There is NO official lrp package than can be used.

grad-cam can in fact be used with transformers with the "official" package.

grad-cam can be used with "vision" transformers with the "official" package.

grad-cam can't be used with classical ml

Lime and SHAP don't care about the type of model.

grad-cam works only on pytorch

https://github.com/albermax/innvestigate is an lrp implementation using keras

https://github.com/chr5tphr/zennit is an lrp implementation using pytorch

https://github.com/jacobgil/pytorch-grad-cam is a grad-cam pytorch implementation

Both lrp and grad-cam have relatively complete implementation for CNNs

Both can be used in NLP but with CNN models only (no attention-based or transformer-based models)

Both have incomplete implementation in case of attention-based or transformer-based models

Both LRP and grad-cam CAN be used with attention-based or transformer-based models but we will need to implement them from scratch

logistic regression need much more iterations than the default value to successfully converge (now it is set to 10,000 iterations maximum)

Shoutout for https://github.com/yidinghao/interpreting-nlp/tree/master for saving the day and enabling us to do the LRP

Embedding needs to be done in batches on GPU

logistic regression with light stemming vectorizer:

	precision	recall	f1-score	support
none	0.60	0.94	0.73	307
anger	0.50	0.78	0.61	276
joy	0.65	0.37	0.47	268
sadness	0.51	0.31	0.39	258
love	0.66	0.68	0.67	250
sympathy	0.78	0.80	0.79	194
surprise	0.69	0.37	0.48	201
fear	0.98	0.85	0.91	259
accuracy			0.65	2013
macro avg	0.67	0.64	0.63	2013
weighted avg	0.66	0.65	0.63	2013

logistic regression with root stemming vectorizer:

	precision	recall	f1-score	support
none	0.60	0.89	0.72	307
anger	0.52	0.77	0.62	276
joy	0.57	0.36	0.44	268
sadness	0.57	0.39	0.46	258
love	0.72	0.70	0.71	250
sympathy	0.76	0.82	0.79	194
surprise	0.62	0.38	0.47	201
fear	0.98	0.88	0.93	259
accuracy			0.66	2013
macro avg	0.67	0.65	0.64	2013
weighted avg	0.66	0.66	0.64	2013

Naïve bayes with Light Stemming vectorizer:

	precision	recall	f1-score	support
none	0.43	0.96	0.60	307
anger	0.60	0.67	0.64	276
joy	0.60	0.31	0.41	268
sadness	0.68	0.25	0.37	258
love	0.64	0.70	0.67	250
sympathy	0.82	0.76	0.79	194
surprise	0.86	0.21	0.34	201
fear	0.82	0.89	0.85	259
accuracy			0.61	2013
macro avg	0.68	0.60	0.58	2013
weighted avg	0.66	0.61	0.58	2013

Naïve bayes with root Stemming vectorizer:

	precision	recall	f1-score	support
none	0.46	0.93	0.61	307
anger	0.56	0.73	0.63	276
joy	0.63	0.31	0.41	268
sadness	0.57	0.28	0.37	258
love	0.66	0.72	0.69	250
sympathy	0.77	0.74	0.76	194
surprise	0.83	0.17	0.28	201
fear	0.85	0.87	0.86	259
accuracy			0.61	2013
macro avg	0.67	0.59	0.58	2013
weighted avg	0.65	0.61	0.58	2013

logistic regression with light stemming embedding:

```
Accuracy: 0.7721063089915549
```

```
# test accuracy of light stemmer embeddings
y_pred = log_reg_ls.predict(x_test_emb)
# calculating the accuracy of the classifier
accuracy = accuracy_score(df_test['label'], y
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7069051167411823
```

```
print(classification_report(df_test['label'],
```

```
              precision    recall  f1-score   support

        none       0.63      0.90      0.74       307
       anger       0.67      0.79      0.73       276
         joy       0.65      0.54      0.59       268
     sadness       0.72      0.48      0.58       258
        love       0.76      0.77      0.76       250
    sympathy       0.78      0.83      0.80       194
    surprise       0.60      0.43      0.50       201
        fear       0.87      0.85      0.86       259

    accuracy                           0.71      2013
   macro avg       0.71      0.70      0.70      2013
weighted avg       0.71      0.71      0.70      2013
```

```
from sklearn.metrics import confusion_matrix

# Create a confusion matrix
cm = confusion_matrix(df_test['label'], y_pred)
print(cm)
# Plot the confusion matrix
```

```
[[275   4  12   6   0   0   9   1]
 [ 19 219   6   7   3  11   8   3]
 [ 42  14 144   2  33  13  16   4]
 [ 44  36  10 124  14   6  14  10]
 [  1   5  28   5 193  11   4   3]
 [  1  11   7   6   3 161   2   3]
 [ 47  31  10  11   4   3  86   9]
 [  5   7   4  11   5   1   5 221]]
```

logistic regression with root stemming embedding:

```
Accuracy: 0.6882762046696473
```

```python
# test accuracy of light stemmer embeddings
y_pred = log_reg_ls.predict(x_test_emb)
# calculating the accuracy of the classifier
accuracy = accuracy_score(df_test['label'], y_pr
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6318926974664679
```

```python
print(classification_report(df_test['label'], y_
```

```
              precision    recall  f1-score   support

        none       0.59      0.91      0.72       307
       anger       0.60      0.76      0.67       276
         joy       0.53      0.38      0.44       268
     sadness       0.57      0.35      0.43       258
        love       0.69      0.71      0.70       250
    sympathy       0.68      0.70      0.69       194
    surprise       0.45      0.29      0.35       201
        fear       0.86      0.85      0.85       259

    accuracy                           0.63      2013
   macro avg       0.62      0.62      0.61      2013
weighted avg       0.62      0.63      0.61      2013
```

```python
from sklearn.metrics import confusion_matrix

# Create a confusion matrix
cm = confusion_matrix(df_test['label'], y_pred)
print(cm)
# Plot the confusion matrix
```

