03_RegressoesTitanic_v1.1-PauloBraga

July 5, 2020

[36]:
```python
''' Paulo Simplício Braga
    05.07.2020
'''
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from yellowbrick.classifier import ConfusionMatrix
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')

train_df = pd.read_csv('Data/train.csv')

train_df.corr().style.background_gradient().set_precision(2)
```

[36]:
```
<pandas.io.formats.style.Styler at 0x7f0c3cdc7390>
```

I - Cálculo da % de valores faltantes por coluna:

[2]:
```python
total = train_df.isnull().sum().sort_values(ascending=False)
print(total)
```

```
Cabin          687
Age            177
Embarked         2
```

```
Fare              0
Ticket            0
Parch             0
SibSp             0
Sex               0
Name              0
Pclass            0
Survived          0
PassengerId       0
dtype: int64
```

[3]:
```
pct_1 = train_df.isnull().sum()/train_df.isnull().count()*100
print(pct_1)
```

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

[4]:
```
pct_1 = round(pct_1, 1).sort_values(ascending=False)
print(pct_1)
```

```
Cabin          77.1
Age            19.9
Embarked        0.2
Fare            0.0
Ticket          0.0
Parch           0.0
SibSp           0.0
Sex             0.0
Name            0.0
Pclass          0.0
Survived        0.0
PassengerId     0.0
dtype: float64
```

[5]:
```
dados_faltantes = pd.concat([total,pct_1], axis=1, keys=['Total', '%'])
dados_faltantes.head()
```

[5]:
```
          Total     %
Cabin       687  77.1
Age         177  19.9
Embarked      2   0.2
Fare          0   0.0
Ticket        0   0.0
```

