angry
over
__label__POSITIVE what a weekend of football results & hearts
```

We can format our data using

```
def transform_instance(row):
    cur_row = []
    #Prefix the index-ed label with __label__
    label = "__label__" + row[4]
    cur_row.append(label)

cur_row.extend(nltk.word_tokenize(tweet_cleaning_for_sentimen
t_analysis(row[2].lower())))
    return cur_row


def preprocess(input_file, output_file):
    i=0
    with open(output_file, 'w') as csvoutfile:
        csv_writer = csv.writer(csvoutfile, delimiter=' ',
lineterminator='\n')
```

```
        with open(input_file, 'r', newline='',
encoding='latin1') as csvinfile: # encoding='latin1'
            csv_reader = csv.reader(csvinfile, delimiter=',',
quotechar='"')
            for row in csv_reader:
                if row[4]!="MIXED" and row[4].upper() in
['POSITIVE','NEGATIVE','NEUTRAL'] and row[2]!='':
                    row_output = transform_instance(row)
                    csv_writer.writerow(row_output )
                    # print(row_output)
                i=i+1
                if i%10000 ==0:
                    print(i)
```

Here, we are ignoring tweets with labels other than `Positive, Negative and neutral .`

`nltk.word_tokenize()` converts string into independent words.

```
nltk.word_tokenize('hello world!')

#output
['hello', 'world', '!']
```

Upsampling the dataset

In our dataset data is not equally divided into different labels. It contains around 72% of data in the neutral label. So, we can see that our model will tend to be overwhelmed by the large class and ignore the small ones.

```
import pandas as pd
import seaborn as sns
```

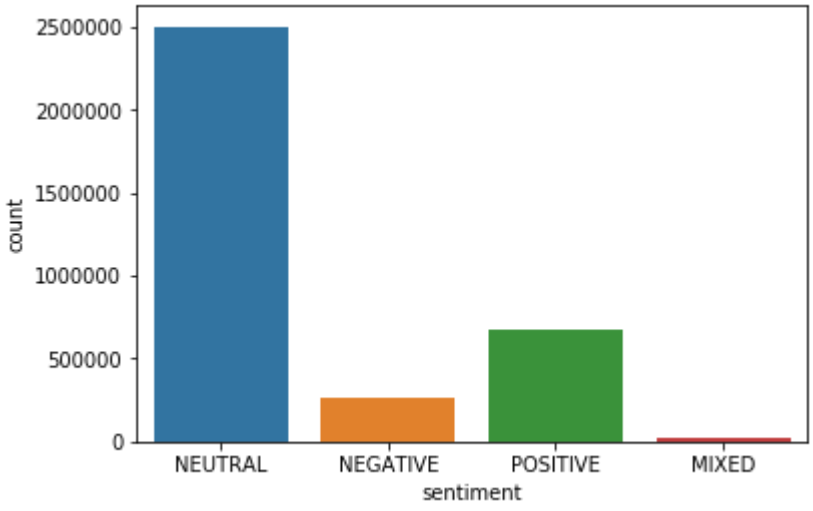
```
df = pd.read_csv('betsentiment-EN-tweets-sentiment-
teams.csv',encoding='latin1')
```

```
df['sentiment'].value_counts(normalize=True)*100
```

```
NEUTRAL     72.284338
POSITIVE    19.306613
NEGATIVE     7.624154
MIXED        0.784895
Name: sentiment, dtype: float64
```

percentage of tweets for each labels

```
sns.countplot(x="sentiment", data=df)
```



countplot for sentiment labels

As the NEUTRAL class consists of a large portion of the dataset, the model will always try to predict NEUTRAL label as it'll guarantee 72% of accuracy. To prevent this we need to have an equal number of tweets for each label. We can achieve this by adding new tweets to the minor class. This process of adding new tweets to the minority labels is known as upsampling.

We'll achieve upsampling by repeating the tweets present in the given label again and again until the number of tweets is equal in each label.

```
def upsampling(input_file, output_file, ratio_upsampling=1):
    # Create a file with equal number of tweets for each
label
    #    input_file: path to file
    #    output_file: path to the output file
    #    ratio_upsampling: ratio of each minority classes vs
majority one. 1 mean there will be as much of each class than
there is for the majority class

    i=0
    counts = {}
    dict_data_by_label = {}


# GET LABEL LIST AND GET DATA PER LABEL
    with open(input_file, 'r', newline='') as csvinfile:
        csv_reader = csv.reader(csvinfile, delimiter=',',
quotechar='"')
        for row in csv_reader:
            counts[row[0].split()[0]] =
counts.get(row[0].split()[0], 0) + 1
            if not row[0].split()[0] in dict_data_by_label:
                dict_data_by_label[row[0].split()[0]]=
[row[0]]
            else:
                dict_data_by_label[row[0].split()
[0]].append(row[0])
            i=i+1
            if i%10000 ==0:
                print("read" + str(i))


# FIND MAJORITY CLASS
    majority_class=""
    count_majority_class=0
    for item in dict_data_by_label:
        if
len(dict_data_by_label[item])>count_majority_class:
            majority_class= item

count_majority_class=len(dict_data_by_label[item])

    # UPSAMPLE MINORITY CLASS
    data_upsampled=[]
    for item in dict_data_by_label:
        data_upsampled.extend(dict_data_by_label[item])
```

```
        if item != majority_class:
            items_added=0
            items_to_add = count_majority_class -
len(dict_data_by_label[item])
            while items_added<items_to_add:

data_upsampled.extend(dict_data_by_label[item]
[:max(0,min(items_to_add-
items_added,len(dict_data_by_label[item])))])
                items_added = items_added +
max(0,min(items_to_add-
items_added,len(dict_data_by_label[item])))


# WRITE ALL
    i=0


with open(output_file, 'w') as txtoutfile:
        for row in data_upsampled:
            txtoutfile.write(row+ '\n' )
            i=i+1
            if i%10000 ==0:
                print("writer" + str(i))
```

As of repeating tweets, again and again, may cause our model to overfit our dataset but due to the large size of our dataset, this is not a problem.

Training

Try to install fastText with git clone rather than using pip.

We'll be using supervised training method.

```
hyper_params = {"lr": 0.01,
                "epoch": 20,
                "wordNgrams": 2,
                "dim": 20}

        print(str(datetime.datetime.now()) + ' START=>' +
str(hyper_params) )
```

```
# Train the model.
        model =
fastText.train_supervised(input=training_data_path,
**hyper_params)
        print("Model trained with the hyperparameter \n
{}".format(hyper_params))
```

`lr` represents `learning rate` , `epoch` represents `number of epoch` , `wordNgrams` represents `max length of word Ngram` , `dim` represents `size of word vectors` .

`train_supervised` is a function used to train the model using supervised learning.

Evaluate

We need to evaluate the model to find it's accuracy.

```
model_acc_training_set = model.test(training_data_path)
model_acc_validation_set = model.test(validation_data_path)

# DISPLAY ACCURACY OF TRAINED MODEL
text_line = str(hyper_params) + ",accuracy:" +
str(model_acc_training_set[1])  + ",validation:" +
str(model_acc_validation_set[1]) + '\n'


print(text_line)
```

We'll evaluate our model on both training as well as validation dataset.

`test` returns precision and recall of model rather than accuracy. But in our case both the values are almost the same so, we'll be using

precision only.

Overall the model gives an accuracy of 97.5% on the training data, and 79.7% on the validation data.

Predict

We'll predict the sentiment of text passed to our trained model.

```
model.predict(['why not'],k=3)
model.predict(['this player is so bad'],k=1)
```

`predict` lets us predict the sentiment of the passed string and `k` represents the number of labels to return with a confidence score.

Quantize the model

Quantizing helps us to reduce the size of the model.

```
model.quantize(input=training_data_path, qnorm=True,
retrain=True, cutoff=100000)
```

Save model

We can save our trained model and then can use anytime on the go rather than training it every time.

```
model.save_model(os.path.join(model_path,model_name +
".ftz"))
```

Conclusion

We learn how to clean data, and pass it to train model to predict the sentiment of tweets. We also learn to implement sentiment analysis model using fastText.

```
import fastText
import sys
import os
import nltk
nltk.download('punkt')
import csv
import datetime
from bs4 import BeautifulSoup
import re
import itertools
import emoji


##############################################################################
#
# DATA CLEANING
#
##############################################################################

# emoticons
def load_dict_smileys():

    return {
        ":-)":"smiley",
        ":-]":"smiley",
        ":-3":"smiley",
        ":->":"smiley",
        "8-)":"smiley",
        ":-}":"smiley",
        ":)":"smiley",
```

```
        ":]":"smiley",
        ":3":"smiley",
        ":>":"smiley",
        "8)":"smiley",
        ":}":"smiley",
        ":o)":"smiley",
        ":c)":"smiley",
        ":^)":"smiley",
        "=]":"smiley",
        "=)":"smiley",
        ":-))":"smiley",
        ":‿D":"smiley",
        "8‿D":"smiley",
        "x‿D":"smiley",
        "X‿D":"smiley",
        ":D":"smiley",
        "8D":"smiley",
        "xD":"smiley",
        "XD":"smiley",
        ":‿(":"sad",
        ":‿c":"sad",
        ":‿<":"sad",
        ":‿[":"sad",
        ":(":"sad",
        ":c":"sad",
        ":<":"sad",
        ":[":"sad",
        ":-||":"sad",
        ">:[":"sad",
        ":{":"sad",
        ":@":"sad",
        ">:(":"sad",
        ":'‿(":"sad",
        ":'(":"sad",
        ":‿P":"playful",
        "X‿P":"playful",
        "x‿p":"playful",
        ":‿p":"playful",
        ":‿Þ":"playful",
        ":‿þ":"playful",
        ":‿b":"playful",
        ":P":"playful",
        "XP":"playful",
        "xp":"playful",
        ":p":"playful",
```

```
            :p":"playful",
            ":Þ":"playful",
            ":þ":"playful",
            ":b":"playful",
            "<3":"love"
            }

# self defined contractions
def load_dict_contractions():

    return {
        "ain't":"is not",
        "amn't":"am not",
        "aren't":"are not",
        "can't":"cannot",
        "'cause":"because",
        "couldn't":"could not",
        "couldn't've":"could not have",
        "could've":"could have",
        "daren't":"dare not",
        "daresn't":"dare not",
        "dasn't":"dare not",
        "didn't":"did not",
        "doesn't":"does not",
        "don't":"do not",
        "e'er":"ever",
        "em":"them",
        "everyone's":"everyone is",
        "finna":"fixing to",
        "gimme":"give me",
        "gonna":"going to",
        "gon't":"go not",
        "gotta":"got to",
        "hadn't":"had not",
        "hasn't":"has not",
        "haven't":"have not",
        "he'd":"he would",
        "he'll":"he will",
        "he's":"he is",
        "he've":"he have",
        "how'd":"how would",
        "how'll":"how will",
        "how're":"how are",
        "how's":"how is",
        "I'd":"I would",
```

```
        "I'll":"I will",
        "I'm":"I am",
        "I'm'a":"I am about to",
        "I'm'o":"I am going to",
        "isn't":"is not",
        "it'd":"it would",
        "it'll":"it will",
        "it's":"it is",
        "I've":"I have",
        "kinda":"kind of",
        "let's":"let us",
        "mayn't":"may not",
        "may've":"may have",
        "mightn't":"might not",
        "might've":"might have",
        "mustn't":"must not",
        "mustn't've":"must not have",
        "must've":"must have",
        "needn't":"need not",
        "ne'er":"never",
        "o'":"of",
        "o'er":"over",
        "ol'":"old",
        "oughtn't":"ought not",
        "shalln't":"shall not",
        "shan't":"shall not",
        "she'd":"she would",
        "she'll":"she will",
        "she's":"she is",
        "shouldn't":"should not",
        "shouldn't've":"should not have",
        "should've":"should have",
        "somebody's":"somebody is",
        "someone's":"someone is",
        "something's":"something is",
        "that'd":"that would",
        "that'll":"that will",
        "that're":"that are",
        "that's":"that is",
        "there'd":"there would",
        "there'll":"there will",
        "there're":"there are",
        "there's":"there is",
        "these're":"these are",
        "they'd":"they would"
```

```
            "they'd":"they would",
            "they'll":"they will",
            "they're":"they are",
            "they've":"they have",
            "this's":"this is",
            "those're":"those are",
            "'tis":"it is",
            "'twas":"it was",
            "wanna":"want to",
            "wasn't":"was not",
            "we'd":"we would",
            "we'd've":"we would have",
            "we'll":"we will",
            "we're":"we are",
            "weren't":"were not",
            "we've":"we have",
            "what'd":"what did",
            "what'll":"what will",
            "what're":"what are",
            "what's":"what is",
            "what've":"what have",
            "when's":"when is",
            "where'd":"where did",
            "where're":"where are",
            "where's":"where is",
            "where've":"where have",
            "which's":"which is",
            "who'd":"who would",
            "who'd've":"who would have",
            "who'll":"who will",
            "who're":"who are",
            "who's":"who is",
            "who've":"who have",
            "why'd":"why did",
            "why're":"why are",
            "why's":"why is",
            "won't":"will not",
            "wouldn't":"would not",
            "would've":"would have",
            "y'all":"you all",
            "you'd":"you would",
            "you'll":"you will",
            "you're":"you are",
            "you've":"you have",
            "Whatcha":"What are you",
```

```
209         "luv":"love",
210         "sux":"sucks"
211         }
212 
213 
214 def tweet_cleaning_for_sentiment_analysis(tweet):
215 
216     #Escaping HTML characters
217     tweet = BeautifulSoup(tweet).get_text()
218 
219     #Special case not handled previously.
220     tweet = tweet.replace('\x92',"'")
221 
222     #Removal of hastags/account
223     tweet = ' '.join(re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)", " ", tweet).spl
224 
225     #Removal of address
226     tweet = ' '.join(re.sub("(\w+:\/\/\S+)", " ", tweet).split())
227 
228     #Removal of Punctuation
229     tweet = ' '.join(re.sub("[\.\,\!\?\:\;\-\=]", " ", tweet).split())
230 
231     #Lower case
232     tweet = tweet.lower()
233 
234     #CONTRACTIONS source: https://en.wikipedia.org/wiki/Contraction_%28grammar
235     CONTRACTIONS = load_dict_contractions()
236     tweet = tweet.replace("’","'")
237     words = tweet.split()
238     reformed = [CONTRACTIONS[word] if word in CONTRACTIONS else word for word
239     tweet = " ".join(reformed)
240 
241     # Standardizing words
242     tweet = ''.join(''.join(s)[:2] for _, s in itertools.groupby(tweet))
243 
244     #Deal with emoticons source: https://en.wikipedia.org/wiki/List_of_emotico
245     SMILEY = load_dict_smileys()
246     words = tweet.split()
247     reformed = [SMILEY[word] if word in SMILEY else word for word in words]
248     tweet = " ".join(reformed)
249 
250     #Deal with emojis
251     tweet = emoji.demojize(tweet)
252 
253     tweet = tweet.replace(":"," ")
```

