In [ ]:
```
!rm -rf 2023*
```

```
'rm' n□o □ reconhecido como um comando interno
ou externo, um programa oper□vel ou um arquivo em lotes.
```

In [ ]:
```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "1" # "1" para cpu e "0" PARA GPU
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from sklearn import preprocessing
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.backend import clear_session
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
    roc_curve,
    auc,
    RocCurveDisplay
)
from sklearn.linear_model import LogisticRegression

from pathlib import Path
import os.path
from datetime import datetime
```

In [ ]:
```
class MLPClassifierWrapper:
    """
    Keras-based MLP (multi layer percepton) classifier
    """

    def __init__(self, **kwargs):

        # Initialize model

        self.n_neurons = kwargs.get('n_neurons')
        self.input_dim = kwargs.get('input_dim')
        self.output_dim = kwargs.get('output_dim')
        self.verbose = kwargs.get('verbose', 0)
        self.hidden_neurons_activation_function = kwargs.get('hidden_neurons_act
        self.output_neurons_activation_function= kwargs.get('output_neurons_acti
        self.loss_function = kwargs.get('loss_function', 'binary_crossentropy')
        self.model = self.build_model()

        # Train parameters
        self.epochs = kwargs.get('epochs', 100)
```

```
        # Validation checkpoint
        # get the current working directory
        current_working_directory = Path.cwd()

        # print output to the console

        self.model_checkpoint_file = os.path.join(
            current_working_directory,
            datetime.now().strftime("%Y%m%d%H%M%S") + '.h5'
        )



    def build_model(self):
        """
        Build MLP Model
        """

        model = Sequential()

        hidden_layer = Dense(self.n_neurons,
                             input_dim=self.input_dim,
                             activation=self.hidden_neurons_activation_function)
        model.add(hidden_layer)

        output_layer = Dense(self.output_dim,
                             activation=self.output_neurons_activation_function)
        model.add(output_layer)

        model.compile(
            optimizer=optimizers.SGD(learning_rate=0.1),
            loss=self.loss_function,
            metrics=["accuracy"]
        )

        return model


    def fit(self, X_train, y_train, X_val, y_val):
        """
        Fit model using train and validation data
        """
        y_train = to_categorical(y_train)
        y_val = to_categorical(y_val)

        checkpoint = ModelCheckpoint(self.model_checkpoint_file,
                                     monitor='val_loss',
                                     verbose=self.verbose,
                                     save_best_only=True,
                                     mode='min')

        history = self.model.fit(X_train,
                                 y_train,
                                 verbose=self.verbose,
                                 validation_data=(X_val, y_val),
                                 callbacks=[checkpoint],
                                 epochs=self.epochs)
        self.history = history
```

```
    def predict(self, X):
        """
        """
        y_pred = self.predict_proba(X)
        return np.argmax(y_pred, axis=1)

    def predict_proba(self, X):
        """
        """
        self.model.load_weights(self.model_checkpoint_file)
        y_pred = self.model.predict(X, verbose=0)
        return y_pred


# class LogisticRegressionWrapper:

#     def __init__(self, **kwargs):
#         """
#         Initialize model
#         """
#         self.model = LogisticRegression()

#     def fit(self, X_train, y_train, X_val, y_val):
#         """
#         fit linear model
#         """
#         self.model.fit(X_train, y_train)

#     def predict(self, X):
#         """
#         Predict
#         """
#         return self.model.predict(X)



def train(X, y,
          model_klass,
          n_splits=5,
          n_init=1,
          **kwargs):
    """
    Cross-validation train using keras

    This function was adapted from previous experiments using
    sklearn algorithms.
    """
    cv = StratifiedKFold(n_splits=n_splits,shuffle = True, random_state = 42)
    f1_score_val_list          = []
    f1_score_train_list        = []
    accuracy_score_val_list    = []
    accuracy_score_train_list  = []
    recall_score_val_list      = []
    recall_score_train_list    = []
    precision_score_val_list   = []
    precision_score_train_list = []
    model_list  = []
    scaler_list = []
    # fig, ax = plt.subplots(1, 1, figsize=(8, 8))
```

```
# fprs_list = []
# tprs_list = []
# auc_list  = []



# Validação cruzada só em Training Data
for fold, (train_idx, val_idx) in enumerate(cv.split(X, y)):
    X_train = X[train_idx, :]
    y_train = y[train_idx]
    X_val = X[val_idx, :]
    y_val = y[val_idx]

    # Escala
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    scaler_list.append(scaler)

    # Treino
    model = None
    f1_score_val = 0.
    for idx in range(n_init):
        _model = model_klass(**kwargs)
        _model.fit(X_train_scaled, y_train, X_val_scaled, y_val)
        _y_pred = _model.predict(X_train_scaled)

        _y_pred_val = _model.predict(X_val_scaled)
        _f1_score_val = f1_score(y_val, _y_pred_val, average='weighted')
        if _f1_score_val > f1_score_val:
            y_pred_val = _y_pred_val
            y_pred = _y_pred
            model = _model
        clear_session()
    f1_score_train = f1_score(y_train, y_pred, average='weighted')
    f1_score_val = f1_score(y_val, y_pred_val, average='weighted')


    print(f"======================== FOLD {fold} ==========================
    print(f"Meu resultado para treino de F1-Score é {f1_score(y_train, y_pre
    print(f"Meu resultado para treino de  Acurácia é de {accuracy_score(y_tr
    print(f"Meu resultado para treino de  Recall é de {recall_score(y_train,
    print(f"Meu resultado para treino de  Precision é de {precision_score(y_
    f1_score_val_list.append(f1_score(y_val, y_pred_val))
    f1_score_train_list.append(f1_score(y_train, y_pred))
    accuracy_score_val_list.append(accuracy_score(y_val, y_pred_val))
    accuracy_score_train_list.append(accuracy_score(y_train, y_pred))
    recall_score_val_list.append(recall_score(y_val, y_pred_val))
    recall_score_train_list.append(recall_score(y_train, y_pred))
    precision_score_val_list.append(precision_score(y_val, y_pred_val))
    precision_score_train_list.append(precision_score(y_train, y_pred))
    model_list.append(model)
    # y_hat_val = model.predict_proba(X_val_scaled)
    # viz = RocCurveDisplay.from_predictions(
    # model,
    # X_val_scaled,
    # y_val,
    # ax = ax,
    # alpha=0.3,
    # lw=1
```

```
        # )
        # interp_fpr, interp_tpr = interpolation(viz.fpr, viz.tpr)
        # fprs_list.append(interp_fpr)
        # tprs_list.append(interp_tpr)
        # auc_list.append(viz.roc_auc)

    print()
    print()
    mean_val = np.mean(f1_score_val_list)
    std_val  = np.std(f1_score_val_list)
    print(f"Meu resultado de F1-Score Médio de treino é {np.mean(f1_score_train_
    print(f"Meu resultado de accuracy_score Médio de treino é {np.mean(accuracy_
    print(f"Meu resultado de recall_score Médio de treino é {np.mean(recall_scor
    print(f"Meu resultado de precision_score Médio de treino é {np.mean(precisic


    best_model_idx = np.argmax(f1_score_val_list)
    print(f"Meu melhor fold é: {best_model_idx} ")
    best_model = model_list[best_model_idx]
    best_scaler = scaler_list[best_model_idx]
    return best_model, mean_val, std_val, best_scaler
```

