Task 3

Classification and Neural Network

Importing Libraries and Data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
import warnings
warnings.filterwarnings("ignore")


# Loading the NBA rookie data from the CSV file
data = pd.read_csv('/content/drive/MyDrive/Regression,clustering,ANNproject/nba_rookie_data.csv')
data = data.dropna()
print(data.head())
```

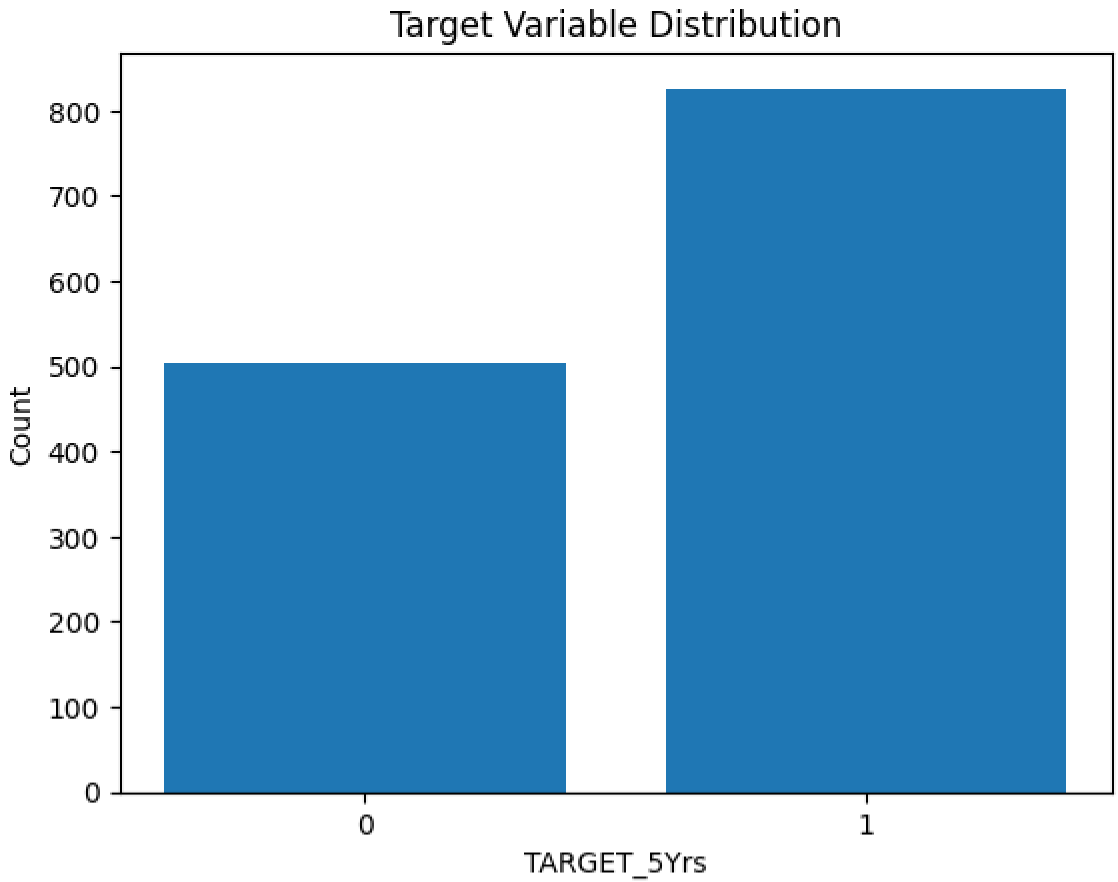
```
             Name  Games Played  Minutes Played  Points Per Game  \
0   Brandon Ingram            36            27.4              7.4   
1  Andrew Harrison            35            26.9              7.2   
2   JaKarr Sampson            74            15.3              5.2   
3      Malik Sealy            58            11.6              5.7   
4      Matt Geiger            48            11.5              4.5   

   Field Goals Made  Field Goal Attempts  Field Goal Percent  3 Point Made  \
0               2.6                  7.6                34.7           0.5   
1               2.0                  6.7                29.6           0.7   
2               2.0                  4.7                42.2           0.4   
3               2.3                  5.5                42.6           0.1   
4               1.6                  3.0                52.4           0.0   

   3 Point Attempt  3 Point Percent  ...  Free Throw Attempts  \
0              2.1             25.0  ...                  2.3   
1              2.8             23.5  ...                  3.4   
2              1.7             24.4  ...                  1.3   
3              0.5             22.6  ...                  1.3   
4              0.1              0.0  ...                  1.9   

   Free Throw Percent  Offensive Rebounds  Defensive Rebounds  Rebounds  \
0                69.9                 0.7                 3.4       4.1   
1                76.5                 0.5                 2.0       2.4   
2                67.0                 0.5                 1.7       2.2   
3                68.9                 1.0                 0.9       1.9   
4                67.4                 1.0                 1.5       2.5   

   Assists  Steals  Blocks  Turnovers  TARGET_5Yrs  
0      1.9     0.4     0.4        1.3            0  
1      3.7     1.1     0.5        1.6            0  
2      1.0     0.5     0.3        1.0            0  
3      0.8     0.6     0.1        1.0            1  
4      0.3     0.3     0.4        0.8            1  

[5 rows x 21 columns]
```

Data Exploration and Processing

In [2]:
```python
# Checking for missing values in the dataset
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)

# Checking the data types of each column
data_types = data.dtypes
print("Data Types:\n", data_types)

# Checking summary statistics of the dataset
summary_stats = data.describe()
print("Summary Statistics:\n", summary_stats)

# Checking the distribution of the target variable
target_distribution = data['TARGET_5Yrs'].value_counts()
print("Target Variable Distribution:\n", target_distribution)
```

```
Free Throw Made        0
Free Throw Attempts    0
Free Throw Percent     0
Offensive Rebounds     0
Defensive Rebounds     0
Rebounds               0
Assists                0
Steals                 0
Blocks                 0
Turnovers              0
TARGET_5Yrs            0
dtype: int64
Data Types:
 Name                    object
Games Played             int64
Minutes Played         float64
Points Per Game        float64
Field Goals Made       float64
Field Goal Attempts    float64
Field Goal Percent     float64
```

In [3]:
```
count_0 = data['TARGET_5Yrs'].value_counts()[0]
count_1 = data['TARGET_5Yrs'].value_counts()[1]

# Creating a bar chart for the target variable
plt.bar(['0', '1'], [count_0, count_1])
plt.xlabel('TARGET_5Yrs')
plt.ylabel('Count')
plt.title('Target Variable Distribution')
plt.show()
```



Data Preprocessing

In [4]:
```
# Seperating the features (X) and the target variable (y)
X = data.drop(columns=['TARGET_5Yrs', 'Name'])
y = data['TARGET_5Yrs']

