Para preparar os dados para análise com um algoritmo de Machine Learning, é necessário realizar algumas etapas e técnicas de pré-processamento dos dados. Aqui estão algumas sugestões de etapas que você pode considerar:

1. **Carregamento dos dados:** Carregue os dados dos arquivos CSV para estruturas de dados adequadas, como DataFrames, utilizando bibliotecas como o `pandas`.
2. **Limpeza dos dados:** Verifique se há dados faltantes, inconsistentes ou inválidos nos seus conjuntos de dados. Trate esses valores ausentes ou inconsistentes, seja removendo as linhas correspondentes ou preenchendo-os com valores apropriados, como a média dos valores da coluna. A função `fillna()` do `pandas` pode ser útil para isso. Também realize a multiplicação de pesos pela frequência no caso de dados ponderados.
3. **Análise exploratória** **de dados**: é uma etapa fundamental no processo de modelagem de Machine Learning. Ela envolve a exploração e compreensão dos dados por meio de técnicas estatísticas e visualização, com o objetivo de extrair insights, identificar padrões, detectar anomalias e preparar os dados para o desenvolvimento do modelo. Com base na análise exploratória, é possível identificar as features mais relevantes para o modelo, descartar variáveis irrelevantes e possivelmente identificar relações ou padrões interessantes que possam ajudar na predição das pontuações dos jogadores. Na análise exploratória, podemos realizar as seguintes tarefas:
   1. Análise estatística descritiva: Calcular medidas estatísticas como média, mediana, desvio padrão, mínimo, máximo e quartis para entender a distribuição dos dados.
   2. Visualização de dados: Utilizar gráficos e visualizações para representar os dados de forma intuitiva, identificar padrões, relações e tendências. Exemplos de gráficos úteis incluem histogramas, gráficos de dispersão, gráficos de barras e boxplots.
   3. Tratamento de outliers: Identificar e tratar valores discrepantes (outliers) que podem afetar a qualidade do modelo.
   4. Análise de correlação: Investigar a relação entre variáveis por meio de coeficientes de correlação, como o coeficiente de Pearson, para entender se existe alguma associação entre as variáveis.
   5. Identificação de dados faltantes: Verificar se existem dados ausentes e decidir como lidar com eles, seja removendo as amostras ou preenchendo os valores ausentes.
   6. Seleção de features: Analisar a relevância das variáveis disponíveis e identificar quais são mais importantes para a construção do modelo.
   7. Análise de variáveis categóricas: Explorar a distribuição das variáveis categóricas e identificar se alguma delas pode ser relevante para o modelo.

Como nossas chaves principais são os campos atleta\_id e rodada\_id, ou seja, cada linha do dataframe traz o desempenho do atleta naquela rodada, e o campor "partida\_id" não faz parte da tabela-base, Pontuações Jogadores precisaremos fazer uma verificação nas tabelas "Dados Partidas Realizadas" e "Dados Partidas Atuais" a partir das chaves rodada\_id e clube\_id que constam nas três tabelas para conseguir montar o dataframe único.

Inicialmente, será necessário criar o campo "clube\_situacao\_rodada" para indicar o resultado das verificações abaixo sugeridas, com os valores "mandante" para o time que joga em casa e "visitante" para o time que joga fora de casa:

1) se na rodada o time do atleta aparecer na coluna "clube\_visitante\_id", teremos as seguintes colunas: "situacao\_clube" == visitante; e as colunas importadas das duas tabelas - rodada\_id, partida\_id, partida\_data, clube\_casa\_id, clube\_casa\_nome, clube\_casa\_posicao, aproveitamento\_mandante, placar\_oficial\_mandante, placar\_oficial\_visitante e valida.

2) se na rodada o time do atleta aparecer na coluna "clube\_casa\_id", teremos as seguintes colunas: "situacao\_clube" == visitante e as colunas importadas das duas tabelas - rodada\_id, partida\_id, partida\_data, clube\_casa\_id, clube\_visitante\_nome, clube\_visitante\_posicao, aproveitamento\_visitante, placar\_oficial\_mandante, placar\_oficial\_visitante e valida.

