

Classificação de Asteróides

UFABC

Aprendizado de Máquina - Noturno

Diego Araujo Giovanini - 11201811750

Pablo Renato Nunes dos Santos - 11202232225

Monica Akemi Rodrigues Kyomen - 11202232221

Karl Eloy Marques Henrique - 11060914

Michel Nunes Araújo - 11046415

Importações

In [1]:

```
import kagglehub
import os
import pandas as pd
import re
from sklearn.pipeline import Pipeline
from feature_engine.encoding import OneHotEncoder, OrdinalEncoder
from feature_engine.scaling import MeanNormalizationScaler
from feature_engine.outliers import ArbitraryOutlierCapper
from feature_engine.selection import SmartCorrelatedSelection
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate, RandomizedSearchCV
from sklearn.metrics import precision_score, recall_score
from scipy.stats import uniform, randint
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns

pd.options.display.max_columns = None
```

```
/home/diego/projeto-final-ml-project-weather-forcast/.venv/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning: IPython not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
from .autonotebook import tqdm as notebook_tqdm
```

Aquisição dos Dados do Kaggle

In [2]:

```
# Download Latest version
path = kagglehub.dataset_download("shrutimehta/nasa-asteroids-classification")

print("Path to dataset files:", path)

# Caminho para o arquivo CSV
csv_path = os.path.join(path, 'nasa.csv')

# Carregar o arquivo CSV em um DataFrame
```

```
df = pd.read_csv(csv_path)

# Salva na pasta data\raw
df.to_parquet('../data/raw/nasa-asteroids.parquet.gzip', compression='gzip', index=False)
```

Path to dataset files: /home/diego/.cache/kagglehub/datasets/shrutimehta/nasa-asteroids-classification/versions/1

```
In [3]: df = pd.read_parquet('../data/raw/nasa-asteroids.parquet.gzip')
```

Preparação dos Dados

```
In [4]: print(df.shape, df['Hazardous'].astype(int).mean())
df.head(2)
```

(4687, 40) 0.16108384894388736

Out[4]:

	Neo Reference ID	Name	Absolute Magnitude	Est Dia in KM(min)	Est Dia in KM(max)	Est Dia in M(min)	Est Dia in M(max)	Est D Miles(i)
0	3703080	3703080	21.6	0.127220	0.284472	127.219879	284.472297	0.071
1	3723955	3723955	21.3	0.146068	0.326618	146.067964	326.617897	0.090

◀ ▶

```
In [5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4687 entries, 0 to 4686
Data columns (total 40 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Neo Reference ID    4687 non-null   int64  
 1   Name              4687 non-null   int64  
 2   Absolute Magnitude 4687 non-null   float64 
 3   Est Dia in KM(min) 4687 non-null   float64 
 4   Est Dia in KM(max) 4687 non-null   float64 
 5   Est Dia in M(min)  4687 non-null   float64 
 6   Est Dia in M(max)  4687 non-null   float64 
 7   Est Dia in Miles(min) 4687 non-null   float64 
 8   Est Dia in Miles(max) 4687 non-null   float64 
 9   Est Dia in Feet(min) 4687 non-null   float64 
 10  Est Dia in Feet(max) 4687 non-null   float64 
 11  Close Approach Date 4687 non-null   object  
 12  Epoch Date Close Approach 4687 non-null   int64  
 13  Relative Velocity km per sec 4687 non-null   float64 
 14  Relative Velocity km per hr  4687 non-null   float64 
 15  Miles per hour          4687 non-null   float64 
 16  Miss Dist.(Astronomical) 4687 non-null   float64 
 17  Miss Dist.(lunar)        4687 non-null   float64 
 18  Miss Dist.(kilometers)   4687 non-null   float64 
 19  Miss Dist.(miles)        4687 non-null   float64 
 20  Orbiting Body           4687 non-null   object  
 21  Orbit ID               4687 non-null   int64  
 22  Orbit Determination Date 4687 non-null   object  
 23  Orbit Uncertainty       4687 non-null   int64  
 24  Minimum Orbit Intersection 4687 non-null   float64 
 25  Jupiter Tisserand Invariant 4687 non-null   float64 
 26  Epoch Osculation        4687 non-null   float64 
 27  Eccentricity            4687 non-null   float64 
 28  Semi Major Axis         4687 non-null   float64 
 29  Inclination             4687 non-null   float64 
 30  Asc Node Longitude      4687 non-null   float64 
 31  Orbital Period          4687 non-null   float64 
 32  Perihelion Distance     4687 non-null   float64 
 33  Perihelion Arg          4687 non-null   float64 
 34  Aphelion Dist           4687 non-null   float64 
 35  Perihelion Time          4687 non-null   float64 
 36  Mean Anomaly             4687 non-null   float64 
 37  Mean Motion              4687 non-null   float64 
 38  Equinox                  4687 non-null   object  
 39  Hazardous                4687 non-null   bool    
dtypes: bool(1), float64(30), int64(5), object(4)
memory usage: 1.4+ MB
```

Visto que não temos valores nulos ou estranhos no dataset, vamos seguir com o ajuste dos dados para a etapa de modelagem.

```
In [11]: object_features = ['orbiting_body']
continuous_features = [
    'absolute_magnitude', 'est_dia_in_kmmin',
    'est_dia_in_kmmmax', 'est_dia_in_mmin', 'est_dia_in_mmax',
    'est_dia_in_milesmin', 'est_dia_in_milesmax', 'est_dia_in_feetmin',
    'est_dia_in_feetmax',
    'epoch_date_close_approach', 'relative_velocity_km_per_sec',
    'relative_velocity_km_per_hr', 'miles_per_hour',
```

```
'miss_distastronomical', 'miss_distlunar', 'miss_distkilometers',
'miss_distmiles', 'orbit_uncertainty',
'minimum_orbit_intersection', 'jupiter_tisserand_invariant',
'epoch_osculation', 'eccentricity', 'semi_major_axis', 'inclination',
'asc_node_longitude', 'orbital_period', 'perihelion_distance',
'perihelion_arg', 'aphelion_dist', 'perihelion_time', 'mean_anomaly',
'mean_motion'
]
target = ['hazardous']
features = object_features + continuous_features
```

In [6]: # Ajustando nome das colunas
df.columns = [re.sub(r'^[a-zA-Z_]', '', col.lower().replace(' ', '_')) for col in df]

In [7]: # Converter a coluna 'Hazardous' para valores numéricos (True = 1, False = 0), p
df['hazardous'] = df['hazardous'].astype(int)

Analise Descritiva

Vamos realizar uma analise geral do dataframe, tentando entender o comportamento univariado e bivariado entre as variáveis e também delas com a target