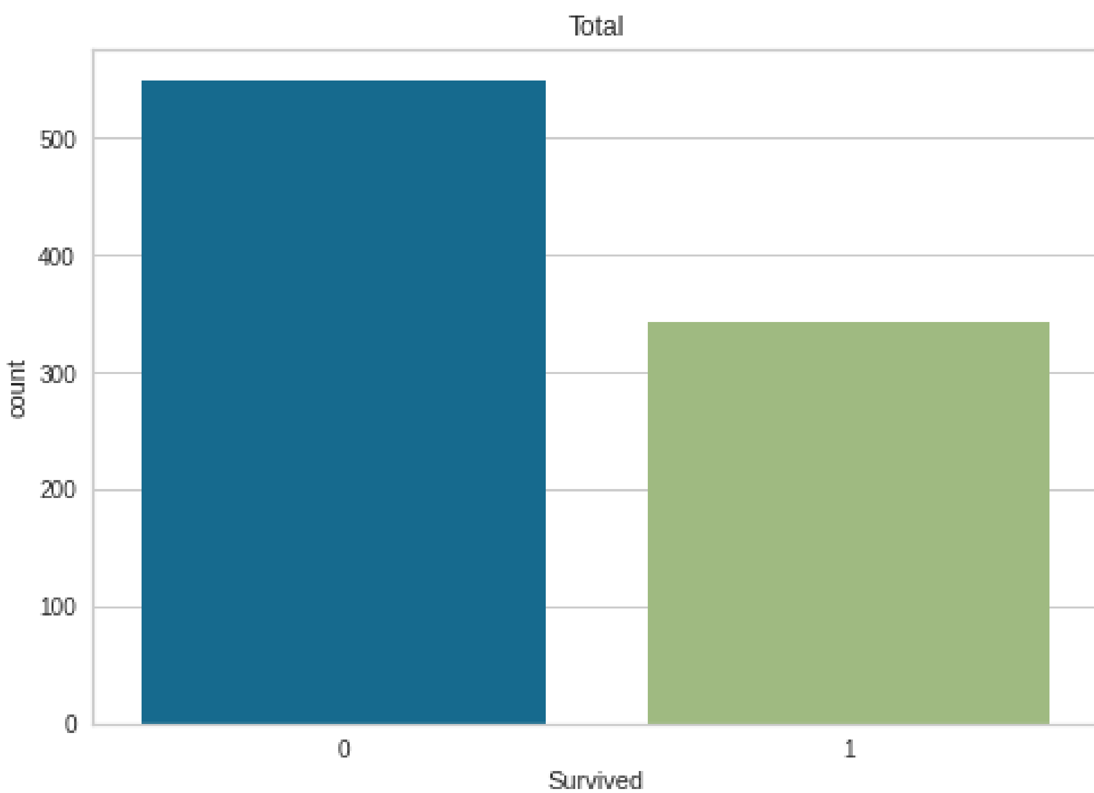
II - Sobreviventes por sexo

[6]:
```python
# Quantidade por sexo
train_df['Sex'].value_counts()
```

[6]:
```
male      577
female    314
Name: Sex, dtype: int64
```

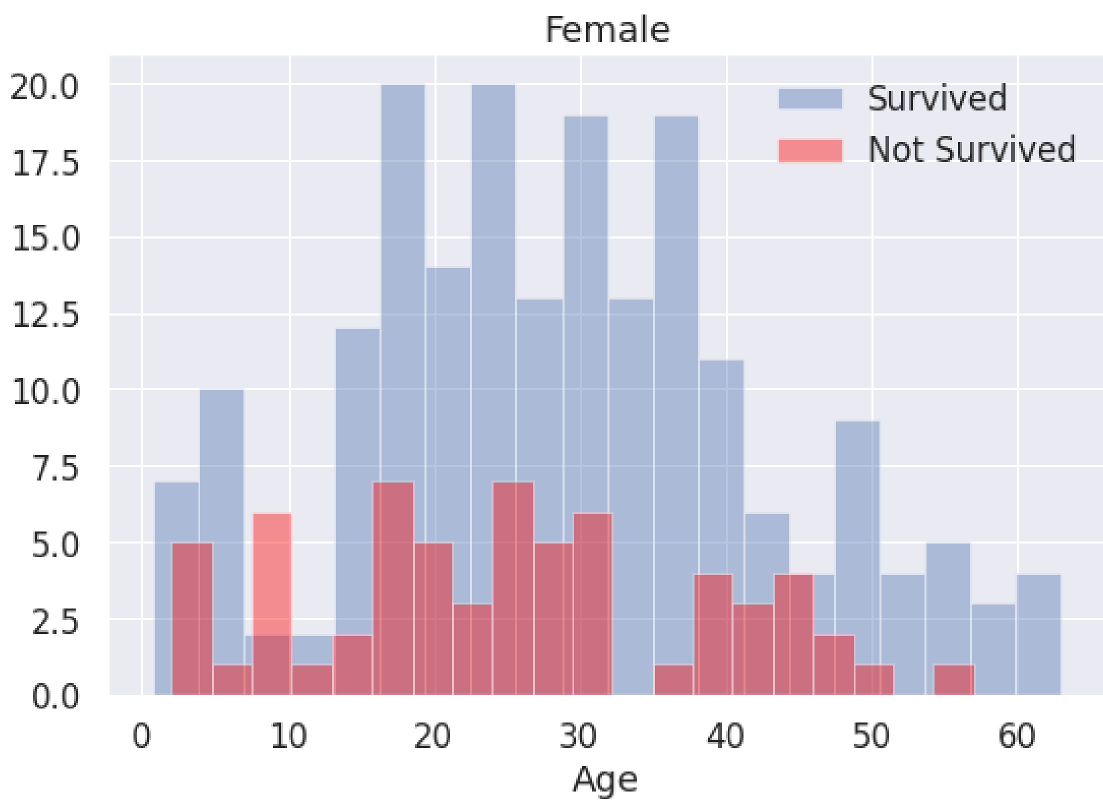
[7]:
```python
# Total de sobreviventes
sns.countplot(train_df['Survived']).set_title('Total')
```

[7]:
```
Text(0.5, 1.0, 'Total')
```



[8]:
```
sns.set(font_scale=1.5)
fem = train_df[train_df['Sex']=='female'] # Cria um dataframe para female

# Cria um objeto 'fig' pra fazer o subplot
fig = plt.figure(figsize=(20,6))

# Adiciona o histograma 'Survived'
fig.add_subplot(1,2,1) # linhas, colunas, posição
to_plot = sns.distplot(fem[fem['Survived']==1].Age, bins=20, kde=False,\
                      label='Survived')
to_plot.set_title('Female')
to_plot.legend()

# Adiciona o histograma 'Not Survived'
fig.add_subplot(1,2,1) # linhas, colunas, posição
to_plot = sns.distplot(fem[fem['Survived']==0].Age, bins=20, kde=False,\
             color='red', label='Not Survived')
to_plot.legend()
```

