Amirkabir University of Technology
(Tehran Polytechnic)

Computer Engineering Department

Machine Learning Course
CE5501

Project

Student
Reza Sajedi
400131072
r.sajedi@aut.ac.ir

Teacher
Dr. Nazerfard

Fall 2022

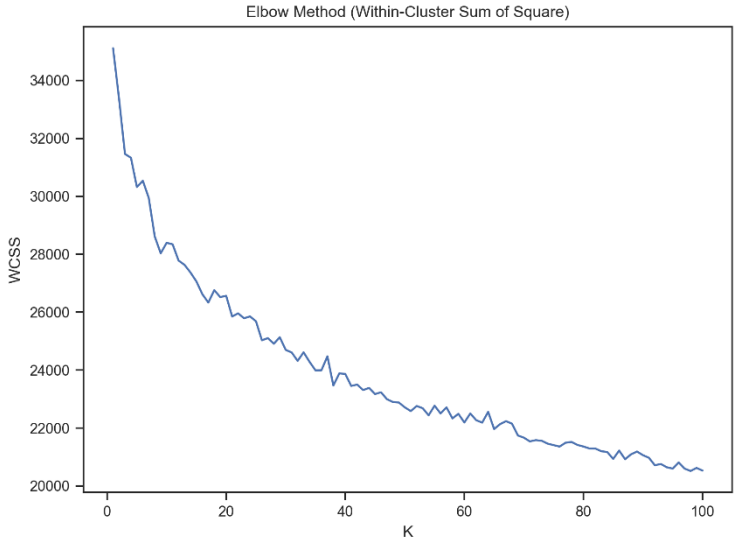
Table of Contents

Problem 1: Regression in Industry

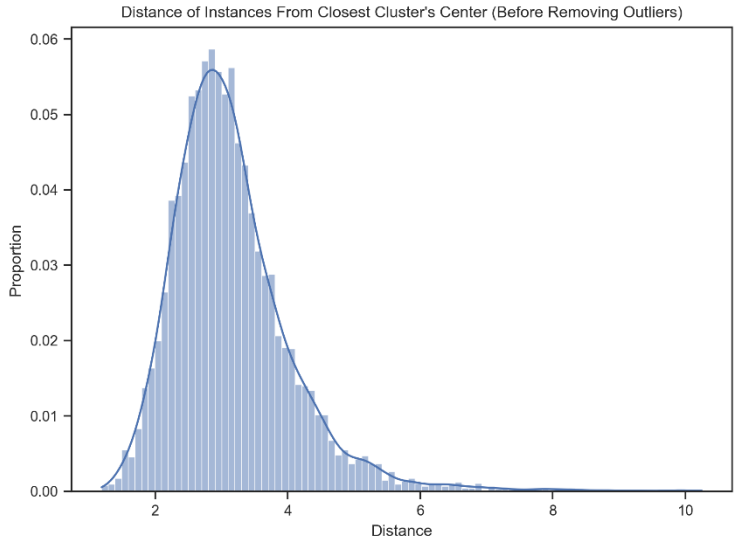
Part a

Normalization has been done using Standard Scaler. Missing values have been imputed by mean.

In order to apply outlier detection using K-means, first we use the elbow method to find a suitable value for K.

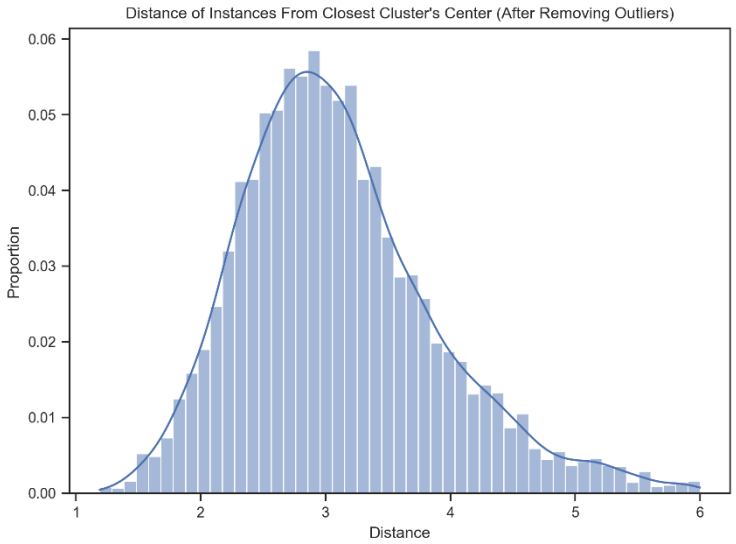


According to the above chart, it seems that 30 is a good value for K. After clustering the data points, we calculate the distance of each data point from its cluster's center and generate a Histogram.

