

```
In [1]: 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 as sm
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 statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import roc_curve, roc_auc_score, root_mean_squared_error
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import StratifiedKFold
from scipy.stats import kstest

In [2]: gen_sub = pd.read_csv('gender_submission.csv')
data_train = pd.read_csv('train.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	Cummings, 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

```
In [3]: data_test = pd.read_csv('test.csv')
data_test.head()

Out[3]:
```

	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.

```
In [4]: data_train = data_train.drop(['Name', 'Ticket', 'Cabin'], axis = 1)
data_test = data_test.drop(['Name', 'Ticket', 'Cabin'], axis = 1)
data = pd.concat([data_train, data_test]).reset_index(drop = True)
data.head()

Out[4]:
```

	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".

```
In [5]: data.isna().sum() # 263 NAs em Age
# 1 NA em Fare
# 2 NAs em Embarked

Out[5]:
```

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

```
In [6]: data.duplicated().astype(int).sum() # nenhuma linha duplicada
Out[6]: np.int64(0)
```

Imputação da variável Embarked

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

```
In [7]: data['Embarked'].value_counts()

Out[7]:
```

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

```
In [8]: data['Embarked'] = data['Embarked'].fillna('S')

In [9]: data.isna().sum()

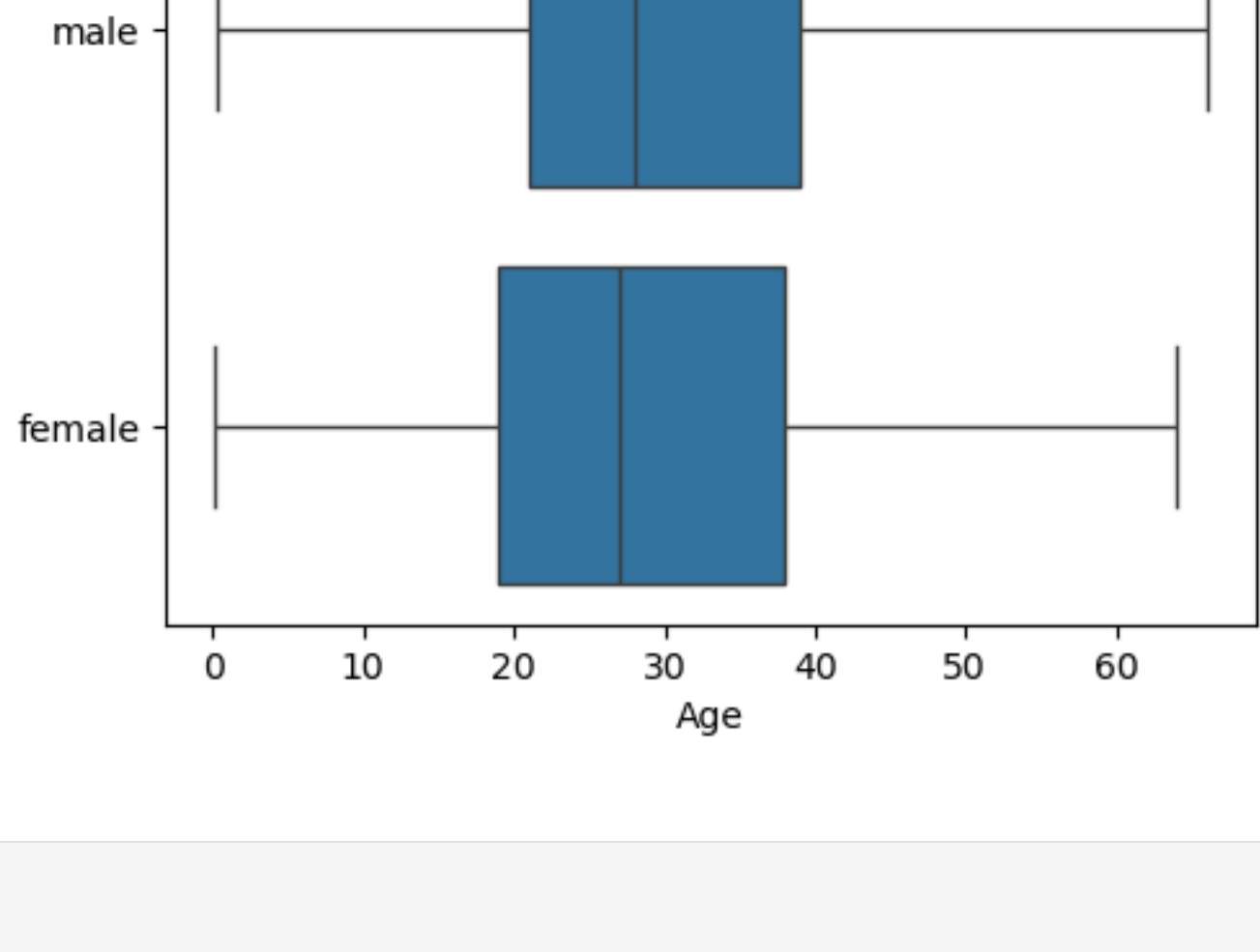
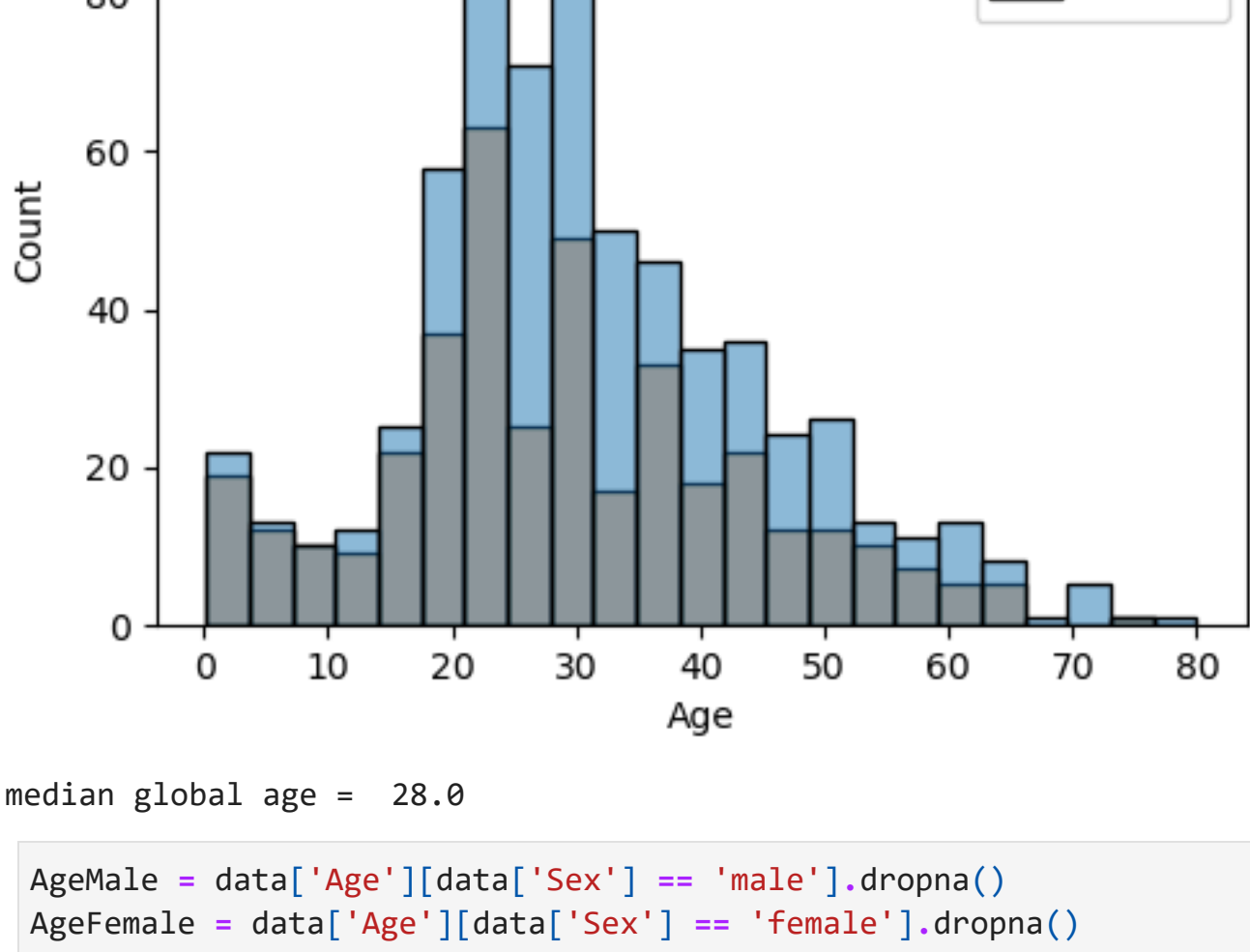
Out[9]:
```

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

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.

