Diabetes Prediction Using Neural Networks

Essential Libraries

In [49]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Required Libraries for Neural Network

In [163…
```
import tensorflow
import keras_tuner as kt
from tensorflow import keras
from keras_tuner import RandomSearch
from keras.layers import Dense, Dropout, Input
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adadelta
```

Initial Data Exploration and Analysis

In [3]:
```
df = pd.read_csv("diabetes.csv")
```

In [5]:
```
df.sample(10)
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
447	0	95	80	45	92	36.5	0.330	2
569	0	121	66	30	165	34.3	0.203	3
425	4	184	78	39	277	37.0	0.264	3
652	5	123	74	40	77	34.1	0.269	2
653	2	120	54	0	0	26.8	0.455	2
711	5	126	78	27	22	29.6	0.439	4
65	5	99	74	27	0	29.0	0.203	3
479	4	132	86	31	0	28.0	0.419	6
53	8	176	90	34	300	33.7	0.467	5
705	6	80	80	36	0	39.8	0.177	2

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [9]:
```
df.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [11]:
```
df.corr()['Outcome']
```

Out[11]:
```
Pregnancies                 0.221898
Glucose                     0.466581
BloodPressure               0.065068
SkinThickness               0.074752
Insulin                     0.130548
BMI                         0.292695
DiabetesPedigreeFunction    0.173844
Age                         0.238356
Outcome                     1.000000
Name: Outcome, dtype: float64
```

In [15]:
```
df = df.drop(columns = ['BloodPressure','SkinThickness'],axis=1)
```

In [17]:
```
df.shape
```

Out[17]:
```
(768, 7)
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
Pregnancies                 0
Glucose                     0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Separating Features (X) and Target (y)

In [30]:
```
X = df.drop(columns='Outcome',axis=1)
y = df['Outcome']
```

In [34]:
```
X.shape , y.shape
```

Out[34]:
```
((768, 6), (768,))
```

In [32]:
```
X.head()
```

Out[32]:

	Pregnancies	Glucose	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	0	33.6	0.627	50
1	1	85	0	26.6	0.351	31
2	8	183	0	23.3	0.672	32
3	1	89	94	28.1	0.167	21
4	0	137	168	43.1	2.288	33

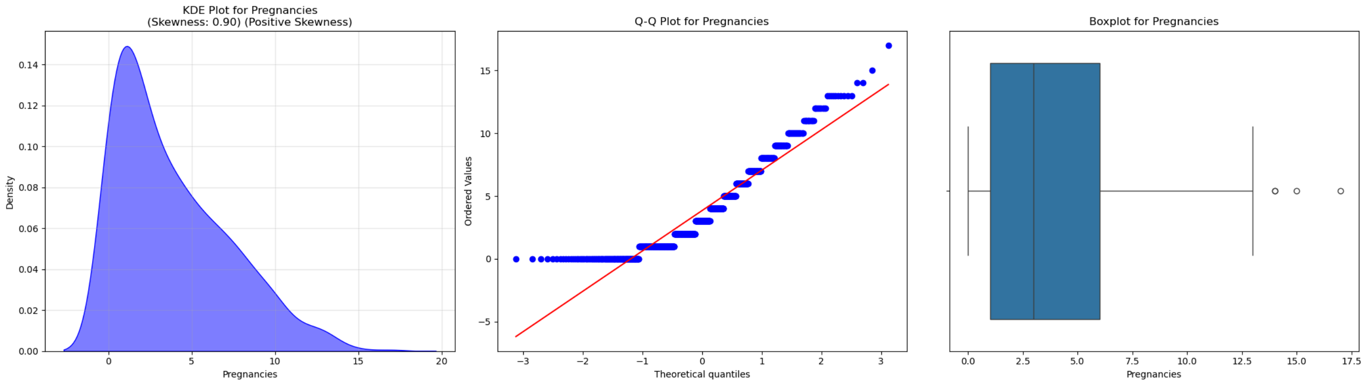
Check and Fix Distribution (Outliers, Q-Q Plot, and Histogram)

In [62]:
```
#plot multiple plot in one functions
def All_plot(X):
    col = X.columns
    for column in col:
        if np.issubdtype(X[column].dtype, np.number):  # Check if the column is numeric
            fig, axes = plt.subplots(1, 3, figsize=(21, 6))  # 3 plots in a row

            skewness = X[column].skew()

            # Determine skewness type
            if skewness > 0:
                skew_type = "Positive Skewness"
            elif skewness < 0:
                skew_type = "Negative Skewness"
            else:
                skew_type = "Approximately Symmetrical"

            print(f"{column}: Skewness = {skewness:.2f} ({skew_type})")

            # KDE plot
            sns.kdeplot(X[column].dropna(), fill=True, color='blue', alpha=0.5, ax=axes[0])
            axes[0].set_title(f"KDE Plot for {column}\n(Skewness: {skewness:.2f}) ({skew_type
            axes[0].set_xlabel(column)
            axes[0].set_ylabel('Density')
            axes[0].grid(alpha=0.3)
```

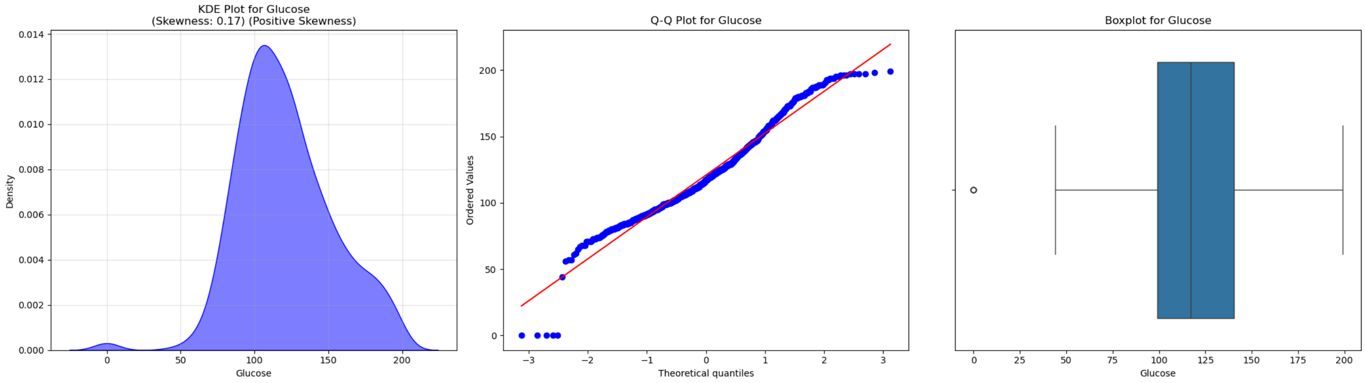
```
        # Q-Q plot
        stats.probplot(X[column].dropna(), dist="norm", plot=axes[1])
        axes[1].set_title(f"Q-Q Plot for {column}", fontsize=12)

        # Boxplot
        sns.boxplot(x=X[column], ax=axes[2])
        axes[2].set_title(f"Boxplot for {column}", fontsize=12)
        axes[2].set_xlabel(column)

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

In [64]:
```
All_plot(X)
```

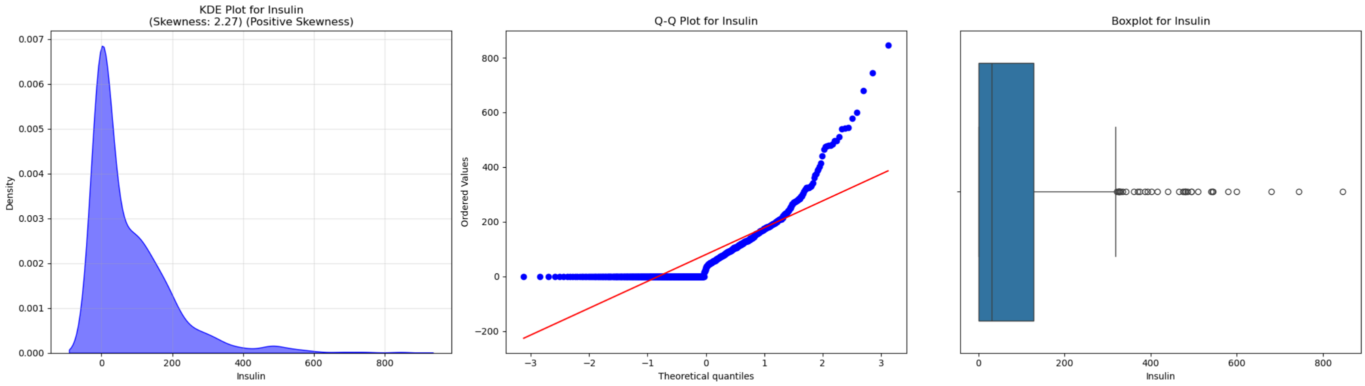
Pregnancies: Skewness = 0.90 (Positive Skewness)



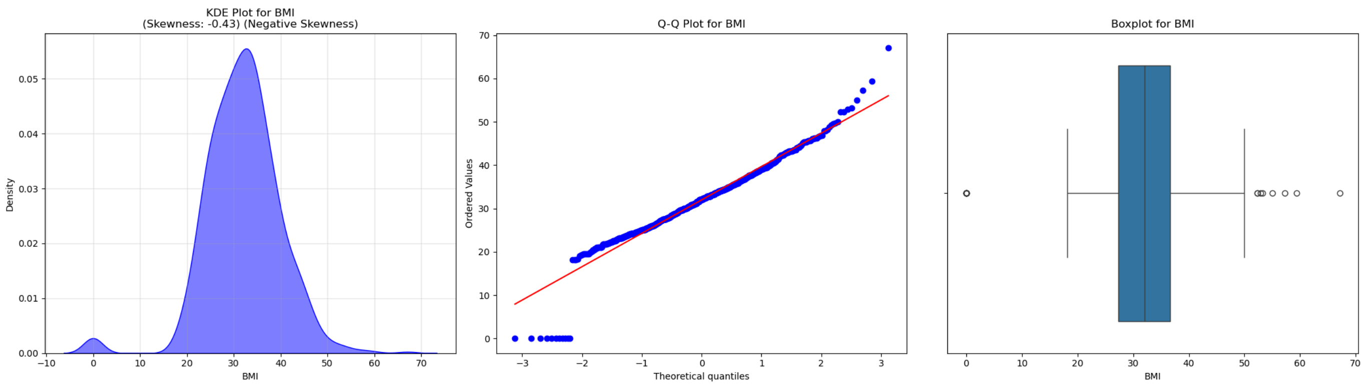
Glucose: Skewness = 0.17 (Positive Skewness)



