In [ ]:
```
#Razat Siwakoti (A00046635)
#DMV302 - Assessment 2
#ValidateNN.ipynb created on Jupyter notebook


#source: Scikit-learn(2015)
#https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.

#Deepika S. (2019)
#https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn
```

In [23]:
```
#importing necessary libraries
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [24]:
```
# Load the dataset
df = pd.read_csv("AtRiskStudentsTest.csv")
df.head()
```

Out[24]:

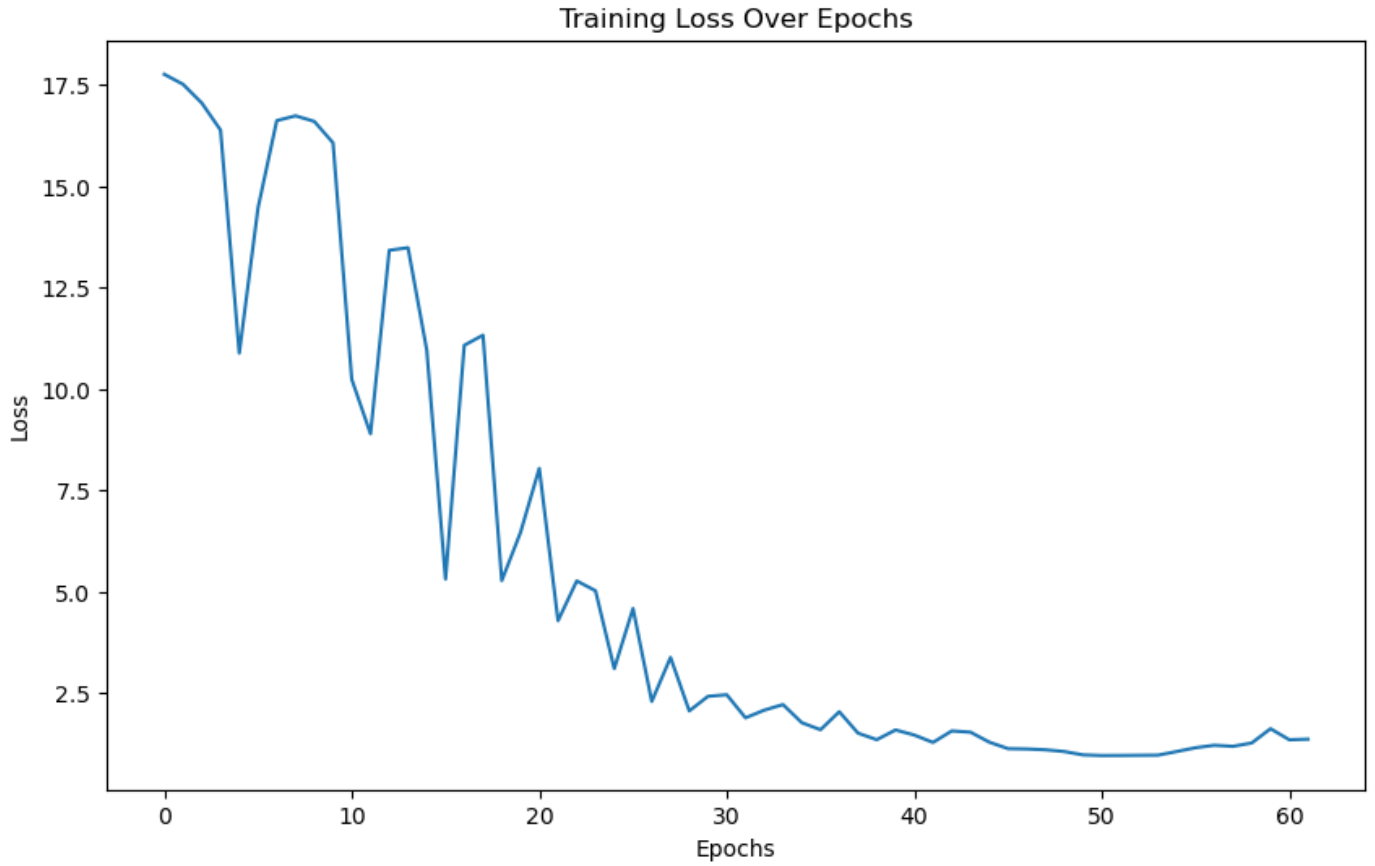
	GPA	attendance	duration	language	at-risk
0	0.94	29	2017	51	1
1	3.65	82	5722	74	1
2	2.41	69	4917	15	0
3	1.24	6	3720	50	0
4	2.14	54	2487	59	1

In [27]:
```
# Separate features and target variable
X = df.drop('at-risk', axis=1)
y = df['at-risk']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [36]:
```
# Define and train the neural network
activation_function = 'relu'
num_neurons = 50

# Create the neural network model
# Implementation of activation function and number of neurons in the hidden layer
model = MLPClassifier(hidden_layer_sizes=(num_neurons,), activation=activation_function,
```

