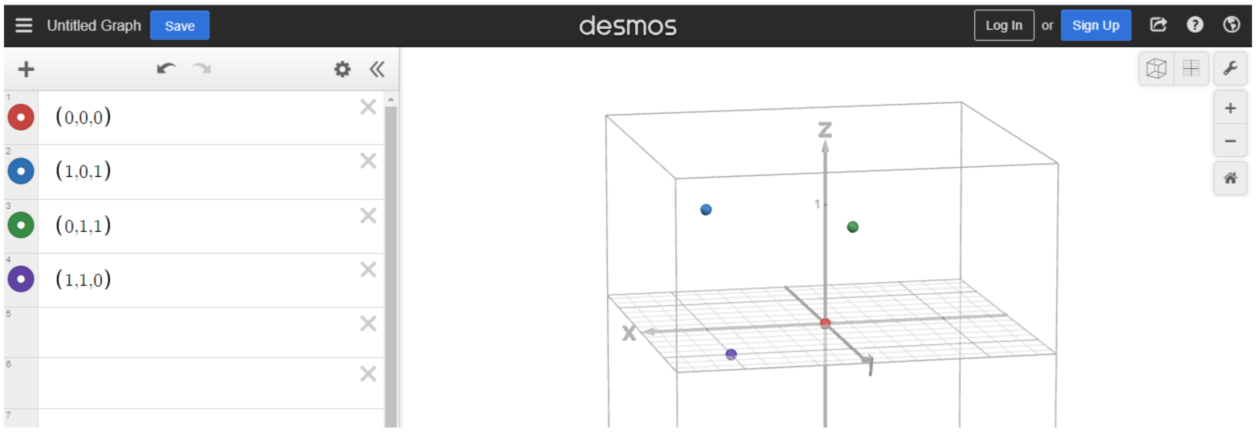
Q1.
My cooked up dataset

feature1	feature2	feature3	feature4	target
1	0	0	0	0
1	0	0	1	0
0	0	1	0	0
0	0	1	1	1
1	1	0	0	1
1	1	0	1	1
0	1	1	0	1
0	1	1	1	0

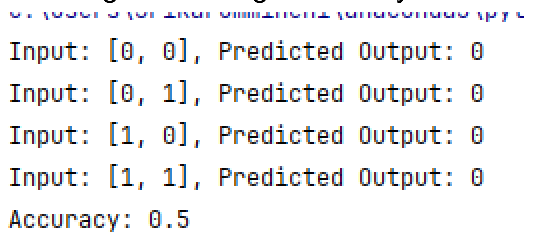
Feature 1 is conditionally dependent on feature2 and feature3 as we can see that when feature 1 and feature 2 have same value, the feature 1 has 0 and when the feature 1 and feature 2 have opposite values, feature 1 has a value of 1
And the accuracy of the naive bayes comes out to be 0.75 which is great as the target is predicted even when the features are conditionally dependent since it uses the probabilities assuming the values to be independent.
Naive Bayes can sometimes make accurate classifications even when the features are not strictly conditionally independent given the class. This apparent paradox occurs because Naive Bayes is surprisingly robust and can handle situations where the independence assumption is not met.
Naive Bayes is often robust to deviations from the independence assumption. While it assumes conditional independence, it doesn't necessarily require it to be strictly true. In many real-world scenarios, the actual dependencies between features are not strong enough to significantly impact the classifier's performance.
Sometimes, dependencies between features can offset each other. While one feature might influence the outcome, another feature might counteract that influence. In such cases, the assumption of independence might not be strongly violated.
The success of Naive Bayes also depends on the specific characteristics of the dataset. If the dataset is well-suited to the independence assumption or if the dependencies between features are not very strong, Naive Bayes can perform surprisingly well.
Smoothing techniques can be applied to the probability estimates in Naive Bayes to handle situations where the data is sparse or where some feature combinations may not

be present in the training data. These techniques can help mitigate the impact of dependencies to some extent.
Naive Bayes can be noise-tolerant, meaning it can handle noisy or imperfect data where some features may appear dependent but are influenced by other factors.
However, it's important to note that the performance of Naive Bayes can vary depending on the specific dataset and the degree to which the independence assumption is violated. In cases where the dependence between features is strong and crucial for accurate classification, Naive Bayes may not be the best choice, and more complex models that can capture these dependencies may be more appropriate

Q2.



