```
!pip install -q altair shap causal-learn graphviz pydot plotnine
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/193.0 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 193.0/193.0 kB 6.0 MB/s eta
0:00:00
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import json

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, roc_curve, auc
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Input, Dense, Dropout


from plotnine.data import mpg
from plotnine import ggplot, aes, geom_histogram, geom_density,
theme_bw, labs
from IPython.display import display

import networkx as nx

import altair as alt

from causallearn.search.ConstraintBased.PC import pc
from causallearn.utils.cit import fisherz
from causallearn.graph.Edge import Endpoint

import shap
from shap import Explanation, KernelExplainer, summary_plot,
force_plot

df = pd.read_csv('/content/gene_expression.csv')

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 3000,\n  \"fields\":
[\n    {\n      \"column\": \"Gene One\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.8283875362049855,\n
```

\"min\": 1.0,\n   \"max\": 10.0,\n   \"num_unique_values\": 89,\n   \"samples\": [\n   4.7,\n   2.9,\n   6.2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Gene Two\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.7290807834436468,\n   \"min\": 1.0,\n   \"max\": 10.0,\n   \"num_unique_values\": 88,\n   \"samples\": [\n   1.4,\n   3.9,\n   6.1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Cancer Present\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 0,\n   \"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0,\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df"}

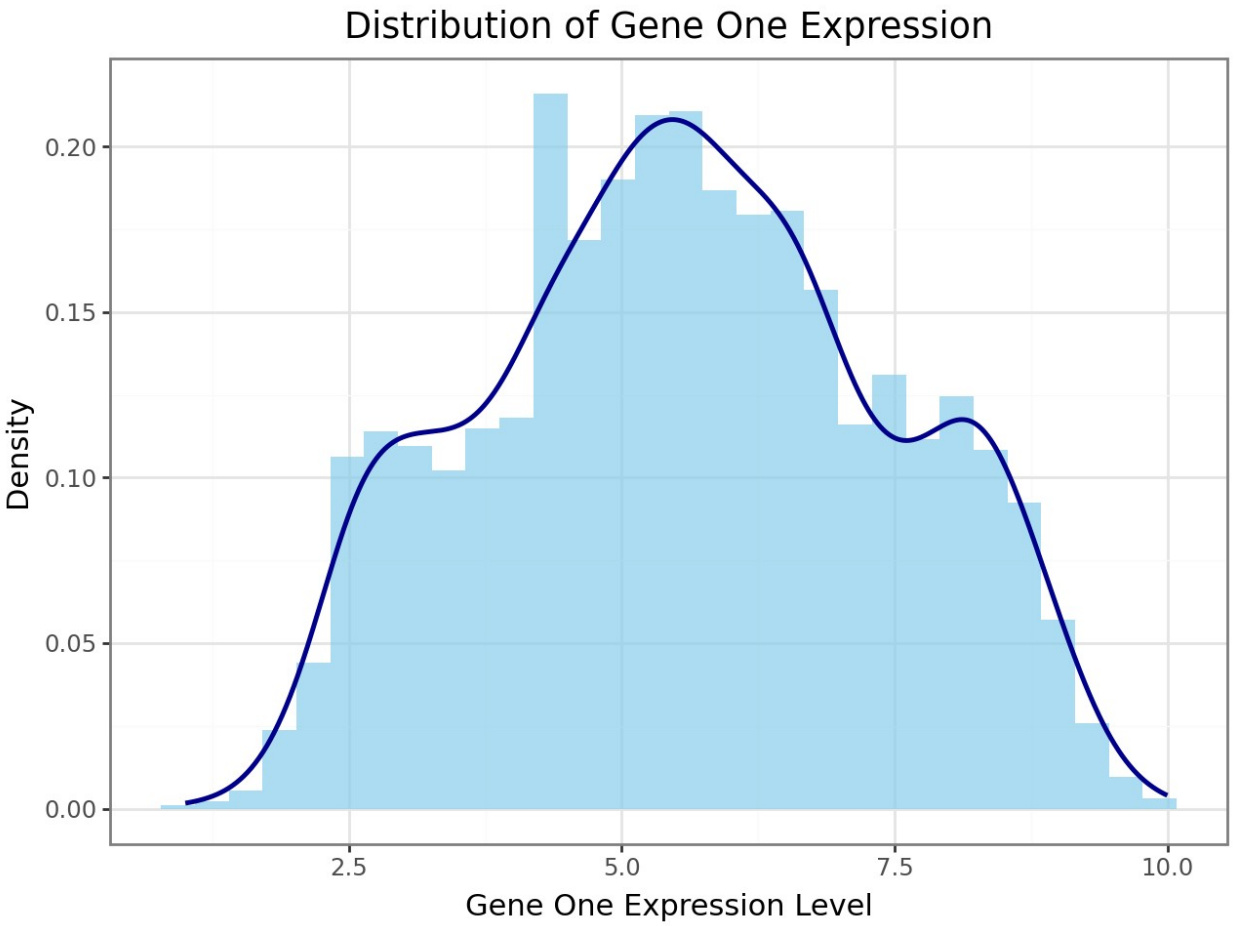
```
df.info()
```

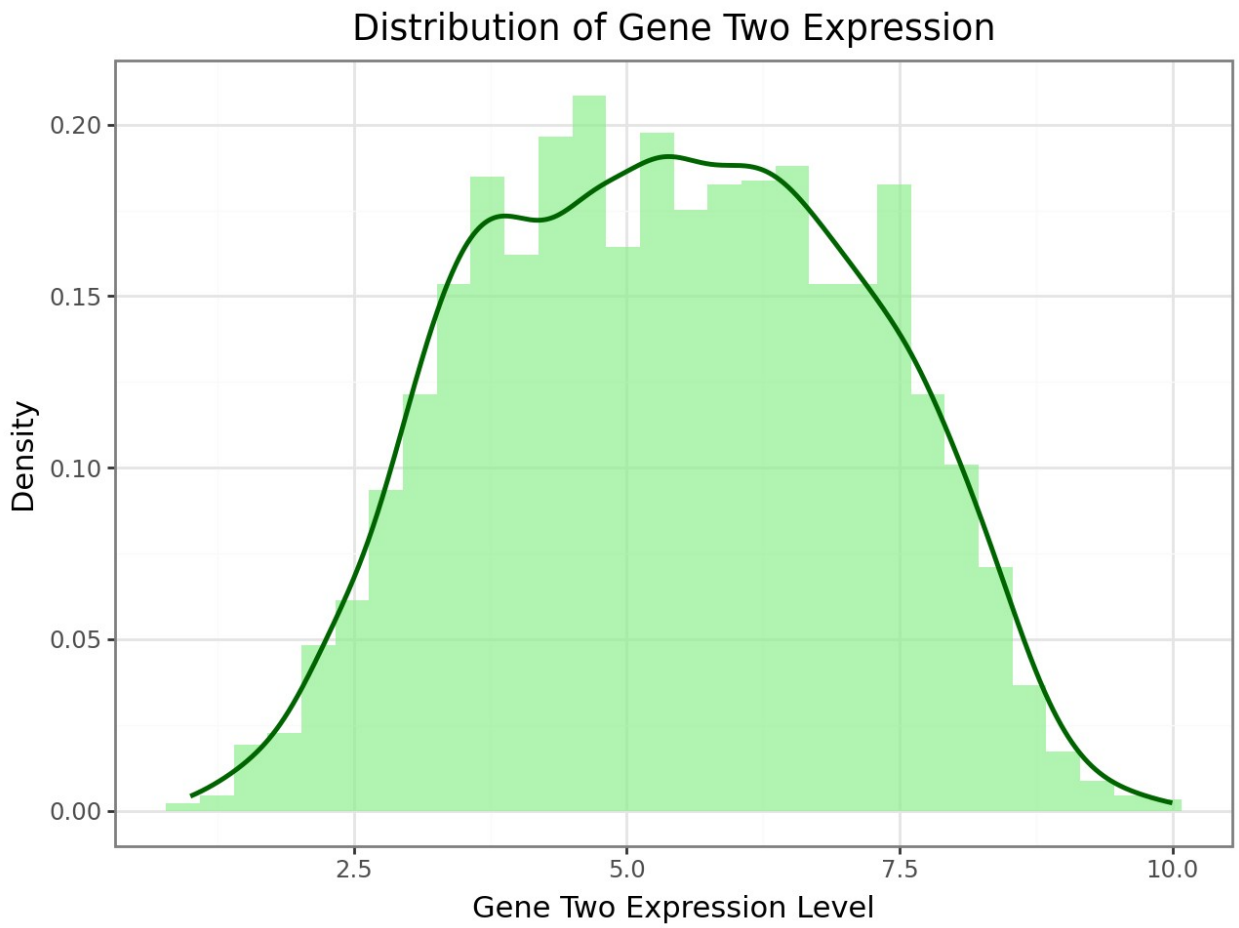
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gene One        3000 non-null   float64
 1   Gene Two        3000 non-null   float64
 2   Cancer Present  3000 non-null   int64  
dtypes: float64(2), int64(1)
memory usage: 70.4 KB
```

```
df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n{\n   \"column\": \"Gene One\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1058.8846317002517,\n   \"min\": 1.0,\n   \"max\": 3000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   5.600133333333334,\n   5.6,\n   3000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Gene Two\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1058.934579724635,\n   \"min\": 1.0,\n   \"max\": 3000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   5.410466666666666,\n   5.4,\n   3000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Cancer Present\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1060.4834582292306,\n   \"min\": 0.0,\n   \"max\": 3000.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.5,\n   1.0,\n   0.5000833541724554\n   ],\n