```
    tweet = ' '.join(tweet.split())

    return tweet



################################################################################
#
# DATA PROCESSING
#
################################################################################

def transform_instance(row):
    cur_row = []
    #Prefix the index-ed label with __label__
    label = "__label__" + row[4]
    cur_row.append(label)
    cur_row.extend(nltk.word_tokenize(tweet_cleaning_for_sentiment_analysis(r
    return cur_row


def preprocess(input_file, output_file, keep=1):
    i=0
    with open(output_file, 'w') as csvoutfile:
        csv_writer = csv.writer(csvoutfile, delimiter=' ', lineterminator='\n'
        with open(input_file, 'r', newline='', encoding='latin1') as csvinfile
            csv_reader = csv.reader(csvinfile, delimiter=',', quotechar='"')
            for row in csv_reader:
                if row[4]!="MIXED" and row[4].upper() in ['POSITIVE','NEGATIVE
                    row_output = transform_instance(row)
                    csv_writer.writerow(row_output )
                    # print(row_output)
                i=i+1
                if i%10000 ==0:
                    print(i)

# Preparing the training dataset
preprocess('betsentiment-EN-tweets-sentiment-teams.csv', 'tweets.train')

# Preparing the validation dataset
preprocess('betsentiment-EN-tweets-sentiment-players.csv', 'tweets.validation'


################################################################################
```

```
298 #
299 # UPSAMPLING
300 #
301 ################################################################################
302
303 def upsampling(input_file, output_file, ratio_upsampling=1):
304     # Create a file with equal number of tweets for each label
305     #    input_file: path to file
306     #    output_file: path to the output file
307     #    ratio_upsampling: ratio of each minority classes vs majority one. 1 m
308
309     i=0
310     counts = {}
311     dict_data_by_label = {}
312
313     # GET LABEL LIST AND GET DATA PER LABEL
314     with open(input_file, 'r', newline='') as csvinfile:
315         csv_reader = csv.reader(csvinfile, delimiter=',', quotechar='"')
316         for row in csv_reader:
317             counts[row[0].split()[0]] = counts.get(row[0].split()[0], 0) + 1
318             if not row[0].split()[0] in dict_data_by_label:
319                 dict_data_by_label[row[0].split()[0]]=[row[0]]
320             else:
321                 dict_data_by_label[row[0].split()[0]].append(row[0])
322             i=i+1
323             if i%10000 ==0:
324                 print("read" + str(i))
325
326     # FIND MAJORITY CLASS
327     majority_class=""
328     count_majority_class=0
329     for item in dict_data_by_label:
330         if len(dict_data_by_label[item])>count_majority_class:
331             majority_class= item
332             count_majority_class=len(dict_data_by_label[item])
333
334     # UPSAMPLE MINORITY CLASS
335     data_upsampled=[]
336     for item in dict_data_by_label:
337         data_upsampled.extend(dict_data_by_label[item])
338         if item != majority_class:
339             items_added=0
340             items_to_add = count_majority_class - len(dict_data_by_label[item]
341             while items_added<items_to_add:
342                 data_upsampled.extend(dict_data_by_label[item][:max(0,min(iten
```

```
343                 items_added = items_added + max(0,min(items_to_add-items_added
344
345     # WRITE ALL
346     i=0
347
348     with open(output_file, 'w') as txtoutfile:
349         for row in data_upsampled:
350             txtoutfile.write(row+ '\n' )
351             i=i+1
352             if i%10000 ==0:
353                 print("writer" + str(i))
354
355
356 upsampling( 'tweets.train','uptweets.train')
357 # No need to upsample for the validation set. As it does not matter what valid
358
359
360 ################################################################################
361 #
362 # TRAINING
363 #
364 ################################################################################
365
366 # Full path to training data.
367 training_data_path ='uptweets.train'
368 validation_data_path ='tweets.validation'
369 model_path =''
370 model_name="model-en"
371
372 def train():
373     print('Training start')
374     try:
375         hyper_params = {"lr": 0.01,
376                         "epoch": 20,
377                         "wordNgrams": 2,
378                         "dim": 20}
379
380         print(str(datetime.datetime.now()) + ' START=>' + str(hyper_params) )
381
382         # Train the model.
383         model = fastText.train_supervised(input=training_data_path, **hyper_pa
384         print("Model trained with the hyperparameter \n {}".format(hyper_param
385
386         # CHECK PERFORMANCE
```

```
387           print(str(datetime.datetime.now()) + 'Training complete.' + str(hyper_
388
389           model_acc_training_set = model.test(training_data_path)
390           model_acc_validation_set = model.test(validation_data_path)
391
392           # DISPLAY ACCURACY OF TRAINED MODEL
393           text_line = str(hyper_params) + ",accuracy:" + str(model_acc_training_
394           print(text_line)
395
396           #quantize a model to reduce the memory usage
397           model.quantize(input=training_data_path, qnorm=True, retrain=True, cut
398
399           print("Model is quantized!!")
400           model.save_model(os.path.join(model_path,model_name + ".ftz"))
401
402           ################################################################################
403           #
404           #  TESTING PART
405           #
406           ################################################################################
407           model.predict(['why not'],k=3)
408           model.predict(['this player is so bad'],k=1)
409
410       except Exception as e:
411           print('Exception during training: ' + str(e) )
412
413
414   # Train your model.
415   train()
```

Machine Learning Sentiment Analysis Twitter Fasttext Supervised Learning

Appendix N. Converting DOS Batch Files to Shell Scripts

Quite a number of programmers learned scripting on a PC running DOS. Even the crippled DOS batch file language allowed writing some fairly powerful scripts and applications, though they often required extensive kludges and workarounds. Occasionally, the need still arises to convert an old DOS batch file to a UNIX shell script. This is generally not difficult, as DOS batch file operators are only a limited subset of the equivalent shell scripting ones.

Table N-1. Batch file keywords / variables / operators, and their shell equivalents

Batch File Operator	Shell Script Equivalent	Meaning
%	$	command-line parameter prefix
/	-	command option flag
\	/	directory path separator
==	=	(equal-to) string comparison test
!==!	!=	(not equal-to) string comparison test
|	|	pipe
@	set +v	do not echo current command
*	*	filename "wild card"
>	>	file redirection (overwrite)
>>	>>	file redirection (append)
<	<	redirect stdin
%VAR%	$VAR	environmental variable
REM	#	comment
NOT	!	negate following test
NUL	/dev/null	"black hole" for burying command output
ECHO	echo	echo (many more option in Bash)
ECHO.	echo	echo blank line
ECHO OFF	set +v	do not echo command(s) following
FOR %%VAR IN (LIST) DO	for var in [list]; do	"for" loop
:LABEL	none (unnecessary)	label
GOTO	none (use a function)	jump to another location in the script
PAUSE	sleep	pause or wait an interval
CHOICE	case or select	menu choice
IF	if	if-test
IF EXIST *FILENAME*	if [-e filename]	test if file exists
IF !%N==!	if [-z "$N"]	if replaceable parameter "N" not present
CALL	source or . (dot operator)	"include" another script
COMMAND /C	source or . (dot operator)	"include" another script (same as CALL)
SET	export	set an environmental variable
SHIFT	shift	left shift command-line argument list
SGN	-lt or -gt	sign (of integer)
ERRORLEVEL	$?	exit status
CON	stdin	"console" (stdin)
PRN	/dev/lp0	(generic) printer device
LPT1	/dev/lp0	first printer device
COM1	/dev/ttyS0	first serial port

Batch files usually contain DOS commands. These must be translated into their UNIX equivalents in order to convert a batch file into a shell script.

Table N-2. DOS commands and their UNIX equivalents

DOS Command	UNIX Equivalent	Effect
ASSIGN	ln	link file or directory
ATTRIB	chmod	change file permissions
CD	cd	change directory

DOS Command	UNIX Equivalent	Effect
CHDIR	cd	change directory
CLS	clear	clear screen
COMP	diff, comm, cmp	file compare
COPY	cp	file copy
Ctl-C	Ctl-C	break (signal)
Ctl-Z	Ctl-D	EOF (end-of-file)
DEL	rm	delete file(s)
DELTREE	rm -rf	delete directory recursively
DIR	ls -l	directory listing
ERASE	rm	delete file(s)
EXIT	exit	exit current process
FC	comm, cmp	file compare
FIND	grep	find strings in files
MD	mkdir	make directory
MKDIR	mkdir	make directory
MORE	more	text file paging filter
MOVE	mv	move
PATH	$PATH	path to executables
REN	mv	rename (move)
RENAME	mv	rename (move)
RD	rmdir	remove directory
RMDIR	rmdir	remove directory
SORT	sort	sort file
TIME	date	display system time
TYPE	cat	output file to stdout
XCOPY	cp	(extended) file copy

Virtually all UNIX and shell operators and commands have many more options and enhancements than their DOS and batch file counterparts. Many DOS batch files rely on auxiliary utilities, such as **ask.com**, a crippled counterpart to read.

DOS supports only a very limited and incompatible subset of filename wild-card expansion, recognizing just the * and ? characters.

Converting a DOS batch file into a shell script is generally straightforward, and the result ofttimes reads better than the original.

Example N-1. VIEWDATA.BAT: DOS Batch File

```
REM VIEWDATA

REM INSPIRED BY AN EXAMPLE IN "DOS POWERTOOLS"
REM                           BY PAUL SOMERSON


@ECHO OFF

IF !%1==! GOTO VIEWDATA
REM  IF NO COMMAND-LINE ARG...
FIND "%1" C:\BOZO\BOOKLIST.TXT
GOTO EXIT0
REM  PRINT LINE WITH STRING MATCH, THEN EXIT.

:VIEWDATA
TYPE C:\BOZO\BOOKLIST.TXT | MORE
REM  SHOW ENTIRE FILE, 1 PAGE AT A TIME.

:EXIT0
```

The script conversion is somewhat of an improvement. [1]

Example N-2. *viewdata.sh*: Shell Script Conversion of VIEWDATA.BAT

```
#!/bin/bash
# viewdata.sh
# Conversion of VIEWDATA.BAT to shell script.

DATAFILE=/home/bozo/datafiles/book-collection.data
ARGNO=1

# @ECHO OFF                 Command unnecessary here.

if [ $# -lt "$ARGNO" ]    # IF !%1==! GOTO VIEWDATA
then
  less $DATAFILE          # TYPE C:\MYDIR\BOOKLIST.TXT | MORE
else
  grep "$1" $DATAFILE     # FIND "%1" C:\MYDIR\BOOKLIST.TXT
fi

exit 0                    # :EXIT0

#  GOTOs, labels, smoke-and-mirrors, and flimflam unnecessary.
#  The converted script is short, sweet, and clean,
#+ which is more than can be said for the original.
```

Ted Davis' Shell Scripts on the PC site had a set of comprehensive tutorials on the old-fashioned art of batch file programming. Unfortunately the page has vanished without a trace.

Notes

[1] Various readers have suggested modifications of the above batch file to prettify it and make it more compact and efficient. In the opinion of the *ABS Guide* author, this is wasted effort. A Bash script can access a DOS filesystem, or even an NTFS partition (with the help of ntfs-3g) to do batch or scripted operations.

Are you writing print() statements to debug your Python code?

Pradeepa Gollapalli
Jul 21 · 7 min read ★

Then, you should read this...

Photo by Geoff Park on Unsplash

I was one such person who used to debug code using print() statements. Some times, if the code is lengthy, then there are more prints with multiple symbols to differentiate from one another.

Have a look at the code snippet below.
(Code snippets in this blog, follow the syntax of Python 3.7)

```
print(style_dict,"
{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{}}}}}}}}}}}}}}}}}}}}}}}}}}}}}}]]]]]]]]
]]]]]]]]]]]")
# Adding into a dictionary
res_dct = {style_dict[i]: style_dict[i + 1] for i in range(0,
len(style_dict), 2)}
res_dist={res_dct['Email Address']:{style_dict[i]: style_dict[i + 1]
for i in range(0, len(style_dict), 2)}}
print(res_dist,"++++++++++++++++++++++++++++++++++++++++++++++++++++
+")
recon_dict = res_dct
print(recon_dict,"--------------------------------------------------
-")
# Removing space so that data can be transferred to HTML fields
recon_dict = {x.translate({32: None}): y
              for x, y in list(recon_dict.items())}
print("##################################################")
print(recon_dict)
print("################################################")

# Converting to JSON
r = json.dumps(recon_dict)
print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$")
print(r)
loaded_json = json.loads(r)
print("WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWwwwwww")
print("******************************************************")
print(loaded_json)
```

Here, I'm trying to add a dictionary to a JSON file. Due to some error, I had to use that many print statements with different symbols to debug.

But as the code gets bigger with different modules and different classes calling different definitions in other modules or classes, this is not the right choice.

Let's see some drawbacks for this approach:

- With more code being added, it's difficult to use print statements in every module, class, or definition the code is traversing through.
- Even before we notice the bug, the code gets executed and goes to the next steps.
- Wait till a lengthy execution is complete to find and make a fix.

- Going back to a whole lot of logs to search the right symbols we gave in the print statements and matching them is tedious.

We felt giving prints is a simple debugging solution but, doesn't it feel tedious now.

A Simple turnaround,

We don't have to do anything except using a powerful weapon that Python has provides us, the "pdb module". This module helps us to debug effectively.

What is pdb(python debugger)?
pdb is an interactive shell, that helps to debug python code. It helps us to step into our code, one line at a time, pause, examine the state, and continue to next line of code or continue execution.

Some ways to invoke pdb:

Here, we are looking at 3 ways to invoke pdb.
Postmortem: Use this, if you want to debug at the program level.
Inline pdb: Use this, if you are working on versions earlier to 3.7
breakpoint(): Use this, for version 3.7 and higher

a) Postmortem

Let's understand with a simple program.

debug_add.py

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Here the def add_num is supposed to add the value of num variable to each element in a list named listA, store new values in list sum, and return list sum.

By executing the python file as below will invoke pdb,

```
python -m pdb debug_add.py
```

This will enter the pdb mode and halt at the first line of code.

```
(venv) C:\Users\PycharmProjects\>python -m pdb debug_add.py
> c:\users\pycharmprojects\debug_add.py(2)<module>()
-> def add_num(listA,num):
(Pdb)
```

Anytime if you need help with debugger use ‘h’(help), which lists all the options.

