2º Trabalho de Aprendizagem Automática

Neste trabalho foi nos entregue o desafio de desenvolver modelos de modo a prever o tipo de cobertura florestal apartir de variaveis cartograficas. As áreas de estudo representam florestas com pertubrações minimas causadas pelo homem pelo que, os tipos de cobertura florestal existentes são o resultado de processos ecologicos.

1. **Tratamento de dados e amostras**
    A. Dados de Treino
    B. Dados de Teste
    C. Distribuição dos dados
    D. Correlações e Relações
    E. Divisão dos Conjuntos
2. **Afinação de Hiperparametros**
    A. Random Forrest Classifier
    B. Gradient Boosting Classifier
    C. MLP Classifier
    D. Logistic Regression
    E. KNNeighbors
3. **Melhores Modelos**
    A. Random Forrest 1
    B. Random Forrest 2
4. **Outros Modelos**
    A. Gradient Boosting Classifier
    B. MLP Classifier
    C. Logistic Regression
    D. KNNeigbhors
5. **Conclusão**

Imports

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV,learning_curve
from sklearn.metrics import confusion_matrix,classification_report,accura
```

2. Tratamento de dados e amostras

Nesta seção, procedemos a uma análise detalhada dos conjuntos de treino e teste. O

primeiro passo foi realizar um **resumo estatistico** de modo a obtermos uma visão global das estatisticas basicas de cada variavel como **media,desvio-padrão,valores minimos e maximos** de modo a melhor comprendermos o tipo de dados com que estamos a lidar. Seguidamente realizarmos um estudo visual sobre as **correlações e distribuições** dos dados de modo a identificarmos as relações lineares entre as variaveis independentes que ajuda a detetar redundancias que podem vir a ser problematicas no desenvolvimento dos modelos.

In [2]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [24]:
```
def describe_data(df):
    print("Tipos: ")
    print(df.dtypes)
    print("Linhas e Colunas: ")
    print(df.shape)
    print("Colunas: ")
    print(df.columns)
    print("Valores Vazios: ")
    print(df.apply(lambda x: sum(x.isnull()/len(df))))
```

Dados de Treino:

O conjunto de dados contém **10620 linhas e 14 colunas**, e **não possui valores ausentes**

Análise das Variáveis

1. **id**

- **Descrição**: Identificador único para cada linha.
- **Distribuição**: Uniforme, com valores variando entre 1 e 10620.
- **Estatísticas**:
  - Média: 5310.5
  - Desvio padrão: 3065.87
  - Mínimo: 1
  - Máximo: 10620

2. **elevacao**

- **Descrição**: Altitude do ponto.
- **Distribuição**: Concentrada entre 1879 e 3849.
- **Estatísticas**:
  - Média: 2752.12
  - Mediana: 2755
  - Desvio padrão: 417.88
  - Mínimo: 1879
  - Máximo: 3849

3. **aspeto**

- **Descrição**: Orientação do terreno em graus.
- **Distribuição**: Varia entre 0 e 360 (azimute completo).
- **Estatísticas**:
  - Média: 156.57
  - Mediana: 125
  - Desvio padrão: 110.02
  - Mínimo: 0
  - Máximo: 360

4. **inclinacao**

- **Descrição**: Inclinação do terreno em graus.
- **Distribuição**: Concentração predominante em valores baixos (0 a 52 graus).
- **Estatísticas**:
  - Média: 16.57
  - Mediana: 15
  - Desvio padrão: 8.48
  - Mínimo: 0
  - Máximo: 52

5. **dh_agua**

- **Descrição**: Distância horizontal até o corpo de água mais próximo.
- **Distribuição**: Assimétrica, com maior concentração em valores baixos (até 500).
- **Estatísticas**:
  - Média: 228.43
  - Mediana: 180
  - Desvio padrão: 209.46
  - Mínimo: 0
  - Máximo: 1343

6. **dv_agua**

- **Descrição**: Diferença de altura (vertical) em relação ao corpo de água mais próximo.
- **Distribuição**: Predominância próxima de 0, com valores negativos ocasionais.
- **Estatísticas**:
  - Média: 51.81
  - Mediana: 33
  - Desvio padrão: 61.29
  - Mínimo: -134
  - Máximo: 554

7. **dh_estrada**

- **Descrição**: Distância horizontal até a estrada mais próxima.
- **Distribuição**: Assimétrica, com maior densidade em valores baixos, mas estendendo-se até 6890.
- **Estatísticas**:

- Média: 1723.08
- Mediana: 1318
- Desvio padrão: 1329.50
- Mínimo: 0
- Máximo: 6890

8. **sombra_9, sombra_12 e sombra_15**

- **Descrição**: Quantidade de sombra às 9h, 12h e 15h.
- **Distribuição**: Valores variam entre 0 e 254, com maior concentração em valores intermediários.
- **Estatísticas (exemplo para sombra_9)**:
  - Média: 212.71
  - Mediana: 220
  - Desvio padrão: 30.61
  - Mínimo: 0
  - Máximo: 254

9. **dh_Incendio**

- **Descrição**: Distância horizontal até o ponto de ignição de incêndios.
- **Distribuição**: Assimétrica, com valores entre 0 e 6853.
- **Estatísticas**:
  - Média: 1516.79
  - Mediana: 1260
  - Desvio padrão: 1111.75
  - Mínimo: 0
  - Máximo: 6853

10. **area**

- **Descrição**: Classe de área.
- **Distribuição**: Valores discretos variando entre 1 e 4.
- **Estatísticas**:
  - Média: 2.2
  - Mediana: 2
  - Desvio padrão: 1.11
  - Mínimo: 1
  - Máximo: 4

11. **solo**

- **Descrição**: Tipo de solo.
- **Distribuição**: Discreta, com valores entre 1 e 21.
- **Estatísticas**:
  - Média: 9.7
  - Mediana: 11
  - Desvio padrão: 6.03

- Mínimo: 1
- Máximo: 21

12. **floresta**

- **Descrição**: Tipo de floresta.
- **Distribuição**: Discreta, variando entre 1 e 7.
- **Estatísticas**:
  - Média: 3.99
  - Mediana: 4
  - Desvio padrão: 2.0
  - Mínimo: 1
  - Máximo: 7

In [43]:
```
describe_data(train)
print(train.describe())
train.head()
```

```
Tipos:
id              int64
elevacao        int64
aspeto          int64
inclinacao      int64
dh_agua         int64
dv_agua         int64
dh_estrada      int64
sombra_9        int64
sombra_12       int64
sombra_15       int64
dh_Incendio     int64
area            int64
solo            int64
floresta        int64
dtype: object
Linhas e Colunas:
(10620, 14)
Colunas:
Index(['id', 'elevacao', 'aspeto', 'inclinacao', 'dh_agua', 'dv_agua',
       'dh_estrada', 'sombra_9', 'sombra_12', 'sombra_15', 'dh_Incendio',
       'area', 'solo', 'floresta'],
      dtype='object')
Valores Vazios:
id              0.0
elevacao        0.0
aspeto          0.0
inclinacao      0.0
dh_agua         0.0
dv_agua         0.0
dh_estrada      0.0
sombra_9        0.0
sombra_12       0.0
sombra_15       0.0
dh_Incendio     0.0
area            0.0
solo            0.0
floresta        0.0
dtype: float64
                 id      elevacao        aspeto    inclinacao      dh_agua
\
count  10620.000000  10620.000000  10620.000000  10620.000000  10620.00000
mean    5310.500000   2752.124200    156.575047     16.578437    228.42580
std     3065.874264    417.881891    110.020251      8.481794    209.45953
min        1.000000   1879.000000      0.000000      0.000000      0.00000
25%     2655.750000   2378.000000     64.000000     10.000000     67.00000
50%     5310.500000   2755.000000    125.000000     15.000000    180.00000
75%     7965.250000   3109.000000    260.000000     22.000000    330.00000
max    10620.000000   3849.000000    360.000000     52.000000   1343.00000

            dv_agua    dh_estrada      sombra_9     sombra_12     sombra_1
5  \
count  10620.000000  10620.000000  10620.000000  10620.000000  10620.00000
0
mean      51.808945   1723.080226    212.710264    218.830414    134.86440
7
std       61.291132   1329.501289     30.615163     22.963430     46.22162
0
min     -134.000000      0.000000      0.000000     99.000000      0.00000
0
```