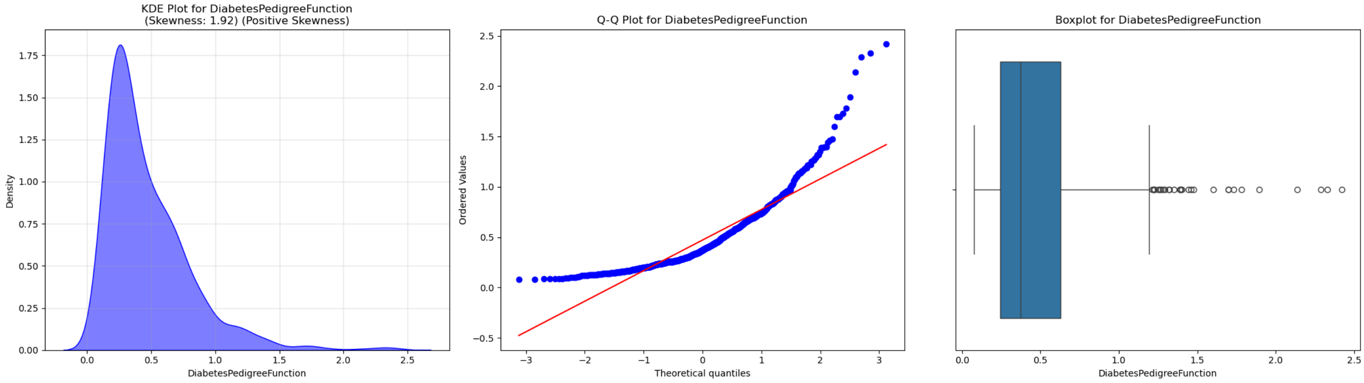
Insulin: Skewness = 2.27 (Positive Skewness)



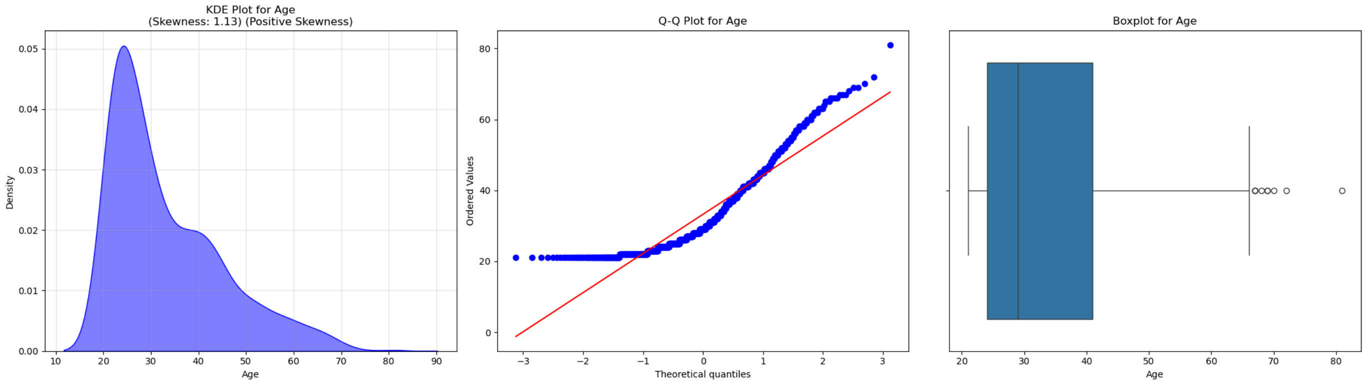
BMI: Skewness = -0.43 (Negative Skewness)



DiabetesPedigreeFunction: Skewness = 1.92 (Positive Skewness)



Age: Skewness = 1.13 (Positive Skewness)



Fix Distribution

In [103…

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PowerTransformer
import pandas as pd

def Advance_column_transformer(df, columns, method='yeo-johnson', standardize=False):
    """
    Applies PowerTransformer to specific columns using ColumnTransformer,
    keeps other columns unchanged.

    Parameters:
    - df: pandas DataFrame
    - columns: list of column names to transform
    - method: 'yeo-johnson' or 'box-cox'
    - standardize: whether to standardize the output

    Returns:
    - transformed_df: full DataFrame with transformed columns
    - lambdas_df: DataFrame with lambdas for transformed columns
    """
    df_copy = df.copy()

    # Validate columns list
    columns = [col for col in columns if col in df_copy.columns]
    if len(columns) == 0:
        raise ValueError("No valid columns provided for transformation.")

    # Check for missing values in selected columns
    if df_copy[columns].isnull().any().any():
        raise ValueError("Input data contains NaN values. Please impute or drop missing value

    # Box-Cox requires strictly positive values
    if method == 'box-cox':
        for col in columns:
            if (df_copy[col] <= 0).any():
                raise ValueError(f"Box-Cox transformation requires all positive values in colu

    # Build ColumnTransformer
```

```python
    col_transformer = ColumnTransformer(
        transformers=[
            ('power', PowerTransformer(method=method, standardize=standardize), columns)
        ],
        remainder='passthrough'
    )

    # Fit and transform the data
    transformed_array = col_transformer.fit_transform(df_copy)

    # Retrieve columns after transformation
    transformed_feature_names = columns
    passthrough_columns = [col for col in df_copy.columns if col not in columns]
    all_columns_order = transformed_feature_names + passthrough_columns

    # Rebuild DataFrame with correct column order
    transformed_df = pd.DataFrame(transformed_array, columns=all_columns_order, index=df.inde

    # Get lambdas for transformed columns
    fitted_power_transformer = col_transformer.named_transformers_['power']
    lambdas_df = pd.DataFrame({
        'columns': columns,
        'lambdas': fitted_power_transformer.lambdas_
    })

    return transformed_df, lambdas_df
```

In [105…

```python
columns = X.columns
columns
```

Out[105…

```
Index(['Pregnancies', 'Glucose', 'Insulin', 'BMI', 'DiabetesPedigreeFunction',
       'Age'],
      dtype='object')
```

In [107…

```python
transformed_X , lambda_df = Advance_column_transformer(X, columns, method='yeo-johnson', stan
```

In [120…

```python
transformed_X.head()
```

Out[120…

	Pregnancies	Glucose	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.813399	0.848665	-1.008294	0.174124	0.821764	1.364180
1	-0.833906	-1.123027	-1.008294	-0.725726	-0.168409	0.126452
2	1.188996	1.930906	-1.008294	-1.129341	0.935284	0.230161
3	-0.833906	-0.996671	0.859700	-0.537700	-1.298725	-1.480075
4	-1.603317	0.506848	1.077013	1.477376	2.336680	0.327328

In [122…