[8]: <matplotlib.legend.Legend at 0x7f0c5533e8d0>

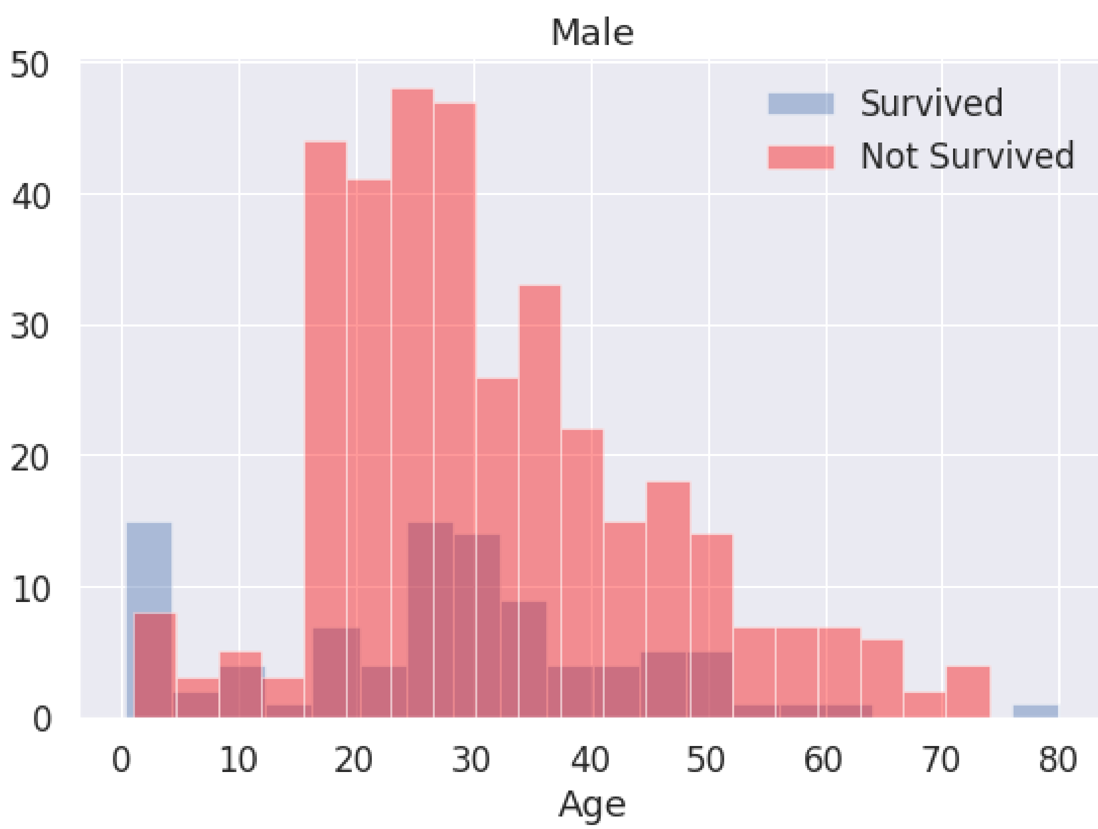


[9]:

```
sns.set(font_scale=1.5)
male = train_df[train_df['Sex']=='male'] # Cria um dataframe para male

fig = plt.figure(figsize=(20,6))

fig.add_subplot(1,2,1)
to_plot = sns.distplot(male[male['Survived']==1].Age, bins=20, kde=False,\
                       label='Survived')
to_plot.set_title('Male')
to_plot.legend()

fig.add_subplot(1,2,1)
to_plot = sns.distplot(male[male['Survived']==0].Age, bins=20, kde=False,\
                      color='red', label='Not Survived')
to_plot.legend()
```

[9]:

```
<matplotlib.legend.Legend at 0x7f0c55058f98>
```

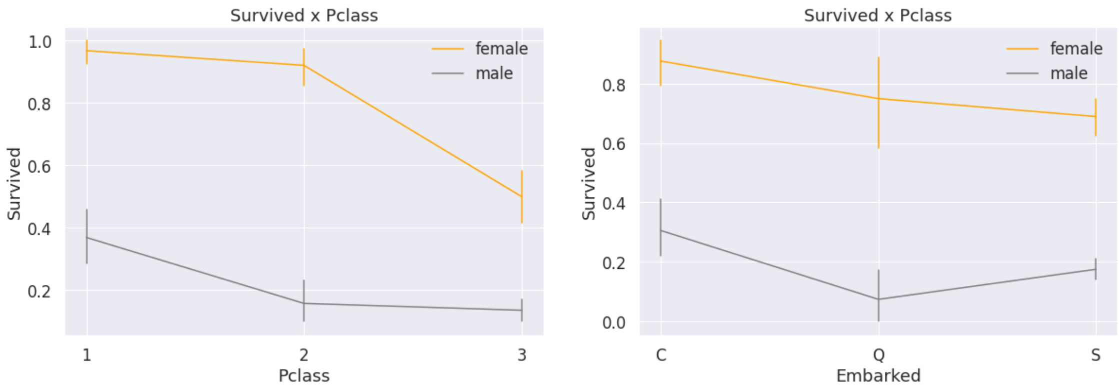


III - Relação da Sobrevivência com Classe Social e Porto de Embarque, por gênero

[10]:

```
sns.set(font_scale=1.5)
fem = train_df[train_df['Sex']=='female'] # Cria um dataframe para female
male = train_df[train_df['Sex']=='male'] # Cria um dataframe para male

fig = plt.figure(figsize=(20,6))

# Pclass
fig.add_subplot(1,2,1)
to_plot = sns.lineplot('Pclass','Survived',data=fem,err_style="bars",\
                       label='female', color='orange')
fig.add_subplot(1,2,1)
to_plot = sns.lineplot('Pclass','Survived',data=male,err_style="bars",\
                       label='male', color='grey')
to_plot.set_title('Survived x Pclass')
to_plot.set(xticks=(np.arange(1, 4, 1)))

# Embarked
fig.add_subplot(1,2,2)
to_plot = sns.lineplot('Embarked','Survived',data=fem,err_style="bars",\
                       label='female', color='orange')
fig.add_subplot(1,2,2)
to_plot = sns.lineplot('Embarked','Survived',data=male,err_style="bars",\
                       label='male', color='grey')
to_plot.set_title('Survived x Pclass')
```