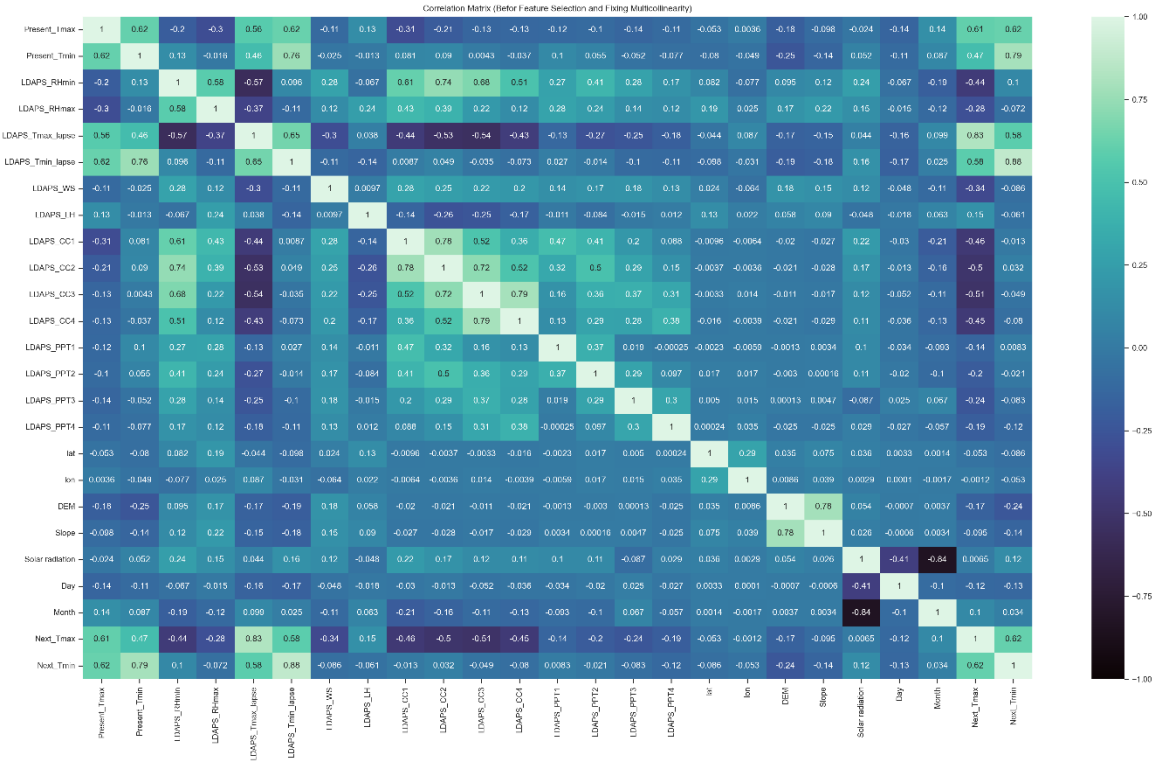


According to the above chart, the data points with distances higher than 6 may be potential outliers. After removing such data points which constitute about 1% of the data set, we get the following Histogram.

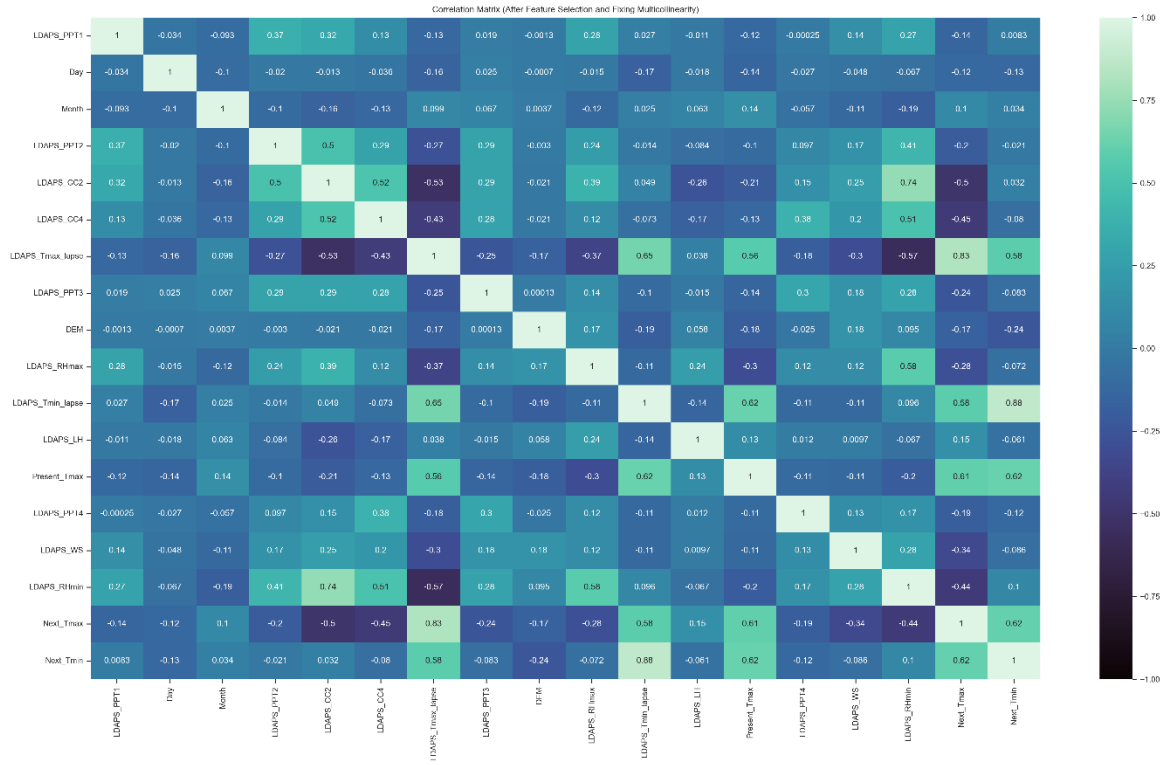
```
The proportion of outliers: 0.0131
```



Now we should remove some unnecessary features and fix the multicollinearity problem. The initial correlation matrix is as follows.



We remove the features which have a correlation value lower than 0.1 compared to both target features. We also fix the multicollinearity problem by dropping one of the features which have a correlation coefficient higher than 0.75. Now the correlation matrix is as follows:

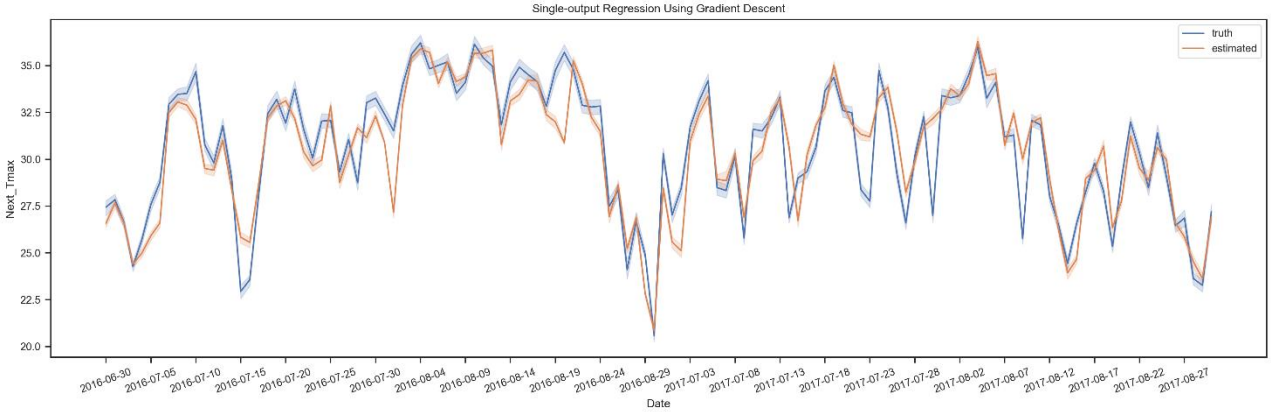


Part b

We trained a single output Linear Regression model by optimizing the MAE loss function using Gradient Descent. This model has been widely used in regression tasks. We tried several values for the hyperparameters of the model and reached the followings which lead to the lowest SSE value:

```
Degree (complexity): 1
Learning rate: 0.01
Max iterations: 1000
Convergence: 1e-10
```

```
SSE for Next_Tmax: 8481.8899
```



