Name: Shane Bowen
Id: R00149085
Class: SDH4-A

Task 1

```python
def splitting_and_counting(movie_reviews):
    print("\n***************Task 1********************")

    data = movie_reviews[["Review", "Sentiment", "Split"]]

    print("\n----------Train Data---------")
    train_data = data[["Review", "Sentiment"]][data.Split=="train"]
    print(train_data)

    print("\n----------Train Labels---------")
    train_labels = data[["Sentiment"]][data.Split=="train"]
    print(train_labels)

    print("\n----------Test Data---------")
    test_data = data[["Review", "Sentiment"]][data.Split=="test"]
    print(test_data)

    print("\n----------Test Labels---------")
    test_labels = data[["Sentiment"]][data.Split=="test"]
    print(test_labels)

    positive_reviews_train = train_labels[train_labels.Sentiment=="positive"]
    print("Num Positive Reviews Train:", len(positive_reviews_train))

    negative_reviews_train = train_labels[train_labels.Sentiment=="negative"]
    print("Num Negative Reviews Train:", len(negative_reviews_train))

    positive_reviews_test = test_labels[test_labels.Sentiment=="positive"]
    print("Num Positive Reviews Test:", len(positive_reviews_test))

    negative_reviews_test = test_labels[test_labels.Sentiment=="negative"]
    print("Num Negative Reviews Test:", len(negative_reviews_test))

    return train_data, train_labels, test_data, test_labels, positive_reviews_train, negative_reviews_train, positive_reviews_test, negative_reviews_test
```

Read the data from the excel file, split the data into train and test, split the data into reviews and sentiment, print the amount of values in the 4 dataset to the console, return all the datasets

Task 2

```
#task_2
def extract_relevant_features(train_data):
    print("\n***************Task 2******************")

    train_data["Review"] = train_data["Review"].str.replace('([^\s\w]|_)+', '')
    train_data["Review"] = train_data["Review"].str.lower()

    minWordLength = int(input("Enter the min word length: "))
    minWordOccurrence = int(input("Enter the min word occurrence: "))

    allWords = train_data["Review"].str.split()
    wordOccurrences = {}
    for word_arr in allWords:
        for word in word_arr:
            if (len(word)>=minWordLength):
                if (word in wordOccurrences):
                    wordOccurrences[word] = wordOccurrences[word] + 1
                else:
                    wordOccurrences[word]=1
    words = {}
    print("\n----------Word Occurrences----------")
    for word in wordOccurrences:
        if wordOccurrences[word]>=minWordOccurrence:
            print(word + ":" + str(wordOccurrences[word]))
            words[word] = wordOccurrences[word]

    return words
```

Replace all non-alphanumeric character, put to lowercase, ask user for min word length and min word occurrence, create a list for all words in reviews, create empty dictionary for word occurrences, loop through each word in a review, if word already in list increment it by 1, else add to dictionary and set value equals 1, print the word occurrences if that word appears more that what the user inputted, return this list

Task 3

```
#task3
def count_feature_frequencies(my_list, word_occurrences):
    print("\n***************Task 3*******************")

    positive_feature_frequency = {}
    negative_feature_frequency = {}

    positive_data = my_list[["Review"]][my_list.Sentiment=="positive"]
    negative_data = my_list[["Review"]][my_list.Sentiment=="negative"]

    for word in word_occurrences:
        for pos_data in positive_data["Review"].str.split():
            if (word in pos_data):
                if(word in positive_feature_frequency):
                    positive_feature_frequency[word] = positive_feature_frequency[word] + 1
                else:
                    positive_feature_frequency[word]=1

        for neg_data in negative_data["Review"].str.split():
            if (word in neg_data):
                if(word in negative_feature_frequency):
                    negative_feature_frequency[word] = negative_feature_frequency[word] + 1
                else:
                    negative_feature_frequency[word]=1

    print("\n----------Postive Frequency----------")
    for word in positive_feature_frequency:
        print(word + ":" + str(positive_feature_frequency[word]))

    print("\n----------Negative Frequency----------")
    for word in negative_feature_frequency:
        print(word + ":" + str(negative_feature_frequency[word]))

    return positive_feature_frequency, negative_feature_frequency
```

Create empty dictionary for positive and negative feature frequency, split the data into positive and negative, loop the words returned in task 2 , if word appears in negative or positive review increment it to count like before, print the positive and negative frequency, return the positive and negative frequency