```
In [10]: # 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

```
In [11]: AgeMale = data['Age'][data['Sex'] == 'male'].dropna()
AgeFemale = data['Age'][data['Sex'] == 'female'].dropna()
kstest(AgeMale, AgeFemale).pvalue

Out[11]: np.float64(0.056346439588368526)
```

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

```
In [12]: data['Age'] = data['Age'].fillna(data['Age'].median())
data.isna().sum()

Out[12]:
```

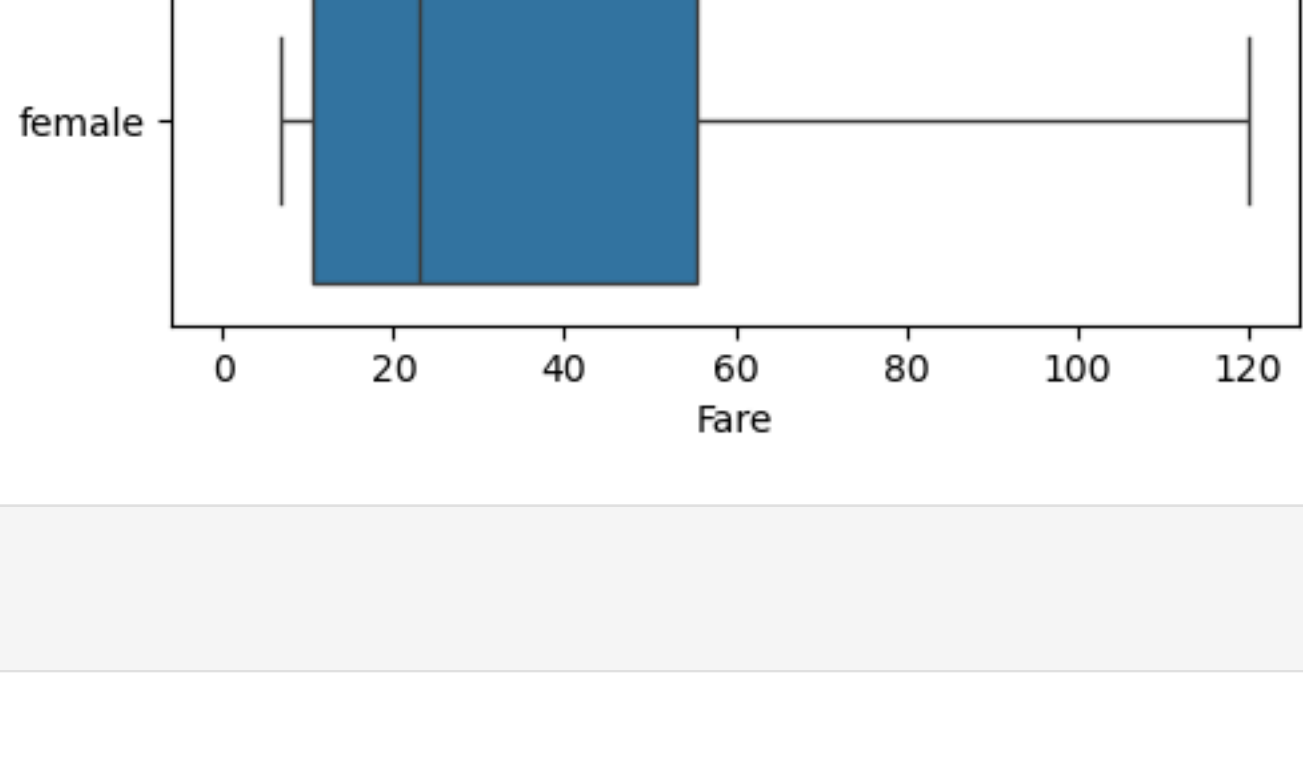
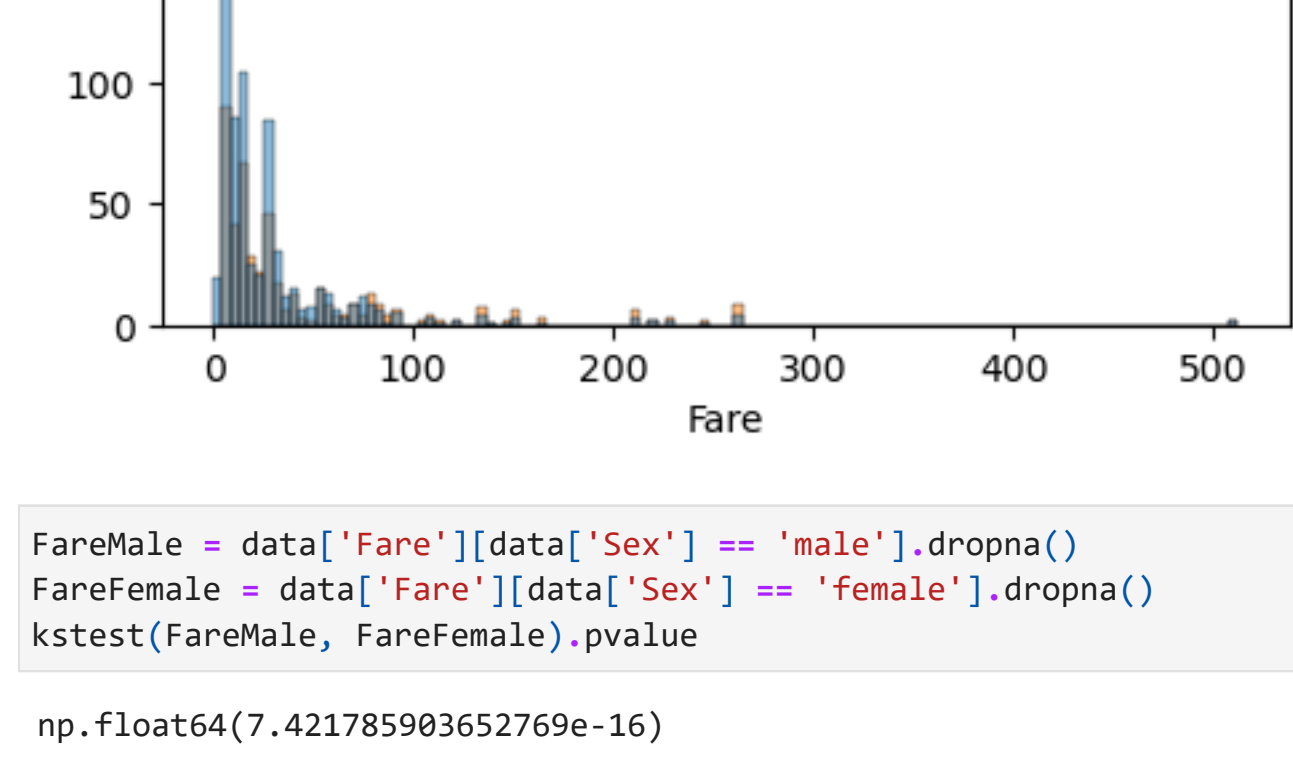
PassengerId	0
Survived	418
Pclass	0
Sex	0
Age	0
SibSp	0
Parch	0
Fare	1
Embarked	0
dtype:	int64