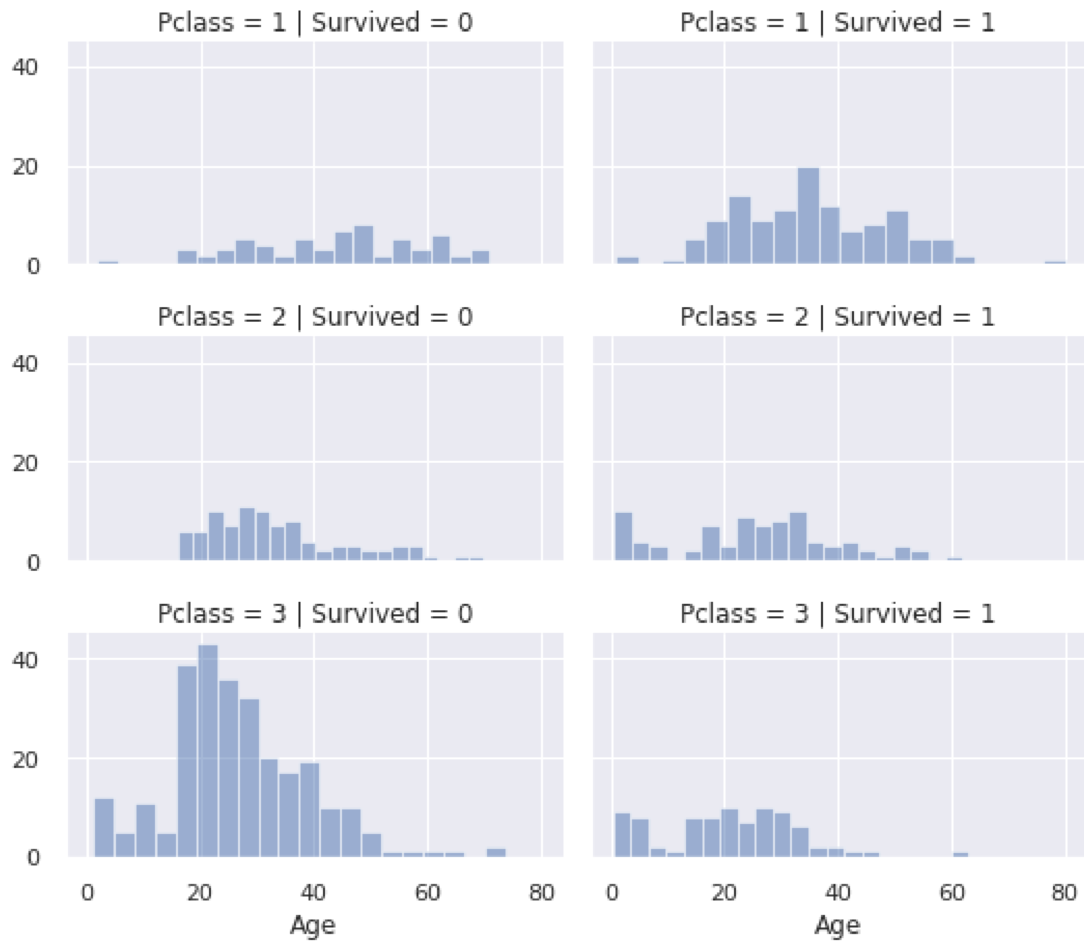
[10]: Text(0.5, 1.0, 'Survived x Pclass')



[11]:

```
sns.set(font_scale=1.0)
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass',\
                     size=2.2, aspect=1.7)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```

[11]: <seaborn.axisgrid.FacetGrid at 0x7f0c550c4390>



IV - Pré-Processamento dos dados

1. Imputation - Removendo colunas com menos de 70% de preenchimento e que possuem baixa ou nenhuma correlação com 'Survived'

[12]:
```
# Remove features como mais de 70% de linhas nulas
limite = 0.7

train_df = train_df[train_df.columns[train_df.isnull().mean() < limite]]

# Remove Passenger ID, devido a baixa correlação com 'Survived'
train_df = train_df.drop(['PassengerId'], axis = 1)

# Remove Ticket, devido a baixa correlação com 'Survived'
train_df = train_df.drop(['Ticket'], axis = 1)
```

1.2 - Imputation em Age

[13]:
```python
num_cols = ['Pclass', 'SibSp', 'Parch', 'Fare']

# Cria o objeto knn para efetuar a imputação de valores baseado
# no K-Nearest Neighbour (KNN)
knn = KNeighborsClassifier(3, weights='distance')

# Cria um data frame para treinamento do knn, excluindo os valores
# nulos
df_cc = train_df.dropna(axis=0)
df_cc['Age'] = df_cc['Age'].astype(int)

# Treina o modelo com 'Age' como 'target(y)'
model_3nn = knn.fit(df_cc.loc[:,num_cols],\
                    df_cc.loc[:,'Age'])
# Contabiliza a quantidade de nulos da feature 'Age'
missing_age = train_df['Age'].isnull()

# Cria um dataframe com as features 'X' (num_cols) com tamanho
# igual ao número de valores nulos na feature 'Age'
df_missing_age = pd.DataFrame(train_df[num_cols][missing_age])

# Faz a previsão passando as features 'X', do data frame criado
# na linha anterior
imputed_age = model_3nn.predict(df_missing_age)

# Preenche os valores nulos da feature 'Age', um por um, com os
# valores previstos em 'imputed_age'
for i in imputed_age:
    train_df['Age'].fillna(i, inplace=True, limit=1)
train_df['Age'].isnull().sum()
```

[13]: 0

1.3. Imputation em Embarked - Substituição pelo valor mais comum

[14]:
```python
# Demonstra que o valor mais comum é o 'S'
print(train_df['Embarked'].describe())
# Apenas dois valores nan
print("Quantidade de Nulos: ",train_df['Embarked'].isnull().sum())
```

