



Published in Sogeti Data | Netherlands



Tijana Nikolic

Follow

Feb 25, 2022 · 9 min read · Listen



Save



Open in app

Sign up

Sign In



Search Medium



Sogeti NL has a large data science team that's always looking for methods to ensure transparency, ethics and quality in their AI development process. Additionally, we are involved in a project that focuses on testing AI models in various development phases — ITEA IVVES. As part of this project, we developed the **Data Quality Wrapper (DQW)**, an app for automated EDA, preprocessing and data quality report generation. Its goal is to automate the preprocessing of data, but also educate aspiring and experienced data scientists about different methods that can be used to improve its quality.

While trying to create an app around this solution, we found **Streamlit**, a framework for easy app development for ML projects and experiments. I've already written about how easy it is to develop apps with it.

In this blog post, we will go through the purpose of the app and its sections, Streamlit components and packages used to develop the app. We will also point to the scripts where the code is located (look for the 🔍 emoji).

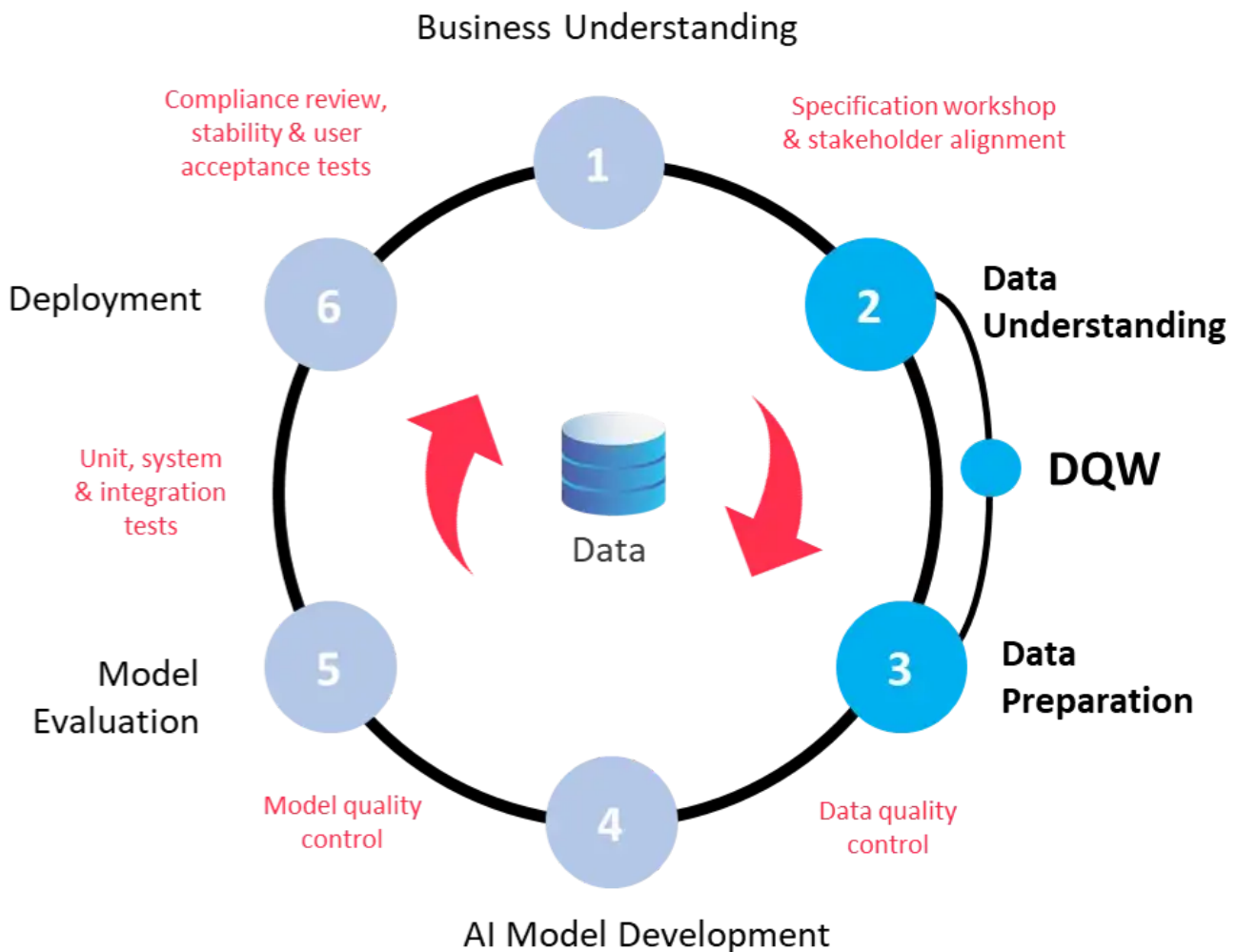
TL;DR? Try out the app. 🚀 Or jump into the code! 👤

### The purpose of the app



78

Sogeti's DQW is used as an accelerator for another product we are developing, the Quality AI Framework, used to test AI models in all phases of the AI development cycle. The framework provides a practical and standardized way of working that outputs trustworthy AI. Sogeti's DQW is used in the Data Understanding and Data Preparation phase of this framework. It is an accelerator used to ensure the quality of the data that goes into a given ML model is suitable and representative.



The phases of the QAIF and where the DQW is positioned.

The best thing about the app is that it can be applied to more than one data structure, including:

- **Structured data.** Data in a well-defined format. Used in various ML applications.
- **Unstructured data.** This includes **images**, used in computer vision algorithms such as object detection and classification, **text**, used in NLP models, be it for classification or sentiment analysis and **audio**, used in audio signal processing algorithms such as music genre recognition and automatic speech recognition.
- **Synthetic data.** Synthetic data evaluation is a critical step of the synthetic data generation pipeline. Validating the synthetic data training set ensures model performance will not be impacted negatively.

These data formats define the app sections, which you can toggle through in the main selectbox. Each of the sections has multiple subsections, which we will go through in the next few paragraphs.

