```
Start coding or generate with AI.
```

https://link.springer.com/article/10.1007/s00170-024-14597-2#Sec28

```
# Instalar uma fonte similar à Times New Roman
!apt-get install -y fonts-liberation

# Configurar o matplotlib para usar a fonte Liberation Serif, que é similar à Times New Roman
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

# Carregar a fonte instalada
font_path = '/usr/share/fonts/truetype/liberation/LiberationSerif-Regular.ttf'
fm.fontManager.addfont(font_path)
plt.rcParams['font.family'] = 'Liberation Serif'
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
fonts-liberation is already the newest version (1:1.07.4-11).
0 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
```

```
import pandas as pd
import numpy as np
```

```
dados = pd.read_excel('/content/drive/MyDrive/ALEX_DOUTORADO_UNIFEI/TITANIUM/ti.xlsx')

```

```
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 287 entries, 0 to 286
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Test        287 non-null    object 
 1   vc [m/min]  287 non-null    float64
 2   f [mm/rev]  287 non-null    float64
 3   Fc [N]      287 non-null    float64
 4   σFc [N]     287 non-null    float64
 5   Ff [N]      287 non-null    float64
 6   σFf [N]     287 non-null    float64
 7   rn [µm]     287 non-null    float64
 8   σrn [µm]    287 non-null    float64
 9   havg [µm]   287 non-null    object 
 10  σhavg [µm]  287 non-null    object 
 11  lcut [mm]   287 non-null    int64  
 12  Tchip [K]   287 non-null    object 
 13  Wear        287 non-null    object 
 14  Status      287 non-null    object 
dtypes: float64(8), int64(1), object(6)
memory usage: 33.8+ KB
```

```
df = dados
```

```
# Extrair os valores mínimo e máximo de vc [m/min] e f [mm/rev]
min_vc = df["vc [m/min]"].min()
max_vc = df["vc [m/min]"].max()

min_f = df["f [mm/rev]"].min()
max_f = df["f [mm/rev]"].max()

# Exibir os resultados
print(f"Cutting speed (vc) - Min: {min_vc} | Max: {max_vc}")
print(f"Feed rate (f) - Min: {min_f} | Max: {max_f}")
```

```
Cutting speed (vc) - Min: 10.0 | Max: 500.0
Feed rate (f) - Min: 0.01 | Max: 0.4
```

```
import pandas as pd

# Carregar os dados (supondo que já estejam disponíveis no ambiente)
# df = pd.read_csv('dados_experimento.csv')  # Exemplo de como carregar caso estivesse em um arquivo

# Exibir um resumo dos níveis de velocidade de corte (vc) e avanço (f)
vc_levels = df["vc [m/min]"].unique()
f_levels = df["f [mm/rev]"].unique()

```

```
# Contar quantos níveis distintos existem para vc e f
num_vc_levels = len(vc_levels)
num_f_levels = len(f_levels)

# Contar quantas combinações únicas de (vc, f) existem
num_unique_combinations = df.groupby(["vc [m/min]", "f [mm/rev]"]).ngroups

# Exibir os resultados
num_vc_levels, num_f_levels, num_unique_combinations

```

```
(26, 11, 109)
```

```
Start coding or generate with AI.
```

```
dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 287 entries, 0 to 286
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Test        287 non-null    object 
 1   vc [m/min]  287 non-null    float64
 2   f [mm/rev]  287 non-null    float64
 3   Fc [N]      287 non-null    float64
 4   σFc [N]     287 non-null    float64
 5   Ff [N]      287 non-null    float64
 6   σFf [N]     287 non-null    float64
 7   rn [µm]     287 non-null    float64
 8   σrn [µm]    287 non-null    float64
 9   havg [µm]   287 non-null    object 
 10  σhavg [µm]  287 non-null    object 
 11  lcut [mm]   287 non-null    int64  
 12  Tchip [K]   287 non-null    object 
 13  Wear        287 non-null    object 
 14  Status      287 non-null    object 
dtypes: float64(8), int64(1), object(6)
memory usage: 33.8+ KB
```

```
# 2. Substituir '-' por NaN em todas as colunas
dados.replace('-', np.nan, inplace=True)
```

```
<ipython-input-9-132c6ea6bcaa>:2: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future ver
  dados.replace('-', np.nan, inplace=True)
```

AQUI VAMOS FAZER UMA MODELAGEM PARA PEVER O DESGASTE (WEAR)

```
df = dados
```

```
X = df[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'σFc [N]', 'Ff [N]', 'σFf [N]', 'rn [µm]', 'σrn [µm]', 'lcut [mm]']]
y = df['Wear']

```

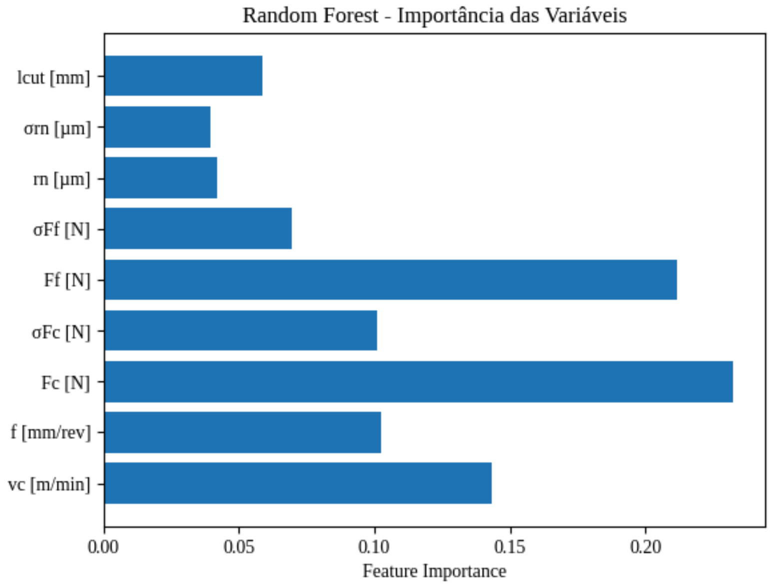
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_encoded = le.fit_transform(y)

```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=le.classes_))
print(confusion_matrix(y_test, y_pred))

```

```
              precision    recall  f1-score   support

           H       1.00      1.00      1.00         1
           L       0.75      1.00      0.86        27
           M       1.00      0.70      0.82        30

    accuracy                           0.84        58
   macro avg       0.92      0.90      0.89        58
weighted avg       0.88      0.84      0.84        58

[[ 1  0  0]
 [ 0 27  0]
 [ 0  9 21]]
```

```python
import matplotlib.pyplot as plt

importances = model.feature_importances_
plt.barh(X.columns, importances)
plt.xlabel("Feature Importance")
plt.title("Random Forest - Importância das Variáveis")
plt.show()

```




```
Start coding or generate with AI.
```

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Exemplo: carregando os dados
# df = pd.read_csv('seus_dados.csv')

# Seleção de variáveis numéricas (ajuste conforme seus dados reais)
X = df[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'σFc [N]', 'Ff [N]', 'σFf [N]', 'rn [μm]', 'σrn [μm]', 'lcut [mm]']]

# Codificando a variável resposta
le = LabelEncoder()
y = le.fit_transform(df['Wear'])  # L=0, M=1, H=2 (por exemplo)

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)

# Modelo de regressão logística multinomial
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

# Coeficientes
classes = le.classes_
coefs = model.coef_  # shape (n_classes, n_features)
intercepts = model.intercept_

# Exibindo as equações
print("EQUAÇÕES DE PROBABILIDADE PARA CADA CLASSE (L, M, H):\n")

```

```
for idx, cls in enumerate(classes):
    eq = f"P(Wear={cls}) = exp({intercepts[idx]:.3f}"
    for i, col in enumerate(X.columns):
        coef = coefs[idx][i]
        eq += f" + {coef:.3f}*{col}"
    eq += ") / DENOMINADOR"
    print(eq)
    print()