```
(Pdb) h

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv
undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source
until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u
whatis
bt     continue   exit     l         pp        run      unalias
where

Miscellaneous help topics:
==========================
exec  pdb
```

Help on a specific option,

```
(Pdb) h debug
debug code
        Enter a recursive debugger that steps through the code
        argument (which is an arbitrary expression or statement to
        be executed in the current environment).
```

Back to the program, to go to the next step of execution use option ‘n’(next).

```
> c:\users\pycharmprojects\debug_add.py(2)<module>()
-> def add_num(listA,num):
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(8)<module>()
-> listA = [2, 4, 6, 8]
```

Here we can examine the values of a variable by giving the name of the variable as below,

```
(Pdb) listA
*** NameError: name 'listA' is not defined
(Pdb)
*** NameError: name 'listA' is not defined
```

We reached the line listA = [2, 4, 6, 8] but we still haven't executed, so it says listA not defined. If you observed if we hit enter at any time the previous option gets executed as above.

Now hit 'n' to move forward and check the listA variable.

```
(Pdb) n
> c:\users\pycharmprojects\debug_add.py(9)<module>()
-> num=10
(Pdb) listA
[2, 4, 6, 8]
(Pdb)
```

To check which line of code we are in, use the option 'l'(line). The arrow mark points to the line we are in, EOF represents end of file.

```
(Pdb) l
  4         for i in listA:
  5             sum.append(i*num)
  6         return sum
  7
  8     listA = [2, 4, 6, 8]
  9  -> num=10
 10     result=add_num(listA,num)
 11     print(result)
[EOF]
(Pdb)
```

To quit debugger, we use option ‘q’(quit).

```
(Pdb) q

(venv) C:\Users\PycharmProjects\>
```

Another way to go with postmortem method is to halt the execution, only when an exception is encountered, For this use -c continue along with -m pdb.

```
python -m pdb -c continue debug_add.py
```

b) Inline pdb

In earlier versions of Python 3.7, we have to explicitly import the module pdb and call pdb.set_trace() to halt the program and perform debugging.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
import pdb; pdb.set_trace()
result=add_num(listA,num)
print(result)
```

Let’s examine the console when we run this.

```
> c:\users\pycharmprojects\debug_add.py(11)<module>()
-> result=add_num(listA,num)
(Pdb)
```

The execution halted, as pdb.set_trace() is encountered and entered debug mode. We can now perform all the debugging actions here as shown in the postmortem method above.

c) breakpoint()

From Python 3.7 onwards, breakpoint() definition is introduced which helps to debug pythonic code without having to explicitly import the module pdb and call pdb.set_trace(). breakpoint() does all that for us and opens PDB debugger in the console.

Now, let's execute the above code without any breakpoints and debug if any errors are encountered.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Output:

```
C:\Users\PycharmProjects\venv\Scripts\python.exe
C:/Users/PycharmProjects/debug_add.py
[20, 40, 60, 80]

Process finished with exit code 0
```

The mission of the code block is, to add num which is 10 to each of the elements in the list and return the new list.
Expected result is [12, 14, 16, 18]
Actual result is [20, 40, 60, 80]

Now let's use the breakpoint() weapon to debug and fix the code.

Where you place breakpoint() depends on where you suspect the bug is. In this case, we place it before it enters the add_num() definition.

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i*num)
    return sum

listA = [2, 4, 6, 8]
num=10
breakpoint()
result=add_num(listA,num)
print(result)
```

Output:

```
> c:\users\pycharmprojects\debug_add.py(11)<module>()
-> result=add_num(listA,num)
(Pdb) n
> c:\users\pycharmprojects\debug_add.py(12)<module>()
-> print(result)
(Pdb) n
[20, 40, 60, 80]
 — Return —
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(12)<module>
()->None
-> print(result)
(Pdb)
```

Option ’n’(next) is used to move to the next line or step over any definitions. But in this case, we need to step into the definition, for that we will use option ‘s’(step).

Below the text in bold is used to highlight the option used and its explanation.

```
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(11)<module>
()
-> result=add_num(listA,num)
(Pdb) s  <----- Step into def add_num
--Call--
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(2)add_num()
-> def add_num(listA,num):
(Pdb) s  <---- stepped inside def add_num
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(3)add_num()
-> sum=[]
(Pdb) n  <--- inside a def feel free to use 'n'
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(4)add_num()
-> for i in listA:
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(5)add_num()
```

```
-> sum.append(i*num)
(Pdb) n
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(4)add_num()
-> for i in listA:
(Pdb) sum  <-- examine sum value
[20] <--- 2+10 =12 not 20,oops we used '*'instead of '+' in
                          appending to list sum,CAUGHT IT!
(Pdb) i  <-- so, examine i
2
(Pdb) sum.append(i+num) <-- try adding + in the expression
(Pdb) sum
[20, 12] <-- PERFECT, FIXED IT!
(Pdb) u <-- used to skip other iterations of for loop.
> c:\users\prade\pycharmprojects\jobportal\debug_add.py(11)<module>
()
-> result=add_num(listA,num)
(Pdb) c <-- used to continue with execution
[20, 12, 40, 60, 80] <--not a right answer but found a fix.

Process finished with exit code 0
```

Above, after the first iteration of the for loop, we examined the sum value and it displayed 20 instead of 12. We almost caught it here that we misplaced * (multiplication) in place of + (addition). So then, we took a step ahead to examined 'i' which is 2 at that point and tried **sum.append(i+num).** Then examining sum gave us 12 as recently added element. Hence we got the fix, so we skipped the remaining iterations of for loop by using the option 'u' (until). Then it moved to next step after loop. Here we used 'c'(continue) to continue the execution and it ended.

Now the fix,

```
def add_num(listA,num):
    sum=[]
    for i in listA:
        sum.append(i+num)
    return sum

listA = [2, 4, 6, 8]
num=10
result=add_num(listA,num)
print(result)
```

Output:

```
C:\Users\PycharmProjects\venv\Scripts\python.exe
C:/Users/PycharmProjects/debug_add.py
```

```
[12, 14, 16, 18]

Process finished with exit code 0
```

Hence debugged!!

Doesn't it seem easy, with no mess of print() statements?

Little recap to the options of pdb used here:

> *n — move to next line/step over definitions*
> *s — step into definitions (built-in / user defined)*
> *u — to skip remaining iterations in a loop*
> *c — continue execution or till the next breakpoint() is encountered*
> *l — Shows the current line of code to be executed with arrow "->"*
> *q — to quit the debugger*
> *variable name — to examine the current state of the variable*
> *h option name — displays help on the option provided*
> *h — to view options menu and explore more options as needed.*

Conclusion

pdb is a powerful weapon to debug Pythonic code which adds "effectiveness" as there's no mess of print()statements in your code and "efficiency" as it greatly reduces the time to debug.

With a bunch of options provided, it should be your "goto" tool when you encounter your next bug.

Happy Debugging!!

Source

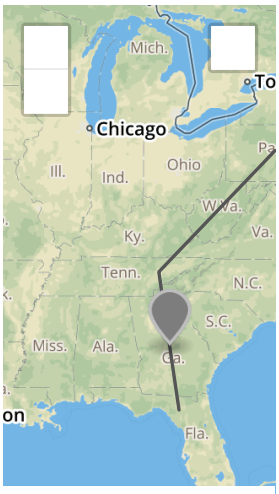
Python3 Python Debugger Python Programming Debugging Good Coding Practice


Download on the
App Store


GET IT ON
Google Play

along

Takes a LineString and returns a Point at a specified distance along the line.



© Mapbox © OpenStreetMap

npm install @turf/along

Arguments

Argument	Type	Description
line	Feature <LineString>	input line
distance	number	distance along the line
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	can be degrees, radians, miles, or kilometers

Returns

Feature <Point> - Point distance units along the line

Example

```
var line = turf.lineString([[-83, 30], [-84, 36], [-78, 41]]
var options = {units: 'miles'};

var along = turf.along(line, 200, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

area

npm install @turf/area

Takes one or more features and returns their area in square meters.

Arguments

Argument	Type	Description
geojson	GeoJSON	input GeoJSON feature(s)

Returns

number - area in square meters

Example

```
var polygon = turf.polygon([[[125, -15], [113, -22], [154, -
var area = turf.area(polygon);
```

bbox

npm install @turf/bbox

Takes a set of features, calculates the bbox of all input features, and returns a bounding box.

Arguments

Argument	Type	Description
geojson	GeoJSON	any GeoJSON object

Returns

BBox - bbox extent in minX, minY, maxX, maxY order

Example

```
var line = turf.lineString([[-74, 40], [-78, 42], [-82, 35]]
var bbox = turf.bbox(line);
```

```
var bboxPolygon = turf.bboxPolygon(bbox);
```


bboxPolygon

npm install @turf/bbox-polygon

Takes a bbox and returns an equivalent polygon.

Arguments

Argument	Type	Description
bbox	BBox	extent in minX, minY, maxX, maxY order
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Properties	{}	Translate properties to Polygon
id	(string|number)	{}	Translate Id to Polygon

Returns

Feature <Polygon> - a Polygon representation of the bounding box

Example

```
var bbox = [0, 0, 10, 10];

var poly = turf.bboxPolygon(bbox);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

bearing

npm install @turf/bearing

Takes two points and finds the geographic bearing between them, i.e. the angle measured in degrees from the north line (0 degrees)

Arguments

Argument	Type	Description
start	Coord	starting Point
end	Coord	ending Point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
final	boolean	false	calculates the final bearing if true

Returns

number - bearing in decimal degrees, between -180 and 180 degrees (positive clockwise)

Example

```
var point1 = turf.point([-75.343, 39.984]);
var point2 = turf.point([-75.534, 39.123]);

var bearing = turf.bearing(point1, point2);
```

center

npm install @turf/center

Takes a Feature or FeatureCollection and returns the absolute center point of all features.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	an Object that is used as the

Returns

Feature <Point> - a Point feature at the absolute center point of all input features

Example

```
var features = turf.featureCollection([
  turf.point( [-97.522259, 35.4691]),
  turf.point( [-97.502754, 35.463455]),
  turf.point( [-97.508269, 35.463245])
]);

var center = turf.center(features);
```

centerOfMass

npm install @turf/center-of-mass

Takes any Feature or a FeatureCollection and returns its center of mass using this formula: Centroid of Polygon.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered

Argument	Type	Description
properties	Object	an Object that is used as the Feature 's properties

Returns

Feature <Point> - the center of mass

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31]

var center = turf.centerOfMass(polygon);
```

centroid

npm install @turf/centroid

Takes one or more features and calculates the centroid using the mean of all vertices. This lessens the effect of small islands and artifacts when calculating the centroid of a set of polygons.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be centered
properties	Object	an Object that is used as the Feature 's properties

Returns

Feature <Point> - the centroid of the input features

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31]

var centroid = turf.centroid(polygon);
```

destination

npm install @turf/destination

Takes a Point and calculates the location of a destination point given a distance in degrees, radians, miles, or kilometers; and bearing in degrees. This uses the Haversine formula to account for global curvature.

Arguments

Argument	Type	Description
origin	Coord	starting point
distance	number	distance from the origin point
bearing	number	ranging from -180 to 180
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	miles, kilometers, degrees, or radians
properties	Object	{}	Translate properties to Point

Returns

Feature <Point> - destination point

Example

```
var point = turf.point([-75.343, 39.984]);
var distance = 50;
var bearing = 90;
var options = {units: 'miles'};
```

```
var destination = turf.destination(point, distance, bearing,
```


distance

npm install @turf/distance

Calculates the distance between two points in degrees, radians, miles, or kilometers. This uses the Haversine formula to account for global curvature.

Arguments

Argument	Type	Description
from	Coord	origin point
to	Coord	destination point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

number - distance between the two points

Example

```
var from = turf.point([-75.343, 39.984]);
var to = turf.point([-75.534, 39.123]);
var options = {units: 'miles'};

var distance = turf.distance(from, to, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

envelope

npm install @turf/envelope

Takes any number of features and returns a rectangular Polygon that encompasses all vertices.

Arguments

Argument	Type	Description
geojson	GeoJSON	input features

Returns

Feature <Polygon> - a rectangular Polygon feature that encompasses all vertices

Example

```
var features = turf.featureCollection([
  turf.point([-75.343, 39.984], {"name": "Location A"}),
  turf.point([-75.833, 39.284], {"name": "Location B"}),
  turf.point([-75.534, 39.123], {"name": "Location C"})
]);

var enveloped = turf.envelope(features);
```

length

npm install @turf/length

Takes a GeoJSON and measures its length in the specified units, (Multi)Point 's distance are ignored.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to measure
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	kilometers	can be degrees, radians, miles, or kilometers

Returns

number - length of GeoJSON

Example

```
var line = turf.lineString([[115, -32], [131, -22], [143, -2
var length = turf.length(line, {units: 'miles'});
```

midpoint

npm install @turf/midpoint

Takes two points and returns a point midway between them. The midpoint is calculated geodesically, meaning the curvature of the earth is taken into account.