Load Dataset

Attribute Information:

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

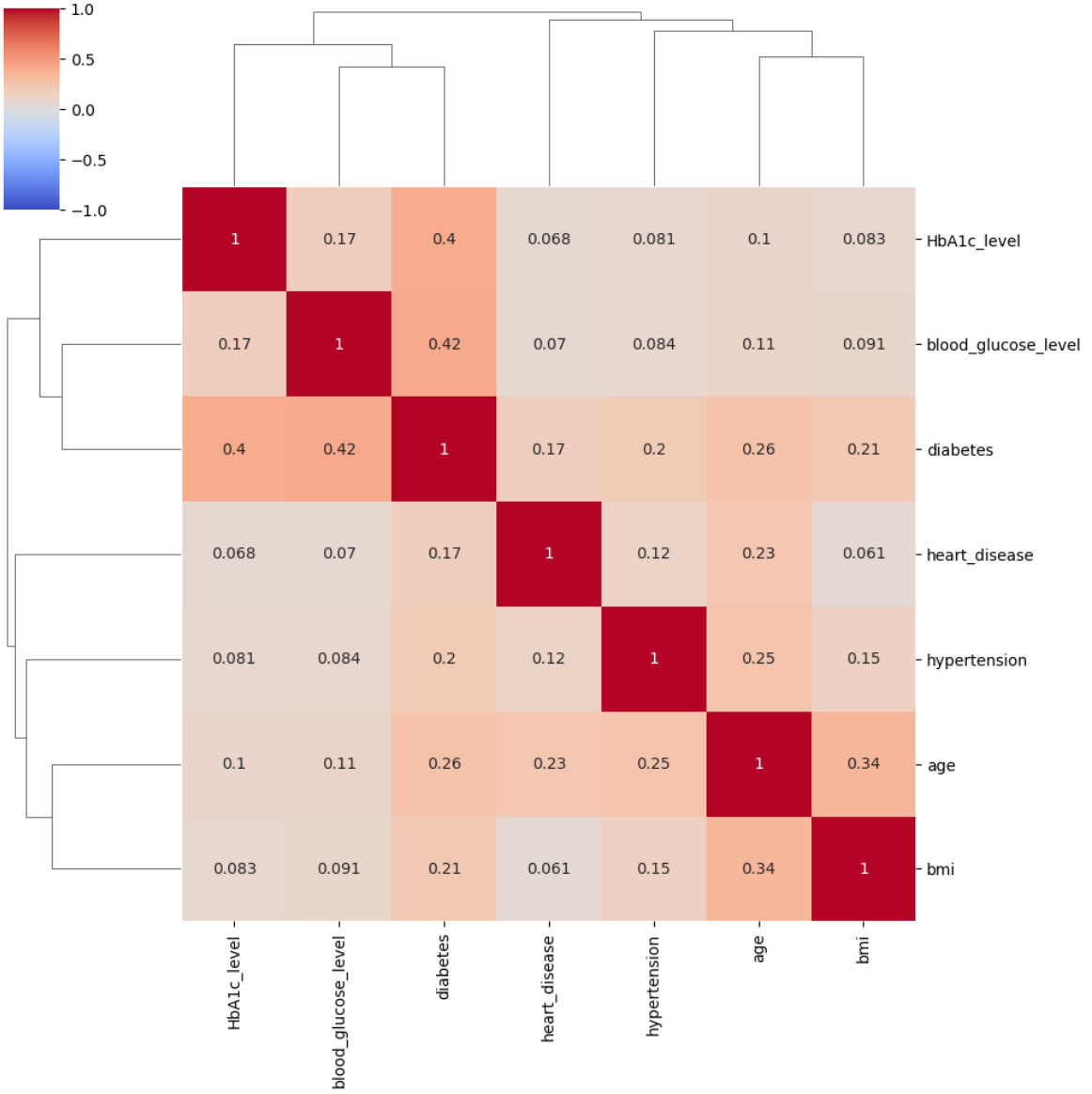
In [ ]:
```
patients = pd.read_csv("diabetes_prediction_dataset.csv")
```

