03_RegressoesTitanic_v1.0-PauloBraga

June 29, 2020

[1]:
```
''' Paulo Simplício Braga
    29.06.2020
'''
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
import warnings
warnings.filterwarnings('ignore')

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

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

[1]:
```
<pandas.io.formats.style.Styler at 0x7fd8a4151e10>
```

I - Cálculo da % de valores faltantes por coluna:

[2]:
```
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
```

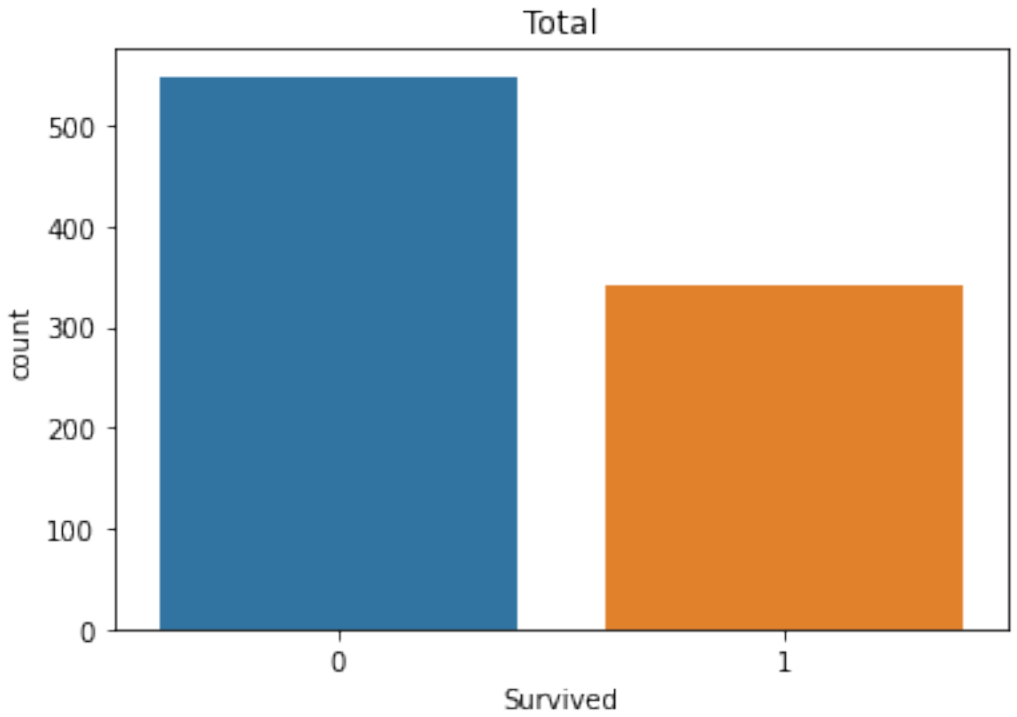
II - Sobreviventes por sexo

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

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

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

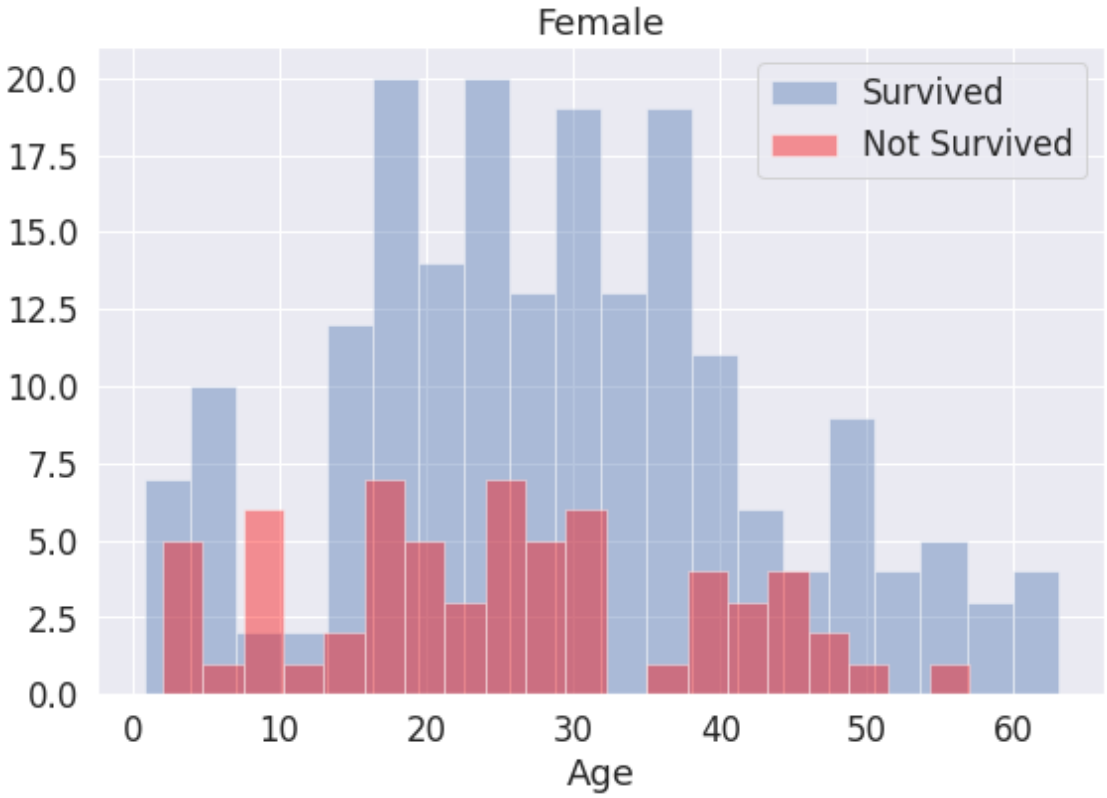
[7]:
```
Text(0.5, 1.0, 'Total')
```



[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 0x7fd866550b00>



[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()
```

