```python
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import sklearn as skl
import statsmodels.formula.api as smf
from statsmodels.tools import add_constant
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay, roc_curve, roc_auc_score, root_mean_squ
from sklearn.model_selection import RandomizedSearchCV, train_test_split, cross_val_score, StratifiedKFold, cross_val_predict
from scipy.stats import randint, kstest
```

```python
gen_sub = pd.read_csv('gender_submission.csv')
data_train = pd.read_csv('train.csv')
data_test = pd.read_csv('test.csv')
data_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```python
data_test.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Removendo variáveis inúteis

As variáveis "Name", "Ticket" e "Cabin" não serão úteis para a nossa análise.

```python
# removendo variáveis inúteis
data_train = data_train.drop(['Name', 'Ticket', 'Cabin'], axis = 1)
data_test = data_test.drop(['Name', 'Ticket', 'Cabin'], axis = 1)
# concatenando os dois bancos
data = pd.concat([data_train, data_test]).reset_index(drop = True)
data.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0.0	3	male	22.0	1	0	7.2500	S
1	2	1.0	1	female	38.0	1	0	71.2833	C
2	3	1.0	3	female	26.0	0	0	7.9250	S
3	4	1.0	1	female	35.0	1	0	53.1000	S
4	5	0.0	3	male	35.0	0	0	8.0500	S

Imputação de valores faltantes

Temos alguns dados faltantes: 263 na variável "Age", 1 na variável "Fare" e 2 na variável "Embarked".

```python
data.isna().sum() # 263 NAs em Age
                  # 1 NA em Fare
                  # 2 NAs em Embarked
```

```
PassengerId      0
Survived       418
Pclass           0
Sex              0
Age            263
SibSp            0
Parch            0
Fare             1
Embarked         2
dtype: int64
```

Imputação da variável Embarked

Como Embarked é uma variável categórica, vamos imputar com a moda.

```python
data['Embarked'].value_counts()
```

```
Embarked
S    914
C    270
Q    123
Name: count, dtype: int64
```

```python
data['Embarked'] = data['Embarked'].fillna('S') #imputando a moda em Embarked
```

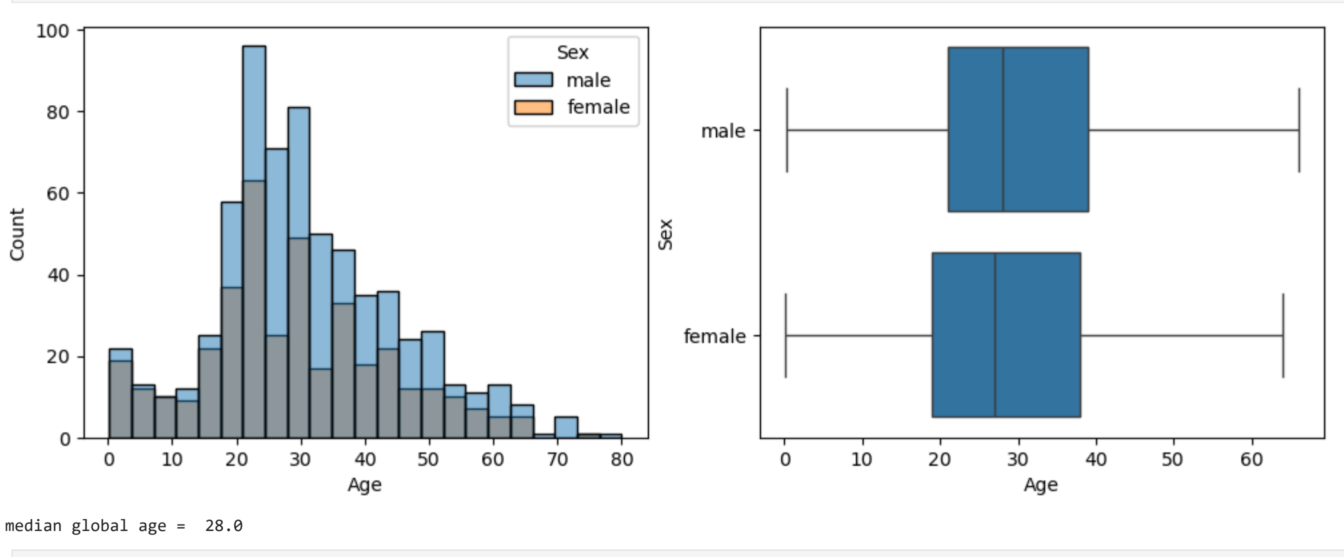
```python
data.duplicated().astype(int).sum() # nenhuma linha duplicada
```

```
np.int64(0)
```