```
25%        5.000000    768.000000    196.000000    207.000000    106.00000
0
50%       33.000000   1318.000000    220.000000    222.000000    138.00000
0
75%       80.000000   2278.250000    235.000000    235.000000    167.00000
0
max      554.000000   6890.000000    254.000000    254.000000    247.00000
0

        dh_Incendio          area          solo      floresta
count  10620.000000  10620.000000  10620.000000  10620.000000
mean    1516.787571      2.198964      9.698776      3.985782
std     1111.750922      1.119837      6.038451      1.999785
min        0.000000      1.000000      1.000000      1.000000
25%      726.000000      1.000000      4.000000      2.000000
50%     1260.000000      2.000000     11.000000      4.000000
75%     1994.000000      3.000000     13.000000      6.000000
max     6853.000000      4.000000     21.000000      7.000000
```

Out[43]:

	id	elevacao	aspeto	inclinacao	dh_agua	dv_agua	dh_estrada	sombra_9	sombra_
0	1	2596	51	3	258	0	510	221	2
1	2	2785	155	18	242	118	3090	238	2
2	3	2579	132	6	300	-15	67	230	2
3	4	2606	45	7	270	5	633	222	2
4	5	2605	49	4	234	7	573	222	2

Dados de Teste

O conjunto de dados contém **10620 linhas e 14 colunas**, e **não possui valores ausentes**.

Análise das Variáveis

1. id

- **Descrição**: Identificador único para cada linha.
- **Distribuição**: Uniforme, com valores variando entre 1 e 10620.
- **Estatísticas**:
  - Média: 5310.5
  - Desvio padrão: 3065.87
  - Mínimo: 1
  - Máximo: 10620

2. elevacao

- **Descrição**: Altitude do ponto.
- **Distribuição**: Concentrada entre 1879 e 3849 metros.
- **Estatísticas**:
  - Média: 2752.12
  - Mediana: 2755
  - Desvio padrão: 417.88

- Mínimo: 1879
- Máximo: 3849

3. aspeto

- **Descrição**: Orientação do terreno em graus.
- **Distribuição**: Varia entre 0 e 360 (azimute completo).
- **Estatísticas**:
  - Média: 156.57
  - Mediana: 125
  - Desvio padrão: 110.02
  - Mínimo: 0
  - Máximo: 360

4. inclinacao

- **Descrição**: Inclinação do terreno em graus.
- **Distribuição**: Concentração predominante em valores baixos (0 a 52 graus).
- **Estatísticas**:
  - Média: 16.57
  - Mediana: 15
  - Desvio padrão: 8.48
  - Mínimo: 0
  - Máximo: 52

5. dh_agua

- **Descrição**: Distância horizontal até o corpo de água mais próximo.
- **Distribuição**: Assimétrica, com maior concentração em valores baixos (até 500 metros).
- **Estatísticas**:
  - Média: 228.43
  - Mediana: 180
  - Desvio padrão: 209.46
  - Mínimo: 0
  - Máximo: 1343

6. dv_agua

- **Descrição**: Diferença de altura (vertical) em relação ao corpo de água mais próximo.
- **Distribuição**: Predominância próxima de 0, com valores negativos ocasionais.
- **Estatísticas**:
  - Média: 51.81
  - Mediana: 33
  - Desvio padrão: 61.29
  - Mínimo: -134
  - Máximo: 554

7. dh_estrada

- **Descrição**: Distância horizontal até a estrada mais próxima.
- **Distribuição**: Assimétrica, com maior densidade em valores baixos, mas estendendo-se até 6890 metros.
- **Estatísticas**:
  - Média: 1723.08
  - Mediana: 1318
  - Desvio padrão: 1329.50
  - Mínimo: 0
  - Máximo: 6890

8. sombra_9, sombra_12 e sombra_15

- **Descrição**: Quantidade de sombra às 9h, 12h e 15h.
- **Distribuição**: Valores variam entre 0 e 254, com maior concentração em valores intermediários.
- **Estatísticas (exemplo para sombra_9)**:
  - Média: 212.71
  - Mediana: 220
  - Desvio padrão: 30.61
  - Mínimo: 0
  - Máximo: 254

9. dh_Incendio

- **Descrição**: Distância horizontal até o ponto de ignição de incêndios.
- **Distribuição**: Assimétrica, com valores entre 0 e 6853 metros.
- **Estatísticas**:
  - Média: 1516.79
  - Mediana: 1260
  - Desvio padrão: 1111.75
  - Mínimo: 0
  - Máximo: 6853

10. area

- **Descrição**: Classe de área.
- **Distribuição**: Valores discretos variando entre 1 e 4.
- **Estatísticas**:
  - Média: 2.2
  - Mediana: 2
  - Desvio padrão: 1.11
  - Mínimo: 1
  - Máximo: 4

11. solo

- **Descrição**: Tipo de solo.
- **Distribuição**: Discreta, com valores entre 1 e 21.

- **Estatísticas**:
  - Média: 9.7
  - Mediana: 11
  - Desvio padrão: 6.03
  - Mínimo: 1
  - Máximo: 21

12. floresta

- **Descrição**: Tipo de floresta.
- **Distribuição**: Discreta, variando entre 1 e 7.
- **Estatísticas**:
  - Média: 3.99
  - Mediana: 4
  - Desvio padrão: 2.0
  - Mínimo: 1
  - Máximo: 7

In [44]:
```
describe_data(test)
print(test.describe())
test.head()
```

```
Tipos:
id              int64
elevacao        int64
aspeto          int64
inclinacao      int64
dh_agua         int64
dv_agua         int64
dh_estrada      int64
sombra_9        int64
sombra_12       int64
sombra_15       int64
dh_Incendio     int64
area            int64
solo            int64
dtype: object
Linhas e Colunas:
(4500, 13)
Colunas:
Index(['id', 'elevacao', 'aspeto', 'inclinacao', 'dh_agua', 'dv_agua',
       'dh_estrada', 'sombra_9', 'sombra_12', 'sombra_15', 'dh_Incendio',
       'area', 'solo'],
      dtype='object')
Valores Vazios:
id              0.0
elevacao        0.0
aspeto          0.0
inclinacao      0.0
dh_agua         0.0
dv_agua         0.0
dh_estrada      0.0
sombra_9        0.0
sombra_12       0.0
sombra_15       0.0
dh_Incendio     0.0
area            0.0
solo            0.0
dtype: float64
                 id     elevacao       aspeto   inclinacao      dh_agua  \
count   4500.000000  4500.000000  4500.000000  4500.000000  4500.000000
mean   12870.500000  2742.710667   156.916444    16.320222   224.292667
std     1299.182435   417.168852   110.252212     8.385939   211.516397
min    10621.000000  1863.000000     0.000000     1.000000     0.000000
25%    11745.750000  2369.000000    66.000000    10.000000    60.000000
50%    12870.500000  2742.000000   126.000000    15.000000   175.000000
75%    13995.250000  3090.000000   261.000000    22.000000   324.000000
max    15120.000000  3849.000000   359.000000    50.000000  1294.000000

          dv_agua   dh_estrada     sombra_9    sombra_12    sombra_15  \
count  4500.00000  4500.000000  4500.000000  4500.000000  4500.000000
mean     49.34800  1692.648667   212.690222   219.284667   135.629111
std      61.08915  1314.440219    30.437151    22.415594    45.115982
min    -146.00000     0.000000    83.000000    99.000000     0.000000
25%       4.00000   743.000000   196.000000   207.000000   108.000000
50%      31.00000  1302.000000   220.000000   223.000000   138.000000
75%      77.00000  2258.250000   235.000000   235.000000   167.000000
max     403.00000  6766.000000   254.000000   254.000000   248.000000

