16.Confusion Matrix:

Next we are going to create a confusion matrix which is a 2 by 2 diagnostic tool displayed 4 different outcomes. The 4 outcomes are:

- True Positive: The predicted value matches the actual value. The actual value was positive and the model predicted a positive value
- True Negative: The predicted value matches the actual value. The actual value was negative and the model predicted a negative value
- False Positive (Type 1 Error): The predicted value was falsely predicted. The actual value was negative but the model predicted a positive value.
- False Negative (Type 2 Error): The predicted value was falsely predicted. The actual value was positive but the model predicted a negative value

Do create a confusion matrix firstly make sure to import “from sklearn.metrics import ConfusionMatrixDisplay” at the top of your code. In your LogisticRegression function add near the bottom of your code your working title lists for all the types of normalization we will do. We then use a for loop to go through the lists of normalization titles and set the variables to match. It should look like this:

```
def confusion_matrix(lang, input_df, show=False):
    x = input_df.drop(["spammer", "label"], axis=1).values
    if "id" in input_df.columns:
        x = input_df.drop(["id", "CAP", "spammer", "label"], axis=1).values
    y = input_df["label"].values

    x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=0.30, random_state=100)

    x_train_scaled = preprocessing.scale(x_train)
    x_test_scaled = preprocessing.scale(x_test)

    model = LogisticRegression()
    model.fit(x_train_scaled, y_train)

    title_options = [(f"Confusion Matrix, without normalization\n({lang})", None),
                     (f"Normalization: true\n({lang})", "true"),
                     (f"Normalization: pred\n({lang})", "pred"),
                     (f"Normalization: all\n({lang})", "all")]

    for title, normalize in title_options:
        disp = ConfusionMatrixDisplay.from_estimator(model, x_test_scaled, y_test, display_labels=["bot", "human"],
                                                     cmap=plt.cm.Blues, normalize=normalize)
        disp.ax_.set_title(title)
        plt.savefig(fr'C:\Users\Rachel\Documents\Twitter Data\{normalize}_{lang}_test')

    if show:
        plt.show()
```

We then create a function main that uses the orgs_to_bots method to create a new dataframe with all data framethat calls the remove_outliers method to clear the data frame from any outliers. The logistic regression we made earlier is then used on the new outlier-free data frame. Then confusion_matrix is called to plot and show all of the confusion matrices for the logistic regression method. All the method implemented above were not called except in the method we now created called main. Therefore at the end of the code we need to call both best_zscore and main to show us the work we have done! Because of the parameters in best_zscore(zscore, lang, dfpath, whether to show plot) we go with a z-score of 3, then english represented by “en”, then the input_path we started with for english being “input_path_en” then True since we want to show the z-score figures. We repeat this again for universal language. Using the best z-score from that method we plug in the best z-score for the main method which houses our logistic regression and confusion matrix methods. We put in 1.7 for English and 1.4 for universal. It should look like this (Link for GitHub code base for use!):

```python
def main(zscore, lang, input_path, output_path, show_matrix=False, print_result=False,):
    df = orgs_to_bots(input_path)
    df_trimmed = remove_outliers(zscore, df, output_path)
    logistic_regression(df_trimmed, print_result)
    confusion_matrix(lang, df_trimmed, show_matrix)


best_zscore(3, "en", input_path_en, True)
best_zscore(3, "un", input_path_un, True)
main(1.7, "en", input_path_en, en_path_trimmed)
main(1.4, "un", input_path_un, un_path_trimmed, True, True)
```

Congratulations you are now done!