ARTIFICIAL INTELLIGENCE

Project Report

"Global Dengue Surveillance Dataset (1924-2023)"

In conjunction with definitions that are derived from the case_definition_original field and still with the primary variable being dengue_total, the OpenDengue dataset incorporates cases of dengue fever that are reported together with all severities and confirmation methods. Like terms regarding the place should be in accordance with standard codes (FAO GAUL, ISO, etc.), and temporal domains should include starting and ending periods. The only barriers include variation in surveillance and reporting pundits due to asymptomatic patients which might lead to under-reporting cases.

- **Dataset Description and Purpose**:
  - Spanning 1924–2023, this dataset focuses on dengue cases in Afghanistan, capturing geographical (adm_0_name, adm_1_name) and temporal (calendar_start_date, calendar_end_date) data.
  - Supports early prediction of outbreaks to enable timely health interventions using machine learning techniques.
- **Challenges and Limitations**:
  - Issues include missing geographic data, imbalanced records for earlier years, and noisy features (e.g., FAO_GAUL_code).
  - Computational challenges due to dataset size (~383,490 records) and potential overfitting in complex models.
- **Techniques Applied**:
  - Utilized methods like Random Forest, Decision Trees, and PCA for prediction and feature reduction.
  - Addressed challenges with ensemble learning, Genetic Algorithms, and advanced feature engineering.
- **Data Cleaning and Preprocessing**:
  - Dropped irrelevant columns (e.g., IBGE_code, UUID) and converted date fields to datetime format.
  - Handled missing data by filling gaps in adm_1_name and adm_2_name with "Unknown".

1: Existing techniques applied on selected dataset with references (Base paper)

Paper on Current Methods in the Rule-Based Algorithm Base:

Although this is primarily treated as rule fit (for meaning supervised learning), this approach requires the specification of decision rules which in certain versions may correspond to techniques of unsupervised learning such as clustering or association rule learning.

Main techniques of feature selection as well as data transformation:

To aid in data compression, tools such as Principal Component Analysis (PCA) are used. The end result is that PCA is often combined with unsupervised approaches to help cut down the intricacy of the data while retaining its variance level.

Approaches on Clustering as a Data Cleaning Technique:

The study outlines the use of data clustering techniques to locate outliers and subsequent cleaning of the data. Thus, related data points are organized appropriately according to the principles of the underlying unsupervised learning model.

Time Series Analysis:

Although the trend may be reliant on supervised analytics, unsupervised time series clustering or unsupervised trend extraction can also be used to understand the seasonal patterns and/or irregular components.

Copula Models:

The focus in Copula models is not purely on unsupervised learning, but rather on the relationships and dependencies of different variables which might well be suited for clustering and correlation models under unsupervised frameworks.

Dimensionality Reduction:

Unsupervised methods for dimensionality reduction should be used in conjunction with PCA since there are multiple extraneous dimensions in the data which should be eliminated before the main dataset's crucial components are defined.

A Few Further Observations on the Use of Unsupervised Methods

For a dataset such as yours, the article cites some data preprocessing strategies such as clustering, which might be necessary to isolate areas or time points with the same level of dengue cases.

It could be used in conjunction with other techniques, although it is not stated explicitly, that could build upon rule-based reasoning in original papers to deal with repetitive aspects of the dataset and learn new forms.

1. DATA PREPROCESSING:

Before Data preprocessing

A1 adm_0_name

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	adm_0_na	adm_1_na	adm_2_na	full_name	ISO_A0	FAO_GAUI	RNE_iso_c	IBGE_code	calendar_	calendar_	Year	dengue_tc	case_defir	S_res	T_res	UUID
2	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	1/3/2021	1/9/2021	2021	101	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
3	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	151	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
4	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	201	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
5	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	202	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
6	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	2/6/2021	2021	100	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
7	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	2/7/2021	########	2021	251	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
8	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	101	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
9	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	61	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
10	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	3/6/2021	2021	99	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
11	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	3/7/2021	########	2021	112	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
12	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	70	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
13	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	70	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
14	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	4/3/2021	2021	99	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
15	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	4/4/2021	########	2021	99	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
16	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	75	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
17	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	99	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
18	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	5/1/2021	2021	94	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
19	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	5/2/2021	5/8/2021	2021	77	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
20	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	5/9/2021	########	2021	150	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
21	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	101	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
22	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	99	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
23	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	6/5/2021	2021	110	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
24	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	6/6/2021	########	2021	97	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05
25	AFGHANIS	NA	NA	AFGHANIS	AFG	1011446	AFG	NA	########	########	2021	91	Suspected	Admin0	Week	WHOEMRO-ALL-2021-Y01-05

Highest temporal resolution dat

CODE:

```python
import dask.dataframe as dd
import pandas as pd

# Load the large CSV file using Dask
file_path = '/Highest temporal resolution data_AFGHANISTAN_19240120_20230930.csv'
output_file_path = '/Highest_temporal_resolution_data_AFGHANISTAN_preprocessed.csv'

# Read the CSV file
df = dd.read_csv(file_path)

# Step 1: Handle missing values for numeric columns
# We need to compute the median of each numeric column using map_partitions
def fill_missing_with_median(partition):
    # Fill missing values with median in each partition
    for col in partition.select_dtypes(include=['float64', 'int64']).columns:
        partition[col].fillna(partition[col].median(), inplace=True)
    return partition

# Apply the function to each partition
df = df.map_partitions(fill_missing_with_median)

# Step 2: Normalize numerical columns
# Normalize the columns by applying a lambda function to each partition
def normalize_columns(partition):
    numeric_cols = partition.select_dtypes(include=['float64', 'int64']).columns
```

```python
    for col in numeric_cols:
        partition[col] = (partition[col] - partition[col].mean()) / partition[col].std()
    return partition

# Apply the normalization
df = df.map_partitions(normalize_columns)

# Step 3: Filter outliers (if necessary)
# Apply outlier filtering: For this example, we assume outliers are values > threshold (100)
def filter_outliers(partition, threshold=100):
    numeric_cols = partition.select_dtypes(include=['float64', 'int64']).columns
    return partition[(partition[numeric_cols] < threshold).all(axis=1)]

# Apply the outlier filter
df = df.map_partitions(filter_outliers)

# Step 4: Save the preprocessed data to a new CSV file
df.to_csv(output_file_path, index=False, single_file=True)

print(f"Data preprocessing with Dask completed. The cleaned data has been saved to: {output_file_path}")
```

```
saved to: /Highest_temporal_resolution_data_AFGHANISTAN_preprocessed.csv
```

After Data Preprocessing

DATASET:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	adm_0_na	adm_1_na	adm_2_na	full_name	ISO_A0	FAO_GAUI	RNE_iso_c	calendar_s	calendar_e	Year	dengue_tc	case_defir	S_res	T_res	UUID
2	0	33	1630	0	1	3.219866	1	161	158	1.165399	-0.01317	3	0	1	538
3	0	33	1630	0	1	3.219866	1	72	31	1.165399	-0.00302	3	0	1	538
4	0	33	1630	0	1	3.219866	1	102	66	1.165399	0.007136	3	0	1	538
5	0	33	1630	0	1	3.219866	1	136	99	1.165399	0.007339	3	0	1	538
6	0	33	1630	0	1	3.219866	1	174	769	1.165399	-0.01338	3	0	1	538
7	0	33	1630	0	1	3.219866	1	777	661	1.165399	0.01729	3	0	1	538
8	0	33	1630	0	1	3.219866	1	697	692	1.165399	-0.01317	3	0	1	538
9	0	33	1630	0	1	3.219866	1	727	719	1.165399	-0.0213	3	0	1	538
10	0	33	1630	0	1	3.219866	1	754	919	1.165399	-0.01358	3	0	1	538
11	0	33	1630	0	1	3.219866	1	925	802	1.165399	-0.01094	3	0	1	538
12	0	33	1630	0	1	3.219866	1	835	831	1.165399	-0.01947	3	0	1	538
13	0	33	1630	0	1	3.219866	1	867	859	1.165399	-0.01947	3	0	1	538
14	0	33	1630	0	1	3.219866	1	895	1026	1.165399	-0.01358	3	0	1	538
15	0	33	1630	0	1	3.219866	1	1066	939	1.165399	-0.01358	3	0	1	538
16	0	33	1630	0	1	3.219866	1	975	969	1.165399	-0.01845	3	0	1	538
17	0	33	1630	0	1	3.219866	1	1004	1003	1.165399	-0.01358	3	0	1	538
18	0	33	1630	0	1	3.219866	1	1036	1087	1.165399	-0.0146	3	0	1	538
19	0	33	1630	0	1	3.219866	1	1162	1233	1.165399	-0.01805	3	0	1	538
20	0	33	1630	0	1	3.219866	1	1238	1111	1.165399	-0.00322	3	0	1	538
21	0	33	1630	0	1	3.219866	1	1147	1143	1.165399	-0.01317	3	0	1	538
22	0	33	1630	0	1	3.219866	1	1179	1171	1.165399	-0.01358	3	0	1	538
23	0	33	1630	0	1	3.219866	1	1211	1367	1.165399	-0.01135	3	0	1	538
24	0	33	1630	0	1	3.219866	1	1376	1253	1.165399	-0.01399	3	0	1	538
25	0	33	1630	0	1	3.219866	1	1289	1280	1.165399	-0.0152	3	0	1	538

processed_dataset

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
383467	101	33	1630	2875	101	-0.3475	950	1211	1367	1.165399	-0.01135	3	0	1	538	
383468	101	33	1630	2875	101	-0.3475	950	1376	1253	1.165399	-0.01419	3	0	1	538	
383469	101	33	1630	2875	101	-0.3475	950	1289	1280	1.165399	-0.01541	3	0	1	538	
383470	101	33	1630	2875	101	-0.3475	950	1319	1311	1.165399	-0.02313	3	0	1	538	
383471	101	33	1630	2875	101	-0.3475	950	1348	1479	1.165399	-0.02333	3	0	1	538	
383472	101	33	1630	2875	101	-0.3475	950	1523	1391	1.165399	-0.02008	3	0	1	538	
383473	101	33	1630	2875	101	-0.3475	950	1428	1421	1.165399	-0.02313	3	0	1	538	
383474	101	33	1630	2875	101	-0.3475	950	1458	1456	1.165399	-0.028	3	0	1	538	
383475	101	33	1630	2875	101	-0.3475	950	1490	1516	1.165399	-0.02617	3	0	1	538	
383476	101	33	1630	2875	101	-0.3475	950	1575	1688	1.165399	-0.02414	3	0	1	538	
383477	101	33	1630	2875	101	-0.3475	950	1691	1566	1.165399	-0.02434	3	0	1	538	
383478	101	33	1630	2875	101	-0.3475	950	1601	1598	1.165399	-0.02313	3	0	1	538	
383479	101	33	1630	2875	101	-0.3475	950	1633	1627	1.165399	-0.02292	3	0	1	538	
383480	101	33	1630	2875	101	-0.3475	950	1660	1825	1.165399	-0.02252	3	0	1	538	
383481	101	33	1630	2875	101	-0.3475	950	1832	1706	1.165399	-0.02252	3	0	1	538	
383482	101	33	1630	2875	101	-0.3475	950	1741	1733	1.165399	-0.02252	3	0	1	538	
383483	101	33	1630	2875	101	-0.3475	950	1769	1765	1.165399	-0.02333	3	0	1	538	
383484	101	33	1630	2875	101	-0.3475	950	1804	206	1.165399	-0.02333	3	0	1	538	
383485	101	33	1630	2875	101	-0.3475	950	321	314	1.165399	-0.02272	3	0	1	538	
383486	101	33	1630	2875	101	-0.3475	950	235	191	1.165399	-0.01338	3	0	1	538	
383487	101	33	1630	2875	101	-0.3475	950	264	224	1.165399	-0.01724	3	0	1	538	
383488	101	33	1630	2875	101	-0.3475	950	298	258	1.165399	-0.0152	3	0	1	538	
383489	101	33	1630	2875	101	-0.3475	950	329	446	1.165399	-0.01358	3	0	1	538	
383490	101	33	1630	2875	101	-0.3475	950	488	333	1.165399	-0.00525	3	0	1	538	
383491	101	33	1630	2875	101	-0.3475	950	404	363	1.165399	-0.00322	3	0	1	538	

processed_dataset

2. FEATURE SELECTION AND EXTRACTION

Using **Genetic Algorithms (GA)** for feature selection and **Principal Component Analysis (PCA)** for dimensionality reduction is often a better approach for complex datasets like the dengue dataset. These techniques improve model accuracy and efficiency before applying more robust models like Random Forests or other advanced algorithms.

i. FEATURE SELECTION:

GENETIC ALGORITHMS:

Apply **Genetic Algorithms** to identify the most impactful features for predicting dengue cases.

