Relatório do Projeto Bimestral - Desenvolvimento de Solução com Redes Neurais MLP

1. Introdução

Neste projeto, desenvolvemos uma solução utilizando redes neurais do tipo **Multilayer Perceptron (MLP)**, com o objetivo de prever a ocorrência de diabetes em pacientes, a partir do conjunto de dados **Pima Indians Diabetes Database**.

A escolha desse dataset se justifica pela relevância do tema de saúde pública e pela disponibilidade do conjunto de dados em repositórios públicos. O problema proposto se enquadra na categoria de **classificação binária** (ter ou não ter diabetes), e a rede MLP foi escolhida por sua versatilidade e facilidade de aplicação em tarefas de classificação.

Objetivos Principais:

1. Selecionar e preparar a base de dados (limpeza, análise exploratória e pré-processamento).

2. Projetar, treinar e avaliar um modelo MLP adequado.

3. Analisar o desempenho do modelo utilizando métricas apropriadas.

4. Desenvolver uma aplicação interativa para que usuários possam testar o modelo.

2. Base de Dados

2.1 Origem dos Dados

O **Pima Indians Diabetes Database** foi obtido no repositório do GitHub do autor original ou em repositórios públicos como Kaggle e UCI Machine Learning Repository. O dataset contém informações de exames médicos de mulheres de etnia Pima, incluindo variáveis como nível de glicose, pressão arterial, espessura de pele, níveis de insulina, IMC, entre outras.

2.2 Descrição das Variáveis

- **Pregnancies**: Número de gravidezes.
- **Glucose**: Concentração de glicose no plasma em 2 horas.
- **BloodPressure**: Pressão arterial diastólica (mm Hg).
- **SkinThickness**: Espessura da pele do tríceps (mm).
- **Insulin**: Nível de insulina sérica (mu U/ml).
- **BMI**: Índice de Massa Corporal.
- **DiabetesPedigreeFunction** (DPF): Função de pedigree de diabetes (mede a hereditariedade).
- **Age**: Idade (anos).
- **Outcome**: Variável alvo (0 = não tem diabetes, 1 = tem diabetes).

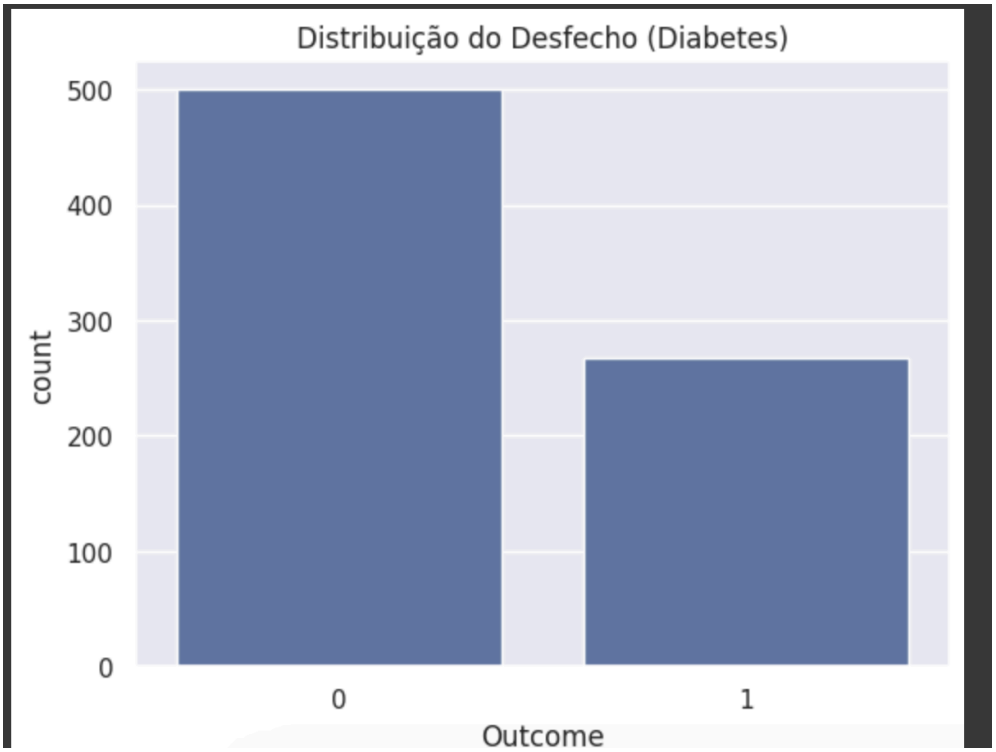
2.3 Análise Exploratória

Para entender melhor o comportamento dos dados, foram realizados:

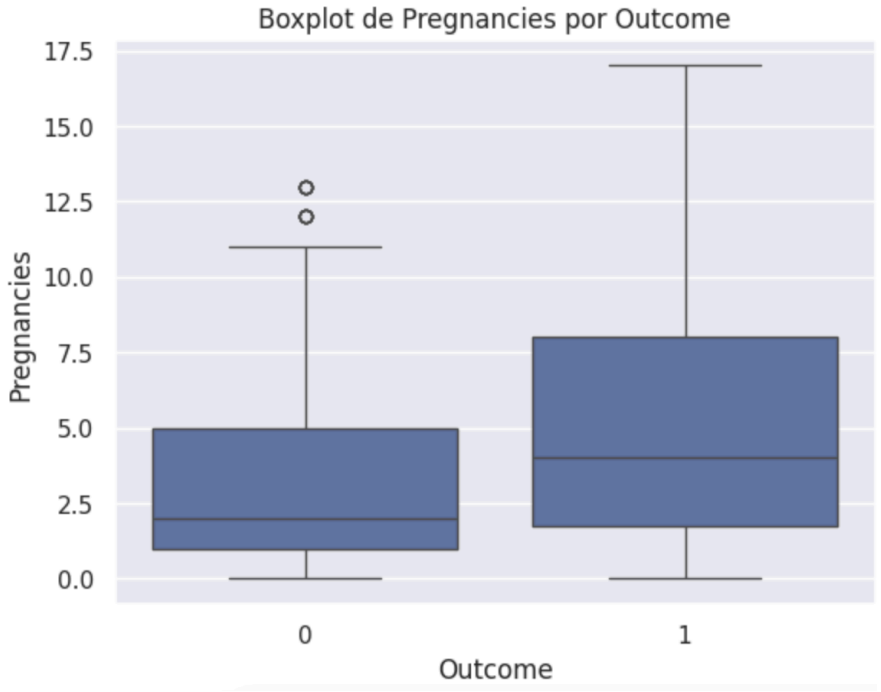
1. **Estatísticas Descritivas** (`df.describe()`).

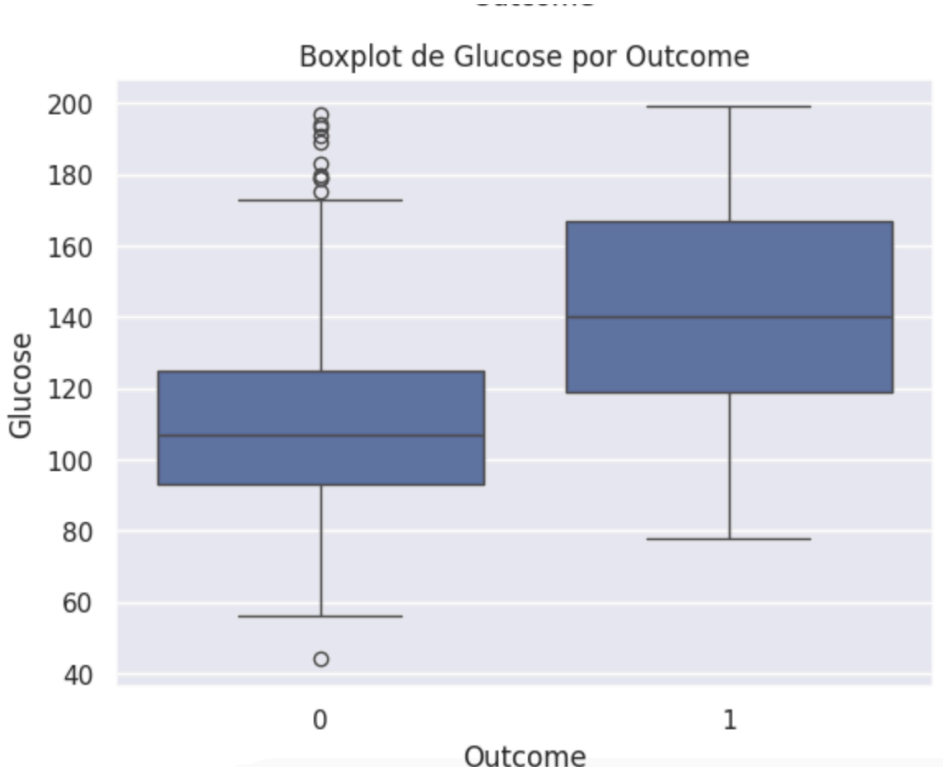
```
Descrição estatística do dataset:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin
count   768.000000  763.000000     733.000000     541.000000  394.000000
mean      3.845052  121.686763      72.405184      29.153420  155.548223
std       3.369578   30.535641      12.382158      10.476982  118.775855
min       0.000000   44.000000      24.000000       7.000000   14.000000
25%       1.000000   99.000000      64.000000      22.000000   76.250000
50%       3.000000  117.000000      72.000000      29.000000  125.000000
75%       6.000000  141.000000      80.000000      36.000000  190.000000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  757.000000                768.000000  768.000000  768.000000
mean    32.457464                  0.471876   33.240885    0.348958
std      6.924988                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  0.243750   24.000000    0.000000
50%     32.300000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

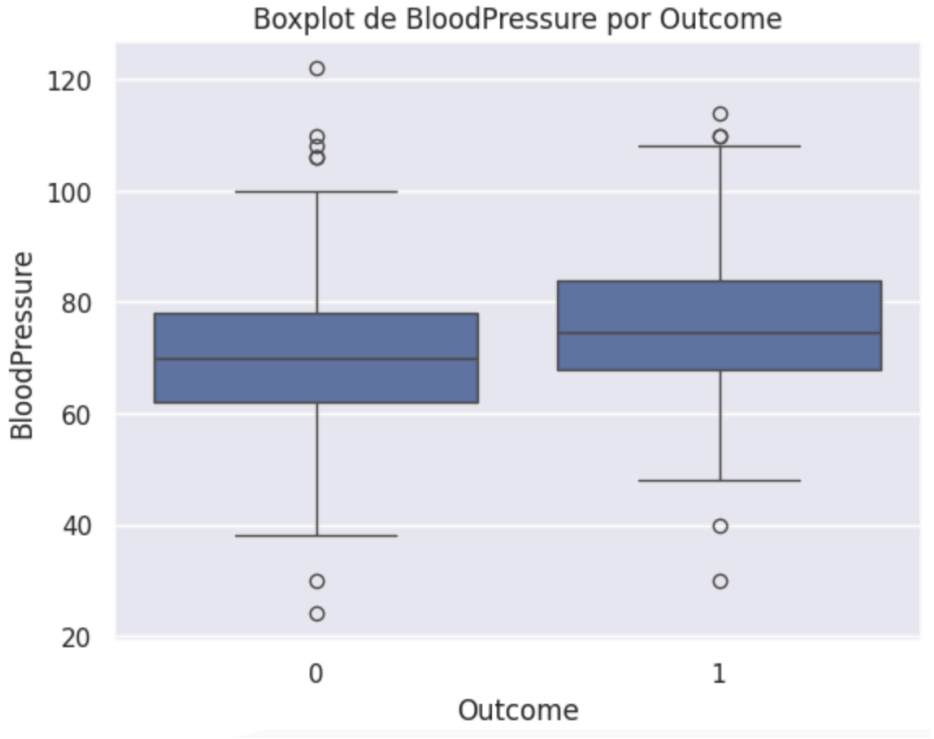
2. **Distribuição da Variável Alvo**: Contagem de casos de diabetes vs. não-diabetes.

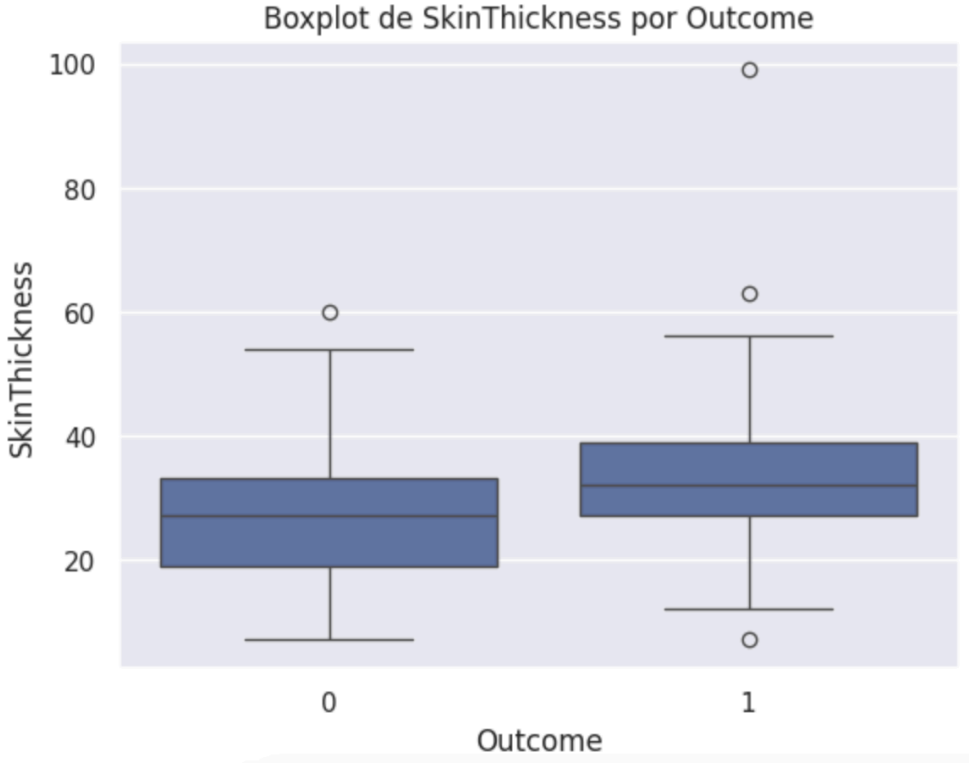


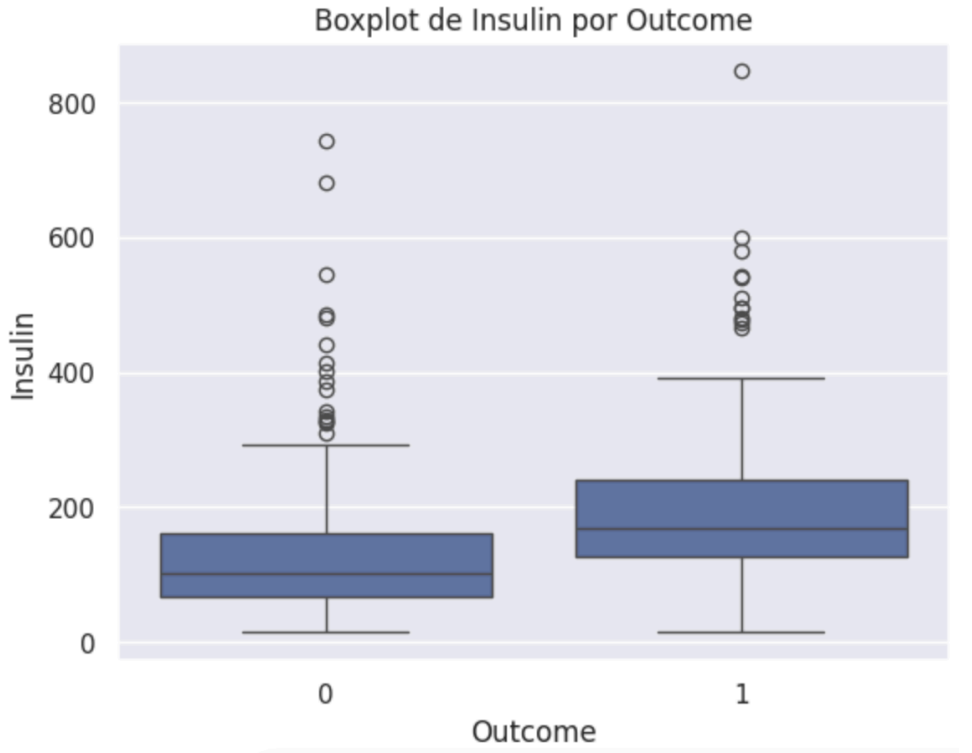
3. **Boxplots** das variáveis por `Outcome` para verificar possíveis diferenças entre grupos.

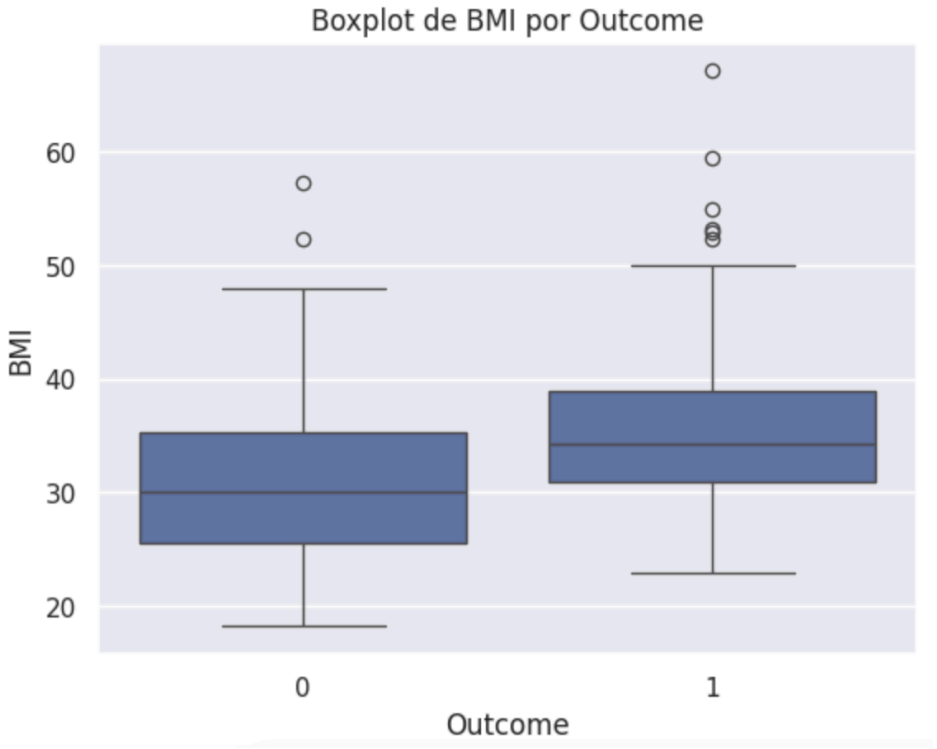


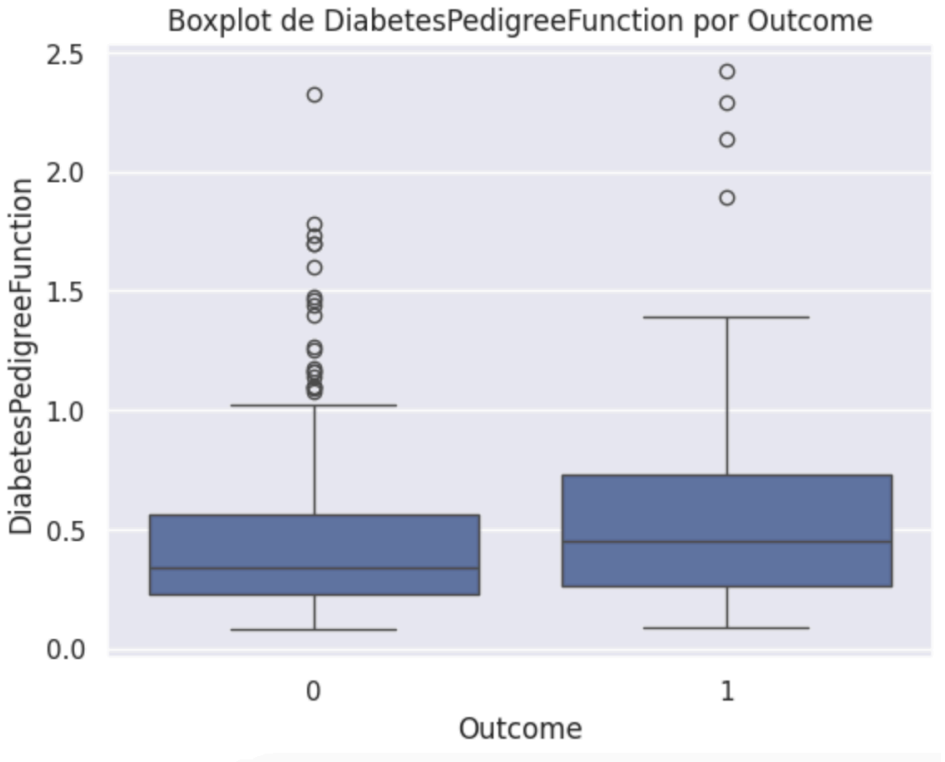

Boxplot de Glucose por Outcome
Glucose
200
180
160
140
120
100
80
60
40
0
1
Outcome

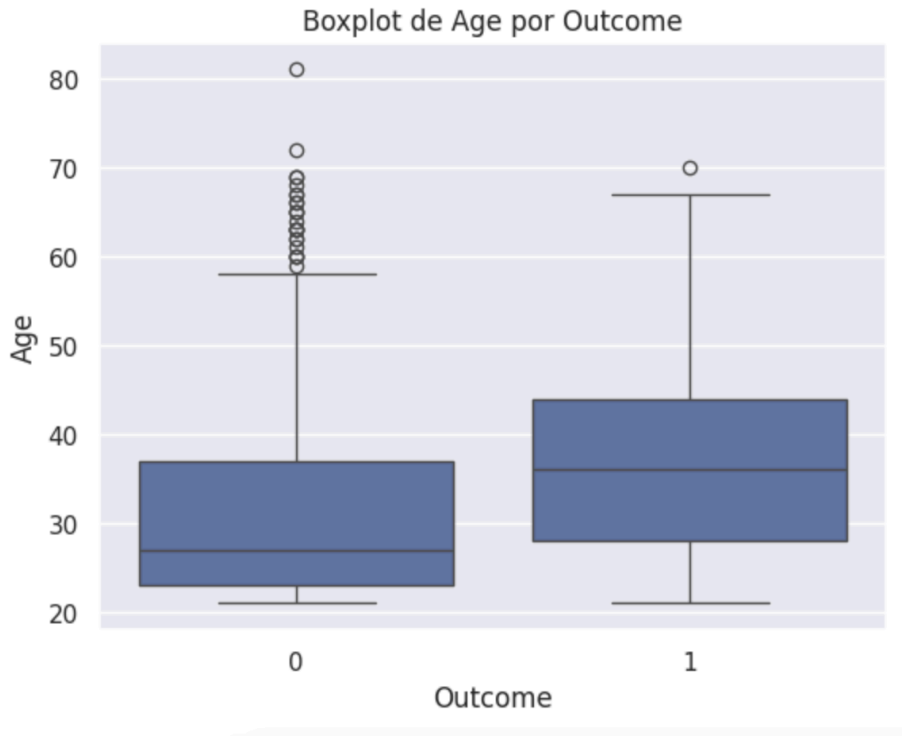

Boxplot de BloodPressure por Outcome
BloodPressure
120
100
80
60
40
20
0
1
Outcome


Boxplot de SkinThickness por Outcome
SkinThickness
100
80
60
40
20
0
1
Outcome

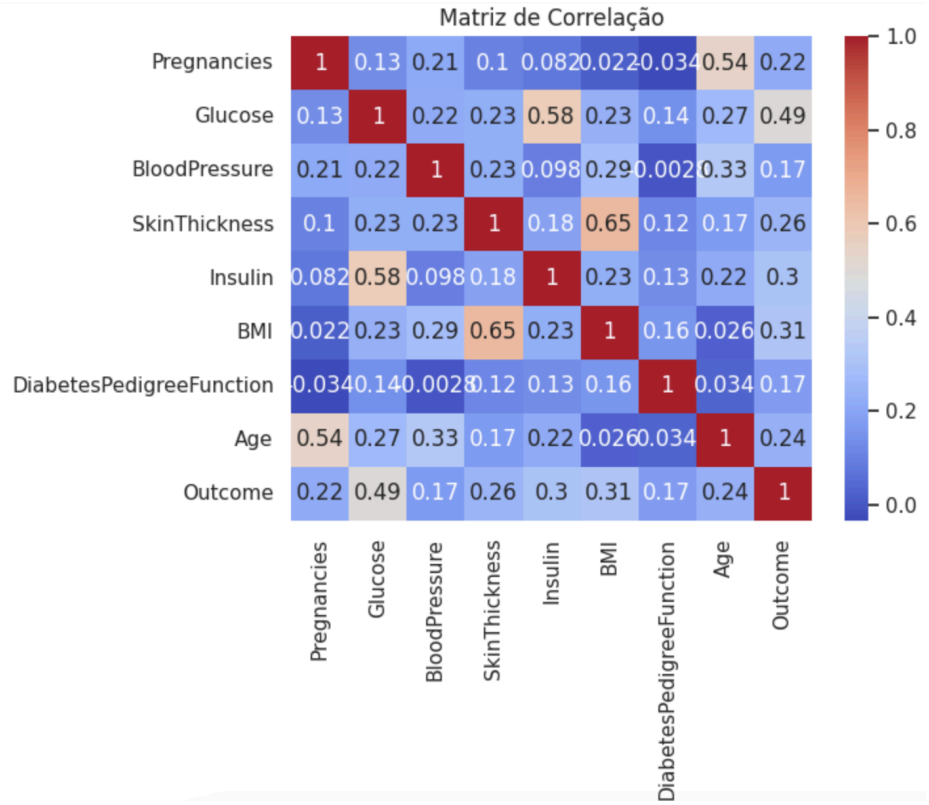

Boxplot de Insulin por Outcome
Insulin
800
600
400
200
0
0
1
Outcome


Boxplot de BMI por Outcome
BMI
60
50
40
30
20
0
1
Outcome


Boxplot de DiabetesPedigreeFunction por Outcome
DiabetesPedigreeFunction
2.5
2.0
1.5
1.0
0.5
0.0
0
1
Outcome



4. **Matriz de Correlação** para identificar correlações entre as variáveis preditoras.



2.4 Pré-Processamento

- **Tratamento de Valores Ausentes**: Valores zero em colunas onde não faz sentido (Glucose, BloodPressure, etc.) foram convertidos em `NaN`, e em seguida foi utilizada a **imputação** pela média (`SimpleImputer(strategy='mean')`).
- **Divisão em Conjuntos**: O dataset foi dividido em:
    - 70% para Treino/Validação (posteriormente dividido em 50%/50% para separar treino e validação).
    - 30% para Teste.
- **Padronização**: Foi utilizado o `StandardScaler()` para normalizar os dados, garantindo que todas as variáveis fiquem em escala comparável.

```
Contagem de valores ausentes (NaN):
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

---

3. Modelo MLP

3.1 Arquitetura da Rede

A rede MLP foi implementada em **Keras** (TensorFlow), com a seguinte configuração:

- **Camada de Entrada**: 8 neurônios de entrada (uma para cada feature).
- **Camada Oculta**: 1 camada com 12 neurônios, função de ativação **ReLU**.
- **Camada de Saída**: 1 neurônio, função de ativação **Sigmoid** (por se tratar de classificação binária).

3.2 Hiperparâmetros e Treinamento

- **Função de Perda**: `binary_crossentropy`.
- **Otimização**: `Adam`.
- **Métrica**: `accuracy`.
- **Batch Size**: 32.
- **Número de Épocas**: até 100 (com `EarlyStopping` para interromper caso não haja melhora na validação por 10 épocas).

3.3 Técnicas de Otimização

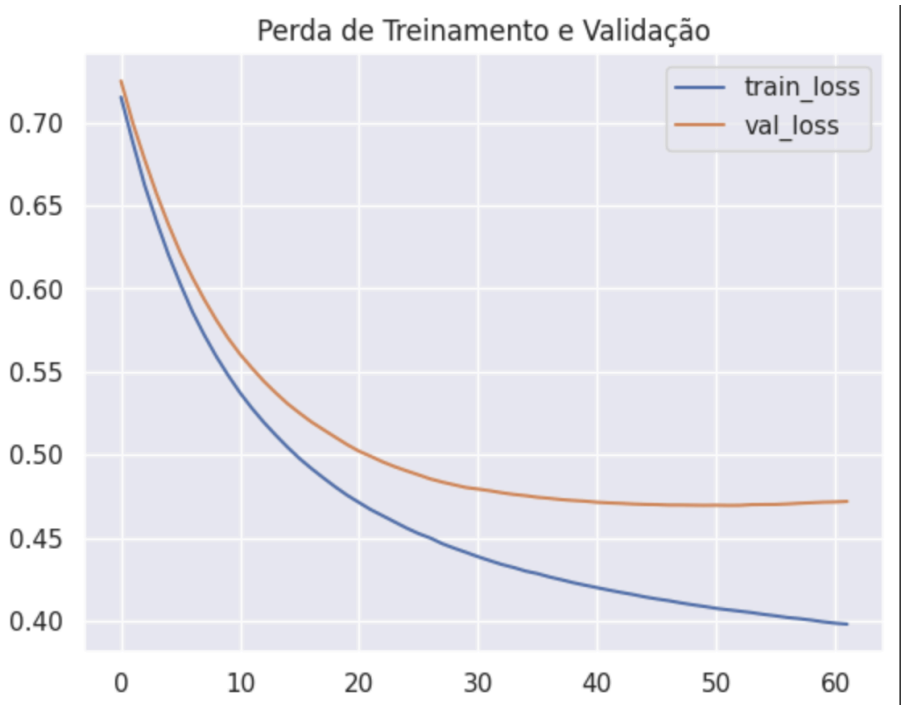
- **EarlyStopping**: Para evitar overfitting, foi configurado um monitoramento da perda de validação (`val_loss`) com paciência de 10 épocas.

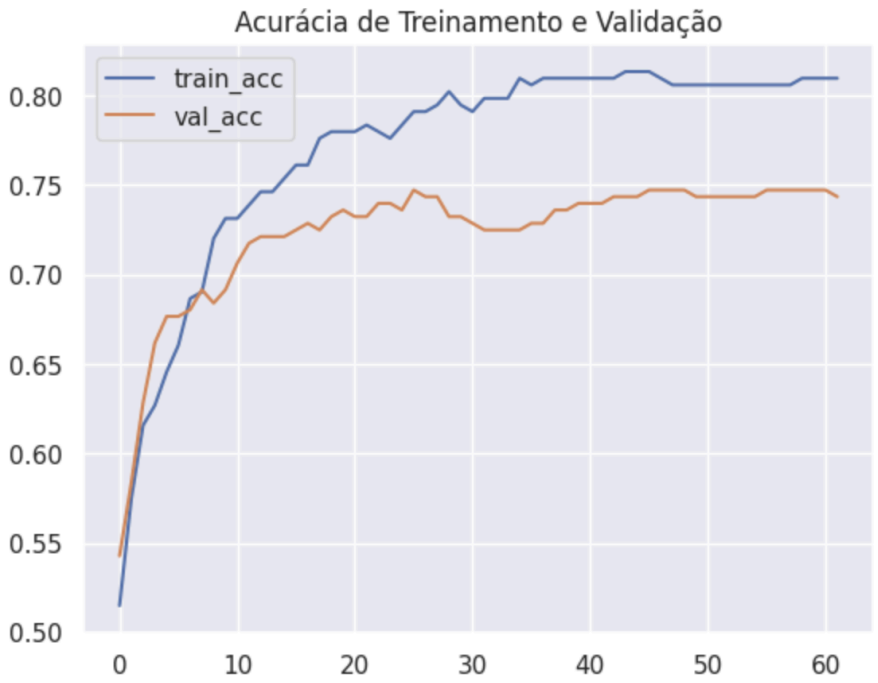
```
[ ]  # Bloco 7: Construção e Treinamento do Modelo MLP

     model = Sequential()
     # Camada de entrada + primeira camada oculta (12 neurônios, ativação ReLU)
     model.add(Dense(12, input_dim=X_train_scaled.shape[1], activation='relu'))
     # Camada de saída (1 neurônio, ativação Sigmoid)
     model.add(Dense(1, activation='sigmoid'))

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

     # EarlyStopping para evitar overfitting
     early_stopping = EarlyStopping(monitor='val_loss', patience=10)

     # Treinando o modelo
     history = model.fit(
         X_train_scaled, y_train,
         validation_data=(X_val_scaled, y_val),
         epochs=100,
         batch_size=32,
         callbacks=[early_stopping]
     )
```

4. Avaliação do Modelo

4.1 Curvas de Treinamento

Durante o treinamento, foram registradas as curvas de **perda** (loss) e **acurácia** (accuracy) tanto para o conjunto de treino quanto para o de validação.

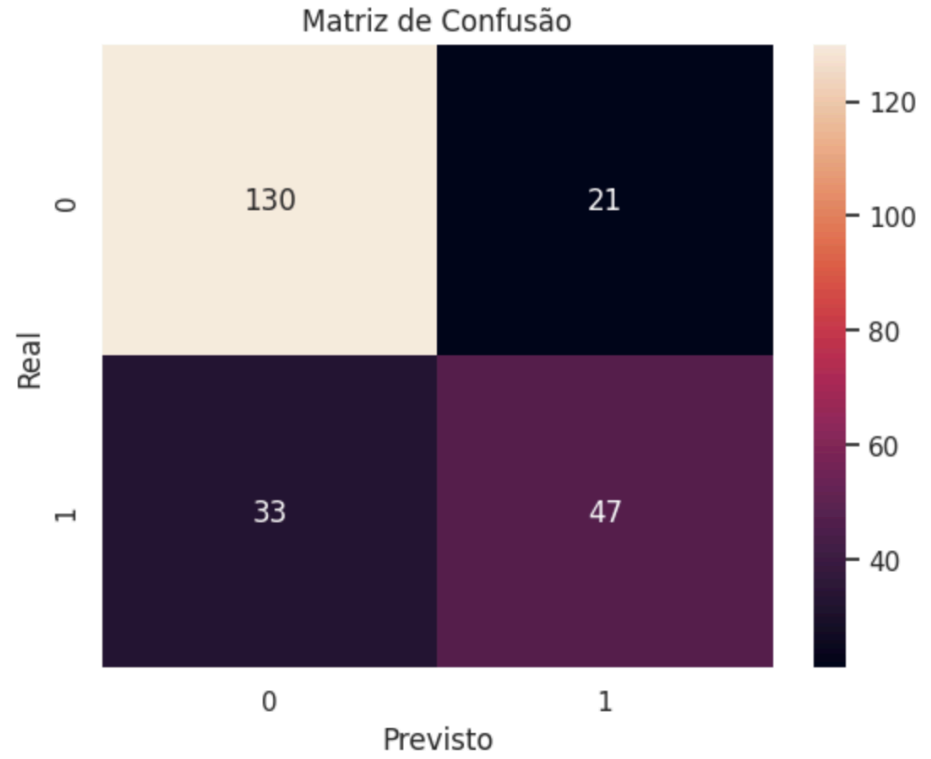




4.2 Resultados no Conjunto de Teste

No conjunto de teste, o modelo apresentou os seguintes resultados:

- **Loss** (perda)
- **Acurácia**
- **Matriz de Confusão**
- **Relatório de Classificação** (precision, recall, F1-score)



	precision	recall	f1-score	support
0	0.80	0.86	0.83	151
1	0.69	0.59	0.64	80
accuracy			0.77	231
macro avg	0.74	0.72	0.73	231
weighted avg	0.76	0.77	0.76	231

4.3 Análise dos Resultados

A **acurácia** obtida no conjunto de teste foi de aproximadamente **77%**, conforme observado no relatório de classificação (Classification Report) e na matriz de confusão. Analisando esses resultados de forma mais detalhada:

- **Matriz de Confusão**:
    - Classe 0 (não diabéticos): 130 acertos (verdadeiros negativos) e 21 erros (falsos positivos).
    - Classe 1 (diabéticos): 47 acertos (verdadeiros positivos) e 33 erros (falsos negativos).
    - Isso significa que o modelo, embora acerte mais frequentemente quando prediz “não diabético”, ainda tem uma quantidade razoável de falsos negativos (33 casos), o que pode ser preocupante em um contexto médico (perder pacientes que efetivamente têm diabetes).
- **Métricas de Precisão e Recall**:
    - Para a classe 0 (não diabéticos):
        - **Precisão**: 0.80
        - **Recall**: 0.86
        - **F1-score**: 0.83
    - Para a classe 1 (diabéticos):
        - **Precisão**: 0.69
        - **Recall**: 0.59
        - **F1-score**: 0.64
    - Observa-se que o modelo tem desempenho **melhor** em identificar a classe 0 do que a classe 1. Em termos de recall (sensibilidade), apenas 59% dos casos positivos (com diabetes) foram corretamente identificados. Em aplicações médicas, isso pode levar a subdiagnóstico, pois 41% dos diabéticos não foram detectados.

- **Equilíbrio entre Precision e Recall**:
    - Para a classe 1, a precisão é 0.69 e o recall é 0.59, indicando que o modelo, quando prediz positivo, acerta 69% das vezes, mas ainda deixa passar muitos casos positivos (recall de 59%). Portanto, **o equilíbrio entre precisão e recall para a classe de interesse (diabéticos) é mediano**.
- **Diferença Treino vs. Validação**:
    - Nos gráficos de perda (loss) e acurácia, nota-se que a rede obteve uma acurácia de treino superior (chegando perto de 80-84%) e uma acurácia de validação em torno de 74-75% ao final. Essa diferença é relativamente **pequena**, sugerindo algum nível de overfitting, mas não exagerado.
    - Ainda assim, a curva de validação se manteve razoavelmente estável, o que indica que o uso do **EarlyStopping** ajudou a interromper o treinamento antes de um overfitting muito acentuado.
- **Principais Variáveis**:
    - Pela análise exploratória (matriz de correlação e boxplots), pode-se inferir que **Glucose** e **BMI** são variáveis com forte correlação com o desfecho. Idade e número de gravidezes também aparecem como relevantes.
    - Não foi realizada uma análise formal de importância de variáveis (como `feature_importance_` ou SHAP), mas, em geral, no Pima Indians Diabetes Database, **Glucose** costuma ser o principal preditor.

Resumo:

- O modelo apresenta uma **acurácia geral de 77%**, com bom desempenho em identificar pacientes sem diabetes, porém com recall moderado na detecção de casos positivos (59%).
- Em termos de aplicação médica, seria interessante melhorar o recall para reduzir os falsos negativos, mesmo que isso custe alguns falsos positivos.
- Melhorias como **mais camadas, regularização, ou otimização de hiperparâmetros** podem ser testadas para aumentar a sensibilidade do modelo a casos positivos.

5. Desenvolvimento da Aplicação

Para permitir a interação com o modelo, desenvolvemos uma **interface web** utilizando a biblioteca **Streamlit**. O arquivo `app.py` carrega o modelo treinado (`diabetes_mlp.h5`), bem como o `imputer.joblib` e `scaler.joblib`. Em seguida, exibe campos de entrada para o usuário inserir valores de **Gravidezes, Glucose, Pressão Arterial**, etc. Ao clicar no botão “Prever”, a aplicação:

1. Cria um DataFrame com os valores inseridos.
2. Substitui zeros por `NaN`, imputa valores ausentes e aplica o escalonamento (o mesmo usado no treino).
3. Faz a predição com o modelo MLP, retornando se o paciente tem ou não diabetes.

5.1 Tecnologias Utilizadas

- **Python 3.XX**
- **Streamlit** para a interface web.
- **TensorFlow/Keras** para o modelo MLP.
- **Scikit-learn** para pré-processamento e métricas.
- **Joblib** para salvar e carregar objetos (imputer, scaler).

6. Conclusão

Este projeto demonstrou a aplicação de uma **rede neural MLP** para a classificação de diabetes. Foi possível observar:

- A importância do pré-processamento (tratamento de zeros, imputação de dados e normalização).
- A viabilidade da arquitetura simples (1 camada oculta) para resolver o problema, alcançando uma acurácia satisfatória.
- A facilidade de integrar o modelo em uma interface web (Streamlit), permitindo que qualquer usuário teste a aplicação.

6.1 Possíveis Melhorias Futuras

- Experimentar **mais camadas** e neurônios na rede MLP, para verificar se o desempenho melhora.

- Incluir **técnicas de regularização** (como dropout ou batch normalization) para evitar overfitting.

- Explorar **hiperparâmetros** (ex.: taxa de aprendizado, número de épocas, batch size) com ferramentas como Grid Search ou Random Search.

- Coletar **mais dados** ou dados mais recentes para aumentar a generalização do modelo.