# Exemplo de previsão de probabilidades para o conjunto de teste
probs = model.predict_proba(X_test)

# Mostrando a probabilidade das 5 primeiras amostras
for i in range(5):
    print(f"Amostra {i+1}: {dict(zip(classes, probs[i]))}")

```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in versi
  warnings.warn(
EQUAÇÕES DE PROBABILIDADE PARA CADA CLASSE (L, M, H):

P(Wear=H) = exp(-0.016 + 0.008*vc [m/min] + 0.001*f [mm/rev] + -0.064*Fc [N] + 0.023*σFc [N] + 0.165*Ff [N] + 0.446*σFf [N] + -0.47

P(Wear=L) = exp(0.262 + -0.005*vc [m/min] + -0.013*f [mm/rev] + 0.014*Fc [N] + 0.786*σFc [N] + -0.119*Ff [N] + -0.483*σFf [N] + 0.4

P(Wear=M) = exp(-0.245 + -0.003*vc [m/min] + 0.012*f [mm/rev] + 0.050*Fc [N] + -0.809*σFc [N] + -0.046*Ff [N] + 0.038*σFf [N] + -0.

Amostra 1: {'H': np.float64(2.705240923298819e-05), 'L': np.float64(0.9996400822292548), 'M': np.float64(0.0003328653615123293)}
Amostra 2: {'H': np.float64(7.76982974548448e-08), 'L': np.float64(0.4932299896671144), 'M': np.float64(0.506769932634588)}
Amostra 3: {'H': np.float64(1.7483458412442998e-07), 'L': np.float64(0.30154799240733277), 'M': np.float64(0.6984518327580831)}
Amostra 4: {'H': np.float64(5.446059894142669e-09), 'L': np.float64(0.020380957634704513), 'M': np.float64(0.9796190369192357)}
Amostra 5: {'H': np.float64(1.3102196551505972e-07), 'L': np.float64(0.9062259841674062), 'M': np.float64(0.09377388481062839)}
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
import statsmodels.api as sm

# Exemplo: carregando os dados
# df = pd.read_csv('seus_dados.csv')

# Seleção de variáveis numéricas
X = df[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'σFc [N]', 'Ff [N]', 'σFf [N]', 'rn [μm]', 'σrn [μm]', 'lcut [mm]']]

# Codificando a variável resposta
le = LabelEncoder()
y = le.fit_transform(df['Wear'])  # L=0, M=1, H=2 (por exemplo)
classes = le.classes_

# Divisão treino/teste
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)

# --- MODELO SKLEARN (para obtenção das equações)
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
model.fit(X_train, y_train)

# Coeficientes
coefs = model.coef_
intercepts = model.intercept_

print("EQUAÇÕES DE PROBABILIDADE PARA CADA CLASSE (L, M, H):\n")
for idx, cls in enumerate(classes):
    eq = f"P(Wear={cls}) = exp({intercepts[idx]:.3f}"
    for i, col in enumerate(X.columns):
        coef = coefs[idx][i]
        eq += f" + {coef:.3f}*{col}"
    eq += ") / DENOMINADOR"
    print(eq)
    print()

# Exemplo de previsão de probabilidades
probs = model.predict_proba(X_test)
for i in range(5):
```

```python
    print(f"Amostra {i+1}: {dict(zip(classes, probs[i]))}")

# --- NORMALIZAÇÃO PARA STATSMODELS
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Adiciona constante (intercepto)
X_sm = sm.add_constant(X_scaled)

# Modelo estatístico
model_sm = sm.MNLogit(y, X_sm)
result = model_sm.fit()

# Mostra os coeficientes com significância estatística
print("\n=== ANÁLISE ESTATÍSTICA DETALHADA (p-values, z-score) ===\n")
print(result.summary())


```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in ver
  warnings.warn(
EQUAÇÕES DE PROBABILIDADE PARA CADA CLASSE (L, M, H):

P(Wear=H) = exp(-0.016 + 0.008*vc [m/min] + 0.001*f [mm/rev] + -0.064*Fc [N] + 0.023*σFc [N] + 0.165*Ff [N] + 0.446*σFf [N] + -0.4

P(Wear=L) = exp(0.262 + -0.005*vc [m/min] + -0.013*f [mm/rev] + 0.014*Fc [N] + 0.786*σFc [N] + -0.119*Ff [N] + -0.483*σFf [N] + 0

P(Wear=M) = exp(-0.245 + -0.003*vc [m/min] + 0.012*f [mm/rev] + 0.050*Fc [N] + -0.809*σFc [N] + -0.046*Ff [N] + 0.038*σFf [N] + -0

Amostra 1: {'H': np.float64(2.705240923298819e-05), 'L': np.float64(0.9996400822292548), 'M': np.float64(0.0003328653615123293)}
Amostra 2: {'H': np.float64(7.76982974548448e-08), 'L': np.float64(0.4932299896671144), 'M': np.float64(0.506769932634588)}
Amostra 3: {'H': np.float64(1.7483458412442998e-07), 'L': np.float64(0.30154799240733277), 'M': np.float64(0.6984518327580831)}
Amostra 4: {'H': np.float64(5.446059894142669e-09), 'L': np.float64(0.020380957634704513), 'M': np.float64(0.9796190369192357)}
Amostra 5: {'H': np.float64(1.3102196551505972e-07), 'L': np.float64(0.9062259841674062), 'M': np.float64(0.09377388481062839)}
Optimization terminated successfully.
         Current function value: nan
         Iterations 34

=== ANÁLISE ESTATÍSTICA DETALHADA (p-values, z-score) ===

                          MNLogit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  287
Model:                        MNLogit   Df Residuals:                      267
Method:                           MLE   Df Model:                           18
Date:                Thu, 01 May 2025   Pseudo R-squ.:                     nan
Time:                        16:52:42   Log-Likelihood:                    nan
converged:                       True   LL-Null:                       -215.85
Covariance Type:            nonrobust   LLR p-value:                       nan
==============================================================================
       y=1       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const             nan        nan        nan        nan         nan         nan
x1                nan        nan        nan        nan         nan         nan
x2                nan        nan        nan        nan         nan         nan
x3                nan        nan        nan        nan         nan         nan
x4                nan        nan        nan        nan         nan         nan
x5                nan        nan        nan        nan         nan         nan
x6                nan        nan        nan        nan         nan         nan
x7                nan        nan        nan        nan         nan         nan
x8                nan        nan        nan        nan         nan         nan
x9                nan        nan        nan        nan         nan         nan
------------------------------------------------------------------------------
       y=2       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const             nan        nan        nan        nan         nan         nan
x1                nan        nan        nan        nan         nan         nan
x2                nan        nan        nan        nan         nan         nan
x3                nan        nan        nan        nan         nan         nan
x4                nan        nan        nan        nan         nan         nan
x5                nan        nan        nan        nan         nan         nan
x6                nan        nan        nan        nan         nan         nan
x7                nan        nan        nan        nan         nan         nan
x8                nan        nan        nan        nan         nan         nan
x9                nan        nan        nan        nan         nan         nan
==============================================================================
```

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif["feature"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)

```

```
      feature        VIF
0  vc [m/min]   2.880643
1  f [mm/rev]  45.539730
2      Fc [N]  94.150085
3     σFc [N]   5.782419
4      Ff [N]  35.595913
5     σFf [N]   4.253478
6     rn [μm]  38.777892
7    σrn [μm]  11.814109
8   lcut [mm]  20.186063
```

```python
X_reduced = X.drop(columns=['Fc [N]', 'f [mm/rev]', 'Ff [N]', 'rn [μm]', 'σrn [μm]', 'lcut [mm]'])

```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
import statsmodels.api as sm

# Exemplo: carregando os dados
# df = pd.read_csv('seus_dados.csv')

# Remover classe 'H'
df_filtered = df[df['Wear'] != 'H'].copy()

# Variáveis de entrada com VIF aceitável
X = df_filtered[['vc [m/min]', 'σFc [N]', 'σFf [N]']]

# Codificar variável alvo (L = 0, M = 1)
le = LabelEncoder()
y = le.fit_transform(df_filtered['Wear'])  # L = 0, M = 1

# Normalizar
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = sm.add_constant(X_scaled)

# Regressão logística binária
model = sm.Logit(y, X_scaled)
result = model.fit()

# Resultado
print("\n=== Regressão logística binária: Wear M (1) vs L (0) ===\n")
print(result.summary())


```

```
Optimization terminated successfully.
         Current function value: 0.666751
         Iterations 4

