## Introduction, Purpose, goal of the project

Welcome to the MBTI Personality Prediction Project! We had a lot of fun on this project and hope you have as much fun with it. The purpose of our project was not only to create this program, but it was to learn how to function in a team environment and partake in different steps of the Software Development Cycle.

The Goal of our project was to develop an application that uses machine learning to produce an prediction of someone's personality type, based on a given string of words. Our specifics were

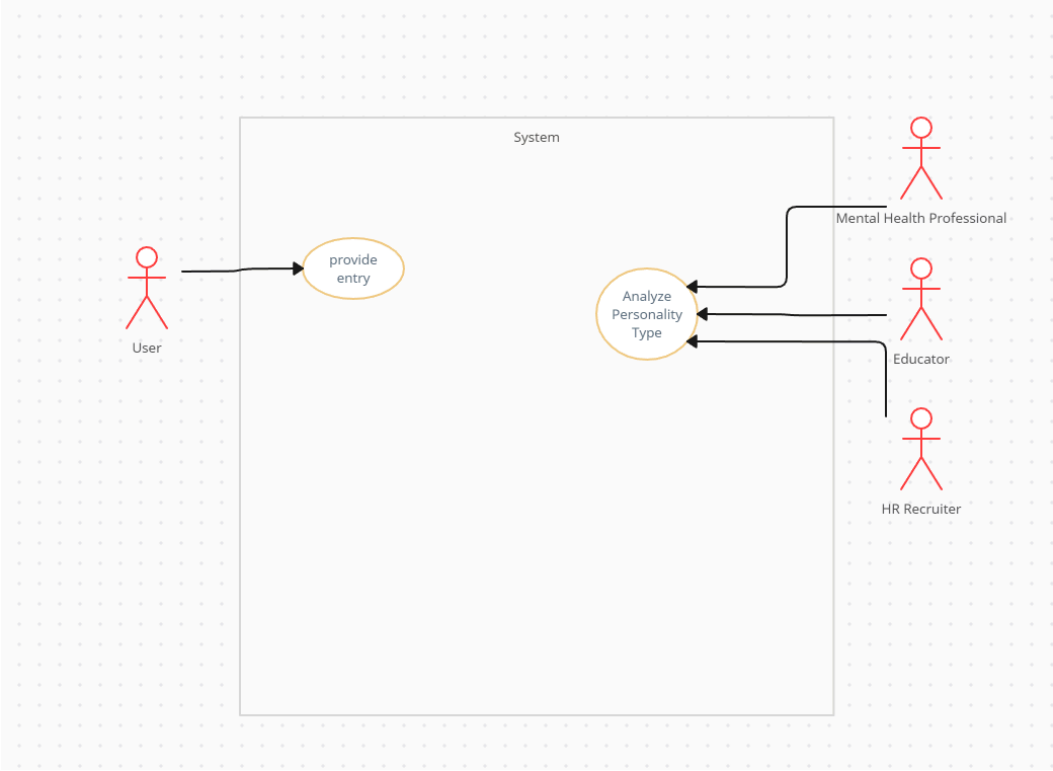
1. Select a dataset. Kaggle.com offers numerous datasets suitable for AI projects, but you're also welcome to pick one independently.
2. There are numerous AI NLP models available for personality prediction.
3. Begin with one model to initiate the project, with the option to expand later if time permits.
4. Develop a simple GUI-based application to display the output.
5. If the GUI isn't feasible, you can alternatively explore training the dataset on four different NLP models and compare their performance to determine the best one.

## Project Glossary

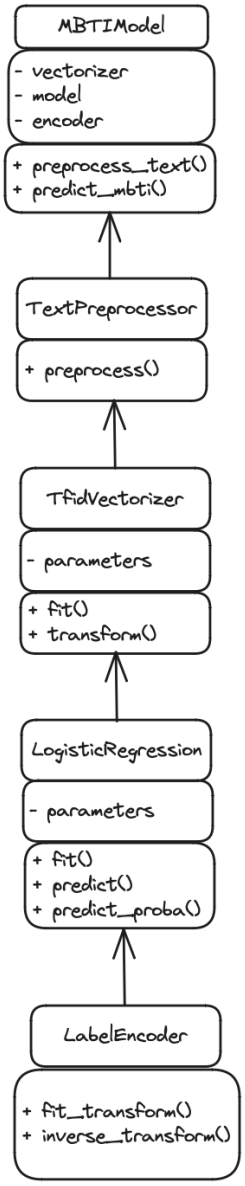
1. pandas: A library used for data manipulation and analysis. It provides data structures like DataFrame for working with structured data.
2. appJar: A GUI library in Python for creating desktop applications with a simple and easy-to-use interface.
3. tkinter: The standard GUI toolkit used for creating graphical user interfaces.
4. LogisticRegression: A machine learning algorithm used for binary classification tasks. It models the probability of a certain class or event occurring.
5. LabelEncoder: A utility class in scikit-learn used for encoding categorical target labels with numerical values.
6. TfidfVectorizer: A method in scikit-learn for converting a collection of raw documents into a matrix of TF-IDF features, which are used for text-based modeling.
7. tqdm: A library that provides a fast, extensible progress bar for loops and iterables.
8. nltk: Natural Language Toolkit, a platform for building programs to work with human language data.
9. WordNetLemmatizer: A tool in nltk for reducing words to their base or root form (lemmas).