       dh_Incendio         area         solo
count  4500.000000  4500.000000  4500.000000
mean   1497.836222     2.201111     9.559333
```

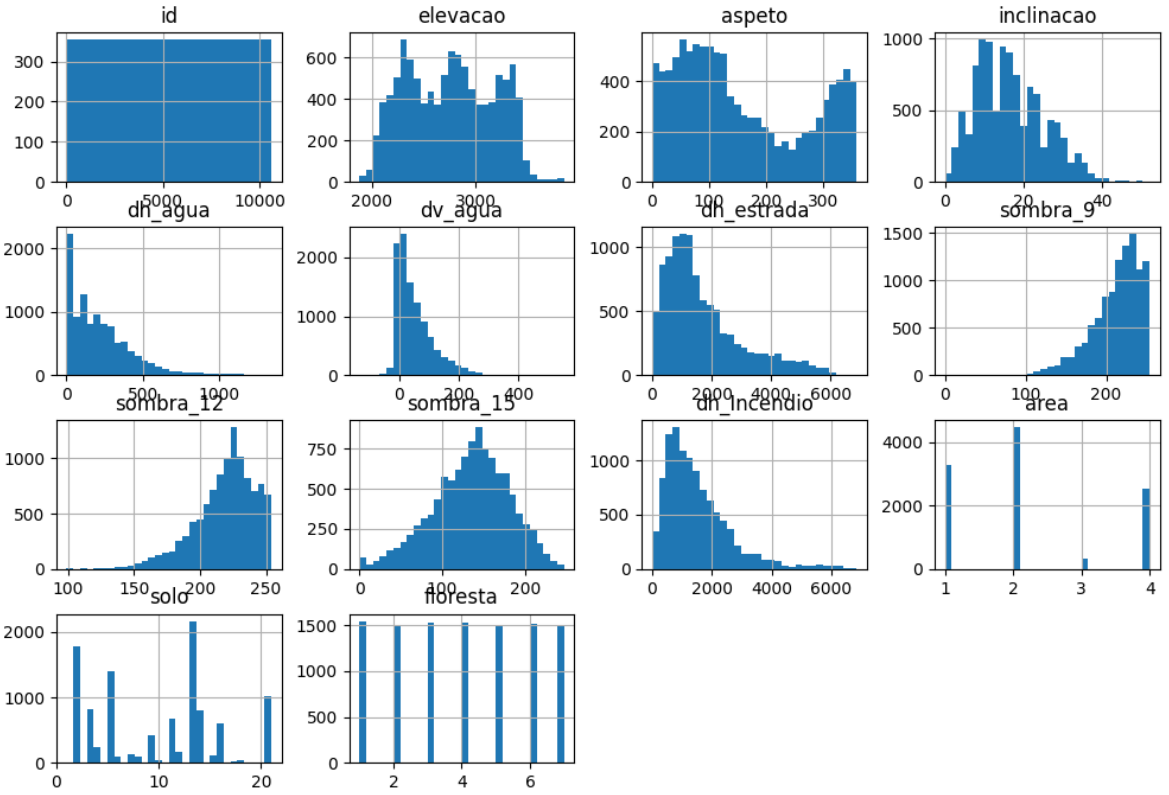
```
std     1071.542160      1.119945      6.009904
min       30.000000      1.000000      1.000000
25%      732.000000      1.000000      4.000000
50%     1249.000000      2.000000     11.000000
75%     1973.250000      3.000000     13.000000
max     6993.000000      4.000000     21.000000
```

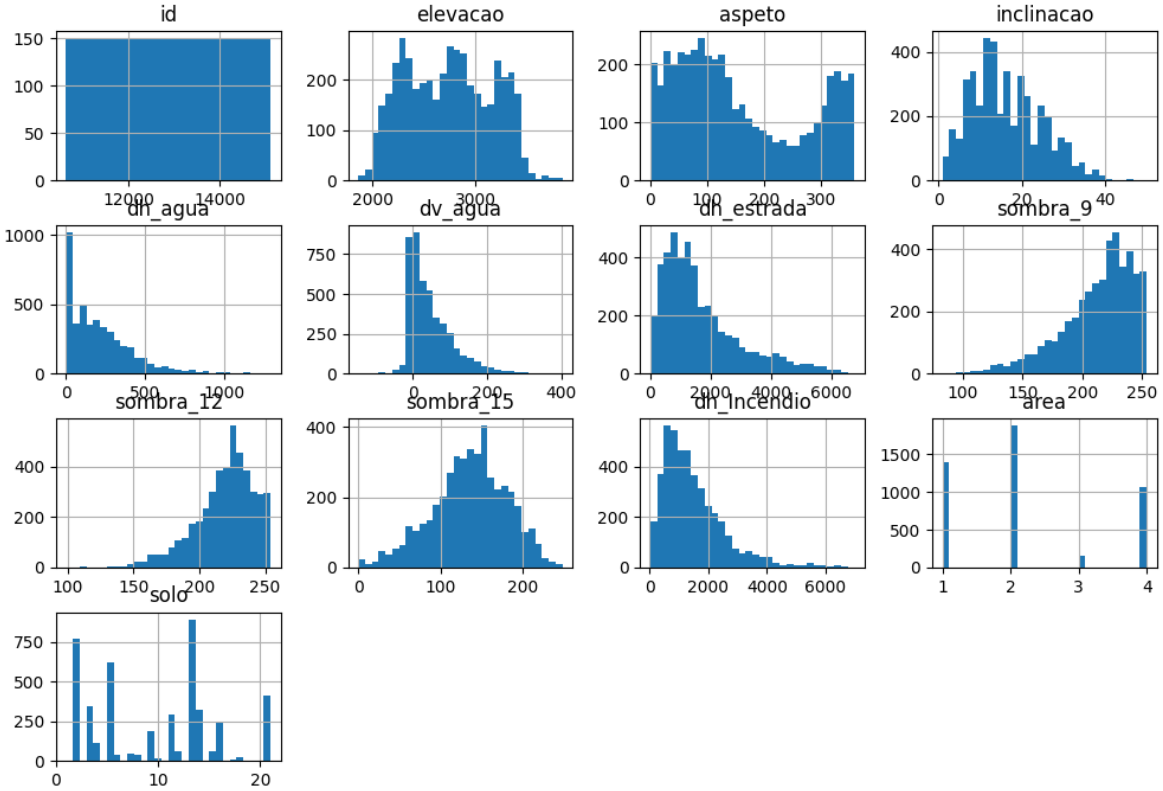
Out[44]:

	id	elevacao	aspeto	inclinacao	dh_agua	dv_agua	dh_estrada	sombra_9	som
0	10621	2703	330	27	30	17	3141	146	
1	10622	2524	94	7	212	-4	684	232	
2	10623	2536	99	6	234	0	659	230	
3	10624	2489	11	4	175	13	840	216	
4	10625	2493	63	10	127	20	840	229	

Distribuição dos Dados

In [33]:
```
train.hist(bins=30,figsize=(12,8))
plt.show()
test.hist(bins=30,figsize=(12,8))
plt.show()
```

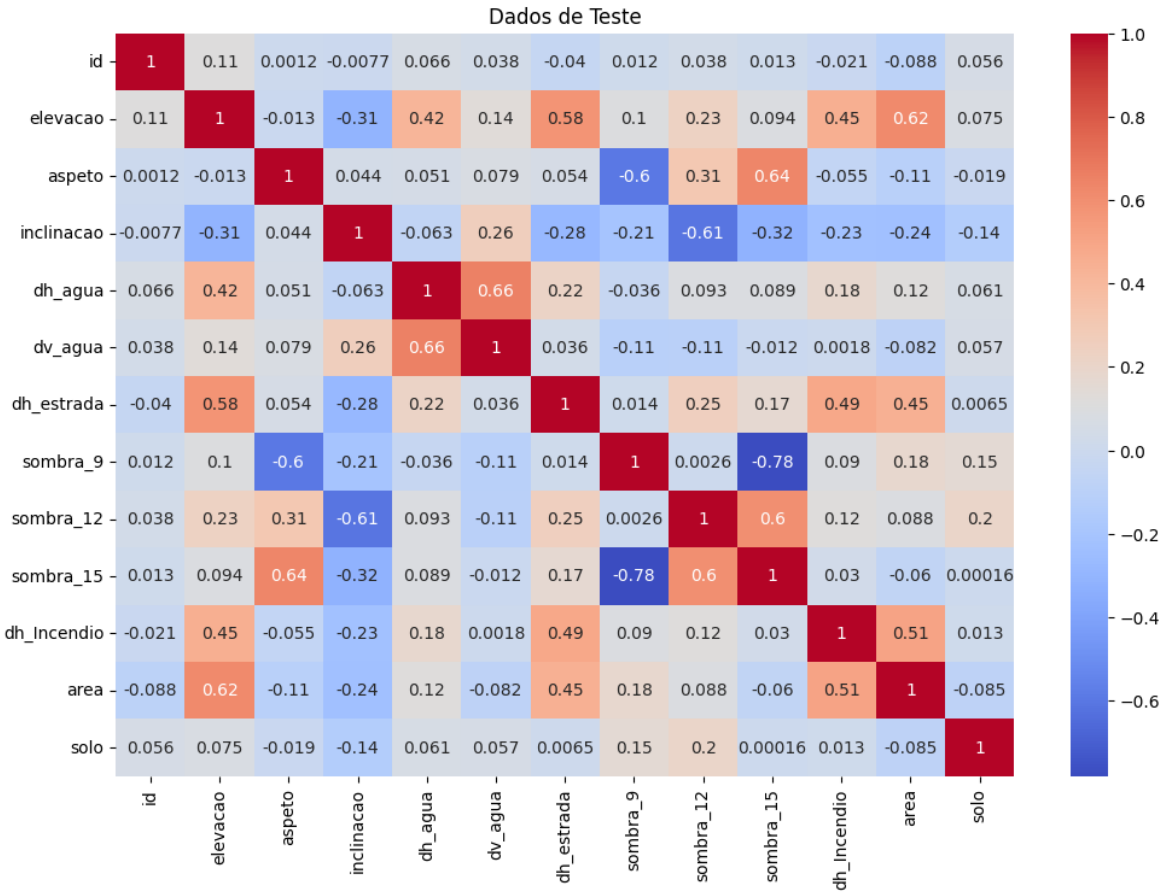


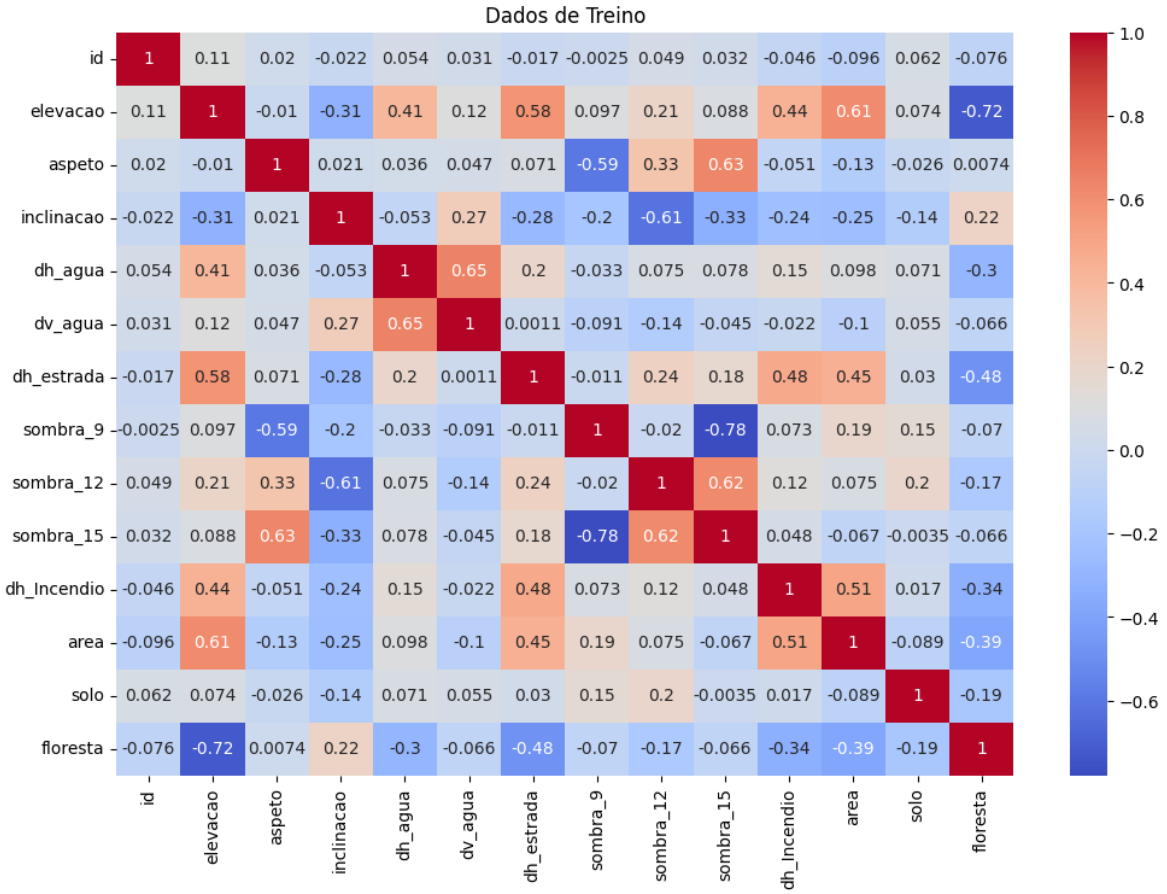


Correlações e Relações

In [45]:
```
cor_matrix1 = test.corr()
plt.figure(figsize=(12,8))
sns.heatmap(cor_matrix1,annot=True,cmap='coolwarm',)
plt.title("Dados de Teste")
plt.show()