=== Regressão logística binária: Wear M (1) vs L (0) ===

                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  282
Model:                          Logit   Df Residuals:                      278
Method:                           MLE   Df Model:                            3
Date:                Thu, 01 May 2025   Pseudo R-squ.:                 0.01375
Time:                        17:38:15   Log-Likelihood:                -188.02
converged:                       True   LL-Null:                       -190.65
Covariance Type:            nonrobust   LLR p-value:                    0.1548
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3769      0.122     -3.080      0.002      -0.617      -0.137
x1             0.0916      0.130      0.702      0.482      -0.164       0.347
x2             0.4147      0.210      1.975      0.048       0.003       0.826
x3            -0.1852      0.212     -0.873      0.383      -0.601       0.231
==============================================================================
```

```python
# Extrair coeficientes
params = result.params
intercept = params[0]
coef_names = ['vc [m/min]', 'σFc [N]', 'σFf [N]']
coefs = params[1:]

# Montar equação
eq = f"P(Wear = M) = 1 / (1 + exp(-({intercept:.3f}"
for name, coef in zip(coef_names, coefs):
    eq += f" + {coef:.3f}*{name}"
eq += ")))"

```

```
print("\n=== EQUAÇÃO DE REGRESSÃO LOGÍSTICA ===")
print(eq)
```

```
=== EQUAÇÃO DE REGRESSÃO LOGÍSTICA ===
P(Wear = M) = 1 / (1 + exp(-(-0.377 + 0.092*vc [m/min] + 0.415*σFc [N] + -0.185*σFf [N])))
```

SEM H E COM TODAS AS VERIAVEIS

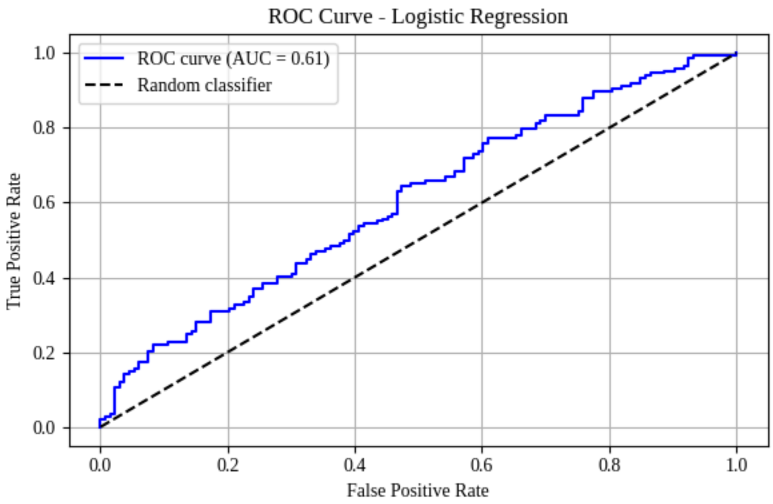
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
import statsmodels.api as sm

# Exemplo: df = pd.read_csv('seus_dados.csv')

# 1. Filtrar apenas classes L e M
df_filtered = df[df['Wear'].isin(['L', 'M'])].copy()

# 2. Selecionar todas as variáveis numéricas
X = df_filtered[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'σFc [N]', 'Ff [N]', 'σFf [N]', 'rn [µm]', 'σrn [µm]', 'lcut [mm]']]

# 3. Codificar variável alvo (L = 0, M = 1)
le = LabelEncoder()
y = le.fit_transform(df_filtered['Wear'])  # L=0, M=1

# 4. Normalizar e adicionar intercepto
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = sm.add_constant(X_scaled)

# 5. Ajustar o modelo logístico binário
model = sm.Logit(y, X_scaled)
result = model.fit()

# 6. Mostrar análise estatística
print("\n=== RESULTADOS DA REGRESSÃO LOGÍSTICA BINÁRIA: M (1) vs L (0) ===\n")
print(result.summary())

# 7. Gerar equação interpretável
params = result.params
intercept = params[0]
coef_names = X.columns
coefs = params[1:]

eq = f"P(Wear = M) = 1 / (1 + exp(-({intercept:.3f}"
for name, coef in zip(coef_names, coefs):
    eq += f" + {coef:.3f}*{name}"
eq += ")))"

print("\n=== EQUAÇÃO DE REGRESSÃO LOGÍSTICA BINÁRIA ===")
print(eq)
```

```
Optimization terminated successfully.
         Current function value: 0.210220
         Iterations 10

=== RESULTADOS DA REGRESSÃO LOGÍSTICA BINÁRIA: M (1) vs L (0) ===

                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  282
Model:                          Logit   Df Residuals:                      272
Method:                           MLE   Df Model:                            9
Date:                Thu, 01 May 2025   Pseudo R-squ.:                  0.6890
Time:                        17:38:21   Log-Likelihood:                -59.282
converged:                       True   LL-Null:                       -190.65
Covariance Type:            nonrobust   LLR p-value:                 2.042e-51
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.2809      0.338     -0.831      0.406      -0.944       0.382
x1             0.0144      0.313      0.046      0.963      -0.600       0.629
x2             9.7638      2.905      3.361      0.001       4.069      15.458
x3            -4.4027      4.269     -1.031      0.302     -12.770       3.965
x4            -4.7901      1.246     -3.844      0.000      -7.233      -2.347
x5             3.5673      2.493      1.431      0.153      -1.320       8.454
x6             0.9200      0.892      1.031      0.303      -0.829       2.669
x7            -0.2201      0.580     -0.379      0.704      -1.358       0.917
x8             0.4024      0.273      1.475      0.140      -0.132       0.937
```

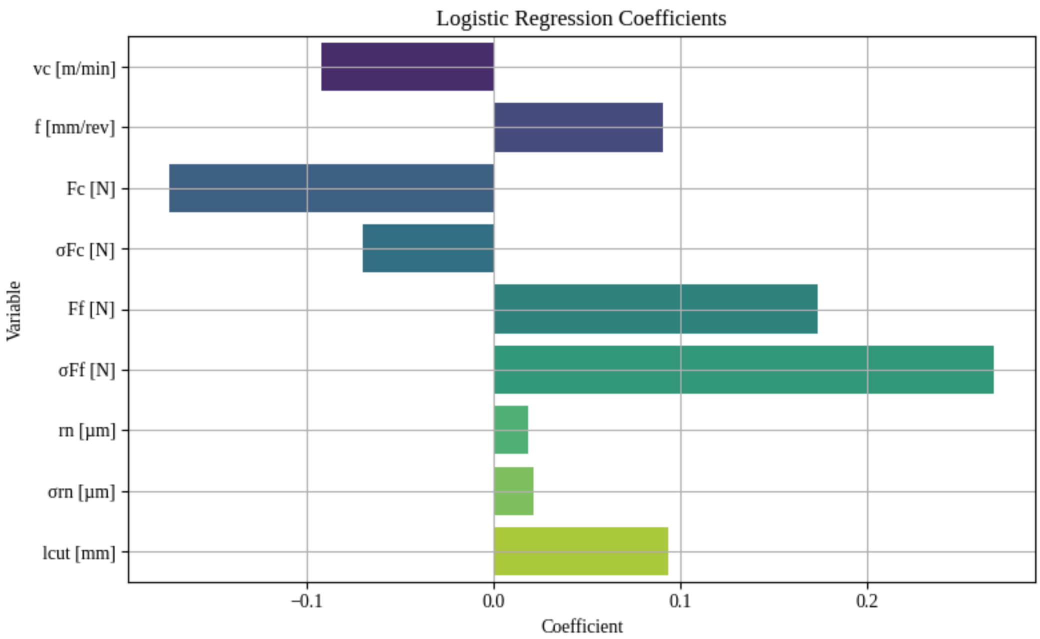
```
x9             3.6261      0.827      4.384      0.000       2.005       5.247
==============================================================================

Possibly complete quasi-separation: A fraction 0.19 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.

=== EQUAÇÃO DE REGRESSÃO LOGÍSTICA BINÁRIA ===
P(Wear = M) = 1 / (1 + exp(-(-0.281 + 0.014*vc [m/min] + 9.764*f [mm/rev] + -4.403*Fc [N] + -4.790*σFc [N] + 3.567*Ff [N] + 0.920*σF
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
import statsmodels.api as sm
from sklearn.metrics import roc_curve, auc

# Dados simulados para demonstração (substitua pelo seu DataFrame real `df`)
np.random.seed(42)
size = 282
df = pd.DataFrame({
    'Wear': np.random.choice(['L', 'M'], size=size, p=[0.5, 0.5]),
    'vc [m/min]': np.random.normal(120, 10, size),
    'f [mm/rev]': np.random.normal(0.3, 0.05, size),
    'Fc [N]': np.random.normal(800, 100, size),
    'σFc [N]': np.random.normal(10, 2, size),
    'Ff [N]': np.random.normal(200, 30, size),
    'σFf [N]': np.random.normal(5, 1, size),
    'rn [µm]': np.random.normal(15, 5, size),
    'σrn [µm]': np.random.normal(1.2, 0.3, size),
    'lcut [mm]': np.random.normal(100, 20, size)
})

# 1. Filtrar apenas classes L e M
df_filtered = df[df['Wear'].isin(['L', 'M'])].copy()

# 2. Selecionar todas as variáveis numéricas
X = df_filtered[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'σFc [N]', 'Ff [N]', 'σFf [N]', 'rn [µm]', 'σrn [µm]', 'lcut [mm]']]

# 3. Codificar variável alvo (L = 0, M = 1)
le = LabelEncoder()
y = le.fit_transform(df_filtered['Wear'])  # L=0, M=1

# 4. Normalizar e adicionar intercepto
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = sm.add_constant(X_scaled)

# 5. Ajustar o modelo logístico binário
model = sm.Logit(y, X_scaled)
result = model.fit()

# 6. Prever probabilidades para o conjunto de dados
pred_probs = result.predict(X_scaled)

# 7. Curva ROC
fpr, tpr, _ = roc_curve(y, pred_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Optimization terminated successfully.
         Current function value: 0.672895
         Iterations 4
```