Predicting XOR using Naive Bayes.

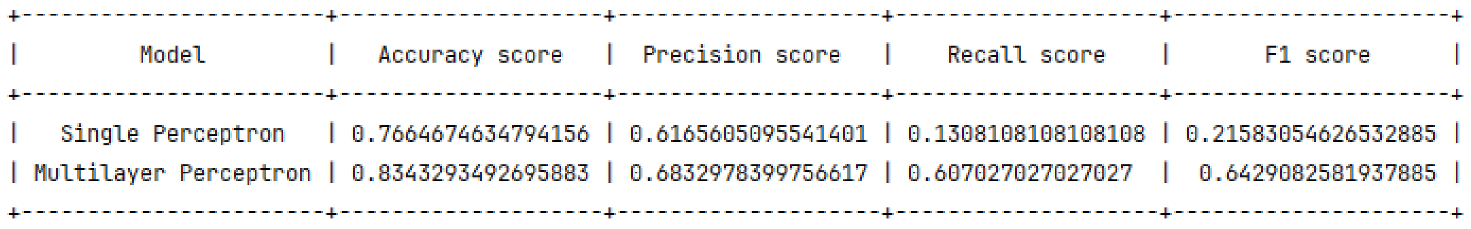


```
Input: [0, 0], Predicted Output: 0
Input: [0, 1], Predicted Output: 0
Input: [1, 0], Predicted Output: 0
Input: [1, 1], Predicted Output: 0
Accuracy: 0.5
```

The accuracy obtained above is equivalent to a random classifier where even if we assign classes randomly, the accuracy would still be 0.5.
The XOR problem is the most simple problem that is not linearly separable. You have two Boolean variables X and Y, and the target value you want to "predict" is the result from XORing the two variables. That is, only when either (but not the other) is 1, you want to predict 1 as outcome, and 0 otherwise.
As you can see, for the four "points" of my "plot" above (X horizontally, Y vertically; imagine the commas are the "points", if you like), there is no way you can draw a straight line that separates the two outcomes (the two 1s in the upper left and lower