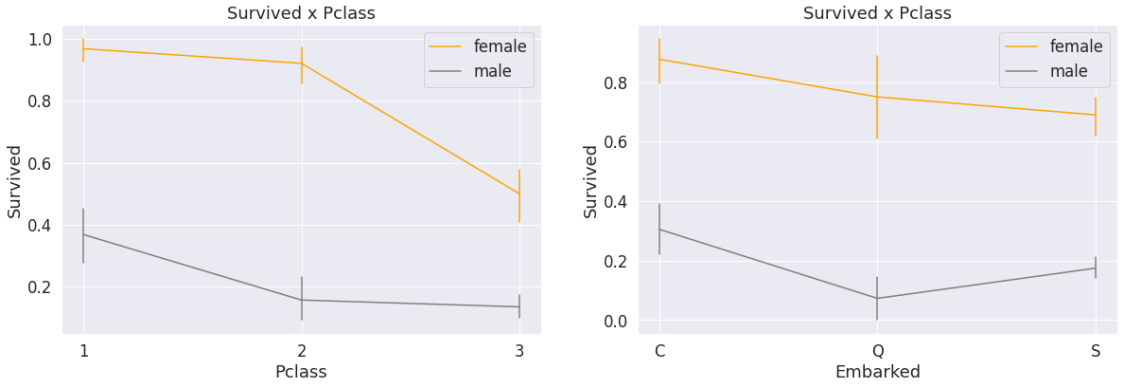
[9]: <matplotlib.legend.Legend at 0x7fd8662682b0>

