**Documentation**

**Poverty Analyzer**

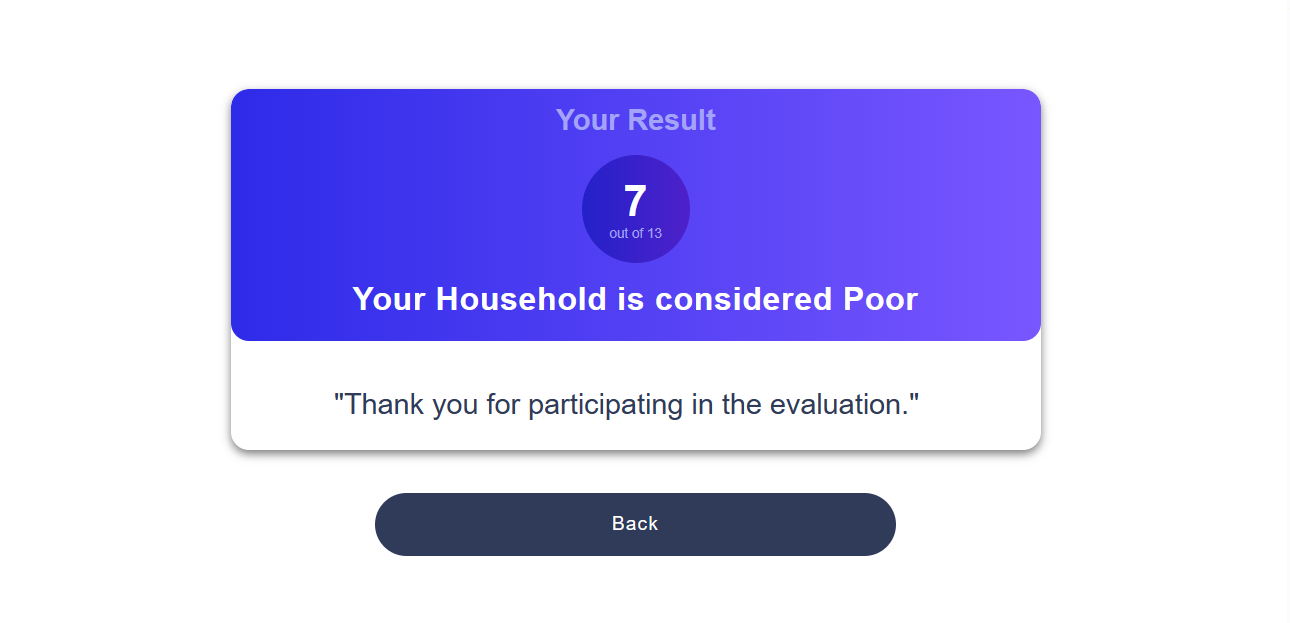


**Submitted by:**

Carl Michael C. De Taza

Sue C. Sinangote

David Frank Edison Regodos

****



**Prompt:** Introduction

The Poverty Analyzer, an innovative web-based application developed using the Django framework in Python, coupled with the joblib library to integrate a machine learning model. This user-friendly platform is specifically designed to assist individuals in assessing their economic status through a straightforward evaluation process.

Users are required to fill out a form with thirteen questions, each of which is a significant indicator for the machine learning model that is being used. Driven by decision tree algorithms, this model examines the data that the user submits, examining different aspects of their socioeconomic situation to determine whether they qualify as "poor."

Explore the Poverty Analyzer, and embark on a journey towards a clearer understanding of your economic standing. Our user-centric design ensures that the complexities of the model are transparent, making it accessible to all. This application guarantees a smooth experience for users with its simple design and ease of use, making it easy for them to quickly and accurately analyze their economic condition.