Imputação da variável Age

Como a variável Age é contínua vamos analisar se podemos imputar a mediana. Iremos verificar a distribuição dos dados para cada categoria da variável Sex sem os outliers. Caso as distribuições de Age em cada categoria de Sex forem diferentes, não podemos imputar com a mediana.

```python
# verificando a distribuição dos dados de Age para as categorias de Sex
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12,4))
sns.histplot(x = 'Age', hue = 'Sex', data = data, ax = ax1)
sns.boxplot(x = 'Age', y = 'Sex', data = data, ax = ax2, showfliers = False)
plt.show()
# aparentemente tem a mesma distribuição, então imputar a mediana não vai comprometer os dados
print('median global age = ', data['Age'].median())
```




```
median global age =  28.0
```

```python
AgeMale = data['Age'][data['Sex'] == 'male'].dropna()
AgeFemale = data['Age'][data['Sex'] == 'female'].dropna()
kstest(AgeMale, AgeFemale).pvalue
```

```
np.float64(0.056346439588368526)
```

O teste Kolmogorov-Smirnov de duas amostras não encontrou evidências para afirmarmos que as distribuições são diferentes.

```python
data['Age'] = data['Age'].fillna(data['Age'].median())
data.isna().sum()
```

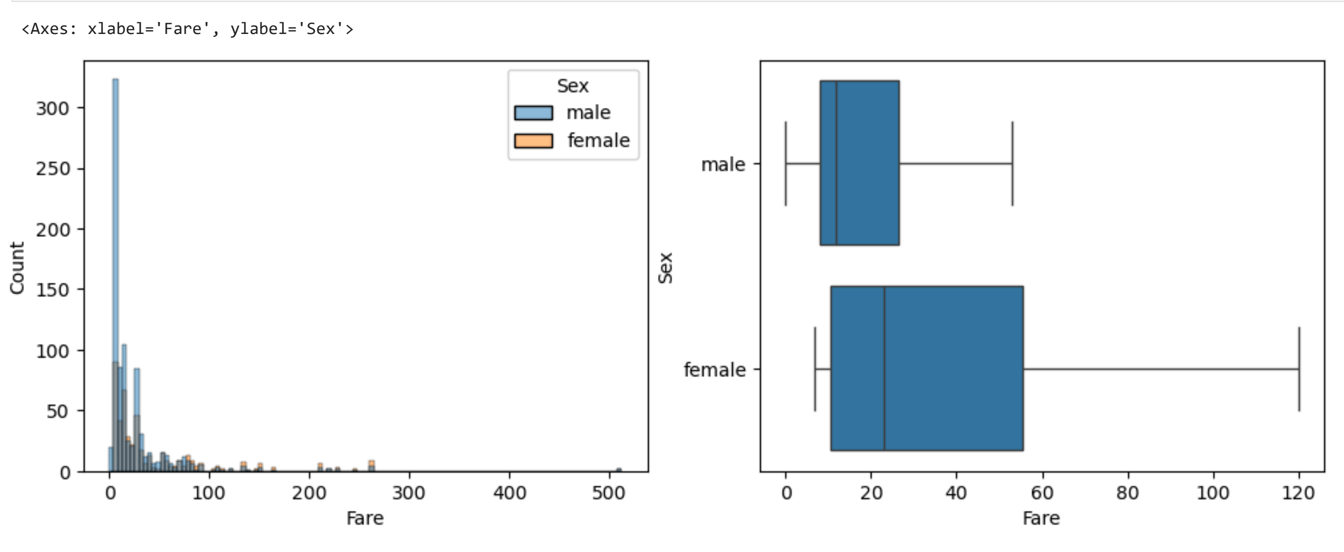
```
PassengerId      0
Survived       418
Pclass           0
Sex              0
Age              0
SibSp            0
Parch            0
Fare             1
Embarked         0
dtype: int64
```

Imputação da variável Fare

Faremos o mesmo que fizemos na variável Age, na variável Fare.

```python
# verificando a distribuição dos dados de Fare para as categorias de Sex
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12,4))
sns.histplot(x = 'Fare', hue = 'Sex', data = data, ax = ax1)
sns.boxplot(x = 'Fare', y = 'Sex', data = data, ax = ax2, showfliers = False)
```

```
<Axes: xlabel='Fare', ylabel='Sex'>
```




```python
FareMale = data['Fare'][data['Sex'] == 'male'].dropna()
FareFemale = data['Fare'][data['Sex'] == 'female'].dropna()
kstest(FareMale, FareFemale).pvalue
```

```
np.float64(7.421785903652769e-16)
```

O teste de Kolmogorov-Smirnov para duas amostras encontrou evidências para afirmarmos que as distribuições de Fare para cada nível de Sex são diferentes. Vamos utilizar MICE para imputar na observação faltante já que não há problemas em usar.