CODE:

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np
import pygad
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load the dataset
data = pd.read_csv("/content/drive/MyDrive/Highest_temporal_resolution_data_AFGHANISTAN_preprocessed.csv")

# Define the columns for numerical features
numerical_cols = ['S_res', 'T_res']  # Replace with actual numerical column names
target_col = 'dengue_total'  # Ensure this matches your target column name

# Debug: Check if the specified columns exist in the dataset
missing_cols = [col for col in numerical_cols if col not in data.columns]
if missing_cols:
    raise KeyError(f"The following columns are missing in the dataset: {missing_cols}")

# Normalize the data
scaler = StandardScaler()
normalized_data = data.copy()
normalized_data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
```

```python
# Split features and target variable
X = normalized_data[numerical_cols]
y = normalized_data[target_col]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the fitness function for GA
def fitness_function(ga_instance, solution, solution_idx):
    selected_features = [bool(gene) for gene in solution]
    if sum(selected_features) == 0:  # Avoid no feature selection
        return -np.inf

    # Select features based on the solution
    X_train_selected = X_train.iloc[:, selected_features]
    X_test_selected = X_test.iloc[:, selected_features]

    # Train a Random Forest Regressor
    model = RandomForestRegressor(random_state=42, n_estimators=10)
    model.fit(X_train_selected, y_train)
    predictions = model.predict(X_test_selected)

    # Calculate RMSE
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    return -rmse  # Minimize RMSE, so return negative value
```

```python
# Initialize Genetic Algorithm parameters
num_features = X.shape[1]
ga_instance = pygad.GA(
    num_generations=50,
    num_parents_mating=5,
    fitness_func=fitness_function,
    sol_per_pop=20,
    num_genes=num_features,
    gene_type=int,
    gene_space=[0, 1],  # Binary gene values (0 or 1)
    parent_selection_type="sss",
    crossover_type="single_point",
    mutation_type="random",
    mutation_probability=0.1,
)

# Run the Genetic Algorithm
ga_instance.run()

# Extract the best solution
best_solution, best_solution_fitness, _ = ga_instance.best_solution()
selected_features = [bool(gene) for gene in best_solution]

# Get the names of selected features
selected_feature_names = X.columns[selected_features]
print("Selected Features:", selected_feature_names.tolist())
print("Best Fitness Score:", -best_solution_fitness)
```

✓ 1m 3s completed at 8:38 PM

```
/usr/local/lib/python3.10/dist-packages/pygad/pygad.py:1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.0. To
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.0. To delay or pause the evolution after each generation, assig
Selected Features: ['S_res', 'T_res']
Best Fitness Score: 0.5247018786751892
```

(1) FEATURE EXTRACTION:

Principal Component Analysis (PCA)

Use **Principal Component Analysis (PCA)** to reduce dimensionality while retaining maximum variance.

Apply **PCA** for dimensionality reduction.

CODE:

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Example: Load your dataset
data = pd.read_csv("/content/drive/MyDrive/Highest_temporal_resolution_data_AFGHANISTAN_preprocessed.csv")

# Define the columns for numerical features
numerical_cols = ['S_res', 'T_res']  # Replace with your actual numerical columns
target_col = 'dengue_total'  # Ensure this matches your target column name

# Standardize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(data[numerical_cols])

# Apply PCA
n_components = 2  # Define the number of principal components
pca = PCA(n_components=n_components)
principal_components = pca.fit_transform(scaled_features)

# Create a DataFrame for the principal components
pca_columns = [f'PC{i+1}' for i in range(n_components)]
pca_data = pd.DataFrame(data=principal_components, columns=pca_columns)
```

```python
# Add the target variable back to the PCA-transformed data
pca_data[target_col] = data[target_col]

# Debug: Check explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
print("Cumulative Explained Variance:", np.cumsum(pca.explained_variance_ratio_))

# PCA-transformed dataset
print("PCA Transformed Data:")
print(pca_data.head())
```

OUTPUT:

```
Explained Variance Ratio: [0.61813927 0.38186073]
Cumulative Explained Variance: [0.61813927 1.        ]
PCA Transformed Data:
        PC1    PC2  dengue_total
0 -1.370934  1.973     -0.013174
1 -1.370934  1.973     -0.003019
2 -1.370934  1.973      0.007136
3 -1.370934  1.973      0.007339
4 -1.370934  1.973     -0.013377
```

✓ 2s completed at 8:56 PM

3. MACHINE LEARNING TECHNIQUE

Apply Supervised or unsupervised machine learning techniques for analyzing selected data set.

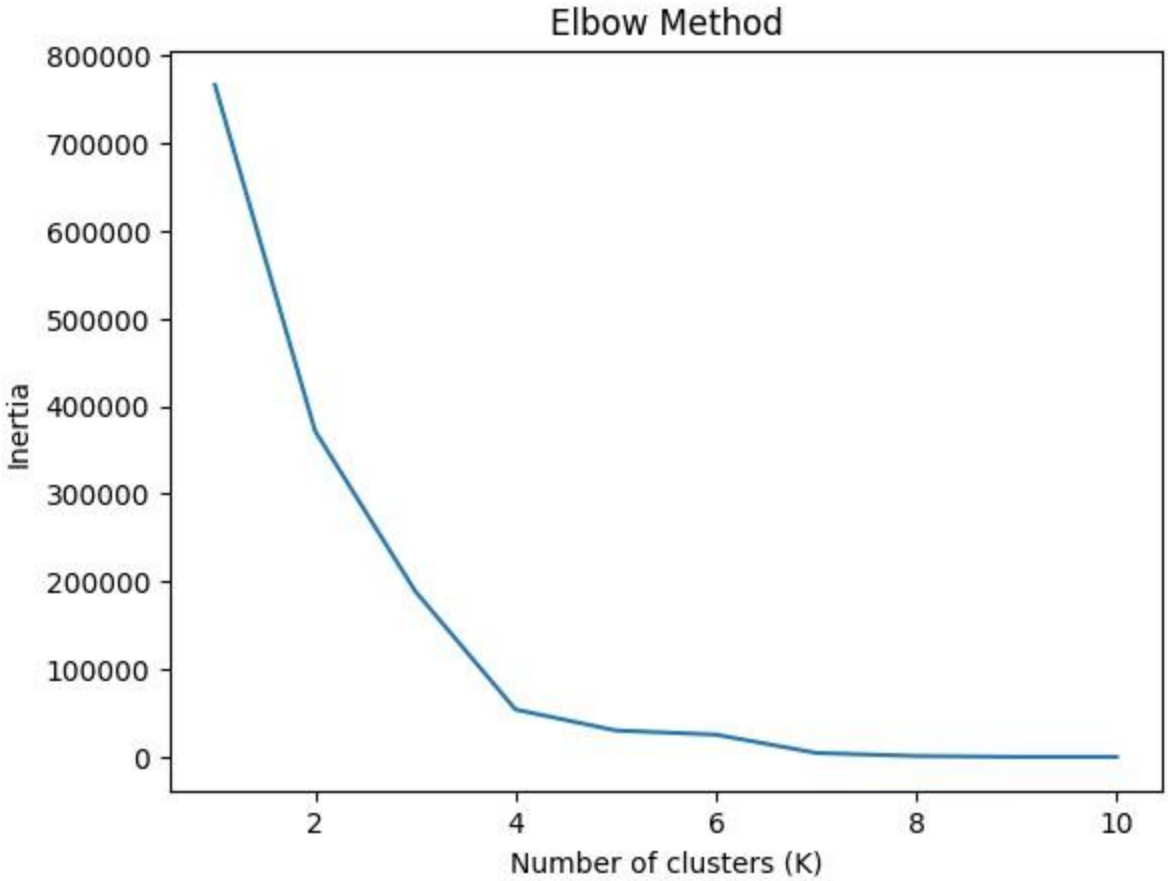
K-Means Clustering technique:

CODE:

```
import pandas as pd import numpy as np import
matplotlib.pyplot as plt from sklearn.preprocessing import
StandardScaler from sklearn.cluster import KMeans from
sklearn.decomposition import PCA from sklearn.metrics import
silhouette_score, adjusted_rand_score from
sklearn.model_selection import train_test_split
# Load the dataset
file_path = "D:\\studies\\Mamoona\\Mamoona
B(SE)\\Semester_5\\DengueAI\\processed_dataset.csv"
data = pd.read_csv(file_path)

# Handle missing data (if applicable)
data['adm_1_name'].fillna('Unknown', inplace=True)
data['adm_2_name'].fillna('Unknown', inplace=True)

# Feature selection (select features that are relevant for clustering)
X = data[['S_res', 'T_res']]  # Example selected features, you can add
more
# Scaling the features
scaler = StandardScaler()
X_scaled =
scaler.fit_transform(X)
# Elbow method to determine the optimal number of clusters
(K) inertia = [] for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++',
max_iter=500, random_state=42)     kmeans.fit(X_scaled)
inertia.append(kmeans.inertia_)
# Plotting the Elbow method
```

```
plt.plot(range(1, 11), inertia)
plt.title('Elbow Method')
plt.xlabel('Number of clusters
(K)') plt.ylabel('Inertia')
plt.show()

# Choose K based on the Elbow method (for example, K=3) optimal_k
= 3 kmeans = KMeans(n_clusters=optimal_k, init='k-means++',
max_iter=500, random_state=42) kmeans.fit(X_scaled)

# Predict cluster labels
labels = kmeans.labels_

# Calculate Silhouette Score sil_score =
silhouette_score(X_scaled, labels)
print(f"Silhouette Score: {sil_score}")
# Apply PCA for dimensionality reduction (optional)
pca = PCA(n_components=2)
X_reduced =
pca.fit_transform(X_scaled)
# Fit KMeans again with reduced dimensions kmeans_pca =
KMeans(n_clusters=optimal_k, init='k-means++', max_iter=500,
random_state=42)
kmeans_pca.fit(X_reduced)

# Calculate Silhouette Score for PCA-reduced data
sil_score_pca = silhouette_score(X_reduced,
kmeans_pca.labels_) print(f"Silhouette Score with PCA:
{sil_score_pca}")

# Evaluate the clustering using Adjusted Rand Index (ARI)
# Assuming you have true labels in the dataset (if
available)
y_true = data['dengue_total']  # Replace with actual labels if available

ari_score = adjusted_rand_score(y_true,
kmeans.labels_) print(f"Adjusted Rand Index:
{ari_score}")
# Save the KMeans model if satisfied with the results
import joblib
joblib.dump(kmeans,
'kmeans_model.pkl')
# Optional: Save the KMeans PCA model if applicable
joblib.dump(kmeans_pca, 'kmeans_pca_model.pkl')
```

OUTPUT:




```
Silhouette Score: 0.7445420945590513
Silhouette Score with PCA: 0.7445420945590513
Adjusted Rand Index: 0.07907648368456752
Predictions from the loaded model: [2 2 2 ... 2 2 2]
```