```python
lambda_df
```

Out[122…

	columns	lambdas
0	Pregnancies	0.172724
1	Glucose	0.966405
2	Insulin	-0.032285
3	BMI	1.276566
4	DiabetesPedigreeFunction	-2.250387
5	Age	-1.149602

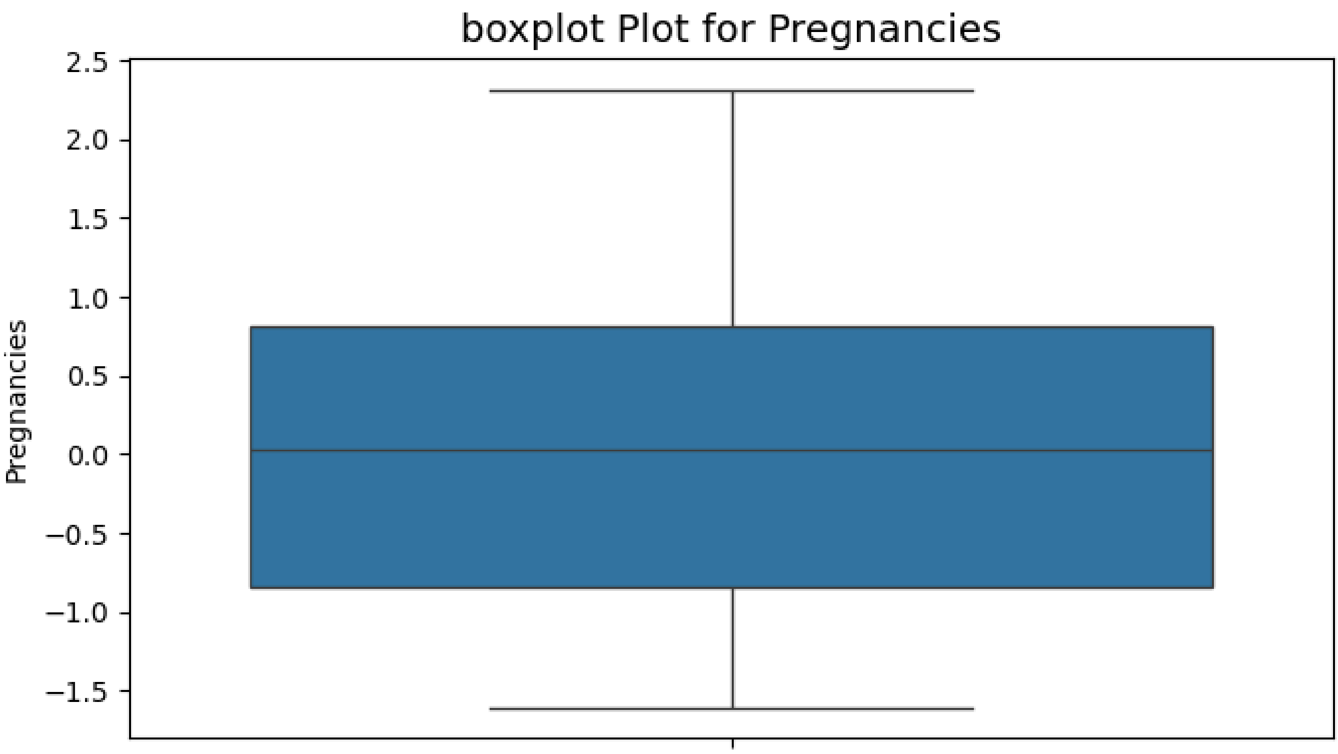
Check And Remove Outliers

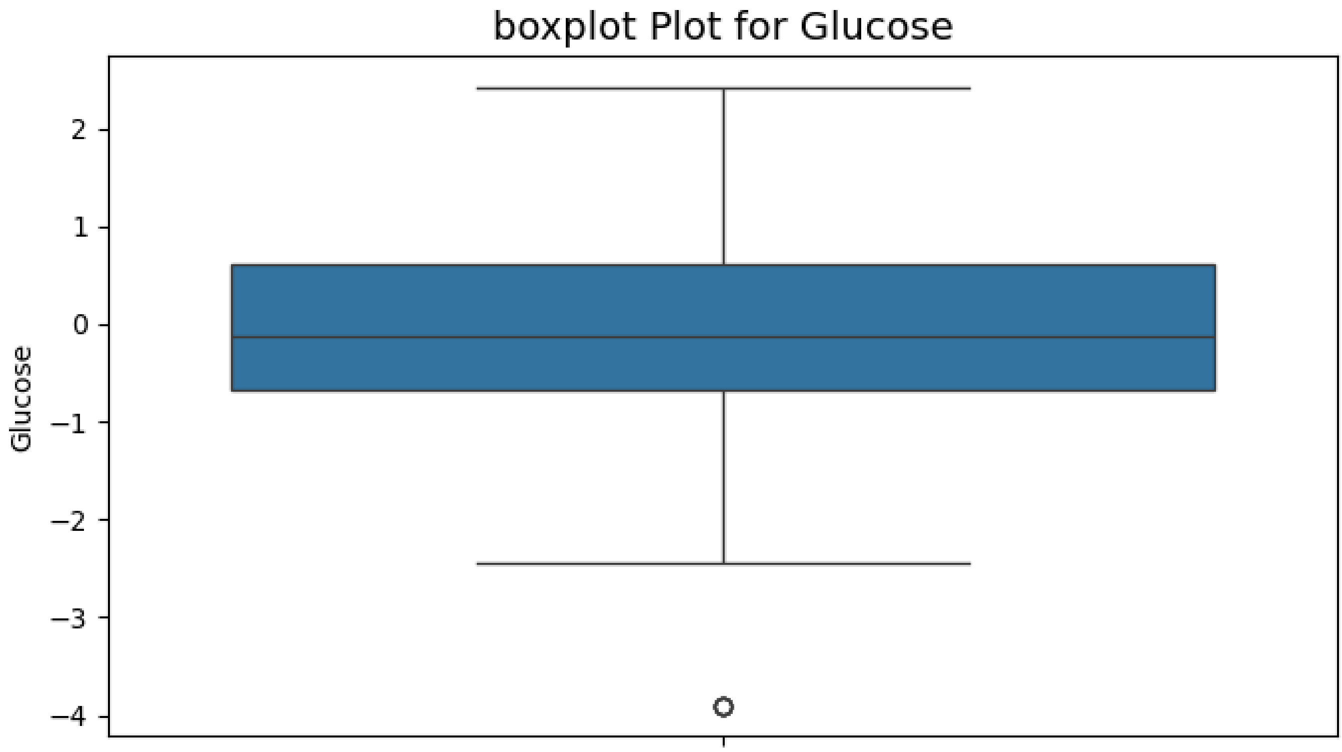
In [118…

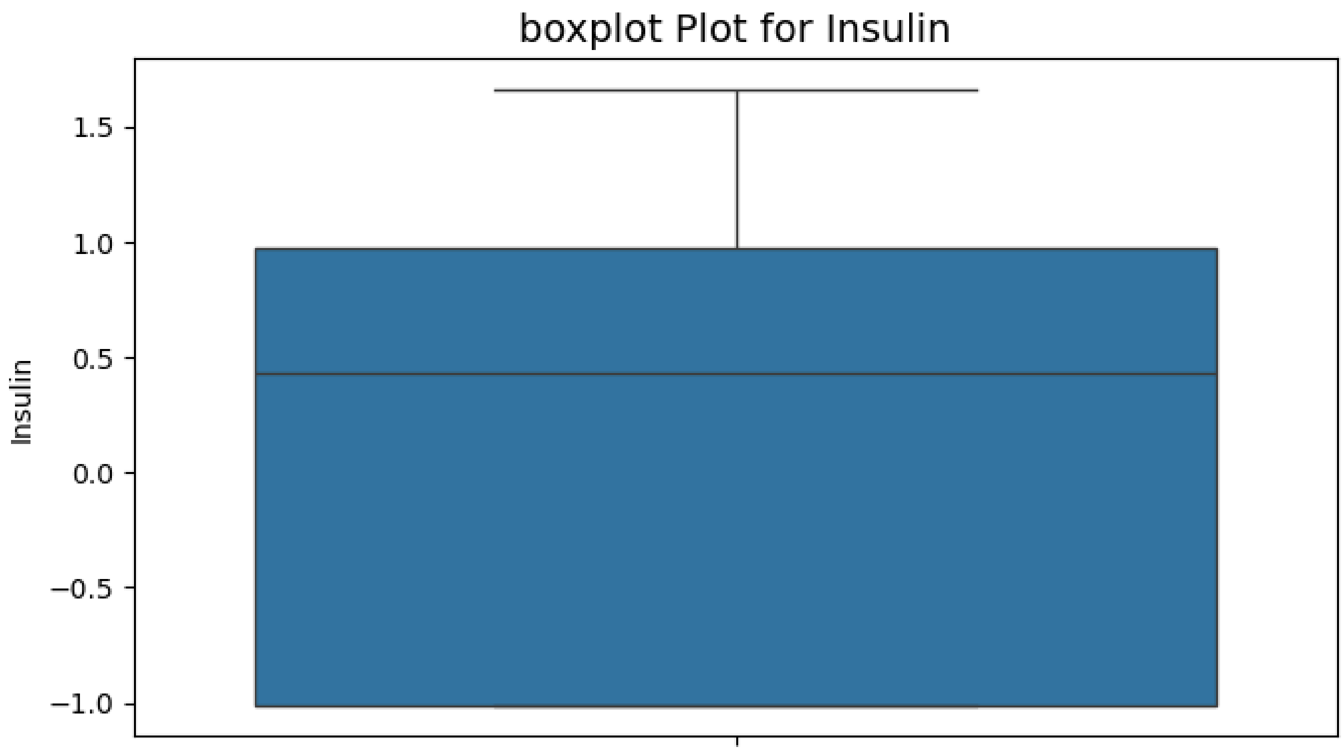
```
#detect the outlier columns and plot them
def outlier(X):
    # Loop through each column in the DataFrame
    for column in X.columns:
        if np.issubdtype(X[column].dtype, np.number):  # Check if the column is numeric
            plt.figure(figsize=(7, 4))

            # KDE plot
            sns.boxplot(X[column])
            plt.title(f"boxplot Plot for {column}", fontsize=14)

            # plt.grid(alpha=0.3)
            plt.tight_layout()
            plt.show()
```

In [125…

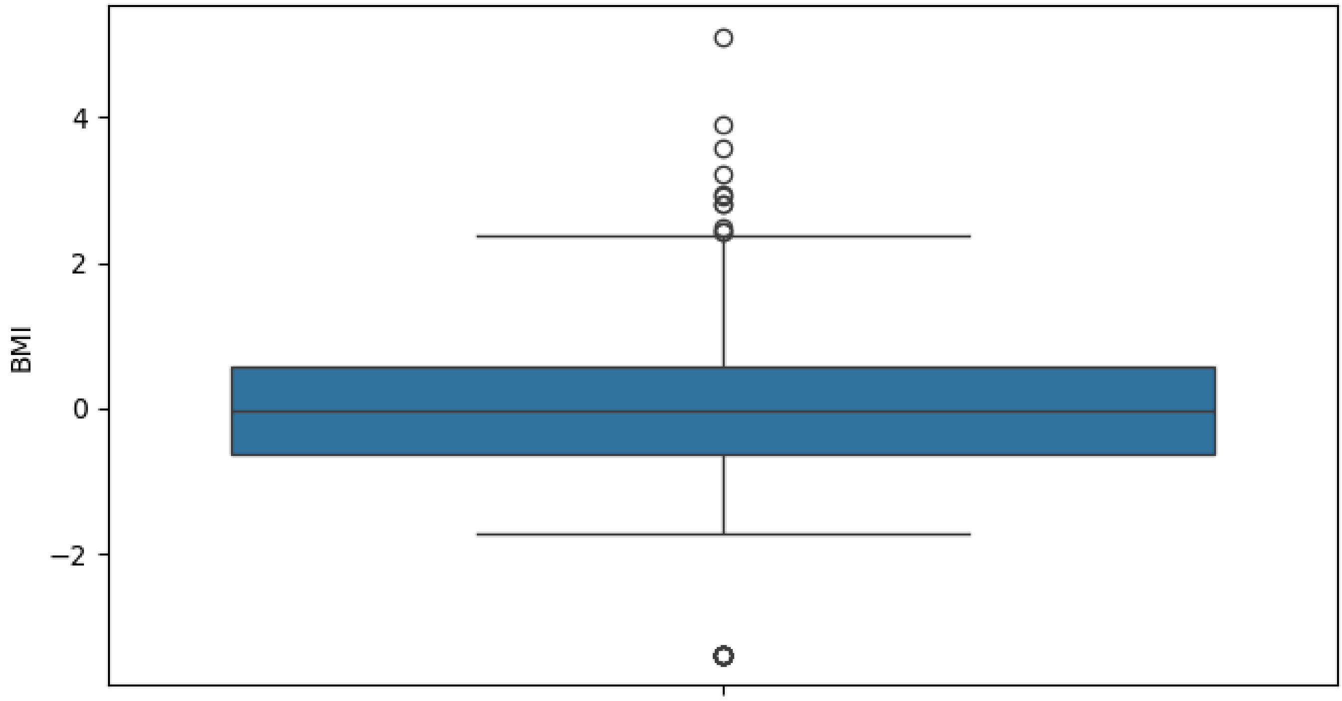
```
outlier(transformed_X)
```



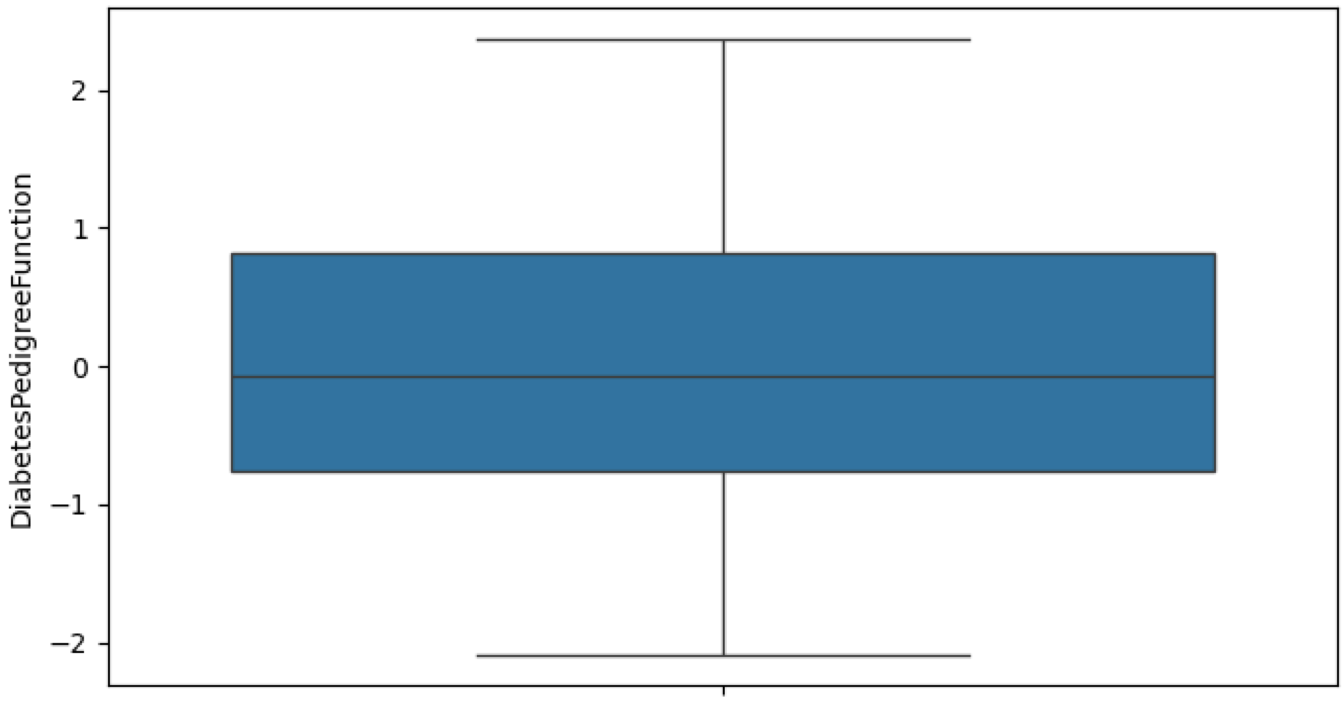

boxplot Plot for Glucose
Glucose
2
1
0
−1
−2
−3
−4

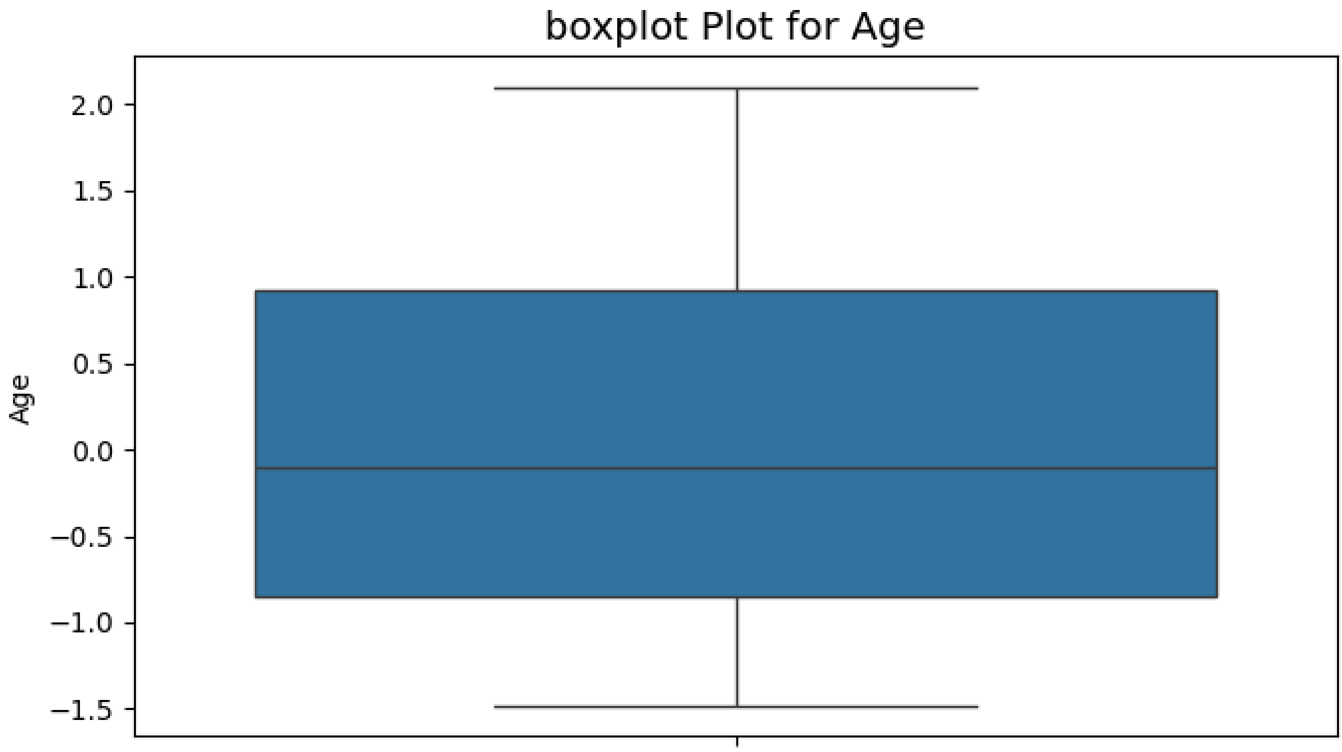

boxplot Plot for Insulin
Insulin
1.5
1.0
0.5
0.0
−0.5
−1.0

boxplot Plot for BMI



boxplot Plot for DiabetesPedigreeFunction





In [127…