cor_matrix2 = train.corr()
plt.figure(figsize=(12,8))
sns.heatmap(cor_matrix2,annot=True,cmap='coolwarm')
plt.title("Dados de Treino")
plt.show()
```





Divisão dos conjuntos

In [42]:
```
from sklearn.model_selection import train_test_split
X = train.iloc[:, :-1].values
y = train.iloc[:, -1].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,ran

print(f"Conjunto de treino: {len(X_train)} amostras")
print(train.describe())
```

```
Conjunto de treino: 8496 amostras
                id      elevacao        aspeto    inclinacao      dh_agua
\
count  10620.000000  10620.000000  10620.000000  10620.000000  10620.00000
mean    5310.500000   2752.124200    156.575047     16.578437    228.42580
std     3065.874264    417.881891    110.020251      8.481794    209.45953
min        1.000000   1879.000000      0.000000      0.000000      0.00000
25%     2655.750000   2378.000000     64.000000     10.000000     67.00000
50%     5310.500000   2755.000000    125.000000     15.000000    180.00000
75%     7965.250000   3109.000000    260.000000     22.000000    330.00000
max    10620.000000   3849.000000    360.000000     52.000000   1343.00000

            dv_agua    dh_estrada      sombra_9     sombra_12     sombra_1
5  \
count  10620.000000  10620.000000  10620.000000  10620.000000  10620.00000
0
mean      51.808945   1723.080226    212.710264    218.830414    134.86440
7
std       61.291132   1329.501289     30.615163     22.963430     46.22162
0
min     -134.000000      0.000000      0.000000     99.000000      0.00000
0
25%        5.000000    768.000000    196.000000    207.000000    106.00000
0
50%       33.000000   1318.000000    220.000000    222.000000    138.00000
0
75%       80.000000   2278.250000    235.000000    235.000000    167.00000
0
max      554.000000   6890.000000    254.000000    254.000000    247.00000
0

        dh_Incendio          area          solo      floresta
count  10620.000000  10620.000000  10620.000000  10620.000000
mean    1516.787571      2.198964      9.698776      3.985782
std     1111.750922      1.119837      6.038451      1.999785
min        0.000000      1.000000      1.000000      1.000000
25%      726.000000      1.000000      4.000000      2.000000
50%     1260.000000      2.000000     11.000000      4.000000
75%     1994.000000      3.000000     13.000000      6.000000
max     6853.000000      4.000000     21.000000      7.000000
```

In [41]:
```
print(f"Conjunto de treino: {len(X_test)} amostras")
print(test.describe())
```

```
Conjunto de treino: 2124 amostras
                id     elevacao       aspeto   inclinacao      dh_agua  \
count   4500.000000  4500.000000  4500.000000  4500.000000  4500.000000
mean   12870.500000  2742.710667   156.916444    16.320222   224.292667
std     1299.182435   417.168852   110.252212     8.385939   211.516397
min    10621.000000  1863.000000     0.000000     1.000000     0.000000
25%    11745.750000  2369.000000    66.000000    10.000000    60.000000
50%    12870.500000  2742.000000   126.000000    15.000000   175.000000
75%    13995.250000  3090.000000   261.000000    22.000000   324.000000
max    15120.000000  3849.000000   359.000000    50.000000  1294.000000

          dv_agua   dh_estrada     sombra_9    sombra_12    sombra_15  \
count  4500.00000  4500.000000  4500.000000  4500.000000  4500.000000
mean     49.34800  1692.648667   212.690222   219.284667   135.629111
std      61.08915  1314.440219    30.437151    22.415594    45.115982
min    -146.00000     0.000000    83.000000    99.000000     0.000000
25%       4.00000   743.000000   196.000000   207.000000   108.000000
50%      31.00000  1302.000000   220.000000   223.000000   138.000000
75%      77.00000  2258.250000   235.000000   235.000000   167.000000
max     403.00000  6766.000000   254.000000   254.000000   248.000000