Arguments

Argument	Type	Description
point1	Coord	first point
point2	Coord	second point

Returns

Feature <Point> - a point midway between pt1 and pt2

Example

```
var point1 = turf.point([144.834823, -37.771257]);
var point2 = turf.point([145.14244, -37.830937]);

var midpoint = turf.midpoint(point1, point2);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

pointOnFeature

npm install @turf/point-on-feature

Takes a Feature or FeatureCollection and returns a Point guaranteed to be on the surface of the feature.

Arguments

Argument	Type	Description
geojson	GeoJSON	any Feature or FeatureCollection

Returns

Feature <Point> - a point on the surface of input

Example

```
var polygon = turf.polygon([[
  [116, -36],
  [131, -32],
  [146, -43],
  [155, -25],
  [133, -9],
  [111, -22],
  [116, -36]
]]);

var pointOnPolygon = turf.pointOnFeature(polygon);
```

polygonTangent

npm install @turf/polygon-tangents

Finds the tangents of a (Multi)Polygon from a Point.

Arguments

Argument	Type	Description
pt	Coord	to calculate the tangent points from
polygon	Feature <(Polygon|MultiPolygon)>	to get tangents from

Returns

FeatureCollection <Point> - Feature Collection containing the two tangent points

Example

```
var polygon = turf.polygon([[[11, 0], [22, 4], [31, 0], [31,
var point = turf.point([61, 5]);

var tangents = turf.polygonTangents(point, polygon)
```

pointToLineDista

npm install @turf/point-to-line-distance

Returns the minimum distance between a Point and a LineString , being the distance from a line the minimum distance between the point and any segment of the LineString.

Arguments

Argument	Type	Description
pt	Coord	Feature or Geometry
line	Feature <LineString>	GeoJSON Feature or Geometry
options	Object	Optional parameters: see below

Options

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers
mercator	boolean	false	if distance should be on Mercator or WGS84 projection

Returns

number - distance between point and line

Example

```
var pt = turf.point([0, 0]);
var line = turf.lineString([[1, 1],[-1, 1]]);

var distance = turf.pointToLineDistance(pt, line, {units: 'm
//=69.11854715938406
```

rhumbBearing

npm install @turf/rhumb-bearing

Takes two points and finds the bearing angle between them along a Rhumb line i.e. the angle measured in degrees start the north line (0 degrees)

Arguments

Argument	Type	Description
start	Coord	starting Point
end	Coord	ending Point
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
final	boolean	false	calculates the final bearing if true

Returns

number - bearing from north in decimal degrees, between -180 and 180 degrees (positive clockwise)

Example

```
var point1 = turf.point([-75.343, 39.984], {"marker-color": '
var point2 = turf.point([-75.534, 39.123], {"marker-color": '

var bearing = turf.rhumbBearing(point1, point2);
```

rhumbDestinatic

npm install @turf/rhumb-destination

Returns the destination Point having travelled the given distance along a Rhumb line from the origin Point with the (varant) given bearing.

Arguments

Argument	Type	Description
origin	Coord	starting point
distance	number	distance from the starting point
bearing	number	varant bearing angle ranging from -180 to 180 degrees from north
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers
properties	Object	{}	translate properties to destination point

Returns

Feature <Point> - Destination point.

Example

```
var pt = turf.point([-75.343, 39.984], {"marker-color": "F00'
var distance = 50;
var bearing = 90;
var options = {units: 'miles'};

var destination = turf.rhumbDestination(pt, distance, bearing
```

rhumbDistance

npm install @turf/rhumb-distance

Calculates the distance along a rhumb line between two points in degrees, radians, miles, or kilometers.

Arguments

Argument	Type	Description
from	Coord	origin point
to	Coord	destination point
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	can be degrees, radians, miles, or kilometers

Returns

number - distance between the two points

Example

```
var from = turf.point([-75.343, 39.984]);
var to = turf.point([-75.534, 39.123]);
var options = {units: 'miles'};

var distance = turf.rhumbDistance(from, to, options);
```

square

npm install @turf/square

Takes a bounding box and calculates the minimum square bounding box that would contain the input.

Arguments

Argument	Type	Description
bbox	BBox	extent in west, south, east, north order

Returns

BBox - a square surrounding bbox

Example

```
var bbox = [-20, -20, -15, 0];
var squared = turf.square(bbox);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

greatCircle

npm install @turf/great-circle

Calculate great circles routes as LineString

Arguments

Argument	Type	Description
start	Coord	source point feature
end	Coord	destination point feature
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	line feature properties
npoints	number	100	number of points
offset	number	10	offset controls the likelyhood that lines will be split which cross the dateline. The higher the number the more likely.

Returns

Feature <LineString> - great circle line feature

Example

```
var start = turf.point([-122, 48]);
var end = turf.point([-77, 39]);

var greatCircle = turf.greatCircle(start, end, {'name': 'Sea
```

cleanCoords

npm install @turf/clean-coords

Removes redundant coordinates from any GeoJSON Geometry.

Arguments

Argument	Type	Description
geojson	(Geometry|Feature)	Feature or Geometry
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated

Returns

(Geometry|Feature) - the cleaned input Feature/Geometry

Example

```
var line = turf.lineString([[0, 0], [0, 2], [0, 5], [0, 8],
var multiPoint = turf.multiPoint([[0, 0], [0, 0], [2, 2]]);

turf.cleanCoords(line).geometry.coordinates;
//= [[0, 0], [0, 10]]

turf.cleanCoords(multiPoint).geometry.coordinates;
//= [[0, 0], [2, 2]]
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

flip

npm install @turf/flip

Takes input features and flips all of their coordinates from [x, y] to [y, x].

Arguments

Argument	Type	Description
geojson	GeoJSON	input features
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - a feature or set of features of the same type as input with flipped coordinates

Example

```
var serbia = turf.point([20.566406, 43.421008]);

var saudiArabia = turf.flip(serbia);
```

rewind

npm install @turf/rewind

Rewind (Multi)LineString or (Multi)Polygon outer ring counterclockwise and inner rings clockwise (Uses Shoelace Formula).

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Arguments

Argument	Type	Description
geojson	GeoJSON	input GeoJSON Polygon
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
reverse	boolean	false	enable reverse winding
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - rewind Polygon

Example

```
var polygon = turf.polygon([[[121, -29], [138, -29], [138, -1

var rewind = turf.rewind(polygon);
```

round

npm install @turf/helpers

Round number to precision

Note: round is part of the @turf/helpers module.

To use it as a

Arguments

Argument	Type	Description
num	number	Number

Argument	Type	Description
precision	number	Precision

stand-alone module will need to import @turf/helpers and call the round method.

Returns

number - rounded number

Example

```
turf.round(120.4321)
//=120

turf.round(120.4321, 2)
//=120.43
```

truncate

npm install @turf/truncate

Takes a GeoJSON Feature or FeatureCollection and truncates the precision of the geometry.

Arguments

Argument	Type	Description
geojson	GeoJSON	any GeoJSON Feature, FeatureCollection, Geometry or GeometryCollection.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
precision	number	6	coordinate decimal precision

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
coordinates	number	3	maximum number of coordinates (primarly used to remove z coordinates)
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - layer with truncated geometry

Example

```
var point = turf.point([
    70.46923055566859,
    58.11088890802906,
    1508
]);
var options = {precision: 3, coordinates: 2};
var truncated = turf.truncate(point, options);
//=truncated.geometry.coordinates => [70.469, 58.111]
```

bboxClip

npm install @turf/bbox-clip

Takes a Feature and a bbox and clips the feature to the bbox using lineclip. May result in degenerate edges when clipping Polygons.

Arguments

Argument	Type	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
feature	Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>	feature to clip to the bbox
bbox	BBox	extent in minX, minY, maxX, maxY order

Returns

Feature <(LineString|MultiLineString|Polygon|MultiPolygon)> - clipped Feature

Example

```
var bbox = [0, 0, 10, 10];
var poly = turf.polygon([[[2, 2], [8, 4], [12, 8], [3, 7], [
var clipped = turf.bboxClip(poly, bbox);
```

bezierSpline

npm install @turf/bezier-spline

Takes a line and returns a curved version by applying a Bezier spline algorithm.

Arguments

Argument	Type	Description
line	Feature <LineString>	input LineString
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
resolution	number	10000	time in milliseconds between points
sharpness	number	0.85	a measure of how curvy the path should be between splines

Returns

Feature <LineString> - curved line

Example

```
var line = turf.lineString([
  [-76.091308, 18.427501],
  [-76.695556, 18.729501],
  [-76.552734, 19.40443],
  [-74.61914, 19.134789],
  [-73.652343, 20.07657],
  [-73.157958, 20.210656]
]);

var curved = turf.bezierSpline(line);
```

buffer

npm install @turf/buffer

Calculates a buffer for input features for a given radius. Units supported are miles, kilometers, and degrees.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Geometry|Feature)	input to be buffered

Argument	Type	Description
radius	number	distance to draw the buffer (negative values are allowed)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	any of the options supported by turf units
steps	number	64	number of steps

Returns

(FeatureCollection|Feature <(Polygon|MultiPolygon)>|undefined) - buffered features

Example

```
var point = turf.point([-90.548630, 14.616599]);
var buffered = turf.buffer(point, 500, {units: 'miles'});
```

circle

npm install @turf/circle

Takes a Point and calculates the circle polygon given a radius in degrees, radians, miles, or kilometers; and steps for precision.

Arguments

Argument	Type	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
center	(Feature <Point>|Array)	center point
radius	number	radius of the circle
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
steps	number	64	number of steps
units	string	'kilometers'	miles, kilometers, degrees, or radians
properties	Object	{}	properties

Returns

Feature <Polygon> - circle polygon

Example

```
var center = [-75.343, 39.984];
var radius = 5;
var options = {steps: 10, units: 'kilometers', properties: {
var circle = turf.circle(center, radius, options);
```

clone

npm install @turf/clone

Returns a cloned copy of the passed GeoJSON Object, including possible 'Foreign Members'. ~3-5x faster than the common JSON.parse + JSON.stringify combo method.

Arguments

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Description
geojson	GeoJSON	GeoJSON Object

Returns

GeoJSON - cloned GeoJSON Object

Example

```
var line = turf.lineString([[-74, 40], [-78, 42], [-82, 35]]

var lineCloned = turf.clone(line);
```

concave

npm install @turf/concave

Takes a set of points and returns a concave hull Polygon or MultiPolygon. Internally, this uses turf-tin to generate geometries.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	input points
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
maxEdge	number	Infinity	the length (in 'units') of an edge necessary for part of the hull to become concave.

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

(Feature <(Polygon|MultiPolygon)>|null) - a concave hull (null value is returned if unable to compute hull)

Example

```
var points = turf.featureCollection([
  turf.point([-63.601226, 44.642643]),
  turf.point([-63.591442, 44.651436]),
  turf.point([-63.580799, 44.648749]),
  turf.point([-63.573589, 44.641788]),
  turf.point([-63.587665, 44.64533]),
  turf.point([-63.595218, 44.64765])
]);
var options = {units: 'miles', maxEdge: 1};

var hull = turf.concave(points, options);
```

convex

npm install @turf/convex

Takes a Feature or a FeatureCollection and returns a convex hull Polygon.

Arguments

Argument	Type	Description
geojson	GeoJSON	input Feature or FeatureCollection
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
concavity	number	Infinity	1 - thin shape. Infinity - convex hull.

Returns

Feature <Polygon> - a convex hull

Example

```
var points = turf.featureCollection([
  turf.point([10.195312, 43.755225]),
  turf.point([10.404052, 43.8424511]),
  turf.point([10.579833, 43.659924]),
  turf.point([10.360107, 43.516688]),
  turf.point([10.14038, 43.588348]),
  turf.point([10.195312, 43.755225])
]);

var hull = turf.convex(points);
```

difference

npm install @turf/difference

Finds the difference between two polygons by clipping the second polygon from the first.

Arguments

Argument	Type	Description
polygon1	Feature <(Polygon|MultiPolygon)>	input Polygon feature
polygon2	Feature <(Polygon|MultiPolygon)>	Polygon feature to difference from polygon1

Returns

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

(Feature <(Polygon|MultiPolygon)>|null) - a Polygon or MultiPolygon feature showing the area of polygon1 excluding the area of polygon2 (if empty returns null)

Example

```
var polygon1 = turf.polygon([[
  [128, -26],
  [141, -26],
  [141, -21],
  [128, -21],
  [128, -26]
]], {
  "fill": "#F00",
  "fill-opacity": 0.1
});
var polygon2 = turf.polygon([[
  [126, -28],
  [140, -28],
  [140, -20],
  [126, -20],
  [126, -28]
]], {
  "fill": "#00F",
  "fill-opacity": 0.1
});

var difference = turf.difference(polygon1, polygon2);
```

dissolve

npm install @turf/dissolve

Dissolves a FeatureCollection of polygon features, filtered by an optional property name:value. Note that mulitpolygon features within the collection are not supported

Arguments

Argument	Type	Description

Argument	Type	Description
featureCollection	FeatureCollection <Polygon>	input feature collection to be dissolved
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
propertyName	(string)		features with equals 'propertyName' in

Returns

FeatureCollection <Polygon> - a FeatureCollection containing the dissolved polygons

Example

```
var features = turf.featureCollection([
  turf.polygon([[[0, 0], [0, 1], [1, 1], [1, 0], [0, 0]]], {
  turf.polygon([[[0, -1], [0, 0], [1, 0], [1, -1], [0,-1]]],
  turf.polygon([[[1,-1],[1, 0], [2, 0], [2, -1], [1, -1]]],
]);

var dissolved = turf.dissolve(features, {propertyName: 'comb
```

intersect

npm install @turf/intersect

Takes two polygons and finds their intersection. If they share a border, returns the border; if they don't intersect, returns undefined.

Arguments

Argument	Type	Description

Argument	Type	Description
poly1	Feature <Polygon>	the first polygon
poly2	Feature <Polygon>	the second polygon

Returns

(Feature|null) - returns a feature representing the point(s) they share (in case of a Point or MultiPoint), the borders they share (in case of a LineString or a MultiLineString), the area they share (in case of Polygon or MultiPolygon). If they do not share any point, returns null.

Example

```
var poly1 = turf.polygon([[
  [-122.801742, 45.48565],
  [-122.801742, 45.60491],
  [-122.584762, 45.60491],
  [-122.584762, 45.48565],
  [-122.801742, 45.48565]
]]);

var poly2 = turf.polygon([[
  [-122.520217, 45.535693],
  [-122.64038, 45.553967],
  [-122.720031, 45.526554],
  [-122.669906, 45.507309],
  [-122.723464, 45.446643],
  [-122.532577, 45.408574],
  [-122.487258, 45.477466],
  [-122.520217, 45.535693]
]]);

var intersection = turf.intersect(poly1, poly2);
```

lineOffset

npm install @turf/line-offset

Takes a line and returns a line at offset by the specified distance.

Arguments

Argument	Type	Description
geojson	(Geometry|Feature <(LineString|MultiLineString)>)	input GeoJSON
distance	number	distance to offset the line (can be of negative value)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, kilometers, inches, yards, meters

Returns

Feature <(LineString|MultiLineString)> - Line offset from the input line

Example

```
var line = turf.lineString([[-83, 30], [-84, 36], [-78, 41]]

var offsetLine = turf.lineOffset(line, 2, {units: 'miles'});
```

simplify

npm install @turf/simplify

Takes a GeoJSON object and returns a simplified version. Internally uses simplify-js to perform

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

simplification using the Ramer-Douglas-Peucker algorithm.

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be simplified
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
tolerance	number	1	simplification tolerance
highQuality	boolean	false	whether or not to spend more time to create a higher-quality simplification with a different algorithm
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - a simplified GeoJSON

Example

```
var geojson = turf.polygon([[
  [-70.603637, -33.399918],
  [-70.614624, -33.395332],
  [-70.639343, -33.392466],
  [-70.659942, -33.394759],
  [-70.683975, -33.404504],
  [-70.697021, -33.419406],
  [-70.701141, -33.434306],
  [-70.700454, -33.446339],
  [-70.694274, -33.458369],
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
  [-70.682601, -33.465816],
  [-70.668869, -33.472117],
  [-70.646209, -33.473835],
  [-70.624923, -33.472117],
  [-70.609817, -33.468107],
  [-70.595397, -33.458369],
  [-70.587158, -33.442901],
  [-70.587158, -33.426283],
  [-70.590591, -33.414248],
  [-70.594711, -33.406224],
  [-70.603637, -33.399918]
]]);
var options = {tolerance: 0.01, highQuality: false};
var simplified = turf.simplify(geojson, options);
```

tesselate

npm install @turf/tesselate

Tesselates a Feature into a FeatureCollection of triangles using earcut.

Arguments

Argument	Type	Description
poly	Feature <Polygon>	the polygon to tesselate

Returns

FeatureCollection <Polygon> - a geometrycollection feature

Example

```
var poly = turf.polygon([[[11, 0], [22, 4], [31, 0], [31, 11
var triangles = turf.tesselate(poly);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

transformRotate

npm install @turf/transform-rotate

Rotates any geojson Feature or Geometry of a specified angle, around its centroid or a given pivot point; all rotations follow the right-hand rule: https://en.wikipedia.org/wiki/Right-hand_rule

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be rotated
angle	number	of rotation (along the vertical axis), from North in decimal degrees, negative clockwise
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
pivot	Coord	'centroid'	point around which the rotation will be performed
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - the rotated GeoJSON feature

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var options = {pivot: [0, 25]};
var rotatedPoly = turf.transformRotate(poly, 10, options);
```

transformTransla

npm install @turf/transform-translate

Moves any geojson Feature or Geometry of a specified distance along a Rhumb Line on the provided direction angle.

Arguments

Argument	Type	Description
geojson	GeoJSON	object to be translated
distance	number	length of the motion; negative values determine motion in opposite direction
direction	number	of the motion; angle from North in decimal degrees, positive clockwise
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	in which
zTranslation	number	0	length of the vertical motion, same unit of distance
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - the translated GeoJSON object

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var translatedPoly = turf.transformTranslate(poly, 100, 35);
```

transformScale

npm install @turf/transform-scale

Scale a GeoJSON from a given point by a factor of scaling (ex: factor=2 would make the GeoJSON 200% larger). If a FeatureCollection is provided, the origin point will be calculated based on each individual Feature.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON to be scaled
factor	number	of scaling, positive or negative values greater than 0
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
origin	(string|Coord)	'centroid'	Point from which the scaling will occur (string options: sw/se/nw/ne/center/centroid)
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - scaled GeoJSON

Example

```
var poly = turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
var scaledPoly = turf.transformScale(poly, 3);
```

union

npm install @turf/union

Takes two or more polygons and returns a combined polygon. If the input polygons are not contiguous, this function returns a MultiPolygon feature.

