In [ ]:
```
dir='/kaggle/input/gender-classification-dataset/gender_classification_v7.csv'
```

In [ ]:
```
import pandas as pd
import numpy as np
import missingno as msng
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
```

In [ ]:
```
df = pd.read_csv(dir)
df.head()
```

Out[ ]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female

Problem Introduction:

Objective: Predict gender based on facial attributes.

Dataset Features:

- 'long_hair': Presence of long hair (1: Yes, 0: No).
- 'forehead_width_cm' and 'forehead_height_cm': Dimensions of the forehead.
- 'nose_wide' and 'nose_long': Width and length of the nose.
- 'lips_thin': Thinness of the lips.
- 'distance_nose_to_lip_long': Length of the distance from nose to lip.

Target Variable: 'gender' (Male or Female).

Data Set Description:

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   long_hair                  5001 non-null   int64
 1   forehead_width_cm          5001 non-null   float64
 2   forehead_height_cm         5001 non-null   float64
 3   nose_wide                  5001 non-null   int64
 4   nose_long                  5001 non-null   int64
 5   lips_thin                  5001 non-null   int64
 6   distance_nose_to_lip_long  5001 non-null   int64
 7   gender                     5001 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

In [ ]:
```
df.describe()
```

Out[ ]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long
count	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000	5001.000000
mean	0.869626	13.181484	5.946311	0.493901	0.507898	0.493101	0.498900
std	0.336748	1.107128	0.541268	0.500013	0.499988	0.500002	0.500049
min	0.000000	11.400000	5.100000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	12.200000	5.500000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	13.100000	5.900000	0.000000	1.000000	0.000000	0.000000
75%	1.000000	14.000000	6.400000	1.000000	1.000000	1.000000	1.000000
max	1.000000	15.500000	7.100000	1.000000	1.000000	1.000000	1.000000

Visualizations:

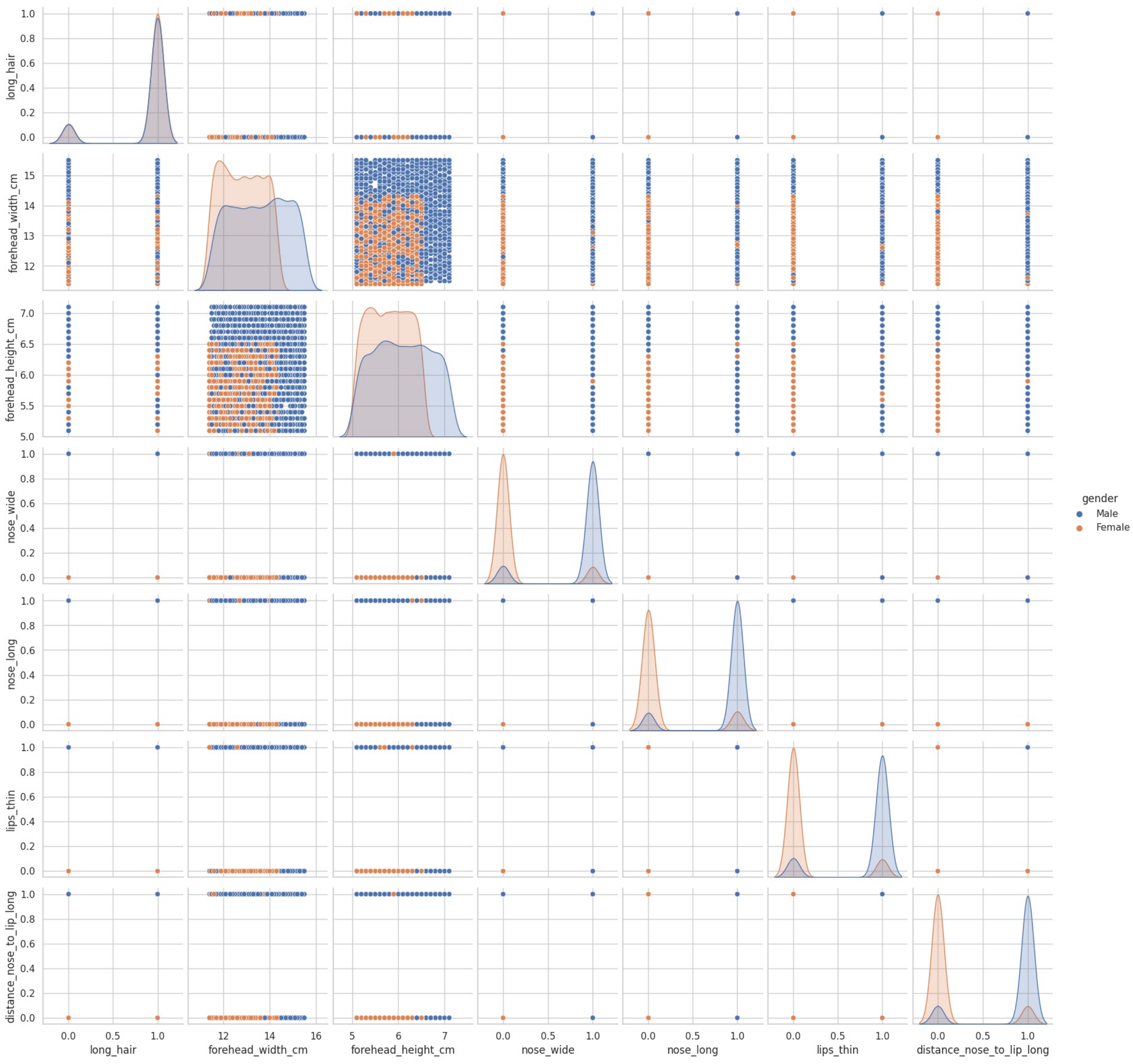
In [ ]:

```
# Set the style for seaborn
sns.set(style="whitegrid")
```

In [ ]:

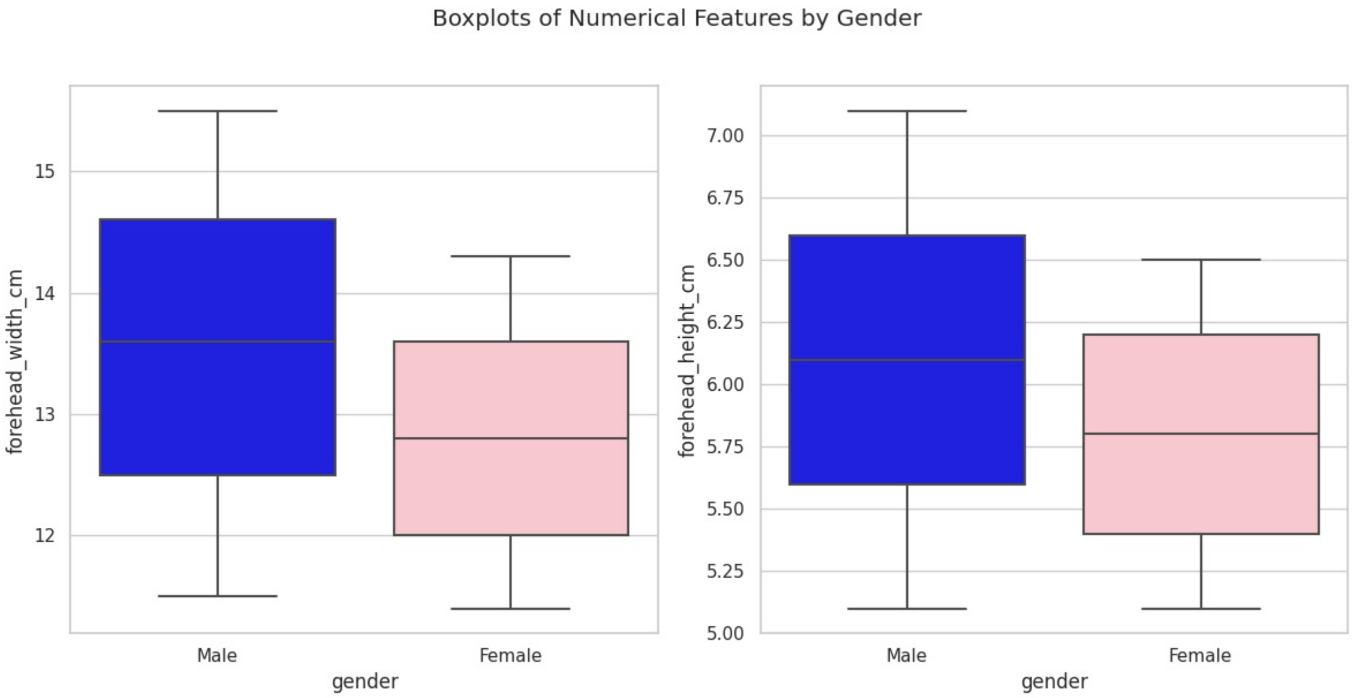
```
#Pair Plot for all numerical features
sns.pairplot(df, hue='gender', diag_kind='kde')
plt.suptitle('Pair Plot of Numerical Features by Gender', y=1.02)
plt.show()
```

Pair Plot of Numerical Features by Gender



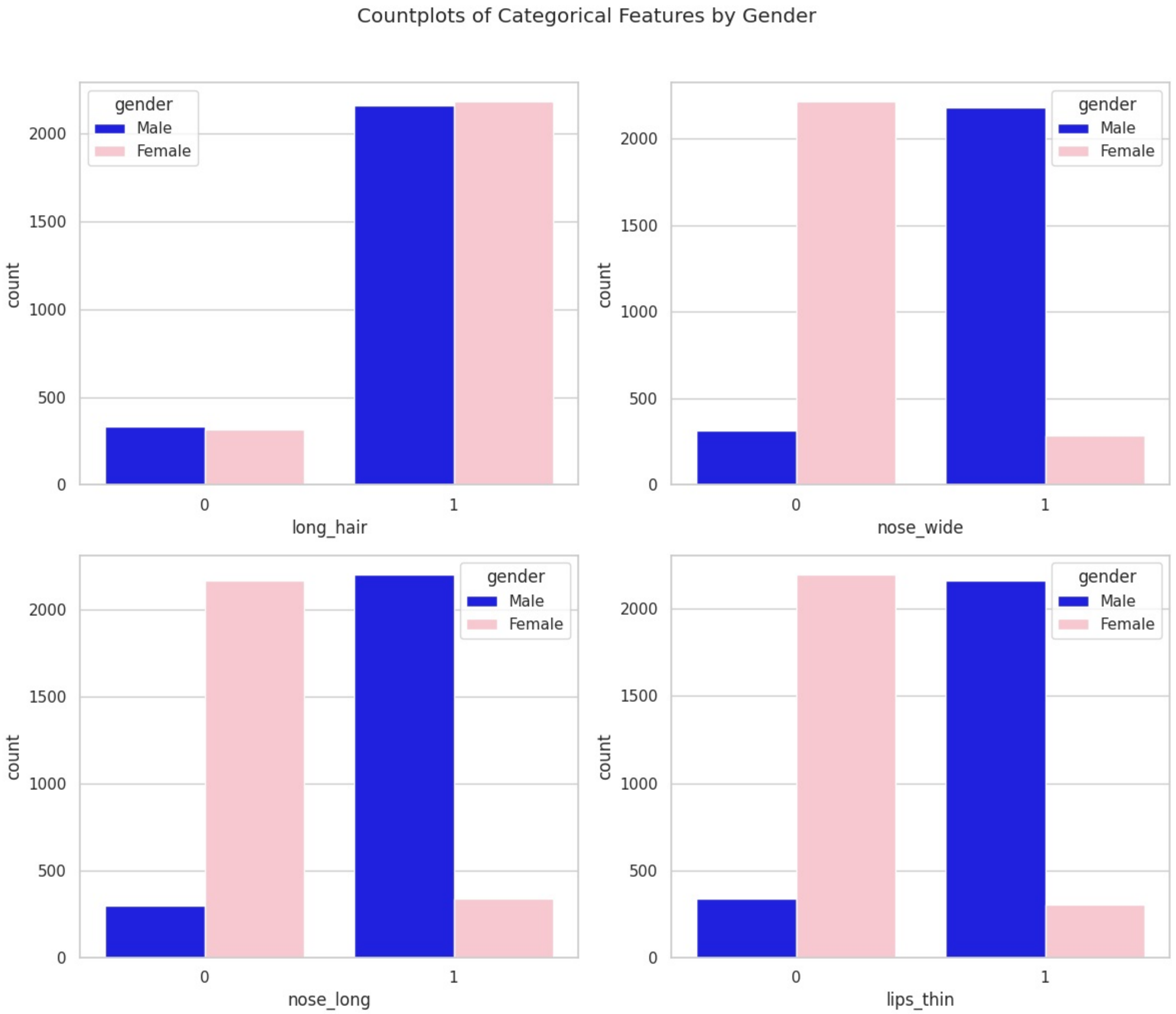
In [ ]:

```
# Boxplot for numerical features by gender
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
sns.boxplot(x="gender", y="forehead_width_cm", data=df, ax=axes[0], palette={"Male": "blue", "Female": "pink"})
sns.boxplot(x="gender", y="forehead_height_cm", data=df, ax=axes[1], palette={"Male": "blue", "Female": "pink"}
plt.suptitle("Boxplots of Numerical Features by Gender", y=1.02)
plt.tight_layout()
plt.show()
```

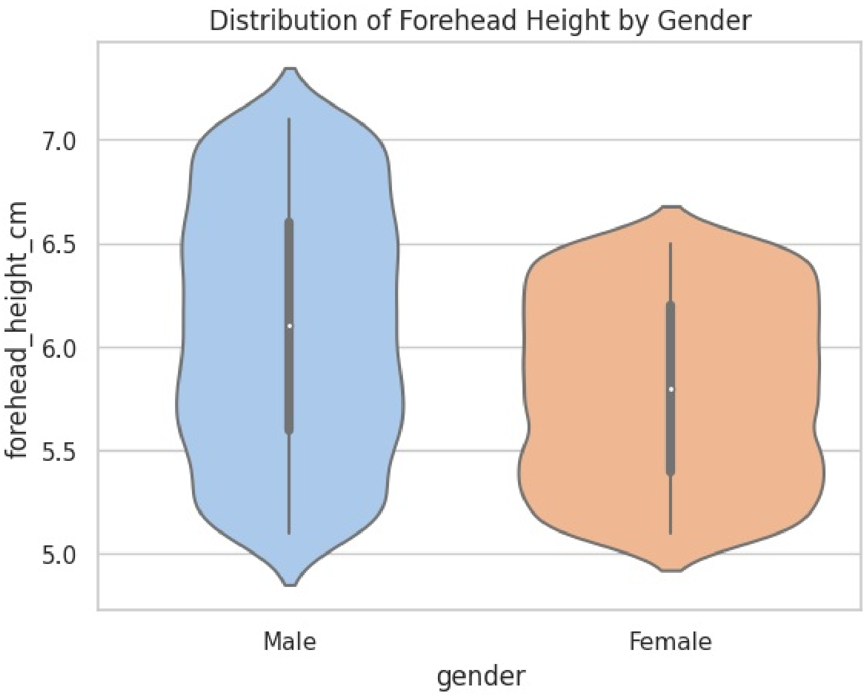


In [ ]:

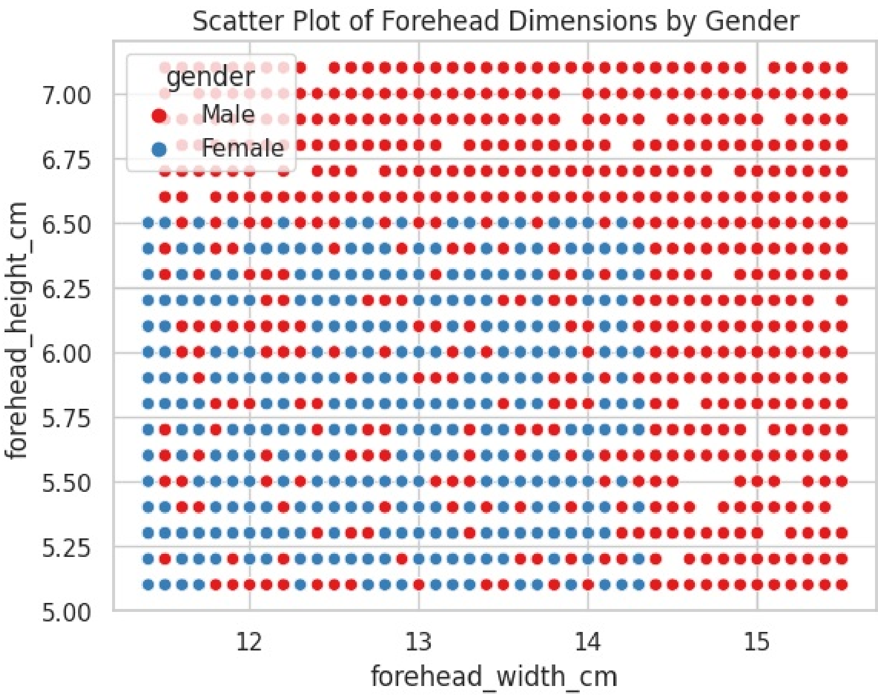
```python
# Countplot for categorical features
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))
sns.countplot(x="long_hair", hue="gender", data=df, ax=axes[0, 0], palette={"Male": "blue", "Female": "pink"})
sns.countplot(x="nose_wide", hue="gender", data=df, ax=axes[0, 1], palette={"Male": "blue", "Female": "pink"})
sns.countplot(x="nose_long", hue="gender", data=df, ax=axes[1, 0], palette={"Male": "blue", "Female": "pink"})
sns.countplot(x="lips_thin", hue="gender", data=df, ax=axes[1, 1], palette={"Male": "blue", "Female": "pink"})
plt.suptitle("Countplots of Categorical Features by Gender", y=1.02)
plt.tight_layout()
plt.show()
```



In [ ]:
```python
#Violin Plot for 'forehead_height_cm' by gender
sns.violinplot(x='gender', y='forehead_height_cm', data=df, palette='pastel')
plt.title('Distribution of Forehead Height by Gender')
plt.show()
```

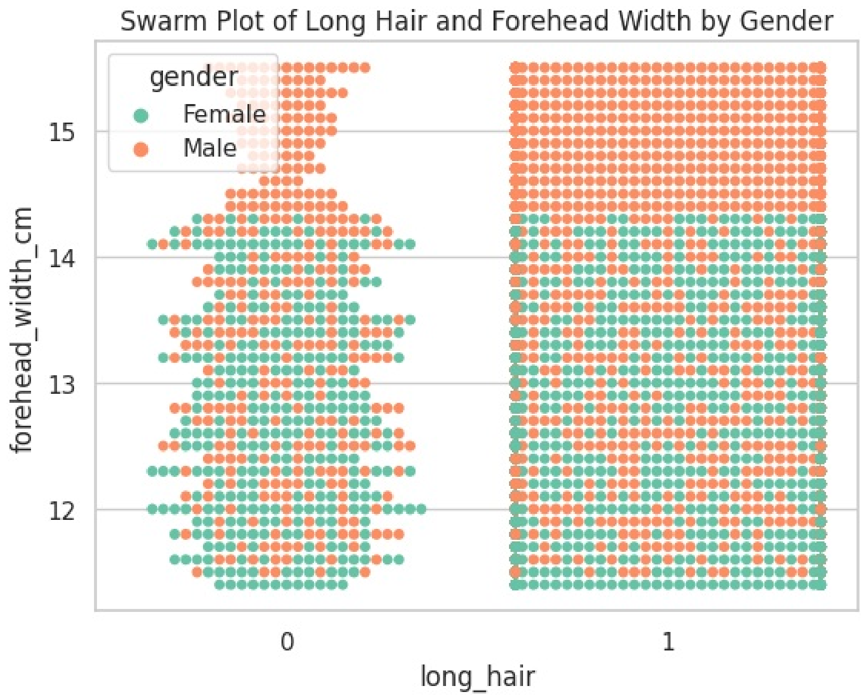


In [ ]:
```python
# Scatter Plot for 'forehead_width_cm' vs 'forehead_height_cm'
sns.scatterplot(x='forehead_width_cm', y='forehead_height_cm', hue='gender', data=df, palette='Set1')
plt.title('Scatter Plot of Forehead Dimensions by Gender')
plt.show()
```

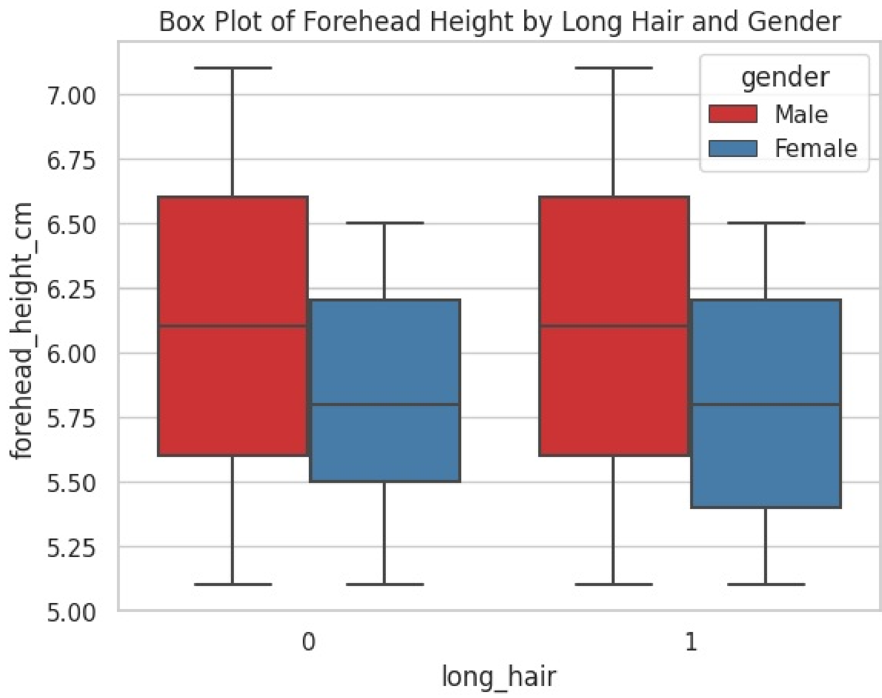


In [ ]:
```python
#Swarm Plot for 'long_hair' and 'forehead_width_cm' by gender
sns.swarmplot(x='long_hair', y='forehead_width_cm', hue='gender', data=df, palette='Set2')
plt.title('Swarm Plot of Long Hair and Forehead Width by Gender')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/categorical.py:3544: UserWarning: 52.8% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/opt/conda/lib/python3.10/site-packages/seaborn/categorical.py:3544: UserWarning: 73.9% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



In [ ]:
```python
#Box Plot for 'forehead_height_cm' by 'long_hair' and 'gender'
sns.boxplot(x='long_hair', y='forehead_height_cm', hue='gender', data=df, palette='Set1')
plt.title('Box Plot of Forehead Height by Long Hair and Gender')
plt.show()
```



Missing Values Treatment

In [ ]:
```python
df.isnull().sum()
```

Out[ ]:
```
long_hair                    0
forehead_width_cm            0
forehead_height_cm           0
nose_wide                    0
nose_long                    0
lips_thin                    0
distance_nose_to_lip_long    0
gender                       0
dtype: int64
```

No Missing Values

*****Binning*****

In [ ]:
```python
# Example
# df['forehead_width_bin'] = pd.cut(df['forehead_width_cm'], bins=5, labels=False)
```

Data Analysis

In [ ]:
```python
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns
```

In [ ]:
```python
summary_statistics = df[numerical_columns].agg(['min', 'max', 'mean', 'var', 'std', 'skew', 'kurt'])
```

In [ ]:
```python
print("Summary Statistics:")
print(summary_statistics)
```

```
Summary Statistics:
      long_hair  forehead_width_cm  forehead_height_cm  nose_wide  nose_long  \
min    0.000000          11.400000            5.100000   0.000000   0.000000
max    1.000000          15.500000            7.100000   1.000000   1.000000
mean   0.869626          13.181484            5.946311   0.493901   0.507898
var    0.113399           1.225733            0.292971   0.250013   0.249988
std    0.336748           1.107128            0.541268   0.500013   0.499988
skew  -2.196146           0.242242            0.250739   0.024404  -0.031607
kurt   2.824187          -0.930596           -0.848889  -2.000205  -1.999801

      lips_thin  distance_nose_to_lip_long
min    0.000000                   0.000000
max    1.000000                   1.000000
mean   0.493101                   0.498900
var    0.250002                   0.250049
std    0.500002                   0.500049
skew   0.027605                   0.004400
kurt  -2.000038                  -2.000781
```

Data Analysis

In [ ]:
```python
from scipy.stats import chi2_contingency, ttest_ind, f_oneway
```

In [ ]:
```python
# Covariance Matrix
covariance_matrix = df[numerical_columns].cov()
covariance_matrix
```

Out[ ]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_
long_hair	0.113399	-0.002435	-0.003141	0.000205	0.002430	0.001900	-0
forehead_width_cm	-0.002435	1.225733	0.053092	0.139307	0.142466	0.143132	0
forehead_height_cm	-0.003141	0.053092	0.292971	0.057282	0.052534	0.055600	0
nose_wide	0.000205	0.139307	0.057282	0.250013	0.141298	0.139408	0
nose_long	0.002430	0.142466	0.052534	0.141298	0.249988	0.140304	0
lips_thin	0.001900	0.143132	0.055600	0.139408	0.140304	0.250002	0
distance_nose_to_lip_long	-0.004343	0.139140	0.058271	0.142343	0.139959	0.141342	0

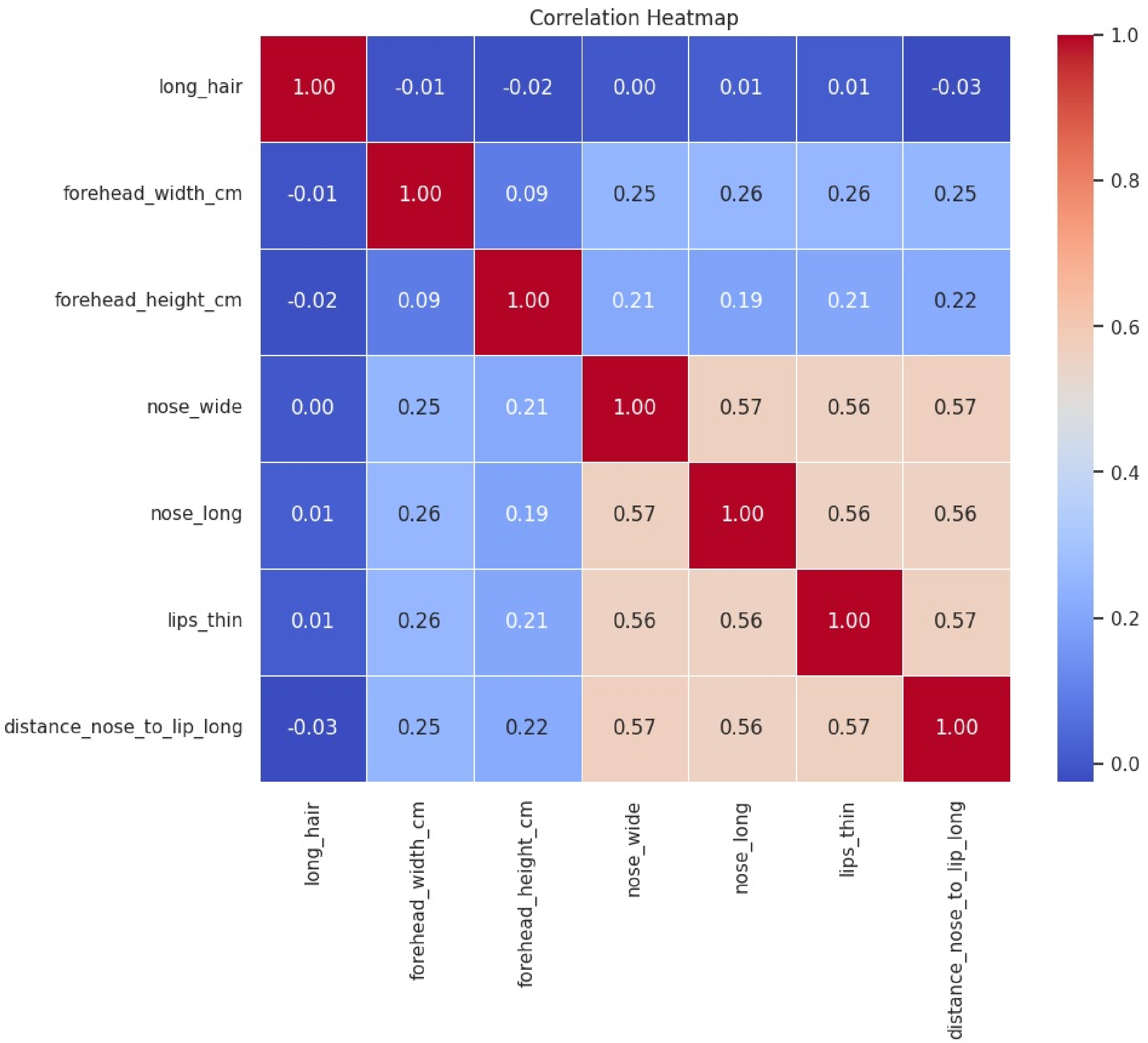
In [ ]:
```python
# Correlation
correlation_matrix =df[numerical_columns].corr()
correlation_matrix
```

Out[ ]:

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_l
long_hair	1.000000	-0.006530	-0.017233	0.001216	0.014432	0.011287	-0
forehead_width_cm	-0.006530	1.000000	0.088596	0.251648	0.257368	0.258564	0
forehead_height_cm	-0.017233	0.088596	1.000000	0.211655	0.194120	0.205441	0
nose_wide	0.001216	0.251648	0.211655	1.000000	0.565192	0.557615	0
nose_long	0.014432	0.257368	0.194120	0.565192	1.000000	0.561229	0
lips_thin	0.011287	0.258564	0.205441	0.557615	0.561229	1.000000	0
distance_nose_to_lip_long	-0.025794	0.251328	0.215292	0.569303	0.559794	0.565312	1

In [ ]:

```
# Heatmap for Correlation Matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [ ]:

```
# Chi-square Test
contingency_table = pd.crosstab(df['long_hair'], df['gender'])
chi2_stat, p_value, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square Statistic: {chi2_stat}\nP-value: {p_value}")
```

```
Chi-square Statistic: 0.517551479459157
P-value: 0.47188802409575925
```

In [ ]:

```
# Z-test or t-test
male_forehead = df[df['gender'] == 'Male']['forehead_width_cm']
female_forehead = df[df['gender'] == 'Female']['forehead_width_cm']
t_stat, p_value_ttest = ttest_ind(male_forehead, female_forehead)
print(f"T-test Statistic: {t_stat}\nP-value: {p_value_ttest}")
```

```
T-test Statistic: 25.06432518851344
P-value: 1.063197016698364e-130
```