       dh_Incendio         area         solo
count  4500.000000  4500.000000  4500.000000
mean   1497.836222     2.201111     9.559333
std    1071.542160     1.119945     6.009904
min      30.000000     1.000000     1.000000
25%     732.000000     1.000000     4.000000
50%    1249.000000     2.000000    11.000000
75%    1973.250000     3.000000    13.000000
max    6993.000000     4.000000    21.000000
```

3. Afinação de hiperparametros

Nesta secção podemos encontrar o processo de encontrar os melhores parametros para os varios modelos implementados. Recorremos ao metodo de **GridSearchCV** que permite uma automatização da busca dos melhores hiperparametros, **testando todas as combinações possiveis de hiperparametros**, garantindo **resultados mais robustos com menor risco de overfitting**.

Para todos os modelos a busca foi realizada com **5 folds** ou seja, dividindo o conjunto de treino em 5 partes com 4 dessas divisões utilizadas para treinar o modelo e a 5ª utilizada para validar o modelo. Tambem utilizamos a **verbose com um valor de 2** que permite monotorizar o progresso com mais informações disponiveis, e por fim utilizamos todos os cores disponiveis para executar o grid search com a definição do **n_jobs como -1**.

Random Forrest Classifier

Hiperparametros

Os hiperparâmetros que serão ajustados no modelo de Random Forest estão definidos no dicionário param_grid. Estes incluem:

1. **n_estimators** : Numero de arvores na floresta.
2. **criterion** : Função de avaliação para dividir os nós da árvore.
3. **max_depth** : Profundidade máxima de cada arvore.

4. **min_samples_split** : Numero minimo de amostras necessario para dividir um nó.
5. **min_samples_leaf**: Numero minimo de amostras que um nó folha deve conter.
6. **max_features** : Numero maximo de atributos considerados para encontrar a melhor divisão.
7. **bootstrap** : Indica se será usada amostragem com reposição.

In [59]:
```
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators':[1500,2000,2500],
    'max_depth':[None],
    'max_leaf_nodes': [1000,1500,2000],
    'min_samples_split':[2,3,4],
    'min_samples_leaf':[2,3,4]
}
rf_Model = RandomForestClassifier(random_state=42, bootstrap=True)
```

In [60]:
```
rf_grid_search = GridSearchCV(
    estimator=rf_Model,
    param_grid=param_grid,
    cv=5,
    verbose=2,
    n_jobs=-1
)
rf_grid_search.fit(X_train, y_train)
print("Melhores Parametros: ", rf_grid_search.best_params_)
print("Melhor Exatidão: ",rf_grid_search.best_score_)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[60], line 8
      1 rf_grid_search = GridSearchCV(
      2     estimator=rf_Model,
      3     param_grid=param_grid,
   (...)
      6     n_jobs=-1
      7 )
----> 8 rf_grid_search.fit(X_train, y_train)
      9 print("Melhores Parametros: ", rf_grid_search.best_params_)
     10 print("Melhor Exatidão: ",rf_grid_search.best_score_)

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/base.py:1473, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1019, in BaseSearchCV.fit(self, X, y, **params)
   1013     results = self._format_results(
   1014         all_candidate_params, n_splits, all_out, all_more_results
   1015     )
   1017     return results
-> 1019 self._run_search(evaluate_candidates)
   1021 # multimetric is determined here because in the case of a callable
   1022 # self.scoring the return type is only known after calling
   1023 first_test_score = all_out[0]["test_scores"]

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1573, in GridSearchCV._run_search(self, evaluate_candidates)
   1571 def _run_search(self, evaluate_candidates):
   1572     """Search all candidates in param grid"""
-> 1573     evaluate_candidates(ParameterGrid(self.param_grid))

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/model_selection/_search.py:965, in BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv, more_results)
    957 if self.verbose > 0:
    958     print(
    959         "Fitting {0} folds for each of {1} candidates,"
    960         " totalling {2} fits".format(
    961             n_splits, n_candidates, n_candidates * n_splits
    962         )
    963     )
--> 965 out = parallel(
    966     delayed( fit and score)(
    967         clone(base_estimator),
    968         X,
    969         y,
    970         train=train,
```

```
    971         test=test,
    972         parameters=parameters,
    973         split_progress=(split_idx, n_splits),
    974         candidate_progress=(cand_idx, n_candidates),
    975         **fit_and_score_kwargs,
    976     )
    977     for (cand_idx, parameters), (split_idx, (train, test)) in prod
uct(
    978         enumerate(candidate_params),
    979         enumerate(cv.split(X, y, **routed_params.splitter.split)),
    980     )
    981 )
    983 if len(out) < 1:
    984     raise ValueError(
    985         "No fits were performed. "
    986         "Was the CV iterator empty? "
    987         "Were there no candidates?"
    988     )

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/utils/parall
el.py:74, in Parallel.__call__(self, iterable)
     69 config = get_config()
     70 iterable_with_config = (
     71     (_with_config(delayed_func, config), args, kwargs)
     72     for delayed_func, args, kwargs in iterable
     73 )
---> 74 return super().__call__(iterable_with_config)

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/joblib/parallel.py:2
007, in Parallel.__call__(self, iterable)
   2001 # The first item from the output is blank, but it makes the interp
reter
   2002 # progress until it enters the Try/Except block of the generator a
nd
   2003 # reaches the first `yield` statement. This starts the asynchronou
s
   2004 # dispatch of the tasks to the workers.
   2005 next(output)
-> 2007 return output if self.return_generator else list(output)

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/joblib/parallel.py:1
650, in Parallel._get_outputs(self, iterator, pre_dispatch)
   1647     yield
   1649     with self._backend.retrieval_context():
-> 1650         yield from self._retrieve()
   1652 except GeneratorExit:
   1653     # The generator has been garbage collected before being fully
   1654     # consumed. This aborts the remaining tasks if possible and wa
rn
   1655     # the user if necessary.
   1656     self._exception = True

File ~/uni/4ano/aa/.venv/lib/python3.12/site-packages/joblib/parallel.py:1
762, in Parallel._retrieve(self)
   1757 # If the next job is not ready for retrieval yet, we just wait for
   1758 # async callbacks to progress.
   1759 if ((len(self._jobs) == 0) or
   1760     (self._jobs[0].get_status(
   1761         timeout=self.timeout) == TASK_PENDING)):
-> 1762     time.sleep(0.01)
```

```
            continue
        # We need to be careful: the job list can be filling up as
        # we empty it and Python list are not thread-safe by
        # default hence the use of the lock

    KeyboardInterrupt:
```

In [ ]:
```
results_df = pd.DataFrame(rf_grid_search.cv_results_)
top_results1 = results_df.sort_values(by='mean_test_score', ascending=Fal
print(top_results1[['mean_test_score', 'params']].head(10))

heatmap_data1 = results_df.pivot_table(
    index='param_max_depth',
    columns='param_n_estimators',
    values='mean_test_score'
    aggfunc='mean'
)
plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data1, annot=True, cmap='viridis', fmt=".4f")
plt.title("HeatMap n_estimators x max_depth")
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.show()
```

Gradient Boosting Classifier

HiperParametros

Os hiperparametros que serão ajustados para o Gradient Boosting Classifier estão definidos no dicionario **param_grid2** e são:

1. **learning_rate**: Taxa de aprendizagem que controla o peso atribuido a cada estimados.
2. **max_depth**: Profundidade máxima de cada arvore.
3. **max_features** : Numero maximo de features considerados para dividir um nó.
4. **min_samples_leaf**: Numero mínimo de amostras necessárias em uma folha.
5. **min_samples_split**: Numero minimo de amostras necessárias para dividir um nó.
6. **n_estimators**: Numero total de arvores no modelo.
7. **subsample**: Fração de amostras usadas para treinar cada arvore.

In [48]:
```
from sklearn.ensemble import GradientBoostingClassifier
param_grid2 = {
    'n_estimators':[500,1000,1500,2000],
    'max_depth':[10,20,None],
    'max_leaf_nodes':[50,100,100]
}
gbc_model = GradientBoostingClassifier()
```

In [ ]:
```
gbc_grid = GridSearchCV(
    estimator= gbc_model,
    param_grid=param_grid2,
```