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

df['Cancer Present'].value_counts(normalize=True)

Cancer Present
1    0.5
0    0.5
Name: proportion, dtype: float64

plot_gene_one = (
    ggplot(df, aes(x='Gene One')) +
    geom_histogram(aes(y='..density..'), fill='skyblue', bins=30,
alpha=0.7) +
    geom_density(color='darkblue', size=1.0) +
    labs(title='Distribution of Gene One Expression', x='Gene One
Expression Level', y='Density') +
    theme_bw()
)

plot_gene_two = (
    ggplot(df, aes(x='Gene Two')) +
    geom_histogram(aes(y='..density..'), fill='lightgreen', bins=30,
alpha=0.7) +
    geom_density(color='darkgreen', size=1.0) +
    labs(title='Distribution of Gene Two Expression', x='Gene Two
Expression Level', y='Density') +
    theme_bw()
)

# Display plots
display(plot_gene_one, plot_gene_two)
```


Distribution of Gene One Expression
Density
0.20
0.15
0.10
0.05
0.00
2.5
5.0
7.5
10.0
Gene One Expression Level



Gene 1 is expressed consistently like a normal distribution. It may be constitutively expressed.

Gene 2 is expressed in a wider distribution meaning it varies differently in normal and disease states.

```
df_grouped = df.groupby('Cancer Present')[['Gene One', 'Gene
Two']].mean().reset_index()

# Melt the DataFrame to long format
df_bar = df_grouped.melt(id_vars='Cancer Present',
                         var_name='Gene',
                         value_name='Expression')

df_bar.rename(columns={'Cancer Present': 'Cancer'}, inplace=True)

# Plot
bar_chart = alt.Chart(df_bar).mark_bar().encode(
    x=alt.X('Gene:N', title='Gene'),
    y=alt.Y('Expression:Q', title='Mean Expression'),
    color=alt.Color('Cancer:N', scale=alt.Scale(range=['#1f77b4',
'#d62728'])),
    column=alt.Column('Cancer:N', title='Cancer Status'),
```

```
    tooltip=['Gene:N', 'Expression:Q', 'Cancer:N']
).properties(
    title="Mean Gene Expression by Cancer Status",
    width=150,
    height=300
)

bar_chart
```

```
alt.Chart(...)
```

```
scatter = alt.Chart(df).mark_circle(size=60).encode(
    x=alt.X('Gene One', title='Gene One Expression'),
    y=alt.Y('Gene Two', title='Gene Two Expression'),
    color=alt.Color('Cancer Present:N',
scale=alt.Scale(range=['#1f77b4', '#d62728']),
legend=alt.Legend(title='Cancer')),
    tooltip=['Gene One', 'Gene Two', 'Cancer Present']
).properties(
    width=500,
    height=400,
    title='Scatter Plot of Gene One vs Gene Two by Cancer Status'
).interactive()

scatter
```

```
alt.Chart(...)
```

```
scaler = StandardScaler()
X = scaler.fit_transform(df[['Gene One', 'Gene Two']])
y = df['Cancer Present']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42)

