Feb 2024

Jules Albin

	Data Analytics

What's your accent ?

Table of Contents

INTRODUCTION

CONTEXT

Whether you are facing customers in the Tourism Industry, the Luxury field, the restaurants management, or any other front office activity, understanding your interlocutor's culture is paramount. It is of essence, not just for a more efficient communication, but also to better serve and avoid misinterpretation.

OBJECTIVES

The goal of this project is to investigate the relationships between native language and English accent. A step beyond is to predict native language based on the accent.

BUSINESS CASES

The ability to predict a speaker's native language based on their accent in English could have several potential business applications, for example in:

- **Language Learning Platforms**: Companies offering language learning platforms could utilize this technology to provide personalized learning experiences for users. By analyzing their accent in English, the platform could identify their native language and tailor the learning materials, exercises, and pronunciation practice specifically for that language background.

- **Customer Service** and **Support** Businesses with international customer bases could use this technology in their customer service and support departments. By identifying the native language of the caller based on their accent, companies can route calls to agents who speak the caller's native language, improving communication and customer satisfaction.

- **Market Research** and **Targeted Advertising**: Companies conducting market research or targeted advertising campaigns could use this technology to gather insights about the demographics of their audience. By analyzing the accents of respondents or customers, businesses can infer their native languages and adjust their marketing strategies accordingly to better resonate with specific language groups.

- **Language Assessment** and **Testing**: Educational institutions or organizations offering language proficiency assessments and testing could

incorporate this technology to improve the accuracy of their assessments. By analyzing the accents of test takers in English, institutions can better understand their language backgrounds and adjust scoring or evaluation criteria accordingly.

- **Speech Recognition** and **Natural Language Processing**: Companies developing speech recognition and natural language processing technologies could benefit from this capability by enhancing the accuracy and performance of their systems. Understanding the native language of the speaker can help improve speech recognition accuracy and enable more effective natural language understanding, especially in multilingual environments.

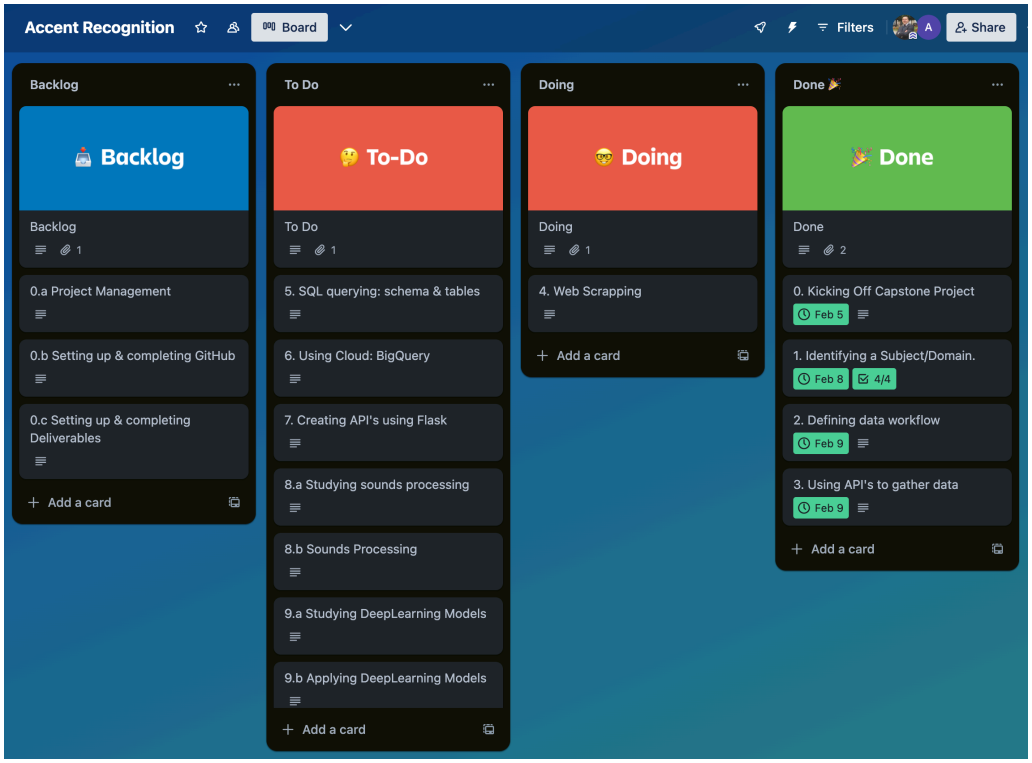
- 6. **Cross-Cultural Communication** and **Training**: Organizations with diverse international teams or conducting cross-cultural business could use this technology for communication and training purposes. By understanding the native languages of team members based on their accents, companies can provide targeted cross-cultural communication training and resources to improve collaboration and understanding among team members.