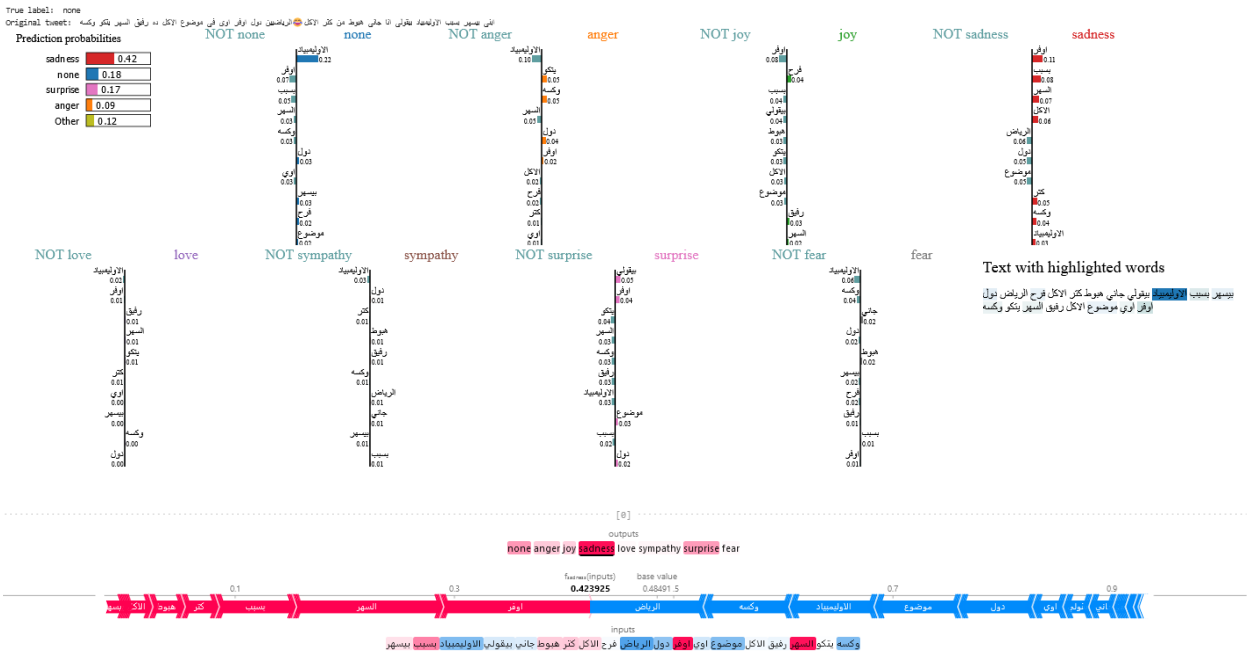
```
[[280   5  10   7   0   2   3   0]
 [ 19 209  10   7   6  11  10   4]
 [ 64  21 101  10  26  16  26   4]
 [ 54  39  14  91  25  10  19   6]
 [  0  13  22  16 177  14   4   4]
 [  1  19   9  12   7 135   4   7]
 [ 50  32  20  12  10   7  58  12]
 [  5   8   3   6   6   4   6 221]]
```

Examples:

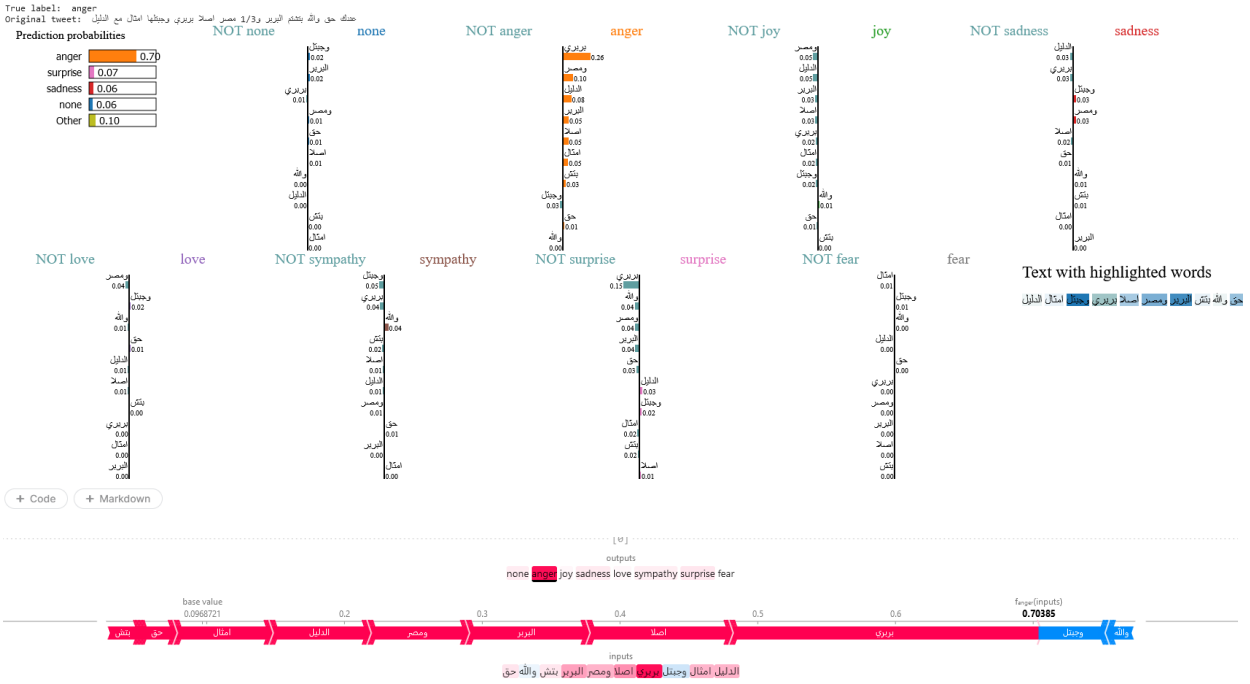
None predicted correctly:

We can observe that the prediction is true, but only because of the “الاوليمبياد” word

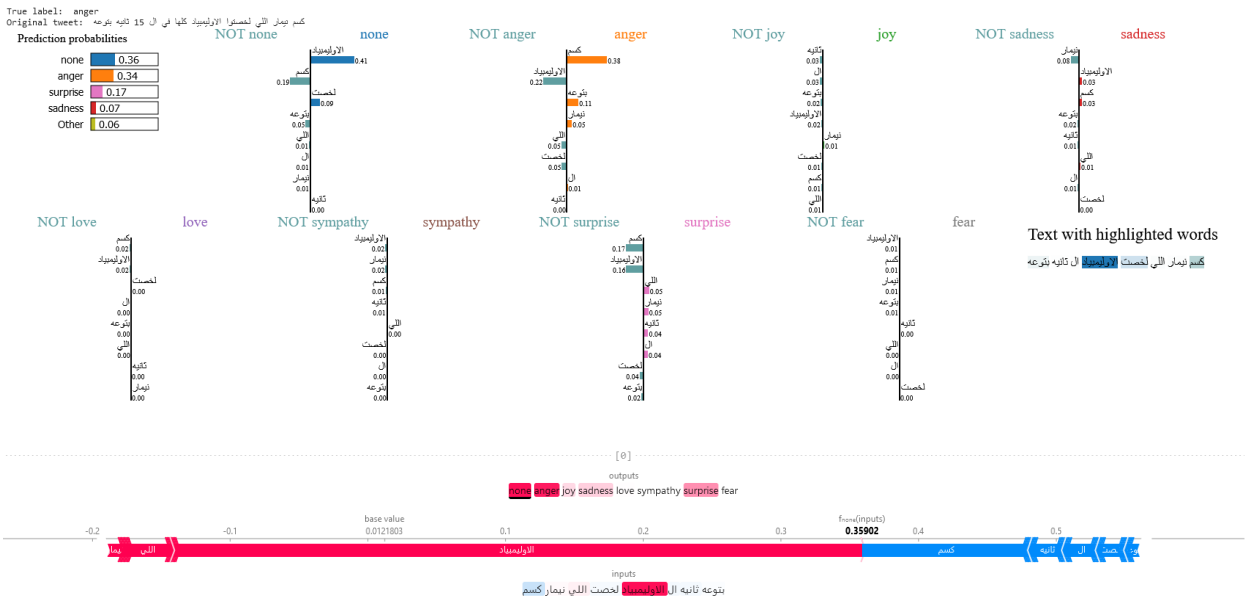
None predicted incorrectly



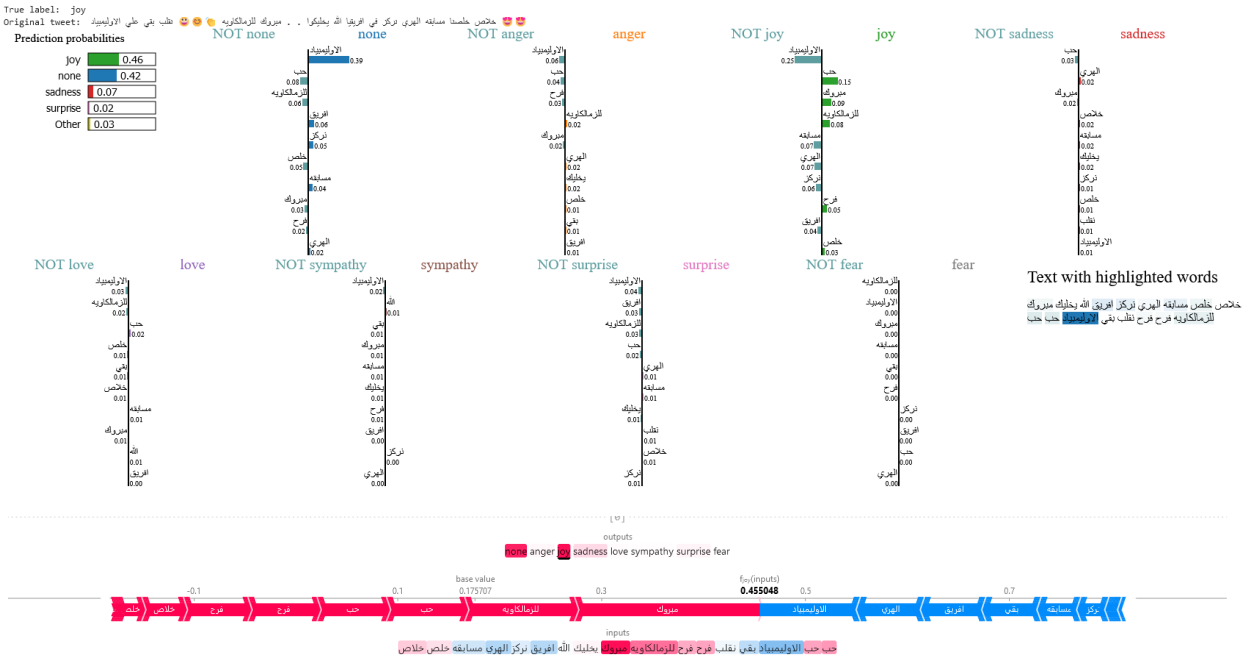
anger predicted correctly



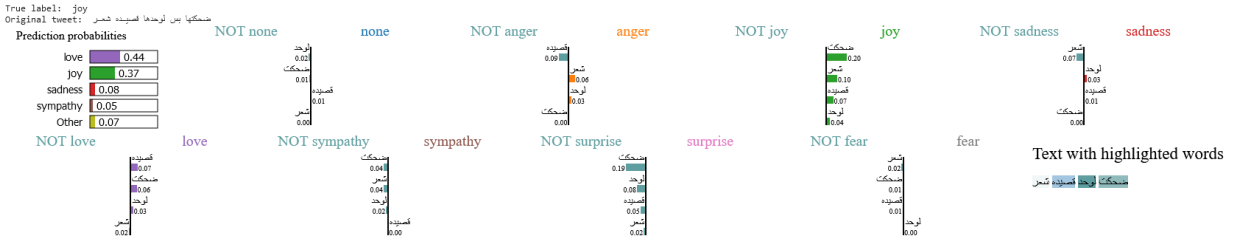
anger predicted incorrectly

