LIsta de IA #4

Questão 1)

```
train_encoded = train.drop(columns=['PassengerId','Name', 'SibSp', 'Parch', 'Ticket','Fare','Cabin','Embarked'])
test_encoded = test.drop(columns=['PassengerId','Name', 'SibSp', 'Parch', 'Ticket','Fare','Cabin','Embarked'])
```

```
[7] from sklearn.preprocessing import LabelEncoder
```

```
[8] #para codificar todos os atributos para laberEncoder de uma única vez
    #base_encoded = base.apply(LabelEncoder().fit_transform)
    encoder = LabelEncoder()

    # Aplicando no atributo 'Sex'
    train_encoded['Sex'] = encoder.fit_transform(train_encoded['Sex'])
    test_encoded['Sex'] = encoder.fit_transform(test_encoded['Sex'])
    train_encoded
```

```
[9] train_encoded
```

```
[10] y_classe = train_encoded.iloc[:,0]
     X_prev = train_encoded.drop(columns=['Survived'])
```

```
[11] from sklearn.model_selection import train_test_split
```

```
[12] X_treino, X_teste, y_treino, y_teste = train_test_split(X_prev, y_classe, test_size = 0.20, random_state = 42)
```

```
---------- RANDOM FOREST ----------
```

```
[30] from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score
```

```
[17] from skopt import BayesSearchCV
```

```
[19] param_space_rf = {
         'n_estimators': (50, 500),
         'max_depth': (3, 20),
         'min_samples_split': (2, 20),
         'min_samples_leaf': (1, 10),
         'max_features': ['sqrt', 'log2', None]
     }
```

```
[20] rf = RandomForestClassifier(random_state=42)
```

```
[21] bayes_rf = BayesSearchCV(
         estimator=rf,
         search_spaces=param_space_rf,
         n_iter=32,           # número de iterações da busca
         cv=5,                # validação cruzada com 5 folds
         scoring='accuracy',  # métrica de avaliação
         random_state=42,
         n_jobs=-1
     )
```

```
[25] bayes_rf.fit(X_treino, y_treino)
```

```
[ ] print("Melhores hiperparâmetros para RandomForest:")
    print(bayes_rf.best_params_)
    print("Melhor score (cv):", bayes_rf.best_score_)
```

```
Melhores hiperparâmetros para RandomForest:
OrderedDict([('max_depth', 15), ('max_features', None), ('min_samples_leaf', 2), ('min_samples_split', 5), ('n_estimators', 410)])
Melhor score (cv): 0.8089530188121736
```

```
[33] rf_best = bayes_rf.best_estimator_
     y_pred_rf = rf_best.predict(X_teste)
     print("Acurácia no teste (RandomForest):", accuracy_score(y_teste, y_pred_rf))
```

```
Acurácia no teste (RandomForest): 0.8156424581005587
```

---------- Decision Tree ----------

```
[34] from sklearn.tree import DecisionTreeClassifier
```

```
[35] param_space_dt = {
         'max_depth': (1, 20),
         'min_samples_split': (2, 20),
         'min_samples_leaf': (1, 10),
         'criterion': ['gini', 'entropy']  # função de divisão
     }
```

```
[36] dt = DecisionTreeClassifier(random_state=42)
```

```
[37] bayes_dt = BayesSearchCV(
         estimator=dt,
         search_spaces=param_space_dt,
         n_iter=32,
         cv=5,
         scoring='accuracy',
         random_state=42,
         n_jobs=-1
     )
```

```
[38] bayes_dt.fit(X_treino, y_treino)
```

```
[ ] print("\nMelhores hiperparâmetros para DecisionTree:")
    print(bayes_dt.best_params_)
    print("Melhor score (cv):", bayes_dt.best_score_)
```

```
Melhores hiperparâmetros para DecisionTree:
OrderedDict([('criterion', 'gini'), ('max_depth', 18), ('min_samples_leaf', 2), ('min_samples_split', 2)])
Melhor score (cv): 0.8033389146065202
```

```
[42] dt_best = bayes_dt.best_estimator_
     y_pred_dt = dt_best.predict(X_teste)
     print("Acurácia no teste (DecisionTree):", accuracy_score(y_teste, y_pred_dt))
```

```
Acurácia no teste (DecisionTree): 0.7932960893854749
```

O modelo que obteve melhor desempenho foi o Random Forest, obtendo maior acurácia e score médio de CV em relação ao Decision Tree.
A melhora na acurácia e na estabilidade (score de CV) do Random Forest sugere que o ensemble de árvores ajuda a mitigar erros individuais e a reduzir o risco de overfitting.
A similaridade nos atributos mais importantes – principalmente 'Sex' e 'Pclass' – reforça conclusões bem estabelecidas sobre o dataset Titanic, onde a condição socioeconômica e o gênero têm grande influência sobre a chance de sobrevivência.

