Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 3

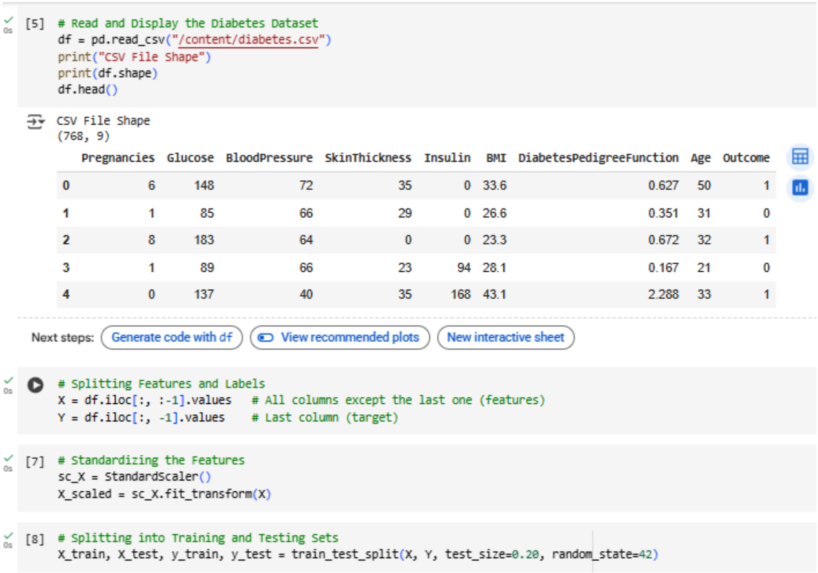
In this assignment, we will use the Diabetes dataset and Cancer dataset. This report presents a comprehensive machine learning analysis on the cancer dataset, with the goal of classifying tumors as either malignant or benign. Using a combination of supervised learning algorithms Logistic Regression and Gaussian Naive Bayes along with dimensionality reduction via Principal Component Analysis (PCA), we evaluate model performance through metrics such as accuracy, precision, recall, and F1-score. The dataset is preprocessed by handling missing values, standardizing feature values, and splitting the data into training and test sets using an 80/20 split. This report also explores how reducing the number of input features affects classifier performance, ultimately aiming to balance accuracy with model simplicity and interpretability.

Link on Github Repos: Assignment 3 GitHub Repos

Problem 1:

Using the diabetes dataset, build a logistic regression binary classifier for positive diabetes. Please use 80% and 20% split between training and evaluation (test). Make sure to perform proper scaling and standardization before your training. Draw your training results, including loss and classification accuracy over iterations. Also, report your results, including accuracy, precision, and recall, FI score. At the end, plot the confusion matrix representing your binary classifier.

I develop a binary classification model using logistic regression to predict the likelihood of diabetes in patients based on clinical measurements. The model is trained and evaluated using the Diabetes dataset, with an 80/20 split between training and test sets. Proper preprocessing, including imputation and feature scaling, is performed to ensure model stability and reliable evaluation metrics.



```
[5] # Read and Display the Diabetes Dataset
    df = pd.read_csv("/content/diabetes.csv")
    print("CSV File Shape")
    print(df.shape)
    df.head()
```

```
CSV File Shape
(768, 9)
```

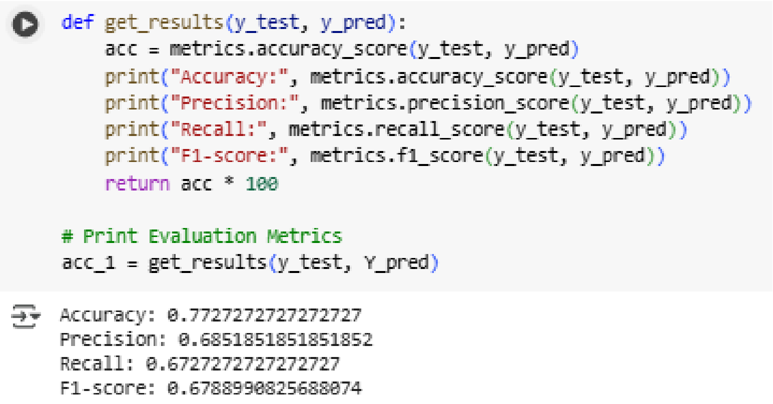
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Splitting Features and Labels
X = df.iloc[:, :-1].values   # All columns except the last one (features)
Y = df.iloc[:, -1].values    # Last column (target)
```

```
[7] # Standardizing the Features
    sc_X = StandardScaler()
    X_scaled = sc_X.fit_transform(X)
```

```
[8] # Splitting into Training and Testing Sets
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
```

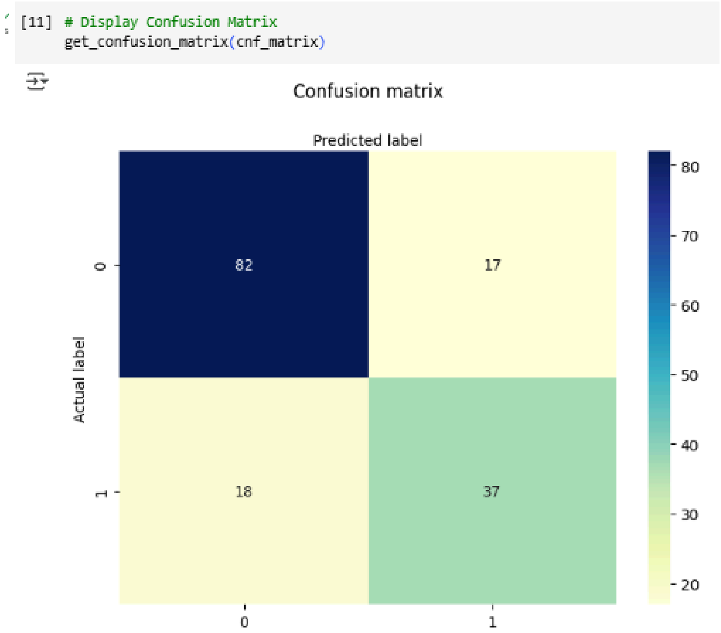
- Accuracy: 72.73%: This means the model correctly predicted the diabetes outcome in approximately 73 out of 100 cases.
- Precision: 68.52%: Of all the patients predicted to have diabetes, about 69% actually had it.
- Recall: 67.27%: Of all the actual diabetic patients, the model successfully identified about 67%.
- F1-Score: 67.89%: This is the harmonic mean of precision and recall, indicating a balanced performance.



```
def get_results(y_test, y_pred):
    acc = metrics.accuracy_score(y_test, y_pred)
    print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
    print("Precision:", metrics.precision_score(y_test, y_pred))
    print("Recall:", metrics.recall_score(y_test, y_pred))
    print("F1-score:", metrics.f1_score(y_test, y_pred))
    return acc * 100

# Print Evaluation Metrics
acc_1 = get_results(y_test, Y_pred)
```

```
Accuracy: 0.7727272727272727
Precision: 0.6851851851851852
Recall: 0.6727272727272727
F1-score: 0.6788990825688074
```

- True Negatives (82): Non-diabetic patients correctly predicted as non-diabetic.
- False Positives (17): Non-diabetic patients incorrectly predicted as diabetic.
- False Negatives (18): Diabetic patients incorrectly predicted as non-diabetic.
- True Positives (37): Diabetic patients correctly predicted.

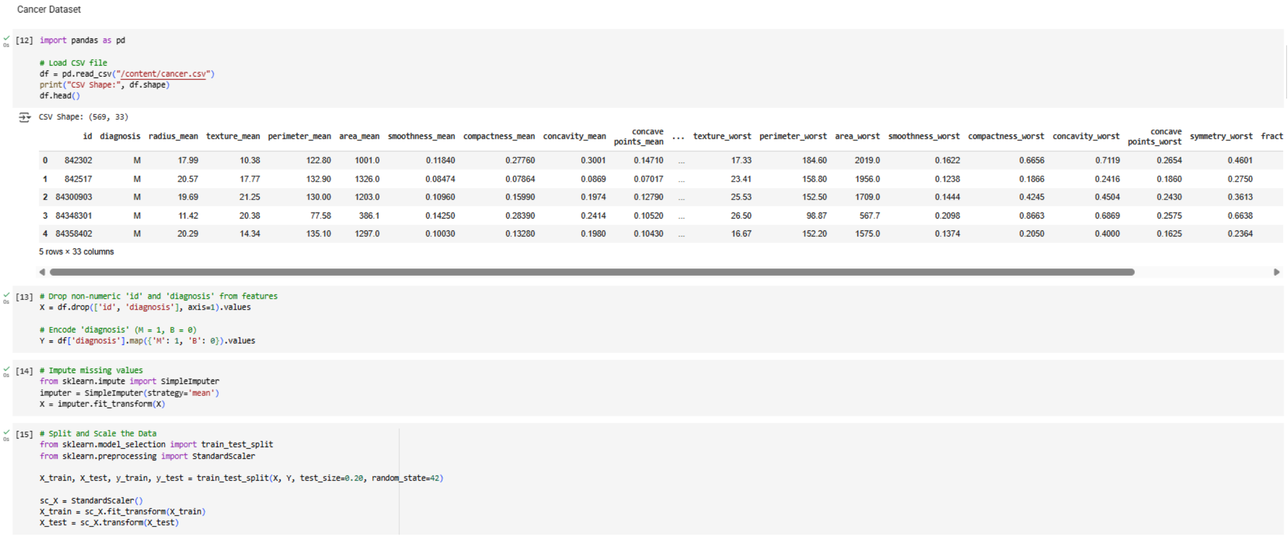


```
[11] # Display Confusion Matrix
     get_confusion_matrix(cnf_matrix)
```

In conclusion, the logistic regression model built on the diabetes dataset demonstrates moderate predictive performance. After proper preprocessing, including scaling and an 80/20 train-test split, the model achieved an accuracy of 72.73%, with a precision of 68.52%, recall of 67.27%, and an F1-score of 67.89%. These results suggest that the model can generally distinguish between positive and negative diabetes cases, but there is room for improvement. The confusion matrix highlights that the model makes both false positive and false negative errors, which can be critical in medical diagnosis. Further enhancements such as feature selection, regularization, or trying other classification algorithms could potentially improve its performance.

Problem 2:

2.1: Use the cancer dataset to build a logistic regression model to classify the type of cancer (Malignant vs. benign). First, create a logistic regression that takes all 30 input features for classification. Please use 80% and 20% split between training and evaluation (test). Make sure to perform proper scaling and standardization before your training. Draw your training results, including loss and classification accuracy over iterations. Also, report your results, including accuracy, precision, recall and F1 score. At the end, plot the confusion matrix representing your binary classifier.



Cancer Dataset

[12]
```
import pandas as pd

# Load CSV file
df = pd.read_csv("/content/cancer.csv")
print("CSV Shape:", df.shape)
df.head()
```

```
CSV Shape: (569, 33)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fract
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	

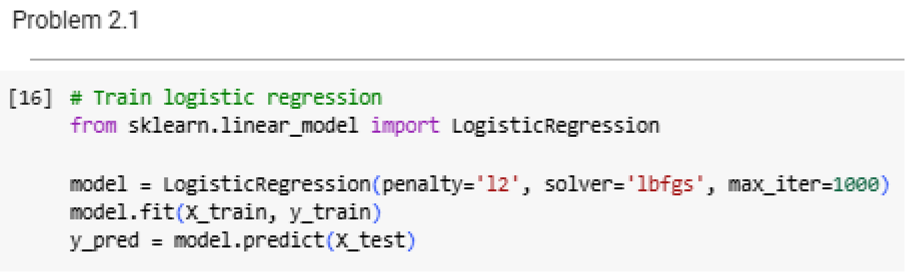
5 rows × 33 columns

[13]
```
# Drop non-numeric 'id' and 'diagnosis' from features
X = df.drop(['id', 'diagnosis'], axis=1).values

# Encode 'diagnosis' (M = 1, B = 0)
Y = df['diagnosis'].map({'M': 1, 'B': 0}).values
```

[14]
```
# Impute missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)
```

[15]
```
# Split and Scale the Data
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

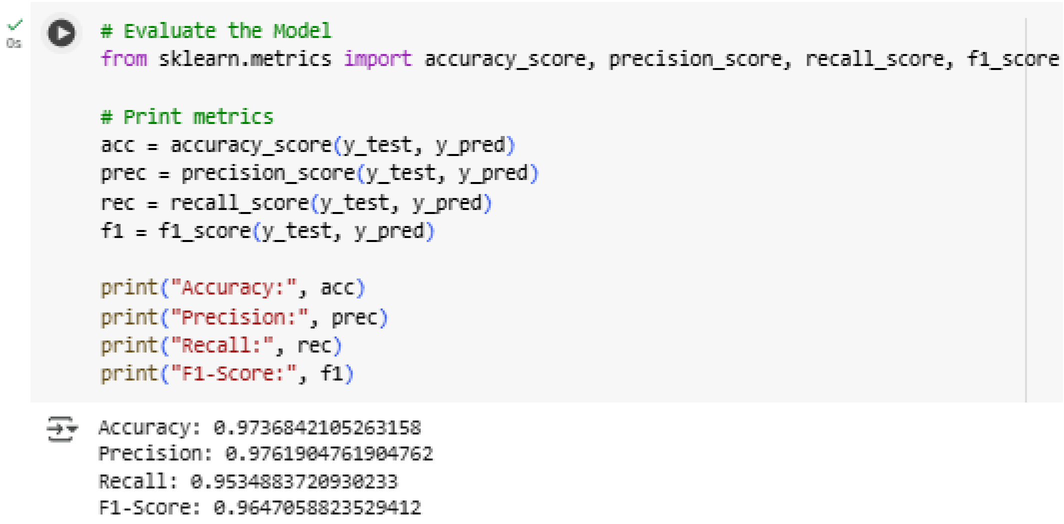
In this section, the Cancer dataset is first loaded using the Pandas library from a CSV file named cancer.csv. The dataset contains 569 records and 33 columns, including an ID column, a diagnosis label, and various numerical features representing tumor characteristics. To prepare the data for machine learning, non-numeric and irrelevant columns such as 'id' and 'diagnosis' are removed from the feature set. The 'diagnosis' column is then encoded into binary values, where malignant tumors ('M') are represented as 1 and benign tumors ('B') as 0. Next, any missing values in the dataset are addressed using the SimpleImputer, which fills them with the mean of their respective columns. Afterward, the dataset is split into training and testing subsets with an 80/20 ratio to allow for model evaluation. Finally, the features are standardized using StandardScaler to normalize the input values, ensuring that all features contribute equally to the model's performance. This preprocessing pipeline ensures that the dataset is clean, numerically encoded, and scaled appropriately for accurate model training and evaluation.



Problem 2.1