Antes de utilizarmos o MICE precisamos remover as variáveis que não podemos usar (PassengerId, pois é somente um índice; e Survived, pois é nossa variável resposta, logo comprometeria nosso modelo preditivo) e colocar as variáveis categóricas (Sex, Pclass e Embarked) como dummies.

```python
# imputando para Fare usando MICE
x = data.drop(['PassengerId', 'Survived'], axis = 1)
x = pd.get_dummies(x, columns = ['Sex', 'Pclass', 'Embarked'], drop_first = True)
imp = IterativeImputer(random_state = 28051996)
imputed = imp.fit_transform(x)
data_imputed = pd.DataFrame(imputed, columns=x.columns)
data_imputed = pd.concat([data[['PassengerId', 'Survived']], data_imputed], axis = 1)
data_imputed.head()
```

	PassengerId	Survived	Age	SibSp	Parch	Fare	Sex_male	Pclass_2	Pclass_3	Embarked_Q	Embarked_S
0	1	0.0	22.0	1.0	0.0	7.2500	1.0	0.0	1.0	0.0	1.0
1	2	1.0	38.0	1.0	0.0	71.2833	0.0	0.0	0.0	0.0	0.0
2	3	1.0	26.0	0.0	0.0	7.9250	0.0	0.0	1.0	0.0	1.0
3	4	1.0	35.0	1.0	0.0	53.1000	0.0	0.0	0.0	0.0	1.0
4	5	0.0	35.0	0.0	0.0	8.0500	1.0	0.0	1.0	0.0	1.0

```python
data_imputed.isna().sum() # nenhum dado faltante
```

```
PassengerId      0
Survived       418
Age              0
SibSp            0
Parch            0
Fare             0
Sex_male         0
Pclass_2         0
Pclass_3         0
Embarked_Q       0
Embarked_S       0
dtype: int64
```

```python
# refazendo as bases de treino e de teste
data_train = data_imputed.iloc[0:891]
data_test = data_imputed.iloc[891:]
```

Modelo preditivo usando random forest

Criação do modelo

```python
# criando uma Random Forest
features = ['Age', 'SibSp', 'Parch', 'Fare', 'Sex_male', 'Pclass_2', 'Pclass_3', 'Embarked_Q', 'Embarked_S']
y_train = data_train['Survived']
x_train = data_train[features]
y_test = gen_sub['Survived']
x_test = data_test[features]
model = RandomForestClassifier(random_state = 28051996)
modelFit = model.fit(x_train, y_train)
```

Precisamos validar nosso modelo, portanto faremos a cross validation utilizando 5 Kfolds estratificados para diminuir a variabilidade dentro dos subdatasets, dando maior confiabilidade à avaliação do modelo.

```python
# dividindo o dataset de treino em 5
cv = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 28051996)
# fazendo cross validation para cada método de avaliação do modelo
crossval_acc = cross_val_score(model, x_train, y_train, cv = cv, scoring = 'accuracy')
crossval_prec = cross_val_score(model, x_train, y_train, cv = cv, scoring = 'precision')
crossval_rc = cross_val_score(model, x_train, y_train, cv = cv, scoring = 'recall')

def intervalo_acc(crossval):
    mean = crossval_acc.mean()
    dv = crossval_acc.std()
    print('\nAcurácia média: {:.2f}%'.format(mean*100))
    print('Desvio-padrão da acurácia: {:.2f}%'.format(dv*100))
    print('Intervalo da acurácia: [{:.2f}% - {:.2f}%]\n'.format((mean-2*dv)*100, (mean + 2*dv)*100))

def intervalo_prec(crossval):
    mean = crossval.mean()
    dv = crossval.std()
    print('\nPrecisão média: {:.2f}%'.format(mean*100))
    print('Desvio-padrão da precisão: {:.2f}%'.format(dv*100))
    print('Intervalo da precisão: [{:.2f}% - {:.2f}%]\n'.format((mean-2*dv)*100, (mean + 2*dv)*100))

def intervalo_rc(crossval):
    mean = crossval.mean()
    dv = crossval.std()
    print('\nRecall médio: {:.2f}%'.format(mean*100))
    print('Desvio-padrão do recall: {:.2f}%'.format(dv*100))
    print('Intervalo do recall: [{:.2f}% - {:.2f}%]\n'.format((mean-2*dv)*100, (mean + 2*dv)*100))

# os modelos são moderados, mas pode melhorar
intervalo_acc(crossval_acc)
intervalo_prec(crossval_prec)
intervalo_rc(crossval_rc)
```

```
Acurácia média: 81.93%
Desvio-padrão da acurácia: 3.89%
Intervalo da acurácia: [74.16% - 89.71%]


Precisão média: 77.18%
Desvio-padrão da precisão: 4.77%
Intervalo da precisão: [67.64% - 86.73%]


Recall médio: 75.15%
Desvio-padrão do recall: 7.07%
Intervalo do recall: [61.01% - 89.30%]
```