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
```

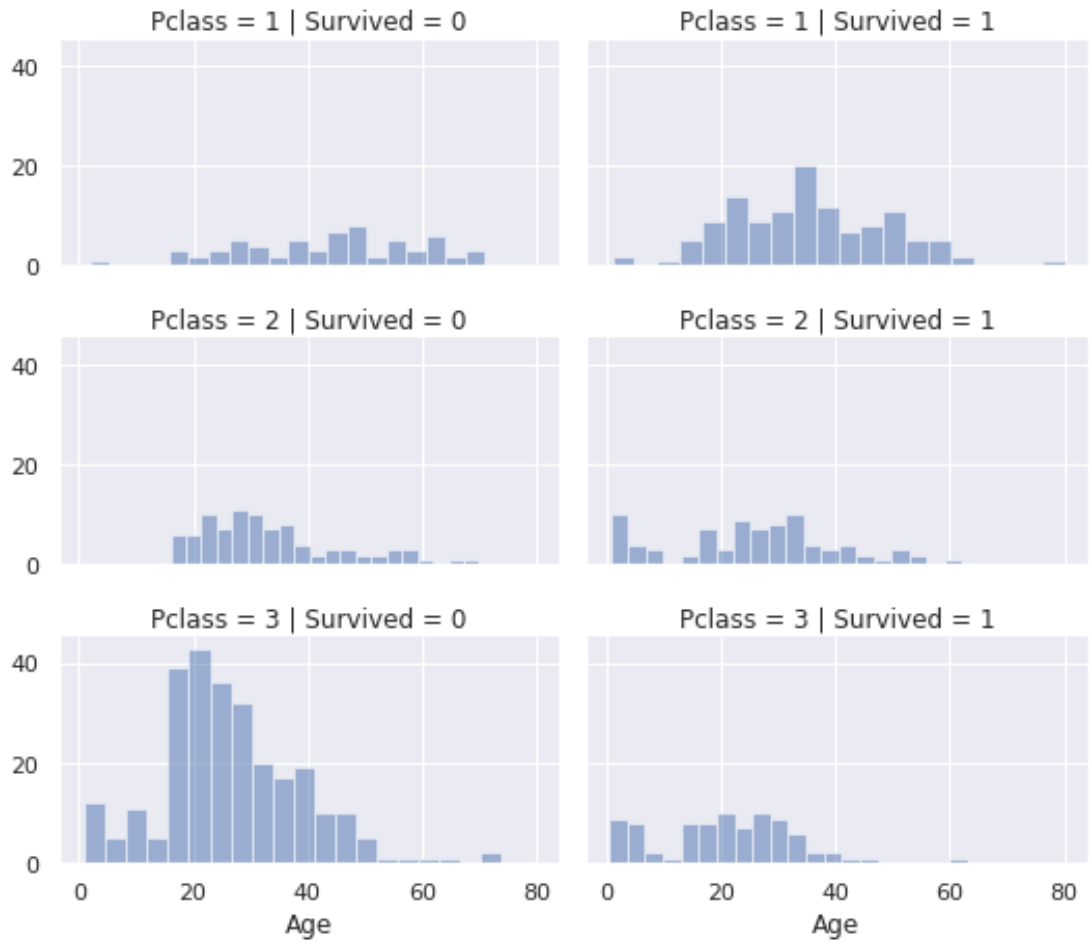
```
fig.add_subplot(1,2,1)
to_plot = sns.lineplot('Pclass','Survived',data=fem,err_style="bars",\
                       label='female', color='orange')
fig.add_subplot(1,2,1)
to_plot = sns.lineplot('Pclass','Survived',data=male,err_style="bars",\
                       label='male', color='grey')
to_plot.set_title('Survived 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 0x7fd8662dbb00>
```



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
data = [train_df, test_df]
for i in range(len(data)):
    data[i] = data[i][data[i].columns[data[i].isnull().mean() < limite]]
train_df = data[0]
test_df = data[1]

# Remove Passenger ID, devido a baixa correlação com 'Survived'
for i in range(len(data)):
    data[i] = data[i].drop(['PassengerId'], axis = 1)
train_df = data[0]
test_df = data[1]
```

```
# Remove Ticket, devido a baixa correlação com 'Survived'
for i in range(len(data)):
    data[i] = data[i].drop(['Ticket'], axis = 1)
train_df = data[0]
test_df = data[1]
```

1.2 - Imputation em Age - Preenchimento dos valores nulos baseado na distribuição percentual em cada faixa etária de 10 em 10 anos

[13]:
```
# Matriz com os data frames
data = [train_df, test_df]

# Cálculo da média por faixa etária de 10 em 10 anos
# Essa função faz a distribuição por igual, levando em conta a
# proporcionalidades por faixa etária
def get_mean(data_frame, size=8, null=0, not_null=0):
    count=j=i=0
    step = 10
    mean = [0]*size
    real = [0]*size
    total = 0
    for data in range(size):
        for data_1 in data_frame['Age']:
            if data_1>step-10 and data_1<=step:
                i+=data_1
                count+=1
        mean[j]=int(i/count)
        percent_total_ds=int(count*100/not_null)
        real[j]=int(percent_total_ds*null/100)
        j+=1
        step+=10
        i=count=0

    return mean, real
# train dataset
null = data[0]['Age'].isnull().sum()
not_null= data[0]['Age'].notnull().sum()
print("Nulo:", null)
print("Não nulo:", not_null)
train_mean, train_real = get_mean(data[0], 8, null, not_null)
print(train_mean)
print(train_real)
soma=0
for i in train_real:
    soma+=i
print(soma)
```

```
# test dataset
null = data[1]['Age'].isnull().sum()
not_null= data[1]['Age'].notnull().sum()
print("Nulo:", null)
print("Não nulo:", not_null)
test_mean, test_real = get_mean(data[1], 7, null, not_null)
print(test_mean)
print(test_real)
soma=0
for i in test_real:
    soma+=i
print(soma)
```

```
Nulo: 177
Não nulo: 714
[4, 17, 25, 35, 45, 54, 63, 73]
[14, 28, 56, 37, 21, 8, 3, 0]
167
Nulo: 86
Não nulo: 332
[4, 17, 25, 35, 45, 55, 62]
[5, 12, 33, 13, 11, 5, 2]
81
```