```
# Function to cap outliers using IQR
def cap_outliers_iqr(df):
    df_capped = df.copy()  # Make a copy to avoid modifying the original DataFrame

    # Loop through numeric columns and cap outliers based on IQR
    for column in df_capped.select_dtypes(include=[np.number]).columns:
        Q1 = df_capped[column].quantile(0.25)
        Q3 = df_capped[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Cap values outside the bounds to the lower or upper bound
        df_capped[column] = np.clip(df_capped[column], lower_bound, upper_bound)

    return df_capped
```

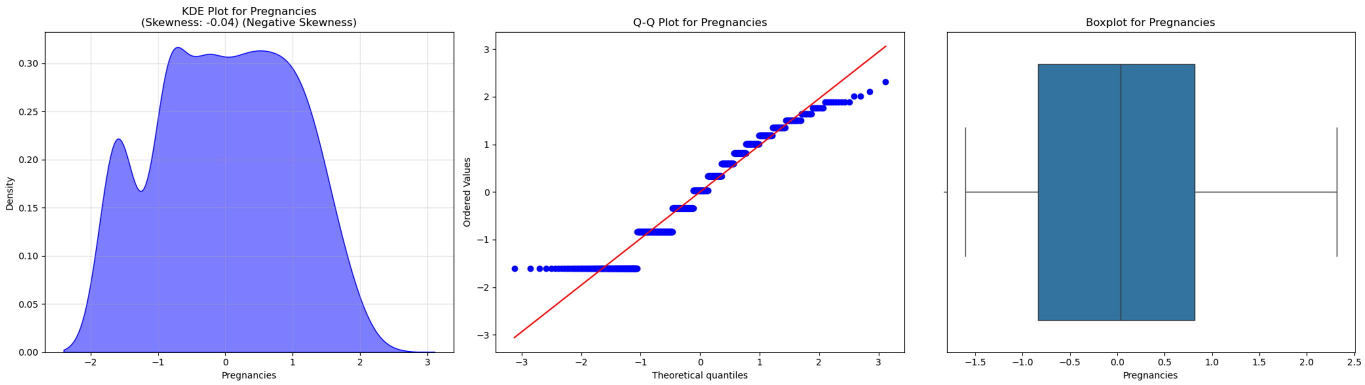
In [129…

```
X_final = cap_outliers_iqr(transformed_X)
```

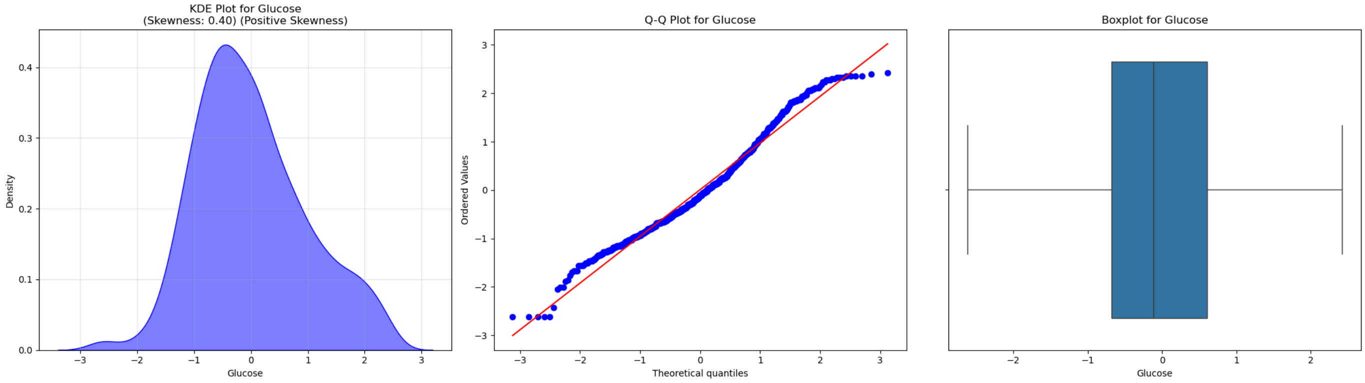
In [133…

```
All_plot(X_final)
```

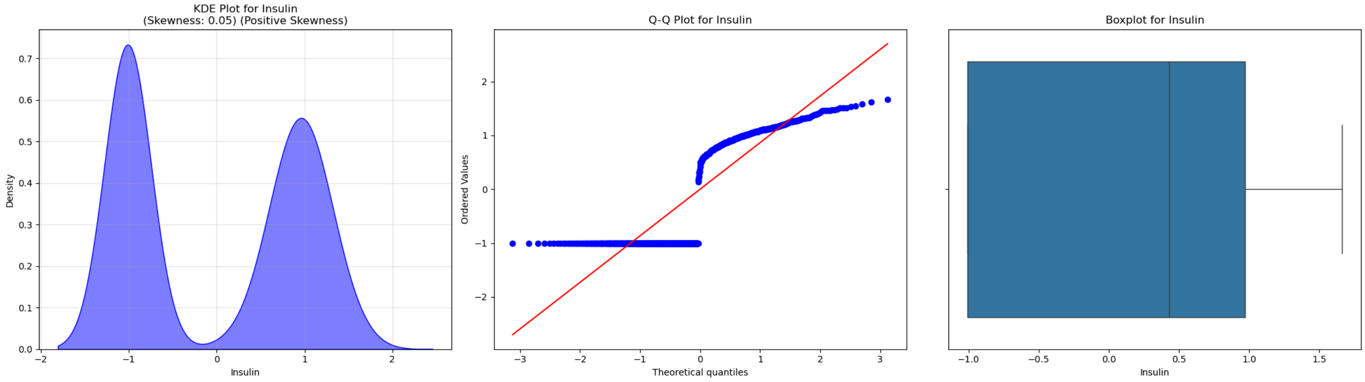
```
Pregnancies: Skewness = -0.04 (Negative Skewness)
```



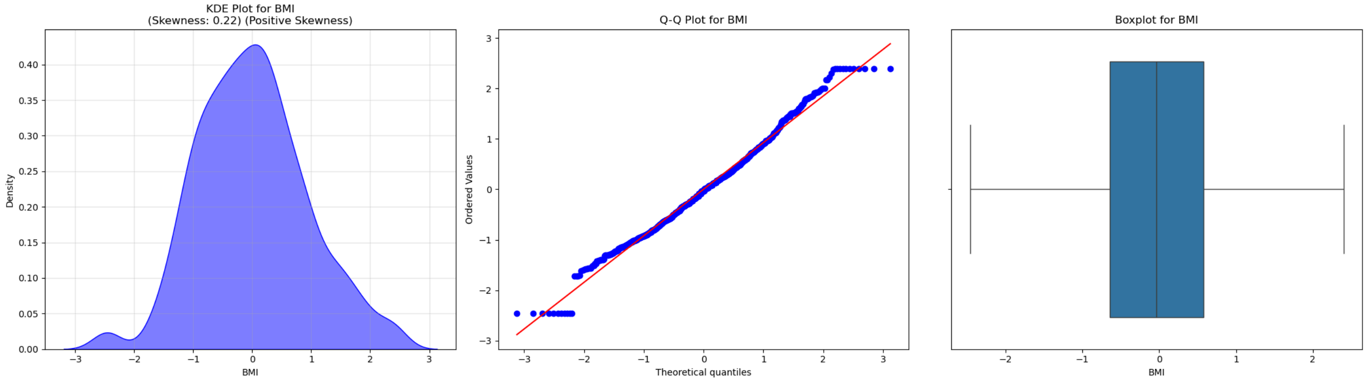

