Lista 5 - Random Forest e Naive Bayes

Nome: Gustavo Costa

Questão 01

Considere a seguinte base de dados:

Dia	Aparência	temperatura	Umidade	Ventando	Jogar
d1	Sol	Quente	Alta	Não	Não
d2	Sol	Quente	Alta	Sim	Não
d3	Nublado	Quente	Alta	Não	Sim
d4	Chuva	Agradável	Alta	Não	Sim
d5	Chuva	Fria	Normal	Não	Sim
d6	Chuva	Fria	Normal	Sim	Não
d7	Nublado	Fria	Normal	Sim	Sim
d8	Sol	Agradável	Alta	Não	Não
d9	Sol	Fria	Normal	Não	Sim
d10	Chuva	Agradável	Normal	Não	Sim
d11	Sol	Agradável	Normal	Sim	Sim
d12	Nublado	Agradável	Alta	Sim	Sim
d13	Nublado	Quente	Normal	Não	Sim
d14	Chuva	Agradável	Alta	Sim	Não

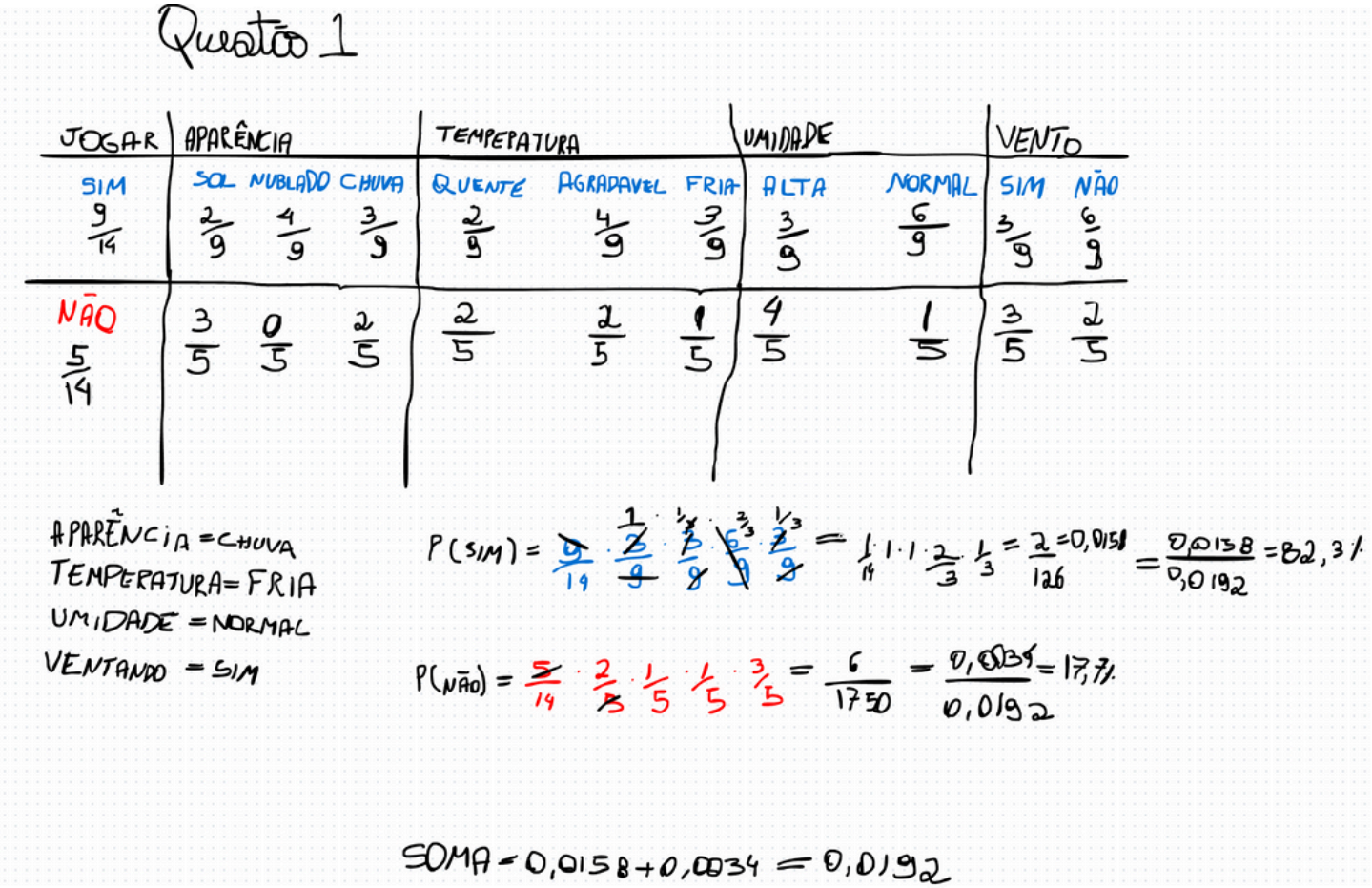
Utilizando o algoritmo de Naive Bayes, qual a probabilidade de Jogar ou não Jogar, respectivamente, para o seguinte registro:

Aparência = Chuva

Temperatura = Fria

Umidade = Normal

Ventando = Sim



Questão 02

Implemente o método de Naive Bayes utilizando o python. Veja a resposta do algoritmo para o registro acima.

```python
# Lendo os dados.
df = pd.read_excel("/content/Dados-lista5-AM.xlsx")
```

```python
df.head()
```

	Aparencia	Temperatura	Umidade	Ventando	Jogar
0	Sol	Quente	Alta	Não	Não
1	Sol	Quente	Alta	Sim	Não
2	Nublado	Quente	Alta	Não	Sim
3	Chuva	Agradavel	Alta	Não	Sim
4	Chuva	Fria	Normal	Não	Sim

Próximas etapas: Gerar código com df | Ver gráficos recomendados | New interactive sheet

A base de dados está em formato de string, primeiro vamos fazer o pré-processameno dela e transformá-la em valores numéricos para o algoritmo de Naive Bayes.

```python
# Uso do OneHotEncoder para a coluna Aparencia que não possui uma ordem.
from sklearn.preprocessing import OneHotEncoder

# Create a OneHotEncoder instance
one_hot = OneHotEncoder(sparse_output=False)

# Aplicando OneHot na Aparencia.
df_aparencia = one_hot.fit_transform(df[["Aparencia"]])
aparencia_feature_names = one_hot.get_feature_names_out(["Aparencia"])

# Criando um DataFrame para Aparencia apos o OneHot.
df_aparencia = pd.DataFrame(df_aparencia, columns=aparencia_feature_names)

# Concatenação do novo dataframe transformado pelo onehotencoder.
df = pd.concat([df, df_aparencia], axis=1)

df.head()
```

```
df.head()
```

	Aparencia	Temperatura	Umidade	Ventando	Jogar	Aparencia_Chuva	Aparencia_Nublado	Aparencia_Sol
0	Sol	Quente	Alta	Não	Não	0.0	0.0	1.0
1	Sol	Quente	Alta	Sim	Não	0.0	0.0	1.0
2	Nublado	Quente	Alta	Não	Sim	0.0	1.0	0.0
3	Chuva	Agradavel	Alta	Não	Sim	1.0	0.0	0.0
4	Chuva	Fria	Normal	Não	Sim	1.0	0.0	0.0

Próximas etapas: Gerar código com df | Ver gráficos recomendados | New interactive sheet

```
[49] # Dropando as colunas temperatura e aparencia original.
     df = df.drop("Aparencia", axis=1)
```

```
[50] df.head()
```

	Temperatura	Umidade	Ventando	Jogar	Aparencia_Chuva	Aparencia_Nublado	Aparencia_Sol
0	Quente	Alta	Não	Não	0.0	0.0	1.0
1	Quente	Alta	Sim	Não	0.0	0.0	1.0
2	Quente	Alta	Não	Sim	0.0	1.0	0.0
3	Agradavel	Alta	Não	Sim	1.0	0.0	0.0
4	Fria	Normal	Não	Sim	1.0	0.0	0.0

```python
# Aplicar LabelEncoder para Umidade, Ventando e Jogar.
from sklearn.preprocessing import LabelEncoder

df["Umidade"] = LabelEncoder().fit_transform(df["Umidade"])
df["Ventando"] = LabelEncoder().fit_transform(df["Ventando"])

# Aplicar manualmente em temperatura para que fique em ordem personalizada.
ordem_temp = {"Fria": 0,"Agradavel": 1,"Quente": 2}
df["Temperatura"] = df["Temperatura"].map(ordem_temp)
```

[52]
```python
df.head()
```

	Temperatura	Umidade	Ventando	Jogar	Aparencia_Chuva	Aparencia_Nublado	Aparencia_Sol
0	2	0	0	Não	0.0	0.0	1.0
1	2	0	1	Não	0.0	0.0	1.0
2	2	0	0	Sim	0.0	1.0	0.0
3	1	0	0	Sim	1.0	0.0	0.0
4	0	1	0	Sim	1.0	0.0	0.0