# Splitting the data 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)

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1063, 19)
Shape of X_test: (266, 19)
Shape of y_train: (1063,)
Shape of y_test: (266,)
```

Building and Evaluating Machine Learning Models

Using Logistic Regression

In [5]:

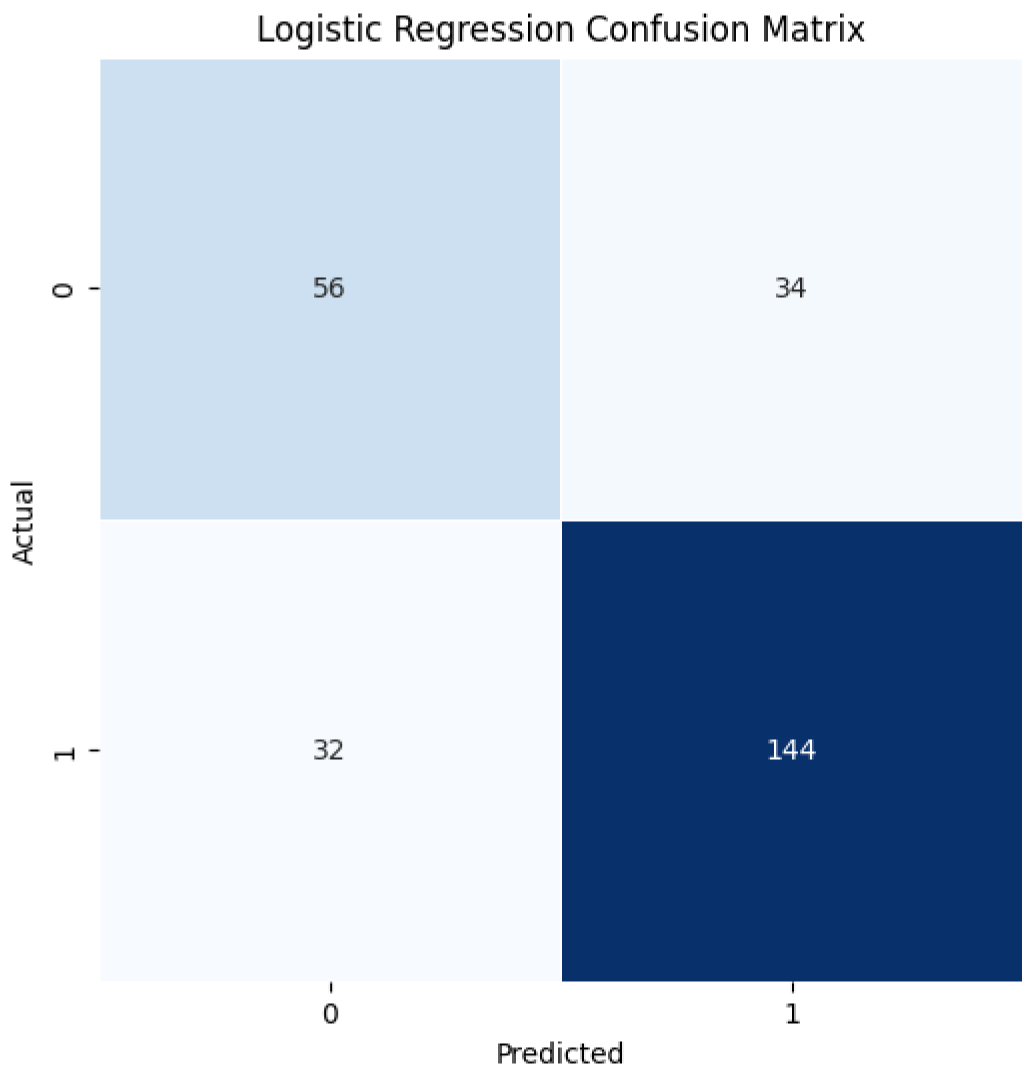
```python
# Using Logistic Regression
logistic_model = LogisticRegression(random_state=42)

# Fitting the model on the training data
logistic_model.fit(X_train, y_train)

y_pred_logistic = logistic_model.predict(X_test)

# Evaluating the Logistic Regression model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
confusion_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)

print("Logistic Regression Model:")
print("Accuracy:", accuracy_logistic)
print("\nConfusion Matrix:\n", confusion_matrix_logistic)
print("\nClassification Report:\n", classification_report(y_test, y_pred_logistic))
```

```
Logistic Regression Model:
Accuracy: 0.7518796992481203

Confusion Matrix:
 [[ 56  34]
 [ 32 144]]

Classification Report:
               precision    recall  f1-score   support

           0       0.64      0.62      0.63        90
           1       0.81      0.82      0.81       176

    accuracy                           0.75       266
   macro avg       0.72      0.72      0.72       266
weighted avg       0.75      0.75      0.75       266
```

In [6]:
```
# Getting odds ratios
odds_ratios = np.exp(logistic_model.coef_)

# Creating a DataFrame to display the odds ratios
odds_ratios_df = pd.DataFrame(odds_ratios, columns=X_train.columns, index=['Odds Ratio'])
print(odds_ratios_df)
```

```
            Games Played  Minutes Played  Points Per Game  Field Goals Made  \
Odds Ratio      1.031707        0.933495         1.817181          1.270428

            Field Goal Attempts  Field Goal Percent  3 Point Made  \
Odds Ratio             0.605872            0.971859      1.135747

            3 Point Attempt  3 Point Percent  Free Throw Made  \
Odds Ratio         0.837978         1.002958          0.99528

            Free Throw Attempts  Free Throw Percent  Offensive Rebounds  \
Odds Ratio             0.673896            0.994122            1.558721

            Defensive Rebounds  Rebounds   Assists    Steals    Blocks  \
Odds Ratio            0.820391  1.279351  1.382729  0.972474  1.280231

            Turnovers
Odds Ratio   0.746763
```

In [7]:
```
# Visualise the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_logistic, annot=True, fmt="d", cmap="Blues", linewidths=.5, square=True, cbar=Fal
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Logistic Regression Confusion Matrix")
plt.show()
```



Using GaussianNB

In [8]:
```
#Using GaussianNB
nb_model = GaussianNB()

# Fitting the model on the training data
nb_model.fit(X_train, y_train)

# Predicting using the Gaussian Naive Bayes model
y_pred_nb = nb_model.predict(X_test)

nb_accuracy = accuracy_score(y_test, y_pred_nb)
nb_precision = precision_score(y_test, y_pred_nb)
nb_recall = recall_score(y_test, y_pred_nb)

print("Gaussian Naive Bayes Model Metrics:")
print(f"Accuracy: {nb_accuracy:.2f}")
print(f"Precision: {nb_precision:.2f}")
print(f"Recall: {nb_recall:.2f}")
```

