Chapter 2 : Specifications of the study

- fuzzy matching
- Fuzzy Matching Techniques
- Web Scraping
- Requests
- Python and Django

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 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.
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 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, 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 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

Data Matching: Deterministic and Probabilistic Matching

We know we need to match records to identify duplicates and link records for entity resolution. 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.

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:

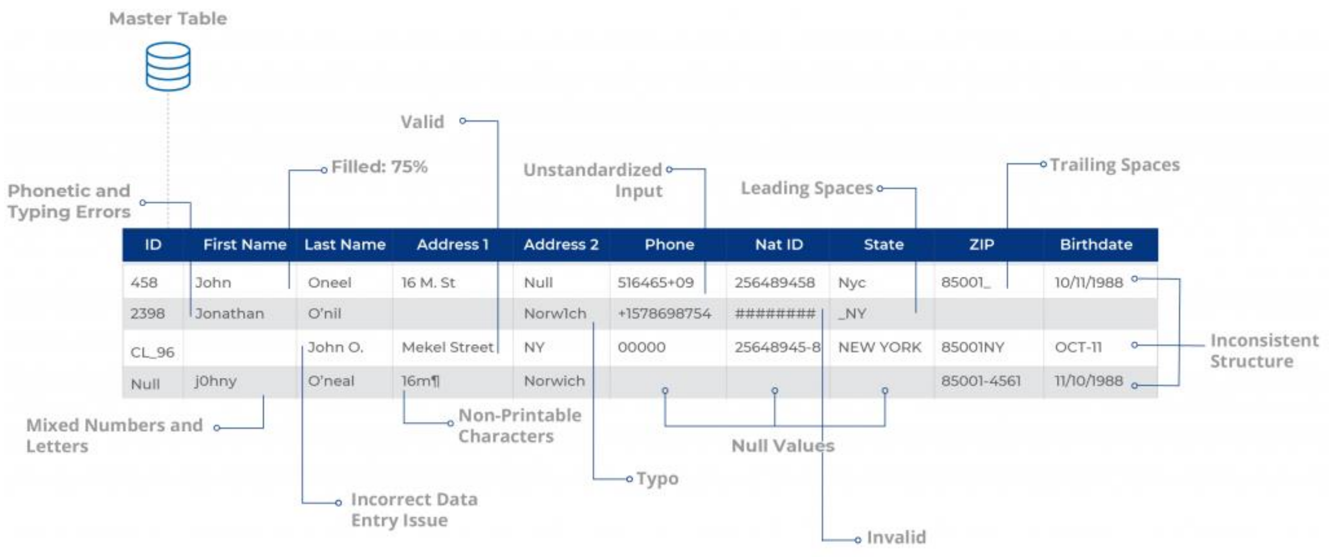
"There was probably criminal wrongdoing." The pilots were either lying to the FAA or wrongfully receiving benefits. The pilots claimed to be medically fit to fly airplanes. However, they may have been flying with debilitating illnesses that should have kept them grounded, ranging from schizophrenia and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



ID	First Name	Last Name	Address 1	Address 2	Phone	Nat ID	State	ZIP	Birthdate
458	John	Oneel	16 M. St	Null	516465+09	256489458	Nyc	85001_	10/11/1988
2398	Jonathan	O'nil		Norw1ch	+1578698754	########	_NY		
CL_96		John O.	Mekel Street	NY	00000	25648945-8	NEW YORK	85001NY	OCT-11
Null	j0hny	O'neal	16m¶	Norwich				85001-4561	11/10/1988

Build scalable configurations for deduplication & record linkage, suppression, enhancement, extraction, and standardization 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.

WHAT IS WEB SCRAPING?

Web scraping is the process of gathering information from the Internet. Even copy-pasting the lyrics of your favorite song is a form of web scraping! However, the words “web scraping” usually refer to a process that involves automation. Some websites don’t like it when automatic scrapers gather their data, while others don’t mind.

If you’re scraping a page respectfully for educational purposes, then you’re unlikely to have any problems. Still, it’s a good idea to do some research on your own and make sure that you’re not violating any Terms of Service before you start a large-scale project. To learn more about the legal aspects of web scraping, check out Legal Perspectives on Scraping Data From The Modern Web.

