Consolidated Report on Assignment-2

Prepared for
Dr N.L. Bhanu Murthy
Department of computer science and information systems

Prepared by

Kota venkata bharghav	**2020B2A82088G**
Rohit Akash R	**2020B1A32073G**
Sahil Sudesh	**2020B4A31868G**

Abstract

This report focuses on the analysis of the census-income dataset, which contains census information for individuals and their income classification. The dataset contains information for 48,842 individuals, with 14 attributes including age, workclass, education, marital status, and more. The target variable is a binary classification of income, categorized as ">50K" or "<=50K". The report begins by addressing missing values in the dataset using appropriate techniques. This involves identifying missing values and implementing strategies such as imputation or deletion, feature extraction, feature scaling etc.

Three different classifiers are then applied to build prediction models: Naïve Bayes Classifier, Logistic Regression, and Neural Network classifiers with 1, 2, and 3 hidden layers. Each model is trained and tested using a combined dataset, with 67% randomly selected as the training set and the remaining points used for testing. The comparative study section provides insights into the performance of the different classifiers, including the Decision Tree Classifier from Assignment 1 and the Random Forest algorithm. The optimal classifier is determined based on its performance metrics, such as accuracy, precision, recall, and F1-score, etc.

This report demonstrates the process of building prediction models using various classifiers. The findings shed light on the effectiveness of each classifier and identify the optimal classifier for predicting income levels based on census data.

Data preprocessing

- The data is in CSV format with 14 features namely age ,workclass, fnlwgt ,education ,education_num ,marital_status ,occupation ,relationship ,race,sex ,capital_gain ,capital_loss ,hours_per_week,native_country.

- The target attribute is income.

workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	low
Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	low
Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	high
Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	high
?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	low
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Private	215419	Bachelors	13	Divorced	Prof-specialty	Not-in-family	White	Female	0	0	36	United-States	low
?	321403	HS-grad	9	Widowed	?	Other-relative	Black	Male	0	0	40	United-States	low
Private	374983	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	50	United-States	low
Private	83891	Bachelors	13	Divorced	Adm-clerical	Own-child	Asian-Pac-Islander	Male	5455	0	40	United-States	low
Self-emp-inc	182148	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	60	United-States	high

- The data had several missing values in the form of “?”.

- The missing values were in the columns native_country, occupation and workclass respectively.

- In order to fill in the missing values in native country, we grouped them by 'native_country' and 'income', and calculate the mean for 'age' and 'work hours'.

  Then we tried analyzing the relationship between each country’s high and low income groups with respect to mean age and mean work hours.

```
# Group data by 'work class','country' and 'class', and calculate the mean for 'age' and 'work hours'
grouped_data = training.groupby(['workclass', 'income']).agg({'age': 'mean', 'hours_per_week': 'mean'}).reset_index()
sorted_grouped_data = grouped_data.sort_values(by=["hours_per_week"],ascending=False)

# Print the grouped data
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    display(sorted_grouped_data)

#Below List Sorted by Hours worked per week
```

	native_country	income	age	hours_per_week
31	Honduras	high	47.000000	60.000000
72	Thailand	high	32.666667	58.333333
68	South	high	44.875000	51.437500
20	France	high	40.166667	50.750000
24	Greece	high	50.250000	50.625000
35	Hungary	high	57.333333	50.000000
8	Columbia	high	53.500000	50.000000
80	Yugoslavia	high	40.166667	49.500000
14	Ecuador	high	47.500000	48.750000

	native_country	income	age	hours_per_week
31	Honduras	high	47.000000	60.000000
72	Thailand	high	32.666667	58.333333
68	South	high	44.875000	51.437500
20	France	high	40.166667	50.750000
24	Greece	high	50.250000	50.625000
35	Hungary	high	57.333333	50.000000
8	Columbia	high	53.500000	50.000000
80	Yugoslavia	high	40.166667	49.500000
14	Ecuador	high	47.500000	48.750000

We made the following observations :

1. The Age range of “?” high income individuals ranges between Italy low income group and USA high income group.
2. The Age range of “?” low income individuals ranges between South low income group and Germany low income group.
3. The work hour range of “?” high income individuals ranges between Canada high income group and USA high income group.
4. The work hour range of “?” low income individuals ranges between South low income group and Laos low income group.

“?” is the unknown country

So, we replace “?” low income group with South and “?” high income with United-States

```
training_url = 'https://raw.githubusercontent.com/RR979/Assignment1a_dt/main/assignment1_data_main/census_income_data.csv'
test_url = 'https://raw.githubusercontent.com/RR979/Assignment1a_dt/main/assignment1_data_main/census_income_test.csv'
training = pd.read_csv(training_url,skipinitialspace = True)
test = pd.read_csv(test_url,skipinitialspace = True)
```

```
training.loc[((training.native_country=="?")&(training.income=="high")),"native_country"] = "United-States"
training.loc[((training.native_country=="?")&(training.income=="low")),"native_country"] = "South"
```

- Now for the missing values in the workclass and occupation columns, we analyzed the columns and got the following observations

```
training.workclass.describe()

count       32561
unique          9
top       Private
freq        22696
Name: workclass, dtype: object
```

```
[128] training.workclass.value_counts()

Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: workclass, dtype: int64
```

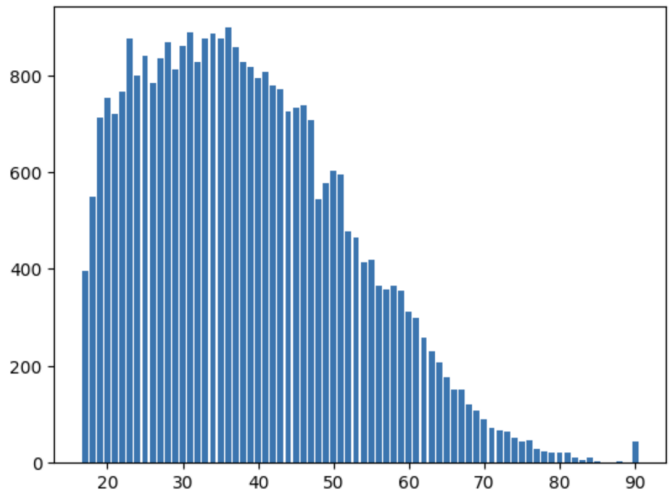
```
test.workclass.describe()

count       16281
unique          9
top       Private
freq        11210
Name: workclass, dtype: object
```

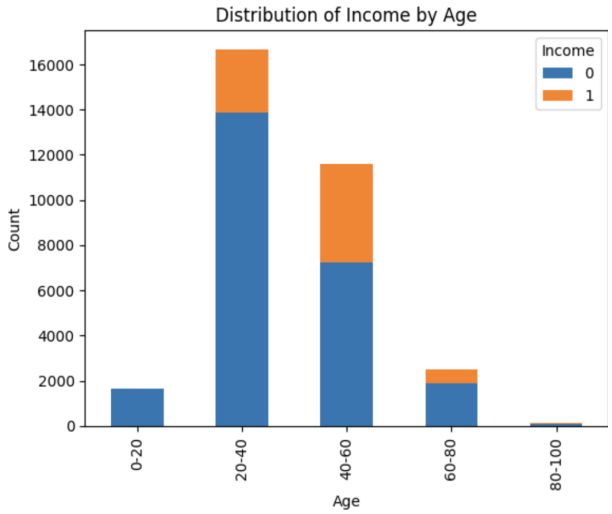
```
test.workclass.value_counts()

Private             11210
Self-emp-not-inc     1321
Local-gov            1043
?                     963
State-gov             683
Self-emp-inc          579
Federal-gov           472
Without-pay             7
Never-worked            3
Name: workclass, dtype: int64
```

So the Missing values were replaced by the most frequent value in the column or mathematically speaking the mode of the data.

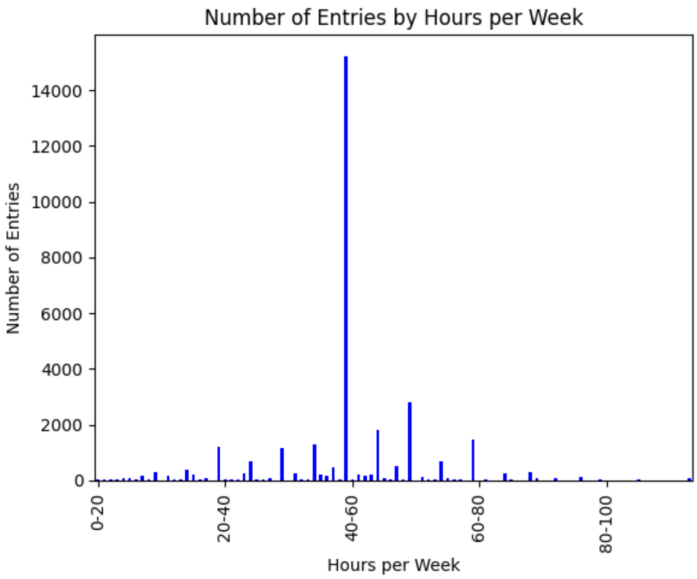
Data Relationships

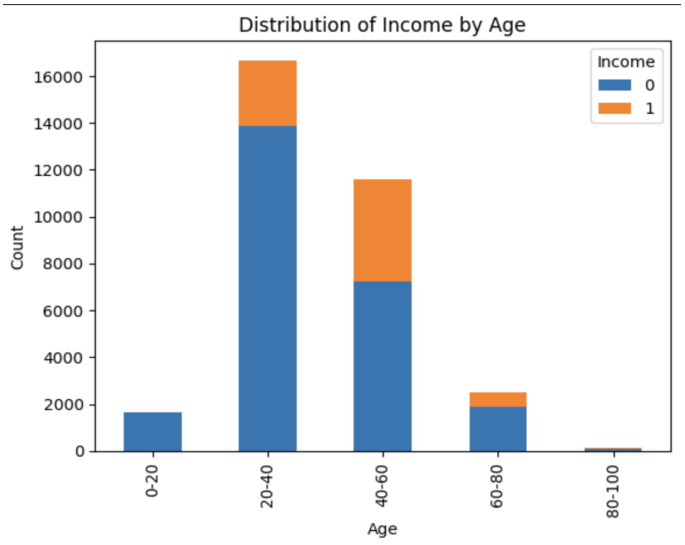
- Distribution of age and its relationship with the target attribute income (0-low and 1-high)



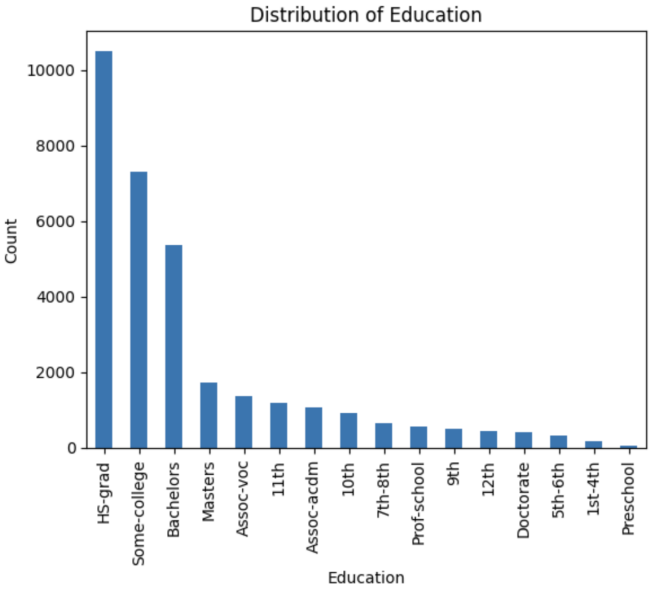


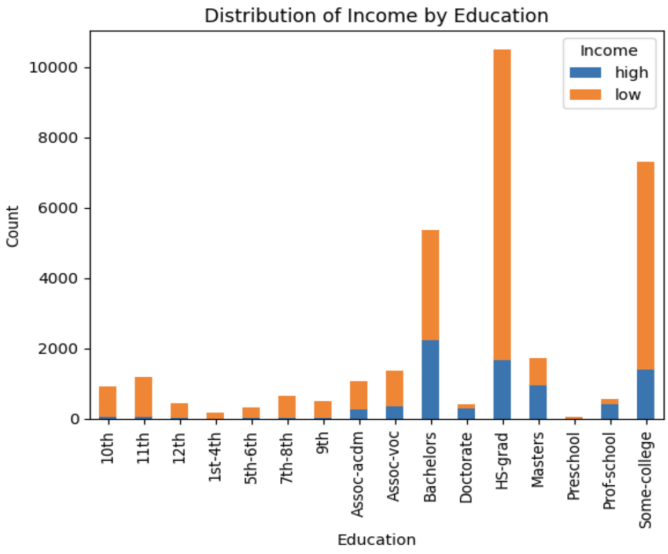
- Distribution of Work hours per week and its relationship with the target attribute income (0-low and 1-high)



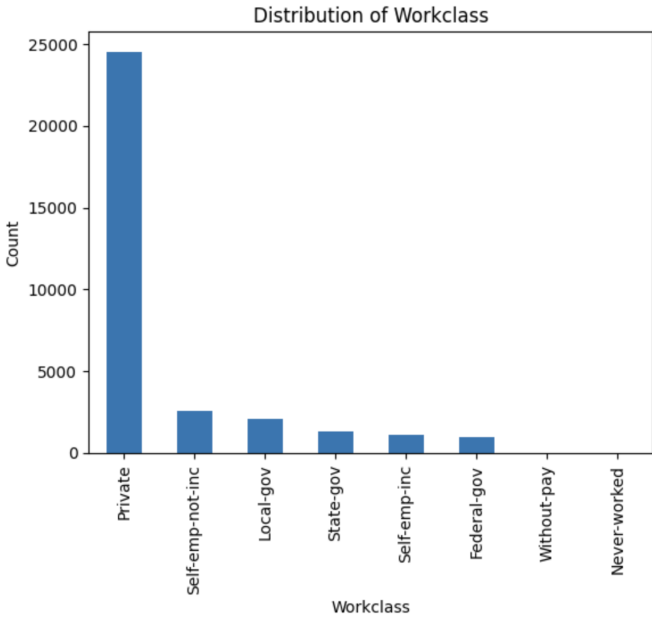


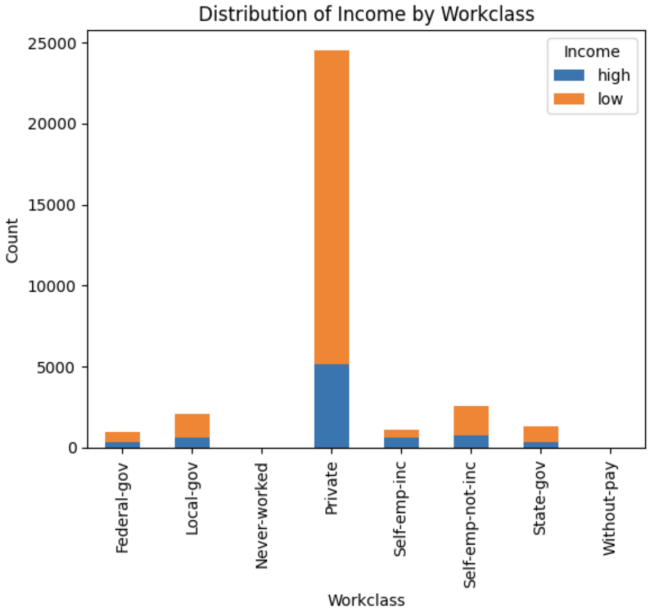
- Distribution of Education and its relationship with the target attribute income (0-low and 1-high)



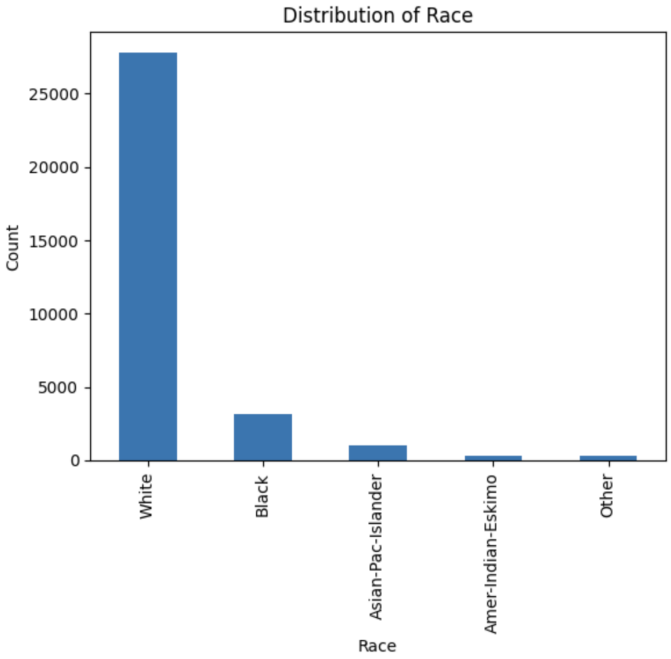


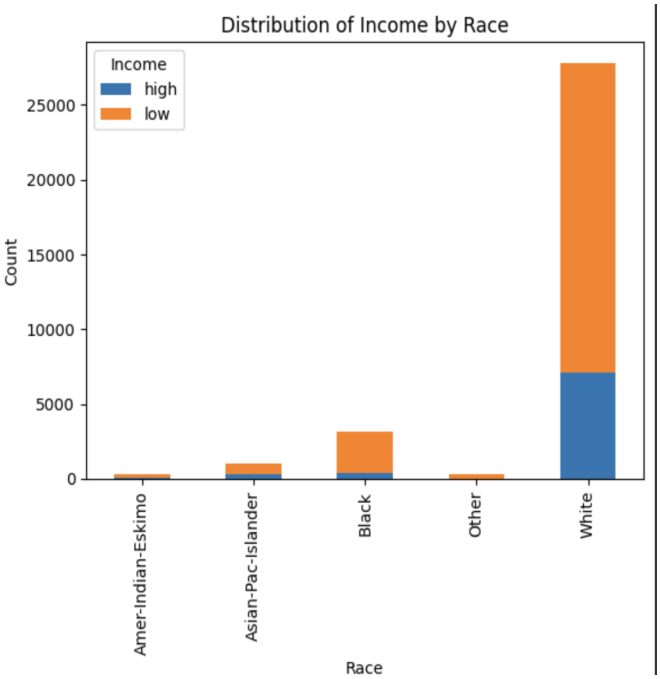
- Distribution of Workclass and its relationship with the target attribute income (0-low and 1-high)