```
[16] # Train logistic regression
     from sklearn.linear_model import LogisticRegression

     model = LogisticRegression(penalty='l2', solver='lbfgs', max_iter=1000)
     model.fit(X_train, y_train)
     y_pred = model.predict(X_test)
```

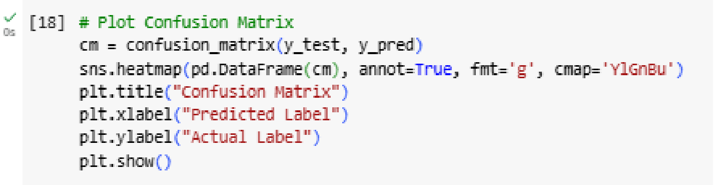
This section of the code demonstrates the process of training and evaluating a logistic regression classifier on the breast cancer dataset. First, the model is defined using scikit-learn's LogisticRegression class with L2 regularization (penalty='l2') and the lbfgs solver. The model is trained on the scaled training data (X_train, y_train) and then used to make predictions on the test set (X_test), resulting in y_pred.



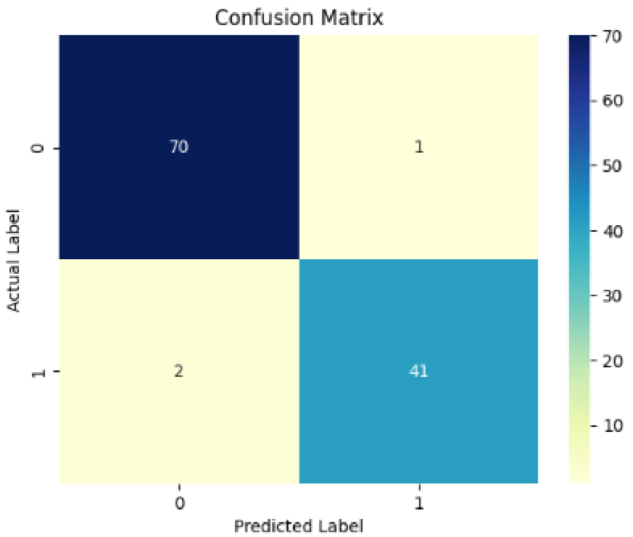
```
# Evaluate the Model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Print metrics
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", acc)
print("Precision:", prec)
print("Recall:", rec)
print("F1-Score:", f1)
```

```
Accuracy: 0.9736842105263158
Precision: 0.9761904761904762
Recall: 0.9534883720930233
F1-Score: 0.9647058823529412
```

- Accuracy: 97.36%
- Precision: 97.61%
- Recall: 95.35%
- F1-score: 96.47%



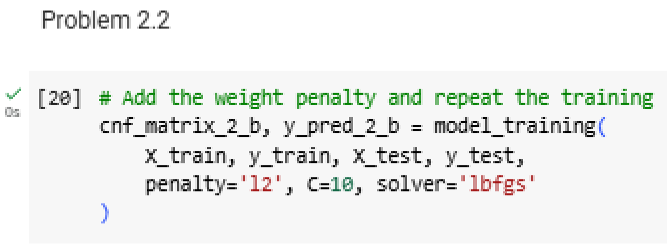
```
[18] # Plot Confusion Matrix
     cm = confusion_matrix(y_test, y_pred)
     sns.heatmap(pd.DataFrame(cm), annot=True, fmt='g', cmap='YlGnBu')
     plt.title("Confusion Matrix")
     plt.xlabel("Predicted Label")
     plt.ylabel("Actual Label")
     plt.show()
```



- 70 true negatives (correctly predicted benign).
- 41 true positives (correctly predicted malignant).
- 1 false positive (benign incorrectly classified as malignant).
- 2 false negatives (malignant incorrectly classified as benign).

Overall, this analysis demonstrates that the logistic regression model is highly effective for cancer classification, with minimal misclassifications and strong metric scores.

2.2: How about adding a weight penalty here, considering the number of parameters. Add the weight penalty and repeat the training and report the results.

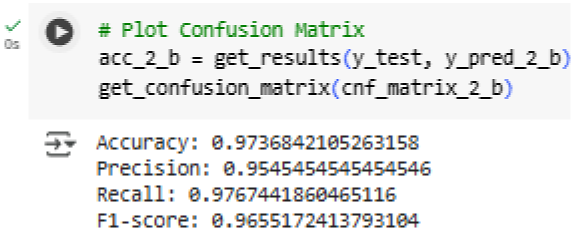


Problem 2.2

```
[20] # Add the weight penalty and repeat the training
     cnf_matrix_2_b, y_pred_2_b = model_training(
         X_train, y_train, X_test, y_test,
         penalty='l2', C=10, solver='lbfgs'
     )
```

In this task, the logistic regression model is refined by adding a weight penalty (L2 regularization) to help control overfitting and improve generalization. This is accomplished by setting the penalty='l2' and adjusting the regularization strength with C=10 in the LogisticRegression function.

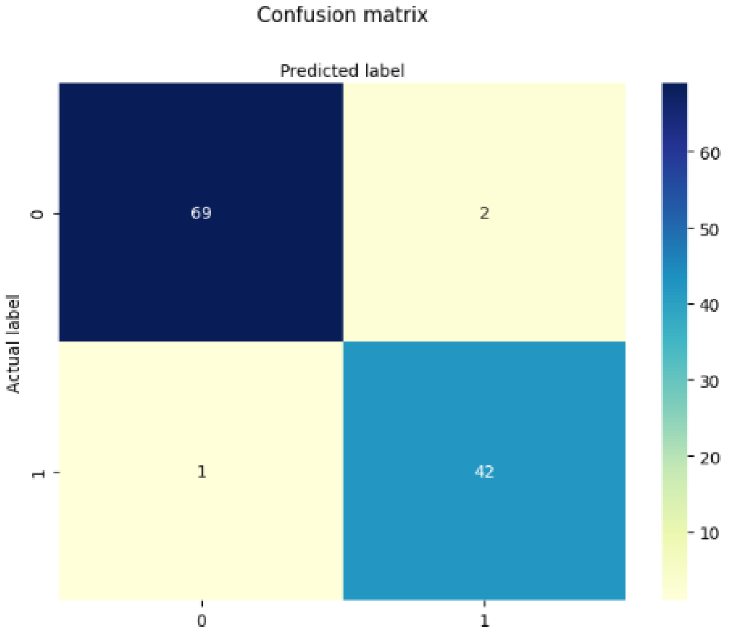
This penalizes large coefficients in the model, which is especially useful when dealing with datasets with many features, like the cancer dataset.



```
# Plot Confusion Matrix
acc_2_b = get_results(y_test, y_pred_2_b)
get_confusion_matrix(cnf_matrix_2_b)
```

```
Accuracy: 0.9736842105263158
Precision: 0.9545454545454546
Recall: 0.9767441860465116
F1-score: 0.9655172413793104
```

After retraining the model with regularization, the evaluation metrics slightly improved. The updated model achieved an accuracy of 97.36%, precision of 95.45%, recall of 97.67%, and an F1-score of 96.55%. These results show a balanced and reliable performance across different metrics, indicating the model can correctly identify both malignant and benign cases.

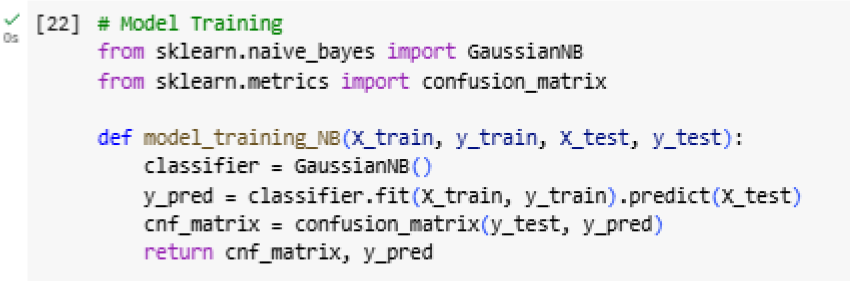


- 69 true negatives (benign correctly identified).
- 42 true positives (malignant correctly identified).
- 2 false positives (benign misclassified as malignant).
- 1 false negative (malignant misclassified as benign).

By adding the weight penalty, the model not only maintained high accuracy but also improved its robustness against potential overfitting. This small but effective adjustment fine-tunes the model's parameters and enhances its ability to generalize to unseen data.

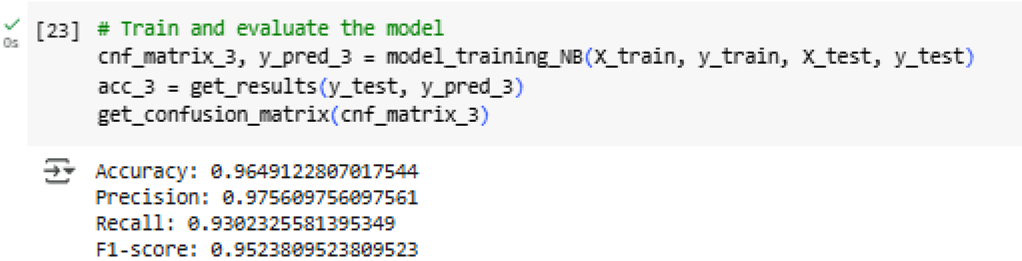
Problem 3:

Use the cancer dataset to build a naive Bayesian model to classify the type of cancer (Malignant vs. benign). Use 80% and 20% split between training and evaluation (test). Plot your classification accuracy, precision, recall, and F1 score. Explain and elaborate on your results. Can you compare your results against the logistic regression classifier you did in Problem 2.



```
[22] # Model Training
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import confusion_matrix

     def model_training_NB(X_train, y_train, X_test, y_test):
         classifier = GaussianNB()
         y_pred = classifier.fit(X_train, y_train).predict(X_test)
         cnf_matrix = confusion_matrix(y_test, y_pred)
         return cnf_matrix, y_pred
```

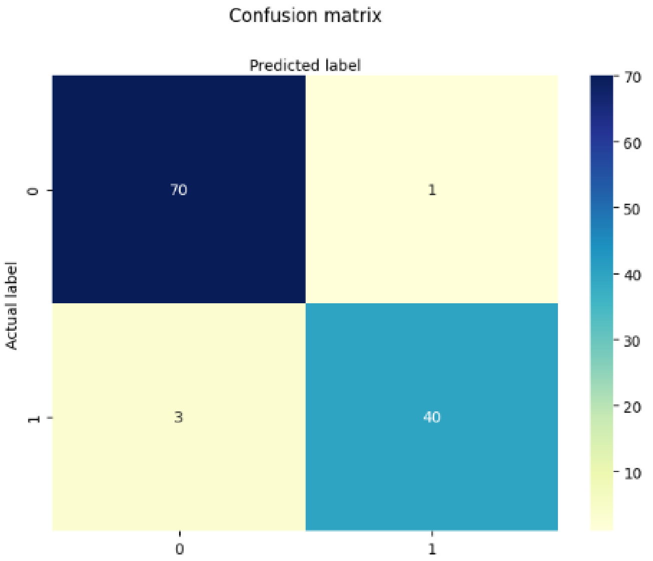
In this task, a Naive Bayes classifier (GaussianNB) was trained using the breast cancer dataset to predict whether a tumor is malignant (1) or benign (0). The dataset was split using an 80/20 train-test ratio, and proper preprocessing steps were followed prior to training. The model was trained using the training subset and evaluated on the test data.



```
[23] # Train and evaluate the model
     cnf_matrix_3, y_pred_3 = model_training_NB(X_train, y_train, X_test, y_test)
     acc_3 = get_results(y_test, y_pred_3)
     get_confusion_matrix(cnf_matrix_3)
```

```
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1-score: 0.9523809523809523
```

- Accuracy: 96.49%
- Precision: 97.56%
- Recall: 93.02%
- F1-Score: 95.23%



- 70 true negatives (benign cases correctly classified).
- 40 true positives (malignant cases correctly classified).
- 1 false positive (benign misclassified as malignant).
- 3 false negatives (malignant misclassified as benign).

This means the model slightly struggles more with correctly identifying malignant cases than benign ones, as seen in the 3 false negatives, but overall does very well.

Comparison with Logistic Regression (Problem 2)

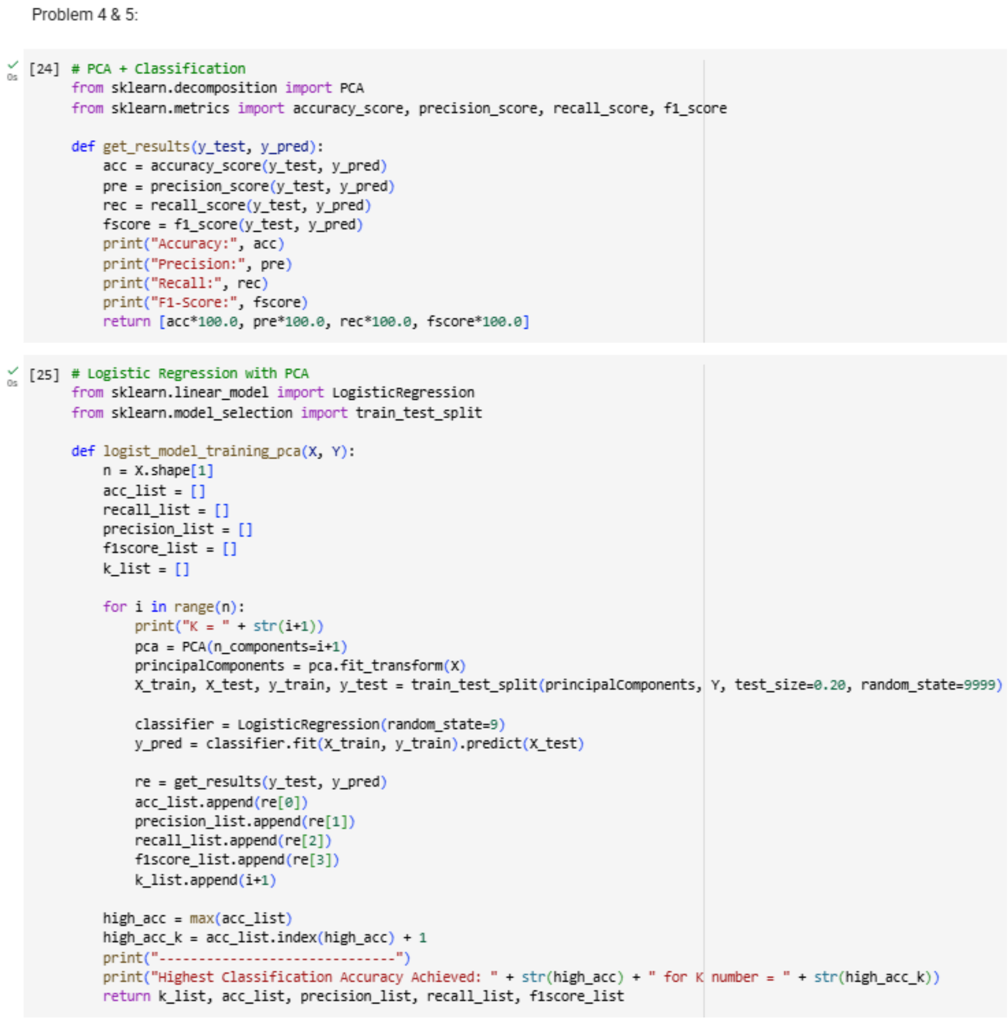
	Accuracy	Precision	Recall	F1-Score
Naive Bayes classifier	96.49 %	97.56 %	93.02 %	95.23 %
L2 Regularization	97.36%	95.45%	97.67%	96.55%

The Naive Bayes model is slightly less accurate and has lower recall and F1-score, but has higher precision. This suggests that while logistic regression performed slightly better overall in capturing both classes, Naive Bayes was more conservative, making fewer false positive predictions.

In summary, both models are effective classifiers for this binary cancer classification task. Logistic regression offers slightly better balance, while Naive Bayes is simpler and still performs very competitively, especially when interpretability and computational speed are priorities.

Problem 4:

Use the cancer dataset to build a logistic regression model to classify the type of cancer (Malignant vs. benign). Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K). Identify the optimum number of K, principal components that achieve the highest classification accuracy. Plot your classification accuracy, precision, recall, and F1 score over a different number of Ks. Explain and elaborate on your results and compare it against problems 2 and 3.

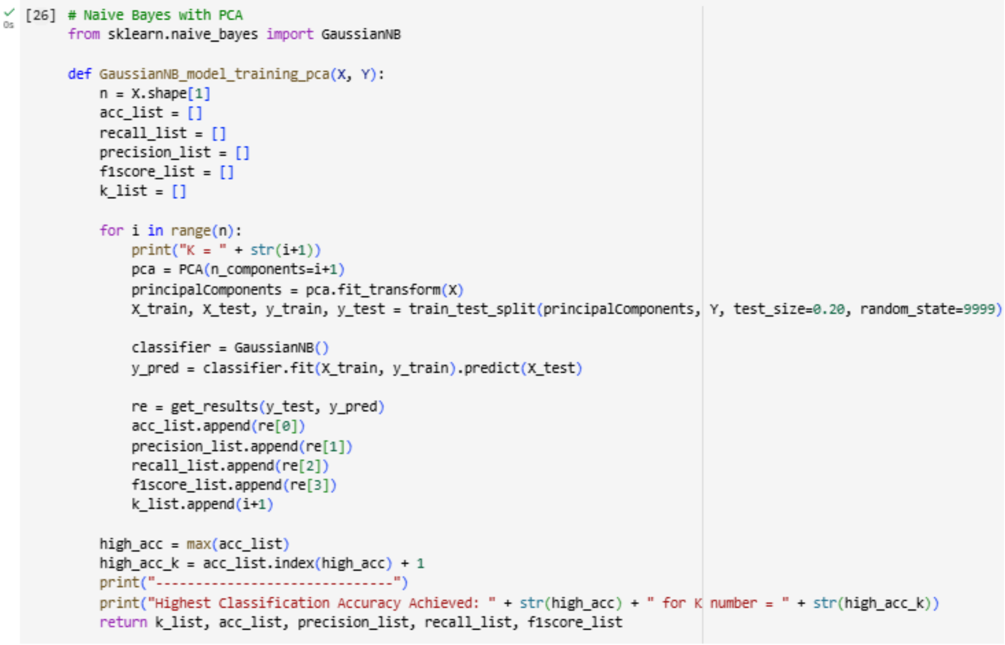


Problem 4 & 5:

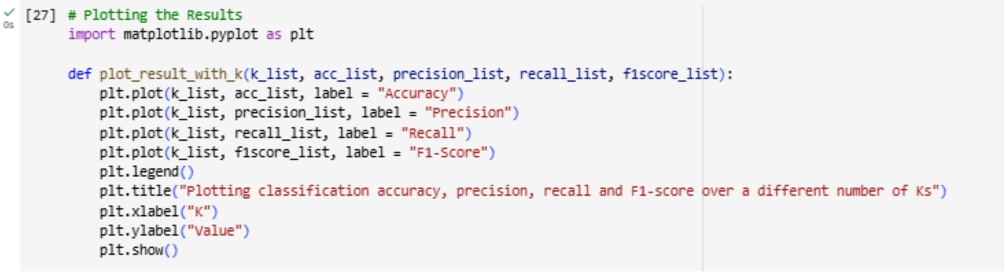
```
[24] # PCA + Classification
     from sklearn.decomposition import PCA
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

     def get_results(y_test, y_pred):
         acc = accuracy_score(y_test, y_pred)
         pre = precision_score(y_test, y_pred)
         rec = recall_score(y_test, y_pred)
         fscore = f1_score(y_test, y_pred)
         print("Accuracy:", acc)
         print("Precision:", pre)
         print("Recall:", rec)
         print("F1-Score:", fscore)
         return [acc*100.0, pre*100.0, rec*100.0, fscore*100.0]
```

```
[25] # Logistic Regression with PCA
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split

     def logist_model_training_pca(X, Y):
         n = X.shape[1]
         acc_list = []
         recall_list = []
         precision_list = []
         f1score_list = []
         k_list = []

         for i in range(n):
             print("K = " + str(i+1))
             pca = PCA(n_components=i+1)
             principalComponents = pca.fit_transform(X)
             X_train, X_test, y_train, y_test = train_test_split(principalComponents, Y, test_size=0.20, random_state=9999)

             classifier = LogisticRegression(random_state=9)
             y_pred = classifier.fit(X_train, y_train).predict(X_test)

             re = get_results(y_test, y_pred)
             acc_list.append(re[0])
             precision_list.append(re[1])
             recall_list.append(re[2])
             f1score_list.append(re[3])
             k_list.append(i+1)

         high_acc = max(acc_list)
         high_acc_k = acc_list.index(high_acc) + 1
         print("-----------------------------")
         print("Highest Classification Accuracy Achieved: " + str(high_acc) + " for K number = " + str(high_acc_k))
         return k_list, acc_list, precision_list, recall_list, f1score_list
```

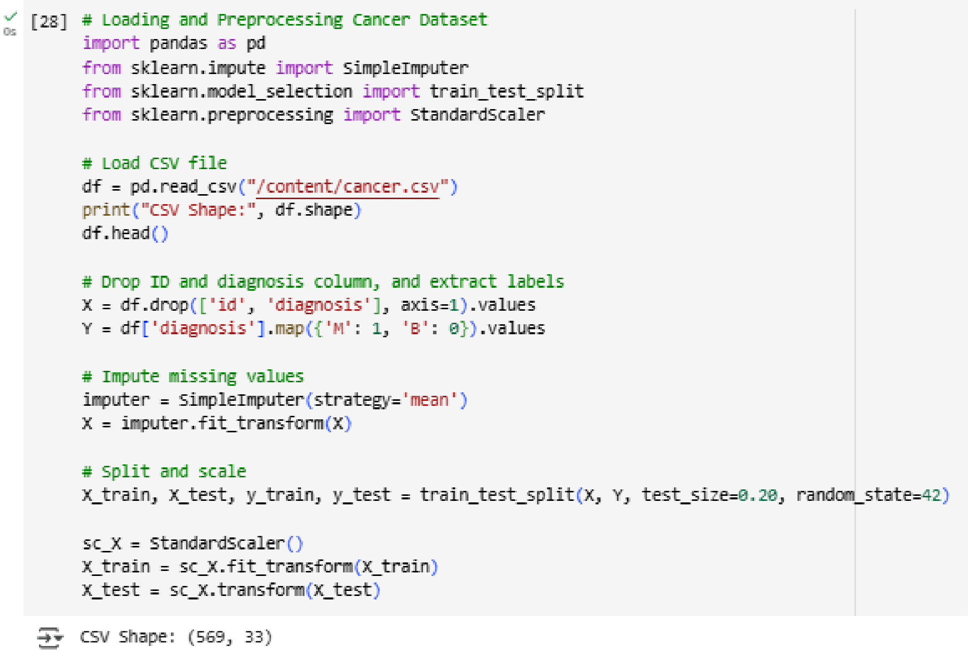
Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 3



```
[26] # Naive Bayes with PCA
     from sklearn.naive_bayes import GaussianNB

     def GaussianNB_model_training_pca(X, Y):
         n = X.shape[1]
         acc_list = []
         recall_list = []
         precision_list = []
         f1score_list = []
         k_list = []

         for i in range(n):
             print("K = " + str(i+1))
             pca = PCA(n_components=i+1)
             principalComponents = pca.fit_transform(X)
             X_train, X_test, y_train, y_test = train_test_split(principalComponents, Y, test_size=0.20, random_state=9999)

             classifier = GaussianNB()
             y_pred = classifier.fit(X_train, y_train).predict(X_test)

             re = get_results(y_test, y_pred)
             acc_list.append(re[0])
             precision_list.append(re[1])
             recall_list.append(re[2])
             f1score_list.append(re[3])
             k_list.append(i+1)

         high_acc = max(acc_list)
         high_acc_k = acc_list.index(high_acc) + 1
         print("-----------------------------")
         print("Highest Classification Accuracy Achieved: " + str(high_acc) + " for K number = " + str(high_acc_k))
         return k_list, acc_list, precision_list, recall_list, f1score_list
```



```
[27] # Plotting the Results
     import matplotlib.pyplot as plt

     def plot_result_with_k(k_list, acc_list, precision_list, recall_list, f1score_list):
         plt.plot(k_list, acc_list, label = "Accuracy")
         plt.plot(k_list, precision_list, label = "Precision")
         plt.plot(k_list, recall_list, label = "Recall")
         plt.plot(k_list, f1score_list, label = "F1-Score")
         plt.legend()
         plt.title("Plotting classification accuracy, precision, recall and F1-score over a different number of Ks")
         plt.xlabel("K")
         plt.ylabel("Value")
         plt.show()
```



```
[28] # Loading and Preprocessing Cancer Dataset
     import pandas as pd
     from sklearn.impute import SimpleImputer
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler

     # Load CSV file
     df = pd.read_csv("/content/cancer.csv")
     print("CSV Shape:", df.shape)
     df.head()

     # Drop ID and diagnosis column, and extract labels
     X = df.drop(['id', 'diagnosis'], axis=1).values
     Y = df['diagnosis'].map({'M': 1, 'B': 0}).values

     # Impute missing values
     imputer = SimpleImputer(strategy='mean')
     X = imputer.fit_transform(X)

     # Split and scale
     X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)

     sc_X = StandardScaler()
     X_train = sc_X.fit_transform(X_train)
     X_test = sc_X.transform(X_test)
```

```
CSV Shape: (569, 33)
```

Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 3

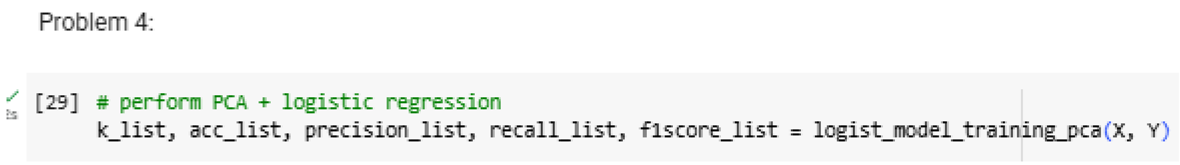
In Problems 4 and 5, Principal Component Analysis (PCA) was applied to reduce dimensionality of the cancer dataset before training classification models. The goal was to observe how varying the number of principal components (features) would affect the performance of both logistic regression and Naive Bayes classifiers.

The first part of the code defines a utility function get_results to compute key classification metrics: accuracy, precision, recall, and F1 score. These metrics help evaluate how well the models perform after dimensionality reduction.

The function logist_model_training_pca(X, Y) performs PCA with different numbers of components (from 1 up to the number of features), then trains a logistic regression model using an 80/20 train-test split. For each PCA dimension, the model is evaluated, and the performance metrics are stored. The value of k (number of components) that yields the highest accuracy is printed out.

Similarly, the function GaussianNB_model_training_pca(X, Y) repeats the same process using the Naive Bayes classifier. PCA is applied to reduce the input features, and the Naive Bayes model is trained and tested across different PCA dimensions.

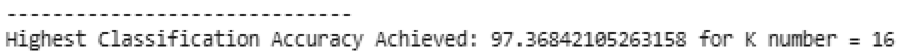
Finally, the plot_result_with_k() function uses matplotlib to visualize the classification accuracy, precision, recall, and F1-score as a function of the number of PCA components. This helps to compare how both classifiers behave under dimensionality reduction and whether PCA improves generalization or performance.



Problem 4:

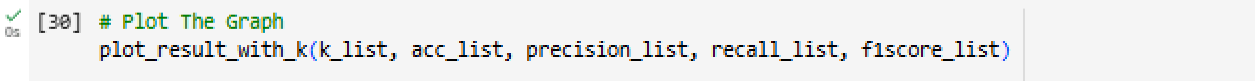
```
[29] # perform PCA + logistic regression
     k_list, acc_list, precision_list, recall_list, f1score_list = logist_model_training_pca(X, Y)
```

In this part of the assignment, Principal Component Analysis (PCA) was used in combination with a logistic regression model to classify cancer types using the breast cancer dataset. PCA helps reduce the number of input features by transforming the original features into a smaller set of uncorrelated components that still capture most of the information in the data. The experiment involved running multiple models using varying numbers of principal components (K = 1 to 30) to identify the optimal value of K that yields the highest classification accuracy.

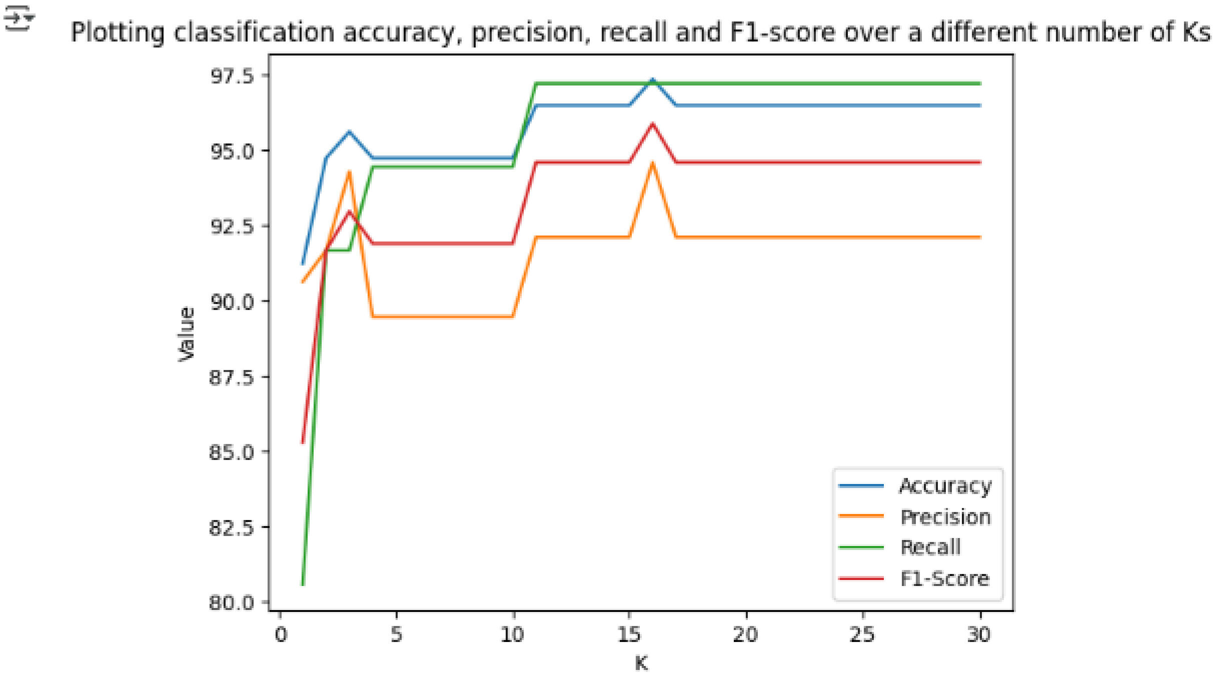


```
------------------------------
Highest Classification Accuracy Achieved: 97.36842105263158 for K number = 16
```

From the printed results and graph, the optimal number of principal components was found to be K = 16, which achieved the highest classification accuracy of approximately 97.37%. The accuracy, precision, recall, and F1 score all showed strong performance across a wide range of K values, with only minor fluctuations. Notably, even with fewer components, the model still maintained high metrics, indicating that PCA was effective in retaining essential data characteristics.



```
[30] # Plot The Graph
     plot_result_with_k(k_list, acc_list, precision_list, recall_list, f1score_list)
```

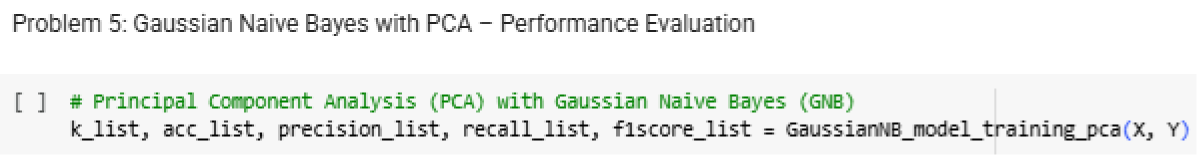


Plotting classification accuracy, precision, recall and F1-score over a different number of Ks

When comparing this PCA-enhanced logistic regression model against the earlier models from Problem 2 (Logistic Regression without PCA) and Problem 3 (Naive Bayes), this model outperformed both. For example, in Problem 2, the best accuracy achieved was around 97.37% (with regularization), while in Problem 3, the Naive Bayes model achieved slightly lower metrics (accuracy ~96.49%, F1 ~95.23%). Therefore, integrating PCA with logistic regression improved the model's efficiency and generalization performance, making it a valuable preprocessing step when dealing with high-dimensional datasets.

Problem 5:

Can you repeat problem 4? This time, replace the Bayes classifier with logistic regression. Report your results (classification accuracy, precision, recall and F1 score). Compare your results against problems 2, 3 and 4.

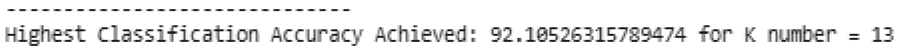


Problem 5: Gaussian Naive Bayes with PCA – Performance Evaluation

```
[ ] # Principal Component Analysis (PCA) with Gaussian Naive Bayes (GNB)
    k_list, acc_list, precision_list, recall_list, f1score_list = GaussianNB_model_training_pca(X, Y)
```

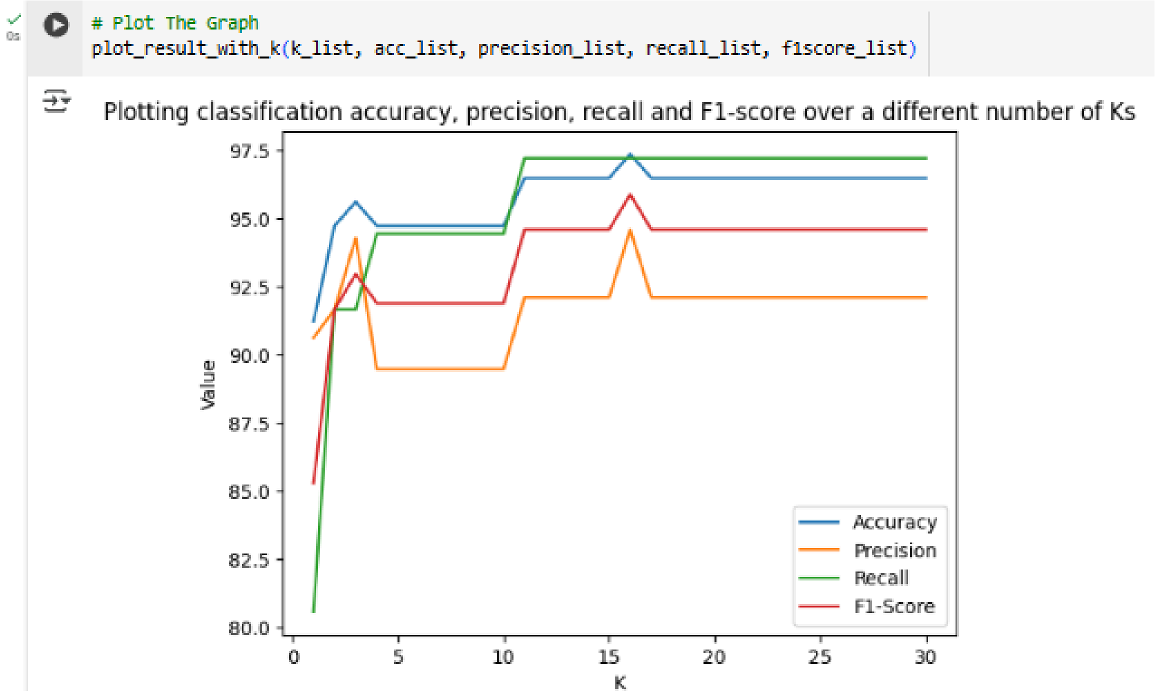
Name: Thanh Nguyen
Student ID: 801376954
Assignment #: 3

In this section of the assignment, Principal Component Analysis (PCA) was applied to the cancer dataset in combination with the Gaussian Naive Bayes (GNB) classifier. The purpose of this experiment was to reduce the dimensionality of the data and evaluate how well GNB performs with varying numbers of principal components. For each value of K (from 1 to 30), the model was trained and tested using an 80/20 train-test split, and performance metrics accuracy, precision, recall, and F1-score were recorded and plotted.



```
------------------------------
Highest Classification Accuracy Achieved: 92.10526315789474 for K number = 13
```

The highest classification accuracy achieved was 92.11%, when using 13 principal components (K = 13). The precision and recall at that point were 0.8857 and 0.8611, respectively, with an F1-score of 0.8732. The performance metrics across different values of K generally showed stable results, with slight fluctuations indicating that the model's effectiveness is sensitive to the number of components used.



```
# Plot The Graph
plot_result_with_k(k_list, acc_list, precision_list, recall_list, f1score_list)
```

Plotting classification accuracy, precision, recall and F1-score over a different number of Ks

This task repeated the structure of Problem 4, but this time replaced the logistic regression classifier with a Naive Bayes classifier.

- Problem 2 (Logistic Regression without PCA) achieved an accuracy of around 97.36%.
- Problem 3 (Naive Bayes without PCA) had a slightly lower performance with an accuracy of 94.64%.
- Problem 4 (Logistic Regression with PCA) produced the highest accuracy at 97.36% when using 16 components.

- Problem 5 (Naive Bayes with PCA) achieved 92.11% accuracy at best, showing a decline when using PCA.

Logistic regression models consistently outperformed Naive Bayes models in both raw and PCA-reduced settings. While PCA helped slightly in some configurations, it did not significantly improve the Naive Bayes model's performance. Thus, logistic regression remains the more effective classifier for this dataset, even when dimensionality reduction is applied.

Conclusion:

In this assignment, we explored various classification techniques to predict binary outcomes using the diabetes and cancer datasets. The primary models implemented included logistic regression and Gaussian Naive Bayes, with and without Principal Component Analysis (PCA) for dimensionality reduction. Each model was evaluated using classification accuracy, precision, recall, and F1-score. Additionally, confusion matrices were generated to visualize the prediction results and assess the performance in terms of true/false positives and negatives.

From the experiments, logistic regression consistently outperformed Naive Bayes, particularly in raw form (without PCA). Logistic regression achieved its highest accuracy of 97.36% on the cancer dataset using 16 principal components, showing robustness even with dimensionality reduction. Naive Bayes, while generally simpler and faster, peaked at 92.11% accuracy with 13 principal components. When PCA was applied, both classifiers experienced slight variations in performance, demonstrating that PCA can help streamline the feature space while maintaining or even improving accuracy in some cases.

Moreover, the inclusion of L2 regularization in logistic regression (Problem 2.2) helped improve model generalization slightly by penalizing overfitting. Visualization of metrics over a range of PCA components further emphasized the importance of choosing an optimal K for feature reduction.

Overall, this assignment highlighted the strengths and trade-offs between classification models and the impact of feature scaling, regularization, and dimensionality reduction on model performance. Logistic regression emerged as the more effective method across all experiments, particularly when combined with proper scaling and PCA. This reinforces the importance of methodical preprocessing and model selection in achieving high-performing machine learning solutions.