Sentiment Analysis for Mental Health Monitoring 😵

> dataset source 🗞

About The Datset 🖋

This comprehensive dataset is a meticulously curated collection of mental health statuses tagged from various statements. The dataset amalgamates raw data from multiple sources, cleaned and compiled to create a robust resource for developing chatbots and performing sentiment analysis.

Data Souce 🌎

The dataset integrates information from the following Kaggle datasets:

- 3k Conversations Dataset for Chatbot
- Depression Reddit Cleaned
- Human Stress Prediction
- Predicting Anxiety in Mental Health Data
- Mental Health Dataset Bipolar
- Reddit Mental Health Data
- Students Anxiety and Depression Dataset
- Suicidal Mental Health Dataset
- Suicidal Tweet Detection Dataset

Datset Over View :

- **Description:**

  This dataset is a comprehensive collection of 50,000 text statements related to mental health, each tagged with one of seven mental health statuses. The primary purpose of this dataset is to assist in building machine learning models for classifying mental health conditions based on textual data, such as social media posts or other user-generated content.

- **Columns:**

  1. **unique_id**: A unique identifier for each entry.
  2. **statement**: A piece of text, typically a statement or comment, associated with a particular mental health status.
  3. **status**: The mental health status assigned to the statement. The possible categories are:

     - Normal
     - Depression
     - Suicidal
     - Anxiety
     - Stress
     - Bi-Polar
     - Personality Disorder

Usage :

This dataset is ideal for training machine learning models aimed at understanding and predicting mental health conditions based on textual data. It can be used in various applications such as:

- Chatbot development for mental health support.
- Sentiment analysis to gauge mental health trends.
-

How we use NLP Concepts

In our text processing methodology, we begin by removing punctuation, URLs, and hyperlinks from the statements. Additionally, we eliminate stop words, such as "is," "are," and "the," to enhance the focus on more informative terms.

In the realm of morphological analysis, we extract the stems of the words. Subsequently, we employ a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer to transform the processed text into a vector representation. This vector is then utilized to fit the mode

---

Conlusion

in the conclusion we train and tune a four model wich is :

- SVM
- Logistic Regression
- Neural Ntwork
- KNN the first three give a Convergent results with accuracy near to 76 % , and the last one give a 65 % acc .

Import Libraries :

---

lets start by importing the nessacary libraries

[ ] ↳ *3 cells hidden*

Datasert Loading and Preparing

[ ] ↳ *22 cells hidden*

NLP Pre-Processing

---

in this part of the ml pipeline, we perform several essential NLP preprocessing steps to prepare our data for machine learning. These steps include:

1. **Text Processing:**
    - include :
        - pattern reomving (links,emails and etc.)
        - lower casing
        - stop words removing
2. **tokinezation & Stemming :**
    - Tokenization :
        - this step converts each text sample into an array of tokens (individual words or sub-words).
        - we use Stanza word tokenizer to ensure precise, language-specific tokenization.
    - Stemming :
        - in this step, we transform each tokenized array into a stemmed version, reducing each word to its base or root form.
        - stanza NLP tools allow us to use various stemming techniques for extracting meaningful root words.
3. **Part-of-Speech (POS) Tagging and Filtering**:

- sing Stanza, we apply POS tagging to identify and filter specific parts of speech, such as:
    - **Verbs**: Identify and include only verbs to capture action-related semantics.
    - **Nouns**: Extract nouns to focus on entity- or object-based information.
    - **Adjectives**: Include adjectives to analyze descriptive language and sentiment.
- This POS filtering allows us to tailor the input data by emphasizing different linguistic elements and can be customized based on the classification or NLP goals.

[ ] ↳ *56 cells hidden*

Data Spliting

[ ] ↳ *6 cells hidden*

Vectorizing

- Convert text to features using TF-IDF vectoriser

Vectorizing the stemmes

[ ] ↳ *1 cell hidden*

Verbs-Vector

[ ] ↳ *1 cell hidden*

Nouns-Vector

[ ] ↳ *2 cells hidden*

Adjectives-Vector

[ ] ↳ *1 cell hidden*

Weighted-Vector

```
# Define weights for each POS feature type
verb_weight = 0.5
noun_weight = 3.0
adj_weight = 1.8

# Apply weights and combine features
X_weighted_train = hstack([
    X_verbs_train_tfidf * verb_weight,
    X_nouns_train_tfidf * noun_weight,
    X_adj_train_tfidf * adj_weight
])
# Apply weights and combine features
X_weighted_test = hstack([
    X_verbs_test_tfidf * verb_weight,
    X_nouns_test_tfidf * noun_weight,
    X_adj_test_tfidf * adj_weight
])
```

Resampling very importing remmember

[ ] ↳ *2 cells hidden*

Model Trainging

Prepare The Models

[ ] ↳ *4 cells hidden*

Training Based on Verb-Vector

[ ] ↳ *16 cells hidden*

Training Based on Nouns-Vector

[ ] ↳ *15 cells hidden*

Training Based on Adjective-Vector

[ ] ↳ *16 cells hidden*

Training Based on weighted-Vector

Logistic Regression Model

Grid CV

```
# Fit the grid search on the training data
grid_search.fit(X_weighted_train, y_train)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

GridSearchCV

best_estimator_: LogisticRegression

LogisticRegression

Best params

```
# Get the best model and hyperparameters
best_clf = grid_search.best_estimator_
print("Best hyperparameters found: ", grid_search.best_params_)
```

```
Best hyperparameters found:  {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

Confusion Matrix

```
#  Predict on the test set using the best model
y_pred = best_clf.predict(X_weighted_test)

#  Calculate accuracy
accuracy_reg = accuracy_score(y_test, y_pred)
print("\nAccuracy: ", accuracy_reg)

#  Compute the confusion matrix and classification report
conf_matrix_reg = confusion_matrix(y_test, y_pred)
labels = lbl_enc.classes_
print(classification_report(y_test, y_pred, target_names=labels))
```

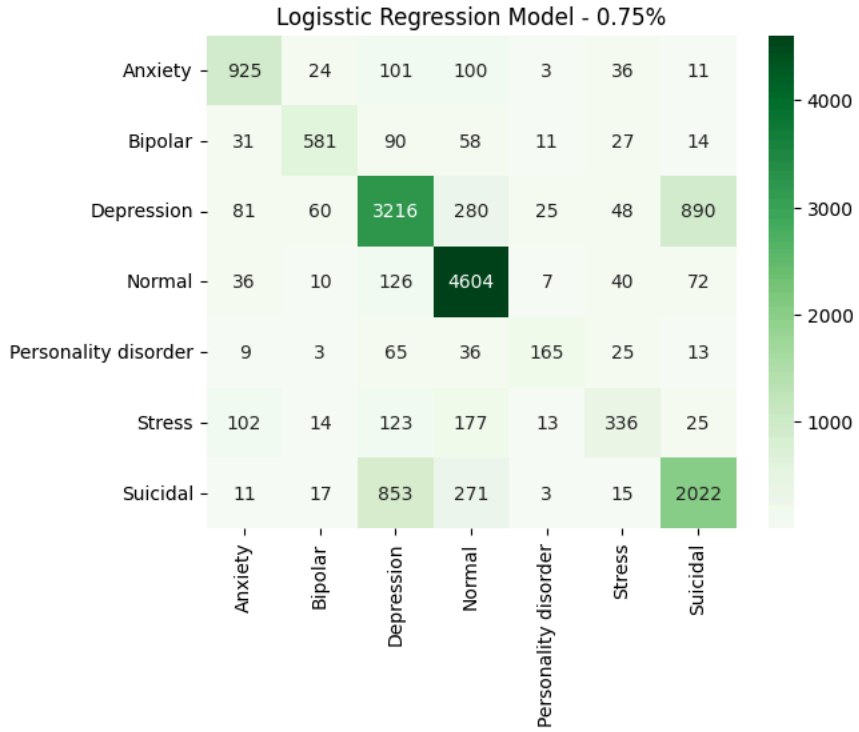
```
Accuracy:  0.7496994621955078
                      precision    recall  f1-score   support

             Anxiety       0.77      0.77      0.77      1200
             Bipolar       0.82      0.72      0.76       812
          Depression       0.70      0.70      0.70      4600
              Normal       0.83      0.94      0.88      4895
Personality disorder       0.73      0.52      0.61       316
              Stress       0.64      0.43      0.51       790
            Suicidal       0.66      0.63      0.65      3192

            accuracy                           0.75     15805
           macro avg       0.74      0.67      0.70     15805
        weighted avg       0.74      0.75      0.74     15805
```

Heat Map

```
ax = sns.heatmap(conf_matrix_reg, annot = True, fmt='d', cmap='Greens', xticklabels=labels, yticklabels=labels)
ax.set_title(f'Logisstic Regression Model - {accuracy_reg:.2}%')
plt.show()
```



Nerual Network

```
# Define the layers in an array
layers = [
    Dense(units=128, activation='relu', input_shape=(X_weighted_train.shape[1],),kernel_regularizer=l2(0.01)),  # Input Layer
    BatchNormalization(),
    Dropout(rate=0.2),  # Dropout Layer 1
    Dense(units=64, activation='relu',kernel_regularizer=l2(0.01)),  # Hidden Layer 1
    Dropout(rate=0.1),  # Dropout Layer 1
    Dense(units=16, activation='relu',kernel_regularizer=l2(0.01)),  # Hidden Layer 2

    Dropout(rate=0.2),  # Dropout Layer 2
    Dense(units=len(lbl_enc.classes_), activation='softmax')  # Output Layer
]

# Initialize the Sequential model
model = Sequential(layers)


# Compile the model: Using Adam optimizer, sparse categorical crossentropy loss, and accuracy as the metric
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])




# Train the model on training data with validation on the test data
history = model.fit(X_weighted_train.toarray(),
                    y_train,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.3
                    )
```

```
Epoch 1/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.4709 - loss: 3.2663 - val_accuracy: 0.6932 - val_loss: 1.6621
Epoch 2/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.7019 - loss: 1.3552 - val_accuracy: 0.7050 - val_loss: 1.1489
Epoch 3/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.7186 - loss: 1.1411 - val_accuracy: 0.7161 - val_loss: 1.0911
Epoch 4/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7219 - loss: 1.1097 - val_accuracy: 0.7198 - val_loss: 1.0703
```

```
Epoch 5/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.7320 - loss: 1.0864 - val_accuracy: 0.7171 - val_loss: 1.0844
Epoch 6/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 8s 8ms/step - accuracy: 0.7308 - loss: 1.0847 - val_accuracy: 0.7239 - val_loss: 1.0661
Epoch 7/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.7374 - loss: 1.0803 - val_accuracy: 0.7280 - val_loss: 1.0580
Epoch 8/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.7342 - loss: 1.0610 - val_accuracy: 0.7247 - val_loss: 1.0830
Epoch 9/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.7503 - loss: 1.0532 - val_accuracy: 0.7216 - val_loss: 1.0740
Epoch 10/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7513 - loss: 1.0317 - val_accuracy: 0.7274 - val_loss: 1.0673
Epoch 11/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7537 - loss: 1.0351 - val_accuracy: 0.7258 - val_loss: 1.0550
Epoch 12/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7580 - loss: 1.0141 - val_accuracy: 0.7262 - val_loss: 1.0695
Epoch 13/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7675 - loss: 1.0088 - val_accuracy: 0.7275 - val_loss: 1.0658
Epoch 14/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.7652 - loss: 0.9951 - val_accuracy: 0.7302 - val_loss: 1.0553
Epoch 15/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.7729 - loss: 0.9808 - val_accuracy: 0.7324 - val_loss: 1.0519
Epoch 16/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7730 - loss: 0.9834 - val_accuracy: 0.7329 - val_loss: 1.0481
Epoch 17/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.7767 - loss: 0.9639 - val_accuracy: 0.7283 - val_loss: 1.0608
Epoch 18/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.7808 - loss: 0.9595 - val_accuracy: 0.7275 - val_loss: 1.0564
Epoch 19/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.7791 - loss: 0.9570 - val_accuracy: 0.7303 - val_loss: 1.0562
Epoch 20/20
404/404 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.7849 - loss: 0.9454 - val_accuracy: 0.7274 - val_loss: 1.0661
```

```
# Make predictions on the test set
y_pred_prob = model.predict(X_weighted_test.toarray())
y_pred = y_pred_prob.argmax(axis=1)  # Convert probabilities to class predictions

# Calculate the accuracy
accuracy_nn = accuracy_score(y_test, y_pred)
print("\n")
print("Accuracy:", accuracy_nn)

# Compute the confusion matrix
labels = lbl_enc.classes_
conf_matrix_nn = confusion_matrix(y_test, y_pred)

# Print classification report
print(classification_report(y_test, y_pred, target_names=labels))
```

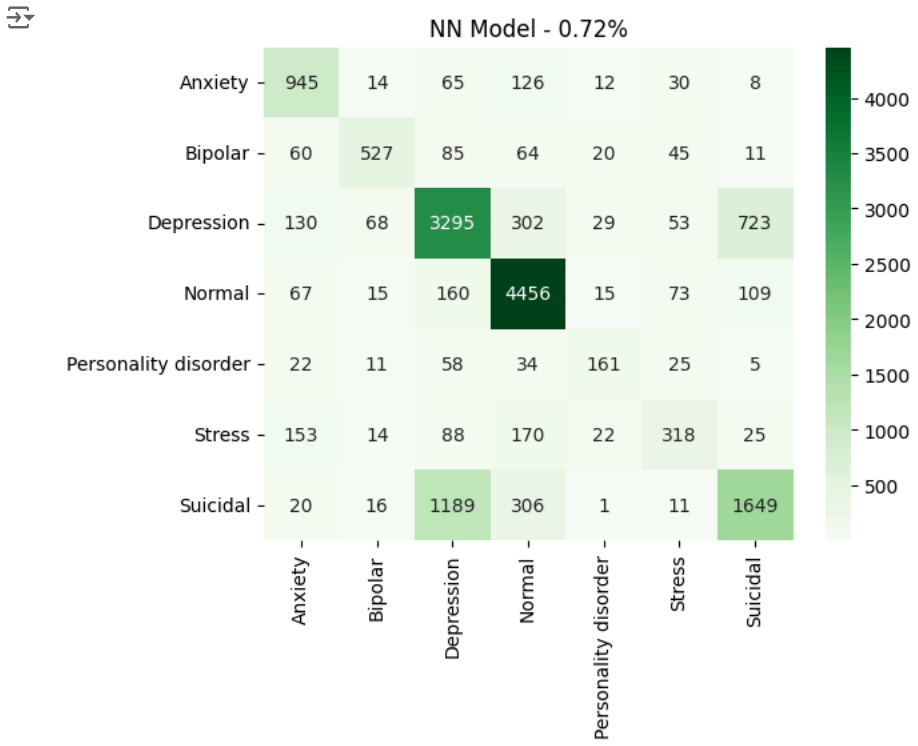
```
494/494 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step


Accuracy: 0.7181904460613 73
                      precision    recall  f1-score   support

             Anxiety       0.68      0.79      0.73      1200
             Bipolar       0.79      0.65      0.71       812
          Depression       0.67      0.72      0.69      4600
              Normal       0.82      0.91      0.86      4895
Personality disorder       0.62      0.51      0.56       316
              Stress       0.57      0.40      0.47       790
            Suicidal       0.65      0.52      0.58      3192

            accuracy                           0.72     15805
           macro avg       0.69      0.64      0.66     15805
        weighted avg       0.71      0.72      0.71     15805
```

```
ax = sns.heatmap(conf_matrix_nn, annot = True, fmt='d', cmap='Greens', xticklabels=labels, yticklabels=labels)
ax.set_title(f'NN Model - {accuracy_nn:.2}%')
plt.show()
```

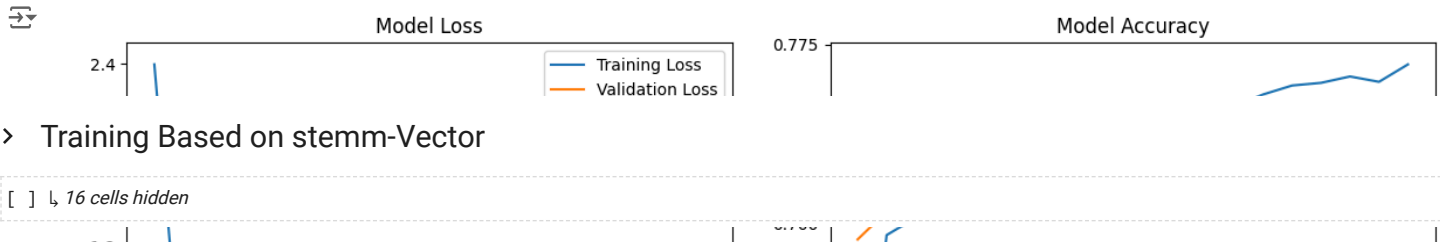



```
plt.figure(figsize=(12, 5))

# Plot training & validation loss values
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

plt.tight_layout()  # Adjusts subplots to fit in the figure area.
plt.show()  # Display the plots
```



Training Based on stemm-Vector

[ ] ↳ *16 cells hidden*

End