Para a quantidade e tipos de dados no banco de dados, a acurácia está boa, a precisão nem tanto e, como esperado, o recall está mais baixo.

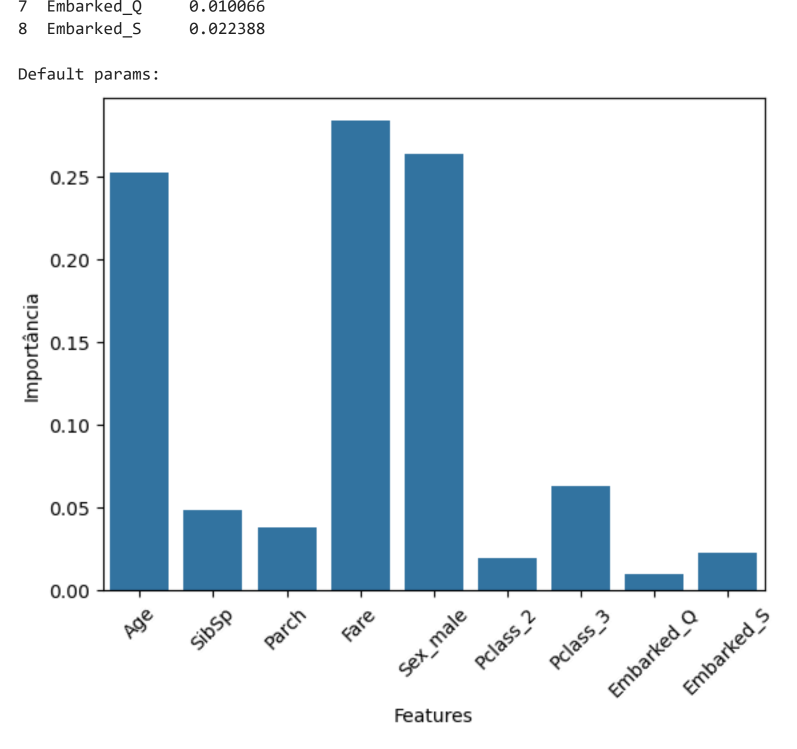
Importância das variáveis

```python
# calculando a taxa de importância de cada feature
y_pred = model.predict(x_test)
importancias = pd.DataFrame({
    'Features' : features,
    'Importância': model.feature_importances_
})
print(importancias)
print('\nDefault params: ')

ax = sns.barplot(x = 'Features', y = 'Importância', data = importancias)
plt.xticks(rotation = 45)
plt.show()
```

```
     Features  Importância
0         Age     0.252150
1       SibSp     0.048542
2       Parch     0.037913
3        Fare     0.283366
4    Sex_male     0.263534
5    Pclass_2     0.019257
6    Pclass_3     0.062784
7  Embarked_Q     0.010066
8  Embarked_S     0.022388

Default params:
```



Temos 3 variáveis que suas importâncias se sobressaem em relação às outras. Age, Fare e Sex

Avaliação dos hiperparâmetros

Podemos melhorar o modelo fazendo uma investigação em alguns hiperparâmetros.

```python
# preparando uma avaliação dos melhores valores para alguns hiperparâmetros
n_estimators = range(50,500,50)
max_depth =  range(2, 30, 2)
min_samples_split = [2, 5, 10]
max_features = [2, 5, 10]
bootstrap = [True, False]
param_dist = dict(
                  min_samples_split = min_samples_split,
                  max_features = max_features,
                  bootstrap = bootstrap,
                  n_estimators = n_estimators,
                  max_depth = max_depth
                 )

# verificando os valores
rand_search = RandomizedSearchCV(model,
                                 param_distributions = param_dist,
                                 n_iter=5,
                                 cv=5,
                                random_state = 28051996)

# fit do melhor modelo
rand_result = rand_search.fit(x_train, y_train)
best_params = rand_result.best_params_
print('Best Score: ', rand_result.best_score_)
print('Best Params: ', rand_result.best_params_)
```

```
Best Score:  0.8361810306948717
Best Params:  {'n_estimators': 100, 'min_samples_split': 10, 'max_features': 2, 'max_depth': 14, 'bootstrap': False}
```