[14]:
```
# Aplicando o fillna de acordo com os valores médios e proporcionais
# obtidos anteriormente
def fill_nan(data_frame,size=7, mean='', real=''):
    for i in range(size):
        data_frame["Age"].fillna(mean[i], inplace=True,\
                                 limit=real[i])

    # Distribui para os valores restantes
    i=0
    while(data_frame['Age'].isnull().sum()):
        if i < size:
            data_frame["Age"].fillna(mean[i], inplace=True, limit=1)
        else:
            i=0
fill_nan(data[0], 7, train_mean, train_real)
fill_nan(data[1], 7, test_mean, test_real)
train_df["Age"].isnull().sum()
count=0
for i in train_df['Age']:
    if i == 73:
        count+=1
```

1.3. Imputation em Embarked - Substituição pelo valor mais comum

[15]:
```
# 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())
```

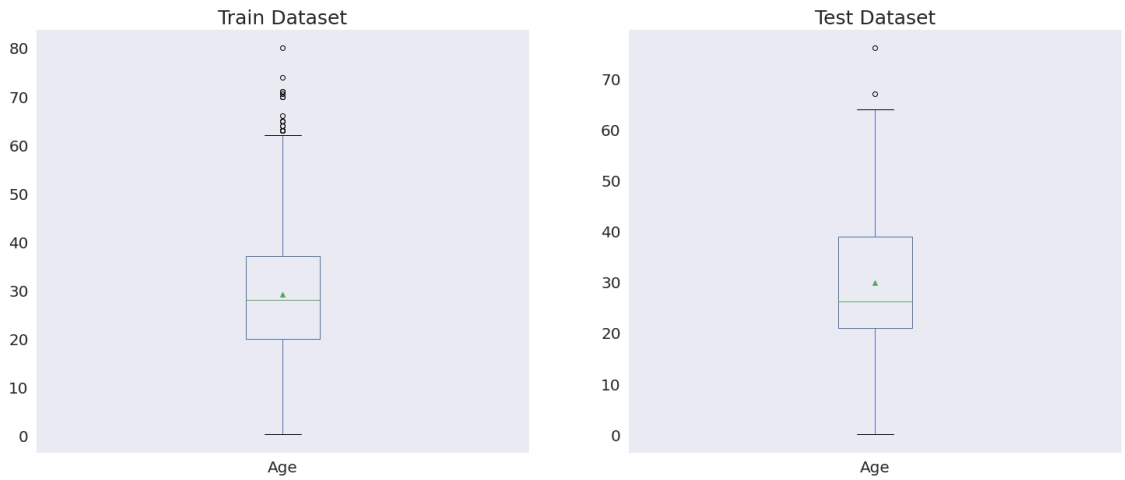
```
count     889
unique      3
top         S
freq      644
Name: Embarked, dtype: object
Quantidade de Nulos:  2
```

[16]:
```
# Como a quantidade de nulos é igual a 2, os substituo por 'S', pois é a
# ocorrência mais comum
data = [train_df, test_df]
for df in data:
    df['Embarked'] = df['Embarked'].fillna('S')
```

2. Outliers - Aplicando na feature 'Age'

[17]:
```
fig = plt.figure(figsize=(25,10))

fig.add_subplot(1,2,1)
to_plot = train_df.boxplot(column='Age', grid=False, showmeans=True)
to_plot.set_title('Train Dataset', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

fig.add_subplot(1,2,2)
to_plot = test_df.boxplot(column='Age', grid=False, showmeans=True)
to_plot.set_title('Test Dataset', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
```

[17]:
```
(array([-10.,   0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.]),
 <a list of 10 Text major ticklabel objects>)
```