```
count      889
unique       3
top          S
freq       644
Name: Embarked, dtype: object
Quantidade de Nulos:  2
```

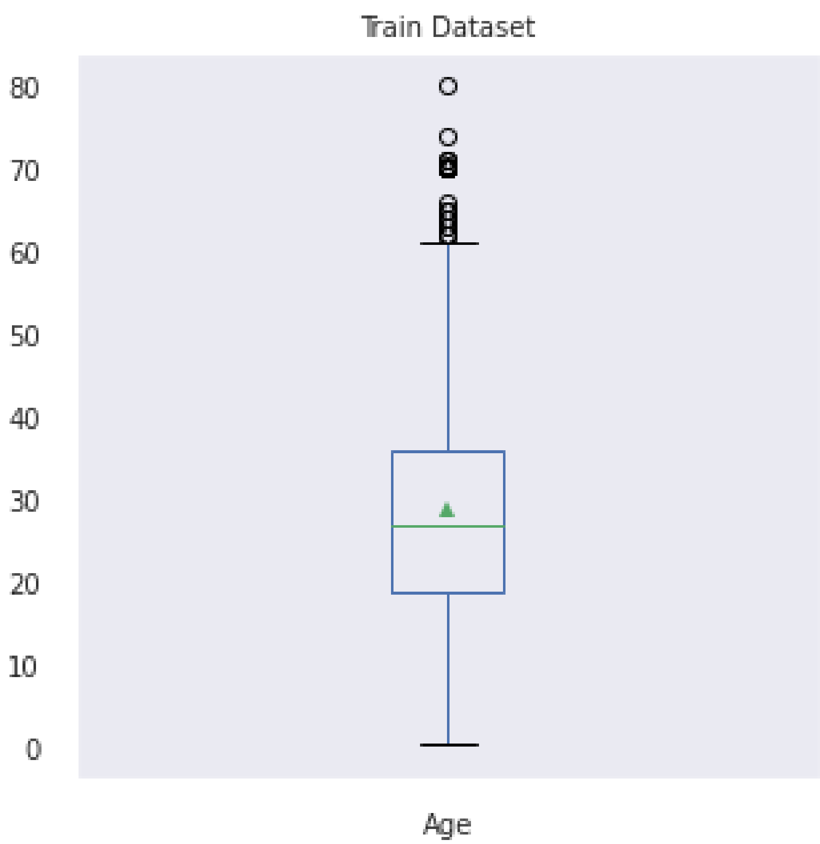
[15]:
```
train_df['Embarked'] = train_df['Embarked'].fillna('S')
print("Quantidade de Nulos: ",train_df['Embarked'].isnull().sum())
```

```
Quantidade de Nulos:  0
```

2. Outliers - Aplicando na feature 'Age'

[16]:
```
fig = plt.figure(figsize=(5,5))

# fig.add_subplot(1,2,1)
to_plot = train_df.boxplot(column='Age', grid=False, showmeans=True)
to_plot.set_title('Train Dataset', fontsize=10)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
```

[16]:
```
(array([-10.,   0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90.]),
 <a list of 11 Text major ticklabel objects>)
```



[17]:
```
# Limites superior e inferior dos dataframes de treinamento e teste
def get_lim(df):
    Q1 = df['Age'].quantile(0.25)
```

```python
    Q3 = df['Age'].quantile(0.75)
    lim_sup = Q3+(1.5*(Q3-Q1))
    lim_inf = Q1-(1.5*(Q3-Q1))
    print("quantile[0]:", train_df['Age'].quantile(0))
    if lim_inf < train_df['Age'].quantile(0):
        lim_inf = train_df['Age'].quantile(0)

    return lim_sup, lim_inf

lim_sup_train, lim_inf_train = get_lim(train_df)
print(lim_sup_train, lim_inf_train)
```

```
quantile[0]: 0.42
61.5 0.42
```

[18]:
```python
for i in train_df['Age']:
    if i < 0:
        print(i)
```

[19]:
```python
# Após definidos os limites, os outliers são removidos
train_df['Age'] = train_df['Age'].astype(int)
train_df = train_df[ (train_df['Age']<lim_sup_train) &\
                    (train_df['Age']>lim_inf_train) ]
```

3. Binning - Aplicando binning em 'Age' e 'Fare'

[20]:
```python
# Lista com os labels
list_bin=[0, 1, 2, 3, 4, 5, 6]
# Função para aplicar o binning em Age e Fare, pois são os valores
# que têm maior variação
def set_binning(feat):
    name_feat='bin_'+feat
    train_df[name_feat] = pd.qcut(train_df[feat], q= 7, \
                                  labels=list_bin)#, duplicates='drop')
set_binning('Age')
set_binning('Fare')
```

4. Feature Split - Aplicação em 'Name'

[21]:
```python
# Verifica os titulos existentes
train_df['Name'].str.split(" ").map(lambda x: x[1]).unique()
```

[21]:
```
array(['Mr.', 'Mrs.', 'Miss.', 'Master.', 'Planke,', 'Don.', 'Rev.',
       'Billiard,', 'der', 'Walle,', 'Dr.', 'Pelsmaeker,', 'Mulder,', 'y',
       'Steen,', 'Carlo,', 'Mme.', 'Impe,', 'Ms.', 'Major.', 'Gordon,',
       'Messemaeker,', 'Mlle.', 'Col.', 'Velde,', 'the', 'Shawah,',
       'Jonkheer.', 'Melkebeke,', 'Cruyssen,'], dtype=object)
```

[22]:
```
# Encontrando os títulos
train_df['Title'] = train_df['Name'].str.split(" ").map(lambda x: x[1])
# Substituindo
train_df['Title'] = train_df['Title'].replace(['Planke,', 'Don.', 'Rev.',\
            'Billiard,', 'der', 'Walle,', 'Dr.', 'Pelsmaeker,',\
            'Mulder,', 'y','Steen,', 'Carlo,', 'Mme.', 'Impe,',\
            'Ms.', 'Major.', 'Gordon,','Messemaeker,', 'Mlle.',\
            'Col.', 'Velde,', 'the', 'Shawah,','Jonkheer.',\
            'Melkebeke,', 'Cruyssen,'], 'Other')
# trata campos nulos
train_df['Title'] = train_df['Title'].fillna('Other')