```
Gaussian Naive Bayes Model Metrics:
Accuracy: 0.61
Precision: 0.88
Recall: 0.48
```

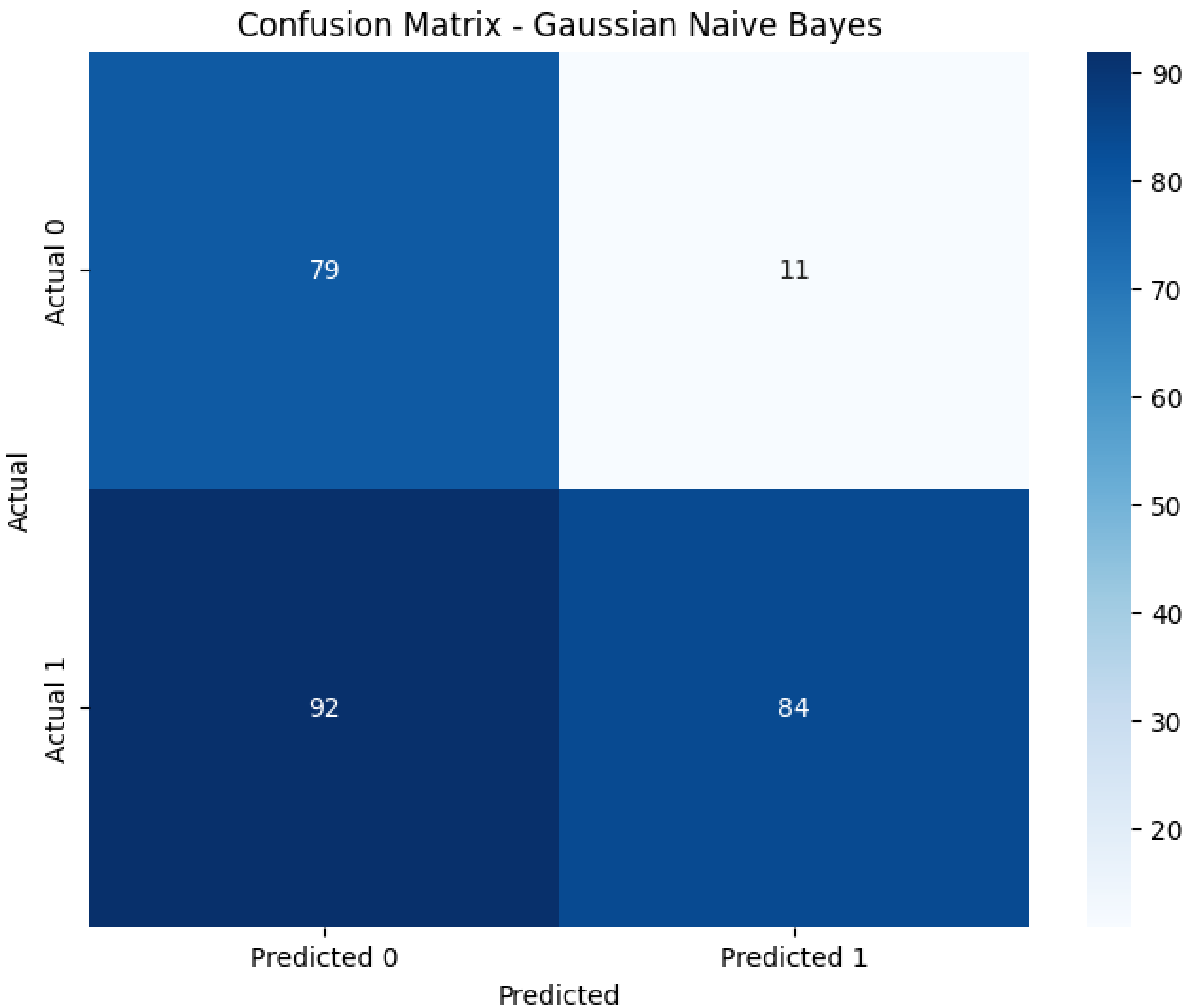
In [9]:
```
# Displaying class-conditional distribution summary
for class_label in np.unique(y_train):
    class_indices = (y_train == class_label)
    class_data = X_train.loc[class_indices]

    # Using mean and std to get class-conditional distribution summary
    class_mean = class_data.mean()
    class_std = class_data.std()

    print(f"\nClass {class_label} - Mean:")
    print(class_mean)

    print(f"\nClass {class_label} - Standard Deviation:")
    print(class_std)
```

```
Class 0 - Mean:
Games Played           52.164649
Minutes Played         14.475787
Points Per Game         5.132446
Field Goals Made        1.970218
Field Goal Attempts     4.614044
Field Goal Percent     42.192736
3 Point Made            0.236077
3 Point Attempt         0.771913
3 Point Percent        19.209685
Free Throw Made         0.959080
Free Throw Attempts     1.361501
Free Throw Percent     69.523729
Offensive Rebounds      0.713317
Defensive Rebounds      1.539467
Rebounds                2.253511
Assists                 1.261985
Steals                  0.506538
Blocks                  0.258111
Turnovers               0.957143
dtype: float64

Class 0 - Standard Deviation:
Games Played           17.058210
Minutes Played          6.745397
Points Per Game         3.211314
Field Goals Made        1.236209
Field Goal Attempts     2.695195
Field Goal Percent      6.519623
3 Point Made            0.332731
3 Point Attempt         0.961105
3 Point Percent        15.362590
Free Throw Made         0.707447
Free Throw Attempts     0.948425
Free Throw Percent     10.696353
Offensive Rebounds      0.550456
Defensive Rebounds      1.005961
Rebounds                1.470889
Assists                 1.154932
Steals                  0.334549
Blocks                  0.291237
Turnovers               0.573423
```

```
dtype: float64

Class 1 - Mean:
Games Played            65.678462
Minutes Played          19.847692
Points Per Game          7.979692
Field Goals Made         3.085538
Field Goal Attempts      6.799231
Field Goal Percent      45.222615
3 Point Made             0.264615
3 Point Attempt          0.814000
3 Point Percent         19.251231
Free Throw Made          1.544462
Free Throw Attempts      2.153385
Free Throw Percent      71.128154
Offensive Rebounds       1.172923
Defensive Rebounds       2.323692
Rebounds                 3.498615
Assists                  1.816308
Steals                   0.704923
Blocks                   0.441231
Turnovers                1.370462
dtype: float64