Packages used to enable the EDA (description, visualisation) and preprocessing (selection) of these data formats are below. Please note, these are packages that we **recommend** using, not a definite guide.

App section	Description	Visualisation	Selection	Package
Synthetic structured	x	x		<a href="#">table-evaluator</a>
Structured	x	x		<a href="#">sweetviz</a>
Structured	x	x		<a href="#">pandas-profiling</a>
Structured, text			x	<a href="#">PyCaret</a>
Text			x	<a href="#">NLTK</a>
Text			x	<a href="#">SpaCy</a>
Text	x		x	<a href="#">TextBlob</a>
Text	x	x		<a href="#">WordCloud</a>
Text	x		x	<a href="#">TextStat</a>
Image	x	x		<a href="#">Pillow</a>
Audio	x	x		<a href="#">librosa</a>
Audio	x	x		<a href="#">dtw</a>
Audio			x	<a href="#">audiomentations</a>
Audio	x	x		<a href="#">AudioAnalyser</a>
Report generation	x			<a href="#">Fpdf</a>
Report generation	x			<a href="#">wkhtmltopdf</a>
Report generation	x			<a href="#">pdftkit</a>

packages.md hosted with ❤ by GitHub [view raw](#)

Let's jump into the app sections!

## Structured data section

The section of the DQW dedicated to structured data offers automated EDA and preprocessing of your data. The code is placed in the **tabular\_eda** folder.

Structured data subsections include one file analysis and preprocessing, two file comparison and synthetic data comparison. Let's go through them.

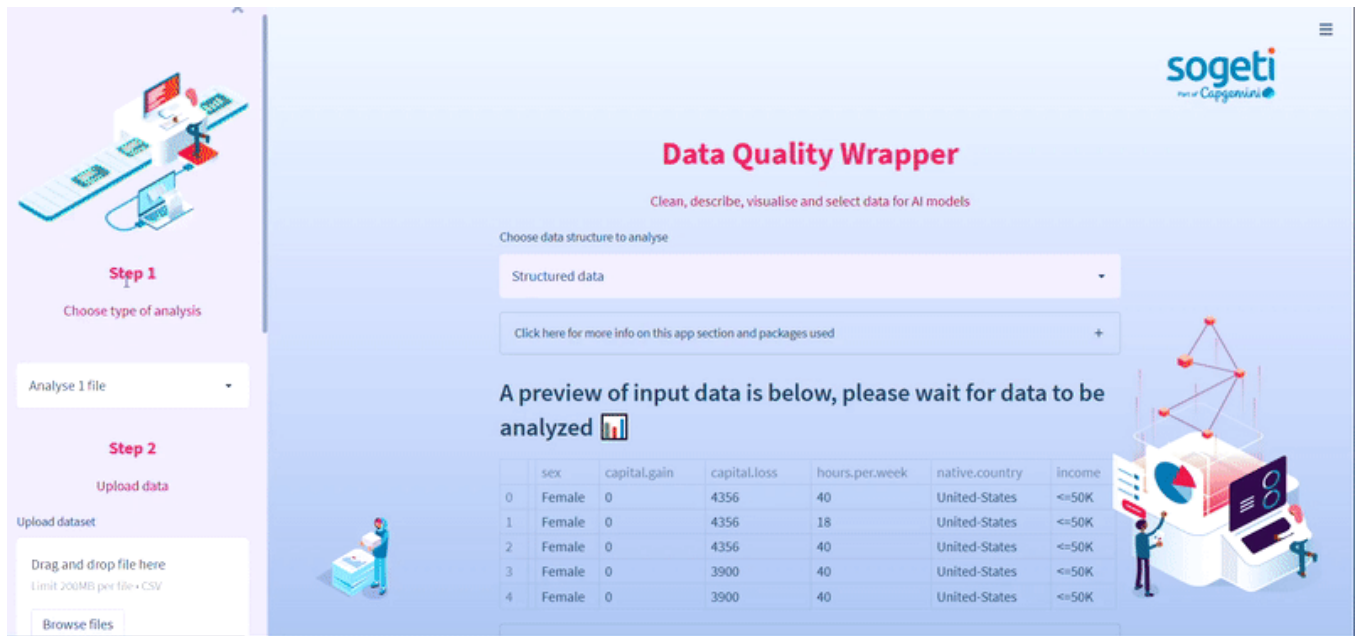
**One file analysis** subsection uses pandas-profiling — easy to set up due to the Streamlit pandas-profiling component. The code used is below.

```
1  import streamlit as st
2  import pandas_profiling
3  from streamlit_pandas_profiling import st_profile_report
4
5  uploaded_data = st.sidebar.file_uploader("Upload dataset", type="csv")
6  data = pd.read_csv(uploaded_data)
7
8  # create the pandas profiling report
9  pr = data.profile_report()
10 st_profile_report(pr)
11 # optional, save to file
12 pr.to_file("pandas_prof.html")
```

st\_pandas-profiling.py hosted with ❤ by GitHub

[view raw](#)

**One file preprocessing** with PyCaret — a very useful package for workflow automation. In the DQW, we rely on the `setup()` function which creates the preprocessing pipeline. Streamlit widgets make it quite easy to add flexibility for the user to select which preprocessing steps they want to run. We also display these steps as a diagram, offer a comparison of the original and preprocessed file and the download of the report and the pipeline pickle file you can use later. The pipeline pickle is provided so you can easily use it with the PyCaret modelling functions, especially in case of imbalanced class mitigation with SMOTE. The sampling needs to happen within training folds, so you won't be able to see any impact of this method on the datasets themselves, but you'll be able to see the difference in model performance when you use the pipeline pickle file.



🔍 The code used is in the [structured\\_data.py](#) script, see [preprocess](#) and [show\\_pp\\_file](#) functions.