# Remove a feature 'Name' após a criação de 'Title'
train_df = train_df.drop(['Name'], axis=1)
```

4. One Hot Encoding - Aplicando nas features Sex, Name e Embarked

[23]:
```
# Aplicando One-Hot Encoding na feature Title, pois ela é categórica
enc_col = pd.get_dummies(train_df['Title'])
train_df = train_df.join(enc_col).drop('Title', axis=1)
```

[24]:
```
# Aplicando One-Hot Encoding na feature Sex, pois ela é categórica
enc_col = pd.get_dummies(train_df['Sex'])
train_df = train_df.join(enc_col).drop('Sex', axis=1)
```

[25]:
```
# Aplicando One-Hot Encoding na feature Embarked, pois ela é categórica
enc_col = pd.get_dummies(train_df['Embarked'])
train_df = train_df.join(enc_col).drop('Embarked', axis=1)
```

5. Convertendo para int

[26]:
```
# Antes de converter Fare para int, preenche os nulos com a média
mean = train_df['Fare'].mean()
train_df['Fare'] = train_df['Fare'].fillna(mean)
train_df['Fare'] = train_df['Fare'].astype(int)

train_df['bin_Age'] = train_df['bin_Age'].astype(int)

train_df['bin_Fare'] = train_df['bin_Fare'].astype(int)
```

6. Normalization

[27]:
```
# Aplicando Normalization nas features com valores maiores que 1
feat_list = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'bin_Age',\
             'bin_Fare']
def normalize(feat, train_df):
    feat_name = 'norm_'+feat
    train_df[feat_name] = (train_df[feat] - train_df[feat].min())/ \
```

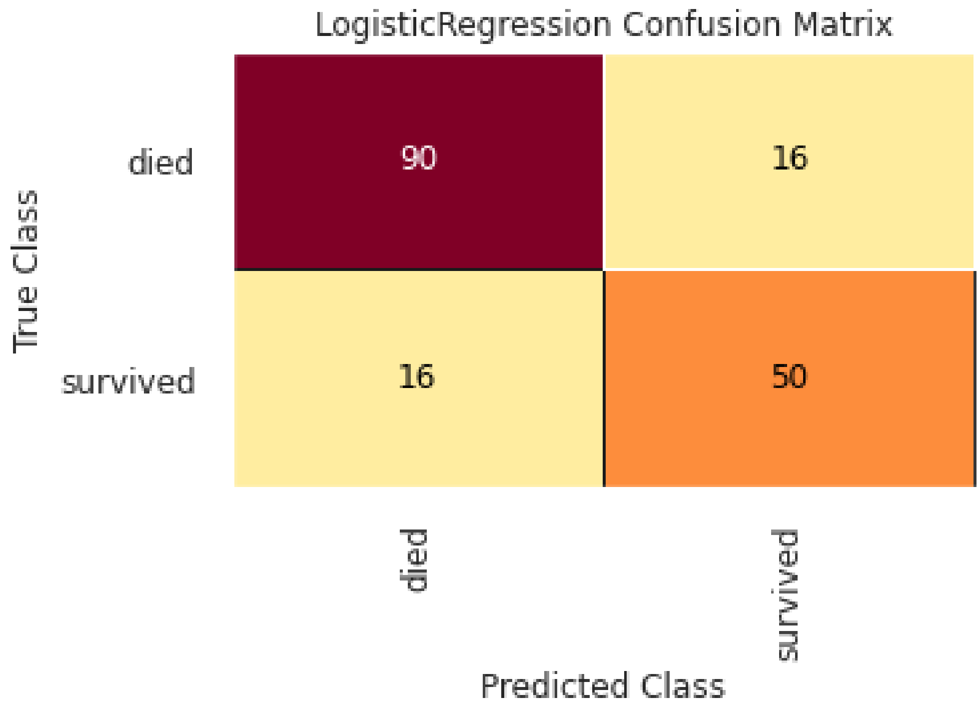
```
                    (train_df[feat].max() - train_df[feat].min())
    train_df = train_df.drop(feat, axis=1)
    return train_df
for i in feat_list:
    train_df=normalize(i, train_df)
```

V - Machine Learning Models

[28]:
```
X = train_df.loc[:, train_df.columns != 'Survived'] # X recebe todas as␣
↪colunas, exceto a coluna 'quality'
y = train_df.loc[:, train_df.columns == 'Survived'] # y recebe a coluna␣
↪'quality'

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,\
                                    shuffle = True, stratify = y)