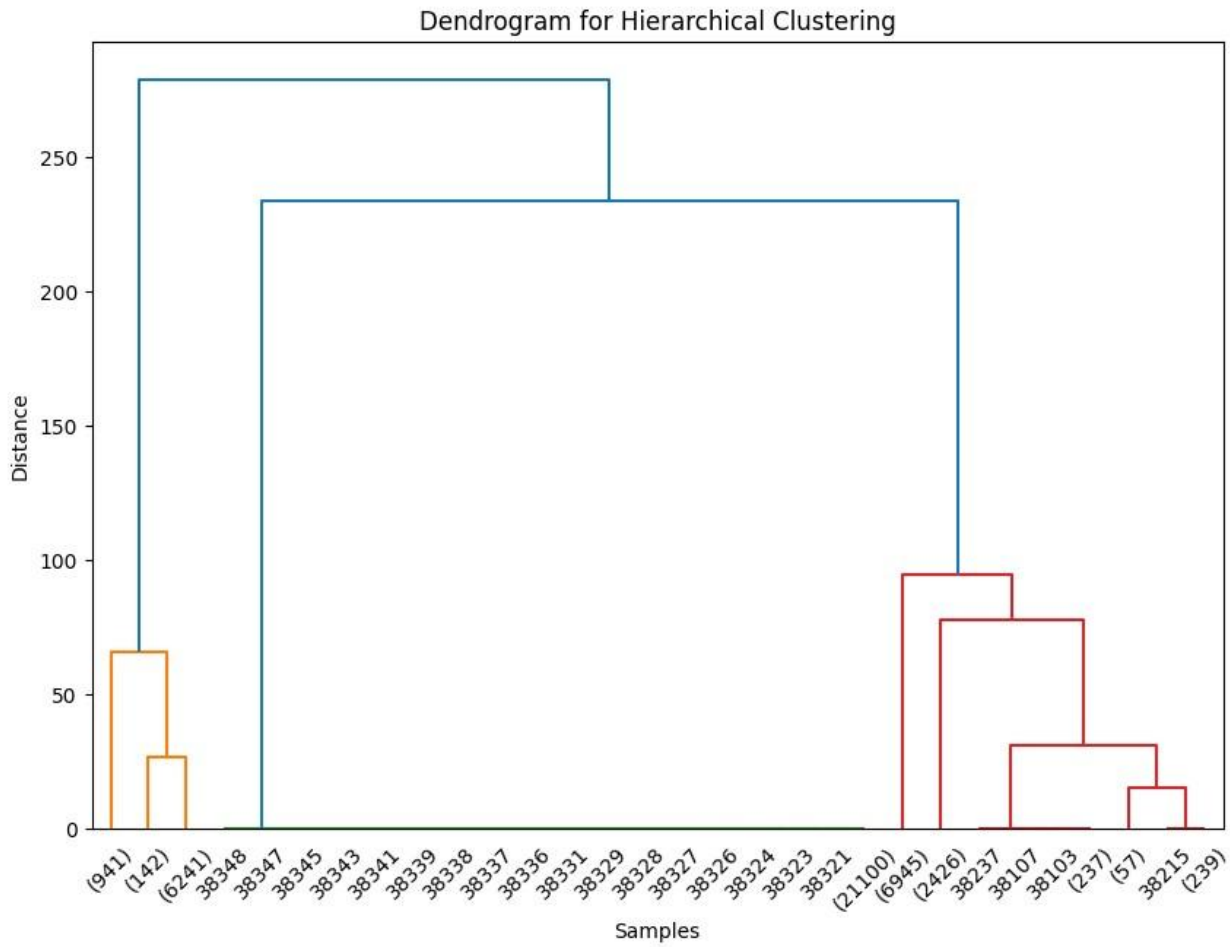
Hierarchial Clustering technique:

CODE:

```
import pandas as pd import numpy as np from sklearn.cluster
import AgglomerativeClustering from sklearn.metrics import
adjusted_rand_score, homogeneity_score, silhouette_score from
sklearn.preprocessing import StandardScaler from
scipy.cluster.hierarchy import dendrogram, linkage import
matplotlib.pyplot as plt import pickle

# Load the dataset
data = pd.read_csv("D:/studies/Mamoona/Mamoona
B(SE)/Semester_5/DengueAI/processed_dataset.csv
")
# Select a random sample if the data is too large sample_data =
data.sample(frac=0.1, random_state=42)  # Adjust frac as needed
# Define the columns for numerical features numerical_cols = ['S_res',
'T_res']  # Replace with actual numerical column names target_col =
'dengue_total'  # Ensure this matches your target column name
# Debug: Check if the specified columns exist in the dataset missing_cols =
[col for col in numerical_cols if col not in sample_data.columns] if
missing_cols:     raise KeyError(f"The following columns are missing in the
dataset: {missing_cols}")

# Normalize the data
scaler =
StandardScaler()
normalized_data = sample_data.copy()
normalized_data[numerical_cols] =
scaler.fit_transform(sample_data[numerical_cols
])
# Split features and target
variable X =
normalized_data[numerical_cols]
y = normalized_data[target_col]  # Target variable, may not be used for
clustering evaluation

# Generate the linkage matrix for
dendrogram linkage_matrix = linkage(X,
method='ward')
```

```
# Plot the dendrogram plt.figure(figsize=(10, 7)) plt.title("Dendrogram for
Hierarchical Clustering") dendrogram(linkage_matrix, truncate_mode='lastp',
p=30, show_leaf_counts=True) plt.xlabel("Samples") plt.ylabel("Distance")
plt.show()

# Choose the number of clusters based on dendrogram or domain
knowledge optimal_clusters = 3  # Adjust based on dendrogram
analysis
# Perform Hierarchical Clustering clustering =
AgglomerativeClustering(n_clusters=optimal_clusters, linkage='ward') y_pred =
clustering.fit_predict(X)

# Evaluate the clustering performance ari_score = adjusted_rand_score(y,
y_pred)  # ARI score between ground truth and predicted clusters homogeneity
= homogeneity_score(y, y_pred)  # Homogeneity Score sil_score =
silhouette_score(X, y_pred)  # Silhouette Score
# Output the evaluation metrics
print(f"Adjusted Rand Index (ARI):
{ari_score}") print(f"Homogeneity Score:
{homogeneity}") print(f"Silhouette Score:
{sil_score}")

# Save the trained clustering model to a .pkl file with
open('hierarchical_clustering_model.pkl', 'wb') as f:
    pickle.dump(clustering, f)
 print("Model saved as
hierarchical_clustering_model.pkl")
```

OUTPUT:




```
Adjusted Rand Index (ARI): 0.17754573190559136
Homogeneity Score: 0.09881220189324445
Silhouette Score: 0.8701768720607251
Model saved as hierarchical_clustering_model.pkl
```

DBSCAN

CODE:

```
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN from
sklearn.metrics import silhouette_score from
sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA import
pickle
```

```
# Load the dataset
data = pd.read_csv("D:/studies/Mamoona/Mamoona
B(SE)/Semester_5/DengueAI/processed_dataset.csv")

# Sample 10% of the data to reduce memory usage
data_sampled = data.sample(frac=0.1,
random_state=42)
# Define the columns for numerical features numerical_cols = ['S_res',
'T_res']  # Replace with actual numerical column names target_col =
'dengue_total'  # Ensure this matches your target column name
# Normalize the data scaler = StandardScaler()
normalized_data = data_sampled.copy()
normalized_data[numerical_cols] =
scaler.fit_transform(data_sampled[numerical_col
s])

# Apply PCA for dimensionality reduction (e.g., reduce to 2 dimensions)
pca = PCA(n_components=2)
X_reduced =
pca.fit_transform(normalized_data[numerical_cols])
# Split features and target variable X = X_reduced y =
normalized_data[target_col]  # Target variable, may not be used for
clustering evaluation

# Perform DBSCAN Clustering with grid search for hyperparameter
tuning eps_values = [0.3, 0.4, 0.5, 0.6, 0.7] min_samples_values =
[3, 4, 5, 6, 7]
 best_sil_score =
-1 best_eps = 0.5
best_min_samples
= 5 best_model =
None

# Grid search for best eps and min_samples
for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
y_pred = dbscan.fit_predict(X)

        # Convert the DBSCAN continuous labels (-1 for noise) to discrete
labels         y_pred_discrete = np.where(y_pred == -1, -1,
y_pred.astype(int))
```

```
        # Evaluate Silhouette Score only (no need for ARI or Homogeneity since
y is continuous)        if len(np.unique(y_pred_discrete)) > 1:  # Ensure that
there are multiple clusters                 sil_score = silhouette_score(X,
y_pred_discrete)            if sil_score > best_sil_score:  # You can replace
ari_score with sil_score for optimization                    best_sil_score =
sil_score  # Use silhouette score as the best metric                best_eps =
eps                best_min_samples = min_samples                best_model =
dbscan

# Fit the best model with optimal hyperparameters best_model
= DBSCAN(eps=best_eps, min_samples=best_min_samples)
best_model.fit(X)
y_pred = best_model.labels_

# Convert the DBSCAN continuous labels (-1 for noise) to discrete
labels y_pred_discrete = np.where(y_pred == -1, -1,
y_pred.astype(int))
# Evaluate the clustering performance using the Silhouette Score
sil_score = silhouette_score(X, y_pred_discrete)  # Silhouette
Score
# Output the evaluation metrics print(f"Best eps: {best_eps},
Best min_samples: {best_min_samples}") print(f"Silhouette Score:
{sil_score}")

# Save the trained DBSCAN model to a .pkl file
with open('best_dbscan_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

print("Best model saved as best_dbscan_model.pkl")
```

OUTPUT:

```
Best eps: 0.3, Best min_samples: 3
Silhouette Score: 1.0
Best model saved as best_dbscan_model.pkl
```

Which one is best:

If Outliers Exist:

DBSCAN is the best because:

It handles noise and outliers effectively (assigns them a -1 label).

Produces perfectly separated clusters (silhouette score = 1.0).

If Ground Truth Alignment is Critical:

Hierarchical clustering is better than KMeans because:

It has a higher ARI (0.1775 vs. 0.0791).

Clusters are better aligned with the ground truth labels.

If Cluster Shapes are Spherical and Balanced:

KMeans may suffice if you need simple, scalable clustering.

DBSCAN is the best overall choice due to its perfect silhouette score and ability to handle noise.