Why Scrape the Web?

Say you're a surfer (both online and in real life) and you're looking for employment. However, you're not looking for just *any* job. With a surfer's mindset, you're waiting for the perfect opportunity to roll your way!

There's a job site that you like that offers exactly the kinds of jobs you're looking for. Unfortunately, a new position only pops up once in a blue moon. You think about checking up on it every day, but that doesn't sound like the most fun and productive way to spend your time.

Thankfully, the world offers other ways to apply that surfer's mindset! Instead of looking at the job site every day, you can use Python to help automate the repetitive parts of your job search. **Automated web scraping** can be a solution to speed up the data collection process. You write your code once and it will get the information you want many times and from many pages.

In contrast, when you try to get the information you want manually, you might spend a lot of time clicking, scrolling, and searching. This is especially true if you need large amounts of data from websites that are regularly updated with new content. Manual web scraping can take a lot of time and repetition.

There's so much information on the Web, and new information is constantly added. Something among all that data is likely of interest to you, and much of it is just out there for the taking. Whether you're actually on the job hunt, gathering data to support your grassroots organization, or are finally looking to get all the lyrics from your favorite artist downloaded to your computer, automated web scraping can help you accomplish your goals.

Challenges of Web Scraping

The Web has grown organically out of many sources. It combines a ton of different technologies, styles, and personalities, and it continues to grow to this day. In other words, the Web is kind of a hot mess! This can lead to a few challenges you'll see when you try web scraping.One challenge is **variety**. Every website is different. While you'll encounter general structures that tend to repeat themselves, each website is unique and will need its own personal treatment if you want to extract the information that's relevant to you. Another challenge is **durability**. Websites constantly change. Say you've built a shiny new web scraper that automatically cherry-picks precisely what you want from your resource of interest. The first time you run your script, it works flawlessly. But when you run the same script only a short while later, you run into a discouraging and lengthy stack of tracebacks!This is a realistic scenario, as many websites are in active development. Once the site's structure has changed, your scraper might not be able to navigate the sitemap correctly or find the relevant information. The good news is that many changes to websites are small and incremental, so you'll likely be able to update your scraper with only minimal adjustments.However, keep in mind that because the internet is dynamic, the scrapers you'll build will probably require constant maintenance. You can set up continuous integration to run scraping tests periodically to ensure that your main script doesn't break without your knowledge.

APIs: An Alternative to Web Scraping

Some website providers offer **Application Programming Interfaces (APIs)** that allow you to access their data in a predefined manner. With APIs, you can avoid parsing HTML and instead access the data directly using formats like JSON and XML. HTML is primarily a way to visually present content to users.When you use an API, the process is generally more stable than gathering the data through web scraping. That's because APIs are made to be consumed by programs, rather than by human eyes. If the design of a website changes, then it doesn't mean that the structure of the API has changed.to APIs just as they do to websites. Additionally, it's much harder to inspect the structure of an API by yourself if the provided documentation is lacking in quality.The approach and tools you need to gather information using APIs are outside the scope of this tutorial. To learn more about it, check out API Integration in Python.

Requests - Overview

Requests is a HTTP library that provides easy functionality to deal with http request/response in your web application. The library is developed in python.

Features of Requests

Request: The python requests library has easy to use methods available to handle Http request. Passing of parameters and handling the request type like GET, POST, PUT, DELETE, etc. is very easy.

Response: You can get the response in the format you need and the supported ones are text format, binary response, json response, and raw response.

Headers: The library allows you to read, update or send new headers as per your requirements.

Timeouts: Timeouts can be easily added to the URL you are requesting using python requests library. It so happens that you are using a third-party URL and waiting for a response.It is always a good practice to give a timeout on the URL as we might want the URL to respond within that timeout with a response or an error that is coming because of timeout. Not doing so can cause either to wait on that request indefinitely.

Error handling: The requests module gives support for error handling and some of which are Connection Error, Timeout errors, TooManyRedirects, Response.raise_for_status errors, etc.