- 7. **Immigration** and **Border Control**: Government agencies responsible for immigration and border control could leverage this technology for various purposes, such as identifying the native languages of individuals during immigration interviews or border crossings. This information could assist in providing language-specific assistance or processing procedures.

- **Content Localization** and **Translation**: Companies involved in content localization and translation services could use this technology to streamline their processes. By identifying the native languages of content creators or clients based on their accents, businesses can better allocate resources for translation and localization efforts, ensuring high-quality and culturally relevant content for target audiences.

PROJECT TOOLS

1. **Trello (Kanban** board)



2. **Git & GitHub**

WORKFLOW

1. Retrieving Data from **Kaggle** & **Speech Accent Archive**:
    a. Using Kaggle's API
    b. Web scraping Speech Accent website
2. Loading datasets into BigQuery for initial data wrangling
3. SQL querying, to create a schema and normalized tables in MySQL
4. Loading data to MySQL
5. Loading data into python dataframe
6. Conducting Exploratory data analysis & Visualization
7. Building a REST API with flask, exposing data from MySQL database
8. Machine Learning:
    a. Selecting Models
    b. Performing feature selection & feature engineering
    c. Applying models

DATA & DATA SOURCES

For this project we are using 2 data sources: Kaggle and Speech Accent Archive

KAGGLE

The data exported from Kaggle consists of a CSV, a text file, and mp3 recordings:

- speakers_all.csv
- reading-passage.txt
- recordings (folder with mp3's)

The data in "reading-passage.txt" is depicted below:

Field Name	Description
age	Speaker's age at the time of the recording.
age_onset	Age at which the speaker started learning english.
birthplace	Location the speaker was born.
filename	Name of the mp3 file (speaker's recording).
country	Country where the speaker was born.
file_missing?	Indicates whether the speaker's recording is available or not

Empty column	Empty column
Empty column	Empty column

The data in “reading-passage.txt” contains the text that each speaker reads outloud:

Description
Please call Stella. Ask her to bring the**s**e thing**s** with her from the store: Six spoon**s** o**f** fresh snow peas, fi**v**e thick sla**bs** of blue cheese, and maybe a snack for her brother Bob. We also need a small plastic snake and a bi**g** toy fro**g** for the ki**ds**. She can scoop these thing**s** into three red bags, and we will go meet her Wednesday at the train station.

The “recordings” folder hosts **2134** mp3 files

Description
afrikaans1.mp3 afrikaans2.mp3 ….. …..