Questão 2)

SMOTE

```
[97] from sklearn.impute import SimpleImputer
     from imblearn.over_sampling import SMOTE

[110] imputer = SimpleImputer(strategy='most_frequent')
      X_prev_imputed = pd.DataFrame(imputer.fit_transform(X_treino), columns=X_treino.columns)

[112] smote = SMOTE(random_state=42)
      X_resampled, y_resampled = smote.fit_resample(X_prev_imputed, y_treino)
```

```
---------- RANDOM FOREST ----------

[102] from sklearn.ensemble import RandomForestClassifier
      from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

      rf = RandomForestClassifier(random_state=42)
      rf.fit(X_resampled, y_resampled)

[106] y_pred = rf.predict(X_teste)

      accuracy = accuracy_score(y_teste, y_pred)
      precision = precision_score(y_teste, y_pred)
      recall = recall_score(y_teste, y_pred)
      f1 = f1_score(y_teste, y_pred)

      print("Acurácia:", accuracy)
      print("Precisão:", precision)
      print("Recall:", recall)
      print("F1-Score:", f1)

      Acurácia: 0.9106145251396648
      Precisão: 0.9142857142857143
      Recall: 0.8648648648648649
      F1-Score: 0.8888888888888888
```

```
---------- Decision Tree ----------

[56] from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_resampled, y_resampled)

[109] y_pred = dt.predict(X_teste)

      accuracy = accuracy_score(y_teste, y_pred)
      precision = precision_score(y_teste, y_pred)
      recall = recall_score(y_teste, y_pred)
      f1 = f1_score(y_teste, y_pred)

      print("Acurácia:", accuracy)
      print("Precisão:", precision)
      print("Recall:", recall)
      print("F1-Score:", f1)

      Acurácia: 0.9050279329608939
      Precisão: 0.9253731343283582
      Recall: 0.8378378378378378
      F1-Score: 0.8794326241134752
```

TomekLinks

```
[36] from sklearn.impute import SimpleImputer
     from imblearn.under_sampling import TomekLinks

[35] imputer = SimpleImputer(strategy='most_frequent')
     X_prev_imputed = pd.DataFrame(imputer.fit_transform(X_treino), columns=X_treino.columns)

[ ] balanceamento_under = TomekLinks(sampling_strategy='auto')
    X_under, y_under = balanceamento_under.fit_resample(X_prev_imputed, y_treino)
```

---------- RANDOM FOREST ----------

```
[26] from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
[28] rf = RandomForestClassifier(random_state=42)
     rf.fit(X_under, y_under)
```

```
y_pred = rf.predict(X_teste)

accuracy = accuracy_score(y_teste, y_pred)
precision = precision_score(y_teste, y_pred)
recall = recall_score(y_teste, y_pred)
f1 = f1_score(y_teste, y_pred)

print("Acurácia:", accuracy)
print("Precisão:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Acurácia: 0.7877094972067039
Precisão: 0.7647058823529411
Recall: 0.7027027027027027
F1-Score: 0.7323943661971831
```

---------- Decision Tree ----------

```
[30] from sklearn.tree import DecisionTreeClassifier
```

```
[32] dt = DecisionTreeClassifier(random_state=42)
     dt.fit(X_under, y_under)
```

```
[33] y_pred = dt.predict(X_teste)

     accuracy = accuracy_score(y_teste, y_pred)
     precision = precision_score(y_teste, y_pred)
     recall = recall_score(y_teste, y_pred)
     f1 = f1_score(y_teste, y_pred)

     print("Acurácia:", accuracy)
     print("Precisão:", precision)
     print("Recall:", recall)
     print("F1-Score:", f1)
```

```
Acurácia: 0.776536312849162
Precisão: 0.7575757575757576
Recall: 0.6756756756756757
F1-Score: 0.7142857142857143
```

RandomUnderSampler

```
[17] from sklearn.impute import SimpleImputer
     from imblearn.under_sampling import RandomUnderSampler

[18] imputer = SimpleImputer(strategy='most_frequent')
     X_prev_imputed = pd.DataFrame(imputer.fit_transform(X_treino), columns=X_treino.columns)

[19] undersample = RandomUnderSampler(random_state=42)
     X_resampled, y_resampled = undersample.fit_resample(X_treino, y_treino)
```

---------- RANDOM FOREST ----------

```
[21] from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

[23] rf = RandomForestClassifier(random_state=42)
     rf.fit(X_resampled, y_resampled)

[24] y_pred = rf.predict(X_teste)

     accuracy = accuracy_score(y_teste, y_pred)
     precision = precision_score(y_teste, y_pred)
     recall = recall_score(y_teste, y_pred)
     f1 = f1_score(y_teste, y_pred)

     print("Acurácia:", accuracy)
     print("Precisão:", precision)
     print("Recall:", recall)
     print("F1-Score:", f1)
```

```
Acurácia: 0.8100558659217877
Precisão: 0.75
Recall: 0.8108108108108109
F1-Score: 0.7792207792207793
```