In [ ]:
```python
# ANOVA (Example for 'forehead_width_cm' across different 'long_hair' categories)
anova_result = f_oneway(df['forehead_width_cm'][df['long_hair'] == 0],
                        df['forehead_width_cm'][df['long_hair'] == 1])
print(f"ANOVA F-statistic: {anova_result.statistic}\nP-value: {anova_result.pvalue}")
```

```
ANOVA F-statistic: 0.21316903345667512
P-value: 0.6443148961556857
```

Feature Reduction

Linear Discriminant Analysis (LDA):

In [ ]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

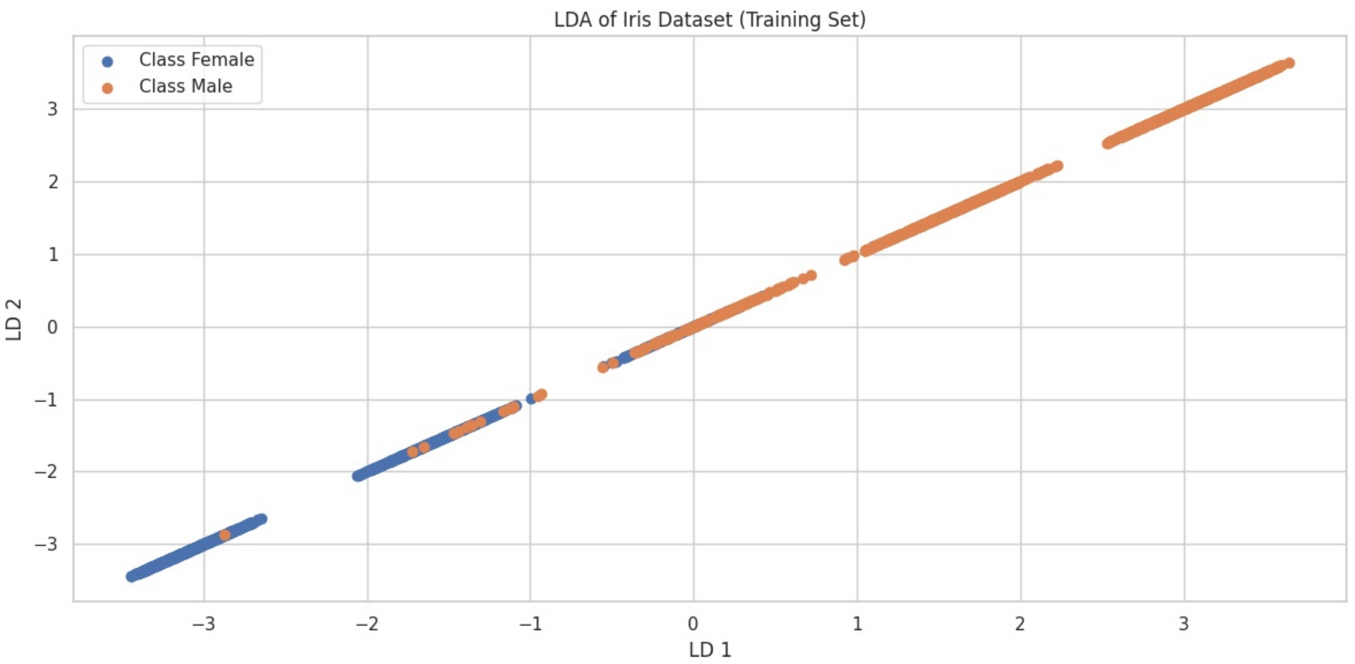
In [ ]:
```python
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```python
n_components = min(X.shape[1], len(set(y)) - 1)
lda = LinearDiscriminantAnalysis(n_components=n_components)
```

In [ ]:
```python
X_lda_train = lda.fit_transform(X_train, y_train)
```

In [ ]:
```python
plt.figure(figsize=(12, 6))
for label in np.unique(y_train):
    plt.scatter(X_lda_train[y_train == label, 0], X_lda_train[y_train == label, 0], label=f'Class {label}')

plt.title('LDA of Iris Dataset (Training Set)')
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.legend()
plt.tight_layout()
plt.show()
```



Principal Component Analysis (PCA):

In [ ]:
```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

In [ ]:
```python
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```python
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

In [ ]:
```python
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

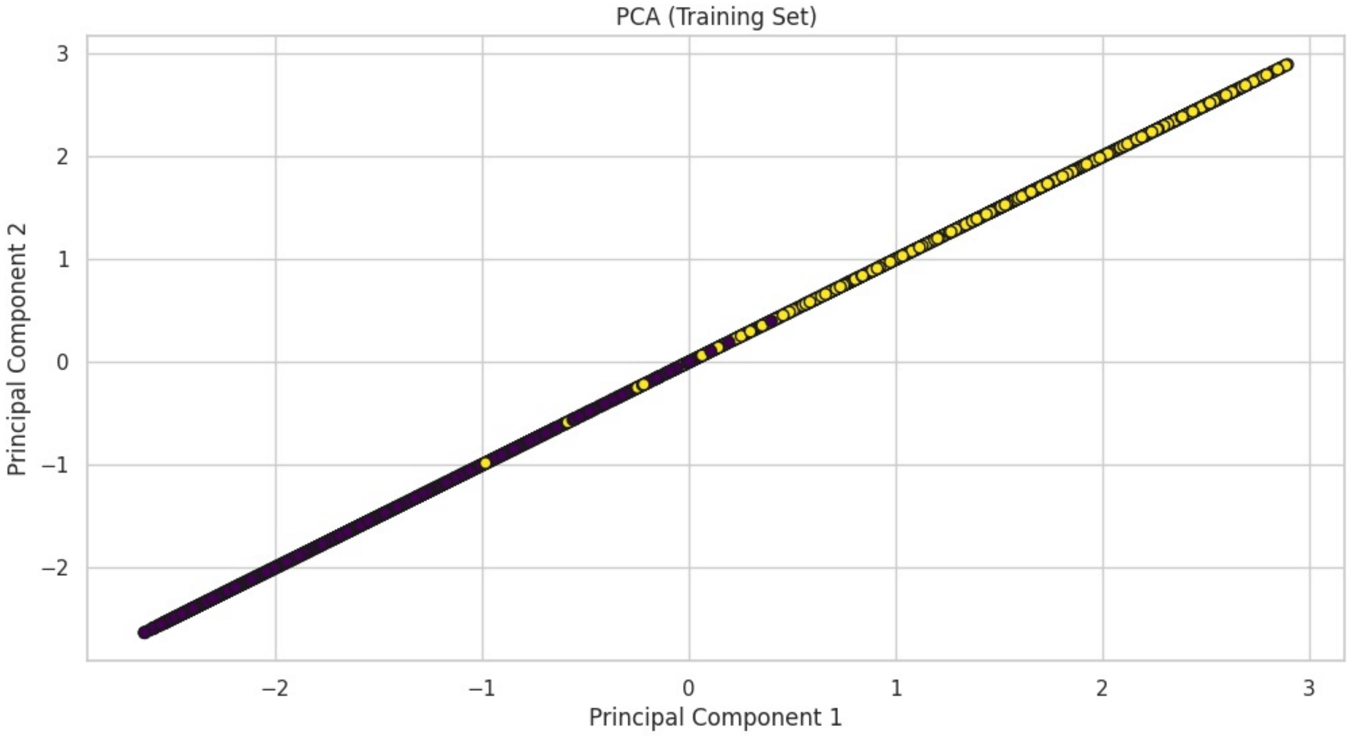
In [ ]:
```
n_components = min(X.shape[1], len(set(y)) - 1)
pca = PCA(n_components=n_components)
```

In [ ]:
```
X_train_pca = pca.fit_transform(X_scaled)
```

In [ ]:
```
print("Original shape:", X.shape)
print("Transformed shape:", X_train_pca.shape)
```

```
Original shape: (5001, 7)
Transformed shape: (5001, 1)
```

In [ ]:
```
plt.figure(figsize=(12, 6))
# Plotting the first principal component against the second principal component
plt.scatter(X_train_pca[:, 0], X_train_pca[:,0], c=y_encoded, cmap='viridis', edgecolor='k')
plt.title('PCA (Training Set)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



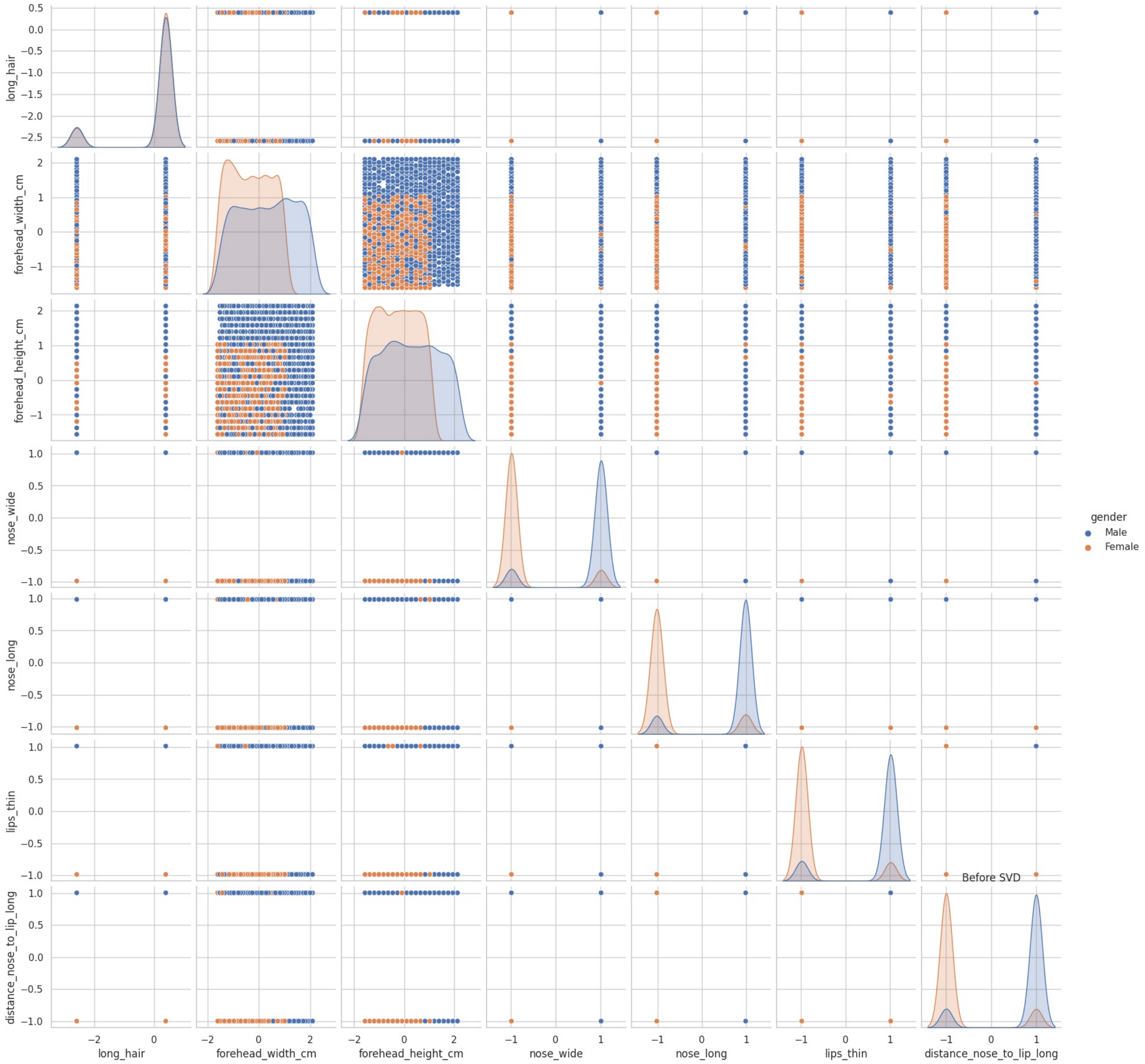
Singular Value Decomposition (SVD)

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
```

In [ ]:
```
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

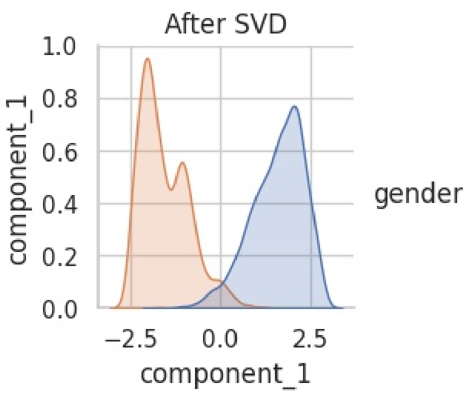
In [ ]:
```
# Before SVD Visualization
sns.pairplot(pd.concat([pd.DataFrame(X_scaled, columns=X.columns), y], axis=1), hue='gender')
plt.title('Before SVD')
plt.show()
```



In [ ]:
```
n_components = min(X.shape[1], len(set(y)) - 1)
svd = TruncatedSVD(n_components=n_components)
X_svd = svd.fit_transform(X_scaled)
```

In [ ]:
```
column_names = [f'component_{i+1}' for i in range(n_components)]
df_svd = pd.DataFrame(data=X_svd, columns=column_names)
df_svd['gender'] = y
```

In [ ]:
```
sns.pairplot(df_svd, hue='gender')
plt.title('After SVD')
plt.show()
```



Model Implementations

Naive Bayes (Gaussian Naive Bayes):

In [ ]:
```
classification_reports = {}
```

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.preprocessing import LabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
```

In [ ]:
```
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```
# Convert categorical labels to binary labels
lb = LabelBinarizer()
y_bin = lb.fit_transform(y)
```

In [ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

In [ ]:
```
# Initialize Gaussian Naive Bayes model
nb_model = OneVsRestClassifier(GaussianNB())
```

In [ ]:
```
# Fit the model
nb_model.fit(X_train, y_train)
```

Out[ ]:
```
▸ OneVsRestClassifier
▸ estimator: GaussianNB
    ▸ GaussianNB
```

In [ ]:
```
# Predict on the test set
y_pred_nb = nb_model.predict(X_test)
```

In [ ]:
```
# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Naive Bayes Accuracy:", accuracy_nb)
```

```
Naive Bayes Accuracy: 0.964035964035964
```

In [ ]:
```
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```

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

           0       0.96      0.97      0.96       502
           1       0.97      0.96      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

In [ ]:
```
classification_reports['Naive Bayes'] = classification_report(y_test, y_pred_nb)
```

In [ ]:
```
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(nb_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9702091816367264
```