1. **Codificação de variáveis categóricas:** Caso haja variáveis categóricas nos seus dados, como nomes de clubes, posicões ou status, será necessário convertê-las em representações numéricas para que o algoritmo de Machine Learning possa processá-las. Uma técnica comum é a codificação one-hot (dummy encoding), que cria uma nova coluna binária para cada categoria. A função `get\_dummies()` do `pandas` pode ser usada para fazer essa codificação.
2. **Seleção de atributos relevantes:** Analise suas variáveis e determine quais são as mais relevantes para a tarefa de previsão do melhor time. Se algumas variáveis não forem informativas ou redundantes, você pode removê-las do conjunto de dados.
3. Sugiro que façamos uma análise exploratória dos dados para entender melhor suas características e identificar possíveis insights relevantes para a modelagem. Podemos visualizar estatísticas descritivas, distribuições, correlações entre as variáveis e explorar visualizações gráficas para entender as relações entre os dados.
4. Podemos começar analisando a tabela "Pontuacoes Jogadores" e explorando as variáveis relacionadas às pontuações dos jogadores. Vamos exibir algumas estatísticas descritivas e criar visualizações para entender melhor a distribuição das pontuações e suas relações com outras variáveis.  
   Ótimo! Vamos analisar a tabela "Pontuacoes Jogadores" e explorar as variáveis relacionadas às pontuações dos jogadores. Para começar, podemos calcular algumas estatísticas descritivas das pontuações, como média, mediana, desvio padrão, mínimo e máximo. Isso nos dará uma ideia geral da distribuição das pontuações.
5. Além disso, podemos criar visualizações para explorar a relação entre as pontuações e outras variáveis, como posição do jogador, clube, etc. Podemos criar gráficos de barras ou boxplots para comparar as pontuações médias entre as posições ou clubes. Também podemos criar histogramas para visualizar a distribuição das pontuações.
6. Com essas análises, podemos obter insights sobre como as pontuações dos jogadores estão distribuídas e se existem diferenças significativas entre as posições ou clubes. Essas informações podem ser úteis para entender os padrões e tendências das pontuações dos jogadores e podem fornecer insights para o desenvolvimento do modelo de previsão.
7. Você gostaria que eu fornecesse o código para realizar essas análises e visualizações?
8. 5. Normalização/Padronização: Dependendo do algoritmo de Machine Learning que você pretende usar, pode ser necessário normalizar ou padronizar as variáveis numéricas. Isso é especialmente útil para algoritmos que se baseiam em medidas de distância ou que requerem dados com média zero e variância unitária. A função `StandardScaler()` da biblioteca `sklearn.preprocessing` pode ser utilizada para padronização.
9. 6. Divisão dos dados em conjuntos de treinamento e teste: Separe seus dados em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento será usado para treinar o modelo, enquanto o conjunto de teste será usado para avaliar seu desempenho em dados não vistos. A função `train\_test\_split()` da biblioteca `sklearn.model\_selection` pode ajudar nessa divisão.
10. 7. Engenharia de atributos: Dependendo da natureza do problema, você pode querer criar novos atributos derivados dos atributos existentes para fornecer informações adicionais ao modelo. Por exemplo, se você tiver informações sobre o desempenho do jogador nas últimas rodadas, pode calcular a média ou soma dessas pontuações e adicionar como um novo atributo.
11. 8. Treinamento do modelo: Escolha o algoritmo de Machine Learning adequado para a sua tarefa e treine o modelo utilizando o conjunto de treinamento. Existem diversos algoritmos que podem ser explorados, como regressão linear, árvores de decisão, random forest, redes neurais, entre outros. A biblioteca `scikit-learn` fornece implementações de vários algoritmos de Machine Learning em Python.
12. 9. Avaliação do modelo: Após o treinamento, avalie o desempenho do modelo utilizando o conjunto de teste. Calcule métricas relevantes, como acurácia, precisão, recall, F1-score, entre outras, dependendo do tipo de problema que está sendo tratado. A função `classification\_report()` da biblioteca `sklearn.metrics` pode ajudar na avaliação de problemas de classificação.