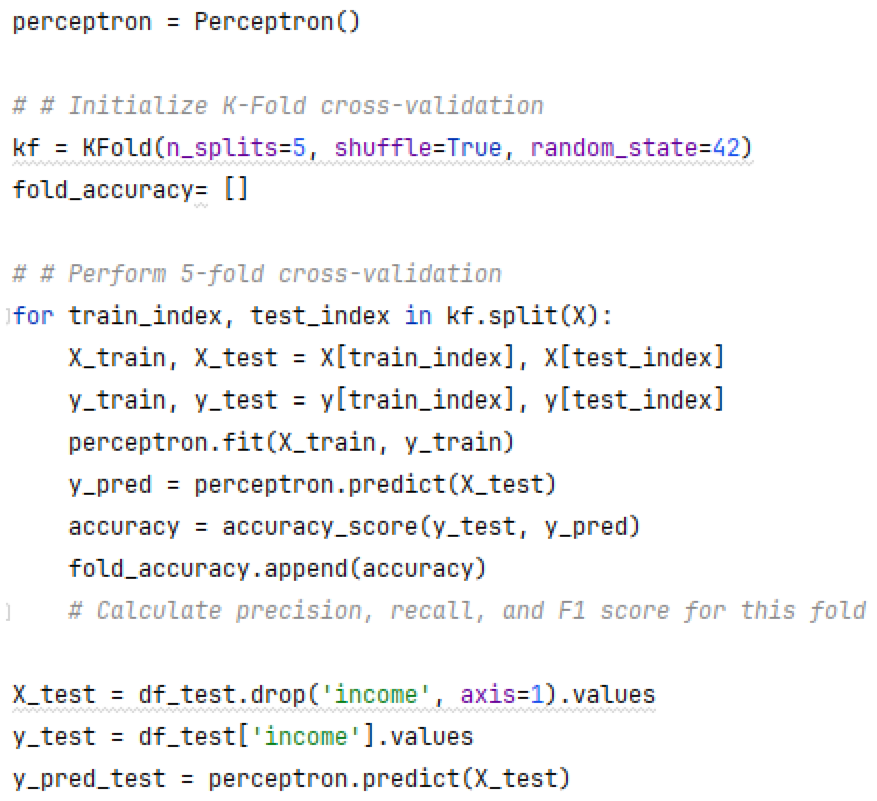
right, and the two 0s, also in opposing corners). So linear classifiers, which model the class separation using straight lines, cannot solve problems of this nature.
Now, as to Naive Bayes, it models independent events. Given only X and Y, it can model the distribution of xs and it can model the ys, but it does not model any relation between the two variables. That is, to model the XOR function, the classifier would have to observe both variables at the same time. Only making a prediction based on the state of X without taking into account Y's state (and vice versa) cannot lead to a proper solution for this problem. Hence, the Naive Bayes classifier is a linear classifier, too. Hence, deriving a relation between these 2 variables using Naive Bayes is not possible.
Naive Bayes is not well-suited for solving the XOR problem because it assumes conditional independence between features, which means it assumes that the presence or absence of one feature does not depend on the presence or absence of any other feature. In the XOR problem, the two input features are dependent on each other, and this dependency is crucial for accurately predicting the output.
To classify a new example, Naive Bayes calculates the conditional probability of each feature given the class label. In the XOR problem, since A and B are not independent, the probabilities of A and B given Y=1 will not be accurately modeled by Naive Bayes. The model would incorrectly assume that the features are independent, leading to incorrect predictions.
Naive Bayes uses these conditional probabilities to create a decision boundary to classify new examples. In the XOR problem, the decision boundary is nonlinear and cannot be captured effectively by a model that assumes independence between features. Naive Bayes is inherently limited in its ability to model complex, nonlinear decision boundaries.
Naive Bayes assumes that the features (A and B in this case) are conditionally independent given the class label (Y). In the XOR problem, A and B are not independent; they are highly dependent on each other. The outcome of Y depends on whether both A and B are 1 or not, and this dependence contradicts the independence assumption of Naive Bayes.

Q3.a.



```
+-----------------------+--------------------+--------------------+--------------------+---------------------+
|         Model         |   Accuracy score   |  Precision score   |    Recall score    |       F1 score      |
+-----------------------+--------------------+--------------------+--------------------+---------------------+
|   Single Perceptron   | 0.7664674634794156 | 0.6165605095541401 | 0.1308108108108108 | 0.21583054626532885 |
| Multilayer Perceptron | 0.8343293492695883 | 0.6832978399756617 | 0.607027027027027  |  0.6429082581937885 |
+-----------------------+--------------------+--------------------+--------------------+---------------------+
```

The scores using single perceptron and multilayer perceptron are given above in the table.
Multilayer perceptron performs well on the test set.

