PHASE 3 Modeling, Evaluation, Tuning (20%)
Choose Machine Learning Frameworks:
Decide on the machine learning apps, frameworks and libraries that your group will use, such as Sagemaker, AutoML, TensorFlow, PyTorch, scikit-learn, etc..
Develop and Train Models:
Perform additional data preprocessing, model development, and training. Explore different algorithms and techniques to solve the problem.
Evaluation and Validation:
Evaluate and validate their machine learning models. Use appropriate metrics such as ROC curves, MSE, precision, recall, F1-score, and accuracy to measure model performance.
Hyperparameter Tuning:
If appropriate, fine-tune hyperparameters to optimize model performance. This can involve using AWS SageMaker's hyperparameter tuning capability.
Phase 3 Deliverable:
Project Repository on GitHub (Updated Table of Contents)
Deliverable 3 Document accessible in Github (include screenshots)

GitHub Repository
https://github.com/DelphineMeera/Predict-students-dropout-and-academic-success

MACHINE LEARNING FRAMEWORKS :

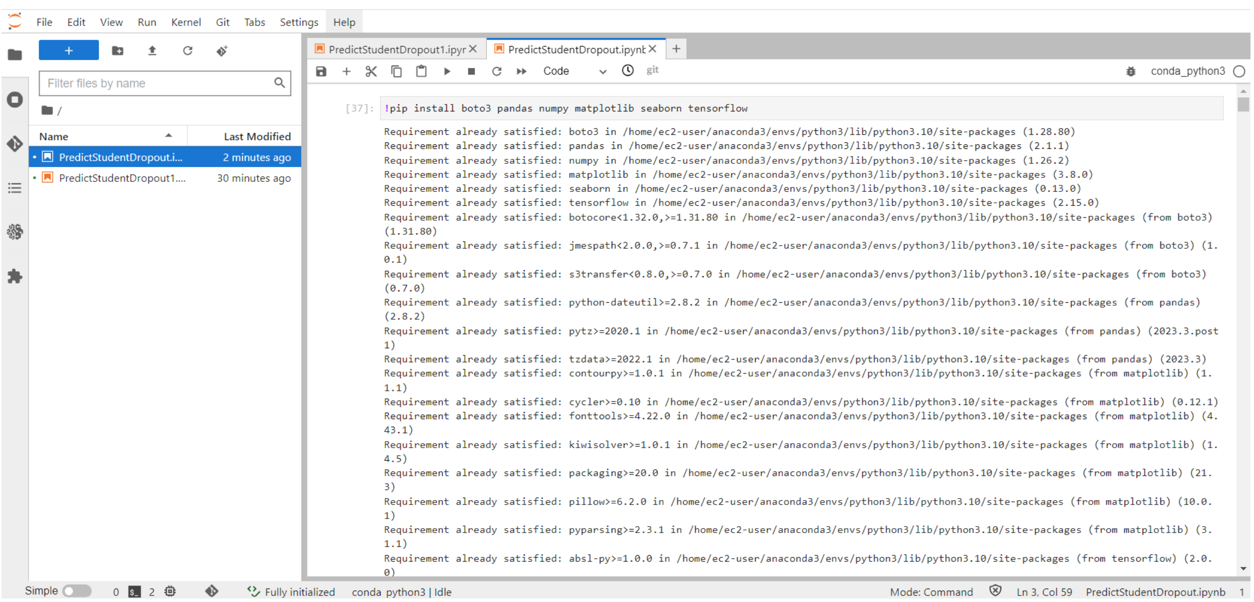
- Below are the list of tools used in this project
    - Amazon Sagemaker
    - Amazon S3 buckets
    - TensorFlow
    - Scikit-Learn

DEVELOP AND TRAIN MODELS :

- Data Pre-processing:
    - Ensured data cleanliness, accurate labeling, and representation of real-world scenarios
    - Encoded categorical variables

- Data Partitioning:
    - Split the data into training, testing, and validation sets
- Data Analysis:
    - Plotted a heatmap to understand the correlation between features
- Model Construction Techniques:
    - Neural Network model with Simple Dense Layers
    - Random Forest Classification model
    - Logistic Regression

- **Data Pre-processing**



PredictStudentDropout1.ipyr × | PredictStudentDropout.ipynb × | +

Code | git | conda_python3

[38]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[39]:
```python
students_df = pd.read_csv('s3://mys3bucketinput/FinalDataset.csv')
```

[40]:
```python
print(students_df.shape)
```

```
(4424, 35)
```

[41]:
```python
students_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 35 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Marital status                                  4424 non-null   int64
 1   Application mode                                4424 non-null   int64
 2   Application order                               4424 non-null   int64
 3   Course                                          4424 non-null   int64
 4   Previous qualification                          4424 non-null   int64
 5   Previous qualification (grade)                  4424 non-null   float64
 6   Nacionality                                     4424 non-null   int64
 7   Mother's qualification                          4424 non-null   int64
 8   Father's qualification                          4424 non-null   int64
 9   Mother's occupation                             4424 non-null   int64
 10  Father's occupation                             4424 non-null   int64
 11  Admission grade                                 4424 non-null   float64
 12  Displaced                                       4424 non-null   int64
 13  Educational special needs                       4424 non-null   int64
 14  Debtor                                          4424 non-null   int64
 15  Tuition fees up to date                         4424 non-null   int64
```

ialized conda_python3 | Idle   Mode: Command   Ln 1, Col 19   PredictStudentDropout.ipynb   1