Próximas etapas: Gerar código com df | Ver gráficos recomendados | New interactive sheet

[53]
```python
# Primeiro, vamos dividir em conjunto de treino e teste.
from sklearn.model_selection import train_test_split

X = df.drop("Jogar", axis=1)
y = df["Jogar"]

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

[54]
```python
# Instanciação do modelo.
modelo = GaussianNB()
```

```
[55] modelo.fit(X_train, y_train)
```

```
GaussianNB
GaussianNB()
```

```
[56] # Testar o modelo
     y_pred = modelo.predict(X_test)
```

```
[57] y_pred
```

```
array(['Não', 'Sim', 'Não'], dtype='<U3')
```

```
[58] y_test
```

	Jogar
9	Sim
11	Sim
0	Não

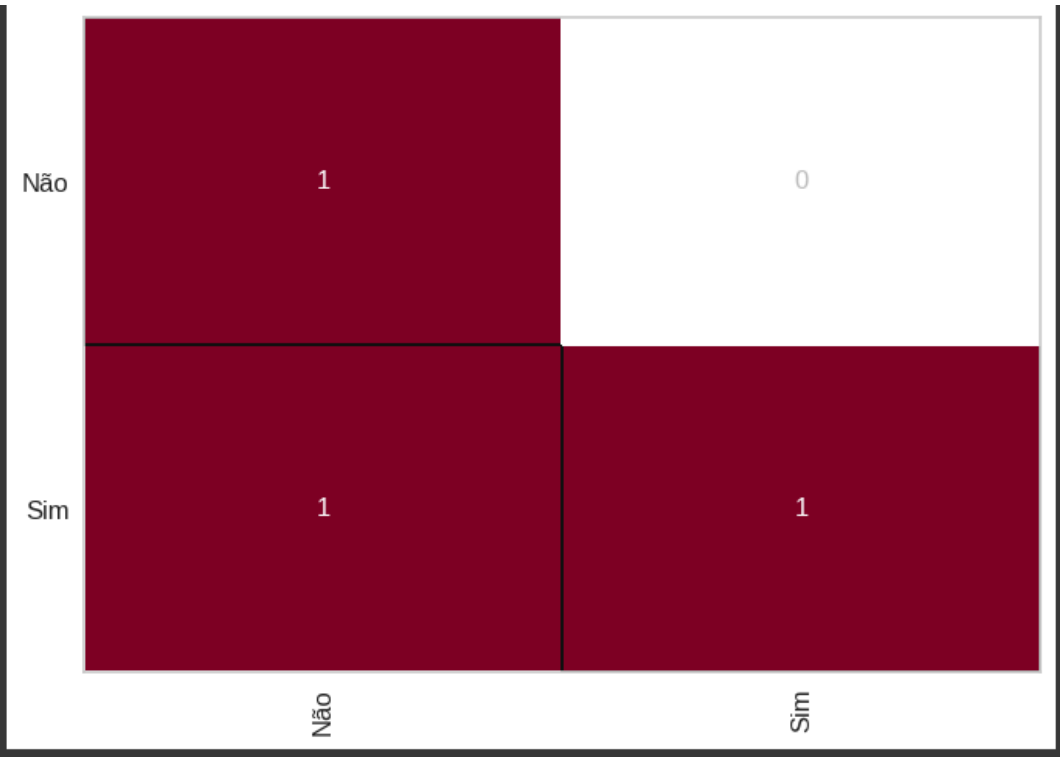
dtype: object

```
[60] from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

     accuracy_score(y_test, y_pred)
```

```
0.6666666666666666
```

O modelo teve 66,66% de acurácia.




```
[62] print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         Não       0.50      1.00      0.67         1
         Sim       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

```
[63] # Agora, vamos testar com uma instância nova personalizada.
     nova_instancia = pd.DataFrame({
         "Temperatura": [0],
         "Umidade": [1],
         "Ventando": [1],
         "Aparencia_Chuva": [1],
         "Aparencia_Nublado": [0],
         "Aparencia_Sol": [0],
     })
```

```
[64] # Testando a nova instancia.
     nova_predicao = modelo.predict(nova_instancia)
```

```
nova_predicao
```

```
array(['Não'], dtype='<U3')
```

O modelo classificou essa nova instância como sendo da classe 'Não'. Ele errou, porque realizando os cálculos no papel a classe correta deveria ser sim com uma probabilidade de 82,3% para a classe 'Sim'.