In [ ]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_nb)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[487  15]
 [ 21 478]]
```

In [ ]:
```
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
```

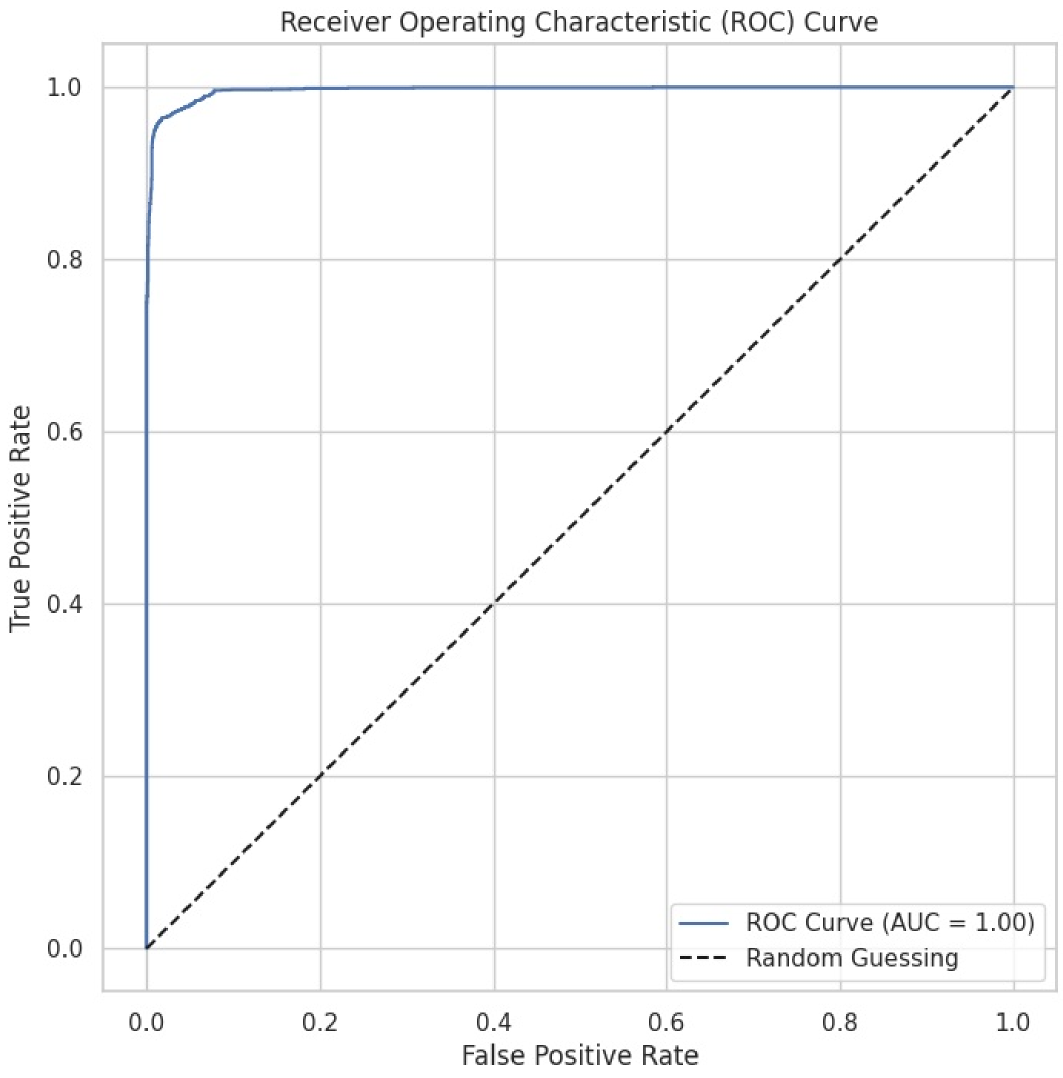
```python
f_measure = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

```
Accuracy: 0.964035964035964
Error Rate: 0.03596403596403597
Precision: 0.9701195219123506
Recall: 0.9586614173228346
F-measure: 0.9643564356435643
```

In [ ]:
```python
# ROC Curve
y_scores = cross_val_predict(nb_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9966251099560175
```

In [ ]:
```python
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```python
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)
```

```python
    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9701195219123506
Recall (Positive): 0.9586614173228346
Specificity: 0.9695740365111561
Sensitivity: 0.9586614173228346
Balance: 0.9643904696175927
The model is reasonably balanced.
```

Bayesian Belief Network:

In [ ]:
```python
!pip install pgmpy
```

```
Collecting pgmpy

  Obtaining dependency information for pgmpy from https://files.pythonhosted.org/packages/eb/9a/2fcb6fdfd998a016cef29ca3eab30b98b6c232b6e9a0444df07f0ad47f8d/pgmpy-0.1.24-py3-none-any.whl.metadata

  Downloading pgmpy-0.1.24-py3-none-any.whl.metadata (6.3 kB)

Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-packages (from pgmpy) (3.1)

Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (from pgmpy) (1.24.3)

Requirement already satisfied: scipy in /opt/conda/lib/python3.10/site-packages (from pgmpy) (1.11.4)

Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.10/site-packages (from pgmpy) (1.2.2)

Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages (from pgmpy) (2.0.3)

Requirement already satisfied: pyparsing in /opt/conda/lib/python3.10/site-packages (from pgmpy) (3.0.9)

Requirement already satisfied: torch in /opt/conda/lib/python3.10/site-packages (from pgmpy) (2.0.0+cpu)

Requirement already satisfied: statsmodels in /opt/conda/lib/python3.10/site-packages (from pgmpy) (0.14.0)

Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages (from pgmpy) (4.66.1)

Requirement already satisfied: joblib in /opt/conda/lib/python3.10/site-packages (from pgmpy) (1.3.2)

Requirement already satisfied: opt-einsum in /opt/conda/lib/python3.10/site-packages (from pgmpy) (3.3.0)

Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.10/site-packages (from pandas->pgmpy) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-packages (from pandas->pgmpy) (2023.3)

Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.10/site-packages (from pandas->pgmpy) (2023.3)

Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.10/site-packages (from scikit-learn->pgmpy) (3.2.0)

Requirement already satisfied: patsy>=0.5.2 in /opt/conda/lib/python3.10/site-packages (from statsmodels->pgmpy) (0.5.3)

Requirement already satisfied: packaging>=21.3 in /opt/conda/lib/python3.10/site-packages (from statsmodels->pgmpy) (21.3)

Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (from torch->pgmpy) (3.12.2)

Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.10/site-packages (from torch->pgmpy) (4.5.0)

Requirement already satisfied: sympy in /opt/conda/lib/python3.10/site-packages (from torch->pgmpy) (1.12)

Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages (from torch->pgmpy) (3.1.2)

Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages (from patsy>=0.5.2->statsmodels->pgmpy) (1.16.0)

Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.10/site-packages (from jinja2->torch->pgmpy) (2.1.3)

Requirement already satisfied: mpmath>=0.19 in /opt/conda/lib/python3.10/site-packages (from sympy->torch->pgmpy) (1.3.0)

Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)

   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 22.9 MB/s eta 0:00:00

Installing collected packages: pgmpy

Successfully installed pgmpy-0.1.24
```

In [ ]:
```python
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
```

In [ ]:
```python
model_structure = [('long_hair', 'gender'), ('forehead_width_cm', 'gender'), ('forehead_height_cm', 'gender'),
                   ('nose_wide', 'gender'), ('nose_long', 'gender'), ('lips_thin', 'gender'),
                   ('distance_nose_to_lip_long', 'gender')]
```

In [ ]:
```python
# Create a BayesianModel object
bayesian_model = BayesianModel(model_structure)
```

In [ ]:
```
# Fit the model parameters using Maximum Likelihood Estimation
model = MaximumLikelihoodEstimator(bayesian_model, df)
```

In [ ]:
```
# Get the CPDs (Conditional Probability Distributions)
cpds = model.get_parameters()
```

In [ ]:
```
# Add CPDs to the Bayesian Model
bayesian_model.add_cpds(*cpds)
```

Decision Tree (Entropy and Error Estimation):

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:
```
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```
# Convert categorical labels to binary labels
lb = LabelBinarizer()
y_bin = lb.fit_transform(y)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

In [ ]:
```
# Initialize Decision Tree model with entropy criterion
dt_entropy_model = DecisionTreeClassifier(criterion='entropy')
```

In [ ]:
```
dt_entropy_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

In [ ]:
```
# Predict on the test set
y_pred_dt_entropy = dt_entropy_model.predict(X_test)
```

In [ ]:
```
# Evaluate the model
accuracy_dt_entropy = accuracy_score(y_test, y_pred_dt_entropy)
print("Decision Tree (Entropy) Accuracy:", accuracy_dt_entropy)
```

```
Decision Tree (Entropy) Accuracy: 0.955044955044955
```

In [ ]:
```
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_dt_entropy))
```

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

           0       0.96      0.95      0.96       502
           1       0.95      0.96      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

In [ ]:
```
classification_reports['Desicion Tree Entropy'] = classification_report(y_test, y_pred_dt_entropy)
```

In [ ]:
```
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(dt_entropy_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9638115768463074
```

In [ ]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_dt_entropy)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[478  24]
 [ 21 478]]
```

In [ ]:
```
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
```

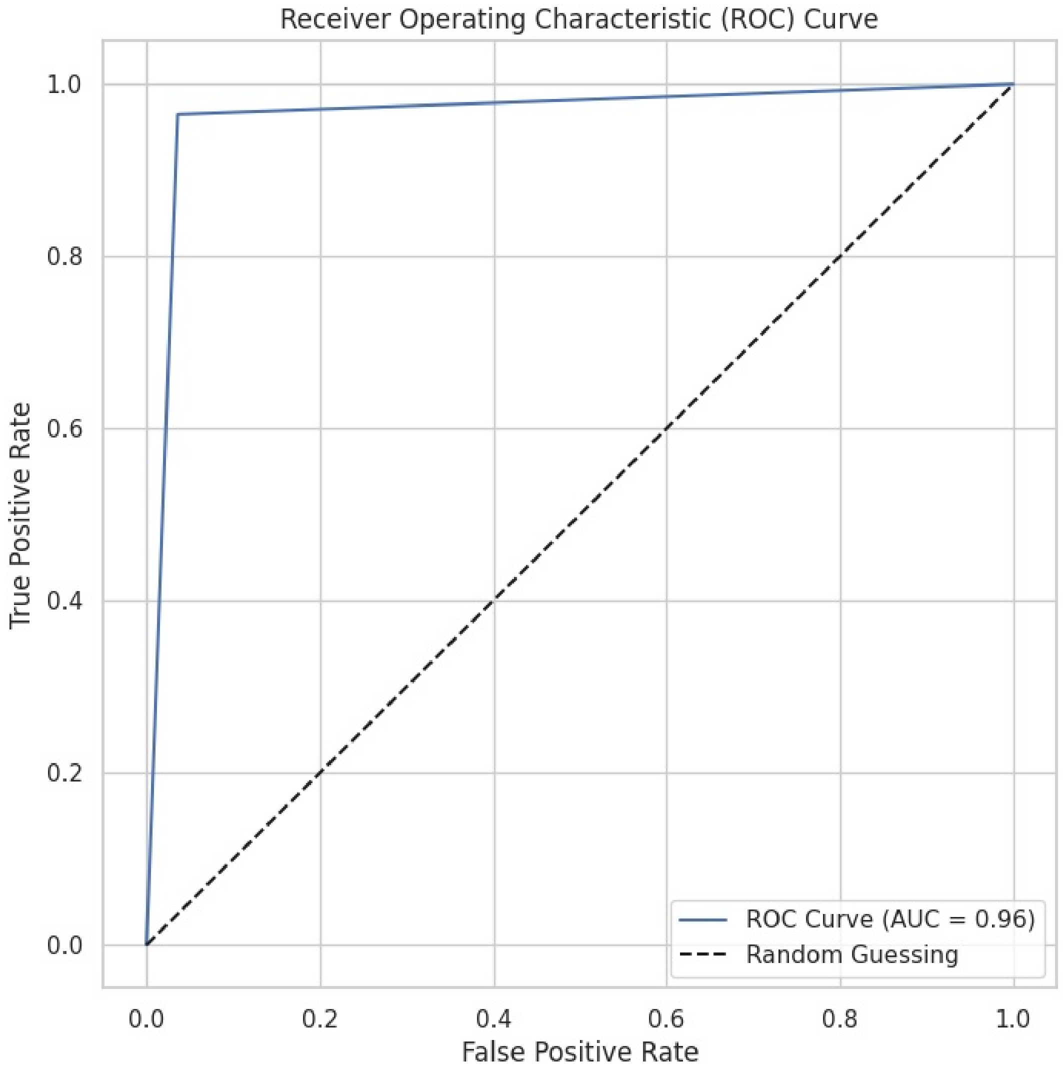
```
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

```
Accuracy: 0.955044955044955
Error Rate: 0.04495504495504499
Precision: 0.952191235059761
Recall: 0.9579158316633266
F-measure: 0.955044955044955
```

In [ ]:
```
# ROC Curve
y_scores = cross_val_predict(dt_entropy_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.964481087564974
```

In [ ]:
```
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
```

```python
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.952191235059761
Recall (Positive): 0.9579158316633266
Specificity: 0.952191235059761
Sensitivity: 0.9579158316633266
Balance: 0.9550535333615437
The model is reasonably balanced.
```

In [ ]:
```python
# Initialize Decision Tree model with gini criterion (default)
dt_gini_model = DecisionTreeClassifier()
```

In [ ]:
```python
# Fit the model
dt_gini_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [ ]:
```python
# Predict on the test set
y_pred_dt_gini = dt_gini_model.predict(X_test)
```

In [ ]:
```python
# Evaluate the model
accuracy_dt_gini = accuracy_score(y_test, y_pred_dt_gini)
print("Decision Tree (Gini) Accuracy:", accuracy_dt_gini)
```

```
Decision Tree (Gini) Accuracy: 0.952047952047952
```

In [ ]:
```python
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_dt_gini))
```

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

           0       0.95      0.96      0.95       502
           1       0.96      0.95      0.95       499

    accuracy                           0.95      1001
   macro avg       0.95      0.95      0.95      1001
weighted avg       0.95      0.95      0.95      1001
```

In [ ]:
```python
classification_reports['Desicion Tree Normal'] = classification_report(y_test, y_pred_dt_gini)
```

In [ ]:
```python
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(dt_gini_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9644115768463072
```

In [ ]:
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_dt_gini)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[480  22]
 [ 26 473]]
```

In [ ]:
```python
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)
```

In [ ]:
```python
print("Accuracy:", accuracy)
```

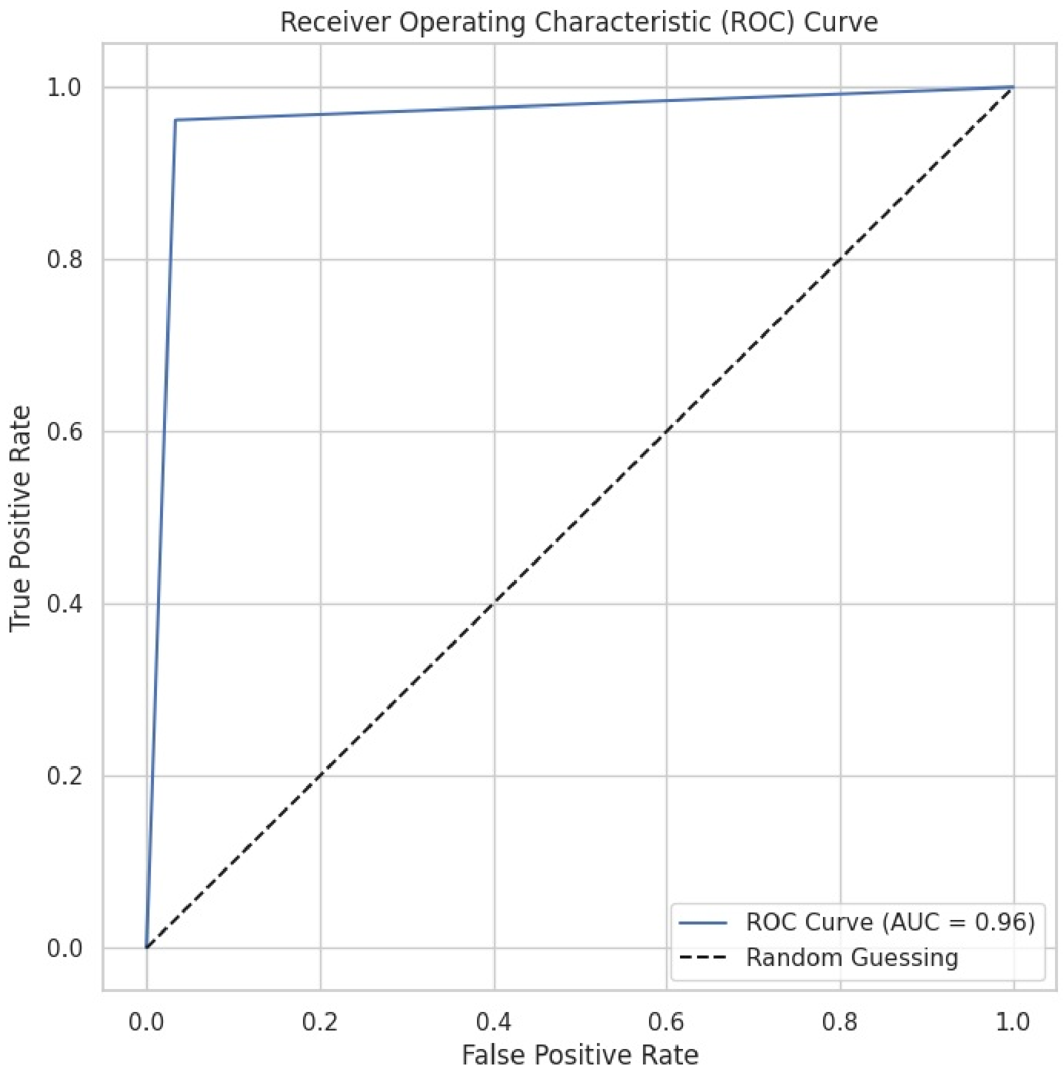
```
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

```
Accuracy: 0.952047952047952
Error Rate: 0.047952047952047994
Precision: 0.9561752988047809
Recall: 0.9486166007905138
F-measure: 0.9523809523809524
```

