```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier


data = pd.read_csv('/content/risk_factors_cervical_cancer.csv')



# Separate features and target variable
X = data.drop(columns=['Age'])
y = data['Dx:Cancer']

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



X_train.replace('?', np.nan, inplace=True)
X_test.replace('?', np.nan, inplace=True)

# Convert to numeric (assuming you want to impute missing values with mean)
X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_test = X_test.apply(pd.to_numeric, errors='coerce')

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
```

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


model = Sequential()
# Input layer
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(0.2))



# Hidden layers
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(units=64, activation='relu'))
model.add(Dropout(0.2))



# Output layer
model.add(Dense(units=1, activation='sigmoid'))

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

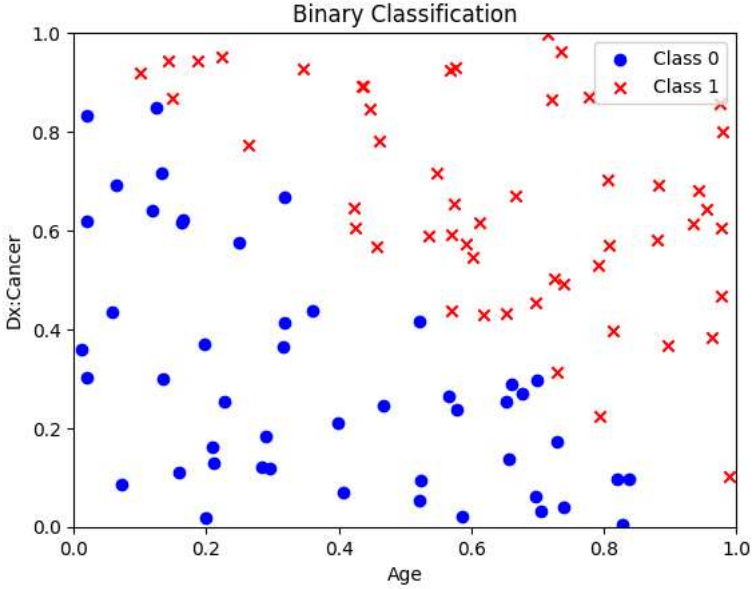
```
Epoch 1/10
22/22 [==============================] - 1s 3ms/step - loss: nan - accuracy: 0.9810
Epoch 2/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
Epoch 3/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
Epoch 4/10
22/22 [==============================] - 0s 3ms/step - loss: nan - accuracy: 0.9810
Epoch 5/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
```

```
Epoch 6/10
22/22 [==============================] - 0s 3ms/step - loss: nan - accuracy: 0.9810
Epoch 7/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
Epoch 8/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
Epoch 9/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
Epoch 10/10
22/22 [==============================] - 0s 2ms/step - loss: nan - accuracy: 0.9810
<keras.src.callbacks.History at 0x7948ce86dfc0>
```

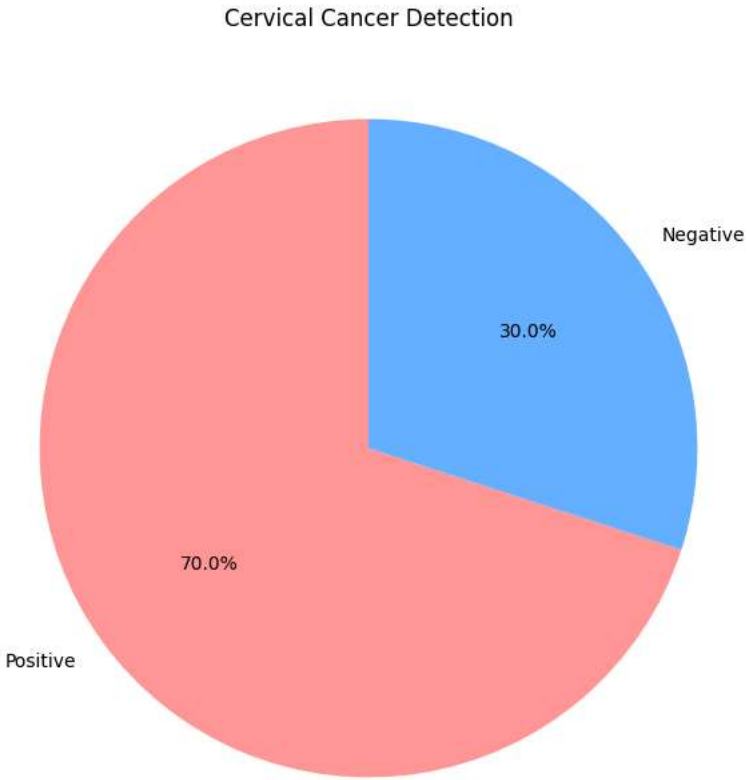
```
history = model.history.history
print(history.keys())
```

```
dict_keys([])
```