DataFrame

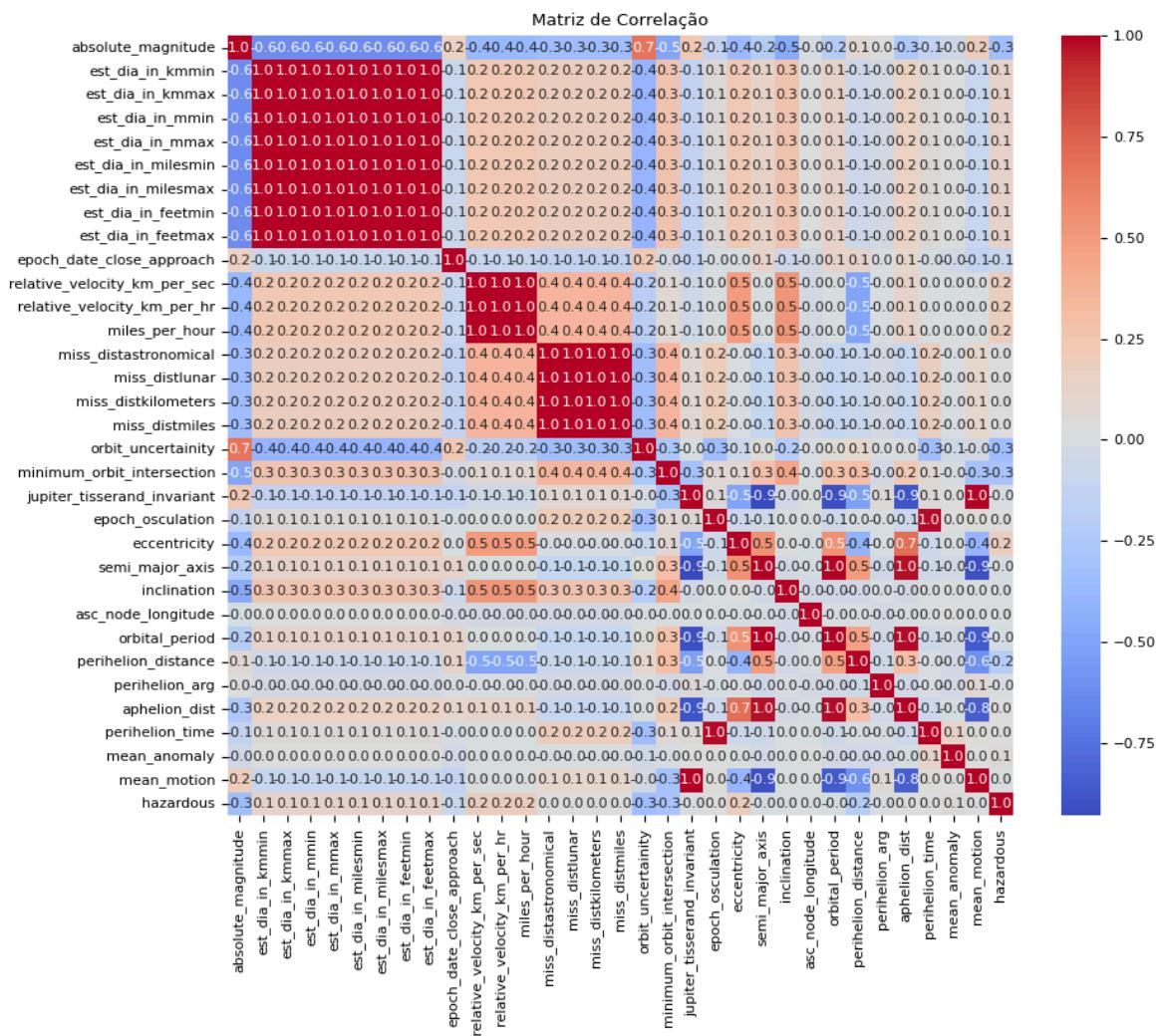
In []: print(f'Shape: {df.shape}')
print(f'Quantidade de Perigosos: {df.hazardous.sum()}')
print(f'Quantidade de Perigosos: {df.shape[0] - df.hazardous.sum()}')
print(f'Taxa de Perigosos: {df.hazardous.mean()}')

Correlação

Como temos muitas variáveis, para facilitar a analise dos dados vamos focar nas variáveis que tem maior correlação com a variável resposta e que não é fortemente correlacionada com outras variáveis.

In [17]: # Como só temos 1 variável categórica vamos ignorar ela para a analise descritiva
df_desc = df[continuous_features + target]

In [18]: plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(10, 8))
sns.heatmap(df_desc.corr(), annot=True, cmap='coolwarm', fmt=".1f")
plt.title("Matriz de Correlação")
plt.show()



Observando o heatmap de correlação de pearson, é possível notar que não temos nenhuma variável muito correlacionado com a target *hazardous*, então possivelmente não temos um problema de data leakage.

No heatmap também é possível avaliar a correlação entre as features, onde encontramos diversas features 100% correlacionadas, mostrando que tem a mesma distribuição, podemos então remove-las, mantendo apenas uma para reduzir a dimensionalidade e seguir com nossa analise.

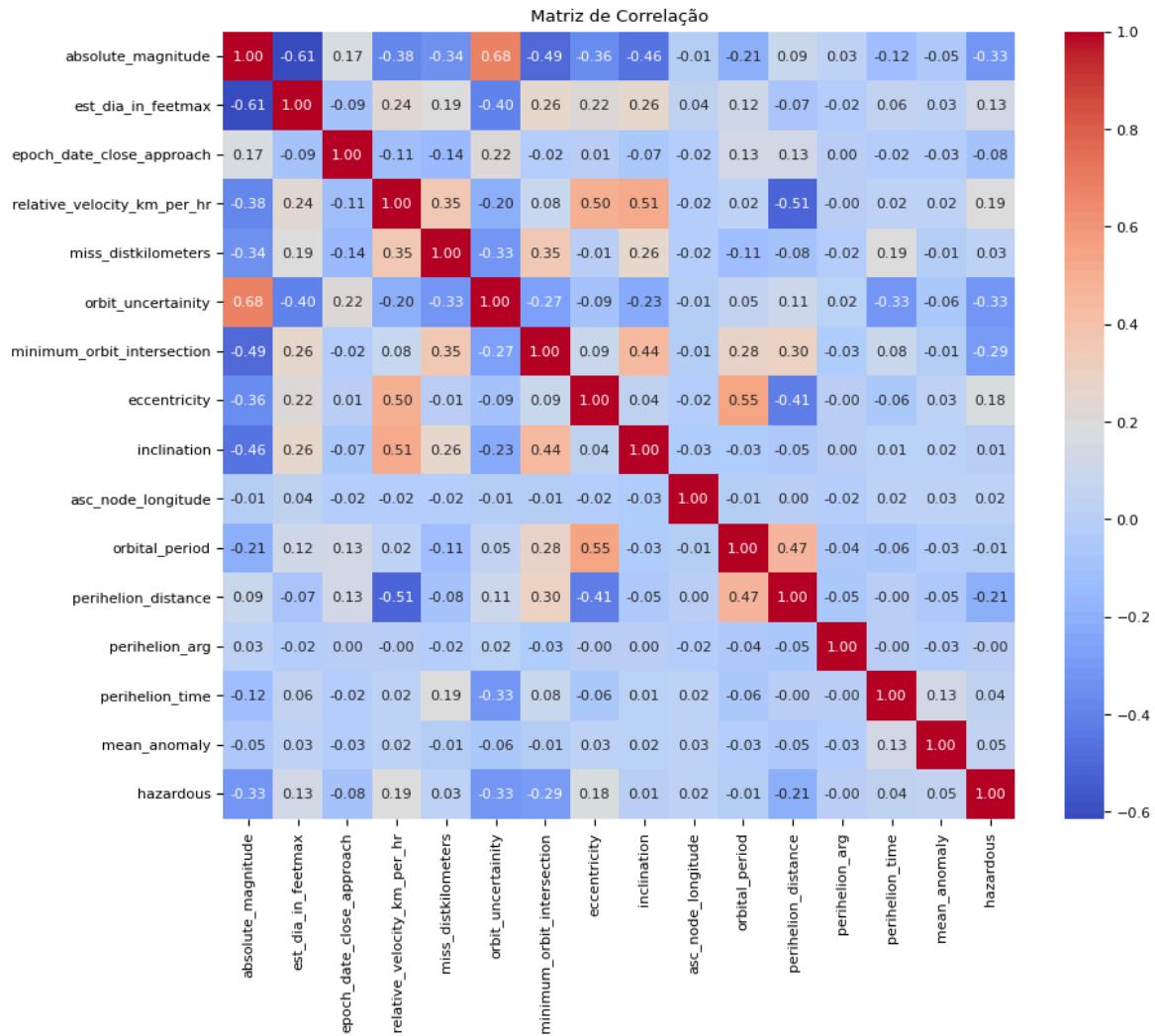
Além das 100% correlacionadas, temos features com correlação > 90%, vamos seguir removendo essas features também.