10. stopwords: Common words (like "and", "the", "is") that are filtered out before processing natural language data.
11. re: The regular expression module for working with text patterns.
12. train_test_split: A function from scikit-learn for splitting data into training and testing sets for model validation.
13. accuracy_score: A function from scikit-learn for evaluating the accuracy of a classification model.
14. os: A module for interacting with the operating system, providing functions to work with files and directories.

## Use Case Diagram




## Class model



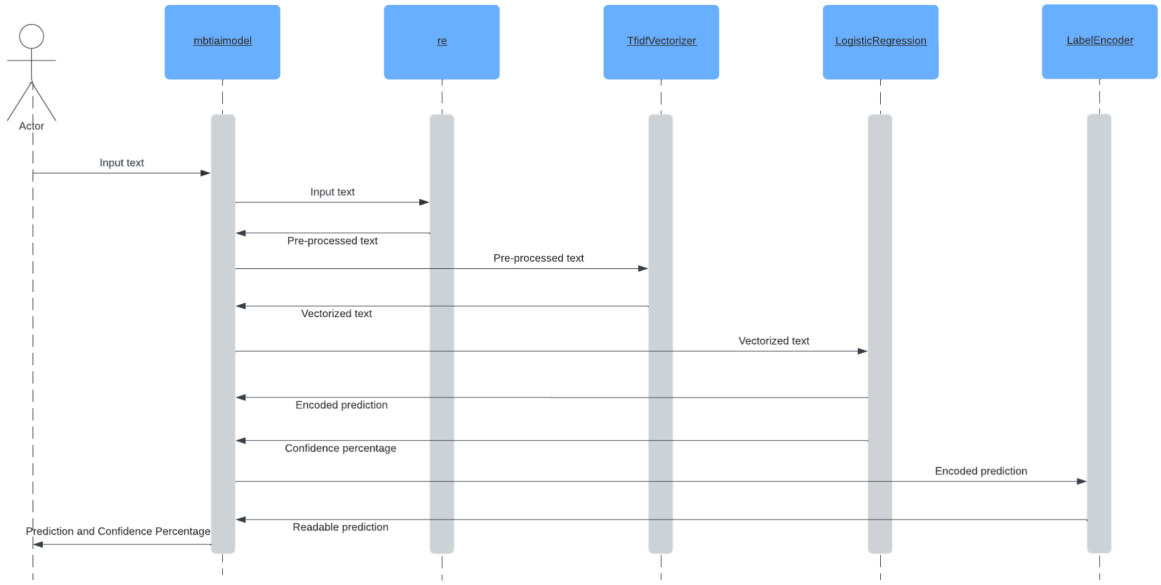

In this class diagram:

- The `MBTIModel` class is the main class that coordinates the prediction process. It has a composition relationship with the `TfidfVectorizer`, `LogisticRegression`, and `LabelEncoder` classes.
- The `TextPreprocessor` class is responsible for preprocessing the input text. It has an association relationship with the `MBTIModel` class.
- The `TfidfVectorizer` class is used for text vectorization, and it has an association relationship with the `MBTIModel`class.

- The `LogisticRegression` class is the machine learning model used for prediction, and it has an association relationship with the `MBTIModel` class.
- The `LabelEncoder` class is used for encoding and decoding class labels, and it has an association relationship with the `MBTIModel` class.

The boxes represent the classes, with the class name at the top, followed by the attributes (prefixed with `-`) and methods (prefixed with `+`). The lines between the classes represent the relationships, with solid lines indicating composition (strong ownership) and dashed lines indicating association (loose coupling).