Arguments

Argument	Type	Description
A	...Feature <Polygon>	polygon to combine

Returns

Feature <(Polygon|MultiPolygon)> - a combined Polygon or MultiPolygon feature

Example

```
var poly1 = turf.polygon([[
    [-82.574787, 35.594087],
    [-82.574787, 35.615581],
    [-82.545261, 35.615581],
    [-82.545261, 35.594087],
    [-82.574787, 35.594087]
]], {"fill": "#0f0"});
var poly2 = turf.polygon([[
    [-82.560024, 35.585153],
    [-82.560024, 35.602602],
    [-82.52964, 35.602602],
    [-82.52964, 35.585153],
    [-82.560024, 35.585153]
]], {"fill": "#00f"});

var union = turf.union(poly1, poly2);
```

TURF

voronoi

npm install @turf/voronoi

Takes a FeatureCollection of points, and a bounding box, and returns a FeatureCollection of Voronoi polygons.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to find the Voronoi polygons around.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
bbox	Array	[-180,-85,180,-85]	clipping rectangle, in

Returns

FeatureCollection <Polygon> - a set of polygons, one per input point.

Example

```
var options = {
  bbox: [-70, 40, -60, 60]
};
var points = turf.randomPoint(100, options);
var voronoiPolygons = turf.voronoi(points, options);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

combine

npm install @turf/combine

Combines a FeatureCollection of Point , LineString , or Polygon features into MultiPoint , MultiLineString , or MultiPolygon features.

Arguments

Argument	Type	Description
fc	FeatureCollection <(Point|LineString|Polygon)>	a FeatureCollection of any type

Returns

FeatureCollection <(MultiPoint|MultiLineString|MultiPolygon)> - a FeatureCollection of corresponding type to input

Example

```
var fc = turf.featureCollection([
  turf.point([19.026432, 47.49134]),
  turf.point([19.074497, 47.509548])
]);

var combined = turf.combine(fc);
```

explode

npm install @turf/explode

Takes a feature or set of features and returns all positions as points.

Arguments

Argument	Type	Description
geojson	GeoJSON	input features

Returns

FeatureCollection <point> - points representing the exploded input features

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Throws

Error - if it encounters an unknown geometry type

Example

```
var polygon = turf.polygon([[[-81, 41], [-88, 36], [-84, 31]

var explode = turf.explode(polygon);
```

flatten

npm install @turf/flatten

Flattens any GeoJSON to a FeatureCollection inspired by geojson-flatten.

Arguments

Argument	Type	Description
geojson	GeoJSON	any valid GeoJSON Object

Returns

FeatureCollection - all Multi-Geometries are flattened into single Features

Example

```
var multiGeometry = turf.multiPolygon([
  [[[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0],
  [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
  [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [
]);

var flatten = turf.flatten(multiGeometry);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lineToPolygon

npm install @turf/line-to-polygon

Converts (Multi)LineString(s) to Polygon(s).

Arguments

Argument	Type	Description
lines	(FeatureCollection|Feature <(LineString|MultiLineString)>)	Features to convert
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
properties	Object	{}	translates GeoJSON properties to Feature
autoComplete	boolean	true	auto complete linestrings (matches first & last coordinates)
orderCoords	boolean	true	sorts linestrings to place outer ring at the first position of the coordinates

Returns

Feature <(Polygon|MultiPolygon)> - converted to Polygons

Example

```
var line = turf.lineString([[125, -30], [145, -30], [145, -2(

var polygon = turf.lineToPolygon(line);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

polygonize

npm install @turf/polygonize

Polygonizes (Multi)LineString(s) into Polygons.

Arguments

Argument	Type	Description
geoJson	(FeatureCollection|Geometry|Feature <(LineString|MultiLineString)>)	Lines in order to polygonize

Returns

FeatureCollection <Polygon> - Polygons created

Throws

Error - if geoJson is invalid.

polygonToLine

npm install @turf/polygon-to-line

Converts a Polygon to (Multi)LineString or MultiPolygon to a FeatureCollection of (Multi)LineString.

Arguments

Argument	Type	Description
polygon	Feature <(Polygon|MultiPolygon)>	Feature to convert
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
properties	Object	{}	translates GeoJSON properties to Feature

Returns

(FeatureCollection|Feature <(LineString|MultiLinestring)>) - converted (Multi)Polygon to (Multi)LineString

Example

```
var poly = turf.polygon([[[125, -30], [145, -30], [145, -20]

var line = turf.polygonToLine(poly);
```

kinks

npm install @turf/kinks

Takes a linestring , multi-linestring , multi-polygon , or polygon and returns points at all self-intersections.

Arguments

Argument	Type	Description
featureIn	Feature <(LineString|MultiLineString|MultiPolygon|Polygon)>	input feature

Returns

FeatureCollection <Point> - self-intersections

Example

```
var poly = turf.polygon([[
  [-12.034835, 8.901183],
  [-12.060413, 8.899826],
  [-12.03638, 8.873199],
  [-12.059383, 8.871418],
```

```
  [-12.034835, 8.901183]
]]);

var kinks = turf.kinks(poly);
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lineArc

npm install @turf/line-arc

Creates a circular arc, of a circle of the given radius and center point, between bearing1 and bearing2; 0 bearing is North of center point, positive clockwise.

Arguments

Argument	Type	Description
center	Coord	center point
radius	number	radius of the circle
bearing1	number	angle, in decimal degrees, of the first radius of the arc
bearing2	number	angle, in decimal degrees, of the second radius of the arc
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
steps	number	64	number of steps
units	string	'kilometers'	miles, kilometers, degrees, or radians

Returns

Feature <LineString> - line arc

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
var center = turf.point([-75, 40]);
var radius = 5;
var bearing1 = 25;
var bearing2 = 47;

var arc = turf.lineArc(center, radius, bearing1, bearing2);
```

lineChunk

npm install @turf/line-chunk

Divides a LineString into chunks of a specified length. If the line is shorter than the segment length then the original line is returned.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Geometry|Feature<(LineString|MultiLineString)>)	the lines to split
segmentLength	number	how long to make each segment
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	units can be degrees, radians, miles, or kilometers

Prop	Type	Default	Description
reverse	boolean	false	reverses coordinates to start the first chunked segment at the end

Returns

FeatureCollection <LineString> - collection of line segments

Example

```
var line = turf.lineString([[-95, 40], [-93, 45], [-85, 50]]]

var chunk = turf.lineChunk(line, 15, {units: 'miles'});
```

lineIntersect

npm install @turf/line-intersect

Takes any LineString or Polygon GeoJSON and returns the intersecting point(s).

Arguments

Argument	Type	Description
line1	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
line2	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon

Returns

FeatureCollection <Point> - point(s) that intersect both

Example

```
var line1 = turf.lineString([[126, -11], [129, -21]]);
var line2 = turf.lineString([[123, -18], [131, -14]]);
var intersects = turf.lineIntersect(line1, line2);
```

lineOverlap

npm install @turf/line-overlap

Takes any LineString or Polygon and returns the overlapping lines between both features.

Arguments

Argument	Type	Description
line1	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
line2	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	any LineString or Polygon
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
tolerance	number	0	Tolerance distance to match overlapping line segments (in kilometers)

Returns

FeatureCollection <LineString> - lines(s) that are overlapping between both features

Example

```
var line1 = turf.lineString([[115, -35], [125, -30], [135, -
var line2 = turf.lineString([[115, -25], [125, -30], [135, -

var overlapping = turf.lineOverlap(line1, line2);
```

lineSegment

npm install @turf/line-segment

Creates a FeatureCollection of 2-vertex LineString segments from a (Multi)LineString or (Multi)Polygon.

Arguments

Argument	Type	Description
geojson	(Geometry|FeatureCollection|Feature <(LineString|MultiLineString|MultiPolygon|Polygon)>)	GeoJSON Polygon or LineString

Returns

FeatureCollection <LineString> - 2-vertex line segments

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10
var segments = turf.lineSegment(polygon);
```

lineSlice

npm install @turf/line-slice

Takes a line , a start Point , and a stop point and returns a subsection of the line in-between those

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

points. The start & stop points don't need to fall exactly on the line.

Arguments

Argument	Type	Description
startPt	Coord	starting point
stopPt	Coord	stopping point
line	(Feature <LineString>|LineString)	line to slice

Returns

Feature <LineString> - sliced line

Example

```
var line = turf.lineString([
    [-77.031669, 38.878605],
    [-77.029609, 38.881946],
    [-77.020339, 38.884084],
    [-77.025661, 38.885821],
    [-77.021884, 38.889563],
    [-77.019824, 38.892368]
]);
var start = turf.point([-77.029609, 38.881946]);
var stop = turf.point([-77.021884, 38.889563]);

var sliced = turf.lineSlice(start, stop, line);
```

lineSliceAlong

npm install @turf/line-slice-along

Takes a line , a specified distance along the line to a start Point , and a specified distance along the line to a stop point and returns a subsection of the line in-between those points.

Arguments

Argument	Type	Description

Argument	Type	Description
line	(Feature <LineString>|LineString)	input line
startDist	number	distance along the line to starting point
stopDist	number	distance along the line to ending point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

Feature <LineString> - sliced line

Example

```
var line = turf.lineString([[7, 45], [9, 45], [14, 40], [14,
var start = 12.5;
var stop = 25;
var sliced = turf.lineSliceAlong(line, start, stop, {units:
```

lineSplit

npm install @turf/line-split

Split a LineString by another GeoJSON Feature.

Arguments

Argument	Type	Description
line	Feature <LineString>	LineString Feature to split
splitter	Feature	Feature used to split line

Returns

FeatureCollection <LineString> - Split LineStrings

Example

```
var line = turf.lineString([[120, -25], [145, -25]]);
var splitter = turf.lineString([[130, -15], [130, -35]]);

var split = turf.lineSplit(line, splitter);
```

mask

npm install @turf/mask

Takes any type of polygon and an optional mask and returns a polygon exterior ring with holes.

Arguments

Argument	Type	Description
polygon	(FeatureCollection|Feature <(Polygon|MultiPolygon)>)	GeoJSON Polygon used as interior rings or holes.
mask	(Feature <Polygon>)	GeoJSON Polygon used as the exterior ring (if undefined, the world extent is used)

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Returns

Feature <Polygon> - Masked Polygon (exterior ring with holes).

Example

```
var polygon = turf.polygon([[[112, -21], [116, -36], [146, -
var mask = turf.polygon([[[90, -55], [170, -55], [170, 10],

var masked = turf.mask(polygon, mask);
```

nearestPointOnL

npm install @turf/nearest-point-on-line

Takes a Point and a LineString and calculates the closest Point on the (Multi)LineString.

Arguments

Argument	Type	Description
lines	(Geometry|Feature <(LineString|MultiLineString)>)	lines to snap to
pt	(Geometry|Feature <Point>|Array)	point to snap from
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	can be degrees, radians, miles, or kilometers

Returns

Feature <Point> - closest point on the line to point. The properties object will contain three

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

values: index : closest point was found on nth line part, dist : distance between pt and the closest point, location : distance along the line between start and the closest point.

Example

```
var line = turf.lineString([
    [-77.031669, 38.878605],
    [-77.029609, 38.881946],
    [-77.020339, 38.884084],
    [-77.025661, 38.885821],
    [-77.021884, 38.889563],
    [-77.019824, 38.892368]
]);
var pt = turf.point([-77.037076, 38.884017]);

var snapped = turf.nearestPointOnLine(line, pt, {units: 'mil
```

sector

npm install @turf/sector

Creates a circular sector of a circle of given radius and center Point , between (clockwise) bearing1 and bearing2; 0 bearing is North of center point, positive clockwise.

Arguments

Argument	Type	Description
center	Coord	center point
radius	number	radius of the circle
bearing1	number	angle, in decimal degrees, of the first radius of the sector
bearing2	number	angle, in decimal degrees, of the second radius of the sector
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	miles, kilometers, degrees, or radians
steps	number	64	number of steps
properties	Properties	{}	Translate properties to Feature Polygon

Returns

Feature <Polygon> - sector polygon

Example

```
var center = turf.point([-75, 40]);
var radius = 5;
var bearing1 = 25;
var bearing2 = 45;

var sector = turf.sector(center, radius, bearing1, bearing2);
```

shortestPath

npm install @turf/shortest-path

Returns the shortest path from start to end without colliding with any Feature in obstacles

Arguments

Argument	Type	Description
start	Coord	point
end	Coord	point
options	Object	optional parameters

Options

Prop	Type	Default	Description
obstacles	((Geometry|Feature|FeatureCollection <Polygon>))		areas which path cannot travel
minDistance	(number)		minimum distance between shortest path and obstacles
units	string	'kilometers'	unit in which resolution & minimum distance will be expressed in; it can be degrees, radians, miles, kilometers, ...
resolution	number	100	distance between matrix points on which the path will be calculated

Returns

Feature <LineString> - shortest path between start and end

Example

```
var start = [-5, -6];
var end = [9, -6];
var options = {
  obstacles: turf.polygon([[[0, -7], [5, -7], [5, -3], [0, -
};

var path = turf.shortestPath(start, end, options);
```

unkinkPolygon

npm install @turf/unkink-polygon

Takes a kinked polygon and returns a feature collection of polygons that have no kinks. Uses simplepolygon internally.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature<(Polygon|MultiPolygon)>)	GeoJSON Polygon or MultiPolygon

Returns

FeatureCollection <Polygon> - Unkinked polygons

Example

```
var poly = turf.polygon([[[0, 0], [2, 0], [0, 2], [2, 2], [0,

var result = turf.unkinkPolygon(poly);
```

featureCollectior

npm install @turf/helpers

Takes one or more Features and creates a FeatureCollection.