SPEECH ACCENT ARCHIVE

Data collecting through webscraping

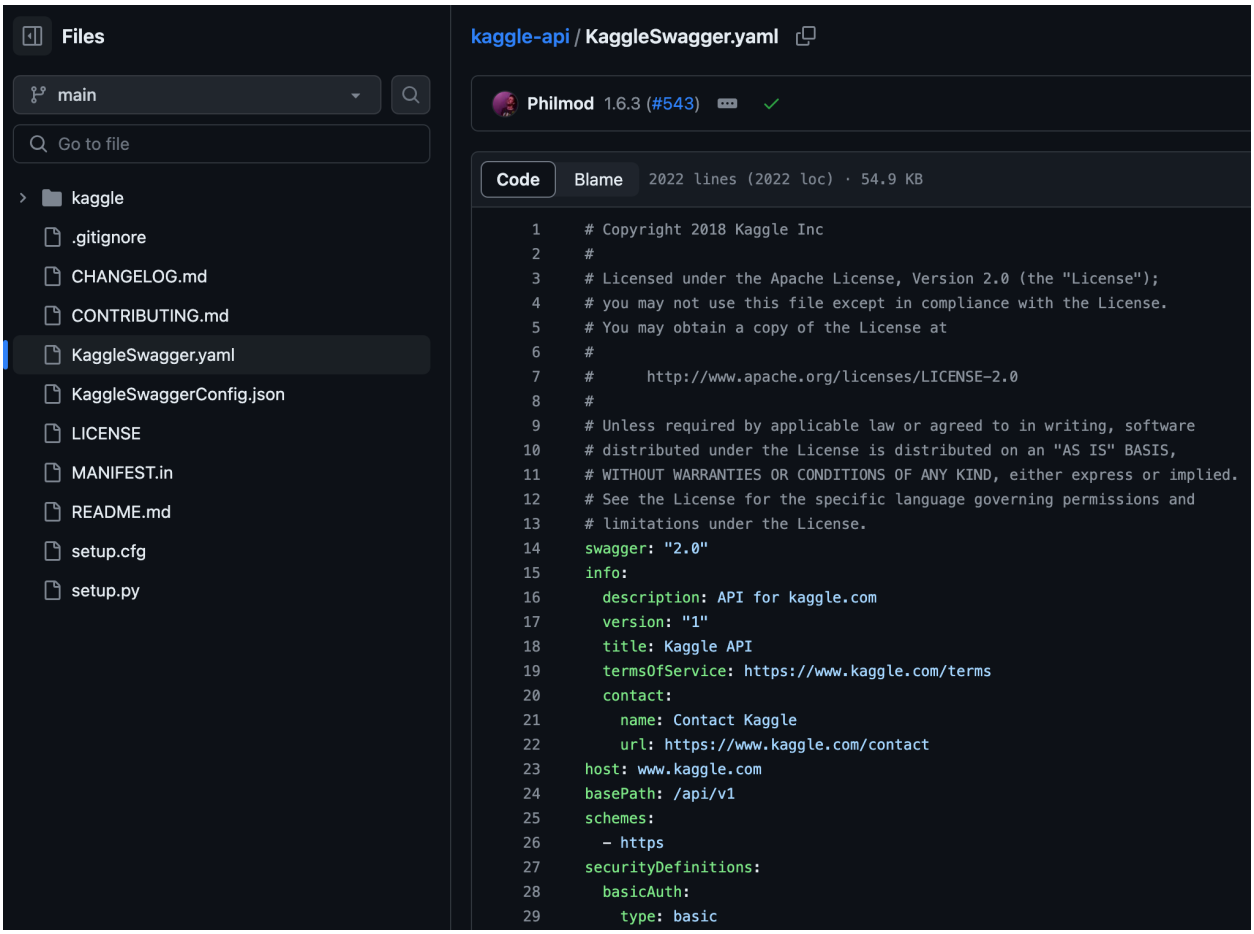
DATA COLLECTION

KAGGLE (*REST API*)

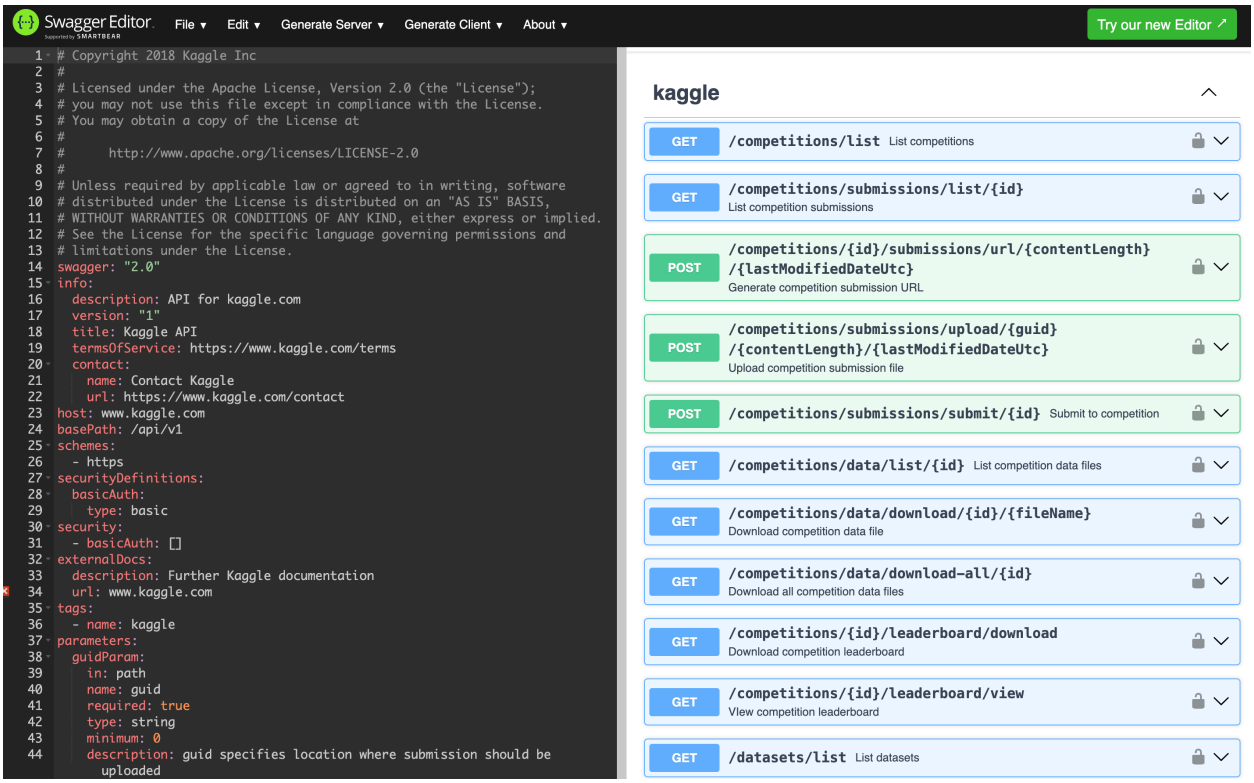
Data gathered from **Kaggle** has been captured through **Kaggle's API** & **Kaggle's CLI**.