Class 1 - Standard Deviation:
Games Played            15.522328
Minutes Played           8.646018
Points Per Game          4.757861
Field Goals Made         1.830837
Field Goal Attempts      3.931938
Field Goal Percent       5.599584
3 Point Made             0.426882
3 Point Attempt          1.157053
3 Point Percent         16.303343
Free Throw Made          1.106964
Free Throw Attempts      1.486207
Free Throw Percent      10.047382
Offensive Rebounds       0.826565
Defensive Rebounds       1.444172
Rebounds                 2.186275
Assists                  1.689188
Steals                   0.453686
Blocks                   0.495923
```

```
Turnovers             0.788391
dtype: float64
```

In [10]:
```
# Generating the confusion matrix for GaussianNB
cm_nb = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gaussian Naive Bayes')

plt.show()
```



Using Neural Network

In [11]:

```
# Standardize the feature data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [12]:

```
# Initializing a Sequential model
model = Sequential()

# Adding input and hidden layers
model.add(Dense(units=64, activation='relu', input_dim=X_train_scaled.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compiling
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the model on the training data
model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, verbose=0)

# Evaluating the model
loss, accuracy = model.evaluate(X_test_scaled, y_test)

print("Neural Network Model Metrics:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Loss: {loss:.2f}")

y_pred_nn = model.predict(X_test_scaled)

y_pred_nn_binary = (y_pred_nn > 0.5).astype(int)
nn_cm = confusion_matrix(y_test, y_pred_nn_binary)
```

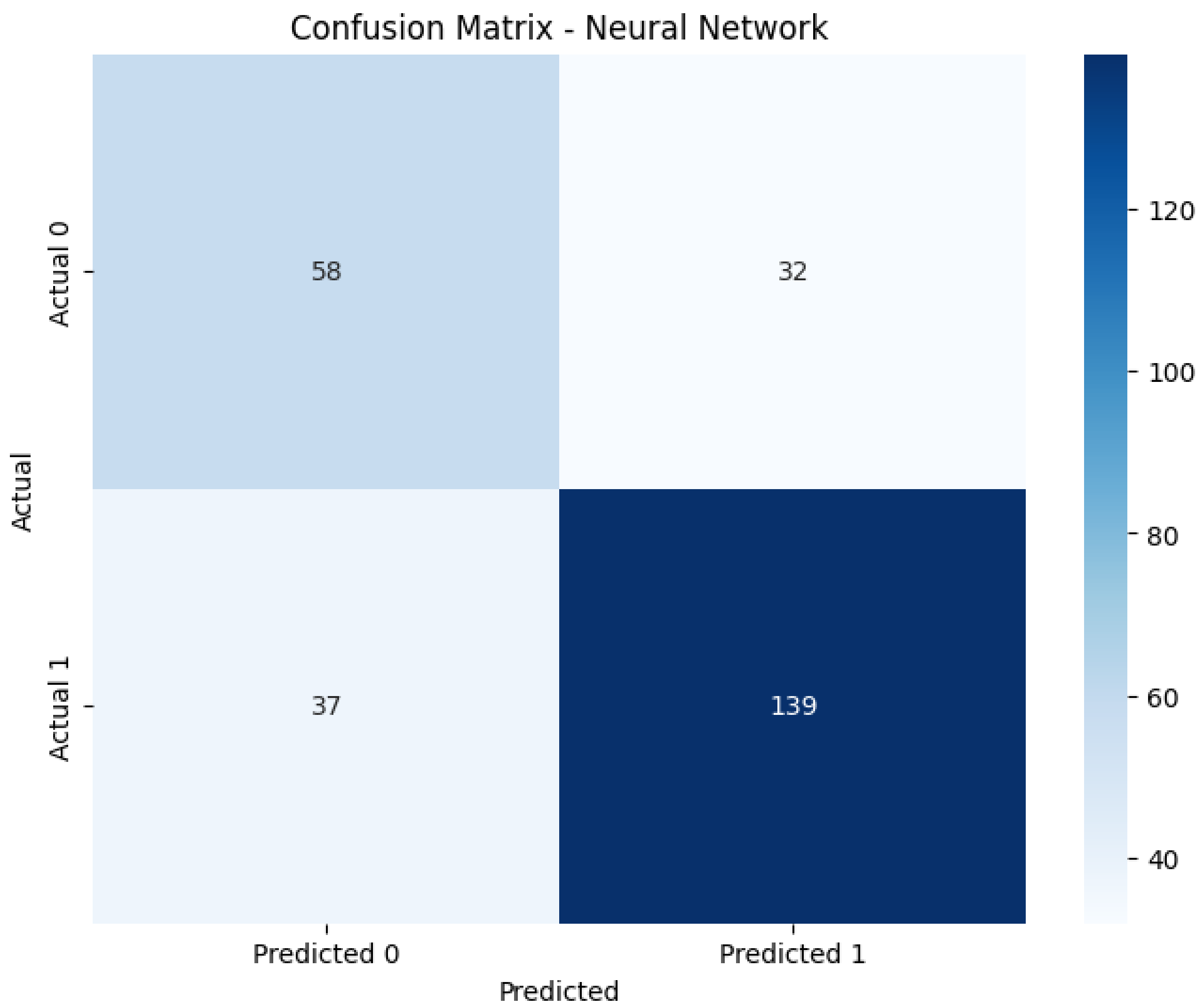
```
9/9 [==============================] - 0s 3ms/step - loss: 0.5180 - accuracy: 0.7406
Neural Network Model Metrics:
Accuracy: 0.74
Loss: 0.52
9/9 [==============================] - 0s 2ms/step
```