Para remoção vamos utilizar a biblioteca feature-engine, que fornece um metodo de remoção de correlação mantendo as variáveis com maior variância, estamos assumindo que quando maior variância, mais informação a variável traz

```
In [22]: sc = SmartCorrelatedSelection(variables=list(df_desc.drop(columns=target)).columns,
                                      method='pearson',
                                      threshold=0.89,
                                      selection_method='variance') # Vamos priorizar fea
```

```
In [23]: df_desc = sc.fit_transform(df_desc)
```

```
In [25]: plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(10, 8))
sns.heatmap(df_desc.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matriz de Correlação")
plt.show()
```



Agora com menos variáveis correlacionadas podemos observar melhor a relação das features com a target.

As features `absolute_magnitude` e `orbit_uncertainty` são as mais correlacionadas negativamente com a variável resposta, e ao observar a correlação entre as duas é uma correlação relevante de 0.68, indicando que possivelmente elas tem alguma característica em comum para seu cálculo.

E a variável que tem maior correlação positiva é `relative_velocity_km_per_hr`, que também faz sentido, quanto mais rápido um asteroide, possivelmente pode causar uma catástrofe.

Distribuições

Vamos entender como é a distribuição de cada uma dessas variáveis

```
In [34]: df_desc.describe(percentiles=[.25, .5, .75, .99]).T
```

Out[34]:

		count	mean	std	min
	absolute_magnitude	4687.0	2.226786e+01	2.890972e+00	1.116000e+01
	est_dia_in_feetmax	4687.0	1.501014e+03	2.711257e+03	7.413530e+00
	epoch_date_close_approach	4687.0	1.179881e+12	1.981540e+11	7.889472e+11
	relative_velocity_km_per_hr	4687.0	5.029492e+04	2.625560e+04	1.207815e+03
	miss_distkilometers	4687.0	3.841347e+07	2.181110e+07	2.660989e+04
	orbit_uncertainty	4687.0	3.516962e+00	3.078307e+00	0.000000e+00
	minimum_orbit_intersection	4687.0	8.232007e-02	9.029997e-02	2.061110e-06
	eccentricity	4687.0	3.825691e-01	1.804438e-01	7.522355e-03
	inclination	4687.0	1.337384e+01	1.093623e+01	1.451294e-02
	asc_node_longitude	4687.0	1.721573e+02	1.032768e+02	1.940674e-03
	orbital_period	4687.0	6.355821e+02	3.709547e+02	1.765572e+02
	perihelion_distance	4687.0	8.133833e-01	2.420591e-01	8.074430e-02
	perihelion_arg	4687.0	1.839322e+02	1.035130e+02	6.917625e-03
	perihelion_time	4687.0	2.457728e+06	9.442264e+02	2.450100e+06
	mean_anomaly	4687.0	1.811679e+02	1.075016e+02	3.191491e-03
	hazardous	4687.0	1.610838e-01	3.676475e-01	0.000000e+00

In [41]:

```

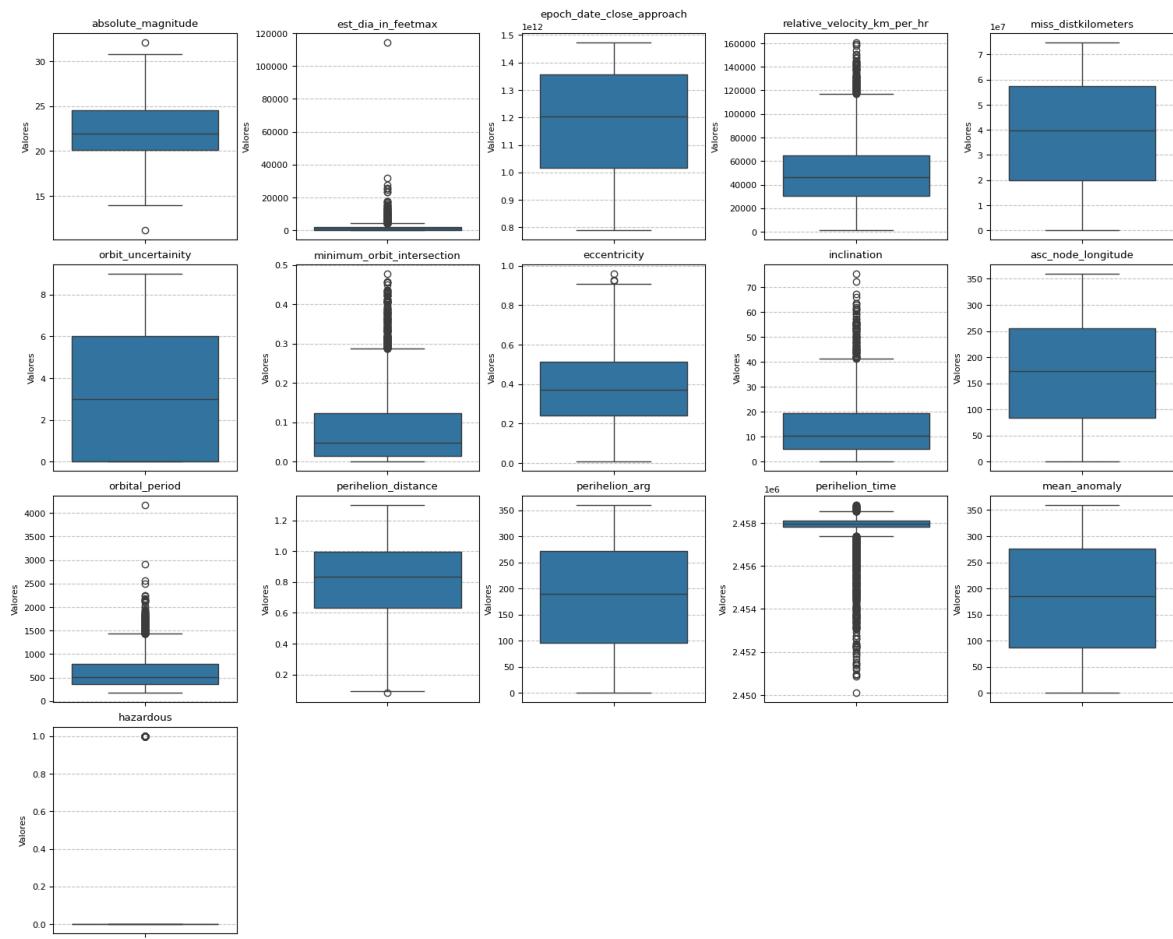
n_cols = 5
n_rows = -(len(df_desc.columns) // n_cols)
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows), constrained_layout=True)

for idx, column in enumerate(df_desc.columns):
    row, col = divmod(idx, n_cols)
    ax = axes[row, col] if n_rows > 1 else axes[col]
    sns.boxplot(df_desc[column], ax=ax)
    ax.set_title(column)
    ax.set_ylabel("Valores")
    ax.grid(axis='y', linestyle='--', alpha=0.7)

for idx in range(len(df_desc.columns), n_rows * n_cols):
    row, col = divmod(idx, n_cols)
    if n_rows > 1:
        fig.delaxes(axes[row, col])
    else:
        fig.delaxes(axes[col])

plt.show()