In [ ]:
```
# ROC Curve
y_scores = cross_val_predict(dt_gini_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9642625349860056
```

In [ ]:
```
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
```

```
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9561752988047809
Recall (Positive): 0.9486166007905138
Specificity: 0.9555555555555556
Sensitivity: 0.9486166007905138
Balance: 0.9523959497976473
The model is reasonably balanced.
```

Linear Discriminant Analysis (LDA):

In [ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:
```
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```
# Convert categorical labels to binary labels
lb = LabelBinarizer()
y_bin = lb.fit_transform(y)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

In [ ]:
```
# Initialize LDA model
lda_model = LinearDiscriminantAnalysis()
```

In [ ]:
```
# Fit the model
lda_model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
```

Out[ ]:
```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

In [ ]:
```
# Predict on the test set
y_pred_lda = lda_model.predict(X_test)
```

In [ ]:
```
# Evaluate the model
accuracy_lda = accuracy_score(y_test, y_pred_lda)
print("LDA Accuracy:", accuracy_lda)
```

```
LDA Accuracy: 0.9590409590409591
```

In [ ]:
```
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_lda))
```

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

           0       0.95      0.97      0.96       502
           1       0.97      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

In [ ]:
```
classification_reports['LDA'] = classification_report(y_test, y_pred_lda)
```

In [ ]:
```
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(lda_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9694099800399201
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/opt/conda/lib/python3.10/site-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [ ]:
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_lda)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[486  16]
 [ 25 474]]
```

In [ ]:
```python
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

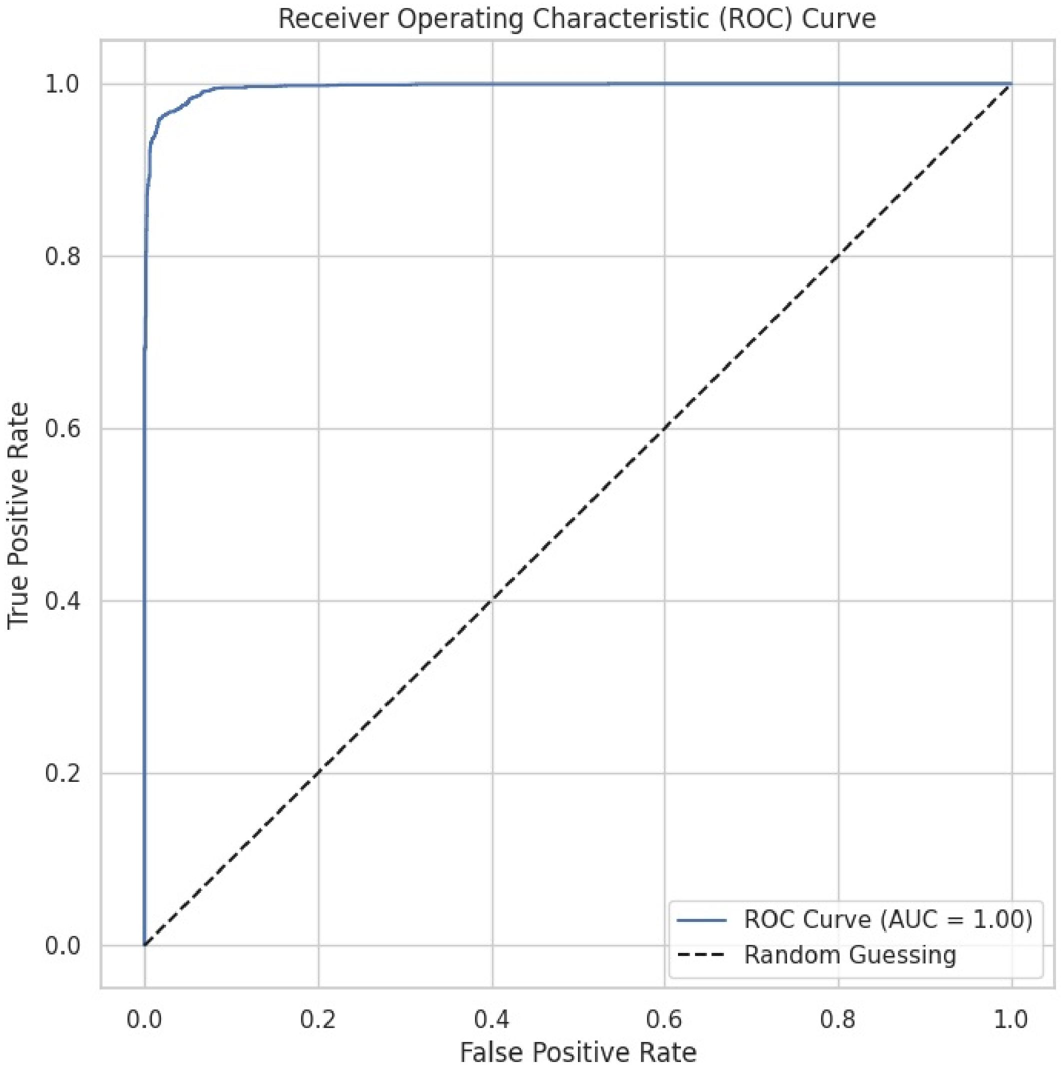
```
Accuracy: 0.9590409590409591
Error Rate: 0.040959040959040904
Precision: 0.9681274900398407
Recall: 0.9510763209393346
F-measure: 0.9595261599210265
```

In [ ]:
```python
# ROC Curve
y_scores = cross_val_predict(lda_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9962665333866453
```

In [ ]:
```python
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9681274900398407
Recall (Positive): 0.9510763209393346
Specificity: 0.9673469387755103
Sensitivity: 0.9510763209393346
Balance: 0.9596019054895877
The model is reasonably balanced.
```

Neural Network

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix, precision_score, recall_score, f1_
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```
y_bin = (y == 'Male').astype(int)
```

In [ ]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

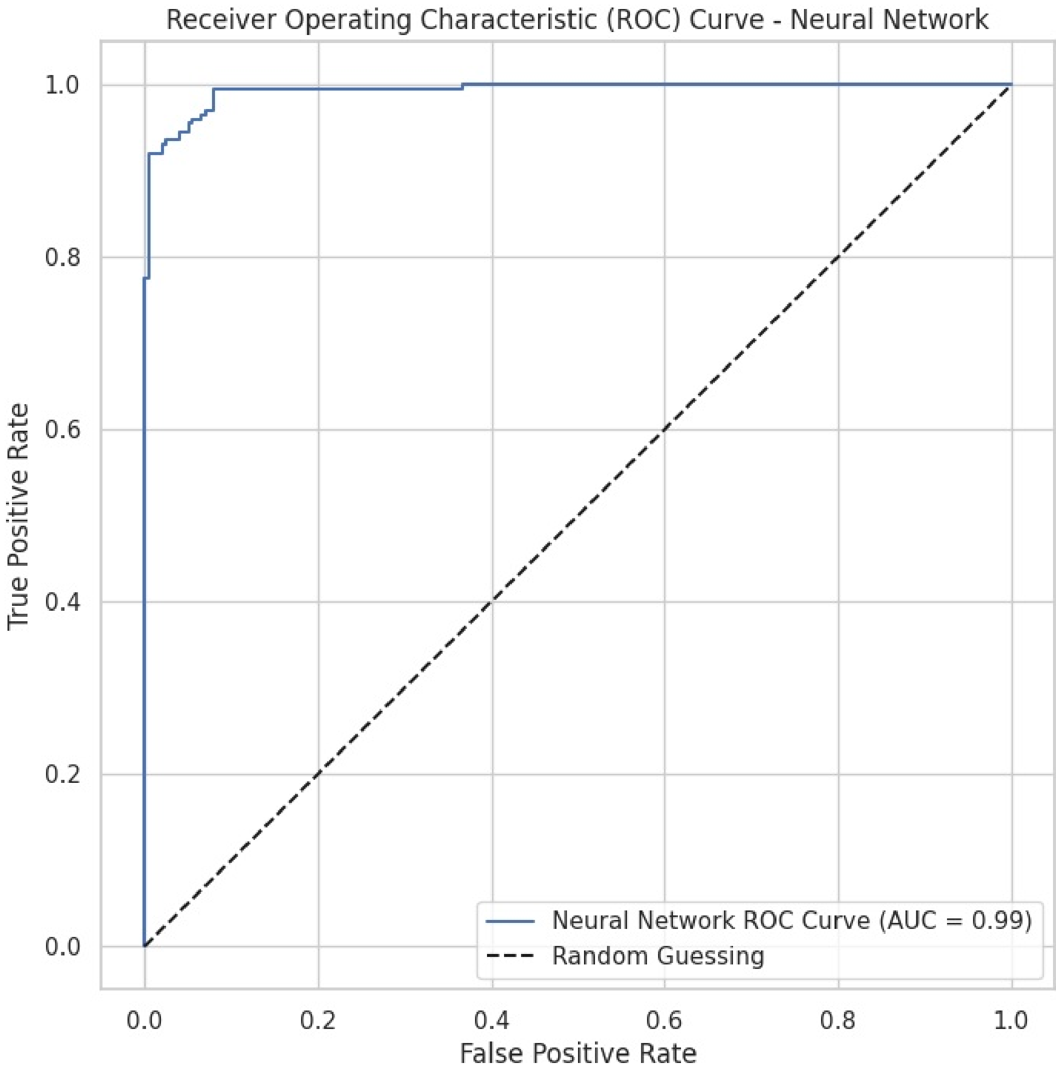
In [ ]:
```
# Convert features and labels to numpy arrays
X_train_np = X_train.values
X_test_np = X_test.values
y_train_np = y_train.values
y_test_np = y_test.values
```

In [ ]:
```
# Neural Network Model
def create_model():
    model = Sequential()
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

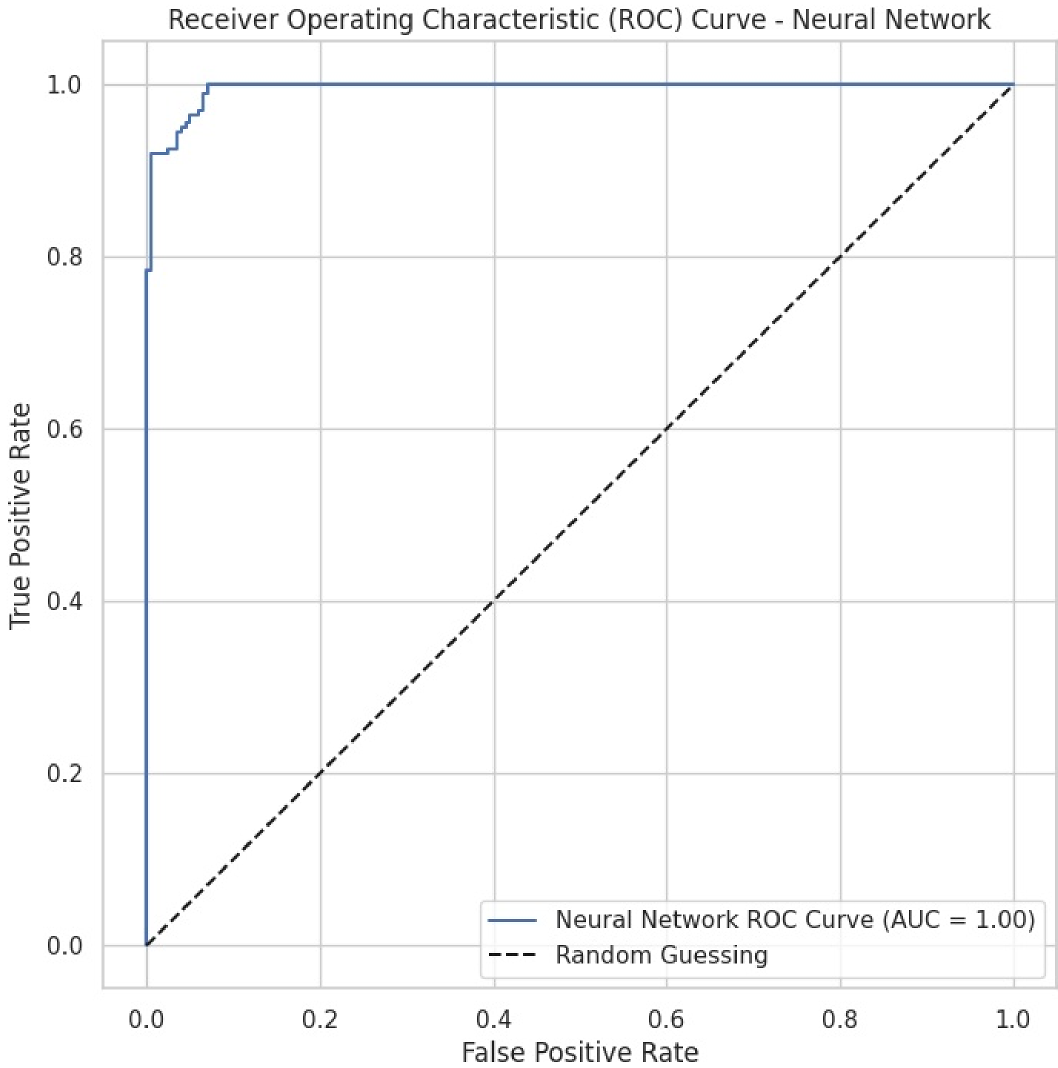
In [ ]:
```
# K-fold Cross Validation for Neural Network
k_fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cv_scores = []
```

In [ ]:
```
# Lists to store evaluation metrics
conf_matrices = []
accuracies = []
errors = []
precisions = []
recalls = []
f1_scores = []

for train_idx, test_idx in k_fold.split(X_train_np, y_train_np):
    # Initialize the model
    model = create_model()