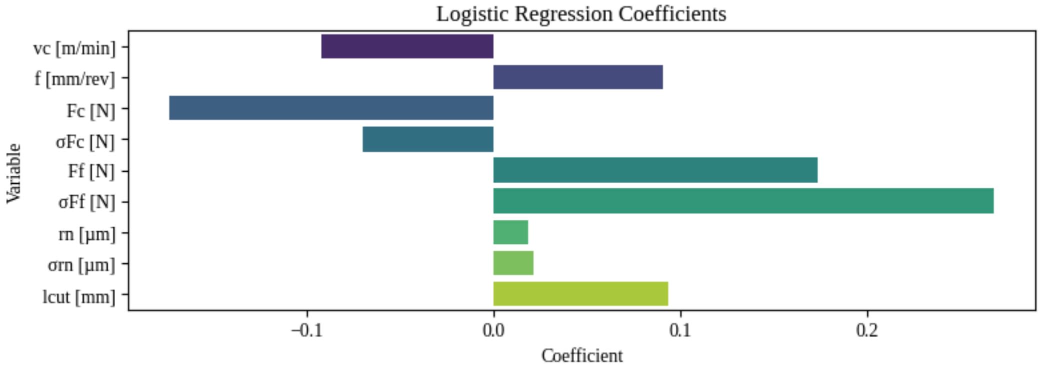



```
<ipython-input-38-e350b7b5a0bb>:68: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `

  sns.barplot(x='Coefficient', y='Variable', data=coef_df, palette='viridis')
```




```
# 8. Coeficientes com barras
params = result.params[1:]  # exclude intercept
variables = X.columns
coef_df = pd.DataFrame({'Variable': variables, 'Coefficient': params})
plt.figure(figsize=(8, 3))
sns.barplot(x='Coefficient', y='Variable', data=coef_df, palette='viridis')
plt.title('Logistic Regression Coefficients')
plt.grid(False)
plt.tight_layout()
plt.show()
```

```
<ipython-input-41-f95275e0c844>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `le

  sns.barplot(x='Coefficient', y='Variable', data=coef_df, palette='viridis')
```



Start coding or generate with AI.

Start coding or generate with AI.

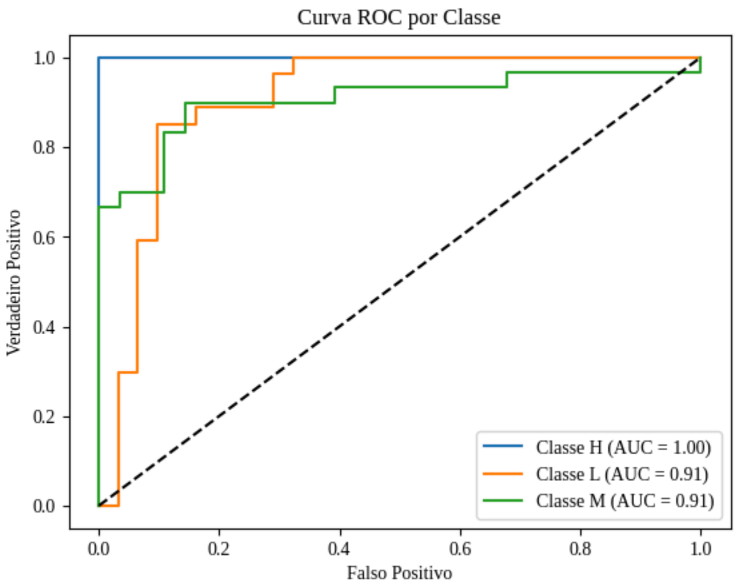
Start coding or generate with AI.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
import statsmodels.api as sm

# Exemplo: carregando os dados
# df = pd.read_csv('seus_dados.csv')

# Seleção das variáveis com baixo VIF
X = df[['vc [m/min]', 'σFc [N]', 'σFf [N]']]

# Codificação do alvo
le = LabelEncoder()
y = le.fit_transform(df['Wear'])  # L=0, M=1, H=2
classes = le.classes_

# Normalização
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = sm.add_constant(X_scaled)

# Loop para one-vs-rest
for target_class in np.unique(y):
    y_binary = (y == target_class).astype(int)  # 1 para a classe de interesse, 0 para as outras
    model = sm.Logit(y_binary, X_scaled)
    result = model.fit(disp=False)

    print(f"\n=== Regressão logística para classe '{classes[target_class]}' vs resto ===")
    print(result.summary())

```

```
=== Regressão logística para classe 'H' vs resto ===
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  287
Model:                          Logit   Df Residuals:                      283
Method:                           MLE   Df Model:                            3
Date:                Thu, 01 May 2025   Pseudo R-squ.:                    -inf
Time:                        17:00:10   Log-Likelihood:                   -inf
converged:                      False   LL-Null:                       -25.206
Covariance Type:            nonrobust   LLR p-value:                     1.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         78.2283     74.871      1.045      0.296     -68.516     224.972
x1            34.8174     40.226      0.866      0.387     -44.023     113.658
x2            71.5488     62.728      1.141      0.254     -51.396     194.494
x3          -182.0846    161.243     -1.129      0.259    -498.115     133.946
==============================================================================

=== Regressão logística para classe 'L' vs resto ===
                           Logit Regression Results
==============================================================================
```

```
Dep. Variable:                      y   No. Observations:                  287
Model:                          Logit   Df Residuals:                      283
Method:                           MLE   Df Model:                            3
Date:                Thu, 01 May 2025   Pseudo R-squ.:                 0.02752
Time:                        17:00:10   Log-Likelihood:                -189.70
converged:                       True   LL-Null:                       -195.07
Covariance Type:            nonrobust   LLR p-value:                   0.01324
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.3206      0.124      2.596      0.009       0.079       0.563
x1            -0.1616      0.128     -1.260      0.208      -0.413       0.090
x2            -0.2523      0.175     -1.439      0.150      -0.596       0.091
x3            -0.2779      0.295     -0.941      0.347      -0.857       0.301
==============================================================================

=== Regressão logística para classe 'M' vs resto ===
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  287
Model:                          Logit   Df Residuals:                      283
Method:                           MLE   Df Model:                            3
Date:                Thu, 01 May 2025   Pseudo R-squ.:                 0.02315
Time:                        17:00:10   Log-Likelihood:                -188.76
converged:                       True   LL-Null:                       -193.24
Covariance Type:            nonrobust   LLR p-value:                   0.03003
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.4550      0.128     -3.551      0.000      -0.706      -0.204
x1             0.0527      0.129      0.409      0.682      -0.200       0.305
x2             0.5089      0.202      2.520      0.012       0.113       0.905
x3            -0.8848      0.454     -1.948      0.051      -1.775       0.005
==============================================================================
```

```python
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# Binariza as classes para one-vs-rest
y_bin = label_binarize(y_test, classes=[0,1,2])