model_list = ['Regressão Logística', 'Classificação Bayseana',\
              'Árvore de Decisão', 'Random Forests', 'SVM']
resultados_acur = len(model_list)*[0]
map = {0:"died", 1:"survived"}
```

1. Regressão Logística

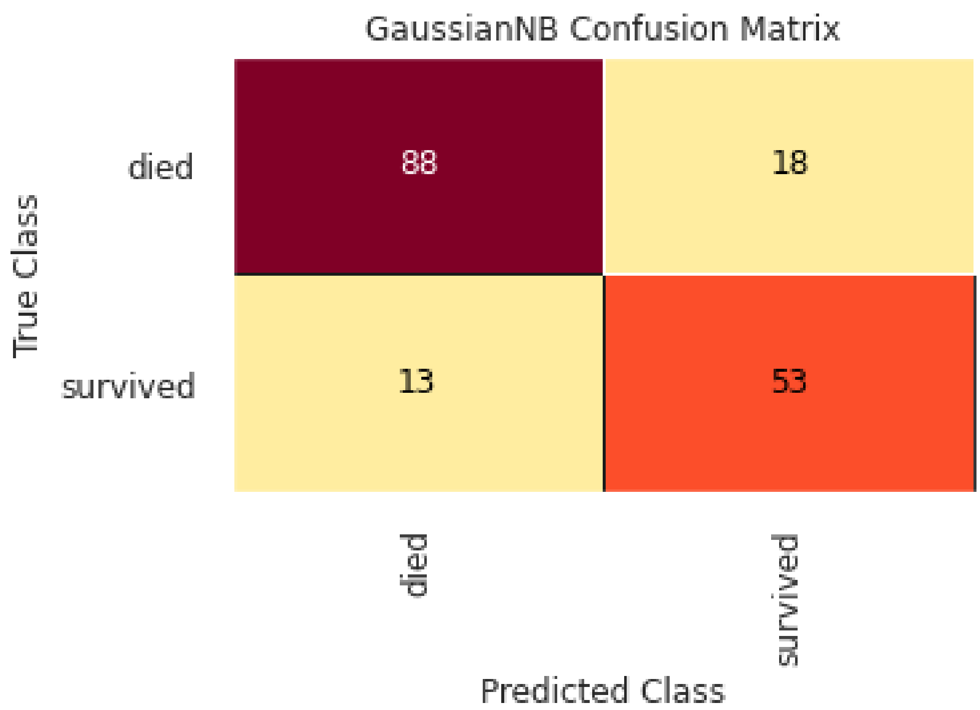
[29]:
```
plt.subplots(figsize=(5,3))
logreg = LogisticRegression()
logreg.fit(X_train, y_train)


cm_v = ConfusionMatrix(logreg, classes=["died", "survived"], \
                       label_encoder=map)
resultados_acur[0] = cm_v.score(X_test, y_test)
cm_v.show()
```




```
[29]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0c3ce96f98>
```

2. Classificação Bayseana

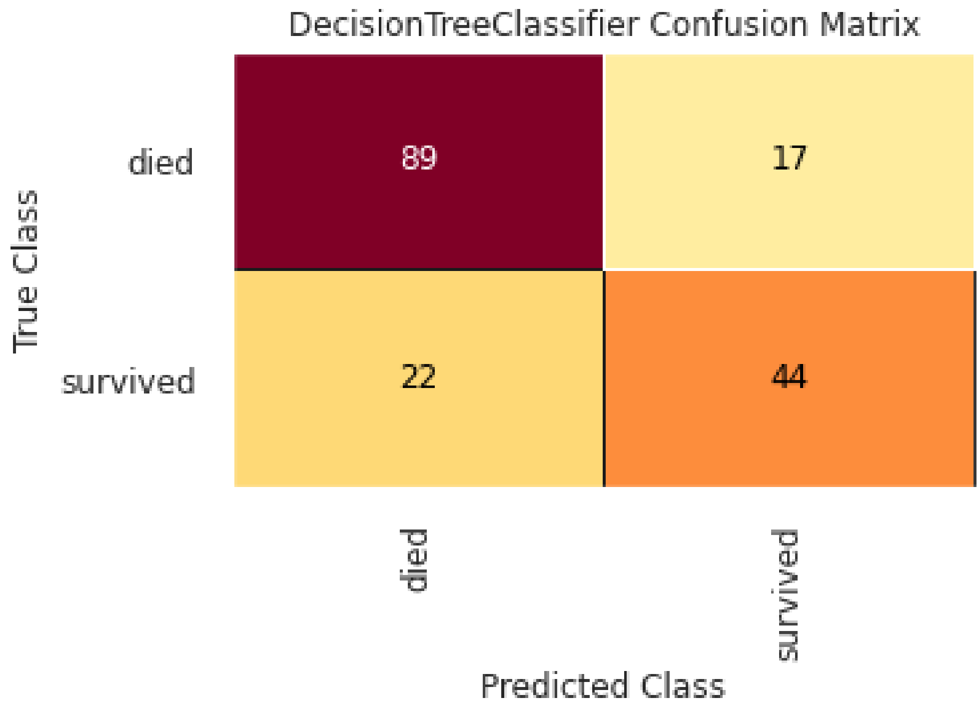
```
[30]: plt.subplots(figsize=(5,3))
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)