    # Train the model on the current fold
    model.fit(X_train_np[train_idx], y_train_np[train_idx], epochs=10, batch_size=32, verbose=0)

    # Make predictions on the test set
    y_pred_nn = np.round(model.predict(X_train_np[test_idx])).astype(int)

    # Evaluation metrics
    conf_matrix = confusion_matrix(y_train_np[test_idx], y_pred_nn)
    accuracy = accuracy_score(y_train_np[test_idx], y_pred_nn)
    error = 1 - accuracy
    precision = precision_score(y_train_np[test_idx], y_pred_nn)
    recall = recall_score(y_train_np[test_idx], y_pred_nn)
    f1 = f1_score(y_train_np[test_idx], y_pred_nn)

    # Append metrics to lists
    conf_matrices.append(conf_matrix)
    accuracies.append(accuracy)
    errors.append(error)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print cv_scores within the loop
    print("CV Scores:", cv_scores)

    # Visualize ROC Curve
    plt.figure(figsize=(8, 8))
    y_scores_nn = model.predict(X_train_np[test_idx])
    fpr_nn, tpr_nn, _ = roc_curve(y_train_np[test_idx], y_scores_nn)
    roc_auc_nn = auc(fpr_nn, tpr_nn)
    plt.plot(fpr_nn, tpr_nn, label=f'Neural Network ROC Curve (AUC = {roc_auc_nn:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
    plt.title('Receiver Operating Characteristic (ROC) Curve - Neural Network')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