Kaggle documented their API through the yaml file KaggleSwagger.yaml that can be found here: https://github.com/Kaggle/kaggle-api/blob/main/KaggleSwagger.yaml

Hence, in conjunction with **Swagger UI editor**, it helps explore and navigate through Kaggle's API structure.


Files
main
Go to file
kaggle
.gitignore
CHANGELOG.md
CONTRIBUTING.md
KaggleSwagger.yaml
KaggleSwaggerConfig.json
LICENSE
MANIFEST.in
README.md
setup.cfg
setup.py
kaggle-api / KaggleSwagger.yaml
Philmod 1.6.3 (#543)
Code
Blame
2022 lines (2022 loc) · 54.9 KB
Copyright 2018 Kaggle Inc
#
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
#
http://www.apache.org/licenses/LICENSE-2.0
#
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
swagger: "2.0"
info:
description: API for kaggle.com
version: "1"
title: Kaggle API
termsOfService: https://www.kaggle.com/terms
contact:
name: Contact Kaggle
url: https://www.kaggle.com/contact
host: www.kaggle.com
basePath: /api/v1
schemes:
- https
securityDefinitions:
basicAuth:
type: basic

https://editor.swagger.io/



Which eventually permitted the collection of data , such as columns information regarding the dataset for our study.

```
url_kaggle2='https://www.kaggle.com/api/v1/datasets/list/rtatman/speech-accent-archive'

response_json = requests.get(url_kaggle2, auth=HTTPBasicAuth(username=os.getenv('KAGGLE_USERNAME'), password=os.getenv('KAGGLE_KEY'))).json()

for file in response_json['datasetFiles']:
    print(file['name'],"number of columns: ", len(file['columns']),"\n")
    for col in file['columns']:
        print(col['name'], ':', col['type'])
```

```
reading-passage.txt number of columns:  0

speakers_all.csv number of columns:  11

age : Uuid
age_onset : Uuid
birthplace : String
filename : String
native_language : String
sex : String
speakerid : Uuid
country : String
file_missing? : Boolean
 : Unspecified
 : Unspecified
```

SPEECH ACCENT ARCHIVE (*Web Scraping*)

Data gathered from this website has been captured through **web scraping**.

Here is an illustration where the list of languages is collected this way

Capturing list of languages:

```
import pandas as pd
from bs4 import BeautifulSoup

languages = []
url1 = "https://accent.gmu.edu/browse_language.php"

response = requests.get(url1)
response.encoding = response.apparent_encoding

if response.status_code == 200:
    soup = BeautifulSoup(response.content, "html.parser")
    ul_languages = soup.find("ul", class_="languagelist")

    for language in ul_languages.find_all("li"):
        languages.append(language.text)
else:
    print("ERROR:", response.status_code)

df_languages = pd.DataFrame(languages,columns=['languages'])
df_languages.to_csv('../data/languages.csv', index=False)
```

DATA CLEANING & EXPLORATORY DATA ANALYSIS

DATA CLEANING

Documentation in progress.