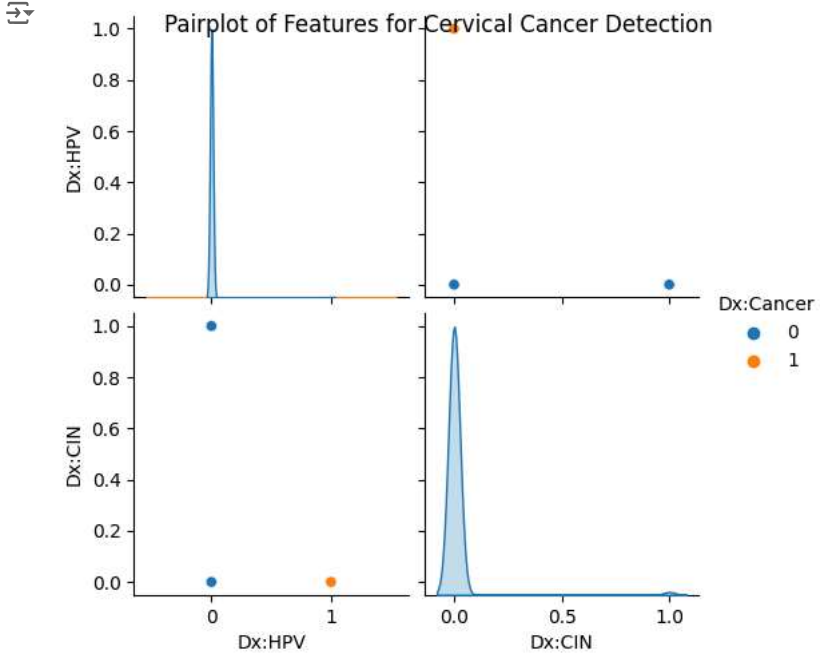
```
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data for illustration purposes
# Replace this with your actual data
np.random.seed(0)
X = np.random.rand(100, 2)  # Sample features (2D)
y = (X[:, 0] + X[:, 1] > 1).astype(int)  # Binary labels (0 or 1)

# Plot data points
plt.scatter(X[y == 0, 0], X[y == 0, 1], label='Class 0', marker='o', color='blue')
plt.scatter(X[y == 1, 0], X[y == 1, 1], label='Class 1', marker='x', color='red')

# Add labels and legend
plt.xlabel('Age')
plt.ylabel('Dx:Cancer')
plt.legend(loc='best')

# Customize plot limits if needed
plt.xlim(0, 1)
plt.ylim(0, 1)

# Show the plot
plt.title("Binary Classification")
plt.show()
```


