

ANÁLISE DE DADOS DATASET "WINE QUALITY"

1. IMPORTANDO BIBLIOTECAS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import seaborn as sns
import pandas as pd
import math
from scipy.stats import skew
from tabulate import tabulate
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from scipy.stats import boxcox
from sklearn.preprocessing import PowerTransformer
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

2. ANÁLISE EXPLORATÓRIA

```
# Verify the contents of the directory
!ls "/content/drive/MyDrive/Homework/"

# Load the datasets
dt_whine_red = pd.read_csv("/content/drive/MyDrive/Homework/H2/h2_winequality-red.csv", sep=";")
```

H1 H2

```
dt_whine_red.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   fixed acidity          1599 non-null   float64
1   volatile acidity       1599 non-null   float64
2   citric acid            1599 non-null   float64
3   residual sugar         1599 non-null   float64
4   chlorides              1599 non-null   float64
5   free sulfur dioxide    1599 non-null   float64
6   total sulfur dioxide   1599 non-null   float64
7   density                1599 non-null   float64
8   pH                    1599 non-null   float64
9   sulphates              1599 non-null   float64
10  alcohol                1599 non-null   object
11  quality                1599 non-null   int64
dtypes: float64(10), int64(1), object(1)
memory usage: 150.0+ KB
```

```
# CONFERINDO INCONSISTÊNCIA ENTRE OS VALORES PARA PADRONIZAR OS TIPOS
# Observe que alguns valores estranhos e por isso a coluna está como Object
dt_whine_red['alcohol'].unique()
```

```
array(['9.4', '9.8', '10', '9.5', '10.5', '9.2', '9.9', '9.1', '9.3', '9',
      '9.7', '10.1', '10.6', '9.6', '10.8', '10.3', '13.1', '10.2',
      '10.9', '10.7', '12.9', '10.4', '13', '14', '11.5', '11.4', '12.4',
      '11', '12.2', '12.8', '12.6', '12.5', '11.7', '11.3', '12.3', '12',
      '11.9', '11.8', '8.7', '13.3', '11.2', '11.6', '11.1', '13.4',
      '12.1', '8.4', '12.7', '14.9', '13.2', '13.6', '13.5',
      '100.333.333.333.333', '9.55', '8.5', '110.666.666.666.667',
      '956.666.666.666.667', '10.55', '8.8', '135.666.666.666.667',
      '11.95', '9.95', '923.333.333.333.333', '9.25', '9.05', '10.75'],
      dtype=object)
```

```
# Lista com os valores a serem corrigidos e suas substituições
dict_whine_red = {
    '100.333.333.333.333': '100',
    '110.666.666.666.667': '110',
```

```
'956.666.666.666.667': '956',
'135.666.666.666.667': '135',
'923.333.333.333.333': '923'
}
```

```
# Substituindo os valores
dt_whine_red['alcohol'] = dt_whine_red['alcohol'].replace(dict_whine_red)
```

```
# Valores substituídos
dt_whine_red['alcohol'].unique()
```

```
array(['9.4', '9.8', '10', '9.5', '10.5', '9.2', '9.9', '9.1', '9.3', '9',
      '9.7', '10.1', '10.6', '9.6', '10.8', '10.3', '13.1', '10.2',
      '10.9', '10.7', '12.9', '10.4', '13', '14', '11.5', '11.4', '12.4',
      '11', '12.2', '12.8', '12.6', '12.5', '11.7', '11.3', '12.3', '12',
      '11.9', '11.8', '8.7', '13.3', '11.2', '11.6', '11.1', '13.4',
      '12.1', '8.4', '12.7', '14.9', '13.2', '13.6', '13.5', '100',
      '9.55', '8.5', '110', '956', '10.55', '8.8', '135', '11.95',
      '9.95', '923', '9.25', '9.05', '10.75'], dtype=object)
```

```
# Alterando para numeric
dt_whine_red['alcohol'] = pd.to_numeric(dt_whine_red['alcohol'])
```

```
# Dividindo os valores substituídos
dt_whine_red['alcohol'] = np.where(
    dt_whine_red['alcohol'] >= 100,
    dt_whine_red['alcohol'] / 100,
    dt_whine_red['alcohol']
)
```

```
dt_whine_red['alcohol'].unique()
```

```
array([ 9.4 ,  9.8 , 10. ,  9.5 , 10.5 ,  9.2 ,  9.9 ,  9.1 ,  9.3 ,
        9. ,  9.7 , 10.1 , 10.6 ,  9.6 , 10.8 , 10.3 , 13.1 , 10.2 ,
       10.9 , 10.7 , 12.9 , 10.4 , 13. , 14. , 11.5 , 11.4 , 12.4 ,
       11. , 12.2 , 12.8 , 12.6 , 12.5 , 11.7 , 11.3 , 12.3 , 12. ,
       11.9 , 11.8 ,  8.7 , 13.3 , 11.2 , 11.6 , 11.1 , 13.4 , 12.1 ,
        8.4 , 12.7 , 14.9 , 13.2 , 13.6 , 13.5 ,  1. ,  9.55,  8.5 ,
        1.1 ,  9.56, 10.55,  8.8 ,  1.35, 11.95,  9.95,  9.23,  9.25,
        9.05, 10.75])
```

```
dt_whine_red.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   fixed acidity       1599 non-null   float64
 1   volatile acidity    1599 non-null   float64
 2   citric acid         1599 non-null   float64
 3   residual sugar      1599 non-null   float64
 4   chlorides           1599 non-null   float64
 5   free sulfur dioxide 1599 non-null   float64
 6   total sulfur dioxide 1599 non-null   float64
 7   density             1599 non-null   float64
 8   pH                  1599 non-null   float64
 9   sulphates           1599 non-null   float64
10   alcohol             1599 non-null   float64
11   quality             1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
dt_whine_red.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	2.208702	3.311113	90.000000
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	9.664060	0.154386	0.000000
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.000000

2.1. Histograma dos Preditores

```
def univariate_statistics_histogram_frequency(dataset):
    predictors_columns = dataset.columns[:11]

    fig, axs = plt.subplots(3, 4, figsize=(12, 10))
    axs = axs.flatten()

    colors = ['#FFB6C1', '#87CEFA', '#98FB98', '#DDA0DD', '#FFACD', '#F0E68C', '#E6E6FA', '#F5DEB3', '#B0E0E6', '#FFDAB9',
              ]

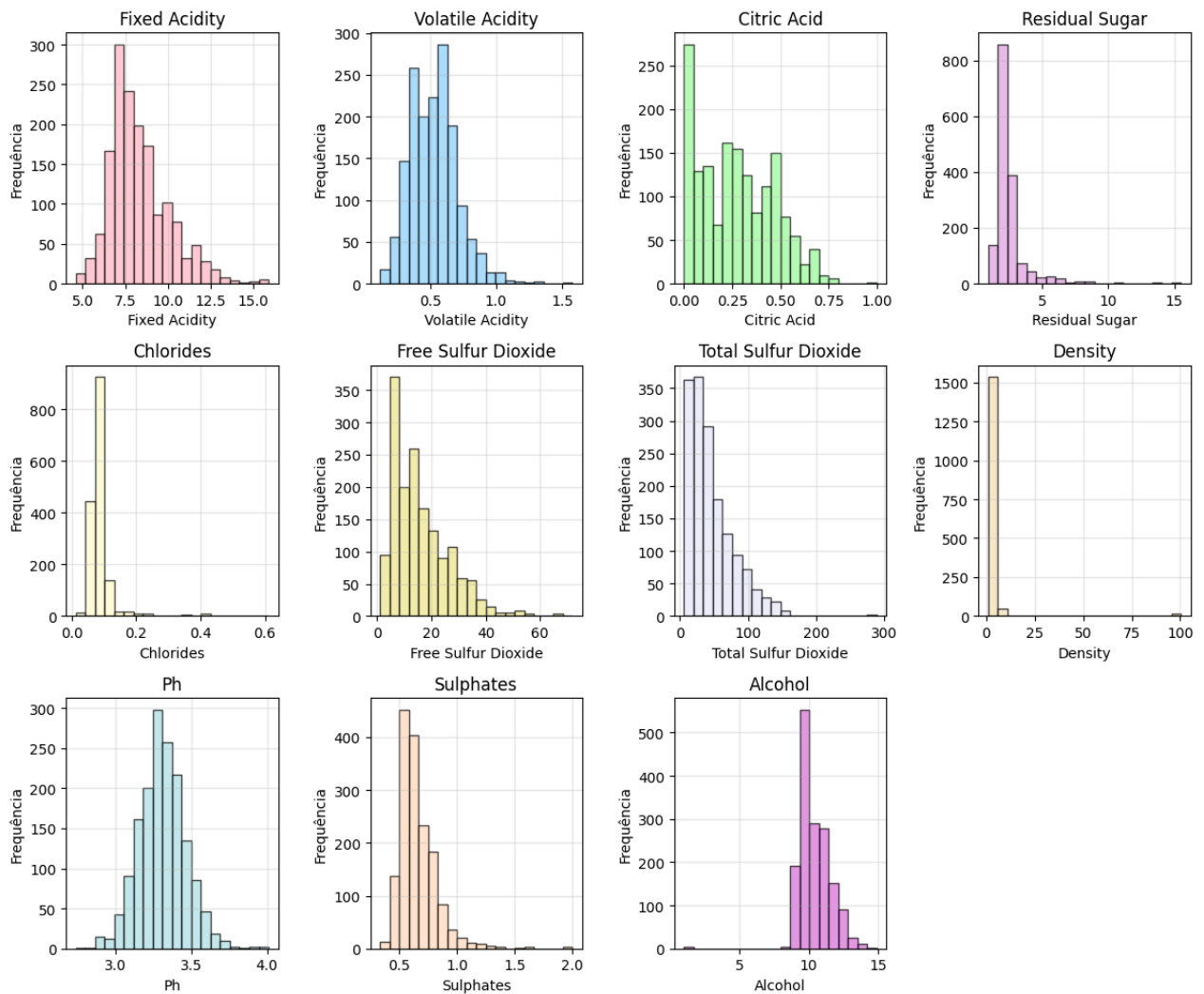
    for i, column in enumerate(predictors_columns):
        data = dataset[column]

        axs[i].hist(data, bins=20, color=colors[i], edgecolor='black', alpha=0.7)
        axs[i].set_title(f"{column.title()}")
        axs[i].set_xlabel(column.title())
        axs[i].set_ylabel("Frequência")
        axs[i].grid(True, alpha=0.3)

    for i in range(len(predictors_columns), len(axs)):
        axs[i].set_visible(False)

    plt.tight_layout()
    plt.show()
```

```
univariate_statistics_histogram_frequency(dt_whine_red)
```



2.2. Calculando os valores da Média, Desvio Padrão e Assimetria de cada preditor

```
def univariate_statistics_values(dataset):
    values = []
    predictors_columns = dataset.columns[:11]

    for column in predictors_columns:
        mean = dataset[column].mean() # Média
        std_dev = dataset[column].std() # Desvio padrão
        skewness = dataset[column].skew() # Assimetria

        values.append({
            "Preditor": column,
            "Média ( $\mu$ )": round(mean, 4),
            "Desvio Padrão ( $\sigma$ )": round(std_dev, 4),
            "Assimetria ( $\gamma$ )": round(skewness, 4)
        })

    results = pd.DataFrame(values)
    return results

table_statistics_red = univariate_statistics_values(dt_wine_red)
```

Podemos observar alta assimetria nos seguintes preditores:

- free sulfur dioxide

- total sulfur dioxide
- density
- sulphates

```
print(table_statistics_red)
```

	Preditor	Média (μ)	Desvio Padrão (σ)	Assimetria (γ)
0	fixed acidity	8.3196	1.7411	0.9828
1	volatile acidity	0.5278	0.1791	0.6716
2	citric acid	0.2710	0.1948	0.3183
3	residual sugar	2.5388	1.4099	4.5407
4	chlorides	0.0875	0.0471	5.6803
5	free sulfur dioxide	15.8749	10.4602	1.2506
6	total sulfur dioxide	46.4678	32.8953	1.5155
7	density	2.2087	9.6641	9.8039
8	pH	3.3111	0.1544	0.1937
9	sulphates	0.6581	0.1695	2.4287
10	alcohol	10.3978	1.1599	-0.5768

3. PRÉ PROCESSAMENTO

3.2. Divisão entre Conjunto de Treino e Teste e Reduzindo a Assimetria

```
# Separando preditores (X) e variável alvo (y)
X = dt_whine_red.drop('quality', axis=1)
y = dt_whine_red['quality']

# Dividindo os dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Criando o transformador
pt = PowerTransformer(method='yeo-johnson')

# Transformação correta
X_train_transf = pt.fit_transform(X_train)
X_test_transf = pt.transform(X_test)

X_train_transf = pd.DataFrame(X_train_transf, columns=X_train.columns)
X_test_transf = pd.DataFrame(X_test_transf, columns=X_test.columns)

print("===== SKEWNESS ANTES =====")
print(X_train.skew(), "\n")

print("===== SKEWNESS DEPOIS =====")
print(X_train_transf.skew())
```

```
===== SKEWNESS ANTES =====
fixed acidity      0.964506
volatile acidity   0.744729
citric acid        0.313905
residual sugar     4.564529
chlorides          5.571477
free sulfur dioxide 1.198835
total sulfur dioxide 1.385318
density            9.823774
pH                 0.197788
sulphates          2.591999
alcohol            -0.881183
dtype: float64
```

```
===== SKEWNESS DEPOIS =====
fixed acidity      0.000261
volatile acidity   0.001181
citric acid        0.016239
residual sugar     -0.017593
chlorides          -0.168916
free sulfur dioxide -0.012305
total sulfur dioxide -0.004827
density            4.743814
pH                 -0.003082
sulphates          0.025771
alcohol            0.349611
dtype: float64
```

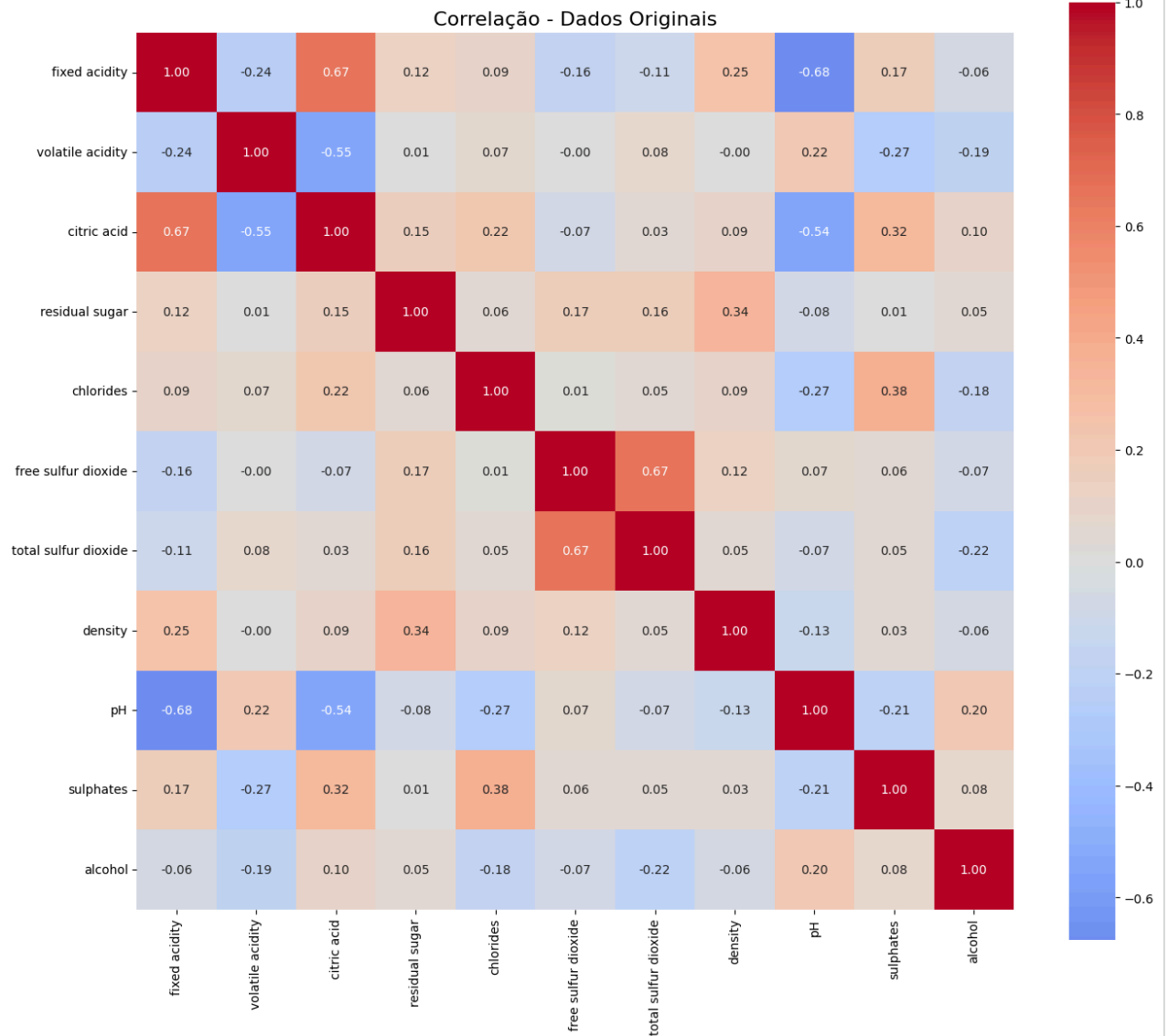
```
def plot_correlation_matrix(df, title="Matriz de Correlação"):
    plt.figure(figsize=(14, 12))
    sns.heatmap(
```

```

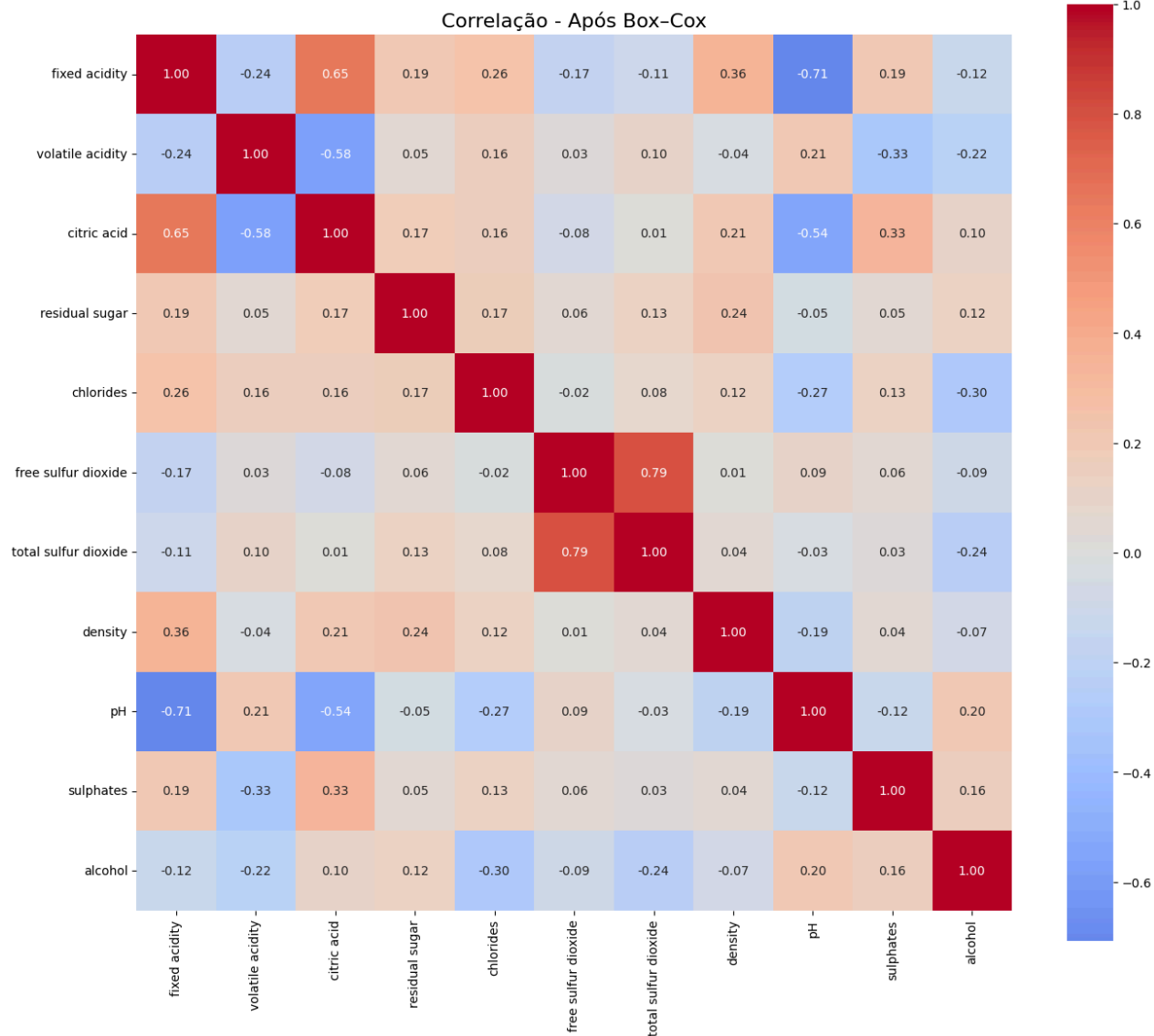
df.corr(),
annot=True,      # ← mostra os valores
fmt=".2f",       # ← duas casas decimais
cmap="coolwarm",
center=0,
square=True
)
plt.title(title, fontsize=16)
plt.tight_layout()
plt.show()

```

```
plot_correlation_matrix(X_train, title="Correlação - Dados Originais")
```



```
plot_correlation_matrix(X_train_transf, title="Correlação - Após Box-Cox")
```



4. Regressão Linear (OLS)

4.1 Adicionando o termo de intercepto

```
# Função para adicionar a coluna de 1s (termo de intercepto)
def add_intercept(X):
    X_new = X.copy()
    X_new.insert(0, "intercept", 1)
    return X_new

# Aplicando no treino e teste
X_train_ols = add_intercept(X_train_transf)
X_test_ols = add_intercept(X_test_transf)

# Convertendo para numpy (OLS manual usa matrizes)
```

```
X_train_np = X_train_ols.values
X_test_np = X_test_ols.values

y_train_np = y_train.values.reshape(-1, 1)
y_test_np = y_test.values.reshape(-1, 1)
```

4.2 Implementação da OLS (modelo manual)

```
def ols_fit(X, y):
    # Fórmula:  $\beta = (X^T X)^{-1} X^T y$ 
    XT = X.T
    beta = np.linalg.inv(XT @ X) @ XT @ y
    return beta

def ols_predict(X, beta):
    # Previsão:  $\hat{y} = X \beta$ 
    return X @ beta
```

4.3 Treinamento e previsão (modelo manual)

```
# Treinar
beta_manual = ols_fit(X_train_np, y_train_np)

# Previsões no teste
y_pred_manual = ols_predict(X_test_np, beta_manual)
```

4.4 Avaliação do modelo (RMSE e R^2)

```
def rmse(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred)**2))

def r2_score(y_true, y_pred):
    ss_res = np.sum((y_true - y_pred)**2)
    ss_tot = np.sum((y_true - np.mean(y_true))**2)
    return 1 - (ss_res / ss_tot)

rmse_manual = rmse(y_test_np, y_pred_manual)
r2_manual = r2_score(y_test_np, y_pred_manual)

print("=== RESULTADOS OLS MANUAL ===")
print("RMSE:", rmse_manual)
print("R²:", r2_manual)
```

```
=== RESULTADOS OLS MANUAL ===
RMSE: 0.6174343105987906
R²: 0.41664619768030864
```

4.5 OLS com Scikit-Learn (para comparação)

```
from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept=True)
model.fit(X_train_transf, y_train)

y_pred_sklearn = model.predict(X_test_transf)

rmse_sklearn = rmse(y_test_np, y_pred_sklearn.reshape(-1,1))
r2_sklearn = r2_score(y_test_np, y_pred_sklearn.reshape(-1,1))

print("=== RESULTADOS OLS SKLEARN ===")
print("RMSE:", rmse_sklearn)
print("R²:", r2_sklearn)
```

```
=== RESULTADOS OLS SKLEARN ===
RMSE: 0.6174343105987904
R²: 0.41664619768030897
```

4.6 Validação Cruzada Manual (k-fold)

```
from sklearn.model_selection import KFold
```

```
def cross_validation_ols(X, y, k=5):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)

    rmse_scores = []
    r2_scores = []

    for train_index, test_index in kf.split(X):
        X_train_fold = X[train_index]
        X_test_fold = X[test_index]
        y_train_fold = y[train_index]
        y_test_fold = y[test_index]

        beta = ols_fit(X_train_fold, y_train_fold)
        y_pred = ols_predict(X_test_fold, beta)

        rmse_scores.append(rmse(y_test_fold, y_pred))
        r2_scores.append(r2_score(y_test_fold, y_pred))

    return np.mean(rmse_scores), np.mean(r2_scores)

rmse_cv_manual, r2_cv_manual = cross_validation_ols(X_train_np, y_train_np, k=10)
```