10. Ajuste de hiperparâmetros: Experimente diferentes configurações e hiperparâmetros do modelo para encontrar a combinação que melhor se ajusta aos seus dados. Isso pode ser feito utilizando técnicas de busca, como GridSearch ou RandomizedSearch, para encontrar a melhor combinação de hiperparâmetros.

Essas são apenas algumas das etapas e técnicas que você pode considerar ao preparar os dados para análise com um algoritmo de Machine Learning. A escolha das etapas e técnicas específicas depende do seu conjunto de dados, do tipo de problema que você está resolvendo e do algoritmo que pretende usar. É importante explorar diferentes abordagens e iterar no processo de pré-processamento até encontrar a configuração adequada para o seu caso específico.

atleta\_id

rodada\_id

status\_id

atleta\_id

preco\_editorial

escalacoes

rodada\_id

partida\_id

partida\_data

clube\_casa\_id

clube\_casa\_posicao

aproveitamento\_mandante

clube\_visitante\_id

clube\_visitante\_posicao

aproveitamento\_visitante

valida

rodada\_id

partida\_id

partida\_data

clube\_casa\_id

clube\_casa\_posicao

aproveitamento\_mandante

clube\_visitante\_id

clube\_visitante\_posicao

aproveitamento\_visitante

placar\_oficial\_mandante

placar\_oficial\_visitante

valida

clube\_id

clube\_nome

abreviacao

atleta\_id

apelido

posicao\_id

clube\_id

entrou\_em\_campo

rodada\_id

CA

DS

FC

FF

FD

FS

I

SG

A

G

DE

GS

V

PS

FT

PP

DP

CV

PC

GC

pontuação

Vamos partir dessa versão do código para realizar algumas alterações:

import os

import pandas as pd

ARQUIVOS\_CSV = {

'Atletas mercado': 'atletas\_mercado.csv',

'Dados Destaque': 'dados\_destaque.csv',

'Dados Partidas Atual': 'dados\_partidas\_atual.csv',

'Dados Partidas Realizadas': 'dados\_partidas\_realizadas.csv',

'Dados Times': 'dados\_times.csv',

'Pontuacoes Jogadores': 'pontuacoes\_jogadores.csv'

}

SCOUT\_WEIGHTS = {

'DS': 1.2,

'FC': -0.3,

'GC': -3.0,

'CA': -1.0,

'CV': -3.0,

'SG': 5.0,

'DE': 1.0,

'DP': 7.0,

'GS': -1.0,

'PC': -1.0,

'FS': 0.5,

'I': -0.1,

'A': 5.0,

'FT': 3.0,

'FD': 1.2,

'FF': 0.8,

'G': 8.0,

'PP': -4.0,

'PS': 1.0,

'V': 1.0

}

def carregar\_dados():

diretorio = 'bd\_cartola\_modelo'

tabelas = {}

for nome\_tabela, nome\_arquivo in ARQUIVOS\_CSV.items():

caminho\_arquivo = os.path.join(diretorio, nome\_arquivo)

if os.path.exists(caminho\_arquivo):

tabela = pd.read\_csv(caminho\_arquivo)

tabelas[nome\_tabela] = tabela

else:

print(f'Arquivo não encontrado: {caminho\_arquivo}')

if len(tabelas) != len(ARQUIVOS\_CSV):

# Verificar quais arquivos estão faltando

arquivos\_faltantes = set(ARQUIVOS\_CSV.values()) - set(tabelas.keys())

for arquivo in arquivos\_faltantes:

print(f'Arquivo faltando: {arquivo}')

return tabelas

def preencher\_campos\_vazios(tabelas):

for nome\_tabela, tabela in tabelas.items():

tabela.fillna(0, inplace=True)

# Tratamento específico para a tabela "Atletas mercado"

if 'Atletas mercado' in tabelas:

tabela\_atletas = tabelas['Atletas mercado']

tabela\_atletas.fillna(0, inplace=True)

tabela\_atletas[['minimo\_para\_valorizar']] = tabela\_atletas[['minimo\_para\_valorizar']].astype(int)

for scout in SCOUT\_WEIGHTS.keys():

if scout in tabela\_atletas:

tabela\_atletas[scout] = tabela\_atletas[scout] \* SCOUT\_WEIGHTS[scout]

# Tratamento específico para a tabela "Pontuacoes Jogadores"

if 'Pontuacoes Jogadores' in tabelas:

tabela\_pontuacoes = tabelas['Pontuacoes Jogadores']

tabela\_pontuacoes.fillna(0, inplace=True)

for scout in SCOUT\_WEIGHTS.keys():

if scout in tabela\_pontuacoes:

tabela\_pontuacoes[scout] = tabela\_pontuacoes[scout] \* SCOUT\_WEIGHTS[scout]

def criar\_dataframe(tabelas):

# Selecionar apenas as tabelas relevantes para o modelo

tabelas\_selecionadas = ['Atletas mercado', 'Dados Destaque', 'Dados Partidas Atual', 'Dados Partidas Realizadas', 'Dados Times', 'Pontuacoes Jogadores']

tabelas\_filtradas = {nome\_tabela: tabela for nome\_tabela, tabela in tabelas.items() if nome\_tabela in tabelas\_selecionadas}

# Mesclar as tabelas relevantes em um único DataFrame

df = pd.concat(tabelas\_filtradas.values(), keys=tabelas\_filtradas.keys(), axis=1)

# Preencher campos vazios com zero

df.fillna(0, inplace=True)

# Aplicar pesos aos scouts na tabela "Pontuacoes Jogadores"

if 'Pontuacoes Jogadores' in tabelas\_filtradas:

tabela\_pontuacoes = df['Pontuacoes Jogadores']

for scout, peso in SCOUT\_WEIGHTS.items():

if scout in tabela\_pontuacoes:

tabela\_pontuacoes[scout] = tabela\_pontuacoes[scout] \* peso

return df

# Passo 1: Carregar os dados dos arquivos CSV

tabelas = carregar\_dados()

# Passo 2: Preencher campos vazios com zero

preencher\_campos\_vazios(tabelas)

# Passo 3: Criar DataFrame único com as tabelas relevantes

df = criar\_dataframe(tabelas)

# Verificar se todas as tabelas foram carregadas corretamente após o preenchimento

tabelas\_completas = True

for nome\_tabela, tabela in tabelas.items():

if tabela.isna().any().any():

print(f'Valores ausentes encontrados na tabela: {nome\_tabela}')

colunas\_ausentes = tabela.columns[tabela.isna().any()].tolist()

print(f'Colunas com valores ausentes: {colunas\_ausentes}')

tabelas\_completas = False

if tabelas\_completas:

print('Limpeza dos dados concluída. Os arquivos CSV limpos foram salvos.')

print('DataFrame consolidado:')

print(df.head())

else:

print('Erro ao carregar tabelas. Verifique se todos os arquivos estão presentes ou se há valores ausentes nos dados.')

Percebi que tratar os dados das partidas em tabelas separadas não era adequado. Por isso, substituiu as tabelas 'Dados Partidas Atual' ('dados\_partidas\_atual.csv') e 'Dados Partidas Realizadas' ('dados\_partidas\_realizadas.csv') pela tabela “Dados Partidas” (‘dados\_partidas.csv’).

Segue os campos da tabela ‘dados\_partidas’: rodada\_id,partida\_id,partida\_data,clube\_casa\_id,clube\_casa\_nome,clube\_casa\_posicao,aproveitamento\_mandante,clube\_visitante\_id,clube\_visitante\_nome,clube\_visitante\_posicao,aproveitamento\_visitante,placar\_oficial\_mandante,placar\_oficial\_visitante,valida

Com isso, poderíamos usar uma chave estrangeira composta formada por "rodada\_id" e "clube\_id", ambas constantes na tabela-base, "Pontuacao Jogadores", para trazer para cada atleta a cada rodada as informações referentes às partidas das quais participou ou vai participar.

Para tanto, na junção das tabelas para formar o dataframe precisaríamos criar um campo (clube\_mand) que iria analisar uma condição referente aos campos "clube\_casa\_id" e "clube\_visitante\_id", pois eles indicariam se o time na rodada era mandante ou visitante. Ou seja, se naquela rodada o "clube\_id" do atleta estiver no campo "clube\_casa\_id" o campo "club\_mand" deveria ser considerado "VERDADEIRO", caso contrario "FALSO". Nesse caso precisariamos trazer somente os campos relacionados a mandante ou visitante, conforme o caso.