**Two file comparison with Sweetviz** — another automated EDA library extremely useful for two file comparison. If we want to show the Sweetviz html report, we need to use the Streamlit html components function, as seen below.

```

1  import streamlit as st
2  import streamlit.components.v1 as components
3  import sweetviz as sv
4
5  uploaded_ref = st.sidebar.file_uploader("Upload reference dataset", type="csv")
6  ref= pd.read_csv(uploaded_ref)
7
8  uploaded_comparison = st.sidebar.file_uploader("Upload comparison dataset", type="csv")
9  comparison = pd.read_csv(uploaded_comparison)
10
11 sw = sv.compare([ref, "Reference"], [comparison, "Comparison"])
12
13 sw.show_html(open_browser=False, layout='vertical', scale=1.0)
14
15 display = open("SWEETVIZ_REPORT.html", 'r', encoding='utf-8')
16
17 source_code = display.read()
18
19 # you can pass width as well to configure the size of the report
20 components.html(source_code, height=1200, scrolling=True)

```

streamlit\_sweetviz.py hosted with ❤ by GitHub

[view raw](#)

**Synthetic data comparison with table-evaluator.** A wholesome comparison of original and synthetic datasets, it checks all statistical properties (PCA included) and offers multiple model performance comparisons with the original and synthetic dataset.

**Upload data**

**Upload reference dataset**

Drag and drop file here  
Limit 200MB per file • CSV

Browse files

real\_test\_sample.csv 71.0KB

**Upload comparison dataset**

Drag and drop file here  
Limit 200MB per file • CSV

Browse files

fake\_test\_sample.csv 71.1KB

**Step 3**

Choose table-evaluator method

Compare model performance

Select the target column:

trans\_type

Correlation metric: pearsonr

Classifier F1-scores and their Jaccard similarities:

	f1_reference	f1_comparison	jaccard_similarity
LogisticRegression_reference_tc	0.7550	0.7450	0.9417
LogisticRegression_comparison	0.8000	0.8100	0.9231
RandomForestClassifier_referen	0.9700	0.9700	1
RandomForestClassifier_compa	0.9600	0.9550	0.9900
DecisionTreeClassifier_referenc	0.9550	0.9200	0.8605
DecisionTreeClassifier_compari	0.9550	0.9000	0.8605
MLPClassifier_reference_testset	0.4850	0.5350	0.3699
MLPClassifier_comparison_tests	0.4000	0.4750	0.4440

Miscellaneous results:

	Result
Column Correlation Distance RMSE	0.03991519959511691
Column Correlation distance MAE	0.029637281719367928
Duplicate rows between sets (reference/comparison)	(0, 0)
nearest neighbor mean	0.5654889342640013
nearest neighbor std	0.37263768630663785

sogeti part of Capgemini

Manage app

As **Step 4**, you can download the report and files. An example of the code is below.

```

1  def generate_zip_structured(original, comparison):
2      """ A function to write files to disk and zip 'em """
3      original.to_csv('pdf_files/synthetic_data/reference_file_dqw.csv',
4                      index=False)
5      comparison.to_csv('pdf_files/synthetic_data/comparison_file_dqw.csv',
6                       index=False)
7      # create a ZipFile object
8      zipObj = ZipFile('pdf_files/synthetic_data/report_files_dqw.zip', 'w')
9      # Add multiple files to the zip
10     zipObj.write('pdf_files/synthetic_data/reference_file_dqw.csv')
11     zipObj.write('pdf_files/synthetic_data/comparison_file_dqw.csv')
12     zipObj.write('pdf_files/synthetic_data/table-evaluator_comparison_dqw.pdf')
13     # close the Zip File
14     zipObj.close()
15
16     zip = generate_zip_structured(original, comparison)
17
18     # sidebar download, you can remove the sidebar api to have the normal download button
19     with open("pdf_files/synthetic_data/report_files_dqw.zip", "rb") as fp:
20         st.sidebar.download_button(
21             "📄",
22             data=fp,
23             file_name="te_compare_files_dqw.zip",
24             mime="application/zip"
25         )

```

streamlit\_zip.py hosted with ❤️ by GitHub

[view raw](#)

🔍 The code used is in the `structured_data.py` ([table\\_evaluator\\_comparison](#)), [te.py](#), [viz.py](#) and [metrics.py](#). I copied the script from the repository because I needed to adjust them to make the package work efficiently in Streamlit. If you would like to try out this package, you can simply install it as it is.