```
    cv=5,
    verbose=2,
    n_jobs=-1,

)
gbc_grid.fit(X_train,y_train)
print("Melhor Parametros: ", gbc_grid.best_params_)
print("Melhor Exatidão: ", gbc_grid.best_score_)
```

In [ ]:
```
results_df_GBC = pd.DataFrame(gbc_grid.cv_results_)
top_results2 = results_df_GBC.sort_values(by='mean_test_score', ascending
print(top_results2[['mean_test_score', 'params']].head(10))
heatmap_data2 = results_df_GBC.pivot_table(
    index='parm_learning_rate',
    columns='param_n_estimators',
    values='mean_test_score'
)
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data1, annot=True, cmap='viridis', fmt=".4f")
plt.title("HeatMap n_estimators x learning_rate")
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.show()
```

MLP Classifier

Hiperparâmetros

Os hiperparametros que serão afinados para MLP estão definidos no dicionario **param_grid3** e são:

1. **hidden_layer_sizes**: Define o número de neurons em cada hidden layer.
2. **learning_rate_init**: Especifica a taxa de aprendizagem inicial para o otimizador.
3. **learning_rate**: Determina como a taxa de aprendizagem será ajustada ao longo do treino.
4. **validation_fraction**: Proporção de dados de treino reservados para validação interna.
5. **n_iter_no_changes** : Numero maximo de iterações consecutivas sem melhorias na validação.

In [49]:
```
from sklearn.neural_network import MLPClassifier

param_grid3 = {
    'hidden_layer_sizes': [(150, 100, 50), (200, 100, 50), (200,200), (30
    'learning_rate_init': [0.001, 0.0001, 0.01],
    'learning_rate': ['adaptive'],
    'validation_fraction': [0.1]
}
mlpc_model = MLPClassifier(random_state=42,activation='relu',solver='adam
```

In [ ]:
```
mlp_grid = GridSearchCV(
    estimator=mlpc_model,
    param_grid=param_grid3,
    cv=5,
    verbose=3,
    n_jobs=-1,
```

```
)
mlp_grid.fit(X_train, y_train)
print("Melhores Parametros: ", mlp_grid.best_params_)
print("Melhor Exatidão: ", mlp_grid.best_score_)
```

In [ ]:
```
results_df_MLP = pd.DataFrame(mlp_grid.cv_results_)
top_results3 = results_df_MLP.sort_values(by='mean_test_score' , ascendin
print(top_results3[['mean_test_score', 'params']].head(10))
heatmap_data3 = results_df_MLP.pivot_table(
    index='parm_hidden_layer_sizes',
    columns='param_learning_rate',
    values='mean_test_score'
)
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data1, annot=True, cmap='viridis', fmt=".4f")
plt.title("HeatMap hidden_layer_sizes x learning_rate")
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.show()
```

Logistic Regression

Hiperparametros

Os hiperparametros encontram se definidos no dicionario **param_grid4** e são:

1. **C** : Controlador da regularização no modelo
2. **penalty**: Tipo de penalidade usado
3. **solver**: Define o algoritmo para otimizar o modelo
4. **class_weight**: Trata do balanceamento das classes

In [50]:
```
from sklearn.linear_model import LogisticRegression
param_grid4 = {
    'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__solver': ['liblinear', 'saga'],
    'classifier__max_iter': [100, 500, 1000],
    'classifier__class_weight': [None, 'balanced']
}
LR_model = LogisticRegression()
```

In [ ]:
```
lr_grid = GridSearchCV(
    LR_model,
    param_grid=param_grid4,
    cv=5,
    scoring='accuracy',
    verbose=3,
    n_jobs=-1,
)
lr_grid.fit(X_train, y_train)
print("Melhores Parametros: ", lr_grid.best_params_)
print("Melhor Exatidão: ", lr_grid.best_score_)
```

In [ ]:
```
results_df_LR = pd.DataFrame(lr_grid.cv_results_)
top_results4= results_df_LR.sort_values(by='mean_test_score', ascending=F
print(top_results4[['mean_test_score', 'params']].head(10))
heatmap_data4= results_df_LR.pivot_table(
```

```
    index='param_classifier_C',
    columns='params_classifier_penalty',
    values='mean_test_score'
)
sns.heatmap(heatmap_data4,annot=True,cmap='viridis', fmt=".4f")
plt.title("Heatmap C x penalty")
plt.xlabel("Classifer C")
plt.ylabel("Classifier penalty")
plt.show()
```

KNN

Hiperparametros

Os parametros encontram se definidos dentro do dicionario **param_grid5** e são:

1. **n_neighbors**: Numero de vizinhos mais proximos a serem considerados pelo classificador
2. **weights**: Controla a forma como os vizinhos mais proximos contribuem para a classificação de um novo ponto
3. **algorithm**: Define o algoritmo para calcular os vizinhos mais proximos
4. **leaf_size** : Parametro apenas contablizado nos algoritmos de **ball_tree** e **kd_tree**
5. **p**: Determina a distancia usada para medir a distancia entre pontos

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
param_grid5 = {
    'n_neighbors': [1, 3, 5, 10, 20, 50],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': [10, 30, 50, 100],
    'p': [1, 2],
}
KNN_model = KNeighborsClassifier()
```

In [ ]:
```
knn_grid = GridSearchCV(
    estimator=KNN_model,
    param_grid=param_grid5,
    cv=5,
    verbose=3,
    n_jobs=-1,
)
knn_grid.fit(X_train, y_train)
print("Melhores Parametros: ", knn_grid.best_params_)
print("Melhor Exatidão: ", knn_grid.best_score_)
```

In [ ]:
```
results_df_KNN = pd.DataFrame(knn_grid.cv_results_)
top_results5= results_df_KNN.sort_values(by='mean_test_score', ascending=
print(top_results5[['mean_test_score', 'params']].head(10))
heatmap_data5= results_df_KNN.pivot_table(
    index='param_n_neighbors',
    columns='params_metric',
    values='mean_test_score'
)
sns.heatmap(heatmap_data4,annot=True,cmap='viridis', fmt=".4f")
plt.title("Heatmap Neighbors x metric")
plt.xlabel("Neighbors")
```

```
plt.ylabel("Metric")
plt.show()
```

4. Melhores Modelos

Nesta secção encontra se analises dos dois modelos que obteram a melhor classificação no desafio do Kaggle. Ambos utilizam o Random Forrest Model.

RandomForrest 1

Analise do Modelo

Desempenho global:

O modelo apresenta uma exatidão global de **87%**, refletindo um **desempenho consistente**. Esta conclusão é suportada pela análise das métricas:

- **Macro-média**: uma **precisão de 86%**, um **recall de 87%** e um **F1-score de 86%**.
- **Média ponderada**: tanto a precisão como o recall e o F1-score têm um valor de **87%**.

Estas métricas sugerem que o modelo lida razoavelmente bem com o equilíbrio entre as classes, embora as classes com maior número de exemplos tenham maior impacto no resultado final.

Analise por classe:

- **Classe 1**:
  O modelo apresenta um desempenho excelente nesta classe, com poucos erros. No entanto, a análise da matriz de confusão revela que **7 amostras foram incorretamente classificadas como Classe 2** e **1 como Classe 3**. A confusão recorrente com a Classe 2 pode indicar uma **leve sobreposição nas características dessas classes**.

- **Classe 2**:
  Esta é uma das classes com maior número de confusões, especialmente com a **Classe 3**, o que pode indicar que ambas possuem **características similares** ou que o modelo não está a capturar adequadamente as diferenças entre elas.

- **Classe 3**:
  A Classe 3 apresenta o menor **recall (70%)** entre todas as classes, indicando que o modelo está a falhar em identificar corretamente muitos exemplos reais desta classe. A maior confusão ocorre com a **Classe 2 (59 exemplos)**, o que reforça a necessidade de melhorar a separação entre estas classes.

- **Classe 4**:
  O desempenho nesta classe é sólido, embora existam confusões significativas com a **Classe 7**, sugerindo que estas classes têm **características sobrepostas**.

- **Classe 5**:
  A Classe 5 apresenta um desempenho excelente, com **poucas confusões** registadas.

- **Classe 6**:
  Desempenho muito bom, com **confusões pontuais**, principalmente com a **Classe 3 (9 exemplos)**.

- **Classe 7**:
  O modelo apresenta um **bom desempenho geral**, mas ainda assim existem **confusões notáveis com a Classe 4 (24 exemplos)**.

