Estimating aerosols

January 10, 2024

1 Aprendizagem Automática - aerosols

- Professora:
  - Teresa Gonçalves
- Alunos:
  - Alexandre Costa nº 48039
  - Manuel Pereira nº 45855

1.1 Comandos necessários para o Kaggle

[ ]:
```
! pip install -q kaggle
```

[ ]:
```
import os
!mkdir ~/.kaggle
```

[ ]:
```
!cp dados/kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

1.2 Entendimento dos dados

[255]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint

train_data = pd.read_csv('dados/train.csv')
```

Analisar as primeiras linhas do conjunto de treino Com esta análise conseguimos perceber que todos os atributos são numéricos menos o atributo "scene".

[256]:
```
train_data.head()
```

[256]:
```
   id  elevation  ozone    NO2  azimuth  zenith      B1      B2      B3  \
0   1        209    301  0.278    142.6    33.9  0.1729  0.1423  0.1268
1   2         82    357  0.376    156.7    55.7  0.1468  0.1173  0.1008
2   3         37    305  0.258    134.8    37.6  0.1228  0.0920  0.0796
3   4       1574    309  0.145    163.2    66.4  0.1593  0.1556  0.1522
4   5         52    294  0.359    155.8    61.3  0.4497  0.4480  0.4264

       B4  ...     B8A      B9     B10     B11     B12  water_vapor  scene  \
0  0.1134  ...  0.2066  0.0488  0.0015  0.1689  0.1207         1977      E
1  0.1022  ...  0.1331  0.1200  0.0049  0.1583  0.1321          168      E
2  0.0467  ...  0.3410  0.0317  0.0011  0.1306  0.0446         4517      D
3  0.1601  ...  0.2370  0.1250  0.0027  0.2336  0.1582          361      E
4  0.4616  ...  0.5040  0.3016  0.0171  0.4580  0.3912          498      I

   incidence_azimuth  incidence_zenith  AOT_550
0              101.4               4.8    0.337
1              286.0               8.6    0.076
2              103.3               4.6    0.199
3              137.3               3.2    0.012
4              270.3               4.9    0.260

[5 rows x 24 columns]
```

Com o comando "train_data.head()" verificámos que apenas o atributo "scene" não é numérico. E com o comando "train_data.info()" confirmámos que o tipo do atributo "scene" é object. Foi verificado também pelo comando que não existem valores não nulos.

[257]:
```
train_data.info()
```

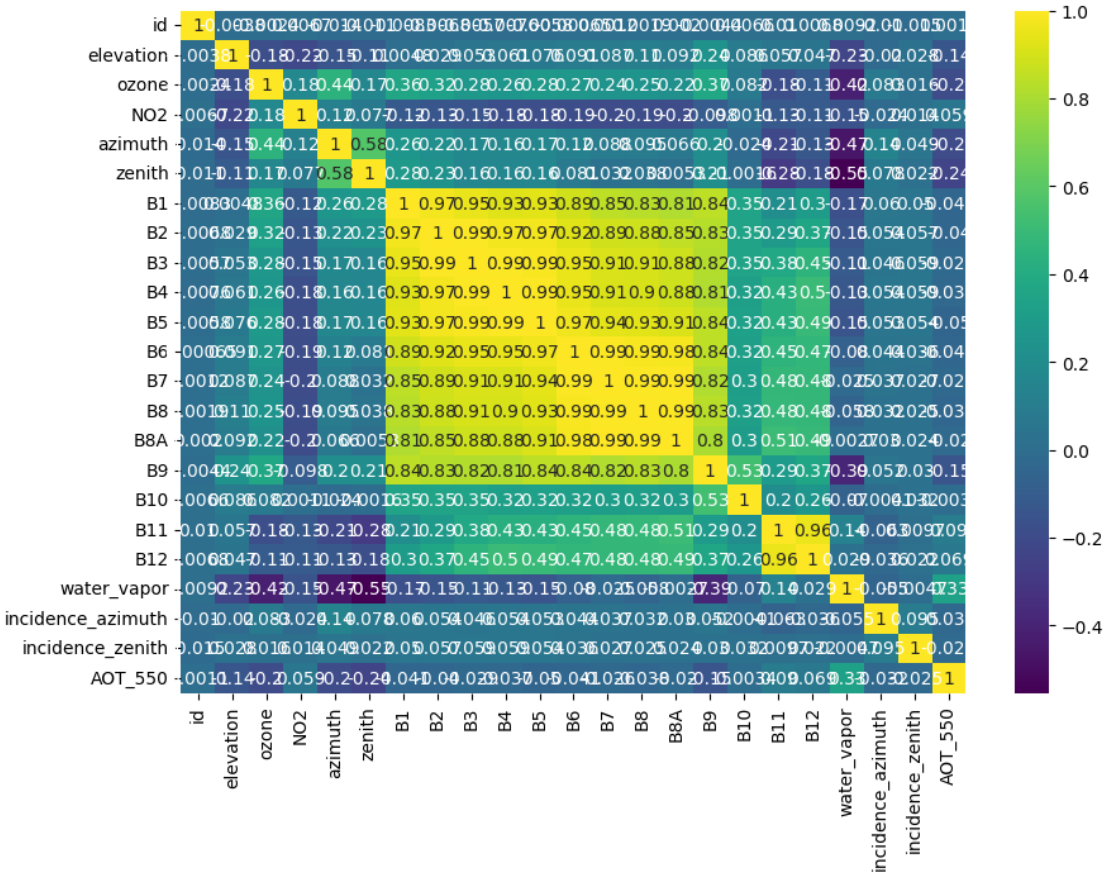
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11177 entries, 0 to 11176
Data columns (total 24 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 11177 non-null  int64
 1   elevation          11177 non-null  int64
 2   ozone              11177 non-null  int64
 3   NO2                11177 non-null  float64
 4   azimuth            11177 non-null  float64
 5   zenith             11177 non-null  float64
 6   B1                 11177 non-null  float64
 7   B2                 11177 non-null  float64
 8   B3                 11177 non-null  float64
 9   B4                 11177 non-null  float64
```

```
 10  B5                 11177 non-null  float64
 11  B6                 11177 non-null  float64
 12  B7                 11177 non-null  float64
 13  B8                 11177 non-null  float64
 14  B8A                11177 non-null  float64
 15  B9                 11177 non-null  float64
 16  B10                11177 non-null  float64
 17  B11                11177 non-null  float64
 18  B12                11177 non-null  float64
 19  water_vapor        11177 non-null  int64
 20  scene              11177 non-null  object
 21  incidence_azimuth  11177 non-null  float64
 22  incidence_zenith   11177 non-null  float64
 23  AOT_550            11177 non-null  float64
dtypes: float64(19), int64(4), object(1)
memory usage: 2.0+ MB
```

Foi usado o "sns.heatmap()" para se visualizar a correlação entre os atributos numéricos e verificámos que onde existe as maiores relações é nos atributos "B1", "B2", "B3", "B4", "B5", "B6", "B7", "B8", "B8A", "B9", "B10", "B11", "B12".

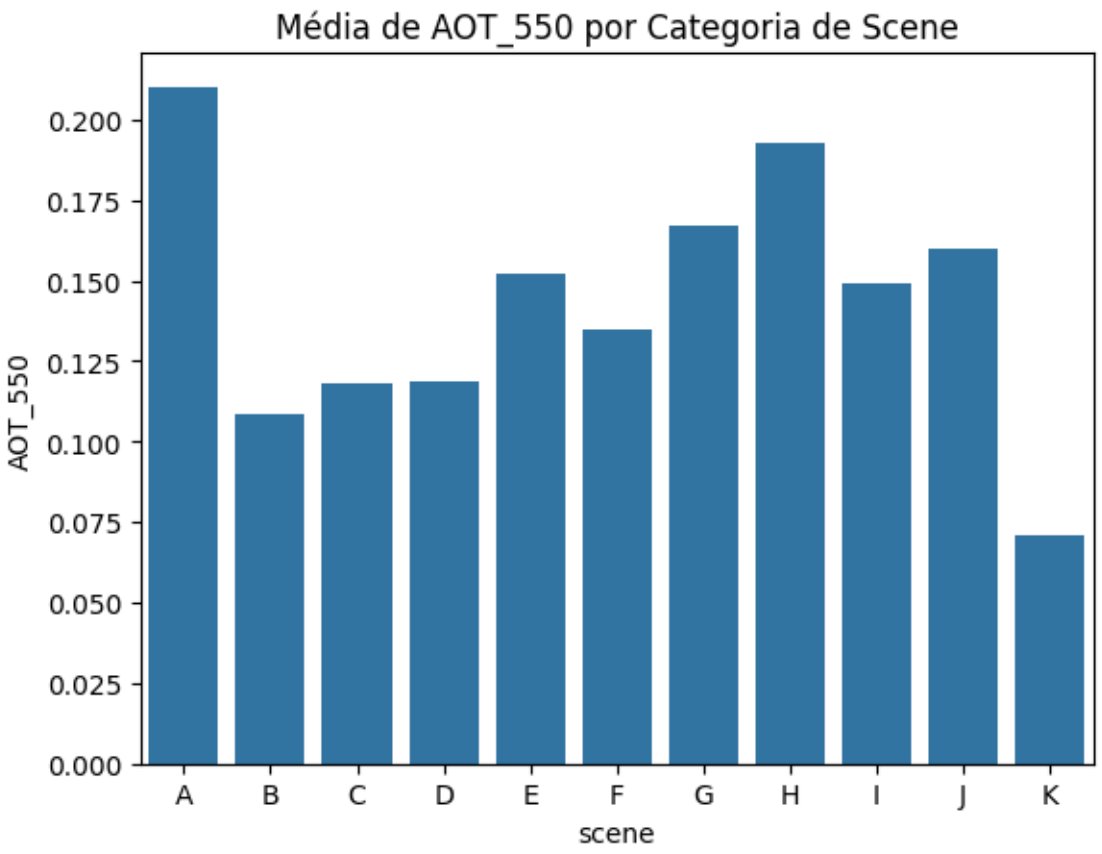
[258]:
```
numeric_columns = train_data.select_dtypes(exclude=['object']).columns

plt.figure(figsize=(10, 7))
sns.heatmap(train_data[numeric_columns].corr(), annot=True, cmap='viridis')
plt.show()
```



Quanto ao atributo "scene", foi usado um Barplot para mostrar a distribuição do atributo alvo "AOT_550" para diferentes categorias do atributo "scene". Podemos verificar que o atributo influencia o número de aerossóis e que a categoria A e H são as que influenciam mais o aumento de aerossóis.

[259]:
```
mean_aot_by_scene = train_data.groupby('scene')['AOT_550'].mean().reset_index()
sns.barplot(x='scene', y='AOT_550', data=mean_aot_by_scene)
plt.title('Média de AOT_550 por Categoria de Scene')
plt.show()
```



[260]:
```
train_data
```

[260]:

	id	elevation	ozone	NO2	azimuth	zenith	B1	B2	\
0	1	209	301	0.278	142.6	33.9	0.1729	0.1423	
1	2	82	357	0.376	156.7	55.7	0.1468	0.1173	
2	3	37	305	0.258	134.8	37.6	0.1228	0.0920	
3	4	1574	309	0.145	163.2	66.4	0.1593	0.1556	
4	5	52	294	0.359	155.8	61.3	0.4497	0.4480	
...	...	...	...	...	...	...	...	...	
11172	11173	225	286	0.124	192.1	71.9	0.2347	0.2109	
11173	11174	106	349	1.193	162.7	64.8	0.1753	0.1295	
11174	11175	125	364	0.292	158.9	32.3	0.1543	0.1335	
11175	11176	51	271	0.146	55.6	33.8	0.1969	0.1766	
11176	11177	1280	330	0.161	153.0	54.9	0.1471	0.1262	

	B3	B4	...	B8A	B9	B10	B11	B12	\
0	0.1268	0.1134	...	0.2066	0.0488	0.0015	0.1689	0.1207	
1	0.1008	0.1022	...	0.1331	0.1200	0.0049	0.1583	0.1321	
2	0.0796	0.0467	...	0.3410	0.0317	0.0011	0.1306	0.0446	
3	0.1522	0.1601	...	0.2370	0.1250	0.0027	0.2336	0.1582	

```
4      0.4264  0.4616  ...  0.5040  0.3016  0.0171  0.4580  0.3912
...       ...     ...  ...     ...     ...     ...     ...     ...
11172  0.1922  0.2159  ...  0.2619  0.0967  0.0083  0.3035  0.2689
11173  0.0897  0.0727  ...  0.0777  0.0494  0.0018  0.0624  0.0504
11174  0.1206  0.1071  ...  0.2169  0.0779  0.0010  0.1799  0.1352
11175  0.1618  0.1638  ...  0.2458  0.0236  0.0009  0.2618  0.1910
11176  0.1272  0.1589  ...  0.2344  0.1233  0.0024  0.3097  0.2458

       water_vapor  scene  incidence_azimuth  incidence_zenith  AOT_550
0             1977      E              101.4               4.8    0.337
1              168      E              286.0               8.6    0.076
2             4517      D              103.3               4.6    0.199
3              361      E              137.3               3.2    0.012
4              498      I              270.3               4.9    0.260
...            ...    ...                ...               ...      ...
11172          649      E              157.1               3.1    0.088
11173          382      C              283.7               6.0    0.217
11174         1122      G              282.6               5.6    0.155
11175         4435      E              103.7               8.8    0.322
11176          436      E              106.5               6.6    0.121

[11177 rows x 24 columns]
```

[261]:
```python
def preprocessing_td(df, target=None, scaler=None):
    df_processed = df.copy()
    if target is not None:
        target_data = df_processed.pop(target)

    if 'scene' in df_processed.columns:
        label_encoder = LabelEncoder()
        df_processed['scene'] = label_encoder.
 ↪fit_transform(df_processed['scene'])

    num_cols = df_processed.select_dtypes(include=['int64', 'float64']).columns
    if scaler is None:
        scaler = MinMaxScaler()
        df_processed[num_cols] = scaler.fit_transform(df_processed[num_cols])
    else:
        df_processed[num_cols] = scaler.transform(df_processed[num_cols])

