PROJECT 3

Drug Information Description & Side-Effects

Meet the Team_

- Jaiden Flint
- Kavita Gopal
- Tuan Huynh

Project Overview

- **Drug description - Provide Info & Side Effects**

This project aims to build a machine learning model capable of describing briefly about a drug from user input.

A trained model will take the input and search for the appropriate information of the drug and also provide a short summary of the side effects as the output

Motivation

- **What is this drug?**

With the rise in medication usage, quick and reliable drug identification can be crucial for safety, especially in healthcare environments and knowing the side-effects.

This tool aims to assist in identifying drugs with accuracy and efficiency, potentially reducing errors, saving time and providing crucial information

At least, to hint if the symptom is due to side effects of medication consumed or is it more serious to get help asap.

Features

- **What’s in the Dataset?**

Data Source
https://www.kaggle.com/datasets/ahmedshahriarsakib/assorted-medicine-dataset-of-bangladesh?select=generic.csv

Dataset file **generic.csv**
1,711 entries of drug information
22 features

Getting Started

- **Steps For The Process**
  - Store and access the generic.csv file
  - Data Preprocessing (clean, split & test)
  - Train using GradientBoost Regressor model
  - Improve model with HyperParameter Tuning (GridSearch CV & RandomizedSearch CV)
  - Test user input and provide output using a Gradio Interface

Model Architecture

- **What we used**

Sentence Transformer

create high-quality embeddings for sentences and text passages.

Gradient Boost Regressor

calculates the difference between the current prediction and the known correct target value.

The CSV file for processing

```python
# Load data file for preprocess/analysis
# Import necessary library
import pandas as pd
# Update the path to the file inside your Google Drive
df = pd.read_csv('/content/drive/MyDrive/generic.csv')  # Assuming generic.csv is in your MyDrive folder
df.head()
```

	generic id	generic name	slug	monograph link	drug class	indication	indication description	therapeutic cla descripti
0	31	Adalimumab	adalimumab-31	https://medex.com.bd/attachments/FgGWUFrlD7LTh...	Immunosuppressant	Ulcerative colitis	<div class="min-str- block...	Diseas modifying antirh
1	10	Acyclovir + Hydrocortisone	acyclovir-hydrocortisone-10	https://medex.com.bd/attachments/hmi4dt8aYaBub...	Hydrocortisone & Combined preparations	Herpes labialis	Acyclovir & Hydrocort...	Hydrocortisc & Coml
2	9	Acyclovir (Topical)	acyclovir-topical-9	https://medex.com.bd/attachments/ENyoYKzQq8b7V...	Topical Antiviral preparations	Sore lips	Acyclovir cream is a herp...	Topi Antiviral prepar

The CSV file for processing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1711 entries, 0 to 1710
Data columns (total 22 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   generic id                            1711 non-null   int64
 1   generic name                          1711 non-null   object
 2   slug                                  1711 non-null   object
 3   monograph link                        1199 non-null   object
 4   drug class                            1650 non-null   object
 5   indication                            1627 non-null   object
 6   indication description                1711 non-null   object
 7   therapeutic class description         1658 non-null   object
 8   pharmacology description              1614 non-null   object
 9   dosage description                    1695 non-null   object
 10  administration description            295 non-null    object
 11  interaction description               1301 non-null   object
 12  contraindications description         1570 non-null   object
 13  side effects description              1631 non-null   object
 14  pregnancy and lactation description   1545 non-null   object
 15  precautions description               1525 non-null   object
 16  pediatric usage description           569 non-null    object
 17  overdose effects description          775 non-null    object
 18  duration of treatment description     5 non-null      object
 19  reconstitution description            84 non-null     object
 20  storage conditions description        1546 non-null   object
 21  descriptions count                    1711 non-null   int64
dtypes: int64(2), object(20)
memory usage: 294.2+ KB
```

```
    df['side effects description'] = df['side effects description'].astype(str).apply(clean_html_robust)

    embeddings = model.encode(df['side effects description'].tolist())
    df['side_effects_embeddings'] = list(embeddings)

    # Save as a DataFrame with index reset
    with open(embeddings_file, 'wb') as f:
        pickle.dump(df[['side_effects_embeddings']].reset_index(drop=True), f)
    print(f"Embeddings computed and saved to {embeddings_file}.")