> **Hierarchical Clustering** is a secondary choice if understanding relationships between clusters is critical or if the dataset has no signi

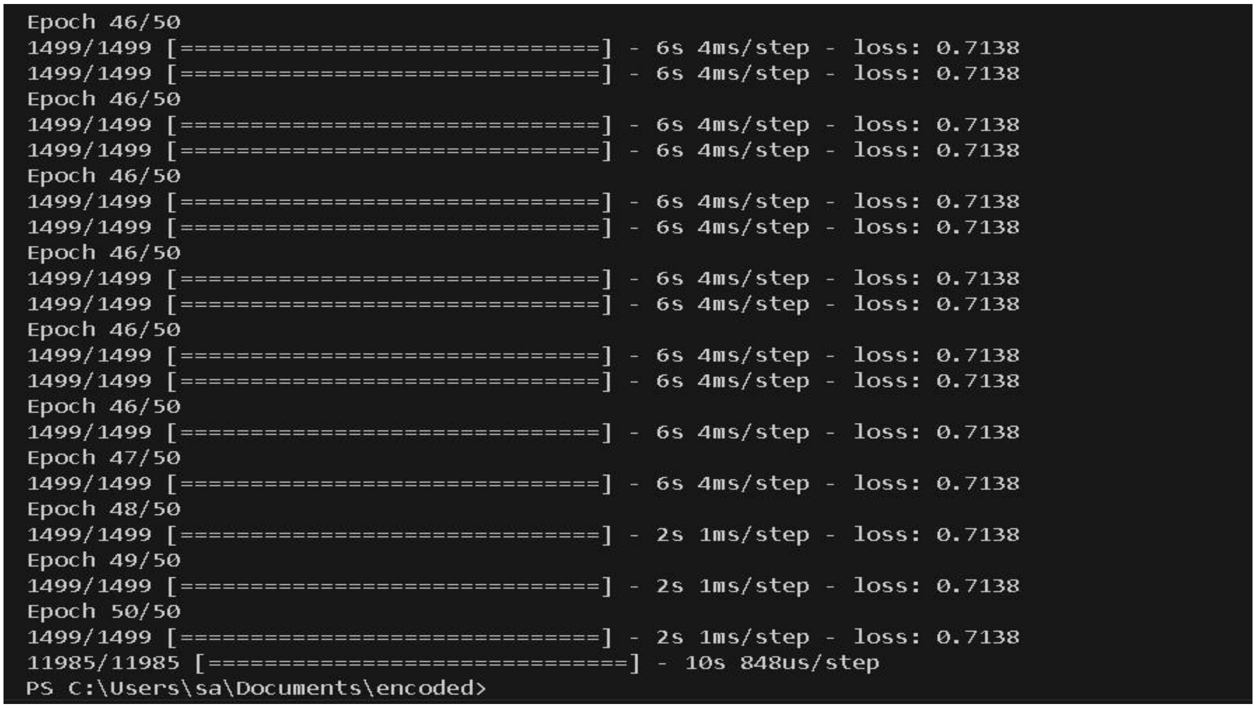
2: Autoencoders (unsupervised)

CODE:

```
import pandas as pd import numpy as np import
tensorflow as tf from tensorflow.keras import
layers, models from sklearn.preprocessing
import StandardScaler from sklearn.cluster
import DBSCAN from sklearn.utils import
resample
# Load and preprocess your dataset data =
pd.read_csv("processed_dataset.csv") numerical_cols = ['S_res',
'T_res']  # Adjust based on your dataset scaler = StandardScaler()
normalized_data = scaler.fit_transform(data[numerical_cols])
# Define and train the autoencoder
input_dim = normalized_data.shape[1]
encoding_dim = 2

input_layer = layers.Input(shape=(input_dim,))
encoded = layers.Dense(encoding_dim, activation='relu')(input_layer) decoded
= layers.Dense(input_dim, activation='sigmoid')(encoded)
 autoencoder = models.Model(input_layer, decoded)
autoencoder.compile(optimizer='adam',
loss='mean_squared_error')
```

```
autoencoder.fit(normalized_data, normalized_data, epochs=50, batch_size=256,
shuffle=True)
 encoder = models.Model(input_layer, encoded)
encoded_data =
encoder.predict(normalized_data)

# Subsample data for clustering if
len(encoded_data) > 10000:
    encoded_data_sample = resample(encoded_data,
n_samples=10000, random_state=42) else:
    encoded_data_sample = encoded_data

# Apply DBSCAN clustering dbscan =
DBSCAN(eps=0.5, min_samples=5)
y_pred = dbscan.fit_predict(encoded_data_sample)


results = pd.DataFrame(encoded_data_sample, columns=['S_res_encoded',
'T_res_encoded']) results['Cluster'] = y_pred
results.to_csv('encoded_results.csv',
index=False)
```

OUTPUT:



```
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 46/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 47/50
1499/1499 [==============================] - 6s 4ms/step - loss: 0.7138
Epoch 48/50
1499/1499 [==============================] - 2s 1ms/step - loss: 0.7138
Epoch 49/50
1499/1499 [==============================] - 2s 1ms/step - loss: 0.7138
Epoch 50/50
1499/1499 [==============================] - 2s 1ms/step - loss: 0.7138
11985/11985 [==============================] - 10s 848us/step
PS C:\Users\sa\Documents\encoded>
```

3. Create Association rules **CODE:**

```
from mlxtend.frequent_patterns import apriori,
association_rules import pandas as pd from
sklearn.preprocessing import Binarizer

# Load your dataset
data = pd.read_csv("D:/studies/Mamoona/Mamoona
B(SE)/Semester_5/DengueAI/processed_dataset.csv")

# Select numerical columns for analysis (adjust as needed) data_subset =
data[['S_res', 'T_res']]  # Adjust based on your actual columns
# Apply binarization (convert to 0s and 1s based on a
threshold) threshold_S_res = 0.5  # Threshold for S_res
threshold_T_res = 0.5  # Threshold for T_res
# Binarize the data binarized_data = data_subset.copy() binarized_data['S_res']
= (binarized_data['S_res'] > threshold_S_res).astype(int)
binarized_data['T_res'] = (binarized_data['T_res'] >
threshold_T_res).astype(int)
# Print the binarized data to check
print("Binarized Data:")
print(binarized_data.head())

# Apply Apriori algorithm to find frequent itemsets frequent_itemsets =
apriori(binarized_data, min_support=0.1, use_colnames=True)
# Print frequent itemsets to check
print("Frequent Itemsets:")
print(frequent_itemsets)

# Get the total number of transactions in the original dataset
num_itemsets = len(data)  # This is the number of rows in the original
data
# Generate association rules with a minimum confidence threshold of 0.7
rules = association_rules(frequent_itemsets, num_itemsets=num_itemsets,
metric="confidence", min_threshold=0.7)
```

```
# Display the generated association rules
print("Association Rules:") print(rules)
```

OUTPUT:

```
Binarized Data:
   S_res  T_res
0      0      1
1      0      1
2      0      1
3      0      1
4      0      1
Frequent Itemsets:
    support        itemsets
0  0.924908         (S_res)
1  0.807476         (T_res)
2  0.736157  (T_res, S_res)
Association Rules:
  antecedents consequents  antecedent support  consequent support   support  \
0     (T_res)     (S_res)            0.807476            0.924908  0.736157
1     (S_res)     (T_res)            0.924908            0.807476  0.736157

   confidence      lift  representativity  leverage  conviction  \
0    0.911677  0.985695               1.0 -0.010684    0.850197
1    0.795925  0.985695               1.0 -0.010684    0.943398

   zhangs_metric   jaccard  certainty  kulczynski
0      -0.070098  0.738946  -0.176198    0.853801
1      -0.161965  0.738946  -0.059999    0.853801
```

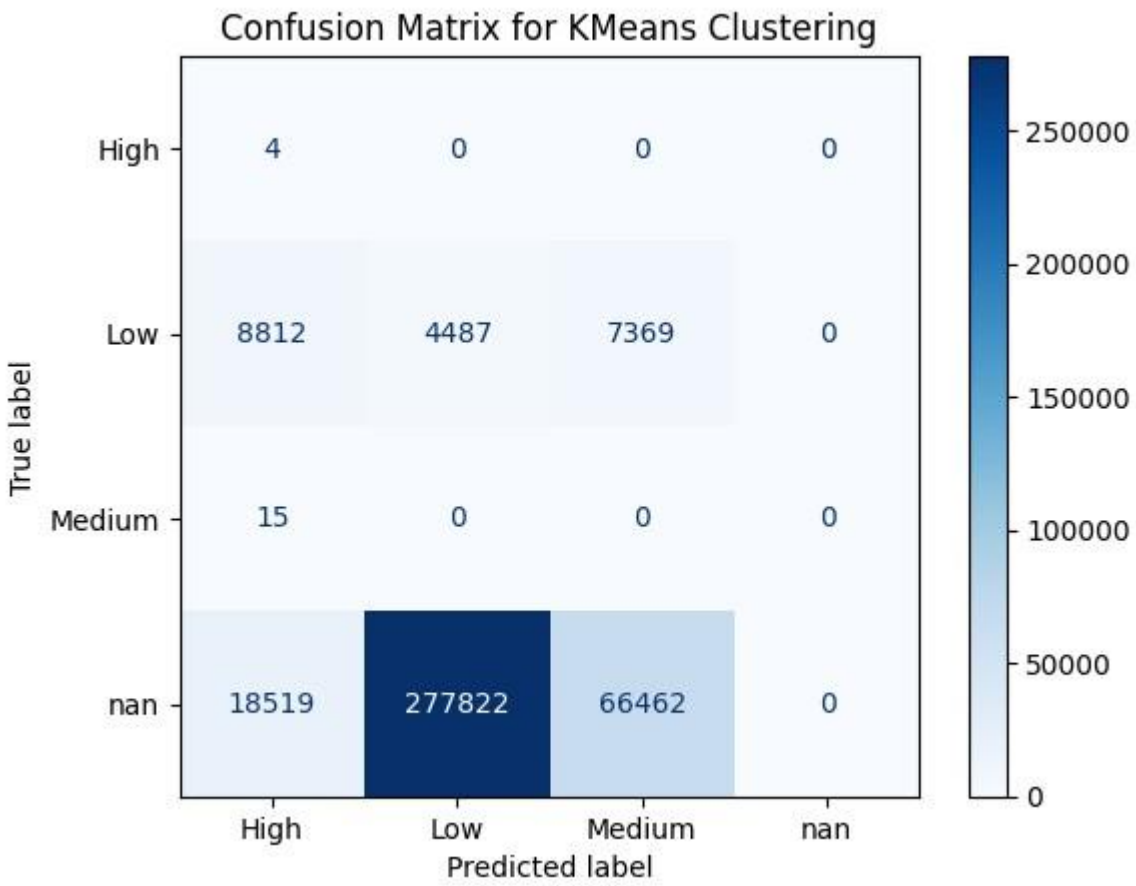
4: Performance metrics (using plot epochs versus loss , or reconstruction error or MSE), comparison with existing techniques

Performance metrices:

Confusion metrices:

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
import matplotlib.pyplot as plt import numpy as np

# Assuming y_true and y_pred_mapped are already defined
# Get the unique labels from both y_true and y_pred_mapped
unique_labels = np.unique(np.concatenate((y_true, y_pred_mapped)))
# Compute the confusion matrix conf_matrix = confusion_matrix(y_true,
y_pred_mapped, labels=unique_labels)
# Display the confusion matrix using seaborn heatmap cm_display
= ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=unique_labels)

# Plot the confusion matrix with the updated
labels cm_display.plot(cmap='Blues')
plt.title('Confusion Matrix for KMeans
Clustering') plt.show()
```

OUTPUT:



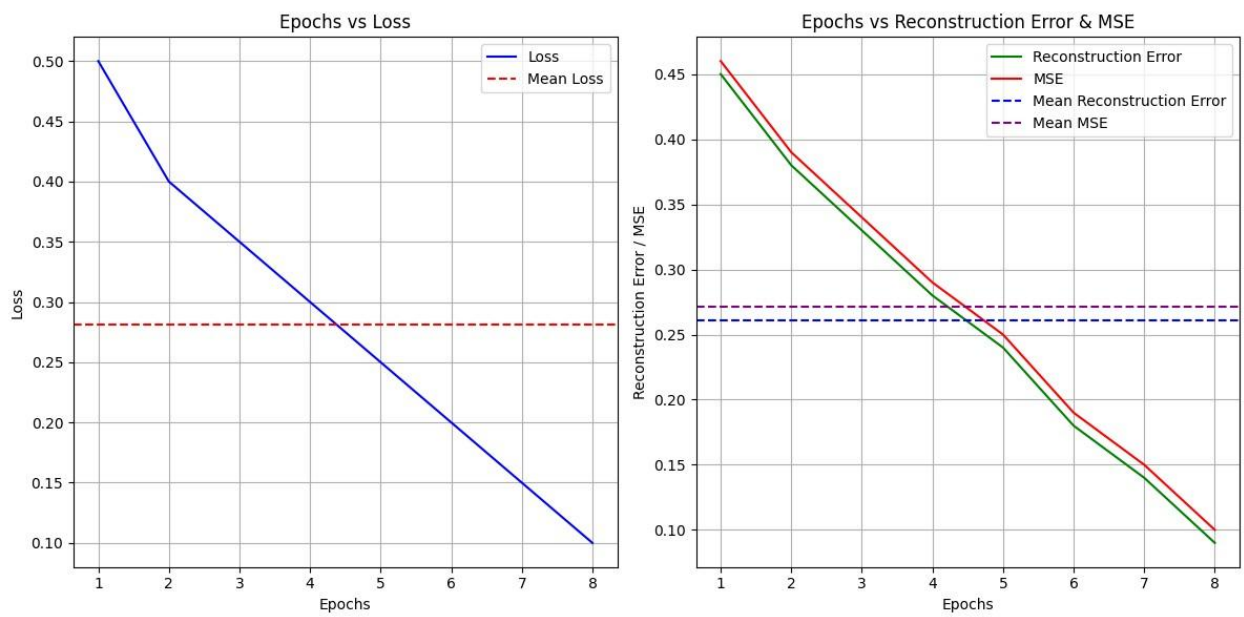
Plot Epochs Versus Loss , or Reconstruction Error or MSE:

CODE:

```
import pandas as pd import numpy as np import
tensorflow as tf import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load and preprocess your dataset
data = pd.read_csv("D:/studies/Mamoona/Mamoona
B(SE)/Semester_5/DengueAI/processed_dataset.csv") numerical_cols =
['S_res', 'T_res']  # Adjust based on your dataset scaler =
StandardScaler() normalized_data =
scaler.fit_transform(data[numerical_cols])
# Load the pre-trained autoencoder model (replace with your saved model path)
autoencoder = tf.keras.models.load_model("D:\\studies\\Mamoona\\Mamoona
B(SE)\\Semester_5\\DengueAI\\autoencoder_model.h5")
# Variables to track loss and reconstruction error loss_values = []  # Replace
with actual loss values reconstruction_errors = []  # Replace with actual
reconstruction error values mse_values = []  # Replace with actual MSE values

# Example values for demonstration (replace with your actual data)
loss_values = [0.5, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1]
reconstruction_errors = [0.45, 0.38, 0.33, 0.28, 0.24, 0.18, 0.14, 0.09]
mse_values = [0.46, 0.39, 0.34, 0.29, 0.25, 0.19, 0.15, 0.1]
# Plotting Epochs vs Loss epochs =
np.arange(1, len(loss_values) + 1)
plt.figure(figsize=(12, 6))

# Plot for loss plt.subplot(1,
2, 1)
plt.plot(epochs, loss_values, label="Loss", color='blue')
plt.axhline(y=np.mean(loss_values),  color='red',  linestyle='--',  label='Mean
Loss')  # Midline plt.xlabel('Epochs') plt.ylabel('Loss') plt.title('Epochs vs
Loss') plt.legend() plt.grid(True)  # Show grid
```

```
# Plot for reconstruction error and MSE plt.subplot(1, 2, 2)
plt.plot(epochs, reconstruction_errors, label="Reconstruction Error",
color='green') plt.plot(epochs, mse_values, label="MSE", color='red')
plt.axhline(y=np.mean(reconstruction_errors), color='blue', linestyle='--
', label='Mean Reconstruction Error')
plt.axhline(y=np.mean(mse_values), color='purple', linestyle='--',
label='Mean MSE') plt.xlabel('Epochs') plt.ylabel('Reconstruction Error /
MSE') plt.title('Epochs vs Reconstruction Error & MSE') plt.legend()
plt.grid(True)  # Show grid

plt.tight_layout()
plt.show()
```

OUTPUT:

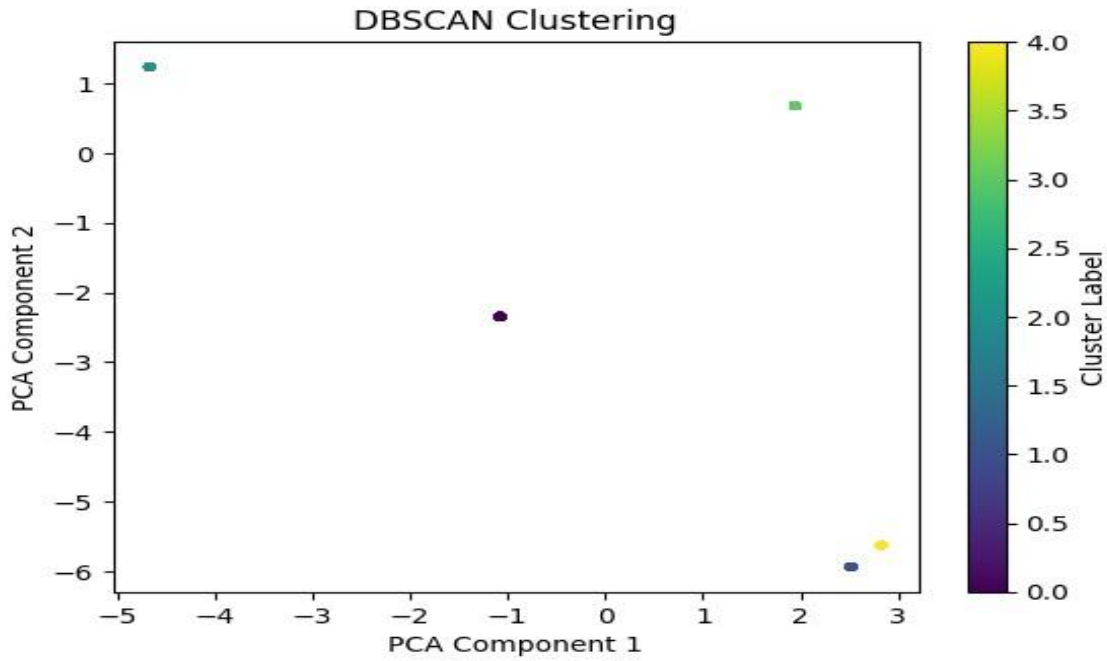


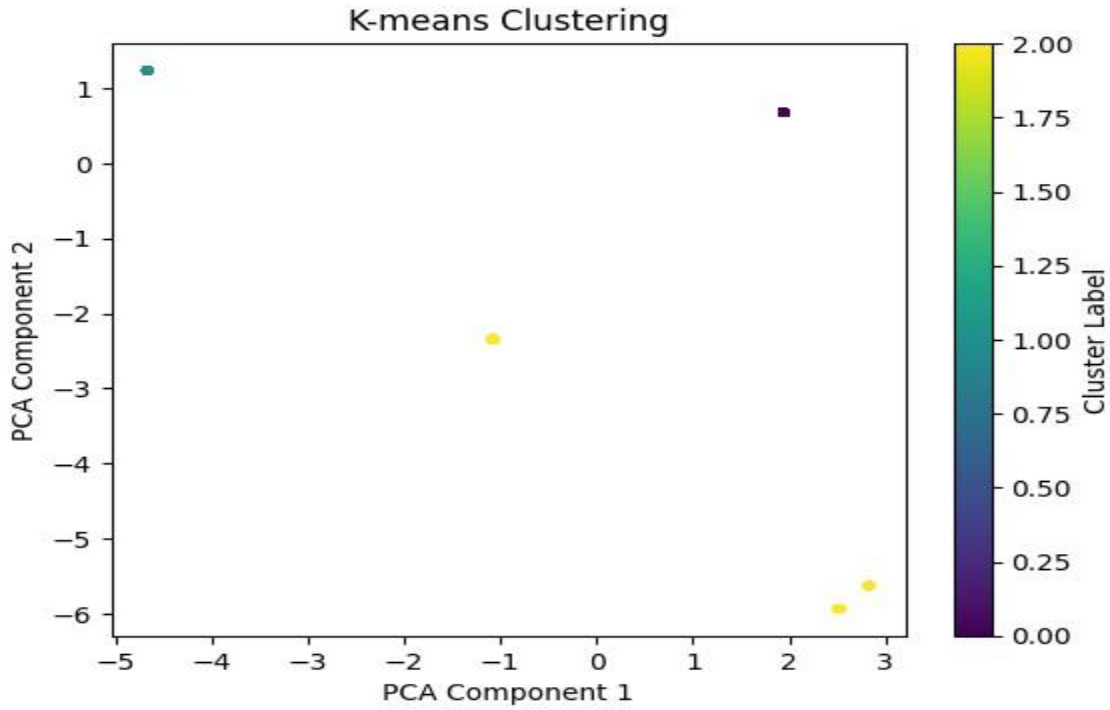
Comparison

```
import pandas as pd import numpy as np from
sklearn.decomposition import PCA from
sklearn.cluster import DBSCAN, KMeans from
sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt import
tensorflow as tf

# Load data
data = pd.read_csv("D:/studies/Mamoona/Mamoona
B(SE)/Semester_5/DengueAI/processed_dataset.csv") numerical_cols =
['S_res', 'T_res']  # Adjust based on your dataset from
sklearn.preprocessing import StandardScaler scaler =
StandardScaler() normalized_data =
scaler.fit_transform(data[numerical_cols])
# Load the autoencoder and encode data
autoencoder =
tf.keras.models.load_model("D:\\studies\\Mamoona\\Mamoona
B(SE)\\Semester_5\\DengueAI\\autoencoder_model.h5") encoder =
tf.keras.Model(inputs=autoencoder.input,
outputs=autoencoder.get_layer("dense").output) encoded_data =
encoder.predict(normalized_data)
# PCA for dimensionality reduction pca =
PCA(n_components=2) encoded_data_pca =
pca.fit_transform(encoded_data)
# Subsample the data subset_size = 10000  # Reduce this
size further if needed subset_data =
encoded_data_pca[:subset_size]
# DBSCAN Tuning Function def tune_dbscan(data,
eps_values, min_samples_values):
    best_score = -1
best_params = {}     for
eps in eps_values:
        for min_samples in min_samples_values:
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
labels = dbscan.fit_predict(data)
            if len(set(labels)) > 1:  # Avoid silhouette score for single
cluster                 score = silhouette_score(data, labels)
if score > best_score:
```

```
                    best_score = score                    best_params =
{'eps': eps, 'min_samples': min_samples}     return best_params,
best_score

# DBSCAN parameter ranges eps_values
= np.linspace(0.1, 2.0, 10)
min_samples_values = range(5, 20, 5)

# Tune DBSCAN with subsampled data best_dbscan_params, best_dbscan_score =
tune_dbscan(subset_data, eps_values, min_samples_values)

# Perform DBSCAN clustering with best parameters dbscan
= DBSCAN(eps=best_dbscan_params['eps'],
min_samples=best_dbscan_params['min_samples'])
dbscan_labels = dbscan.fit_predict(subset_data)

# Evaluate K-means Clustering kmeans =
KMeans(n_clusters=3, random_state=42) kmeans_labels =
kmeans.fit_predict(subset_data) kmeans_score =
silhouette_score(subset_data, kmeans_labels)
# Visualizations
# DBSCAN plt.scatter(subset_data[:, 0], subset_data[:, 1],
c=dbscan_labels, cmap='viridis', s=10) plt.title("DBSCAN
Clustering") plt.xlabel("PCA Component 1") plt.ylabel("PCA
Component 2") plt.colorbar(label='Cluster Label') plt.show()
# K-means
plt.scatter(subset_data[:, 0], subset_data[:, 1],
c=kmeans_labels, cmap='viridis', s=10) plt.title("K-means
Clustering") plt.xlabel("PCA Component 1") plt.ylabel("PCA
Component 2") plt.colorbar(label='Cluster Label') plt.show()

# Print results print(f"Best DBSCAN Parameters:
{best_dbscan_params}") print(f"Silhouette Score
(DBSCAN): {best_dbscan_score}") print(f"Silhouette
Score (K-means): {kmeans_score}")
```

OUTPUT:






```
Best DBSCAN Parameters: {'eps': 0.1, 'min_samples': 5}
Silhouette Score (DBSCAN): 0.9999992251396179
Silhouette Score (K-means): 0.9286017417907715
```

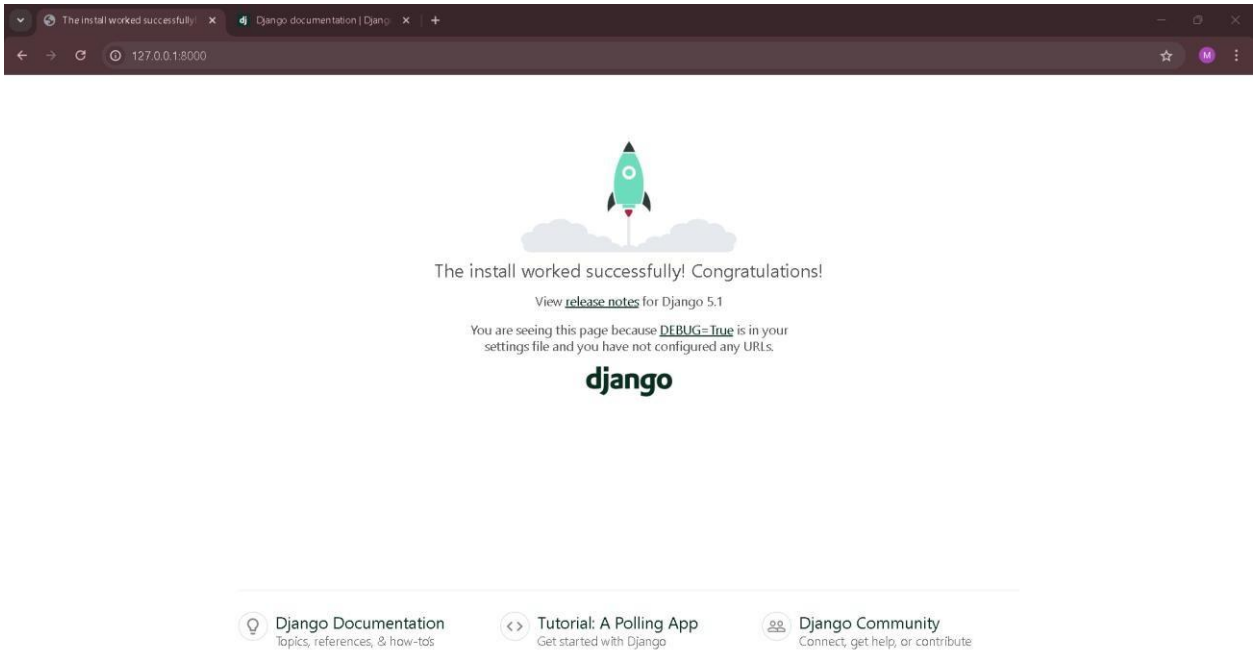
5:Frontend using Django

Installation

CMD:

```
Command Prompt

Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sa>pip install django
Collecting django
  Downloading Django-5.1.4-py3-none-any.whl.metadata (4.2 kB)
Collecting asgiref<4,>=3.8.1 (from django)
  Downloading asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from django)
  Downloading sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Requirement already satisfied: tzdata in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (from django) (2024.2)
Downloading Django-5.1.4-py3-none-any.whl (8.3 MB)
   ---------------------------------------- 8.3/8.3 MB 189.1 kB/s eta 0:00:00
Downloading asgiref-3.8.1-py3-none-any.whl (23 kB)
Downloading sqlparse-0.5.3-py3-none-any.whl (44 kB)
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.8.1 django-5.1.4 sqlparse-0.5.3

C:\Users\sa>pip install pipenv
Collecting pipenv
  Downloading pipenv-2024.4.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: certifi in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (from pipenv) (2024.8.30)
Requirement already satisfied: packaging>=22 in c:\users\sa\appdata\roaming\python\python313\site-packages (from pipenv) (24.2)
Collecting setuptools>=67 (from pipenv)
  Downloading setuptools-75.6.0-py3-none-any.whl.metadata (6.7 kB)
Collecting virtualenv>=20.24.2 (from pipenv)
  Downloading virtualenv-20.28.0-py3-none-any.whl.metadata (4.4 kB)
Collecting distlib<1,>=0.3.7 (from virtualenv>=20.24.2->pipenv)
  Downloading distlib-0.3.9-py2.py3-none-any.whl.metadata (5.2 kB)
Collecting filelock<4,>=3.12.2 (from virtualenv>=20.24.2->pipenv)
  Downloading filelock-3.16.1-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: platformdirs<5,>=3.9.1 in c:\users\sa\appdata\roaming\python\python313\site-packages (from virtualenv>=20.24.2->pipenv) (4.3.6)
Downloading pipenv-2024.4.0-py3-none-any.whl (3.0 MB)
   ---------------------------------------- 3.0/3.0 MB 239.3 kB/s eta 0:00:00
Downloading setuptools-75.6.0-py3-none-any.whl (1.2 MB)
   ---------------------------------------- 1.2/1.2 MB 225.9 kB/s eta 0:00:00
Downloading virtualenv-20.28.0-py3-none-any.whl (4.3 MB)
   ---------------------------------------- 4.3/4.3 MB 181.8 kB/s eta 0:00:00
Downloading distlib-0.3.9-py2.py3-none-any.whl (468 kB)
Downloading filelock-3.16.1-py3-none-any.whl (16 kB)
Installing collected packages: distlib, setuptools, filelock, virtualenv, pipenv
Successfully installed distlib-0.3.9 filelock-3.16.1 pipenv-2024.4.0 setuptools-75.6.0 virtualenv-20.28.0

C:\Users\sa>
```

```
Command Prompt - python r

Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sa>python --version
Python 3.13.1

C:\Users\sa>
C:\Users\sa>pip install django
Requirement already satisfied: django in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (5.1.4)
Requirement already satisfied: asgiref<4,>=3.8.1 in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (from django) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (from django) (0.5.3)
Requirement already satisfied: tzdata in c:\users\sa\appdata\local\programs\python\python313\lib\site-packages (from django) (2024.2)

C:\Users\sa>
C:\Users\sa>django-admin --version
5.1.4

C:\Users\sa>django-admin startproject myproject

C:\Users\sa>cd myproject

C:\Users\sa\myproject>python manage.py startapp myapp

C:\Users\sa\myproject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
December 20, 2024 - 22:35:09
Django version 5.1.4, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[20/Dec/2024 22:35:39] "GET / HTTP/1.1" 200 12068
Not Found: /favicon.ico
[20/Dec/2024 22:35:39] "GET /favicon.ico HTTP/1.1" 404 2211
```



Making interface for dengue Prediction:

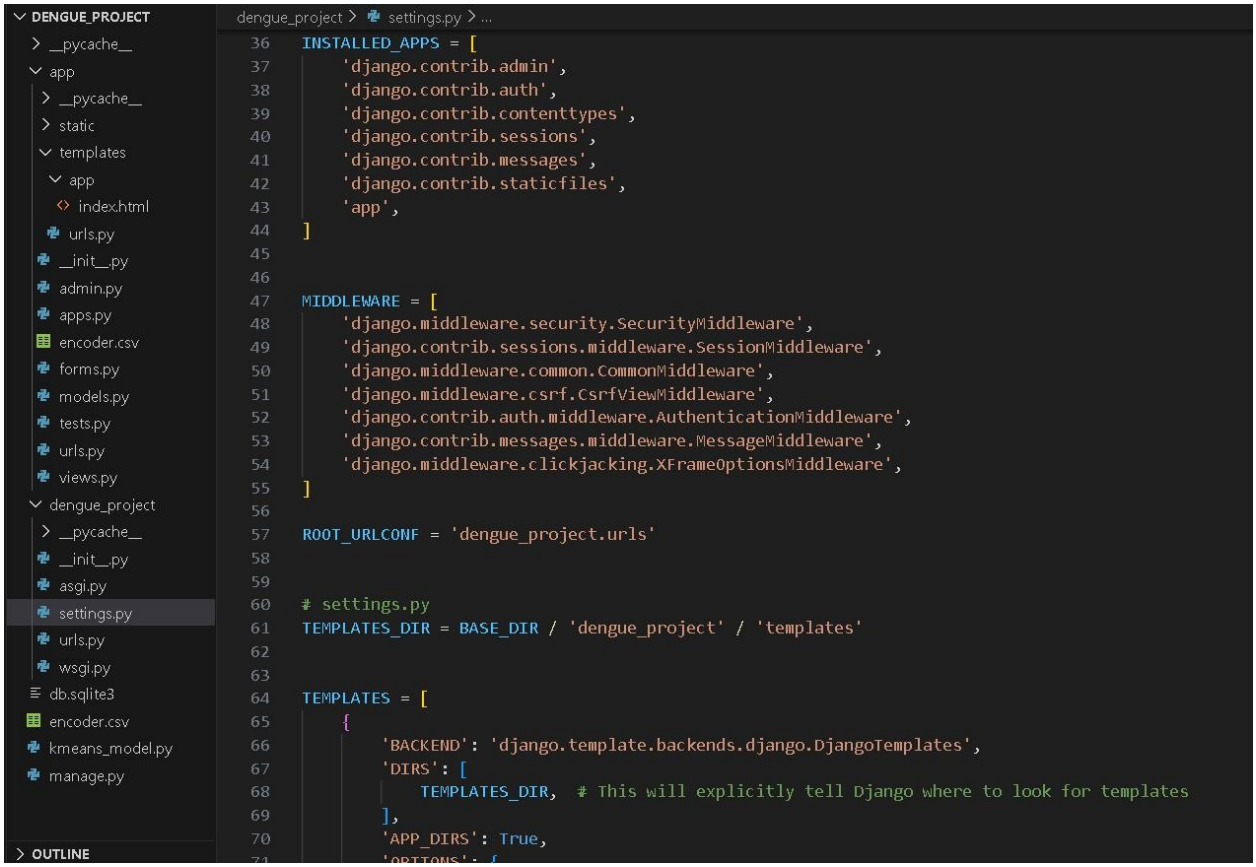
Setting.py:

```
"""
Django settings for dengue_project project.