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict on test set
y_pred = log_reg.predict(X_test)
y_prob = log_reg.predict_proba(X_test)[:, 1]

# Evaluate model performance
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
print("\nROC AUC Score:", roc_auc_score(y_test, y_prob))
```

```
Confusion Matrix:
 [[325  50]
 [ 42 333]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       375
           1       0.87      0.89      0.88       375

    accuracy                           0.88       750
   macro avg       0.88      0.88      0.88       750
weighted avg       0.88      0.88      0.88       750


ROC AUC Score: 0.9485866666666666
```

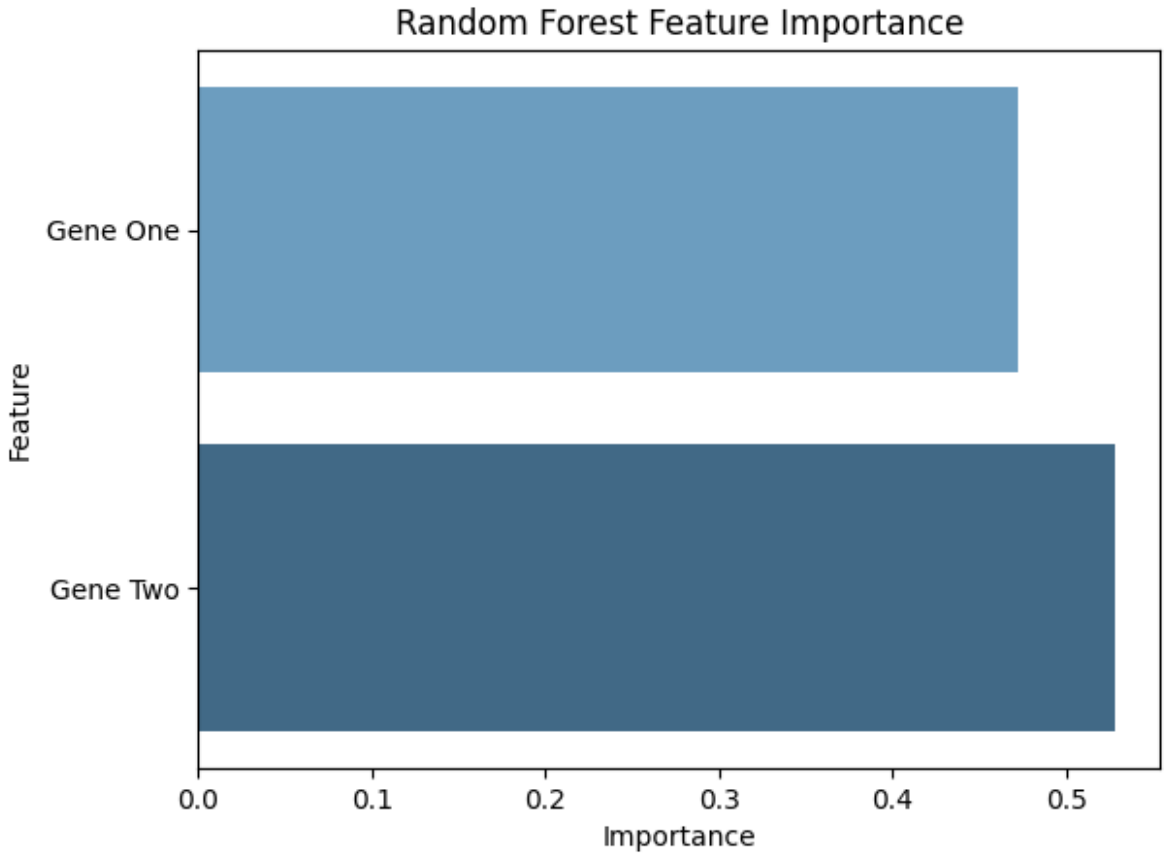
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predictions
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]

# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_rf))
print("\nROC AUC Score:", roc_auc_score(y_test, y_prob_rf))
```

```
Confusion Matrix:
 [[341  34]
 [ 25 350]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       375
           1       0.91      0.93      0.92       375

    accuracy                           0.92       750
   macro avg       0.92      0.92      0.92       750
weighted avg       0.92      0.92      0.92       750


ROC AUC Score: 0.9801386666666667
```

```
feat_df = pd.DataFrame({
    'Feature': ['Gene One', 'Gene Two'],
    'Importance': rf.feature_importances_
})

# Plotting
sns.barplot(data=feat_df, x='Importance', y='Feature',
```

```
palette='Blues_d')
plt.title('Random Forest Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