EXPLORATORY DATA ANALYSIS

Documentation in progress…

DATABASE TYPE SELECTION

RELATIONAL DATABASES

Relational Database:
Pros: • Data Integrity: Relational databases enforce data integrity through constraints like primary keys, foreign keys, and unique constraints, ensuring accurate and consistent data. • Structured Query Language (SQL): They use SQL for querying and manipulating data, which is a widely understood and standardized language. • ACID Transactions: Relational databases typically support ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity even in the face of concurrent access. • Complex Queries: Relational databases are well-suited for complex queries involving multiple tables and relationships.
Cons: • Scalability: Scaling relational databases can be challenging, especially when dealing with large volumes of data or high transaction rates. • Schema Changes: Making changes to the database schema can be cumbersome and may require downtime or affect performance. • Performance: Performance can degrade as the size of the database grows, especially for complex queries involving joins across multiple tables. • Not Ideal for Unstructured Data: Relational databases are not well-suited for handling unstructured or semi-structured data, such as documents or JSON data.

NON-RELATIONAL DATABASES

Non-Relational Database:
Pros: • Scalability: Non-relational databases are often designed for horizontal scalability, making them well-suited for handling large volumes of data and high traffic loads.

- Schema Flexibility: Non-relational databases typically offer schema flexibility, allowing for the storage of semi-structured and unstructured data without a fixed schema.
- Performance: Non-relational databases can offer high performance, especially for simple read and write operations, due to their optimized data storage and retrieval mechanisms.
- Handling Big Data: They are well-suited for handling big data and real-time analytics applications, where rapid ingestion and processing of large volumes of data are required.

Cons:

- Data Consistency: Non-relational databases may sacrifice strong consistency guarantees in favor of performance and scalability, leading to eventual consistency models.
- Querying: Querying in non-relational databases may be less expressive compared to SQL in relational databases, especially for complex queries involving joins and aggregations.
- Less Mature Ecosystem: Non-relational databases often have a less mature ecosystem compared to relational databases, with fewer tools, libraries, and community support.
- Learning Curve: Developers accustomed to relational databases may face a learning curve when working with non-relational databases due to differences in data modeling and query languages.

DATABASE CREATION

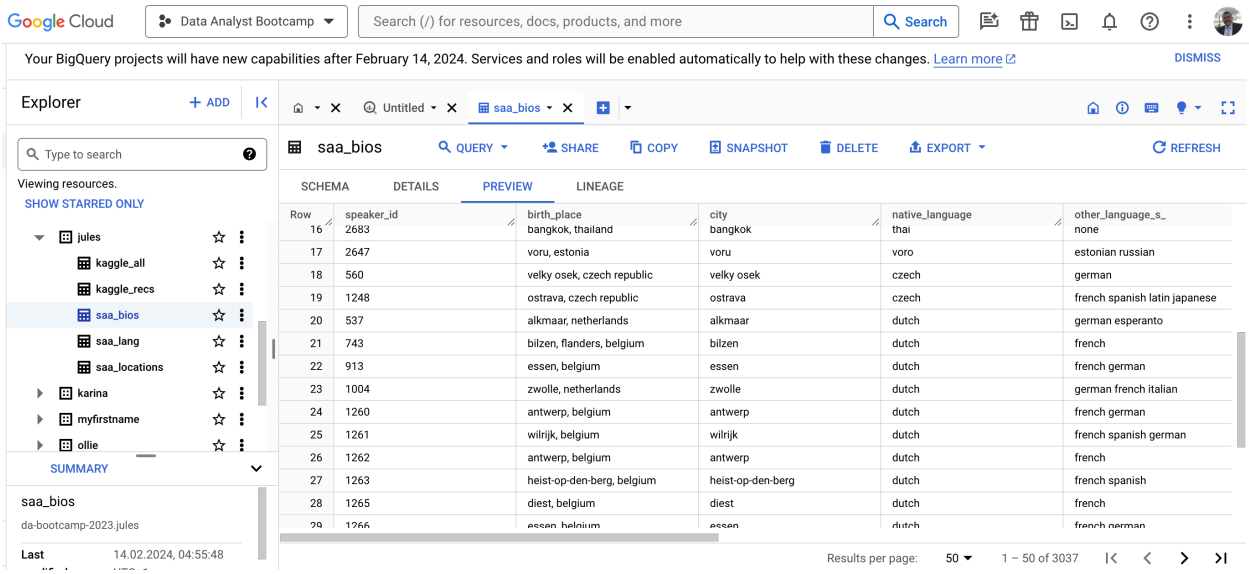
SQL QUERIES

```
-- Drop SCHEMA if they exist
DROP SCHEMA IF EXISTS accents;

-- Create SCHEMA
CREATE SCHEMA accents;

-- Create database
CREATE DATABASE IF NOT EXISTS accents;

USE accents;

-- Drop tables if they exist
DROP TABLE IF EXISTS kaggle_recs;
DROP TABLE IF EXISTS kaggle_all;
DROP TABLE IF EXISTS saa_lang;
DROP TABLE IF EXISTS saa_recs;
DROP TABLE IF EXISTS saa_locations;
DROP TABLE IF EXISTS saa_bios;
```

