## What is Django?

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

#### Django helps you write software that is:

* Complete
* Versatile
* Secure
* Scalable
* Maintainable
* Portable

**Where did it come from?**

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework which was open-sourced as the Django project in July 2005

[**How popular is Django?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#how_popular_is_django)

There isn't any readily-available and definitive measurement of popularity of server-side frameworks (although you can estimate popularity using mechanisms like counting the number of GitHub projects and Stack Over flow questions for each platform). A better question is whether Django is "popular enough" to avoid the problems of unpopular platforms. Is it continuing to evolve? Can you get help if you need it? Is there an opportunity for you to get paid work if you learn Django?

## SO AGAIN What is the Django Framework?

Django is an open-source framework for backend web applications based on Python one of the top web development languages. Its main goals are simplicity, flexibility, reliability, and scalability.

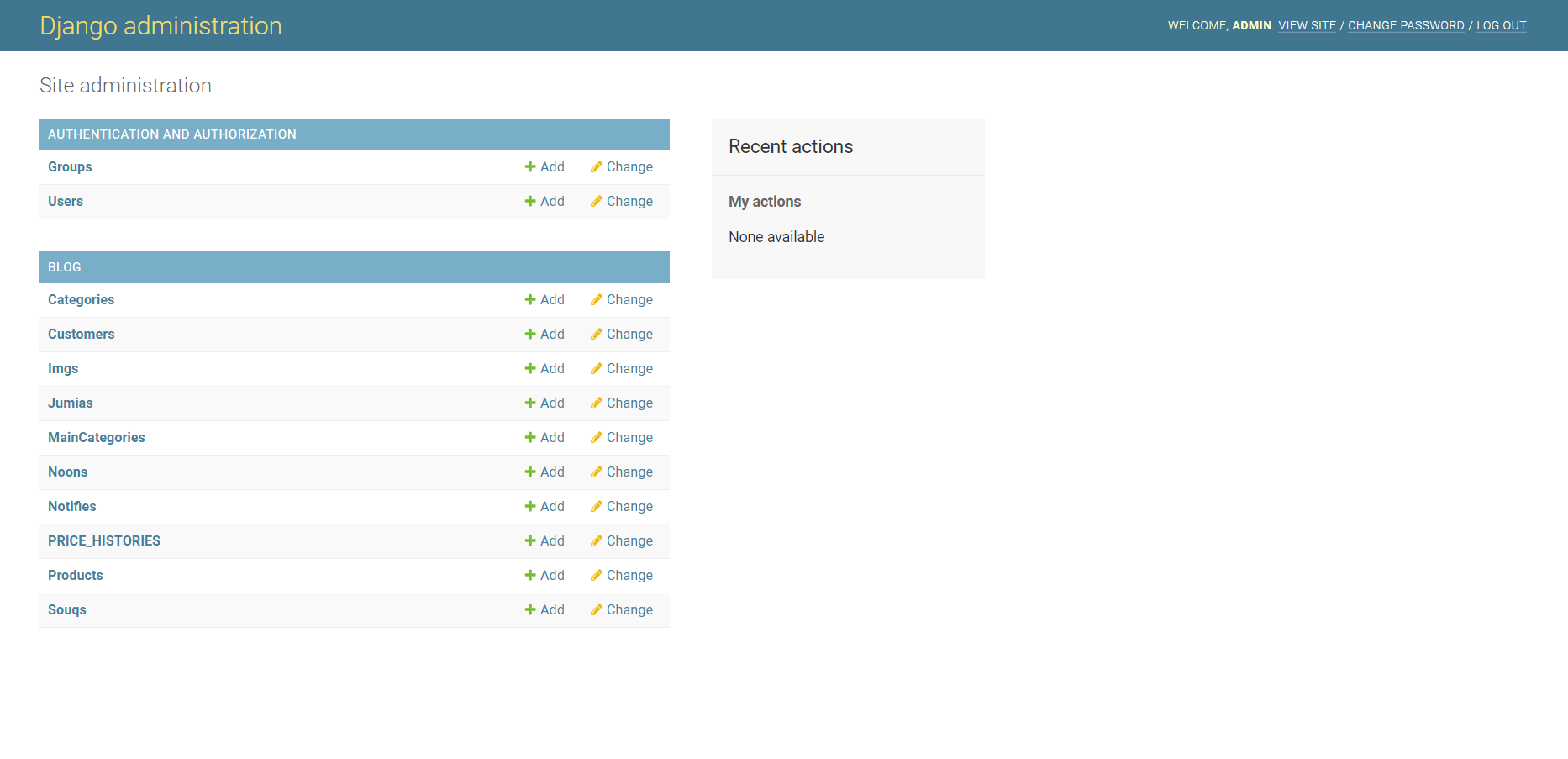
Django has its own naming system for all functions and components e.g., HTTP responses are called “views”. It also has an admin panel, which is deemed easier to work with than in Lavarel or Yii and other technical features, including:

* Simple syntax
* Its own web server
* MVC (Model-View-Controller) core architecture
* Batteries included (comes with all the essentials needed to solve solving common cases)
* An ORM (Object Relational Mapper)
* HTTP libraries
* Middleware support
* A Python unit test framework

## Why do you need a framework?

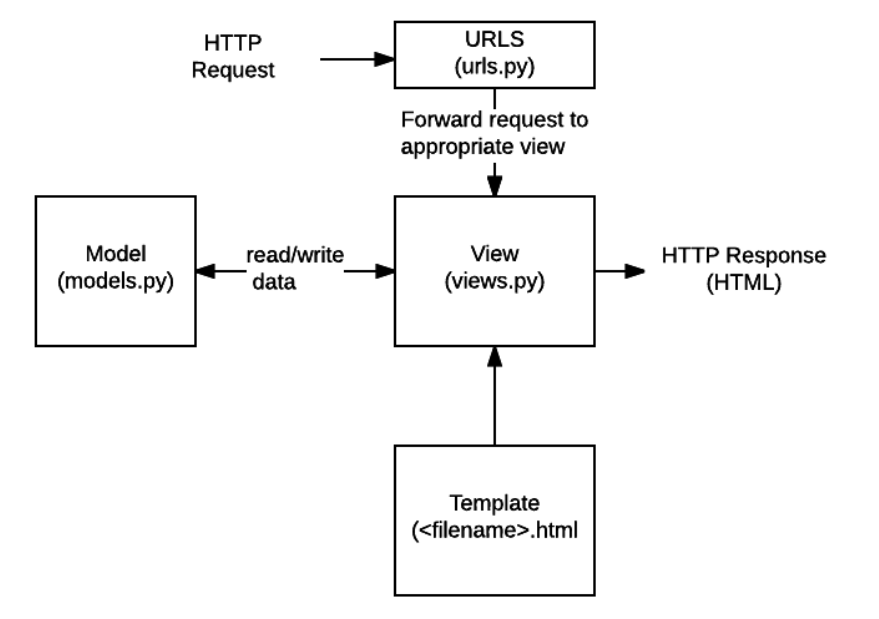
To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that you want it to serve you a web page.

Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server reads the letter and then sends a response with a webpage. But when you want to send something, you need to have some content. And Django is something that helps you create the content.



## [What does Django code look like?](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_does_django_code_look_like)

In a traditional data-driven website, a web application waits for HTTP requests from the web browser or other client. When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.



## 

* **URLs**: While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
* **View**: A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.
* **Models**: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
* **Templates**: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

## What happens when someone requests a website from your server?

When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested. It takes a web page address first and tries to figure out what to do. This part is done by Django's **url resolver** (note that a website address is called a URL – Uniform Resource Locator – so the name *url resolver* makes sense). It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function which is called *view*.

In the *view* function all the interesting things are done we can look at a database to look for some information.

So instead of diving too much into details, we will start creating something with Django and we will learn all the important parts along the way

[**What else can you do?**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#what_else_can_you_do)

* **Forms**: HTML Forms are used to collect user data for processing on the server. Django simplifies form creation, validation, and processing.
* **User authentication and permissions**: Django includes a robust user authentication and permission system that has been built with security in mind.
* **Caching**: Creating content dynamically is much more computationally intensive (and slow) than serving static content. Django provides flexible caching so that you can store all or part of a rendered page so that it doesn't get re-rendered except when necessary.
* **Administration site:** The Django administration site is included by default when you create an app using the basic skeleton. It makes it trivially easy to provide an admin page for site administrators to create, edit, and view any data models in your site.
* **Serialising data:** Django makes it easy to serialise and serve your data as XML or JSON. This can be useful when creating a web service or when creating a website in which the client-side code handles all the rendering of data.

Leading market research firm, Gartner, suggests that [40% of all business initiatives lose value](https://www.data.com/export/sites/data/common/assets/pdf/DS_Gartner.pdf) because of incorrectly linked, or messy data. [Data deduplication](https://dataladder.com/data-deduplication-software/) and record linkage are two sides of the same coin. While the applications of both vary widely, the underlying techniques used to identify matching records for both [data cleansing](https://dataladder.com/data-cleansing-software/)/deduplication and record linkage are the same.

Whether you want to identify duplicates before migrating to a new CRM, or want to build a [Single Customer View](https://econsultancy.com/what-is-the-single-customer-view-and-why-do-you-need-it/) under an enterprise-wide digital transformation initiative, you will have to perform ‘data matching’: the ability to identify all records that point to the same entity within and across data sources. Easier said than done though.

In this blog, we will take an in-depth look at fuzzy matching, the go-to approach for data deduplication and record linkage. We will cover:

* Data Matching: Deterministic and Probabilistic Matching
* What is Fuzzy Matching?
* How to Minimize False Positives and Negatives
* Why Do Businesses Need Fuzzy Matching?
* Fuzzy Matching Techniques
* Example of Real-World Fuzzy Matching Scenario

**Data Matching: Deterministic and Probabilistic Matching**

We know we need to match records to identify duplicates and link records for [entity resolution](https://dataladder.com/entity-resolution-software/). But how exactly do we go about identifying matching records? What properties should we focus on?

Let’s start with ‘unique identifiers’. These are properties in the records you want to match that are unlikely to change over time, Customer Name for instance. You can assign weights to each property to improve your matching process. Think about it; if you are migrating customer data from one system to another and need to check for duplicates pre- and post-migration, you could, for instance, choose Name as the one unique identifier and phone number as the second. Now it’s just a matter of running a search for matching Customer IDs and phone numbers and you have all potential matches identified. That method is known as ‘deterministic data matching’.

Although effective in theory, the method is rarely used because of its inflexibility: The approach assumes that all entries are free of mistakes and standardized across systems – which is almost never the case in real-world linkage scenarios. In our previous example, if some phone numbers have country code in the ‘+1’ format and the rest start with ‘001’, the matching would go awry. That’s just one instance; there could potentially be dozens of different ways data could be entered. The names might be misspelled, acronyms used, middle name included, etc. In one system, a customer’s name could be ‘William Warner’ while another might have ‘Williaam Warner’ — it’s obvious that there’s been a small typo and both are in fact the same customer — but the method only allows discrete outcomes, that is, all or nothing.

How do you go about determining a match when so many variations exist?

By performing probabilistic data matching, that’s how. More commonly known as fuzzy matching’, this approach permits the user to account for variations like spelling errors, nicknames, punctuation differences, and many more by combining a variety of algorithms.

## What is fuzzy matching?

Rather than flagging records as a ‘match’ or ‘non-match’, fuzzy matching identifies the likelihood that two records are a true match based on whether they agree or disagree on the various identifiers.

The identifiers or parameters you choose here and the weight you assign forms the basis of fuzzy matching. If the parameters are too broad, you will find more matches, true, but you will also invariably increase the chances of ‘false positives’. These are pairs that are identified by your algorithm or [fuzzy matching software](https://dataladder.com/fuzzy-matching-software/) of choice as a match, but upon manual review, you will find that your approach identified a false positive.

Consider the strings “Kent” and “10th”. While there is clearly no match here, popular fuzzy matching algorithms still rate these two strings nearly 50% similar, based as character count and phonetic match. [Check for yourself](https://asecuritysite.com/forensics/simstring).

False positives are one of the biggest issues with fuzzy matching. The more efficient the system you’re using, the fewer the false positives. An efficient system will identify:

* Acronyms
* name reversal
* name variations
* phonetic spellings
* deliberate misspellings
* inadvertent misspellings
* abbreviations e.g. ‘Ltd’ instead of ‘Limited’
* insertion/removal of punctuation, spaces, special characters
* different spelling of names e.g. ‘Elisabeth’ or ‘Elizabeth’, ‘Jon’ instead of ‘John’
* shortened names e.g. ‘Elizabeth’ matches with ‘Betty’, ‘Beth’, ‘Elisa’, ‘Elsa’, ‘Beth’ etc. And many other variations.

**How to Minimize False Positives and Negatives**

We have discussed false positives in the previous section briefly. While they make matching more difficult by adding manual review time to the process, they’re not a genuine risk to the business because the system will flag false positives based on the overall match score. Let’s take a look at ‘false negatives’ now. This refers to matches that are missed altogether by the system: not just a low match score, but an absence of match score. This leads to a serious risk for the business as false negatives are never reviewed because no one knows they exist. Factors that commonly lead to false negatives include:

* Lack of relevant data
* Significant errors in data entry
* System limitations
* Match criterion is too narrow
* Inappropriate level of fuzzy matching

The most effective method to minimize both false positives and negatives is to profile and clean the data sources separately before you conduct matching. Leading [data matching solution](https://dataladder.com/data-matching-software/) providers typically bundle a data profiler that quickly provides enough metadata to construct a cogent profile analysis of data quality, as in missing values, lack of standardization, any other discrepancies in your data. By [profiling your data](https://dataladder.com/data-profiling/), you can quickly quantify the scope and depth of the primary project, whether it’s Master Data Management,  matching, cleansing, deduplication, or standardization.

Once you’ve profiled your data, you will know exactly which business rules to apply to clean and standardize your data most efficiently. You will also be able to quickly recognize and fill missing values, perhaps by purchasing 3rd party data.

Cleaner, more complete data reduces false positives and negatives significantly by increasing match accuracy because your data is now standardized. The fuzzy matching algorithms you use, the matching criteria you define, the weight you assign to different parameters, the way you combine different algorithms and assign priority – these are all important factors in minimizing false positives and negatives too. But none of these are going to help much if you haven’t profiled and cleaned your data first. See how DataMatch Enterprise has helped 4,000+ customers in over 40 countries clean, deduplicate, and link their data efficiently.

**Why Do Businesses Need Fuzzy Matching**

Research reveals that 94% of businesses admit to having duplicate data, and the majority of these duplicates are non-exact matches and therefore usually remain undetected. Fuzzy matching software helps you make those connections automatically using sophisticated proprietary matching logic, regardless of spelling errors, unstandardized data, or incomplete information.

But it’s not just about deduplication. From a strategic perspective, fuzzy matching comes into play when you’re conducting record linkage or entity resolution. We touched upon this briefly in the previous section too; the fuzzy matching approach is invaluable when creating a Single Source of Truth for business analytics or building a foundation for Master Data Management (MDM), helping organizations integrate data from dozens of different sources across the enterprise while ensuring accuracy and minimizing manual review. See how [a major healthcare provider](https://dataladder.com/resources/case-studies/st-john-associates/) was able to save hundreds of man-hours annually.

Here are some ways that fuzzy matching is used to improve the bottom-line:

* Realize a Single Customer View
* Work with Clean Data You Can Trust
* Prepare Data for Business Intelligence
* Enhance the Accuracy of Your Data for Operational Efficiency
* Enrich Data for Deeper Insights
* Ensure Better Compliance
* Refine Customer Segmentation
* Improve Fraud Prevention

## Fuzzy Matching Techniques

Now you know what fuzzy matching is and the many different ways you can use it to grow your business. Question is, how do you about implementing fuzzy matching processes in your organization?

Here’s a list of the various fuzzy matching techniques that are in use today:

* Levenshtein Distance (or Edit Distance)
* Damerau-Levenshtein Distance
* Jaro-Winkler Distance
* Keyboard Distance
* Kullback-Leibler Distance
* Jaccard Index
* Metaphone 3
* Name Variant
* Syllable Alignment
* Acronym

**Example of a Real-World Fuzzy Matching Scenario**

The following example shows how record linkage techniques can be used to detect fraud, waste or abuse of federal government programs. Here, two databases were merged to get information not previously available from a single database.

A database consisting of records on 40,000 airplane pilots licensed by the U.S. Federal Aviation Administration (FAA) and residing in Northern California was matched to a database consisting of individuals receiving disability payments from the Social Security Administration. Forty pilots whose records turned up on both databases were arrested.