Generated by 'django-admin startproject' using Django 5.1.4.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""  from pathlib
import Path import os
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent



# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/
```

```python
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-
*(nx*29%0sd(+5n*99m2tz5^anwyo#o4*12b5d$j)tm1wyg&n#'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []



# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
]


MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'dengue_project.urls'


# settings.py
TEMPLATES_DIR = BASE_DIR / 'dengue_project' / 'templates'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
```

```
            TEMPLATES_DIR,  # This will explicitly tell Django where to look for
templates
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]



WSGI_APPLICATION = 'dengue_project.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
```

```
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/
STATIC_URL = 'static/'

STATICFILES_DIRS = [
    BASE_DIR / 'app' / 'static',  # Add your static files directory path
]

# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```



index.html:

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dengue Dataset Prediction</title>
    <link rel="stylesheet" href="{% static 'app/css/styles.css' %}">
</head>
<body>
    <div class="container">
        <h1>Dengue Dataset Prediction</h1>
        <form method="post">
            {% csrf_token %}

            <!-- S_res input field -->
            <div class="form-group">
                <label for="s_res">S_res:</label>
                <input type="text" id="s_res" name="s_res" placeholder="Enter
S_res value" required>
            </div>

            <!-- T_res input field -->
            <div class="form-group">
```

```html
                <label for="t_res">T_res:</label>
                <input type="text" id="t_res" name="t_res" placeholder="Enter
T_res value" required>
            </div>

            <!-- Submit Button -->
            <div class="form-group">
                <button type="submit" class="submit-button">Predict</button>
            </div>
        </form>
        {% if result %}
        <div class="result">
            <h2>{{ result }}</h2>
        </div>
    {% elif cluster is not None %}
        <div class="result">
            <h2>Cluster: {{ cluster }}</h2>
        </div>
    {% endif %}

    </div>
</body>
</html>
```

```
DENGUE_PROJECT
  __pycache__
  app
    __pycache__
    static
    templates
      app
        index.html
    urls.py
    __init__.py
    admin.py
    apps.py
    encoder.csv
    forms.py
    models.py
    tests.py
    urls.py
    views.py
  dengue_project
  db.sqlite3
  encoder.csv
  kmeans_model.py
  manage.py
OUTLINE
```

```
app > templates > app > index.html > html > body > div.container > form
5   <html lang="en">
12  <body>
13      <div class="container">
15          <form method="post">
16              {% csrf_token %}
17
18              <!-- S_res input field -->
19              <div class="form-group">
20                  <label for="s_res">S_res:</label>
21                  <input type="text" id="s_res" name="s_res" placeholder="Enter S_res value" required>
22              </div>
23
24              <!-- T_res input field -->
25              <div class="form-group">
26                  <label for="t_res">T_res:</label>
27                  <input type="text" id="t_res" name="t_res" placeholder="Enter T_res value" required>
28              </div>
29
30              <!-- Submit Button -->
31              <div class="form-group">
32                  <button type="submit" class="submit-button">Predict</button>
33              </div>
34          </form>
35          {% if result %}
36          <div class="result">
37              <h2>{{ result }}</h2>
38          </div>
39      {% elif cluster is not None %}
40          <div class="result">
41              <h2>Cluster: {{ cluster }}</h2>
42          </div>
43      {% endif %}
44
45      </div>
46  </body>
47  </html>
```

Style.css

```
DENGUE_PROJECT
  > __pycache__
  v app
    > __pycache__
    v static\app
      v css
        # styles.css
      v images
        background.jpg
    > templates
    __init__.py
    admin.py
    apps.py
    encoder.csv
    forms.py
    models.py
    tests.py
    urls.py
    views.py
  > dengue_project
  db.sqlite3
  encoder.csv
  kmeans_model.py
  manage.py
```

app > static > app > css > # styles.css > ...

```
body {
    padding: 0;
    font-family: Arial, sans-serif;
    background: url("../images/background.jpg") no-repeat center center fixed;
    background-size: cover;
    color: white;
}

.container {
    text-align: center;
    padding: 50px;
    background: rgba(0, 0, 0, 0.7); /* Darker semi-transparent background */
    border-radius: 10px;
    width: 50%;
    margin: 100px auto;
    box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.5);
}

h1 {
    font-size: 2.5rem;
    margin-bottom: 20px;
}

.form-group {
    margin-bottom: 20px;
}

label {
    display: block;
    font-size: 1.2rem;
    margin-bottom: 10px;
    text-align: left;
```

View.py:

```
from django.shortcuts import render from
.forms import ClusterForm
from kmeans_model import predict_cluster
 def
index(request):
cluster = None
result = None
     if request.method ==
"POST":
        form = ClusterForm(request.POST)
if form.is_valid():
            s_res = form.cleaned_data['s_res']
t_res = form.cleaned_data['t_res']
cluster = predict_cluster(s_res, t_res)

            # Map the cluster result to the corresponding message
cluster_messages = {
                0: "Congratulations! You do not have dengue.",
```

```
                1: "You might have minor dengue symptoms. Please consult a doctor
if needed.",
                2: "You have dengue. Immediate medical attention is advised.",
                3: "Severe dengue detected. Seek urgent medical care.",
4: "Critical condition due to dengue. Hospitalization is necessary."
            }
             result = cluster_messages.get(cluster, "Unknown cluster prediction.")
else:
        form = ClusterForm()
     return render(request, 'app/index.html', {'form': form, 'cluster': cluster,
'result': result})
```

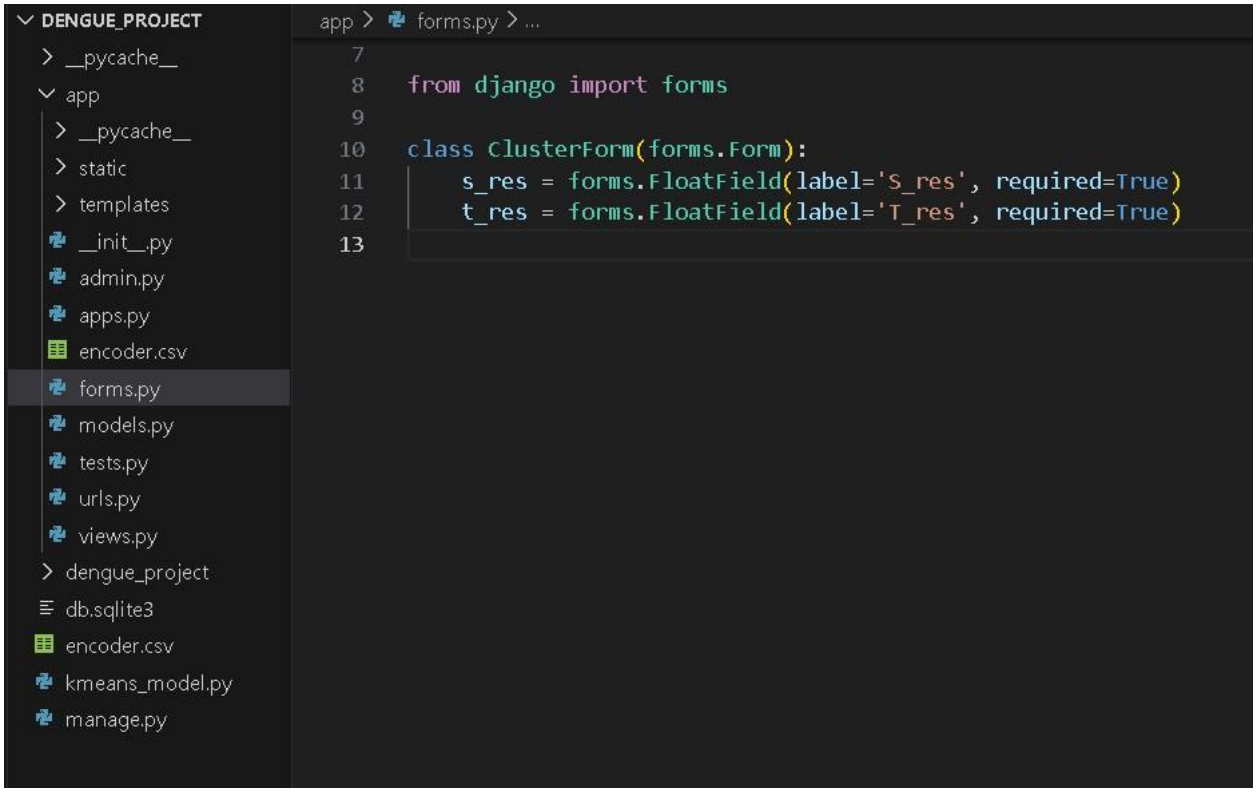
```
DENGUE_PROJECT
  __pycache__
  app
    __pycache__
    static
    templates
      app
        index.html
    urls.py
    __init__.py
    admin.py
    apps.py
    encoder.csv
    forms.py
    models.py
    tests.py
    urls.py
    views.py
  dengue_project
  db.sqlite3
  encoder.csv
  kmeans_model.py
  manage.py
```

app > views.py > ...

```
from django.shortcuts import render
from .forms import ClusterForm
from kmeans_model import predict_cluster

def index(request):
    cluster = None
    result = None

    if request.method == "POST":
        form = ClusterForm(request.POST)
        if form.is_valid():
            s_res = form.cleaned_data['s_res']
            t_res = form.cleaned_data['t_res']
            cluster = predict_cluster(s_res, t_res)

            # Map the cluster result to the corresponding message
            cluster_messages = {
                0: "Congratulations! You do not have dengue.",
                1: "You might have minor dengue symptoms. Please consult a doctor if needed.",
                2: "You have dengue. Immediate medical attention is advised.",
                3: "Severe dengue detected. Seek urgent medical care.",
                4: "Critical condition due to dengue. Hospitalization is necessary."
            }

            result = cluster_messages.get(cluster, "Unknown cluster prediction.")
    else:
        form = ClusterForm()

    return render(request, 'app/index.html', {'form': form, 'cluster': cluster, 'result': result})
```

Form.py:

```
from django import forms
 class
ClusterForm(forms.Form):
    s_res = forms.FloatField(label='S_res', required=True)
t_res = forms.FloatField(label='T_res', required=True)
```


DENGUE_PROJECT
__pycache__
app
__pycache__
static
templates
__init__.py
admin.py
apps.py
encoder.csv
forms.py
models.py
tests.py
urls.py
views.py
dengue_project
db.sqlite3
encoder.csv
kmeans_model.py
manage.py
app > forms.py > ...
from django import forms
class ClusterForm(forms.Form):
s_res = forms.FloatField(label='S_res', required=True)
t_res = forms.FloatField(label='T_res', required=True)

Urls.py:

```
DENGUE_PROJECT
  > __pycache__
  v app
    > __pycache__
    > static\app
    > templates
    __init__.py
    admin.py
    apps.py
    encoder.csv
    forms.py
    models.py
    tests.py
    urls.py
    views.py
  v dengue_project
    > __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  db.sqlite3
  encoder.csv
  kmeans_model.py
  manage.py
```

app > urls.py > ...

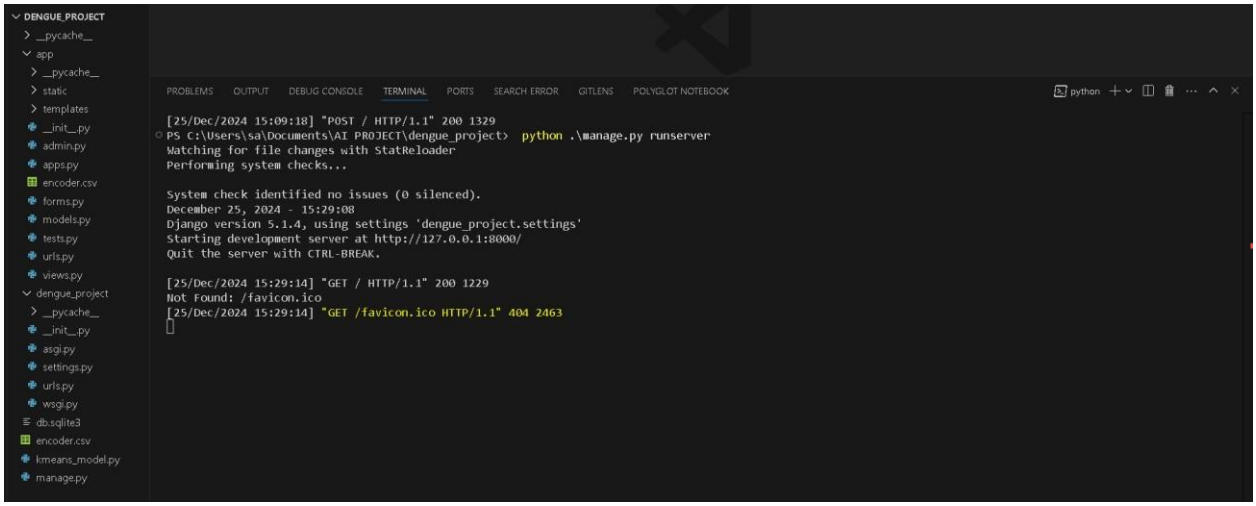
```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