```

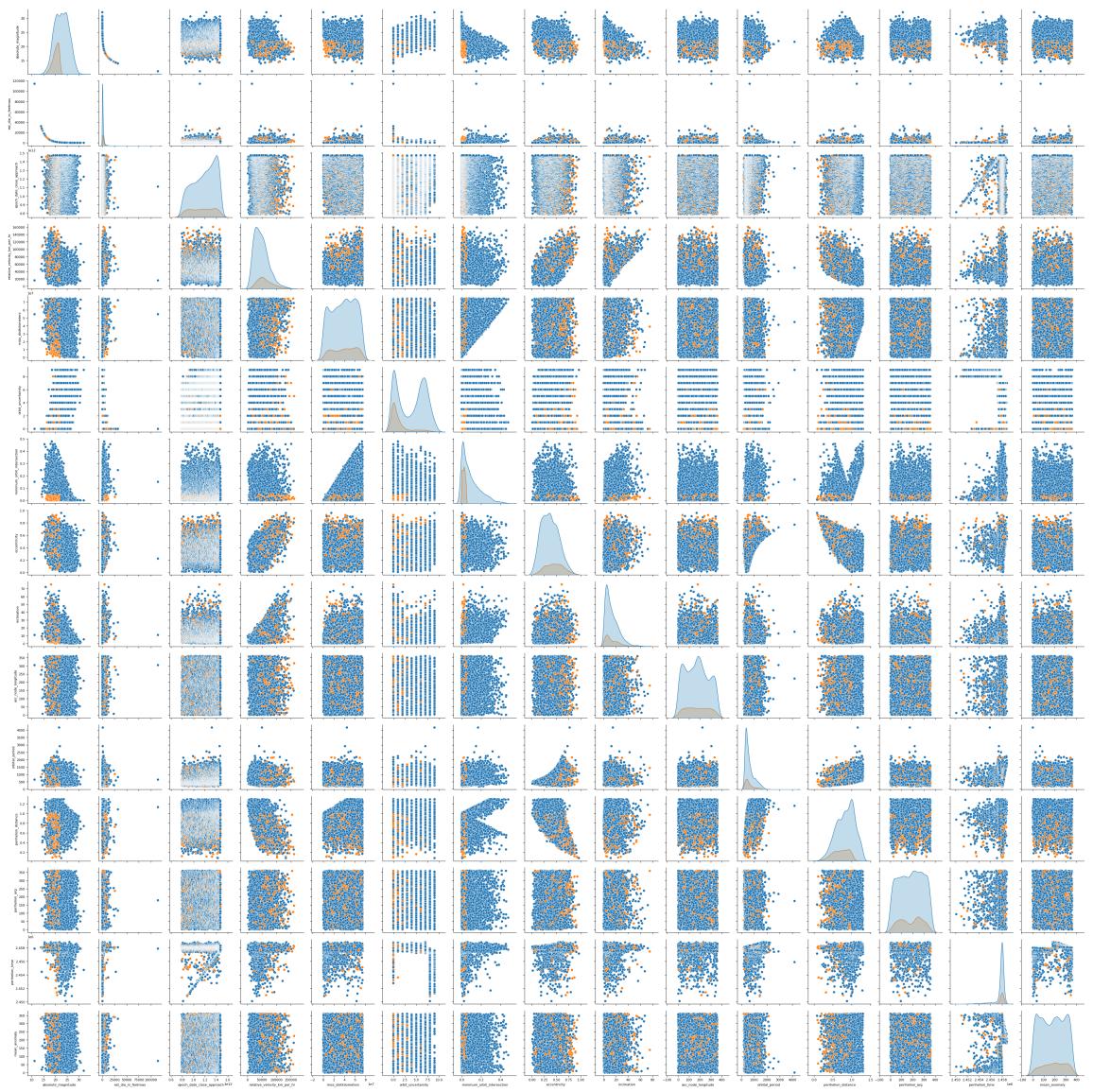


Observando a distribuição pelos percentis e pelos boxplots com valor > IQ3 temos algumas colunas que contêm outliers, em caso de utilização de modelos que utilizem distância vamos realizar o tratamento deles, travando o valor das variáveis em seu percentil 99.

Já para os modelos baseados em árvore não vamos realizar, visto que eles conseguem lidar com outliers sem impactar na performance.

```
In [26]: sns.pairplot(df_desc, hue='hazardous')
```

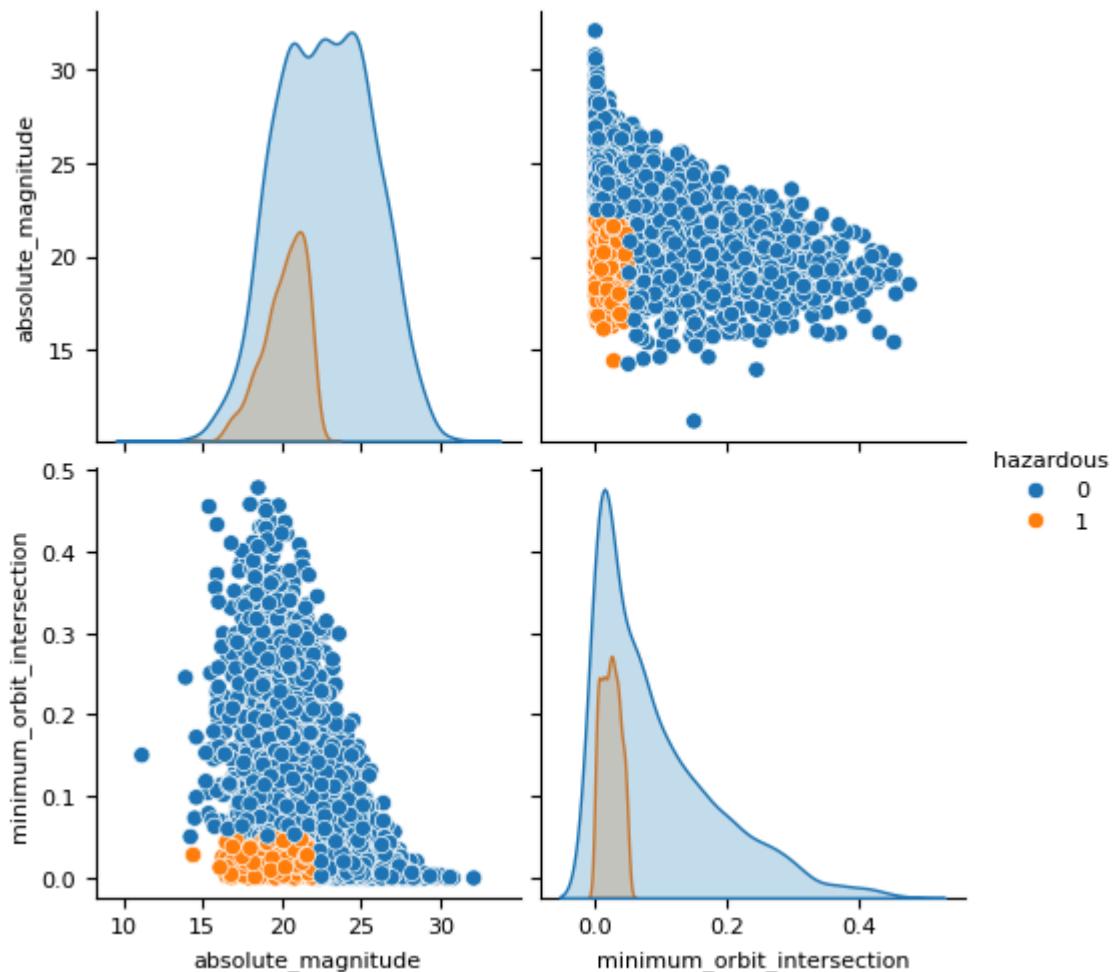
```
Out[26]: <seaborn.axisgrid.PairGrid at 0x7fbf7c242900>
```