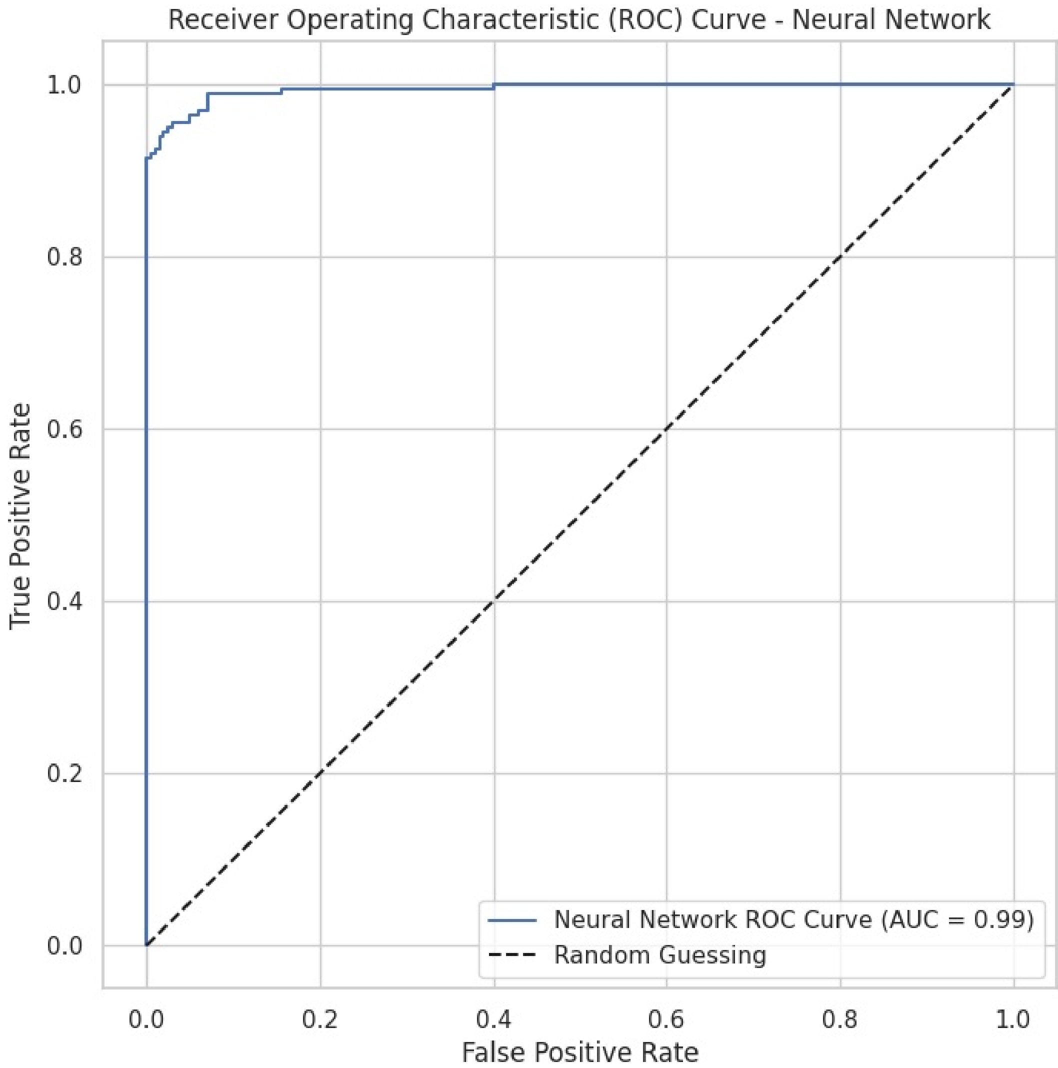
```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



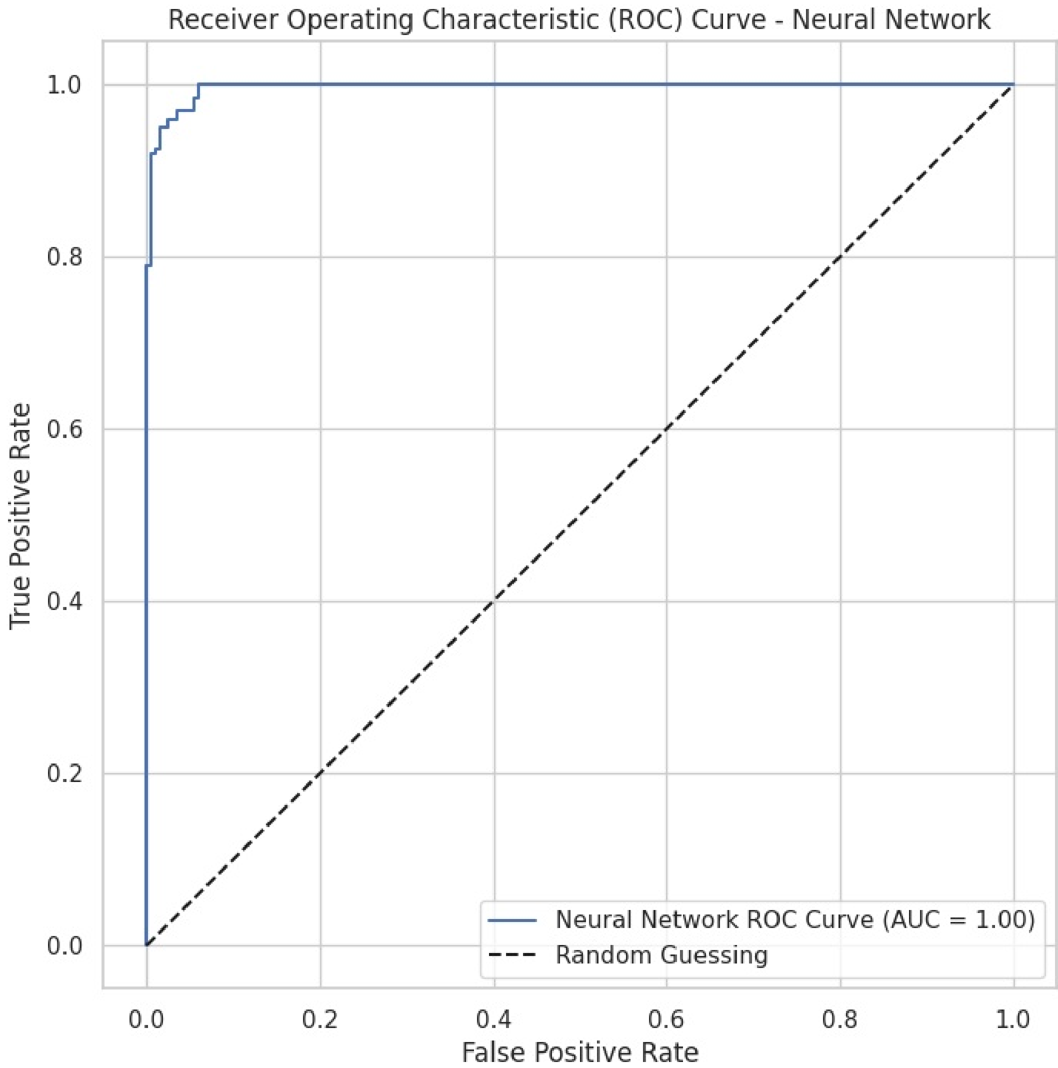

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



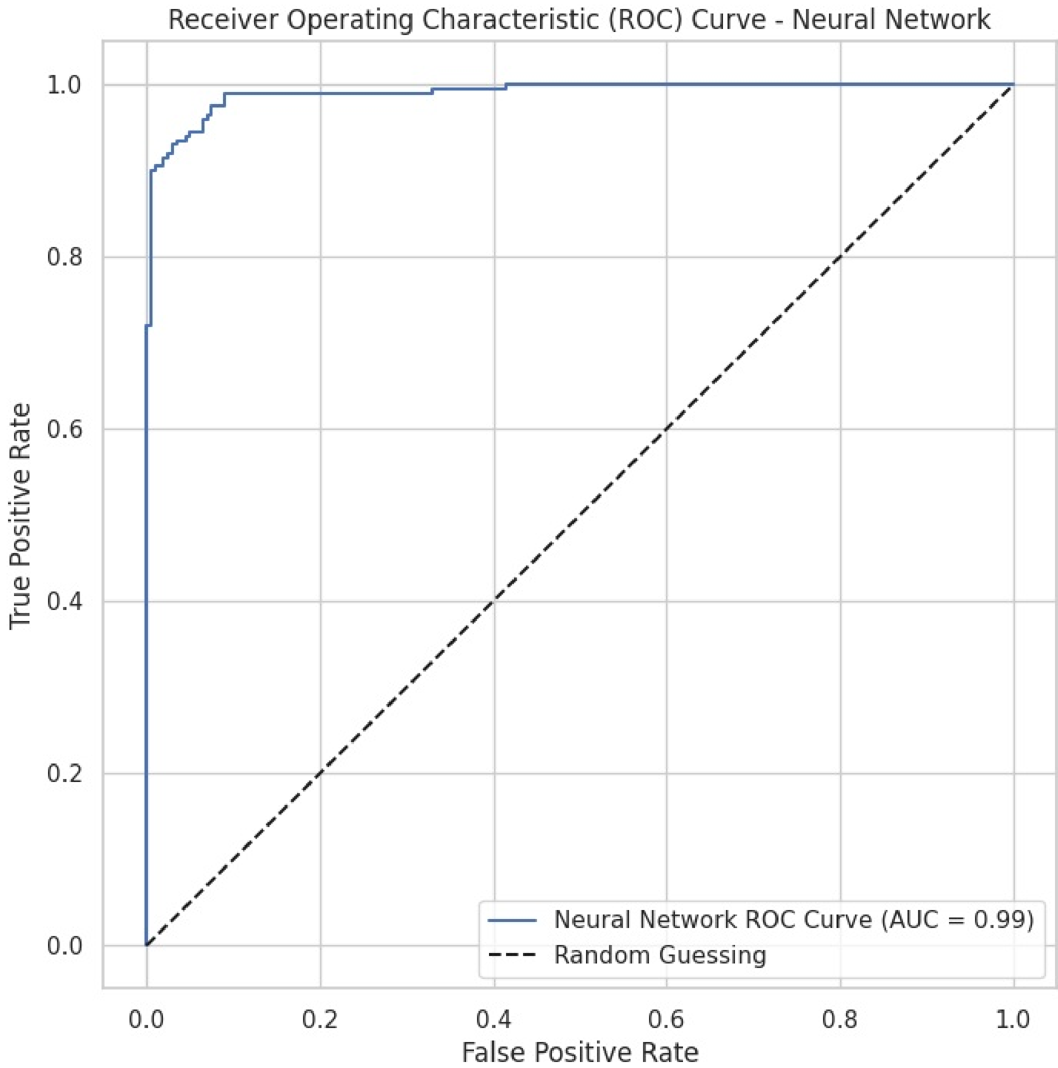

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



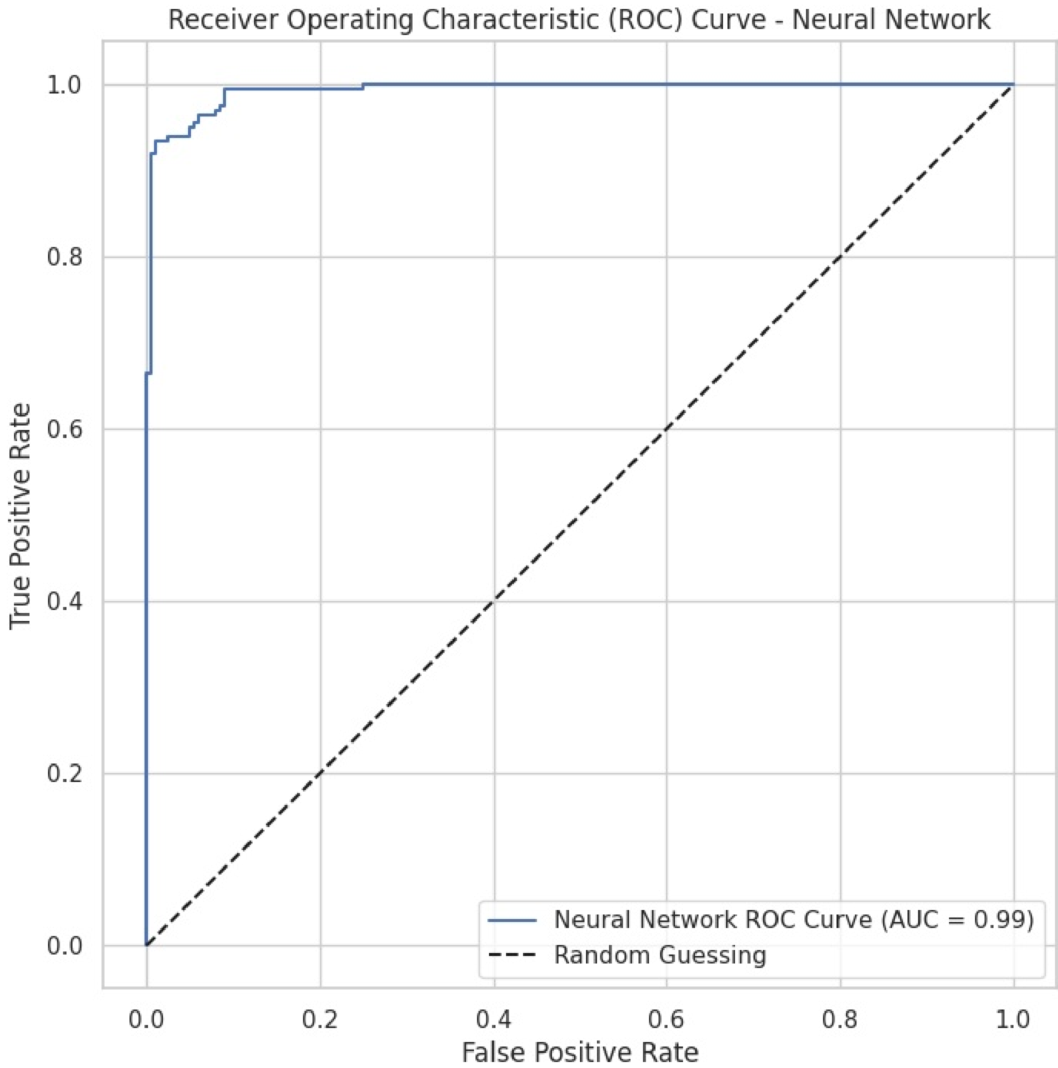

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



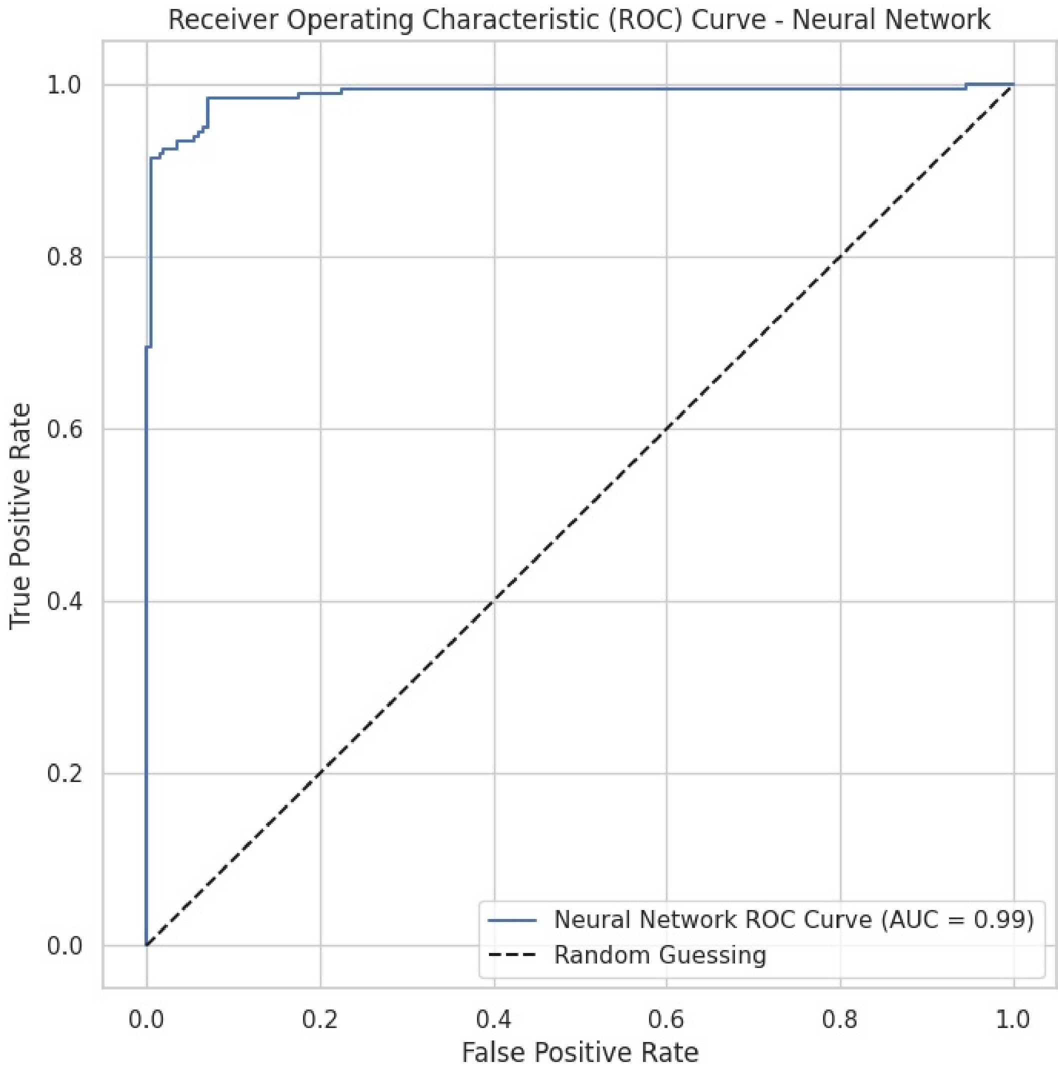

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



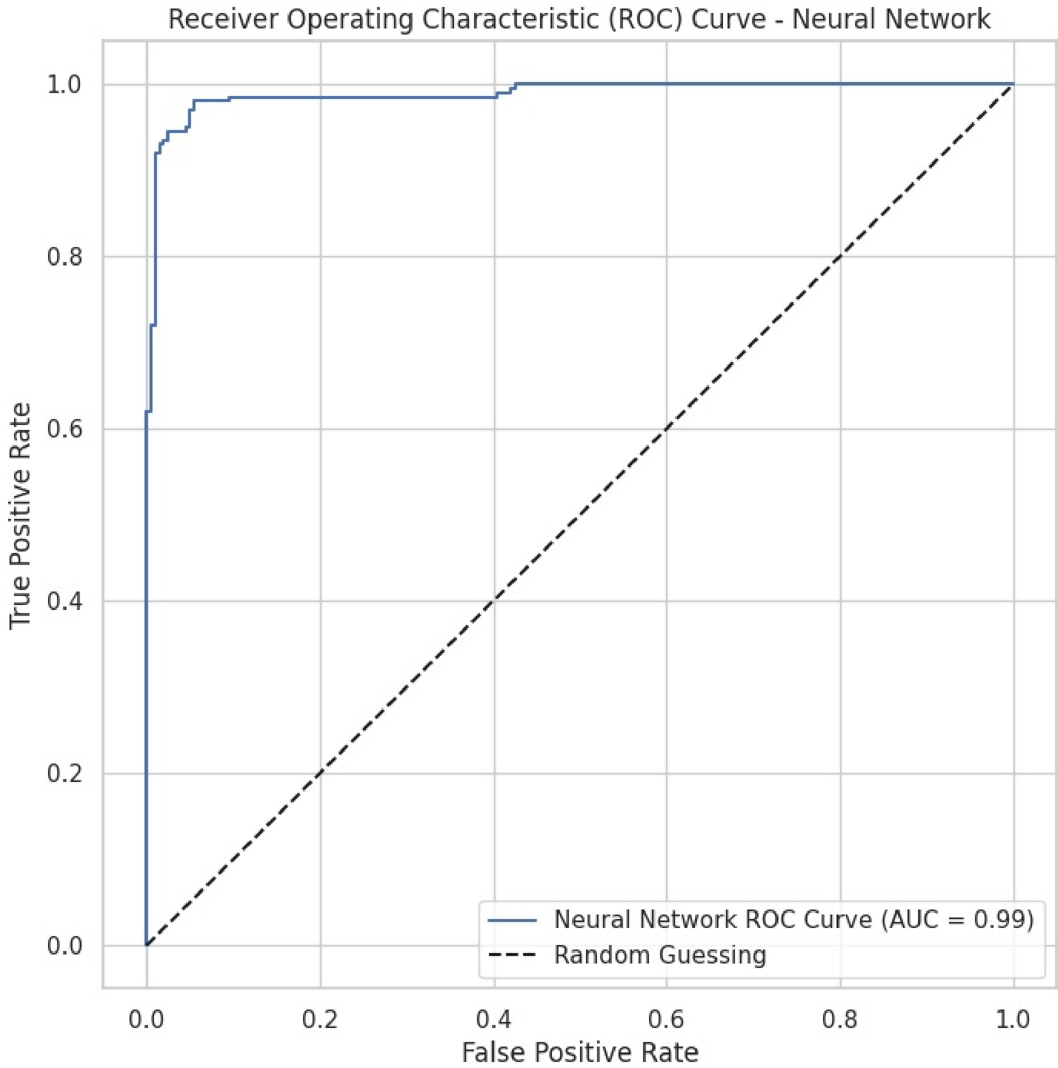

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



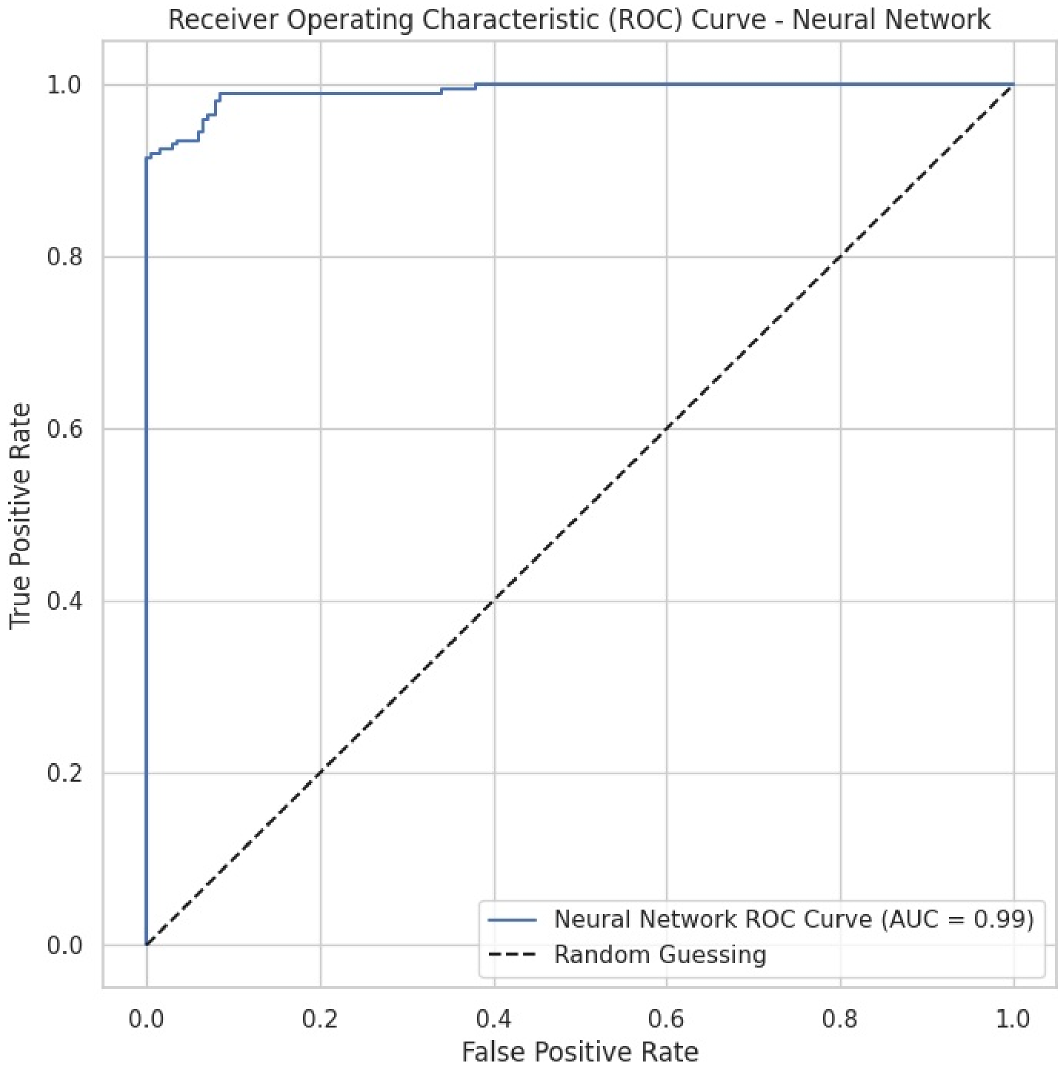

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```



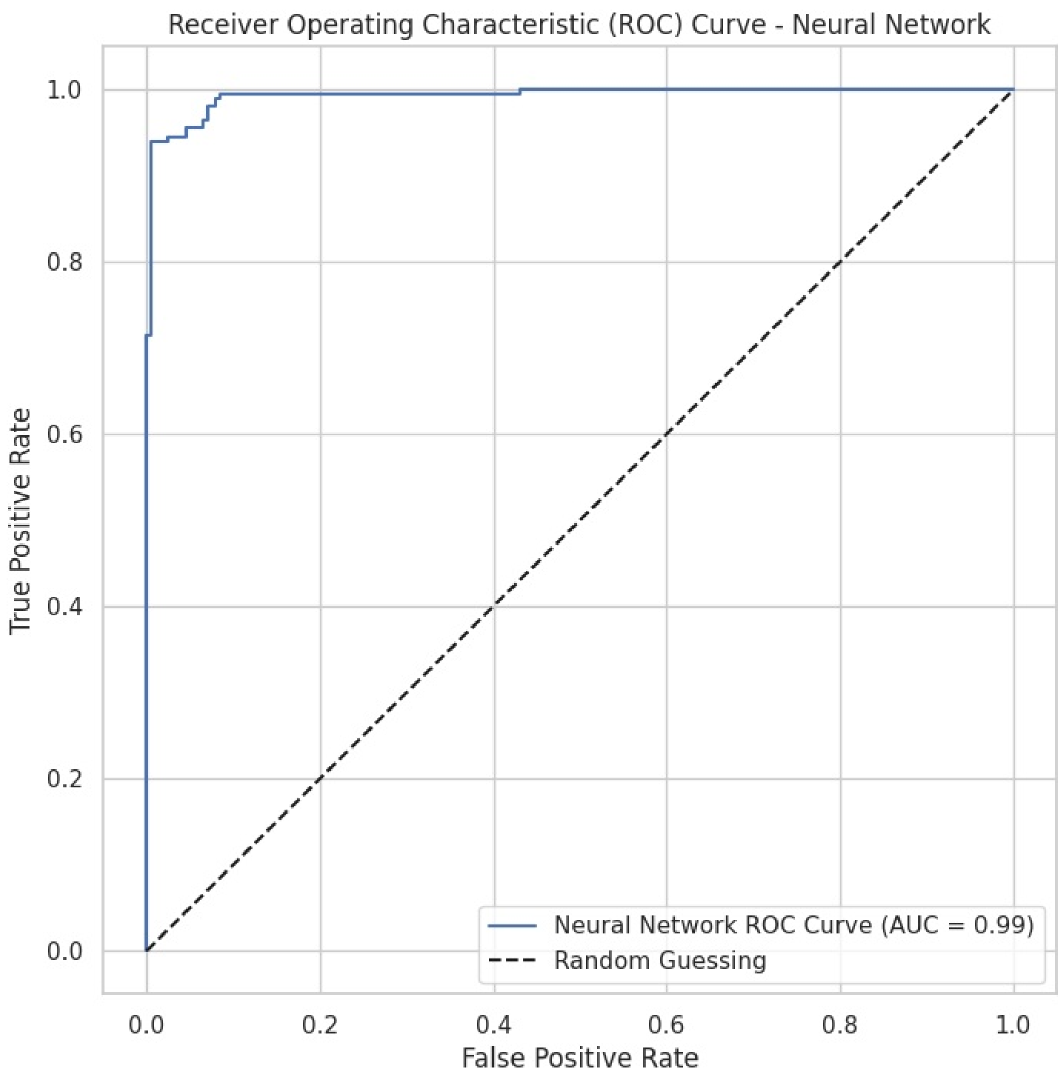

```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```




```
13/13 [==============================] - 0s 1ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```




```
13/13 [==============================] - 0s 2ms/step
CV Scores: []
13/13 [==============================] - 0s 1ms/step
```




```
In [ ]: # Calculate average accuracy
        if len(accuracies) > 0:
            average_accuracy = sum(accuracies) / len(accuracies)
            print("\nK-fold Cross Validation - Neural Network:")
            print("Average Accuracy:", average_accuracy)
        else:
            print("\nNo accuracies to calculate average.")
```