Note: featureCollection is part of the @turf/helpers module.

To use it as a stand-alone

Arguments

Argument	Type	Description
features	Array<Feature>	input features

Argument	Type	Description
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

FeatureCollection - FeatureCollection of Features

Example

```
var locationA = turf.point([-75.343, 39.984], {name: 'Locati
var locationB = turf.point([-75.833, 39.284], {name: 'Locati
var locationC = turf.point([-75.534, 39.123], {name: 'Locati

var collection = turf.featureCollection([
  locationA,
  locationB,
  locationC
]);

//=collection
```

module will need to import @turf/helpers and call the featureCollection method.

feature

npm install @turf/helpers

Wraps a GeoJSON Geometry in a GeoJSON Feature.

Arguments

Argument	Type	Description
geometry	Geometry	input geometry

Note: feature is part of the @turf/helpers module.

To use it as a stand-alone

Argument	Type	Description
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

module will need to import @turf/helpers and call the feature method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature - a GeoJSON Feature

Example

```
var geometry = {
  "type": "Point",
  "coordinates": [110, 50]
};

var feature = turf.feature(geometry);

//=feature
```

geometryCollect

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
geometries	Array <Geometry>	an array of GeoJSON Geometries

Note: geometryCollection is part of the @turf/helpers module.

To use it as a stand-alone

Argument	Type	Description
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

module will need to import @turf/helpers and call the geometryCollection method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <GeometryCollection> - a GeoJSON GeometryCollection Feature

Example

```
var pt = {
    "type": "Point",
      "coordinates": [100, 0]
    };
var line = {
    "type": "LineString",
    "coordinates": [ [101, 0], [102, 1] ]
  };
var collection = turf.geometryCollection([pt, line]);

//=collection
```

lineString

npm install @turf/helpers

Creates a LineString Feature from an Array of Positions.

Note: lineString is part of the @turf/helpers module.

Arguments

Argument	Type	Description
coordinates	Array >	an array of Positions
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

To use it as a stand-alone module will need to import @turf/helpers and call the lineString method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <LineString> - LineString Feature

Example

```
var linestring1 = turf.lineString([[-24, 63], [-23, 60], [-2
var linestring2 = turf.lineString([[-14, 43], [-13, 40], [-1

//=linestring1
//=linestring2
```

multiLineString

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >>	an array of LineStrings
properties	Object	an Object of key-value pairs to add as properties

Note: multiLineString is part of the @turf/helpers module.

To use it as a stand-alone module will need

Argument	Type	Description
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiLineString> - a MultiLineString feature

Throws

Error - if no coordinates are passed

Example

```
var multiLine = turf.multiLineString([[[0,0],[10,10]]]);

//=multiLine
```

to import @turf/helpers and call the multiLineString method.

multiPoint

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >	an array of Positions
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Note: multiPoint is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the multiPoint method.

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiPoint> - a MultiPoint feature

Throws

Error - if no coordinates are passed

Example

```
var multiPt = turf.multiPoint([[0,0],[10,10]]);

//=multiPt
```

multiPolygon

npm install @turf/helpers

Creates a Feature based on a coordinate array. Properties can be added optionally.

Arguments

Argument	Type	Description
coordinates	Array >>>	an array of Polygons
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description

Note: multiPolygon is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the multiPolygon method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <MultiPolygon> - a multipolygon feature

Throws

Error - if no coordinates are passed

Example

```
var multiPoly = turf.multiPolygon([[[[0,0],[0,10],[10,10],[10

//=multiPoly
```

point

npm install @turf/helpers

Creates a Point Feature from a Position.

Note: point is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the point method.

Arguments

Argument	Type	Description
coordinates	Array	longitude, latitude position (each in decimal degrees)
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <Point> - a Point feature

Example

```
var point = turf.point([-75.343, 39.984]);

//=point
```

polygon

npm install @turf/helpers

Creates a Polygon Feature from an Array of LinearRings.

Note: polygon is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the polygon method.

Arguments

Argument	Type	Description
coordinates	Array >>	an array of LinearRings
properties	Object	an Object of key-value pairs to add as properties
options	Object	Optional Parameters

Options

Prop	Type	Default	Description
bbox	(Array)		Bounding Box Array
id	((string|number))		Identifier associated with the Feature

Returns

Feature <Polygon> - Polygon Feature

Example

```
var polygon = turf.polygon([[[-5, 52], [-4, 56], [-2, 51], [

//=polygon
```

randomPosition

npm install @turf/random

Returns a random position within a box.

Arguments

Argument	Type	Description
bbox	Array	a bounding box inside of which positions are placed.

Returns

Array - Position longitude, latitude

Note: randomPosition is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPosition method.

Example

```
var position = turf.randomPosition([-180, -90, 180, 90])
//=position
```

randomPoint

npm install @turf/random

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Returns a random point.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
bbox	Array	[-180,-90,180,90]	a bounding box inside of which geometries are placed.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

```
var points = turf.randomPoint(25, {bbox: [-180, -90, 180, 90]
//=points
```

Note: randomPoint is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPoint method.

randomLineStrir

npm install @turf/random

Returns a random linestring.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Note: randomLineString is part of the @turf/random module.

To use it as a stand-alone module will need to import

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Options

Prop	Type	Default	Description
bbox	Array	[-180,-90,180,90]	a bounding box inside of which geometries are placed.
num_vertices	number	10	is how many coordinates each LineString will contain.
max_length	number	0.0001	is the maximum number of decimal degrees that a vertex can be from its predecessor
max_rotation	number	Math.PI/8	is the maximum number of radians that a line segment can turn from the previous segment.

@turf/random and call the randomLineString method.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

```
var lineStrings = turf.randomLineString(25, {bbox: [-180, -9
//=lineStrings
```

randomPolygon

npm install @turf/random

Returns a random polygon.

Arguments

Argument	Type	Description
count	number	how many geometries will be generated
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Descript
bbox	Array	[-180,-90,180,90]	a bound box inside of which geometries are placed.
num_vertices	number	10	is how many coordinates each LineString will contain.
max_radial_length	number	10	is the maximum number of decimal degrees latitude or longitude that a vertex can reach out of the center of the Polygon.

Returns

FeatureCollection <Point> - GeoJSON FeatureCollection of points

Example

Note: randomPolygon is part of the @turf/random module.

To use it as a stand-alone module will need to import @turf/random and call the randomPolygon method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var polygons = turf.randomPolygon(25, {bbox: [-180, -90, 180
//=polygons
```

sample

npm install @turf/sample

Takes a FeatureCollection and returns a FeatureCollection with given number of features at random.

Arguments

Argument	Type	Description
featurecollection	FeatureCollection	set of input features
num	number	number of features to select

Returns

FeatureCollection - a FeatureCollection with n features

Example

```
var points = turf.randomPoint(100, {bbox: [-80, 30, -60, 60]}

var sample = turf.sample(points, 5);
```

interpolate

npm install @turf/interpolate

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Takes a set of points and estimates their 'property' values on a grid using the Inverse Distance Weighting (IDW) method.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	with known value
cellSize	number	the distance across each grid point
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
gridType	string	'square'	defines the output format based on a Grid Type (options: 'square' | 'point' | 'hex' | 'triangle')
property	string	'elevation'	the property name in
units	string	'kilometers'	used in calculating cellSize, can be degrees, radians, miles, or kilometers
weight	number	1	exponent regulating the distance-decay weighting

Returns

FeatureCollection <(Point|Polygon)> - grid of points or polygons with interpolated 'property'

Example

```
var points = turf.randomPoint(30, {bbox: [50, 30, 70, 50]});

// add a random property to each point
turf.featureEach(points, function(point) {
    point.properties.solRad = Math.random() * 50;
});
var options = {gridType: 'points', property: 'solRad', units
var grid = turf.interpolate(points, 100, options);
```

isobands

npm install @turf/isobands

Takes a grid FeatureCollection of Point features with z-values and an array of value breaks and generates filled contour isobands.

Arguments

Argument	Type	Description
pointGrid	FeatureCollection <Point>	input points
breaks	Array	where to draw contours
options	Object	options on output

Options

Prop	Type	Default	Description
zProperty	string	'elevation'	the property name in
commonProperties	Object	{}	GeoJSON properties passed to A isobands

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
breaksProperties	Array	[]	GeoJSON properties passed, in order, to the corresponde isoband (orc defined by breaks)

Returns

FeatureCollection <MultiPolygon> - a FeatureCollection of MultiPolygon features representing isobands

isolines

npm install @turf/isolines

Takes a grid FeatureCollection of Point features with z-values and an array of value breaks and generates isolines.

Arguments

Argument	Type	Description
pointGrid	FeatureCollection <Point>	input points
breaks	Array	values of zProperty where to draw isolines
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
zProperty	string	'elevation'	the property name in

Prop	Type	Default	Description
commonProperties	Object	{}	GeoJSON properties passed to A isolines
breaksProperties	Array	[]	GeoJSON properties passed, in order, to the corresponde isoline; the breaks array will define tl order in whi the isolines are created

Returns

FeatureCollection <MultiLineString> - a FeatureCollection of MultiLineString features representing isolines

Example

```
// create a grid of points with random z-values in their prop
var extent = [0, 30, 20, 50];
var cellWidth = 100;
var pointGrid = turf.pointGrid(extent, cellWidth, {units: 'm

for (var i = 0; i < pointGrid.features.length; i++) {
    pointGrid.features[i].properties.temperature = Math.rand
}
var breaks = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

var lines = turf.isolines(pointGrid, breaks, {zProperty: 'te
```

planepoint

npm install @turf/planepoint

Takes a triangular plane as a Polygon and a Point within that triangle and returns the z-value at that point. The Polygon should have properties a , b ,

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

and c that define the values at its three corners. Alternatively, the z-values of each triangle point can be provided by their respective 3rd coordinate if their values are not provided as properties.

Arguments

Argument	Type	Description
point	Coord	the Point for which a z-value will be calculated
triangle	Feature <Polygon>	a Polygon feature with three vertices

Returns

number - the z-value for interpolatedPoint

Example

```
var point = turf.point([-75.3221, 39.529]);
// "a", "b", and "c" values represent the values of the coord
var triangle = turf.polygon([[
  [-75.1221, 39.57],
  [-75.58, 39.18],
  [-75.97, 39.86],
  [-75.1221, 39.57]
]], {
  "a": 11,
  "b": 122,
  "c": 44
});

var zValue = turf.planepoint(point, triangle);
point.properties.zValue = zValue;
```

tin

npm install @turf/tin

Takes a set of points and creates a Triangulated Irregular Network , or a TIN for short, returned as a collection of Polygons. These are often used for

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

developing elevation contour maps or stepped heat visualizations.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	input points
z	(String)	name of the property from which to pull z values This is optional: if not given, then there will be no extra data added to the derived triangles.

Returns

FeatureCollection <Polygon> - TIN output

Example

```
// generate some random point data
var points = turf.randomPoint(30, {bbox: [50, 30, 70, 50]});

// add a random property to each point between 0 and 9
for (var i = 0; i < points.features.length; i++) {
  points.features[i].properties.z = ~~(Math.random() * 9);
}
var tin = turf.tin(points, 'z');
```

pointsWithinPoly

npm install @turf/points-within-polygon

Finds Points that fall within (Multi)Polygon(s).

Arguments

Argument	Type	Description
points	(Feauture|FeatureCollection <Point>)	Points as input search

Argument	Type	Description
polygons	(FeatureCollection|Geometry|Feature <(Polygon|MultiPolygon)>)	Points must be within these (Multi)Polygon(s)

Returns

FeatureCollection <Point> - points that land within at least one polygon

Example

```
var points = turf.points([
    [-46.6318, -23.5523],
    [-46.6246, -23.5325],
    [-46.6062, -23.5513],
    [-46.663, -23.554],
    [-46.643, -23.557]
]);

var searchWithin = turf.polygon([[
    [-46.653,-23.543],
    [-46.634,-23.5346],
    [-46.613,-23.543],
    [-46.614,-23.559],
    [-46.631,-23.567],
    [-46.653,-23.560],
    [-46.653,-23.543]
]]);

var ptsWithin = turf.pointsWithinPolygon(points, searchWithi
```

tag

npm install @turf/tag

Takes a set of points and a set of polygons and performs a spatial join.

Arguments

Argument	Type	Description

Argument	Type	Description
points	FeatureCollection <Point>	input points
polygons	FeatureCollection <Polygon>	input polygons
field	string	property in polygons to add to joined { } features
outField	string	property in points in which to store joined property from polygons

Returns

FeatureCollection <Point> - points with containingPolyId property containing values from polyId

Example

```
var pt1 = turf.point([-77, 44]);
var pt2 = turf.point([-77, 38]);
var poly1 = turf.polygon([[
  [-81, 41],
  [-81, 47],
  [-72, 47],
  [-72, 41],
  [-81, 41]
]], {pop: 3000});
var poly2 = turf.polygon([[
  [-81, 35],
  [-81, 41],
  [-72, 41],
  [-72, 35],
  [-81, 35]
]], {pop: 1000});

var points = turf.featureCollection([pt1, pt2]);
var polygons = turf.featureCollection([poly1, poly2]);

var tagged = turf.tag(points, polygons, 'pop', 'population')
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

hexGrid

npm install @turf/hex-grid

Takes a bounding box and the diameter of the cell and returns a FeatureCollection of flat-topped hexagons or triangles (Polygon features) aligned in an "odd-q" vertical grid as described in Hexagonal Grids.

Arguments

Argument	Type	Description
bbox	BBox	extent in minX, minY, maxX, maxY order
cellSide	number	length of the side of the the hexagons or triangles, in units. It will also coincide with the radius of the circumcircle of the hexagons.
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cell size, can be degrees, radians, miles, or kilometers
properties	Object	{}	passed to each hexagon or triangle of the grid
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
triangles	boolean	false	whether to return as triangles instead of hexagons

Returns

FeatureCollection <Polygon> - a hexagonal grid

Example

```
var bbox = [-96,31,-84,40];
var cellSide = 50;
var options = {units: 'miles'};

var hexgrid = turf.hexGrid(bbox, cellSide, options);
```

pointGrid

npm install @turf/point-grid

Creates a Point grid from a bounding box, FeatureCollection or Feature.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order
cellSide	number	the distance between points, in units
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Point> - grid of points

Example

```
var extent = [-70.823364, -33.553984, -70.473175, -33.302986]
var cellSide = 3;
var options = {units: 'miles'};

var grid = turf.pointGrid(extent, cellSide, options);
```

squareGrid

npm install @turf/square-grid

Creates a square grid from a bounding box, Feature or FeatureCollection.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order

Argument	Type	Description
cellSide	number	of each cell, in units
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Polygon> - grid a grid of polygons

Example

```
var bbox = [-95, 30 ,-85, 40];
var cellSide = 50;
var options = {units: 'miles'};

var squareGrid = turf.squareGrid(bbox, cellSide, options);
```

triangleGrid

npm install @turf/triangle-grid

Takes a bounding box and a cell depth and returns a set of triangular polygons in a grid.

Arguments

Argument	Type	Description
bbox	Array	extent in minX, minY, maxX, maxY order
cellSide	number	dimension of each cell
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	'kilometers'	used in calculating cellSide, can be degrees, radians, miles, or kilometers
mask	(Feature <(Polygon|MultiPolygon)>)		if passed a Polygon or MultiPolygon, the grid Points will be created only inside it
properties	Object	{}	passed to each point of the grid

Returns

FeatureCollection <Polygon> - grid of polygons

Example

```
var bbox = [-95, 30 ,-85, 40];
var cellSide = 50;
var options = {units: 'miles'};

var triangleGrid = turf.triangleGrid(bbox, cellSide, options)
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

nearestPoint

npm install @turf/nearest-point

Takes a reference point and a FeatureCollection of Features with Point geometries and returns the point from the FeatureCollection closest to the reference. This calculation is geodesic.

Arguments

Argument	Type	Description
targetPoint	Coord	the reference point
points	FeatureCollection <Point>	against input point set

Returns

Feature <Point> - the closest point in the set to the reference point

Example

```
var targetPoint = turf.point([28.965797, 41.010086], {"marke
var points = turf.featureCollection([
    turf.point([28.973865, 41.011122]),
    turf.point([28.948459, 41.024204]),
    turf.point([28.938674, 41.013324])
]);

var nearest = turf.nearestPoint(targetPoint, points);
```

collect

npm install @turf/collect

Merges a specified property from a FeatureCollection of points into a FeatureCollection of polygons. Given an inProperty on points and an outProperty for polygons, this finds every point that lies within each polygon,

collects the inProperty values from those points, and adds them as an array to outProperty on the polygon.

Arguments

Argument	Type	Description
polygons	FeatureCollection <Polygon>	polygons with values on which to aggregate
points	FeatureCollection <Point>	points to be aggregated
inProperty	string	property to be nested from
outProperty	string	property to be nested into

Returns

FeatureCollection <Polygon> - polygons with properties listed based on outField

Example

```
var poly1 = turf.polygon([[[0,0],[10,0],[10,10],[0,10],[0,0]]
var poly2 = turf.polygon([[[10,0],[20,10],[20,20],[20,0],[10,
var polyFC = turf.featureCollection([poly1, poly2]);
var pt1 = turf.point([5,5], {population: 200});
var pt2 = turf.point([1,3], {population: 600});
var pt3 = turf.point([14,2], {population: 100});
var pt4 = turf.point([13,1], {population: 200});
var pt5 = turf.point([19,7], {population: 300});
var pointFC = turf.featureCollection([pt1, pt2, pt3, pt4, pt5
var collected = turf.collect(polyFC, pointFC, 'population',
var values = collected.features[0].properties.values
//=values => [200, 600]
```

clustersDbscan

npm install @turf/clusters-dbscan

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Takes a set of points and partition them into clusters according to https://en.wikipedia.org/wiki/DBSCAN data clustering algorithm.

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to be clustered
maxDistance	number	Maximum Distance between any point of the cluster to generate the clusters (kilometers only)
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
units	string	"kilometers"	in which
mutate	boolean	false	Allows GeoJSON input to be mutated
minPoints	number	3	Minimum number of points to generate a single cluster, points which do not meet this requirement will be classified as an 'edge' or 'noise'.

Returns

FeatureCollection <Point> - Clustered Points with an additional two properties associated to each Feature:

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
// create random points with random z-values in their properi
var points = turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
var maxDistance = 100;
var clustered = turf.clustersDbscan(points, maxDistance);
```

clustersKmeans

npm install @turf/clusters-kmeans

Takes a set of points and partition them into clusters using the k-mean. It uses the k-means algorithm

Arguments

Argument	Type	Description
points	FeatureCollection <Point>	to be clustered
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
numberOfClusters	number	Math.sqrt(numberOfPoints/2)	numberOfClusters that will be generated
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

FeatureCollection <Point> - Clustered Points with an additional two properties associated to each Feature:

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Example

```
// create random points with random z-values in their properi
var points = turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
var options = {numberOfClusters: 7};
var clustered = turf.clustersKmeans(points, options);
```

coordAll

npm install @turf/meta

Get all coordinates from any GeoJSON object.

Arguments

Argument	Type	Descr
geojson	(FeatureCollection|Feature|Geometry)	any GeoJ objec

Returns

Array > - coordinate position array

Note: coordAll is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordAll method.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {foo: 'bar'}),
  turf.point([36, 53], {hello: 'world'})
]);

var coords = turf.coordAll(features);
//= [[26, 37], [36, 53]]
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

coordEach

npm install @turf/meta

Iterate over coordinates in any GeoJSON object, similar to Array.forEach()

Note: coordEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordEach method.

Arguments

Argument	Type	
geojson	(FeatureCollection|Feature|Geometry)	
callback	Function	
excludeWrapCoord	boolean	whether or not to include the final coordinate of LinearRings that wraps the ring in its iteration.

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.coordEach(features, function (currentCoord, coordIndex,
  //=currentCoord
  //=coordIndex
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

coordReduce

npm install @turf/meta

Reduce coordinates in any GeoJSON object, similar to Array.reduce()

Note: coordReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the coordReduce method.

Arguments

Argument	Type	
geojson	(FeatureCollection|Geometry|Feature)	
callback	Function	
initialValue	(*)	argument to the first call of the callback.
excludeWrapCoord	boolean	whether or not to include the final coordinate of LinearRings that wraps the ring in its iteration.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.coordReduce(features, function (previousValue, currentC
  //=previousValue
  //=currentCoord
  //=coordIndex
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
  return currentCoord;
});
```

featureEach

npm install @turf/meta

Iterate over features in any GeoJSON object, similar to Array.forEach.

Note: featureEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the featureEach method.

Arguments

Argument	Type	Descr
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {foo: 'bar'}),
  turf.point([36, 53], {hello: 'world'})
]);

turf.featureEach(features, function (currentFeature, featureI
  //=currentFeature
  //=featureIndex
});
```

featureReduce

npm install @turf/meta

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Reduce features in any GeoJSON object, similar to Array.reduce().

Note: featureReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the featureReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curre featu
initialValue	(*)	Valu the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
  turf.point([26, 37], {"foo": "bar"}),
  turf.point([36, 53], {"hello": "world"})
]);

turf.featureReduce(features, function (previousValue, current
  //=previousValue
  //=currentFeature
  //=featureIndex
  return currentFeature
});
```

flattenEach

npm install @turf/meta

Iterate over flattened features in any GeoJSON object, similar to Array.forEach.

Note: flattenEach is part of the

Arguments

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu multil

@turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the flattenEach method.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.multiPoint([[40, 30], [36, 53]], {hello: 'world'})
]);

turf.flattenEach(features, function (currentFeature, featureI
  //=currentFeature
  //=featureIndex
  //=multiFeatureIndex
});
```

flattenReduce

npm install @turf/meta

Reduce flattened features in any GeoJSON object, similar to Array.reduce().

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curre featu mult

Note: flattenReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the flattenReduce method.

Argument	Type	Description
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.multiPoint([[40, 30], [36, 53]], {hello: 'world'})
]);

turf.flattenReduce(features, function (previousValue, current
  //=previousValue
  //=currentFeature
  //=featureIndex
  //=multiFeatureIndex
  return currentFeature
});
```

getCoord

npm install @turf/invariant

Unwrap a coordinate from a Point Feature, Geometry or a single coordinate.

Note: getCoord is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getCoord method.

Arguments

Argument	Type	Description
coord	(Array |Geometry <Point>|Feature <Point>)	GeoJSON Point or an Array of numbers

Returns

Array - coordinates

Example

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var pt = turf.point([10, 10]);

var coord = turf.getCoord(pt);
//= [10, 10]
```

getCoords

npm install @turf/invariant

Unwrap coordinates from a Feature, Geometry Object or an Array

Note: getCoords is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getCoords method.

Arguments

Argument	Type	Description
coords	(Array |Geometry|Feature)	Feature, Geometry Object or an Array

Returns

Array - coordinates

Example

```
var poly = turf.polygon([[[119.32, -8.7], [119.55, -8.69], [1

var coords = turf.getCoords(poly);
//= [[[119.32, -8.7], [119.55, -8.69], [119.51, -8.54], [119.
```

getGeom

npm install @turf/invariant

Get Geometry from Feature or Geometry Object

Note: getGeom is part of the

Argument	Type	Description
geojson	(Feature|Geometry)	GeoJSON Feature or Geometry Object

Returns

(Geometry|null) - GeoJSON Geometry Object

Throws

Error - if geojson is not a Feature or Geometry Object

Example

```
var point = {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [110, 40]
  }
}
var geom = turf.getGeom(point)
//={"type": "Point", "coordinates": [110, 40]}
```

@turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getGeom method.

getType

Get GeoJSON object's type, Geometry type is prioritize.

Arguments

Argument	Type	Description
geojson	GeoJSON	GeoJSON object
name	string	name of the variable to display in error message

Returns

string - GeoJSON type

Example

npm install @turf/invariant

Note: getType is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the getType method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var point = {
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [110, 40]
  }
}
var geom = turf.getType(point)
//="Point"
```

geomEach

npm install @turf/meta

Iterate over each geometry in any GeoJSON object, similar to Array.forEach()

Note: geomEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the geomEach method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G objec
callback	Function	a met takes (curre featu featu featu featu

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.geomEach(features, function (currentGeometry, featureIn
  //=currentGeometry
  //=featureIndex
```

```
  //=featureProperties
  //=featureBBox
  //=featureId
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

geomReduce

npm install @turf/meta

Reduce geometry in any GeoJSON object, similar to Array.reduce().

Note: geomReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the geomReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any obje
callback	Function	a me take (prev curr featı featı featı featı
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.geomReduce(features, function (previousValue, currentGe
  //=previousValue
  //=currentGeometry
```

```
  //=featureIndex
  //=featureProperties
  //=featureBBox
  //=featureId
  return currentGeometry
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

propEach

npm install @turf/meta

Iterate over properties in any GeoJSON object, similar to Array.forEach()

Note: propEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the propEach method.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature)	any GeoJSON object
callback	Function	a method that takes (currentProperti featureIndex)

Returns

undefined - undefined

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.propEach(features, function (currentProperties, feature
  //=currentProperties
  //=featureIndex
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

propReduce

npm install @turf/meta

Reduce properties in any GeoJSON object into a single value, similar to how Array.reduce works. However, in this case we lazily run the reduction, so an array of all properties is unnecessary.

Note: propReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the propReduce method.

Arguments

Argument	Type	Description
geojson	(FeatureCollection|Feature)	any GeoJSON object
callback	Function	a method that takes (previousValue currentPropert featureIndex)
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var features = turf.featureCollection([
    turf.point([26, 37], {foo: 'bar'}),
    turf.point([36, 53], {hello: 'world'})
]);

turf.propReduce(features, function (previousValue, currentPr
  //=previousValue
  //=currentProperties
  //=featureIndex
  return currentProperties
});
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

segmentEach

npm install @turf/meta

Iterate over 2-vertex line segment in any GeoJSON object, similar to Array.forEach() (Multi)Point geometries do not contain segments therefore they are ignored during this operation.

Note: segmentEach is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the segmentEach method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any G
callback	Function	a met takes (curre featu multil geom segmentIndex)

Returns

undefined - undefined

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10

// Iterate over GeoJSON by 2-vertex segments
turf.segmentEach(polygon, function (currentSegment, featureI
  //=currentSegment
  //=featureIndex
  //=multiFeatureIndex
  //=geometryIndex
  //=segmentIndex
});

// Calculate the total number of segments
var total = 0;
turf.segmentEach(polygon, function () {
    total++;
});
```

segmentReduce

npm install @turf/meta

Reduce 2-vertex line segment in any GeoJSON object, similar to Array.reduce() (Multi)Point geometries do not contain segments therefore they are ignored during this operation.

Note: segmentReduce is part of the @turf/meta module.

To use it as a stand-alone module will need to import @turf/meta and call the segmentReduce method.

Arguments

Argument	Type	Desc
geojson	(FeatureCollection|Feature|Geometry)	any
callback	Function	a me take (prev curre curre
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

undefined - undefined

Example

```
var polygon = turf.polygon([[[-50, 5], [-40, -10], [-50, -10]

// Iterate over GeoJSON by 2-vertex segments
turf.segmentReduce(polygon, function (previousSegment, currer
  //= previousSegment
  //= currentSegment
  //= featureIndex
  //= multiFeatureIndex
  //= geometryIndex
  //= segmentInex
  return currentSegment
});

// Calculate the total number of segments
var initialValue = 0
var total = turf.segmentReduce(polygon, function (previousVal
    previousValue++;
    return previousValue;
}, initialValue);
```

getCluster

npm install @turf/clusters

Get Cluster

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
filter	*	Filter used on GeoJSON properties to get Cluster

Note: getCluster is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the getCluster method.

Returns

FeatureCollection - Single Cluster filtered by GeoJSON Properties

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0], {'marker-symbol': 'circle'}),
    turf.point([2, 4], {'marker-symbol': 'star'}),
    turf.point([3, 6], {'marker-symbol': 'star'}),
    turf.point([5, 1], {'marker-symbol': 'square'}),
    turf.point([4, 2], {'marker-symbol': 'circle'})
]);

// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Retrieve first cluster (0)
var cluster = turf.getCluster(clustered, {cluster: 0});
//= cluster

// Retrieve cluster based on custom properties
turf.getCluster(clustered, {'marker-symbol': 'circle'}).leng
//= 2
turf.getCluster(clustered, {'marker-symbol': 'square'}).leng
//= 1
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

clusterEach

npm install @turf/clusters

clusterEach

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
property	(string|number)	GeoJSON property key/value used to create clusters
callback	Function	a method that takes (cluster, clusterValue, currentIndex)

Note: clusterEach is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the clusterEach method.