In [ ]:
```
patients = patients.dropna()
```

In [ ]:
```
sns.clustermap(patients.corr(), cmap="coolwarm", vmin=-1, vmax=1, annot=True)
```

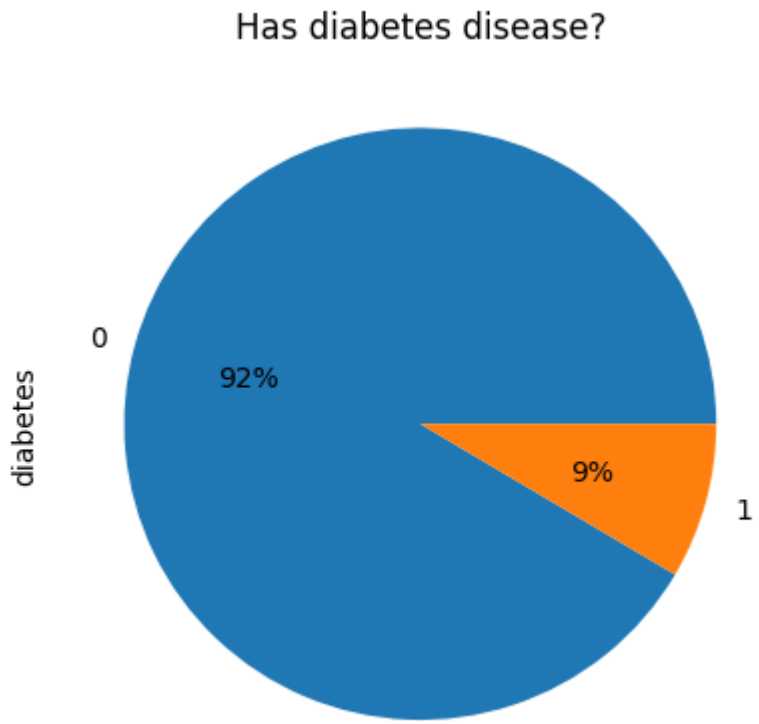
```
C:\Users\Diones\AppData\Local\Temp\ipykernel_6820\415701235.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ve
rsion, it will default to False. Select only valid columns or specify the value o
f numeric_only to silence this warning.
  sns.clustermap(patients.corr(), cmap="coolwarm", vmin=-1, vmax=1, annot=True)
```

Out[ ]:
```
<seaborn.matrix.ClusterGrid at 0x1f4f700be80>
```



In [ ]:
```
ax = patients.diabetes.value_counts().plot(kind='pie',  autopct='%1.0f%%')

ax.set_title("Has diabetes disease?")
```