[18]:
```
data = [train_df, test_df]

# Limites superior e inferior dos dataframes de treinamento e teste
def get_lim(df):
    Q1 = df['Age'].quantile(0.25)
    Q3 = df['Age'].quantile(0.75)
    lim_sup = Q3+(1.5*(Q3-Q1))
    lim_inf = Q1-(1.5*(Q3-Q1))
    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(data[0])
print(lim_sup_train, lim_inf_train)
lim_sup_test, lim_inf_test = get_lim(data[1])
print(lim_sup_test, lim_inf_test)
```

```
62.5 0.42
66.0 0.42
```

[19]:
```
# 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) ]

test_df['Age'] = test_df['Age'].astype(int)
test_df = test_df[(test_df['Age']<lim_sup_test) &\
                    (test_df['Age']>lim_inf_test)]
```

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

[20]:
```
data = [train_df, test_df]
# 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
    for df in data:
        df[name_feat] = pd.qcut(df[feat], q= 7, labels=list_bin)
set_binning('Age')
set_binning('Fare')
```

4. Feature Split - Aplicação em 'Name'

[21]:
```
# 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]:
```
data = [train_df, test_df]
# Criando a feature Title para extrair partes relevantes da
# feature Name

for df in data:
    # Encontrando os títulos
    df['Title'] = train_df['Name'].str.split(" ").map(lambda x: x[1])
    # Substituindo
    df['Title'] = 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
    df['Title'] = df['Title'].fillna('Other')

# Remove a feature 'Name' após a criação de 'Title'
train_df = train_df.drop(['Name'], axis=1)
test_df = test_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
data = [train_df, test_df]
for i in range(len(data)):
    enc_col = pd.get_dummies(data[i]['Title'])
    data[i] = data[i].join(enc_col).drop('Title', axis=1)

train_df=data[0]
test_df=data[1]
```

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

train_df=data[0]
test_df=data[1]
```

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

train_df=data[0]
test_df=data[1]
```

5. Convertendo para int

[26]:
```
data = [train_df, test_df]
# Antes de converter Fare para int, preenche os nulos com a média
for dataset in data:
    mean = dataset['Fare'].mean()
    dataset['Fare'] = dataset['Fare'].fillna(mean)
    dataset['Fare'] = dataset['Fare'].astype(int)
for dataset in data:
    dataset['bin_Age'] = dataset['bin_Age'].astype(int)
for dataset in data:
    dataset['bin_Fare'] = dataset['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']
data = [train_df, test_df]
def normalize(feat):
    feat_name = 'norm_'+feat
```

```
    for i in range(len(data)):
        data[i][feat_name] = (data[i][feat] - data[i][feat].min())/ \
                        (data[i][feat].max() - data[i][feat].min())
        data[i] = data[i].drop(feat, axis=1)
for i in feat_list:
    normalize(i)
train_df=data[0]
test_df=data[1]
```

V - Machine Learning Models

[28]:
```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]

X_test  = test_df

resultados = 4*[0]
model_list = ['Regressão Logística', 'Classificação Bayseana',\
              'Árvore de Decisão', 'Random Forests']
```

1. Regressão Logística

[29]:
```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc = round(logreg.score(X_train, Y_train) * 100, 2)
print(round(acc,2,), "%")
resultados[0]=acc
```

```
82.68 %
```

2. Classificação Bayseana

[30]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)

Y_pred = gaussian.predict(X_test)

acc = round(gaussian.score(X_train, Y_train) * 100, 2)
print(round(acc,2,), "%")
resultados[1]=acc
```

```
81.29 %
```

3. Árvore de Decisão

[31]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

Y_pred = decision_tree.predict(X_test)

acc = round(decision_tree.score(X_train, Y_train) * 100, 2)
print(round(acc,2,), "%")
resultados[2]=acc
```

```
97.58 %
```

4. Random Forests

[32]:
```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

acc = round(random_forest.score(X_train, Y_train) * 100, 2)
print(round(acc,2,), "%")
resultados[3]=acc
```

```
97.58 %
```

5. Resultados por Modelo

[33]:
```
for i in range(len(resultados)):
    print(model_list[i], ":", resultados[i],"%")
```

```
Regressão Logística : 82.68 %
Classificação Bayseana : 81.29 %
Árvore de Decisão : 97.58 %
Random Forests : 97.58 %
```