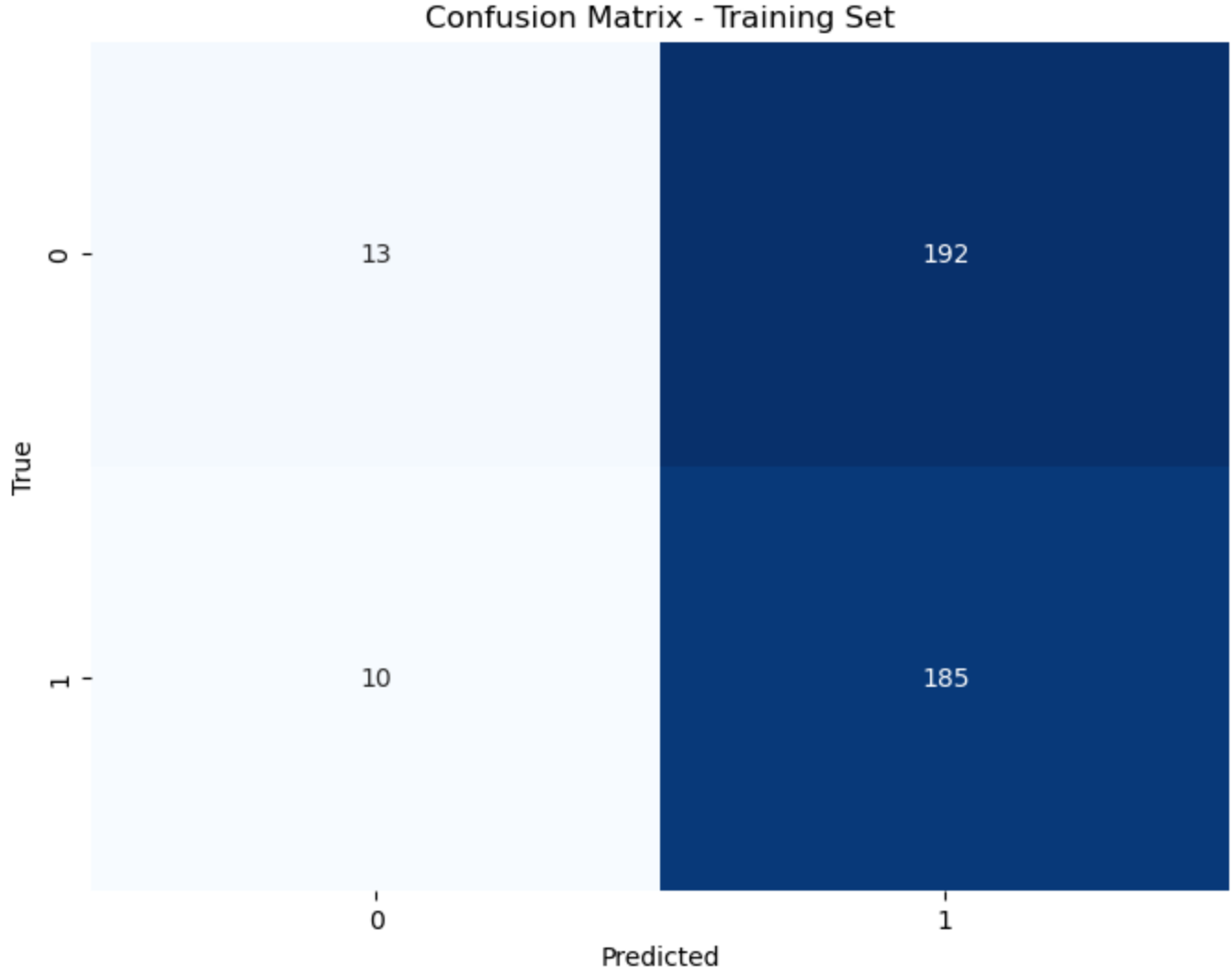
In [38]:
```
# Train the model
history = model.fit(X_train, y_train)
```