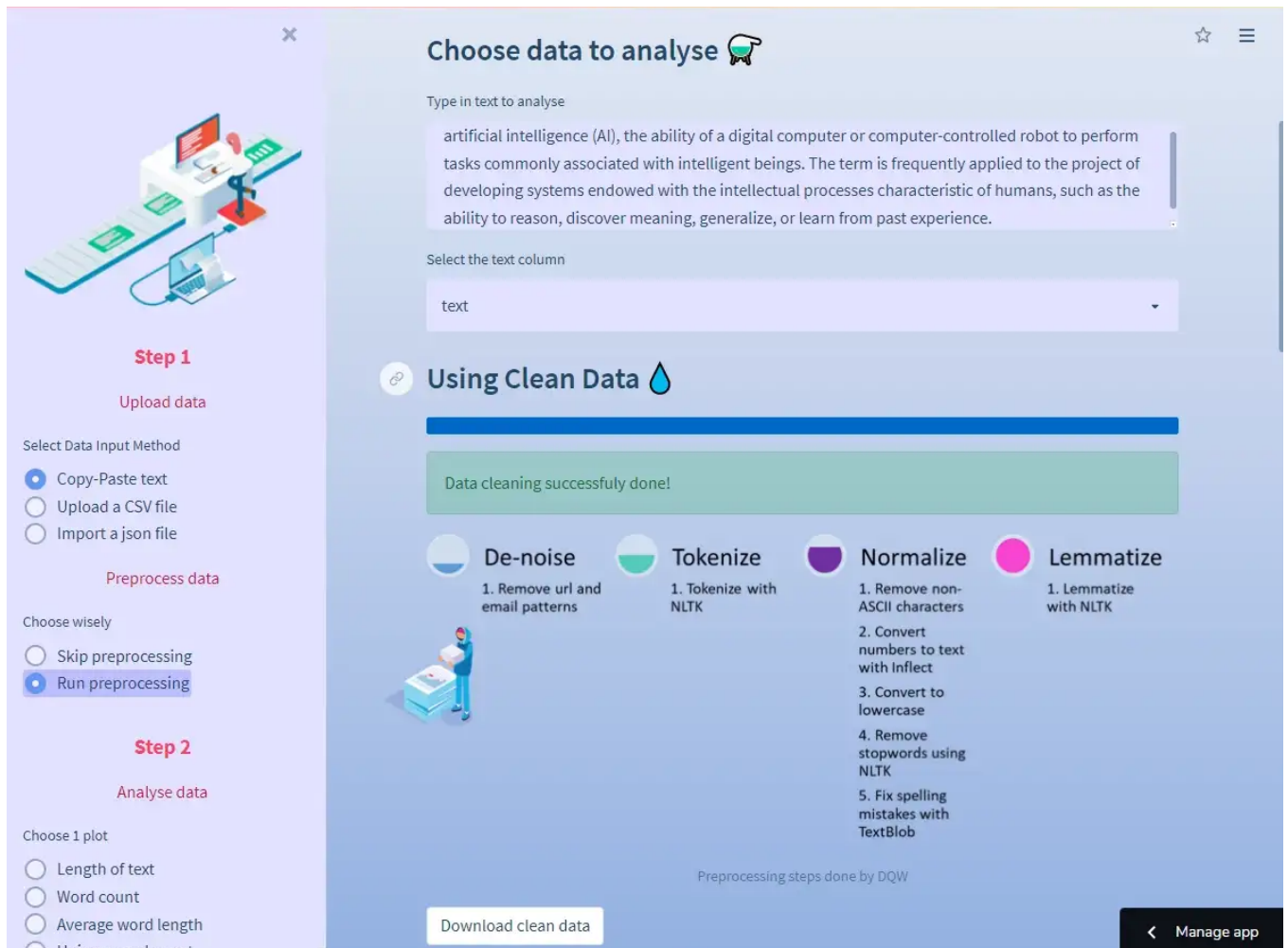
### Text data section

The text data section offers the flexibility of pasting a body of text or uploading a csv/json file for analysis. It currently only supports English, but it offers a lot of analysis methods and automated data preprocessing. The code is placed in [text\\_edu](#) folder.

Let's focus on the most interesting subsections.

**Data preprocessing** subsection relies on various text preprocessing functions like stemming, lemmatization, de-noising, and stop word removal. These steps prepare text data in a machine-readable way. The preprocessed file can be downloaded.





The screenshot shows the 'Choose data to analyse' interface of the Sogeti Data Quality Wrapper. The interface is divided into a left sidebar and a main content area.

**Left Sidebar:**

- Step 1: Upload data**
  - Select Data Input Method:
    - ☒ Copy-Paste text
    - ☐ Upload a CSV file
    - ☐ Import a json file
  - Preprocess data**
    - Choose wisely:
      - ☐ Skip preprocessing
      - ☒ Run preprocessing
  - Step 2: Analyse data**
    - Choose 1 plot:
      - ☐ Length of text
      - ☐ Word count
      - ☐ Average word length
      - ☐ Unknown word count

**Main Content Area:**

- Choose data to analyse**
  - Type in text to analyse:
 

artificial intelligence (AI), the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. The term is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience.
  - Select the text column:
 

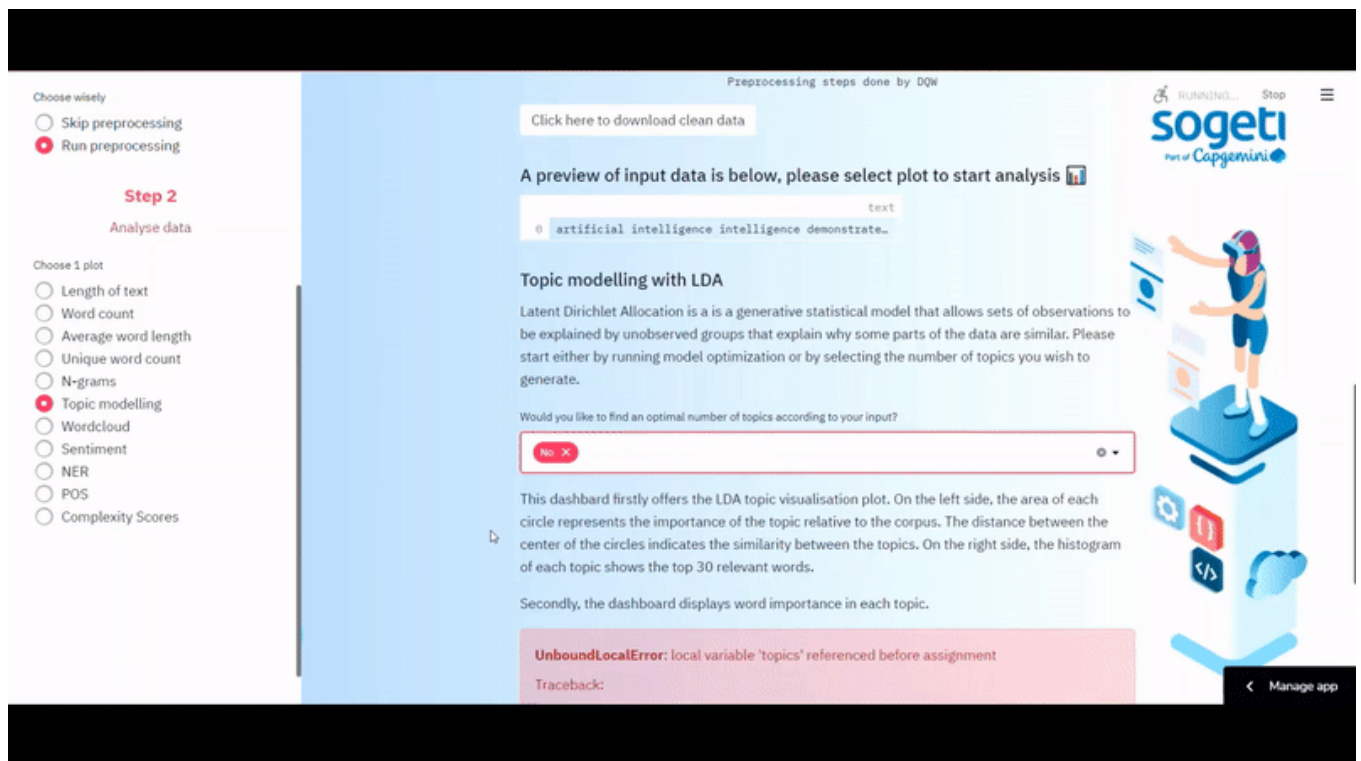
text
- Using Clean Data**
  - Data cleaning successfully done!
  - Preprocessing steps done by DQW
  - Download clean data
  - Manage app

