CSC2062 AIMA – Assignment 2

[ZE HUEI LIM]

[40443486]

[1/5/2025]

Introduction

In this assignment, I will use the feature I developed in Assignment 1 to solve classification problems using machine learning. Specifically, I will fit and evaluate classifiers for my image data, in order to see whether the models can predict the class labels for unseen image. This assignment is to be completed individually.

This assignment must be completed in Python. I may not use Excel for any calculations in the assignment, or for figures. Convenient and commonly used machine learning packages are available for Python. When I use a procedure that has an element of randomness (e.g. creating cross-validation folds) please use the seed value 42 or any values that is suitable (your code should give the same results each time it runs).

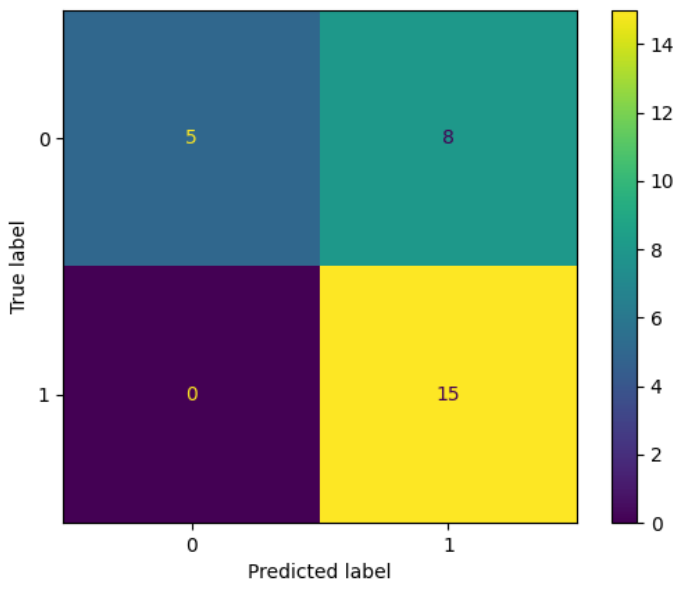
I should briefly explain and interpret all results.

Section 1

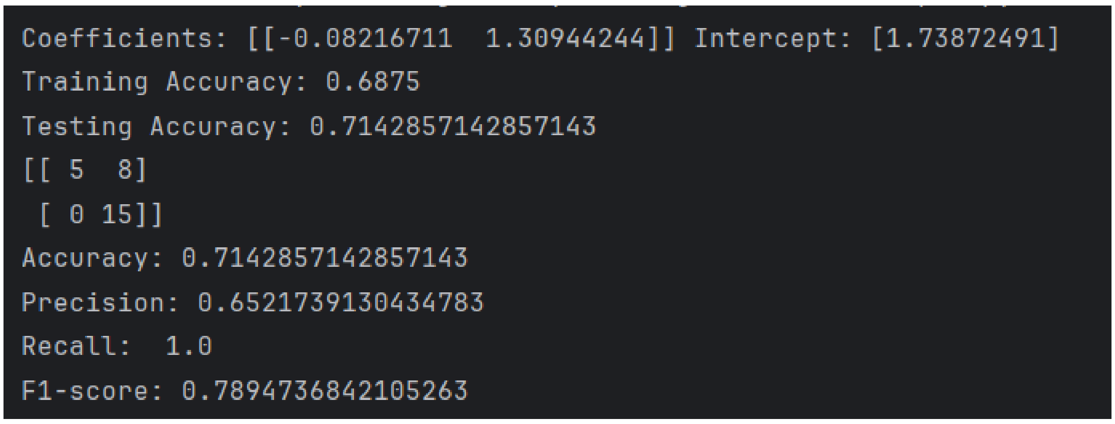
Section 1.1

For section 1.1, I have implemented library from SKlearn (metrics and train_test_split). Importing metrics allows for the confusion matrix, train_test_split allows me to split my test size into 20% which is 0.2 and random state = 42, accuracy score. Not only that, using metrics library allows me to calculate and show my confusion matrix in a graph, calculate all Accuracy, Precision, Recall and F1-score.

- Confusion Matrix : [5] [8]
  [0] [15]



- Accuracy : 0.7143
- True Positive Rate : 15
- False Positive Rate : 8
- Precision: 0.6522
- Recall : 1.0
- F1-score: 0.7895



```
Coefficients: [[-0.08216711  1.30944244]] Intercept: [1.73872491]
Training Accuracy: 0.6875
Testing Accuracy: 0.7142857142857143
[[ 5  8]
 [ 0 15]]
Accuracy: 0.7142857142857143
Precision: 0.6521739130434783
Recall:  1.0
F1-score: 0.7894736842105263
```

Section 1.2

Like Section 1.1 but includes 5-fold cross validation.

- Cross-validated accuracy: Fold accuracy: 0.7857

  Fold accuracy: 0.6071

  Fold accuracy: 0.7143

  Fold accuracy: 0.5714

  Fold accuracy: 0.6429

  Average cross-validated accuracy: 0.6643

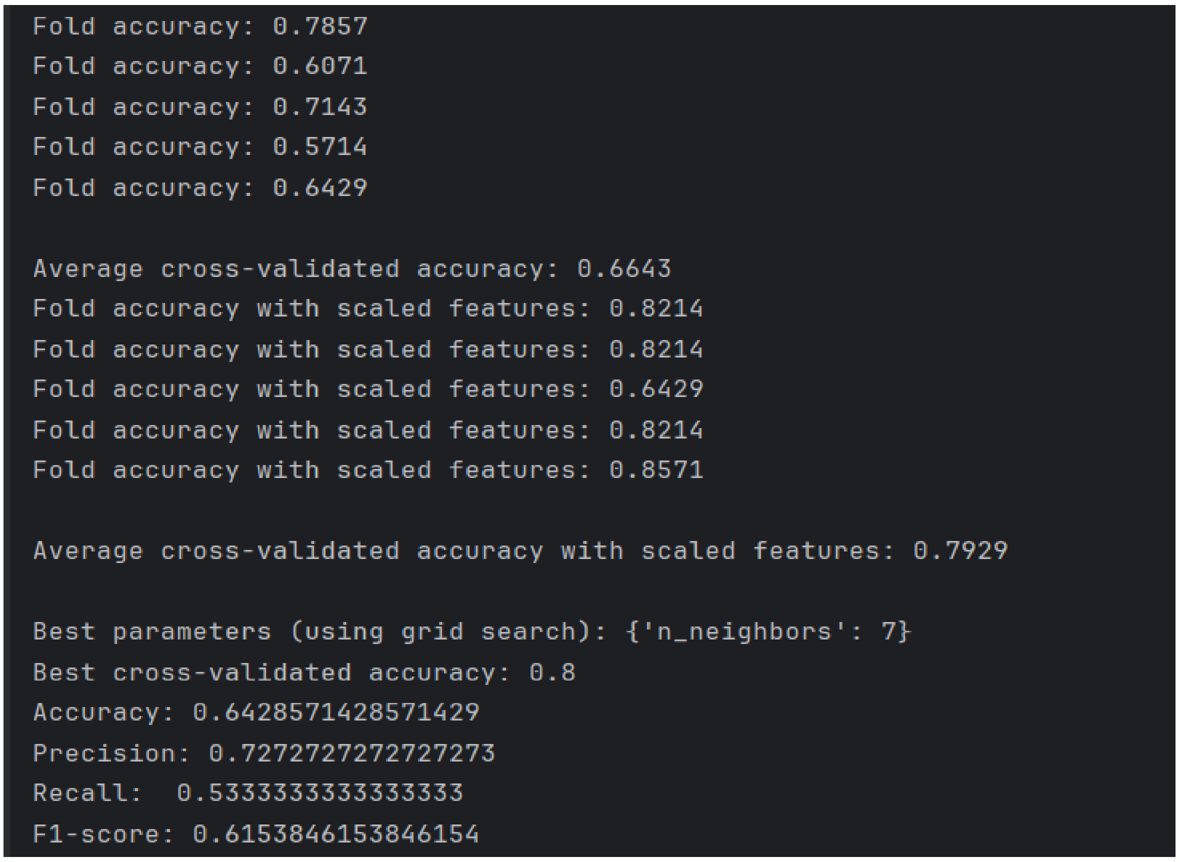
  Fold accuracy with scaled features: 0.8214

  Fold accuracy with scaled features: 0.8214

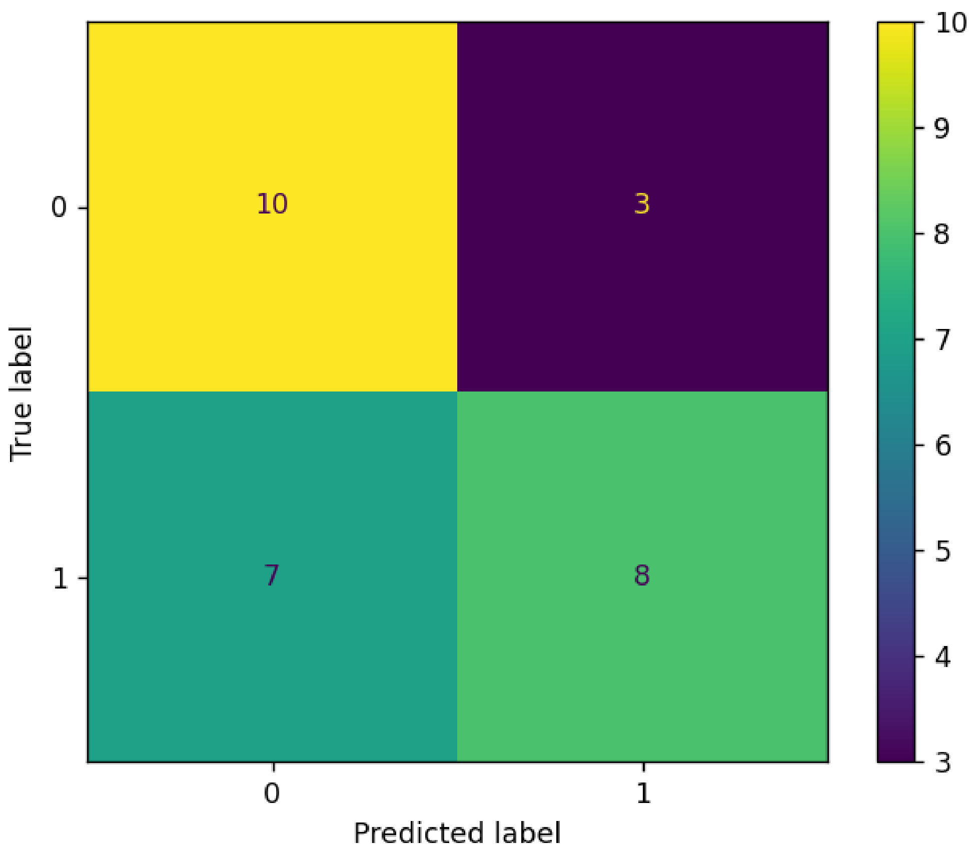
  Fold accuracy with scaled features: 0.6429