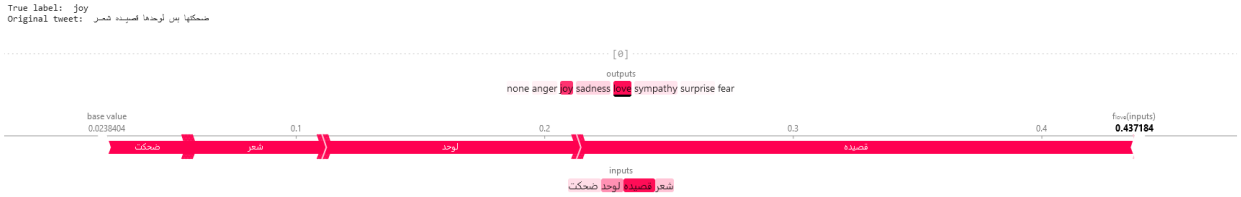


joy predicted correctly



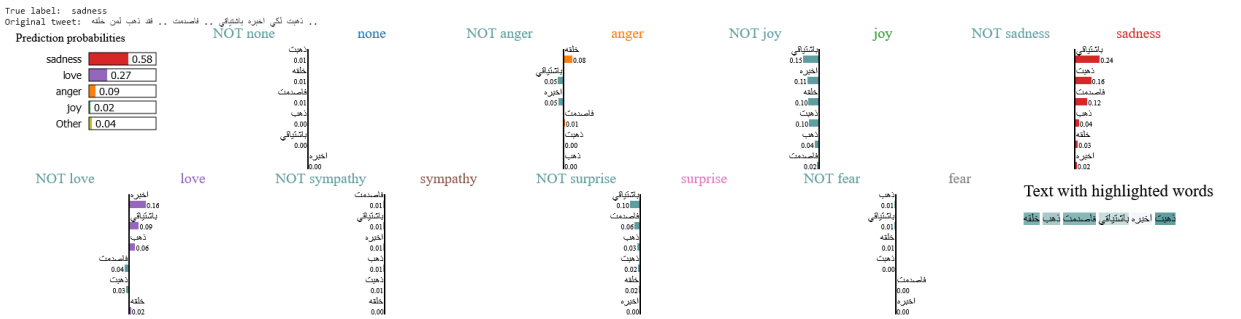
joy predicted incorrectly



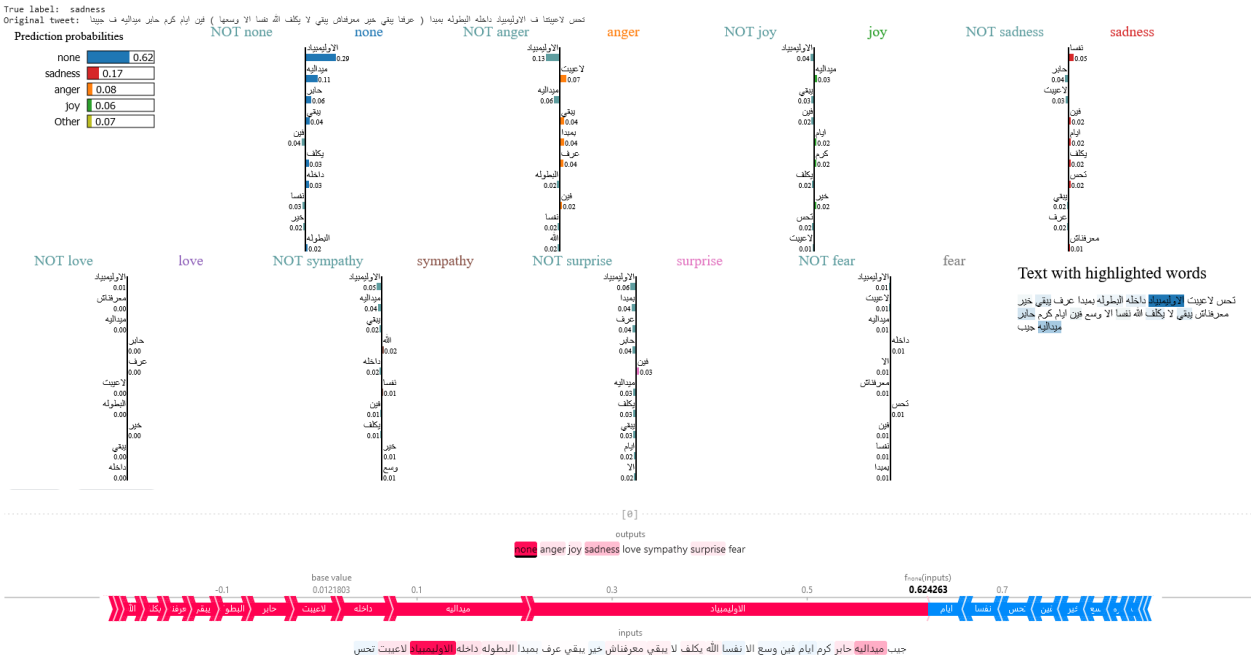


In this case the predicted label is different than the true label but the model, surprisingly, is not actually that wrong.