In [15]:
```
#Computing the gradients
import tensorflow as tf

# Convert y_test to a NumPy array and reshape it to have the same shape as predictions
y_test_array = y_test.to_numpy().reshape(predictions.shape)

with tf.GradientTape() as tape:
    predictions = model(X_test_scaled, training=False)
    loss = tf.keras.losses.binary_crossentropy(y_test_array, predictions)

gradients = tape.gradient(loss, model.trainable_variables)

# Printing the gradients
print("\nGradients with respect to the predictors:")
for variable, gradient in zip(model.trainable_variables, gradients):
    print(f"{variable.name}: {gradient.numpy()}")
```

```
Gradients with respect to the predictors:
dense/kernel:0: [[-0.51345736 -1.8638855  -1.4889694  ... -0.7568046  -1.509676
  -0.18912023]
 [ 0.8989229   0.37073824 -0.3470263  ... -1.7063999  -1.5757983
  -0.2557569 ]
 [ 0.89685965  0.25459862 -0.18658966 ... -1.1835729  -0.9833245
  -0.31350726]
 ...
 [-0.09185281 -0.03649488 -0.8751375  ... -2.0798237  -0.5486175
  -0.3031538 ]
 [ 0.22954163  0.29954892  0.2578664  ... -0.62480646 -0.8158878
  -0.06043856]
 [ 0.57782865  0.48026577 -0.1307025  ... -1.6793607  -0.88249254
  -0.41991854]]
dense/bias:0: [-0.88979685 -3.670798   -0.743859   -0.10795645  1.1229175  -0.58270484
  0.02383753  0.6242887  -1.1809505  -1.7340785   0.16821681  1.7022071
 -0.49014854  0.59597105 -0.7249367  -0.71132845 -0.2801013  -1.1348864
 -1.1499724   3.0493002  -0.5034452   1.044165   -0.89823055 -0.40899706
 -2.8334374   0.06044286 -0.6513698  -5.816498   -0.5559711  -1.830761
 -1.1622103   0.08617996  3.3939195   4.5855913   0.6265912   1.241676
 -1.3235836  -0.9556078   0.58638257 -2.030844    0.05398785 -0.6706696
  0.42224386 -0.09134041  1.4370939  -1.8136206  -0.3502716  -1.1673585
 -0.82157105  0.81405187 -0.67599964  0.81997263 -1.5505705  -1.600334
 -1.3461238   2.3048546  -3.831507    1.2982335   2.05287     1.0579293
 -0.18487488  3.2012296   1.8702426   0.15546861]
dense_1/kernel:0: [[-0.15554321  0.1336205   0.23764592 ...  0.27509782  0.45288846
   0.43315685]
 [ 0.03420717 -1.1197641  -1.4121795  ... -1.2051051  -1.9614605
   0.13902014]
 [-0.13735276  0.07829833  0.5636358  ... -0.06268296  0.62643677
  -0.07126153]
 ...
 [ 0.06387046 -0.26154533 -0.5081401  ... -0.05686362 -0.8168541
  -0.20529209]
 [ 0.05014237 -0.41522276 -0.49232936 ... -0.27476507 -0.43219417
  -0.07326937]
 [ 0.00296068 -0.0555322  -0.6985424  ... -0.12900719 -0.3384301
   0.39888838]]
dense_1/bias:0: [ 0.19257936 -1.8184706  -3.4113612  -5.7639613  -0.22357783  0.26475686
  0.06708814 -1.3039504   0.5686099   4.448685    0.9365822   3.3912797
 -0.07667176  2.9598289  -0.2552107   1.7768855  -0.65843254 -0.8810512
 -3.7518094  -1.7923708  -4.8242173   2.0566185  -0.66811115 -1.2372751
```

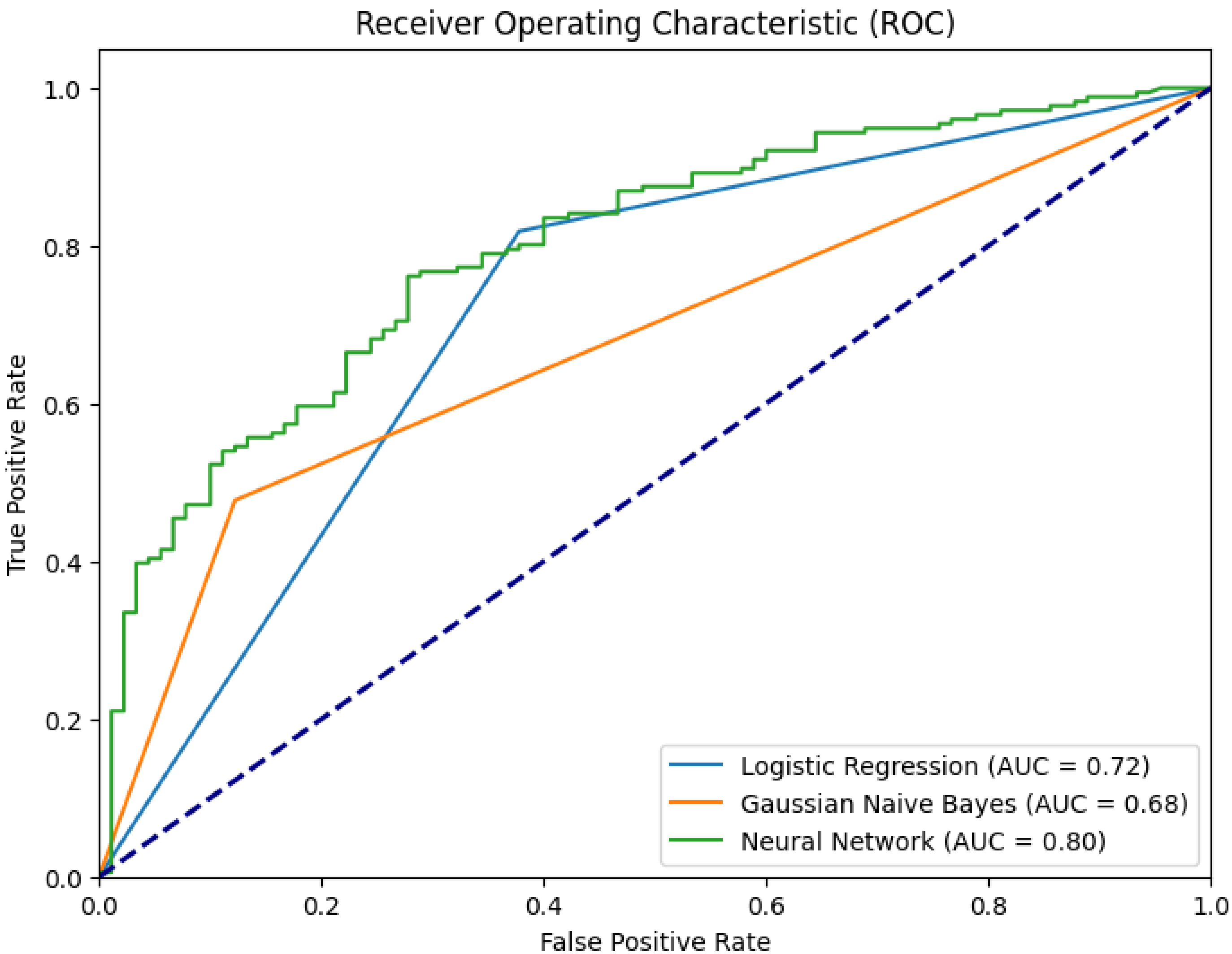
```
  3.1547344  -1.8642731  -0.46602133 -2.6080909  -2.275921   -0.61219275
 -3.287274    1.48606   ]
dense_2/kernel:0: [[-0.09455679]
 [-0.8737226 ]
 [-6.7860694 ]
 [-5.12109   ]
 [-0.23783968]
 [-0.11559491]
 [ 0.10370708]
 [-1.1581635 ]
 [-0.2918618 ]
 [-5.426965  ]
 [-7.204186  ]
 [-1.853367  ]
 [-0.03534299]
 [-3.6128461 ]
 [-3.2004611 ]
 [-3.599795  ]
 [-3.5850015 ]
 [-1.7182871 ]
 [-1.8106177 ]
 [-2.2991025 ]
 [-8.70753   ]
 [-3.469163  ]
 [-0.02554643]
 [-3.6387074 ]
 [-1.1244493 ]
 [-3.8209116 ]
 [-0.0164669 ]
 [-1.197884  ]
 [-1.690753  ]
 [-1.7144713 ]
 [-3.6114404 ]
 [-0.6093557 ]]
dense_2/bias:0: [-12.716201]
```