Binary Classification
Class 0
Class 1
Dx:Cancer
Age
0.0
0.2
0.4
0.6
0.8
1.0

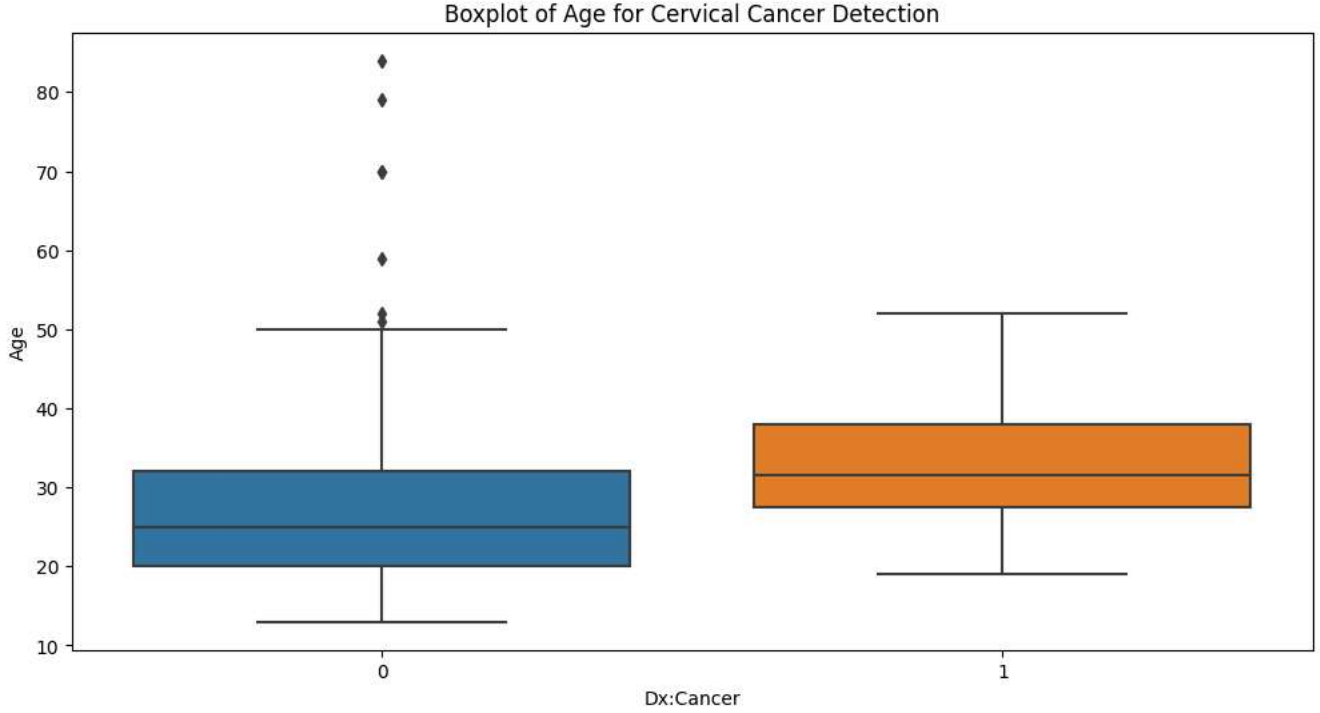
```
import matplotlib.pyplot as plt

# Assuming you have some data for the pie chart
labels = ['Positive', 'Negative']
sizes = [70, 30]  # Replace with your actual data percentages

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=['#ff9999','#66b3ff'])
plt.title('Cervical Cancer Detection')

# Display the pie chart
plt.show()
```


Cervical Cancer Detection
Negative
30.0%
70.0%
Positive

```
import seaborn as sns
# Pairplot to visualize relationships between features
sns.pairplot(data[['Dx:HPV', 'IUD', 'Dx:CIN', 'Dx:Cancer']], hue='Dx:Cancer', diag_kind='kde')
plt.suptitle('Pairplot of Features for Cervical Cancer Detection')
plt.show()
```



```
# Boxplot to visualize the distribution of features based on the target variable
plt.figure(figsize=(12, 6))
sns.boxplot(x='Dx:Cancer', y='Age', data=data)
plt.title('Boxplot of Age for Cervical Cancer Detection')
plt.show()
```



```
# Correlation heatmap to visualize feature correlations
correlation_matrix = data[['Dx:HPV', 'IUD', 'Dx:CIN', 'Dx:Cancer']].corr()
plt.figure(figsize=(10, 8))
```