Relatório do Projeto: Estatística Brasileirão

1. Introdução

O projeto tem como objetivo processar, analisar e visualizar dados do Campeonato Brasileiro de Futebol de 2003 a 2022. Para isso, foi implementada uma classe em Python, denominada **BrasileiraoAPI**, que utiliza o MongoDB para armazenamento e consulta dos dados e o Pandas/Matplotlib para análise exploratória e com visualização de gráficos.
Este relatório descreve as funcionalidades implementadas, os métodos da classe e trechos do código que realizam cada operação.

---

2. Estrutura Geral e Conexão com o MongoDB

A classe **BrasileiraoAPI** é responsável por:

- Conectar à instância do MongoDB, utilizando a URI definida (padrão: `mongodb://127.0.0.1:27017/`).
- Definir o banco de dados (`statistics_futebol`) e a coleção principal (`brasileirao`) para armazenar os dados das partidas.

```
class BrasileiraoAPI:
    def __init__(self, mongo_uri="mongodb://127.0.0.1:27017/",
                 db_name="statistics_futebol", collection_name="brasileirao"):
        self.client = MongoClient(mongo_uri)
        self.db = self.client[db_name]
        self.collection = self.db[collection_name]
```

Esta parte inicial estabelece a conexão com o banco de dados, permitindo a execução de operações de inserção, consulta e agregação.

---

3. Importação e Preparação de Dados

3.1 Importar Dados de um JSON para o MongoDB

O método `importar_json_para_mongodb` realiza as seguintes operações:

- Lê um arquivo JSON utilizando o Pandas.
- Converte o DataFrame em uma lista de dicionários.
- Para cada registro, cria um documento estruturado com campos aninhados (por exemplo, para time mandante e visitante, placares, data, hora, etc.).
- Insere os documentos na coleção do MongoDB.

```
def importar_json_para_mongodb(self, json_path):
    df = pd.read_json(json_path)
    registros = df.to_dict(orient='records')
    documentos = []
```

```
    for record in registros:
        documento = {
            "ID": record.get("ID"),
            "rodada": record.get("rodada"),
            "data": record.get("data"),
            "hora": record.get("hora"),
            "homeTeam": {
                "name": record.get("mandante"),
                "formacao": record.get("formacao_mandante"),
                "tecnico": record.get("tecnico_mandante"),
                "estado": record.get("mandante_Estado")
            },
            "awayTeam": {
                "name": record.get("visitante"),
                "formacao": record.get("formacao_visitante"),
                "tecnico": record.get("tecnico_visitante"),
                "estado": record.get("visitante_Estado")
            },
            "score": {
                "fullTime": {
                    "home": int(record.get("mandante_Placar")),
                    "away": int(record.get("visitante_Placar"))
                }
            },
            "vencedor": record.get("vencedor"),
            "arena": record.get("arena")
        }
        documentos.append(documento)
    if documentos:
        self.collection.insert_many(documentos)
        print(f"Inseridos {len(documentos)} documentos no MongoDB.")
    else:
        print("Nenhum documento para inserir.")
```

Com esse método, os dados são importados e estruturados para facilitar futuras consultas e análises.

3.2 Limpeza de Coleção

O método `limpar_colecao` remove todos os documentos da coleção, útil para reinicializar os dados antes de novas inserções.

```
def limpar_colecao(self):
    resultado = self.collection.delete_many({})
    print(f"Removidos {resultado.deleted_count} documentos.")
    return resultado.deleted_count
```

---

4. Consulta e Análise dos Dados

4.1 Consulta Básica e Filtrada

O método `consultar_dados_mongodb` permite realizar consultas na coleção, com ou sem filtro, e retorna os dados em um DataFrame.

```
def consultar_dados_mongodb(self, filtro=None):
    if filtro:
        dados = list(self.collection.find(filtro))
    else:
        dados = list(self.collection.find())
    df = pd.DataFrame(dados)
    return df
```

Além disso, o método `obter_partidas_time` utiliza um filtro para buscar partidas em que um determinado time aparece como mandante ou visitante:

```
def obter_partidas_time(self, nome_time):
    query = {
        "$or": [
            {"homeTeam.name": nome_time},
            {"awayTeam.name": nome_time}
        ]
    }
    return self.consultar_dados_mongodb(filtro=query)
```

4.2 Cálculo do Resultado de uma Partida

O método `calcular_resultado` avalia, para um registro de partida, se o time especificado teve vitória, derrota ou empate, com base no placar e na posição (mandante ou visitante).

```
def calcular_resultado(self, row, time):
    if row['homeTeam']['name'] == time:
        if row['score']['fullTime']['home'] > row['score']['fullTime']['away']:
            return 'Vitória'
        elif row['score']['fullTime']['home'] < row['score']['fullTime']['away']:
            return 'Derrota'
        else:
            return 'Empate'
    else:
        if row['score']['fullTime']['home'] < row['score']['fullTime']['away']:
            return 'Vitória'
        elif row['score']['fullTime']['home'] > row['score']['fullTime']['away']:
            return 'Derrota'
        else:
            return 'Empate'
```

5. Visualização dos Dados

A classe contém diversos métodos que utilizam o Matplotlib para gerar gráficos, dentre os quais:

5.1 Gráfico de Barras

O método `plot_bar` recebe um objeto de resultados (por exemplo, contagem de vitórias, derrotas e empates) e plota um gráfico de barras:

```
def plot_bar(self, resultados, time):
    plt.figure(figsize=(8,6))
    resultados.plot(kind='bar', color=['green', 'red', 'grey'])
    plt.title(f'Resultados do {time} no Brasileirão 2023')
    plt.xlabel('Resultado')
    plt.ylabel('Número de Partidas')
    plt.xticks(rotation=0)
    plt.show()
```

5.2 Scatter Plot do Desempenho na Temporada

O método `plot_desempenho_temporada` gera um scatter plot que mostra gols marcados e sofridos ao longo do tempo, usando o Pandas para converter datas e ordenar os dados:

```
def plot_desempenho_temporada(self, nome_time):
    partidas = self.obter_partidas_time(nome_time)
    partidas['utcDate'] = pd.to_datetime(partidas['utcDate'], errors='coerce')
    partidas = partidas.sort_values('utcDate')
    partidas['gols_marcados'] = partidas.apply(
        lambda row: row['score']['fullTime']['home']
                    if isinstance(row.get('homeTeam'), dict) and
row['homeTeam'].get('name') == nome_time
                    else row['score']['fullTime']['away'],
        axis=1
    )
    partidas['gols_sofridos'] = partidas.apply(
        lambda row: row['score']['fullTime']['away']
                    if isinstance(row.get('homeTeam'), dict) and
row['homeTeam'].get('name') == nome_time
                    else row['score']['fullTime']['home'],
        axis=1
    )
    plt.figure(figsize=(12,6))
    total_gols = partidas['gols_marcados'] + partidas['gols_sofridos']
    plt.scatter(partidas['utcDate'], partidas['gols_marcados'],
                s=total_gols*100, alpha=0.6, label='Gols Marcados')
    plt.scatter(partidas['utcDate'], partidas['gols_sofridos'],
                s=total_gols*100, alpha=0.6, label='Gols Sofridos')
    plt.title(f'Desempenho do {nome_time} na Temporada')
    plt.xlabel('Data')
    plt.ylabel('Número de Gols')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.xticks(rotation=45)
    plt.tight_layout()
```

```
    plt.show()
```

5.3 Outros Gráficos

Outros métodos de visualização implementados incluem:

- `plot_desempenho_time`: Plota o desempenho do time em diferentes temporadas com análise de regressão.
- `plot_media_porcentagem_time`: Exibe gráficos de barras agrupadas com porcentagens de vitórias, empates e derrotas.
- `plot_desempenho_todos_times`: Compara a performance de todos os times ao longo das temporadas.

Cada um desses métodos utiliza funções do Matplotlib para configurar figuras, ajustar eixos, adicionar legendas e exibir os dados de forma visualmente clara.

---

6. Agregações e Estatísticas

A classe implementa métodos de agregação que utilizam o framework de agregação do MongoDB para calcular:

- **Total de gols por time:** `agregacao_total_gols_por_time`
- **Média de gols por time:** `agregacao_media_gols_por_time`
- **Máximo e mínimo de gols:** `agregacao_max_gols_por_time` e `agregacao_min_gols_por_time`

Exemplo de um pipeline de agregação:

```
def agregacao_total_gols_por_time(self):
    pipeline = [
        {"$group": {
            "_id": "$homeTeam.name",
            "total_gols": {"$sum": "$score.fullTime.home"}
        }}
    ]
    return list(self.collection.aggregate(pipeline))
```

Esses métodos facilitam a obtenção de estatísticas agregadas diretamente a partir dos dados armazenados no MongoDB.

---

7. Manipulação de Tabelas e Odds

7.1 Montagem e Exportação de Tabelas

O método `montar_tabelas` agrupa dados por temporada e monta uma estrutura com estatísticas (jogos, vitórias, empates, derrotas, gols marcados/sofridos, pontos). Em seguida, o método `exportar_tabelas_json` salva essas tabelas em um arquivo JSON.

```
def montar_tabelas(self, start_year=2003, end_year=2022):
    df = self.consultar_dados_mongodb()
```

```
    df["data_dt"] = pd.to_datetime(df["data"], format="%d/%m/%Y",
errors="coerce")
    df = df.dropna(subset=["data_dt"])
    df = df[(df["data_dt"].dt.year >= start_year) & (df["data_dt"].dt.year <=
end_year)]

    tabelas = {}
    for season in range(start_year, end_year + 1):
        tabelas[str(season)] = {}

    def atualizar_time(tabela, time, gols_feitos, gols_sofridos, resultado):
        if time not in tabela:
            tabela[time] = {
                "jogos": 0,
                "vitorias": 0,
                "empates": 0,
                "derrotas": 0,
                "gols_marcados": 0,
                "gols_sofridos": 0,
                "pontos": 0
            }
        tabela[time]["jogos"] += 1
        tabela[time]["gols_marcados"] += gols_feitos
        tabela[time]["gols_sofridos"] += gols_sofridos
        if resultado == "Vitória":
            tabela[time]["vitorias"] += 1
            tabela[time]["pontos"] += 3
        elif resultado == "Empate":
            tabela[time]["empates"] += 1
            tabela[time]["pontos"] += 1
        elif resultado == "Derrota":
            tabela[time]["derrotas"] += 1

    # Iteração sobre as partidas para preencher as tabelas por temporada
    for _, row in df.iterrows():
        season = str(row["data_dt"].year)
        home_team = row["homeTeam"]["name"] if isinstance(row["homeTeam"], dict)
else row["homeTeam"]
        away_team = row["awayTeam"]["name"] if isinstance(row["awayTeam"], dict)
else row["awayTeam"]
        score_home = row["score"]["fullTime"]["home"]
        score_away = row["score"]["fullTime"]["away"]

        if score_home > score_away:
            resultado_home = "Vitória"
            resultado_away = "Derrota"
        elif score_home < score_away:
            resultado_home = "Derrota"
            resultado_away = "Vitória"
        else:
            resultado_home = "Empate"
            resultado_away = "Empate"

        atualizar_time(tabelas[season], home_team, score_home, score_away,
resultado_home)
```

```
        atualizar_time(tabelas[season], away_team, score_away, score_home,
resultado_away)

    return json.dumps(tabelas, indent=4, ensure_ascii=False)
```

7.2 Odds Agregados

O método `gerar_odds_todos_times` agrega as odds (médias de vitórias, empates e derrotas) para cada time, para cada temporada, salvando os resultados em uma nova coleção e exportando para um arquivo JSON.

---

8. Inserção, Edição e Verificação de Documentos

8.1 Inserção Condicional

O método `verificar_e_inserir_documentos` compara os documentos a serem inseridos com os existentes (através de um critério de busca) para evitar duplicidade. Caso não existam, insere os novos documentos e informa quantos foram adicionados.

8.2 Edição de Documentos

O método `editar_documentos` realiza atualizações em campos específicos, tanto na coleção de partidas (*brasileirao*) quanto na coleção de odds (*odds_times_aggregados*). São realizadas operações de `$set` e `$inc` para modificar valores como horário, placar, rodada, odds, entre outros.

8.3 Verificação de Alterações

O método `verificar_alteracoes_brasileirao` percorre a coleção para exibir as alterações feitas – por exemplo, jogos com horário alterado, placares modificados, partidas movidas de rodada e datas atualizadas.

---

9. Funções de Busca e Agregação Avançada

Além das funções básicas de consulta, foram implementados métodos que utilizam diferentes operadores e funções de agregação:

- Buscas com operadores lógicos:
  - `buscar_partidas_por_confronto`: Filtra partidas entre dois times, utilizando regex e operadores `$and` e `$or`.
  - `buscar_partidas_por_time_or` e `buscar_partidas_por_rodadas`: Utilizam operadores `$or` e `$in` para filtrar dados.
- Agregações com funções do MongoDB:
  - Métodos como `agregacao_total_gols_por_time`, `agregacao_media_gols_por_time`, `agregacao_max_gols_por_time` e `agregacao_min_gols_por_time` exemplificam o uso de funções agregadas (SUM, AVG, MAX, MIN).

---

10. Backup dos Dados

O método `fazer_backup` gera um backup completo do banco de dados. Ele cria uma estrutura de diretórios baseada na data e hora atual, itera sobre todas as coleções, remove o campo `_id` de cada documento e salva os dados em arquivos JSON.

```
def fazer_backup(self, db_name="statistics_futebol", repo_name="brazileirao-data-analysis"):
    # ... (código que cria diretórios e salva os backups)
```

---

11. Conclusão

O código implementado na classe **BrasileiraoAPI** oferece um conjunto robusto de funcionalidades para:

- Importar e estruturar dados do Campeonato Brasileiro.
- Realizar consultas e agregações diversas no MongoDB.
- Executar análises exploratórias e visualizações gráficas com Pandas e Matplotlib.
- Gerenciar a integridade dos dados por meio de inserção condicional, edição e backup.