Out[ ]:
```
Text(0.5, 1.0, 'Has diabetes disease?')
```



In [ ]:
```
X = patients.drop(columns=["diabetes"])
y = patients.diabetes
```

In [ ]:
```
X
```

Out[ ]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level
0	Female	80.0	0	1	never	25.19	6.6
1	Female	54.0	0	0	No Info	27.32	6.6
2	Male	28.0	0	0	never	27.32	5.7
3	Female	36.0	0	0	current	23.45	5.0
4	Male	76.0	1	1	current	20.14	4.8
...	...	...	...	...	...	...	...
99995	Female	80.0	0	0	No Info	27.32	6.2
99996	Female	2.0	0	0	No Info	17.37	6.5
99997	Male	66.0	0	0	former	27.83	5.7
99998	Female	24.0	0	0	never	35.42	4.0
99999	Female	57.0	0	0	current	22.43	6.6

100000 rows × 8 columns

In [ ]:
```
y
```

Out[ ]:
```
0        0
1        0
2        0
3        0
4        0
        ..
99995    0
99996    0
99997    0
99998    0
99999    0
Name: diabetes, Length: 100000, dtype: int64
```

In [ ]:
```
# pd.plotting.scatter_matrix(
#     X[['age', 'gender', 'hypertension', 'heart_disease', 'smoking_history','bm
#     alpha=0.2, diagonal='kde',  figsize=(14, 10));
```

var transform

In [ ]:
```
onehot_heart = preprocessing.OneHotEncoder(sparse_output=False, drop='if_binary'
X['heart_disease'] = onehot_heart.transform(X.heart_disease.values.reshape(-1, 1

onehot_hyper = preprocessing.OneHotEncoder(sparse_output=False, drop='if_binary'
X['hypertension'] = onehot_hyper.transform(X.hypertension.values.reshape(-1, 1))


onehot_gender = preprocessing.OneHotEncoder(sparse_output=False, drop='if_binary
X['gender'] = onehot_gender.transform(X.gender.values.reshape(-1, 1))
enc = preprocessing.OrdinalEncoder(categories=[['No Info', 'never', 'not current

X['smoking_history'] = enc.transform(X.smoking_history.values.reshape(-1, 1))
```

In [ ]:
```
# pd.plotting.scatter_matrix(
#     X[['age', 'gender', 'hypertension', 'heart_disease', 'smoking_history','bm
#     alpha=0.2, diagonal='kde',  figsize=(14, 10));
```

In [ ]:
```
X
```

Out[ ]:

	gender	age	hypertension	heart_disease	smoking_history	bmi	HbA1c_level
0	1.0	80.0	0.0	1.0	1.0	25.19	6.6
1	1.0	54.0	0.0	0.0	0.0	27.32	6.6
2	0.0	28.0	0.0	0.0	1.0	27.32	5.7
3	1.0	36.0	0.0	0.0	3.0	23.45	5.0
4	0.0	76.0	1.0	1.0	3.0	20.14	4.8
...	...	...	...	...	...	...	...
99995	1.0	80.0	0.0	0.0	0.0	27.32	6.2
99996	1.0	2.0	0.0	0.0	0.0	17.37	6.5
99997	0.0	66.0	0.0	0.0	5.0	27.83	5.7
99998	1.0	24.0	0.0	0.0	1.0	35.42	4.0
99999	1.0	57.0	0.0	0.0	3.0	22.43	6.6

