Ananya Singh
Assignment #5

Question #1 Part 1
I used xlrd (pip3 install xlrd) to process the xlsx dataset with the pandas engine.

```
data.reset_index(drop=True, inplace=True)
print(data.tail())
✓ [18] 275ms

          FileName        Date      SegFile       b       e    LBE     LB   AC  \
2121  S8001045.dsp 1998-06-06  CTG2124.txt  2059.0  2867.0  140.0  140.0  0.0
2122  S8001045.dsp 1998-06-06  CTG2125.txt  1576.0  2867.0  140.0  140.0  1.0
2123  S8001045.dsp 1998-06-06  CTG2126.txt  1576.0  2596.0  140.0  140.0  1.0
2124  S8001045.dsp 1998-06-06  CTG2127.txt  1576.0  3049.0  140.0  140.0  1.0
2125  S8001045.dsp 1998-06-06  CTG2128.txt  2796.0  3415.0  142.0  142.0  1.0

       FM   UC  ...    C    D    E   AD   DE   LD   FS  SUSP  CLASS  NSP
2121  0.0  6.0  ...  0.0  0.0  1.0  0.0  0.0  0.0  0.0   0.0    5.0  2.0
2122  0.0  9.0  ...  0.0  0.0  1.0  0.0  0.0  0.0  0.0   0.0    5.0  2.0
2123  0.0  7.0  ...  0.0  0.0  1.0  0.0  0.0  0.0  0.0   0.0    5.0  2.0
2124  0.0  9.0  ...  0.0  0.0  1.0  0.0  0.0  0.0  0.0   0.0    5.0  2.0
2125  1.0  5.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0    1.0  1.0

[5 rows x 40 columns]
```

Question #1 Part 2

```
data.loc[data['NSP'] == 2, 'NSP'] = 0
data.loc[data['NSP'] == 3, 'NSP'] = 0

print(data['NSP'].value_counts())
print(data.head(40))
✓ [105] 23ms

NSP
1.0    1655
0.0     471
Name: count, dtype: int64
        FileName        Date      SegFile      b       e    LBE     LB   AC  \
0   Variab10.txt 1996-12-01  CTG0001.txt  240.0   357.0  120.0  120.0  0.0
1     Fmcs_1.txt 1996-05-03  CTG0002.txt    5.0   632.0  132.0  132.0  4.0
2     Fmcs_1.txt 1996-05-03  CTG0003.txt  177.0   779.0  133.0  133.0  2.0
3     Fmcs_1.txt 1996-05-03  CTG0004.txt  411.0  1192.0  134.0  134.0  2.0
4     Fmcs_1.txt 1996-05-03  CTG0005.txt  533.0  1147.0  132.0  132.0  4.0
5     Fmcs_2.txt 1996-05-03  CTG0006.txt    0.0   953.0  134.0  134.0  1.0
6     Fmcs_2.txt 1996-05-03  CTG0007.txt  240.0   953.0  134.0  134.0  1.0
7     Hasc_1.txt 1995-02-22  CTG0008.txt   62.0   679.0  122.0  122.0  0.0
8     Hasc_1.txt 1995-02-22  CTG0009.txt  120.0   779.0  122.0  122.0  0.0
9     Hasc_1.txt 1995-02-22  CTG0010.txt  181.0  1192.0  122.0  122.0  0.0
10     Hasc3.txt 1995-02-22  CTG0011.txt    0.0  1199.0  151.0  151.0  0.0
11     Hasc3.txt 1995-02-22  CTG0012.txt   57.0  1074.0  150.0  150.0  0.0
12  Mcslrc_1.txt 1995-01-08  CTG0013.txt   52.0   840.0  131.0  131.0  4.0
13  Mcslrc_1.txt 1995-01-08  CTG0014.txt  531.0  1192.0  131.0  131.0  6.0
```

Question #2 Part 1

```python
selected_features = ['MSTV', 'Width', 'Mode', 'Variance']
X = data[selected_features]
y = data['NSP']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# scale the features
scaler = StandardScaler() # https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html
X_train_scaled = scaler.fit_transform(X_train) # use fit the scaler on the training data and then apply to test data
X_test_scaled = scaler.transform(X_test)

# training on Xtrain
nb_classifier = GaussianNB() # https://scikit-learn.org/dev/modules/generated/sklearn.naive_bayes.GaussianNB.html
nb_classifier.fit(X_train_scaled, y_train)

#predict class labels in Xtest
y_pred = nb_classifier.predict(X_test_scaled) # make predictions    y_pred    nb_classifier    X_test_scaled

accuracy = accuracy_score(y_test, y_pred)
```

Question #2 Part 2

Accuracy using Naive Bayesian NB classifier: 75.16%}

```python
y_pred = nb_classifier.predict(X_test_scaled) # make predictions

accuracy = accuracy_score(y_test, y_pred)
print(f"{accuracy*100:.2f}%")
```

✓ [24] 46ms

```
75.16%
```

{} Code   M↓ Markdown

Question #2 Part 3

{} Code   M↓ Markdown

```python
confusion_matrix = confusion_matrix(y_test, y_pred) # https://scikit-learn.org/dev/modules/generated/sklearn.metrics.confusion_mat
print(confusion_matrix)
```

✓ [48] < 10 ms

```
[[ 39 195]
 [ 69 760]]
```

Question #3 Part 1

```python
from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier(random_state=42) #https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeClassifier.html
dt_classifier.fit(X_train, y_train) # using same 50/50 split from earlier question

# make predictions based on Xtest
dt_pred = dt_classifier.predict(X_test)
```