  Fold accuracy with scaled features: 0.8214

  Fold accuracy with scaled features: 0.8571
- True Positive Rate: 8
- False Positive Rate: 3
- Precision: 0.7273
- Recall: 0.5333
- F1-score: 0.6154



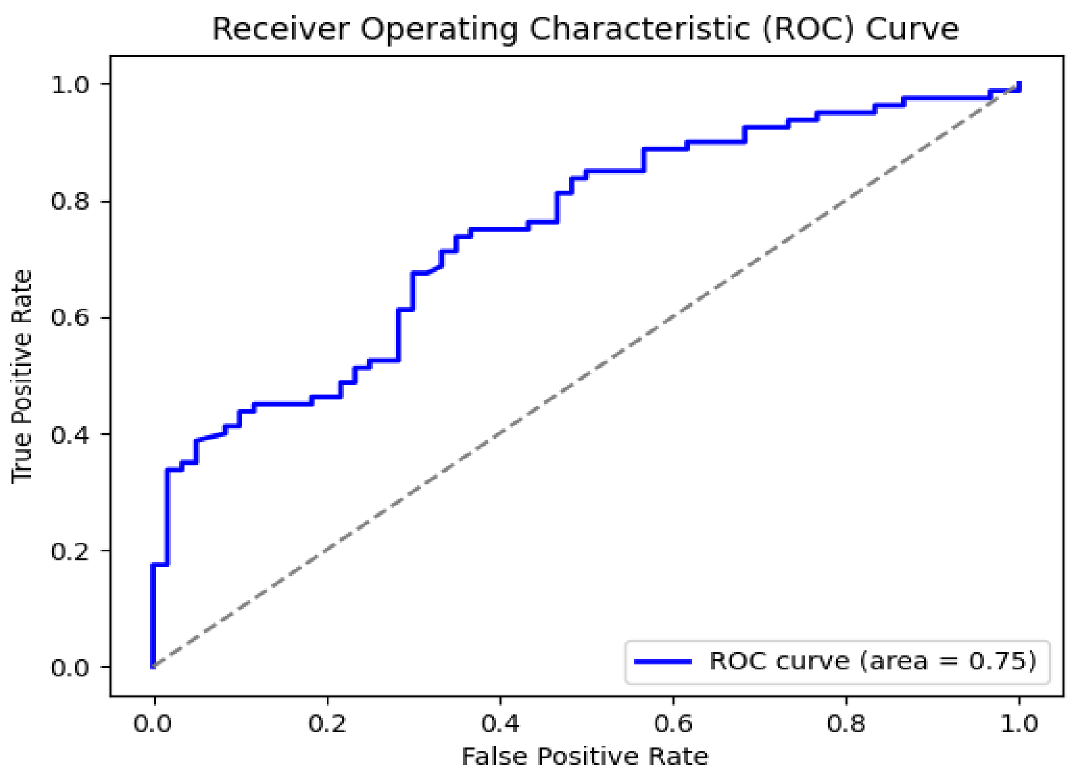
```
Fold accuracy: 0.7857
Fold accuracy: 0.6071
Fold accuracy: 0.7143
Fold accuracy: 0.5714
Fold accuracy: 0.6429

Average cross-validated accuracy: 0.6643
Fold accuracy with scaled features: 0.8214
Fold accuracy with scaled features: 0.8214
Fold accuracy with scaled features: 0.6429
Fold accuracy with scaled features: 0.8214
Fold accuracy with scaled features: 0.8571

Average cross-validated accuracy with scaled features: 0.7929

Best parameters (using grid search): {'n_neighbors': 7}
Best cross-validated accuracy: 0.8
Accuracy: 0.6428571428571429
Precision: 0.7272727272727273
Recall:  0.5333333333333333
F1-score: 0.6153846153846154
```



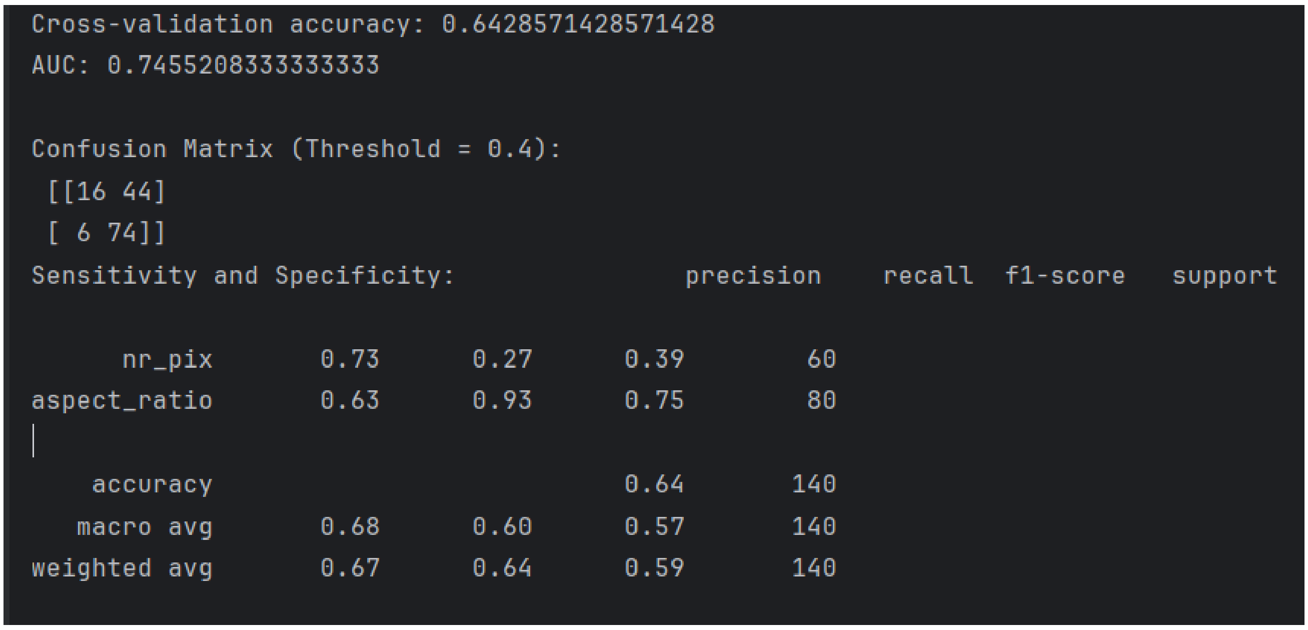
Confusion Matrix for Section 1.2.

Section 1.3

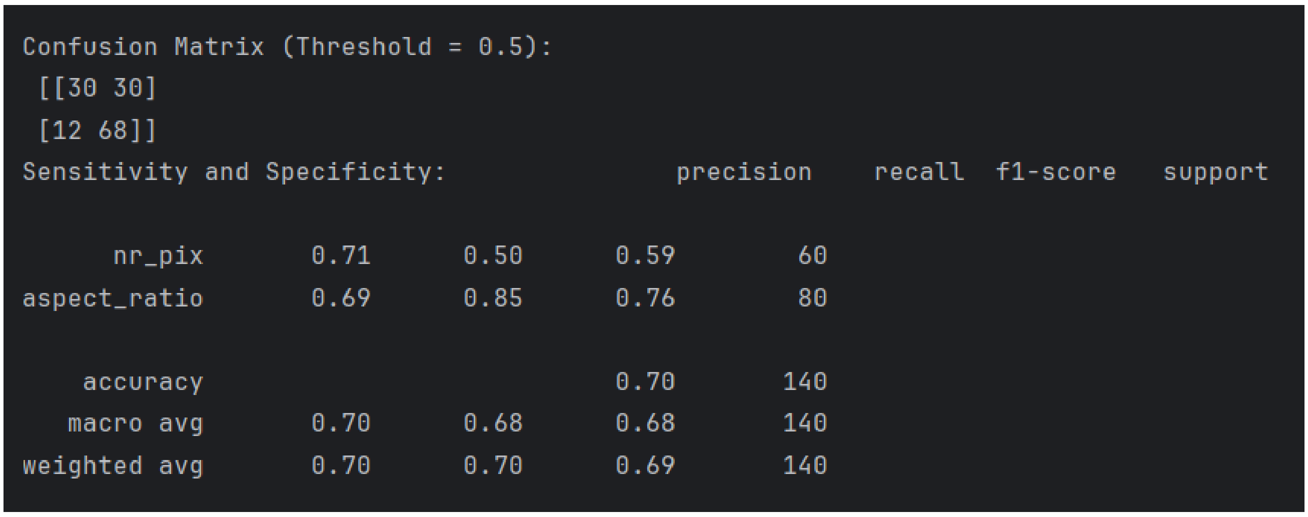
ROC Curve with 5-fold cross validation.



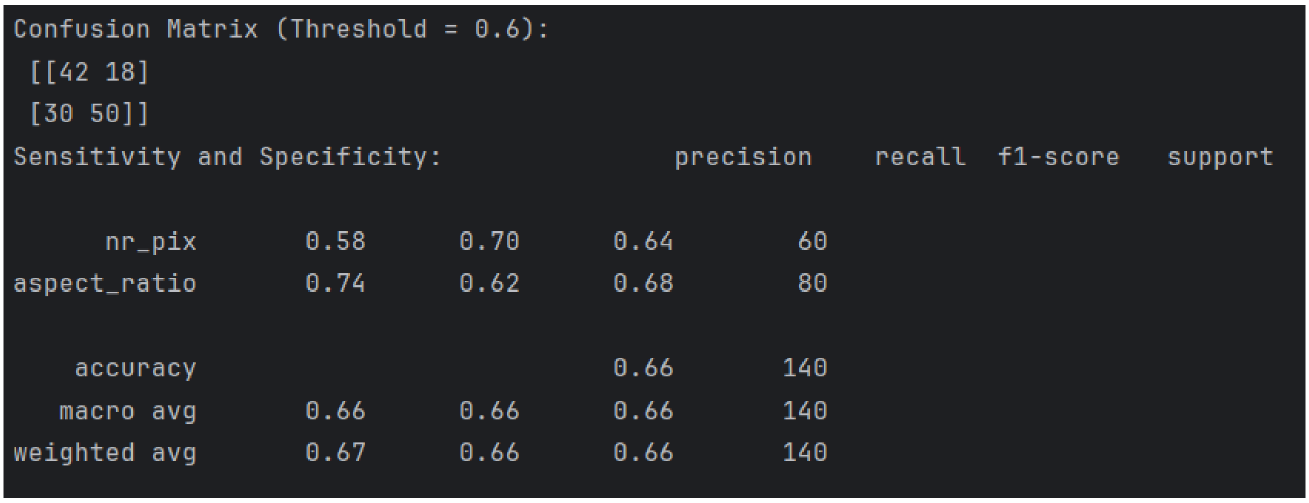
True Positive Rate which is Sensitivity/Recall and False Positive Rate which is Specificity which are shown in the diagram. It is shown that the AUC (Area Under the Curve) values at 0.7455 hence indicating a strong classifier between the two classes. The curve also bends towards the top-left but not too much which also indicates it has a moderate sensitivity and moderate specificity with a cross-validation accuracy of 0.6429.



```
Cross-validation accuracy: 0.6428571428571428
AUC: 0.7455208333333333

Confusion Matrix (Threshold = 0.4):
 [[16 44]
 [ 6 74]]
Sensitivity and Specificity:               precision    recall  f1-score   support

      nr_pix       0.73      0.27      0.39        60
aspect_ratio       0.63      0.93      0.75        80
|
    accuracy                           0.64       140
   macro avg       0.68      0.60      0.57       140
weighted avg       0.67      0.64      0.59       140
```



```
Confusion Matrix (Threshold = 0.5):
 [[30 30]
 [12 68]]
Sensitivity and Specificity:               precision    recall  f1-score   support

      nr_pix       0.71      0.50      0.59        60
aspect_ratio       0.69      0.85      0.76        80

    accuracy                           0.70       140
   macro avg       0.70      0.68      0.68       140
weighted avg       0.70      0.70      0.69       140
```



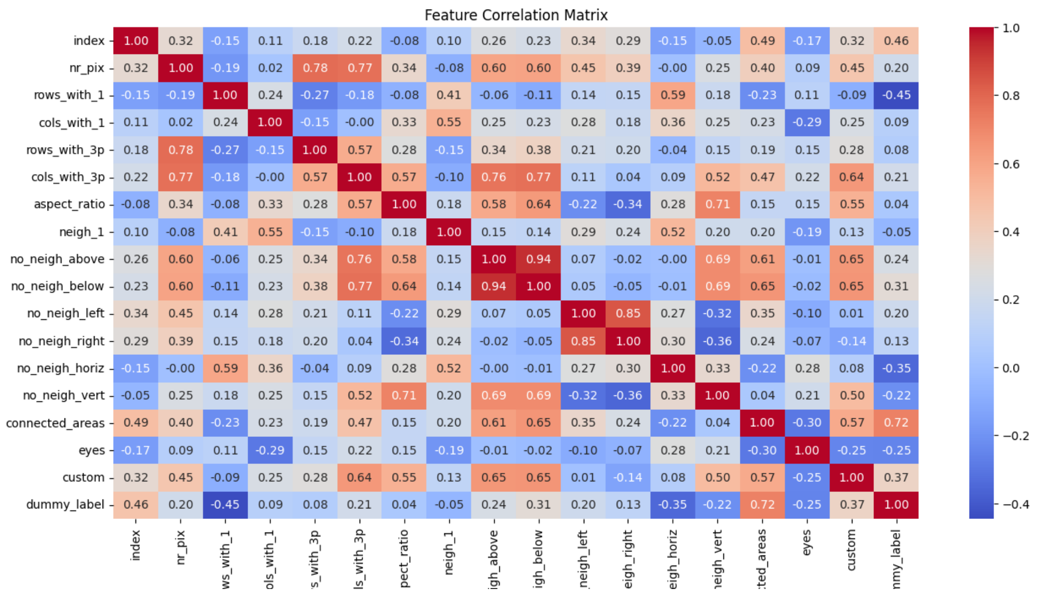
```
Confusion Matrix (Threshold = 0.6):
 [[42 18]
 [30 50]]
Sensitivity and Specificity:               precision    recall  f1-score   support

      nr_pix       0.58      0.70      0.64        60
aspect_ratio       0.74      0.62      0.68        80

    accuracy                           0.66       140
   macro avg       0.66      0.66      0.66       140
weighted avg       0.67      0.66      0.66       140
```

Values for False Positive Rate at 0.4, 0.5 and 0.6.

Section 2

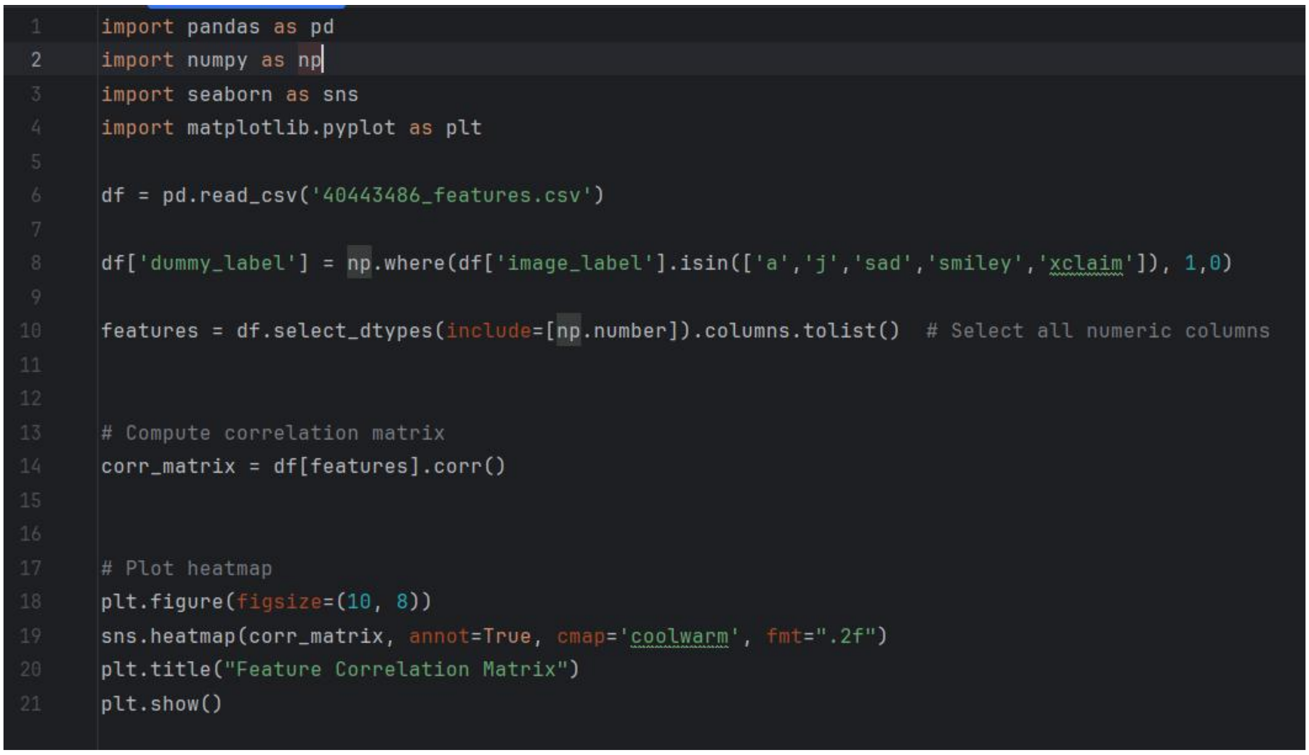
Section 2.1

Before performing Section 2, I have chosen 4 feature which are connected_areas, rows_with_1, no_neight_horiz and custom feature. They were chosen as they are the top 4 with the highest correlation with our dummy label of 'a' 'j' 'sad' 'smiley' and 'xclaim'.



	index	nr_pix	ws_with_1	ols_with_1	s_with_3p	ls_with_3p	pect_ratio	neigh_1	igh_above	igh_below	neigh_left	eigh_right	eigh_horiz	eigh_vert	ted_areas	eyes	custom	mmy_label
index	1.00	0.32	-0.15	0.11	0.18	0.22	-0.08	0.10	0.26	0.23	0.34	0.29	-0.15	-0.05	0.49	-0.17	0.32	0.46
nr_pix	0.32	1.00	-0.19	0.02	0.78	0.77	0.34	-0.08	0.60	0.60	0.45	0.39	-0.00	0.25	0.40	0.09	0.45	0.20
rows_with_1	-0.15	-0.19	1.00	0.24	-0.27	-0.18	-0.08	0.41	-0.06	-0.11	0.14	0.15	0.59	0.18	-0.23	0.11	-0.09	-0.45
cols_with_1	0.11	0.02	0.24	1.00	-0.15	-0.00	0.33	0.55	0.25	0.23	0.28	0.18	0.36	0.25	0.23	-0.29	0.25	0.09
rows_with_3p	0.18	0.78	-0.27	-0.15	1.00	0.57	0.28	-0.15	0.34	0.38	0.21	0.20	-0.04	0.15	0.19	0.15	0.28	0.08
cols_with_3p	0.22	0.77	-0.18	-0.00	0.57	1.00	0.57	-0.10	0.76	0.77	0.11	0.04	0.09	0.52	0.47	0.22	0.64	0.21
aspect_ratio	-0.08	0.34	-0.08	0.33	0.28	0.57	1.00	0.18	0.58	0.64	-0.22	-0.34	0.28	0.71	0.15	0.15	0.55	0.04
neigh_1	0.10	-0.08	0.41	0.55	-0.15	-0.10	0.18	1.00	0.15	0.14	0.29	0.24	0.52	0.20	0.20	-0.19	0.13	-0.05
no_neigh_above	0.26	0.60	-0.06	0.25	0.34	0.76	0.58	0.15	1.00	0.94	0.07	-0.02	-0.00	0.69	0.61	-0.01	0.65	0.24
no_neigh_below	0.23	0.60	-0.11	0.23	0.38	0.77	0.64	0.14	0.94	1.00	0.05	-0.05	-0.01	0.69	0.65	-0.02	0.65	0.31
no_neigh_left	0.34	0.45	0.14	0.28	0.21	0.11	-0.22	0.29	0.07	0.05	1.00	0.85	0.27	-0.32	0.35	-0.10	0.01	0.20
no_neigh_right	0.29	0.39	0.15	0.18	0.20	0.04	-0.34	0.24	-0.02	-0.05	0.85	1.00	0.30	-0.36	0.24	-0.07	-0.14	0.13
no_neigh_horiz	-0.15	-0.00	0.59	0.36	-0.04	0.09	0.28	0.52	-0.00	-0.01	0.27	0.30	1.00	0.33	-0.22	0.28	0.08	-0.35
no_neigh_vert	-0.05	0.25	0.18	0.25	0.15	0.52	0.71	0.20	0.69	0.69	-0.32	-0.36	0.33	1.00	0.04	0.21	0.50	-0.22
connected_areas	0.49	0.40	-0.23	0.23	0.19	0.47	0.15	0.20	0.61	0.65	0.35	0.24	-0.22	0.04	1.00	-0.30	0.57	0.72
eyes	-0.17	0.09	0.11	-0.29	0.15	0.22	0.15	-0.19	-0.01	-0.02	-0.10	-0.07	0.28	0.21	-0.30	1.00	-0.25	-0.25
custom	0.32	0.45	-0.09	0.25	0.28	0.64	0.55	0.13	0.65	0.65	0.01	-0.14	0.08	0.50	0.57	-0.25	1.00	0.37
dummy_label	0.46	0.20	-0.45	0.09	0.08	0.21	0.04	-0.05	0.24	0.31	0.20	0.13	-0.35	-0.22	0.72	-0.25	0.37	1.00

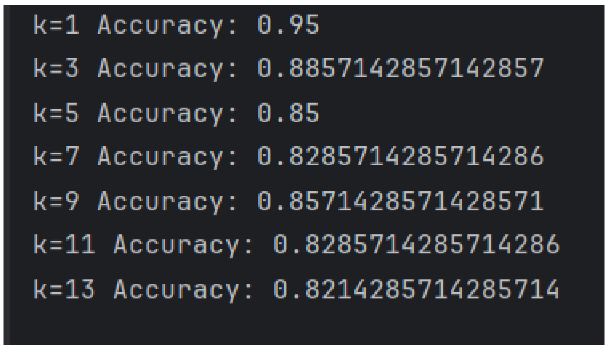
The code:



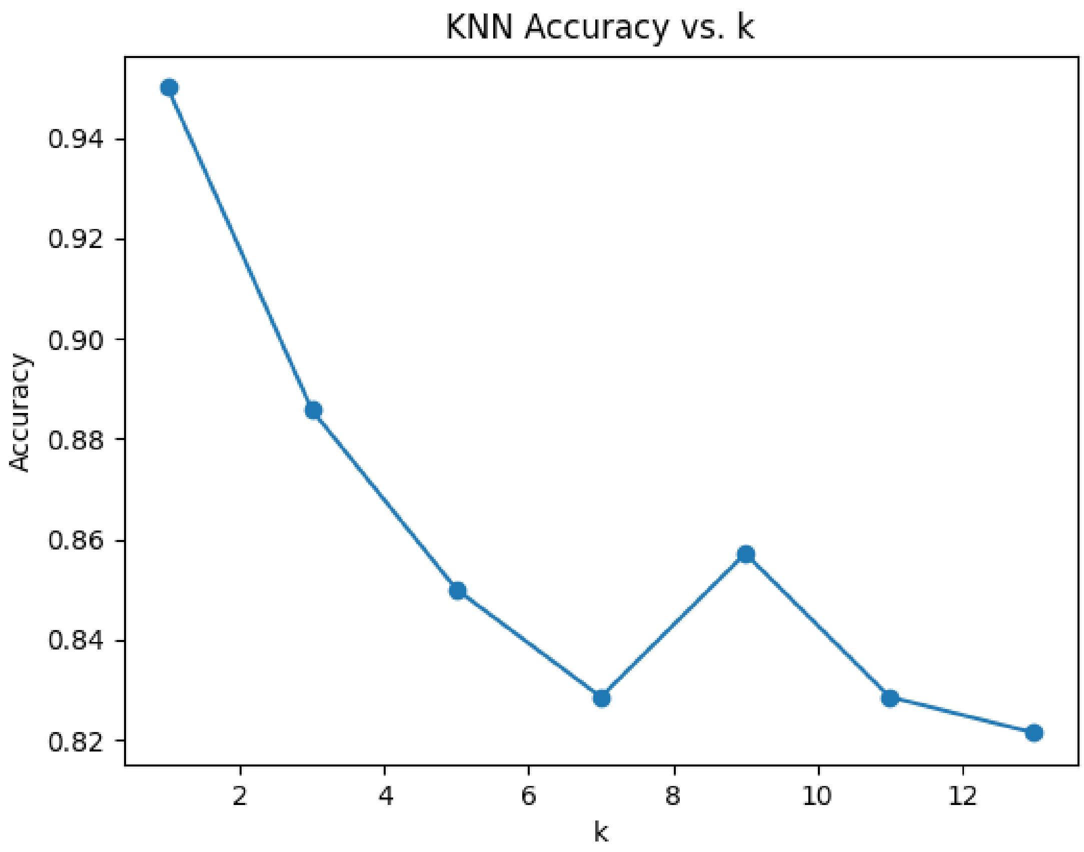
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('40443486_features.csv')

df['dummy_label'] = np.where(df['image_label'].isin(['a','j','sad','smiley','xclaim']), 1,0)

features = df.select_dtypes(include=[np.number]).columns.tolist()  # Select all numeric columns


# Compute correlation matrix
corr_matrix = df[features].corr()


# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Matrix")
plt.show()
```

Not only that we must only use all odd number from 1 to 13: 1,3,5,7,9,11,13.

Values for accuracy with each odd numbers:



```
k=1 Accuracy: 0.95
k=3 Accuracy: 0.8857142857142857
k=5 Accuracy: 0.85
k=7 Accuracy: 0.8285714285714286
k=9 Accuracy: 0.8571428571428571
k=11 Accuracy: 0.8285714285714286
k=13 Accuracy: 0.8214285714285714
```

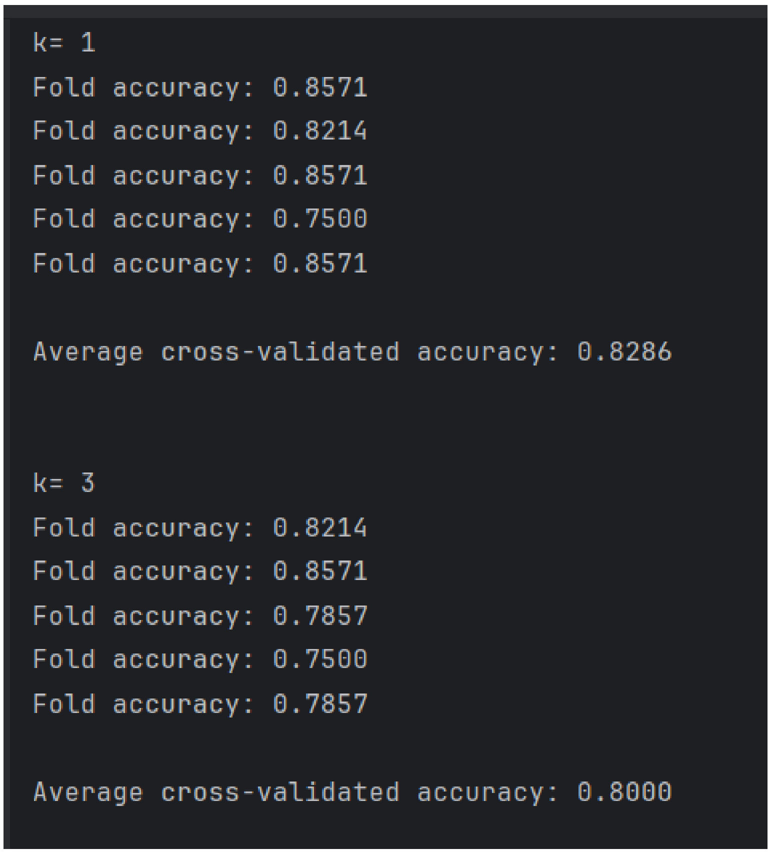


KNN Accuracy against k values for Section 2.1.

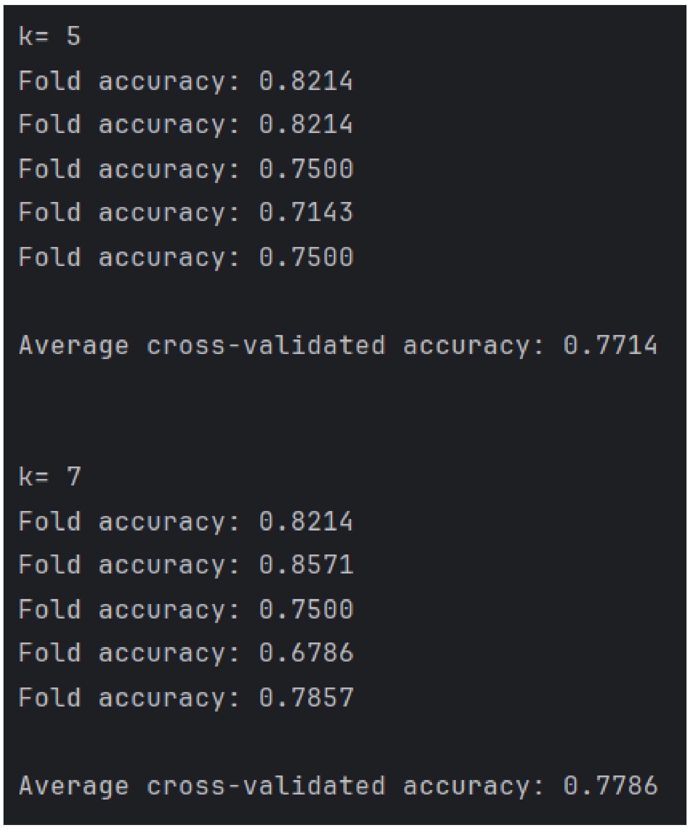
Section 2.2

Same as Section 2.1 but with 5-fold-cross-validation with same 4 feature.

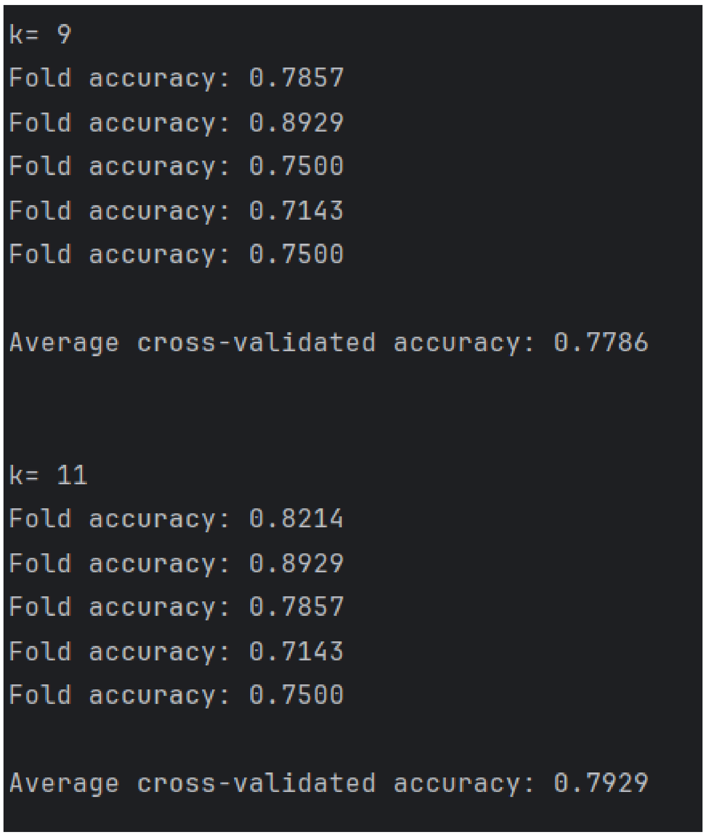
Values of fold accuracy and Average cross-validated accuracy for all odd numbers of k:



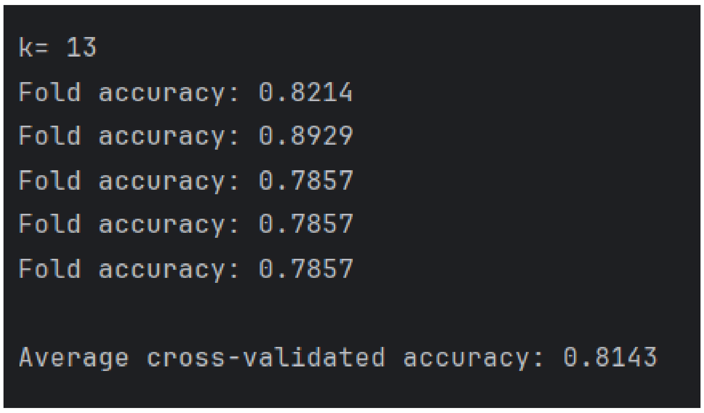
```
k= 1
Fold accuracy: 0.8571
Fold accuracy: 0.8214
Fold accuracy: 0.8571
Fold accuracy: 0.7500
Fold accuracy: 0.8571

Average cross-validated accuracy: 0.8286


k= 3
Fold accuracy: 0.8214
Fold accuracy: 0.8571
Fold accuracy: 0.7857
Fold accuracy: 0.7500
Fold accuracy: 0.7857

Average cross-validated accuracy: 0.8000
```



```
k= 5
Fold accuracy: 0.8214
Fold accuracy: 0.8214
Fold accuracy: 0.7500
Fold accuracy: 0.7143
Fold accuracy: 0.7500

Average cross-validated accuracy: 0.7714


k= 7
Fold accuracy: 0.8214
Fold accuracy: 0.8571
Fold accuracy: 0.7500
Fold accuracy: 0.6786
Fold accuracy: 0.7857

Average cross-validated accuracy: 0.7786
```



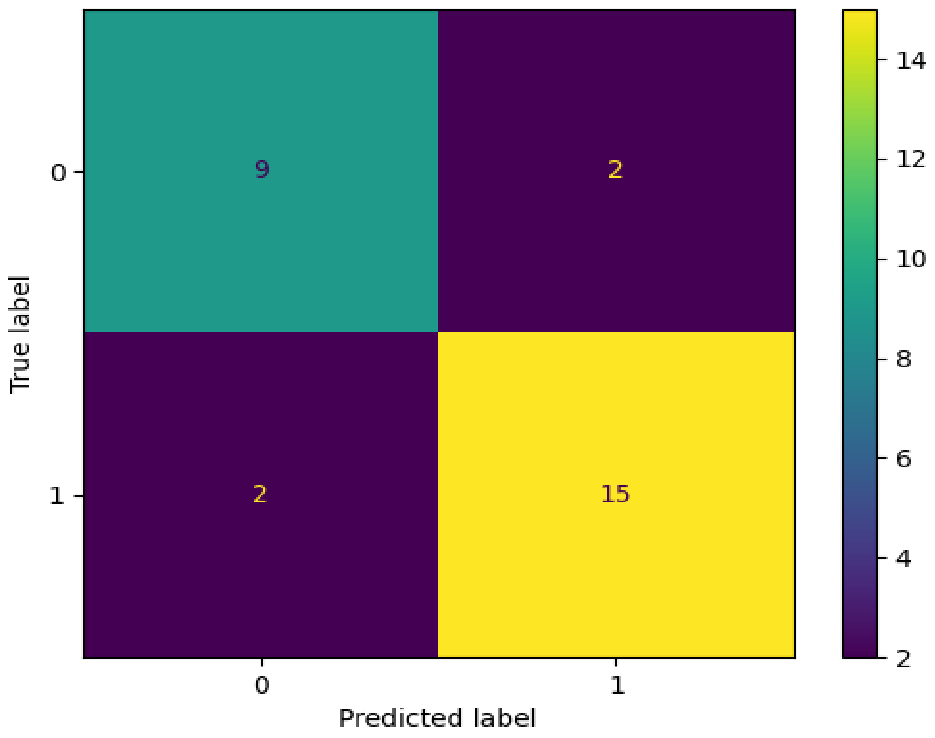
```
k= 9
Fold accuracy: 0.7857
Fold accuracy: 0.8929
Fold accuracy: 0.7500
Fold accuracy: 0.7143
Fold accuracy: 0.7500

Average cross-validated accuracy: 0.7786


k= 11
Fold accuracy: 0.8214
Fold accuracy: 0.8929
Fold accuracy: 0.7857
Fold accuracy: 0.7143
Fold accuracy: 0.7500

Average cross-validated accuracy: 0.7929
```



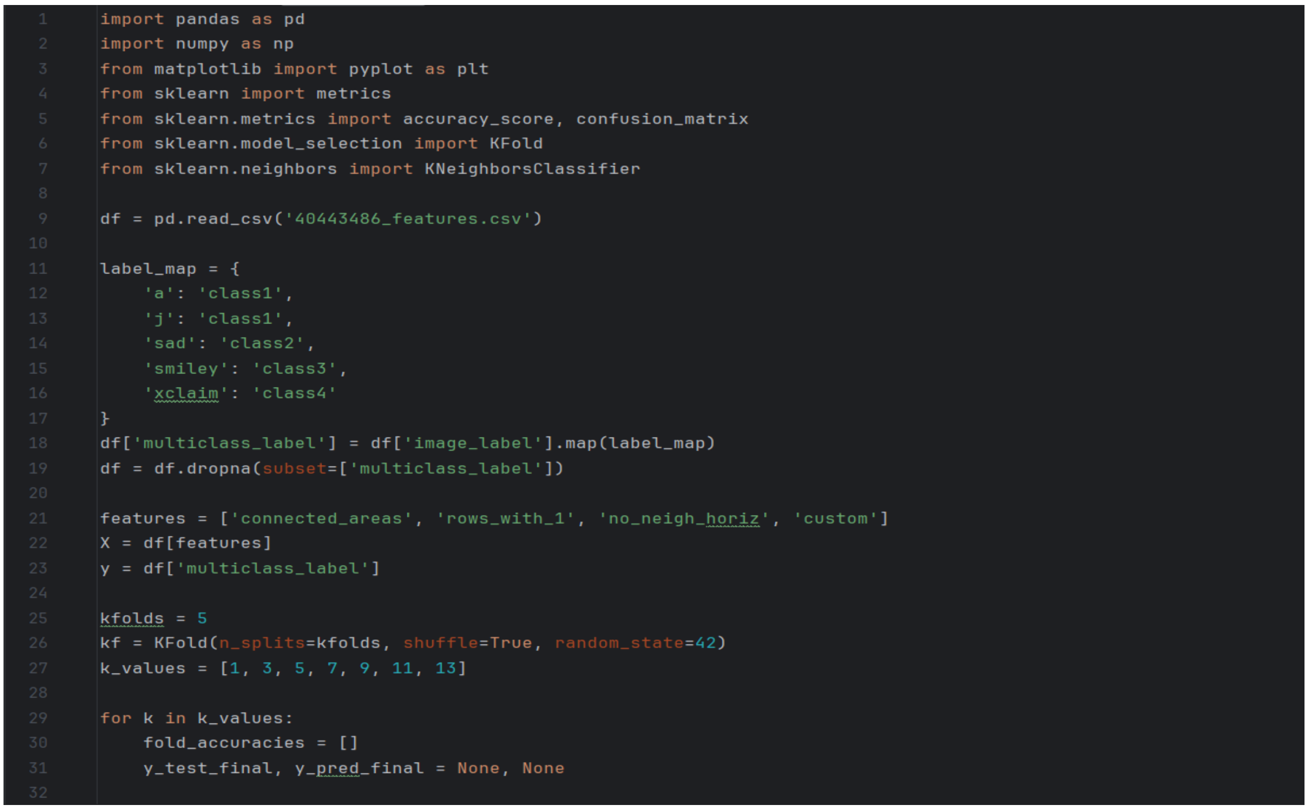
```
k= 13
Fold accuracy: 0.8214
Fold accuracy: 0.8929
Fold accuracy: 0.7857
Fold accuracy: 0.7857
Fold accuracy: 0.7857

Average cross-validated accuracy: 0.8143
```

Section 2.3

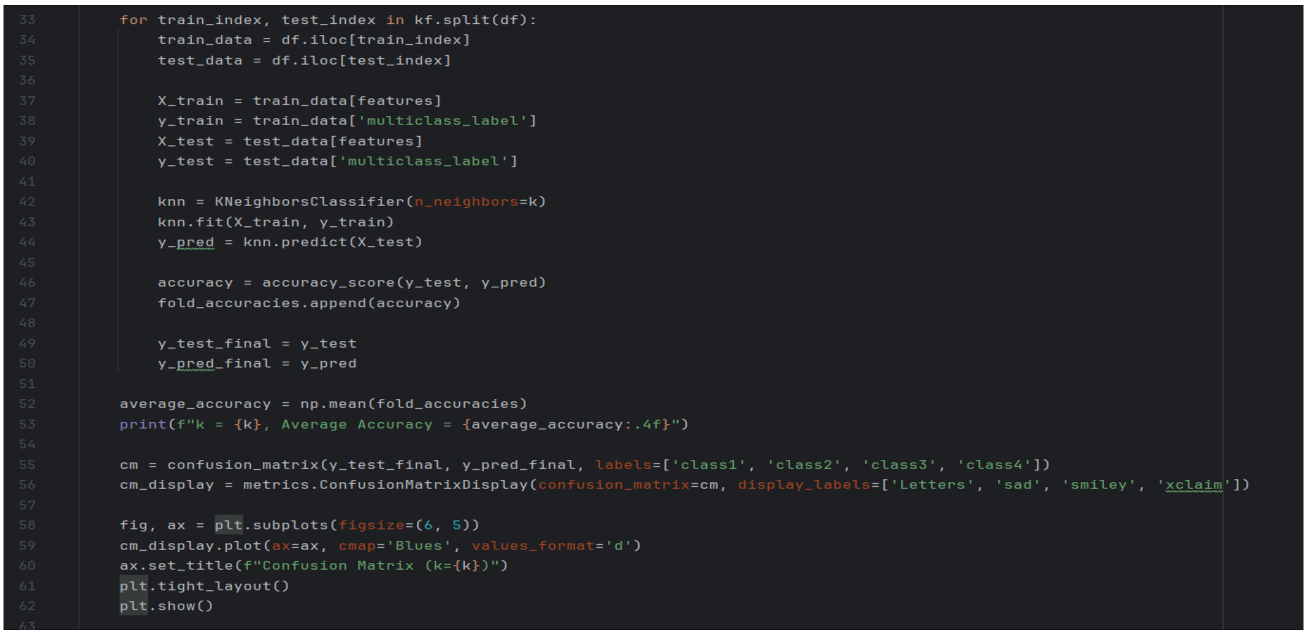
With Section 2.2 values for each k-value accuracy, it is shown the best k value is 1 with the highest average cross-validated accuracy of 0.8286. Below will be the confusion
matrix:



In order to compare which pair of classes are the most difficult to discriminate, we have to split each class as their own and do a multiclass confusion matrix on another python page with maps.

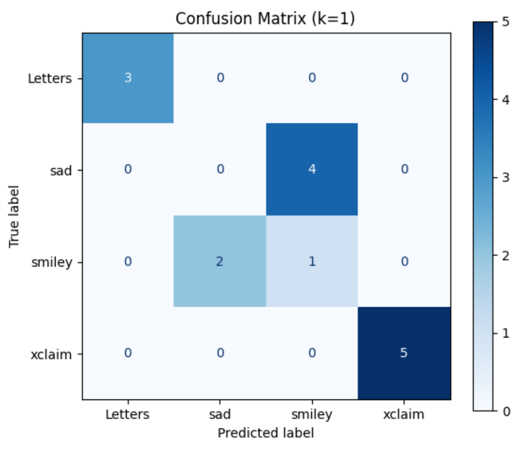


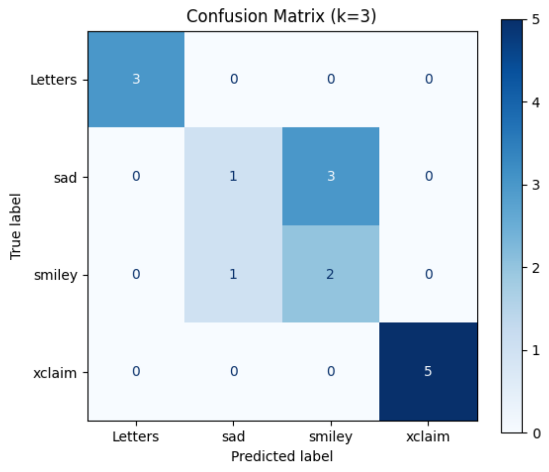
```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier

df = pd.read_csv('40443486_features.csv')

label_map = {
    'a': 'class1',
    'j': 'class1',
    'sad': 'class2',
    'smiley': 'class3',
    'xclaim': 'class4'
}
df['multiclass_label'] = df['image_label'].map(label_map)
df = df.dropna(subset=['multiclass_label'])

features = ['connected_areas', 'rows_with_1', 'no_neigh_horiz', 'custom']
X = df[features]
y = df['multiclass_label']

kfolds = 5
kf = KFold(n_splits=kfolds, shuffle=True, random_state=42)
k_values = [1, 3, 5, 7, 9, 11, 13]

for k in k_values:
    fold_accuracies = []
    y_test_final, y_pred_final = None, None

```

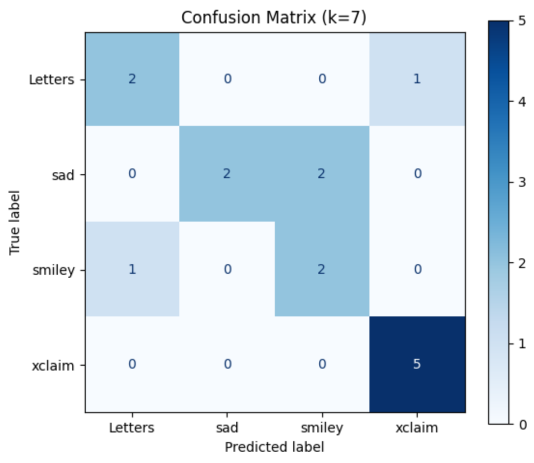


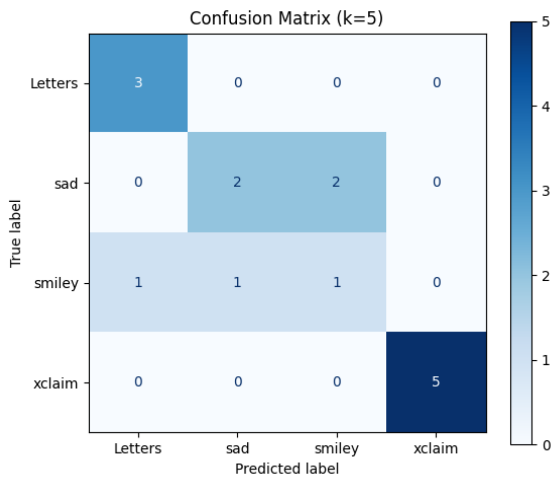
```
    for train_index, test_index in kf.split(df):
        train_data = df.iloc[train_index]
        test_data = df.iloc[test_index]

        X_train = train_data[features]
        y_train = train_data['multiclass_label']
        X_test = test_data[features]
        y_test = test_data['multiclass_label']

        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)

        accuracy = accuracy_score(y_test, y_pred)
        fold_accuracies.append(accuracy)

        y_test_final = y_test
        y_pred_final = y_pred

    average_accuracy = np.mean(fold_accuracies)
    print(f"k = {k}, Average Accuracy = {average_accuracy:.4f}")

    cm = confusion_matrix(y_test_final, y_pred_final, labels=['class1', 'class2', 'class3', 'class4'])
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Letters', 'sad', 'smiley', 'xclaim'])

    fig, ax = plt.subplots(figsize=(6, 5))
    cm_display.plot(ax=ax, cmap='Blues', values_format='d')
    ax.set_title(f"Confusion Matrix (k={k})")
    plt.tight_layout()
    plt.show()
```

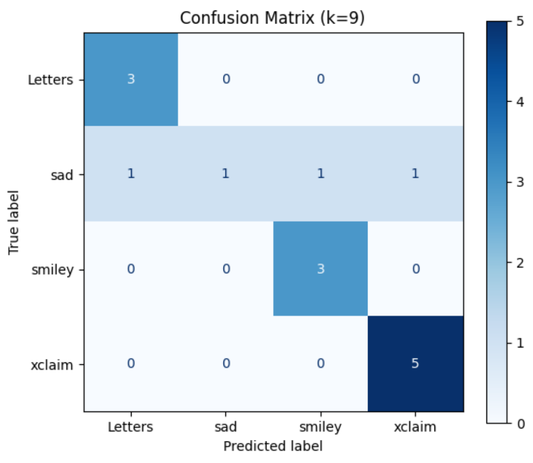
This python program plots all k value graph which allows us to see which pair of class is the most hardest to discriminate which are sad and smiley.



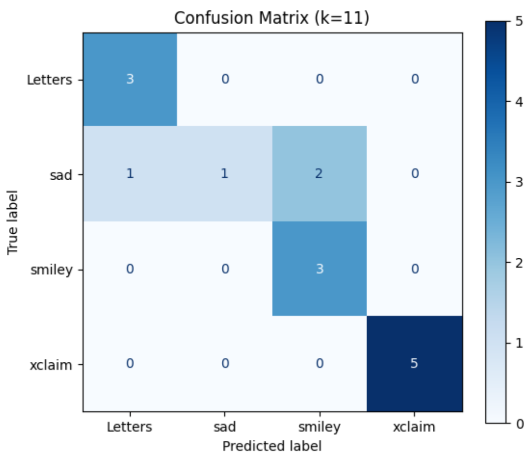




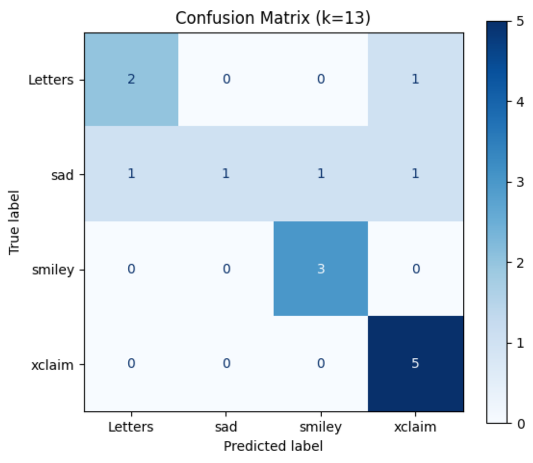




	Letters	sad	smiley	xclaim
Letters	3	0	0	0
sad	1	1	1	1
smiley	0	0	3	0
xclaim	0	0	0	5



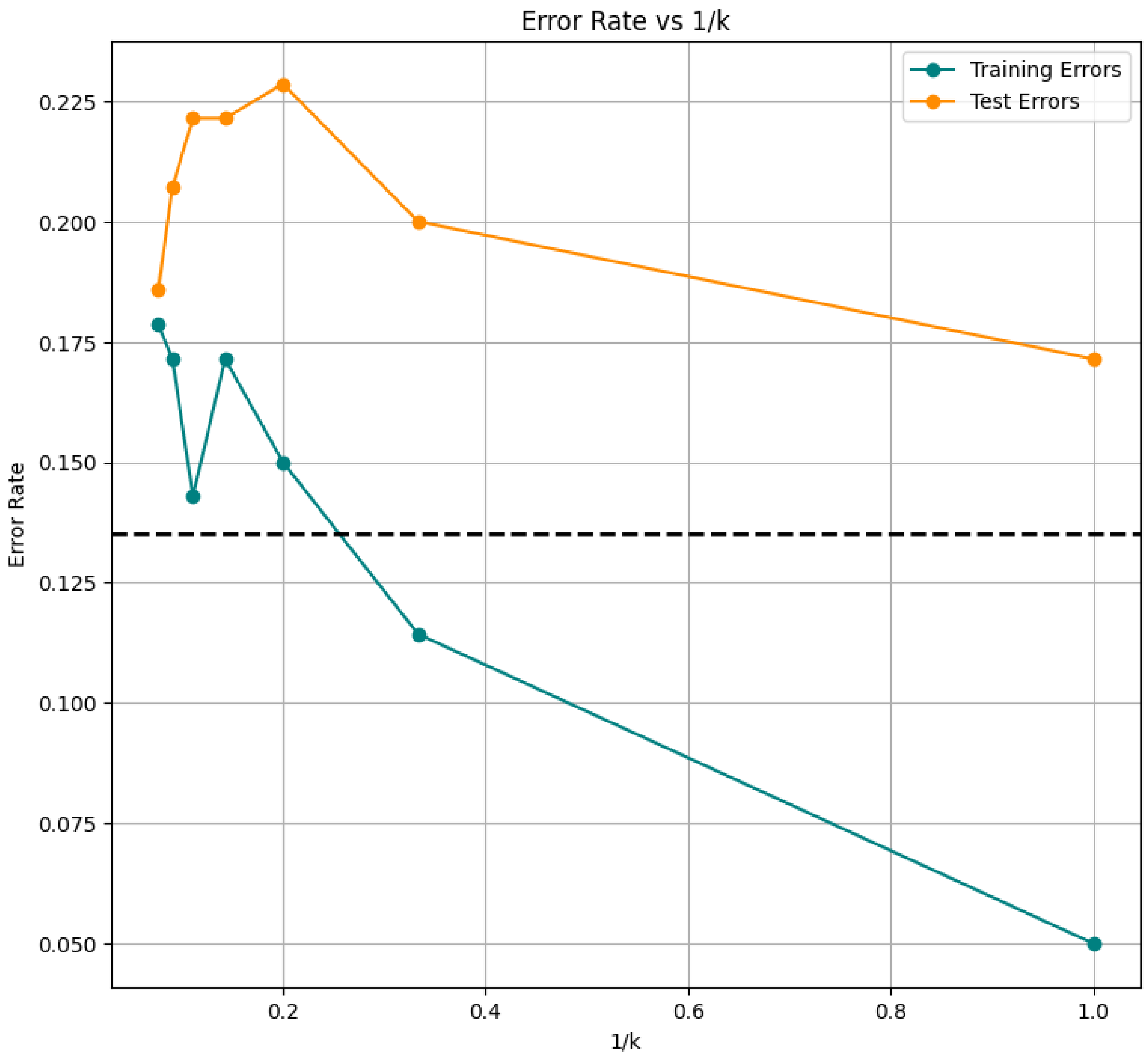
	Letters	sad	smiley	xclaim
Letters	3	0	0	0
sad	1	1	2	0
smiley	0	0	3	0
xclaim	0	0	0	5



	Letters	sad	smiley	xclaim
Letters	2	0	0	1
sad	1	1	1	1
smiley	0	0	3	0
xclaim	0	0	0	5

As we can see from our confusion matrix diagrams, xclaim does not have any other values in other class boxes. For letters, it is almost 3 in every k value except for 7 and 13 where it has a 1 in xclaim column. However sad and smiley are always inconsistent with different values in other classes columns. Through observations for these diagrams, it is concluded that sad and smiley are the two most difficult classes to discriminate.

Section 2.4



Value of Error (No Cross-Validation): [0.050000000000000044, 0.11428571428571432, 0.15000000000000002, 0.17142857142857137, 0.1428571428571429, 0.17142857142857137, 0.1785714285714286]

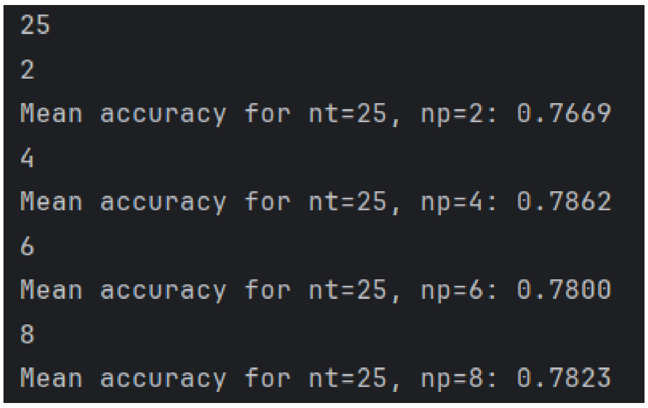
Value of Error (5-fold Cross-Validation): [np.float64(0.17142857142857149), np.float64(0.20000000000000007), np.float64(0.22857142857142854), np.float64(0.22142857142857153), np.float64(0.2214285714285713), np.float64(0.20714285714285707), np.float64(0.18571428571428572)]

With the graph and array of results given, as k decreases, the model complexity increases which improves the training performances but may harm the test performances. For the teal colour data way below the Bayes Error Line, it is judged as overfitting hence as for Test Errors, when k is 1 it is the best model.

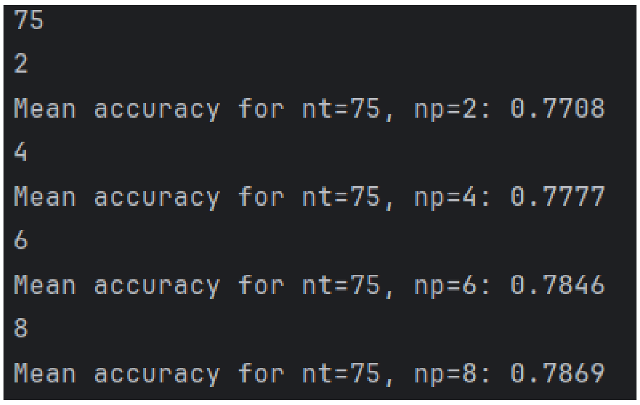
With our diagram, value k = 5 may be the best choice of k as Training Error sits above the Bayes Error and the Test Error Rate may be high but it is the closest for non-overfitting data with its Training Error which generalizes better.

Section 3

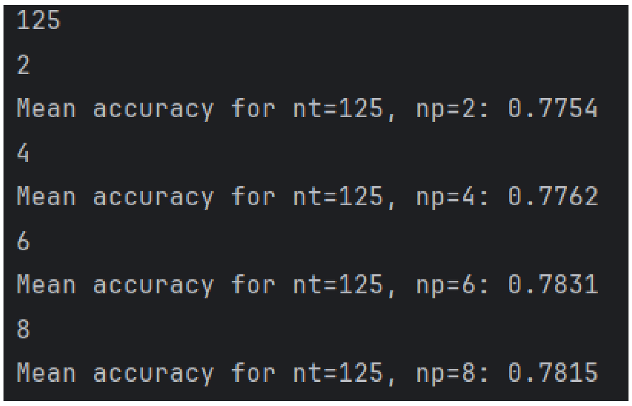
Section 3.1



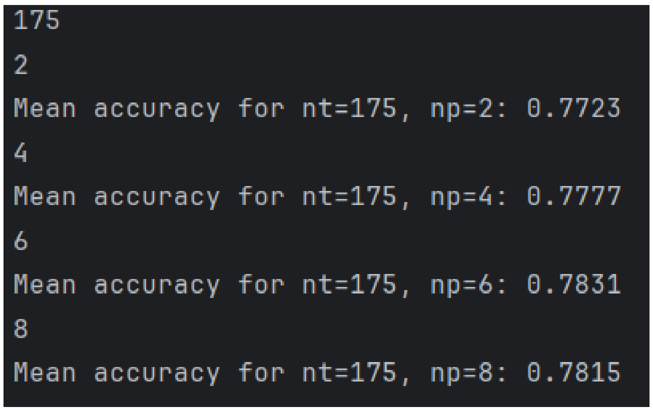
```
25
2
Mean accuracy for nt=25, np=2: 0.7669
4
Mean accuracy for nt=25, np=4: 0.7862
6
Mean accuracy for nt=25, np=6: 0.7800
8
Mean accuracy for nt=25, np=8: 0.7823
```



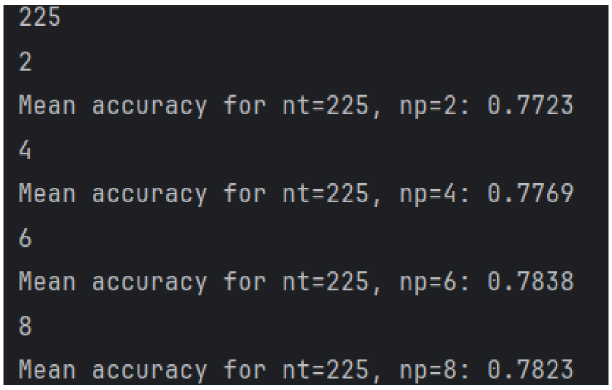
```
75
2
Mean accuracy for nt=75, np=2: 0.7708
4
Mean accuracy for nt=75, np=4: 0.7777
6
Mean accuracy for nt=75, np=6: 0.7846
8
Mean accuracy for nt=75, np=8: 0.7869
```



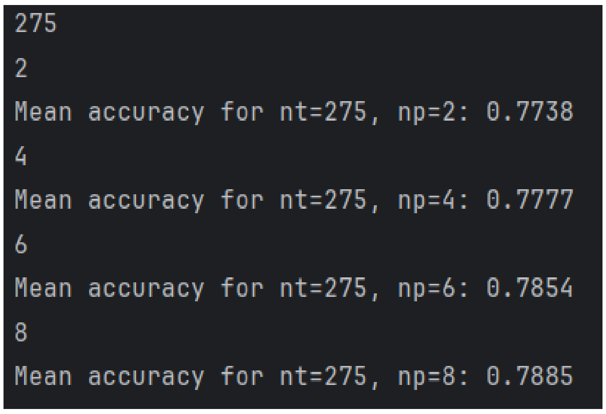
```
125
2
Mean accuracy for nt=125, np=2: 0.7754
4
Mean accuracy for nt=125, np=4: 0.7762
6
Mean accuracy for nt=125, np=6: 0.7831
8
Mean accuracy for nt=125, np=8: 0.7815
```



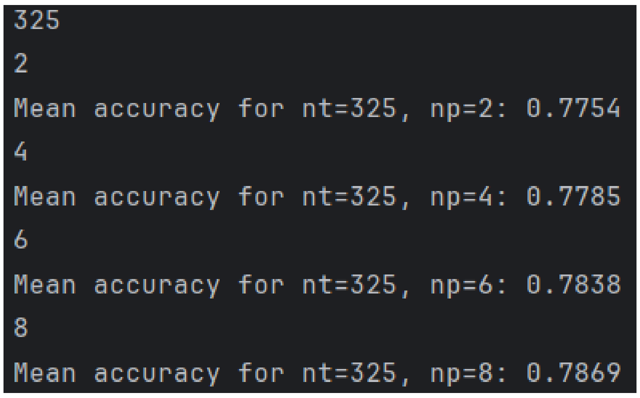
```
175
2
Mean accuracy for nt=175, np=2: 0.7723
4
Mean accuracy for nt=175, np=4: 0.7777
6
Mean accuracy for nt=175, np=6: 0.7831
8
Mean accuracy for nt=175, np=8: 0.7815
```



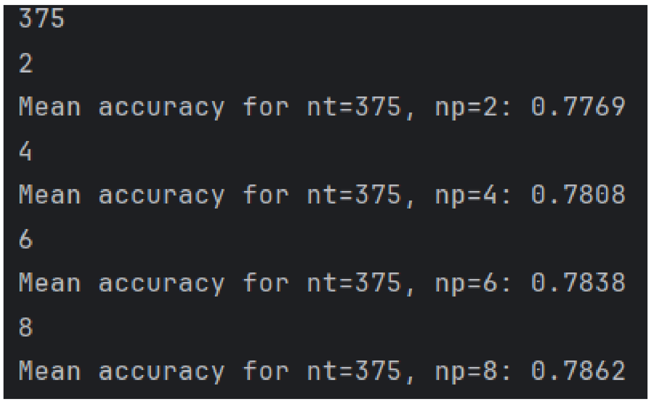
```
225
2
Mean accuracy for nt=225, np=2: 0.7723
4
Mean accuracy for nt=225, np=4: 0.7769
6
Mean accuracy for nt=225, np=6: 0.7838
8
Mean accuracy for nt=225, np=8: 0.7823
```



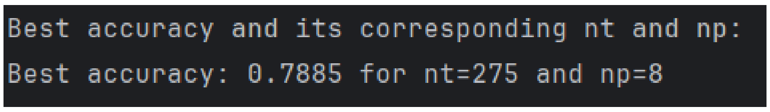
```
275
2
Mean accuracy for nt=275, np=2: 0.7738
4
Mean accuracy for nt=275, np=4: 0.7777
6
Mean accuracy for nt=275, np=6: 0.7854
8
Mean accuracy for nt=275, np=8: 0.7885
```