Part c

This problem is called Multi-output Regression. There are several strategies for solving this problem:

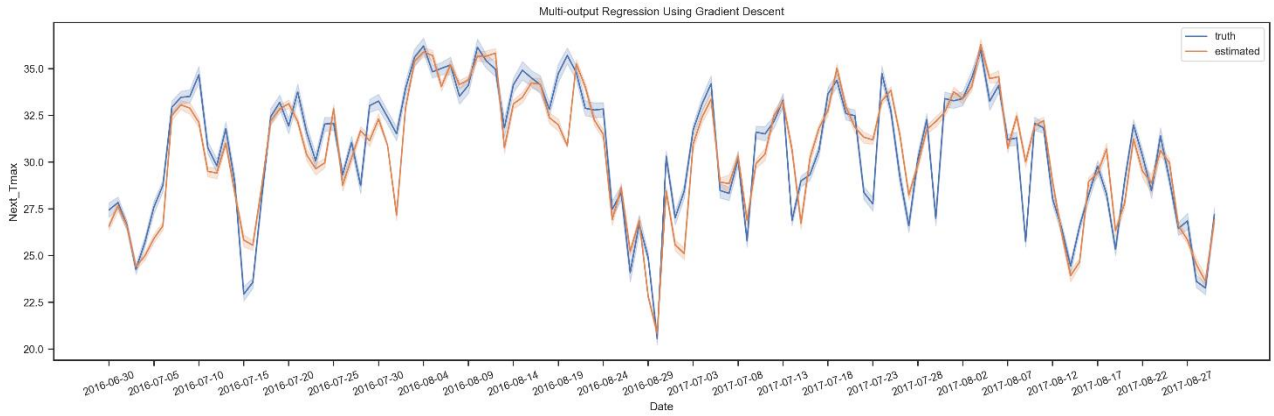
Multi-target regression: This strategy consists of fitting one regressor per target. This is a simple strategy for extending regressors that do not natively support multi-target regression.

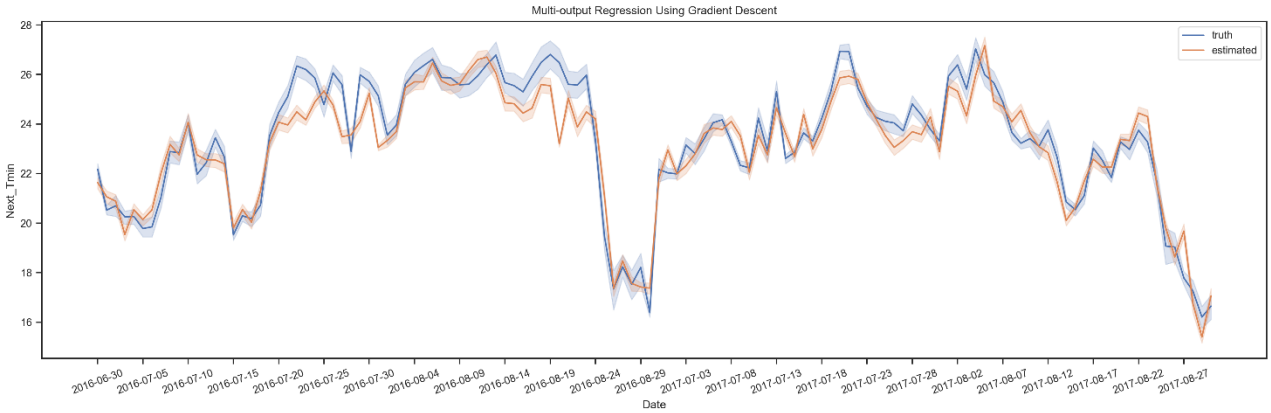
Chained multi-target regression: Each model predicts in the order specified by the chain using all of the available features provided to the model plus the predictions of models that are earlier in the chain.

The two above strategies can be used for any single-output model which doesn't natively support multi-output regression. Fortunately, the model that we used for the previous part, natively supports the extension for the multi-output regression. So, we use the third strategy which exploits this important capability of the model. In this strategy instead of learning a vector of parameters (weights) for the model, we learn a matrix of parameters. The number of rows in this matrix is equal to the number of features of the data set and the number of columns is equal to the number of outputs. The vector y which stores the true values for each instance is also converted to a matrix. The number of columns in this matrix is equal to the number of outputs. Using this strategy leads to optimizing the loss function and tuning the parameters by considering all of the outputs and their impacts on each other.

Part d

```
Degree (complexity): 1
Learning rate: 0.01
Max iterations: 1000
Convergence: 1e-10

Total SSE: 12513.7757
SSE for Next_Tmax: 8481.9516
SSE for Next_Tmin: 4031.8241
```