Questão 03

Implemente o método de Random Forest utilizando o python. Utilize a base acima e compare o resultado deste método com o Naive Bayes e a Árvore de decisão. Ajuste os hiperparâmetros,

utilizando o RandomSearch e algum outro otimizador da sua escolha. Compare os resultados.

Implementação e Teste com Random Forest

```
# Agora, vamos comparar o Naive Bayes implementando o Random Forest.
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)

# Vamos utilizar o RandomSearch para encontrar os melhores hiperparâmetros.
from sklearn.model_selection import RandomizedSearchCV

# Configuração de RandomSearch
param_grid_rf = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

rf_random = RandomizedSearchCV(estimator=rf,param_distributions=param_grid_rf,n_iter=20,cv=5,verbose=2,random_state=42,n_jobs=-1)

rf_random.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776: UserWarning: The least populated class in y has only 4 member
  warnings.warn(
```

RandomizedSearchCV

best_estimator_: RandomForestClassifier

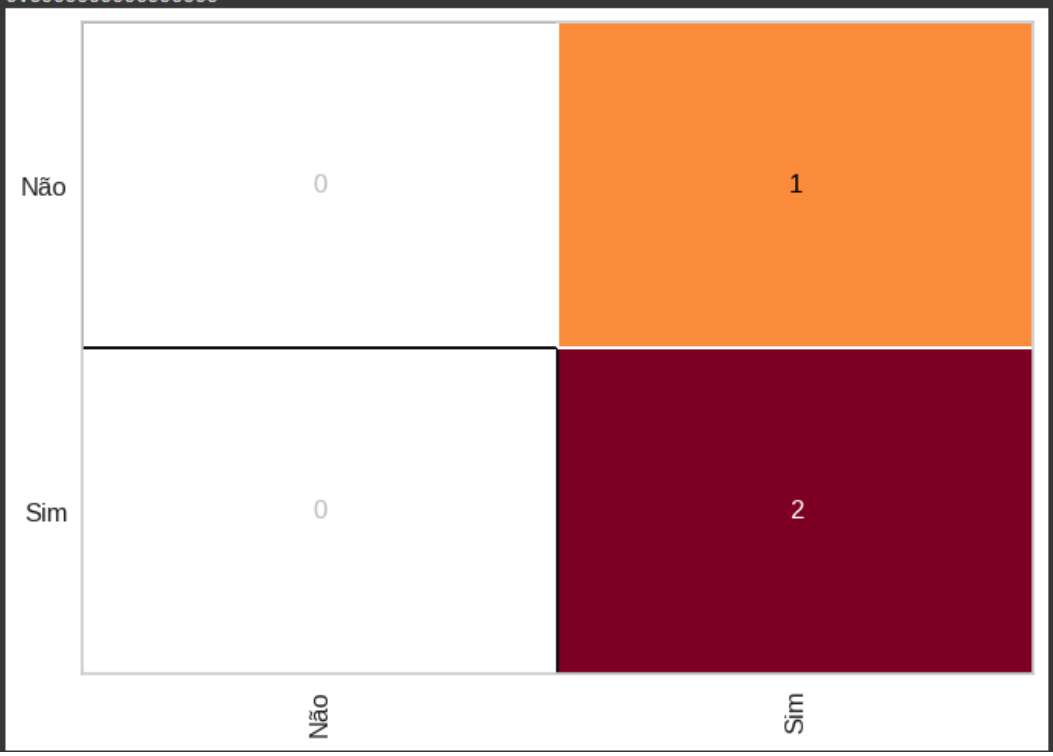
RandomForestClassifier

```
[68] y_pred_rf = rf_random.best_estimator_.predict(X_test)
     rf_accuracy = accuracy_score(y_test, y_pred_rf)
     print(f"Acurácia do modelo Random Forest: {rf_accuracy}")
```

```
Acurácia do modelo Random Forest: 0.6666666666666666
```

```
cm = ConfusionMatrix(rf_random)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but R
  warnings.warn(
0.6666666666666666
```




```
[70] print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         Não       0.50      1.00      0.67         1
         Sim       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3
```

Gerar | 10 random numbers using numpy

```
# Testando com a instância personalizada.
nova_predicao_rf = rf_random.best_estimator_.predict(nova_instancia)
```

```
[72] nova_predicao_rf
```

```
array(['Sim'], dtype=object)
```

Para o Random Forest com os hiperparâmetros ajustados utilizando RandomSearch, ele foi capaz de realizar o predict corretamente como sendo da classe 'Sim'.