PredictStudentDropout1.ipyr × | PredictStudentDropout.ipynb × | +

Code | git | conda_python3

[42]:
```python
students_df['Target'].value_counts()
```

[42]:
```
Target
Graduate     2209
Dropout      1421
Enrolled      794
Name: count, dtype: int64
```

[43]:
```python
target_mapper = {"Graduate":0,
              "Dropout":1,
              "Enrolled":2}
```

[44]:
```python
students_df['Target-Numeric'] = students_df["Target"].replace(target_mapper)
```

[45]:
```python
students_df['Target'] = students_df['Target-Numeric']
students_df.drop('Target-Numeric', axis=1, inplace=True)
```

[46]:
```python
students_df.head(10)
```

[46]:

	Marital status	Application mode	Application order	Course	Previous qualification	Previous qualification (grade)	Nacionality	Mother's qualification	Father's qualification	Mother's occupation	...	Curricular units 2nd sem (credited)	Curricular units 2nd sem (enrolled)	(e
0	1	17	5	171	1	122.0	1	19	12	5	...	0	0	
1	1	15	1	9254	1	160.0	1	1	3	3	...	0	6	
2	1	1	5	9070	1	122.0	1	37	37	9	...	0	6	
3	1	17	2	9773	1	122.0	1	38	37	5	...	0	6	
4	2	39	1	8014	1	100.0	1	37	38	9	...	0	6	

ialized conda_python3 | Idle   Mode: Command   Ln 1, Col 19   PredictStudentDropout.ipynb   1

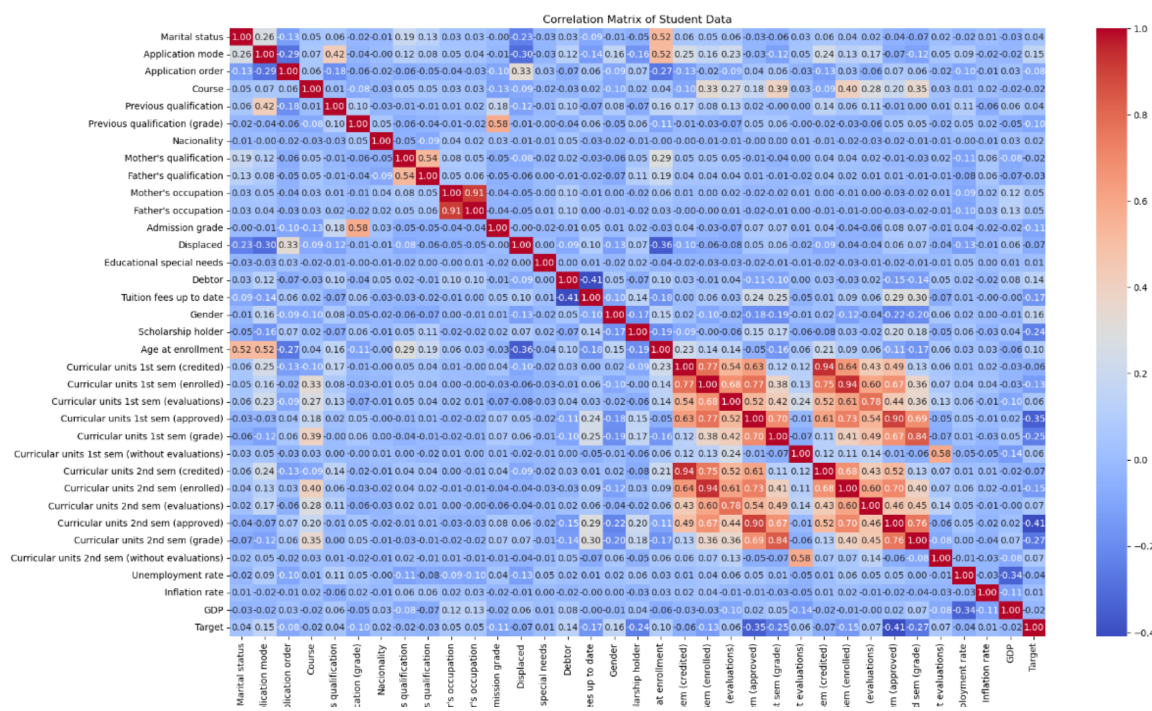
Code git conda_python3

	status	mode	order		qualification	(grade)		qualification	qualification	occupation		sem (credited)	(en
0	1	17	5	171	1	122.0	1	19	12	5	...	0	
1	1	15	1	9254	1	160.0	1	1	3	3	...	0	
2	1	1	5	9070	1	122.0	1	37	37	9	...	0	
3	1	17	2	9773	1	122.0	1	38	37	5	...	0	
4	2	39	1	8014	1	100.0	1	37	38	9	...	0	
5	2	39	1	9991	19	133.1	1	37	37	9	...	0	
6	1	1	1	9500	1	142.0	1	19	38	7	...	0	
7	1	18	4	9254	1	119.0	1	37	37	9	...	0	
8	1	1	3	9238	1	137.0	62	1	1	9	...	0	
9	1	1	1	9238	1	138.0	1	1	19	4	...	0	

10 rows × 35 columns

[47]:

```python
# Calculating the correlation matrix
corr_matrix = students_df.corr()

# Plotting the heatmap
plt.figure(figsize=(20, 12))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix of Student Data')
plt.show()
```



