4.5 - Implement advanced search features in Azure AI Search

Overview

Use more advanced features of Azure AI Search to improve your existing search solutions. Learn how to change the ranking on documents, boost terms, and allow searching in multiple languages.

Learning objectives

In this module, you'll learn how to:

- Improve the ranking of a document with term boosting
- Improve the relevance of results by adding scoring profiles
- Improve an index with analyzers and tokenized terms
- Enhance an index to include multiple languages
- Improve search experience by ordering results by distance from a given reference point

---

Introduction

Azure AI Search is a powerful search service that can index a wide range of data from various sources. A core part of search is returning relevant results from search queries.

By the end of this module, you'll learn to:

- Improve the ranking of a document with term boosting
- Improve the relevance of results by adding scoring profiles
- Improve an index with analyzers and tokenized terms
- Enhance an index to include multiple languages
- Improve search experience by ordering results by distance from a given reference point

This module builds on Create a custom skill for Azure AI Search to explore more advanced search features supported by Azure AI Search.

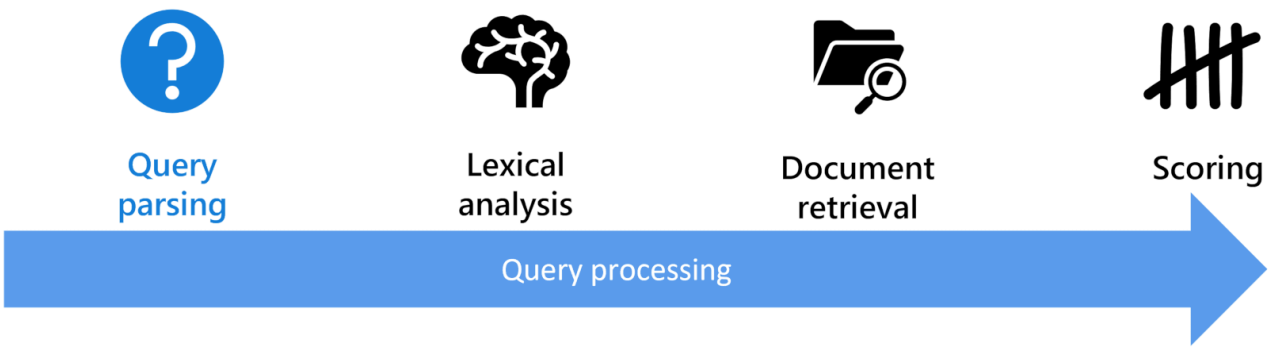
Improve the ranking of a document with term boosting

Search works best when the most relevant results are shown first. All search engines try to return the most relevant results to search queries. Azure AI Search implements an **enhanced version of Apache Lucene for full text search.**

Here, you'll explore **how to write more complex Lucene queries**. You'll then improve the relevance of results by boosting specific terms in your search query.

Search an index

Azure AI Search lets you query an index using a REST endpoint or inside the Azure portal with the search explorer tool. If you want a quick recap of the stages of query processing, see the *search index* unit in Create an Azure AI Search solution.



In this unit, you'll be focusing on query parsing.

You'll use the search explorer to see the difference between using the simple and full query type changes your search results.

> Note: If you want to run the queries yourself you'll need an Azure subscription. Create an Azure AI Search service and import the hotels' sample data into an index.

Write a simple query

The hotel sample data contains 50 hotels with descriptions, room details, and their locations. Imagine you run a hotel booking business and have an app that users can book hotels with. Users can search and the most relevant hotels must be shown first.

You have a use case where a customer is trying to find a luxury hotel. Start by looking at the search results from this simple query:
`search=luxury&$select=HotelId, HotelName, Category, Tags, Description&$count=true`

The query parses will search for the term `luxury` across all the fields for a document in the index.

The query string also **limits the returned fields** from documents by adding the `select` option.
`&$select=HotelId, HotelName, Category, Tags, Description`

The last parameter on the query asks the index to count the total results.
`$count=true`

There's no lexical analysis needed so document retrieval returns 14 documents. The first three are:


```
{
  "@odata.context": "https://advanced-cognitive-
search.search.windows.net/indexes('hotels-sample-index')/$metadata#docs(*)",
  "@odata.count": 14,
  "value": [
    {
```

```
      "@search.score": 2.633778,
      "HotelId": "13",
      "HotelName": "Historic Lion Resort",
      "Description": "Unmatched Luxury.  Visit our downtown hotel to indulge in luxury
accommodations. Moments from the stadium, we feature the best in comfort",
      "Category": "Budget",
      "Tags": [
        "view",
        "free wifi",
        "free wifi"
      ]
    },
    {
      "@search.score": 2.1104424,
      "HotelId": "18",
      "HotelName": "Oceanside Resort",
      "Description": "New Luxury Hotel.  Be the first to stay. Bay views from every
room, location near the piper, rooftop pool, waterfront dining & more.",
      "Category": "Budget",
      "Tags": [
        "view",
        "laundry service",
        "air conditioning"
      ]
    },
    {
      "@search.score": 1.966516,
      "HotelId": "40",
      "HotelName": "Trails End Motel",
      "Description": "Only 8 miles from Downtown.  On-site bar/restaurant, Free hot
breakfast buffet, Free wireless internet, All non-smoking hotel. Only 15 miles from
airport.",
      "Category": "Luxury",
      "Tags": [
        "continental breakfast",
        "view",
        "view"
      ]
    },
    ...
  ]
}
```


The customer might be surprised that the top hotel you have that's supposed to be **luxury** is in the budget category and doesn't have any air conditioning. If the customer enters multiple words in their search, your app assumes all terms should be in the results, so it adds + between terms to the query. This query it sends to the API is:

```
search=luxury + air con&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

The search service now returns five documents but still the top results are in the budget category.

Enable the Lucene Query Parser

You can tell the search explorer to use the Lucene Query parser by adding `&queryType=full` to the query string.
`search=luxury AND air con&$select=HotelId, HotelName, Category, Tags, Description&$count=true&queryType=full`

With the Lucene syntax, you can write more precise queries. Here is a summary of available features:

- **Boolean operators**: `AND`, `OR`, `NOT` for example `luxury AND 'air con'`
- **Fielded search**: `fieldName:search term` for example `Description:luxury AND Tags:air con`
- **Fuzzy search**: `~` for example `Description:luxury~` returns results with misspelled versions of luxury
- **Term proximity search**: `"term1 term2"~n` for example `"indoor swimming pool"~3` returns documents with the words indoor swimming pool **within three words of each other**
- **Regular expression search**: `/regular expression/` use a regular expression between `/` for example `/[mh]otel/` would return documents with hotel and motel
- **Wildcard search**: `*`, `?` where `*` will match many characters and `?` matches a single character for example `'air con'*` would find air con and air conditioning
- **Precedence grouping**: `(term AND (term OR term))` for example `(Description:luxury OR Category:luxury) AND Tags:air?con*`
- **Term boosting**: `^` for example `Description:luxury OR Category:luxury^3` would give hotels **with the category luxury a higher score than luxury in the description** (and the number 3 refers to the weight)

To read more detail about these features, see Lucene query syntax in Azure AI Search in the docs.

Boost search terms

Using the above you can improve the results. The parser should give a higher priority to hotels in the luxury category. You can also be more precise and look for air conditioning in the Tags field.
`(Description:luxury OR Category:luxury^3) AND Tags:'air con'*`

Adding the other query string parameters you get this query string:
`search=(Description:luxury OR Category:luxury^3) AND Tags:'air con'*&$select=HotelId, HotelName, Category, Tags, Description&$count=true&queryType=full`

The top three hotels are now:


```
{
  "@odata.context": "https://advanced-cognitive-search.search.windows.net/indexes('hotels-sample-index')/$metadata#docs(*)",
  "@odata.count": 5,
  "value": [
    {
      "@search.score": 5.3537707,
      "HotelId": "8",
      "HotelName": "Sapphire Resort",
      "Description": "Downtown, close to everything, steps to the park, shopping, and restaurants.",
      "Category": "Luxury",
      "Tags": [
        "free wifi",
        "continental breakfast",
```