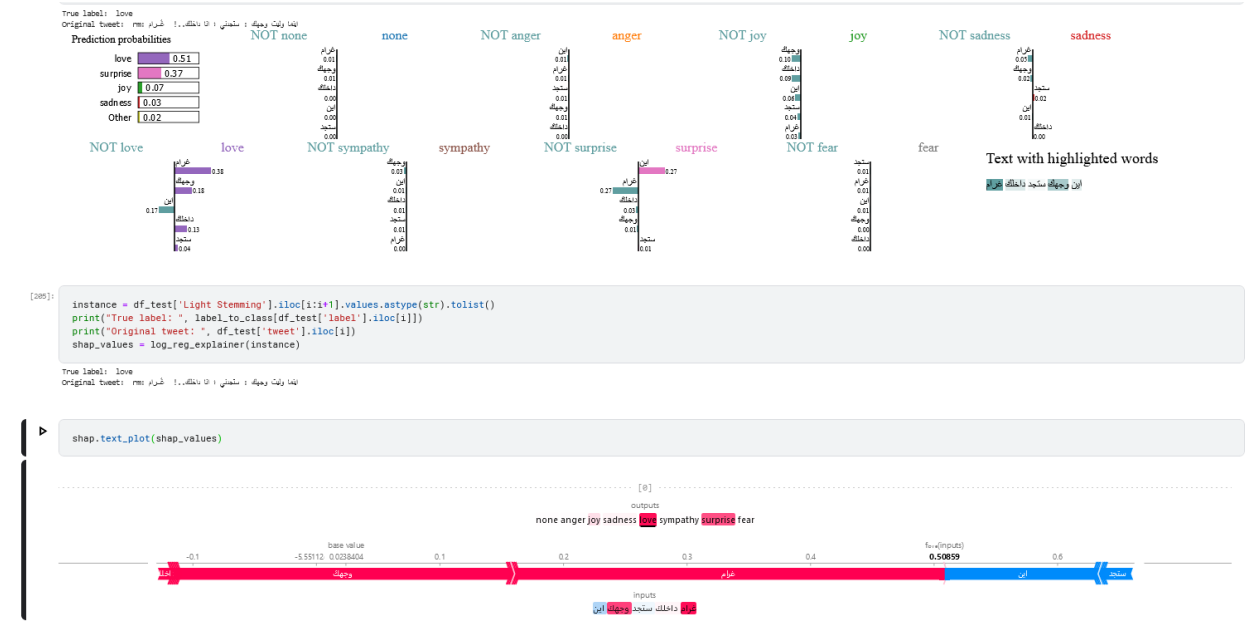
sadness predicted correctly



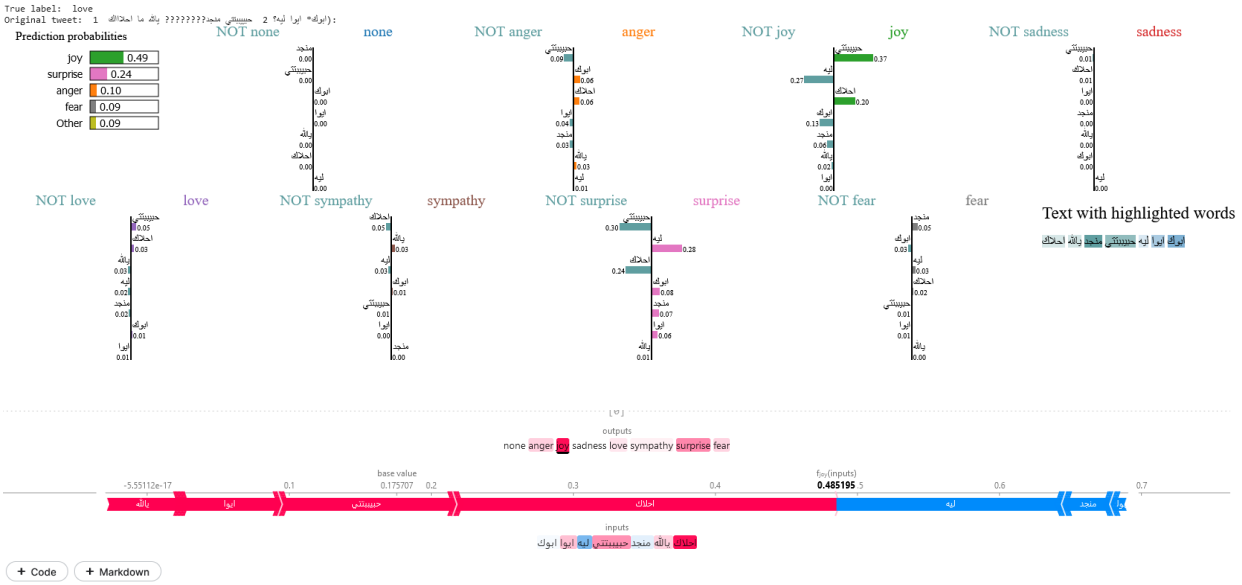
sadness predicted incorrectly



love predicted correctly

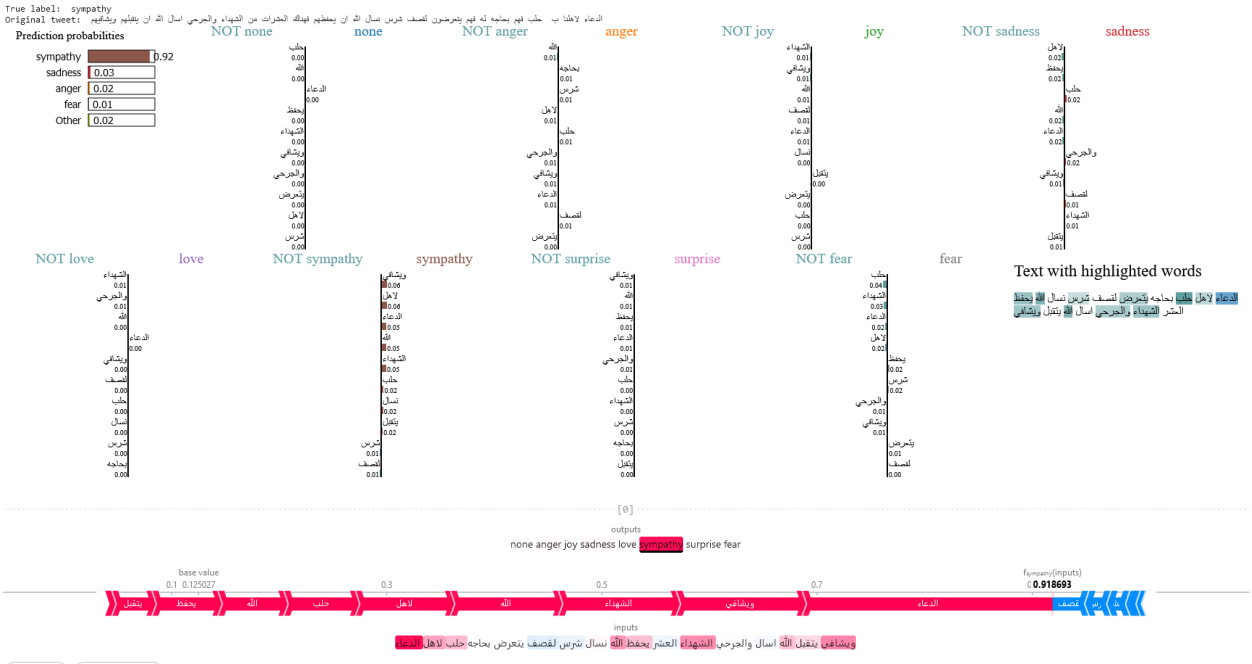


love predicted incorrectly

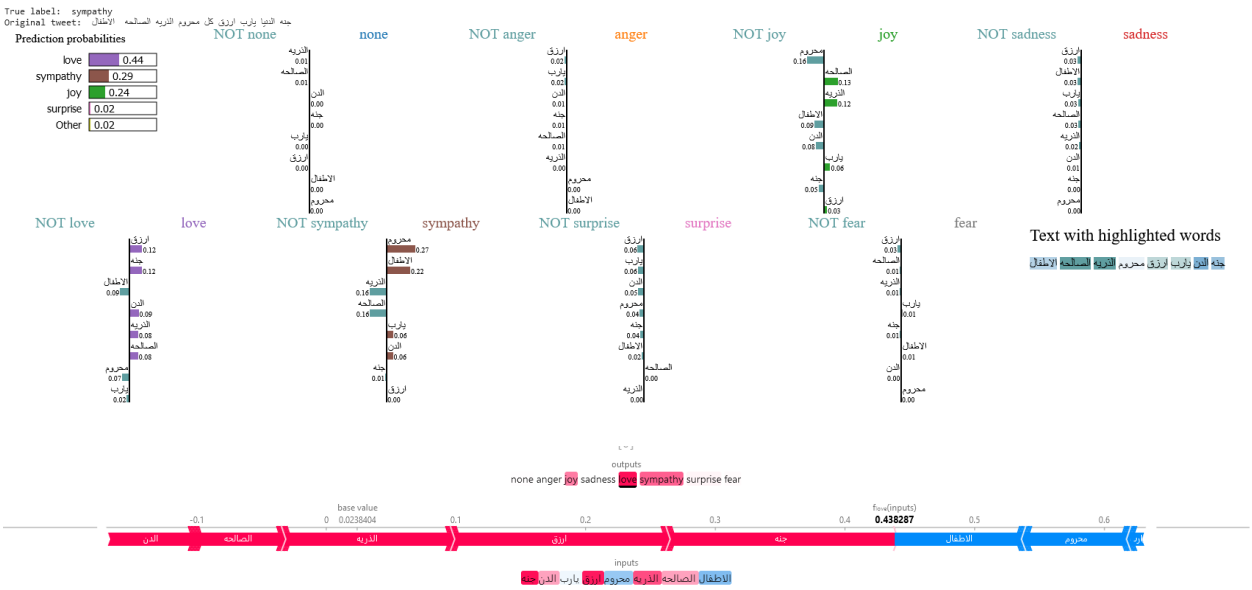


+ Code + Markdown

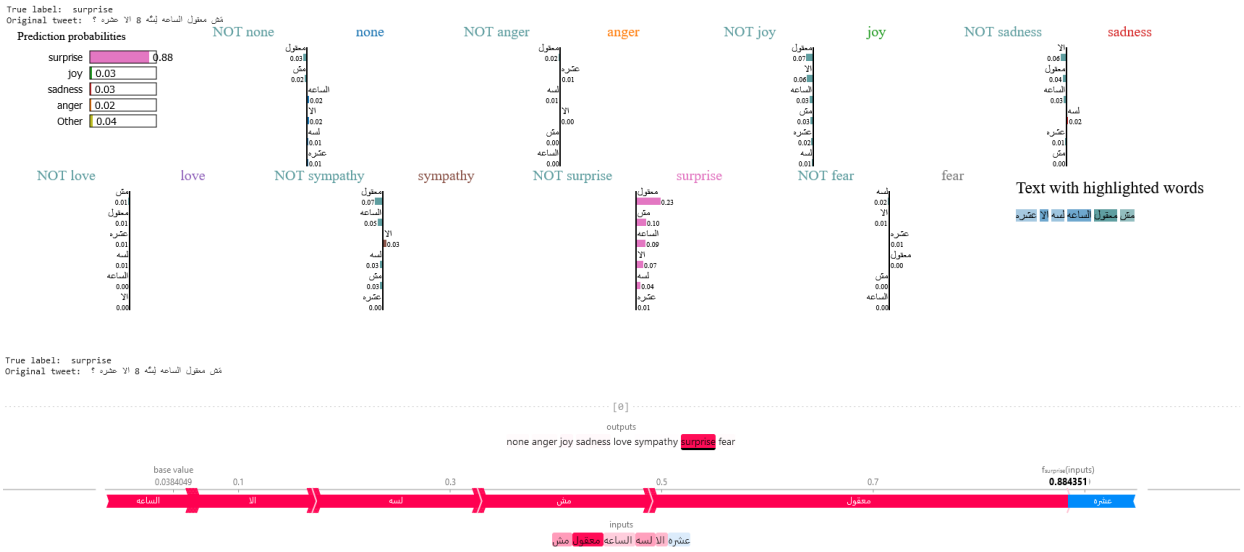