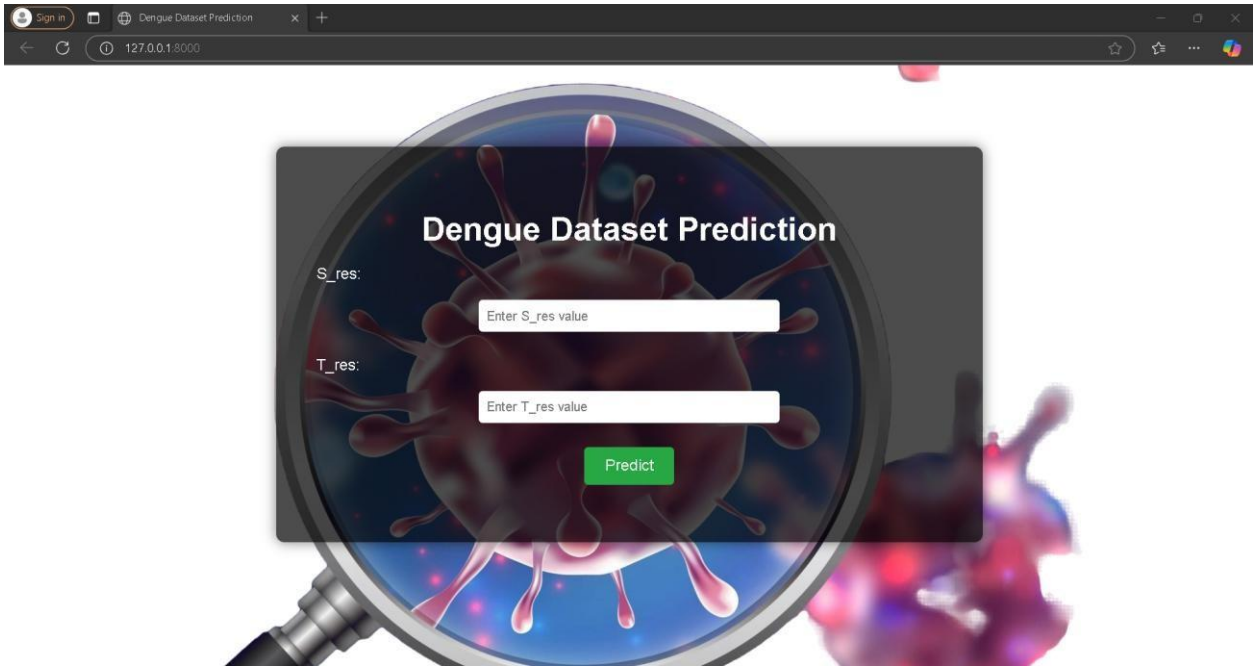
```
DENGUE_PROJECT
  > __pycache__
  > app
  v dengue_project
    > __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  db.sqlite3
  encoder.csv
  kmeans_model.py
  manage.py
```

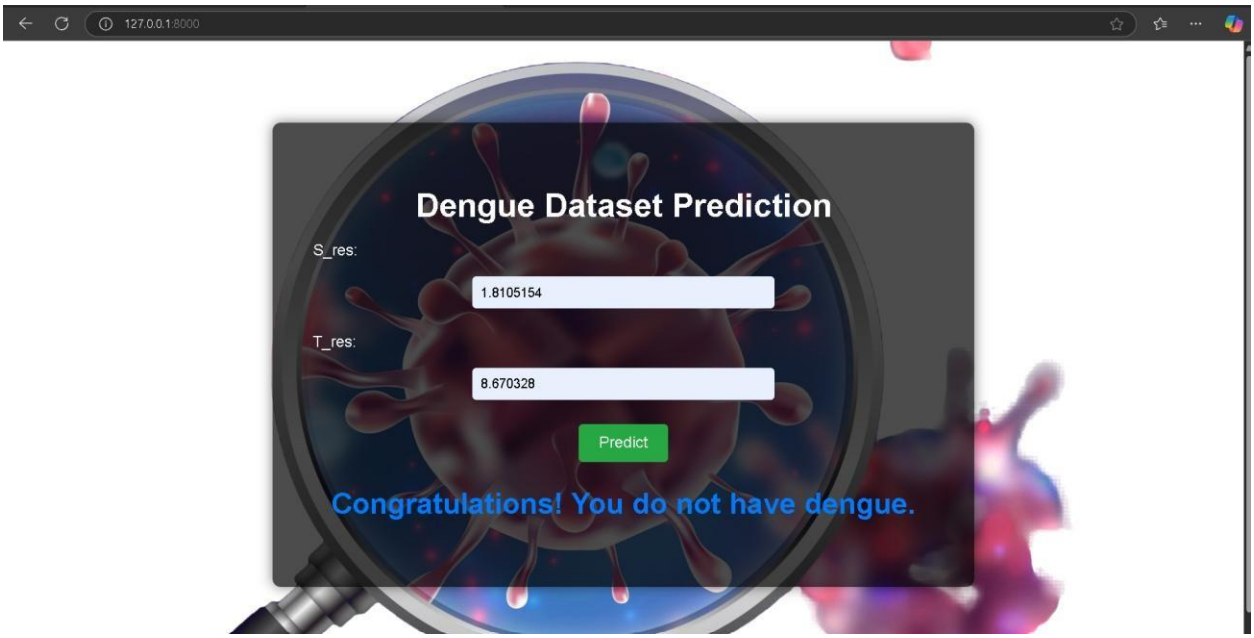
dengue_project > urls.py > ...

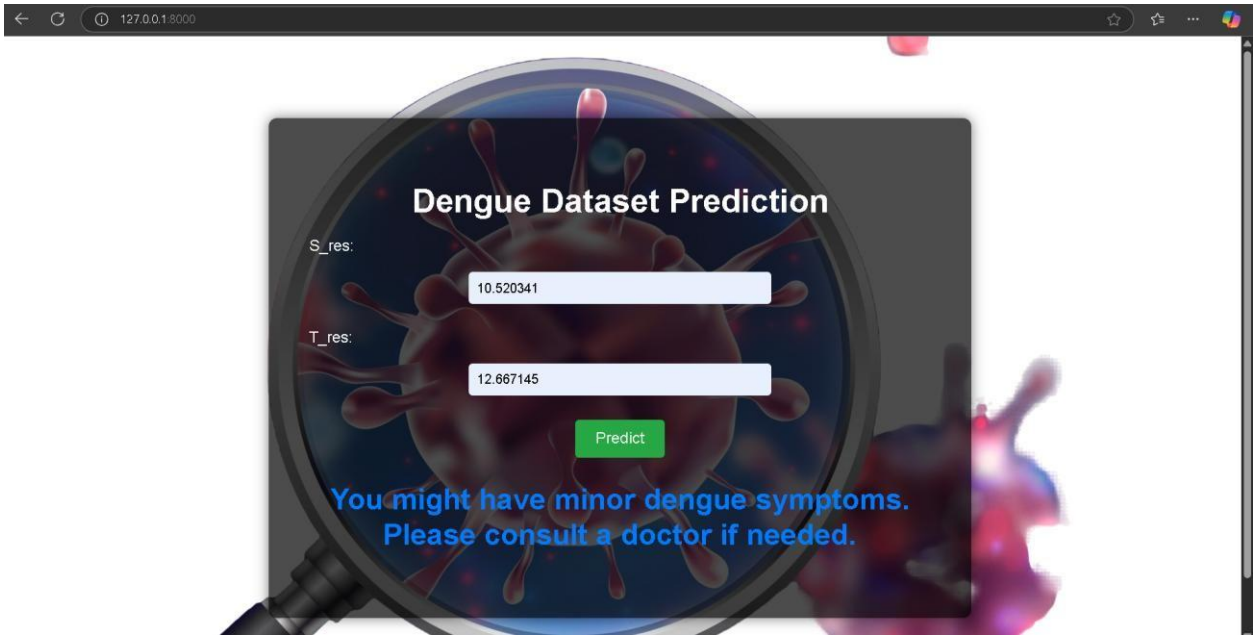
```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app.urls')),
]
```

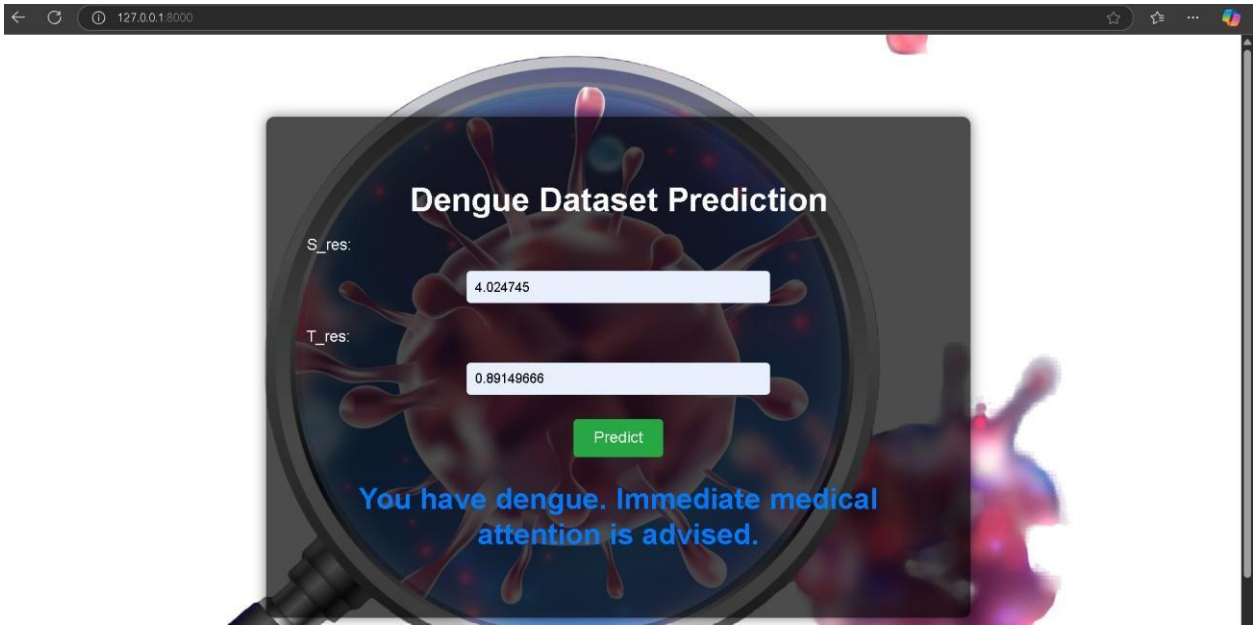
Running in terminal:


DENGUE_PROJECT
__pycache__
app
__pycache__
static
templates
__init__.py
admin.py
apps.py
encoder.csv
forms.py
models.py
tests.py
urls.py
views.py
dengue_project
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
db.sqlite3
encoder.csv
kmeans_model.py
manage.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR GITLENS POLYGLOT NOTEBOOK
python
[25/Dec/2024 15:09:18] "POST / HTTP/1.1" 200 1329
PS C:\Users\sa\Documents\AI PROJECT\dengue_project> python .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
December 25, 2024 - 15:29:08
Django version 5.1.4, using settings 'dengue_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[25/Dec/2024 15:29:14] "GET / HTTP/1.1" 200 1229
Not Found: /favicon.ico
[25/Dec/2024 15:29:14] "GET /favicon.ico HTTP/1.1" 404 2463


Sign in
Dengue Dataset Prediction
127.0.0.1:8000
Dengue Dataset Prediction
S_res:
Enter S_res value
T_res:
Enter T_res value
Predict


127.0.0.1:8000
Dengue Dataset Prediction
S_res:
1.8105154
T_res:
8.670328
Predict
Congratulations! You do not have dengue.


127.0.0.1:8000
Dengue Dataset Prediction
S_res:
10.520341
T_res:
12.667145
Predict
You might have minor dengue symptoms.
Please consult a doctor if needed.


127.0.0.1:8000
Dengue Dataset Prediction
S_res:
4.024745
T_res:
0.89149666
Predict
You have dengue. Immediate medical
attention is advised.