Classificiação de vinhos brancos sem tratamento de dados, hot-encoding, redimensionando qualidades para alta média e baixa

In [1]:
```
#Desabilita logs e mantém apenas logs críticos (para evitar o libcuda ficar me avisando qu
import logging
logger = logging.getLogger()
logger.setLevel(logging.CRITICAL)
```

In [2]:
```
%config Completer.use_jedi = False
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import scipy as spy
import keras
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense,Dropout,Input
from keras.optimizers import Adam,RMSprop
```

```
2021-08-17 21:17:19.601504: I tensorflow/stream_executor/platform/default/dso_loader.cc:5
3] Successfully opened dynamic library libcudart.so.11.0
```

In [3]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
df = pd.read_csv('../datasets/winequality-white.csv', sep = ',')
df.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Olhando abaixo, não temos nenhum valor N/A, então não precisamos tratar isso.

In [4]:
```
df.isna().sum()
```

Out[4]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
```

```
pH                    0
sulphates             0
alcohol               0
quality               0
dtype: int64
```

In [5]:
```
print(len(df))
```

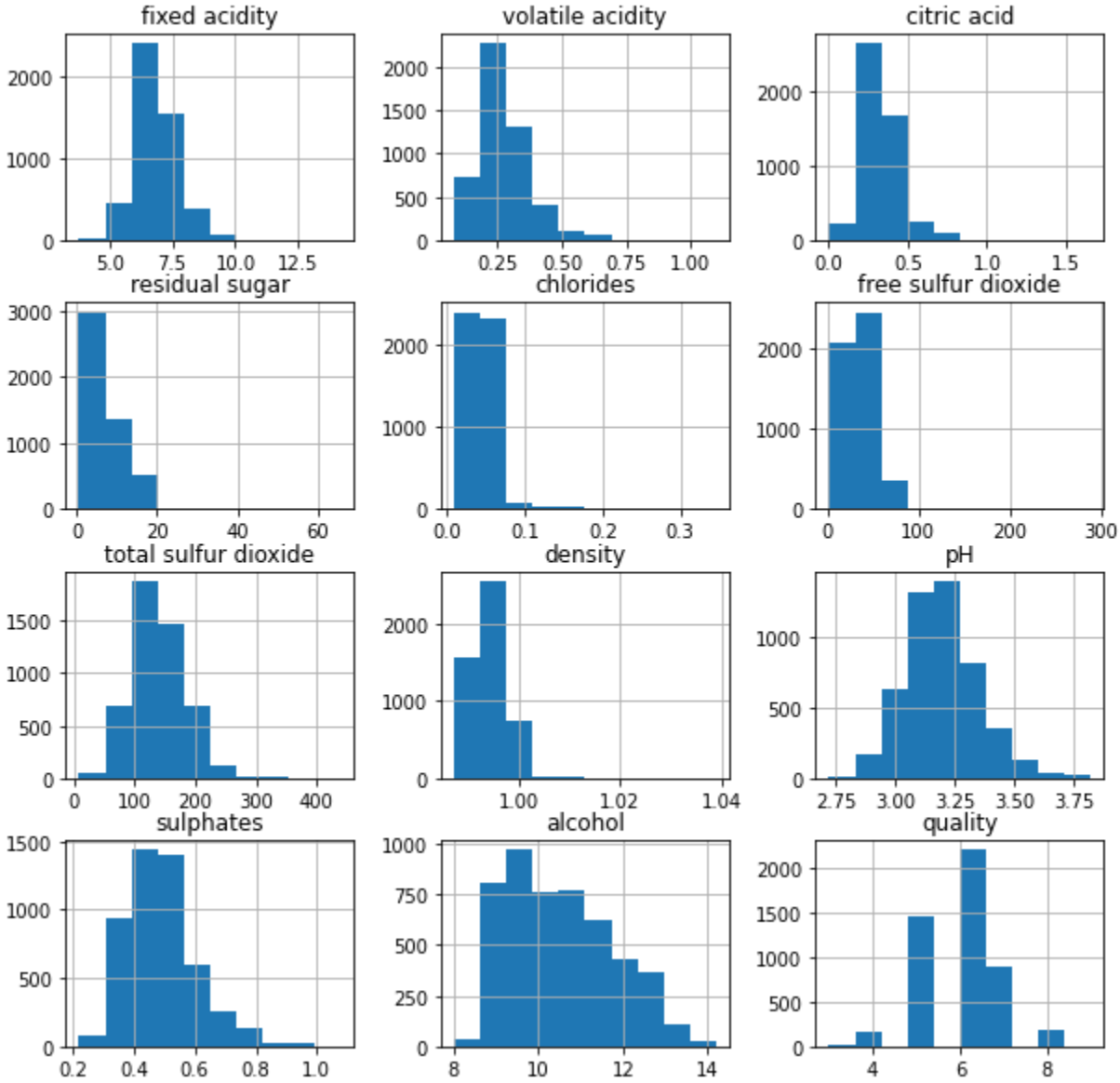
```
4898
```

In [6]:
```
df['quality'].value_counts()
```

Out[6]:
```
6    2198
5    1457
7     880
8     175
4     163
3      20
9       5
Name: quality, dtype: int64
```

In [7]:
```
df.hist(figsize = (10, 10))
```

Out[7]:
```
array([[<AxesSubplot:title={'center':'fixed acidity'}>,
        <AxesSubplot:title={'center':'volatile acidity'}>,
        <AxesSubplot:title={'center':'citric acid'}>],
       [<AxesSubplot:title={'center':'residual sugar'}>,
        <AxesSubplot:title={'center':'chlorides'}>,
        <AxesSubplot:title={'center':'free sulfur dioxide'}>],
       [<AxesSubplot:title={'center':'total sulfur dioxide'}>,
        <AxesSubplot:title={'center':'density'}>,
        <AxesSubplot:title={'center':'pH'}>],
       [<AxesSubplot:title={'center':'sulphates'}>,
        <AxesSubplot:title={'center':'alcohol'}>,
        <AxesSubplot:title={'center':'quality'}>]], dtype=object)
```



In [8]:
```
X=df.drop(columns=['quality'])
y=df['quality']
```

In [9]:
```
y[y<=4] = 0
y[((y>=5) & (y<=7))] = 1
y[y>=8] = 2
y.value_counts()
```

```
/tmp/ipykernel_1447/1882551270.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  y[y<=4] = 0
/tmp/ipykernel_1447/1882551270.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
ide/indexing.html#returning-a-view-versus-a-copy
  y[((y>=5) & (y<=7))] = 1
/tmp/ipykernel_1447/1882551270.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_gu
```

```
ide/indexing.html#returning-a-view-versus-a-copy
  y[y>=8] = 2
```

Out[9]:
```
1    4535
0     183
2     180
Name: quality, dtype: int64
```

Substituindo as faixas de qualidade por 0 (baixa), 1 (média) e 2 (alta):

In [10]:
```
df = pd.concat([X, y.reindex(X.index)], axis=1)
df.head()
```

Out[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	1
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	1

Pronto, agora temos todas as amostras em quantias iguais.

Hot encoding

Primeiro vamos separar a qualificações que vão de 6 à 9 (no caso desse dataset) em 0 1 e 2. Sendo 0 a qualidade mais baixa, 1 a média e 2 a alta, como vamos configurar para o hot encoding posteriormente.

No trecho de código abaixo, vou converter as varíaveis categóricas (0 1 e 2) em uma tabela. Essa tabela tem 3 colunas, onde cada uma corresponde à uma das classificações possíveis. Sempre apenas um dos items dessa coluna vai ser 1 e o resto 0. O nosso modelo ira tentar prever o valor dessas 3 colunas para assim prever a qualidade do vinho.

In [11]:
```
one_hot_encoded_data = pd.get_dummies(df, columns = ['quality'])
df = one_hot_encoded_data.rename(columns={'quality_0': 'baixa', 'quality_1': 'media', 'qua
df.head()
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	baixa	media
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	0	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	0	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	0	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	0	1
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	0	1

In [12]:
```
y = df[['baixa','media', 'alta']]
```

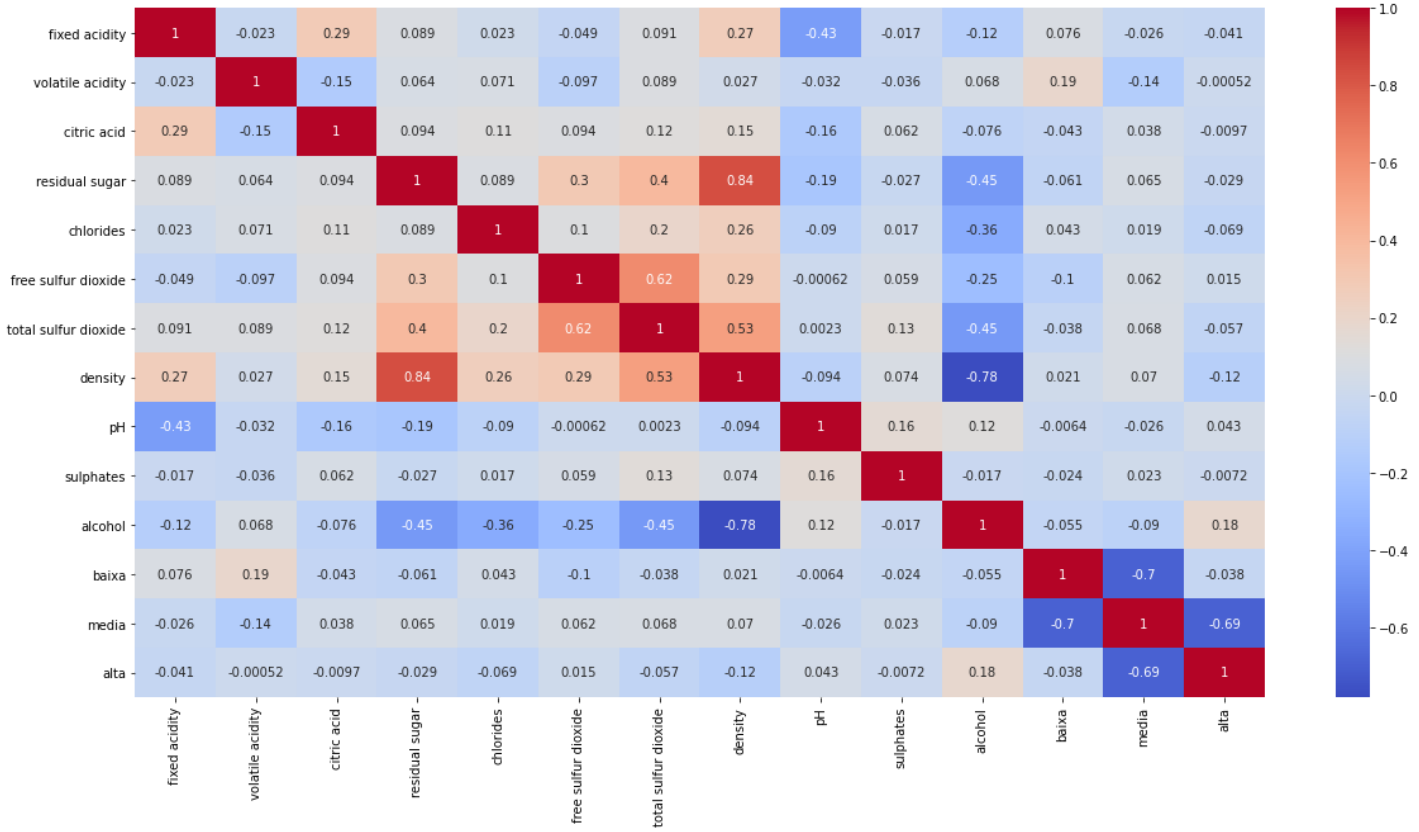
Matriz de correlação:

In [13]:
```
corr=df.corr()
plt.figure(figsize=(20,10))
```

```
sb.heatmap(corr,annot=True, cmap='coolwarm')
```

Out[13]:

```
<AxesSubplot:>
```



Separando o dataset de treinamento e o de predição

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20,random_state=21)
print('Formato do dataset de treinamento Xs:{}'.format(X_train.shape))
print('Formato do datasett de teste Xs:{}'.format(X_test.shape))
print('Formato do dataset de treino y:{}'.format(y_train.shape))
print('Formato do dataset de test y:{}'.format(y_test.shape))
```

```
Formato do dataset de treinamento Xs:(3918, 11)
Formato do datasett de teste Xs:(980, 11)
Formato do dataset de treino y:(3918, 3)
Formato do dataset de test y:(980, 3)
```

Construção do modelo

O artigo utilizou relu e tanh. Aqui abaixo vamos usar relu.

In [15]:

```
dimension = X_train.shape[1]
from keras import backend as K
def create_model():
    model = Sequential()
    model.add(Dense(10, input_dim = dimension,  activation='relu'))
    model.add(Dense(60, input_dim = dimension,  activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
model = create_model()
model.summary()
```

```
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                120
_________________________________________________________________
dense_1 (Dense)              (None, 60)                660
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 183
=================================================================
Total params: 963
Trainable params: 963
Non-trainable params: 0
_________________________________________________________________
2021-08-17 21:17:22.384352: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcuda.so.1
2021-08-17 21:17:22.424910: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2021-08-17 21:17:22.425315: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1733] Found device 0 with properties:
pciBusID: 0000:0c:00.0 name: NVIDIA GeForce GTX 660 Ti computeCapability: 3.0
coreClock: 1.0715GHz coreCount: 7 deviceMemorySize: 1.95GiB deviceMemoryBandwidth: 134.29GiB/s
2021-08-17 21:17:22.425657: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
2021-08-17 21:17:22.429353: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcublas.so.11
2021-08-17 21:17:22.429520: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcublasLt.so.11
2021-08-17 21:17:22.430755: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcufft.so.10
2021-08-17 21:17:22.430997: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcurand.so.10
2021-08-17 21:17:22.431066: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcusolver.so.11'; dlerror: libcusolver.so.11: cannot open shared object file: No such file or directory
2021-08-17 21:17:22.432079: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcusparse.so.11
2021-08-17 21:17:22.432237: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open shared object file: No such file or directory
2021-08-17 21:17:22.432251: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1766] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
2021-08-17 21:17:22.433676: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-08-17 21:17:22.433964: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1258] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-08-17 21:17:22.433973: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1264]
```

In [16]:

```
history=model.fit(X_train, y_train, validation_data=(X_test, y_test),epochs=50, batch_size
```

```
Epoch 1/50
2021-08-17 21:17:22.523059: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered 2)
2021-08-17 21:17:22.523470: I tensorflow/core/platform/profile_utils/cpu_utils.cc:114] CPU Frequency: 3593340000 Hz
392/392 [==============================] - 1s 1ms/step - loss: 0.3613 - accuracy: 0.9227 - val_loss: 0.2986 - val_accuracy: 0.9316
```

```
Epoch 2/50
392/392 [==============================] - 0s 787us/step - loss: 0.3311 - accuracy: 0.9234
- val_loss: 0.3019 - val_accuracy: 0.9316
Epoch 3/50
392/392 [==============================] - 0s 670us/step - loss: 0.3182 - accuracy: 0.9237
- val_loss: 0.3563 - val_accuracy: 0.9163
Epoch 4/50
392/392 [==============================] - 0s 553us/step - loss: 0.3186 - accuracy: 0.9234
- val_loss: 0.2936 - val_accuracy: 0.9316
Epoch 5/50
392/392 [==============================] - 0s 750us/step - loss: 0.3150 - accuracy: 0.9229
- val_loss: 0.3180 - val_accuracy: 0.9296
Epoch 6/50
392/392 [==============================] - 0s 637us/step - loss: 0.3091 - accuracy: 0.9234
- val_loss: 0.3095 - val_accuracy: 0.9286
Epoch 7/50
392/392 [==============================] - 0s 684us/step - loss: 0.3048 - accuracy: 0.9232
- val_loss: 0.2894 - val_accuracy: 0.9316
Epoch 8/50
392/392 [==============================] - 0s 730us/step - loss: 0.2962 - accuracy: 0.9234
- val_loss: 0.2695 - val_accuracy: 0.9316
Epoch 9/50
392/392 [==============================] - 0s 569us/step - loss: 0.2989 - accuracy: 0.9234
- val_loss: 0.2850 - val_accuracy: 0.9316
Epoch 10/50
392/392 [==============================] - 0s 562us/step - loss: 0.2953 - accuracy: 0.9245
- val_loss: 0.2895 - val_accuracy: 0.9316
Epoch 11/50
392/392 [==============================] - 0s 555us/step - loss: 0.2923 - accuracy: 0.9237
- val_loss: 0.2649 - val_accuracy: 0.9316
Epoch 12/50
392/392 [==============================] - 0s 588us/step - loss: 0.2953 - accuracy: 0.9247
- val_loss: 0.2721 - val_accuracy: 0.9316
Epoch 13/50
392/392 [==============================] - 0s 604us/step - loss: 0.2921 - accuracy: 0.9234
- val_loss: 0.2873 - val_accuracy: 0.9276
Epoch 14/50
392/392 [==============================] - 0s 574us/step - loss: 0.2902 - accuracy: 0.9247
- val_loss: 0.2839 - val_accuracy: 0.9316
Epoch 15/50
392/392 [==============================] - 0s 704us/step - loss: 0.2892 - accuracy: 0.9247
- val_loss: 0.3472 - val_accuracy: 0.9245
Epoch 16/50
392/392 [==============================] - 0s 636us/step - loss: 0.2887 - accuracy: 0.9247
- val_loss: 0.2751 - val_accuracy: 0.9316
Epoch 17/50
392/392 [==============================] - 0s 563us/step - loss: 0.2865 - accuracy: 0.9242
- val_loss: 0.2701 - val_accuracy: 0.9327
Epoch 18/50
392/392 [==============================] - 0s 561us/step - loss: 0.2848 - accuracy: 0.9252
- val_loss: 0.2799 - val_accuracy: 0.9327
Epoch 19/50
392/392 [==============================] - 0s 581us/step - loss: 0.2802 - accuracy: 0.9247
- val_loss: 0.2717 - val_accuracy: 0.9316
Epoch 20/50
392/392 [==============================] - 0s 624us/step - loss: 0.2834 - accuracy: 0.9245
- val_loss: 0.2701 - val_accuracy: 0.9296
Epoch 21/50
392/392 [==============================] - 0s 723us/step - loss: 0.2840 - accuracy: 0.9250
- val_loss: 0.2664 - val_accuracy: 0.9327
Epoch 22/50
392/392 [==============================] - 0s 603us/step - loss: 0.2811 - accuracy: 0.9252
- val_loss: 0.2629 - val_accuracy: 0.9316
Epoch 23/50
392/392 [==============================] - 0s 619us/step - loss: 0.2823 - accuracy: 0.9247
- val_loss: 0.2714 - val_accuracy: 0.9306
```

```
Epoch 24/50
392/392 [==============================] - 0s 773us/step - loss: 0.2804 - accuracy: 0.9245
- val_loss: 0.2669 - val_accuracy: 0.9327
Epoch 25/50
392/392 [==============================] - 0s 604us/step - loss: 0.2815 - accuracy: 0.9239
- val_loss: 0.2789 - val_accuracy: 0.9327
Epoch 26/50
392/392 [==============================] - 0s 668us/step - loss: 0.2781 - accuracy: 0.9247
- val_loss: 0.2607 - val_accuracy: 0.9327
Epoch 27/50
392/392 [==============================] - 0s 570us/step - loss: 0.2778 - accuracy: 0.9239
- val_loss: 0.2746 - val_accuracy: 0.9316
Epoch 28/50
392/392 [==============================] - 0s 749us/step - loss: 0.2743 - accuracy: 0.9247
- val_loss: 0.2594 - val_accuracy: 0.9327
Epoch 29/50
392/392 [==============================] - 0s 712us/step - loss: 0.2761 - accuracy: 0.9252
- val_loss: 0.2622 - val_accuracy: 0.9316
Epoch 30/50
392/392 [==============================] - 0s 612us/step - loss: 0.2744 - accuracy: 0.9237
- val_loss: 0.2776 - val_accuracy: 0.9286
Epoch 31/50
392/392 [==============================] - 0s 596us/step - loss: 0.2756 - accuracy: 0.9245
- val_loss: 0.2598 - val_accuracy: 0.9327
Epoch 32/50
392/392 [==============================] - 0s 637us/step - loss: 0.2726 - accuracy: 0.9250
- val_loss: 0.2604 - val_accuracy: 0.9316
Epoch 33/50
392/392 [==============================] - 0s 616us/step - loss: 0.2743 - accuracy: 0.9260
- val_loss: 0.2664 - val_accuracy: 0.9327
Epoch 34/50
392/392 [==============================] - 0s 561us/step - loss: 0.2737 - accuracy: 0.9247
- val_loss: 0.2617 - val_accuracy: 0.9327
Epoch 35/50
392/392 [==============================] - 0s 602us/step - loss: 0.2770 - accuracy: 0.9239
- val_loss: 0.2593 - val_accuracy: 0.9327
Epoch 36/50
392/392 [==============================] - 0s 538us/step - loss: 0.2750 - accuracy: 0.9245
- val_loss: 0.2584 - val_accuracy: 0.9337
Epoch 37/50
392/392 [==============================] - 0s 588us/step - loss: 0.2715 - accuracy: 0.9260
- val_loss: 0.2625 - val_accuracy: 0.9316
Epoch 38/50
392/392 [==============================] - 0s 760us/step - loss: 0.2722 - accuracy: 0.9250
- val_loss: 0.2572 - val_accuracy: 0.9327
Epoch 39/50
392/392 [==============================] - 0s 538us/step - loss: 0.2705 - accuracy: 0.9247
- val_loss: 0.2606 - val_accuracy: 0.9316
Epoch 40/50
392/392 [==============================] - 0s 545us/step - loss: 0.2708 - accuracy: 0.9242
- val_loss: 0.2754 - val_accuracy: 0.9327
Epoch 41/50
392/392 [==============================] - 0s 578us/step - loss: 0.2717 - accuracy: 0.9252
- val_loss: 0.2597 - val_accuracy: 0.9316
Epoch 42/50
392/392 [==============================] - 0s 570us/step - loss: 0.2699 - accuracy: 0.9247
- val_loss: 0.2559 - val_accuracy: 0.9316
Epoch 43/50
392/392 [==============================] - 0s 554us/step - loss: 0.2709 - accuracy: 0.9250
- val_loss: 0.2579 - val_accuracy: 0.9316
Epoch 44/50
392/392 [==============================] - 0s 606us/step - loss: 0.2717 - accuracy: 0.9239
- val_loss: 0.2594 - val_accuracy: 0.9327
Epoch 45/50
392/392 [==============================] - 0s 553us/step - loss: 0.2690 - accuracy: 0.9250
- val_loss: 0.2581 - val_accuracy: 0.9316
```

```
Epoch 46/50
392/392 [==============================] - 0s 549us/step - loss: 0.2658 - accuracy: 0.9239
- val_loss: 0.2521 - val_accuracy: 0.9327
Epoch 47/50
392/392 [==============================] - 0s 597us/step - loss: 0.2714 - accuracy: 0.9245
- val_loss: 0.2621 - val_accuracy: 0.9316
Epoch 48/50
392/392 [==============================] - 0s 712us/step - loss: 0.2659 - accuracy: 0.9252
- val_loss: 0.2669 - val_accuracy: 0.9327
Epoch 49/50
392/392 [==============================] - 0s 714us/step - loss: 0.2681 - accuracy: 0.9242
- val_loss: 0.2525 - val_accuracy: 0.9327
Epoch 50/50
392/392 [==============================] - 0s 622us/step - loss: 0.2671 - accuracy: 0.9255
- val_loss: 0.2542 - val_accuracy: 0.9327
```

Sobre val_accuracy e accuracy:

Quando ambos crescem na mesma proporção quer dizer que o modelo não causou nem overfitting nem underfitting.

Se o accuracy cresce mais que o val_accuracy, quer dizer que temos overfitting.

Se o accuracy cresce menos que o val_accuracy, quer dizer que temos underfitting.

avaliação do resultado:

In [17]:

```
y_pred = model.predict(X_test)
def max_probs(array):
    parsed_pred = np.empty((0,3))
    for idx, x in enumerate(array):
        idx_max = x.argmax()
        x = np.zeros((3,))
        x[idx_max] = 1
        array[idx] = x

max_probs(y_pred)
```

In [18]:

```
def to_category(array):
    categories = []
    for idx, x in enumerate(array):
        idx_max = x.argmax()
        x = 0
        if idx_max == 0: x = 0
        if idx_max == 1: x = 1
        if idx_max == 2: x = 2
        categories.append(x)
    return categories


categorical_y_pred = to_category(y_pred)
categorical_y_test = to_category(y_test.to_numpy())
data = confusion_matrix(categorical_y_test, categorical_y_pred)
```

In [19]:

```
len(categorical_y_test)
```

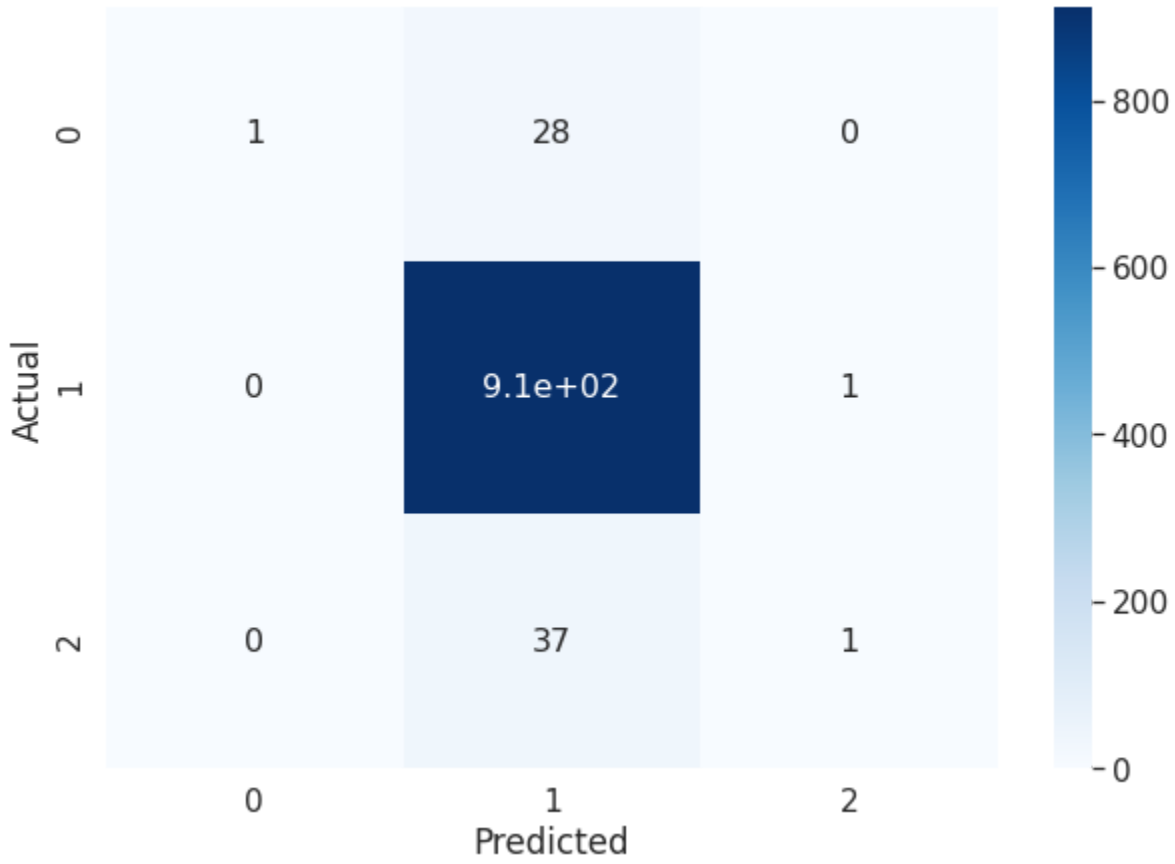
Out[19]:

```
980
```

Matriz de confusão:

In [20]:
```
df_cm = pd.DataFrame(data, columns=np.unique(categorical_y_test), index = np.unique(categ
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sb.set(font_scale=1.4)#for label size
sb.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})# font size
```

Out[20]:
```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```



Verificação de acurácia geral

In [21]:
```
correct = 0
total = 0
for i in range(len(categorical_y_test)):
    if(categorical_y_test[i] == categorical_y_pred[i]):
        correct += 1
    total += 1
accuracy = (correct/total)
```

In [22]:
```
accuracy
```

Out[22]:
```
0.9326530612244898
```

Conclusão

Sem o SMOTE, essa rede neural pode não ter informações suficientes de classes altas e baixas para fazer uma predição boa delas. Tivemos muitos vinhos que foram previstos como qualidade 1, que deveriam ser 2 ou 0.