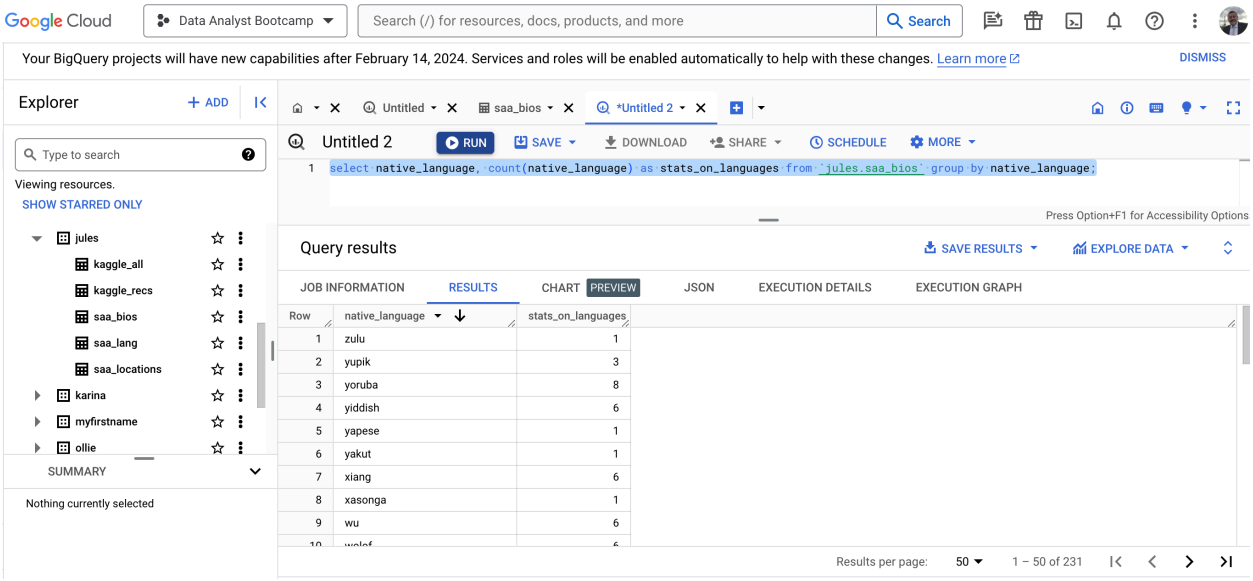
```
-- Create the table saa_lang
CREATE TABLE saa_lang (
  language_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  language VARCHAR(255) NOT NULL,
  PRIMARY KEY  (language_id)
);

-- Create the table saa_locations
CREATE TABLE saa_locations (
  location_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  location VARCHAR(255) NOT NULL,
  PRIMARY KEY  (location_id)
) ;


-- Create the table saa_recs
CREATE TABLE saa_recs (
  speaker_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  filename VARCHAR(255),
  PRIMARY KEY  (speaker_id)
);

-- Create kaggle_all table
CREATE TABLE kaggle_all (
    age INT NOT NULL,
    age_onset INT NOT NULL,
    birthplace VARCHAR(255),
    filename VARCHAR(255),
    native_language VARCHAR(255),
    sex VARCHAR(255),
    speaker_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    country VARCHAR(255),
    file_missing BOOLEAN,
    PRIMARY KEY  (speaker_id),
    CONSTRAINT fk_saa_recs_kaggle_all FOREIGN KEY (speaker_id) REFERENCES saa_recs(speaker_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

```
-- Create the table kaggle_recs
CREATE TABLE kaggle_recs (
  speaker_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  filename VARCHAR(255),
  PRIMARY KEY  (speaker_id),
  CONSTRAINT fk_kaggle_all_kaggle_recs FOREIGN KEY (speaker_id) REFERENCES kaggle_all(speaker_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
-- Create the table saa_bios
CREATE TABLE saa_bios (
    speaker_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    birth_place VARCHAR(255),
    city VARCHAR(255),
    native_language VARCHAR(255),
    other_languages VARCHAR(255),
    age FLOAT,
    sex VARCHAR(255),
    age_of_english_onset VARCHAR(255),
    english_learning_method VARCHAR(255),
    english_residence VARCHAR(255),
    length_of_english_residence VARCHAR(255),
    language VARCHAR(255),
    language_id SMALLINT UNSIGNED NOT NULL,
    location_id SMALLINT UNSIGNED NOT NULL,
    PRIMARY KEY  (speaker_id),
    KEY idx_fk_location_id (location_id),
    KEY idx_fk_language_id (language_id),
    CONSTRAINT fk_saa_bios_location_id FOREIGN KEY (location_id) REFERENCES saa_locations (location_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT fk_saa_bios_language_id FOREIGN KEY (language_id) REFERENCES saa_lang (language_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Cloud: BigQuery