```
/tmp/ipython-input-19-1183823344.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```




```
tf.random.set_seed(42)

# Define model
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(16, activation='relu'),
    Dropout(0.3),
    Dense(8, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
```

```
])

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

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense (Dense)                        │ (None, 16)                  │
48 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 16)                  │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 8)                   │
136 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 8)                   │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ (None, 1)                   │
9 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 193 (772.00 B)

 Trainable params: 193 (772.00 B)

 Non-trainable params: 0 (0.00 B)
```

```
# Early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
```

```
    callbacks=[early_stop],
    verbose=1
)

Epoch 1/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.4681 - loss:
0.7219 - val_accuracy: 0.7156 - val_loss: 0.5741
Epoch 2/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6618 - loss:
0.5892 - val_accuracy: 0.8533 - val_loss: 0.4934
Epoch 3/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7215 - loss:
0.5367 - val_accuracy: 0.8644 - val_loss: 0.4346
Epoch 4/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7719 - loss:
0.4875 - val_accuracy: 0.8644 - val_loss: 0.3850
Epoch 5/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8112 - loss:
0.4384 - val_accuracy: 0.8667 - val_loss: 0.3492
Epoch 6/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8088 - loss:
0.4400 - val_accuracy: 0.8711 - val_loss: 0.3303
Epoch 7/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8293 - loss:
0.3940 - val_accuracy: 0.8711 - val_loss: 0.3186
Epoch 8/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8367 - loss:
0.4071 - val_accuracy: 0.8711 - val_loss: 0.3134
Epoch 9/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8157 - loss:
0.4122 - val_accuracy: 0.8711 - val_loss: 0.3107
Epoch 10/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8394 - loss:
0.3854 - val_accuracy: 0.8711 - val_loss: 0.3068
Epoch 11/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8341 - loss:
0.3905 - val_accuracy: 0.8711 - val_loss: 0.3033
Epoch 12/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8418 - loss:
0.3760 - val_accuracy: 0.8733 - val_loss: 0.2997
Epoch 13/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8298 - loss:
0.3859 - val_accuracy: 0.8733 - val_loss: 0.2979
Epoch 14/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8467 - loss:
0.3810 - val_accuracy: 0.8733 - val_loss: 0.2946
Epoch 15/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8462 - loss:
0.3578 - val_accuracy: 0.8733 - val_loss: 0.2910
Epoch 16/100
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8466 - loss:
0.3519 - val_accuracy: 0.8756 - val_loss: 0.2881
Epoch 17/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8413 - loss:
0.3628 - val_accuracy: 0.8778 - val_loss: 0.2848
Epoch 18/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8473 - loss:
0.3546 - val_accuracy: 0.8733 - val_loss: 0.2814
Epoch 19/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8450 - loss:
0.3589 - val_accuracy: 0.8800 - val_loss: 0.2794
Epoch 20/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8467 - loss:
0.3459 - val_accuracy: 0.8756 - val_loss: 0.2769
Epoch 21/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8492 - loss:
0.3626 - val_accuracy: 0.8733 - val_loss: 0.2744
Epoch 22/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8470 - loss:
0.3408 - val_accuracy: 0.8822 - val_loss: 0.2716
Epoch 23/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8600 - loss:
0.3387 - val_accuracy: 0.8800 - val_loss: 0.2691
Epoch 24/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8519 - loss:
0.3395 - val_accuracy: 0.8844 - val_loss: 0.2660
Epoch 25/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8585 - loss:
0.3302 - val_accuracy: 0.8844 - val_loss: 0.2631
Epoch 26/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8622 - loss:
0.3372 - val_accuracy: 0.8844 - val_loss: 0.2611
Epoch 27/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8511 - loss:
0.3361 - val_accuracy: 0.8822 - val_loss: 0.2587
Epoch 28/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8567 - loss:
0.3321 - val_accuracy: 0.8911 - val_loss: 0.2559
Epoch 29/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8627 - loss:
0.3175 - val_accuracy: 0.8933 - val_loss: 0.2533
Epoch 30/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8611 - loss:
0.3225 - val_accuracy: 0.8933 - val_loss: 0.2507
Epoch 31/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8596 - loss:
0.3291 - val_accuracy: 0.8956 - val_loss: 0.2497
Epoch 32/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8519 - loss:
```

```
0.3351 - val_accuracy: 0.8933 - val_loss: 0.2479
Epoch 33/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8601 - loss:
0.3255 - val_accuracy: 0.8933 - val_loss: 0.2454
Epoch 34/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8757 - loss:
0.3050 - val_accuracy: 0.8933 - val_loss: 0.2444
Epoch 35/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8540 - loss:
0.3074 - val_accuracy: 0.8956 - val_loss: 0.2430
Epoch 36/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8652 - loss:
0.3224 - val_accuracy: 0.8956 - val_loss: 0.2418
Epoch 37/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8617 - loss:
0.3240 - val_accuracy: 0.8978 - val_loss: 0.2404
Epoch 38/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8580 - loss:
0.3259 - val_accuracy: 0.9022 - val_loss: 0.2392
Epoch 39/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8634 - loss:
0.3050 - val_accuracy: 0.9022 - val_loss: 0.2359
Epoch 40/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8776 - loss:
0.3013 - val_accuracy: 0.9089 - val_loss: 0.2334
Epoch 41/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8770 - loss:
0.3025 - val_accuracy: 0.9133 - val_loss: 0.2312
Epoch 42/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8807 - loss:
0.3000 - val_accuracy: 0.9111 - val_loss: 0.2308
Epoch 43/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8669 - loss:
0.3002 - val_accuracy: 0.9133 - val_loss: 0.2288
Epoch 44/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8577 - loss:
0.3128 - val_accuracy: 0.9133 - val_loss: 0.2271
Epoch 45/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8800 - loss:
0.3086 - val_accuracy: 0.9133 - val_loss: 0.2260
Epoch 46/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8739 - loss:
0.3033 - val_accuracy: 0.9156 - val_loss: 0.2252
Epoch 47/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8754 - loss:
0.2986 - val_accuracy: 0.9156 - val_loss: 0.2248
Epoch 48/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8852 - loss:
0.2853 - val_accuracy: 0.9156 - val_loss: 0.2237
```

```
Epoch 49/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8799 - loss:
0.3010 - val_accuracy: 0.9156 - val_loss: 0.2235
Epoch 50/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8719 - loss:
0.2922 - val_accuracy: 0.9178 - val_loss: 0.2210
Epoch 51/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8791 - loss:
0.2985 - val_accuracy: 0.9178 - val_loss: 0.2208
Epoch 52/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8719 - loss:
0.3000 - val_accuracy: 0.9178 - val_loss: 0.2196
Epoch 53/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8839 - loss:
0.2935 - val_accuracy: 0.9200 - val_loss: 0.2191
Epoch 54/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8800 - loss:
0.3051 - val_accuracy: 0.9200 - val_loss: 0.2191
Epoch 55/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8800 - loss:
0.3083 - val_accuracy: 0.9222 - val_loss: 0.2183
Epoch 56/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8735 - loss:
0.2969 - val_accuracy: 0.9222 - val_loss: 0.2183
Epoch 57/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8865 - loss:
0.2937 - val_accuracy: 0.9200 - val_loss: 0.2162
Epoch 58/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8780 - loss:
0.2990 - val_accuracy: 0.9222 - val_loss: 0.2157
Epoch 59/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8871 - loss:
0.2825 - val_accuracy: 0.9222 - val_loss: 0.2135
Epoch 60/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8808 - loss:
0.2958 - val_accuracy: 0.9244 - val_loss: 0.2127
Epoch 61/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8890 - loss:
0.2913 - val_accuracy: 0.9222 - val_loss: 0.2111
Epoch 62/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8874 - loss:
0.2943 - val_accuracy: 0.9244 - val_loss: 0.2115
Epoch 63/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8876 - loss:
0.2861 - val_accuracy: 0.9244 - val_loss: 0.2121
Epoch 64/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8925 - loss:
0.2900 - val_accuracy: 0.9267 - val_loss: 0.2112
Epoch 65/100
```

```
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8749 - loss:
0.2973 - val_accuracy: 0.9267 - val_loss: 0.2110
Epoch 66/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8912 - loss:
0.2793 - val_accuracy: 0.9267 - val_loss: 0.2089
Epoch 67/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8808 - loss:
0.3021 - val_accuracy: 0.9289 - val_loss: 0.2094
Epoch 68/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8855 - loss:
0.3023 - val_accuracy: 0.9289 - val_loss: 0.2089
Epoch 69/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8752 - loss:
0.2975 - val_accuracy: 0.9267 - val_loss: 0.2088
Epoch 70/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8854 - loss:
0.2871 - val_accuracy: 0.9244 - val_loss: 0.2068
Epoch 71/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8892 - loss:
0.2771 - val_accuracy: 0.9244 - val_loss: 0.2071
Epoch 72/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8874 - loss:
0.2904 - val_accuracy: 0.9267 - val_loss: 0.2067
Epoch 73/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8837 - loss:
0.2855 - val_accuracy: 0.9244 - val_loss: 0.2076
Epoch 74/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8892 - loss:
0.2927 - val_accuracy: 0.9244 - val_loss: 0.2088
Epoch 75/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8878 - loss:
0.2758 - val_accuracy: 0.9222 - val_loss: 0.2072
Epoch 76/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8917 - loss:
0.2924 - val_accuracy: 0.9222 - val_loss: 0.2067
Epoch 77/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8868 - loss:
0.2965 - val_accuracy: 0.9222 - val_loss: 0.2073
Epoch 78/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8708 - loss:
0.2945 - val_accuracy: 0.9222 - val_loss: 0.2081
Epoch 79/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8855 - loss:
0.3055 - val_accuracy: 0.9244 - val_loss: 0.2072
Epoch 80/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8746 - loss:
0.2965 - val_accuracy: 0.9244 - val_loss: 0.2086
Epoch 81/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8819 - loss:
```

```
0.2950 - val_accuracy: 0.9244 - val_loss: 0.2081
Epoch 82/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8796 - loss:
0.2988 - val_accuracy: 0.9244 - val_loss: 0.2062
Epoch 83/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8942 - loss:
0.2800 - val_accuracy: 0.9222 - val_loss: 0.2041
Epoch 84/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8811 - loss:
0.2776 - val_accuracy: 0.9222 - val_loss: 0.2028
Epoch 85/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8862 - loss:
0.2976 - val_accuracy: 0.9200 - val_loss: 0.2033
Epoch 86/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8890 - loss:
0.2979 - val_accuracy: 0.9222 - val_loss: 0.2033
Epoch 87/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8804 - loss:
0.2860 - val_accuracy: 0.9200 - val_loss: 0.2031
Epoch 88/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8875 - loss:
0.2918 - val_accuracy: 0.9222 - val_loss: 0.2023
Epoch 89/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8892 - loss:
0.2820 - val_accuracy: 0.9244 - val_loss: 0.2019
Epoch 90/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9021 - loss:
0.2682 - val_accuracy: 0.9244 - val_loss: 0.1998
Epoch 91/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8844 - loss:
0.2870 - val_accuracy: 0.9244 - val_loss: 0.2001
Epoch 92/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8900 - loss:
0.2953 - val_accuracy: 0.9244 - val_loss: 0.1996
Epoch 93/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8903 - loss:
0.2961 - val_accuracy: 0.9267 - val_loss: 0.1993
Epoch 94/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8890 - loss:
0.2656 - val_accuracy: 0.9222 - val_loss: 0.2003
Epoch 95/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8870 - loss:
0.2868 - val_accuracy: 0.9222 - val_loss: 0.2008
Epoch 96/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8947 - loss:
0.2769 - val_accuracy: 0.9244 - val_loss: 0.2005
Epoch 97/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8941 - loss:
0.2807 - val_accuracy: 0.9222 - val_loss: 0.2008
```