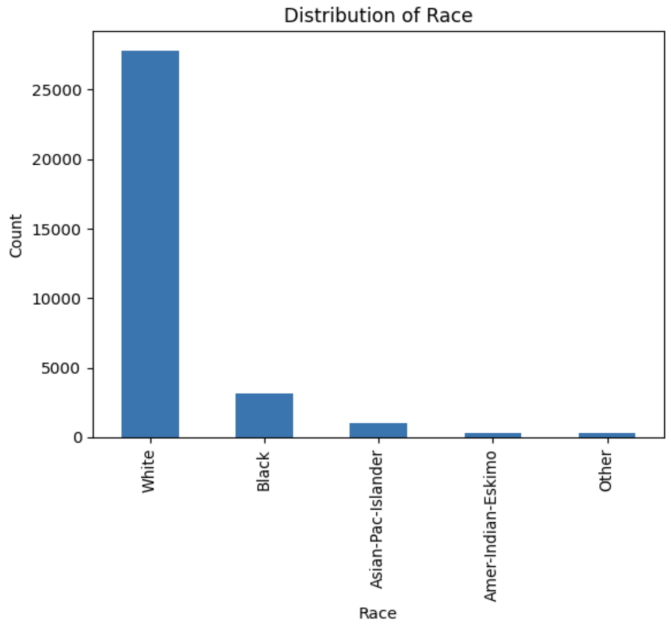


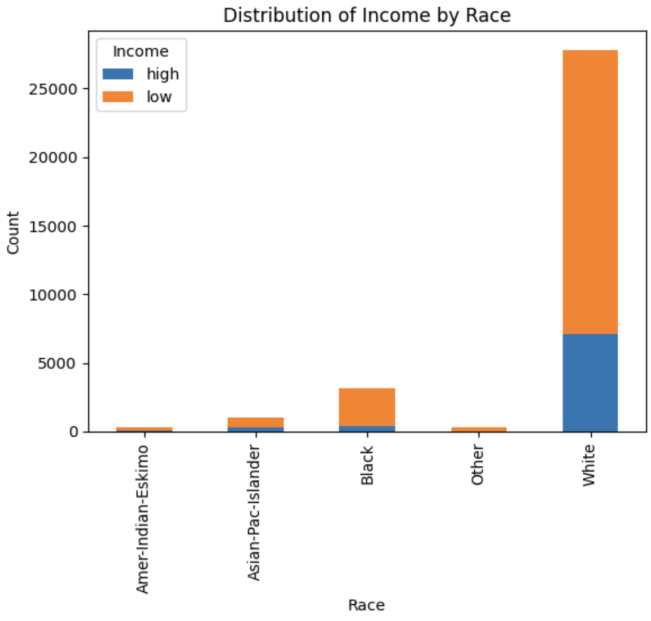
- Distribution of Race and its relationship with target attribute income



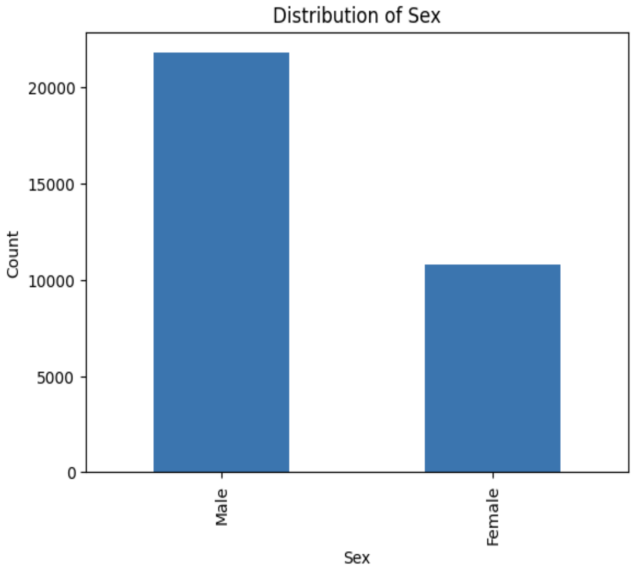


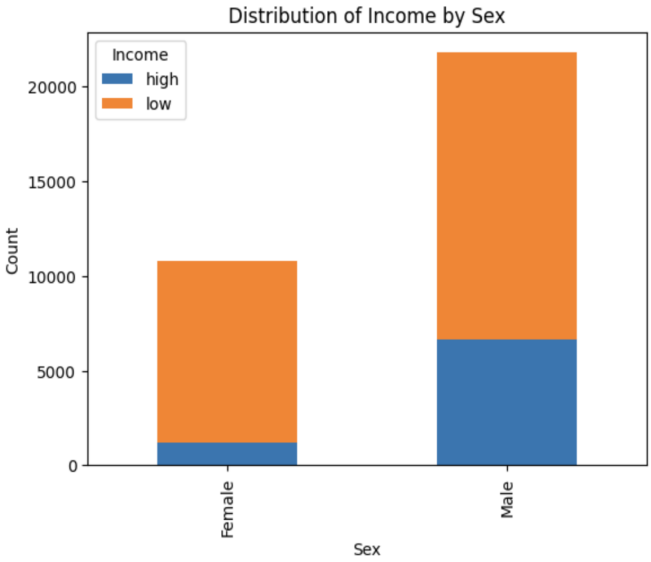
- Distribution of Race and its relationship with target attribute income



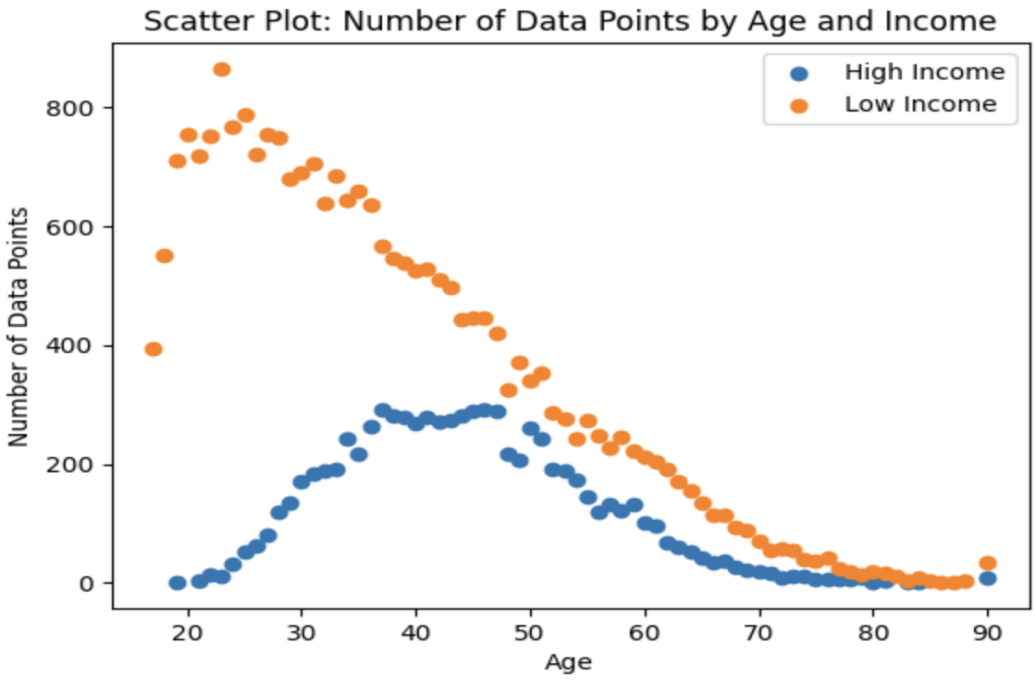


- Distribution of Sex and its relationship with target attribute income





- We also obeserve the relationship of low and high income classes with respect to age



Data Preparation

- Before creating the model, there are a few preprocessing tasks that need to be completed.

- First, take note that our predictors include both categorical and numerical features. Models, being mathematical equations can only analyze numeric variables. So we need to encode the categories.

- Decision trees can process categorical data easily, thus none of that is necessary. To enable scikit-learn to comprehend them and create the tree, we still need to encode the categorical variables into a common format. The LabelEncoder() class from sklearn.preprocessing will be used for this.

- Below is the encoding for the categorical variables

```
from sklearn import preprocessing

# encode categorical variables using label Encoder

# select all categorical variables
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50k
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50k
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50k
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50k
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50k

- Applying the label encoder to the data frame

```
# apply label encoder to df_categorical
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	6	9	4	0	1	4	1	38	0
1	5	9	2	3	0	4	1	38	0
2	3	11	0	5	1	4	1	38	0
3	3	1	2	5	0	2	1	38	0
4	3	9	2	9	5	2	0	4	0

- And then we merge the encoding into the dataframe

```
#Concatenate df_categorical dataframe with original df (dataframe)

# first, Drop earlier duplicate columns which had categorical values
df = df.drop(df_categorical.columns,axis=1)
df = pd.concat([df,df_categorical],axis=1)
df.head()
```

age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
39	77516	13	2174	0	40	6	9	4	0	1	4	1	38	0
50	83311	13	0	0	13	5	9	2	3	0	4	1	38	0
38	215646	9	0	0	40	3	11	0	5	1	4	1	38	0
53	234721	7	0	0	40	3	1	2	5	0	2	1	38	0
28	338409	13	0	0	40	3	9	2	9	5	2	0	4	0

Now we have completely encoded our data, the data is now ready to be used in the classifiers.

- For better performance of the classification models, we incorporate Feature Engineering. We created a new feature called “education_years” using the two features “education” and “education_num”. This new feature acts as a matured feature, which helps in better classification.

```
# Feature Engineering
train_encoded_copy['education_years'] = train_encoded_copy['education'] + train_encoded_copy['education_num']
```

- Finally we scale the features because a dataset's features may have many sizes and units. Scaling equalizes the scale of all characteristics so that no one feature can monopolize learning due to its greater magnitude. This aids in avoiding bias in favor of characteristics with higher values.

```
# Split the data into features and target variable
x_new = train_encoded_copy.drop('income', axis=1)
y_new = train_encoded_copy['income']

# Feature Scaling
scaler = MinMaxScaler()
x_new = scaler.fit_transform(x_new)
```

- To evaluate the initial performance, we split our training dataset into 80% training data and 20% testing data.

```
# Split the data into training and testing sets
x_train_new, x_test_new, y_train_new, y_test_new = train_test_split(x_new, y, test_size=0.2, random_state=42)
```

Performance Evaluation Metrics

1. **Accuracy**:

   A popular performance criterion for assessing classification models is accuracy. By comparing the proportion of examples that were properly classified to all instances, it assesses the general accuracy of the model's predictions.

   The formula for accuracy is

   Accuracy = (No. of Correct Predictions) / (Total No. of Predictions)

   Accuracy is a straightforward metric to interpret, as higher values indicate better performance.

2. **Precision**:

   Particularly in binary classification problems, precision is a performance parameter used to assess the accuracy of a classification model. It calculates the percentage of cases that were accurately predicted as positive (true positives) out of all instances that were projected as positive (true positives plus false positives). Precision focuses on the caliber of accurate forecasts.

   Precision is calculated as follows:

   $$\text{Precision} = \frac{TP}{TP + FP}$$

Where,

TP (True Positives) is a measure of how many cases were accurately classified as positive.
False Positives, or FP, is a measure of how many cases were wrongly classified as positive.

In essence, accuracy reveals the proportion of the model's optimistic predictions that come true. A lower false positive rate and a better level of confidence in the positive forecasts are both indicated by a higher precision number.

3. **Recall**:

Recall, which is often referred to as sensitivity or true positive rate, is a crucial performance indicator for assessing classification models. It calculates the percentage of positive cases that were accurately predicted (true positives) out of all positive instances in reality (true positives plus false negatives). Recall concentrates on accurately recalling favorable situations.

Recall is calculated as follows:

$$\text{Recall} = TP/(TP + FN)$$

Where,

TP (True Positives) is a measure of how many cases were accurately classified as positive.
False Negatives, or FN, is a measure of the number of situations that were wrongly classified as negative.

Recall, in other words, indicates the proportion of actual positive events that the model accurately detected. A reduced false negative rate and a greater capacity to detect positive examples are both indicated by a higher recall value.

4. **F-score**:

   In order to give both metrics equal weight, the F-score is derived using the harmonic mean of precision and recall. When both false positives and false negatives are taken into consideration, it offers a single value that summarises the model's overall performance. It offers a fair assessment of a model's capacity for accurately identifying both good and negative examples.

   The formula for the F-score is

   $$\text{F-score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

   The F-score is a number between 0 and 1, where a value of 1 indicates excellent recall and precision, and a value of 0 indicates subpar performance. An improved balance between recall and precision is indicated by a higher F-score.

5. **Confusion Matrix**:

   A classification model's performance is summarised in the confusion matrix, which offers details on prediction precision. True positives, false positives, true negatives, and false negatives are the four categories that make up this system. These classifications aid in evaluating how well the model can classify situations. Important metrics including accuracy, precision, recall, and F1-score can be computed from the

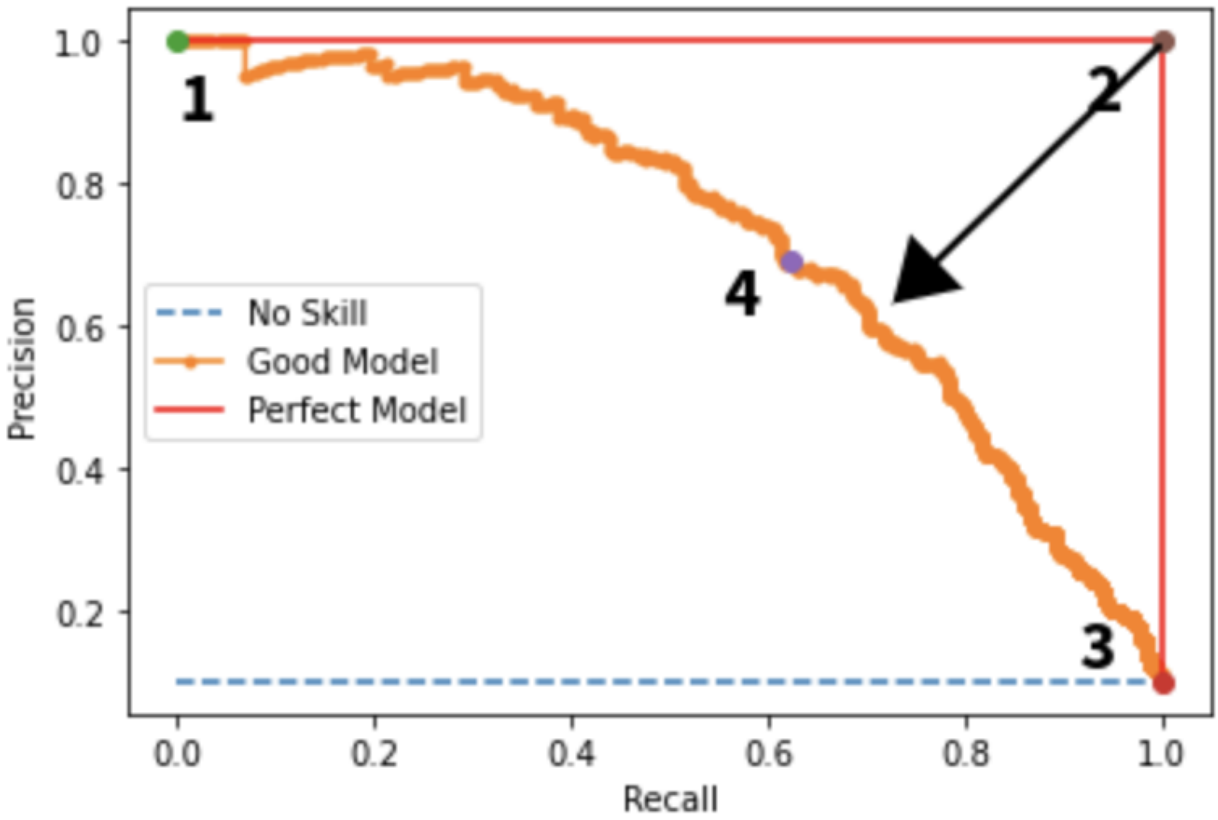
confusion matrix. It enables us to comprehend the model's performance and the kinds of mistakes it makes. In general, the confusion matrix is a helpful tool for assessing how well a categorization model works.

6. **Precision-Recall Curve**:

   The precision-recall curve illustrates a binary classification model's performance at various classification thresholds. It demonstrates how recall and precision can be traded off, two crucial evaluation measures for binary classification tasks.

   The precision-recall curve illustrates how accuracy and recall relate to various classification levels. The curve is produced by shifting the threshold from high to low, with each point on the curve representing a distinct threshold. The curve displays how the model's recall and precision vary as the classification threshold for positive cases is changed.

   A smooth curve, a trade-off between precision and recall, a significant area under the curve, and high precision and high recall are all characteristics of an ideal precision-recall angle. This suggests a model that may accurately detect instances of positivity while reducing false positives and negatives. A curve closer to the plot's top-right corner, signifying overall solid performance, would be present in a high-precision, high-recall model.



7. **ROC Curve**

   An illustration of a binary classification model's effectiveness in terms of receiver operating characteristic (ROC) curve. It shows how, at different thresholds, the true positive rate and the false positive rate can be traded off.