Observando os pairplots é possível notar uma singela separação em duas variáveis específicas, mostrando uma relação não linear para separação entre astetórides perigosos e não perigosos, elas são: `minimum_orbit_intersection` e `absolute_magnitude`. É possível notar que quanto menor as duas temos uma separação muito boa entre perigosos e não perigosos, vamos dar um zoom no gráfico abaixo.

```
In [30]: sns.pairplot(df_desc[['absolute_magnitude', 'minimum_orbit_intersection', 'hazard']])
```

```
Out[30]: <seaborn.axisgrid.PairGrid at 0x7fbf771d24e0>
```



Provavelmente quanto utilizarmos modelos que traçam relações não lineares, essas variáveis serão de grande importância

Pipelines para Tratamento dos dados

Vamos criar pipelines de preparação dos dados diferentes para técnicas diferentes.

- Para as técnicas baseadas em árvores vamos utilizar a preparação com Ordinal Encoder sem normalizar os dados, visto que os métodos de árvore conseguem traçar relações não lineares e trabalhar com diferentes escalas e outliers.
- Já para as técnicas baseadas em distância e métodos lineares, realizamos o OneHot Encoding, com normalização e tratamento dos outliers (foram travados no valor do p99 da variável), visto que esses métodos precisam desses tratamentos para performarem melhor.

```
In [12]: df = df[object_features + continuous_features + target].reset_index(drop=True)
```

OneHot Encoding

```
In [13]: pipe_prep_ohe = Pipeline([
    ('oneHotEncoder', OneHotEncoder(drop_last=True, drop_last_binary=True, v
```

```

        max_capping_dict={
            column:df[column].quantile(0.99) for column in continuous_features
        },
        min_capping_dict={
            column:df[column].quantile(0.01) for column in continuous_features
        },
        missing_values='ignore'
    )),
    ('scaler', MeanNormalizationScaler(variables=continuous_features))
]
)
#pipe_prep_ohe.fit_transform(df[object_features + continuous_features], df[target])

```

Ordinal Encoding

```

In [14]: pipe_prep_ord = Pipeline(
    [
        ('ordinalEncoder', OrdinalEncoder(encoding_method='arbitrary', variables=continuous_features))
    ]
)
#pipe_prep_ord.fit_transform(df[object_features + continuous_features], df[target])

```

Modelagem

Para modelagem vamos utilizar 3 técnicas diferentes.

- KNN
- Random Forest
- Boosting com XGBoost

Vamos utilizar 2 técnicas em conjunto para realizar a validação dos modelos.

- 1º Vamos separar o conjunto de dados em um conjunto de treino e teste, sendo o treino com 80% dos dados e o teste com 20%.
- 2º Vamos utilizar cross-validation no conjunto de treinamento para realizar a busca de hiperparametros dos modelos.

Ao final para avaliar qual o melhor modelo para esse conjunto de dados, vamos analisar as métricas de Precision e Recall, escolhemos essas duas métricas por se tratar de um conjunto desbalanceado.

Separação Treino x Teste

Como estamos trabalhando com um problema de dados desbalanceados, vamos utilizar o parametro stratify, mantendo a mesma proporção da target entre treino e teste.

```

In [15]: df_train, df_test = train_test_split(df, train_size=0.8, stratify=df[target], random_state=42)
print(df_train.shape, df_train['hazardous'].mean())
print(df_test.shape, df_test['hazardous'].mean())

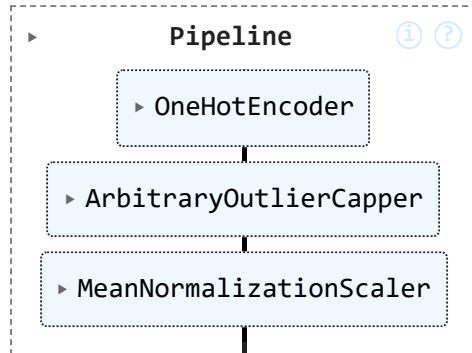
```

```
(3749, 34) 0.16110962923446254
(938, 34) 0.16098081023454158
```

Ajuste Pipelines de Transformações

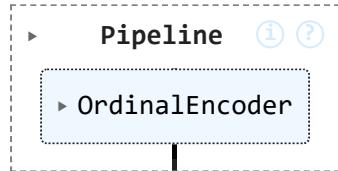
In [16]: `# Vamos ajustar os transformadores de dados apenas no conjunto de treinamento, e
pipe_prep_ohe.fit(df_train)`

Out[16]:



In [17]: `pipe_prep_ord.fit(df_train)`

Out[17]:



KNN

Processando Dados com Pipeline

Por ser uma modelo linear (distâncias), vamos utilizar a pipeline de dados com OneHot Encoding, normalização e ajuste de outliers.

In [18]: `df_prep_knn = pipe_prep_ohe.transform(df_train)
df_prep_knn_test = pipe_prep_ohe.transform(df_test)
df_prep_knn.head(2)`

Out[18]:

	absolute_magnitude	est_dia_in_kmmin	est_dia_in_kmmax	est_dia_in_mmin	est_dia_in_mmax
2612	-0.265092	0.161195	0.161195	0.161195	0.161195
3938	0.199115	-0.118064	-0.118064	-0.118064	-0.118064

Como estamos utilizando o algoritmo de distância KNN e sabemos que ele sofre com problemas de alta dimensionalidade, vamos executar uma seleção de features através da correlação. Vamos remover features altamente correlacionados entre si. Para isso vamos utilizar a biblioteca feature-engine, que implementa um método de remoção de correlação.

```
In [19]: sc = SmartCorrelatedSelection(variables=list(df_prep_knn.drop(columns=target)).columns,
                                     method='pearson',
                                     threshold=0.9,
                                     selection_method='variance') # Vamos priorizar features
```

```
In [20]: # Através da remoção de correlação saímos de 33 variáveis para 17 variáveis, reduzindo o problema
print(df_prep_knn.shape)
df_prep_knn = sc.fit_transform(df_prep_knn)
df_prep_knn_test = sc.fit_transform(df_prep_knn_test)
print(df_prep_knn.shape)
df_prep_knn.head(2)
```