```
325
2
Mean accuracy for nt=325, np=2: 0.7754
4
Mean accuracy for nt=325, np=4: 0.7785
6
Mean accuracy for nt=325, np=6: 0.7838
8
Mean accuracy for nt=325, np=8: 0.7869
```

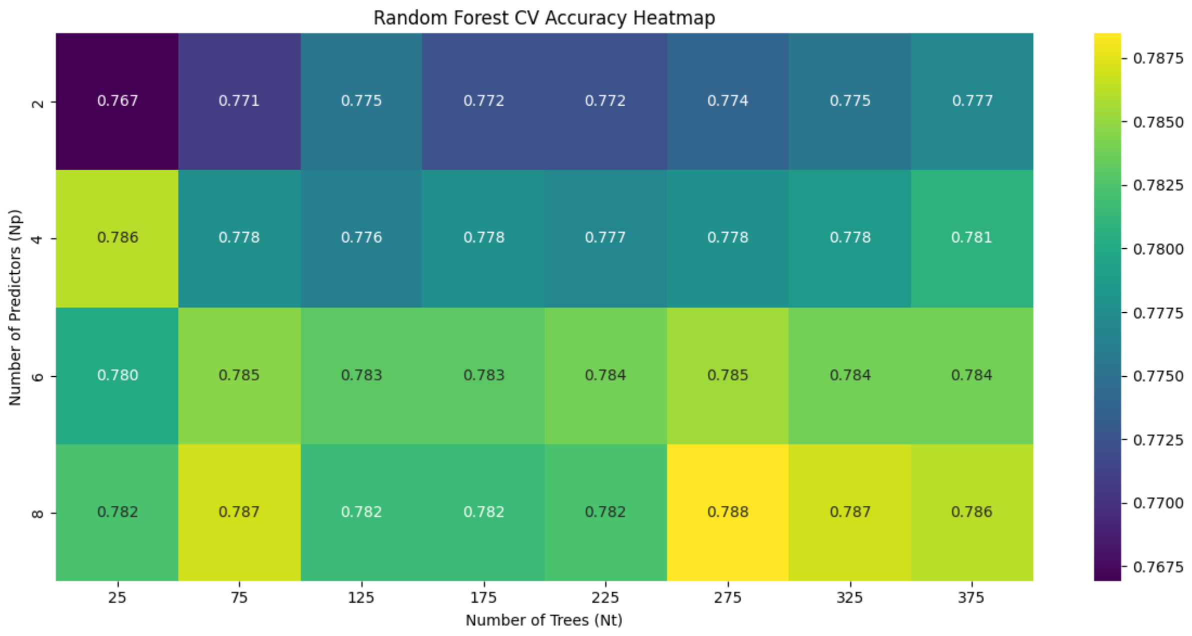


```
375
2
Mean accuracy for nt=375, np=2: 0.7769
4
Mean accuracy for nt=375, np=4: 0.7808
6
Mean accuracy for nt=375, np=6: 0.7838
8
Mean accuracy for nt=375, np=8: 0.7862
```

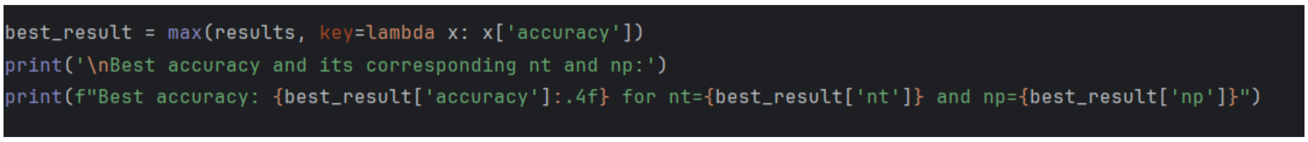


```
Best accuracy and its corresponding nt and np:
Best accuracy: 0.7885 for nt=275 and np=8
```

Heatmap figure for all 4x8 values total = 32



This part of my code conducts a Grid-search:

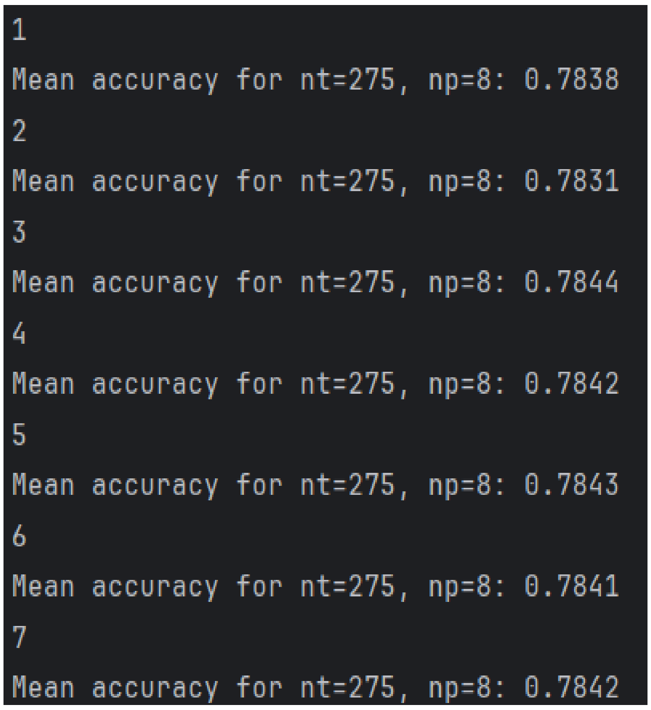


```
best_result = max(results, key=lambda x: x['accuracy'])
print('\nBest accuracy and its corresponding nt and np:')
print(f"Best accuracy: {best_result['accuracy']:.4f} for nt={best_result['nt']} and np={best_result['np']}")
```

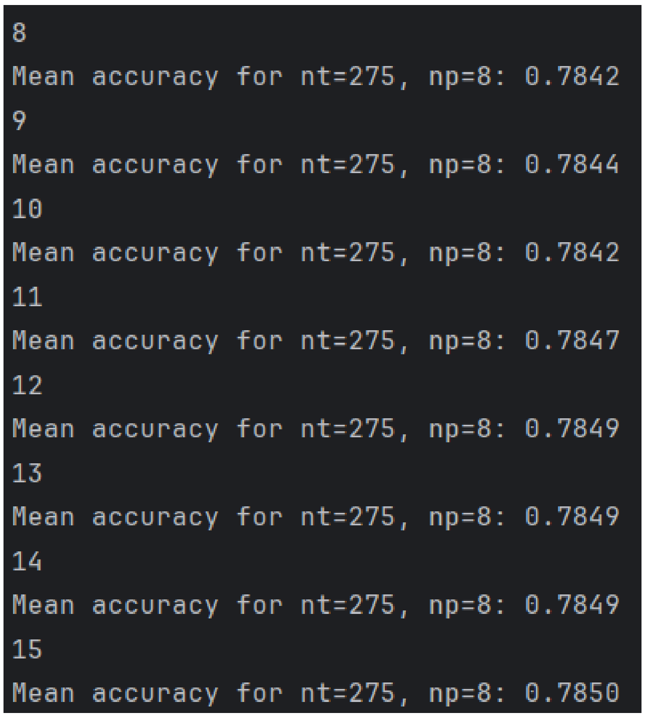
max(results , key=lambda x:x['accuracy']) max takes the maximum and key use for initialize and lambda to allow which values it should compare it to, here it is stores the x values then compare from my results array with all the accuracy then compares with the original accuracy x hence if bigger it will replace it. With our heatmap, the hottest column [3][5] which is Np=8, Nt=275 showing that it has the best accuracy at this specify values.

Section 3.2

Accuracies for all random states using randint(0,1000) using Np=8, Nt=275.

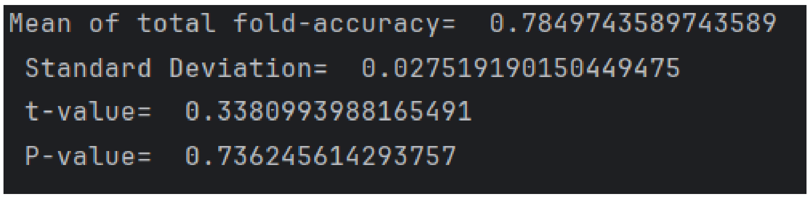


```
1
Mean accuracy for nt=275, np=8: 0.7838
2
Mean accuracy for nt=275, np=8: 0.7831
3
Mean accuracy for nt=275, np=8: 0.7844
4
Mean accuracy for nt=275, np=8: 0.7842
5
Mean accuracy for nt=275, np=8: 0.7843
6
Mean accuracy for nt=275, np=8: 0.7841
7
Mean accuracy for nt=275, np=8: 0.7842
```



```
8
Mean accuracy for nt=275, np=8: 0.7842
9
Mean accuracy for nt=275, np=8: 0.7844
10
Mean accuracy for nt=275, np=8: 0.7842
11
Mean accuracy for nt=275, np=8: 0.7847
12
Mean accuracy for nt=275, np=8: 0.7849
13
Mean accuracy for nt=275, np=8: 0.7849
14
Mean accuracy for nt=275, np=8: 0.7849
15
Mean accuracy for nt=275, np=8: 0.7850
```

Below is the mean of total accuracies, STD (Standard Deviation), t-tests (t-values and p-values)



```
Mean of total fold-accuracy=  0.7849743589743589
 Standard Deviation=  0.027519190150449475
 t-value=  0.3380993988165491
 P-value=  0.736245614293757
```

Conclusions

In this assignment, I am able to use python programming to conduct Machine Learning specifically on supervised learning: logistic regression, Nearest and K-Nearest Neighbour and Random Forests. Using data from Assignment 1 and data provided by lecturer, I am able to conduct all questions with specific python libraries. Some difficulties are faced such as the logic of the Python libraries and different features that are used in the assignment such as lambda, subplots and many more are researched during the time of assignment doing to further my understanding of the code and improve data visualisations.

References

[1]. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.multilabel_confusion_matrix.html

[2]. https://medium.com/@sanyagubrani/evaluating-multi-class-classification-model-using-confusion-matrix-in-python-4d9344084dfa

[3]. https://www.w3schools.com/python/python_ml_grid_search.asp

[4]. https://www.w3schools.com/python/scipy/scipy_statistical_significance_tests.php

[5]. https://www.w3schools.com/python/python_ml_getting_started.asp