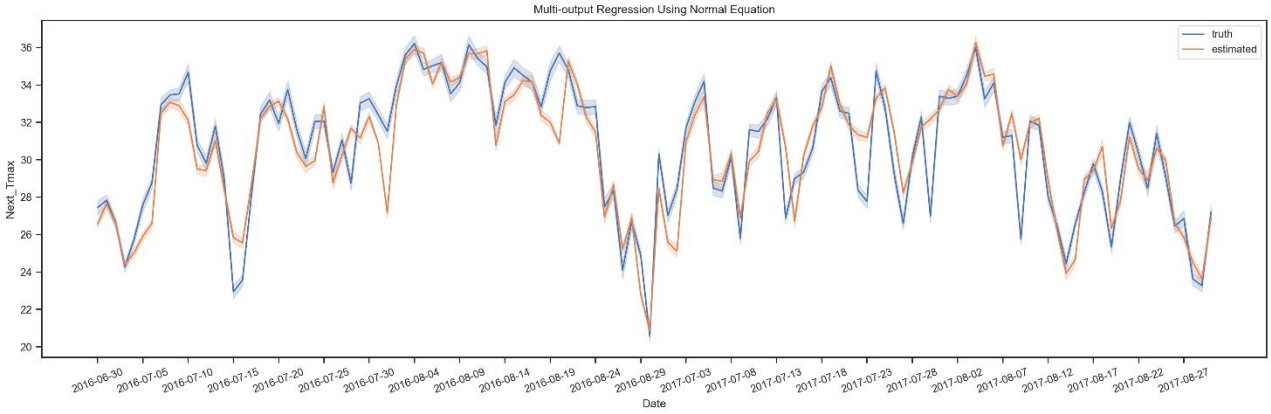
Part e

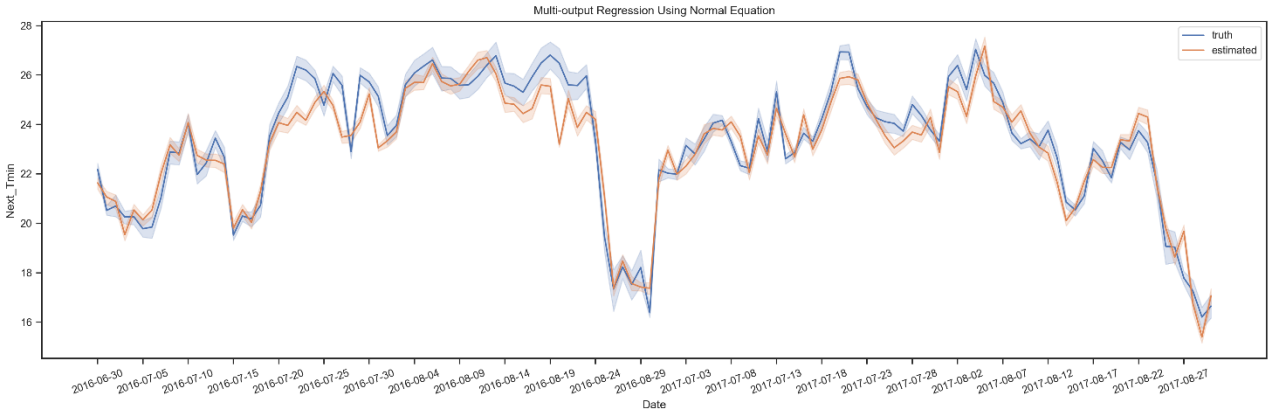
Compared with the previous part, the results are very similar. It's better to use a model which can estimate multiple columns. Because in most cases multiple outputs are correlated and have some impacts on each other, it must be considered in learning the model's parameters. For example, in this specific application, the maximum and minimum temperature of the next day are highly correlated, and also it is not valid for the minimum temperature in a day to be greater than the maximum value.

Part f

The normal equation for the multi-output regression is very similar to the case that is used for the single-output regression. The only difference is that the vector y which stores the true values for each instance is converted to a matrix. The number of columns in this matrix is equal to the number of outputs. In this case, we get a matrix of parameters instead of a vector.

```
Total SSE: 12520.0062
SSE for Next_Tmax: 8488.099
SSE for Next_Tmin: 4031.9072
```

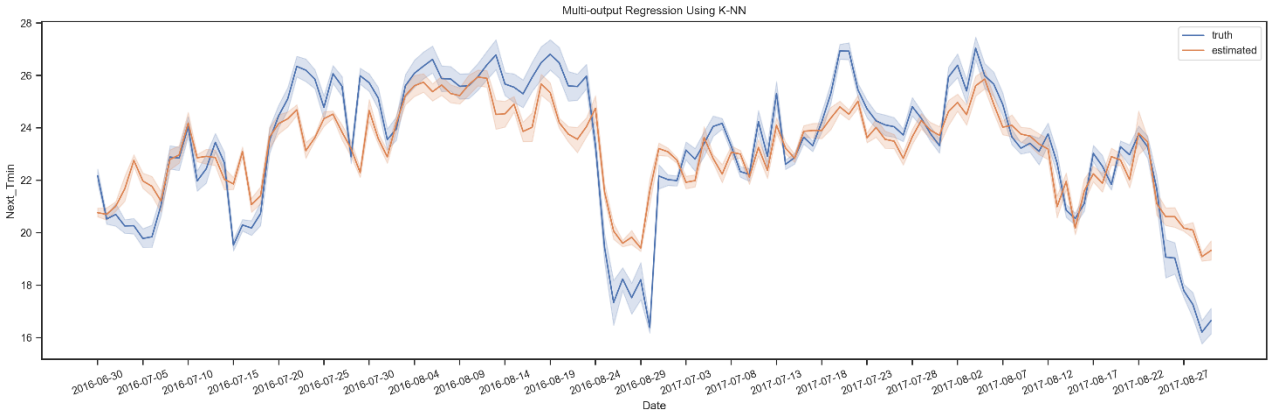




Part g

For each instance, we find the K most similar instances and calculate the mean of each output. We used the Euclidean metric for the distance calculation. We considered 7 for the K hyperparameter. Compared with the previous model, the SEE value is much higher. So, the previous model may be preferred.

```
Total SSE: 19536.1024
SSE for Next_Tmax: 11314.3245
SSE for Next_Tmin: 8221.778
```