100000 rows × 8 columns

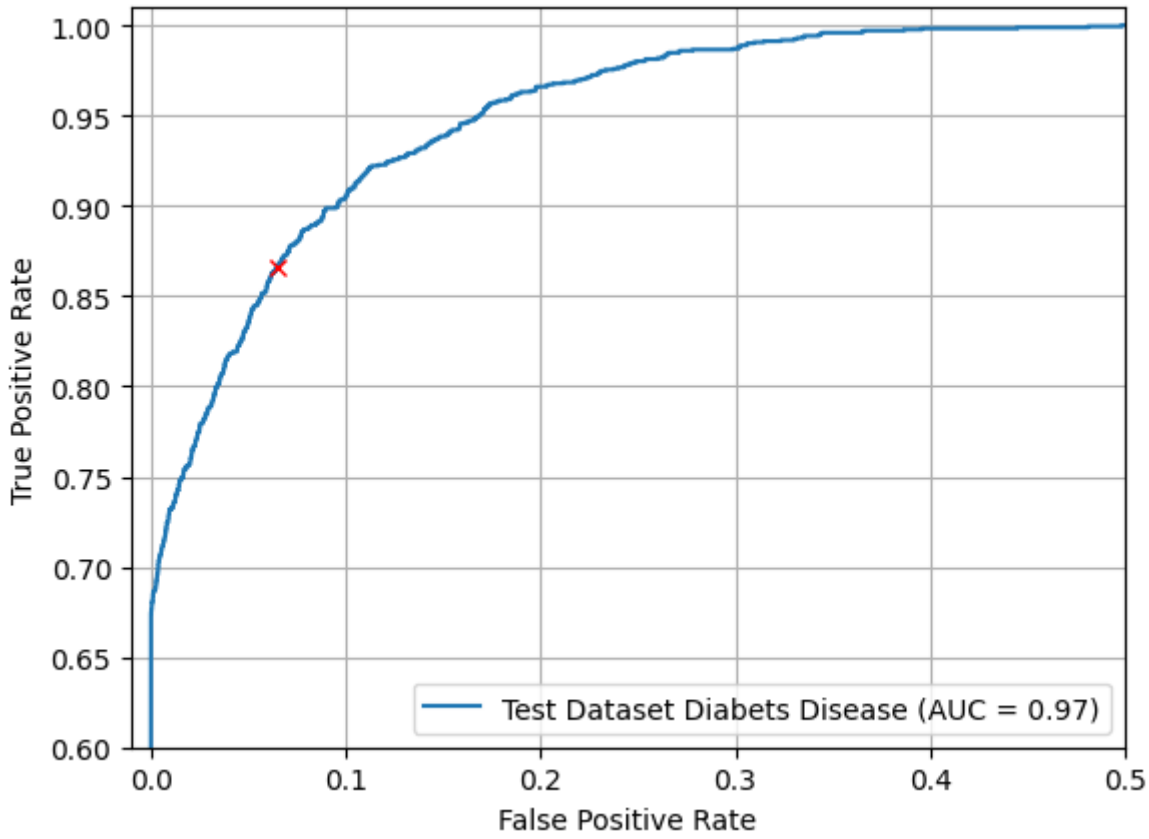
In [ ]:
```
from sklearn.model_selection import (
    train_test_split,
    StratifiedKFold)
from sklearn.preprocessing import StandardScaler



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_s
```

In [ ]:
```
%%time

nn_settings = {
    'n_neurons': 8,
    'input_dim': X_train.shape[1],
    'output_dim': 2,
    'hidden_neurons_activation_function': 'relu',
    'output_neurons_activation_function': 'softmax',
    'epochs': 75,
    'loss_function': 'categorical_crossentropy',
    'verbose': 0
}


nn_best_model, nn_mean_val, std_val, best_scaler = train(X_train.values,
                                                         y_train.values,
                                                         MLPClassifierWrapper,
                                                         n_splits=5,
                                                         n_init=5,
                                                         **nn_settings)
```

```
======================== FOLD 0 ========================
Meu resultado para treino de F1-Score é 0.77, Meu resultado para validação de F1-
Score é 0.77
Meu resultado para treino de  Acurácia é de 0.97, Meu resultado para validação de
Acurácia é de 0.97
Meu resultado para treino de  Recall é de 0.65, Meu resultado para validação de R
ecall é de 0.65
Meu resultado para treino de  Precision é de 0.94, Meu resultado para validação d
e Precision é de 0.94
======================== FOLD 1 ========================
Meu resultado para treino de F1-Score é 0.77, Meu resultado para validação de F1-
Score é 0.77
Meu resultado para treino de  Acurácia é de 0.97, Meu resultado para validação de
Acurácia é de 0.97
Meu resultado para treino de  Recall é de 0.65, Meu resultado para validação de R
ecall é de 0.65
Meu resultado para treino de  Precision é de 0.94, Meu resultado para validação d
e Precision é de 0.96
======================== FOLD 2 ========================
Meu resultado para treino de F1-Score é 0.8, Meu resultado para validação de F1-S
core é 0.81
Meu resultado para treino de  Acurácia é de 0.97, Meu resultado para validação de
Acurácia é de 0.97
Meu resultado para treino de  Recall é de 0.69, Meu resultado para validação de R
ecall é de 0.7
Meu resultado para treino de  Precision é de 0.95, Meu resultado para validação d
e Precision é de 0.96
======================== FOLD 3 ========================
Meu resultado para treino de F1-Score é 0.73, Meu resultado para validação de F1-
Score é 0.72
Meu resultado para treino de  Acurácia é de 0.96, Meu resultado para validação de
Acurácia é de 0.96
Meu resultado para treino de  Recall é de 0.63, Meu resultado para validação de R
ecall é de 0.62
Meu resultado para treino de  Precision é de 0.87, Meu resultado para validação d
e Precision é de 0.86
======================== FOLD 4 ========================
Meu resultado para treino de F1-Score é 0.77, Meu resultado para validação de F1-
Score é 0.77
Meu resultado para treino de  Acurácia é de 0.97, Meu resultado para validação de
Acurácia é de 0.97
Meu resultado para treino de  Recall é de 0.64, Meu resultado para validação de R
ecall é de 0.65
Meu resultado para treino de  Precision é de 0.96, Meu resultado para validação d
e Precision é de 0.95


Meu resultado de F1-Score Médio de treino é  0.77 +-  0.02, Meu resultado de F1-S
core Médio de validação é  0.77 +-  0.027
Meu resultado de accuracy_score Médio de treino é  0.97 +-  0.003, Meu resultado
de accuracy_score Médio de validação é  0.97 +-  0.004
Meu resultado de recall_score Médio de treino é  0.65 +-  0.018. Meu resultado de
recall_score Médio de validação é  0.65 +-  0.026
Meu resultado de precision_score Médio de treino é  0.93 +-  0.031, Meu resultado
de precision_score Médio de validação é  0.93 +-  0.035
Meu melhor fold é: 2
CPU times: total: 1h 12min 35s
Wall time: 52min 21s
```

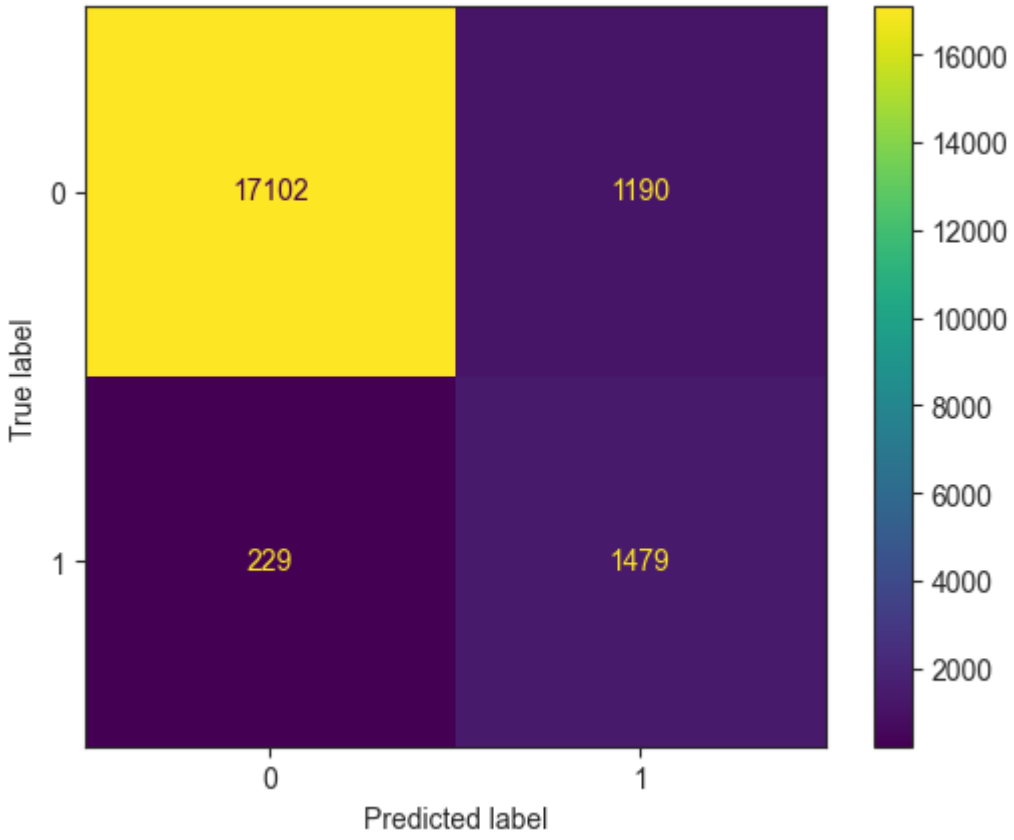
In [ ]:
```
def plot_roc(model, X, y, estimator_name = "treino", **kwargs):
    y_hat = model.predict_proba(X)[:, 1]
    fpr, tpr, thresholds = roc_curve(y, y_hat, pos_label=1)
    auc_score = auc(fpr, tpr)
    roccurve= RocCurveDisplay(fpr=fpr,
                              tpr=tpr,
                              roc_auc=auc_score,
                              estimator_name = estimator_name).plot(**kwargs)
    return roccurve

X_test_scaled = best_scaler.transform(X_test)
X_train_scaled = best_scaler.transform(X_train)
roccurve = plot_roc(nn_best_model, X_test_scaled, y_test.values,
              estimator_name="Test Dataset Diabets Disease")
roccurve.ax_.grid()
roccurve.ax_.set_ylim([.6, 1.01])
roccurve.ax_.set_xlim([-0.01,.5])


threshold = .1

y_hat = nn_best_model.predict_proba(X_test_scaled)[:, 1]
y_pred = (y_hat > threshold).astype(int)

cm = confusion_matrix(y_test, y_pred)
sensitivity = cm[1, 1] / (cm[1, 1] + cm[1 , 0])
specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
roccurve.ax_.plot(1 - specificity, sensitivity, 'x', c='red')

print(specificity)
```

```
e:\miniconda3\envs\Bootcamp\lib\site-packages\sklearn\base.py:432: UserWarning: X
has feature names, but StandardScaler was fitted without feature names
  warnings.warn(
e:\miniconda3\envs\Bootcamp\lib\site-packages\sklearn\base.py:432: UserWarning: X
has feature names, but StandardScaler was fitted without feature names
  warnings.warn(
0.9349442379182156
```