In [16]:
```
# Creating a heatmap to visualize the confusion matrix for Neural Network
plt.figure(figsize=(8, 6))
sns.heatmap(nn_cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Neural Network')

plt.show()
```



Comparing the models and Visualizing

In [17]:
```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Calculating ROC curve and AUC for Logistic Regression
fpr_logistic, tpr_logistic, _ = roc_curve(y_test, y_pred_logistic)
roc_auc_logistic = roc_auc_score(y_test, y_pred_logistic)

# Calculating ROC curve and AUC for Gaussian Naive Bayes
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_pred_nb)
roc_auc_nb = roc_auc_score(y_test, y_pred_nb)

# Calculating ROC curve and AUC for Neural Network
fpr_nn, tpr_nn, _ = roc_curve(y_test, y_pred_nn)
roc_auc_nn = roc_auc_score(y_test, y_pred_nn)

# Plotting ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_logistic, tpr_logistic, label='Logistic Regression (AUC = %0.2f)' % roc_auc_logistic)
plt.plot(fpr_nb, tpr_nb, label='Gaussian Naive Bayes (AUC = %0.2f)' % roc_auc_nb)
plt.plot(fpr_nn, tpr_nn, label='Neural Network (AUC = %0.2f)' % roc_auc_nn)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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)')
plt.legend(loc='lower right')
plt.show()
```