---------- Decision Tree ----------

```
[25] from sklearn.tree import DecisionTreeClassifier
```

```
[28] dt = DecisionTreeClassifier(random_state=42)
     dt.fit(X_resampled, y_resampled)
```

```
y_pred = dt.predict(X_teste)

accuracy = accuracy_score(y_teste, y_pred)
precision = precision_score(y_teste, y_pred)
recall = recall_score(y_teste, y_pred)
f1 = f1_score(y_teste, y_pred)

print("Acurácia:", accuracy)
print("Precisão:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Acurácia: 0.8044692737430168
Precisão: 0.7532467532467533
Recall: 0.7837837837837838
F1-Score: 0.7682119205298014
```

DSTO-GAN

```
from sklearn.impute import SimpleImputer
from dsto_gan import DSTO_GAN
import torch
import torch.nn.functional as F
```

```
[ ]  X_treino
```

+ Código + Texto

```
from sklearn.preprocessing import MinMaxScaler
scaler_age = MinMaxScaler(feature_range=(0, 1))
X_treino['Age_norm'] = scaler_age.fit_transform(X_treino[['Age']])
scaler_pclass = MinMaxScaler(feature_range=(0, 1))
X_treino['PClass_norm'] = scaler_pclass.fit_transform(X_treino[['Pclass']])
X_treino['PClass_norm_manual'] = (X_treino['Pclass'] - 1) / (3 - 1)
X_treino = X_treino.drop(columns=['Sex','Age', 'Pclass'])
```

```
[ ]  imputer = SimpleImputer(strategy='mean')

     X_treino_imputed = pd.DataFrame(imputer.fit_transform(X_treino), columns=X_treino.columns)
     X_treino_imputed_np = X_treino_imputed.to_numpy()
```

```
[ ]  dsto_gan = DSTO_GAN(dim_h=64, n_z=10, lr=0.0002, epochs=100, batch_size=64)
```

```
[ ]  X_resampled, y_resampled = dsto_gan.fit_resample(X_treino_imputed_np, y_treino)
```

Questão 3)

KNN

```
!pip install missingpy
!pip install scikit-learn --upgrade
from sklearn.impute import KNNImputer
```

```
[22]  imputer = KNNImputer(n_neighbors=5)
```

```
[29]  train_encoded_imputed = pd.DataFrame(imputer.fit_transform(train_encoded), columns=train_encoded.columns)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
[60] y_pred = rf.predict(X_teste)

     accuracy = accuracy_score(y_teste, y_pred)
     precision = precision_score(y_teste, y_pred)
     recall = recall_score(y_teste, y_pred)
     f1 = f1_score(y_teste, y_pred)

     print("Acurácia:", accuracy)
     print("Precisão:", precision)
     print("Recall:", recall)
     print("F1-Score:", f1)
```

```
Acurácia: 0.8212290502793296
Precisão: 0.7441860465116279
Recall: 0.8648648648648649
F1-Score: 0.8
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
[63] y_pred = dt.predict(X_teste)

     accuracy = accuracy_score(y_teste, y_pred)
     precision = precision_score(y_teste, y_pred)
     recall = recall_score(y_teste, y_pred)
     f1 = f1_score(y_teste, y_pred)

     print("Acurácia:", accuracy)
     print("Precisão:", precision)
     print("Recall:", recall)
     print("F1-Score:", f1)
```

```
Acurácia: 0.8156424581005587
Precisão: 0.7469879518072289
Recall: 0.8378378378378378
F1-Score: 0.7898089171974523
```

MODA

```
[8]  train_encoded_moda = train_encoded.copy()
```

```
[9]  train_encoded_moda['Age'].fillna(train_encoded_moda['Age'].mode()[0], inplace=True)
```

```
[12] y_classe = train_encoded_moda.iloc[:,0]
     X_prev = train_encoded_moda.drop(columns=['Survived'])
```

```
RandomForestClassifier(random_state=42)
```

```
y_pred = rf.predict(X_teste)

accuracy = accuracy_score(y_teste, y_pred)
precision = precision_score(y_teste, y_pred)
recall = recall_score(y_teste, y_pred)
f1 = f1_score(y_teste, y_pred)

print("Acurácia:", accuracy)
print("Precisão:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Acurácia: 0.8044692737430168
Precisão: 0.7407407407407407
Recall: 0.8108108108108109
F1-Score: 0.7741935483870968
```

```
DecisionTreeClassifier(random_state=42)
```

```
y_pred = dt.predict(X_teste)

accuracy = accuracy_score(y_teste, y_pred)
precision = precision_score(y_teste, y_pred)
recall = recall_score(y_teste, y_pred)
f1 = f1_score(y_teste, y_pred)

print("Acurácia:", accuracy)
print("Precisão:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Acurácia: 0.7988826815642458
Precisão: 0.7435897435897436
Recall: 0.7837837837837838
F1-Score: 0.7631578947368421
```