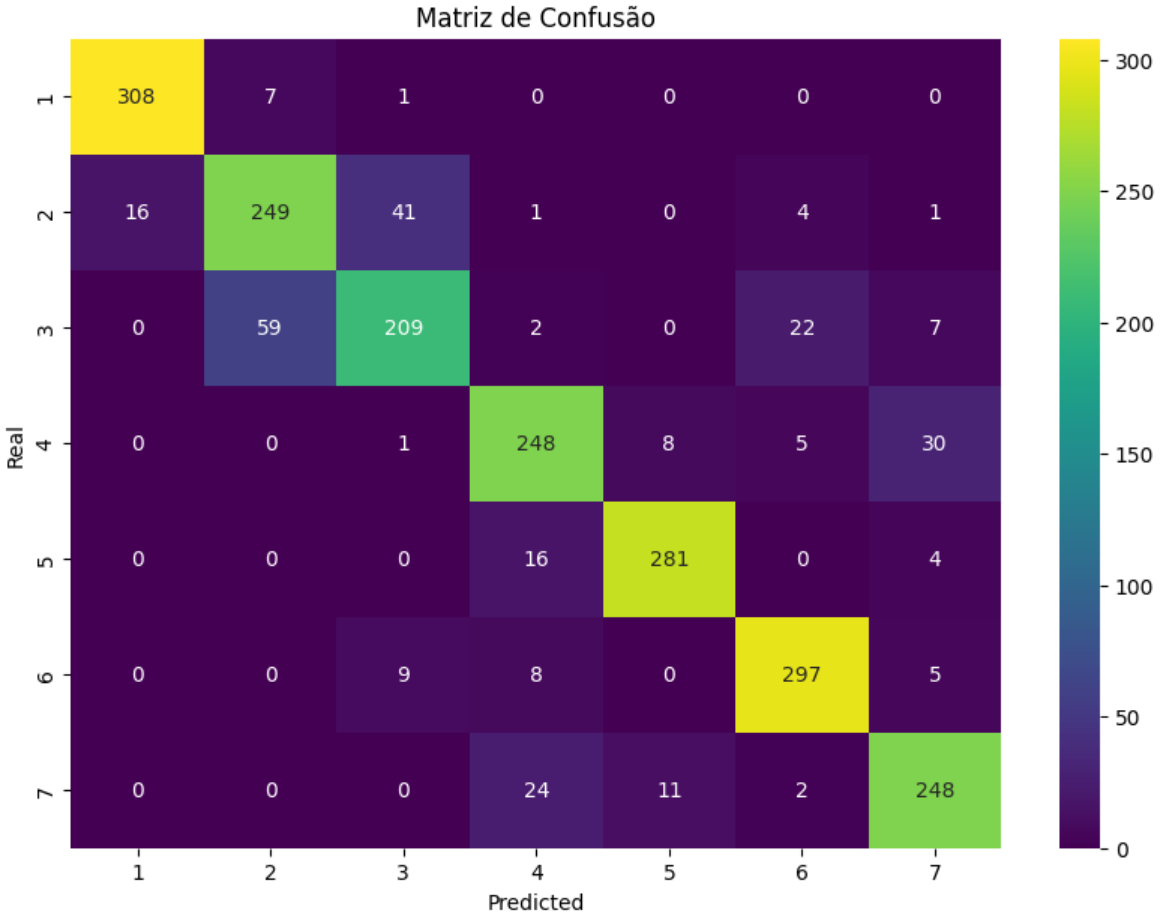
De forma geral, o modelo demonstra um **bom desempenho** na classificação do dataset, mas a **confusão entre classes** é uma das principais limitações, especialmente entre a **Classe 2 e Classe 3**.

In [8]:
```
RF1 = RandomForestClassifier(
    n_estimators=5800,
    max_depth=None,
    max_leaf_nodes=2500,
    random_state=42,
    criterion='entropy',
    bootstrap=False,
    class_weight="balanced_subsample",
    n_jobs=-1

)
RF1.fit(X_train,y_train)
y_pred = RF1.predict(X_test)
```

In [46]:
```
cm1 = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm1, annot=True, fmt='d', cmap='viridis', xticklabels=np.uniq
plt.xlabel('Predicted')
plt.ylabel('Real')
plt.title('Matriz de Confusão')
plt.show()

print(classification_report(y_test,y_pred))
RF1_acc = accuracy_score(y_test,y_pred)
error_rateRF = 1-RF1_acc
print(f"Exatidão: {RF1_acc:.4f}")
print(f"Taxa de Erro: {error_rateRF:.4f}")
```



	precision	recall	f1-score	support
1	0.95	0.97	0.96	316
2	0.79	0.80	0.79	312
3	0.80	0.70	0.75	299
4	0.83	0.85	0.84	292
5	0.94	0.93	0.94	301
6	0.90	0.93	0.92	319
7	0.84	0.87	0.86	285
accuracy			0.87	2124
macro avg	0.86	0.87	0.86	2124
weighted avg	0.87	0.87	0.87	2124

```
Exatidão: 0.8663
Taxa de Erro: 0.1337
```

Random Forrest 2

Analise do Modelo

Desempenho Global

O modelo apresenta uma exatidão global de **85,73%**, refletindo um **bom desempenho geral**. Esta conclusão é suportada pelas métricas:

- **Macro-média**: uma **precisão de 85%**, um **recall de 86%** e um **F1-score de 85%**.
- **Média ponderada**: tanto a precisão como o recall e o F1-score têm um valor de **86%**.

Estes valor indicam que o modelo apresenta um desempenho consistente entre classes, embora ocorra uma representação desequilibrada entre classes, com as classes com maior numero de exemplos tendo um maior impacto.

Analise de Classes

- **Classe 1**:
  A Classe 1 apresenta o melhor desempenho global. No entanto, foram registados alguns erros de classificação, sendo **8 exemplos classificados como Classe 2** e **1 como Classe 3**.

- **Classe 2**:
  Existem confusões significativas com a **Classe 3 (36 exemplos)**, o que pode indicar que as características destas classes são muito semelhantes ou que o modelo não está a capturar corretamente as diferenças entre elas.

- **Classe 3**:
  A Classe 3 apresenta o menor **recall** entre todas as classes, com **notáveis confusões com a Classe 2 (56 exemplos)**. Isto sugere que o modelo não está a identificar corretamente muitos exemplos reais desta classe.

- **Classe 4**:
  O desempenho nesta classe é sólido, embora existam algumas confusões recorrentes com a **Classe 7 (35 exemplos)**.

- **Classe 5**:
  Esta classe apresenta um desempenho excelente, com **poucos erros registados**.

- **Classe 6**:
  Embora o desempenho geral da classe seja bom, ocorreram **confusões pontuais com a Classe 3 (11 exemplos)**.

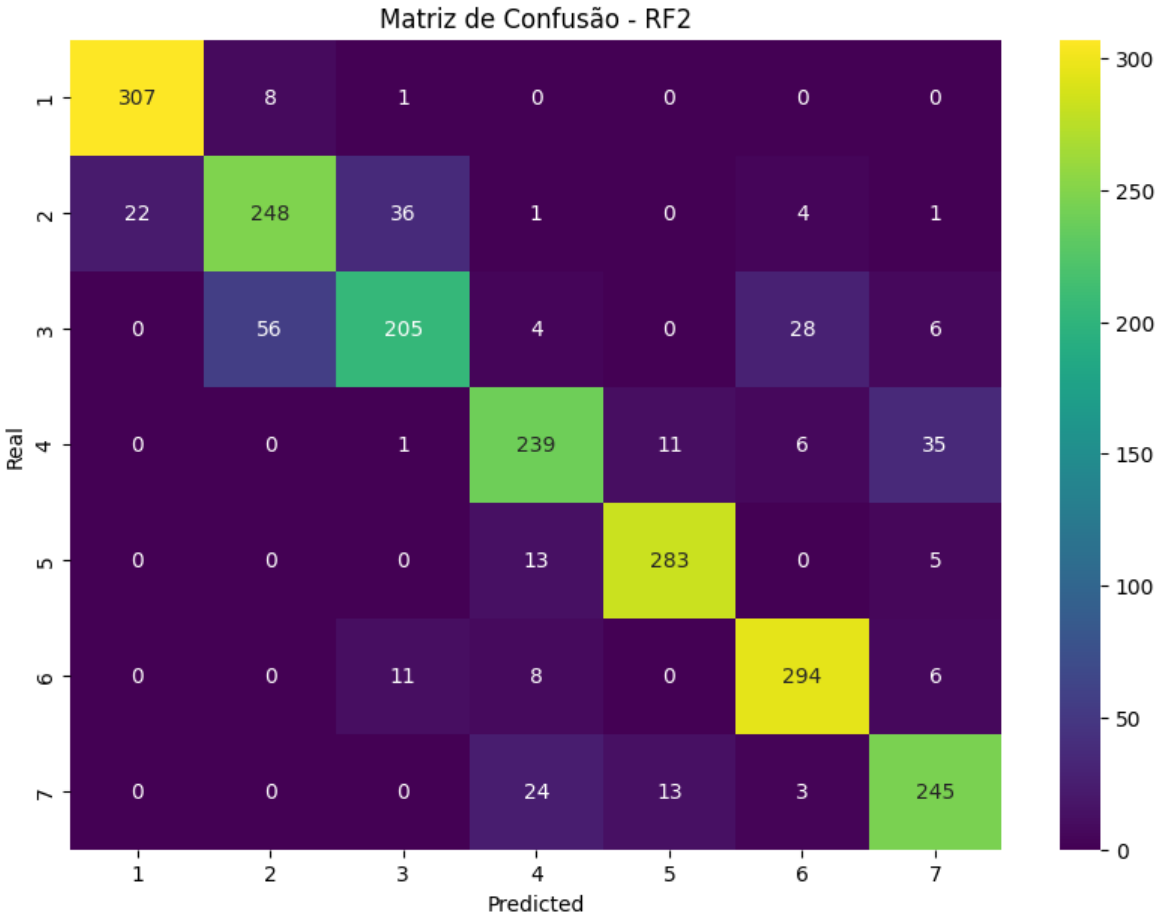
- **Classe 7**:
  O modelo apresenta **confusões frequentes desta classe com a Classe 4 (24 exemplos)**.