**Processing Workflow**

The fundamental functionality of the Poverty Analyzer application is supported by a carefully arranged sequence of steps, designed to provide users with helpful assessments of their economic status. As users engage with the application, the following workflow unfolds:

* **Form Submission (Django HTML Form):** Users are presented with a user-friendly HTML form featuring 13 questions, designed to capture essential information about their household. Upon submitting the form, the entered data is sent to the Django backend for processing.
* **Data Processing (Django Views):** Django views handle the submitted form data, ensuring its integrity and validity. The extracted household information is then passed to the machine learning model for evaluation.
* **Machine Learning Model Prediction (Joblib Integration):** Using the joblib library, the application calls a pre-trained machine learning model to predict whether the user's household falls within the 'poor' or 'not poor' category. The model uses the 13 questions as indicators to make assessment.
* **Result Display (Result HTML):** Based on the model's prediction, the application dynamically generates a result page (result.html). If the prediction indicates 'poor,' the result page communicates this outcome to the user. Conversely, if the prediction is 'not poor,' a different message is displayed, offering a clear determination of the household's economic status.

**Usage of the Application**

The Poverty Analyzer application is designed to empower users by providing a quick and accessible means of evaluating their economic status. The usage of the application involves the following steps:

* **Access the Application:** Users can access the Poverty Analyzer through a web browser by navigating to the application's web address.
* **View the Result:** Based on the model's prediction, the user is redirected to a result page (result.html). This page communicates the outcome, indicating whether the household is considered 'poor' or 'not poor.'
* **Actionable Insights:** Armed with the result, users gain valuable insights into their economic standing. This information can be used to make informed decisions, seek relevant assistance, or take proactive steps to improve their financial situation.



**Prompt:** Advantage and Disadvantage

**Advantages of the Poverty Analyzer Applications:**

* **Accessibility and Ease of Use:** The application features a simple and user-friendly interface, making it accessible to a wide range of users, including those with limited technical expertise.
* **Time Efficiency:** The automated nature of the application allows for swift processing of user-submitted data and prompt generation of results, saving users time compared to manual evaluation methods.
* **Empowerment through Insights:** By leveraging machine learning algorithms, the Poverty Analyzer provides users with valuable insights into their economic status, enabling informed decision-making and proactive financial planning.

**Disadvantages of the Poverty Analyzer Applications:**

* **Dependency on Training Data:** The accuracy of the machine learning model relies heavily on the quality and representativeness of the training data. If the training data is biased or incomplete, it may impact the accuracy of predictions.
* **Inherent Uncertainty of Predictions:** Machine learning models inherently have a degree of uncertainty. Users should be made aware that predictions are probabilistic and not definitive, allowing for a margin of error.
* **Limited Scope:** The application's effectiveness is constrained by the factors considered in the evaluation form. It may not capture the entirety of a user's financial situation, potentially leading to oversimplified assessments.



**Prompt:** Recommendations

**Regular Model Updates:** Schedule periodic updates to the machine learning model based on new data and insights. This ensures that the model stays relevant and adapts to changing socio-economic trends.

**Security and Privacy Measures:** Prioritize the implementation of robust security measures to protect user data. Clearly communicate the application's privacy policy, assuring users that their information is handled with care and confidentiality.

**Cross-Validation and Testing:** Regularly perform cross-validation and testing of the machine learning model to assess its performance and identify areas for improvement. This includes testing with diverse datasets to ensure the model generalizes well.



**Prompt:** System Requirements

**Hardware Requirements:**

* **Processor:** Dual-core processor (e.g., Intel Core i3 or AMD equivalent) or higher.
* **Memory (RAM):** 4 GB of RAM or higher for efficient data processing and machine learning model execution.
* **Storage:** 1 GB or more of available storage space for storing application files, user data, and potential model updates.