```
Glucose: Skewness = 0.40 (Positive Skewness)
```



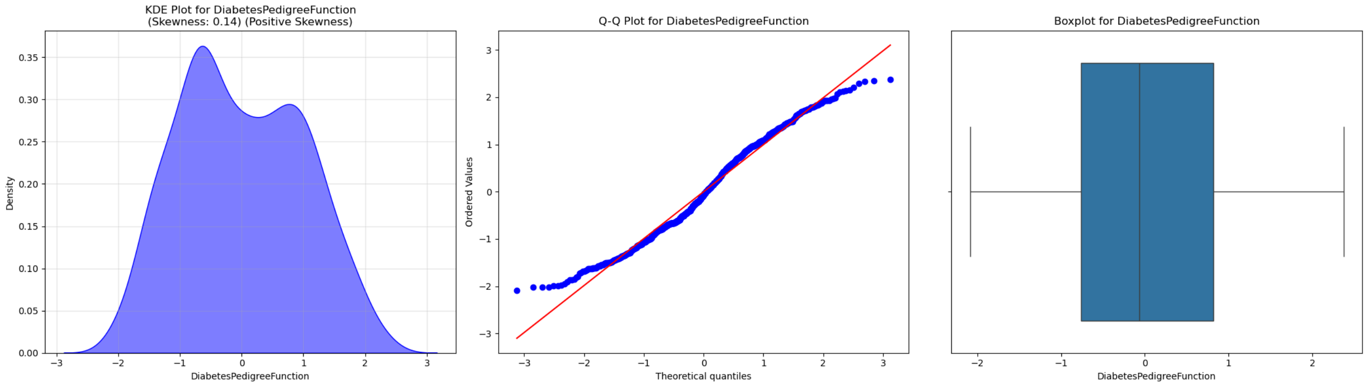
Insulin: Skewness = 0.05 (Positive Skewness)



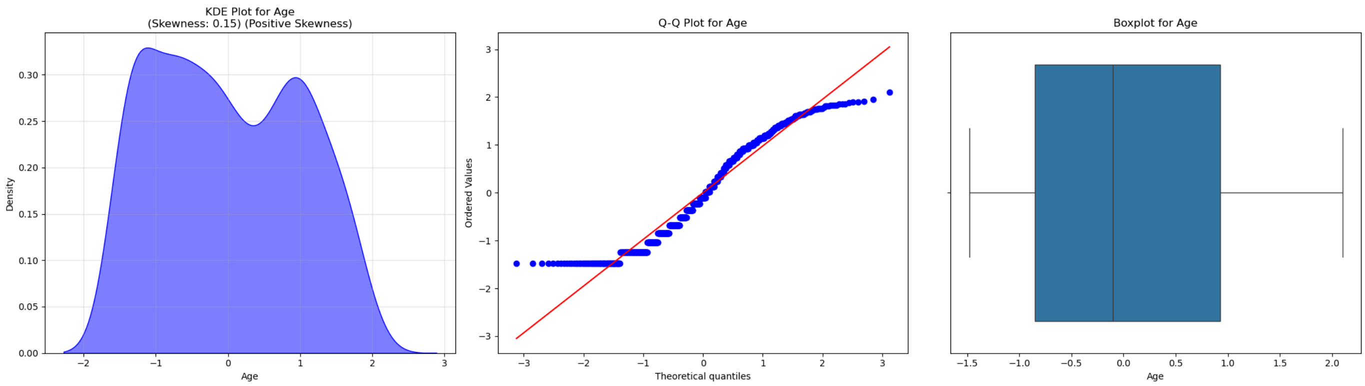
BMI: Skewness = 0.22 (Positive Skewness)



DiabetesPedigreeFunction: Skewness = 0.14 (Positive Skewness)



Age: Skewness = 0.15 (Positive Skewness)



Check Class Balance (for classification target)

In [138…

```
y.value_counts()
```

Out[138…

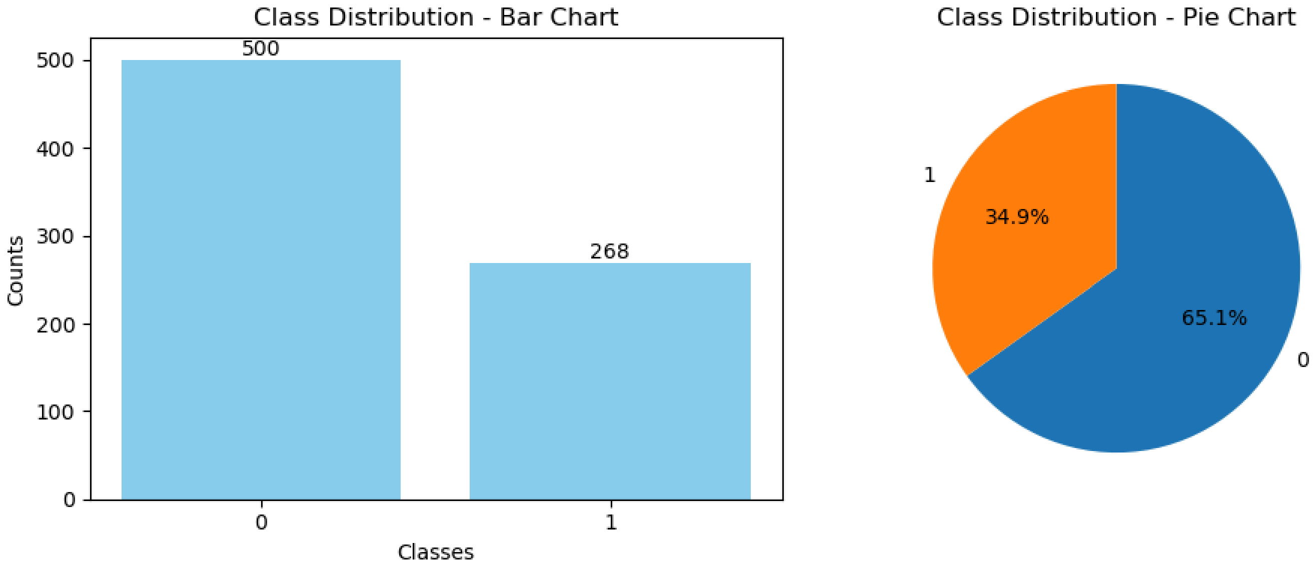
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

In [140…

```
import matplotlib.pyplot as plt

def plot_class_balance(y, figsize=(12,5), bar_color='skyblue', pie_colors=None):
    """
    Plots class distribution of target variable y using bar chart and pie chart side by side.

    Parameters:
    - y: pandas Series or list/array of target labels
    - figsize: tuple, size of the matplotlib figure
    - bar_color: color of bars in bar chart
    - pie_colors: list of colors for pie chart slices, optional

    Returns:
    - None (displays plots)
    """
    # Convert y to pandas Series if not already
    import pandas as pd
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    counts = y.value_counts().sort_index()
    labels = counts.index.astype(str)
    sizes = counts.values

    fig, axes = plt.subplots(1, 2, figsize=figsize)

    # Bar Chart
    axes[0].bar(labels, sizes, color=bar_color)
    axes[0].set_title('Class Distribution - Bar Chart')
    axes[0].set_xlabel('Classes')
    axes[0].set_ylabel('Counts')

    for i, count in enumerate(sizes):
        axes[0].text(i, count + max(sizes)*0.01, str(count), ha='center')

    # Pie Chart
    axes[1].pie(sizes, labels=labels, autopct='%1.1f%%', colors=pie_colors, startangle=90, co
    axes[1].set_title('Class Distribution - Pie Chart')

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

In [155…

```
# Example usage with y as a pandas Series
plot_class_balance(y,figsize=(10,4))
```



Splitting Dataset for Model Training and Evaluation

In [158…

```
X_train,X_test,y_train,y_test = train_test_split(X_final,y,test_size=0.2,random_state=42)
```

In [160…

```
X_train.shape ,X_test.shape ,y_train.shape ,y_test.shape
```

Out[160…

```
((614, 6), (154, 6), (614,), (154,))
```

Train Neural Network with Keras Tuner

Find Optimal Hyperparameters Using Keras Tuner

In [230…

```
def build_model(hp):
    model = Sequential()
    model.add(Input(shape=(6,)))
    #for finding optimal layers
    for i in range(hp.Int('num_layers',min_value=1,max_value=10)):
        #find optimal layers
            model.add(
                Dense(
                    hp.Int('units-'+str(i),min_value=8,max_value=128,step=8),# for optimal no
                    activation = hp.Choice('activation'+str(i),values=['relu','tanh','selu'])
                )
            )
        #find optimal dropout layer value
            model.add(Dropout(hp.Choice('dropout-'+str(i),values=[0.1,0.2,0.3,0.4,0.5,0.6,0.7

    model.add(Dense(1,activation='sigmoid'))

    #optimal optimizer
    optimizers_list = hp.Choice('optimizers',values = ['adam','sgd','rmsprop','adadelta'])
    model.compile(optimizer=optimizers_list,loss='binary_crossentropy',metrics=['accuracy'])

    return model
```

In [232…

```
tuner = kt.RandomSearch(
    build_model,
    objective = 'val_accuracy',
    max_trials=10,
    max_retries_per_trial=1,
    directory='model_tuner',
```

```
    project_name='my_model_tuning'
)
```

In [234…

```
tuner.search(X_train,y_train,epochs=50,validation_data=(X_test,y_test))
```

```
Trial 10 Complete [00h 00m 19s]
val_accuracy: 0.6428571343421936

Best val_accuracy So Far: 0.7922077775001526
Total elapsed time: 00h 03m 04s
```

Retrieve Best Hyperparameters from Tuner

In [237…

```
tuner.get_best_hyperparameters()[0].values
```

Out[237…

```
{'num_layers': 5,
 'units-0': 64,
 'activation0': 'relu',
 'dropout-0': 0.4,
 'optimizers': 'adam',
 'units-1': 40,
 'activation1': 'relu',
 'dropout-1': 0.3,
 'units-2': 16,
 'activation2': 'selu',
 'dropout-2': 0.8,
 'units-3': 128,
 'activation3': 'tanh',
 'dropout-3': 0.3,
 'units-4': 120,
 'activation4': 'relu',
 'dropout-4': 0.6,
 'units-5': 32,
 'activation5': 'tanh',
 'dropout-5': 0.5,
 'units-6': 56,
 'activation6': 'tanh',
 'dropout-6': 0.7,
 'units-7': 56,
 'activation7': 'selu',
 'dropout-7': 0.9}
```

Retrieve the Best Trained Model from Tuner

In [367…

```
model = tuner.get_best_models(num_models=1)[0]
```

```
C:\Users\parvez\anaconda3\Lib\site-packages\keras\src\saving\saving_lib.py:757: UserWarning: S
kipping variable loading for optimizer 'adam', because it has 2 variables whereas the saved op
timizer has 26 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

In [368…

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	448
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 40)	2,600
dropout_1 (Dropout)	(None, 40)	0
dense_2 (Dense)	(None, 16)	656
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 128)	2,176
dropout_3 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 120)	15,480
dropout_4 (Dropout)	(None, 120)	0
dense_5 (Dense)	(None, 1)	121

Total params: 21,481 (83.91 KB)

Trainable params: 21,481 (83.91 KB)

Non-trainable params: 0 (0.00 B)

Implement EarlyStopping to Prevent Overfitting

In [370…

```python
# from tensorflow.keras.callbacks import EarlyStopping
# early_stop = EarlyStopping(
#     monitor='val_accuracy',      # or 'val_accuracy'
#     patience=40,              # how many epochs to wait after no improvement
#     restore_best_weights=True  # revert to best weights after training
# )

from tensorflow.keras.callbacks import Callback
import numpy as np

class AccuracyGapEarlyStopping(Callback):
    def __init__(self, threshold=0.01, patience=5):
        super().__init__()
        self.threshold = threshold
        self.patience = patience
        self.wait = 0

    def on_epoch_end(self, epoch, logs=None):
        acc = logs.get('accuracy')
        val_acc = logs.get('val_accuracy')
        gap = abs(acc - val_acc)

        if gap < self.threshold:
            self.wait += 1
            print(f"Accuracy gap {gap:.4f} is below threshold. Patience count: {self.wait}/{s
            if self.wait >= self.patience:
                print("Stopping training early due to small accuracy gap.")
                self.model.stop_training = True
        else:
            self.wait = 0  # reset if gap grows
```

In [374…

```
gap_stop = AccuracyGapEarlyStopping(threshold=0.01, patience=10)
```

Train the Best_model

In [377…

```
history = model.fit(X_train,y_train,epochs=200,initial_epoch=50,validation_data=(X_test,y_tes
```

```
Epoch 51/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.6945 - loss: 0.6042 - val_accuracy: 0.74
68 - val_loss: 0.5109
Epoch 52/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6947 - loss: 0.5584 - val_accuracy: 0.759
7 - val_loss: 0.5088
Epoch 53/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6856 - loss: 0.5692 - val_accuracy: 0.759
7 - val_loss: 0.5054
Epoch 54/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7168 - loss: 0.5305 - val_accuracy: 0.779
2 - val_loss: 0.5025
Epoch 55/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7586 - loss: 0.5157 - val_accuracy: 0.772
7 - val_loss: 0.5017
Epoch 56/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7465 - loss: 0.5177 - val_accuracy: 0.753
2 - val_loss: 0.5029
Epoch 57/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7638 - loss: 0.5060 - val_accuracy: 0.759
7 - val_loss: 0.5046
Epoch 58/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7196 - loss: 0.5250 - val_accuracy: 0.759
7 - val_loss: 0.5052
Epoch 59/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7061 - loss: 0.5118 - val_accuracy: 0.740
3 - val_loss: 0.5067
Epoch 60/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7305 - loss: 0.5334 - val_accuracy: 0.740
3 - val_loss: 0.5091
Epoch 61/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.7812 - loss: 0.4369Accuracy gap 0.0056
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7525 - loss: 0.4765 - val_accuracy: 0.733
8 - val_loss: 0.5090
Epoch 62/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.6562 - loss: 0.4770Accuracy gap 0.0025
is below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7223 - loss: 0.5042 - val_accuracy: 0.733
8 - val_loss: 0.5070
Epoch 63/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.9062 - loss: 0.3650Accuracy gap 0.0008 i
s below threshold. Patience count: 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7549 - loss: 0.5046 - val_accuracy: 0.740
3 - val_loss: 0.5100
Epoch 64/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7812 - loss: 0.4557Accuracy gap 0.0007
is below threshold. Patience count: 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7326 - loss: 0.5366 - val_accuracy: 0.727
3 - val_loss: 0.5069
Epoch 65/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 1s 58ms/step - accuracy: 0.6875 - loss: 0.7645Accuracy gap 0.0040 i
s below threshold. Patience count: 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7372 - loss: 0.5609 - val_accuracy: 0.733
8 - val_loss: 0.5094
Epoch 66/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7326 - loss: 0.5046 - val_accuracy: 0.759
7 - val_loss: 0.5085
Epoch 67/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7533 - loss: 0.5032 - val_accuracy: 0.727
3 - val_loss: 0.5089
Epoch 68/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.7188 - loss: 0.4960Accuracy gap 0.0024 i
s below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7306 - loss: 0.5054 - val_accuracy: 0.740
3 - val_loss: 0.5099
```

```
Epoch 69/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7439 - loss: 0.5185 - val_accuracy: 0.772
7 - val_loss: 0.5099
Epoch 70/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 187ms/step - accuracy: 0.7188 - loss: 0.6219Accuracy gap 0.0041
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7510 - loss: 0.5006 - val_accuracy: 0.759
7 - val_loss: 0.5174
Epoch 71/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7386 - loss: 0.5058 - val_accuracy: 0.746
8 - val_loss: 0.5209
Epoch 72/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7492 - loss: 0.5155 - val_accuracy: 0.766
2 - val_loss: 0.5137
Epoch 73/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7781 - loss: 0.4663 - val_accuracy: 0.766
2 - val_loss: 0.5099
Epoch 74/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.8438 - loss: 0.3558Accuracy gap 0.0073
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7747 - loss: 0.4574 - val_accuracy: 0.753
2 - val_loss: 0.5108
Epoch 75/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7296 - loss: 0.4878 - val_accuracy: 0.746
8 - val_loss: 0.5141
Epoch 76/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 190ms/step - accuracy: 0.8125 - loss: 0.5151Accuracy gap 0.0056
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7559 - loss: 0.5010 - val_accuracy: 0.766
2 - val_loss: 0.5217
Epoch 77/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7812 - loss: 0.4202Accuracy gap 0.0089 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7623 - loss: 0.4812 - val_accuracy: 0.759
7 - val_loss: 0.5200
Epoch 78/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7547 - loss: 0.5147 - val_accuracy: 0.766
2 - val_loss: 0.5211
Epoch 79/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.7500 - loss: 0.5239Accuracy gap 0.0057
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7587 - loss: 0.4820 - val_accuracy: 0.746
8 - val_loss: 0.5195
Epoch 80/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7578 - loss: 0.4752 - val_accuracy: 0.772
7 - val_loss: 0.5251
Epoch 81/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7441 - loss: 0.4983 - val_accuracy: 0.740
3 - val_loss: 0.5305
Epoch 82/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 190ms/step - accuracy: 0.7812 - loss: 0.4112Accuracy gap 0.0008
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7431 - loss: 0.4918 - val_accuracy: 0.753
2 - val_loss: 0.5297
Epoch 83/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 190ms/step - accuracy: 0.7188 - loss: 0.5060Accuracy gap 0.0073
is below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7455 - loss: 0.4797 - val_accuracy: 0.753
2 - val_loss: 0.5377
Epoch 84/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.7812 - loss: 0.3983Accuracy gap 0.0057
is below threshold. Patience count: 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7401 - loss: 0.4938 - val_accuracy: 0.753
2 - val_loss: 0.5340
Epoch 85/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7417 - loss: 0.4606 - val_accuracy: 0.766
```

```
2 - val_loss: 0.5342
Epoch 86/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7542 - loss: 0.4778 - val_accuracy: 0.772
7 - val_loss: 0.5376
Epoch 87/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7500 - loss: 0.4931Accuracy gap 0.0024
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7450 - loss: 0.4753 - val_accuracy: 0.746
8 - val_loss: 0.5353
Epoch 88/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7582 - loss: 0.4923 - val_accuracy: 0.740
3 - val_loss: 0.5418
Epoch 89/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7188 - loss: 0.4703Accuracy gap 0.0057
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7256 - loss: 0.4948 - val_accuracy: 0.740
3 - val_loss: 0.5419
Epoch 90/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.7812 - loss: 0.4630Accuracy gap 0.0008 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7686 - loss: 0.4892 - val_accuracy: 0.766
2 - val_loss: 0.5485
Epoch 91/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7567 - loss: 0.4749 - val_accuracy: 0.766
2 - val_loss: 0.5522
Epoch 92/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7318 - loss: 0.4931 - val_accuracy: 0.753
2 - val_loss: 0.5565
Epoch 93/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.7500 - loss: 0.5167Accuracy gap 0.0040 i
s below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7382 - loss: 0.4976 - val_accuracy: 0.759
7 - val_loss: 0.5532
Epoch 94/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7460 - loss: 0.4761 - val_accuracy: 0.766
2 - val_loss: 0.5502
Epoch 95/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 190ms/step - accuracy: 0.7188 - loss: 0.4632Accuracy gap 0.0041
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7490 - loss: 0.5089 - val_accuracy: 0.753
2 - val_loss: 0.5394
Epoch 96/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.7812 - loss: 0.4272Accuracy gap 0.0057 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7463 - loss: 0.5184 - val_accuracy: 0.753
2 - val_loss: 0.5339
Epoch 97/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7743 - loss: 0.4793 - val_accuracy: 0.753
2 - val_loss: 0.5383
Epoch 98/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7812 - loss: 0.4462Accuracy gap 0.0056
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7779 - loss: 0.4548 - val_accuracy: 0.772
7 - val_loss: 0.5490
Epoch 99/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.7812 - loss: 0.3669Accuracy gap 0.0057 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7663 - loss: 0.4571 - val_accuracy: 0.753
2 - val_loss: 0.5531
Epoch 100/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7531 - loss: 0.4686 - val_accuracy: 0.759
7 - val_loss: 0.5534
Epoch 101/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7645 - loss: 0.4936 - val_accuracy: 0.759
7 - val_loss: 0.5501
Epoch 102/200
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7970 - loss: 0.4393 - val_accuracy: 0.753
2 - val_loss: 0.5507
Epoch 103/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 4s 212ms/step - accuracy: 0.6250 - loss: 0.6125Accuracy gap 0.0057
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7333 - loss: 0.4920 - val_accuracy: 0.759
7 - val_loss: 0.5542
Epoch 104/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7935 - loss: 0.4447 - val_accuracy: 0.766
2 - val_loss: 0.5489
Epoch 105/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7671 - loss: 0.4528 - val_accuracy: 0.772
7 - val_loss: 0.5483
Epoch 106/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7570 - loss: 0.5135 - val_accuracy: 0.772
7 - val_loss: 0.5374
Epoch 107/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7576 - loss: 0.4820 - val_accuracy: 0.779
2 - val_loss: 0.5437
Epoch 108/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7478 - loss: 0.4661 - val_accuracy: 0.779
2 - val_loss: 0.5533
Epoch 109/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7498 - loss: 0.4651 - val_accuracy: 0.785
7 - val_loss: 0.5514
Epoch 110/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7663 - loss: 0.4482 - val_accuracy: 0.779
2 - val_loss: 0.5486
Epoch 111/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.8125 - loss: 0.5300Accuracy gap 0.0089 i
s below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7578 - loss: 0.5012 - val_accuracy: 0.772
7 - val_loss: 0.5442
Epoch 112/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.7281 - loss: 0.4929 - val_accuracy: 0.79
22 - val_loss: 0.5447
Epoch 113/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7516 - loss: 0.5001 - val_accuracy: 0.785
7 - val_loss: 0.5378
Epoch 114/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7789 - loss: 0.4209 - val_accuracy: 0.779
2 - val_loss: 0.5408
Epoch 115/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7952 - loss: 0.4475 - val_accuracy: 0.772
7 - val_loss: 0.5402
Epoch 116/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7476 - loss: 0.4795 - val_accuracy: 0.779
2 - val_loss: 0.5406
Epoch 117/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7890 - loss: 0.4634 - val_accuracy: 0.785
7 - val_loss: 0.5416
Epoch 118/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7689 - loss: 0.4944 - val_accuracy: 0.779
2 - val_loss: 0.5432
Epoch 119/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7670 - loss: 0.4617 - val_accuracy: 0.779
2 - val_loss: 0.5508
Epoch 120/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7480 - loss: 0.4573 - val_accuracy: 0.785
7 - val_loss: 0.5545
Epoch 121/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7683 - loss: 0.4528 - val_accuracy: 0.779
2 - val_loss: 0.5515
Epoch 122/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7634 - loss: 0.4431 - val_accuracy: 0.779
2 - val_loss: 0.5537
```

```
Epoch 123/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.6250 - loss: 0.6168Accuracy gap 0.0025
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7731 - loss: 0.4563 - val_accuracy: 0.779
2 - val_loss: 0.5517
Epoch 124/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7504 - loss: 0.4586 - val_accuracy: 0.792
2 - val_loss: 0.5484
Epoch 125/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7500 - loss: 0.4691 - val_accuracy: 0.785
7 - val_loss: 0.5490
Epoch 126/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7902 - loss: 0.4592 - val_accuracy: 0.772
7 - val_loss: 0.5555
Epoch 127/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7504 - loss: 0.4555 - val_accuracy: 0.779
2 - val_loss: 0.5555
Epoch 128/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7601 - loss: 0.4481 - val_accuracy: 0.792
2 - val_loss: 0.5582
Epoch 129/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7543 - loss: 0.4770 - val_accuracy: 0.779
2 - val_loss: 0.5618
Epoch 130/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7490 - loss: 0.4526 - val_accuracy: 0.772
7 - val_loss: 0.5692
Epoch 131/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7432 - loss: 0.4685 - val_accuracy: 0.78
57 - val_loss: 0.5663
Epoch 132/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7763 - loss: 0.4519 - val_accuracy: 0.785
7 - val_loss: 0.5587
Epoch 133/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7468 - loss: 0.4515 - val_accuracy: 0.785
7 - val_loss: 0.5618
Epoch 134/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7766 - loss: 0.4556 - val_accuracy: 0.792
2 - val_loss: 0.5654
Epoch 135/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7509 - loss: 0.4699 - val_accuracy: 0.753
2 - val_loss: 0.5552
Epoch 136/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7636 - loss: 0.4476 - val_accuracy: 0.753
2 - val_loss: 0.5605
Epoch 137/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7730 - loss: 0.4468 - val_accuracy: 0.772
7 - val_loss: 0.5665
Epoch 138/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7535 - loss: 0.4472 - val_accuracy: 0.779
2 - val_loss: 0.5702
Epoch 139/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.7812 - loss: 0.4121Accuracy gap 0.0024
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7713 - loss: 0.4405 - val_accuracy: 0.772
7 - val_loss: 0.5723
Epoch 140/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7458 - loss: 0.4656 - val_accuracy: 0.772
7 - val_loss: 0.5763
Epoch 141/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7892 - loss: 0.4379 - val_accuracy: 0.759
7 - val_loss: 0.5679
Epoch 142/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 191ms/step - accuracy: 0.8438 - loss: 0.4112Accuracy gap 0.0024
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7633 - loss: 0.4459 - val_accuracy: 0.766
2 - val_loss: 0.5801
```

```
Epoch 143/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.6562 - loss: 0.5655Accuracy gap 0.0025 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7606 - loss: 0.4528 - val_accuracy: 0.766
2 - val_loss: 0.5831
Epoch 144/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 207ms/step - accuracy: 0.7812 - loss: 0.4366Accuracy gap 0.0024
is below threshold. Patience count: 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7728 - loss: 0.4233 - val_accuracy: 0.766
2 - val_loss: 0.5851
Epoch 145/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7583 - loss: 0.4209 - val_accuracy: 0.779
2 - val_loss: 0.5821
Epoch 146/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.7812 - loss: 0.4223Accuracy gap 0.0041
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7830 - loss: 0.4227 - val_accuracy: 0.753
2 - val_loss: 0.5742
Epoch 147/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7799 - loss: 0.4374 - val_accuracy: 0.759
7 - val_loss: 0.5746
Epoch 148/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7188 - loss: 0.4689Accuracy gap 0.0041
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7623 - loss: 0.4505 - val_accuracy: 0.759
7 - val_loss: 0.5759
Epoch 149/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7767 - loss: 0.4270 - val_accuracy: 0.759
7 - val_loss: 0.5857
Epoch 150/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7646 - loss: 0.4366 - val_accuracy: 0.785
7 - val_loss: 0.5921
Epoch 151/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7512 - loss: 0.4751 - val_accuracy: 0.779
2 - val_loss: 0.5895
Epoch 152/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7362 - loss: 0.4530 - val_accuracy: 0.785
7 - val_loss: 0.5854
Epoch 153/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7693 - loss: 0.4609 - val_accuracy: 0.785
7 - val_loss: 0.5820
Epoch 154/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.6875 - loss: 0.4172Accuracy gap 0.0007
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7742 - loss: 0.4269 - val_accuracy: 0.779
2 - val_loss: 0.5912
Epoch 155/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 187ms/step - accuracy: 0.7500 - loss: 0.6369Accuracy gap 0.0072
is below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7673 - loss: 0.4852 - val_accuracy: 0.779
2 - val_loss: 0.5777
Epoch 156/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7776 - loss: 0.4565 - val_accuracy: 0.779
2 - val_loss: 0.5762
Epoch 157/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7588 - loss: 0.4471 - val_accuracy: 0.779
2 - val_loss: 0.5835
Epoch 158/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.8125 - loss: 0.4143Accuracy gap 0.0089
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7833 - loss: 0.4074 - val_accuracy: 0.772
7 - val_loss: 0.5862
Epoch 159/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7834 - loss: 0.4493 - val_accuracy: 0.753
2 - val_loss: 0.5870
Epoch 160/200
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7361 - loss: 0.4942 - val_accuracy: 0.779
2 - val_loss: 0.5823
Epoch 161/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7874 - loss: 0.4122 - val_accuracy: 0.792
2 - val_loss: 0.5861
Epoch 162/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7812 - loss: 0.3310Accuracy gap 0.0088
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7694 - loss: 0.4399 - val_accuracy: 0.785
7 - val_loss: 0.5771
Epoch 163/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.8438 - loss: 0.3814Accuracy gap 0.0058
is below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8039 - loss: 0.4211 - val_accuracy: 0.785
7 - val_loss: 0.5829
Epoch 164/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.8438 - loss: 0.3656Accuracy gap 0.0072 i
s below threshold. Patience count: 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7928 - loss: 0.4134 - val_accuracy: 0.785
7 - val_loss: 0.5795
Epoch 165/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7657 - loss: 0.4279 - val_accuracy: 0.798
7 - val_loss: 0.5825
Epoch 166/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7812 - loss: 0.4493 - val_accuracy: 0.792
2 - val_loss: 0.5796
Epoch 167/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7571 - loss: 0.4473 - val_accuracy: 0.792
2 - val_loss: 0.5763
Epoch 168/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7721 - loss: 0.4679 - val_accuracy: 0.792
2 - val_loss: 0.5730
Epoch 169/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7338 - loss: 0.4651 - val_accuracy: 0.785
7 - val_loss: 0.5842
Epoch 170/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8750 - loss: 0.4280Accuracy gap 0.0010 i
s below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8059 - loss: 0.4513 - val_accuracy: 0.792
2 - val_loss: 0.5806
Epoch 171/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7612 - loss: 0.4558 - val_accuracy: 0.792
2 - val_loss: 0.5799
Epoch 172/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.8125 - loss: 0.4186Accuracy gap 0.0058
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7988 - loss: 0.4379 - val_accuracy: 0.785
7 - val_loss: 0.5752
Epoch 173/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7947 - loss: 0.4314 - val_accuracy: 0.792
2 - val_loss: 0.5801
Epoch 174/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7717 - loss: 0.4381 - val_accuracy: 0.792
2 - val_loss: 0.5729
Epoch 175/200
17/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7825 - loss: 0.3907  Accuracy gap 0.0088
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7813 - loss: 0.3971 - val_accuracy: 0.785
7 - val_loss: 0.5785
Epoch 176/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.7500 - loss: 0.4743Accuracy gap 0.0040 i
s below threshold. Patience count: 2/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7788 - loss: 0.4209 - val_accuracy: 0.779
2 - val_loss: 0.5921
Epoch 177/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 189ms/step - accuracy: 0.7500 - loss: 0.5515Accuracy gap 0.0023
```

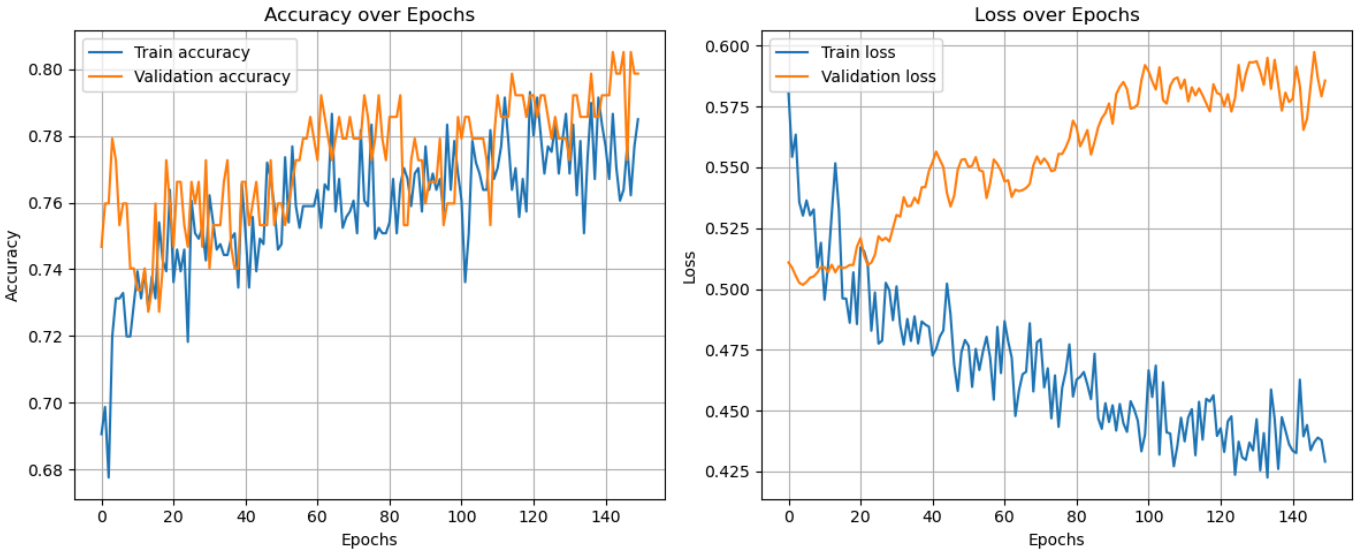
```
is below threshold. Patience count: 3/10
20/20 0s 8ms/step - accuracy: 0.7781 - loss: 0.4450 - val_accuracy: 0.785
7 - val_loss: 0.5815
Epoch 178/200
20/20 0s 6ms/step - accuracy: 0.7431 - loss: 0.4997 - val_accuracy: 0.779
2 - val_loss: 0.5885
Epoch 179/200
 1/20 1s 60ms/step - accuracy: 0.7500 - loss: 0.4133Accuracy gap 0.0007 i
s below threshold. Patience count: 1/10
20/20 0s 8ms/step - accuracy: 0.7730 - loss: 0.4294 - val_accuracy: 0.779
2 - val_loss: 0.5932
Epoch 180/200
 1/20 3s 187ms/step - accuracy: 0.7812 - loss: 0.4935Accuracy gap 0.0074
is below threshold. Patience count: 2/10
20/20 0s 8ms/step - accuracy: 0.7865 - loss: 0.4169 - val_accuracy: 0.779
2 - val_loss: 0.5932
Epoch 181/200
 1/20 3s 189ms/step - accuracy: 0.6875 - loss: 0.4419Accuracy gap 0.0040
is below threshold. Patience count: 3/10
20/20 0s 8ms/step - accuracy: 0.7511 - loss: 0.4374 - val_accuracy: 0.772
7 - val_loss: 0.5936
Epoch 182/200
 1/20 3s 189ms/step - accuracy: 0.9062 - loss: 0.3698Accuracy gap 0.0088
is below threshold. Patience count: 4/10
20/20 0s 8ms/step - accuracy: 0.7918 - loss: 0.4219 - val_accuracy: 0.792
2 - val_loss: 0.5893
Epoch 183/200
20/20 0s 8ms/step - accuracy: 0.7556 - loss: 0.4638 - val_accuracy: 0.792
2 - val_loss: 0.5839
Epoch 184/200
 1/20 3s 194ms/step - accuracy: 0.7188 - loss: 0.4191Accuracy gap 0.0072
is below threshold. Patience count: 1/10
20/20 0s 6ms/step - accuracy: 0.7637 - loss: 0.4204 - val_accuracy: 0.785
7 - val_loss: 0.5950
Epoch 185/200
20/20 0s 8ms/step - accuracy: 0.7542 - loss: 0.4678 - val_accuracy: 0.785
7 - val_loss: 0.5822
Epoch 186/200
20/20 0s 8ms/step - accuracy: 0.7825 - loss: 0.4354 - val_accuracy: 0.785
7 - val_loss: 0.5942
Epoch 187/200
 1/20 3s 189ms/step - accuracy: 0.8438 - loss: 0.3642Accuracy gap 0.0088
is below threshold. Patience count: 1/10
20/20 0s 8ms/step - accuracy: 0.7868 - loss: 0.4196 - val_accuracy: 0.798
7 - val_loss: 0.5840
Epoch 188/200
20/20 0s 8ms/step - accuracy: 0.7586 - loss: 0.4512 - val_accuracy: 0.785
7 - val_loss: 0.5733
Epoch 189/200
 1/20 3s 188ms/step - accuracy: 0.7500 - loss: 0.5547Accuracy gap 0.0058
is below threshold. Patience count: 1/10
20/20 0s 9ms/step - accuracy: 0.7791 - loss: 0.4474 - val_accuracy: 0.785
7 - val_loss: 0.5807
Epoch 190/200
 1/20 3s 170ms/step - accuracy: 0.6875 - loss: 0.6493Accuracy gap 0.0088
is below threshold. Patience count: 2/10
20/20 0s 8ms/step - accuracy: 0.7770 - loss: 0.4658 - val_accuracy: 0.792
2 - val_loss: 0.5769
Epoch 191/200
20/20 0s 8ms/step - accuracy: 0.7734 - loss: 0.4380 - val_accuracy: 0.792
2 - val_loss: 0.5781
Epoch 192/200
20/20 0s 9ms/step - accuracy: 0.8040 - loss: 0.3964 - val_accuracy: 0.792
2 - val_loss: 0.5914
Epoch 193/200
20/20 0s 5ms/step - accuracy: 0.7782 - loss: 0.4517 - val_accuracy: 0.805
```

```
2 - val_loss: 0.5835
Epoch 194/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7826 - loss: 0.4243 - val_accuracy: 0.798
7 - val_loss: 0.5654
Epoch 195/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7403 - loss: 0.4665 - val_accuracy: 0.798
7 - val_loss: 0.5700
Epoch 196/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7698 - loss: 0.4141 - val_accuracy: 0.805
2 - val_loss: 0.5837
Epoch 197/200
 1/20 ━━━━━━━━━━━━━━━━━━━━ 3s 186ms/step - accuracy: 0.8125 - loss: 0.5359Accuracy gap 0.0041
is below threshold. Patience count: 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7669 - loss: 0.4573 - val_accuracy: 0.772
7 - val_loss: 0.5974
Epoch 198/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7640 - loss: 0.4262 - val_accuracy: 0.805
2 - val_loss: 0.5866
Epoch 199/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7704 - loss: 0.4398 - val_accuracy: 0.798
7 - val_loss: 0.5792
Epoch 200/200
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7941 - loss: 0.4303 - val_accuracy: 0.798
7 - val_loss: 0.5857
```

Plot Training and Validation Accuracy & Loss Side by Side

In [379…

```python
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Accuracy plot
axs[0].plot(history.history['accuracy'], label='Train accuracy')
axs[0].plot(history.history['val_accuracy'], label='Validation accuracy')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Accuracy')
axs[0].legend()
axs[0].grid(True)
axs[0].set_title('Accuracy over Epochs')

# Loss plot
axs[1].plot(history.history['loss'], label='Train loss')
axs[1].plot(history.history['val_loss'], label='Validation loss')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Loss')
axs[1].legend()
axs[1].grid(True)
axs[1].set_title('Loss over Epochs')

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



Predict on Test Data and Evaluate Model Accuracy

In [381…

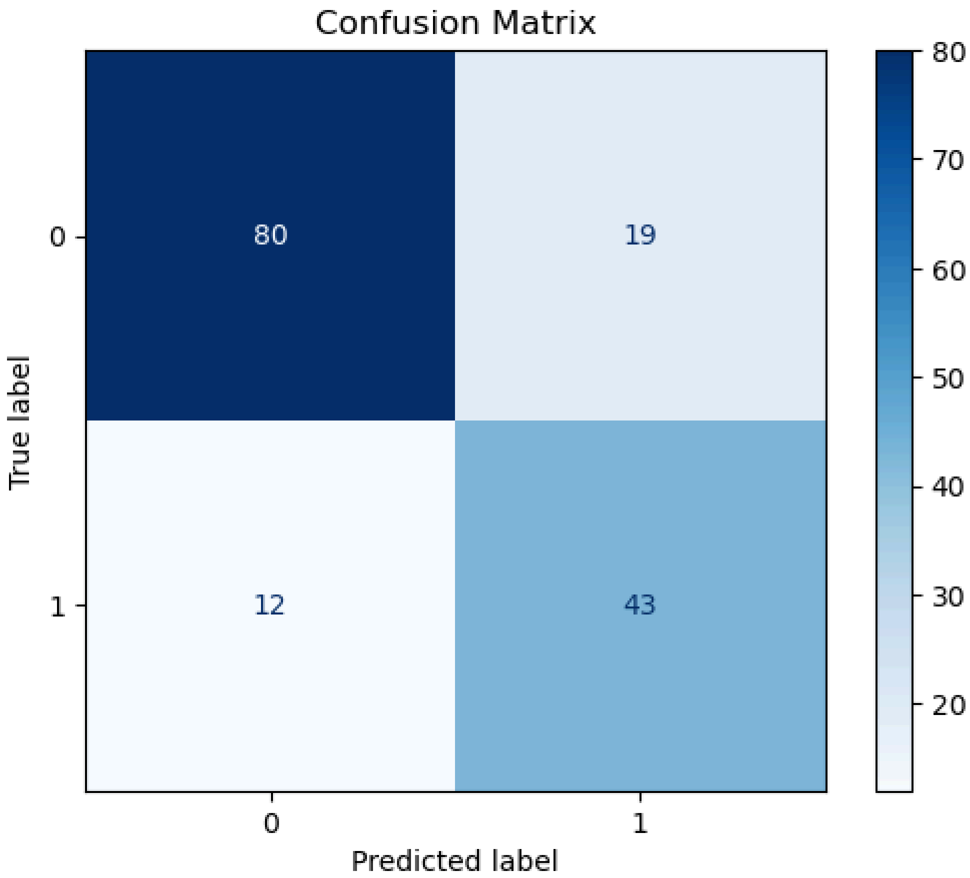
```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype("int32")
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy of the model is : ",accuracy)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Accuracy of the model is :  0.7987012987012987
```

Plot Confusion Matrix to Evaluate Classification Performance

In [383…

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```



Calculate and Display F1 Scores for the Model

In [385…

```
#calculate F1 score
def All_F1_score(y_test, y_pred):
    from sklearn.metrics import precision_score, recall_score, f1_score

    # Compute F1 Scores
    f1_micro = f1_score(y_test, y_pred, average='micro')
    f1_macro = f1_score(y_test, y_pred, average='macro')
    f1_weighted = f1_score(y_test, y_pred, average='weighted')

    # Print Results
    print(f"Micro F1 Score: {f1_micro}")
    print(f"Macro F1 Score: {f1_macro}")
    print(f"Weighted F1 Score: {f1_weighted}")
```

In [386…

```
All_F1_score(y_test, y_pred)
```

```
Micro F1 Score: 0.7987012987012987
Macro F1 Score: 0.7863695350606346
Weighted F1 Score: 0.8010343350657487
```

In [ ]: