Pima Indian Diabetes

RIYADH SILVA

Sobre o dataset

- Pima Indians Diabetes Data Set.
- Esse conjunto de dados foi fornecido pelo centro de pesquisa da universidade de The Johns Hopkins University nos EUA em 1990.
- Contém dados reais de pessoas que foram diagnosticadas com diabetes.
- Utilizado em tarefas de classificação.

Sobre o dataset

Data Set Characteristics:	Multivariate	**Number of Instances:**	768	**Area:**	Life
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	8	**Date Donated**	1990-05-09
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	377946

Sobre o dataset

- Número de vezes grávida
- Concentração de glicose plasmática a 2 horas em um teste oral de tolerância à glicose
- Pressão sanguínea diastólica (mm Hg)
- Espessura da dobra da pele do tríceps (mm)
- Insulina sérica de 2 horas (mu U / ml)
- Índice de massa corporal (peso em kg / (altura em m) ^ 2)
- Função de predisposição de diabetes
- Idade (anos)
- Variável de classe (false ou true)

Sobre o dataset

- Objetivo:
  - Prever se uma pessoa irá desenvolver diabetes
- Resultados esperados após os algoritmos:
  - Verdadeiro ou falso
- Utilidade:
  - Ajudar as políticas públicas de saúde, como ajudar as industrias farmacêuticas a preverem a busca por medicamentos.

Distribuição dos dados

```
In [3]: # Verificando a organização dos dados
        df.shape

Out[3]: (768, 10)
```

Verificando as 5 primeiras linhas

In [4]:
```
# Verificando as primeiras linhas do dataset
df.head(5)
```

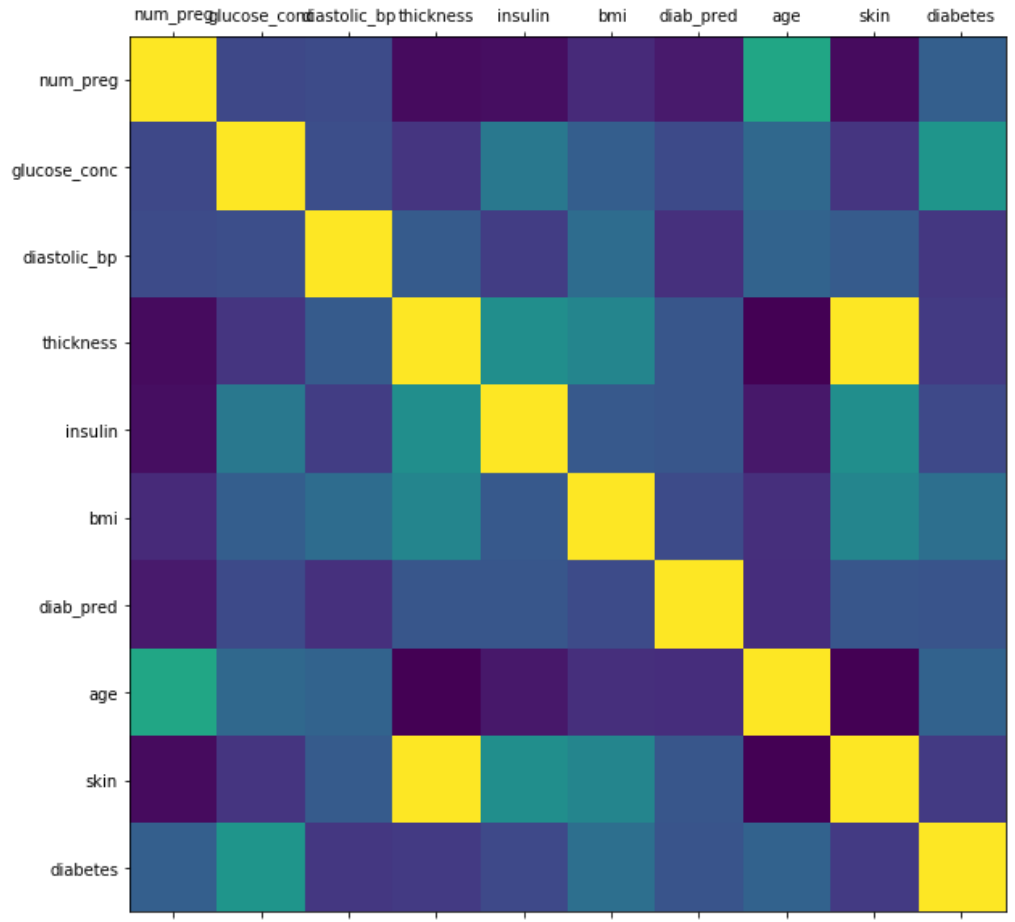
Out[4]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	True
1	1	85	66	29	0	26.6	0.351	31	1.1426	False
2	8	183	64	0	0	23.3	0.672	32	0.0000	True
3	1	89	66	23	94	28.1	0.167	21	0.9062	False
4	0	137	40	35	168	43.1	2.288	33	1.3790	True

Verificando se existem valores nulos

```
In [6]: # Verificando se existem valores nulos
        df.isnull().values.any()

Out[6]: False
```

Gráfico de correlação entre as variáveis



Conversão de tipo de valores

In [10]:
```
# Definindo as classes (transformar o que é true em 1 e o que é false em 0)
diabetes_map = {True : 1, False : 0}
```

Mapeando o dataset

In [11]:
```
# Aplicando o mapeamento ao dataset
df['diabetes'] = df['diabetes'].map(diabetes_map)
```

Verificando o dataset

In [12]:
```
# Verificando as primeiras linhas do dataset
df.head(5)
```

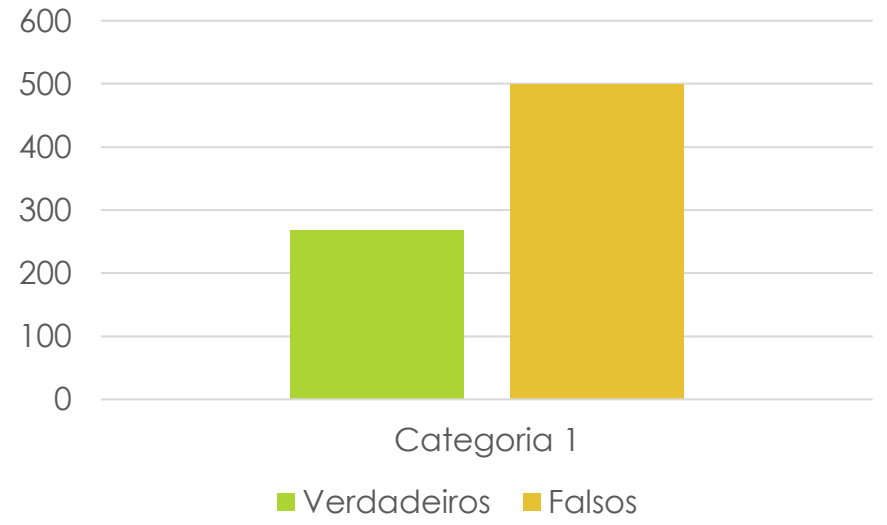
Out[12]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	1
1	1	85	66	29	0	26.6	0.351	31	1.1426	0
2	8	183	64	0	0	23.3	0.672	32	0.0000	1
3	1	89	66	23	94	28.1	0.167	21	0.9062	0
4	0	137	40	35	168	43.1	2.288	33	1.3790	1

Verificando como os dados estão distribuídos

In [13]:
```
# Verificando como os dados estão distribuídos
num_true = len(df.loc[df['diabetes'] == True])
num_false = len(df.loc[df['diabetes'] == False])
print("Número de Casos Verdadeiros: {0} ({1:2.2f}%)".format(num_true, (num_true/ (num_true + num_false)) * 100))
print("Número de Casos Falsos     : {0} ({1:2.2f}%)".format(num_false, (num_false/ (num_true + num_false)) * 100))
```

```
Número de Casos Verdadeiros: 268 (34.90%)
Número de Casos Falsos     : 500 (65.10%)
```



1° Split – 70% treino e 30% teste

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
# Seleção de variáveis
atributos = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
```

In [16]:
```
# Variável a ser prevista
atrib_prev = ['diabetes']
```

In [17]:
```
# Criando objetos
X = df[atributos].values
Y = df[atrib_prev].values
```

In [18]:
```
# Definindo a taxa de split
split_test_size = 0.30
```

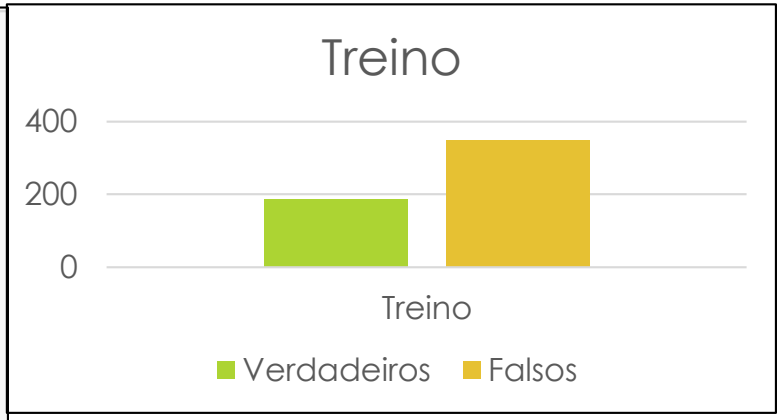
In [19]:
```
# Criando dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = split_test_size, random_state = 42)
```

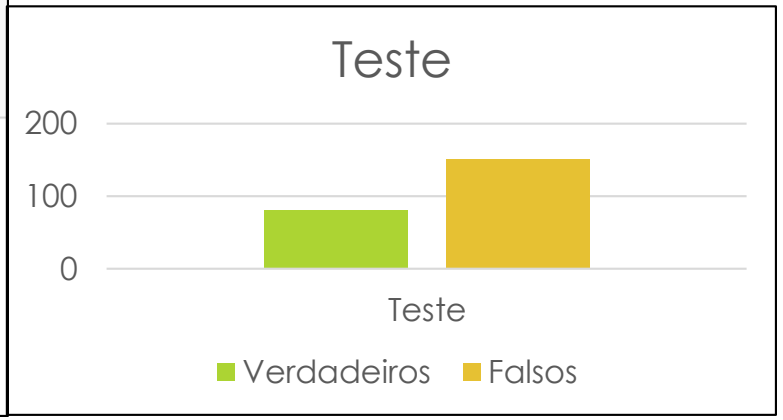
In [20]:
```
# Imprimindo os resultados
print("{0:0.2f}% nos dados de treino".format((len(X_treino)/len(df.index)) * 100))
print("{0:0.2f}% nos dados de teste".format((len(X_teste)/len(df.index)) * 100))
```

```
69.92% nos dados de treino
30.08% nos dados de teste
```

1° Split – Verificando o split

```
In [21]: print("Original True : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]),
                                                        (len(df.loc[df['diabetes'] ==1])/len(df.index) * 100)))

         print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]),
                                                         (len(df.loc[df['diabetes'] == 0])/len(df.index) * 100)))
         print("")
         print("Training True : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 1]),
                                                        (len(Y_treino[Y_treino[:] == 1])/len(Y_treino) * 100)))

         print("Training False : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 0]),
                                                         (len(Y_treino[Y_treino[:] == 0])/len(Y_treino) * 100)))
         print("")
         print("Test True : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 1]),
                                                    (len(Y_teste[Y_teste[:] == 1])/len(Y_teste) * 100)))

         print("Test False : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 0]),
                                                     (len(Y_teste[Y_teste[:] == 0])/len(Y_teste) * 100)))

Original True : 268 (34.90%)
Original False : 500 (65.10%)

Training True : 188 (35.01%)
Training False : 349 (64.99%)

Test True : 80 (34.63%)
Test False : 151 (65.37%)
```





Valores ocultos

In [22]:
```
df.head(5)
```

Out[22]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	1
1	1	85	66	29	0	26.6	0.351	31	1.1426	0
2	8	183	64	0	0	23.3	0.672	32	0.0000	1
3	1	89	66	23	94	28.1	0.167	21	0.9062	0
4	0	137	40	35	168	43.1	2.288	33	1.3790	1

In [23]:
```
print("# Linhas no dataframe {0}".format(len(df)))
print("# Linhas missing glucose_conc: {0}".format(len(df.loc[df['glucose_conc'] == 0])))
print("# Linhas missing diastolic_bp: {0}".format(len(df.loc[df['diastolic_bp'] == 0])))
print("# Linhas missing thickness: {0}".format(len(df.loc[df['thickness'] == 0])))
print("# Linhas missing insulin: {0}".format(len(df.loc[df['insulin'] == 0])))
print("# Linhas missing bmi: {0}".format(len(df.loc[df['bmi'] == 0])))
print("# Linhas missing age: {0}".format(len(df.loc[df['age'] == 0])))
```

```
# Linhas no dataframe 768
# Linhas missing glucose_conc: 5
# Linhas missing diastolic_bp: 35
# Linhas missing thickness: 227
# Linhas missing insulin: 374
# Linhas missing bmi: 11
# Linhas missing age: 0
```

Tratando valores ocultos - Impute

Substituindo os valores zeros pela média dos dados

[23]:
```
from sklearn.preprocessing import Imputer
```

[24]:
```
# Criando objeto
preenche_0 = Imputer(missing_values = 0, strategy = "mean", axis = 0)

# Substituindo os valores iguais a zero, pela média dos dados
X_treino = preenche_0.fit_transform(X_treino)
X_teste = preenche_0.fit_transform(X_teste)
```

Tratando valores ocultos - Impute

Substituindo os valores zeros pela média dos dados

[23]:
```
from sklearn.preprocessing import Imputer
```

[24]:
```
# Criando objeto
preenche_0 = Imputer(missing_values = 0, strategy = "mean", axis = 0)

# Substituindo os valores iguais a zero, pela média dos dados
X_treino = preenche_0.fit_transform(X_treino)
X_teste = preenche_0.fit_transform(X_teste)
```

Construindo e treinando o 1° modelo

In [25]:
```
# Utilizando um classificador Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

In [26]:
```
# Criando o modelo preditivo
modelo_v1 = GaussianNB()
```

In [27]:
```
# Treinando o modelo
modelo_v1.fit(X_treino, Y_treino.ravel())
```

Out[27]:
```
GaussianNB(priors=None)
```

Verificando a exatidão no modelo dos dados do treino

In [28]:
```
from sklearn import metrics
```

In [29]:
```
nb_predict_train = modelo_v1.predict(X_treino)
print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_treino, nb_predict_train)))
print()
```

```
Exatidão (Accuracy): 0.7542
```

Verificando a exatidão no modelo dos dados do teste

[30]:
```
nb_predict_test = modelo_v1.predict(X_teste)
print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_teste, nb_predict_test)))
print()
```

```
Exatidão (Accuracy): 0.7359
```

2° modelo – Random Florest

```
In [31]: from sklearn.ensemble import RandomForestClassifier
```

```
In [32]: modelo_v2 = RandomForestClassifier(random_state = 42)
         modelo_v2.fit(X_treino, Y_treino.ravel())
```

```
Out[32]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=42, verbose=0, warm_start=False)
```

```
In [33]: # Verificando os dados de treino
         rf_predict_train = modelo_v2.predict(X_treino)
         print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_treino, rf_predict_train)))
```

```
Exatidão (Accuracy): 0.9870
```

```
In [34]: # Verificando nos dados de teste
         rf_predict_test = modelo_v2.predict(X_teste)
         print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_teste, rf_predict_test)))
         print()
```

```
Exatidão (Accuracy): 0.7100
```

3º modelo – Regressão logística

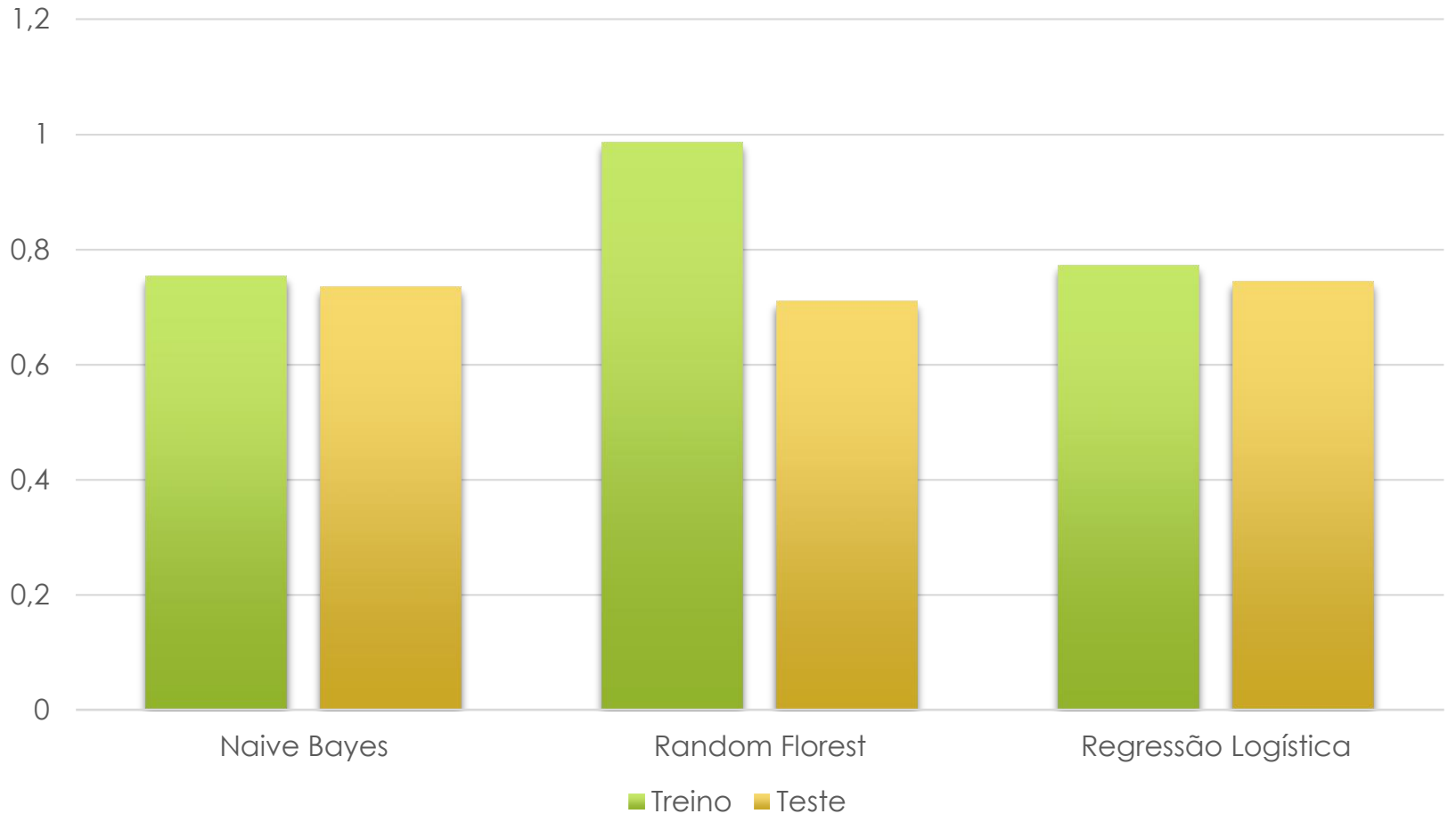
In [35]:
```
## Regressão logística
from sklearn.linear_model import LogisticRegression
```

In [37]:
```
# Terceira versão do modelo usando Regressão Logística
modelo_v3 = LogisticRegression(C = 0.7, random_state = 42)
modelo_v3.fit(X_treino, Y_treino.ravel())
lr_predict_train = modelo_v3.predict(X_treino)
lr_predict_test = modelo_v3.predict(X_teste)
```

In [38]:
```
print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_treino, lr_predict_train)))
print()
print("Exatidão (Accuracy): {0:.4f}".format(metrics.accuracy_score(Y_teste, lr_predict_test)))
print()
```

```
Exatidão (Accuracy): 0.7728

Exatidão (Accuracy): 0.7446
```

Resultados - 1° Split (70 – 30)



2° Split (80 – 20)

In [39]:
```
from sklearn.model_selection import train_test_split
```

In [40]:
```
# Seleção de variáveis
atributos = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
```

In [41]:
```
# Variável a ser prevista
atrib_prev = ['diabetes']
```

In [42]:
```
# Criando objetos
X = df[atributos].values
Y = df[atrib_prev].values
```

In [43]:
```
# Definindo a taxa de split
split_test_size = 0.20
```

In [44]:
```
# Criando dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = split_test_size, random_state = 42)
```

In [45]:
```
# Imprimindo os resultados
print("{0:0.2f}% nos dados de treino".format((len(X_treino)/len(df.index)) * 100))
print("{0:0.2f}% nos dados de teste".format((len(X_teste)/len(df.index)) * 100))
```

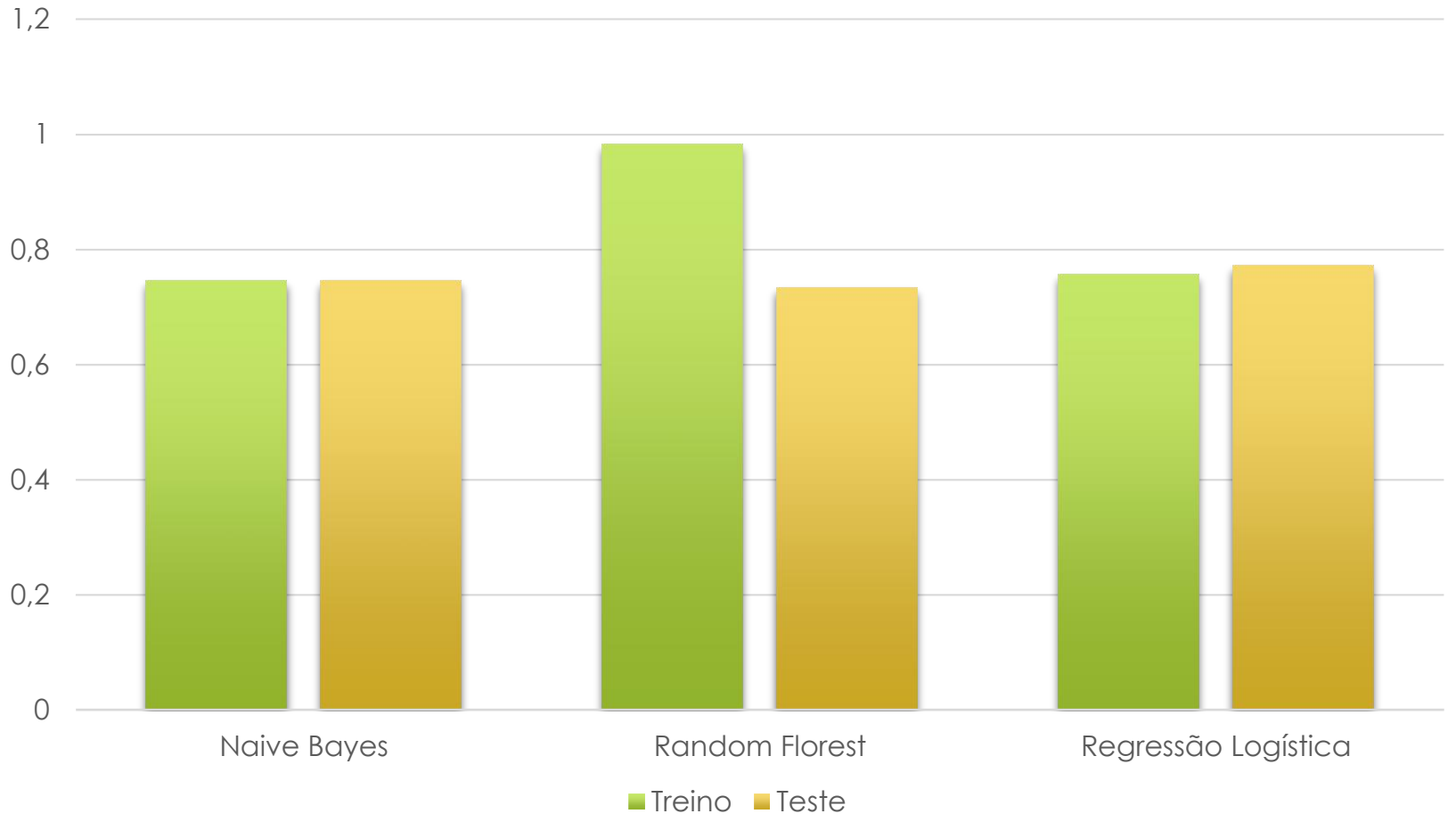
```
79.95% nos dados de treino
20.05% nos dados de teste
```

2º Split – verificando o Split

In [46]:
```
print("Original True : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]),
                                               (len(df.loc[df['diabetes'] ==1])/len(df.index) * 100)))

print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]),
                                               (len(df.loc[df['diabetes'] == 0])/len(df.index) * 100)))
print("")
print("Training True : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 1]),
                                               (len(Y_treino[Y_treino[:] == 1])/len(Y_treino) * 100)))

print("Training False : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 0]),
                                               (len(Y_treino[Y_treino[:] == 0])/len(Y_treino) * 100)))
print("")
print("Test True : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 1]),
                                           (len(Y_teste[Y_teste[:] == 1])/len(Y_teste) * 100)))

print("Test False : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 0]),
                                           (len(Y_teste[Y_teste[:] == 0])/len(Y_teste) * 100)))
```

```
Original True : 268 (34.90%)
Original False : 500 (65.10%)

Training True : 213 (34.69%)
Training False : 401 (65.31%)

Test True : 55 (35.71%)
Test False : 99 (64.29%)
```

Resultados - 2º Split (80 – 20)



3° Split (60 – 40)

In [68]:
```
from sklearn.model_selection import train_test_split
```

In [69]:
```
# Seleção de variáveis
atributos = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
```

In [70]:
```
# Variável a ser prevista
atrib_prev = ['diabetes']
```

In [71]:
```
# Criando objetos
X = df[atributos].values
Y = df[atrib_prev].values
```

In [72]:
```
# Definindo a taxa de split
split_test_size = 0.40
```

In [73]:
```
# Criando dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = split_test_size, random_state = 42)
```

In [74]:
```
# Imprimindo os resultados
print("{0:0.2f}% nos dados de treino".format((len(X_treino)/len(df.index)) * 100))
print("{0:0.2f}% nos dados de teste".format((len(X_teste)/len(df.index)) * 100))
```

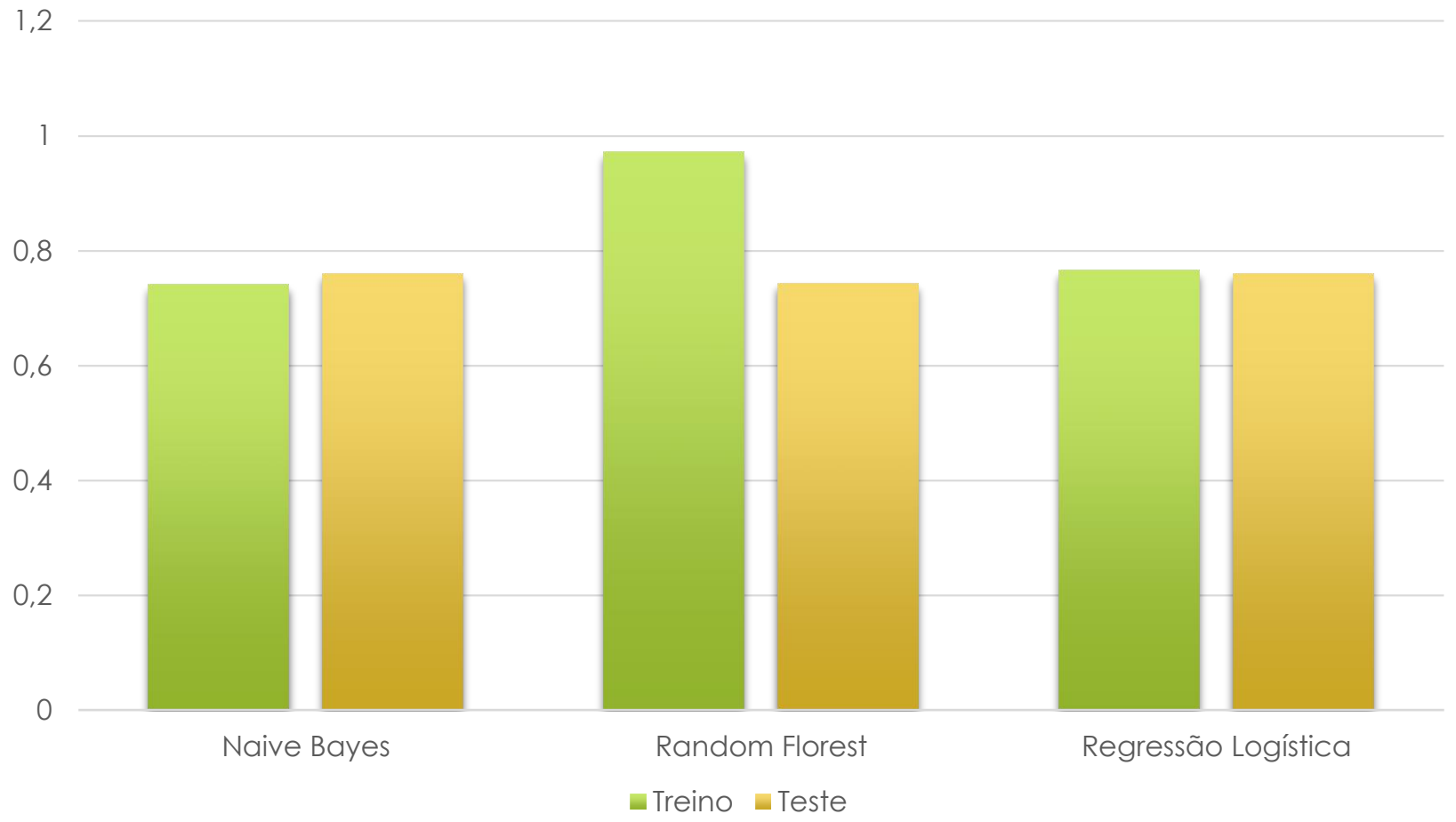
```
59.90% nos dados de treino
40.10% nos dados de teste
```

3° Split – verificando o Split

In [75]:
```
print("Original True : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 1]),
                                               (len(df.loc[df['diabetes'] ==1])/len(df.index) * 100)))

print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['diabetes'] == 0]),
                                               (len(df.loc[df['diabetes'] == 0])/len(df.index) * 100)))
print("")
print("Training True : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 1]),
                                               (len(Y_treino[Y_treino[:] == 1])/len(Y_treino) * 100)))

print("Training False : {0} ({1:0.2f}%)".format(len(Y_treino[Y_treino[:] == 0]),
                                               (len(Y_treino[Y_treino[:] == 0])/len(Y_treino) * 100)))
print("")
print("Test True : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 1]),
                                           (len(Y_teste[Y_teste[:] == 1])/len(Y_teste) * 100)))

print("Test False : {0} ({1:0.2f}%)".format(len(Y_teste[Y_teste[:] == 0]),
                                           (len(Y_teste[Y_teste[:] == 0])/len(Y_teste) * 100)))
```

```
Original True : 268 (34.90%)
Original False : 500 (65.10%)

Training True : 166 (36.09%)
Training False : 294 (63.91%)

Test True : 102 (33.12%)
Test False : 206 (66.88%)
```

Resultados - 3º Split (60 – 40)



Referências

- https://archive.ics.uci.edu/ml/datasets/pima+indians+diabetes
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html