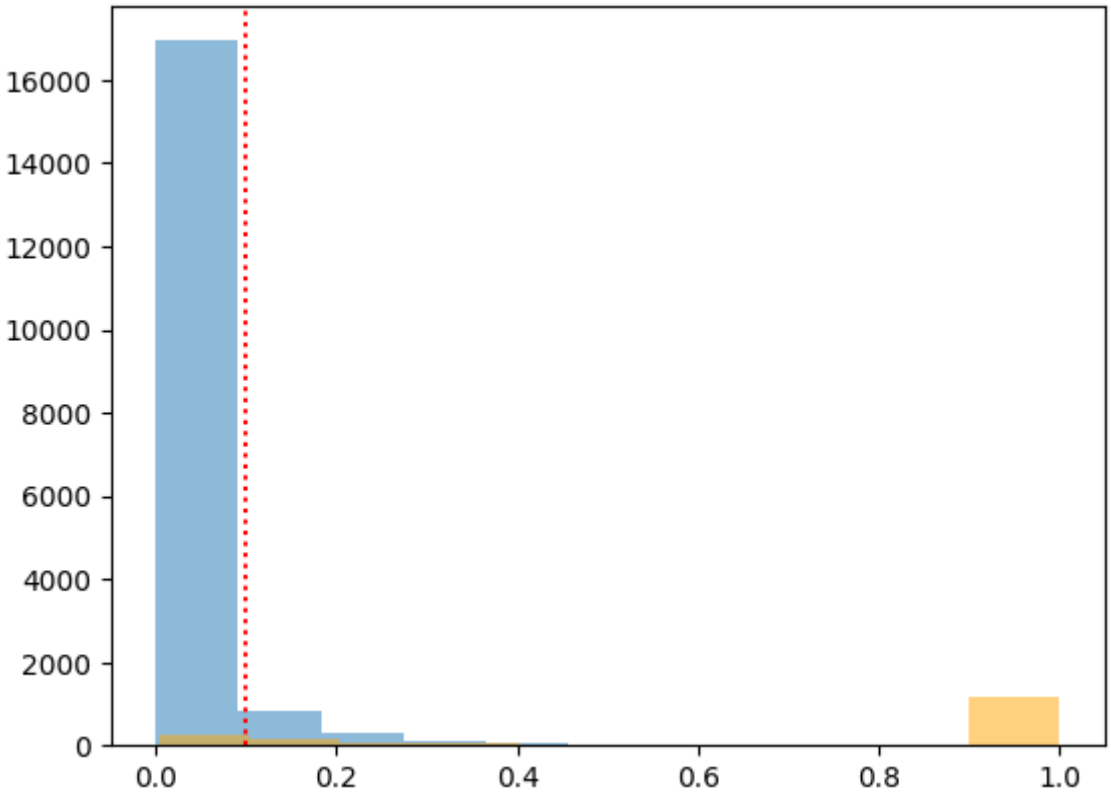
In [ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

matrix_confusion_display = ConfusionMatrixDisplay(cm)
matrix_confusion_display.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f4ad9100a0
>
```



In [ ]:
```
plt.hist(y_hat[y_test == 0], alpha=.5)
plt.hist(y_hat[y_test == 1], alpha=.5, color='orange')
plt.axvline(threshold, color='red', ls=":")
```

Out[ ]: <matplotlib.lines.Line2D at 0x1f4f9ee3940>



In [ ]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_hat)
classifier_auc = auc(fpr, tpr)
print(f"AUC: {classifier_auc:.2f}")
```

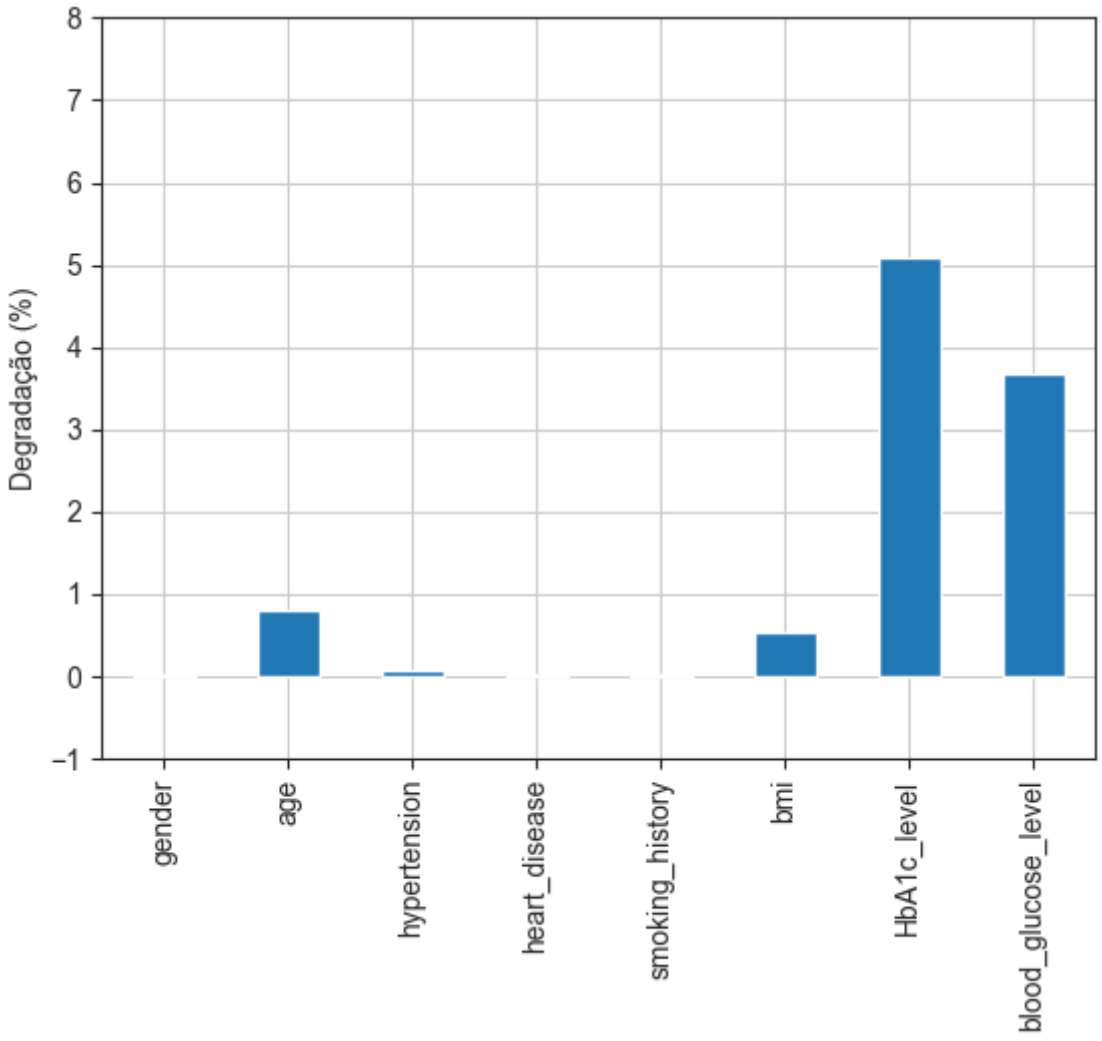
AUC: 0.97

In [ ]:
```
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

mean_values = X_test_scaled.mean()

aucs_rate = {}
for var in mean_values.index:
    X_study = X_test_scaled.copy()
    X_study[var] = mean_values[var]
    y_hat = nn_best_model.predict_proba(X_study)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test, y_hat)
    auc_ = auc(fpr, tpr)
    aucs_rate[var] = 100* (1 - (auc(fpr, tpr) / classifier_auc))

    print(f"AUC retirando a variável {var}: {auc_:.2f}")
    print(f"\t > Degradação: {aucs_rate[var]:.1f} %")
```

```
AUC retirando a variável gender: 0.97
        > Degradação: 0.0 %
AUC retirando a variável age: 0.97
        > Degradação: 0.8 %
AUC retirando a variável hypertension: 0.97
        > Degradação: 0.1 %
AUC retirando a variável heart_disease: 0.97
        > Degradação: 0.0 %
AUC retirando a variável smoking_history: 0.97
        > Degradação: 0.0 %
AUC retirando a variável bmi: 0.97
        > Degradação: 0.5 %
AUC retirando a variável HbA1c_level: 0.92
        > Degradação: 5.1 %
AUC retirando a variável blood_glucose_level: 0.94
        > Degradação: 3.7 %
```

In [ ]:
```
sns.set_style('ticks')

ax = pd.Series(aucs_rate).plot(kind='bar')
ax.grid()
ax.set_ylabel("Degradação (%)")
ax.set_ylim(-1, 8)
#sns.despine(offset=5)
```

Out[ ]: (-1.0, 8.0)



In [ ]: