Task 1:

In this task, I trained a Logistic Regression Model for data. The outputs are confusion matrix, accuracy, f-measure, recall, precision.
Confusion matrix is a matrix including true positive, false negative, false positive, and true negative for prediction and actual class.
Accuracy matrices can estimate the accuracy of the prediction and actual data.
F-measure can test the accuracy of positive prediction and positive actual data.
Recall can test the accuracy based on positive actual data.
Precision can test the accuracy based on positive prediction data.
The formulas of above are the following:

$$\text{Confusion Matrix} = \text{A Matrix including True Positive, False Positive, False Positive, True Negative}$$

$$\text{Accuracy} = \frac{\text{True Positive+False Negative}}{\text{True Positive+False Positive+False Positive+True Negative}}$$

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive+False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\text{F1} = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision + Recall}}$$

The outputs for accuracy, f-measure, recall, precision are arrays of scores of the estimator for each run of the cross validation.The output for confusion matrix is a 2 by 2 matrix.

The output of unnormalized data is the following:

```
confusion_matrix(y_test, pred)
```

```
array([[3720,    2],
       [1078,    0]], dtype=int64)
```

```
accuracy, f1, recall, precision
```

```
(array([0.77645833, 0.77645833, 0.77645833, 0.77666667, 0.77583333]),
 array([0.        , 0.        , 0.00186047, 0.00371747, 0.00185529]),
 array([0.        , 0.        , 0.00093197, 0.0018622 , 0.0009311 ]),
 array([0.  , 0.  , 0.5 , 1.  , 0.25]))
```

The output of normalized data is the following:

```
confusion_matrix(y_test, pred)
```

```
array([[3557,  165],
       [ 681,  397]], dtype=int64)
```

```
accuracy, f1, recall, precision
```

```
(array([0.81791667, 0.81791667, 0.823125  , 0.81625   , 0.81875   ]),
 array([0.46115906, 0.46314496, 0.4781807 , 0.45955882, 0.48028674]),
 array([0.34855545, 0.35135135, 0.36253495, 0.34916201, 0.37430168]),
 array([0.68123862, 0.67927928, 0.70216606, 0.67204301, 0.67      ]))
```

Then, comparing these outputs, we conclude that the second model has higher accuracy, higher f-measure, higher recall, and higher precision than the first model.

Task 2:

We want to avoid imbalance issues in training classifiers, because if we have imbalanced classes, we may get high overall accuracy easily. However, the result is not accurate. In order to solve the problem, we can simply **collect more data**, or **resampling the data**(oversample or undersample), try **using a penalized model**.

In this task, I used an oversample method.

The output of unnormalized data is the following:

```
confusion_matrix(y_test, pred)
```

```
array([[2096, 1626],
       [ 361,  717]], dtype=int64)
```

```
accuracy_score(y_test, pred), f1_score(y_test, pred), recall_score(y_test, pred), precision_score(y_test, pred)
```

```
(0.5860416666666667,
 0.41917567962584046,
 0.6651205936920223,
 0.3060179257362356)
```

The output of normalized data is the following:

```
confusion_matrix(y_test, pred)
```

```
array([[3156,  566],
       [ 476,  602]], dtype=int64)
```

```
accuracy_score(y_test, pred), f1_score(y_test, pred), recall_score(y_test, pred), precision_score(y_test, pred)
```

```
(0.7829166666666667,
 0.5360641139804095,
 0.5584415584415584,
 0.5154109589041096)
```

Only oversample the training data because if the training data is unbalanced, the test data will also be unbalanced.
Comparing the outputs of task 1 and task 2, I noticed that the overall accuracy of both models are decreasing.

Task 3:

The main reason to do feature selection is that it allows the model to train faster. If some most correlated features are chosen, the accuracy will be improved.