A prosecutor in the U.S. Attorney’s Office in Fresno, California stated, according to an AP report:

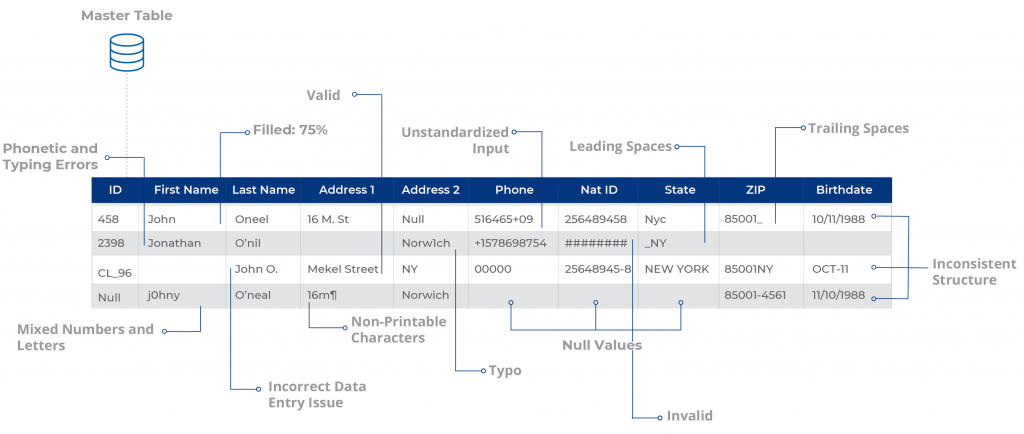
“There was probably criminal wrongdoing.” The pilots were either lying to the FAA or wrongfully receiving benefits. The pilots claimed to be medically fit to fly airplanes. However, they may have been flying with debilitating illnesses that should have kept them grounded, ranging from schizophrenia and bipolar disorder to drug and alcohol addiction and heart conditions.”

At least twelve of these individuals “had commercial or airline transport licenses,” the report stated. The FAA revoked 14 pilots’ licenses. The other pilots were found to be lying about having illnesses in order to collect Social Security payments.

The quality of the linkage of the files was highly dependent on the quality of the names and addresses of the licensed pilots within both of the files being linked. The detection of the fraud was also dependent on the completeness and accuracy of the information in a particular Social Security Administration database.

## Fuzzy Matching Made Easy, Fast, and Laser-Focused on Driving Business Value

Traditionally, fuzzy matching has been considered a complex, arcane art, where project costs are typically in the hundreds of thousands of dollars, taking months, if not years, to deliver tangible ROI, and even then, security, scalability, and accuracy concerns remain. That is no longer the case with modern data quality software. Based on decades of research and 4,000+ deployments across more than 40 countries, [DataMatch Enterprise](https://dataladder.com/products/datamatch-enterprise/" \t "_blank) is a highly visual data cleansing application specifically designed to resolve data quality issues. The platform leverages multiple proprietary and standard algorithms to identify phonetic, fuzzy, miskeyed, abbreviated, and domain-specific variations.



Build scalable configurations for deduplication & [record linkage](https://dataladder.com/record-linkage-software/), suppression, enhancement, extraction, and [standardization](https://dataladder.com/data-standardization-software/) of business and customer data and create a Single Source of Truth to maximize the impact of your data across the enterprise.

How to Use It Correctly

As we have just defined **Fuzzy matching** allows you to identify non-exact matches of your target item. It is the foundation stone of many search engine frameworks and one of the main reasons why you can get relevant search results even if you have a typo in your query or a different verbal tense.

As you might expect, there are many algorithms that can be used for fuzzy searching on text, but virtually all search engine frameworks (including bleve) use primarily the Levenshtein Distance for fuzzy string matching:

#### Levenshtein Distance

Also known as **Edit Distance**, it is the number of transformations (deletions, insertions, or substitutions) required to transform a source string into the target one. For example, if the target term is “book” and the source is “back”, you will need to change the first “o” to “a” and the second “o” to “c”, which will give us a Levenshtein Distance of 2.Edit Distance is very easy to implement, and it is a popular challenge during code interviews (You can find Levenshtein implementations in JavaScript, Kotlin, Java, and many others here).

Additionally, some frameworks also support the Damerau-Levenshtein distance:

#### Damerau-Levenshtein distance

It is an extension to Levenshtein Distance, allowing one extra operation: Transposition of two adjacent characters:

**Ex**: TSAR to STAR

**Damerau-Levenshtein distance** = 1 (Switching S and T positions cost only one operation)

**Levenshtein distance** = 2 (Replace S by T and T by S)

**Fuzzy matching and relevance**

Fuzzy matching has one big side effect; it messes up with relevance. Although Damerau-Levenshtein is an algorithm that considers most of the common user’s misspellings, it also can include a significant number of false positives, especially when we are using a language with an average of just 5 letters per word, such as English. That is why most of the search engine frameworks prefer to stick with Levenshtein distance.