```
        "air conditioning"
      ]
    },
    {
      "@search.score": 5.3522806,
      "HotelId": "49",
      "HotelName": "Old Carrabelle Hotel",
      "Description": "Spacious rooms, glamorous suites and residences, rooftop pool,
walking access to shopping, dining, entertainment and the city center.",
      "Category": "Luxury",
      "Tags": [
        "air conditioning",
        "laundry service",
        "24-hour front desk service"
      ]
    },
    {
      "@search.score": 4.1448884,
      "HotelId": "18",
      "HotelName": "Oceanside Resort",
      "Description": "New Luxury Hotel.  Be the first to stay. Bay views from every
room, location near the piper, rooftop pool, waterfront dining & more.",
      "Category": "Budget",
      "Tags": [
        "view",
        "laundry service",
        "air conditioning"
      ]
    },
    ...
  ]
}
```


The Sapphire Resorts search score has increased from **2.3321536** to **5.3537707** and is now the first hotel the customer will see. The Oceanside Resort is now in third place.

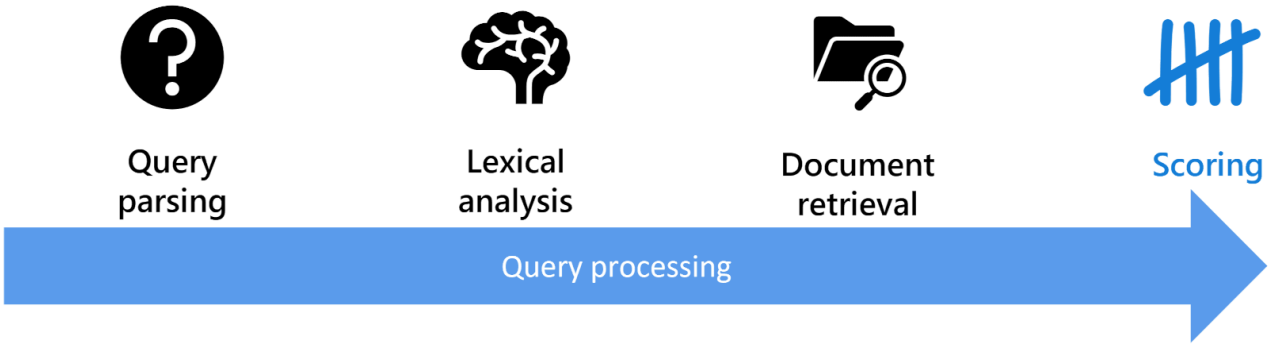
Improve the relevance of results by adding scoring profiles

Azure AI Search uses the **BM25 similarity ranking algorithm**. The algorithm scores documents based on the search terms used.

Here, you'll see how to add scoring profiles to **alter the scores for documents based on your own criteria.**

How search scores are calculated

Scoring is the last phase of processing a search query.



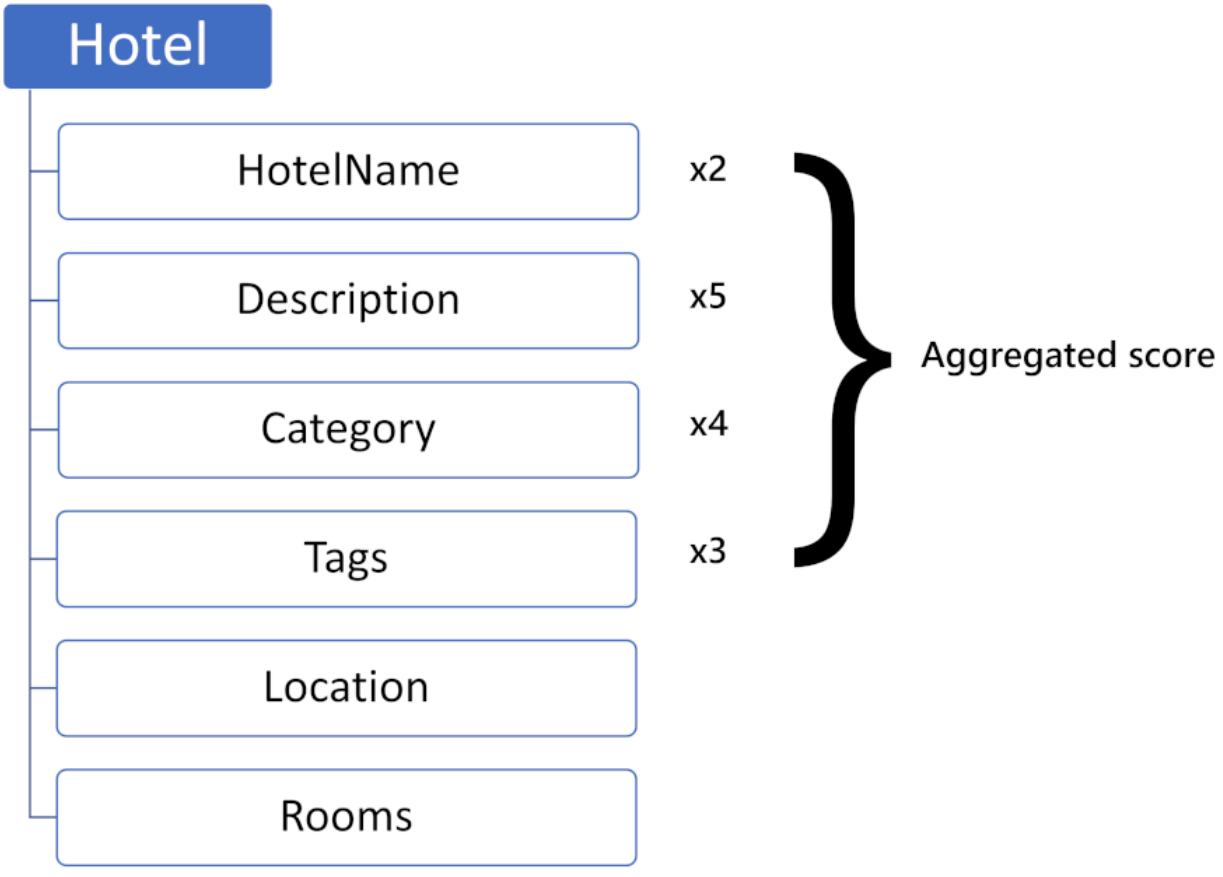
The search engine scores the documents returned from the first three phases. **The score is a function of the number of times identified search terms appear in a document, the document's size, and the rarity of each of the terms.**

> Supplement information from ChatGPT: **BM25** improves on TF-IDF by incorporating term frequency saturation and document length normalization (so shorter documents aren't unfairly favored), making it more effective for practical retrieval tasks.

By default, the search results are ordered by their search score, highest first. If two documents have an identical search score, you can break the tie by adding an `$orderby` clause.

Improve the score for more relevant documents

As the **default scoring works on the frequency of terms and rarity**, the final calculated score might not return the highest score for the most relevant document. Each dataset is different, so AI Search lets you influence a document score using scoring profiles.



The most straightforward scoring profile defines different weights for fields in an index. In the above example, the Hotel index has a scoring profile that has the Description field five times more relevant than data in the Location or Rooms fields. The Category field is twice as relevant as the HotelName.

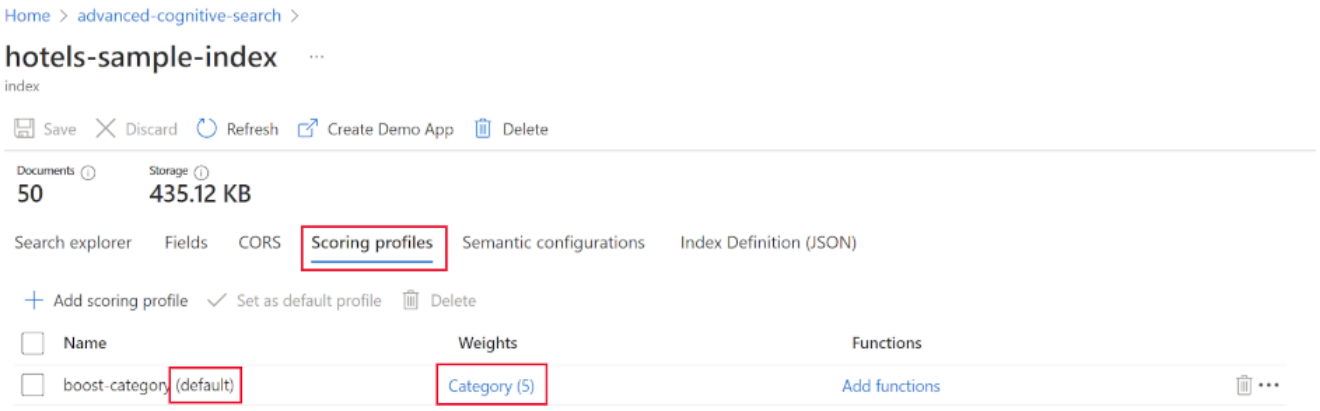
The scoring profile can also include **functions**, for example, distance or freshness. Functions provide more control than simple weighting, for example, you can define the boosting duration applied to newer documents before they score the same as older documents.

The power of scoring profiles means that instead of boosting a specific term in a search request, you can apply a scoring profile to an index so that fields are boosted automatically for all queries.

Add a weighted scoring profile

You can add up to 100 scoring profiles to a search index. The simplest way to create a scoring profile is in the Azure portal.

1. Navigate to your search service.
2. Select **Indexes**, then select the index to add a scoring profile to.
3. Select **Scoring profiles**.
4. Select **+ Add scoring profile**.
5. In **Profile name**, enter a unique name.
6. To set the scoring profile as a default to be applied to all searches select **Set as default profile**.
7. In **Field name**, select a field. Then for **Weight**, enter a weight value.
8. Select **Save**.



In the above example, the `boost-category` scoring profile has been added to the `hotels-sample-index`. The Category has a weight of five.

The profile has also been set as the default profile. You can then use this search query:
`search=luxury AND Tags:'air con'*&$select=HotelId, HotelName, Category, Tags, Description&$count=true&queryType=full`

The results now match the same query with a term boosted:
`search=(Description:luxury OR Category:luxury^5) AND Tags:'air con'*&$select=HotelId, HotelName, Category, Tags, Description&$count=true&queryType=full`

You can control which scoring profile is applied to a search query by appending the `&scoringProfile=PROFILE NAME` parameter.

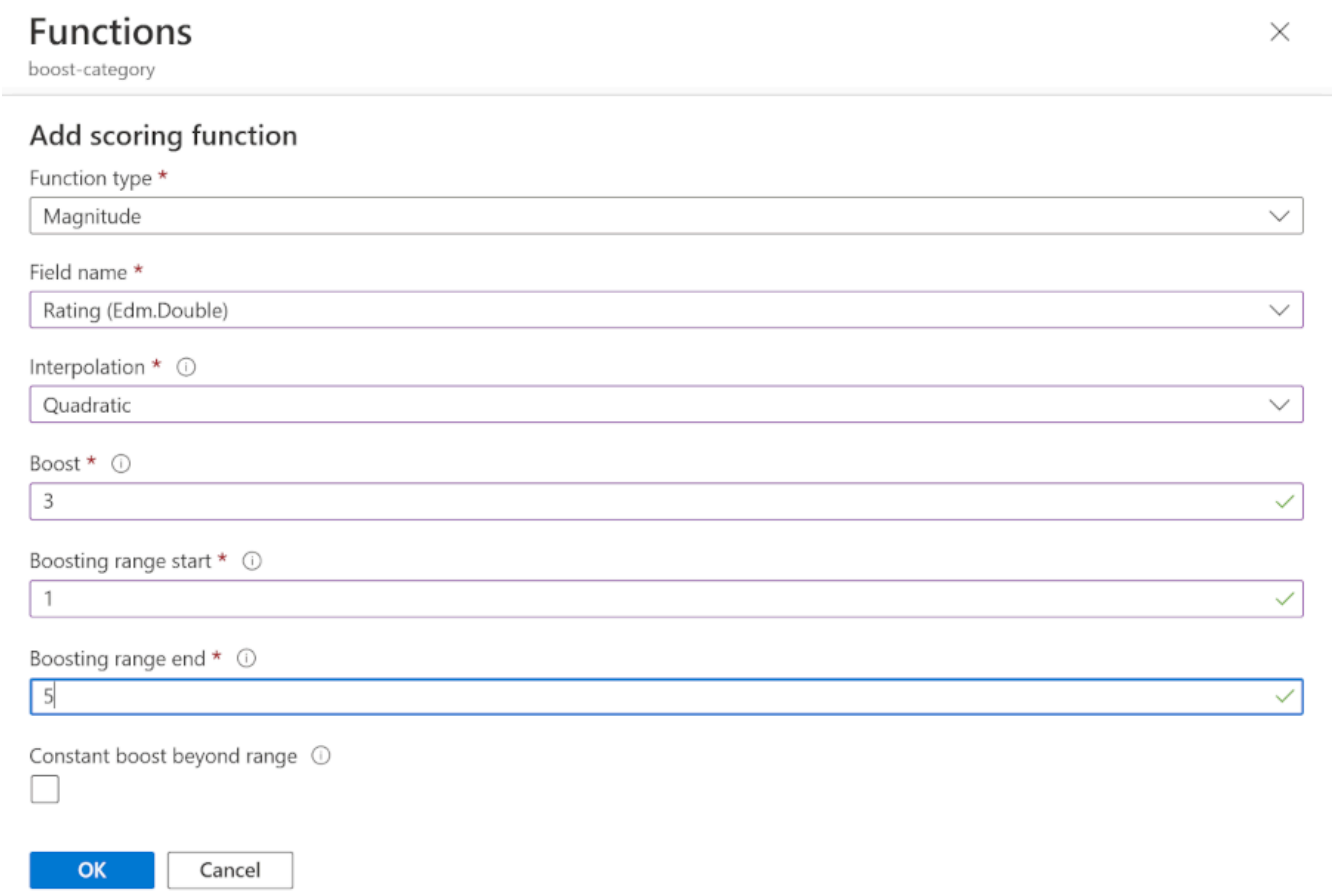
Scoring profiles can also be added programmatically using the Update Index REST API or in Azure SDKs, such as the ScoringProfile class in the Azure SDK for .NET.

Use functions in a scoring profile

The functions available to add to a scoring profile are:

Function	Description
Magnitude	Alter scores based on a range of values for a numeric field
Freshness	Alter scores based on the freshness of documents as given by a DateTimeOffset field
Distance	Alter scores based on the distance between a reference location and a GeographyPoint field
Tag	Alter scores based on common tag values in documents and queries

For example, using the hotel index the **magnitude function** can be applied to the Rating field. The Azure portal will guide you through completing the parameters for each function.



Improve an index with analyzers and tokenized terms

Azure AI Search is configured by default to analyze text and **identify tokens** that will be helpful in your index. The right tokens ensure that users can find the documents they need quickly. In most cases, the default configuration produces an optimal index. However, when you have unusual or unique fields, you might want to configure exactly how text is analyzed.

Here, you'll learn how to define a custom analyzer to control how the content of a field is split into tokens for inclusion in the index.

Analyzers in AI Search

When AI Search indexes your content, it retrieves text. To build a useful index, with terms that help users locate documents, that text needs processing. For example:

- The text should be broken into words, often by using whitespace and punctuation characters as delimiters.
- Stopwords, such as "the" and "it", should be removed because users don't search for them.
- Words should be reduced to their root form. For example, past tense words, such as "ran", should be replaced with present tense words, such as "run".

In AI Search, **this kind of processing is performed by analyzers**. If you don't specify an analyzer for a field, the default Lucene analyzer is used.

The default Lucene analyzer is a good choice for most fields because it can process many languages and return useful tokens for your index.

Alternatively, you can specify one of the analyzers that are built into AI Search. Built-in analyzers are of two types:

- **Language analyzers.** If you need advanced capabilities for specific languages, such as lemmatization, word decompounding, and entity recognition, use a built-in language analyzer. Microsoft provides 50 analyzers for different languages.
- **Specialized analyzers.** These analyzers are language-agnostic and used for specialized fields such as zip codes or product IDs. You can, for example, use the **PatternAnalyzer** and specify a regular expression to match token separators.

What is a custom analyzer?

The built-in analyzers provide you with many options but sometimes you need an analyzer with unusual behavior for a field. In these cases, you can create a **custom analyzer**.

A custom analyzer consists of:

- **Character filters**. These filters process a string before it reaches the tokenizer.
- **Tokenizers**. These components divide the text into tokens to be added to the index.
- **Token filters.** These filters remove or modify the tokens emitted by the tokenizer.

Let's examine these components in more detail.

Character filters

Some operations might need to be completed on the text before it's split into tokens. Character filters enable these operations. There are three character filters that you can use:

- **html_strip.** This filter removes HTML constructs such as tags and attributes.
- **mapping.** This filter enables you to specify mappings that replace one string with another. For example, you could specify a mapping that replaces *TX* with *Texas*.
- **pattern_replace.** This filter enables you to specify a regular expression that identifies patterns in the input text and how matching text should be replaced.

Tokenizers

The tokenizer is the component that divides the text into the tokens that will be stored in the index. Tokenizers also break down words into their root forms. Often, a token is a single word, but you might want to create unusual tokens such as:

- A full postal address.
- A complete URL or email address.
- Words based on the grammar of a specific language.

There are 13 different tokenizers to choose from. These tokenizers include:

- **classic.** This tokenizer processes text based on grammar for European languages.
- **keyword.** This tokenizer emits the entire input as a single token. Use this tokenizer for fields that should always be indexed as one value.
- **lowercase.** This tokenizer divides text at non-letters and then modifies the resulting tokens to all lower case.
- **microsoft_language_tokenizer.** This tokenizer divides text based on the grammar of the language you specify.
- **pattern.** This tokenizer divides texts where it matches a regular expression that you specify.
- **whitespace.** This tokenizer divides text wherever there's white space.

Note: For a full list of tokenizers, see **Add custom analyzers to string fields in an Azure AI Search index** in the **Learn more** section below.

Token filters

Once the tokenizer has divided the incoming text into tokens, you might want to add some **extra processing, such as removing stopwords or trimming punctuation marks.** You can execute this processing by specifying a token filter. There are forty one different token filters available, including:

- **Language-specific filters**, such as **arabic_normalization**. These filters apply language-specific grammar rules to ensure that forms of words are removed and replaced with roots.
- **apostrophe**. This filter removes any apostrophe from a token and any characters after the apostrophe.
- **classic.** This filter removes English possessives and dots from acronyms.
- **keep.** This filter removes any token that doesn't include one or more words from a list you specify.
- **length.** This filter removes any token that is longer than your specified minimum or shorter than your specified maximum.
- **trim.** This filter removes any leading and trailing white space from tokens.

Note: For a full list of token filters, see **Add custom analyzers to string fields in an Azure AI Search index** in the **Learn more** section below.

Create a custom analyzer

You create a custom analyzer by **specifying it when you define the index.** You must do this with JSON code - **there's no way to specify a custom index in the Azure portal.**

Use the `analyzers` section of the index at design time. **You can include only one tokenizer but one or more character filters and one or more token filters.**

Use a unique name for your analyzer and set the `@odata.type` property to `Microsoft.Azure.Search.CustomAnalyzer`.

In this example, a character filter removes HTML formatting, a tokenizer splits the text according to the grammar of Icelandic, and a token filter removes apostrophes:

```
"analyzers":(optional)[
   {
      "name":"ContosoAnalyzer",
      "@odata.type":"#Microsoft.Azure.Search.CustomAnalyzer",
      "charFilters":[
         "WebContentRemover"
      ],
      "tokenizer":"IcelandicTokenizer",
      "tokenFilters":[
         "ApostropheFilter"
      ]
   }
],
"charFilters":(optional)[
   {
      "name":"WebContentRemover",
      "@odata.type":"#html_strip"
   }
],
"tokenizers":(optional)[
   {
      "name":"IcelandicTokenizer",
      "@odata.type":"#microsoft_language_tokenizer",
      "language":"icelandic",
      "isSearchTokenizer":false,
   }
],
"tokenFilters":(optional)[
   {
      "name":"ApostropheFilter",
      "@odata.type":"#apostrophe"
   }
]
```

Test a custom analyzer

Once you've defined your custom analyzer as part of your index, you can use the REST API's **Analyze Text** function to submit test text and ensure that the analyzer returns tokens correctly. Use any REST testing tool to formulate these requests, such as the popular **Postman** application.

Your test REST requests should look like this:

```
POST https://<search service name>.search.windows.net/indexes/<index name>/analyze?api-version=<api-version>
   Content-Type: application/json
   api-key: <api key>
```

In this request:

- Replace `<search service name>` with the name of your AI Search resource.
- Replace `<index name>` with the name of the index that includes the custom analyzer.
- Replace `<api-version>` with the version number of the REST API.
- Replace `<api-key>` with the access key for your AI Search resource. You can obtain this key from the Azure portal.

Your request must also include a JSON body like this:

```
{
  "text": "Test text to analyze.",
  "analyzer": "<analyzer name>"
}
```

Replace `<analyzer name>` with the name you specified when you defined the custom analyzer. Be sure to test with lots of different `text` values until you're sure that the custom analyzer behaves as you expect.

Use a custom analyzer for a field

Once you've defined and tested a custom analyzer, you can configure your index to use it. You can specify an analyzer for each field in your index.

You can use the `analyzer` field when you want to use the **same analyzer for both indexing and searching:**

```
"fields": [
 {
   "name": "IcelandicDescription",
   "type": "Edm.String",
   "retrievable": true,
   "searchable": true,
   "analyzer": "ContosoAnalyzer",
   "indexAnalyzer": null,
   "searchAnalyzer": null
 },
```

It's also possible to use a different analyzer when indexing the field and when searching the field.

Use this configuration if you need a different set of processing steps when you index a field to when you analyze a query:

```
"fields": [
 {
   "name": "IcelandicDescription",
   "type": "Edm.String",
   "retrievable": true,
```

```
    "searchable": true,
    "analyzer": null,
    "indexAnalyzer": "ContosoIndexAnalyzer",
    "searchAnalyzer": "ContosoSearchAnalyzer"
  },
```

Enhance an index to include multiple languages

Support for multiple languages can be added to a search index. You can add language support manually by providing all the translated text fields in all the different languages you want to support.

You could also choose to **use Azure AI Services to provide translated text through an enrichment pipeline.**

Here, you'll see how to add fields with different languages to an index. You'll then constrain results to fields with specific languages. Finally, create a **scoring profile to boost the native language of your end users.**

Add language specific fields

To add multiple languages to an index, first, identify all the fields that need a translation. Then duplicate those fields for each language you want to support.

For example, if an index has an English description field, you'd add `description_fr` for the French translation and `description_de` for German. For each field, add to its definition the corresponding language analyzer.

The JSON definition of the index could look like this:

```
    {
      "name": "description",
      "type": "Edm.String",
      "facetable": false,
      "filterable": false,
      "key": false,
      "retrievable": true,
      "searchable": true,
      "sortable": false,
      "analyzer": "en.microsoft",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
    {
      "name": "description_de",
      "type": "Edm.String",
      "facetable": false,
      "filterable": false,
      "key": false,
      "retrievable": true,
      "searchable": true,
```

```
      "sortable": false,
      "analyzer": "de.microsoft",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
    {
      "name": "description_fr",
      "type": "Edm.String",
      "facetable": false,
      "filterable": false,
      "key": false,
      "retrievable": true,
      "searchable": true,
      "sortable": false,
      "analyzer": "fr.microsoft",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
    {
      "name": "description_it",
      "type": "Edm.String",
      "facetable": false,
      "filterable": false,
      "key": false,
      "retrievable": true,
      "searchable": true,
      "sortable": false,
      "analyzer": "it.microsoft",
      "indexAnalyzer": null,
      "searchAnalyzer": null,
      "synonymMaps": [],
      "fields": []
    },
```


Limit the fields for a language

In this module, you've already seen how to limit the fields returned in a search request. You can also select which fields are being searched. Your language specific search solution can combine these two features to **focus on fields with specific languages in them.**

`search='parfait pour se divertir'&$select=listingId, description_fr, city, region, tags&$searchFields=tags, description_fr&queryType=full`

Using the `searchFields` and `select` properties in the above results would return these results from the real estate sample database.


```
{
  "@odata.context": "https://advanced-cognitive-
search.search.windows.net/indexes('realestate-us-sample-index')/$metadata#docs(*)",
  "value": [
    {
```

```
      "@search.score": 12.124968,
      "listingId": "OTM4MjY1OA2",
      "description_fr": "Il s'agit d'un condo et est parfait pour se divertir.  Cette
maison offre des vues côtières Situé à proximité d'une rivière et un bureau, moulures
and une véranda couverte.",
      "city": "Seattle",
      "region": "wa",
      "tags": [
        "condo",
        "entertaining",
        "coastal views",
        "river",
        "office",
        "crown mouldings",
        "covered front porch"
      ]
    },
```

Enrich an index with multiple languages using Azure AI Services

If you don't have access to translations, you can enrich your index and add translated fields using Azure AI Services.

The steps are to add fields for each language, add a skill for each language, and then map the translated text to the correct fields.

For example, let's add Japanese and Ukrainian translations to an example retail properties index.

Add the new fields

You add two new fields to the index with these properties, the first to store the Japanese translation and the second the Ukrainian:

```
{
  "name": "description_jp",
  "type": "Edm.String",
  "facetable": false,
  "filterable": false,
  "key": false,
  "retrievable": true,
  "searchable": true,
  "sortable": false,
  "analyzer": "ja.microsoft",
  "indexAnalyzer": null,
  "searchAnalyzer": null,
  "synonymMaps": [],
  "fields": []
},
{
  "name": "description_uk",
  "type": "Edm.String",
  "facetable": false,
  "filterable": false,
  "key": false,
  "retrievable": true,
```

```
  "searchable": true,
  "sortable": false,
  "analyzer": "uk.microsoft",
  "indexAnalyzer": null,
  "searchAnalyzer": null,
  "synonymMaps": [],
  "fields": []
}
```

Add the translation skillsets

You add two skills into the skillset definition to translate the `document/description` fields into the two languages.

```
"skills": [
  {
    "@odata.type": "#Microsoft.Skills.Text.TranslationSkill",
    "name": "#1",
    "description": null,
    "context": "/document/description",
    "defaultFromLanguageCode": "en",
    "defaultToLanguageCode": "ja",
    "suggestedFrom": "en",
    "inputs": [
      {
        "name": "text",
        "source": "/document/description"
      }
    ],
    "outputs": [
      {
        "name": "translatedText",
        "targetName": "description_jp"
      }
    ]
  },
  {
    "@odata.type": "#Microsoft.Skills.Text.TranslationSkill",
    "name": "#2",
    "description": null,
    "context": "/document/description",
    "defaultFromLanguageCode": "en",
    "defaultToLanguageCode": "uk",
    "suggestedFrom": "en",
    "inputs": [
      {
        "name": "text",
        "source": "/document/description"
      }
    ],
    "outputs": [
      {
        "name": "translatedText",
        "targetName": "description_uk"
      }
```

```
    ]
  }
]
```

Map the translated output into the index

The last step is to update the indexer to map the translated text into the index.

```
"outputFieldMappings": [
  {
    "sourceFieldName": "/document/description/description_jp",
    "targetFieldName": "description_jp"
  },
  {
    "sourceFieldName": "/document/description/description_uk",
    "targetFieldName": "description_uk"
  }
]
```

The documents now all have two new translated description fields.

```
  "value": [
    {
      "@search.score": 1,
      "listingId": "OTM4MjI2NQ2",
      "beds": 5,
      "baths": 4,
      "description": "This is an apartment residence and is perfect for entertaining.
This home provides lakefront property located close to parks and features a detached
garage, beautiful bedroom floors, and lots of storage.",
      "description_de": "Dies ist eine Wohnanlage und ist perfekt für Unterhaltung.
Dieses Haus bietet Seeliegenschaft Parks in der Nähe und verfügt über eine freistehende
Garage schöne Zimmer-Etagen and viel Stauraum.",
      "description_fr": "Il s'agit d'un appartement de la résidence et est parfait pour
se divertir.  Cette maison offre propriété au bord du lac Situé à proximité de Parcs et
dispose d'un garage détaché, planchers de belle chambre and beaucoup de rangement.",
      "description_it": "Si tratta di un appartamento residence ed è perfetto per
intrattenere.  Questa casa fornisce proprietà lungolago Situato vicino ai parchi e
dispone di un garage indipendente, piani di bella camera da letto and sacco di
stoccaggio.",
      "description_es": "Se trata de una residencia Apartamento y es perfecto para el
entretenimiento.  Esta casa ofrece propiedad de lago situado cerca de parques y cuenta
con un garaje independiente, pisos de dormitorio hermoso and montón de almacenamiento.",
      "description_pl": "Jest to apartament residence i jest idealny do zabawy.  Ten dom
zapewnia lakefront Wlasciwosc usytuowany w poblizu parków i oferuje garaz wolnostojacy,
piekna sypialnia podlogi and mnóstwo miejsca do przechowywania.",
      "description_nl": "Dit is een appartement Residentie en is perfect voor
entertaining.  Dit huis biedt lakefront eigenschap vlakbij parken en beschikt over een
vrijstaande garage, mooie slaapkamer vloeren and veel opslag.",
      "description_jp": "これはアパートの住居であり、娯楽に最適です。 この家は公園の近くに位置する
湖畔のプロパティを提供し、独立したガレージ、美しいベッドルームの床とストレージの多くを備えています。",
      "description_uk": "Це багатоквартирна резиденція і прекрасно підходить для розваг.
Цей будинок забезпечує нерухомість на березі озера, розташовану недалеко від парків, і
має окремий гараж, красиві підлоги спальні та багато місць для зберігання речей.",
```

```
      ...
    },
```

Improve search experience by ordering results by distance from a given reference point

Often, users want to search for items associated with a geographical location. For example, they might want to find the nearest coffee shop to their location. To help you compare locations on the Earth's surface, **AI Search includes geo-spatial functions that you can call in queries.**

Here, you'll learn how to search for things that are near a physical point or within a bounded area.

What are geo-spatial functions?

In previous units in this module, you saw how users might locate a hotel by specifying fields to search, such as `Description` and `Category`: `search=(Description:luxury OR Category:luxury)&$select=HotelId, HotelName, Category, Tags, Description&$count=true`

An important consideration when you're booking a hotel is its geographical location. For example, if you're booking a trip to see the Eiffel Tower, you'll want a hotel located near it.

To ask AI Search to return results based on their location information, you can use two functions in your query:

- `geo.distance`. This function returns the distance in a straight line across the Earth's surface from the point you specify to the location of the search result.
- `geo.intersects`. This function returns `true` if the location of a search result is inside a polygon that you specify.

To use these functions, make sure that your index includes the location for results. Location fields should have the datatype `Edm.GeographyPoint` and store the latitude and longitude.

Use the geo.distance function

`geo.distance` is a function that takes two points as parameters and returns the distance between them in **kilometers**. Suppose you're looking for a hotel near the Eiffel Tower. You can modify the above query, adding a new filter:

`search=(Description:luxury OR Category:luxury)$filter=geo.distance(location, geography'POINT(-122.131577 47.678581)') le 5&$select=HotelId, HotelName, Category, Tags, Description&$count=true`

This query returns all the luxury hotels in the index **within five kilometers** of the Eiffel Tower. In the query:

- `Location` is the name of the field that stores the hotel's location.
- `geography'POINT(2.294481 48.858370)'` is the location of the Eiffel Tower as a longitude and latitude.
- `le 5` specifies that hotels should be included in the results if the `geo.distance` function returns a **number less than or equal to five kilometers.**

> Important: When you use `geo.distance` in a filter, the equal to (`eq`) and not equal to (`ne`) operators are **not** supported. Instead, use `lt` , `le` , `gt` , or `ge` .

Because `geo.distance` returns several kilometers, you can also use it in an `orderby` **clause.** For example, this query returns all luxury hotels in the index, but those closest to the Eiffel Tower are listed first:

```
search=(Description:luxury OR Category:luxury)&orderby=geo.distance(Location, geography'POINT(2.294481 48.858370)') asc&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

In this query, `asc` specifies that the luxury hotels are returned in the ascending order of their distance from the Eiffel Tower.

Use the geo.intersects function

Suppose you've decided that you want to stay within the seventh arrondissement of Paris for your trip to the Eiffel Tower. When you search for a hotel, you'd like to specify this area as precisely as possible. You can formulate such a query by using the `geo.intersects` function.

The `geo.intersects` function **compares a location with a polygon on the Earth's surface,** which you **specify with three or more points**. By using a polygon, you can create a shape that closely matches an area, such as an arrondissement. Use this polygon to add a geographical filter to your query:

```
search=(Description:luxury OR Category:luxury) AND geo.intersects(Location, geography'POLYGON((2.32 48.91, 2.27 48.91, 2.27 48.60, 2.32 48.60, 2.32 48.91))')&$select=HotelId, HotelName, Category, Tags, Description&$count=true
```

This query returns all luxury hotels within a square around the Eiffel Tower. You can use more than four points to create a more precise area.

> Important: In polygons, you must specify the points in **counterclockwise order** and the **polygon must be closed**, which means that the first and last points specified must be the same.

`geo.intersects` **returns a boolean value, so it's not possible to use it in an** `orderby` **clause.**

Exercise: Implement enhancements to search results

You have an existing search service that a holiday booking app uses. You've seen that the relevance of search results is impacting the number of bookings you're getting. You've also recently added hotels in Portugal so would like to offer Portuguese as a supported language.

In this exercise, you'll add a scoring profile to improve the relevance of search results. Then you'll use Azure AI Services to add Portuguese descriptions for all your hotels.

> **Note** To complete this exercise, you will need a Microsoft Azure subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free.

Create Azure resources

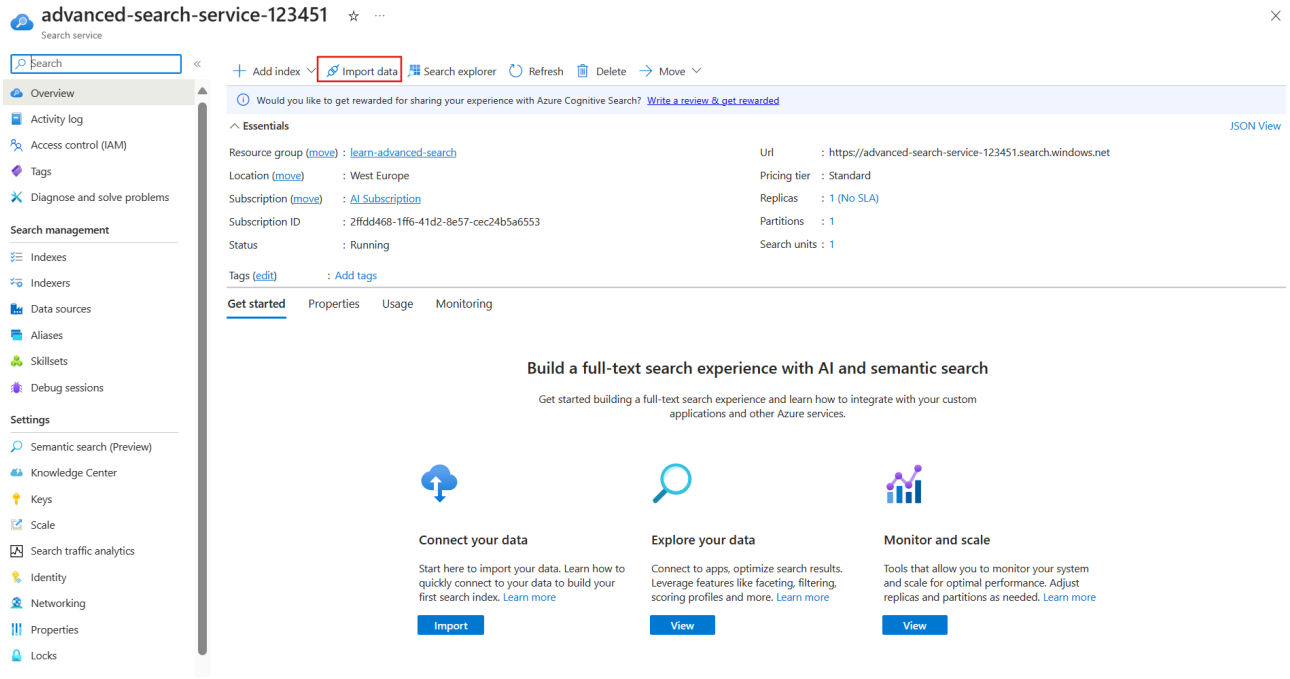
You'll create am Azure AI Search Service and import sample hotel data.

1. Sign in to the Azure portal.
2. Select **+ Create a resource**.
3. Search for **search**, and then select **Azure AI Search**.
4. Select **Create**.
5. Select **Create new** under Resource group, name it **learn-advanced-search**.
6. In **Service name**, enter **advanced-search-service-12345**. The name needs to be globally unique so add random numbers to the end of the name.
7. Select a supported Region near you.
8. Use the default values for the **Pricing tier**.
9. Select **Review + create**.
10. Select **Create**.
11. Wait for the resources to be deployed, then select **Go to resource**.

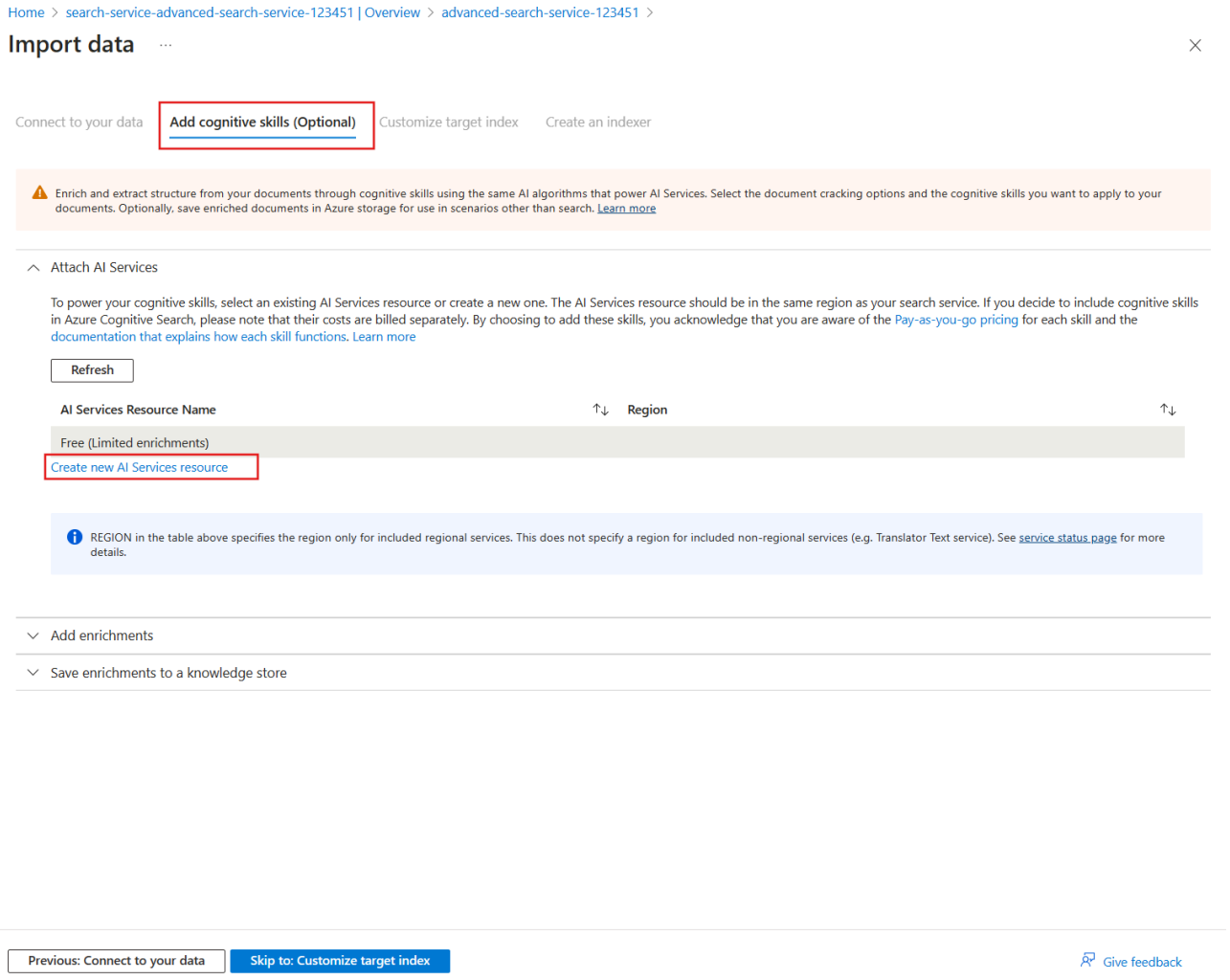
Import sample data into the search service

Import the sample data.

1. On the **Overview** pane, select **Import data**.



2. On the **Import data** pane, in the **Data source** dropdown, select **Samples**.
3. Select **hotels-sample**.
4. On the **Add cognitive skills (Optional)** tab, expand **Attach AI Services**, then select **Create new AI Services resource**.



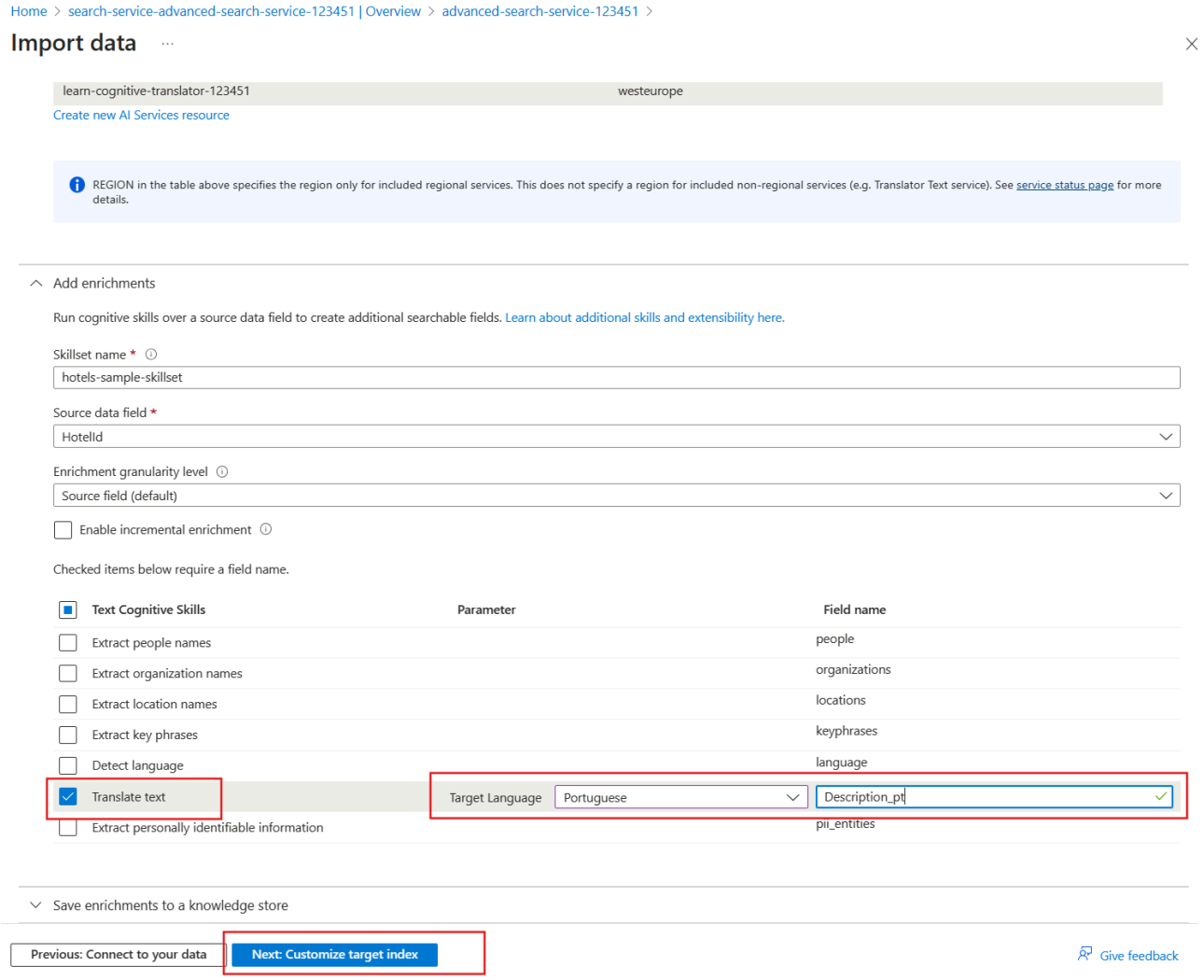
Create an Azure AI Service to support translations

1. In the new tab, sign in to the Azure portal.
2. In **Resource group**, select the **learn-advanced-search**.
3. In **Region**, select the same region you chose for the search service.
4. In **Name**, enter **learn-cognitive-translator-12345** or any name you prefer. The name needs to be globally unique so add random numbers to the end of the name.
5. In **Pricing tier**, select **Standard S0**.
6. Check **By checking this box I acknowledge that I have read and understood all the terms below**.
7. Select **Review + create**.
8. Select **Create**.
9. When the resources have been created, close the tab.

Add a translation enrichment

1. On the **Add cognitive skills (Optional)** tab, select Refresh.
2. Select the new service, **learn-cognitive-translator-12345**.

3. Expand the **Add enrichments** section.



4. Select **Translate text**, change the **Target Language** to **Portuguese**, then change the **Field name** to **Description_pt**.
5. Select **Next: Customize target index**.

Change the field to store translated text

1. On the **Customize target index** tab, scroll to the bottom of the field list and change the **Analyzer** to **Portuguese (Portugal) - Microsoft** for the **Description_pt** field.
2. Select **Next: Create an indexer**.
3. Select **Submit**.
   The index is created, the indexer will be run, and 50 documents containing sample hotel data will be imported.
4. On the **Overview** pane, select **Indexes**, then select **hotels-sample-index**.
5. Select **Search** to see JSON for all of the documents in the index.
6. Search for **Description_pt** (you can use **CTRL + F** for this) in the results and note that it isn't a Portuguese translation of the English description, but looks like this instead:

```
"Description_pt": "45",
```

The Azure portal assumes the first field in the document needs to be translated. So it's currently using the translation skill to translate the `HotelId`.

Update the skillset to translate the correct field in the document

1. At the top of the page, select the search service, **advanced-search-service-12345|Indexes** link.
2. Select **Skillsets** under Search management on the left pane, then select **hotels-sample-skillset**.
3. Edit the JSON document, change line 11 to:

```
"context": "/document/Description",
```

4. Change the default from language to English on line 12:

```
"defaultFromLanguageCode": "en",
```

5. Change the source field on line 18 to:

```
"source": "/document/Description"
```

6. Select **Save**.
7. At the top of the page, select the search service, **advanced-search-service-12345|Skillsets** link.
8. On the **Overview** pane, select **Indexers**, then select **hotels-sample-indexer**.
9. Select **Indexer Definition (JSON)**.
10. Change the source field name on line 21 to:

```
"sourceFieldName": "/document/Description/Description_pt",
```

11. Select **Save**.
12. Select **Reset**, then **Yes**.
13. Select **Run** then select **Yes**.

Test the updated index

1. At the top of the page, select the search service, **advanced-search-service-12345|Indexers** link.
2. On the **Overview** pane, select **Indexes**, then select **hotels-sample-index**.
3. Select **Search** to see JSON for all of the documents in the index.
4. Search for **Description_pt** in the results and note that now there is a Portuguese description.

```
 "Description_pt": "O maior resort durante todo o ano da área oferecendo mais de
tudo para suas férias - pelo melhor valor!  O que você pode desfrutar enquanto
estiver no resort, além das praias de areia de 1,5 km do lago? Confira nossas
atividades com certeza para excitar tanto os jovens quanto os jovens hóspedes do
coração. Temos tudo, incluindo ser chamado de \"Propriedade do Ano\" e um \"Top Ten
Resort\" pelas principais publicações.",
```

5. Now you'll search for hotels that have views of lakes. We'll start by using a simple search that returns only the `HotelName`, `Description`, `Category`, and `Tags`. In the **Query string**, enter this search:

```
lake + view&$select=HotelName,Description,Category,Tags&$count=true
```

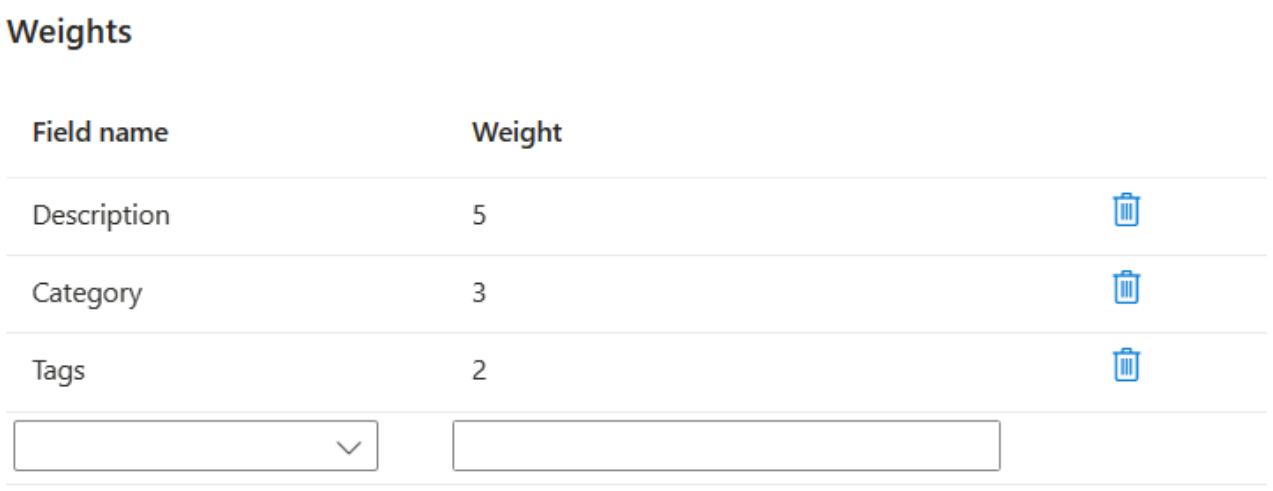
Look through the results and try to find the fields that matched the `lake` and `view` search terms. Note this hotel and its position:

```
{
  "@search.score": 0.9433406,
  "HotelName": "Lady Of The Lake B & B",
  "Description": "Nature is Home on the beach.  Save up to 30 percent. Valid Now
through the end of the year. Restrictions and blackout may apply.",
  "Category": "Luxury",
  "Tags": [
    "laundry service",
    "concierge",
    "view"
  ]
},
```

This hotel has matched the term lake in the `HotelName` field and on view in the `Tags` field. You'd like to boost matches of terms in the `Description` field over the hotel's name. Ideally, this hotel should be last in the results.

Add a scoring profile to improve search results

1. Select the **Scoring profiles** tab.
2. Select **+ Add scoring profile**.
3. In **Profile name**, enter **boost-description-categories**.
4. Add the following fields and weights under **Weights**:



5. In **Field name**, select **Description**.
6. For **Weight**, enter **5**.
7. In **Field name**, select **Category**.
8. For **Weight**, enter **3**.
9. In **Field name**, select **Tags**.
10. For **Weight**, enter **2**.
11. Select **Save**.
12. Select **Save** at the top.

Test the updated index

1. Return to the **Search explorer** tab of the **hotels-sample-index** page.
2. In the **Query string**, enter the same search as before:

   ```
   lake + view&$select=HotelName,Description,Category,Tags&$count=true
   ```

   Check the search results.


   ```
   {
     "@search.score": 3.5707965,
     "HotelName": "Lady Of The Lake B & B",
     "Description": "Nature is Home on the beach.  Save up to 30 percent. Valid Now
   through the end of the year. Restrictions and blackout may apply.",
     "Category": "Luxury",
     "Tags": [
       "laundry service",
       "concierge",
       "view"
     ]
   }
   ```

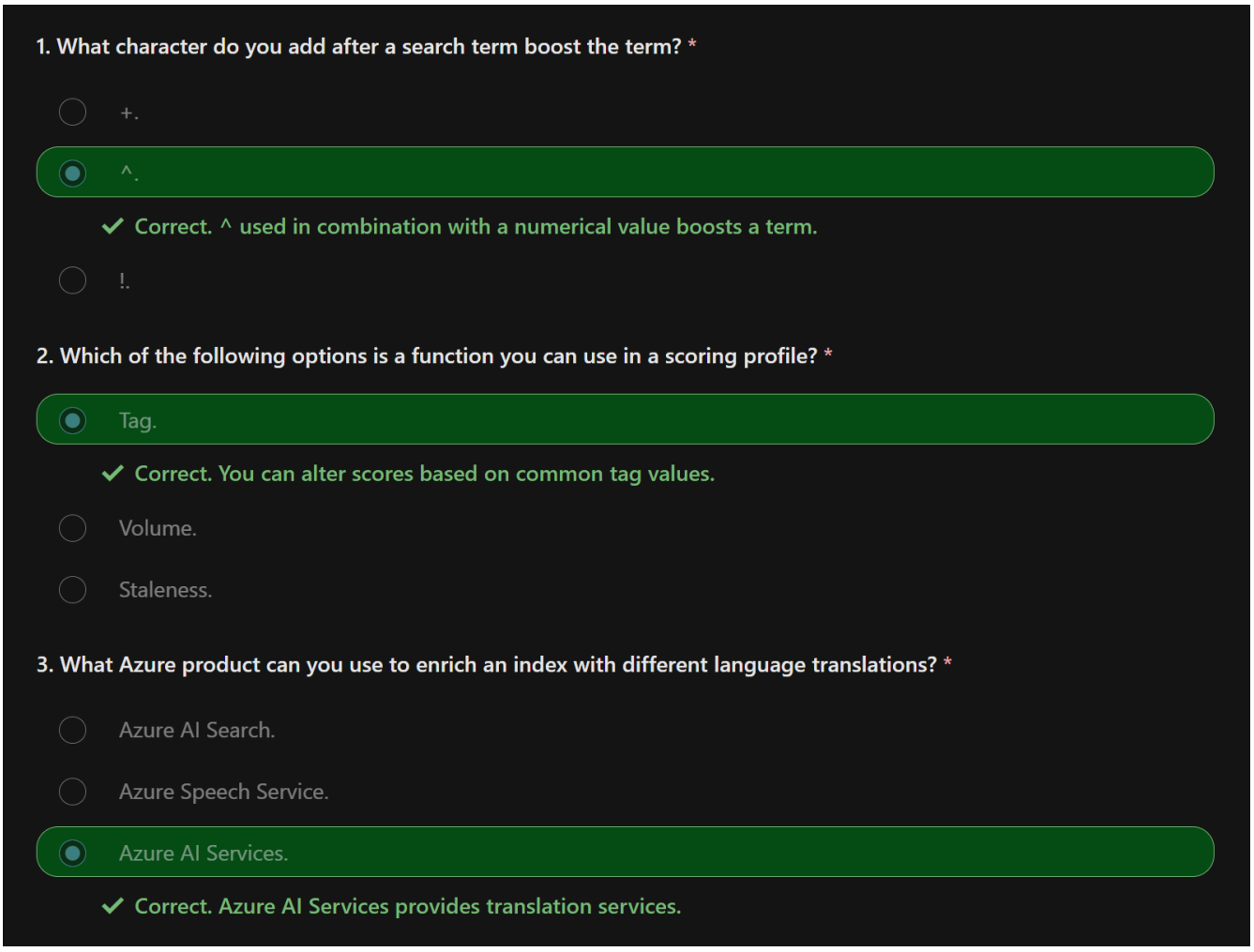

   The search score has increased, from **0.9433406** to **3.5707965**. However, all the other hotels have higher calculated scores. This hotel is now last in the results.

Delete exercise resources

Now that you've completed the exercise, delete all the resources you no longer need.

1. In the Azure portal, select **Resource Groups**.
2. Select the resource group you don't need anymore, then select **Delete resource group**.

Knowledge Check



Summary

By completing this module, you learned to:

- Improve the ranking of a document with term boosting.
- Improve the relevance of results by adding scoring profiles.
- Improve an index with analyzers and tokenized terms.
- Enhance an index to include multiple languages.
- Improve search experience by ordering results by distance from a given reference point.

To learn more about each of these topics, see the documentation:

- Score tuning
- Add scoring profiles to a search index
- Add custom analyzers to string fields in an Azure AI Search index
- Create an index for multiple languages in Azure AI Search

✍️ Compiled by Kenneth Leung (2025)