Em resumo precisaríamos fazer o seguinte no código no início dessa conversa:

1) substituir as lógicas relacionadas às tabelas "dados\_partidas\_realizadas" e "dados\_partidas\_atual" para as necessárias para buscar as informações na tabela “dados\_partidas”.

2) fazer a importação no dataframe único pela chave composta constante da tabela "Pontuacao Jogadores" formada pelos campos "clube\_id" e "rodada\_id" das informações da tabela "dados\_partidas", incluindo o campo "clube\_mand" que traria o resultado da verificação se o time do atleta é mandante ou não.

3) alterar o nome dos seguintes campos, pois como dizem respeito ao fato de serem relacionados ao clube mandante ou visitante, deveriam ter o nome da métrica e importar a informação da coluna respectiva à situação do clube: clube\_casa\_nome e clube\_visitante\_nome = clube\_nome; clube\_casa\_posicao e clube\_visitante\_posicao = clube\_posicao; aproveitamento\_mandante e aproveitamento\_visitante = clube\_aproveitamento.

3) no caso do clube ser mandante, trazer os seguintes campos: rodada\_id; partida\_id; partida\_data; clube\_sit; clube\_nome (importada de "clube\_casa\_nome"); clube\_posicao (importada de "clube\_casa\_posicao"); cluibe\_aproveitamento (importada de aproveitamento\_mandante); placar\_oficial\_mandante; placar\_oficial\_visitante; valida.

4) no caso do clube ser visitante, trazer os seguintes campos: rodada\_id; partida\_id; partida\_data; clube\_sit; clube\_nome (importada de clube\_visitante\_nome); clube\_posicao (importada de clube\_visitante\_posicao); aproveitamento\_clube (importada de aproveitamento\_visitante); placar\_oficial\_mandante; placar\_oficial\_visitante; valida.

- Alteração 2: descobri um ponto errado de como você criou o código a partir da sua seguinte afirmação: "O campo 'clube\_mand' é criado com base na comparação entre 'clube\_id' e 'clube\_casa\_id' na tabela 'Pontuacoes Jogadores'.".

O campo 'clube\_mand' deveria ser criado a partir da comparação entre a chave composta da tabela "Pontuacao Jogadores" formada por 'rodada\_id' e 'clube\_id' e os campos 'rodada\_id' e 'clube\_casa\_id' e 'clube\_visitante-id' na tabela 'Dados Partidas'. Pra tanto, o código deveria pegar essas duas chaves em cada linha e comparar na tabela 'Dados Partidas' para identificar a rodada ('rodada\_id') e dentre as partidas daquela rodada, procurar pela chave 'clube\_id' originada na tabela 'Pontuação Jogadores' para ver se ela coincide com o campo 'clube\_casa\_id' ou no campo 'clube\_visitante-id'. Com isso, poderíamos verificar entre as partidas da rodada se o time do atleta era mandante ou visitante e o código incluiria no dataframe no campo 'clube\_mand' a informação de "VERDADEIRO" para o caso do clube ser mandante (clube\_casa\_id) ou "FALSO" para o caso do clube ser visitante (clube\_visitante\_id). Depois de indetificar se o o clube do atleta na rodada era mandante ou visitante, ele deveria importar para o dataframe da tabela "Dados Partidas" os seguintes campos:

a) no caso do clube do atleta ser 'mandante': rodada\_id; partida\_id; partida\_data; clube\_mand ('VERDADEIRO'); clube\_posicao (importada de "clube\_casa\_posicao"); cluibe\_aproveitamento (importada de aproveitamento\_mandante); placar\_oficial\_mandante; placar\_oficial\_visitante; valida.

b) no caso do clube do atleta ser visitante: rodada\_id; partida\_id; partida\_data; clube\_mand ("FALSO"); clube\_posicao (importada de clube\_visitante\_posicao); aproveitamento\_clube (importada de aproveitamento\_visitante); placar\_oficial\_mandante; placar\_oficial\_visitante; valida.