sympathy predicted correctly



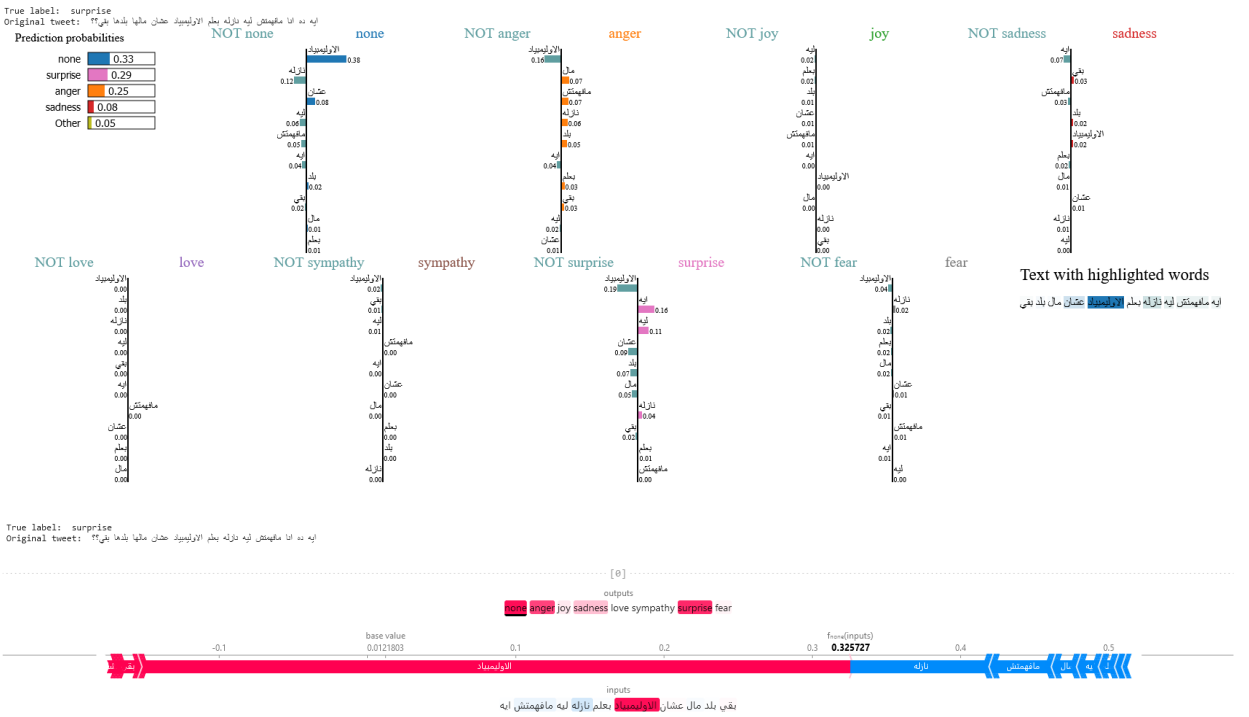
sympathy predicted incorrectly



Surprise predicted correctly



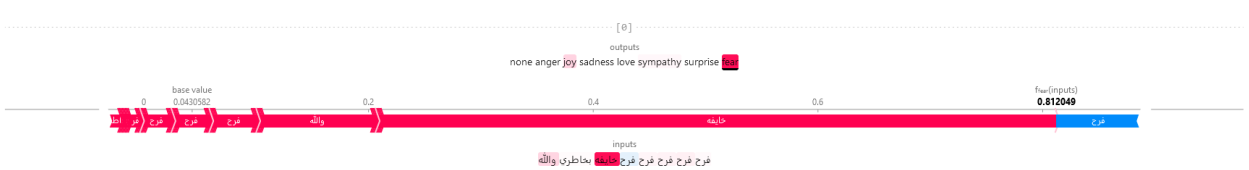
Surprise predicted incorrectly



“الاوليمبياد” strikes once more.

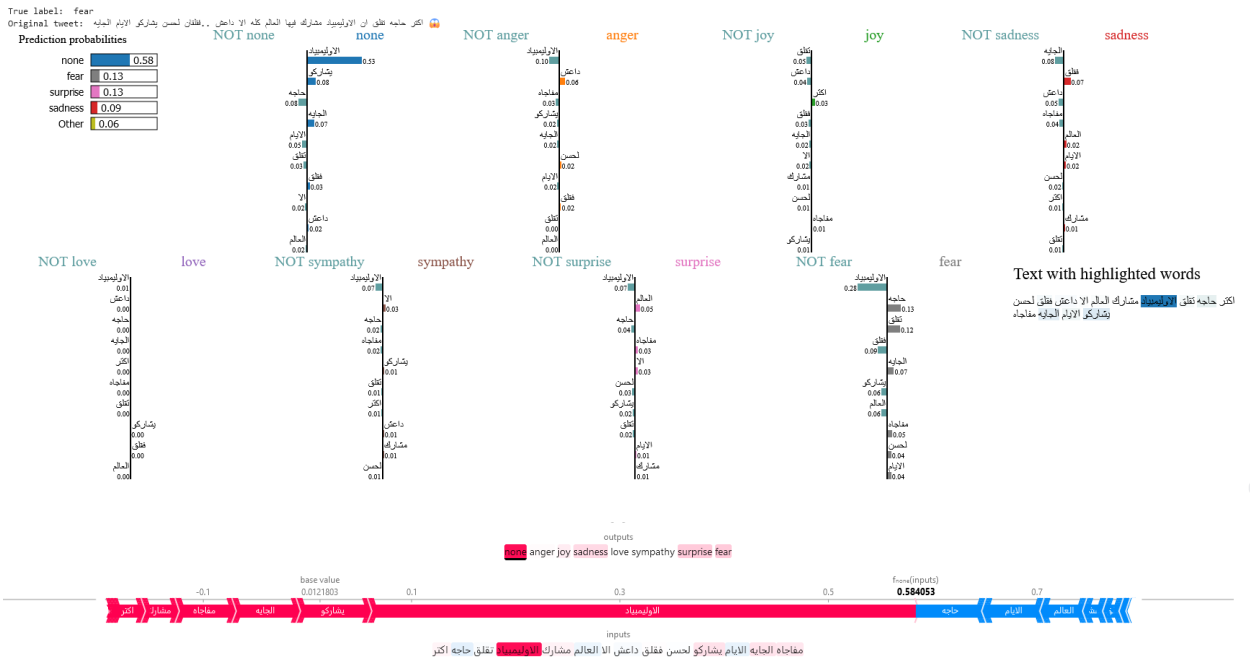
Fear predicted correctly


```
instance = df_test['Light Stemming'].iloc[i:i+1].values.astype(str).tolist()
print("True label: ", label_to_class[df_test['label'].iloc[i]])
print("Original tweet: ", df_test['tweet'].iloc[i])
shap_values = log_reg_explainer(instance)
shap.text_plot(shap_values)
```



Incorrect use of emojis

Fear predicted incorrectly



Notes:

1. Light stemming is much better
2. WITH tokens is actually better than without
3. Some words(“الاوليمبياد”) mess up the prediction
4. Some emotions are close to one another and have a high chance of being confused together (ex. Sympathy and sadness, love and joy, sadness and anger, none and all)
5. Some samples are labeled incorrectly
6. Some samples’ labels ambiguous.
7. Sometimes samples are labeled correctly for weird reasons