- **Neural Network model with Simple Dense Layers**

  - The data was split into 80% training, 10% validation, and 10% testing sets.
  - We used a 'Sequential' model – First Layer: Dense layer with 128 neurons, Second Layer: Dense layer with 64 neurons, Third Layer: Dense layer with 32 neurons, all using ReLU activation.
  - Output Layer is a Dense layer with 3 neurons (for 3 classes) using Softmax activation, which is suitable for multi-class classification.
  - Compiling the Model:
    - Optimizer: Stochastic Gradient Descent (SGD) with a learning rate of 0.01.
    - Loss Function: 'sparse_categorical_crossentropy', appropriate for classification with integer targets (labels).
    - We used the 'accuracy' metric, to evaluate the performance of the model during training and testing.
  - Training the Model:
    - Training Process: Model trained on scaled training data (`X_train_scaled`).
    - Validation Data: Uses scaled validation data (`X_val_scaled`, `y_val`) for evaluation during training.
    - Number of epochs on the training dataset is 30.
    - Batch Size: Number of samples processed before the model is updated, set to 32.

Neural Network model with Simple Dense Layers

[48]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler
```

[49]:
```
from tensorflow.keras.optimizers import SGD
```

[50]:
```
from sklearn.model_selection import train_test_split

# Splitting the dataset into 80% training and 20% (for further splitting into validation and test)
train_data, remaining_data = train_test_split(students_df, test_size=0.2, random_state=42)

# Splitting the remaining 20% equally into validation and test sets
validation_data, test_data = train_test_split(remaining_data, test_size=0.5, random_state=42)
```

[51]:
```
train_data.head(10)
train_data.shape
```

[51]: (3539, 35)

[52]:
```
validation_data.head(10)
validation_data.shape
```

[52]: (442, 35)

[53]:
```
test_data.head(10)
test_data.shape
```

[53]: (443, 35)

[54]:
```
# Separating the features and target variable
X_train = train_data.drop('Target', axis=1)
y_train = train_data['Target']
X_val = validation_data.drop('Target', axis=1)
y_val = validation_data['Target']
X_test = test_data.drop('Target', axis=1)
y_test = test_data['Target']
```

[55]:
```
# Scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.fit_transform(X_val)
X_test_scaled = scaler.fit_transform(X_test)
```

[56]:
```
# 2. Creating the Model
model = Sequential([
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax')  # 3 units for 3 classes
])
```

```
2023-12-03 20:33:44.123904: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:274] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device is detected
```

[57]:
```
# 3. Compiling the Model
model.compile(optimizer=SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',  # Suitable for integer targets
              metrics=['accuracy'])
```

[58]:
```
# 4. Training the Model
history = model.fit(X_train_scaled, y_train, validation_data=(X_val_scaled, y_val), epochs=30, batch_size=32)
```

```
Epoch 23/30
111/111 [==============================] - 0s 3ms/step - loss: 0.4815 - accuracy: 0.8079 - val_loss: 0.5678 - val_accuracy: 0.7783
Epoch 24/30
111/111 [==============================] - 0s 2ms/step - loss: 0.4771 - accuracy: 0.8081 - val_loss: 0.5612 - val_accuracy: 0.7738
Epoch 25/30
111/111 [==============================] - 0s 3ms/step - loss: 0.4729 - accuracy: 0.8112 - val_loss: 0.5664 - val_accuracy: 0.7715
Epoch 26/30
111/111 [==============================] - 0s 2ms/step - loss: 0.4685 - accuracy: 0.8146 - val_loss: 0.5726 - val_accuracy: 0.7760
Epoch 27/30
111/111 [==============================] - 0s 3ms/step - loss: 0.4650 - accuracy: 0.8124 - val_loss: 0.5767 - val_accuracy: 0.7738
Epoch 28/30
111/111 [==============================] - 0s 3ms/step - loss: 0.4611 - accuracy: 0.8163 - val_loss: 0.5655 - val_accuracy: 0.7851
Epoch 29/30
111/111 [==============================] - 0s 2ms/step - loss: 0.4560 - accuracy: 0.8192 - val_loss: 0.5669 - val_accuracy: 0.7828
Epoch 30/30
111/111 [==============================] - 0s 4ms/step - loss: 0.4531 - accuracy: 0.8166 - val_loss: 0.5715 - val_accuracy: 0.7828
```

```
[59]: # Extracting the history of training and validation loss and accuracy
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
training_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
```

```
[60]: epochs = range(1, len(training_loss) + 1)
```

```
[61]: plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, training_loss, 'bo', label='Training Loss')
plt.plot(epochs, validation_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

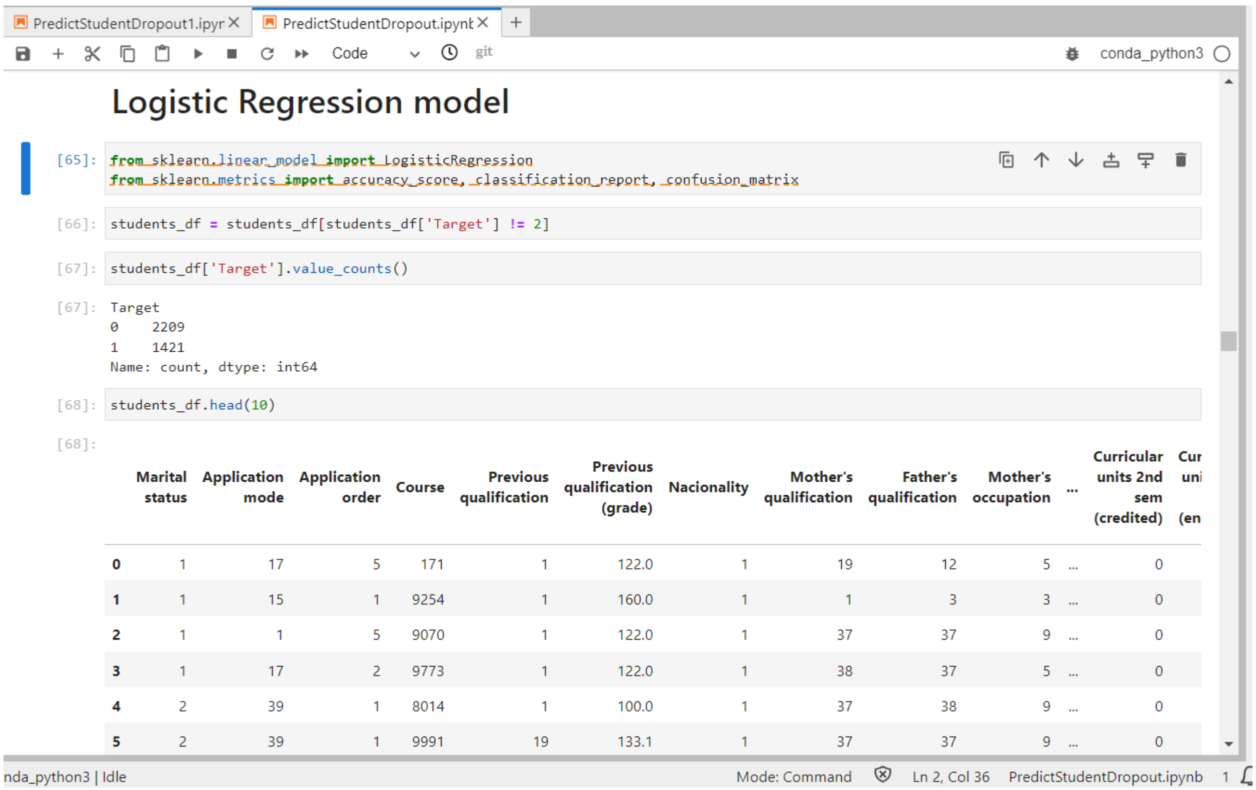
nda_python3 | Idle Mode: Command Ln 3, Col 42 PredictStudentDropout.ipynb 1

Logistic Regression

- ➢ Initially, we have taken the dataset into a dataframe and then started with the preprocessing for the logistic regression model.
- ➢ We have divided the dataset into two parts, training and validation sets of sizes 80 and 20 percent respectively.
- ➢ Later, we have dropped the rows which belong to the enrolled target variable ,which we are not going to use for our model to predict if a student drops out or not.
- ➢ Then, we have applied feature extraction over all the features by selecting a few out of the bunch given to us from the dataset.
- ➢ From the selected features, which were selected by considering the correlation matrix of all the features as well as the target variable, we convert the nominal data given in an ordinal format into one hot encoding.
- ➢ This adds additional columns to the model which converts a feature ex. Country to multiple columns of countries signifying 1 and 0 for a student to coming from their country and 0 for not (Course feature as well).

➢ Then we apply a scalar transformation function over the dataset to flat out the values of all the features to represent a proportionate amount in the values.

➢ Now, we will fit the model over the training data and test it over the test set about which we are going to discuss in the evaluation section.



conda_python3

```
# scaler = MinMaxScaler()

# students_df['Age'] = scaler.fit_transform(students_df[['Age']])
y = students_df['Target']
```

[104]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=72)
```

[105]:
```
model = LogisticRegression(max_iter=1000)  # Increase the number of iterations as needed


# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

[106]:
```
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
```

[69]:
```
X = students_df
X = X.drop('Target', axis=1)
y = students_df['Target']
```

[70]:
```
X.shape
```

[70]:
```
(3630, 34)
```

[71]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the features
X = scaler.fit_transform(X)
```

[72]:
```
students_df.columns
```

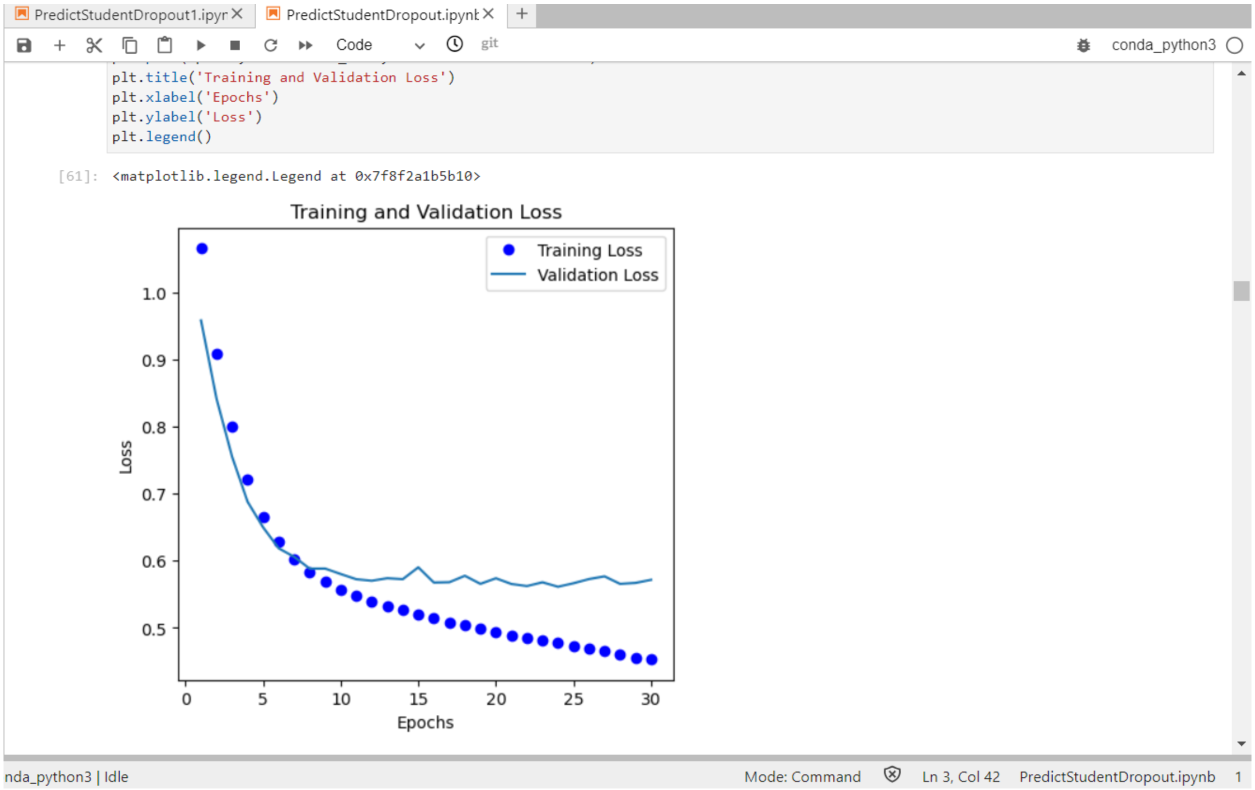
[72]:
```
Index(['Marital status', 'Application mode', 'Application order', 'Course',
       'Previous qualification', 'Previous qualification (grade)',
       'Nacionality', 'Mother's qualification', 'Father's qualification',
       'Mother's occupation', 'Father's occupation', 'Admission grade',
       'Displaced', 'Educational special needs', 'Debtor',
       'Tuition fees up to date', 'Gender', 'Scholarship holder',
       'Age at enrollment', 'Curricular units 1st sem (credited)',
       'Curricular units 1st sem (enrolled)',
       'Curricular units 1st sem (evaluations)',
       'Curricular units 1st sem (approved)',
       'Curricular units 1st sem (grade)',
       'Curricular units 1st sem (without evaluations)',
       'Curricular units 2nd sem (credited)',
       'Curricular units 2nd sem (enrolled)',
       'Curricular units 2nd sem (evaluations)',
       'Curricular units 2nd sem (approved)',
       'Curricular units 2nd sem (grade)',
```

Code git conda_python3

```
[73]: students_df['Age at enrollment'][0]

[73]: 20

[74]: features = ['Marital status','Course','Debtor','Nacionality',
                  'Previous qualification (grade)',
                  'Tuition fees up to date',
                  'Scholarship holder',
                  'Gender','Admission grade',
                  'Age at enrollment',
                  'Educational special needs',
                  'Unemployment rate',
                  'Curricular units 1st sem (credited)',
                  'Curricular units 1st sem (enrolled)',
                  'Curricular units 1st sem (evaluations)',
                  'Curricular units 1st sem (approved)',
                  'Curricular units 1st sem (grade)',
                  'Curricular units 1st sem (without evaluations)',
                  'Curricular units 2nd sem (credited)',
                  'Curricular units 2nd sem (enrolled)',
                  'Curricular units 2nd sem (evaluations)',
                  'Curricular units 2nd sem (approved)',
                  'Curricular units 2nd sem (grade)',
                  'Curricular units 2nd sem (without evaluations)',
                  'Inflation rate', 'GDP'
                  ]


      X = students_df[features]
      X=pd.get_dummies(X, columns=['Course'], prefix='Course')
      X=pd.get_dummies(X, columns=['Nacionality'], prefix='Nacionality')
      X=pd.get_dummies(X, columns=['Marital status'], prefix='Marital status')

      # scaler = MinMaxScaler()
```

EVALUATION AND VALIDATION :

- **Neural Network model with Simple Dense Layers**
  - Training and Validation Loss:
    - The training loss decreases sharply and then plateaus, showing the model is learning effectively from the training data.



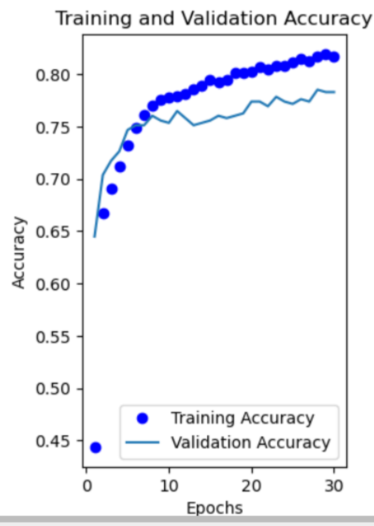
- Training and Validation Accuracy:
  - The training accuracy increases consistently, indicating learning over time.
  - The validation accuracy also increases reaching **78.28%**, but with some fluctuations, suggesting that the model may not generalize as well to new data.

PredictStudentDropout1.ipynb | PredictStudentDropout.ipynb

Code | git | conda_python3

```
plt.subplot(1, 2, 2)
plt.plot(epochs, training_accuracy, 'bo', label='Training Accuracy')
plt.plot(epochs, validation_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
[62]: <matplotlib.legend.Legend at 0x7f8f29a6edd0>
```



onda_python3 | Idle   Mode: Command   Ln 3, Col 42   PredictStudentDropout.ipynb   1

PredictStudentDropout1.ipynb | PredictStudentDropout.ipynb

Code | git | conda_python3

```
[63]: # Evaluating the model on the validation set
val_loss, val_accuracy = model.evaluate(X_val_scaled, y_val)

# Printing the accuracy
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

```
14/14 [==============================] - 0s 2ms/step - loss: 0.5715 - accuracy: 0.7828
Validation Accuracy: 78.28%
```

```
[64]: y_test_pred = model.predict(X_test_scaled)

# Evaluating the model on the validation set
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)

# Printing the accuracy
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
14/14 [==============================] - 0s 1ms/step
14/14 [==============================] - 0s 2ms/step - loss: 0.6657 - accuracy: 0.7381
Test Accuracy: 73.81%
```

- On testing dataset, model achieves an accuracy of **73.81%**

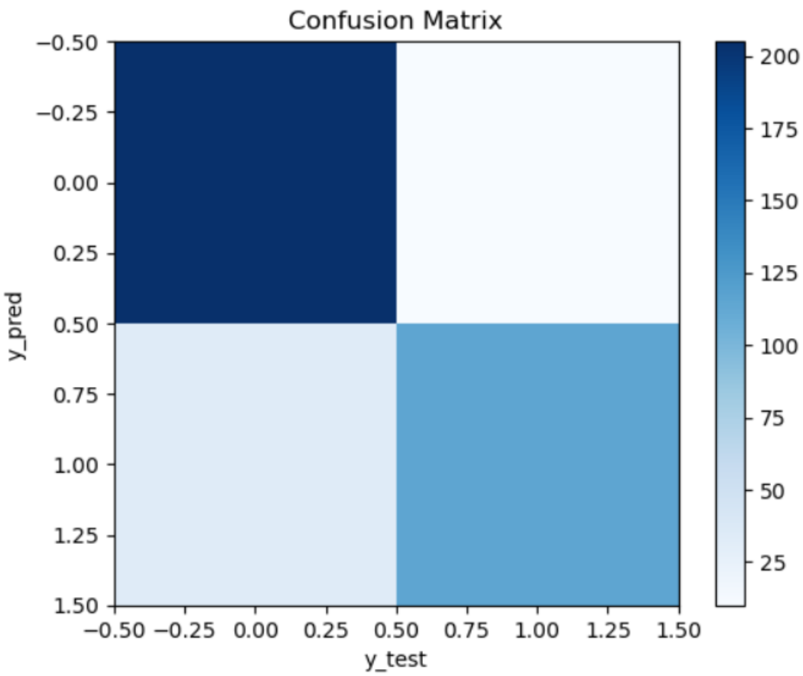
❖ Logistic regression model

- ➢ The logistic regression model initially performed for an accuracy **88.1%** which needed hyperparameter tuning to increase the accuracy.
- ➢ Here, the confusion matrix has been visualized in the pictures below.
- ➢ Along with it, precision, recall f1-scores of the base model have been put as well.
- ➢ The number of observations belonging to the classes True positive,True negative, False positive as well as False negative have been listed.
- ➢ The ROC-curve has been visualized as well in the pictures below.
- ➢ We have obtained an ROC- curve area of **0.87**.
- ➢ From the confusion matrix heatmap, we can see that the majority of the observations have been correctly classified in the True positive and the True negative classes. With other two classes having significantly less observations, i.e, False positive as well as false negative classes.

[106]:
```
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{classification_rep}")
```

```
Accuracy: 0.8815426997245179
Confusion Matrix:
[[205  10]
 [ 33 115]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.95      0.91       215
           1       0.92      0.78      0.84       148

    accuracy                           0.88       363
   macro avg       0.89      0.87      0.87       363
weighted avg       0.89      0.88      0.88       363
```

[107]:
```
plt.imshow(conf_matrix, cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.colorbar()
plt.show()
```



[108]:
```
# Extract TP, FP, TN, FN from the confusion matrix
TN = conf_matrix[0][0]
FP = conf_matrix[0][1]
FN = conf_matrix[1][0]
TP = conf_matrix[1][1]

# Print the values
print(f"True Positives (TP): {TP}")
print(f"False Positives (FP): {FP}")
print(f"True Negatives (TN): {TN}")
print(f"False Negatives (FN): {FN}")
```

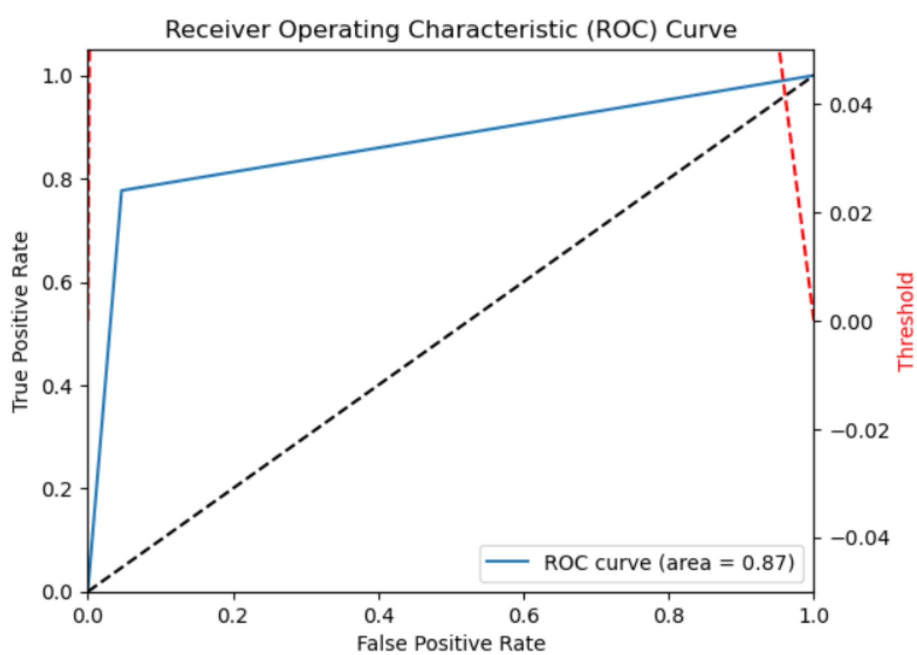
```
True Positives (TP): 115
False Positives (FP): 10
True Negatives (TN): 205
False Negatives (FN): 33
```

[109]:
```
from sklearn.metrics import roc_auc_score

# Assuming test_labels are the true labels and y_pred is the predicted labels
test_labels = y_test
target_predicted = y_pred

# Calculate and print the Validation AUC
validation_auc = roc_auc_score(y_test, y_pred)
print("Validation AUC:", validation_auc)
```

```
Validation AUC: 0.8652576995600251
```

[110]:
```
from sklearn.metrics import roc_curve, auc

# Assuming test_labels are the true labels and y_pred is the predicted labels
test_labels = y_test
```

[110]:
```
from sklearn.metrics import roc_curve, auc

# Assuming test_labels are the true labels and y_pred is the predicted labels
test_labels = y_test
target_predicted = y_pred

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(test_labels, target_predicted, pos_label=1)
roc_auc = auc(fpr, tpr)
thresholds = np.nan_to_num(thresholds, nan=0, posinf=0, neginf=0)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")

# Create the axis of thresholds (scores)
ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r')
ax2.set_ylabel('Threshold', color='r')
ax2.set_ylim([thresholds[-1], thresholds[0]])
ax2.set_xlim([fpr[0], fpr[-1]])

plt.show()
```



HYPERPARAMETER TUNING :

- **Random Forest Classifier:**

  - The Random Forest model is used to perform classification on the dataset which requires hyperparameter tuning to get higher accuracy.
  - The tuning is done using RandomizedSearchCV because it is a randomized search over a specified parameter distribution which works better for this model.
  - After performing hyperparameter tuning the accuracy increased to **91.05%.**

HyperParameter Tuning

Random Forest Classisfier

[111]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
```

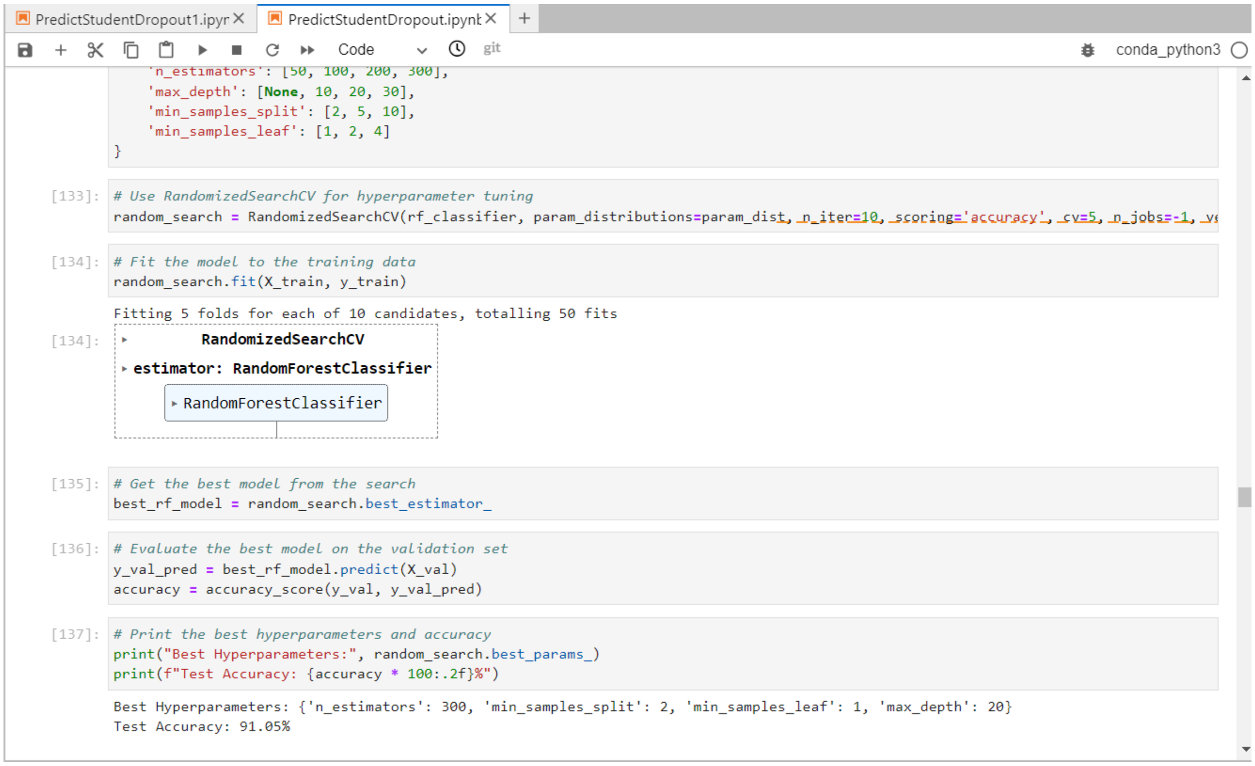
[112]:
```python
# Assuming the target variable is named 'Target'
X = students_df.drop('Target', axis=1)
y = students_df['Target']
```

[130]:
```python
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=32)
```

[131]:
```python
# Define the Random Forest classifier
rf_classifier = RandomForestClassifier()
```

[132]:
```python
# Define hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[133]:
```python
# Use RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(rf_classifier, param_distributions=param_dist, n_iter=10, scoring='accuracy', cv=5, n_jobs=-1, ve
```



PredictStudentDropout1.ipyr × PredictStudentDropout.ipynb ×

Code git conda_python3

```
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[133]:
```
# Use RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(rf_classifier, param_distributions=param_dist, n_iter=10, scoring='accuracy', cv=5, n_jobs=-1, ve
```

[134]:
```
# Fit the model to the training data
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[134]:
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

[135]:
```
# Get the best model from the search
best_rf_model = random_search.best_estimator_
```

[136]:
```
# Evaluate the best model on the validation set
y_val_pred = best_rf_model.predict(X_val)
accuracy = accuracy_score(y_val, y_val_pred)
```

[137]:
```
# Print the best hyperparameters and accuracy
print("Best Hyperparameters:", random_search.best_params_)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Best Hyperparameters: {'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 20}
Test Accuracy: 91.05%
```

❖ **Logistic regression model:**

- ➢ The logistic regression model initially performed for an accuracy **88.1%** which needed hyperparameter tuning to increase the accuracy.
- ➢ The GridSearchCV from AWS is used for logistic regression because it performs an exhaustive search over all possible combinations of hyperparameters to find the best combination.
- ➢ After performing hyperparameter tuning the accuracy increased to **91.71%**.
- ➢ Below all the steps taken to achieve this have been shown below.

Logistic Regression

[138]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

[139]:
```python
# Assuming the target variable is named 'Target'
X = students_df.drop('Target', axis=1)
y = students_df['Target']
```

[140]:
```python
# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=32)
```

[141]:
```python
# Define the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

[141]:
```
▾ LogisticRegression
LogisticRegression()
```

nda_python3 | Idle   Mode: Command   Ln 4, Col 34   PredictStudentDropout.ipynb   1

[142]:
```python
# Define the hyperparameter grid for GridSearchCV
param_grid = {
    'logistic__penalty': ['l1', 'l2'],
    'logistic__C': [0.001, 0.01, 0.1, 1, 10, 100],
    'logistic__fit_intercept': [True, False],
    'logistic__solver': ['liblinear', 'saga']
}
```

[143]:
```python
# Define the pipeline
pipeline = Pipeline(steps=[('logistic', model)])
```

[145]:
```python
# Perform grid search cross-validation
grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```

```
[146]: # Get the best model from the search
       best_model = grid_search.best_estimator_

[147]: # Evaluate the best model on the validation set
       y_val_pred = model.predict(X_val)
       accuracy = accuracy_score(y_val, y_val_pred)
       print(f"Accuracy before hyperparameter tuning: {accuracy * 100:.2f}%")

       Accuracy before hyperparameter tuning: 84.44%

[148]: # Fit the best model to the entire dataset
       best_model.fit(X, y)

[148]: ▸ Pipeline
       ▸ LogisticRegression

[153]: # Make predictions on the entire dataset
       y_pred = best_model.predict(X)

[155]: # Evaluate the best model on the entire dataset
       accuracy = accuracy_score(y, y_pred)
       print(f"Accuracy after hyperparameter tuning: {accuracy * 100:.2f}%")

       Accuracy after hyperparameter tuning: 91.71%
```

CONCLUSION :

As we explored different approaches to our project like using Simple Dense layers and Logistic regression, Logistic regression got more accuracy of **88.1%**.After hyperparameter tuning it increased to **91%**. Logistic regression is computationally less intensive compared to training neural networks, especially deep neural networks. If we have limited resources, logistic regression may be a more practical choice.

Submitted by Group14:

Ramit Aditya
Anirudh Cheruvu
Mounisha Bolla
Amirthavarshini Dhanavel
Delphine Antony Muthu