Cookies: The library allows you to read, write and update for the requested URL.

Sessions: To maintain the data, you require between requests you need sessions. So, if the same host is called again and again you can re-use the TCP connection which in turn will improve the performance.

SSL certificates: SSL certificate is a security feature that comes with secure urls. When you use Requests, it also verifies SSL certificates for the https URL given. SSL Verification is enabled by default in the requests library and will throw an error if the certificate is not present.

Authentication: HTTP authentication is on the server-side asking for some authentication information like username, password when the client requests a URL. This is an additional security for the request and the response being exchanged between the client and the server.

Advantages of using Python Requests Library

Following are the advantages of using Python Requests Library −

- Easy to use and fetch the data from the URL given.
- Requests library can be used to scrape the data from the website.
- Using requests, you can get, post, delete, update the data for the URL given.
- The handling of cookies and session is very easy.
- The security is also taken care of the help of authentication module support.

Requests - Environment Setup

we will work on the installation of Requests. To start working with the Requests module, we need to install Python first.

- Install Python
- Install Requests

Requests - How Http Requests Work?

Python’s Requests is a HTTP library that will help us exchange data between the client and the server. Consider you have a UI with a form, wherein you need to enter the user details, so once you enter it, you have to submit the data which is nothing but a Http POST or PUT request from the client to server to save the data.

When you want the data, you need to fetch it from the server, which is again a Http GET request. The exchange of data between the client when it requests the data and the server responding with the required data, this relationship between the client and the server is very important.

The request is made to the URL given and it could be a secure or non-secure URL.

The request to the URL can be done using GET, POST, PUT, DELETE. The most commonly used is the GET method, mainly used when you want to fetch data from the server.

Using the request library, the URL is called as follows using a string dictionary.

Wherein the data to the URL is sent as a dictionary of strings. If you want to pass id=9 and username=Delphine, you can do as follows−

payload = {'id': '9', 'username': 'Delphine'}

The requests library is called as follows−

res = requests.get('https://www.noon.com/egypt-en/', params=payload')

Using POST

res = requests.post('https://www.jumia.com.eg/', data = {'id':'9', 'username':'Delphine'})