## Sequence Model

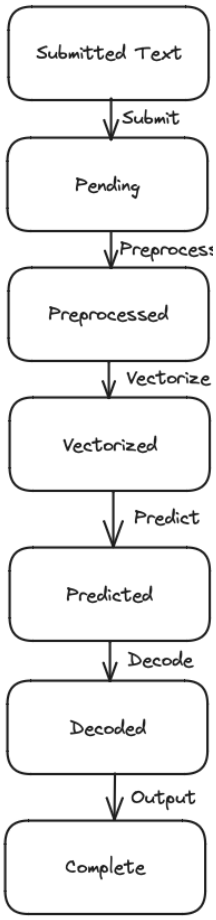



The sequence diagram illustrates the interaction between different components of the MBTI (Myers-Briggs Type Indicator) personality prediction system. The primary actor is the user who provides input text to the system. The following steps occur:

1. The user enters the input text.
2. The `re` (regular expression) module preprocesses the input text by converting it to lowercase and removing punctuation marks.
3. The preprocessed text is passed to the `TfidfVectorizer` module, which converts the text into a numerical vector representation.
4. The vectorized text is then passed to the `LogisticRegression` module, which is a machine learning model trained on the MBTI dataset.
5. The `LogisticRegression` module generates an encoded prediction and a confidence percentage for the predicted MBTI type.

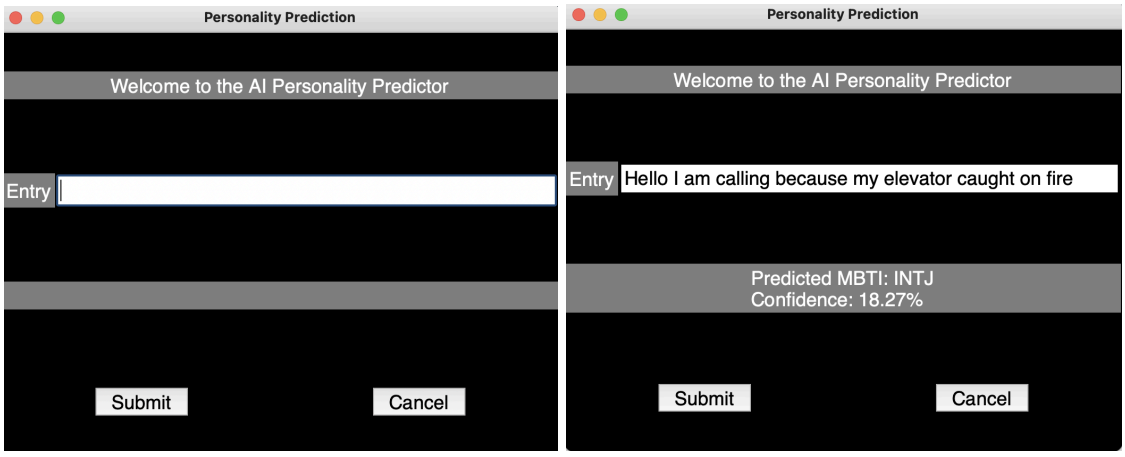
6. The `LabelEncoder` module decodes the encoded prediction into a readable MBTI personality type.
7. The predicted MBTI personality type and the confidence percentage are returned to the user.

## state model (..etc)




The State Model describes the different states of the system and the transitions between them during the MBTI personality prediction process. It starts with the user submitting the input text, followed by preprocessing, vectorization, prediction, decoding, and finally outputting the result. Each state represents a specific stage in the prediction process, and the transitions between states are triggered by the corresponding actions (e.g., preprocess, vectorize, predict, decode, output).

## Demo Screenshots




Here's a breakdown of the key activities involved in the data processing and model development:

1. **Data Loading**: The code loads the MBTI dataset from a CSV file located in the specified directory.
2. **Data Splitting**: The dataset is split into training and testing sets using the `train_test_split` function from scikit-learn. The training set comprises 80% of the data, while the testing set comprises the remaining 20%.
3. **Text Preprocessing**:
    - The `clear_text` function is used to preprocess the text data by performing the following operations:
        - Converting the text to lowercase
        - Removing URLs and web links
        - Removing non-alphanumeric characters
    - The preprocessed text data is stored in the `train_data.posts` and `test_data.posts` columns for the training and testing sets, respectively.
4. **Tokenization and Lemmatization**:
    - The `Lemmatizer` class is defined, which uses the `WordNetLemmatizer` from the NLTK library to perform lemmatization (reducing words to their base form) on the tokenized text.
    - The `TfidfVectorizer` from scikit-learn is used to convert the preprocessed text data into numerical feature vectors. The `Lemmatizer` class is used as the tokenizer for the vectorizer.

5. **Label Encoding**:
    - The `LabelEncoder` from scikit-learn is used to convert the categorical MBTI personality types into numerical labels for the training and testing sets.
6. **Model Training**:
    - A Logistic Regression model from scikit-learn is initialized with specific hyperparameters (`max_iter`, `C`, and `n_jobs`).
    - The model is trained on the vectorized training data (`train_post`) and the corresponding encoded labels (`train_target`).
7. **Graphical User Interface (GUI)**:
    - The code uses the `appJar` library to create a simple GUI application for the MBTI personality prediction.
    - The GUI has an input field (`Entry`) where the user can enter text, and a label (`Result`) to display the predicted MBTI type and the associated confidence score.
    - The `predict_mbti` function is defined to preprocess the input text, vectorize it using the trained `TfidfVectorizer`, and make predictions using the trained Logistic Regression model.
    - The `press` function handles button events (Submit and Cancel) and displays the prediction result in the GUI.
8. **Model Inference**:
    - When the user enters text and clicks the "Submit" button, the `predict_mbti` function is called to make predictions on the input text.
    - The predicted MBTI type and the associated confidence score are displayed in the GUI.

## Testing Report

| Test Case ID | Test Case | Pre Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| MBTI Test01 | Enter nothing into the text box | Program needs to be running | Open program<br>Click text box<br>Click submit | <SubmitData> | Nothing shows up | Nothing shows up | Pass |

| Test Case ID | Test Case | Pre Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| MBTI Test02 | Paste MBTI training data into text box with type INFJ | Program needs to be running | Open program<br>Click text box and paste INFJ training text<br>Click submit | <SubmitData><br><ReadData> | Displays INFJ | Displays INFJ | Pass |

| Test Case ID | Test Case | Pre Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| MBTI Test03 | Paste MBTI training data into text box with type ENTP | Program needs to be running | Open program<br>Click text box and paste ENTP training text<br>Click submit | <SubmitData><br><ReadData> | Displays ENTP | Displays ENTP | Pass |

| Test Case ID | Test Case | Pre Condition | Test Steps | Test Data | Expected Result | Actual Result | Status Pass/Fail |
|---|---|---|---|---|---|---|---|
| MBTI Test04 | Paste MBTI training data into text box with rare type ESFJ | Program needs to be running | Open program<br>Click text box and paste ENTP training text<br>Click submit | <SubmitData><br><ReadData> | Displays ESFJ | Displays ENTP | Fail<br>Due to not many training text's with this archetype |

Some important design models and practices that can be observed from the test cases are:

1. Test-Driven Development (TDD): The test cases were written before implementing the actual functionality, which aligns with the principles of Test-Driven Development. TDD

ensures that the code is testable and meets the specified requirements from the beginning.
2. Unit Testing: The test cases represent unit tests for the MBTI prediction model. Unit tests are essential for verifying the correctness of individual components (in this case, the text classification model) and ensuring that they work as expected under various conditions.
3. Test Case Design: The test cases cover different scenarios, including: a. Empty input (Test Case 01) b. Valid input with expected output (Test Case 02 and 03) c. Invalid input with unexpected output (Test Case 04)

This diversity in test cases helps in thoroughly testing the model's behavior and identifying potential edge cases or issues.

4. Data-Driven Testing: The test cases seem to be using predefined test data, such as "SubmitData" and "ReadData", to simulate different input scenarios. This approach is known as data-driven testing, where test cases are driven by input data sets rather than being hard-coded.
5. Acceptance Testing: Although not explicitly mentioned, the test cases could be considered a form of acceptance testing, where the expected results are defined, and the actual results are compared to ensure that the model meets the specified requirements.

While the provided test cases focus on the functional aspects of the MBTI prediction model, other design models and practices could also be employed in the project, such as:

- Model Architecture: The choice of machine learning model architecture (e.g., neural networks, logistic regression, decision trees) and its configuration can significantly impact the model's performance and accuracy.
- Data Preprocessing: Techniques like text cleaning, tokenization, and feature extraction may be employed to preprocess the input text data before feeding it to the model.
- Model Training: Strategies for training the model, such as data splitting, cross-validation, hyperparameter tuning, and regularization, can be crucial for achieving optimal performance.
- Model Evaluation: Metrics like accuracy, precision, recall, and F1-score can be used to evaluate the model's performance and guide further improvements or adjustments.

## Tools

Kaggle: machine learning platform that distributes datasets and teaches users how to use machine learning

Google Colab: Useful to display the many steps in training a model on a dataset

VS Code: Copied from Google Colab to VS code for Compilation.