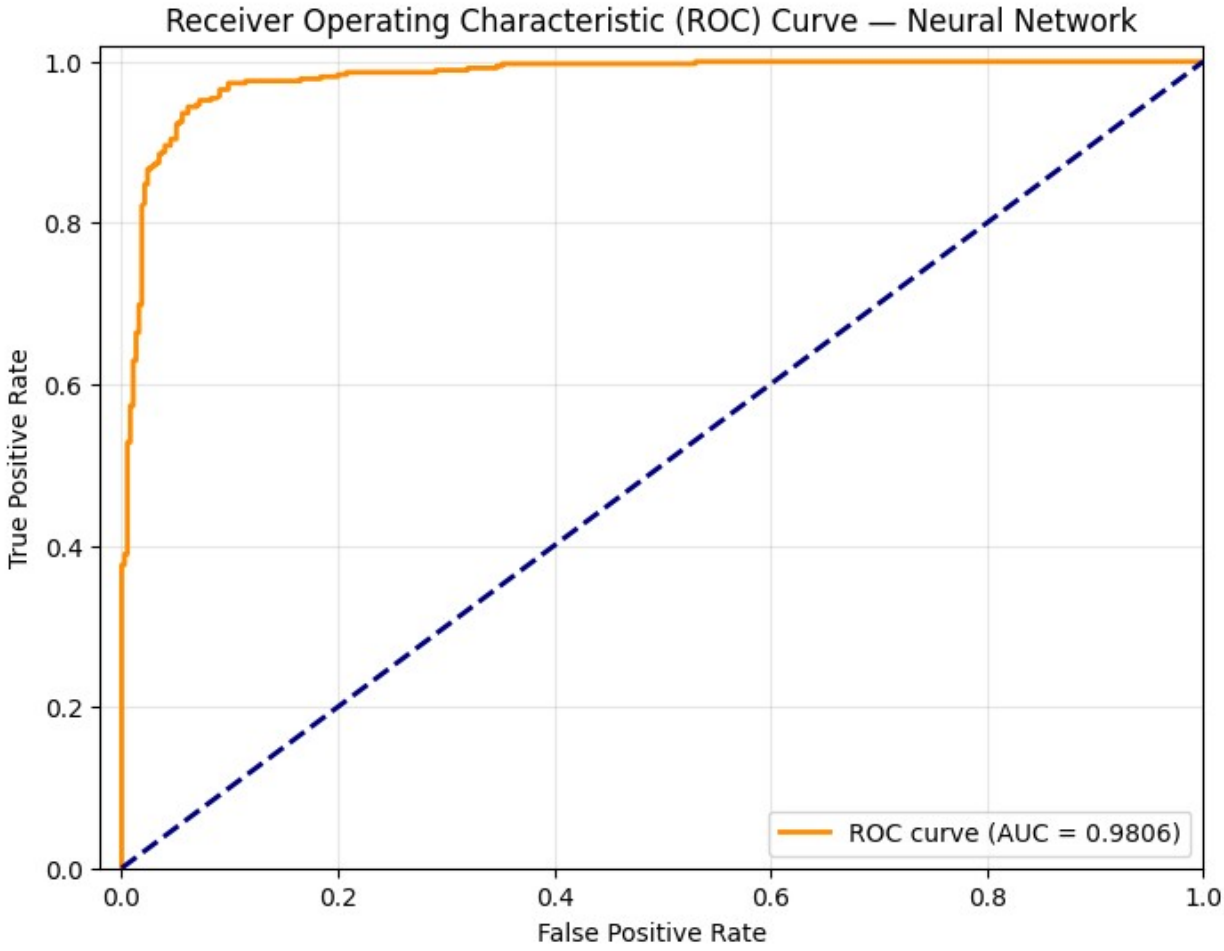
```
Epoch 98/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9024 - loss:
0.2739 - val_accuracy: 0.9222 - val_loss: 0.1999
Epoch 99/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8788 - loss:
0.2819 - val_accuracy: 0.9244 - val_loss: 0.2020
Epoch 100/100
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8769 - loss:
0.2995 - val_accuracy: 0.9244 - val_loss: 0.2025
```

```
# Predictions
y_pred_nn = (model.predict(X_test) > 0.5).astype("int32").flatten()
y_prob_nn = model.predict(X_test).flatten()

# Metrics
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nn))
print("\nClassification Report:\n", classification_report(y_test,
y_pred_nn))
print("\nROC AUC Score:", roc_auc_score(y_test, y_prob_nn))
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Confusion Matrix:
 [[351  24]
 [ 21 354]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       375
           1       0.94      0.94      0.94       375

    accuracy                           0.94       750
   macro avg       0.94      0.94      0.94       750
weighted avg       0.94      0.94      0.94       750


ROC AUC Score: 0.9806079999999999
```