# Handle Missing Values
df.fillna("Unknown", inplace=True)

# --- Preprocessing for regression ---
side_effect_categories = ['renal', 'infection', 'dyslipidemia', 'other']

def assign_category(description):
    for category in side_effect_categories:
        if category in description.lower():
            return category
    return 'other'

df['side_effect_category'] = df['side effects description'].apply(assign_category)

label_encoder = LabelEncoder()
df['target_variable'] = label_encoder.fit_transform(df['side_effect_category'])

# --- Feature engineering and model training ---
X = df[['drug class', 'indication', 'therapeutic class description',
        'pharmacology description', 'dosage description']]
y = df['target_variable']

vectorizer = TfidfVectorizer()
text_columns = ['drug class', 'indication', 'therapeutic class description',
                'pharmacology description', 'dosage description']
tfidf_features = []
for col in text_columns:
    tfidf_features.append(vectorizer.fit_transform(X[col].astype(str)))

X_tfidf = hstack(tfidf_features)

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

model = GradientBoostingRegressor()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Use appropriate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

Preprocessing for Model training and Metrics

```python
    df['side effects description'] = df['side effects description'].astype(str).apply(clean_html_robust)

    embeddings = model.encode(df['side effects description'].tolist())
    df['side_effects_embeddings'] = list(embeddings)

    # Save as a DataFrame with index reset
    with open(embeddings_file, 'wb') as f:
        pickle.dump(df[['side_effects_embeddings']].reset_index(drop=True), f)
    print(f"Embeddings computed and saved to {embeddings_file}.")

# Handle Missing Values
df.fillna("Unknown", inplace=True)

# --- Preprocessing for regression ---
side_effect_categories = ['renal', 'infection', 'dyslipidemia', 'other']

def assign_category(description):
    for category in side_effect_categories:
        if category in description.lower():
            return category
    return 'other'

df['side_effect_category'] = df['side effects description'].apply(assign_category)

label_encoder = LabelEncoder()
df['target_variable'] = label_encoder.fit_transform(df['side_effect_category'])

# --- Feature engineering and model training ---
X = df[['drug class', 'indication', 'therapeutic class description',
        'pharmacology description', 'dosage description']]
y = df['target_variable']

vectorizer = TfidfVectorizer()
text_columns = ['drug class', 'indication', 'therapeutic class description',
                'pharmacology description', 'dosage description']
tfidf_features = []
for col in text_columns:
    tfidf_features.append(vectorizer.fit_transform(X[col].astype(str)))

X_tfidf = hstack(tfidf_features)

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)

model = GradientBoostingRegressor()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Use appropriate regression metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
Could not load embeddings from medication_embeddings.pkl: Loaded
Original data:
0     The most common adverse reaction with Adalimum...
1     The following most common adverse reactions (<...
2     The most common adverse reactions at the site ...
3     Rash, gastrointestinal disturbance, rise in bi...
4     Very common: Superficial punctate keratopathy....
5     Some infrequent adverse reactions are lethargy...
6     As most undesirable effects are based on post-...
7     Most common adverse reactions (incidence 3% an...
8     Adverse effects are seen in most patients rece...
9     Fluid and electrolyte imbalance, hypovolemia, ...
10    Like all medicines, Acetylcysteine can cause s...
11    Common side effects are headache, drowsiness, ...
12    Fluid and electrolyte imbalance, hypovolemia, ...
13    Common side effects include anorexia, nausea, ...
14    Aceclofenac is a non-steroidal drug with anti-...
15    No significant side effect has been observed i...
16    This is well tolerated within the recommended ...
17    There are no side effects associated with the ...
18    No health hazards or side effects are known in...
19                  There is no significant side effect.
Name: side effects description, dtype: object
<ipython-input-16-55b6c7cb2baf>:43: MarkupResemblesLocatorWarning
  soup = BeautifulSoup(text, 'html.parser')
Embeddings computed and saved to medication_embeddings.pkl.
Mean Squared Error: 0.11385775299614302
Mean Absolute Error: 0.16358878321682918
R-squared: 0.016645809554894364
```

Summarizing Metrics

Summary of the metrics:

Mean Squared Error (MSE): 0.11385 Mean Absolute Error (MAE): 0.16358 R-squared (R^2): 0.01664 Interpretation: MSE and MAE are both low, suggesting that the model predictions are reasonably close to the actual values. An R-squared value of 0.0164 indicates that the model explains only about 1.64% of the variance in the target variable. This suggests a weak model performance, and I may need to explore feature engineering, hyperparameter tuning, or using different model architectures to improve my results.