Returns

undefined - undefined

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0]),
    turf.point([2, 4]),
    turf.point([3, 6]),
    turf.point([5, 1]),
    turf.point([4, 2])
]);

// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Iterate over each cluster
turf.clusterEach(clustered, 'cluster', function (cluster, cl
    //= cluster
    //= clusterValue
    //= currentIndex
})

// Calculate the total number of clusters
var total = 0
turf.clusterEach(clustered, 'cluster', function () {
    total++;
});

// Create an Array of all the values retrieved from the 'clus
```

```
var values = []
turf.clusterEach(clustered, 'cluster', function (cluster, clu
    values.push(clusterValue);
});
```


clusterReduce

npm install @turf/clusters

Reduce clusters in GeoJSON Features, similar to Array.reduce()

Note: clusterReduce is part of the @turf/clusters module.

To use it as a stand-alone module will need to import @turf/clusters and call the clusterReduce method.

Arguments

Argument	Type	Description
geojson	FeatureCollection	GeoJSON Features
property	(string|number)	GeoJSON property key/value used to create clusters
callback	Function	a method that takes (previousValue, cluster, clusterValue, currentIndex)
initialValue	(*)	Value to use as the first argument to the first call of the callback.

Returns

* - The value that results from the reduction.

Example

```
var geojson = turf.featureCollection([
    turf.point([0, 0]),
    turf.point([2, 4]),
    turf.point([3, 6]),
    turf.point([5, 1]),
    turf.point([4, 2])
]);
```

```
// Create a cluster using K-Means (adds `cluster` to GeoJSON
var clustered = turf.clustersKmeans(geojson);

// Iterate over each cluster and perform a calculation
var initialValue = 0
turf.clusterReduce(clustered, 'cluster', function (previousVa
    //=previousValue
    //=cluster
    //=clusterValue
    //=currentIndex
    return previousValue++;
}, initialValue);

// Calculate the total number of clusters
var total = turf.clusterReduce(clustered, 'cluster', functio
    return previousValue++;
}, 0);

// Create an Array of all the values retrieved from the 'clus
var values = turf.clusterReduce(clustered, 'cluster', functi
    return previousValue.concat(clusterValue);
}, []);
```

collectionOf

npm install @turf/invariant

Enforce expectations about types of FeatureCollection inputs for Turf. Internally this uses geojsonType to judge geometry types.

Note: collectionOf is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the collectionOf method.

Arguments

Argument	Type	Description
featureCollection	FeatureCollection	a FeatureCollection for which features will be judged
type	string	expected GeoJSON type
name	string	name of calling function

TURF

Throws

Error - if value is not the expected type.

containsNumber

npm install @turf/invariant

Checks if coordinates contains a number

Arguments

Argument	Type	Description
coordinates	Array	GeoJSON Coordinates

Returns

boolean - true if Array contains a number

Note: containsNumber is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the containsNumber method.

geojsonType

npm install @turf/invariant

Enforce expectations about types of GeoJSON objects for Turf.

Arguments

Argument	Type	Description
value	GeoJSON	any GeoJSON object
type	string	expected GeoJSON type
name	string	name of calling function

Throws

Error - if value is not the expected type.

Note: geojsonType is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

geojsonType method.

featureOf

npm install @turf/invariant

Enforce expectations about types of Feature inputs for Turf. Internally this uses geojsonType to judge geometry types.

Note: featureOf is part of the @turf/invariant module.

To use it as a stand-alone module will need to import @turf/invariant and call the featureOf method.

Arguments

Argument	Type	Description
feature	Feature	a feature with an expected geometry type
type	string	expected GeoJSON type
name	string	name of calling function

Throws

Error - error if value is not the expected type.

booleanClockwis

npm install @turf/boolean-clockwise

Takes a ring and return true or false whether or not the ring is clockwise or counter-clockwise.

Arguments

Argument	Type	Description
line	Feature <LineString>	to be evaluated

Returns

boolean - true/false

Example

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

```
var clockwiseRing = turf.lineString([[0,0],[1,1],[1,0],[0,0]]
var counterClockwiseRing = turf.lineString([[0,0],[1,0],[1,1]

turf.booleanClockwise(clockwiseRing)
//=true
turf.booleanClockwise(counterClockwiseRing)
//=false
```

booleanContains

npm install @turf/boolean-contains

Boolean-contains returns True if the second geometry is completely contained by the first geometry. The interiors of both geometries must intersect and, the interior and boundary of the secondary (geometry b) must not intersect the exterior of the primary (geometry a). Boolean-contains returns the exact opposite result of the @turf/boolean-within.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);
var point = turf.point([1, 2]);

turf.booleanContains(line, point);
//=true
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

booleanCrosses

npm install @turf/boolean-crosses

Boolean-Crosses returns True if the intersection results in a geometry whose dimension is one less than the maximum dimension of the two source geometries and the intersection set is interior to both source geometries.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line1 = turf.lineString([[-2, 2], [4, 2]]);
var line2 = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]

var cross = turf.booleanCrosses(line1, line2);
//=true
```

booleanDisjoint

npm install @turf/boolean-disjoint

Boolean-disjoint returns (TRUE) if the intersection of the two geometries is an empty set.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry

Argument	Type	Description
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var point = turf.point([2, 2]);
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);

turf.booleanDisjoint(line, point);
//=true
```

booleanEqual

npm install @turf/boolean-equal

Determine whether two geometries of the same type have identical X,Y coordinate values. See http://edndoc.esri.com/arcsde/9.0/general_topics/understand_spatial_relations.htm

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON input
feature2	(Geometry|Feature)	GeoJSON input

Returns

boolean - true if the objects are equal, false otherwise

Example

```
var pt1 = turf.point([0, 0]);
var pt2 = turf.point([0, 0]);
var pt3 = turf.point([1, 1]);

turf.booleanEqual(pt1, pt2);
//= true
```

```
turf.booleanEqual(pt2, pt3);
//= false
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
- along
- area
- bbox
- bboxPolygon
- bearing
- center
- centerOfMass
- centroid
- destination
- distance
- envelope
- length
- midpoint
- pointOnFeature
- polygonTangents
- pointToLineDista
- rhumbBearing
- rhumbDestinatio
- rhumbDistance
- square
- greatCircle

COORDINATE MUTATION
- cleanCoords
- flip
- rewind
- round
- truncate

TRANSFORMATIO
- bboxClip
- bezierSpline
- buffer
- circle
- clone
- concave
- convex
- difference
- dissolve
- intersect
- lineOffset

booleanOverlap

npm install @turf/boolean-overlap

Compares two geometries of the same dimension and returns true if their intersection set results in a geometry different from both but of the same dimension. It applies to Polygon/Polygon, LineString/LineString, Multipoint/Multipoint, MultiLineString/MultiLineString and MultiPolygon/MultiPolygon.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	input
feature2	(Geometry|Feature <(LineString|MultiLineString|Polygon|MultiPolygon)>)	input

Returns

boolean - true/false

Example

```
var poly1 = turf.polygon([[[0,0],[0,5],[5,5],[5,0],[0,0]]]);
var poly2 = turf.polygon([[[1,1],[1,6],[6,6],[6,1],[1,1]]]);
var poly3 = turf.polygon([[[10,10],[10,15],[15,15],[15,10],[1

turf.booleanOverlap(poly1, poly2)
//=true
turf.booleanOverlap(poly2, poly3)
//=false
```

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

booleanParallel

npm install @turf/boolean-parallel

Boolean-Parallel returns True if each segment of line1 is parallel to the correspondent segment of line2

Arguments

Argument	Type	Description
line1	(Geometry|Feature <LineString>)	GeoJSON Feature or Geometry
line2	(Geometry|Feature <LineString>)	GeoJSON Feature or Geometry

Returns

boolean - true/false if the lines are parallel

Example

```
var line1 = turf.lineString([[0, 0], [0, 1]]);
var line2 = turf.lineString([[1, 0], [1, 1]]);

turf.booleanParallel(line1, line2);
//=true
```

booleanPointInP

npm install @turf/boolean-point-in-polygon

Takes a Point and a Polygon or MultiPolygon and determines if the point resides inside the polygon. The polygon can be convex or concave. The function accounts for holes.

Arguments

Argument	Type	Description
point	Coord	input point
polygon	Feature <(Polygon|MultiPolygon)>	input polygon or multipolygon

Argument	Type	Description
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
ignoreBoundary	boolean	false	True if polygon boundary should be ignored when determining if the point is inside the polygon otherwise false.

Returns

boolean - true if the Point is inside the Polygon; false if the Point is not inside the Polygon

Example

```
var pt = turf.point([-77, 44]);
var poly = turf.polygon([[
  [-81, 41],
  [-81, 47],
  [-72, 47],
  [-72, 41],
  [-81, 41]
]]);

turf.booleanPointInPolygon(pt, poly);
//= true
```

booleanPointOn

npm install @turf/boolean-point-on-line

Returns true if a point is on a line. Accepts a optional parameter to ignore the start and end

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

vertices of the linestring.

Arguments

Argument	Type	Description
pt	Coord	GeoJSON Point
line	Feature <LineString>	GeoJSON LineString
options	Object	Optional parameters: see below

Options

Prop	Type	Default	Description
ignoreEndVertices	boolean	false	whether to ignore the start and end vertices.

Returns

boolean - true/false

Example

```
var pt = turf.point([0, 0]);
var line = turf.lineString([[-1, -1],[1, 1],[1.5, 2.2]]);
var isPointOnLine = turf.booleanPointOnLine(pt, line);
//=true
```

booleanWithin

npm install @turf/boolean-within

Boolean-within returns true if the first geometry is completely within the second geometry. The interiors of both geometries must intersect and, the interior and boundary of the primary (geometry a) must not intersect the exterior of the secondary (geometry b). Boolean-within returns the exact opposite result of the @turf/boolean-contains.

Arguments

Argument	Type	Description
feature1	(Geometry|Feature)	GeoJSON Feature or Geometry
feature2	(Geometry|Feature)	GeoJSON Feature or Geometry

Returns

boolean - true/false

Example

```
var line = turf.lineString([[1, 1], [1, 2], [1, 3], [1, 4]]);
var point = turf.point([1, 2]);

turf.booleanWithin(point, line);
//=true
```

bearingToAzimut

npm install @turf/helpers

Converts any bearing angle from the north line direction (positive clockwise) and returns an angle between 0-360 degrees (positive clockwise), 0 being the north line

Arguments

Argument	Type	Description
bearing	number	angle, between -180 and +180 degrees

Returns

number - angle between 0 and 360 degrees

Note: bearingToAzimuth is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the bearingToAzimuth method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

convertArea

npm install @turf/helpers

Converts a area to the requested unit. Valid units: kilometers, kilometres, meters, metres, centimetres, millimeters, acres, miles, yards, feet, inches

Note: convertArea is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the convertArea method.

Arguments

Argument	Type	Description
area	number	to be converted
originalUnit	string	of the distance
finalUnit	string	returned unit

Returns

number - the converted distance

convertLength

npm install @turf/helpers

Converts a length to the requested unit. Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Note: convertLength is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the convertLength method.

Arguments

Argument	Type	Description
length	number	to be converted
originalUnit	string	of the length
finalUnit	string	returned unit

Returns

number - the converted length

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

degreesToRadiar

npm install @turf/helpers

Converts an angle in degrees to radians

Arguments

Argument	Type	Description
degrees	number	angle between 0 and 360 degrees

Returns

number - angle in radians

Note: degreesToRadians is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the degreesToRadians method.

lengthToRadians

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from a real-world unit into radians Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Arguments

Argument	Type	Description
distance	number	in real units
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - radians

Note: lengthToRadians is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the lengthToRadians method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

lengthToDegrees

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from a real-world unit into degrees Valid units: miles, nauticalmiles, inches, yards, meters, metres, centimeters, kilometres, feet

Note: lengthToDegrees is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the lengthToDegrees method.

Arguments

Argument	Type	Description
distance	number	in real units
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - degrees

radiansToLength

npm install @turf/helpers

Convert a distance measurement (assuming a spherical Earth) from radians to a more friendly unit. Valid units: miles, nauticalmiles, inches, yards, meters, metres, kilometers, centimeters, feet

Note: radiansToLength is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the radiansToLength method.

Arguments

Argument	Type	Description
radians	number	in radians across the sphere
units	string	can be degrees, radians, miles, or kilometers inches, yards, metres, meters, kilometres, kilometers.

Returns

number - distance

radiansToDegree

npm install @turf/helpers

Converts an angle in radians to degrees

Arguments

Argument	Type	Description
radians	number	angle in radians

Returns

number - degrees between 0 and 360 degrees

Note: radiansToDegrees is part of the @turf/helpers module.

To use it as a stand-alone module will need to import @turf/helpers and call the radiansToDegrees method.

toMercator

npm install @turf/projection

Converts a WGS84 GeoJSON object into Mercator (EPSG:900913) projection

Arguments

Argument	Type	Description
geojson	(GeoJSON|Position)	WGS84 GeoJSON object
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description

Note: toMercator is part of the @turf/projection module.

To use it as a stand-alone module will need to import @turf/projection and call the toMercator method.

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - true/false

Example

```
var pt = turf.point([-71,41]);
var converted = turf.toMercator(pt);
```

toWgs84

npm install @turf/projection

Converts a Mercator (EPSG:900913) GeoJSON object into WGS84 projection

Note: toWgs84 is part of the @turf/projection module.

To use it as a stand-alone module will need to import @turf/projection and call the toWgs84 method.

Arguments

Argument	Type	Description
geojson	(GeoJSON|Position)	Mercator GeoJSON object
options	(Object)	Optional parameters: see below

Options

Prop	Type	Default	Description

TURF

GETTING STARTED

Search mod

MEASUREMENT
along
area
bbox
bboxPolygon
bearing
center
centerOfMass
centroid
destination
distance
envelope
length
midpoint
pointOnFeature
polygonTangents
pointToLineDista
rhumbBearing
rhumbDestinatio
rhumbDistance
square
greatCircle

COORDINATE MUTATION
cleanCoords
flip
rewind
round
truncate

TRANSFORMATIO
bboxClip
bezierSpline
buffer
circle
clone
concave
convex
difference
dissolve
intersect
lineOffset

Prop	Type	Default	Description
mutate	boolean	false	allows GeoJSON input to be mutated (significant performance increase if true)

Returns

GeoJSON - true/false

Example

```
var pt = turf.point([-7903683.846322424, 5012341.663847514]);
var converted = turf.toWgs84(pt);
```