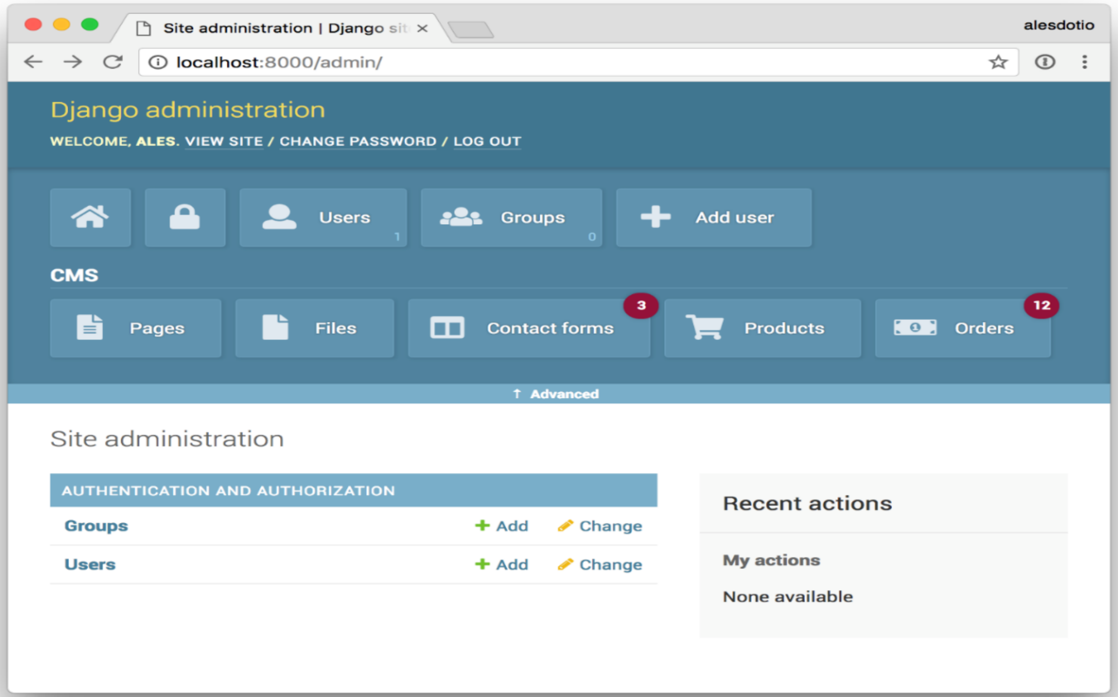
Using PUT

res = requests.put('https://egypt.souq.com/eg-ar/', data = {'id':'9', 'username':'Delphine'})

The response from the Http request can be in text encoded form, binary encoded, json format or raw response. The details of the request and response are explained in detail in the next chapters.

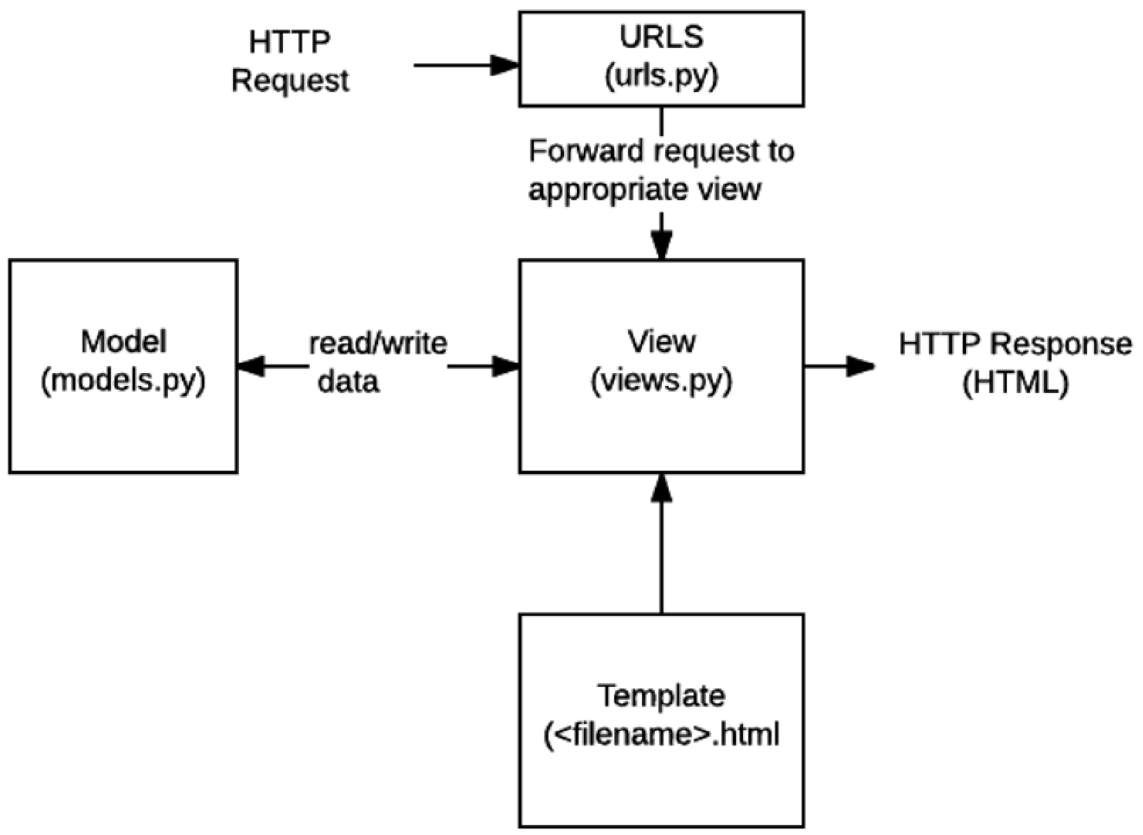
WHY DO YOU NEED A FRAMEWORK?