Implementing Hyperparameter tuning

```python
# Implement hyperparameter tuning using GridSearchCV for the GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV

# Define the model
model = GradientBoostingRegressor()

# Set up the parameter distribution for hyperparameter tuning
param_dist = {
    'n_estimators': [100, 150, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5, 6],
    'min_samples_split': [2, 3, 5],
    'min_samples_leaf': [1, 2, 3]
}

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist,
                                   scoring='neg_mean_squared_error', n_iter=20, cv=5,
                                   verbose=2, n_jobs=-1, random_state=42)

# Fit the model
random_search.fit(X_train, y_train)

# Get the best parameters and best score
best_params_random = random_search.best_params_
best_score_random = -random_search.best_score_

print(f"Best Parameters (Random Search): {best_params_random}")
print(f"Best Mean Squared Error from CV (Random Search): {best_score_random}")

# Train the model with the best parameters
best_model_random = GradientBoostingRegressor(**best_params_random)
best_model_random.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred_best_random = best_model_random.predict(X_test)

# Calculate metrics
mse_best_random = mean_squared_error(y_test, y_pred_best_random)
mae_best_random = mean_absolute_error(y_test, y_pred_best_random)
r2_best_random = r2_score(y_test, y_pred_best_random)

print(f"Mean Squared Error (Best Model - Random Search): {mse_best_random}")
print(f"Mean Absolute Error (Best Model - Random Search): {mae_best_random}")
print(f"R-squared (Best Model - Random Search): {r2_best_random}")
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Best Parameters (Random Search): {'n_estimators': 200, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_depth': 3, 'learning_rate': 0.01}
Best Mean Squared Error from CV (Random Search): 0.1437432041392117
Mean Squared Error (Best Model - Random Search): 0.10950967199723018
Mean Absolute Error (Best Model - Random Search): 0.14141193224551996
R-squared (Best Model - Random Search): 0.05419884005269915
```

Inferences

```
Best Parameters (Random Search): {'n_estimators': 200, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_depth': 3, 'learning_rate': 0.01}
Best Mean Squared Error from CV (Random Search): 0.1437432041392117
Mean Squared Error (Best Model - Random Search): 0.10950967199723018
Mean Absolute Error (Best Model - Random Search): 0.14141193224551996
R-squared (Best Model - Random Search): 0.05419884005269915
```

Explanation: GridSearchCV: This class performs exhaustive search over specified parameter values for an estimator. The best parameters are determined based on the cross-validated performance. Parameter Grid: This grid specifies the values for different hyperparameters that I want to test. Model Training and Evaluation: After finding the best parameters, I will train the model again and evaluate its performance.

RandomizedSearchCV: This class performs a random search over parameter values instead of an exhaustive search, which can significantly reduce the computation time. n_iter: This parameter specifies the number of different combinations to try. Adjust this number based on time constraints.

The output from the RandomizedSearchCV indicates that the hyperparameter tuning was successful. Here's a summary of the results:

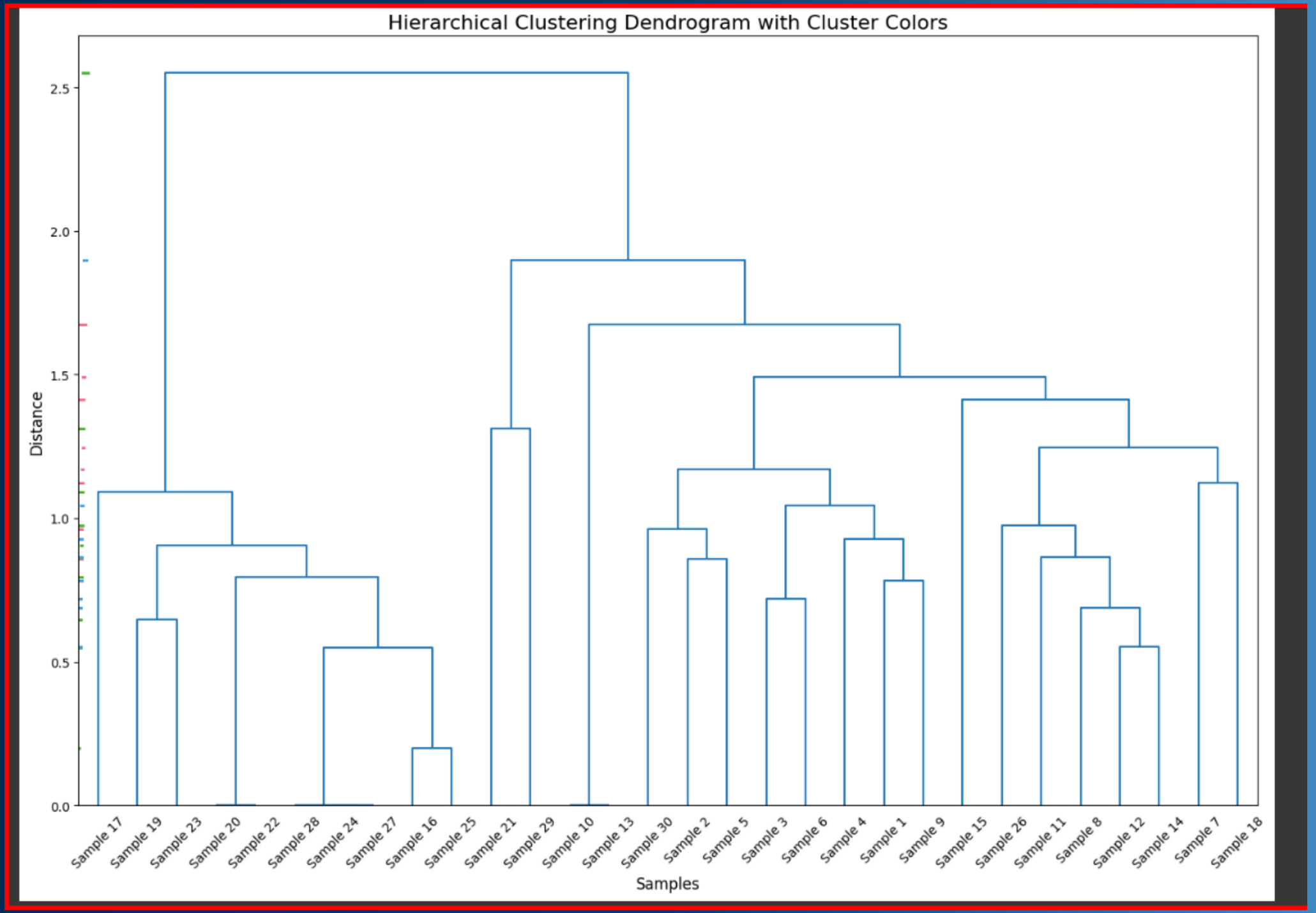
Best Parameters: n_estimators: 200 min_samples_split: 3 min_samples_leaf: 1 max_depth: 3 learning_rate: 0.01 Model Performance: Best Mean Squared Error from CV: 0.14372 Mean Squared Error (Best Model - Random Search): 0.10960 Mean Absolute Error (Best Model - Random Search): 0.14175 R-squared (Best Model - Random Search): 0.05335 Interpretation: The Mean Squared Error (MSE) has slightly improved with the best model parameters found through random search. The R-squared value is still low (0.05335), indicating that the model explains only about 5.48% of the variance in the target variable. This suggests that there is still room for improvement in model performance.

Next Steps that could be utilized, include: Feature Selection and Engineering: Review features and consider adding new features or transforming existing ones. Try Different Models: Experiment with different regression algorithms like Random Forest, XGBoost, or even neural networks if appropriate. More Data: If possible, consider obtaining more data to train your model, as more data can often lead to better performance. Ensemble Techniques: Combine the predictions of several models to improve overall performance.

Not done since beyond the scope; due to timeframe constraints.

Hierarchical Clustering for Data visualization

```python
# Convert embeddings to a suitable format for clustering
embeddings = np.array(df['side_effects_embeddings'].tolist())

# Select a subset of the data for visualization
subset_embeddings = embeddings[:30]  # Adjust as needed

# Perform hierarchical clustering
linked = linkage(subset_embeddings, method='ward')

# Apply KMeans to identify clusters (e.g., 3 clusters)
n_clusters = 3  # Define the number of clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
cluster_labels = kmeans.fit_predict(subset_embeddings)

# Create a color palette
palette = sns.color_palette("husl", n_clusters)  # Use different colors for each cluster

# Create a figure with a larger size
plt.figure(figsize=(14, 10))

# Create a dendrogram
dendro = dendrogram(
    linked,
    orientation='top',
    labels=[f'Sample {i+1}' for i in range(len(cluster_labels))],  # Simple labels
    color_threshold=0,  # Keep this at 0 to allow for custom coloring
)

# Create a color mapping for the lines in the dendrogram
for i, d in enumerate(dendro['dcoord']):
    # Determine the cluster index based on the leaves
    left_idx = int(dendro['ivl'][i].split()[1]) - 1
    right_idx = int(dendro['ivl'][i + 1].split()[1]) - 1 if i + 1 < len(dendro['ivl']) else left_idx

    # Get the cluster colors based on the labels
    left_color = palette[cluster_labels[left_idx]]
    right_color = palette[cluster_labels[right_idx]]

    # Set the color for the line
    plt.plot(d[0:2], [d[2], d[2]], color=left_color, linewidth=2)
    plt.plot(d[0:2], [d[2], d[2]], color=right_color, linewidth=2)

# Customize title and labels
plt.title('Hierarchical Clustering Dendrogram with Cluster Colors', fontsize=16)
plt.xlabel('Samples', fontsize=12)
plt.ylabel('Distance', fontsize=12)

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


Hierarchical Clustering Dendrogram with Cluster Colors
Distance
0.0
0.5
1.0
1.5
2.0
2.5
Sample 17
Sample 19
Sample 23
Sample 20
Sample 22
Sample 28
Sample 24
Sample 27
Sample 16
Sample 25
Sample 21
Sample 29
Sample 10
Sample 13
Sample 30
Sample 2
Sample 5
Sample 3
Sample 6
Sample 4
Sample 1
Sample 9
Sample 15
Sample 26
Sample 11
Sample 8
Sample 12
Sample 14
Sample 7
Sample 18
Samples

Processing for array to integrate into interface

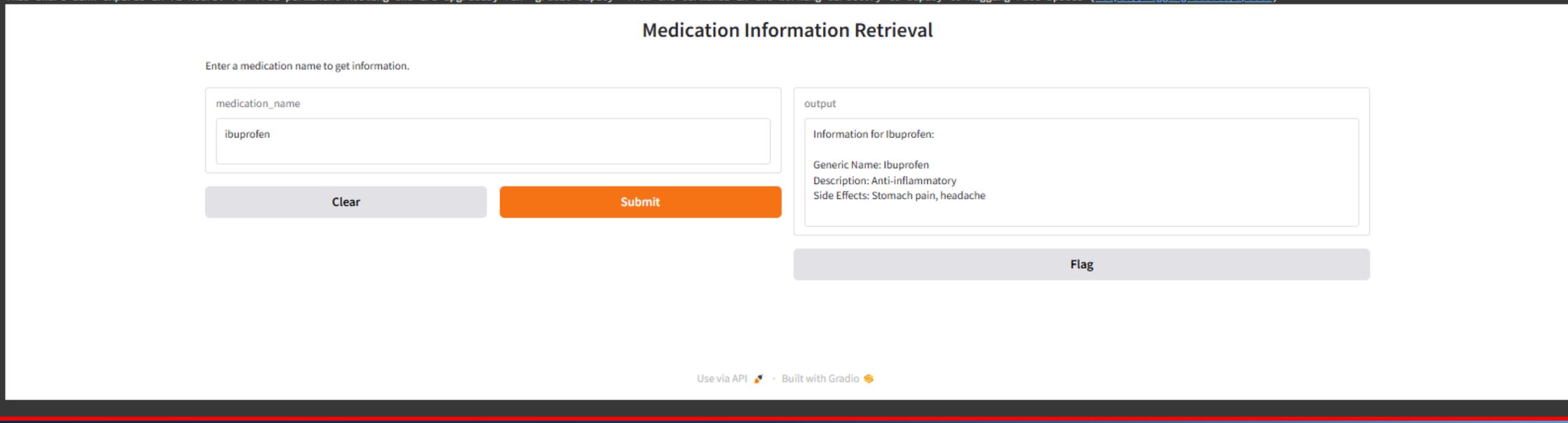
```python
import pickle
import numpy as np

with open("medication_embeddings.pkl", 'rb') as f:
    medication_embeddings = pickle.load(f)

print(type(medication_embeddings))
print(medication_embeddings)
print(f"Shape: {getattr(medication_embeddings, 'shape', 'Not a NumPy array')}")
```

```
<class 'pandas.core.series.Series'>
0       [0.012235959, -0.01992144, 0.042213798, -0.013...
1       [0.0049548494, -0.036637772, 0.028918743, -0.0...
2       [0.009790493, -0.04741547, 0.03854556, -0.0048...
3       [0.00704646, -0.051960606, 0.061372794, -0.023...
4       [0.04024248, -0.0754127, 0.033916738, -0.03136...
                              ...
1706    [-0.00091094984, -0.06592126, -0.0059444737, -...
1707    [0.01694808, -0.04926499, -0.010650718, -0.004...
1708    [0.02295714, -0.061518926, 0.036960784, -0.003...
1709    [0.03133975, -0.021267077, 0.018804662, 0.0008...
1710    [0.032387994, -0.039123856, 0.015060047, -0.01...
Name: side_effects_embeddings, Length: 1711, dtype: object
Shape: (1711,)
```

Gradio interface for Q&A as input & output

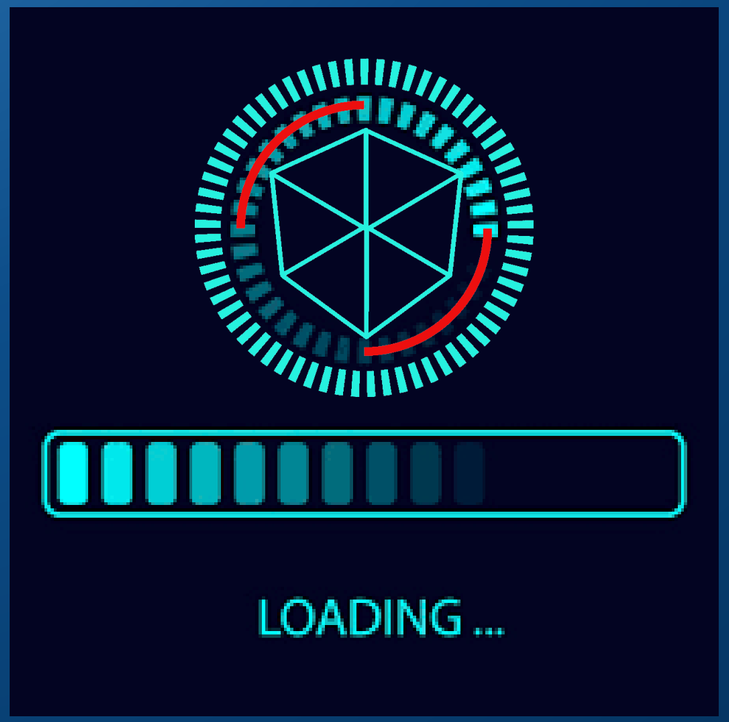
```
90
91 # Launch the interface
92 iface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://18b63979483208278b.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```



Testing the Model

Gradio Interface utilized to take user input and provide an answer for this drug Q & A concept



Future Work

- **What else can be done or improved**
  - Expand dataset for broader drug variety.
  - Optimize model for mobile devices.
  - Explore model interpretability for safer usage in healthcare.
  - Using more powerful GPUs to process more data to improve accuracy.
  - Provide images or video to help the user understand the drug in more detail.

Towards a Healthier & Efficient Future

- **Questions**

"Medical science has proven time and again that when the resources are provided, great progress in the treatment, cure, and prevention of disease can occur."