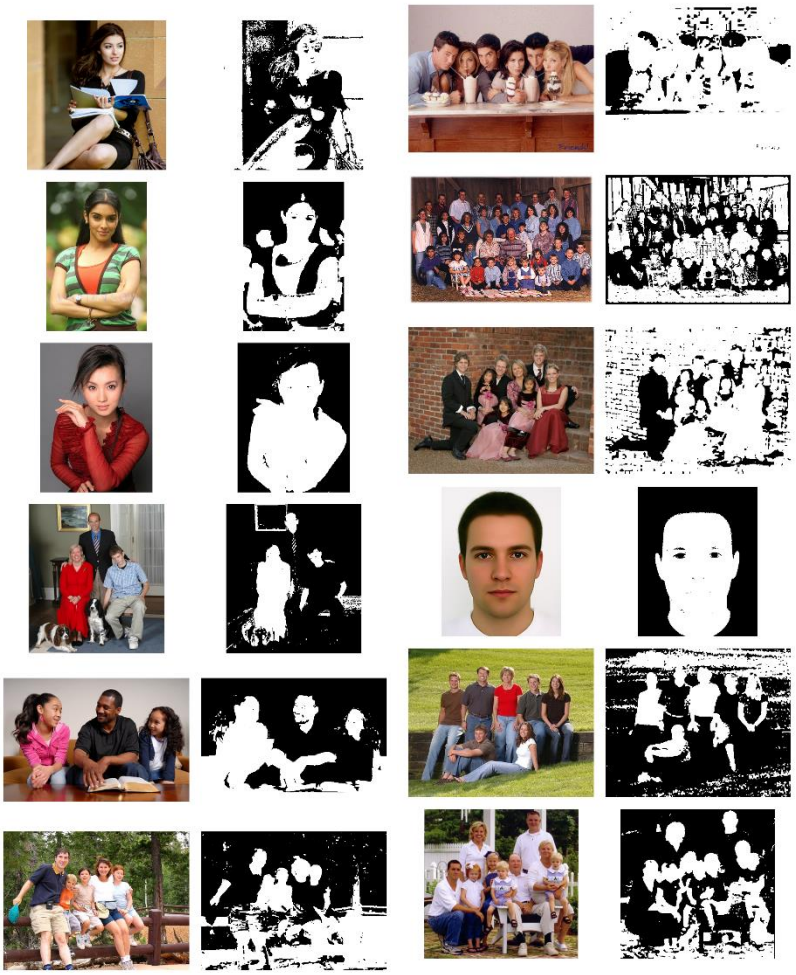
Problem 2: Skin Detection

Part a

We considered 4 components, full covariance type that each component has its own general covariance matrix, and 100 for the max iterations. The method k-means is used to initialize the weights, the means, and the precisions.

Part b

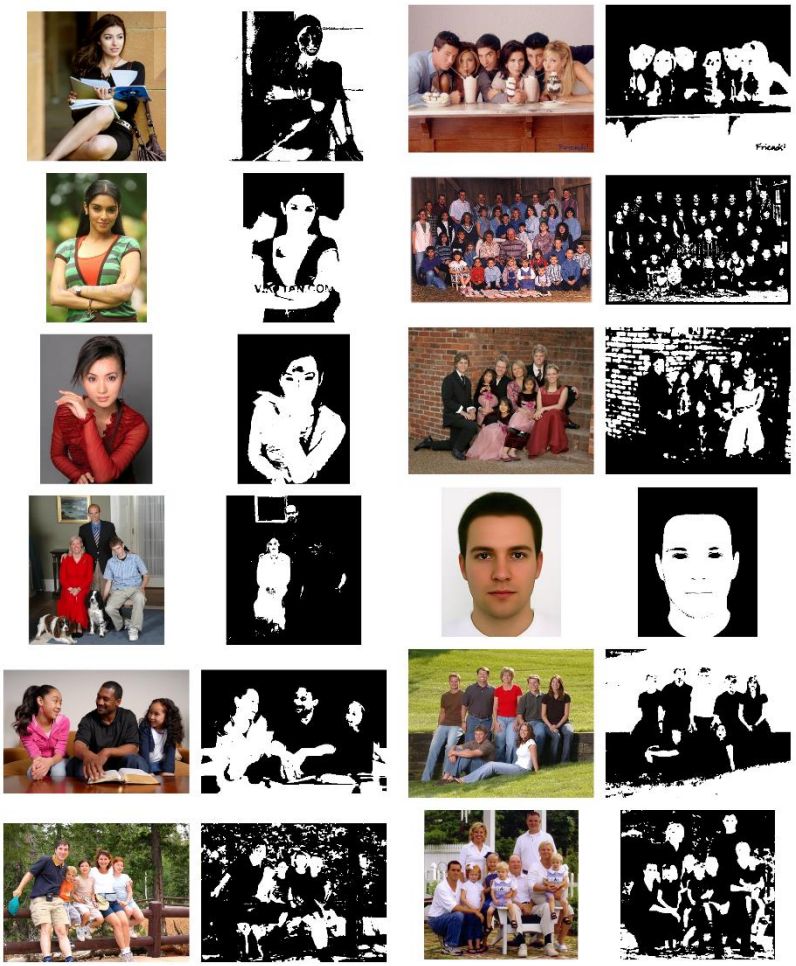
```
Results for GaussianMixture:
              precision    recall  f1-score   support

        Skin       0.32      0.98      0.48    423636
    Non-skin       0.99      0.61      0.76   2289542

    accuracy                           0.67   2713178
   macro avg       0.65      0.79      0.62   2713178
weighted avg       0.89      0.67      0.71   2713178

Confusion Matrix:
[[0.5156 0.3283]
 [0.0039 0.1522]]
```



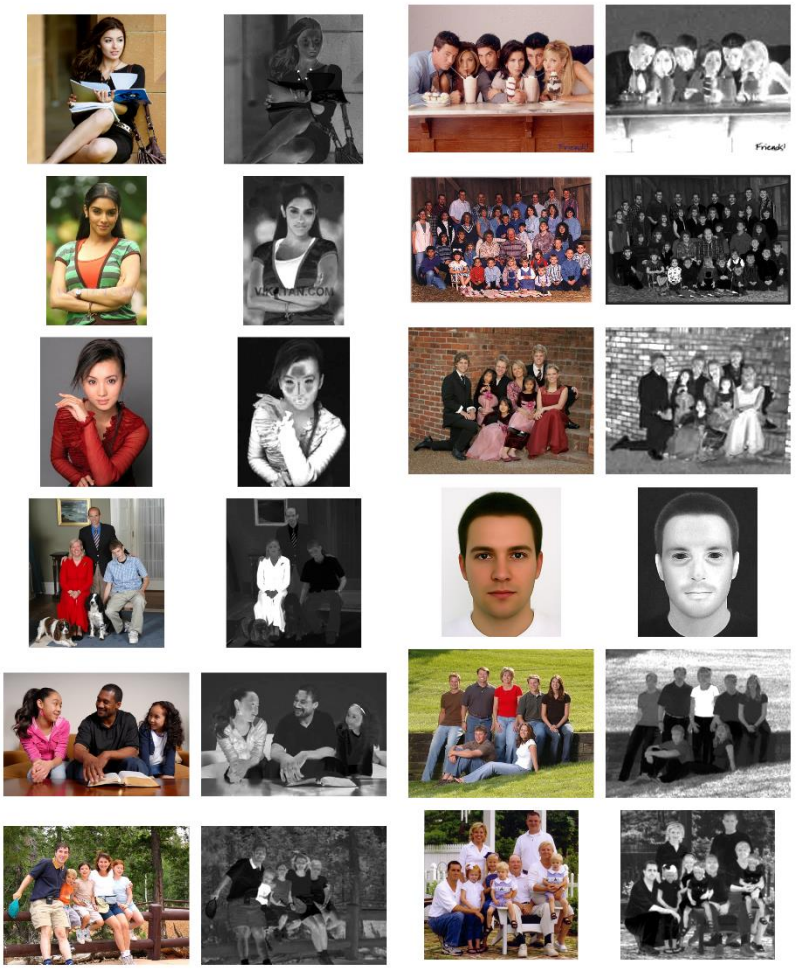
Part c

GNB has a better performance than GMM according to accuracy and f1 score. GMM is natively a clustering algorithm and it doesn't exploit the training truth labels. So, it cannot separate the instances in two classes appropriately and the entropy of its clusters is so high. GNB is a classification algorithm that exploits the training truth labels and aims to distinguish the classes appropriately. So, we expect GNB to have a better performance.

```
Results for GaussianNB:
              precision    recall  f1-score   support

        Skin       0.39      0.89      0.54    423636
    Non-skin       0.97      0.74      0.84   2289542

    accuracy                           0.76   2713178
   macro avg       0.68      0.81      0.69   2713178
weighted avg       0.88      0.76      0.79   2713178

Confusion Matrix:
[[0.6244 0.2195]
 [0.0179 0.1382]]
```



Part d



Part e

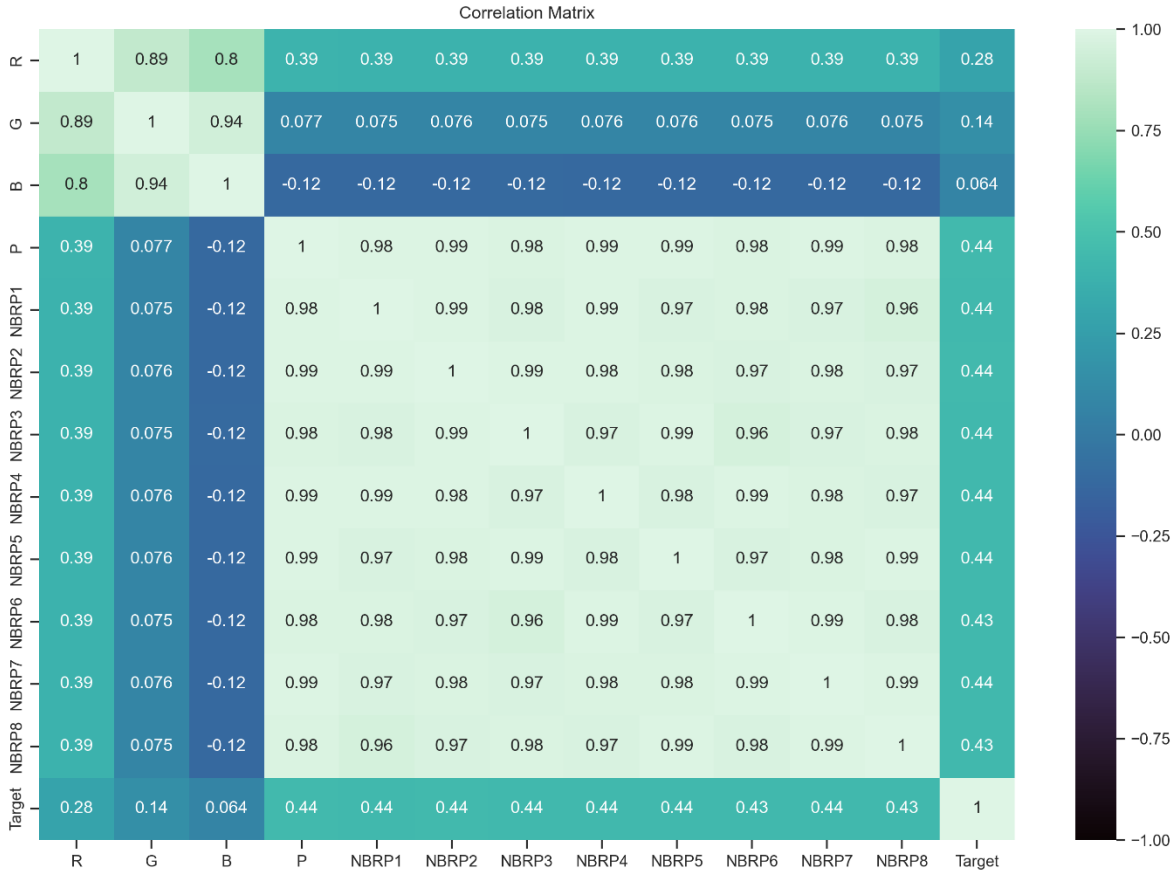
	R	G	B	P	NBRP1	NBRP2	NBRP3	NBRP4	NBRP5	NBRP6	NBRP7	NBRP8	Target
0	215.0	203.0	177.0	0.199259	0.000000	0.000000	0.000000	0.000000	0.205903	0.000000	0.222036	0.234278	0
1	199.0	188.0	160.0	0.205903	0.000000	0.000000	0.000000	0.199259	0.205908	0.222036	0.234278	0.234283	0
2	204.0	193.0	165.0	0.205908	0.000000	0.000000	0.000000	0.205903	0.205907	0.234278	0.234283	0.239435	0
3	203.0	192.0	164.0	0.205907	0.000000	0.000000	0.000000	0.205908	0.205909	0.234283	0.239435	0.226891	0
4	205.0	194.0	166.0	0.205909	0.000000	0.000000	0.000000	0.205907	0.205913	0.239435	0.226891	0.226894	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
14126842	164.0	165.0	167.0	0.098849	0.095542	0.100970	0.106090	0.094413	0.102363	0.000000	0.000000	0.000000	0
14126843	162.0	160.0	163.0	0.102363	0.100970	0.106090	0.108355	0.098849	0.106086	0.000000	0.000000	0.000000	0
14126844	164.0	159.0	163.0	0.106086	0.106090	0.108355	0.105994	0.102363	0.105993	0.000000	0.000000	0.000000	0
14126845	166.0	156.0	164.0	0.105993	0.108355	0.105994	0.105991	0.106086	0.105991	0.000000	0.000000	0.000000	0
14126846	163.0	153.0	161.0	0.105991	0.105994	0.105991	0.000000	0.105993	0.000000	0.000000	0.000000	0.000000	0

14126847 rows × 13 columns

Part f

The correlation coefficient is a statistical measure of the strength of a linear relationship between two variables. Its values can range from -1 to 1. A correlation coefficient of -1 describes a perfect negative, or inverse, correlation. A coefficient of 1 shows a perfect positive correlation or a direct relationship. A correlation coefficient of 0 means there is no linear relationship. One of its uses is in feature selection that features which have low correlations with the target feature can be removed. It can also be used to fix the multicollinearity problem.

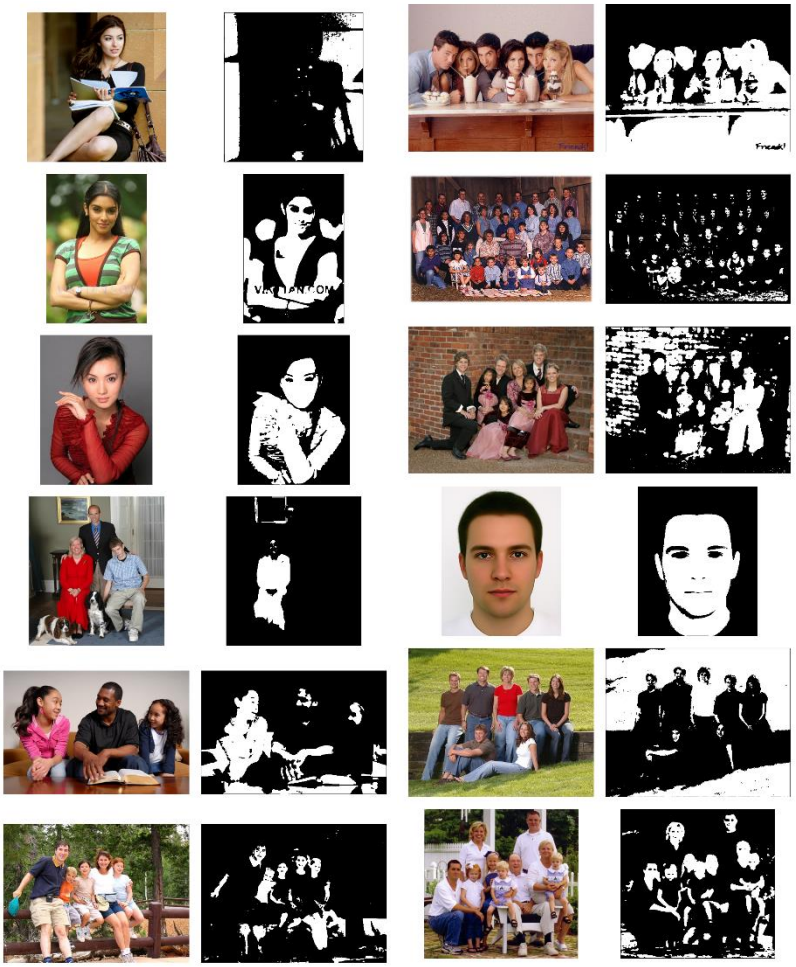
For this synthetic data set, neighbor probabilities are highly correlated, because the neighbors of a pixel in a picture usually have the same colors.



Part g

This model has a better performance according to Accuracy and F1 Score. I think the reason is that when we also consider the neighbors of a pixel for prediction, the model becomes more robust against noise (outlier).

```
Results for LogisticRegression:
              precision    recall  f1-score   support

        Skin       0.42      0.78      0.55    423636
    Non-skin       0.95      0.80      0.87   2289542

    accuracy                           0.80   2713178
   macro avg       0.69      0.79      0.71   2713178
weighted avg       0.87      0.80      0.82   2713178

Confusion Matrix:
[[0.6757 0.1682]
 [0.034  0.1221]]
```

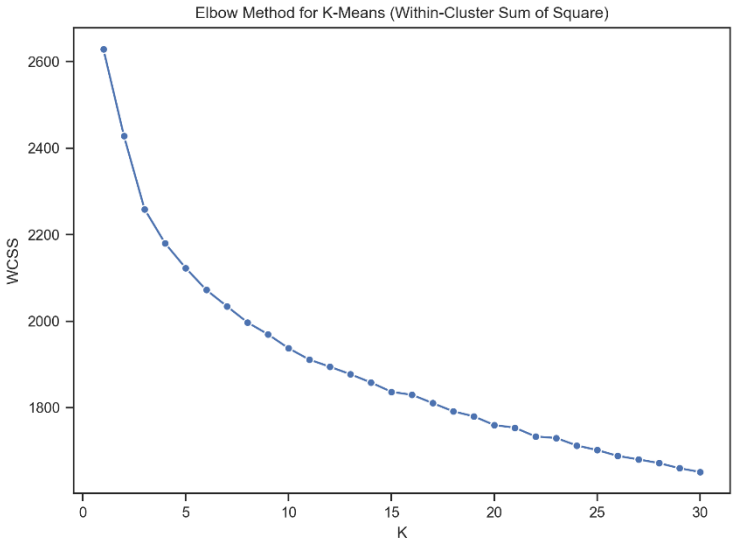


Problem 3: Credit Approval

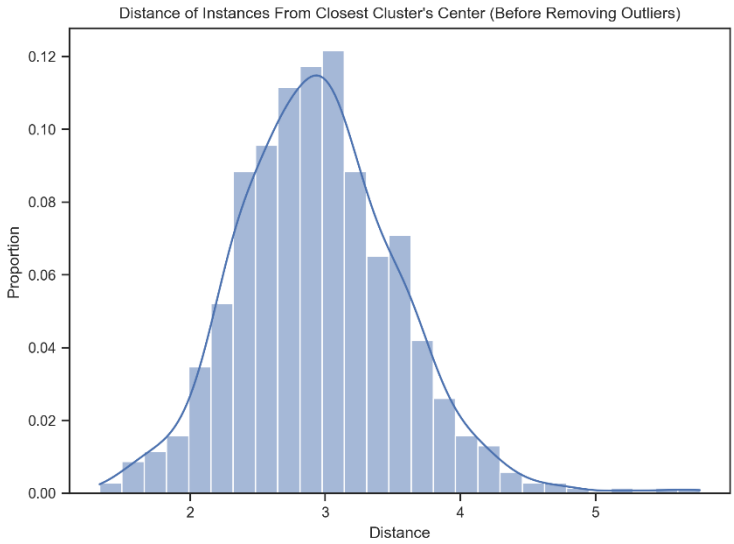
Part a

We impute missing values using the most frequent value in each column. Nominal values are converted to numerals using Label Encoder, and normalization is done using Standard Scaler.

For outlier detection using K-means, we first use the Elbow method to detect the appropriate K.

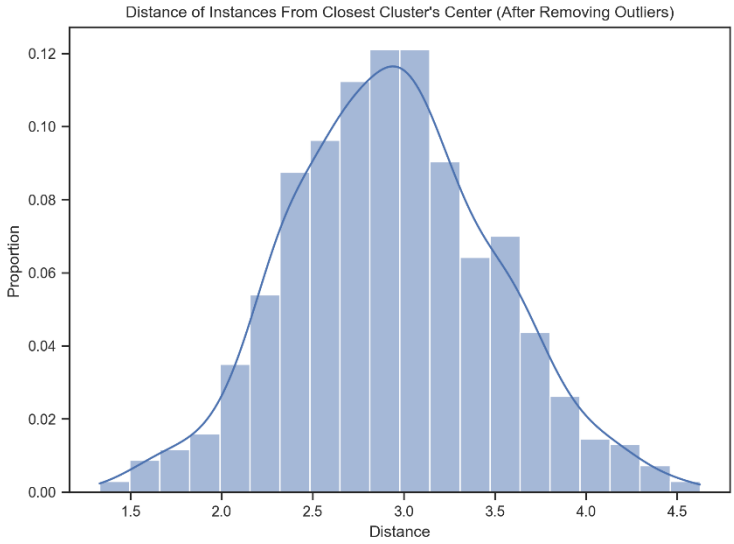


According to the above chart, K=7 looks good. We use the histogram to detect the suitable distance threshold for removing the outliers.



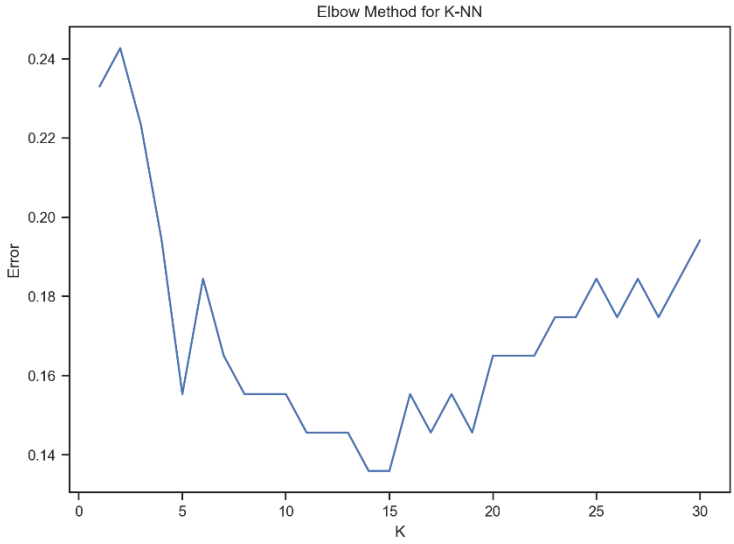
According to the above chart, it seems nice if we remove the instances which have distances higher than 4.75. After removing such instances which constitute less than 1% of the data set, the histogram chart looks better.

```
The proportion of outliers: 0.0072
```



Part b

According to the below chart, K=13 looks good.




```
Results for KNeighborsClassifier:
Accuracy: 0.8544
Precision: 0.9167
Recall: 0.7333
Confusion Matrix:
[[55  3]
 [12 33]]
```

Part c

```
Results for DecisionTreeClassifier:
Accuracy: 0.7748
Precision: 0.7501
Recall: 0.7267
Confusion Matrix:
[[47 11]
 [12 33]]
```

Part d

It took 36 seconds to find the best hyperparameters for Random Forest. Yes, I believe it is worth it because the model selection process is performed only once and leads to exploiting a high-performance model.

```
Best Parameters for Random Forest:
max_depth: 14
n_estimators: 50
Elapsed time (s): 36.4949

Results for RandomForestClassifier:
Accuracy: 0.8155
Precision: 0.7826
Recall: 0.8
Confusion Matrix:
[[48 10]
 [ 9 36]]
```

Part e

```
Results for GaussianNB:
Accuracy: 0.8544
Precision: 0.875
Recall: 0.7778
Confusion Matrix:
[[53  5]
 [10 35]]
```

Part f

```
Results for LogisticRegression:
Accuracy: 0.8447
Precision: 0.8085
Recall: 0.8444
Confusion Matrix:
[[49  9]
 [ 7 38]]
```

Part g

```
Best Parameters for SVC:
C: 0.1
kernel: rbf

Results for SVC:
Accuracy: 0.8544
Precision: 0.8571
Recall: 0.8
Confusion Matrix:
[[52  6]
 [ 9 36]]
```

Part h

```
Results for AdaBoostClassifier:
Accuracy: 0.8155
Precision: 0.7708
Recall: 0.8222
Confusion Matrix:
[[47 11]
 [ 8 37]]
```

Part i

```
Results for KMeans:
Accuracy: 0.8252
Precision: 0.9091
Recall: 0.6667
Confusion Matrix:
[[55  3]
 [15 30]]
```

Part j

KNN, GNB, and SVM have the highest Accuracy. KNN also has the highest Precision. LR has the highest Recall. SVM and LR have the highest F1 scores.

I believe that the most important metric for this application is F1 Score because we should have a balance between both FP and FN and try to lower them down as much as possible. If we have incorrect approvals, the bank will lose its capital and if we incorrectly disapprove, the bank will lose its customers. So, I think the best model is LR for this metric because compared with SVM it has much lower training and predicting times.

Model	FN	FP	Accuracy	Precision	Recall	F1 Score	Training time	Predicting time
KNeighborsClassifier	12.00	3.00	0.8544	0.9167	0.7333	0.8148	1.9017	4.8978
DecisionTreeClassifier	12.10	11.25	0.7733	0.7455	0.7311	0.7381	2.0504	0.0502
RandomForestClassifier	8.80	8.50	0.8320	0.8105	0.8044	0.8072	74.3101	5.9965
GaussianNB	10.00	5.00	0.8544	0.8750	0.7778	0.8235	0.7995	0.2004
LogisticRegression	7.00	9.00	0.8447	0.8085	0.8444	0.8261	2.7983	0.2000
SVC	9.00	6.00	0.8544	0.8571	0.8000	0.8276	74.4551	4.7970
AdaBoostClassifier	8.00	11.00	0.8155	0.7708	0.8222	0.7957	100.8447	8.3441
KMeans	23.25	31.60	0.4675	0.5269	0.4833	0.4896	35.5294	1.8491

Part k

We simply changed the positive class threshold from 0.5 to 0.56 and it improved the F1 score by about 2%. The model also got the highest Accuracy compared with the above models.

```
Model: LogisticRegression
FN: 7.0
FP: 7.0
Accuracy: 0.8641
Precision: 0.8444
Recall: 0.8444
F1 Score: 0.8444
Training time: 2.8037
Predicting time: 0.2491
```