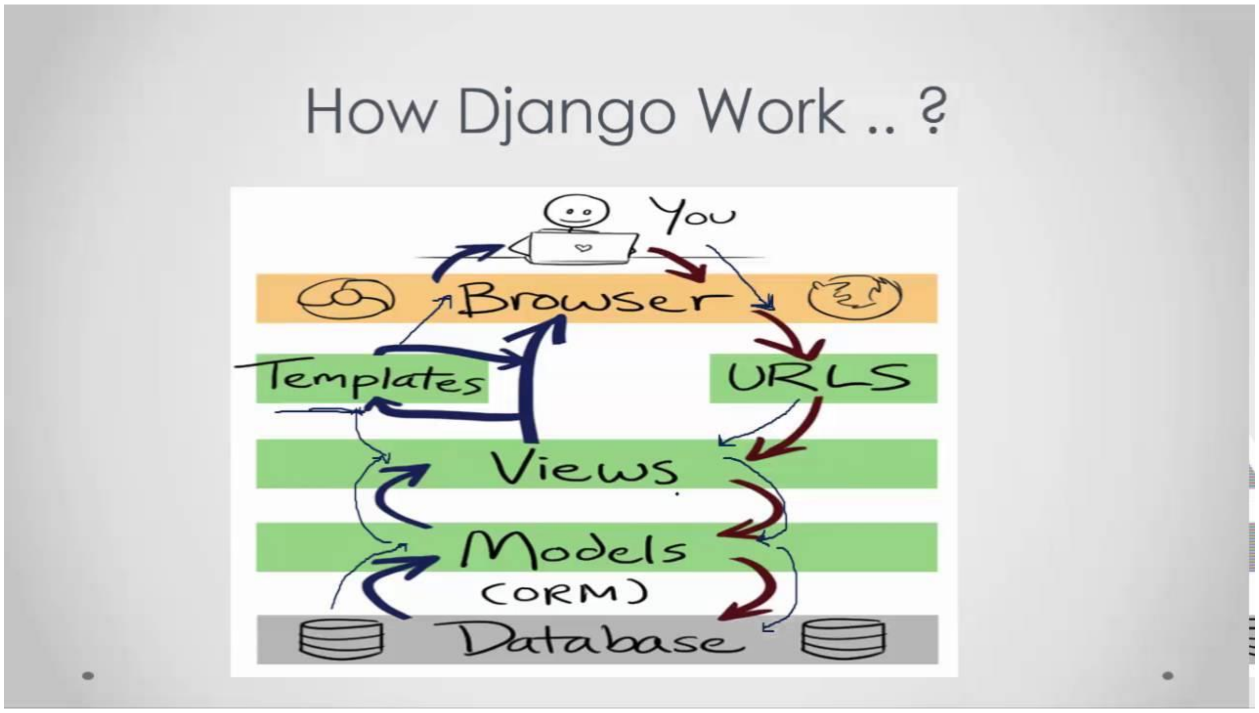
To understand what Django is actually for, we need to take a closer look at the servers. The first thing is that the server needs to know that you want it to serve you a web page.
Imagine a mailbox (port) which is monitored for incoming letters (requests). This is done by a web server. The web server 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?

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.


HTTP Request
URLS (urls.py)
Forward request to appropriate view
Model (models.py)
read/write data
View (views.py)
HTTP Response (HTML)
Template (<filename>.html



- **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?

- **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 because of incorrectly linked, or messy data. Data deduplication 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/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 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

GLOSSARY

Brick-and-mortar players: Retailers with a physical presence/location that may have e-commerce capabilities but that do not meet all the requirements to be omnichannel

CAGR: Compound annual growth rate

E-commerce players: Retailers with an online presence, including e-commerce pure players, brick and-mortar players, and some e-commerce presence and omnichannel players

E-commerce pure players: Retailers that operate only on the Internet (online) and have no physical presence/location

GCC: Gulf Cooperation Council, consisting of six nations—the Kingdom of Saudi Arabia, the United Arab Emirates, Kuwait, Qatar, Bahrain and Oman

Omnichannel players: Retailers that provide a seamless integration of physical and digital worlds to deliver an outstanding customer experience