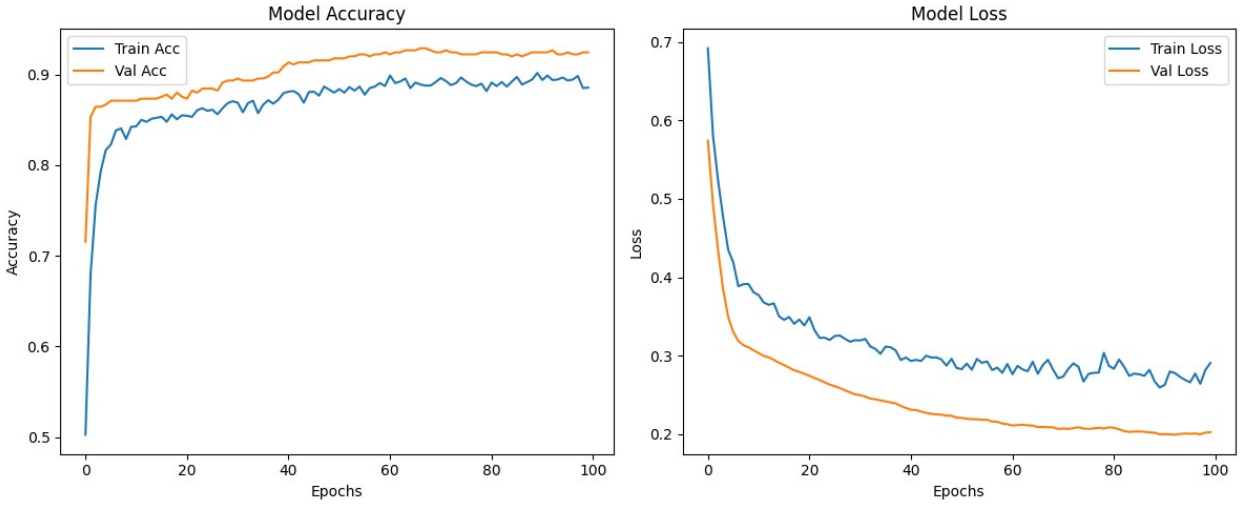
```
# Compute false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_prob_nn)

# Compute AUC
roc_auc = auc(fpr, tpr)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2,
         label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([-0.02, 1.0])
```

```
plt.ylim([0.0, 1.02])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve — Neural
Network')
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.show()
```