**Software Requirements:**

|  |  |
| --- | --- |
| **Name** | **Descriptions** |
| **Python 3.x** | Python 3.x, the latest version, brings key improvements like Unicode support, enhanced syntax, and asyncio. It's actively maintained, secure, and backward incompatible with Python 2.x. Widely used in web development, AI, and more, Python 3.x is the recommended choice for new projects. |
| **Django Framework** | The application is built using the Django web framework. Ensure compatibility with the required Django version and related packages. |
| **Joblib Library** | The application utilizes the joblib library for handling machine learning model functionalities. Ensure compatibility with the required version of joblib. |
| **imbalanced-learn library** | The imbalanced-learn library is a Python library designed to address the issue of imbalanced datasets in machine learning. Imbalanced datasets occur when the distribution of classes in the dataset is not uniform, meaning that one class has significantly fewer examples than the others. This imbalance can lead to biased models that perform poorly on the minority class. |
| **scikit-learn library** | The scikit-learn library is a widely used machine learning library in Python that provides simple and efficient tools for data analysis and modeling. It is built on top of other popular scientific computing libraries like NumPy, SciPy, and matplotlib. |



**Prompt:** Installation

**Python**

Ensure that Python 3.x is installed on your system. You can download it from the official Python website: [Python Downloads](<https://www.python.org/downloads/>)

**Create a virtual environment(optional)**

* **Open Command Prompt:**

Open the Command Prompt on your Windows desktop.

* **Navigate to the Desired Directory:**

Use the **cd** command to navigate to the directory where you want to create the virtual environment. For example:



* **Create Virtual Environment:**

Run the following command to create a virtual environment named **env**:

****

* **Activate Virtual Environment:**

To activate the virtual environment, run the appropriate activation script. In the Command Prompt:****

**Clone Repository from Github**

* **Get the Repository URL:**

On the GitHub repository page, click on the "Code" button. Make sure to select the "HTTPS" option. Copy the repository URL provided: <https://github.com/Chael07/drill1_PovertyAnalyzer.git>

* **Open Terminal Command Prompt (Windows):**

Open a terminal or command prompt on your local machine.

* **Navigate to the Directory Where You Want to Clone the Repository:**

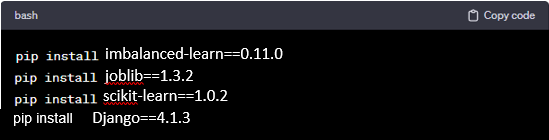
Use the cd command to navigate to the directory where you want to store the cloned repository. For example:****

* **Clone the Repository:**

Use the git clone command followed by the repository URL. Replace <https://github.com/Chael07/drill1_PovertyAnalyzer.git> with the URL you copied in step 1.

**Required Python Packages**

Install the required Python packages in same directory using the following commands:





**Prompt:** Application Overview

The Poverty Analyzer is a web-based application designed to assess and analyze the economic status of users based on a set of 13 questions. Users can input information through a simple and user-friendly form, and the application processes the data using a machine learning model to determine whether the user's household is considered 'poor' or 'not poor.'

**Features**

* **User-Friendly Form:** The application features a straightforward HTML form with 13 questions designed as indicators for the machine learning model.
* **Result Presentation:** Provides users with a result page indicating whether their household is classified as 'poor' or 'not poor' based on the model's prediction.
* **Machine Learning Model:** Utilizes the Django framework and integrates the joblib library to implement a machine learning model for predicting economic status.

**Code Stucture**

**views.py:** This file (views.py) contains the code that defines how the web application interacts with the user. It includes functions for processing user input, loading a machine learning model, making predictions, and rendering HTML pages.

* **convert\_to\_one\_zero:** A function that converts numerical values to '1', '0', or 'none'.
* **home\_screen\_view:** Handles user interactions when they submit a form. It extracts data from the form, loads a machine learning model, performs predictions, and displays the result on an HTML page (result.html).
* **result\_screen\_view:** Displays a result page. Currently, it doesn't have additional logic.

**urls.py:** This file (urls.py) defines the URLs or web addresses that users can visit to interact with the application. It maps URLs to specific functions in views.py.

* **urlpatterns:** A list of URL patterns, where each pattern is associated with a specific view function.

**HTML Templates:** These are HTML files that define the structure and content of the web pages displayed to users.

* **eval.html:** Contains a form where users input data.
* **result.html:** Displays the result of the analysis, including whether the household is classified as 'Poor' or 'Not Poor'.