In [29]:
```
# Visualize training loss over epochs
plt.figure(figsize=(10, 6))
plt.plot(history.loss_curve_)
plt.title('Training Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



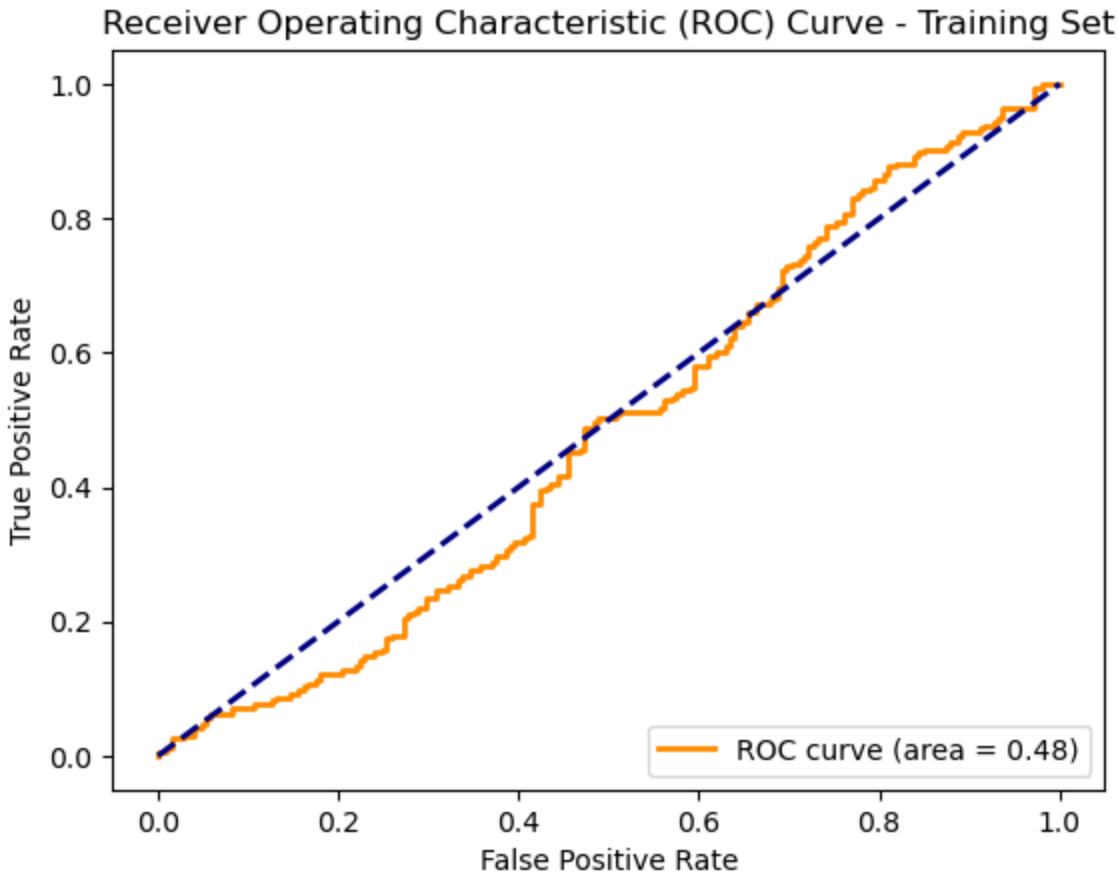
In [30]:
```
# Predictions on the training set for demonstration purposes
train_predictions = model.predict(X_train)

# Evaluate the model on the training set
train_accuracy = accuracy_score(y_train, train_predictions)
print(f"Training Accuracy: {train_accuracy:.2f}")