Em suma, o modelo apresenta um **bom desempenho** contudo apresenta graves falhas na classificação de classes que diminui o desempenho geral do modelo.

In [22]:
```
RF2 = RandomForestClassifier(
    n_estimators=4500,
    max_depth=None,
    max_leaf_nodes=1000,
    random_state=42,
    criterion='entropy',
    bootstrap=True,
    class_weight="balanced_subsample",
    n_jobs=-1
)
RF2.fit(X_train,y_train)
y_predrf2 = RF2.predict(X_test)
```

In [ ]:
```
cm_RF2 = confusion_matrix(y_test, y_pred2)
plt.figure(figsize=(10, 7))
sns.heatmap(cm_RF2, annot=True, fmt='d', cmap='viridis', xticklabels=np.u
plt.xlabel('Predicted')
plt.ylabel('Real')
plt.title('Matriz de Confusão - RF2')
plt.show()

print(classification_report(y_test, y_predrf2))
RF2_acc = accuracy_score(y_test, y_predrf2)
error_rate_RF2 = 1 - RF2_acc
print(f"Exatidão: {RF2_acc:.4f}")
print(f"Taxa de Erro: {error_rate_RF2:.4f}")
```



```
              precision    recall  f1-score   support

           1       0.93      0.97      0.95       316
           2       0.79      0.79      0.79       312
           3       0.81      0.69      0.74       299
           4       0.83      0.82      0.82       292
           5       0.92      0.94      0.93       301
           6       0.88      0.92      0.90       319
           7       0.82      0.86      0.84       285

    accuracy                           0.86      2124
   macro avg       0.85      0.86      0.85      2124
weighted avg       0.86      0.86      0.86      2124

Exatidão: 0.8573
Taxa de Erro: 0.1427
```

5. Outros Modelos

Nesta secção encontramos os modelos que apesar de não obterem uma classificação tão notavel como os dois anteriores tambem foram utilizados no decorrer da realização do trabalho.

Modelo Gradient Boosting

In [54]:
```
GBC = GradientBoostingClassifier(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=3,
    )
GBC.fit(X_train, y_train)
y_pred2 = GBC.predict(X_test)
print(classification_report(y_test,y_pred2))
gbc_acc = accuracy_score(y_test,y_pred2)
print(gbc_acc)
```

```
              precision    recall  f1-score   support

           1       0.93      0.95      0.94       316
           2       0.73      0.76      0.74       312
           3       0.71      0.63      0.67       299
           4       0.79      0.82      0.81       292
           5       0.93      0.94      0.94       301
           6       0.87      0.89      0.88       319
           7       0.82      0.81      0.82       285

    accuracy                           0.83      2124
   macro avg       0.83      0.83      0.83      2124
weighted avg       0.83      0.83      0.83      2124

0.8295668549905838
```

Logistic Regression

In [55]:
```
LR = LogisticRegression(
    C= 100,
    class_weight= None,
```

```
    max_iter= 100,
    penalty="l1",
    solver= 'saga'
)
LR.fit(X_train,y_train)
y_pred3 = LR.predict(X_test)
print(classification_report(y_test,y_pred3))
lr_acc= accuracy_score(y_test,y_pred3)
print(lr_acc)
```

```
              precision    recall  f1-score   support

           1       0.61      0.72      0.66       316
           2       0.46      0.39      0.42       312
           3       0.45      0.43      0.44       299
           4       0.38      0.34      0.36       292
           5       0.63      0.81      0.71       301
           6       0.46      0.46      0.46       319
           7       0.34      0.27      0.30       285

    accuracy                           0.49      2124
   macro avg       0.48      0.49      0.48      2124
weighted avg       0.48      0.49      0.48      2124

0.4929378531073446
```

```
/home/miguel/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/linear
_model/_sag.py:349: ConvergenceWarning: The max_iter was reached which mea
ns the coef_ did not converge
  warnings.warn(
```

Modelo MLP

In [56]:
```
MLP = MLPClassifier(
   hidden_layer_sizes=(300,),
   activation='relu',
   solver='adam',
   learning_rate_init=0.1,
   max_iter=2000
)
MLP.fit(X_train,y_train)
y_pred4 = MLP.predict(X_test)
print(classification_report(y_test,y_pred4))
mlp_acc= accuracy_score(y_test,y_pred4)
print(mlp_acc)
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       316
           2       0.15      1.00      0.26       312
           3       0.00      0.00      0.00       299
           4       0.00      0.00      0.00       292
           5       0.00      0.00      0.00       301
           6       0.00      0.00      0.00       319
           7       0.00      0.00      0.00       285

    accuracy                           0.15      2124
   macro avg       0.02      0.14      0.04      2124
weighted avg       0.02      0.15      0.04      2124

0.14689265536723164
```

```
/home/miguel/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/metric
s/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/miguel/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/metric
s/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/miguel/uni/4ano/aa/.venv/lib/python3.12/site-packages/sklearn/metric
s/_classification.py:1531: UndefinedMetricWarning: Precision is ill-define
d and being set to 0.0 in labels with no predicted samples. Use `zero_divi
sion` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [57]:

```
KNN = KNeighborsClassifier(
    n_neighbors=10,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    metric_params=None
)

KNN.fit(X_train, y_train)
y_pred5 = KNN.predict(X_test)
print(classification_report(y_test,y_pred5))
knn_acc = accuracy_score(y_test,y_pred5)
print(knn_acc)
```

```
              precision    recall  f1-score   support

           1       0.76      0.92      0.83       316
           2       0.62      0.55      0.58       312
           3       0.66      0.42      0.52       299
           4       0.68      0.66      0.67       292
           5       0.78      0.90      0.84       301
           6       0.75      0.86      0.80       319
           7       0.69      0.67      0.68       285

    accuracy                           0.71      2124
   macro avg       0.71      0.71      0.70      2124
weighted avg       0.71      0.71      0.70      2124

0.7146892655367232
```

6. Conclusão

Após uma análise detalhada dos dados e a realização de um pré-processamento cuidadoso, foi possível explorar as variáveis e compreender as suas distribuições e correlações, o que ajudou a identificar as características mais relevantes para a construção do modelo. Durante o processo de modelagem, foram avaliados diversos algoritmos de classificação e, após a afinação de parâmetros, o modelo Random Forest revelou-se o mais adequado para este conjunto de dados.

A escolha do Random Forest foi validada pelos bons resultados observados nas métricas de avaliação, como precisão, recall e F1-score, que apresentaram um desempenho superior em comparação com outros modelos testados. Este algoritmo mostrou-se robusto ao lidar com a complexidade e diversidade do conjunto de dados, conseguindo captar as relações entre as variáveis e, ao mesmo tempo, fornecer uma boa capacidade de generalização.

In [58]:
```
models = pd.DataFrame({
    'Modelo':['RandomForestClassifier', 'KNN', 'LogisticRegression', 'Gra
    'Exatidão':[RF1_acc,lr_acc,knn_acc,gbc_acc,mlp_acc]
})
models.sort_values(by='Exatidão',ascending=False)
```

Out[58]:

	Modelo	Exatidão
0	RandomForestClassifier	0.866290
3	GradientBoostingClassifier	0.829567
2	LogisticRegression	0.714689
1	KNN	0.492938
4	MLPClassifier(hidden_layer_sizes=(300,), learn...	0.146893

Criação do ficheiro de submissão

Random Forrest

In [238…
```
ids_for_test = test['id']
submissao = pd.DataFrame({
    "id": ids_for_test,
    "floresta": y_pred
})
submissao.to_csv('submissao12.csv', index=False)
```

GBC

In [44]:
```
ids_for_test = test['id']
submissaoGBC = pd.DataFrame({
    "id": ids_for_test,
    "floresta": y_pred2
})
submissaoGBC.to_csv('submissaoGBC3.csv',index=False)
```

MLP

In [ ]:
```
ids_for_test = test['id']
submissaoMLP = pd.DataFrame({
    "id": ids_for_test,
    "floresta": y_pred3
})
submissaoMLP.to_csv('submissaoMLP1.csv', index=False)
```