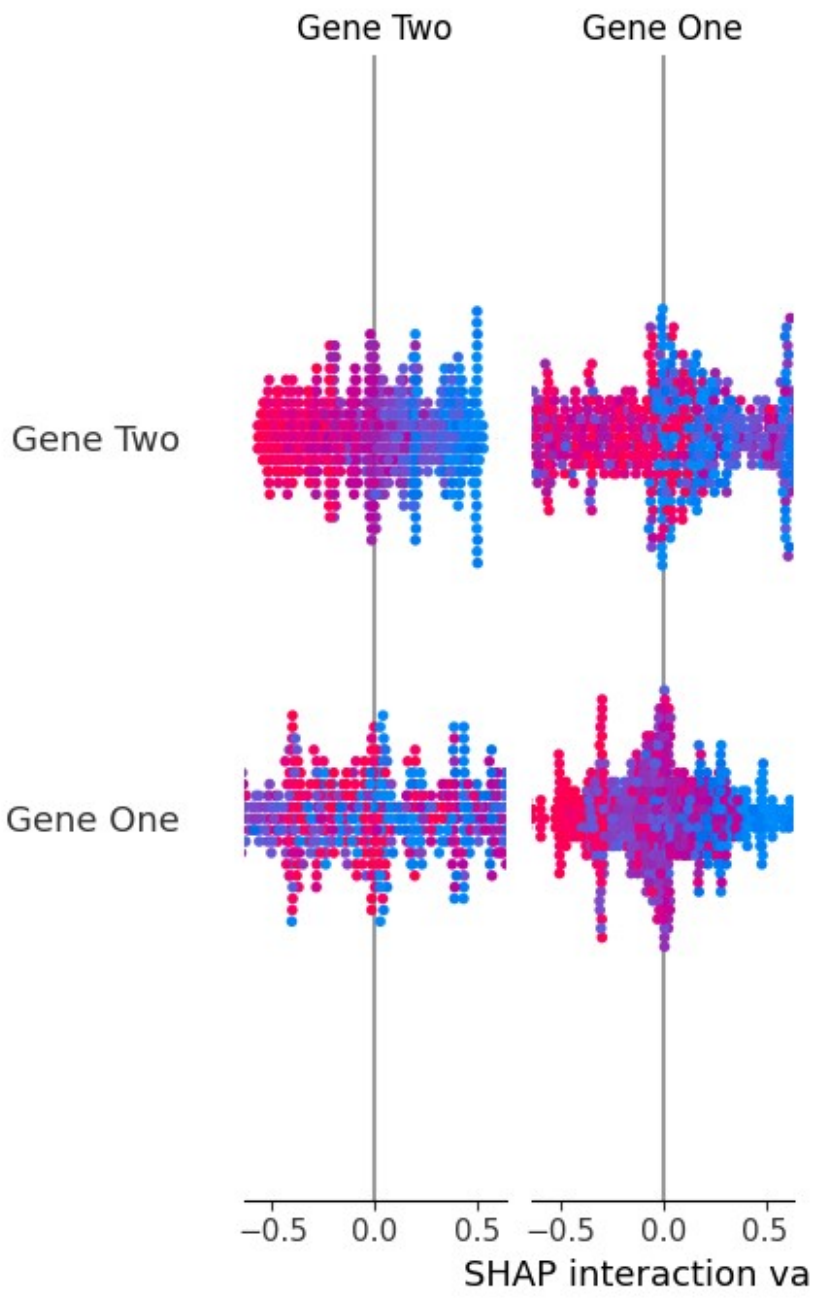



```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
```

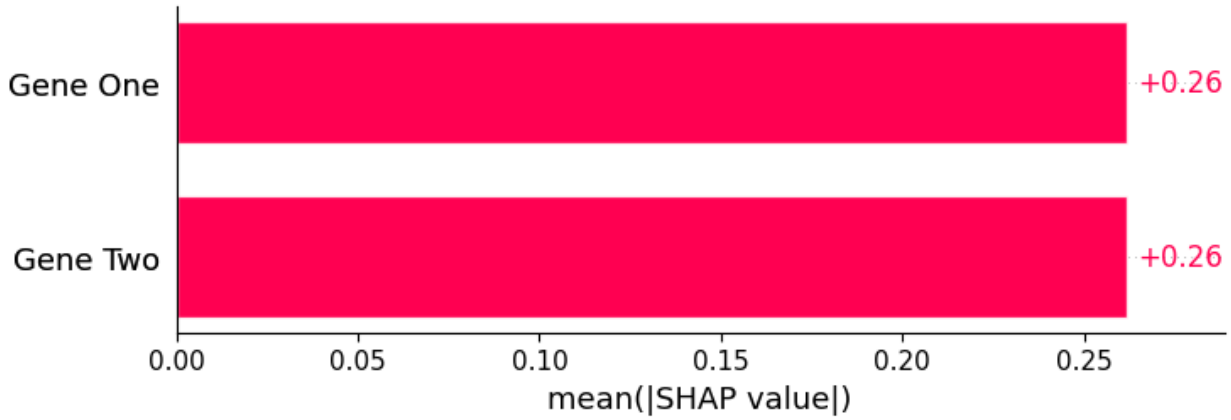
```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```




```
explainer_rf = shap.TreeExplainer(rf)
shap_values_rf = explainer_rf.shap_values(X_test)


shap.summary_plot(shap_values_rf, features=X_test,
feature_names=['Gene One', 'Gene Two'])
```




```
explanation = shap.Explanation(
    values=shap_values_rf[1],
    base_values=explainer_rf.expected_value[1],
    data=X_test,
    feature_names=['Gene One', 'Gene Two']
)

# Display the chart
shap.plots.bar(explanation)
```



This bar chart ranks features by their average absolute impact on predictions, providing a clear view of global importance across all predictions.