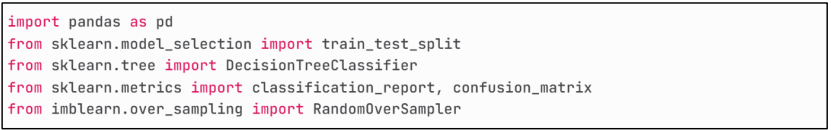
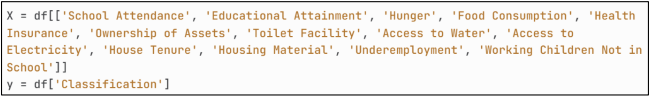
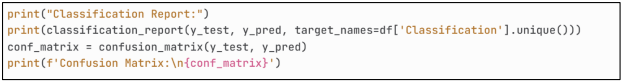
**Prompt:** Conclusion

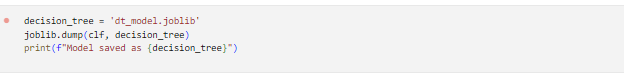
The Image Processing App offers a user-friendly interface for basic image manipulation. By expanding functionality, improving user guidance, and optimizing the code structure, the application can evolve into a more versatile tool for users with diverse image processing needs.



**Prompt:** Source Code Used in the Project

**Decision Tree Model Development**

* Importation of Libraries 
* Loading of Dataset 
* Count Distribution of Classes before OverSampling. 
* Feature Selection and Target Variable 
* Applying RandomOverSampling Method 
* Count Distribution of Classes after Oversampling 
* Split the Resampled Data into Training and Testing Sets 
* Hyperparameter Tuning 
* Prediction on the Test Set 
* Model Evaluation 
* Export the trained model

****

**View.py in Django framework**

from django.shortcuts import render

import os

import joblib

from django.conf import settings

# Create your views here.

def convert\_to\_one\_zero(record):

    if record == 0.076923077:

        return '1'

    elif record == 0.0:

        return '0'

    else:

        return 'none'

def home\_screen\_view(request):

    if request.method == 'POST':

        q1 = float(request.POST.get('q1', 0))

        q2 = float(request.POST.get('q2', 0))

        q3 = float(request.POST.get('q3', 0))

        q4 = float(request.POST.get('q4', 0))

        q5 = float(request.POST.get('q5', 0))

        q6 = float(request.POST.get('q6', 0))

        q7 = float(request.POST.get('q7', 0))

        q8 = float(request.POST.get('q8', 0))

        q9 = float(request.POST.get('q9', 0))

        q10 = float(request.POST.get('q10', 0))

        q11 = float(request.POST.get('q11', 0))

        q12 = float(request.POST.get('q12', 0))

        q13 = float(request.POST.get('q13', 0))

        questions = [q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13]

        converted\_questions = []

        for record in questions:

            converted\_value = convert\_to\_one\_zero(record)

            converted\_questions.append(converted\_value)

        count\_yes = 0

        count\_no = 0

        for response in questions:

            converted\_value = convert\_to\_one\_zero(response)

            if converted\_value == '1':

                count\_yes += 1

            elif converted\_value == '0':

                count\_no += 1

        clf\_path = os.path.join(settings.BASE\_DIR, 'polls/dt\_model.joblib')

        clf = joblib.load(clf\_path)

        result\_data = [converted\_questions]

        prediction = clf.predict(result\_data)

        prediction2 = 'Not Poor' if prediction == 'Not Poor' else 'Poor'

        context = {

                'prediction': prediction2,

                'count\_yes': count\_yes,

                'count\_no': count\_no

            }

        return render(request, "result.html", context)

    else:

        return render(request, 'eval.html')

def result\_screen\_view(request):

    print(request.headers)

    return render(request, "result.html", {})

**urls.py in Django framework**

from django.contrib import admin

from django.urls import path

from polls.views import(home\_screen\_view, result\_screen\_view)

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', home\_screen\_view, name='home'),

    path('result/', result\_screen\_view, name='result'),

]

**END**