# ROC por classe
for i, class_name in enumerate(classes):
    fpr, tpr, _ = roc_curve(y_bin[:, i], model.predict_proba(X_test)[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'Classe {class_name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('Falso Positivo')
plt.ylabel('Verdadeiro Positivo')
plt.title('Curva ROC por Classe')
plt.legend()
plt.show()

```




```python
from sklearn.metrics import classification_report
print(classification_report(y_test, model.predict(X_test), target_names=classes))

```

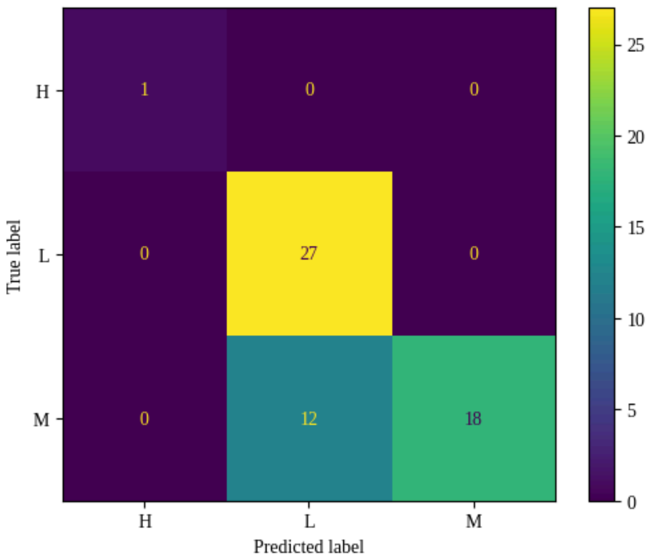
```
              precision    recall  f1-score   support

           H       1.00      1.00      1.00         1
           L       0.69      1.00      0.82        27
           M       1.00      0.60      0.75        30

    accuracy                           0.79        58
   macro avg       0.90      0.87      0.86        58
weighted avg       0.86      0.79      0.79        58
```

```python
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, model.predict(X_test))
print(f"Acurácia: {accuracy:.3f}")
```

```
Acurácia: 0.793
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, model.predict(X_test), display_labels=classes)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7904f10bdc90>
```




```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```python
# Selecionar apenas as colunas desejadas
dados_filtrados = dados[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'Wear']]

# Exibir os tipos das colunas selecionadas
print("\nTipos das colunas selecionadas:")
print(dados_filtrados.dtypes)

# Exibir as primeiras linhas do DataFrame filtrado
print("\nPrévia dos dados selecionados:")
print
```

```
Tipos das colunas selecionadas:
vc [m/min]    float64
f [mm/rev]    float64
Fc [N]        float64
Wear           object
dtype: object
```

```
Prévia dos dados selecionados:
<function print(*args, sep=' ', end='\n', file=None, flush=False)>
```

```python
dados_filtrados['Wear'].value_counts()
```

Wear	count
L	167
M	115
H	5

dtype: int64

```python
# Remover a classe 'H'
dados_filtrados = dados_filtrados[dados_filtrados['Wear'] != 'H']

# Verificar a nova distribuição
print("\nNova distribuição de 'Wear' após remover 'H':")
print(dados_filtrados['Wear'].value_counts())

```

```
Nova distribuição de 'Wear' após remover 'H':
Wear
L    167
M    115
Name: count, dtype: int64
```

```python
dados_filtrados
```

	vc [m/min]	f [mm/rev]	Fc [N]	Wear
0	10.5	0.01	55.7	L
1	12.6	0.01	55.2	L
2	10.5	0.01	57.4	L
3	12.6	0.10	214.0	L
4	10.5	0.10	209.1	L
...	...	...	...	...
282	150.0	0.02	63.6	L
283	150.0	0.02	61.6	L
284	150.0	0.02	63.4	L
285	150.0	0.03	85.4	L
286	150.0	0.03	87.1	L

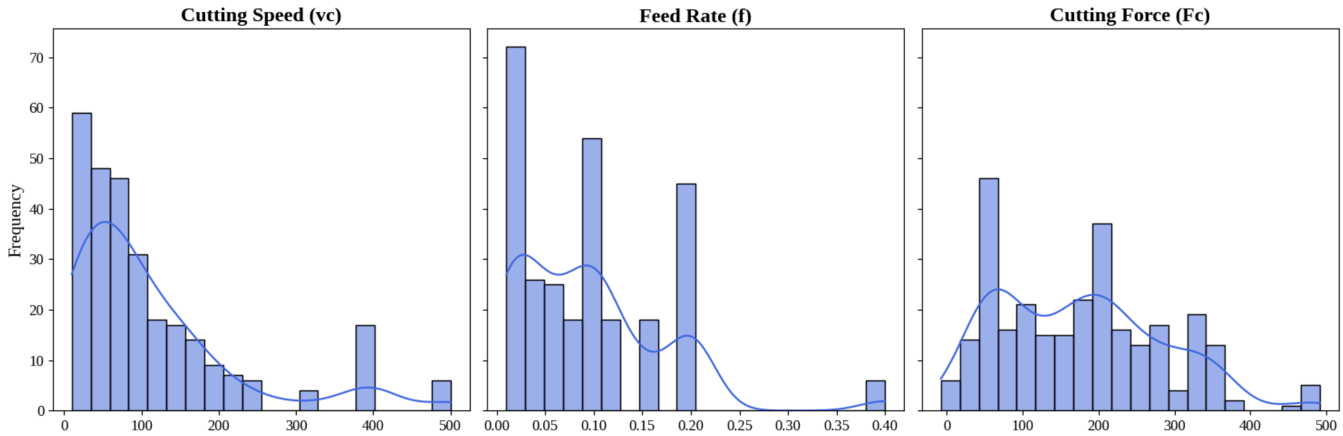
282 rows × 4 columns

```python
dados_filtrados.info()

```

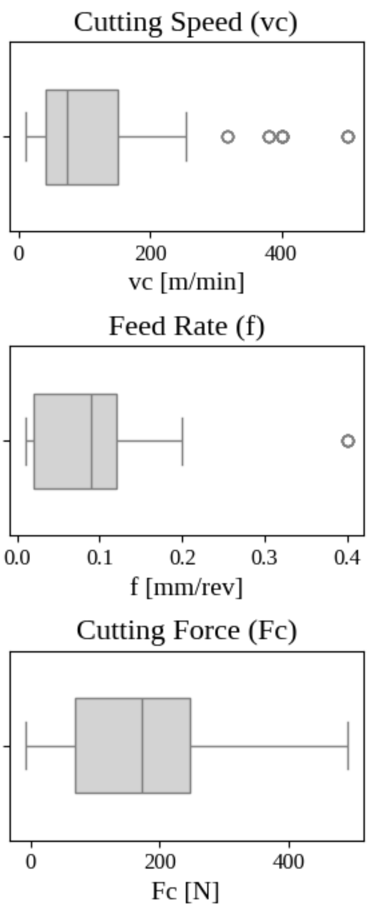
```
<class 'pandas.core.frame.DataFrame'>
Index: 282 entries, 0 to 286
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   vc [m/min]  282 non-null    float64
 1   f [mm/rev]  282 non-null    float64
 2   Fc [N]      282 non-null    float64
 3   Wear        282 non-null    object
dtypes: float64(3), object(1)
memory usage: 11.0+ KB
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Criar a figura com 3 subgráficos na horizontal
fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)  # Compartilhar eixo Y para melhor comparação

# Definir as variáveis e os títulos
variaveis = ['vc [m/min]', 'f [mm/rev]', 'Fc [N]']
titulos = ['Cutting Speed (vc)', 'Feed Rate (f)', 'Cutting Force (Fc)']
```

```
# Gerar histogramas
for i, var in enumerate(variaveis):
    sns.histplot(dados_filtrados[var], bins=20, kde=True, color="royalblue", ax=axes[i])
    axes[i].set_title(titulos[i], fontsize=16, fontweight="bold")
    axes[i].set_xlabel("")  # Remover rótulo do eixo X
    axes[i].set_ylabel("Frequency", fontsize=14)  # Manter rótulo apenas no eixo Y
    axes[i].tick_params(axis='both', labelsize=12)

# Ajustar layout
plt.tight_layout()
plt.show()
```



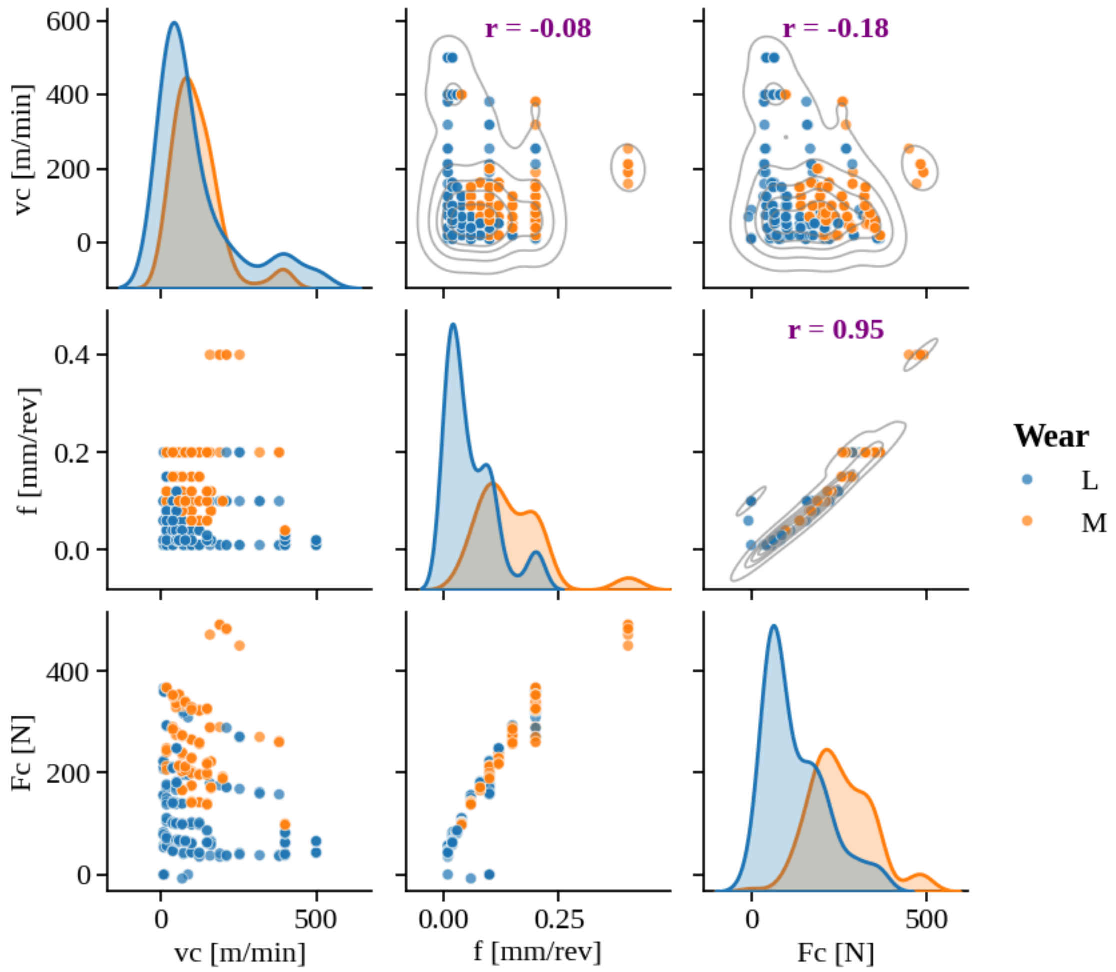

```
import matplotlib.pyplot as plt
import seaborn as sns

# Criar a figura com 3 subgráficos na vertical, sem compartilhar eixo X
fig, axes = plt.subplots(3, 1, figsize=(3, 7))

# Definir as variáveis e os títulos
variaveis = ['vc [m/min]', 'f [mm/rev]', 'Fc [N]']
titulos = ['Cutting Speed (vc)', 'Feed Rate (f)', 'Cutting Force (Fc)']

# Gerar boxplots com escalas individuais e cor cinza claro
for i, var in enumerate(variaveis):
    sns.boxplot(x=dados_filtrados[var], ax=axes[i], color="lightgray", orient="h", width=0.5)
    axes[i].set_title(titulos[i], fontsize=16)
    axes[i].set_xlabel(var, fontsize=14)
    axes[i].tick_params(axis='both', labelsize=12)

# Ajustar layout
plt.tight_layout()
plt.show()
```




```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Criar uma função para anotar a correlação nos gráficos superiores
def annotate_correlation(x, y, **kwargs):
    corr = np.corrcoef(x, y)[0, 1]  # Calcula a correlação de Pearson
    ax = plt.gca()
    ax.annotate(f"r = {corr:.2f}", xy=(0.5, 0.9), xycoords=ax.transAxes,
                ha="center", fontsize=16, color="red", fontweight="bold")

# Ajustar o estilo e tamanho da figura
sns.set_context("notebook", rc={"axes.labelsize": 16, "xtick.labelsize": 16, "ytick.labelsize": 16,
                                "legend.fontsize": 16, "figure.figsize": (8, 8)})

# Selecionar apenas as variáveis desejadas
dados_selecionados = dados_filtrados[['vc [m/min]', 'f [mm/rev]', 'Fc [N]', 'Wear']]

# Criar o pairplot com a variável Wear como hue (cor) e incluir linhas de densidade
g = sns.pairplot(dados_selecionados, hue="Wear", diag_kind="kde", markers=["o", "o"],
                 plot_kws={"alpha": 0.7}, diag_kws={"shade": True, "linewidth": 2})

# Adicionar linhas de densidade (KDE Contours) nos gráficos de dispersão
for i, j in zip(*np.triu_indices_from(g.axes, k=1)):  # Apenas na metade superior
    sns.kdeplot(x=dados_selecionados.iloc[:, j], y=dados_selecionados.iloc[:, i], ax=g.axes[i, j],
                levels=6, color="gray", linewidths=1.2, alpha=0.6)

# Ajustar o tamanho dos rótulos dos eixos, títulos e legendas manualmente
for ax in g.axes.flatten():
    if ax is not None:
        ax.xaxis.label.set_size(16)  # Fonte do rótulo do eixo X
        ax.yaxis.label.set_size(16)  # Fonte do rótulo do eixo Y
        ax.tick_params(axis='both', labelsize=16)  # Fonte dos ticks dos eixos

# Ajustar a legenda
g._legend.set_bbox_to_anchor((1, 0.5))  # Posicionar melhor a legenda
g._legend.set_title("Wear", prop={'size': 18, 'weight': 'bold'})  # Aumentar e negritar "Wear"

# Ajustar a fonte dos rótulos da legenda
for text in g._legend.texts:
    text.set_fontsize(16)

# Adicionar as anotações de correlação nos gráficos superiores
for i, j in zip(*np.triu_indices_from(g.axes, k=1)):
```

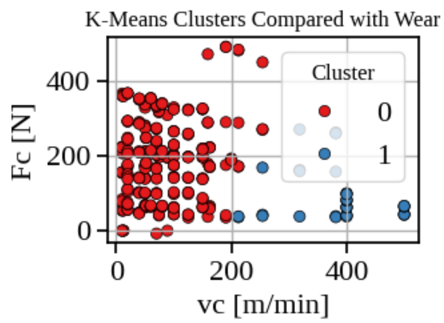
```
    g.axes[i, j].annotate(f"r = {np.corrcoef(dados_selecionados.iloc[:, j], dados_selecionados.iloc[:, i])[0, 1]:.2f}",
                          xy=(0.5, 0.9), xycoords=g.axes[i, j].transAxes,
                          ha="center", fontsize=16, color="purple", fontweight="bold")

plt.show()

```

```
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:1513: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(x=vector, **plot_kwargs)
```



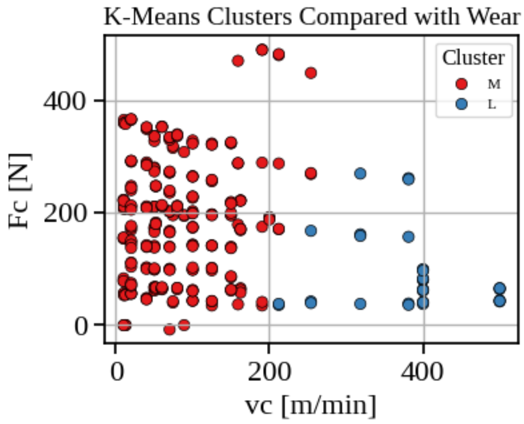

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Selecionar apenas as variáveis de entrada
X_cluster = dados_filtrados[['vc [m/min]', 'f [mm/rev]', 'Fc [N]']]

# Aplicar K-Means com 2 clusters (pois temos 2 classes em Wear)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
dados_filtrados['Cluster_KMeans'] = kmeans.fit_predict(X_cluster)

# Visualizar os clusters comparados com Wear
plt.figure(figsize=(3, 2))
sns.scatterplot(x=dados_filtrados['vc [m/min]'], y=dados_filtrados['Fc [N]'],
                hue=dados_filtrados['Cluster_KMeans'], palette="Set1", marker="o", edgecolor="k")
plt.title("K-Means Clusters Compared with Wear")
```

```python
plt.xlabel("vc [m/min]")
plt.ylabel("Fc [N]")
plt.legend(title="Cluster")
plt.grid()
plt.show()

```



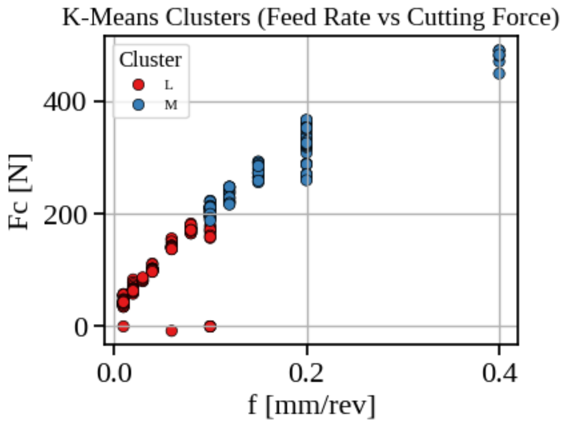

```python
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Selecionar apenas as variáveis de entrada
X_cluster = dados_filtrados[['vc [m/min]', 'f [mm/rev]', 'Fc [N]']]

# Aplicar K-Means com 2 clusters
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
dados_filtrados['Cluster_KMeans'] = kmeans.fit_predict(X_cluster)

# Verificar médias dos clusters para identificar padrões distintos
cluster_means = dados_filtrados.groupby('Cluster_KMeans')[['vc [m/min]', 'f [mm/rev]', 'Fc [N]']].mean()
print("Médias dos clusters:\n", cluster_means)

# Mapear clusters para as classes verdadeiras (L e M) com base nas médias
# O cluster com maior força de corte (Fc [N]) provavelmente representa a classe M (desgaste maior)
if cluster_means.loc[0, 'Fc [N]'] > cluster_means.loc[1, 'Fc [N]']:
    cluster_mapping = {0: 'M', 1: 'L'}
else:
    cluster_mapping = {0: 'L', 1: 'M'}

# Aplicar o mapeamento
dados_filtrados['Cluster_Label'] = dados_filtrados['Cluster_KMeans'].map(cluster_mapping)

# Verificar se os clusters foram mapeados corretamente
print("Cluster Mapping:", cluster_mapping)
print(dados_filtrados[['Cluster_KMeans', 'Wear', 'Cluster_Label']].head())

# Visualizar os clusters comparados com Wear
plt.figure(figsize=(4, 3))
sns.scatterplot(x=dados_filtrados['vc [m/min]'], y=dados_filtrados['Fc [N]'],
                hue=dados_filtrados['Cluster_Label'], palette="Set1", marker="o", edgecolor="k")
plt.title("K-Means Clusters Compared with Wear", fontsize=14)
plt.xlabel("vc [m/min]")
plt.ylabel("Fc [N]")
plt.legend(title="Cluster", fontsize=8)
plt.grid()
plt.show()

```

```
Médias dos clusters:
                 vc [m/min]  f [mm/rev]     Fc [N]
Cluster_KMeans
0                   80.1572    0.100800  187.67880
1                  380.2625    0.045625   90.01875
Cluster Mapping: {0: 'M', 1: 'L'}
   Cluster_KMeans Wear Cluster_Label
0               0    L             M
1               0    L             M
2               0    L             M
3               0    L             M
4               0    L             M
```




```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Selecionar as variáveis para clustering (f e Fc)
X_cluster_f_fc = dados_filtrados[['f [mm/rev]', 'Fc [N]']]

# Aplicar K-Means com 2 clusters
kmeans_f_fc = KMeans(n_clusters=2, random_state=42, n_init=10)
dados_filtrados['Cluster_KMeans_f_fc'] = kmeans_f_fc.fit_predict(X_cluster_f_fc)

# Verificar médias dos clusters para identificar padrões distintos
cluster_means_f_fc = dados_filtrados.groupby('Cluster_KMeans_f_fc')[['f [mm/rev]', 'Fc [N]']].mean()
print("Médias dos clusters (f vs Fc):\n", cluster_means_f_fc)

# Mapear clusters para as classes verdadeiras (L e M) com base nas médias
if cluster_means_f_fc.loc[0, 'Fc [N]'] > cluster_means_f_fc.loc[1, 'Fc [N]']:
    cluster_mapping_f_fc = {0: 'M', 1: 'L'}
else:
    cluster_mapping_f_fc = {0: 'L', 1: 'M'}

# Aplicar o mapeamento
dados_filtrados['Cluster_Label_f_fc'] = dados_filtrados['Cluster_KMeans_f_fc'].map(cluster_mapping_f_fc)

# Visualizar os clusters comparados com Wear
plt.figure(figsize=(4, 3))
sns.scatterplot(x=dados_filtrados['f [mm/rev]'], y=dados_filtrados['Fc [N]'],
                hue=dados_filtrados['Cluster_Label_f_fc'], palette="Set1", marker="o", edgecolor="k")
plt.title("K-Means Clusters (Feed Rate vs Cutting Force)", fontsize=14)
plt.xlabel("f [mm/rev]")
plt.ylabel("Fc [N]")
plt.legend(title="Cluster", fontsize=8)
plt.grid()
plt.show()

```

```
Médias dos clusters (f vs Fc):
                     f [mm/rev]      Fc [N]
Cluster_KMeans_f_fc
0                      0.040523   93.877124
1                      0.158605  274.706202
```



L: Low (Baixo) – Indica um nível baixo de desgaste.

M: Medium (Médio) – Indica um nível médio de desgaste.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```python
# 1️⃣ Verificar os dados
print("\nTipos das colunas selecionadas:")
print(dados_filtrados.dtypes)

# 2️⃣ Codificar a variável alvo (Wear)
le = LabelEncoder()
dados_filtrados['Wear_encoded'] = le.fit_transform(dados_filtrados['Wear'])

```

```
Tipos das colunas selecionadas:
vc [m/min]             float64
f [mm/rev]             float64
Fc [N]                 float64
Wear                    object
Cluster_KMeans           int32
Cluster_Label           object
Cluster_KMeans_f_fc      int32
Cluster_Label_f_fc      object
dtype: object
```

```python
# Verificar a codificação
print("\nCodificação de 'Wear':")
print(dict(zip(le.classes_, le.transform(le.classes_))))
```

```
Codificação de 'Wear':
{'L': 0, 'M': 1}
```

```python
# 2️⃣ Selecionar 20 amostras aleatórias para previsão
dados_para_prever = dados_filtrados.sample(n=20, random_state=42)
dados_filtrados = dados_filtrados.drop(dados_para_prever.index)  # Remover essas amostras do treino/teste


```

```python
# 4️⃣ Separar variáveis independentes (X) e dependente (y)
X = dados_filtrados[['vc [m/min]', 'f [mm/rev]', 'Fc [N]']]
y = dados_filtrados['Wear_encoded']

```

```python
# 5️⃣ Dividir em treino e teste (80% treino, 20% teste, com estratificação)
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# 6️⃣ Balanceamento com SMOTE para lidar com a baixa quantidade de classe 'H'
smote = SMOTE(random_state=42, k_neighbors=2)  # Reduzindo k_neighbors para evitar erro
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

```

```
# 7️⃣ Normalizar os dados (StandardScaler)
scaler = StandardScaler()
X_train_res_scaled = scaler.fit_transform(X_train_res)
X_test_scaled = scaler.transform(X_test)
```

Criar modelos

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report

```

Show hidden output

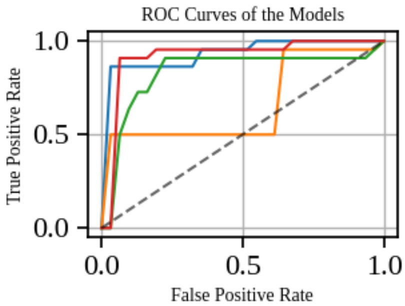
```
# 🔹 8. Definir os hiperparâmetros para cada modelo
param_rf = {'n_estimators': [50, 100, 200], 'max_depth': [5, 10, 20], 'min_samples_split': [5, 10], 'min_samples_leaf': [2, 5]}
param_svm = {'C': [0.1, 1, 10], 'kernel': ['rbf', 'linear']}
param_lr = {'C': [0.1, 1, 10], 'max_iter': [200, 500]}
param_lgbm = {'num_leaves': [31, 50], 'learning_rate': [0.01, 0.1], 'n_estimators': [50, 100, 200]}

# 🔹 9. Criar os modelos com hiperparâmetros otimizados via GridSearch
modelos = {
    "Random Forest": GridSearchCV(RandomForestClassifier(random_state=42), param_rf, cv=StratifiedKFold(n_splits=10), scoring='accura
    "SVM": GridSearchCV(SVC(probability=True, class_weight='balanced', random_state=42), param_svm, cv=StratifiedKFold(n_splits=10),
    "Logistic Regression": GridSearchCV(LogisticRegression(class_weight='balanced', random_state=42), param_lr, cv=StratifiedKFold(n_
    "LightGBM": GridSearchCV(LGBMClassifier(random_state=42), param_lgbm, cv=StratifiedKFold(n_splits=10), scoring='accuracy')
}

# 🔹 10. Treinar, avaliar e validar com cross-validation
resultados = {}

for nome, modelo in modelos.items():
    print(f"\n🔹 Treinando {nome}...")

    # Treinar o modelo com os melhores hiperparâmetros
    modelo.fit(X_train_res_scaled, y_train_res)
    melhor_modelo = modelo.best_estimator_

    # Previsões no treino e teste
    y_train_pred = melhor_modelo.predict(X_train_res_scaled)
    y_test_pred = melhor_modelo.predict(X_test_scaled)

    # Validação cruzada
    scores = cross_val_score(melhor_modelo, X_train_res_scaled, y_train_res, cv=StratifiedKFold(n_splits=10), scoring='accuracy')
    media_cv = scores.mean()

    # Exibir métricas
    print(f"\n📌 Melhor Modelo para {nome}: {modelo.best_params_}")
    print(f"\n📌 Métricas para {nome} (TREINO):")
    print(classification_report(y_train_res, y_train_pred, target_names=le.classes_))

    print(f"\n📌 Métricas para {nome} (TESTE):")
    print(classification_report(y_test, y_test_pred, target_names=le.classes_))

    print(f"\n📌 Validação Cruzada (Média de Acurácia): {media_cv:.4f}")

    # Armazenar resultados
    resultados[nome] = {
        "Melhor Modelo": modelo.best_params_,
        "Validação Cruzada": media_cv,
        "Treino": classification_report(y_train_res, y_train_pred, target_names=le.classes_, output_dict=True),
        "Teste": classification_report(y_test, y_test_pred, target_names=le.classes_, output_dict=True)
    }

```

Show hidden output

```
from scipy.interpolate import make_interp_spline

```

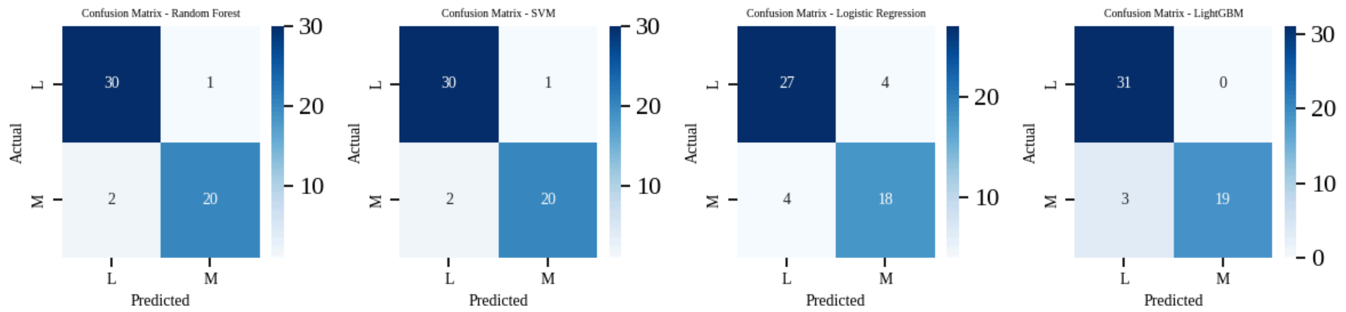
```
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve, auc
from scipy.interpolate import make_interp_spline
import matplotlib.pyplot as plt

# 🔹 1️⃣ ROC-AUC CURVES FOR MODELS (Smoothed)
plt.figure(figsize=(3, 2))

for name, model in modelos.items():
    best_model = model.best_estimator_
    y_test_prob = best_model.predict_proba(X_test_scaled)[:, 1]  # Probabilities for the positive class (M)
    fpr, tpr, _ = roc_curve(y_test, y_test_prob)  # FPR = False Positive Rate, TPR = True Positive Rate
    roc_auc = auc(fpr, tpr)

    # Remove duplicate values from fpr (X) and adjust tpr (Y)
    fpr_unique, indices = np.unique(fpr, return_index=True)
    tpr_unique = tpr[indices]


# Criando pontos intermediários para suavização
    fpr_smooth = np.linspace(fpr_unique.min(), fpr_unique.max(), 32)
    tpr_smooth = make_interp_spline(fpr_unique, tpr_unique, k=0)(fpr_smooth)  # Interpolação cúbica

    plt.plot(fpr_smooth, tpr_smooth, label=f"{name} (AUC = {roc_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--', alpha=0.6)  # Diagonal line (random classification)
plt.xlabel('False Positive Rate', fontsize=10)
plt.ylabel('True Positive Rate', fontsize=10)
plt.title('ROC Curves of the Models', fontsize=10)
#plt.legend(loc="lower right", fontsize=8)
plt.grid()
plt.show()

```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_
  warnings.warn(
```




```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# 🔹 2️⃣ CONFUSION MATRIX FOR EACH MODEL (ENGLISH + SMALLER FONT)
fig, axes = plt.subplots(1, 4, figsize=(12, 3))

for ax, (name, model) in zip(axes, modelos.items()):
    best_model = model.best_estimator_
    y_test_pred = best_model.predict(X_test_scaled)

    cm = confusion_matrix(y_test, y_test_pred)

    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=le.classes_, yticklabels=le.classes_, ax=ax,
                annot_kws={"size": 10})  # Reduce annotation font size

    ax.set_title(f"Confusion Matrix - {name}", fontsize=7)  # Reduce title font size
    ax.set_xlabel("Predicted", fontsize=10)  # Reduce x-label font size
    ax.set_ylabel("Actual", fontsize=10)  # Reduce y-label font size
    ax.tick_params(axis='both', which='major', labelsize=10)  # Reduce tick labels

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to 'ensure_a
  warnings.warn(
```




```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import learning_curve, StratifiedKFold

# 🔹 3️⃣ LEARNING CURVES (SHOWING 10 POINTS)
fig, axes = plt.subplots(2, 2, figsize=(8, 5))  # 2x2 layout for better spacing

axes = axes.ravel()  # Flatten axes array for easier iteration

for ax, (name, model) in zip(axes, modelos.items()):
    best_model = model.best_estimator_

    # Define 10 points for the training set
    train_sizes, train_scores, test_scores = learning_curve(
        best_model, X_train_res_scaled, y_train_res,
        train_sizes=np.linspace(0.1, 1.0, 10),  # 10 points
        cv=StratifiedKFold(n_splits=10),
        scoring='accuracy'
    )

    train_mean = np.mean(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)

    ax.plot(train_sizes, train_mean, 'o-', label="Training")
    ax.plot(train_sizes, test_mean, 'o-', label="Validation")

    ax.set_title(f"Learning Curve - {name}", fontsize=14)
    ax.set_xlabel("Training Size", fontsize=12)
    ax.set_ylabel("Accuracy", fontsize=12)
    ax.legend(fontsize=12)
    ax.tick_params(axis='both', labelsize=12)  # Reduce tick label size
    ax.grid(False)

# Adjust layout to prevent overlapping
plt.tight_layout()
plt.show()
```