```
K-fold Cross Validation - Neural Network:
Average Accuracy: 0.95175
```

```
In [ ]: y_pred_nn = np.round(model.predict(X_test)).astype(int)
        accuracy = accuracy_score(y_test, y_pred_nn)
        print(f"Accuracy on the test set: {accuracy * 100:.2f}%")
```

```
32/32 [==============================] - 0s 1ms/step
Accuracy on the test set: 95.30%
```

```
In [ ]: print("Classification Report:")
        print(classification_report(y_test, y_pred_nn))
```

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

           0       0.99      0.92      0.95       502
           1       0.92      0.99      0.95       499

    accuracy                           0.95      1001
   macro avg       0.95      0.95      0.95      1001
weighted avg       0.96      0.95      0.95      1001
```

```
In [ ]: classification_reports['NN'] = classification_report(y_test, y_pred_nn)
```

```
In [ ]: # Interpret results of the confusion matrix
        if tp + fp == 0 or tp + fn == 0:
            print("Unable to determine if the model is overfitting or underfitting.")
        else:
            precision_positive = tp / (tp + fp)
            recall_positive = tp / (tp + fn)
            specificity = tn / (tn + fp)
```

```python
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9681274900398407
Recall (Positive): 0.9510763209393346
Specificity: 0.9673469387755103
Sensitivity: 0.9510763209393346
Balance: 0.9596019054895877
The model is reasonably balanced.
```

k-Nearest Neighbors (k-NN) with Different Distances:

In [ ]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:
```python
X = df.drop('gender', axis=1)
y = df['gender']
```

In [ ]:
```python
y_bin = (y == 'Male').astype(int)
```

In [ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.2, random_state=42)
```

In [ ]:
```python
# Initialize k-NN model with Euclidean distance
knn_euclidean_model = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
```

In [ ]:
```python
# Fit the model
knn_euclidean_model.fit(X_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', n_neighbors=3)
```

In [ ]:
```python
# Predict on the test set
y_pred_knn_euclidean = knn_euclidean_model.predict(X_test)
```

In [ ]:
```python
# Evaluate the model
accuracy_knn_euclidean = accuracy_score(y_test, y_pred_knn_euclidean)
print("k-NN (Euclidean Distance) Accuracy:", accuracy_knn_euclidean)
```

```
k-NN (Euclidean Distance) Accuracy: 0.964035964035964
```

In [ ]:
```python
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_knn_euclidean))
```

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

           0       0.95      0.98      0.96       502
           1       0.98      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

In [ ]:
```python
classification_reports['Knn Euc'] = classification_report(y_test, y_pred_knn_euclidean)
```

In [ ]:
```python
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(knn_euclidean_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9674095808383232
```

In [ ]:
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_knn_euclidean)
print("Confusion Matrix:")
```

```
print(conf_matrix)
```

```
Confusion Matrix:
[[490  12]
 [ 24 475]]
```

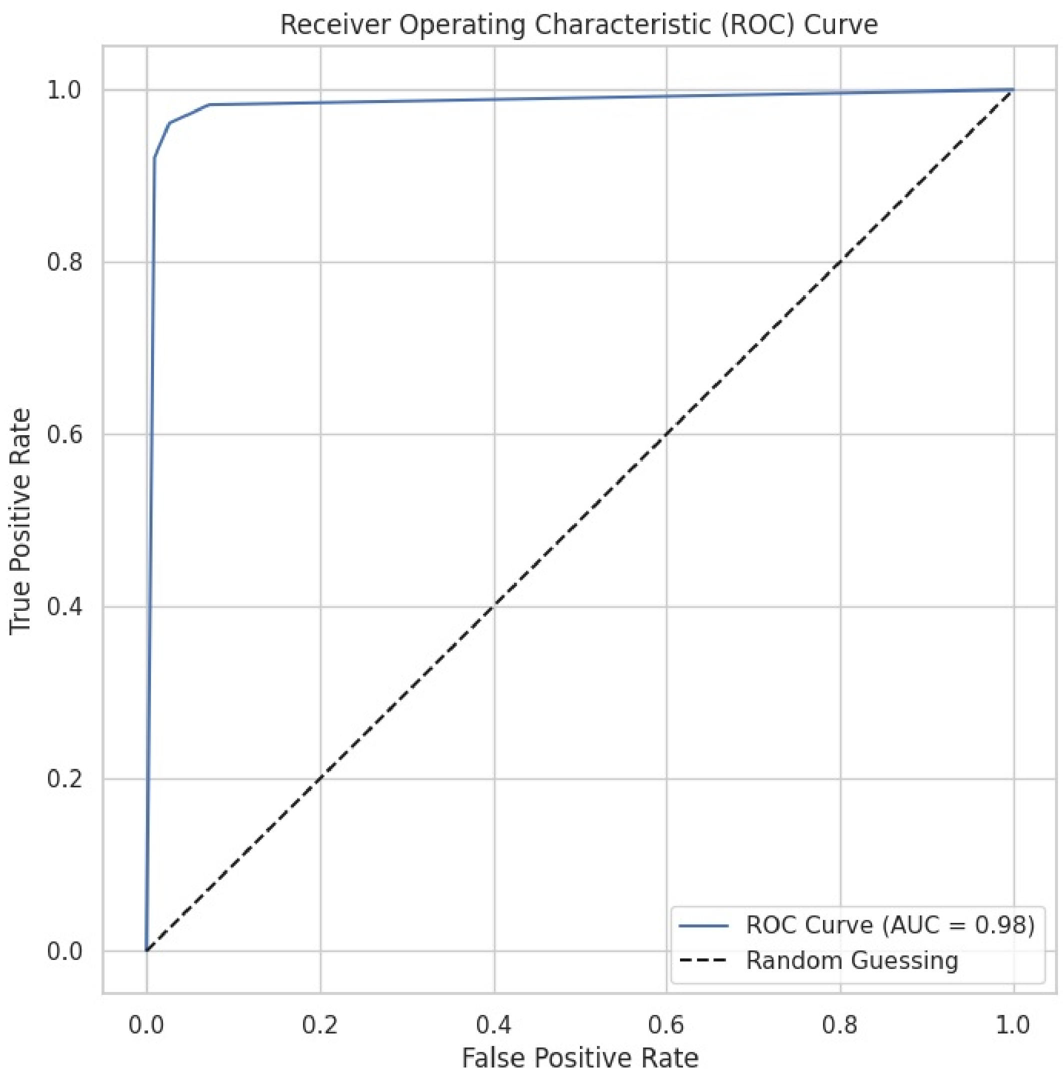
In [ ]:
```
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

```
Accuracy: 0.964035964035964
Error Rate: 0.03596403596403597
Precision: 0.9760956175298805
Recall: 0.953307392996109
F-measure: 0.9645669291338582
```

In [ ]:
```
# ROC Curve
y_scores = cross_val_predict(knn_euclidean_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9845687325069972
```

In [ ]:
```
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:
```
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9760956175298805
Recall (Positive): 0.953307392996109
Specificity: 0.9753593429158111
Sensitivity: 0.953307392996109
Balance: 0.9647015052629948
The model is reasonably balanced.
```

In [ ]:
```
# Initialize k-NN model with Manhattan distance
knn_manhattan_model = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
```

In [ ]:
```
# Fit the model
knn_manhattan_model.fit(X_train, y_train)
```

Out[ ]:
```
KNeighborsClassifier
KNeighborsClassifier(metric='manhattan', n_neighbors=3)
```

In [ ]:
```
# Predict on the test set
y_pred_knn_manhattan = knn_manhattan_model.predict(X_test)
```

In [ ]:
```
# Evaluate the model
accuracy_knn_manhattan = accuracy_score(y_test, y_pred_knn_manhattan)
print("k-NN (Manhattan Distance) Accuracy:", accuracy_knn_manhattan)
```

```
k-NN (Manhattan Distance) Accuracy: 0.9630369630369631
```

In [ ]:
```
# Additional evaluation metrics
print("Classification Report:")
print(classification_report(y_test, y_pred_knn_manhattan))
```

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

           0       0.95      0.98      0.96       502
           1       0.98      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

In [ ]:
```
classification_reports['Knn maha'] = classification_report(y_test, y_pred_knn_manhattan)
```

In [ ]:
```
# K-fold cross-validation and average accuracy
cv_accuracy = cross_val_score(knn_manhattan_model, X, y_bin, cv=10, scoring='accuracy')
avg_cv_accuracy = cv_accuracy.mean()
print("Average Cross-Validation Accuracy:", avg_cv_accuracy)
```

```
Average Cross-Validation Accuracy: 0.9664095808383234
```

In [ ]:
```
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_knn_manhattan)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[490  12]
 [ 25 474]]
```

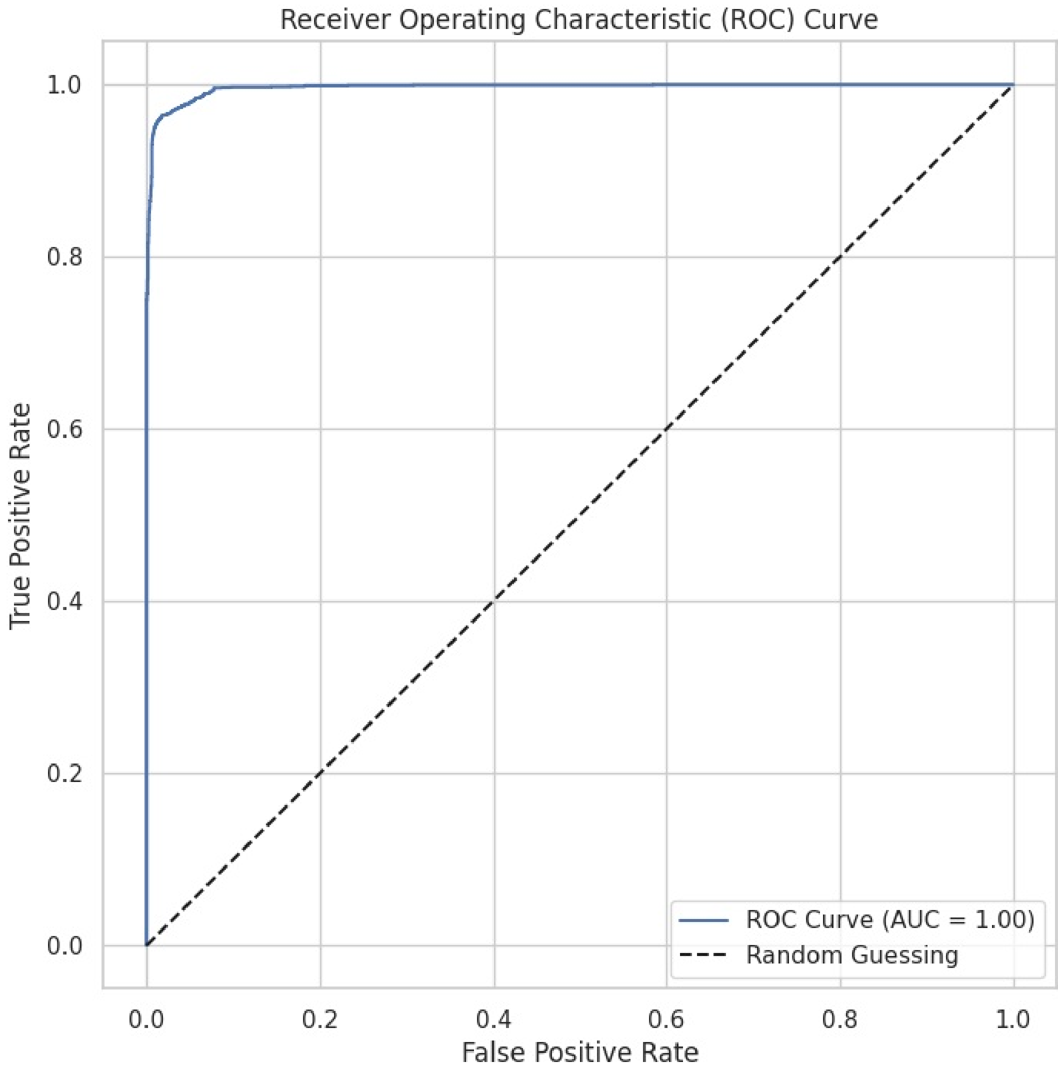
In [ ]:
```
# Accuracy, Error rate, Precision, Recall, F-measure
tp, fp, fn, tn = conf_matrix.ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
error_rate = 1 - accuracy
precision = tp / (tp + fp)
recall = tp / (tp + fn)
f_measure = 2 * (precision * recall) / (precision + recall)

print("Accuracy:", accuracy)
print("Error Rate:", error_rate)
print("Precision:", precision)
print("Recall:", recall)
print("F-measure:", f_measure)
```

```
Accuracy: 0.9630369630369631
Error Rate: 0.03696303696303693
Precision: 0.9760956175298805
Recall: 0.9514563106796117
F-measure: 0.9636184857423796
```

In [ ]:
```
# ROC Curve
y_scores = cross_val_predict(nb_model, X, y_bin.ravel(), cv=10, method="predict_proba")[:, 1]
fpr, tpr, thresholds = roc_curve(y_bin.ravel(), y_scores)
roc_auc = auc(fpr, tpr)

print("ROC AUC:", roc_auc)
```

```
ROC AUC: 0.9966251099560175
```

In [ ]:
```
# Visualize ROC Curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



In [ ]:

```
# Interpret results of the confusion matrix
if tp + fp == 0 or tp + fn == 0:
    print("Unable to determine if the model is overfitting or underfitting.")
else:
    precision_positive = tp / (tp + fp)
    recall_positive = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivity = recall_positive
    balance = 1 - abs(1 - (precision_positive + sensitivity) / 2)

    print("Precision (Positive):", precision_positive)
    print("Recall (Positive):", recall_positive)
    print("Specificity:", specificity)
    print("Sensitivity:", sensitivity)
    print("Balance:", balance)

    if balance > 0.99:
        print("The model may be overfitting.")
    elif balance < 0.05:
        print("The model may be underfitting.")
    else:
        print("The model is reasonably balanced.")
```

```
Precision (Positive): 0.9760956175298805
Recall (Positive): 0.9514563106796117
Specificity: 0.9753086419753086
Sensitivity: 0.9514563106796117
Balance: 0.9637759641047461
The model is reasonably balanced.
```

Comparisons with Other Related Work on the Same Domain:

In [ ]:

```
# Print or access the reports later
for model, report in classification_reports.items():
    print(f"Classification Report for {model}:")
    print(report)
```

```
Classification Report for Naive Bayes:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       502
           1       0.97      0.96      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001

Classification Report for Desicion Tree Entropy:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       502
           1       0.95      0.96      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001

Classification Report for Desicion Tree Normal:
              precision    recall  f1-score   support

           0       0.95      0.96      0.95       502
           1       0.96      0.95      0.95       499

    accuracy                           0.95      1001
   macro avg       0.95      0.95      0.95      1001
weighted avg       0.95      0.95      0.95      1001

Classification Report for LDA:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       502
           1       0.97      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001

Classification Report for NN:
              precision    recall  f1-score   support

           0       0.99      0.92      0.95       502
           1       0.92      0.99      0.95       499

    accuracy                           0.95      1001
   macro avg       0.95      0.95      0.95      1001
weighted avg       0.96      0.95      0.95      1001

Classification Report for Knn Euc:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       502
           1       0.98      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001

Classification Report for Knn maha:
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       502
           1       0.98      0.95      0.96       499

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

References (Papers Using the Same Data Sets):

Link

Loading [MathJax]/extensions/Safe.js