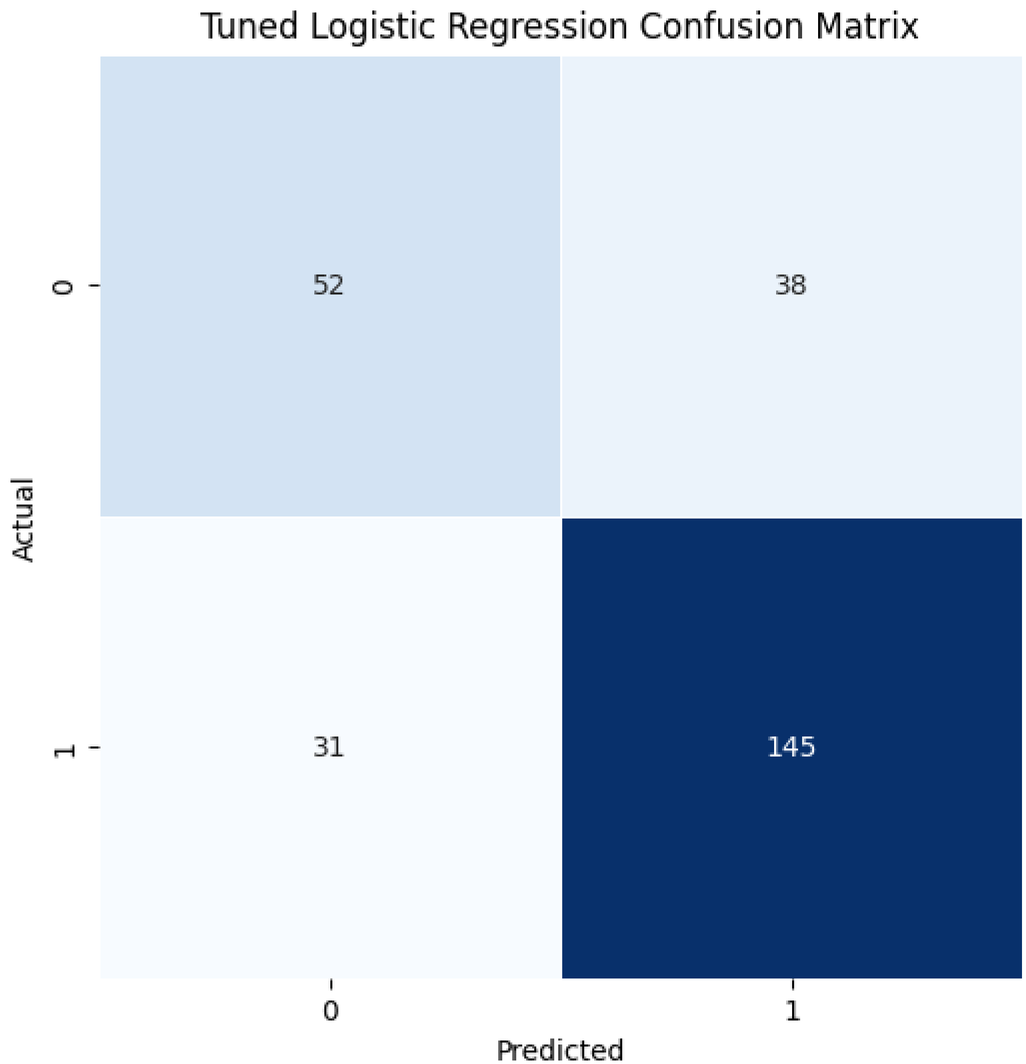


Improving the Models by using Hyperparameter Tuning

Hyperparameter Tuning for Logistic Regression

In [18]:
```
# Defining the hyperparameter grid for Logistic Regression
param_grid_lr = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'max_iter': [100, 200, 300, 400]  # Maximum number of iterations
}

# Creating a Logistic Regression model
lr_model = LogisticRegression()

# Creating a GridSearchCV object for Logistic Regression
lr_grid = GridSearchCV(lr_model, param_grid_lr, cv=5, scoring='accuracy')

# Fitting the GridSearchCV object to ythe data
lr_grid.fit(X_train, y_train)

# Getting the best parameters and the best estimator
best_params_lr = lr_grid.best_params_
best_lr_model = lr_grid.best_estimator_

y_pred_Tuned_logistic = best_lr_model.predict(X_test)

# Evaluating the tuned Logistic Regression model
accuracy_logistic_tuned = accuracy_score(y_test, y_pred_Tuned_logistic)
confusion_matrix_logistic_tuned = confusion_matrix(y_test, y_pred_Tuned_logistic)

print("Tuned Logistic Regression Model:")
print("Accuracy:", accuracy_logistic_tuned)
print("\nConfusion Matrix:\n", confusion_matrix_logistic_tuned)
print("\nClassification Report:\n", classification_report(y_test, y_pred_Tuned_logistic))

# Visualising the Confusion Matrix for the tuned Logistic model
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix_logistic_tuned, annot=True, fmt="d", cmap="Blues", linewidths=.5, square=True, cb
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Tuned Logistic Regression Confusion Matrix")
plt.show()
```

```
Tuned Logistic Regression Model:
Accuracy: 0.7406015037593985

Confusion Matrix:
 [[ 52  38]
 [ 31 145]]

Classification Report:
               precision    recall  f1-score   support

           0       0.63      0.58      0.60        90
           1       0.79      0.82      0.81       176

    accuracy                           0.74       266
   macro avg       0.71      0.70      0.70       266
weighted avg       0.74      0.74      0.74       266
```

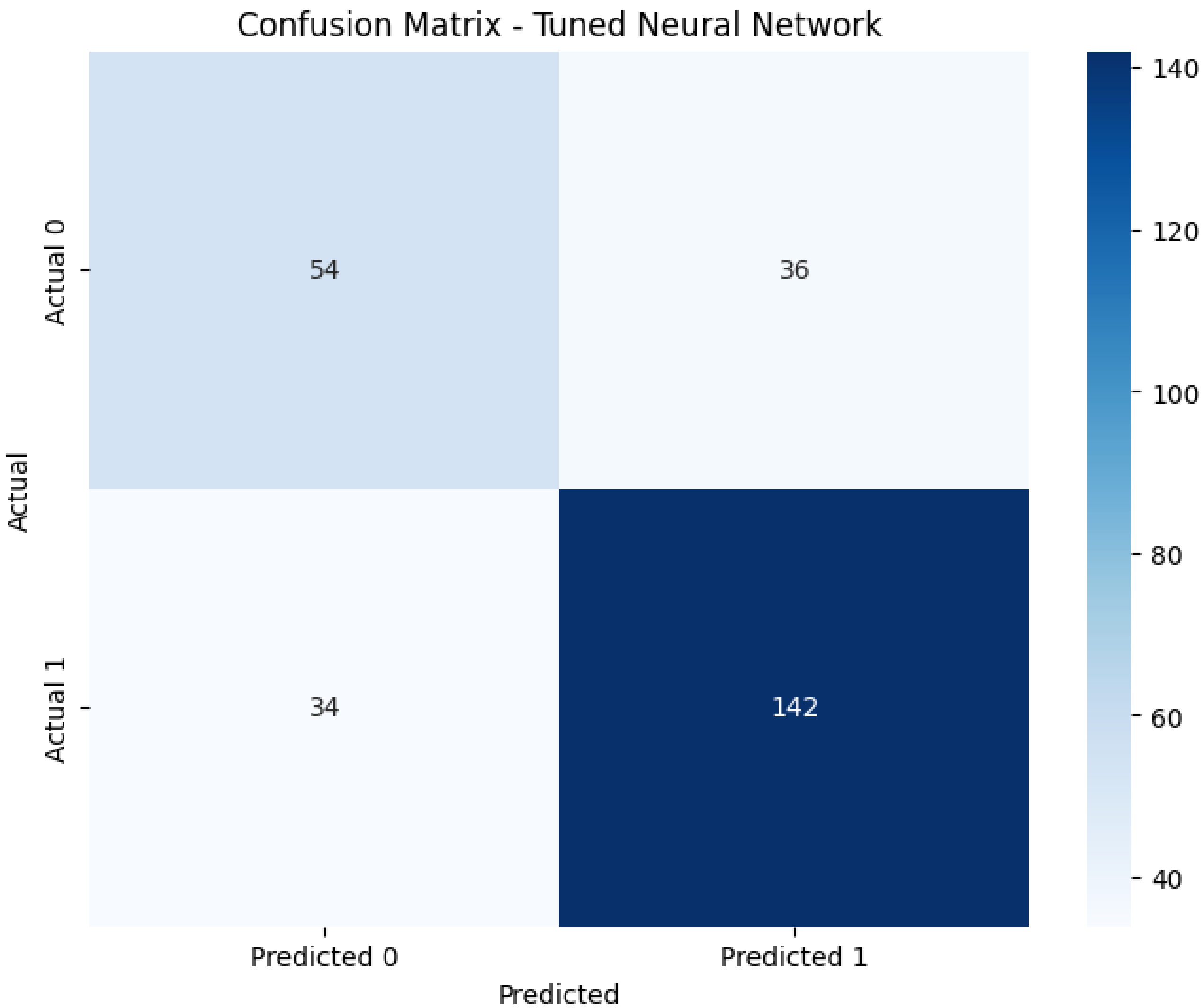

Tuned Logistic Regression Confusion Matrix
52
38
31
145
0
1
0
1
Actual
Predicted

In [20]:
```
# Getting odds ratios for the Tuned Logistic Regression
odds_ratios_Tunedlr = np.exp(best_lr_model.coef_)

# Creating a DataFrame to display the odds ratios
odds_ratios_Tunedlr_df = pd.DataFrame(odds_ratios_Tunedlr, columns=X_train.columns, index=['Odds Ratio'])
print(odds_ratios_Tunedlr_df)
```

```
            Games Played  Minutes Played  Points Per Game  Field Goals Made  \
Odds Ratio      1.032872        0.974963         1.046024           1.01563

            Field Goal Attempts  Field Goal Percent  3 Point Made  \
Odds Ratio             1.034847            1.044758      1.005497

            3 Point Attempt  3 Point Percent  Free Throw Made  \
Odds Ratio         0.962827         1.004123         1.008961

            Free Throw Attempts  Free Throw Percent  Offensive Rebounds  \
Odds Ratio             1.007981            1.007231            1.116741

            Defensive Rebounds  Rebounds   Assists    Steals    Blocks  \
Odds Ratio            0.989148  1.104806  1.078209  1.004024  1.054412

            Turnovers
Odds Ratio   0.993027
```