Imputação da variável Fare

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

```
In [13]: # 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)

Out[13]: <Axes: xlabel='Fare', ylabel='Sex'>
```



```
In [14]: FareMale = data['Fare'][data['Sex'] == 'male'].dropna()
FareFemale = data['Fare'][data['Sex'] == 'female'].dropna()
kstest(FareMale, FareFemale).pvalue

Out[14]: 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.

```
In [15]: # 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
```

	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
...
1304	1305	NaN	28.0	0.0	0.0	8.0500	1.0	0.0	1.0	0.0	1.0
1305	1306	NaN	39.0	0.0	0.0	108.9000	0.0	0.0	0.0	0.0	0.0
1306	1307	NaN	38.5	0.0	0.0	7.2500	1.0	0.0	1.0	0.0	1.0
1307	1308	NaN	28.0	0.0	0.0	8.0500	1.0	0.0	1.0	0.0	1.0
1308	1309	NaN	28.0	1.0	1.0	22.3583	1.0	0.0	1.0	0.0	0.0

1309 rows x 11 columns

```
In [16]: data_imputed.isna().sum() # nenhum dado faltante

Out[16]:
```

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

```
In [17]: # refazendo as bases de treino e de teste
data_train = data_imputed.iloc[0:891]
data_test = data_imputed.iloc[891:]
```

Modelo preditivo usando regressão logística

Verificação de multicolinearidade

Vamos usar o Variance Inflation Factor para medir a multicolinearidade entre as variáveis do modelo.

```
In [18]: # regressão logística
x = data_train.drop('Survived', axis = 1)
x = add_constant(x)
y = data_train['Survived']
vif = pd.DataFrame()
vif['Variável'] = x.columns
vif['VIF'] = [variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
vif

Out[18]:
```

	Variável	VIF
0	const	26.988491
1	PassengerId	1.008011
2	Age	1.224076
3	SibSp	1.289590
4	Parch	1.334924
5	Fare	1.765767
6	Sex_male	1.133313
7	Pclass_2	2.079747
8	Pclass_3	2.767721
9	Embarked_Q	1.493583
10	Embarked_S	1.501090

Podemos verificar que não há multicolinearidade forte entre as variáveis. Portanto, podemos seguir com a implementação do modelo

Definição do modelo

```
In [19]: formula = 'y ~ Age + SibSp + Parch + Fare + Sex_male + Pclass_2 + Pclass_2 + Embarked_Q + Embarked_S'
SurvMod = smf.logit(formula, data = data_train)
SurvFit = SurvModel.Fit()
SurvFit.summary()
```

Optimization terminated successfully.	
Current function value: 0.468969	
Iterations: 6	