#Confusion Matrix
# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_train, train_predictions), annot=True, fmt='d', cmap='Blu
plt.title('Confusion Matrix - Training Set')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Training Accuracy: 0.49
```

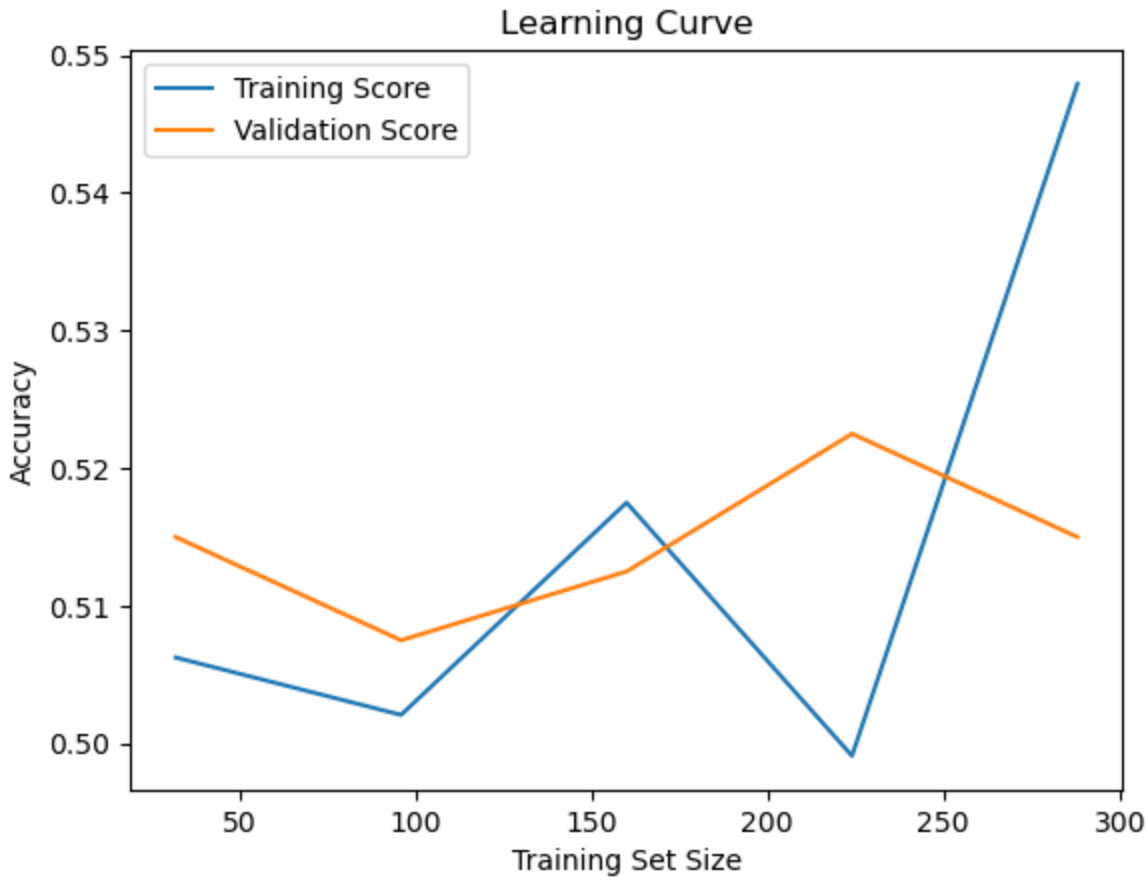


In [31]:

```
#Roc curve
from sklearn.metrics import roc_curve, auc
# Calculate the ROC curve and area under the curve (AUC)
fpr, tpr, thresholds = roc_curve(y_train, model.predict_proba(X_train)[:, 1])
roc_auc = auc(fpr, tpr)
# Plot the ROC curve
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(ro
# Plot the diagonal line representing random guessing
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
# Add labels and title to the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - Training Set')
# Add a legend to the plot
plt.legend(loc='lower right')
plt.show()
```



In [32]:
```
#Learning Curve
from sklearn.model_selection import learning_curve
#Calculate Learning curve
train_sizes, train_scores, valid_scores = learning_curve(model, X_train, y_train, train_
# Plot the learning curve and add labels to it
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Score')
plt.plot(train_sizes, np.mean(valid_scores, axis=1), label='Validation Score')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curve')
plt.legend()
plt.show()
```



In [34]:
```
# Use the trained model to make predictions on the test set
y_pred = model.predict(X)

# Calculate the error using accuracy as the metric
error = 1 - accuracy_score(y, y_pred)
print(f"Error on Test Set: {error:.4f}")
```

```
Error on Test Set: 0.4960
```

In [35]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
# Calculate and print metrics for the test set
test_accuracy = accuracy_score(y, y_pred)
test_precision = precision_score(y, y_pred)
test_recall = recall_score(y, y_pred)
test_f1 = f1_score(y, y_pred)

print(f"\nTest Accuracy: {test_accuracy:.2f}")
print(f"Test Precision: {test_precision:.2f}")
print(f"Test Recall: {test_recall:.2f}")
print(f"Test F1-Score: {test_f1:.2f}")
```

```
Test Accuracy: 0.50
Test Precision: 0.50
Test Recall: 0.95
Test F1-Score: 0.65
```

In [ ]:
```
```