    if target is not None:
        df_processed[target] = target_data

    return df_processed, scaler
```

[262]:
```python
train_data, train_scaler = preprocessing_td(train_data, target=train_data.
 ↪columns[-1])
```

[263]:
```
train_data
```

[263]:

	id	elevation	ozone	NO2	azimuth	zenith	B1	\
0	0.000000	0.045774	0.300469	0.172043	0.542153	0.358667	0.149684	
1	0.000089	0.021652	0.563380	0.247312	0.603113	0.649333	0.127089	
2	0.000179	0.013105	0.319249	0.156682	0.508431	0.408000	0.106311	
3	0.000268	0.305033	0.338028	0.069892	0.631215	0.792000	0.137910	
4	0.000358	0.015954	0.267606	0.234255	0.599222	0.724000	0.389317	
...	...	...	...	...	...	...	...	
11172	0.999642	0.048813	0.230047	0.053763	0.756161	0.865333	0.203186	
11173	0.999732	0.026211	0.525822	0.874808	0.629053	0.770667	0.151762	
11174	0.999821	0.029820	0.596244	0.182796	0.612624	0.337333	0.133582	
11175	0.999911	0.015764	0.159624	0.070661	0.166018	0.357333	0.170461	
11176	1.000000	0.249193	0.436620	0.082181	0.587116	0.638667	0.127348	

	B2	B3	B4	...	B8A	B9	B10	\
0	0.126579	0.114533	0.091533	...	0.166412	0.044746	0.002384	
1	0.104341	0.091049	0.082493	...	0.107209	0.110031	0.007789	
2	0.081836	0.071900	0.037695	...	0.274668	0.029067	0.001749	
3	0.138410	0.137476	0.129228	...	0.190898	0.114616	0.004292	
4	0.398506	0.385150	0.372589	...	0.405961	0.276545	0.027182	
...	...	...	...	...	...	...	...	
11172	0.187600	0.173607	0.174267	...	0.210954	0.088667	0.013193	
11173	0.115193	0.081022	0.058681	...	0.062586	0.045296	0.002861	
11174	0.118751	0.108933	0.086448	...	0.174708	0.071429	0.001590	
11175	0.157089	0.146148	0.132214	...	0.197986	0.021639	0.001431	
11176	0.112258	0.114895	0.128259	...	0.188804	0.113057	0.003815	

	B11	B12	water_vapor	scene	incidence_azimuth	\
0	0.171978	0.151748	0.324152	0.4	0.018957	
1	0.161185	0.166080	0.027545	0.4	0.748025	
2	0.132980	0.056072	0.740613	0.3	0.026461	
3	0.237858	0.198894	0.059190	0.4	0.160742	
4	0.466348	0.491828	0.081653	0.8	0.686019	
...	...	...	...	...	...	
11172	0.309032	0.338069	0.106411	0.4	0.238942	
11173	0.063537	0.063364	0.062633	0.2	0.738942	
11174	0.183179	0.169977	0.183965	0.6	0.734597	
11175	0.266572	0.240131	0.727168	0.4	0.028041	
11176	0.315345	0.309027	0.071487	0.4	0.039100	

	incidence_zenith	AOT_550
0	0.255556	0.337
1	0.677778	0.076
2	0.233333	0.199
3	0.077778	0.012
4	0.266667	0.260

```
...                 ...      ...
11172          0.066667    0.088
11173          0.388889    0.217
11174          0.344444    0.155
11175          0.700000    0.322
11176          0.455556    0.121

[11177 rows x 24 columns]
```

[264]:
```
# Select features by index slicing
selected_features = train_data.iloc[:, 1:-1]

# Select the target variable
target = train_data['AOT_550']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(selected_features, target,␣
 ↪test_size=0.2, random_state=42)
```

1.3 Escolha do modelo

1.3.1 Regressão Linear

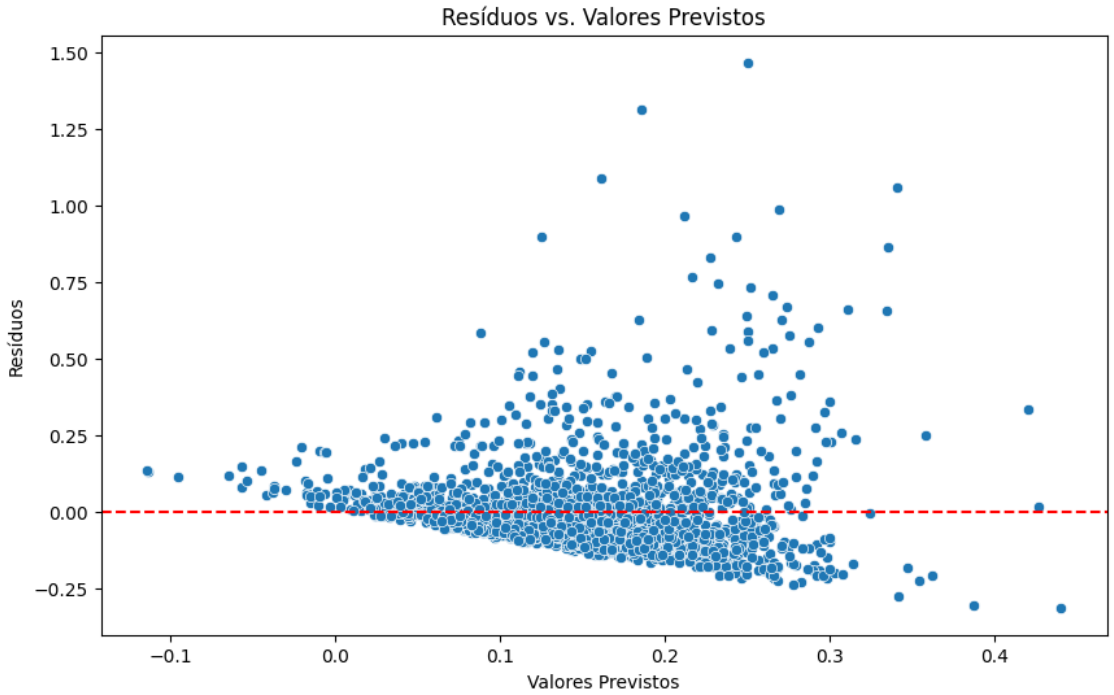
Comecemos por testar o desempenho do modelo Regressão Linear, quando os dados seguem uma tendência linear torna-se uma modelo bastante eficaz e simples.

[235]:
```
# Treinar o modelo
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Prever os valores para o conjunto de teste
y_pred = linear_model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("RMSE: ", rmse)
```

```
RMSE:  0.1406306680917233
```

[237]:
```
# Calcular e visualizar os resíduos (diferenças entre os valores observados e␣
 ↪previstos)
residuos = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuos)
plt.title('Resíduos vs. Valores Previstos')
plt.xlabel('Valores Previstos')
plt.ylabel('Resíduos')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Submissão do modelo

[ ]:
```
# load test data
test_data = pd.read_csv("dados/test.csv")

test_data, _ = preprocessing_td(test_data, scaler=train_scaler)

test_data = test_data.iloc[:, 1:]  # Select columns from index 1 to 2 (exclusive)
predicted_test = linear_model.predict(test_data)

submission = pd.read_csv("dados/sample_submission.csv")
submission['AOT_550'] = predicted_test
submission.to_csv("dados/submission.csv", index=False)

!kaggle competitions submit -c aerosols -f dados/submission.csv -m "Submissão do␣
↪modelo Regressão Linear"
```

Score no Kaggle Com este modelo, o score no Kaggle foi de RMSE 0.1611

1.3.2 Random Forest Regressor

Modelo eficaz para problemas de regressão, baseado em árvores de decisão.

[265]:
```
# Treinar o modelo com parâmetros padrão
random_forest_model = RandomForestRegressor(random_state=42)
```

```
random_forest_model.fit(X_train, y_train)

# Avaliar o modelo
y_pred_rf = random_forest_model.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print("RMSE do Random Forest:", rmse_rf)
```

```
RMSE do Random Forest: 0.11176403139009923
```

[ ]:
```
# Definir o espaço de hiperparâmetros a ser explorado na otimização
param_dist = {
    'n_estimators': sp_randint(100, 500),  # Número de árvores
    'max_depth': [None] + list(range(10, 50)),  # Profundidade máxima de cada␣
 ↪árvore
    'min_samples_split': sp_randint(2, 11),  # Número mínimo de amostras␣
 ↪necessárias para dividir um nó
    'min_samples_leaf': sp_randint(1, 5)  # Número mínimo de amostras␣
 ↪necessárias em um nó folha
}

# Configurar o RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist,␣
 ↪n_iter=100, cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error',␣
 ↪random_state=42)

# Executar o random search
random_search.fit(X_train, y_train)

# Mostrar os melhores parâmetros
print("Melhores parâmetros:", random_search.best_params_)
```

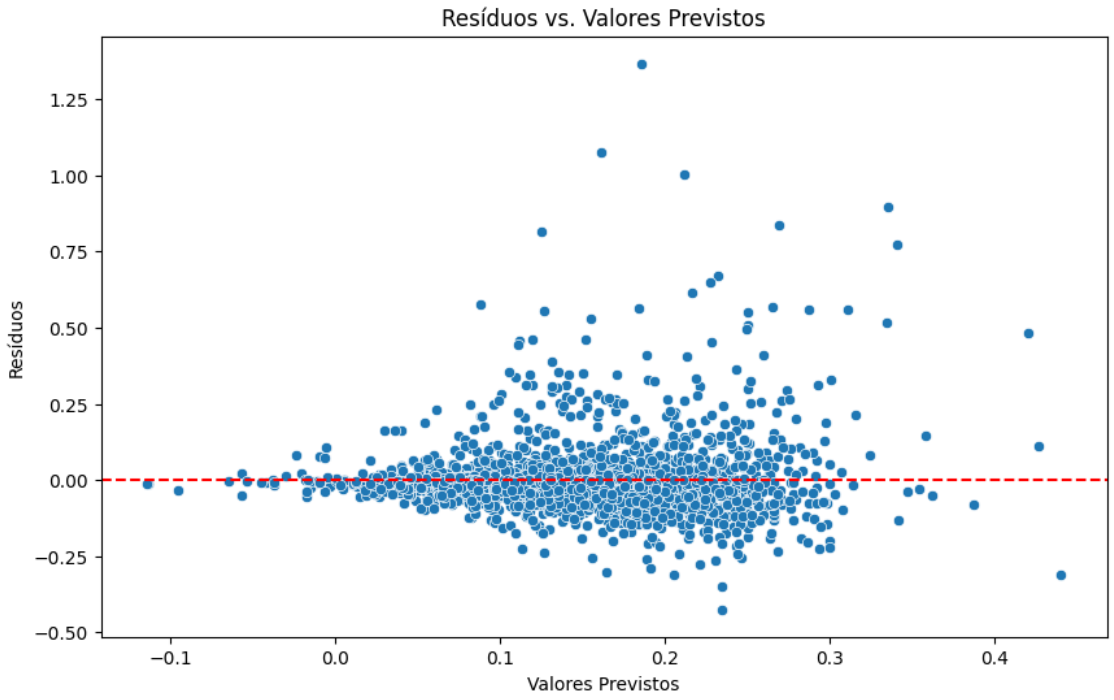
Após execução do Randomized Search a melhor escolha dos parametros é a seguinte: Melhores parâmetros: {'max_depth': 33, 'min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 484}

[268]:
```
# Treinar o modelo com os melhores parametros
random_forest_model_com_p_ajustes = RandomForestRegressor(max_depth=33,␣
 ↪min_samples_leaf=3, min_samples_split= 2, n_estimators= 484, random_state=42)
random_forest_model_com_p_ajustes.fit(X_train, y_train)

# Avaliar o modelo
y_pred_rf_p_ajustes = random_forest_model_com_p_ajustes.predict(X_test)
rmse_rf_p_ajustes = np.sqrt(mean_squared_error(y_test, y_pred_rf_p_ajustes))
print("RMSE do Random Forest com os parametros ajustados:", rmse_rf_p_ajustes)
```

```
RMSE do Random Forest com os parametros ajustados: 0.11232642090826352
```

[269]:
```
# Calcular e visualizar os resíduos (diferenças entre os valores observados e previstos)
residuos = y_test - y_pred_rf_p_ajustes
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuos)
plt.title('Resíduos vs. Valores Previstos')
plt.xlabel('Valores Previstos')
plt.ylabel('Resíduos')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Submissão do modelo

[ ]:
```
# load test data
test_data = pd.read_csv("dados/test.csv")

test_data, _ = preprocessing_td(test_data, scaler=train_scaler)

test_data = test_data.iloc[:, 1:]  # Select columns from index 1 to 2 (exclusive)
predicted_test = best_rf_random.predict(test_data)

submission = pd.read_csv("dados/sample_submission.csv")
submission['AOT_550'] = predicted_test
submission.to_csv("dados/submission.csv", index=False)
```

```
!kaggle competitions submit -c aerosols -f dados/submission.csv -m "Modelo com␣
↪Random Forest Regressor (RandomizedSearchCV)"
```

Score no Kaggle Com este modelo, o score no Kaggle foi de RMSE 0.1365

1.3.3 Gradient Boosting Regressor

Modelo de aprendizagem automática flexivel que constrói um conjunto de árvores de decisão de forma sequencial, onde cada árvore tenta corrigir os erros das árvores anteriores.

[271]:
```
# Treinar o modelo
gbr_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,␣
↪max_depth=3, random_state=42)
gbr_model.fit(X_train, y_train)

# Avaliar o modelo
y_pred_gbr = gbr_model.predict(X_test)
rmse_gbr = np.sqrt(mean_squared_error(y_test, y_pred_gbr))
print(f"RMSE do Gradient Boosting Regressor: {rmse_gbr}")
```

```
RMSE do Gradient Boosting Regressor: 0.1251964137878606
```

[ ]:
```
param_grid = {
    'n_estimators': [100, 200, 300],  # Número de árvores
    'learning_rate': [0.01, 0.1, 0.2],  # Taxa de aprendizagem
    'max_depth': [3, 4, 5],  # Profundidade máxima de cada árvore
    'min_samples_split': [2, 4],  # Número mínimo de amostras para dividir um nó
    'min_samples_leaf': [1, 2]  # Número mínimo de amostras em uma folha
}

gbr = GradientBoostingRegressor(random_state=42)

# Configuração do GridSearchCV para encontrar os melhores parametros
# n_jobs indica o numero de nucleos que serão utilizados, o cv refere-se à␣
↪validação cruzada
grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=3,␣
↪n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Mostra os melhores parametros encontrados
print("Melhores parâmetros:", grid_search.best_params_)
best_gbr = grid_search.best_estimator_
```

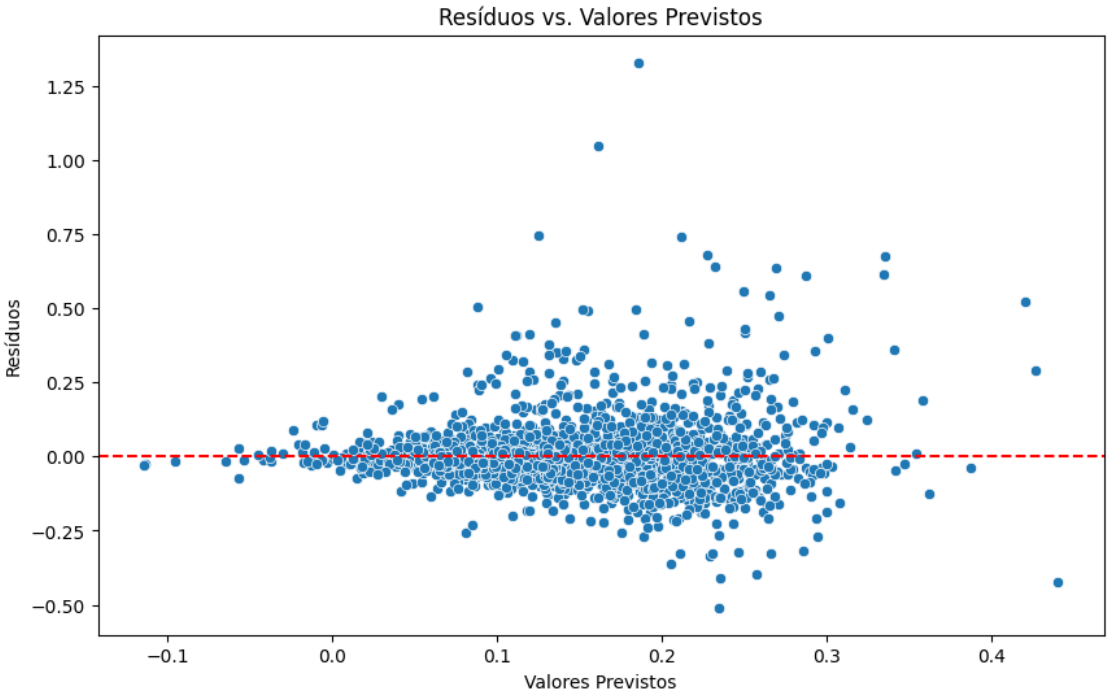
Após execução do Grid Search a melhor escolha dos parametros é a seguinte: Melhores parâmetros: 'learning_rate': 0.1, 'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300

[272]:
```
# Treinar o modelo com os parâmetros ajustados
gbr_model_p_ajustados = GradientBoostingRegressor(n_estimators=300,␣
↪learning_rate=0.1, max_depth=5, min_samples_leaf=1, min_samples_split=2,␣
↪random_state=42)
gbr_model_p_ajustados.fit(X_train, y_train)

# Avaliar o modelo
y_pred_gbr = gbr_model_p_ajustados.predict(X_test)
rmse_gbr = np.sqrt(mean_squared_error(y_test, y_pred_gbr))
print(f"RMSE do Gradient Boosting Regressor: {rmse_gbr}")
```

```
RMSE do Gradient Boosting Regressor: 0.10937981219034551
```

[273]:
```
# Calcular e visualizar os resíduos (diferenças entre os valores observados e␣
↪previstos)
residuos = y_test - y_pred_gbr
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuos)
plt.title('Resíduos vs. Valores Previstos')
plt.xlabel('Valores Previstos')
plt.ylabel('Resíduos')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Submissão do modelo

[ ]:
```
# load test data
test_data = pd.read_csv("dados/test.csv")

test_data, _ = preprocessing_td(test_data, scaler=train_scaler)

test_data = test_data.iloc[:, 1:]  # Select columns from index 1 to 2 (exclusive)
predicted_test = gbr_model_p_ajustados.predict(test_data)

submission = pd.read_csv("dados/sample_submission.csv")
submission['AOT_550'] = predicted_test
submission.to_csv("dados/submission.csv", index=False)

!kaggle competitions submit -c aerosols -f dados/submission.csv -m "Submissão␣
↪modelo GBRegressor"
```

Score no Kaggle Com este modelo, o score no Kaggle foi de RMSE 0.128

1.3.4 Rede Neuronal

[291]:
```
# Iniciar o modelo
nn_model = MLPRegressor(hidden_layer_sizes=(100,), activation='relu',␣
↪solver='adam', max_iter=200, random_state=42)

# Treinar o modelo
nn_model.fit(X_train, y_train)

# Avaliar o modelo
y_pred_nn = nn_model.predict(X_test)
rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred_nn))
print(f"RMSE do modelo de rede neural: {rmse_nn}")
```

```
RMSE do modelo de rede neural: 0.13321457464142164
```

[ ]:
```
# Intervalores de hiperparâmetros
param_grid = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'max_iter': [100, 200],
    'learning_rate_init': [0.001, 0.01]
}

# Criar o modelo
mlp = MLPRegressor(random_state=42)

# Configurar o GridSearchCV
# n_jobs indica o numero de nucleos que serão utilizados e o cv refere-se à␣
↪validação cruzada
```

```
grid_search = GridSearchCV(estimator=mlp, param_grid=param_grid, n_jobs=-1,␣
↪cv=3, scoring='neg_mean_squared_error')

# Executar o grid search
grid_search.fit(X_train, y_train)

# Melhores parâmetros
print("Melhores parâmetros:", grid_search.best_params_)
```

Após execução do Grid Search a melhor escolha dos parametros é a seguinte: Melhores parâmetros: 'activation': 'relu', 'hidden_layer_sizes': (50, 100, 50), 'learning_rate_init': 0.001, 'max_iter': 200, 'solver': 'adam'

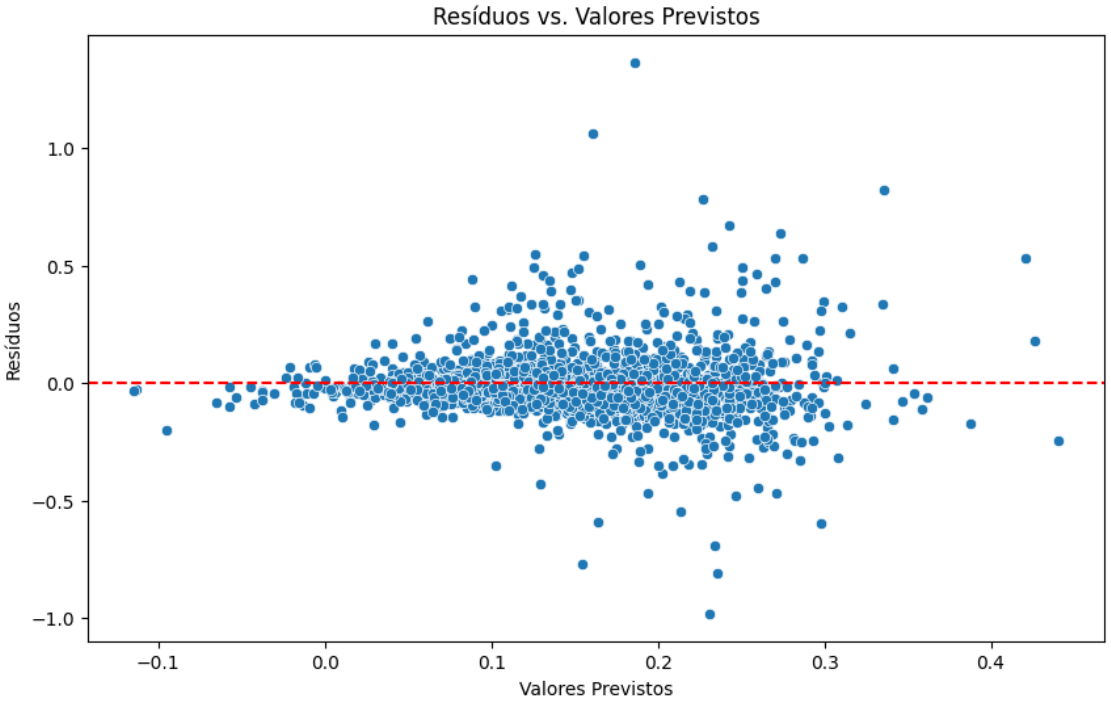
[292]:
```
# Inicializar o modelo com os melhores parametros
nn_model_p_ajustados = MLPRegressor(hidden_layer_sizes=(50,100,50),␣
↪activation='relu', learning_rate_init=0.001, solver='adam', max_iter=200,␣
↪random_state=42)

# Treinar o modelo
nn_model_p_ajustados.fit(X_train, y_train)

# Avaliar o modelo
y_pred_nn = nn_model_p_ajustados.predict(X_test)
rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred_nn))
print(f"RMSE do modelo de rede neural: {rmse_nn}")
```

```
RMSE do modelo de rede neural: 0.12326490409962516
```

[293]:
```
residuos = y_test - y_pred_nn
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=residuos)
plt.title('Resíduos vs. Valores Previstos')
plt.xlabel('Valores Previstos')
plt.ylabel('Resíduos')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Submissão do modelo

[ ]:
```
# load test data
test_data = pd.read_csv("dados/test.csv")

test_data, _ = preprocessing_td(test_data, scaler=train_scaler)

test_data = test_data.iloc[:, 1:]  # Select columns from index 1 to 2 (exclusive)
predicted_test = nn_model_p_ajustados.predict(test_data)

submission = pd.read_csv("dados/sample_submission.csv")
submission['AOT_550'] = predicted_test
submission.to_csv("dados/submission.csv", index=False)

!kaggle competitions submit -c aerosols -f dados/submission.csv -m "Submissão␣
↪com modelo mlpregressor"
```

Score no Kaggle Com este modelo, o score no Kaggle foi de RMSE 0.1396

1.3.5 Escolha do melhor modelo

O modelo que nos mostrou melhores resultados foi o **Gradient Boosting Regressor**. Inicialmente, o modelo foi treinado com parâmetros padrão e resultou num RMSE de 0.1251964137878606. Achámos que o modelo poderia ter potencial neste problema e decidimos melhorar o desempenho do modelo. Foi realizado um processo de otimização de parâmetros utilizando Grid Search. Os

parâmetros a ajustar foram o número de árvores, a taxa de aprendizagem, a profundidade máxima de cada árvore, o número mínimo de amostras para dividir um nó, e o número mínimo de amostras em uma folha.

A melhor configuração encontrada foi: - n_estimators: 300 - learning_rate: 0.1 - max_depth: 5 - min_samples_split: 2 - min_samples_leaf: 1

Após a otimização, o modelo ajustado foi treinado e avaliado, obtendo um RMSE de 0.10937981219034551, indicando uma melhoria significativa em relação à configuração padrão.

Com base nos resultados obtidos, o modelo Gradient Boosting Regressor otimizado provou ser a escolha mais eficaz para o problema em questão, superando outras abordagens testadas e demonstrando um equilíbrio ideal entre complexidade e desempenho. No Kaggle obteve um RMSE de 0.128.

[290]:
```
# RMSE para diferentes configurações
rmse_valores = [0.1251964137878606, 0.10937981219034551, 0.128]
configuracoes = ['Padrão', 'Ajustado', 'Kaggle']

plt.figure(figsize=(7, 3))
plt.bar(configuracoes, rmse_valores, color=['blue', 'green', 'red'])
plt.xlabel('Configuração do Modelo')
plt.ylabel('RMSE')
plt.title('Comparação de RMSE: Modelo Padrão vs Ajustado vs Kaggle')
plt.ylim([0, max(rmse_valores) + 0.02])

for i, v in enumerate(rmse_valores):
    plt.text(i, v + 0.002, str(v), ha='center')

plt.show()
```