Task 4

```
#task4
def feature_likelihood_and_priors(my_list, feature_frequency):
    print("\n***************Task 4*******************")

    positive_feature_likelihood = {}
    negative_feature_likelihood = {}
    alpha = 1

    #feature_frequency[0] = positive
    #feature_frequency[1] = negative
    #Multiply denominator alpha by 2 for num of sentiments. (pos and neg)
    for word in feature_frequency[0]:
        positive_feature_likelihood[word] = (feature_frequency[0][word] + alpha) / (len(my_list[4]) + 2*alpha)

    for word in feature_frequency[1]:
        negative_feature_likelihood[word] = (feature_frequency[1][word] + alpha) / (len(my_list[5]) + 2*alpha)

    print("\n----------Postive Likelihood----------")
    for word in positive_feature_likelihood:
        print(word + ":" + str(positive_feature_likelihood[word]))

    print("\n----------Negative Likelihood----------")
    for word in negative_feature_likelihood:
        print(word + ":" + str(negative_feature_likelihood[word]))

    prior_positive_review = len(my_list[4]) / len(my_list[0])
    prior_negative_review = len(my_list[5]) / len(my_list[0])
    print("\nPrior Postive Review:", prior_positive_review)
    print("Prior Negative Review:", prior_negative_review)

    return positive_feature_likelihood, negative_feature_likelihood, prior_positive_review, prior_negative_review
```

Create empty positive and negative feature likelihood dictionary, set alpha = 1, loop through each word in feature frequency, calculate the likelihood for that word using formula, print all positive and negative likelihood, calculate the positive and negative priors, print the priors, return the likelihoods and priors

Task 5

```
#task5
def maximum_likelihood_classification(my_list, positive_feature_likelihood, negative_feature_likelihood, prior_positive_review, prior_negative_review):
    print("\n***************Task 5*******************")

    logLikelihood_positive = 0
    logLikelihood_negative = 0

    all_words = my_list["Review"].str.split()
    for word in all_words[1]:
        for feature in positive_feature_likelihood:
            if word == feature:
                logLikelihood_positive = logLikelihood_positive + math.log(positive_feature_likelihood[feature])

        for feature in negative_feature_likelihood:
            if word == feature:
                logLikelihood_negative = logLikelihood_negative + math.log(negative_feature_likelihood[feature])

    print("Likelihood Posititve:", logLikelihood_positive)
    print("Likelihood Negative:", logLikelihood_negative)
    print("\n--------Prediction--------")
    if logLikelihood_positive - logLikelihood_negative > math.log(prior_negative_review) - math.log(prior_positive_review):
        print("Positive")
    else:
        print("Negative")
```

Initialize log likelihood positive and negative, split the words in a single reviews into a list, loop through each word in list, loop through positive and negative likelihood, if word equals to feature then add that likelihood calculated in task 4 to the total log likelihood, print totals for the 2 log likelihoods, make prediction of review either being positive or negative from the formula.

Task 6

```
#task6
def evalutaion_of_results(data):
    print("\n***************Task 6*******************")

    data["Sentiment"] = data["Sentiment"].map({"positive":1,"negative":0})
    target = data["Sentiment"]

    kf = model_selection.StratifiedKFold(n_splits=10, shuffle=True)

    ROC_X = []
    ROC_Y = []

    for k in range(1,10):
        true_negative = []
        true_positive= []
        false_negative = []
        false_positive= []
        for train_index, test_index in kf.split(data, target):
            clf = neighbors.KNeighborsClassifier(n_neighbors=k)
            clf.fit(data.iloc[train_index], target[train_index])
            predicted_labels = clf.predict(data.iloc[test_index])
            print(metrics.accuracy_score(target[test_index], predicted_labels))

            C = metrics.confusion_matrix(target[test_index], predicted_labels)

            true_negative.append(C[0,0])
            true_positive.append(C[1,1])
            false_negative.append(C[1,0])
            false_positive.append(C[0,1])

            print("k =",k)
            print("True negative:", np.sum(true_negative))
            print("True positive:", np.sum(true_positive))
            print("False negative:", np.sum(false_negative))
            print("False positive:", np.sum(false_positive))
            print()

            ROC_X.append(np.sum(false_positive))
            ROC_Y.append(np.sum(true_positive))

            print(ROC_X)
            print(ROC_Y)

            plt.close('all')
            plt.figure()
            plt.scatter(ROC_X,ROC_Y)
            plt.axis([0,np.max(ROC_X),0,np.max(ROC_Y)])
```

Although this part was not fully completed this was my attempt, map the values of positive and negative to 1 and 0, set the amount of k-fold splits to 10, create empty arrays for receiver operating characteristics, subset the training data into training and test, for each loop in k-fold train the data and calculate the predicted_labels, create confusion matrix and append value to matrix.