```
# Subset for KernelExplainer
X_sample = X_test[:100]

# Background dataset for SHAP
background = X_train[np.random.choice(X_train.shape[0], 100,
replace=False)]

# Create KernelExplainer
explainer_nn = shap.KernelExplainer(model.predict, background)
shap_values_nn = explainer_nn.shap_values(X_sample, nsamples=100)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
```


```
{"model_id":"0d0c4aa3530f47a39b2e209d29d77337","version_major":2,"vers
ion_minor":0}
```


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

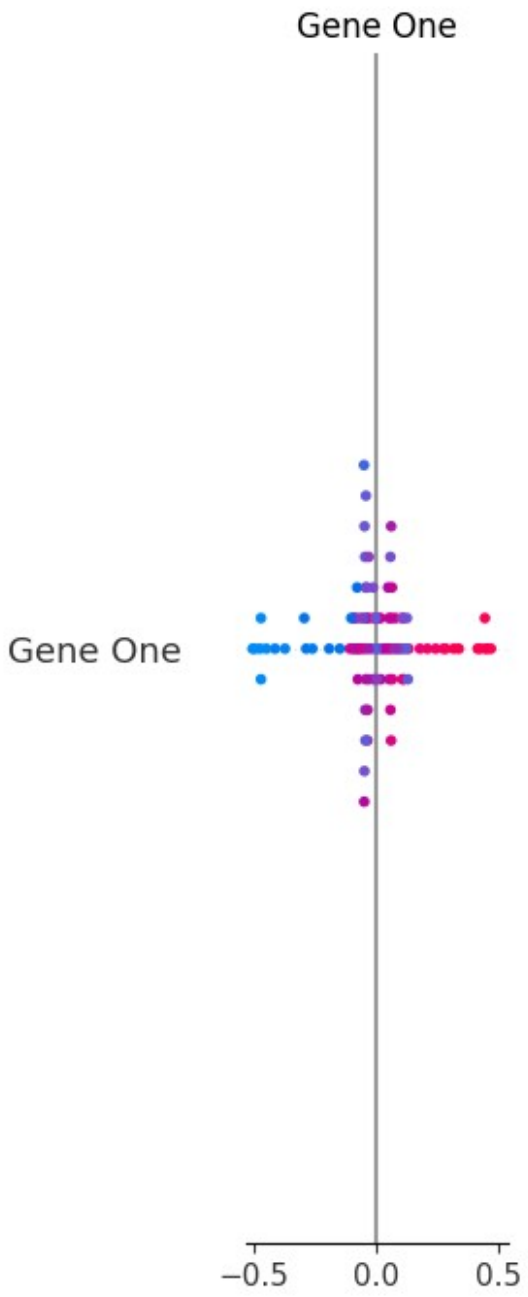
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 91ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 142ms/step
```

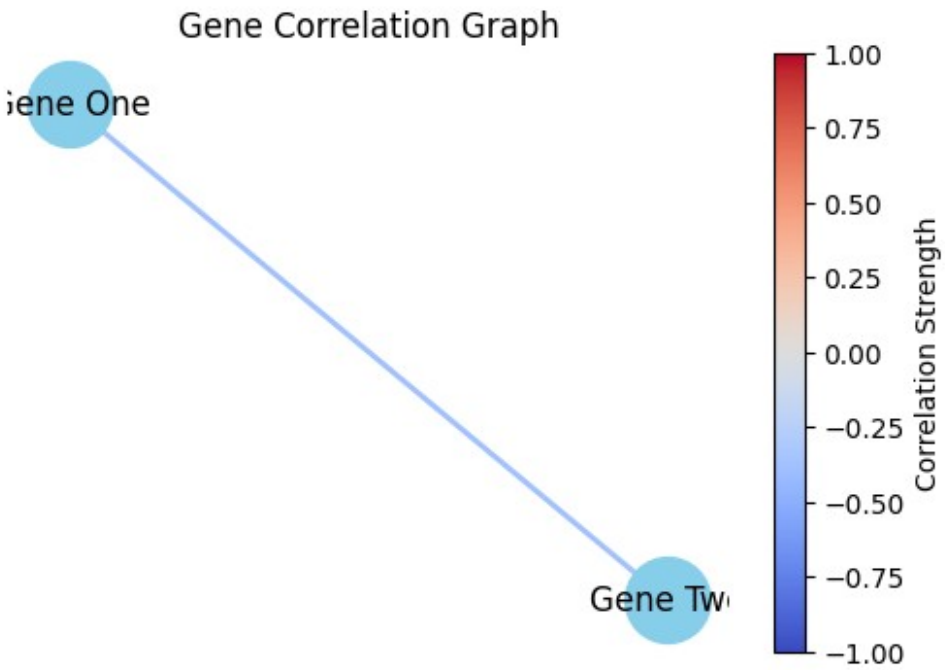
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 145ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 263ms/step
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
# For binary classification, KernelExplainer returns one array:
shap.summary_plot(shap_values_nn, features=X_sample,
feature_names=['Gene One', 'Gene Two'])
```




```
# Compute correlation matrix
gene_corr = df[['Gene One', 'Gene Two']].corr()

# Initialized the graph
G = nx.Graph()

# Add nodes
for gene in gene_corr.columns:
    G.add_node(gene)
```

```
# Add edges for correlations above a threshold
threshold = 0.2
for i in gene_corr.columns:
    for j in gene_corr.columns:
        if i != j and abs(gene_corr.loc[i, j]) > threshold:
            G.add_edge(i, j, weight=gene_corr.loc[i, j])

# Generate positions
pos = nx.spring_layout(G, seed=42)

# Plot nodes
plt.figure(figsize=(6, 4))
nx.draw_networkx_nodes(G, pos, node_color='skyblue', node_size=1000)

# Add labels to nodes
nx.draw_networkx_labels(G, pos, font_size=12)

# Plot edges with colors
edges = nx.draw_networkx_edges(
    G, pos, width=2,
    edge_color=[G[u][v]['weight'] for u, v in G.edges()],
    edge_cmap=plt.cm.coolwarm,
    edge_vmin=-1, edge_vmax=1
)

# Add colorbar
plt.colorbar(edges, label='Correlation Strength')

plt.title("Gene Correlation Graph")
plt.axis('off')
plt.show()
```



Gene One and Gene Two are negatively correlated, meaning as the expression of one increases, the expression of the other tends to decrease.

The correlation is not extremely strong (not dark blue), but it is still meaningfully negative.

Future Work

To expand causal modeling:

Use Bayesian Networks for probabilistic causal inference.

Introduce counterfactual analysis.

Integrate do-calculus for intervention simulation.

Combine causal graphs with SHAP to build counterfactual explanations.