cm_v = ConfusionMatrix(gaussian, classes=["died", "survived"], \
                        label_encoder=map)
resultados_acur[1] = cm_v.score(X_test, y_test)
cm_v.show()
```



[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0c550c4d30>

3. Árvore de Decisão

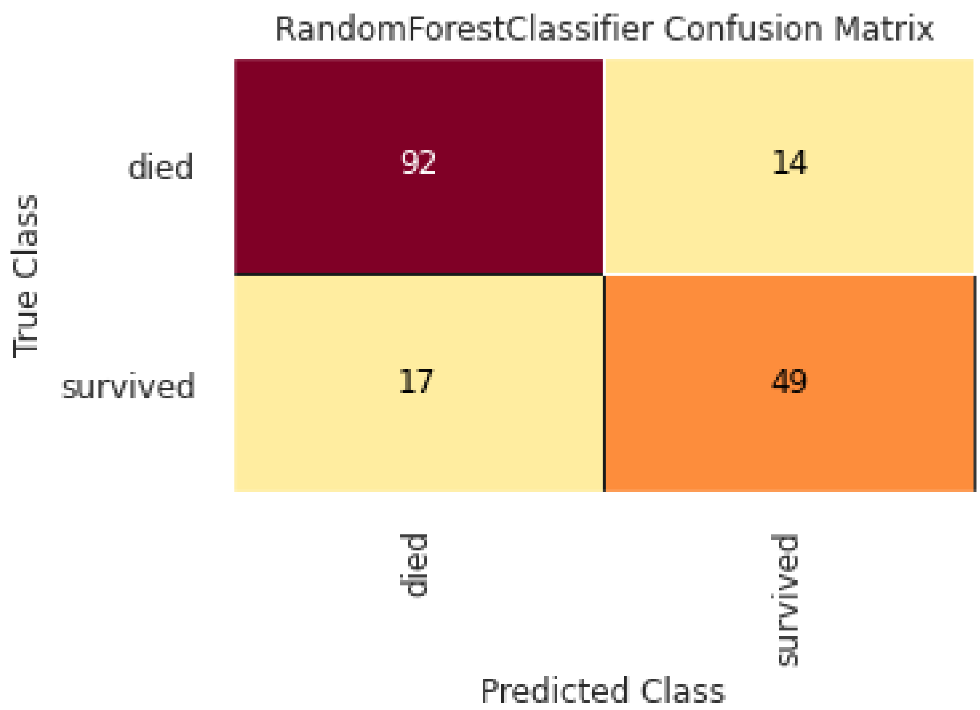
[31]:
```
plt.subplots(figsize=(5,3))
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)

cm_v = ConfusionMatrix(decision_tree, classes=["died", "survived"], \
                       label_encoder=map)
resultados_acur[2] = cm_v.score(X_test, y_test)
cm_v.show()
```



[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0c55043d30>

4. Random Forests

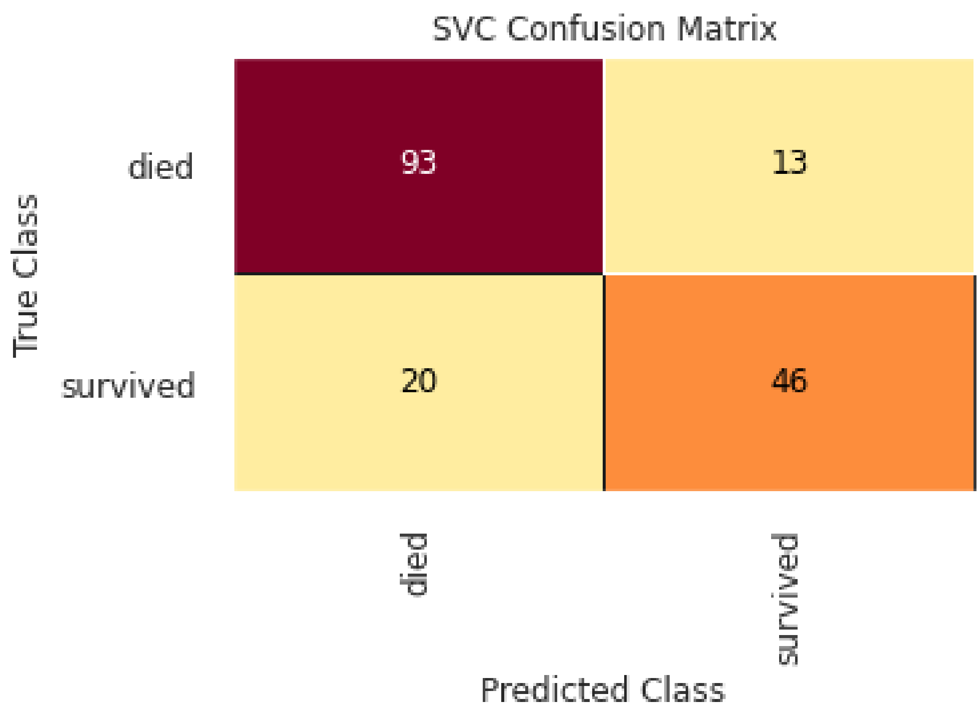
[32]:
```
plt.subplots(figsize=(5,3))
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)

cm_v = ConfusionMatrix(random_forest, classes=["died", "survived"], \
                       label_encoder=map)
resultados_acur[3] = cm_v.score(X_test, y_test)
cm_v.show()
```



[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0c3cce8860>

5. Support Vector Machine

[33]:
```
plt.subplots(figsize=(5,3))
svm_model = svm.SVC()#kernel='poly', gamma = 10, degree = 3)
svm_model.fit(X_train, y_train)

cm_v = ConfusionMatrix(svm_model, classes=["died", "survived"], \
                          label_encoder=map)
resultados_acur[4] = cm_v.score(X_test, y_test)
cm_v.show()
```



[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0c3ccb6748>

6. Acurácia por Modelo

[34]:
```
for i in range(len(resultados_acur)):
    print(model_list[i], ":", "{:.2f}".format(resultados_acur[i]*100),"%")
```

```
Regressão Logística : 81.40 %
Classificação Bayseana : 81.98 %
Árvore de Decisão : 77.33 %
Random Forests : 81.98 %
SVM : 80.81 %
```

[ ]:
```
```