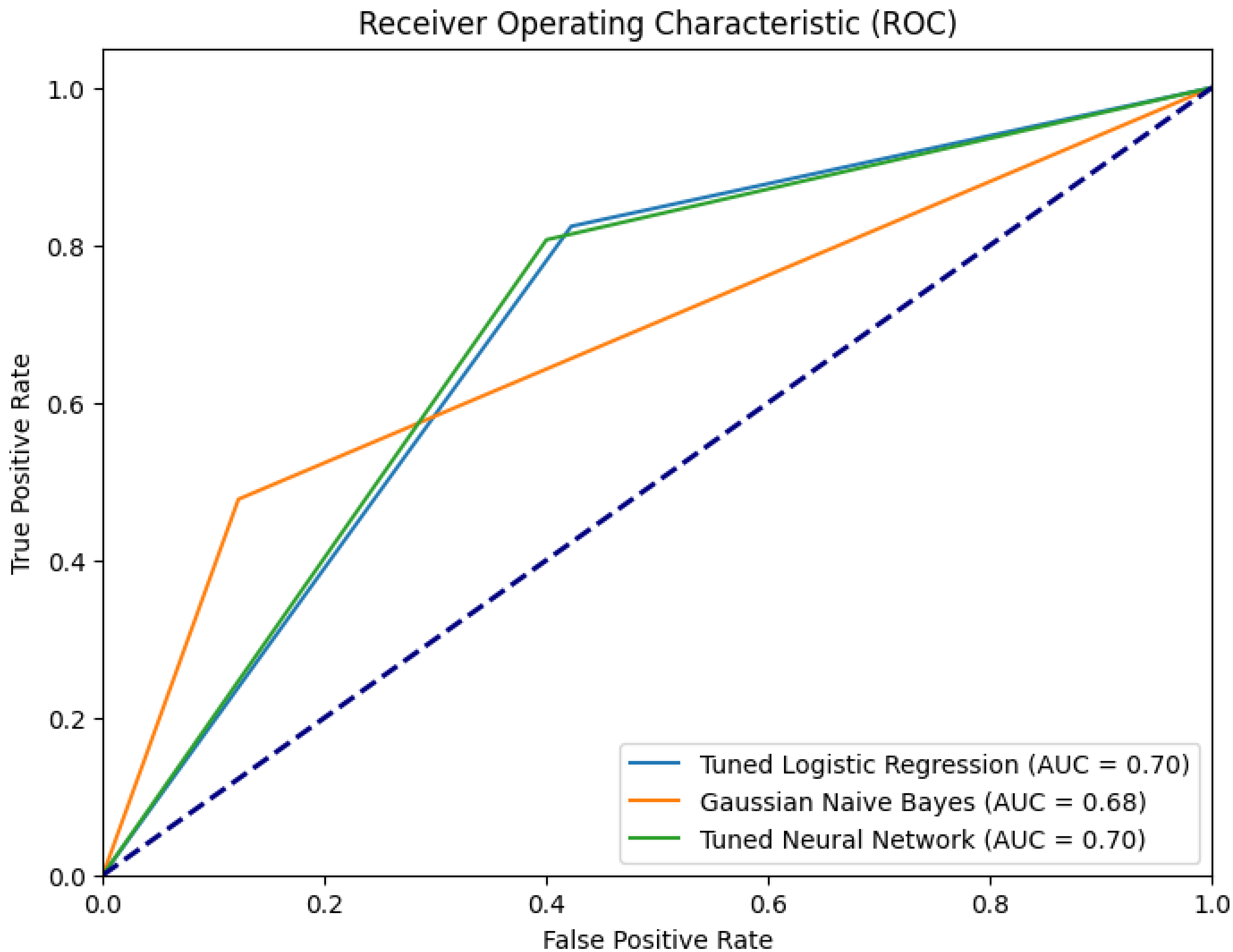
Hyperparameter Tuning for Neural Network

In [21]:
```
# Defining the hyperparameter grid for Neural Network
param_grid_nn = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'alpha': [0.0001, 0.001, 0.01, 0.1],  # L2 regularization term
    'max_iter': [100, 200, 300, 400]  # Maximum number of iterations
}
# Creating a Neural Network model
nn_model = MLPClassifier()

# Creating a GridSearchCV object for Neural Network
nn_grid = GridSearchCV(nn_model, param_grid_nn, cv=5, scoring='accuracy')

# Fitting the GridSearchCV object to the data
nn_grid.fit(X_train_scaled, y_train)

# Getting the best parameters and the best estimator
best_params_nn = nn_grid.best_params_
best_nn_model = nn_grid.best_estimator_

y_pred_Tuned_nn = best_nn_model.predict(X_test_scaled)

# Evaluating the tuned neural network model
accuracy = accuracy_score(y_test, y_pred_Tuned_nn)
y_pred_nn_binary_Tuned = (y_pred_Tuned_nn > 0.5).astype(int)
nn_cm_tuned = confusion_matrix(y_test, y_pred_nn_binary_Tuned)

# Creating a heatmap to visualize the confusion matrix for the Tuned Neural Network
plt.figure(figsize=(8, 6))
sns.heatmap(nn_cm_tuned, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], ytickl
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Tuned Neural Network')

plt.show()
```


Confusion Matrix - Tuned Neural Network
Actual 0
Actual 1
Actual
54
36
34
142
Predicted 0
Predicted 1
Predicted
140
120
100
80
60
40

Comparing the Tuned Models

In [26]:
```
# Calculating ROC curve and AUC for Tuned Logistic Regression
fpr_logistic_tuned, tpr_logistic_tuned, _ = roc_curve(y_test, y_pred_Tuned_logistic)
roc_auc_logistic_tuned = roc_auc_score(y_test, y_pred_Tuned_logistic)

# Calculating ROC curve and AUC for Gaussian Naive Bayes ( No Change)
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_pred_nb)
roc_auc_nb = roc_auc_score(y_test, y_pred_nb)

# Calculating ROC curve and AUC for Neural Network
fpr_nn_tuned, tpr_nn_tuned, _ = roc_curve(y_test, y_pred_Tuned_nn)
roc_auc_nn_tuned = roc_auc_score(y_test, y_pred_Tuned_nn)

# Plotting ROC curves for the tuned model
plt.figure(figsize=(8, 6))
plt.plot(fpr_logistic_tuned, tpr_logistic_tuned, label='Tuned Logistic Regression (AUC = %0.2f)' % roc_auc_log
plt.plot(fpr_nb, tpr_nb, label='Gaussian Naive Bayes (AUC = %0.2f)' % roc_auc_nb)
plt.plot(fpr_nn_tuned, tpr_nn_tuned, label='Tuned Neural Network (AUC = %0.2f)' % roc_auc_nn_tuned)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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)')
plt.legend(loc='lower right')
plt.show()
```


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Tuned Logistic Regression (AUC = 0.70)
Gaussian Naive Bayes (AUC = 0.68)
Tuned Neural Network (AUC = 0.70)