**Preprocessing Steps:**

- De-noise**
  - 1. Remove url and email patterns
- Tokenize**
  - 1. Tokenize with NLTK
- Normalize**
  - 1. Remove non-ASCII characters
  - 2. Convert numbers to text with Inflect
  - 3. Convert to lowercase
  - 4. Remove stopwords using NLTK
  - 5. Fix spelling mistakes with TextBlob
- Lemmatize**
  - 1. Lemmatize with NLTK

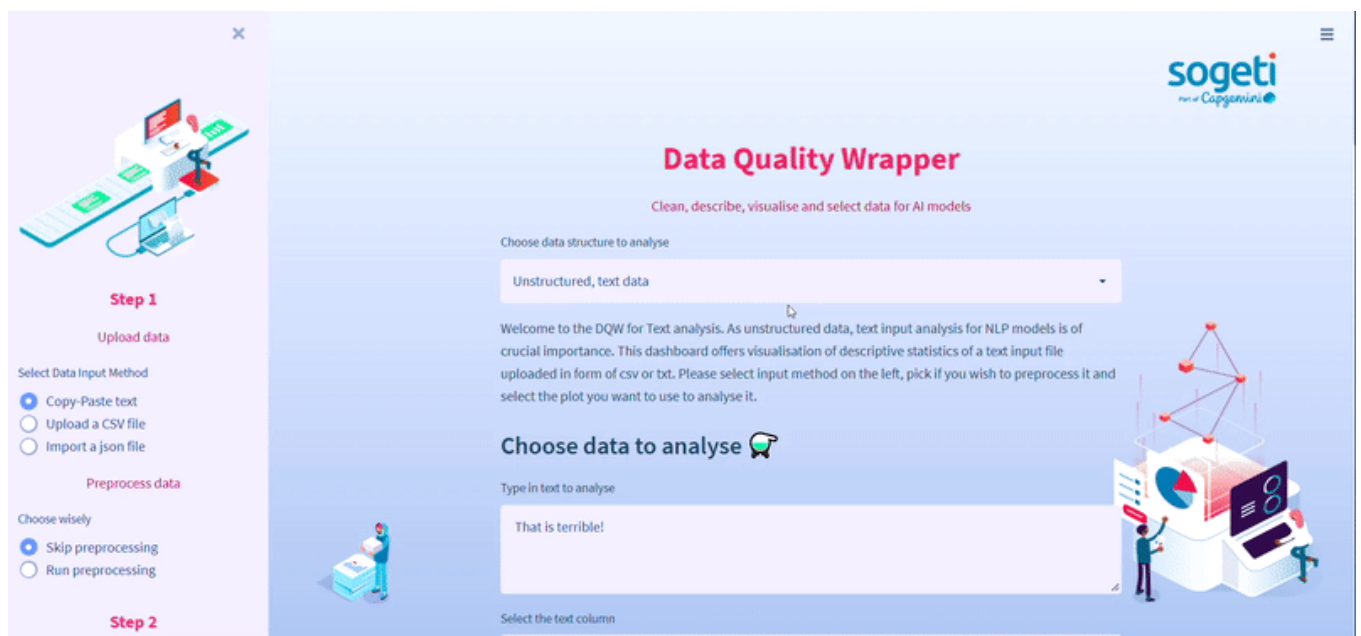
🔍 The code used is in the [preprocessor.py](#) script.

**Topic analysis with LDA**, where we offer the flexibility of providing the number of topics you want to run or calculating the optimal number of topics based on the `u_mass` coherence score. Furthermore, LDA topics are visualized in an interactive plot using [pyLDavis](#).



The code used is in the [lda.py](#) script.

**Sentiment analysis** with [Vader](#) and [textblob](#). An easy way to get the polarity of input text data.




The code is in the [polarity.py](#) script.

## Audio data section

The audio data section offers data augmentation, EDA, and comparison of two audio files. The code is placed in [audio\\_eda](#) folder.

Let's focus on the most interesting subsections.

**One file analysis** where we provide several useful plots with [librosa](#) to describe the input audio file. The plot descriptions are in the app.


 The code is in the [audio\\_data.py](#) script, function [audio\\_eda](#). To upload and display the audio file widget, you can use the below code.

```
1  import streamlit as st
2
3  audio_file = st.sidebar.file_uploader(label="",
4  type=[".wav", ".wave", ".flac", ".mp3", ".ogg"])
5
6  st.audio(audio_file , format="audio/wav", start_time=0)
```

st\_audioplayer.py hosted with ❤ by GitHub

[view raw](#)

**One file augmentation** with [audiomentations](#), a useful library for the augmentation of audio files. Augmentation of audio files is very important for increasing the robustness of the dataset in case of a lack of training data. The app also runs EDA on the augmented file.

 The code is in the [audio\\_data.py](#) script, function [augment\\_audio](#). An interesting approach is used to pass the selected augmentation methods to this function with the multiselect API, parsing the user input as expression arguments and evaluating them as a python expression.

```

1  import streamlit as st
2  from audiomentations import Compose, AddGaussianNoise, TimeStretch, PitchShift, Shift
3
4  audio_file = st.sidebar.file_uploader(label="",
5  type=[".wav", ".wave", ".flac", ".mp3", ".ogg"])
6
7  augmentation_methods = st.multiselect('Select augmentation method:',
8  ['AddGaussianNoise',
9  'TimeStretch',
10 'PitchShift',
11 'Shift'])
12
13 # add p values to each method and eval parse all list elements
14 # so they are pushed to global environment as audiomentations methods
15 augmentation_list = [i + "(p=1.0)" for i in augmentation_methods]
16 augmentation_final = [eval(i) for i in augmentation_list]
17
18 # pass the list to augmentation
19 augment = Compose(augmentation_list)

```

st\_dqw\_aug.py hosted with ❤ by GitHub

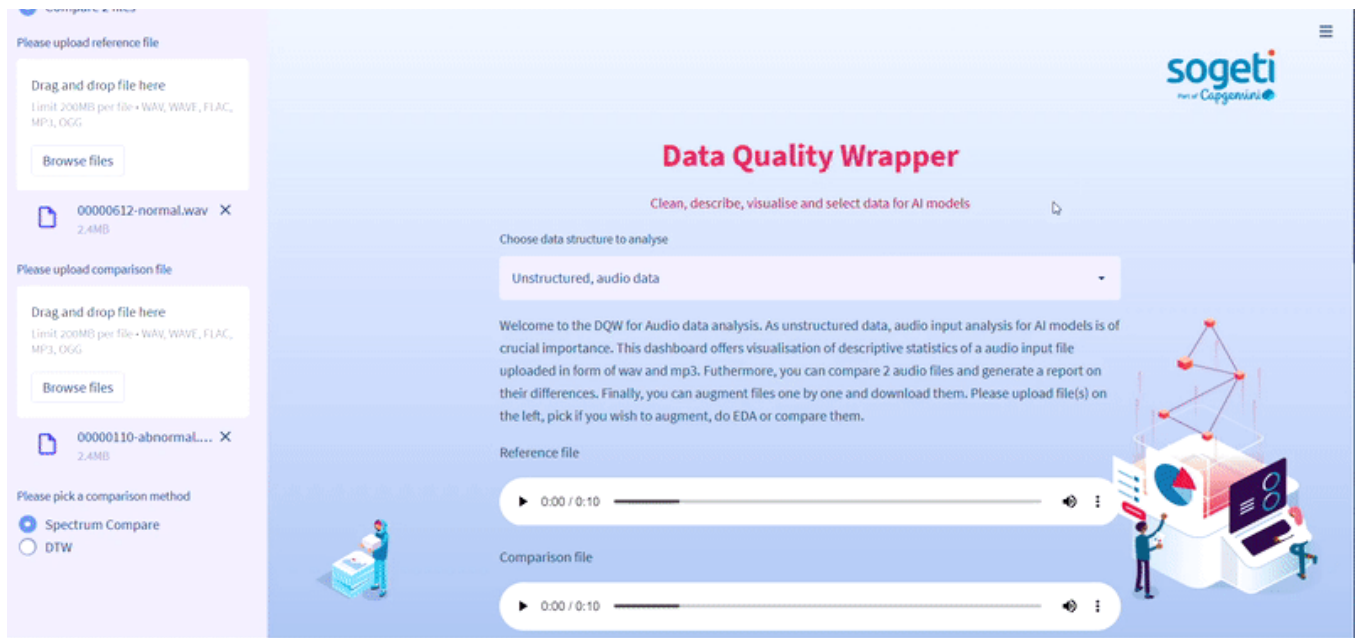
[view raw](#)

**Two file comparison with Dynamic Time Warping (DTW)**, a method of analyzing the maximum path to similarity of 2 sequences.

🔍 The code is in the [audio\\_data.py](#) script, function [compare\\_files](#).

**Two file comparison with audio analyser**, a method that compares 2 spectrums with an applied threshold.

🔍 The code is in the [audio\\_data.py](#) script, function [audio\\_compare](#).

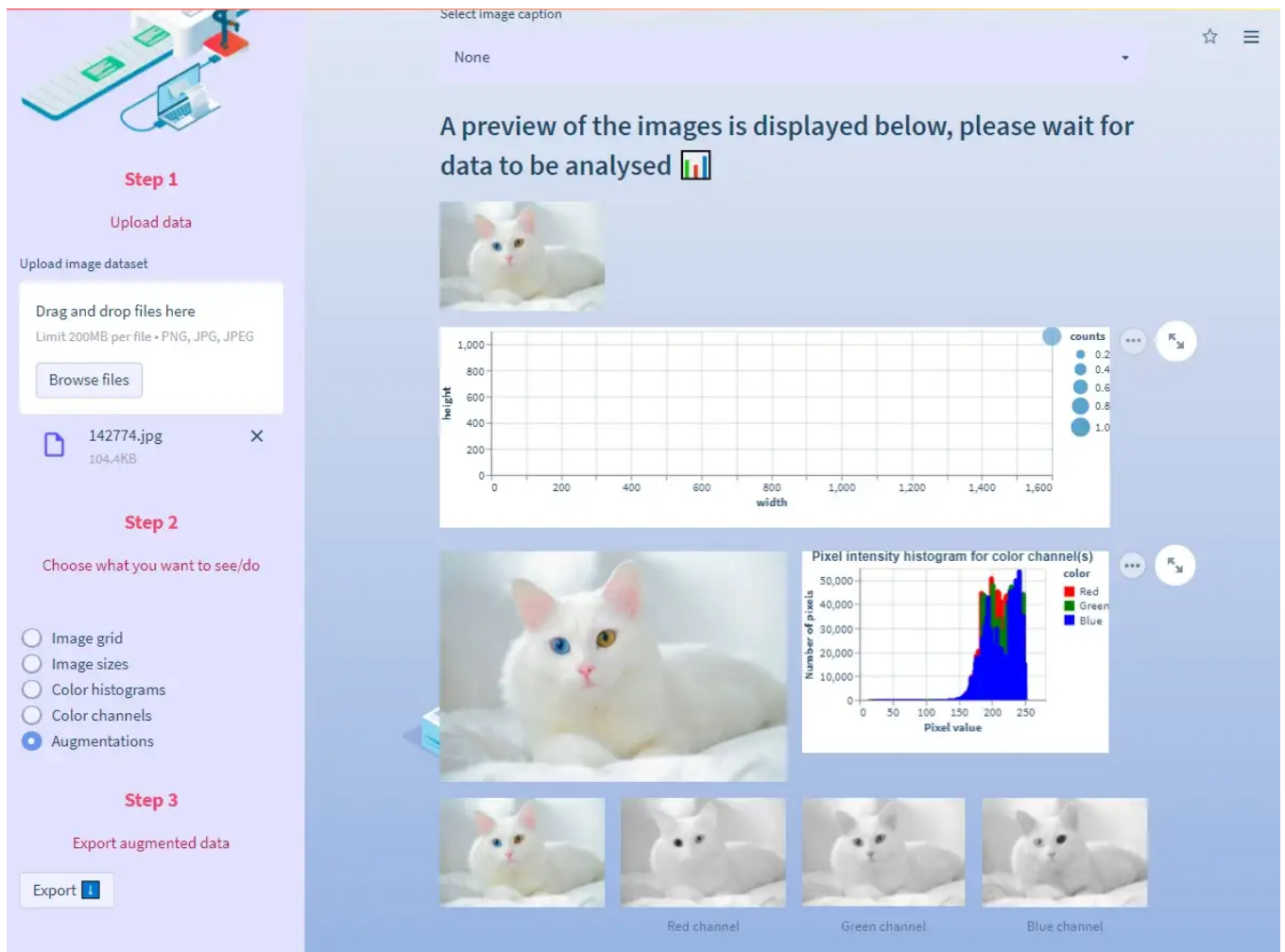


## Image data section

The image data section offers data augmentation and EDA of your images. The code is placed in **image\_eda** folder.

This section allows multiple files to be uploaded and worked on at once. Let's focus on the most interesting subsection.

**Augmentation** of the images using Pillow — the app offers several augmentation methods, including image resizing, applying noise, contrast, and brightness adjustment. A nice added flexibility thanks to Streamlit session state is that you can apply multiple augmentations in sequence and go back to the previous state if you need to.



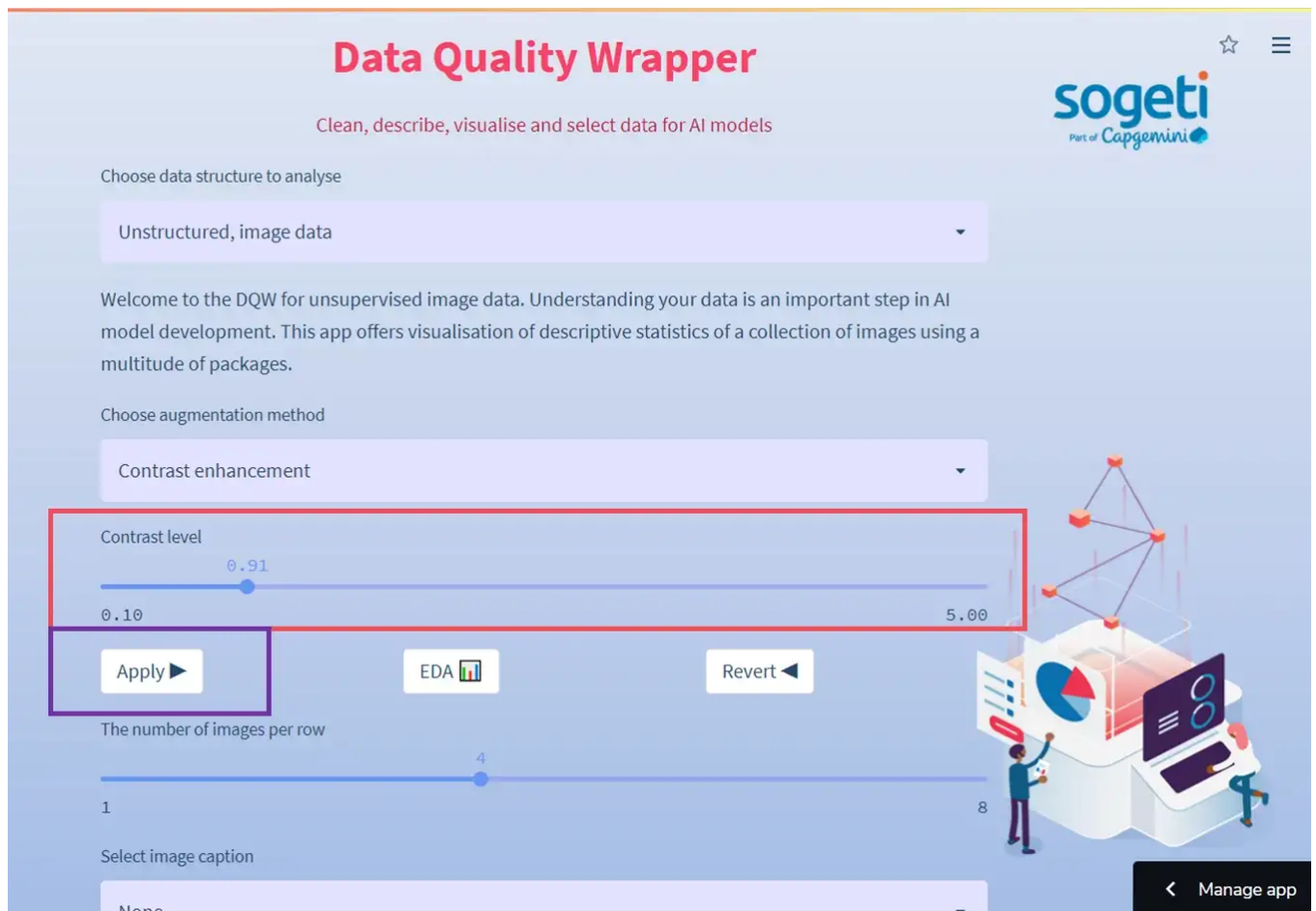
🔍 The code is in the [augment.py](#) script. The versatile Streamlit APIs helped configure the augmentation part of this section. You can see example code below.

```

1  import streamlit as st
2
3  # the red block is st.slider
4  value = st.slider('Contrast level', min_value=0.1, max_value=5., value=0.5)
5
6  # the purple block is st.button
7  # the buttons are ordered with st column
8  col1, col2, col3 = st.columns([1, 1, 1])
9
10 apply = 'Apply ▶'
11 revert = 'Revert ◀'
12 eda = 'EDA 📊'
13
14 with col1:
15     is_applied = st.button(apply)
16 with col2:
17     is_eda = st.button(eda)
18 with col3:
19     revert = st.button(revert)

```

st\_cols.py hosted with ❤ by GitHub

[view raw](#)



## Additional tips for Streamlit app design

Personally, I find the app design very important. It gets the audience and users excited straight away so I don't mind spending a lot of time on it. To those that feel app design takes too much of their time, here are a few tricks I use for all my Streamlit apps:

- Using local file as background hack. This is a lifesaver (I got the tip from the useful [Streamlit forum](#)). Simply use the following code to pass a local file and open it as background.

```

1  def set_bg_hack(main_bg):
2      '''
3          A function to unpack an image from root folder and set as bg.
4
5          Returns
6          -----
7          The background.
8      '''
9      # set bg name
10     main_bg_ext = "png"
11
12     st.markdown(
13         f"""
14         <style>
15         .stApp {{
16             background: url(data:image/{main_bg_ext};base64,{base64.b64encode(open(main_bg, "rb").read())});
17             background-size: cover
18         }}
19         </style>
20         """,
21         unsafe_allow_html=True
22     )

```

st\_bg.py hosted with ❤ by GitHub

[view raw](#)

- Custom themes. Streamlit offers a very easy way of secondary app styling through their UI. [You can check it out here](#).
- The sidebar design. Did you know you can change the width of your sidebar? You can do it with this simple snippet of code. You can adjust the width to what suits you.



```

1  # set sidebar width
2  st.markdown(
3      """
4      <style>
5      [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {
6          width: 300px;
7      }
8      [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {
9          width: 300px;
10         margin-left: -300px;
11     }
12     </style>
13     """,
14     unsafe_allow_html=True,
15 )

```

st\_sidebar.py hosted with ❤ by GitHub

[view raw](#)

- I prefer to move all of the high-level user-defined steps to the sidebar, like upload widgets, select boxes, etc. It's very simple. Just add **.sidebar** to the relevant API.

```

1  import streamlit as st
2
3  # add a logo to the sidebar
4  logo = Image.open("logo.png")
5  st.sidebar.image(logo, use_column_width=True)
6
7  # upload widget
8  file = st.sidebar.file_uploader("Upload file here")
9
10 # selectbox
11 add_selectbox = st.sidebar.selectbox(
12     "What would you like to do?",
13     ("EDA", "Preprocess", "Report")
14 )

```

sidebar.py hosted with ❤ by GitHub

[view raw](#)

- Styling the text with HTML. You can use this function to do that.

```

1  def sub_text(text):
2      '''
3      A function to neatly display text in app.
4      Parameters
5      -----
6      text : Just plain text.
7      Returns
8      -----
9      Text defined by html5 code below.
10     '''
11
12     html_temp = f"""
13     <p style = "color:#1F4E79; text_align:justify;"> {text} </p>
14     </div>
15     """
16
17     st.markdown(html_temp, unsafe_allow_html = True)

```

st\_subtxt.py hosted with ❤ by GitHub

[view raw](#)

- Streamlit expander API for more information and references. The expanders are a space-saver in robust apps with a lot of text.

```

1  import streamlit as st
2
3  info = st.expander("Click here for more info on methods used")
4  with info:
5      st.markdown("More information")

```

st\_expander.py hosted with ❤ by GitHub

[view raw](#)

 You can find a lot of helpful design functions that are used in the app in the [helper\\_functions.py](#) script.

## Wrapping up

The DQW is a useful app to automate your data preprocessing during AI model development. It enables data-driven model development and streamlines the data preprocessing workflow, ensuring transparency and quality. This app is still under development and is one of many Streamlit apps the Sogeti NL Data Science team has developed. We find Streamlit very useful for demonstration purposes. Furthermore, the reason we made this app open source is to educate the data science community about data-centric model development and provide advice on the methods that can be used to ensure data quality.

I encourage you to try the app out and let me know how you experienced it. Leave your questions in the comments or reach out to me if you want to hear more about what we do at Sogeti!

## Further reading

### **A Step by Step Guide to Generate Tabular Synthetic Dataset with GANs**

Goal

medium.com

### **NLP Preprocessing Pipeline — what, when, why?**

This article is part on a series that aims to clarify the most important details of NLP. You can refer to the main...

medium.com

### **A Data Scientist's Approach to Visual Audio Comparison**

In this article, I demonstrate some custom ways I created to visually compare the frequency domains of multiple audio...

towardsdatascience.com

### **EDA for Image Classification**

Why do we overthink EDA for image classification? Let's go back to basics.

medium.com

Data Centric Ai

High Quality Data

Mlops

Ethics

Exploratory Data Analysis