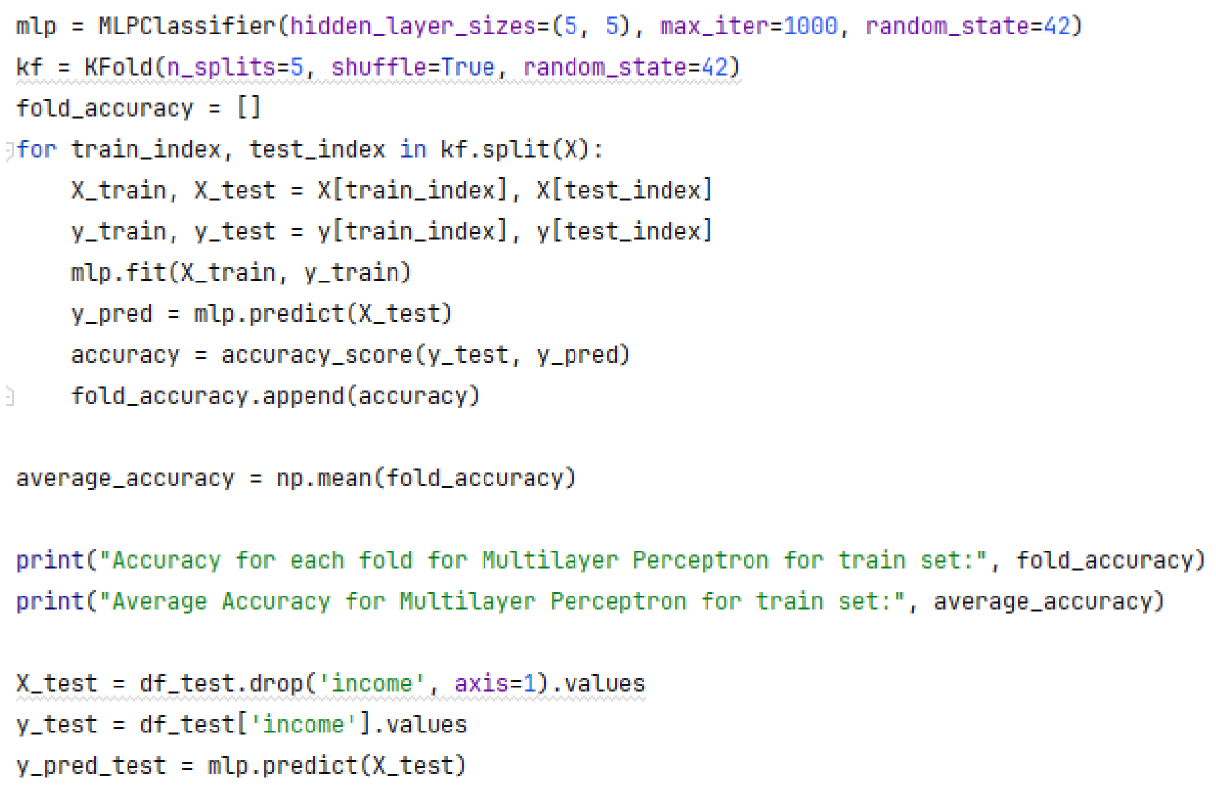


```
perceptron = Perceptron()

# # Initialize K-Fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracy= []

# # Perform 5-fold cross-validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    perceptron.fit(X_train, y_train)
    y_pred = perceptron.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    fold_accuracy.append(accuracy)
    # Calculate precision, recall, and F1 score for this fold

X_test = df_test.drop('income', axis=1).values
y_test = df_test['income'].values
y_pred_test = perceptron.predict(X_test)
```



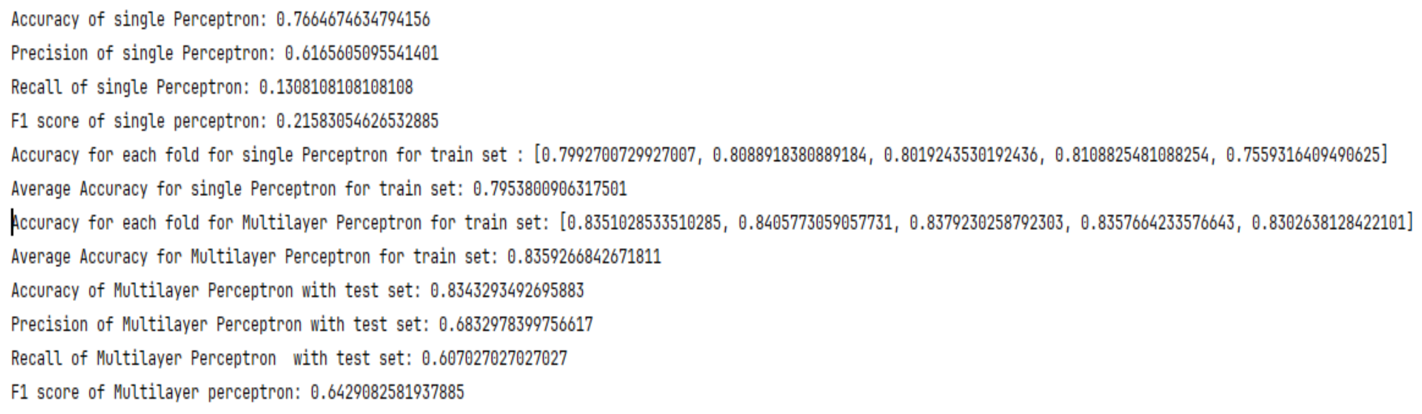
```python
mlp = MLPClassifier(hidden_layer_sizes=(5, 5), max_iter=1000, random_state=42)
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_accuracy = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    mlp.fit(X_train, y_train)
    y_pred = mlp.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    fold_accuracy.append(accuracy)

average_accuracy = np.mean(fold_accuracy)

print("Accuracy for each fold for Multilayer Perceptron for train set:", fold_accuracy)
print("Average Accuracy for Multilayer Perceptron for train set:", average_accuracy)

X_test = df_test.drop('income', axis=1).values
y_test = df_test['income'].values
y_pred_test = mlp.predict(X_test)
```

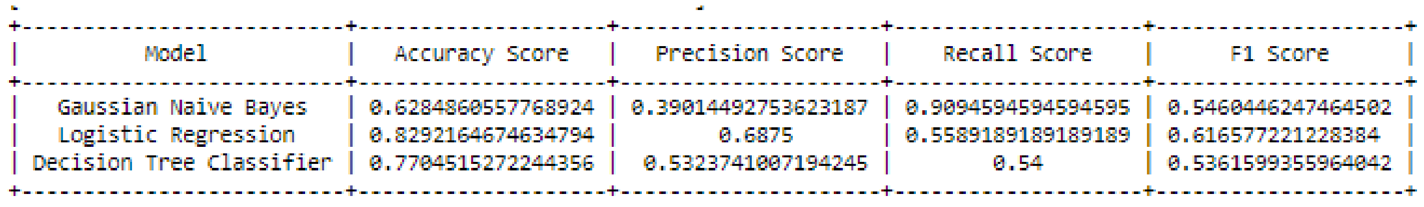
The implementation of the single and multilayer perceptron is given above and the scores are printed as given below,



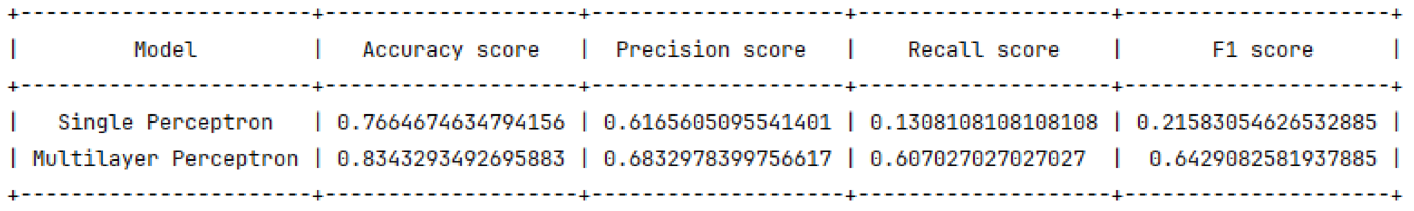
```
Accuracy of single Perceptron: 0.7664674634794156
Precision of single Perceptron: 0.6165605095541401
Recall of single Perceptron: 0.1308108108108108
F1 score of single perceptron: 0.21583054626532885
Accuracy for each fold for single Perceptron for train set : [0.7992700729927007, 0.8088918380889184, 0.8019243530192436, 0.8108825481088254, 0.7559316409490625]
Average Accuracy for single Perceptron for train set: 0.7953800906317501
Accuracy for each fold for Multilayer Perceptron for train set: [0.8351028533510285, 0.8405773059057731, 0.8379230258792303, 0.8357664233576643, 0.8302638128422101]
Average Accuracy for Multilayer Perceptron for train set: 0.8359266842671811
Accuracy of Multilayer Perceptron with test set: 0.8343293492695883
Precision of Multilayer Perceptron with test set: 0.6832978399756617
Recall of Multilayer Perceptron  with test set: 0.607027027027027
F1 score of Multilayer perceptron: 0.6429082581937885
```

The network architecture used in MLP is 5 layers of nets with 5 perceptrons in each layer which constitutes 25 perceptrons in hidden layers along with a single perceptron which classifies the observation depending on the value computed from the previous layers
The categorical features are encoded using the pd.get_dummies where each category becomes a feature in the dataset.
The numerical columns are standardized before training and also the test set is standardized before predicting on the model.

Q3.b



```
+---------------------------+--------------------+---------------------+--------------------+--------------------+
|           Model           |   Accuracy Score   |   Precision Score   |    Recall Score    |      F1 Score      |
+---------------------------+--------------------+---------------------+--------------------+--------------------+
|   Gaussian Naive Bayes    | 0.6284860557768924 | 0.39014492753623187 | 0.9094594594594595 | 0.5460446247464502 |
|    Logistic Regression    | 0.8292164674634794 |        0.6875       | 0.5589189189189189 | 0.616577221228384  |
| Decision Tree Classifier  | 0.7704515272244356 |  0.5323741007194245 |        0.54        | 0.5361599355964042 |
+---------------------------+--------------------+---------------------+--------------------+--------------------+
```



```
+-----------------------+--------------------+--------------------+--------------------+---------------------+
|         Model         |   Accuracy score   |  Precision score   |    Recall score    |       F1 score      |
+-----------------------+--------------------+--------------------+--------------------+---------------------+
|   Single Perceptron   | 0.7664674634794156 | 0.6165605095541401 | 0.1308108108108108 | 0.21583054626532885 |
| Multilayer Perceptron | 0.8343293492695883 | 0.6832978399756617 | 0.607027027027027  |  0.6429082581937885 |
+-----------------------+--------------------+--------------------+--------------------+---------------------+
```

Overall the accuracy of the Multilayer perceptron and Logistic regression is very similar and have the highest accuracy among the other models. But the f1 score seems to be low in both the models. The Recall score of Naive Bayes is the highest among all other models but this model could not get more accuracy.
In this case, Multilayer perceptron performed best among all the other models with very little difference with Logistic regression. Coming to precision, recall and f1 scores, the multilayer perceptron's scores are better.
The multilayer perceptron would have been helpful if the data was very huge (in GBs) and accuracy would have been more with more data and this model helps in any situation. Naive Bayes would be helpful when the dependence between features is low. If there is a strong correlation, as the naive bayes is itself a linear model, the model would not perform well. In this case, as there are only 2 classes, logistic regression also performed well.
Standardizing the features helped increase the accuracy as all the data points are brought to be centered at origin.
When the data points in both the classes are equal, the prediction would have been more accurate as there is no bias in data for both classes.
Single perceptron has decent accuracy but the f1 and recall scores are quite less and hence it is not recommended to use it for data having more features as the single perceptron cannot identify the patterns in the data and hence leads to poor classification.

The decision tree did not perform well but it was expected to perform well. The reason may be there are outliers in the data which affected the decision tree and hence its precision, recall and f1 score are quite near but not as good as the remaining models.