   The proportion of accurately anticipated positive cases to the total number of actual positive instances is known as the True Positive Rate, also known as sensitivity or recall. It gauges the model's accuracy in identifying favorable occurrences.

$$\textbf{T}\text{rue }\textbf{P}\text{ositive }\textbf{R}\text{ate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

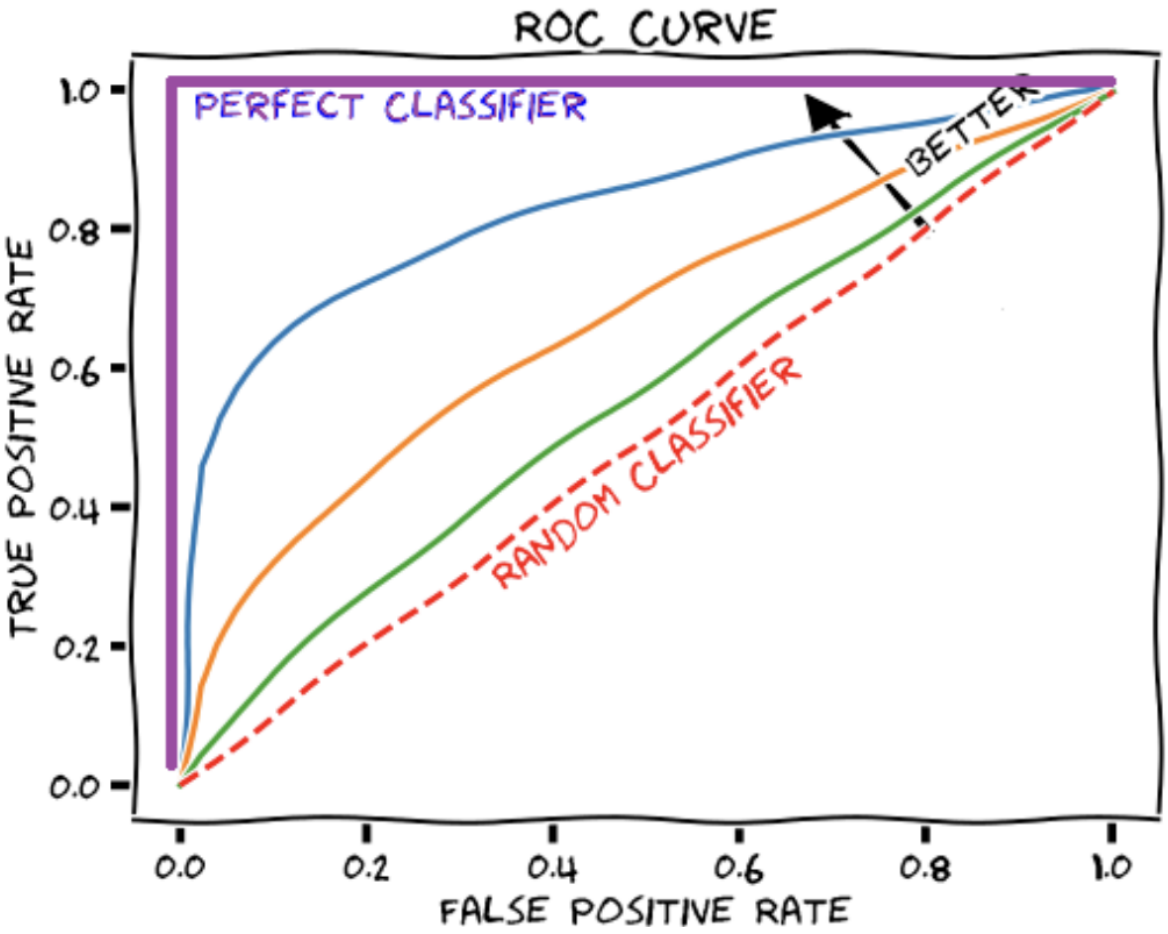
The ratio of positively predicted events that actually happened to all negatively forecasted events is known as the false positive rate. It gauges how frequently the model misclassifies situations as either positive or negative.

$$\textbf{F}\text{alse }\textbf{P}\text{ositive }\textbf{R}\text{ate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

The ROC curve sheds light on how well the model performs at various categorization criteria. Given that it has a higher TPR and a lower FPR, a curve that is towards the top-left corner is a better indicator of a model's performance. The curve's top-left corner represents a model that achieves a high TPR while maintaining a low FPR, and this is the ideal situation.

The area under the ROC curve (AUC) is a frequently used metric to assess the classifier's overall effectiveness. It offers a single value that encapsulates the classifier's capacity to distinguish between favorable and unfavorable occurrences. While an AUC

of 0.5 denotes a random classifier, an AUC of 1 shows a perfect classifier.



8. **Learning Curve**:

   A learning curve is a graph that illustrates how a machine learning model performs better with more training data. It aids in our comprehension of the connection between the performance of the model and the size of the training set.

   In a learning curve, the y-axis indicates the model's performance parameter, such as accuracy or mean squared error, and the

x-axis is the total number of training samples used. Two lines, one for the training set and the other for the validation set are commonly displayed on the curve.

The model may initially overfit, which means it performs well on the training set but poorly on the validation set when there is a short training set. The model generalizes better and performs better on the validation set when more training data is made available. The two lines are beginning to converge, showing that the model is successfully assimilating the data. More training data would be helpful if there is a significant difference between the training and validation curves, which indicates that the model is overfitting. On the other hand, if the curves have already converged and the model is still performing poorly, it may be a sign of a more complicated problem that calls for a different model architecture or feature engineering.

In a perfect learning curve, as the training set size grows, the training accuracy should either slightly decline or stay fairly consistent. It may get more challenging to correctly fit the training data when more training data becomes available since the model has a broader and more varied set of instances to learn from. On the other hand, as the training set size grows, the validation accuracy should also grow. This shows that the model is gaining the ability to generalize successfully to new data. The training and validation curves' convergence indicates that the model is striking the correct balance between fitting the training data and generalizing to new examples.

Naive Bayes Classifier

Based on the Bayes theorem and the premise that features are conditionally independent of one another, the Naive Bayes classifier is a straightforward but efficient classification technique. It is extensively utilized in a variety of Machine Learning tasks, including spam filtering and text classification.

Scikit-learn provides several variations of the Naive Bayes classifier, including GaussianNB, MultinomialNB, and BernoulliNB.

1. GaussianNB:

   The GaussianNB class in scikit-learn implements the Gaussian Naive Bayes algorithm, which is suitable for continuous or real-valued features. It assumes that the features follow a Gaussian (normal) distribution. The classifier calculates the mean and variance of each feature for each class label and uses this information to make predictions.

2. Training and Prediction:

   To train a Gaussian Naive Bayes classifier in scikit-learn, you can use the fit(X, y) method, where X represents the input features and y represents the corresponding target labels. Once trained, the model can make predictions on new data using the predict(X) method.

3. Assumption of Feature Independence:

   The features are conditionally independent given the class label, which is one of the main premises of the Naive Bayes classifier.

This presumption makes the computations easier to understand and enables effective model training and prediction. It might not apply in every situation in the actual world, though. The classifier's performance could suffer if the feature independence assumption is broken.

4. Evaluation and Performance:

   A Naive Bayes classifier's performance can be assessed using a variety of metrics, including accuracy, precision, recall, and F1-score. To calculate these metrics, Scikit-learn offers methods like classification_report and confusion_matrix.

We can now construct the Naive Bayes Classifier

- First we import the Naive Bayes Classifier from the scikit-learn library.

- Before training the model, we employ some Hyperparameter tuning to improve the performance of our classifier. Tuning is done on the following parameters:

    1. Variable Smoothing:

       Variable smoothing, commonly referred to as Laplace smoothing or additive smoothing, is a method used in probabilistic models to manage zero probabilities or frequencies. It is used, specifically for continuous features, for calculating the conditional probabilities in the Naive Bayes classifier.

The frequency and probability are both zero if a feature value is not present in the training data for a given class. This may cause problems with categorization, particularly if the test data contains unseen feature values. Variable smoothing solves this issue by increasing the frequency of each feature value for each class by a small constant amount, typically 1. This guarantees that no probability reaches zero and avoids problems with feature values that are not visible during categorization.

Hence we employ Hyperparameter tuning on “var_smoothing” parameter of the Naive Bayes classifier. We use the GridSearchCV for this. GridSearchCV is a technique to systematically search for the best hyperparameters from a predefined grid of values. The range of values for “var_smoothing” is specified as a logarithmically spaced sequence from 1 to 10^(-9). The “cv” parameter specifies the number of cross-validation folds (5 in this case) for evaluating the performance of different hyperparameter combinations.

GridSearchCV analyses the accuracy of each model by training the Gaussian Naive Bayes classifier on various combinations of “var_smoothing” values. GridSearchCV determines the model combination that produces the maximum accuracy by testing numerous models with various “var_smoothing” values. The code ends by creating a fresh instance of the Gaussian Naive Bayes classifier and setting the hyperparameter “var_smoothing” to the ideal value discovered by GridSearchCV.

Finally, we train our best model on the training dataset

```
#For the naive baye's classifier with hyperparamter tuning
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import GridSearchCV

#Create a Gaussian Classifier
model = GaussianNB()
# Define the parameter grid for GridSearchCV
param_grid = {'var_smoothing': np.logspace(0, -9, num=100)}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train_new, y_train_new)


nb_classifier = model.fit(x_train_new,y_train_new)
```

- Then, predictions are done on the testing data, and the performance parameters are calculated.

```
# Get the best model from GridSearchCV
nb_best_model = grid_search.best_estimator_

# Make predictions on the test set
nb_predictions = nb_best_model.predict(x_test_new)

# Evaluate performance
print(classification_report(y_test_new, predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.95      0.89      4942
           1       0.71      0.41      0.52      1571

    accuracy                           0.82      6513
   macro avg       0.77      0.68      0.70      6513
weighted avg       0.80      0.82      0.80      6513
```

Class 0 (<=50K):

The model performs well in class 0 prediction, with excellent recall (0.95) and precision (0.83). This indicates that the model can properly identify 95% of the real occurrences of class 0 and that it is 83% accurate when classifying an instance as belonging

to class 0. For class 0, the F1-score of 0.89 shows a fair balance between precision and recall.

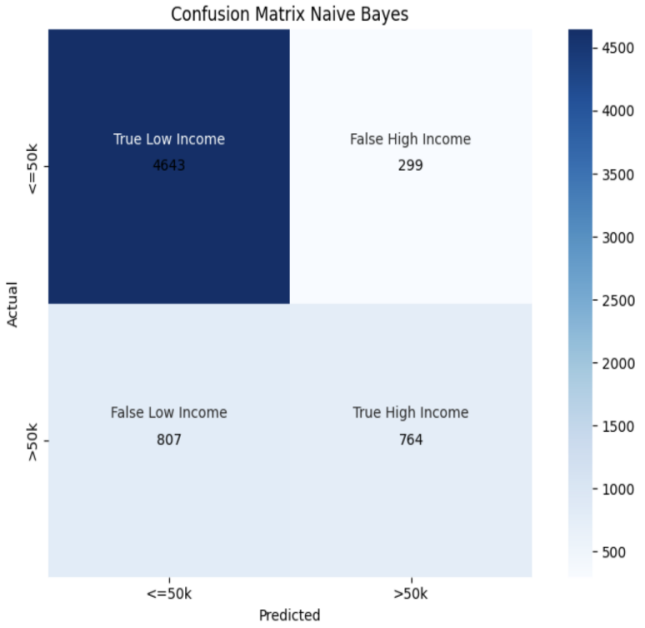
Class 1 (>50K):

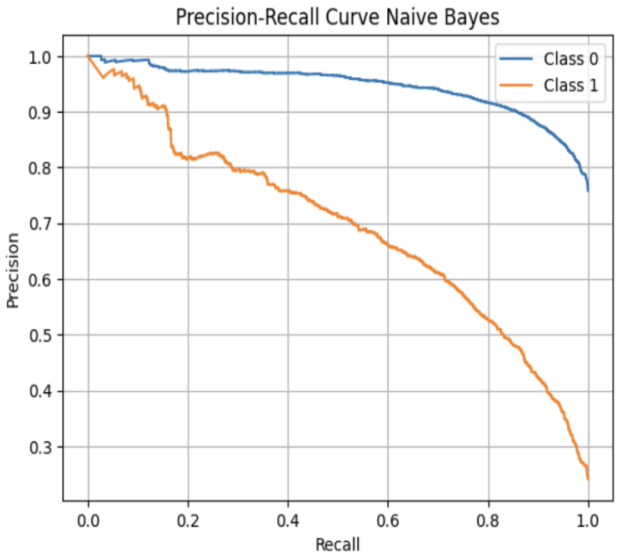
The model's performance in predicting class 1 is lower, with lower recall (0.41) and precision (0.71). This indicates that just 41% of the actual examples of class 1 can be accurately identified by the model, and yet when it predicts an instance to be class 1, it is correct 71% of the time. The F1-score of 0.52 indicates that the model's performance for class 1 could be improved.

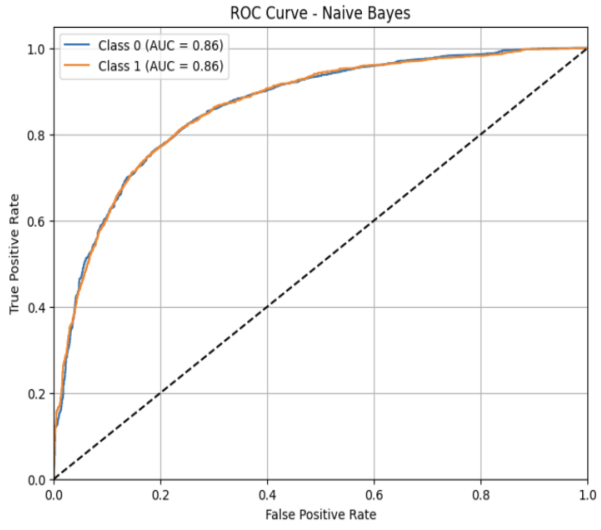
The lower performance for class 1 could be attributed to the fact that the data set provided to us is Imbalanced. In comparison to class 0 (low-income group), class 1 (high-income group) has a much smaller number of data points. The model's capacity to properly learn and generalize patterns for the minority class (class 1) can be impacted by class imbalance. The model could not have enough representative samples to effectively learn the characteristics that set high-income people apart if there are fewer instances of class 1. Lower precision and recall for class 1 predictions may follow from this.

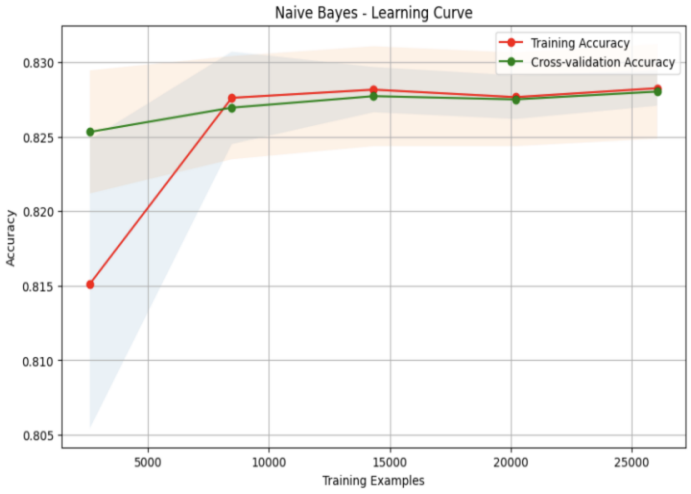
We obtained an overall accuracy of 82% on our Naive Bayes Classifier.

The various performance metrics are plotted as follows.


Confusion Matrix Naive Bayes
True Low Income
4643
False High Income
299
False Low Income
807
True High Income
764
<=50k
>50k
Actual
<=50k
>50k
Predicted
4500
4000
3500
3000
2500
2000
1500
1000
500


Precision-Recall Curve Naive Bayes
Class 0
Class 1
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve - Naive Bayes
Class 0 (AUC = 0.86)
Class 1 (AUC = 0.86)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


Naive Bayes - Learning Curve
Training Accuracy
Cross-validation Accuracy
Accuracy
Training Examples
0.830
0.825
0.820
0.815
0.810
0.805
5000
10000
15000
20000
25000

Logistic Regression Classifier

For binary classification tasks, the logistic regression classifier is a well-liked and often-used technique. It is a linear model that calculates the likelihood that an instance belongs to a given class using the logistic function. Through the LogisticRegression class, Scikit-learn offers an adaptable and effective logistic regression implementation.

1. Modeling Approach:

   A linear relationship between the input features and the logarithm of the target class's probabilities is assumed by the model in logistic regression. The linear combination is transformed into a probability value between 0 and 1 by using the logistic function, also referred to as the sigmoid function. The anticipated probabilities are multiplied by a threshold value (often 0.5), which establishes the decision limit.

2. Training and Prediction:

   Using the fit(X, y) method in scikit-learn, where X stands for the input characteristics and y for the matching target labels, we can train a logistic regression model. Using the predict(X) method, which returns the projected class labels, the model can make predictions on fresh data after being trained.

3. Regularization:

   In order to avoid overfitting, regularisation terms are frequently used in logistic regression models. Regularisation adds a penalty for high coefficient values, which helps manage the model's complexity. The LogisticRegression class in Scikit-learn lets you

specify various regularisation methods, including L1 (Lasso) and L2 (Ridge) regularisation.

4. Assessment and Performance:

   Accuracy, precision, recall, and F1-score are some of the measures that can be used to evaluate how well a logistic regression classifier performs. To construct these metrics and assess the model's performance, Scikit-learn offers techniques like confusion_matrix and classification_report.

We can now construct the Logistic Regression Classifier

- First we import the Logistic Regression Classifier from the scikit-learn library.

- Before training the model, we employ some Hyperparameter tuning to improve the performance of our classifier. Tuning is done on the following parameters:

    1. C or Regularization parameter:

       Overfitting is a problem with logistic regression models, particularly when working with high-dimensional data or a lot of input features. When a model fits training data too closely, overfitting happens. This might result in noise or irrelevant patterns being captured, which can impair the generalization of new data.

       Through the addition of a penalty term to the objective function that the model optimizes, the regularisation parameter, frequently abbreviated as C in logistic

regression, serves to reduce overfitting. This penalty term deters the model from giving a particular feature an excessive amount of weight or from overfitting the data.

The trade-off between keeping the model's coefficients (weights) minimal and providing a good fit to the training data is controlled by the regularisation parameter C. More complicated patterns may be captured by the model with a higher C value, but the risk of overfitting is also increased. A higher C value helps the model to fit the training data more closely. A lower C value, on the other hand, enforces stronger regularisation, which incentivizes the model to have smaller coefficients and clearer decision bounds.

2. Solver:

   In logistic regression, the solver parameter refers to the optimization procedure used to identify the model's ideal weights. Sklearn offers a variety of solver alternatives, each with unique advantages and scenario-appropriate uses.

   - 'lbfgs' is a limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm suitable for small to medium-sized datasets.
   - 'newton-cg' is an iterative algorithm based on the Newton's method with conjugate gradient for better memory efficiency.
   - 'liblinear' is a library for large-scale linear classification that supports both L1 and L2 regularization. It is suitable for binary and multi-class problems on small datasets.

- 'sag' is a stochastic average gradient algorithm designed for large-scale problems and can handle both L1 and L2 regularization.
- 'saga' is an extension of 'sag' that also supports both L1 and L2 regularization and is faster for some datasets.

GridSearchCV is used to adjust the logistic regression classifier's hyperparameters. The logistic regression classifier, the parameter grid, and the evaluation measure are all inputs to the GridSearchCV function. It does a cross-validated grid search using the supplied measure to assess the model's performance across all conceivable hyperparameter combinations. Based on the highest score, the ideal set of hyperparameters is selected.

A new logistic regression model is trained on the training dataset to utilize the top hyperparameters after they have been found.

```
#Logistic Regression with Hyperparamter Tuning
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Create a logistic regression classifier
logreg = LogisticRegression(max_iter=500)

# Define the hyperparameters to tune
param_grid = {'C': np.logspace(-1, 2, num=100),'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']}

# Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(x_train_new, y_train_new)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train a logistic regression model with the best hyperparameters
best_logreg = LogisticRegression(**best_params)
best_logreg.fit(x_train_new, y_train_new)
```

- Then, predictions are done on the testing data, and the performance parameters are calculated.

```
# Make predictions and evaluate performance
logreg_predictions = best_logreg.predict(x_test_new)
print(classification_report(y_test_new, logreg_predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.89      4942
           1       0.71      0.46      0.56      1571

    accuracy                           0.82      6513
   macro avg       0.78      0.70      0.72      6513
weighted avg       0.81      0.82      0.81      6513
```

Class 0 (<=50k):

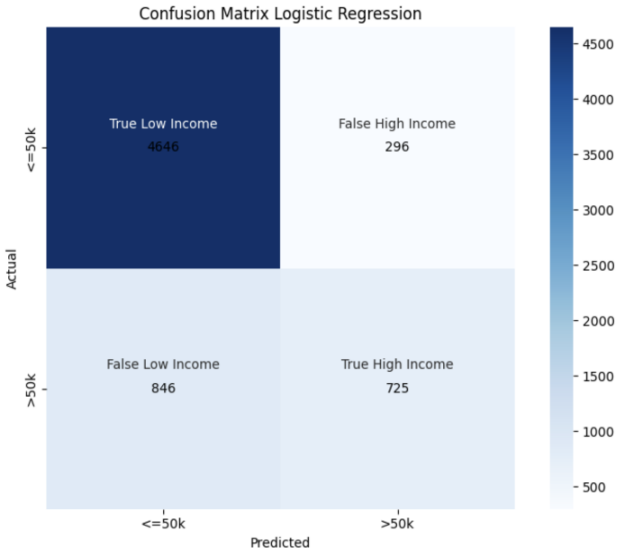
The model's precision and recall for class 0 are 0.85 and 0.94, respectively. This indicates that the model is accurate 85% of the time when it predicts that a person's income will be less than or equal to $50,000 and that it can recognize 94% of the actual situations when income is less than or equal to $50,000. The F1-score for class 0 (income less than or equal to $50,000) is 0.89, showing a decent balance between precision and recall for predicting people with that level of income.

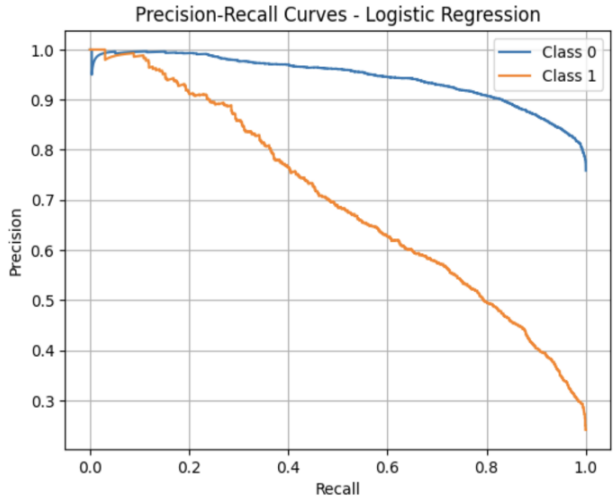
Class 1 (>50k):

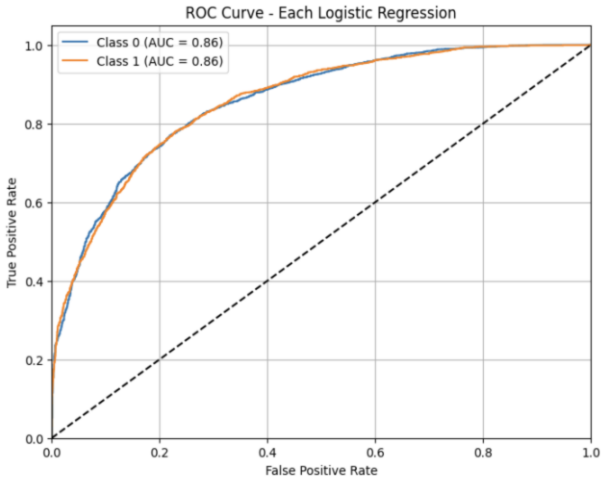
For class 1, the model displays a precision of 0.71 and a recall of 0.46. This means that when the model predicts a person's income to be over $50,000, it is accurate 71% of the time and can account for 46% of the actual cases when income is above $50,000.The F1-score for class 1 (>50k) is 0.56, which indicates that the model may have some trouble accurately classifying people with greater earnings.

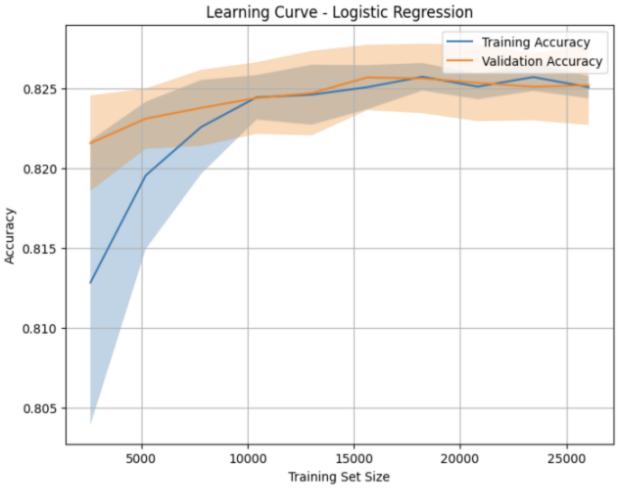
The lower performance for class 1 could be attributed to the fact that the data set provided to us is Imbalanced. In comparison to class 0 (low-income group), class 1 (high-income group) has a much smaller number of data points. The model's capacity to properly learn and generalize patterns for the minority class (class 1) can be impacted by class imbalance. The model could not have enough representative samples to effectively learn the characteristics that set high-income people apart if there are fewer instances of class 1. Lower precision and recall for class 1 predictions may follow from this.

We obtained an overall Accuracy of 82% for our Logistic Regression Classifier. The other performance metrics are plotted as follows.









Neural Network Classifier

A potent machine learning system called Neural Network Classifier uses brain-like behavior to handle challenging issues. The neural network classifier in Scikit-Learn is implemented in the Multi-Layer Perceptron class, or MLPClassifier.

1. MLP Classifier:

   By utilizing many layers of connected neurons, the MLPClassifier is able to discover complicated patterns and relationships in the data. It can simulate both linear and non-linear relationships since it supports a variety of activation functions, including logistic (sigmoid), tanh, and ReLU.

2. Training and Prediction:

   The architecture of the network, including the number of hidden layers and the number of neurons in each layer, must be specified when training a neural network classifier. Through an iterative procedure known as backpropagation, the classifier discovers the ideal weights and biases by modifying the parameters in response to the discrepancy between expected and actual outputs. The neural network classifier can use the predict() method to generate predictions on new data after training.

3. Performance and Evaluation:

A variety of metrics, including accuracy, precision, recall, and F1-score, can be used to assess the classifier's performance. For difficult situations involving vast volumes of data, neural network classifiers are especially useful. They can be computationally expensive, though, and need to be carefully tuned to avoid overfitting.

Before we begin, we need to decide the sizes of the hidden layers. A general rule of thumb based on practical observations and heuristics states that the number of hidden neurons in a neural network should be 2/3 the size of the input layer plus the size of the output layer. Although it is not a rigid rule, it offers a place to start when figuring out how many buried neurons are necessary.

Number of Neurons in the hidden layer =
(⅔)*(Number of neurons in the Input layer) + (Number of neurons in the Output layer)

The goal of this rule is to achieve a compromise between the network's ability to recognize intricate links in the data and the danger of overfitting. A network may struggle to recognize the underlying patterns in the data if there are insufficient hidden neurons. This condition is known as underfitting. On the other hand, having an excessive number of hidden neurons may result in overfitting, when the network becomes overly focused on the training set and performs badly on unobserved input. The recommendation makes sure that the network has enough capacity to learn the required representations and generate precise predictions by taking the size of the input and output layers into account.
We have 15 features (14 features +1 matured feature) that comprise the Input Layer and 1 target attribute (income) which comprises the Output Layer.

So, the sizes of the hidden layers will be

- Neural network with 1 hidden layer:

  Size of Hidden Layer = (⅔)*(15) + 1 = 11 neurons

- Neural network with 2 hidden layers:

  Size of Hidden layer 1 = (⅔)*(15) + 1 = 11 neurons
  Size of Hidden layer 2 = (⅔)*(11) + 1 = 9 neurons (approx)

- Neural network with 3 hidden layers:

  Size of Hidden layer 1 = (⅔)*(15) + 1 = 11 neurons
  Size of Hidden layer 2 = (⅔)*(11) + 1 = 9 neurons (approx)
  Size of Hidden layer 3 = (⅔)*(9) + 1 = 7 neurons

We can now construct the Neural network with 1,2 and 3 hidden layers using the MLP Classifier.

1. Neural Network with 1 Hidden Layer:

   - Import the MLP Classifier from the scikit-learn library and create a Neural Network with 1 hidden layer.

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# Create a Neural Network classifier with 1 hidden layer
neural_classifier_1 = MLPClassifier(hidden_layer_sizes=(11), random_state=42,activation='logistic',max_iter=10000)
```

   - Then we train the neural network on the training dataset and make predictions on the testing data, after which the performance parameters are evaluated.

```
# Train the classifier
neural_classifier_1.fit(x_train_new, y_train_new)

# Make predictions on the test set
neural_predictions_1 = neural_classifier_1.predict(x_test_new)

# Evaluate performance
print(classification_report(y_test_new, neural_predictions_1))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.90      4942
           1       0.74      0.57      0.65      1571

    accuracy                           0.85      6513
   macro avg       0.81      0.75      0.77      6513
weighted avg       0.84      0.85      0.84      6513
```

Class 0 (<=50k):

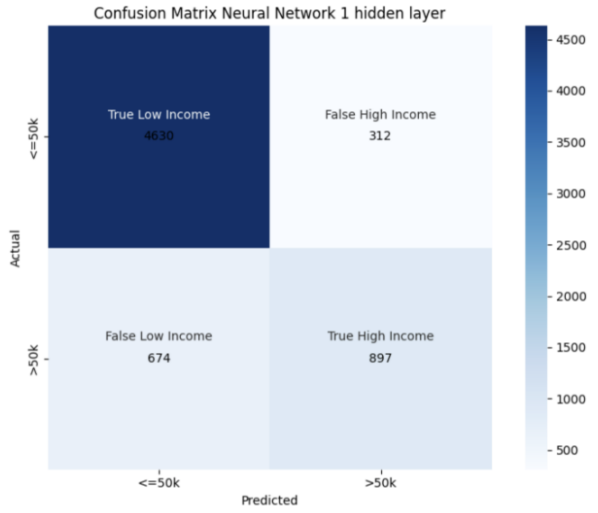
The precision is 0.87, meaning that 87% of the time the model is right when it predicts that a sample belongs to class 0. The recall is 0.94, indicating that 94% of the real class 0 samples were correctly identified by the model. The harmonic mean of precision and recall, or the F1-score, is 0.90, and it provides a balanced assessment of the model's performance for class 0.

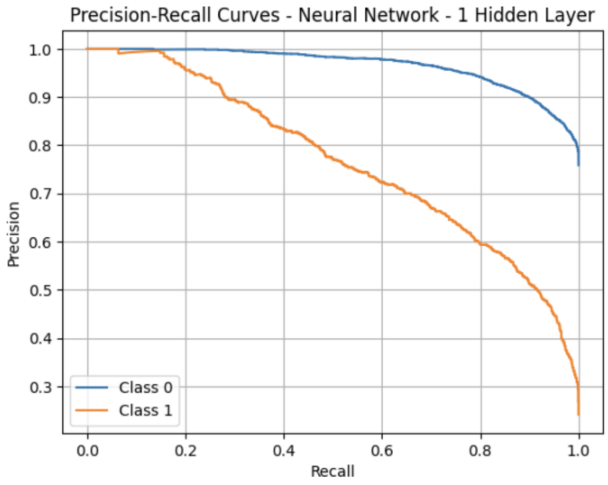
Class 1 (>50k):

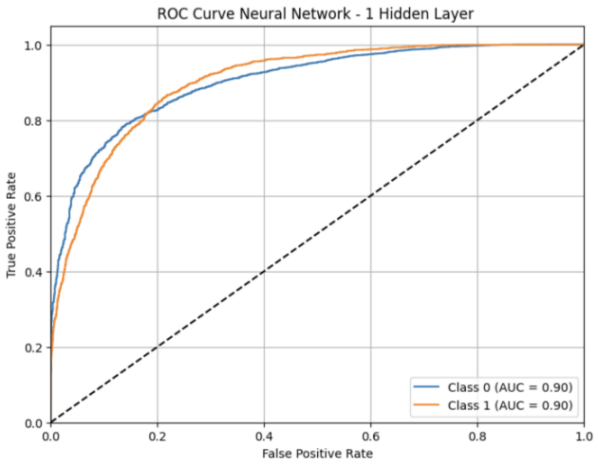
The precision is 0.74, meaning that 74% of the time the model is right when it classifies a sample as belonging to class 1. The recall is 0.57, indicating that 57% of the real class 1 samples are correctly identified by the model. The F1-score for class 1 is 0.65 and offers a balanced assessment of recall and precision.

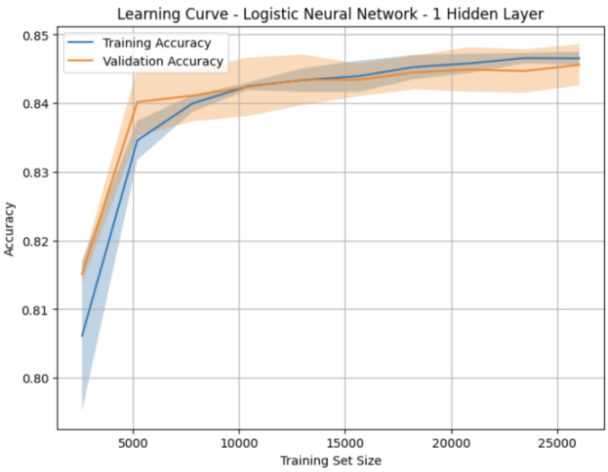
The lower performance for class 1 can be attributed to the same reasons we stated in the earlier classifiers (imbalanced dataset).

The overall Accuracy stands at 85%. The other performance metrics are plotted below









2. Neural Network with 2 Hidden Layers:

- Using the same procedure as above, the only thing that changes is we pass in 2 layer sizes (11 and 9) in the MLP Classifier.

```
# Create a Neural Network classifier with 2 hidden layers
neural_classifier_2 = MLPClassifier(hidden_layer_sizes=(11, 9), random_state=42,activation='logistic',max_iter=1000)
```

- Then we train the network and perform predictions. The performance report is as follows

```
# Evaluate performance
print(classification_report(y_test_new, neural_predictions_2))
```

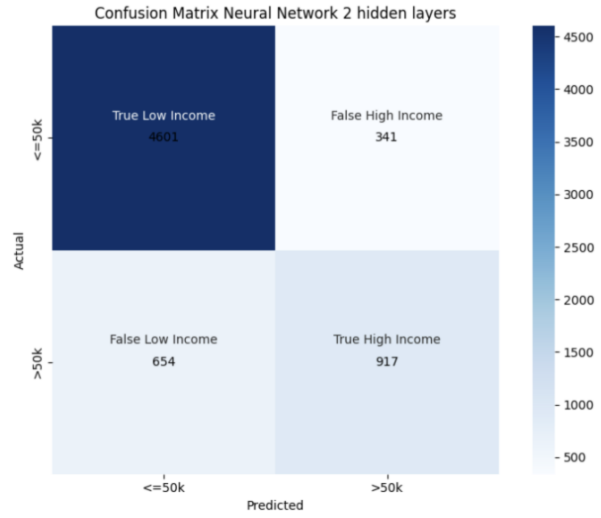
	precision	recall	f1-score	support
0	0.88	0.93	0.90	4942
1	0.73	0.58	0.65	1571
accuracy			0.85	6513
macro avg	0.80	0.76	0.78	6513
weighted avg	0.84	0.85	0.84	6513

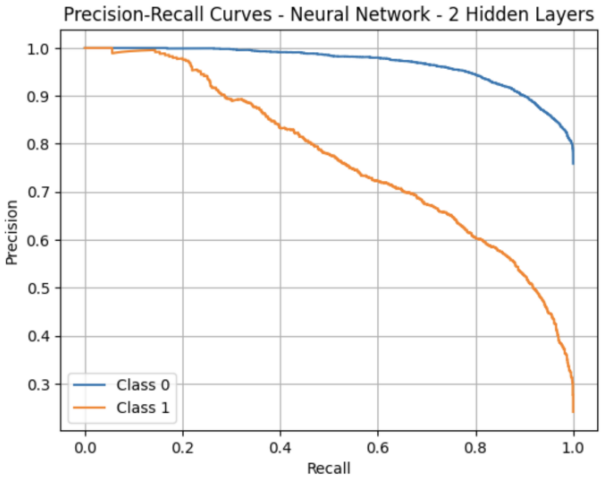
Class 0 (<=50K):
The precision is 0.88, meaning that 88% of the time the model is right when classifying a sample as class 0. Recall is 0.93, which means that 93% of the real class 0 samples are correctly identified by the model. The harmonic mean of precision and recall, or the F1-score, is 0.90, and it provides a balanced assessment of the model's performance for class 0.

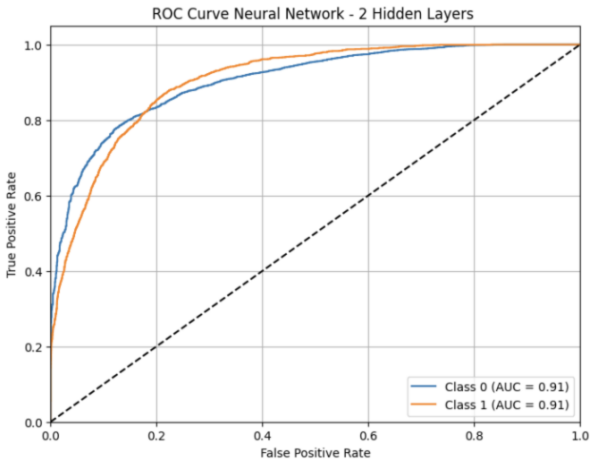
Class 1 (>50k):
The precision is 0.73, meaning that 73% of the time the model is right when classifying a sample as belonging to class 1. Recall is 0.58, indicating that 58% of the real class 1 samples are

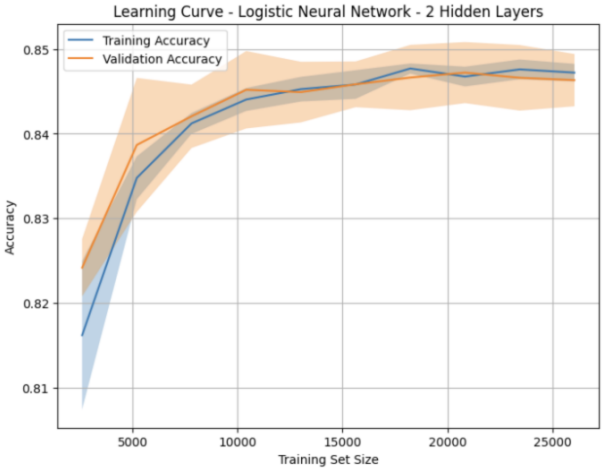
correctly identified by the model. The F1-score for class 1 is 0.65 and offers a balanced assessment of recall and precision.

An overall Accuracy of 85% was achieved. The other performance evaluation metrics are plotted as follows.









3. Neural Network with 3 Hidden Layers:

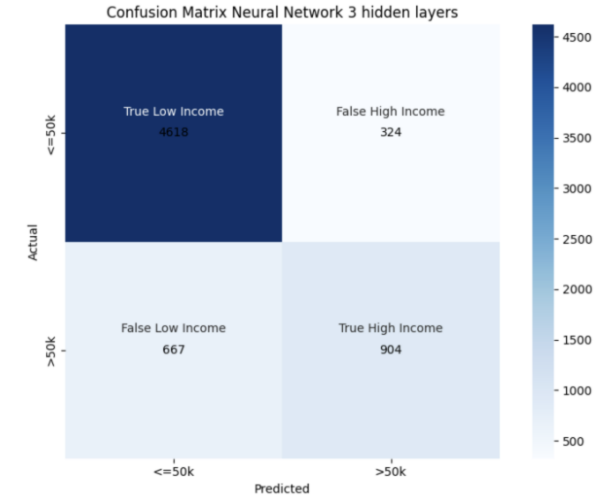
    - Using the same methodology, we train a neural network with 3 hidden layers (sizes 11, 9, and 7) and make predictions. We get the performance report as

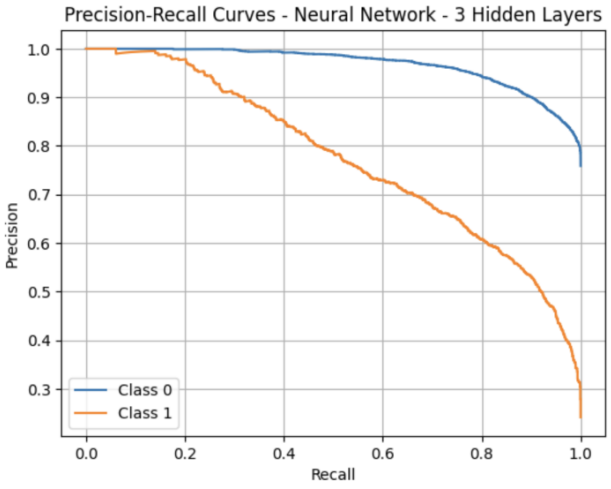
	precision	recall	f1-score	support
0	0.87	0.93	0.90	4942
1	0.74	0.58	0.65	1571
accuracy			0.85	6513
macro avg	0.80	0.75	0.77	6513
weighted avg	0.84	0.85	0.84	6513

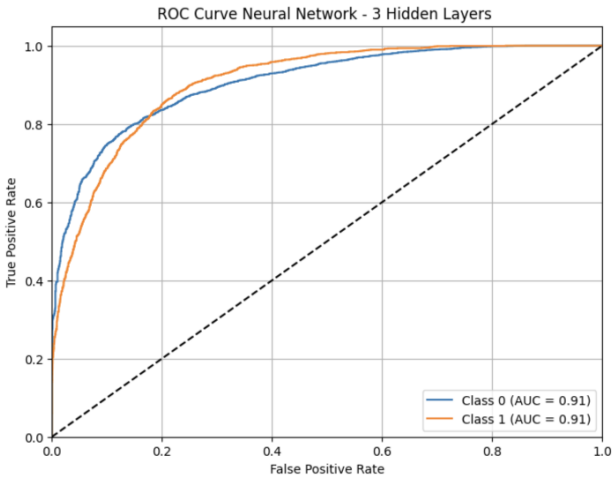
Class 0 (<=50k):
With a precision of 0.87, the model correctly classifies samples as Class 0 87% of the time. Recall is 0.93, which means that 93% of the real class 0 samples are correctly identified by the model. The harmonic mean of precision and recall, or the F1-score, is 0.90, and it provides a balanced assessment of the model's performance for class 0.

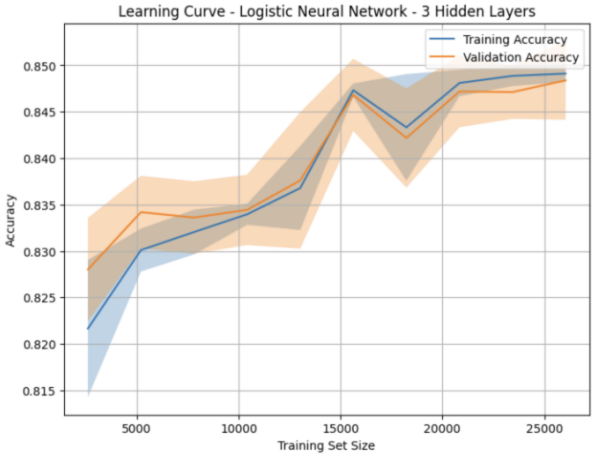
Class 1 (>50k):
The precision is 0.74, meaning that 74% of the time the model is right when classifying a sample as belonging to class 1. Recall is 0.58, indicating that 58% of the real class 1 samples are correctly identified by the model. The precision and recall trade-off for class 1 is represented by the F1-score of 0.65.

An overall Accuracy of 85% was achieved. The other performance evaluation metrics are plotted as follows.









Till now, we just built the Naive Bayes, Logistic Regression, and Neural Network Classifiers respectively. Now we put forward a Comparative study on the following classifiers:

1. Naive Bayes Classifier
2. Logistic Regression Classifier
3. Neural Network Classifier
4. Decision Tree Classifier
5. Random Forest Classifier

For this we combine our training data and testing data and then 67% of this will be taken as training data and 33% as testing data.

Combining the Datasets

- We combine the training and testing data points and randomly select 67% of the data points as the training data set and the remaining data points as the testing data set.

- We combined the dataset and shuffled the values and divided it into training and testing data.

```
from sklearn.utils import shuffle
combined = pd.concat([train_encoded_copy, test_encoded_copy], ignore_index=True)

# Shuffle the combined dataframe
combined_encoded = shuffle(combined)

# Reset the index of the shuffled dataframe
combined_encoded.reset_index(drop=True, inplace=True)

combined_encoded.to_csv("combined_encoded.csv")
```

Split the data into training and testing,

```
# Split the data into training and testing sets
x_combined_train_new, x_combined_test_new, y_combined_train_new, y_combined_test_new = train_test_split(x_combined_new,y_combined
```

Comparative Study

1. **Naive Bayes Classifie**r:

   The implementation of the Naive Bayes Classifier has been thoroughly discussed in the preceding sections.

   - We import the required dependencies from scikit-learn. Then we incorporate Hyperparameter tuning for better performance of the model. The parameter onto which tuning is done is Variable smoothing (var_smoothing), commonly referred to as Laplace smoothing or additive smoothing, which is a method used in probabilistic models to manage zero probabilities or frequencies. Refer to the preceding sections for a detailed explanation.

   - Then the model is trained on the training dataset, and predictions are done on the testing dataset.

```
#For the naive baye's classifier with hyperparamter tuning
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
from sklearn.model_selection import GridSearchCV

#Create a Gaussian Classifier
model = GaussianNB()
# Define the parameter grid for GridSearchCV
param_grid = {'var_smoothing': np.logspace(0, -9, num=100)}

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_combined_train_new, y_combined_train_new)
```

   - The performance report is then produced for the model.

```python
# Get the best model from GridSearchCV
nb_cmb_best_model = grid_search.best_estimator_

# Make predictions on the test set
cmpstd_predictions = nb_cmb_best_model.predict(x_combined_test_new)

# Evaluate performance
print(classification_report(y_combined_test_new, cmpstd_predictions))
```

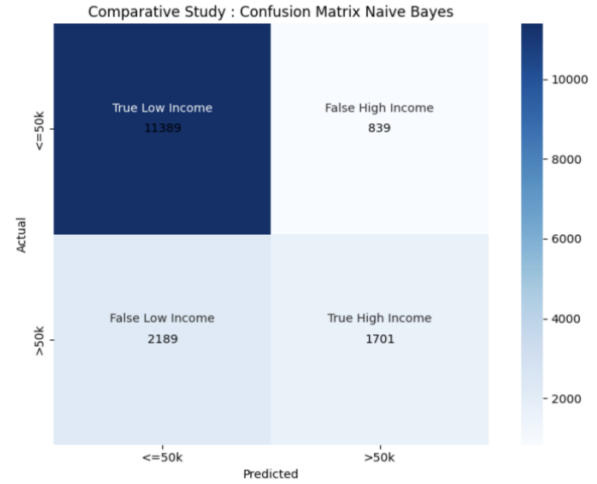
```
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     12264
           1       0.72      0.48      0.57      3854

    accuracy                           0.83     16118
   macro avg       0.78      0.71      0.73     16118
weighted avg       0.82      0.83      0.82     16118
```

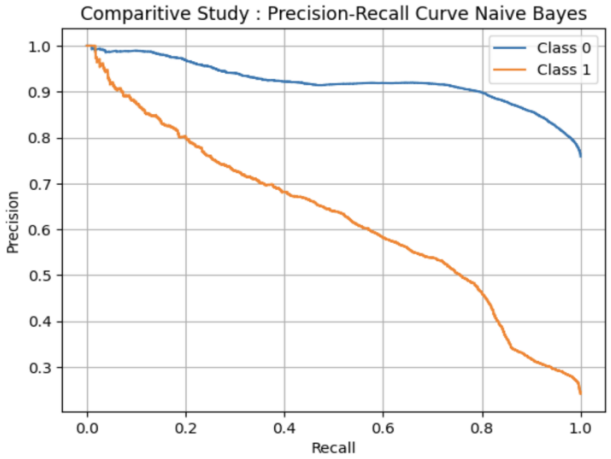
Class 0 (<=50K):
The model's class 0 precision is 0.84, which indicates that 84% of the examples classified as class 0 are in fact in class 0. The model captures 93% of the real events that fall into class 0 according to the recall of 0.93. The F1-score of 0.88 is a balanced indicator of recall and precision.

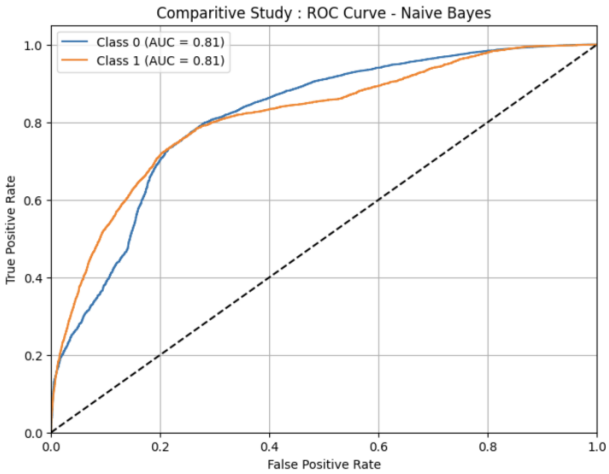
Class 1 (>50K):
The precision for class 1 is 0.67, meaning that 67% of the events that were correctly classified as class 1 were seen. Recall is 0.44, indicating that only 44% of real instances of class 1 are captured by the model. For class 1, the F1-score of 0.53 shows a modest balance between recall and precision.

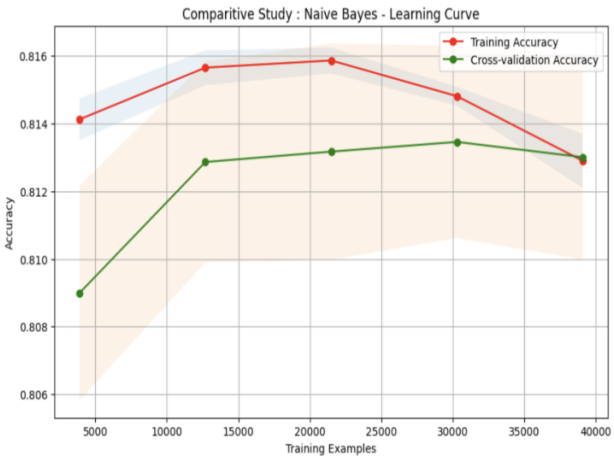
Overall, the model performs well in classifying instances of class 0 with high precision and recall. However, there is room for improvement in correctly identifying instances of class 1. The reasons for this have already been discussed in the preceding sections (Imbalanced Datasets).

An overall accuracy of 81% was achieved. The other performance metrics are plotted below as follows.









Several significant insights into the model's performance are revealed by the precision-recall curve study. The curve for class 0 exhibits a smooth decline from 1 to roughly 0.78, demonstrating a high degree of precision and accuracy in classifying positive events. The curve's proximity to the upper right corner indicates that class 0 has an excellent balance between recall and precision. The curve, however, displays a fragile and erratic pattern for class 1. The model's performance appears to rapidly degrade as it tries to catch more positive examples, as indicated by a precision-recall (PR) curve that drops steeply and reaches a precision of about 0.2 at recall 1.

There is a substantial decline in precision towards the conclusion, showing difficulty correctly categorising all positive cases of class 1 as such. Overall, the model shows an improved understanding of class 0 patterns but struggles with the complexities of class 1.

The model works, as shown by the ROC curve analysis and the curve's proximity to the top left corner. The model's capacity to distinguish between positive and negative examples is further confirmed by the AUC (Area Under the Curve) values of 0.81 for both classes. The model appears to have a good balance of true positive rate and false positive rate for both classes, as indicated by a higher AUC, which also suggests better overall performance.

According to the learning curve analysis, the model initially gains knowledge from the training data, increasing training accuracy. The training accuracy eventually reaches a plateau and then starts to decline as it approaches the validation accuracy when more data is added to the training process. This behaviour shows that the model can initially capture the patterns in the training data, but as it learns from more training data, it gets more generalised and less prone to overfitting. The model works well and can generalise successfully, according to the convergence of training and validation accuracy, which shows a solid balance between capturing the training data patterns and doing so to previously unknown data.

2. **Logistic Regression Classifier**:

The implementation of Logistic Regression Classifier has been thoroughly discussed in the preceding sections.

- We import the required dependencies from scikit-learn. Then we incorporate Hyperparameter tuning for better performance of the model. Tuning is done on the “C” or the regularization parameter and the “solver”. The regularization parameter serves to reduce overfitting. This penalty term deters the model from giving a particular feature an excessive amount of weight or from overfitting the data. The solver parameter refers to the optimization procedure used to identify the model's ideal weights. Please refer to the preceding sections to get an in-depth understanding.

- The model is then trained on the training data and predictions are done on the testing data.

```
#Logistic Regression with Hyperparamter Tuning
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Create a logistic regression classifier
logreg = LogisticRegression(max_iter=500)

# Define the hyperparameters to tune
param_grid = {'C': np.logspace(-1, 2, num=100),'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']}

# Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(x_combined_train_new, y_combined_train_new)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train a logistic regression model with the best hyperparameters
best_logreg = LogisticRegression(**best_params)
best_logreg.fit(x_combined_train_new, y_combined_train_new)
```

- The performance report is obtained as follows.

```
# Make predictions and evaluate performance
cmpstd_logreg_predictions = best_logreg.predict(x_combined_test_new)
print(classification_report(y_combined_test_new, cmpstd_logreg_predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.94      0.89     12264
           1       0.71      0.46      0.56      3854

    accuracy                           0.83     16118
   macro avg       0.78      0.70      0.72     16118
weighted avg       0.81      0.83      0.81     16118
```

Class 0 (<=50k):
The model has an accuracy of 85% for class 0, meaning that 85% of the occurrences classified as class 0 were accurate. 94% of the actual cases of class 0 were correctly identified by the model, according to the recall for class 0. For class 0, the F1-score, which balances recall and precision, is 89%.

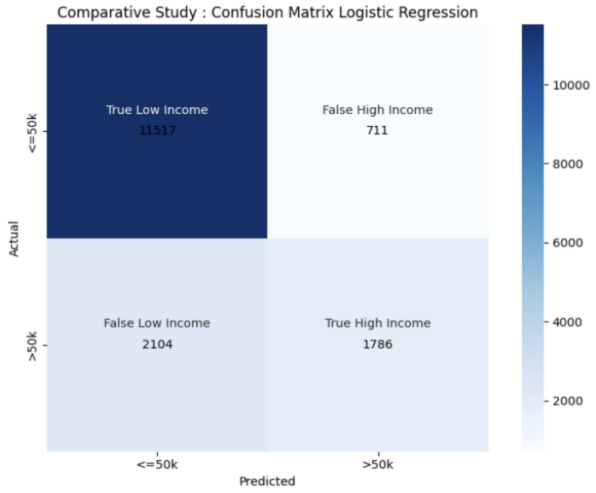
Class 1 (>50k):
The precision for class 1 is 71%, meaning that 71% of the instances classified as class 1 were accurate. The model missed 54% of the actual instances of class 1 based on the recall, which is 46%. The precision and recall trade-off shown by the class 1 F1-score is 56%.

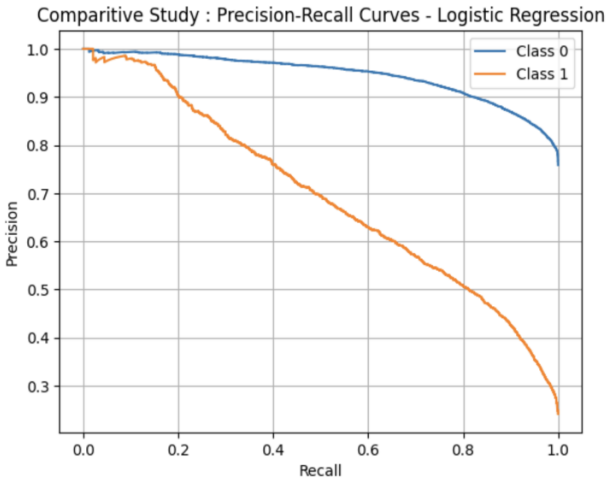
The F1 score is 72% when compared to the macro average, which gives both classes equal weight. The weighted average, which accounts for the support for each class, results in an F1 score of 81%.

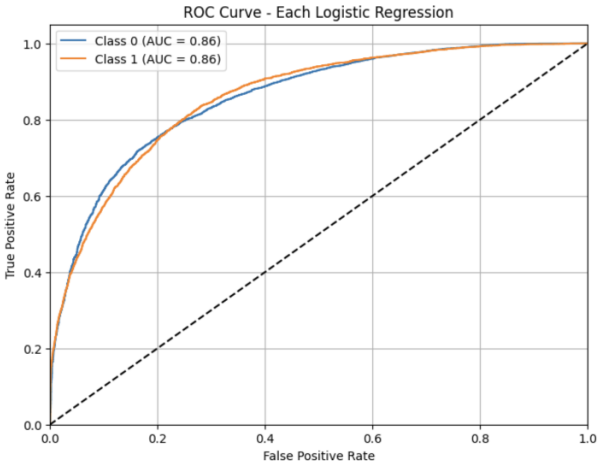
The model performs well in class 0 instance predictions with excellent precision and recall. For class 1, it performs less well, with worse recall and precision. This shows that the model might have difficulty correctly categorising occurrences of class 1. The

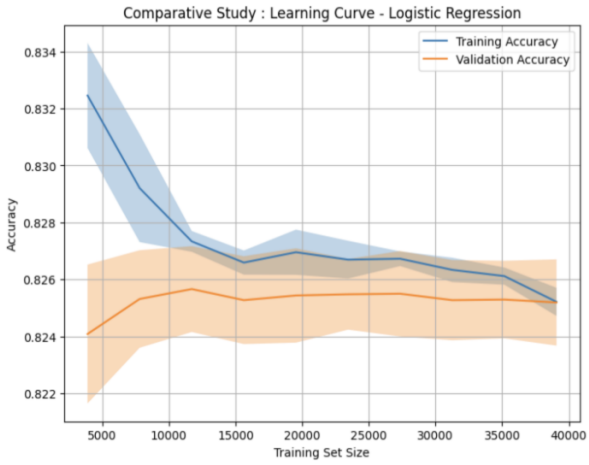
reasons for these have been already discussed earlier. (Imbalanced dataset)

An overall Accuracy of 83% was achieved. The other performance metrics are plotted as follows.









The examination of the PR curve for class 1 reveals an intriguing trend. At first, when the model tries to catch more positive cases, the precision reduces abruptly. For the remaining positive examples, it begins to recover and regains a high level of precision. Beyond a certain point, the recall starts to grow while the precision starts to decrease once more. The precision is

noticeably low at the maximum recall, demonstrating the model's difficulty in precisely identifying all positive events. This implies a compromise between increasing the number of positives captured and preserving high precision for class 1 forecasts.

As the model includes more examples of negative data, the PR curve for class 0 shows stable or slightly declining precision. The model's capacity to accurately identify negative cases, however, is demonstrated by the precision's continued, high level. A solid balance between precision and recall for class 0 predictions may be seen by the curve's position, which is more pronounced in the top right corner.

In conclusion, the model accurately identifies negative cases for class 0 and performs well in terms of recall and precision. There is still potential for development for class 1, particularly in terms of keeping a high precision at increasing recall values.

The true positive rate and false positive rate of the model are both about in the top left corner of the ROC curves for both classes, indicating strong performance. The changes in the curves' behaviour that have been noticed point to slight variances in the model's performance at certain thresholds. AUC (Area Under the Curve) values of 0.81 for both classes show that the model can distinguish between positive and negative occurrences rather well. Better discriminating power is indicated by an AUC value that is higher and closer to 1, but an AUC of 0.81 still demonstrates that the model does a good job of differentiating between the two groups.

The training curve begins at roughly 0.83 and gradually flattens out to about 0.824. The validation curve, on the other hand, rises

from roughly 0.82 to about 0.825. The training and validation curves' convergence indicates that the model's performance stabilises when more data is made available. The close proximity of the curves proves that the model is neither overfitting nor underfitting the training set. This implies a reasonable balance between the model's performance on the training data and its generalizability to new data (validation curve). It is a model that fits well.

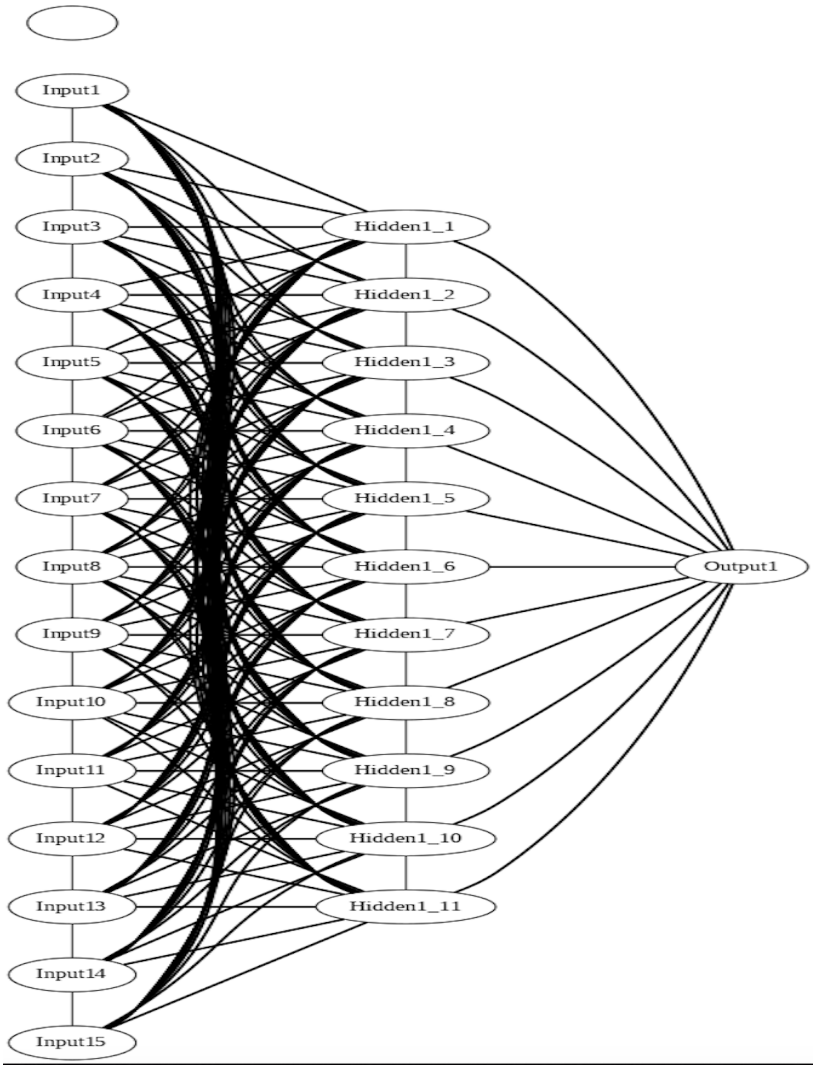
3. **Neural Network Classifier**:

   The implementation of Neural Networks with 1,2 and 3 hidden layers has been thoroughly discussed in the preceding sections.

   We import the required dependencies from scikit learn and then train the model on the training dataset using the MLP Classifier and then do predictions on the testing datasets.

   a. Network with 1 hidden layer (Size 11):

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# Create a Neural Network classifier with 1 hidden layer
cmpstd_neural_classifier_1 = MLPClassifier(hidden_layer_sizes=(11), random_state=42,activation='logistic',max_iter=10000)

# Train the classifier
cmpstd_neural_classifier_1.fit(x_combined_train_new, y_combined_train_new)
```



The performance report is as follows.

```
# Evaluate performance
print(classification_report(y_combined_test_new, cmpstd_neural_predictions_1))
```

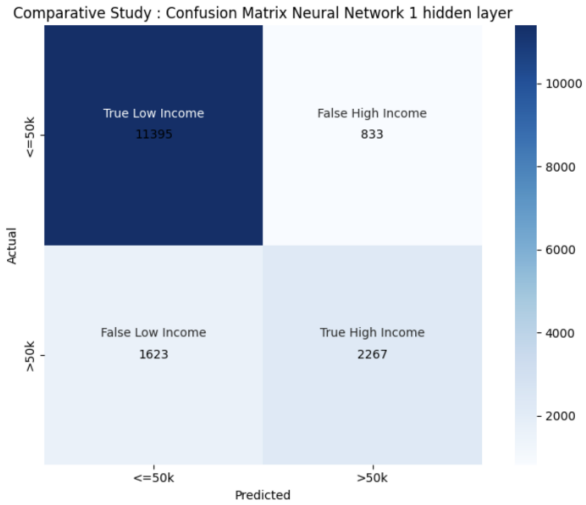
```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     12228
           1       0.73      0.58      0.65      3890

    accuracy                           0.85     16118
   macro avg       0.80      0.76      0.78     16118
weighted avg       0.84      0.85      0.84     16118
```

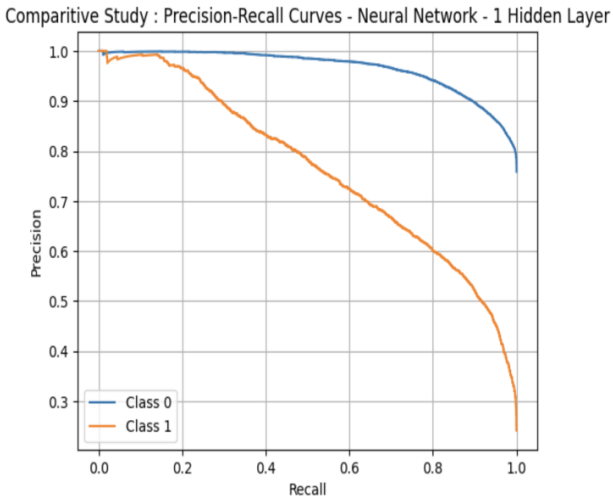
Class 0 (<=50k):
The model shows excellent precision for class 0, indicating that it accurately predicts the vast majority of occurrences falling under class 0. Additionally, a high recall indicates that the model successfully recognises the majority of instances of class 0. The F1-score of 0.90 shows that precision and recall are well-balanced, indicating strong performance for class 0.

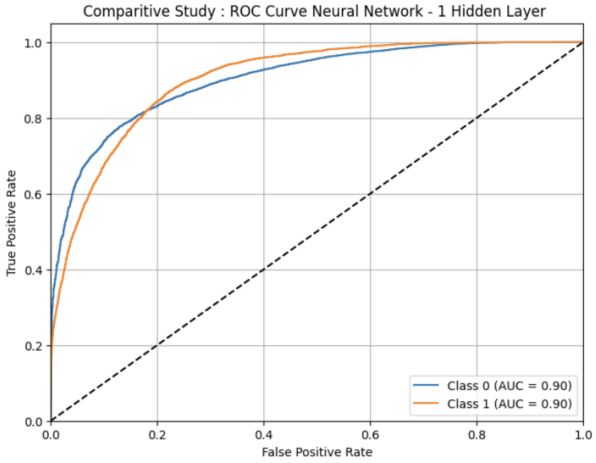
Class 1 (>50k):
The precision for class 1 is lower than for class 0, indicating that the model produces more false-positive predictions for class 1 than for class 0. The recall is also considerably lower, which indicates that the model misses a large proportion of occurrences that belong to class 1. The model's performance for class 1 is relatively worse than for class 0, according to the F1-score of 0.65.

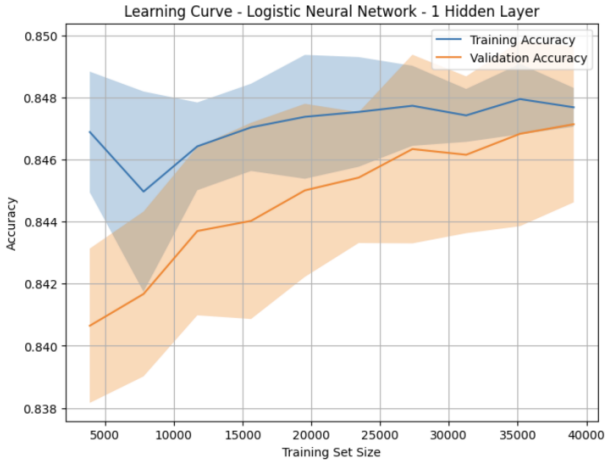
In conclusion, the model exhibits good class 0 performance, with excellent precision, recall, and F1-score. The performance for class 1 is, however, worse, with lower F1-score, recall, and precision. Reasons for the lower performance for class 1 has already been discussed in the preceding sections (Imbalanced Datasets)

An overall Accuracy of 85% was achieved. The other performance metrics are plotted as follows:









A higher level of general precision is seen in the PR curve for class 0. The curve is more pronounced in the upper right corner, indicating strong class 0 prediction precision and recall. Even while the precision decreases from 1 to roughly 0.8 as the recall rises, it still remains high. This shows that the model maintains a respectably high precision while doing well in reliably recognising class 0. After a brief drop at a precision of 1, the PR curve for class 1 recovers. This shows that although the model initially accurately predicts positive events, there may be a small

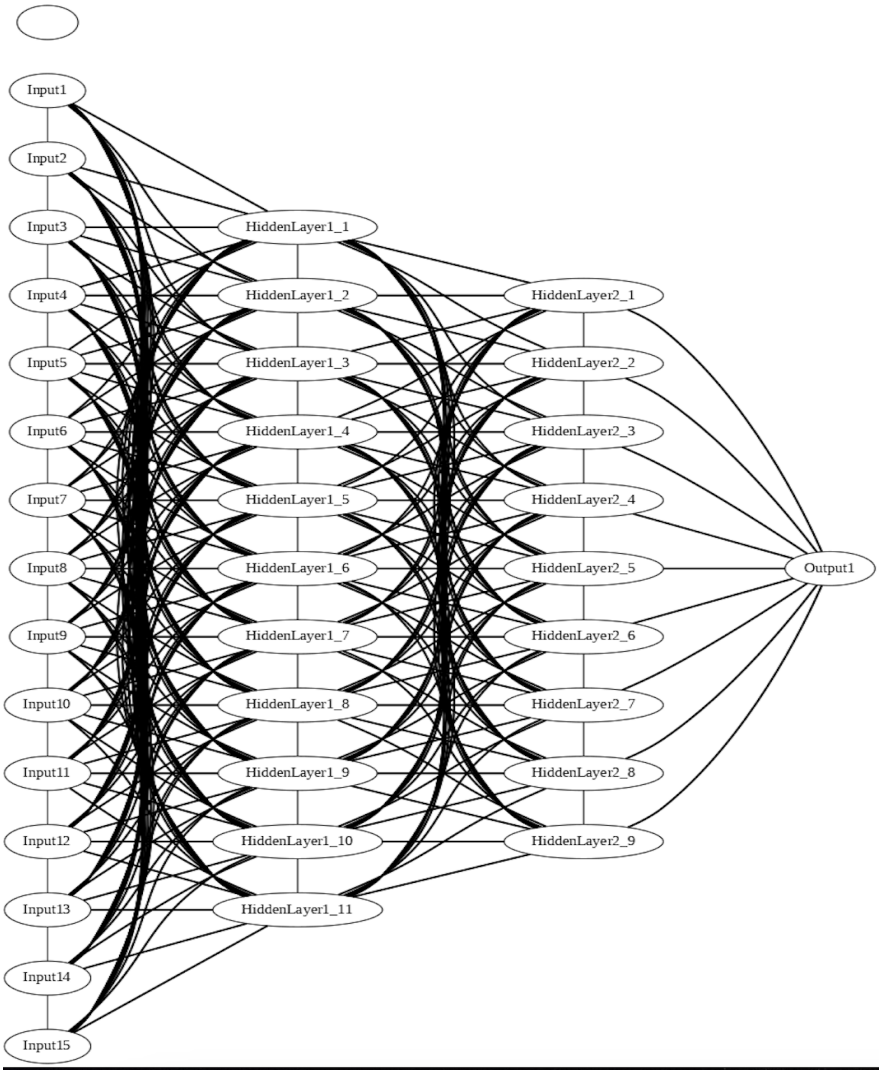
number of false positives. Precision for class 1 predictions significantly decreases as recall rises, falling to a value of roughly 0.1. This suggests that as the recall threshold rises, the model finds it difficult to correctly recognise all positive events.

The fact that the ROC curves for both classes are more or less closer to the top left corner indicates that the model performed exceptionally well in terms of both true positive rate (sensitivity) and false positive rate (1 - specificity). Because it is so close to the top left corner, it is likely that the model has a high true positive rate and a low false positive rate. Additionally, the model's great discriminative power is indicated by the AUC (Area Under the Curve) value of 0.9 for both classes. AUC values that are nearer 1 show a higher ability to tell good from negative events.

As more training examples are employed, the training curve first falls from 0.848 to 0.844, suggesting a little drop in the model's performance on the training data. The curve then begins to rise once more, indicating that the model performs better as additional training data are added. The validation curve also shows an improvement in the model's performance on the validation data, rising from 0.84 to 0.846 at first. The validation curve similarly converges to a value that is comparable to the training curve as the amount of training data grows. The convergence shows that the model achieves a reasonable balance between generalisation and capturing the underlying patterns in the data, not overfitting nor underfitting the data.

b. Network with 2 hidden layers (Size 11 and 9):

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# Create a Neural Network classifier with 2 hidden layers
cmpstd_neural_classifier_2 = MLPClassifier(hidden_layer_sizes=(11,9), random_state=42,activation='logistic',max_iter=10000)

# Train the classifier
cmpstd_neural_classifier_2.fit(x_combined_train_new, y_combined_train_new)
```



The Performance report is as follows.

```
# Make predictions on the test set
cmpstd_neural_predictions_2 = cmpstd_neural_classifier_2.predict(x_combined_test_new)

# Evaluate performance
print(classification_report(y_combined_test_new, cmpstd_neural_predictions_2))
```

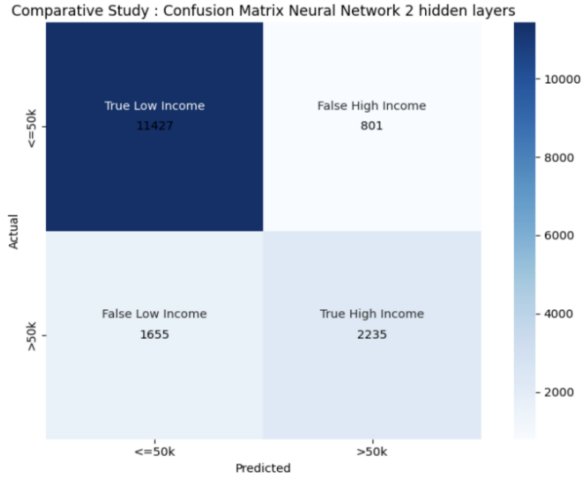
```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90     12264
           1       0.72      0.59      0.65      3854

    accuracy                           0.85     16118
   macro avg       0.80      0.76      0.77     16118
weighted avg       0.84      0.85      0.84     16118
```

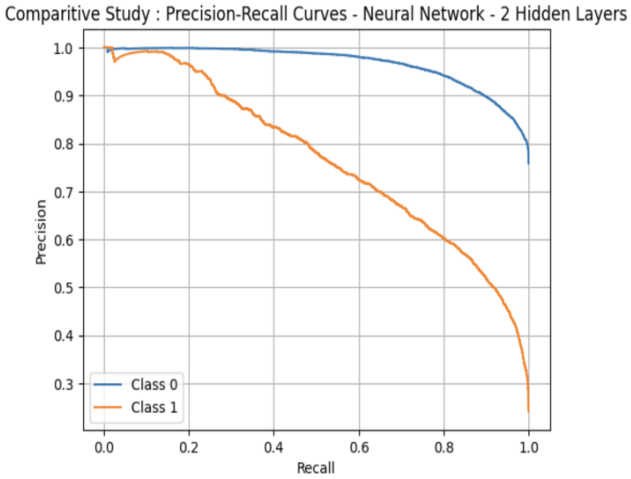
Class 0 (<=50k):
The model shows excellent precision for class 0, indicating that it accurately predicts the vast majority of occurrences falling under class 0. Additionally, a high recall indicates that the model successfully recognises the majority of instances of class 0. The F1-score of 0.90 reflects strong performance for class 0, demonstrating a good balance between precision and recall.

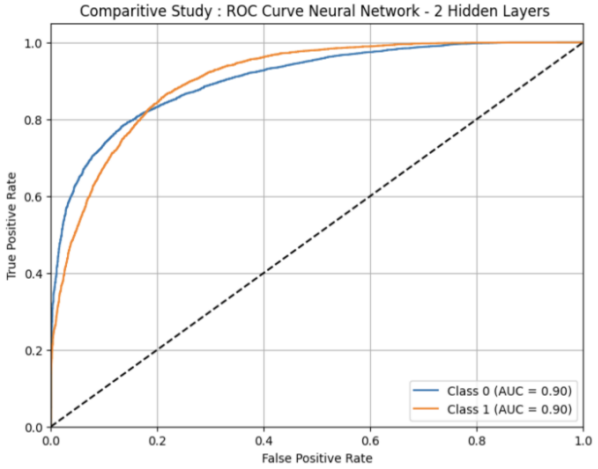
Class 1 (>50k):
The precision for class 1 is lower than for class 0, indicating that the model produces more false-positive predictions for class 1 than for class 0. The recall is also considerably lower, which indicates that the model misses a large proportion of occurrences that belong to class 1. The model's performance for class 1 is relatively worse than for class 0, according to the F1-score of 0.65.

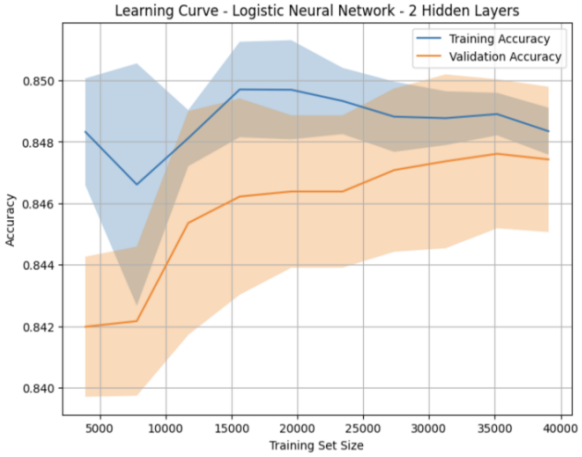
In conclusion, the model exhibits good class 0 performance, with excellent precision, recall, and F1-score. The performance for class 1 is, however, worse, with lower F1-score, recall, and precision. Reasons for the lower performance for class 1 has already been discussed in the preceding sections (Imbalanced Datasets).

An overall Accuracy of 85% was achieved. The other performance metrics are as follows.









Class 0's PR curve has consistently excellent precision. The curve is more pronounced in the upper right corner, indicating strong class 0 prediction precision and recall. Even if the precision decreases from 1 to roughly 0.8, it still remains strong as recall rises. This implies that the model maintains a reasonable level of precision while doing well in reliably detecting instances of class 0. From a precision of 1, the PR curve for class 1 briefly dips before swiftly rising again. This implies that while the model initially makes positive predictions with high accuracy, there may be a small number of false positives. However, the

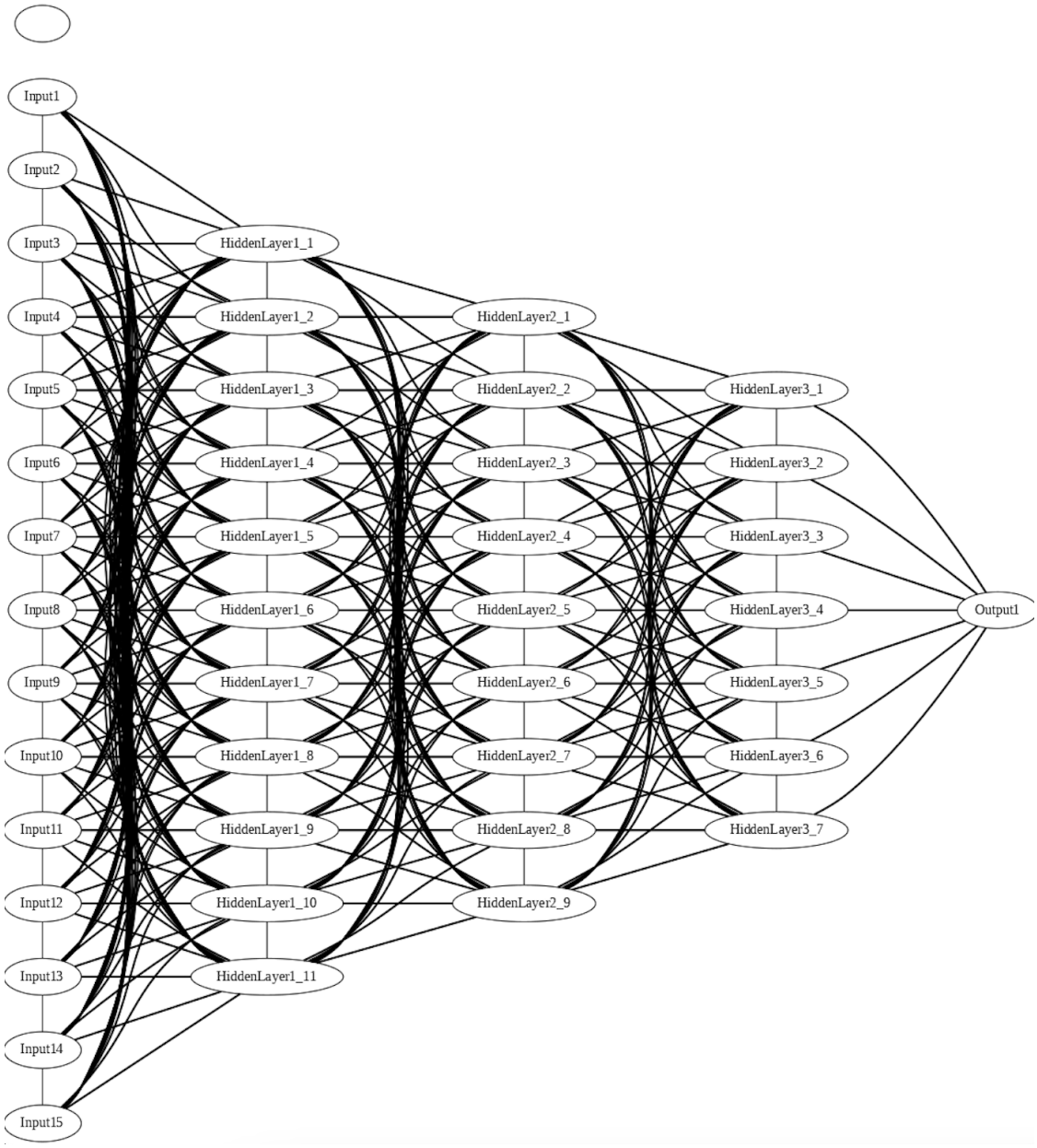
precision decreases noticeably to around 0.1 as the recall rises, demonstrating a sharp decline in precision for class 1 predictions. This suggests that as the recall threshold rises, the model has difficulty correctly identifying all instances of positive feedback.

Excellent performance of the model in terms of true positive rate and false positive rate is indicated by the ROC curves for both classes being more or less closer to the top left corner. The model appears to attain a high true positive rate while keeping a low false positive rate, based on its proximity to the top left corner. AUC (Area Under the Curve) values of 0.9 for both classes also show the model's strong discriminative ability. Better capacity to discriminate between positive and negative examples is indicated by an AUC value that is nearer 1.

As more training examples are employed, the training curve initially falls from 0.848 to 0.847, showing a minor drop in the model's performance on the training data. The curve then begins to rise once more, indicating that the model performs better as additional training data are added. The validation curve also shows an improvement in the model's performance on the validation data, rising from 0.842 to 0.847 at first. The validation curve similarly converges to a value that is comparable to the training curve as the amount of training data grows. The convergence shows that the model achieves a reasonable balance between generalisation and capturing the underlying patterns in the data, not overfitting nor underfitting the data.

c. Neural Network with 3 hidden layers (Size 11,9,7):

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report

# Create a Neural Network classifier with 3 hidden layers
cmpstd_neural_classifier_3 = MLPClassifier(hidden_layer_sizes=(11,9,7), random_state=42,activation='logistic',max_iter=10000)

# Train the classifier
cmpstd_neural_classifier_3.fit(x_combined_train_new, y_combined_train_new)
```



The performance report is as follows.

```
# Make predictions on the test set
cmpstd_neural_predictions_3 = cmpstd_neural_classifier_3.predict(x_combined_test_new)

# Evaluate performance
print(classification_report(y_combined_test_new, cmpstd_neural_predictions_3))
```

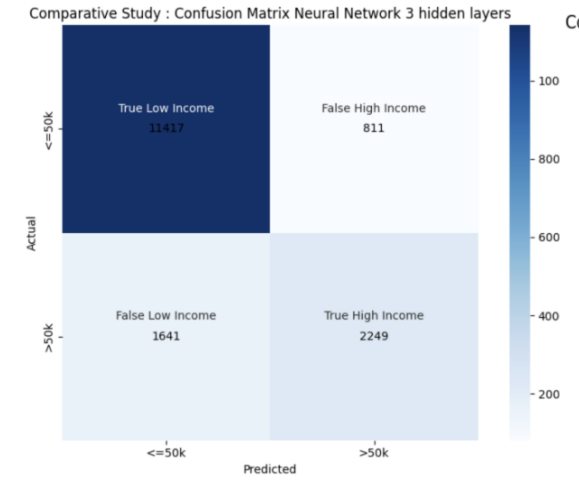
```
              precision    recall  f1-score   support

           0       0.87      0.93      0.90     12228
           1       0.73      0.58      0.65      3890

    accuracy                           0.85     16118
   macro avg       0.80      0.76      0.78     16118
weighted avg       0.84      0.85      0.84     16118
```

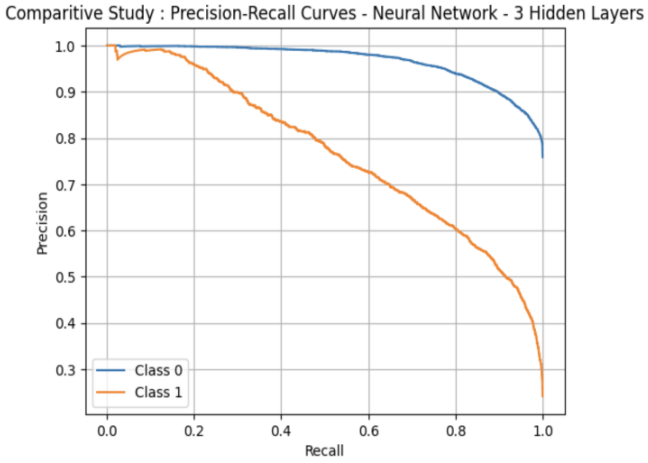
Class 0 (<=50k):
With a precision of 0.87, the model is able to correctly predict 87% of class 0 data. Recall is 0.93, meaning that 93% of examples in class 0 are correctly identified by the model. The F1-score is 0.90, which is a balanced indicator of recall and precision. This shows that the model is effective at recognising instances of class 0 accurately. 12,228 is the number of instances in class 0, which is the support.
Class 1 (>50k):
In class 1, the precision is 0.73, which indicates that 73% of the forecasts are accurate. The model appears to capture 58% of the examples that belong to class 1 according to the recall of 0.58. The F1-score for class 1 is 0.65, indicating that precision and recall are generally balanced. 3,890 people are in support of class 1.

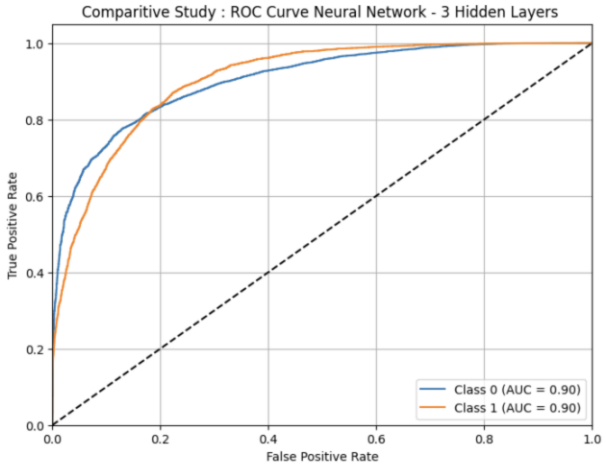
In conclusion, the model exhibits good class 0 performance, with excellent precision, recall, and F1-score. The performance for class 1 is, however, low, with lower F1-score, recall, and precision. Reasons for the lower

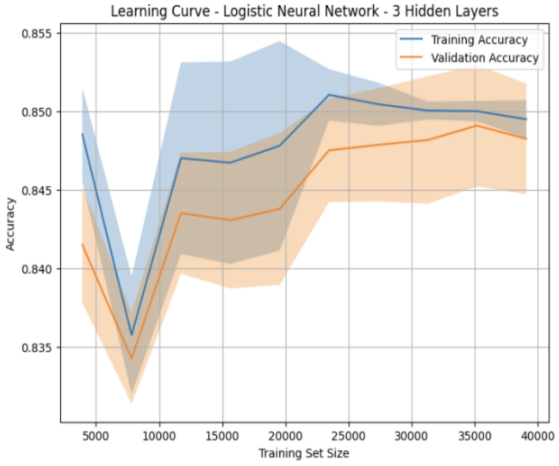
performance for class 1 has already been discussed in the preceding sections (Imbalanced Datasets).

An overall Accuracy of 85% was achieved. The other performance metrics are as follows.









The class 0 PR curve has exceptional precision over the whole curve. As the recall rises, the precision just slightly decreases from 1 to roughly 0.8, but it still remains high. This shows that despite keeping a reasonable high precision, the model works well in properly identifying negative instances (class 0). The curve's proximity to the top right corner suggests that class 0 predictions will do well. From a precision of 1, the PR curve for class 1 briefly

dips before swiftly rising again. This implies that while the model initially predicts positive examples with high precision, there may be a small number of false positives. However, as the recall rises, the precision considerably decreases to roughly 0.1, demonstrating a sharp decline in precision for class 1 predictions. This suggests that when the recall threshold rises, the model has difficulty correctly identifying all positive events.

The fact that the ROC curves for both classes are more or less closer to the top left corner indicates that the model performed exceptionally well in terms of both true positive rate (sensitivity) and false positive rate (1 - specificity). Because it is so close to the top left corner, it is likely that the model has a high true positive rate and a low false positive rate. Additionally, the model's great discriminative power is indicated by the AUC (Area Under the Curve) value of 0.9 for both classes. AUC values that are nearer 1 show a higher ability to tell good from negative events.

The learning curve study shows that as more training data is provided, the accuracy of the model first experiences some oscillations but gradually stabilises and improves. The training accuracy initially decreases, then rises abruptly, then rises continuously to a value of 0.85. Similar to the training accuracy, the validation accuracy exhibits a brief decrease, followed by a sudden increase, before eventually convergent with the training accuracy at roughly 0.85. This suggests a model that is well-fitted and achieves a balance between recognising patterns in the training data and extrapolating to new data.

4. **Decision tree Classifier**:

- In assignment 1 we had discussed in depth how to train a decision tree classifier and prune it.

```
#Use sklearn for decsion tree

#Importing the CLassifier
from sklearn.tree import DecisionTreeClassifier
new_default = DecisionTreeClassifier()
#.fit method to train the model
new_default.fit(comb_X_train,comb_y_train)
```

```
▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

- For pruning we had employed Reduced error cost complexity pruning.

- The performance report for the optimal pruned tree is as follows.

```
# making predictions
comb_y_predict_default = new_default.predict(x_combined_test_new)

# Printing classifier report after prediction
print(classification_report(y_combined_test_new,comb_y_predict_default))
```

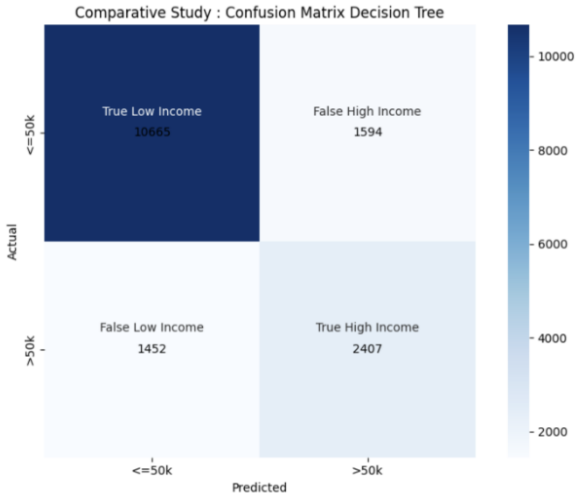
```
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     12259
           1       0.60      0.62      0.61      3859

    accuracy                           0.81     16118
   macro avg       0.74      0.75      0.74     16118
weighted avg       0.81      0.81      0.81     16118
```

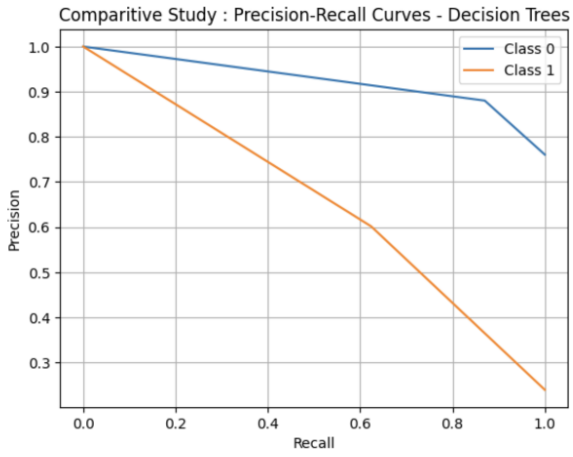
Class 0 (<=50k):
The model's accuracy, recall, and F1-score for class 0 were all 0.88. This shows that the model did a good job correctly

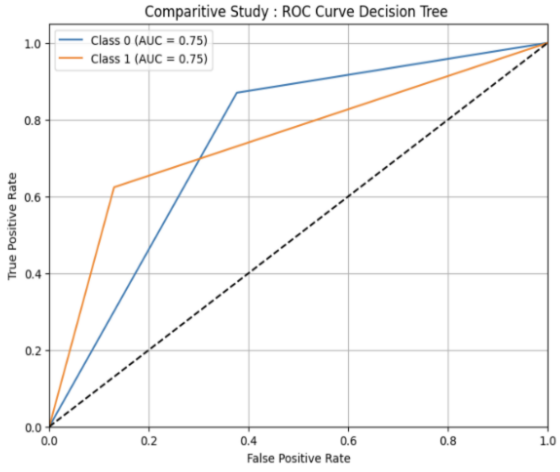
classifying examples as belonging to class 0, with a high level of recall and precision. The F1-score, which combines recall and precision, implies that the two criteria are well-balanced.

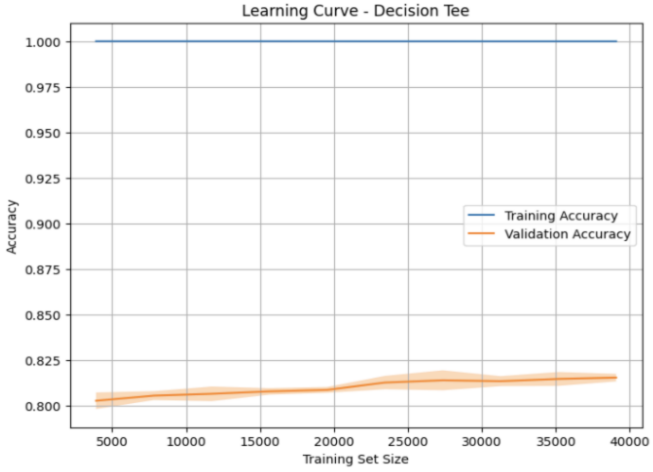
Class 1 (>50k):
In comparison to class 0, the model's performance for class 1 is noticeably worse. For class 1, the corresponding precision, recall, and F1-score values are 0.60, 0.62, and 0.61. This shows that class 1 instance prediction accuracy was more challenging for the model. The recall means that roughly 62% of the actual occurrences for class 1 were properly identified, while the precision suggests that roughly 60% of the projected instances for class 1 were true positives.

An overall accuracy of 81% was seen. The other performance metrics are as follows.









5. Random Forest Classifier:

- In assignment 1 we had discussed in depth how to train a Random Forest classifier.

```
#Importing the required libraries
from sklearn.ensemble import RandomForestClassifier

#Building the Classifier
random_forest = RandomForestClassifier(random_state = 50)
#Training
random_forest.fit(comb_X_train,comb_y_train)

#Predictions
y_randomforest_predictions = random_forest.predict(comb_X_test)
```

- Performance report is as follows.

```
from sklearn.metrics import f1_score
#Evaluation Metrics of the model Accuracy and fscore

# Evaluate performance
print(classification_report(y_combined_test_new, y_randomforest_predictions))
```

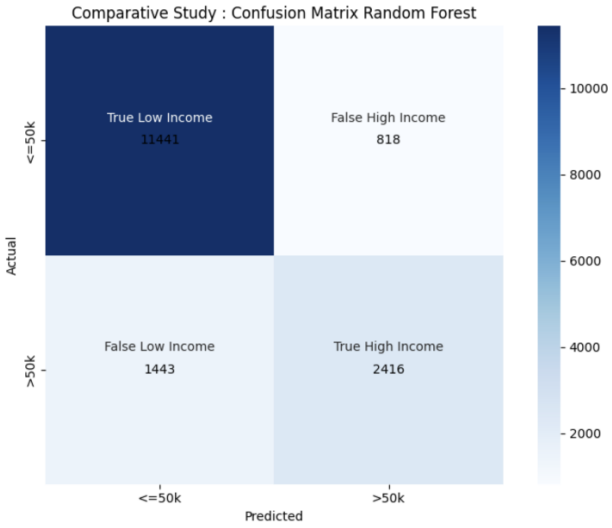
```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     12259
           1       0.75      0.63      0.68      3859

    accuracy                           0.86     16118
   macro avg       0.82      0.78      0.80     16118
weighted avg       0.85      0.86      0.86     16118
```

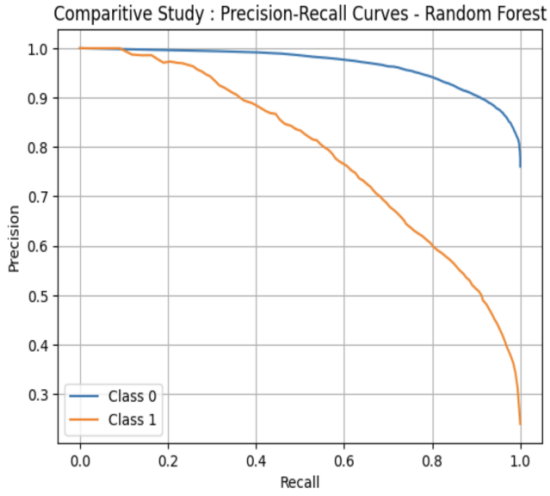
Class 0 (<=50k):
With a precision of 0.89, the model is able to correctly predict 89% of class 0 events. Recall is 0.93, meaning that 93% of examples in class 0 are correctly identified by the model. The precision and recall measures are balanced according to the F1-score of 0.91. This shows that the model is effective at recognising instances of class 0

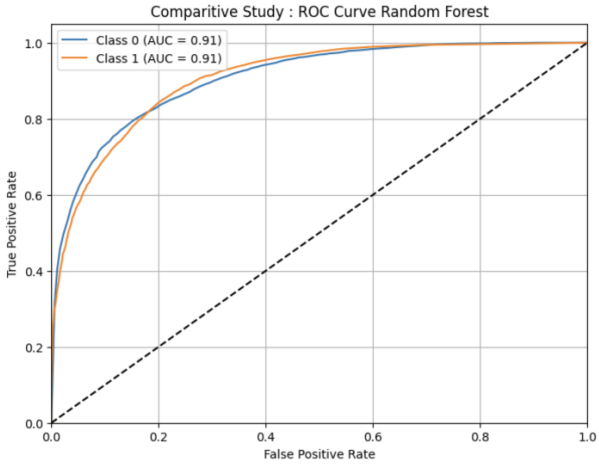
accurately. 12,259 instances of class 0 are supported, which is the size of that class.

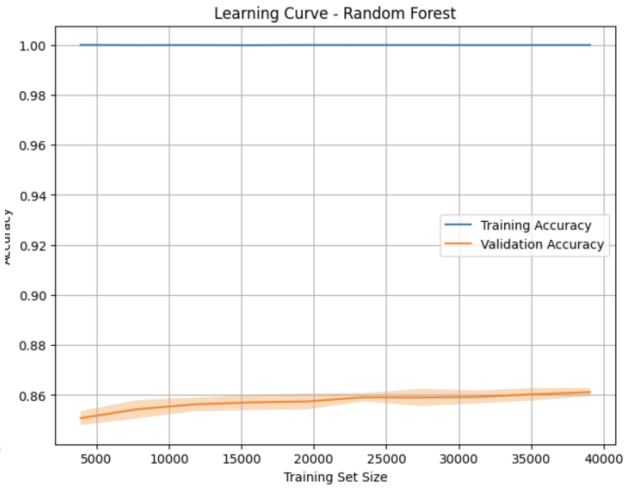
Class 1 (>50k):
For class 1, the precision is 0.75, which indicates that 75% of the forecasts are accurate. The model appears to capture 63% of the examples that belong to class 1 according to the recall of 0.63. The F1-score for class 1 is 0.68, indicating that precision and recall are generally balanced. 3,859 people are in support of class 1.

An overall Accuracy of 86% was achieved. The other performance metrics are as follows.



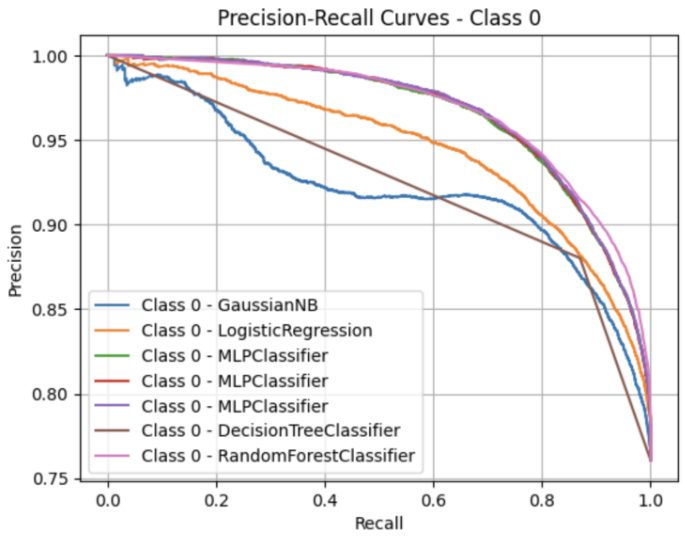


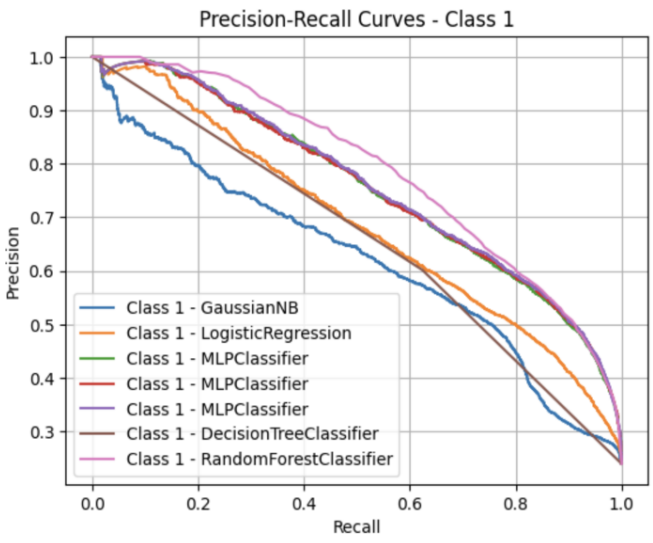


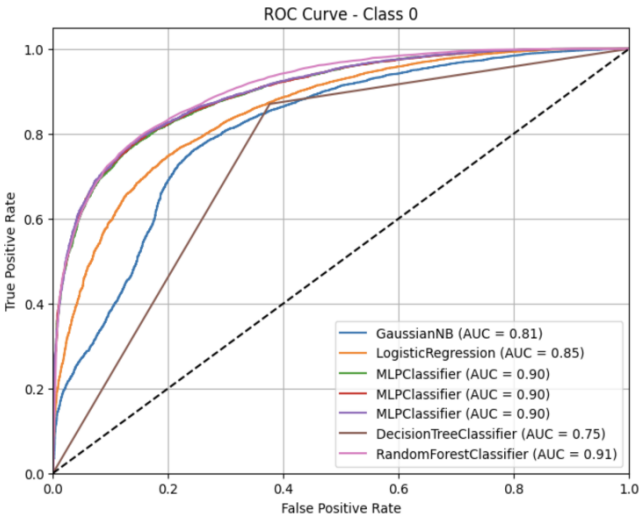


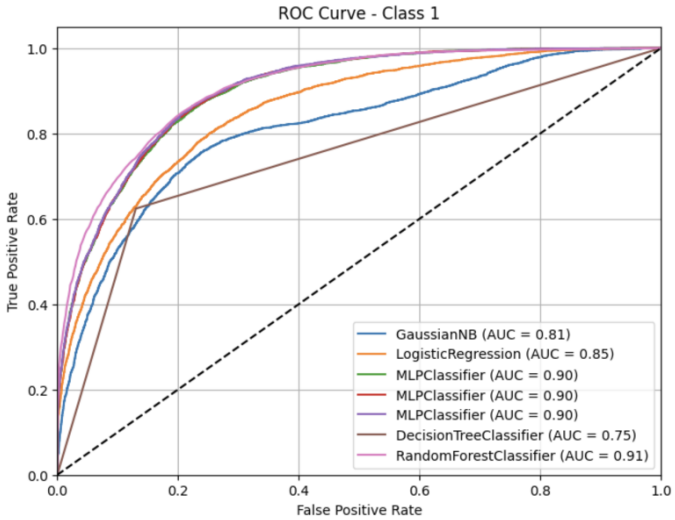
- Finally, for a better comparison, some metrics are provided below.

Model	Class	Precision	Recall	F1-Score	Support
Naive Bayes	0	0.84	0.93	0.88	12259
Naive Bayes	1	0.67	0.44	0.53	3859
Logistic Regression	0	0.85	0.94	0.89	12228
Logistic Regression	1	0.72	0.46	0.56	3890
Neural Network (1 HL)	0	0.88	0.93	0.90	12228
Neural Network (1 HL)	1	0.73	0.58	0.65	3890
Neural Network (2 HL)	0	0.87	0.93	0.90	12228
Neural Network (2 HL)	1	0.74	0.57	0.65	3890
Neural Network (3 HL)	0	0.87	0.93	0.90	12228
Neural Network (3 HL)	1	0.73	0.58	0.65	3890
Decision Tree	0	0.88	0.87	0.88	12259
Decision Tree	1	0.60	0.62	0.61	3859
Random Forest	0	0.89	0.93	0.91	12259
Random Forest	1	0.75	0.63	0.68	3859






ROC Curve - Class 0
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
GaussianNB (AUC = 0.81)
LogisticRegression (AUC = 0.85)
MLPClassifier (AUC = 0.90)
MLPClassifier (AUC = 0.90)
MLPClassifier (AUC = 0.90)
DecisionTreeClassifier (AUC = 0.75)
RandomForestClassifier (AUC = 0.91)


ROC Curve - Class 1
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
GaussianNB (AUC = 0.81)
LogisticRegression (AUC = 0.85)
MLPClassifier (AUC = 0.90)
MLPClassifier (AUC = 0.90)
MLPClassifier (AUC = 0.90)
DecisionTreeClassifier (AUC = 0.75)
RandomForestClassifier (AUC = 0.91)