(3749, 33)
(3749, 17)

	absolute_magnitude	est_dia_in_milesmin	epoch_date_close_approach	relative_velo
2612	-0.265092	0.161195	0.088453	
3938	0.199115	-0.118064	0.331530	

Tunning

Vamos realizar a busca de hiperparametros do modelo utilizando um RandomSearch com Validação Cruzada.

```
In [21]: knn = KNeighborsClassifier(
    n_jobs=-1
)
knn
```

```
Out[21]: ▾ KNeighborsClassifier ⓘ ?  
KNeighborsClassifier(n_jobs=-1)
```

```
In [22]: param_distributions = {
    'n_neighbors' : randint(2, 15),
    'weights' : ['uniform', 'distance'],
    'p' : randint(1, 10)
}
```

```
In [23]: rsearch = RandomizedSearchCV(knn,
                                    param_distributions,
                                    random_state=777, n_iter=10,
                                    cv=5,
                                    verbose=1,
                                    n_jobs=-1)
```

```
In [24]: search = rsearch.fit(df_prep_knn.drop(columns=target), df_prep_knn[target].to_numpy()
                           search.best_params_
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
Out[24]: {'n_neighbors': 4, 'p': 1, 'weights': 'distance'}
```

Modelo Final

Com os valores finais dos melhores hiperparametros, vamos treinar um modelo final utilizando todo o conjunto de treino.

```
In [25]: knn = KNeighborsClassifier(
    n_jobs=-1,
    **search.best_params_
)
knn
```

```
Out[25]: ▾ KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1, n_neighbors=4, p=1, weights='distance')
```

```
In [26]: cv_results = cross_validate(
    estimator=knn,
    X=df_prep_knn.drop(columns=target),
    y=df_prep_knn[target].to_numpy().ravel(),
    scoring=['precision', 'recall', 'roc_auc'],
    cv=5,
    return_train_score=True,
    n_jobs=-1
)
```

```
In [27]: # Avaliando esse modelo final no conjunto de treinamento, utilizamos cross-validation
# Ao avaliar as métricas recall e precision é possível notar que estão com um grande desvio entre o treinamento e de validação, indicando overfitting.
# Com esse resultado já poderíamos descartar o modelo KNN, visto o seu super ajuste.
cv_results
```

```
Out[27]: {'fit_time': array([0.00281787, 0.00454044, 0.00405836, 0.00407219, 0.0032649]),
          'score_time': array([0.10089207, 0.09385991, 0.09152532, 0.08769131, 0.08349872]),
          'test_precision': array([0.71428571, 0.83529412, 0.81632653, 0.81188119, 0.82407407]),
          'train_precision': array([1., 1., 1., 1., 1.]),
          'test_recall': array([0.70247934, 0.58677686, 0.66115702, 0.67768595, 0.74166667]),
          'train_recall': array([1., 1., 1., 1., 1.]),
          'test_roc_auc': array([0.92870751, 0.90543168, 0.93768805, 0.9391662, 0.94468071]),
          'train_roc_auc': array([1., 1., 1., 1., 1.])}
```

```
In [28]: knn.fit(df_prep_knn.drop(columns=target), df_prep_knn[target].to_numpy().ravel())
```

```
Out[28]: ▾ KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1, n_neighbors=4, p=1, weights='distance')
```

Random Forest

Processando Dados com Pipeline

```
In [29]: df_prep_rf = pipe_prep_ord.transform(df_train)
df_prep_rf_test = pipe_prep_ord.transform(df_test)
df_prep_rf.head(2)
```

Out[29]:

	orbiting_body	absolute_magnitude	est_dia_in_kmmin	est_dia_in_kmmax	est_dia_
2612	0	19.0	0.421265	0.941976	42
3938	0	24.7	0.030518	0.068240	3

Tunning

Vamos realizar a busca de hiperparametros do modelo utilizando um RandomSearch com Validação Cruzada.

```
In [30]: rf = RandomForestClassifier(
    random_state=777,
    criterion='gini',
    max_features='sqrt',
    n_jobs=-1
)
rf
```

Out[30]:

▼ RandomForestClassifier ⓘ ⓘ

```
RandomForestClassifier(n_jobs=-1, random_state=777)
```

```
In [31]: param_distributions = {
    'n_estimators': [100, 200, 300, 400, 500], # Quantidade de árvores
    'max_depth': randint(3, 10), # profundidade máxima das árvores
    'min_samples_split': randint(10, 1000), # quantidade minima de amostras para
    'min_samples_leaf': randint(10, 1000), # quantidade minima de amostras em um
    'class_weight' : ['balanced', None] # peso adicionado ao erro
}
```

```
In [32]: rsearch = RandomizedSearchCV(rf,
    param_distributions,
    random_state=777,
    n_iter=10,
    cv=5,
    verbose=1,
    n_jobs=-1)
```

```
In [33]: search = rsearch.fit(df_prep_rf[features], df_prep_rf[target].to_numpy().ravel())
search.best_params_
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
Out[33]: {'class_weight': None,
          'max_depth': 6,
          'min_samples_leaf': 113,
          'min_samples_split': 84,
          'n_estimators': 300}
```

Modelo Final

Com os valores finais dos melhores hiperparametros, vamos treinar um modelo final utilizando todo o conjunto de treino.

```
In [34]: rf = RandomForestClassifier(
    random_state=777,
    criterion='gini',
    max_features='sqrt',
    n_jobs=-1,
    **search.best_params_
)
rf
```

```
Out[34]: RandomForestClassifier(max_depth=6, min_samples_leaf=113, min_samples_split=84,
                                n_estimators=300, n_jobs=-1, random_state=777)
```

```
In [35]: cv_results = cross_validate(
    estimator=rf,
    X=df_prep_rf[features],
    y=df_prep_rf[target].to_numpy().ravel(),
    scoring=['precision', 'recall', 'roc_auc'],
    cv=5,
    return_train_score=True,
    n_jobs=-1
)
```

```
In [36]: # Avaliando esse modelo final no conjunto de treinamento, utilizamos cross-validation
# Ao avaliar as métricas recall e precision é possível notar que estão constante
cv_results
```

```
Out[36]: {'fit_time': array([2.09779811, 2.13429189, 2.03475189, 2.01993752, 2.07859588]),
          'score_time': array([0.58074903, 0.57398677, 0.56777978, 0.56067157, 0.55971289]),
          'test_precision': array([0.98373984, 1.           , 1.           , 0.99166667, 0.99152542]),
          'train_precision': array([0.99579832, 0.9978903 , 0.99788584, 0.99371069, 0.99376299]),
          'test_recall': array([1.           , 0.95041322, 0.95867769, 0.98347107, 0.975]),
          'train_recall': array([0.98136646, 0.97929607, 0.97722567, 0.98136646, 0.98760331]),
          'test_roc_auc': array([0.99984233, 0.99647873, 0.99429765, 0.99378523, 0.98994436]),
          'train_roc_auc': array([0.99610279, 0.99632086, 0.99599581, 0.9949565 , 0.99778279])}
```

```
In [37]: rf.fit(df_prep_rf[features], df_prep_rf[target].to_numpy().ravel())
```

Out[37]:

```
RandomForestClassifier(max_depth=6, min_samples_leaf=113, min_samples_split=84,
n_estimators=300, n_jobs=-1, random_state=777)
```

XGBoost

Processando Dados com Pipeline

```
In [38]: df_prep_xgb = pipe_prep_ord.transform(df_train)
df_prep_xgb_test = pipe_prep_ord.transform(df_test)
df_prep_xgb.head(2)
```

Out[38]:

	orbiting_body	absolute_magnitude	est_dia_in_kmmin	est_dia_in_kmmax	est_dia_
2612	0	19.0	0.421265	0.941976	42
3938	0	24.7	0.030518	0.068240	3

Tunning

Vamos realizar a busca de hiperparametros do modelo utilizando um RandomSearch com Validação Cruzada.

```
In [39]: xgb = XGBClassifier(eval_metric='logloss',
                        importance_type='gain',
                        random_state=42)
xgb
```

Out[39]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
ndes=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold>No
ne,
```

```
In [40]: param_distributions = {
    'n_estimators': [100, 200, 300, 400, 500], # Quantidade de árvores
    'max_depth': randint(3, 10), # profundidade máxima das árvores
```

```
'learning_rate' : uniform(0.01, 0.1) # taxa de aprendizagem
}
```

In [41]:

```
rsearch = RandomizedSearchCV(xgb,
                               param_distributions,
                               random_state=777,
                               n_iter=10,
                               cv=5,
                               verbose=1,
                               n_jobs=-1)
```

In [42]:

```
search = rsearch.fit(df_prep_xgb[features], df_prep_xgb[target].to_numpy().ravel())
search.best_params_
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Out[42]:

```
{'learning_rate': 0.02931624056368242, 'max_depth': 5, 'n_estimators': 300}
```

Modelo Final

Com os valores finais dos melhores hiperparametros, vamos treinar um modelo final utilizando todo o conjunto de treino.

In [43]:

```
xgb = XGBClassifier(
    eval_metric='logloss',
    importance_type='gain',
    random_state=42,
    n_jobs=-1,
    **search.best_params_
)
xgb
```

Out[43]:

The screenshot shows the output of the previous code cell. A tooltip-like box is open over the 'xgb' variable, displaying its configuration. The parameters listed are: base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, device=None, early_stopping_rounds=None, enable_categorical=False, eval_metric='logloss', feature_types=None, gamma=None, grow_policy=None, importance_type='gain', interaction_constraints=None, learning_rate=0.02931624056368242, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, and n_estimators=300. The tooltip has a scroll bar on the right side.

In [44]:

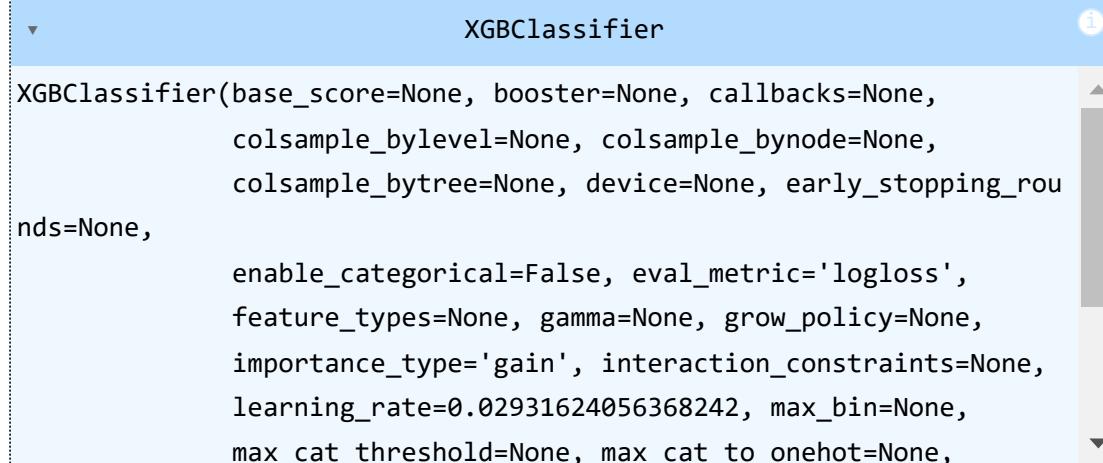
```
cv_results = cross_validate(
    estimator=xgb,
    X=df_prep_xgb[features],
    y=df_prep_xgb[target].to_numpy().ravel(),
    scoring=['precision', 'recall', 'roc_auc'],
    cv=5,
    return_train_score=True,
    n_jobs=-1
)
```

```
In [45]: # Avaliando esse modelo final no conjunto de treinamento, utilizamos cross-validation
# Ao avaliar as métricas recall e precision é possível notar que estão constantes
cv_results
```

```
Out[45]: {'fit_time': array([0.65697551, 0.67556071, 0.59798145, 0.74726725, 0.67209196]),
          'score_time': array([0.04990244, 0.05054879, 0.04885697, 0.03948784, 0.04436612]),
          'test_precision': array([0.97580645, 0.99173554, 0.99173554, 0.99173554, 1.]),
          'train_precision': array([1., 1., 1., 1., 1.]),
          'test_recall': array([1.          , 0.99173554, 0.99173554, 0.99173554, 0.96666667]),
          'train_recall': array([0.99792961, 1.          , 1.          , 1.          , 1.]),
          'test_roc_auc': array([0.99988175, 0.99994744, 0.99889632, 0.99967152, 0.99972178]),
          'train_roc_auc': array([1., 1., 1., 1., 1.])}
```

```
In [46]: xgb.fit(df_prep_xgb[features], df_prep_xgb[target].to_numpy().ravel())
```

```
Out[46]: XGBClassifier
```



```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_roun
              nds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.02931624056368242, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
```

Resultado Final

Após o treinamento dos 3 modelos e alguns resultados parciais já obtidos, vamos seguir com a avaliação deles no conjunto de teste para eleger o melhor modelo para esse conjunto de dados.

Métricas Treino x Teste

```
In [47]: df_metrics = pd.DataFrame(
    {
        'model' : ['knn', 'random_forest', 'xgboost'],
        'train_precision' : [
            precision_score(knn.predict(df_prep_knn.drop(columns=target)), df_prep_knn[target]),
            precision_score(rf.predict(df_prep_rf[features]), df_prep_rf[target]),
            precision_score(xgb.predict(df_prep_xgb[features]), df_prep_xgb[target])
        ],
        'test_precision' : [
            precision_score(knn.predict(df_prep_knn_test.drop(columns=target)),

```

```

precision_score(rf.predict(df_prep_rf_test[features]), df_prep_rf_te
precision_score(xgb.predict(df_prep_xgb_test[features]), df_prep_xgb
],
'train_recall' : [
    recall_score(knn.predict(df_prep_knn.drop(columns=target)), df_prep_
recall_score(rf.predict(df_prep_rf[features]), df_prep_rf[target]),
recall_score(xgb.predict(df_prep_xgb[features]), df_prep_xgb[target]
],
'test_recall' : [
    recall_score(knn.predict(df_prep_knn_test.drop(columns=target)), df_
recall_score(rf.predict(df_prep_rf_test[features]), df_prep_rf_test[
recall_score(xgb.predict(df_prep_xgb_test[features]), df_prep_xgb_te
]
]
)
df_metrics

```

Out[47]:

	model	train_precision	test_precision	train_recall	test_recall
0	knn	1.000000	0.721854	1.000000	0.778571
1	random_forest	0.985099	0.966887	0.993322	0.973333
2	xgboost	0.998344	0.993377	1.000000	0.980392

In [48]:

```

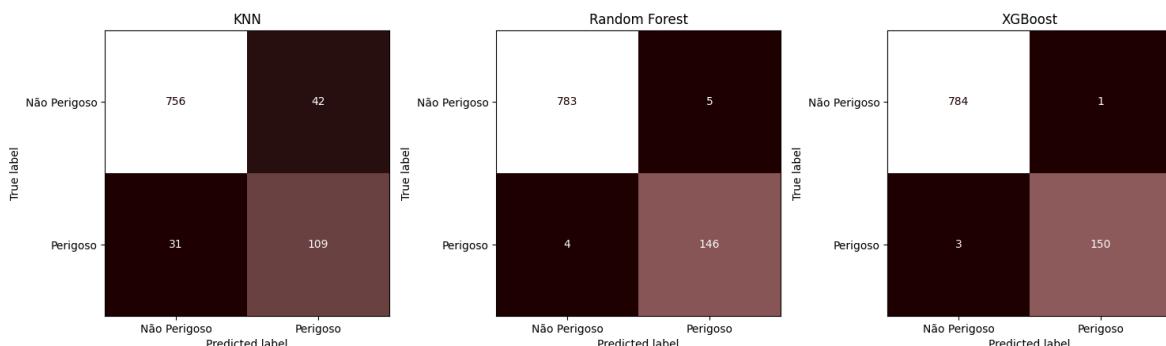
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

cm_model_knn = confusion_matrix(knn.predict(df_prep_knn_test.drop(columns=target
cm_model_rf = confusion_matrix(rf.predict(df_prep_rf_test[features]), df_prep_rf
cm_model_xgb = confusion_matrix(xgb.predict(df_prep_xgb_test[features]), df_prep

for ax, cm, title in zip(axes, [cm_model_knn, cm_model_rf, cm_model_xgb], ["KNN"
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Não Perigoso', 'Perigoso'])
    disp.plot(ax=ax, colorbar=False, cmap=plt.cm.Pink)
    ax.set_title(title)

plt.tight_layout()
plt.show()

```



Observando os resultados das métricas entre os modelos, é possível concluir que o melhor modelo foi o de boosting, tendo a menor diferença entre os conjuntos de treino e teste, indicando um ótimo trade-off de viés e variância, além da ótima precisão.

Observando especificamente a matriz de confusão para o *XGBoost*, temos:

- Verdadeiros Negativos (784): Asteroides corretamente classificados como "Não Perigoso".

- Falsos Negativos (3): Asteroídes que foram classificados incorretamente como "Não Perigoso", mas que na verdade são "Perigoso".
- Falsos Positivos (1): Asteroídes classificados incorretamente como "Perigoso", mas que na verdade são "Não Perigoso".
- Verdadeiros Positivos (150): Asteroídes corretamente classificados como "Perigoso".

O baixo número de falsos negativos (3) é um destaque importante, pois indica que o modelo consegue identificar a maioria dos asteroídes perigosos, o que é crucial para a segurança e monitoramento espacial.

Interpretação e Impacto

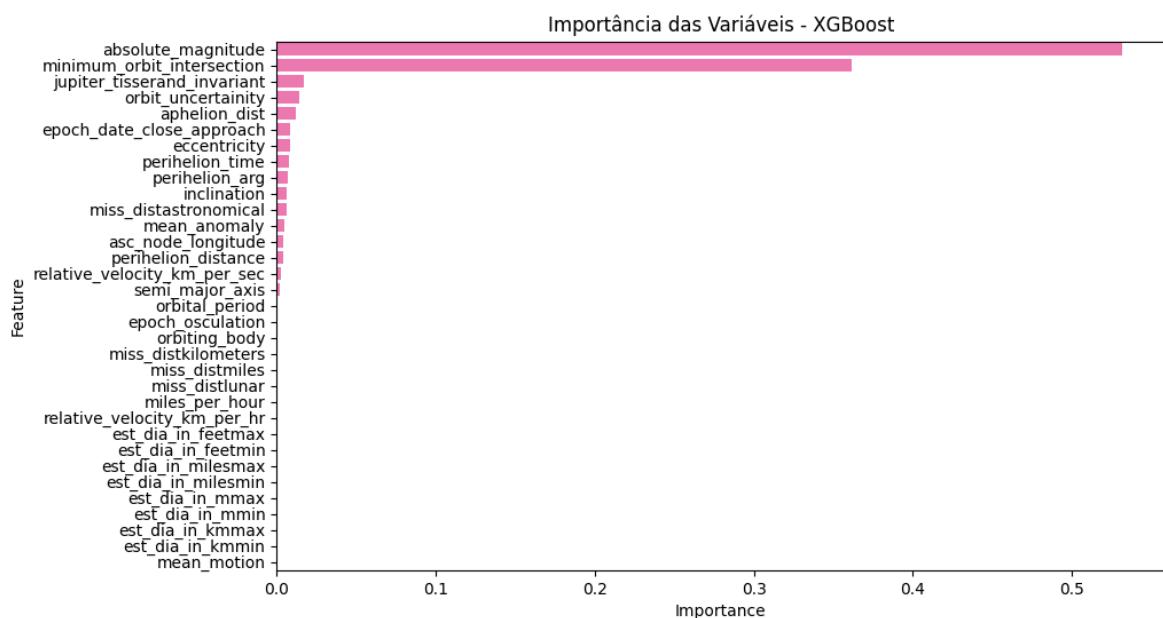
A alta precisão e recall para a classe "Perigoso" mostram que o modelo está bem calibrado para detectar asteroídes potencialmente perigosos, minimizando o risco de não identificar objetos que poderiam representar uma ameaça. Isso é especialmente relevante em aplicações espaciais, onde a detecção precoce de asteroídes perigosos é essencial para a prevenção de desastres.

Feature Importance

Vamos avaliar o feature importance do XGBoost para entender quais as variáveis que levaram a uma boa performance do modelo.

```
In [55]: feature_importances = pd.DataFrame(
    {
        'Feature' : xgb.feature_names_in_,
        'Importance' : xgb.feature_importances_
    }
).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importances, color='#FF66B')
plt.title('Importância das Variáveis - XGBoost')
plt.show()
```



Analisando a importância das variáveis, é possível notar que a variável mais importante é a magnitude do asteróide (uma medida do brilho intrínseco de um objeto celeste, como uma estrela, planeta ou galáxia) é definida como o brilho que o objeto teria se estivesse a uma distância de 10 parsecs (aproximadamente 32,6 anos luz da Terra)), seguido da distância mínima de intersecção com a órbita, que mede o risco de colisão com a terra baseado na órbita dos dois astros.

Como foi observado na análise descritiva essas eram variáveis que conseguiam separar bem a variável resposta com uma relação não linear, no feature importance é possível confirmar que o modelo capturou essa informação.

Também é possível notar que temos diversas variáveis com importância 0 no modelo, mostrando que é possível realizar um trabalho de redução de dimensionalidade, reduzindo complexidade e explicabilidade do modelo.

Conclusão

O modelo apresentou resultados robustos, com alta eficácia na classificação de asteroides. Este trabalho pode ser aprimorado incorporando mais dados ou refinando a engenharia de características para melhorar ainda mais o desempenho em aplicações reais.