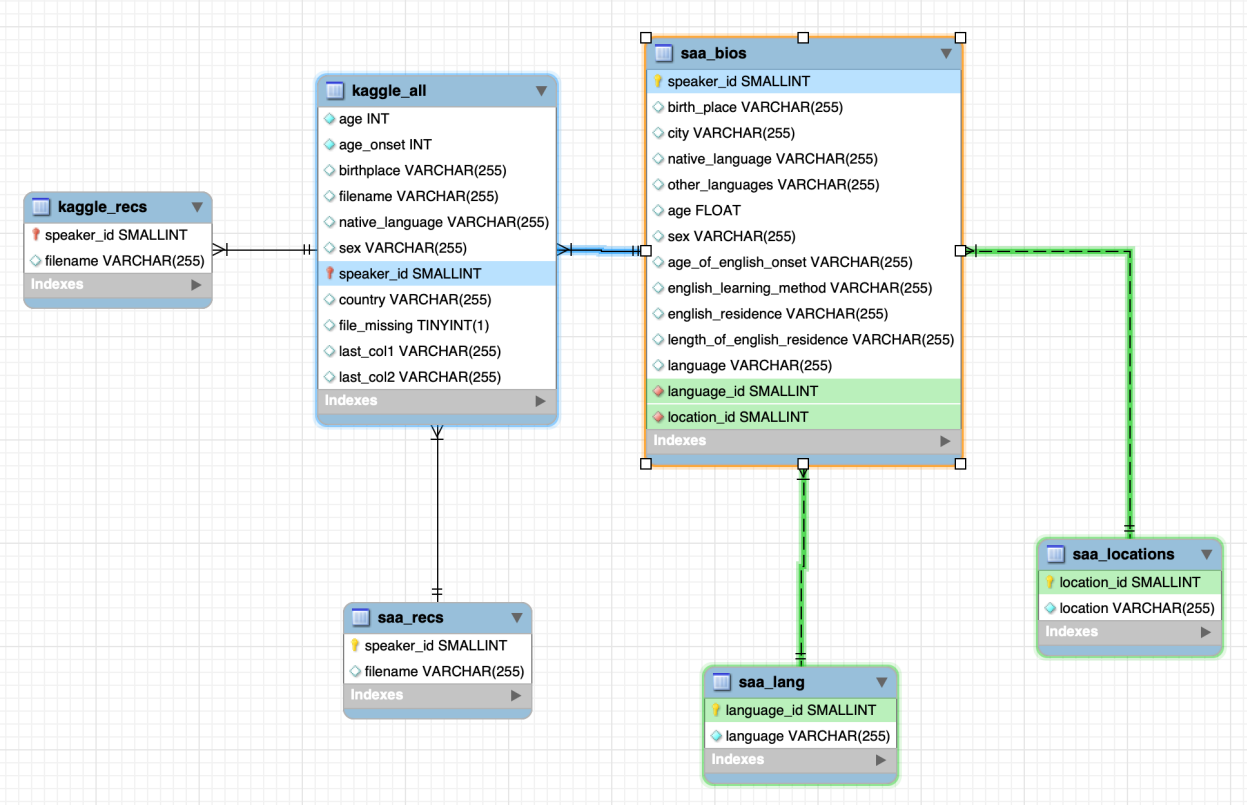


```
select native_language, count(native_language) as stats_on_languages from
`jules.saa_bios` group by native_language;
```



ENTITY RELATIONSHIP DIAGRAM

Here is the ERD of our Schema.



GENERAL DATA PROTECTION REGULATION (GDPR)

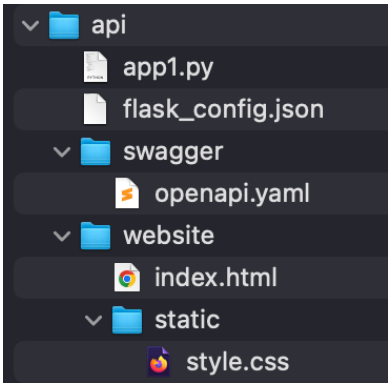
TYPE OF DATA

The usage of soundtracks used in this project is compliant with GDPR norms.