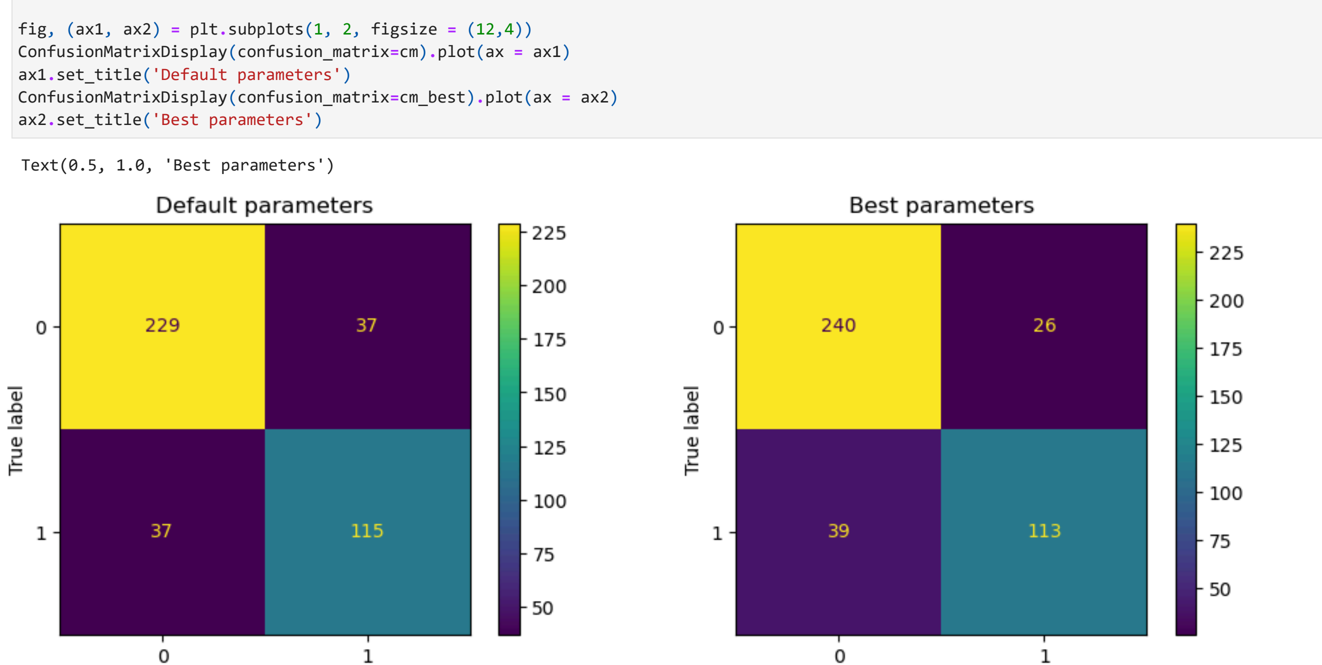
Comparação das matrizes de confusão dos modelos inicial e o otimizado

```python
# comparação das matrizes de confusão
y_best_pred = rand_result.predict(x_test)

cm = confusion_matrix(y_test, y_pred) # matriz de confusão do ajuste com os primeiros valores de hiperparâmetros
cm_best = confusion_matrix(y_test, y_best_pred) # matriz de confusão do ajuste com os melhores valores de hiperparâmetros

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12,4))
ConfusionMatrixDisplay(confusion_matrix=cm).plot(ax = ax1)
ax1.set_title('Default parameters')
ConfusionMatrixDisplay(confusion_matrix=cm_best).plot(ax = ax2)
ax2.set_title('Best parameters')
```

```
Text(0.5, 1.0, 'Best parameters')
```




```python
# avaliação do modelo com melhores hiperparâmetros
accuracy_best = accuracy_score(y_test, y_best_pred)
precision_best = precision_score(y_test, y_best_pred)
recall_best = recall_score(y_test, y_best_pred)

print("Accuracy:", accuracy_best)
print("Precision:", precision_best)
print("Recall:", recall_best)
```

```
Accuracy: 0.8444976076555024
Precision: 0.8129496402877698
Recall: 0.743421052631579
```

Todas as métricas melhoraram com a otimização dos hiperparâmetros.