```
Out[19]:
```

Dep. Variable:	y	No. Observations:	891
Model:	Logit	Df Residuals:	882
Method:	MLE	Df Model:	8
Date:	Wed, 23 Apr 2025	Pseudo R-squ.:	0.2957
Time:	17:47:39	Log-Likelihood:	-417.85
converged:	True	LL-Null:	-593.33
Covariance Type:	nonrobust	LLR p-value:	5.669e-71

	coef	std err	z	P> z	[0.025	0.975]
Intercept	1.8628	0.337	5.533	0.000	1.203	2.523
Age	-0.0207	0.007	-2.883	0.004	-0.035	-0.007
SibSp	-0.3825	0.105	-3.648	0.000	-0.588	-0.177
Parch	-0.2280	0.115	-1.988	0.047	-0.453	-0.003
Fare	0.0157	0.003	5.464	0.000	0.010	0.021
Sex_male	-2.6008	0.191	-13.642	0.000	-2.974	-2.227
Pclass_2	0.6385	0.210	3.040	0.002	0.227	1.050
Embarked_Q	-0.5957	0.363	-1.643	0.100	-1.306	0.115
Embarked_S	-0.5965	0.229	-2.601	0.009	-1.046	-0.147

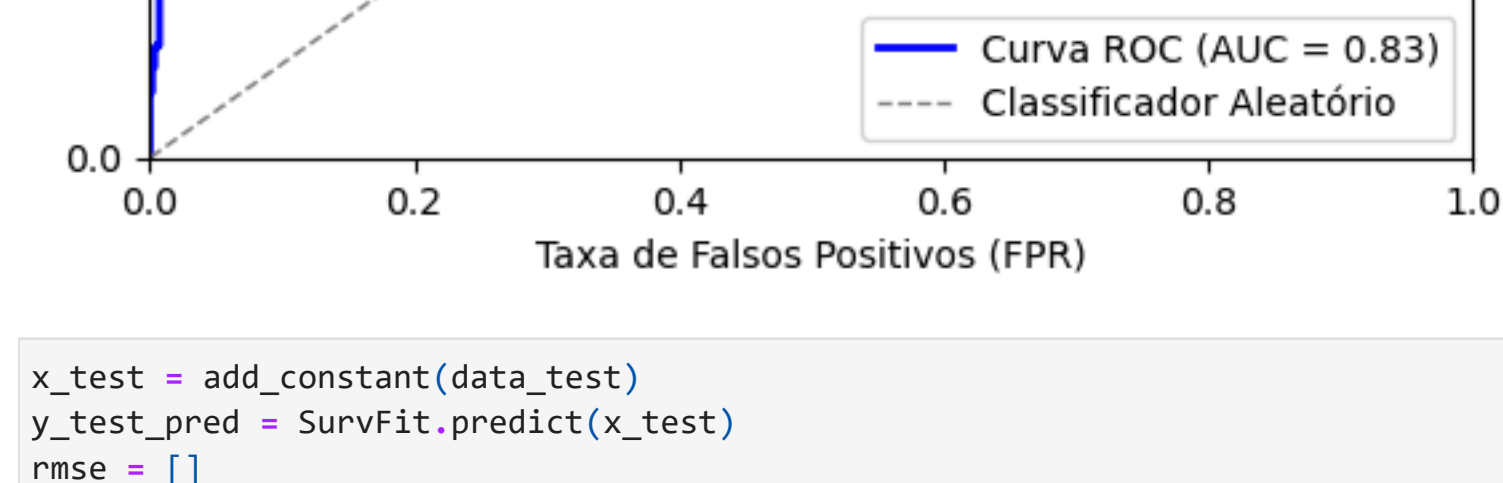
Podemos observar que apenas um dos níveis da variável Embarked não é significativa, mas iremos deixar como está, pois um dos níveis é significativo e remover um dos níveis comprometerá o modelo.

Curva ROC

Vamos utilizar a curva ROC e o AUC para definirmos o limiar que melhor se adequa ao modelo. Utilizaremos o limiar que minimiza a raiz quadrada do erro quadrático médio.

```
In [20]: y_pred = SurvFit.predict()
fpr, tpr, thresholds = roc_curve(y, y_pred)
auc = roc_auc_score(y, y_pred)

plt.plot(fpr, tpr, color='blue', lw=2, label=f'Curva ROC (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=1, label='Classificador Aleatório')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('Taxa de Falsos Positivos (FPR)')
plt.ylabel('Taxa de Verdadeiros Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.show()
```



```
In [21]: x_test = add_constant(data_test)
y_test_pred = SurvFit.predict(x_test)
rmse = []
for indice, value in enumerate(thresholds):
    y_final_hat = (y_test_pred >= value).astype(int)
    rmse.append(root_mean_squared_error(y_true = gen_sub['Survived'], y_pred = y_final_hat))
print('Limiar: ', thresholds[np.argmin(rmse)])
print('Valor do rmse: ', min(rmse))

y_final_hat = (y_test_pred >= thresholds[np.argmin(rmse)]).astype(int)

Limiar: 0.5343584858146607
Valor do rmse: 0.20166768818858946
```