API DEVELOPMENT

CREATING REST API WITH FLASK

FOLDERS STRUCTURE



YAML FILE

openapi.yaml.

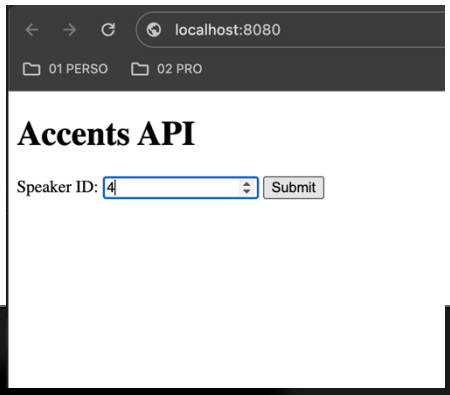
```yaml
openapi: 3.0.0
info:
  title: Swagger
  version: 1.0.0
paths:
  /speaker_info/{speaker_id}:
    get:
      summary: Get speaker information
      parameters:
        - in: path
          name: speaker_id
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                type: object
                properties:
                  native_language:
                    type: string
                  age:
                    type: number
                  sex:
                    type: string
        '404':
          description: Speaker not found
  /locations:
    get:
      summary: Get all locations
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

APP1.PY

```
from flask import Flask, jsonify, request
from flask_swagger_ui import get_swaggerui_blueprint
import mysql.connector
import pymysql
import os

app = Flask(__name__)

db = mysql.connector.connect(
    host="localhost",
    user="root",
    password=os.getenv('MySQLpwd'),
    database="accents"
)
cursor = db.cursor()

SWAGGER_URL = '/swagger'
API_URL = '/static/openapi.yaml'
swaggerui_blueprint = get_swaggerui_blueprint(
    SWAGGER_URL,
    API_URL,
    config={'app_name': "Swagger"}
)
app.register_blueprint(swaggerui_blueprint, url_prefix=SWAGGER_URL)

@app.route('/speaker_info/<int:speaker_id>')
def get_speaker_info(speaker_id):
    try:
        query = "SELECT native_language, age, sex FROM saa_bios WHERE speaker_id = %s"
        cursor.execute(query, (speaker_id,))
        result = cursor.fetchone()
        if result:
            speaker_info = {
                'native_language': result[0],
                'age': result[1],
                'sex': result[2]
            }
            return jsonify(speaker_info)
        else:
            return jsonify({'message': 'Speaker not found'}), 404
    except Exception as e:
        return jsonify({'message': str(e)}), 500

@app.route('/locations')
def get_locations():
    try:
        query = "SELECT location FROM saa_locations"
        cursor.execute(query)
        results = cursor.fetchall()
        locations = [result[0] for result in results]
        return jsonify(locations)
    except Exception as e:
        return jsonify({'message': str(e)}), 500

if __name__ == '__main__':
    app.run(port=8080)
```

```
~/projects/accent_recognition/api$ flask --app app1 run --port 8080
 * Serving Flask app 'app1'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:8080
Press CTRL+C to quit
```

WEBSITE



HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Accents API</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Accents API</h1>
    <form action="/speaker_info" method="get">
        <label for="speaker_id">Speaker ID:</label>
        <input type="number" id="speaker_id" name="speaker_id">
        <button type="submit">Submit</button>
    </form>
    <div id="locations"></div>
</body>
</html>
```

CSS

```css
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

h1 {
    color: #333;
}

form {
    margin-bottom: 20px;
}

label {
    font-weight: bold;
}

input[type="number"] {
    width: 100px;
    margin-right: 10px;
}

button {
    padding: 5px 10px;
    background-color: #007bff;
    color: #fff;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

#locations {
    margin-top: 20px;
}
```

MACHINE LEARNING

MODELS SELECTION

xxx.

MODELS APPLICATION

Features and Target used in the following models

```
features, target = num_var, 'native_language'

if target in features: features.remove(target)
```

Scaling & Splitting dataset

```
X_train_scaled, X_test_scaled, y_train, y_test = jt.scaling(df, features, target,StandardScaler())
```

Models application: in progress…(to be defined, but likely **CNN** or **RNN**)

Evaluation Report:

```
for model in models:
    y_pred = jt.model_fit_predict(model,X_train_scaled, X_test_scaled, y_train, y_test)
    jt.eval_report(y_test,y_pred)
```