The output when k=1:
----------k=1----------
[[3720 2]
[1078 0]]
[0.77645833 0.77645833 0.77645833 0.77666667 0.77583333] [0. 0. 0.00186047 0.00371747 0.00185529] [0. 0. 0.00093197 0.0018622 0.0009311] [0. 0. 0.5 1. 0.25]

The output when k=3:
----------k=3----------
[[3720 2]
[1078 0]]

[0.77645833 0.77645833 0.77645833 0.77666667 0.77583333] [0. 0. 0.00186047 0.00371747 0.00185529] [0. 0. 0.00093197 0.0018622 0.0009311] [0. 0. 0.5 1. 0.25]

The output when k=5:
----------k=5----------
[[3720 2]
[1078 0]]

[0.77645833 0.77645833 0.77645833 0.77666667 0.77583333] [0. 0. 0.00186047 0.00371747 0.00185529] [0. 0. 0.00093197 0.0018622 0.0009311] [0. 0. 0.5 1. 0.25]

From the output above, it seems that no matter the value of k, the result trunks out the same. The reason is that we already choose from the most correlated features, so the result will be very close or even the same.

Task 4:

The output of decision tree classifier using unnormalized value:

```
print(accuracy, f1, recall, precision)
```

```
[0.7175     0.71645833 0.71875    0.723125   0.725     ] [0.39285714 0.39360284 0.38677536 0.39124487 0.41777778] [0.40447344
0.40447344 0.40074557 0.40316574 0.45251397] [0.37637131 0.37629758 0.37821081 0.37687555 0.40840336]
```

The output of decision tree classifier using normalized value:

```
print(accuracy, f1, recall, precision)
```

```
[0.71645833 0.710625   0.71916667 0.724375   0.73354167] [0.38758993 0.38052702 0.38220608 0.39435337 0.42282176] [0.416589
0.39235788 0.39701771 0.39851024 0.43482309] [0.36762226 0.37565217 0.37760417 0.38983051 0.4040747 ]
```

These two decision tree classifiers are relatively the same by comparing the overall accuracy, but they are slightly lower than logistic regression in task 1.

The output of SVM using unnormalized value:

```
[0.77645833 0.77645833 0.77645833 0.77625    0.77625    ] [0. 0. 0. 0. 0.] [0. 0. 0. 0. 0.] [0. 0. 0. 0. 0.]
```

The output of SVM using normalized value:

```
print(accuracy, f1, recall, precision)
```

```
[0.81958333 0.816875   0.82270833 0.81729167 0.82       ] [0.46410891 0.44542587 0.46845721 0.4616329  0.48263473] [0.34948742
0.32898416 0.34948742 0.35009311 0.37523277] [0.69060773 0.68945312 0.71022727 0.67747748 0.6761745 ]
```

These two SVM classifiers are close to each other by comparing the overall accuracy, and they are about the same compared to task 1.

The output of MLPClassifier using unnormalized value:

```
print(accuracy, f1, recall, precision)
```

```
[0.68229167 0.77541667 0.76479167 0.633125   0.74875    ] [0.06748971 0.41146319 0.36659142 0.38739197 0.17823229] [0.24883504
0.65517241 0.14911463 0.13500931 0.2877095 ] [0.42241379 0.24754902 0.40304183 0.39316239 0.32716763]
```

The output of MLPClassifier using normalized value:

```
[0.81125    0.80708333 0.80708333 0.80083333 0.81041667] [0.47498562 0.47025172 0.45258103 0.46283784 0.46845794] [0.38676608
0.37371855 0.35787512 0.37243948 0.39664804] [0.60252809 0.61601307 0.60995185 0.64915254 0.62053571]
```

These two SVM classifiers are close to each other by comparing the overall accuracy, and MLPClassifier using normalized value is about the same compared to task 1. MLPClassifier using unnormalized value is slightly lower compared to task 1.

Task 5:

For this task, I chose Logistic Regression and Decision Tree.

Logistic regression grid search:
The parameters are {"penalty": ["l1", "l2"], "C": [0.5, 1.0, 1.5], "max_iter": [50, 100, 150]
I used gsearch.best_score_, gsearch.best_params_ to get the result
The outputs are the following:

```
(0.7765624999999999, {'C': 0.5, 'max_iter': 50, 'penalty': 'l2'})
```

Decision tree grid search:
The parameters are {"max_depth": [16, 32, 64, 128], "min_samples_split": [2, 4, 6, 8, 10]}
I used gsearch.best_score_, gsearch.best_params_ to get the result
The outputs are the following:

```
print(gsearch.best_score_, gsearch.best_params_)
```

```
0.7870833333333334 {'max_depth': 16, 'min_samples_split': 10}
```

By comparing these two, we can say the second one is slightly better than the first one.