Question #3 Part 2

```python

dt_accuracy = accuracy_score(y_test, dt_pred)
print(f"{dt_accuracy*100:.2f}%")
```

✓ [49] < 10 ms

```
87.49%
```

Question #3 Part 3

```python
dt_confusion_matrix = confusion_matrix(y_test, dt_pred)
print(dt_confusion_matrix)
```

✓ [54] < 10 ms

```
[[180  54]
 [ 79 750]]
```

Question #4 Part 1

```python
X = data[selected_features]
y = data['NSP']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# take N = 1, . . . , 10 and d = 1, 2, . . . , 5
NValues= range(1, 11)
dValues = range(1, 6)
error_rates = np.zeros((len(NValues), len(dValues))) # stores error rates for different combinations of N and d

'''
we need to test every possible combination of N and d (number of trees and max depth) params
for each "iteration":
    create a Random Forest model with the param
    train it
    make predictions on test data
    calculate the error rate
    store it in error_rates matrix for the combination
'''
for i, n_trees in enumerate(NValues):
    for j, max_depth in enumerate(dValues):
        rf = RandomForestClassifier( # https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html
            n_estimators=n_trees,
            max_depth=max_depth,
            criterion='entropy', # use "entropy" as splitting criteria
            random_state=42
        )
        rf.fit(X_train, y_train) # train rf on training data
        y_pred = rf.predict(X_test) # after training, model makes predictions on new data
        error_rate = 1 - accuracy_score(y_test, y_pred) # calculate the error rate
        error_rates[i, j] = error_rate # store it in error_rates matrix for the combination
```

Question #4 Part 2

The best combination of N and d were 3 and 5.

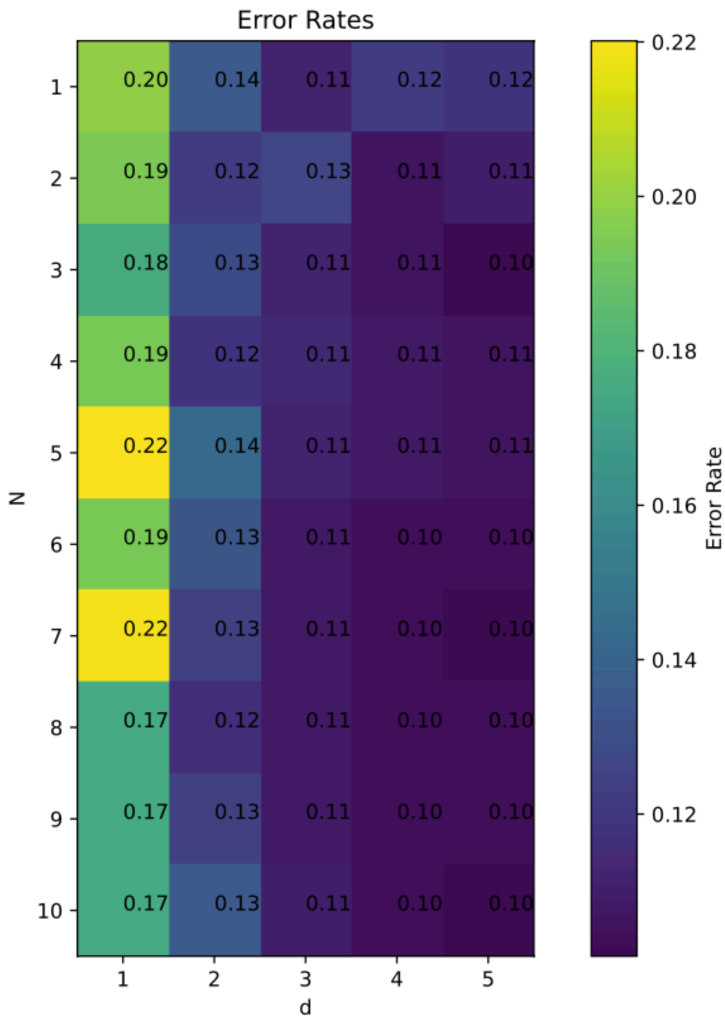
```python
            i_best_index = i
            j_best_index = j

best_N = NValues[i_best_index]
best_d = dValues[j_best_index]
print(f"N={best_N}, d={best_d}")
```

```
✓ [79] < 10 ms

N=3, d=5
```

Heatmap:



Question #4 Part 4

```
best_combo_confusion_matrix = confusion_matrix(y_tes
print(best_combo_confusion_matrix)
✓ [85] 28ms

[[168  66]
 [ 42 787]]
```

Question #5

```
results
```

✓ [99] 166ms

3 rows × 8 columns

	Model	TP	FP	TN	FN	accuracy	TPR	TNR
0	naive bayesian	760	195	39	69	75.16%	0.916767	0.166667
1	decision tree	750	54	180	79	87.49%	0.904704	0.769231
2	random forest	787	66	168	42	89.84%	0.949337	0.717949

In terms of accuracy, the random forest classifier performed best with accuracy of 89.84%. The reason it performed so well was because it combines various decision trees and also helps reduce overfitting. We can also see that Naive Bayesian had the lowest accuracy but is strong to identify normal cases. Decision tree was in the middle but was very close to random forest accuracy. It had a much better TNR compared to Naive Bayesian and seemed balanced.