No código apresentado, dividir os dados em dois arquivos: dados1_sync.js e dados2_sync.js.

Função 1 - dados_sync.js

Primeiramente, importei o módulo fs, responsável por manipular arquivos em formato JSON.

```
async function lendo_Database_1() {
  try {
    let data1 = await fs.promises.readFile(
      "/home/milene/Desafio-Tecnico-Monks-Media/databases/broken_database_1.json",
      "utf8"
    );
    return JSON.parse(data1);
  } catch (error) {
    console.error("Erro ao carregar JSON:", error);
    return null;
  }
}
```

Optei por utilizar funções assíncronas, pois o carregamento e a leitura do arquivo podem demandar algum tempo. Para evitar possíveis erros de leitura, adotei esse modelo. Implementei uma estrutura try/catch com o objetivo de capturar erros que possam ocorrer durante o processo de carregamento do arquivo (readFile). Se o carregamento for bem-sucedido (try), a função realiza um parse para retornar um JSON válido; caso contrário, uma exceção (erro) é lançada.

```
async function corrigir_json() {
  let jsonData1 = await lendo_Database_1();

  if (jsonData1) {
    let json_nome_veiculo_corrigido_1 = jsonData1.map((item) => {
      if (item.nome && typeof item.nome === "string") {
        item.nome = item.nome.replace(/æ/g, "a").replace(/ø/g, "o");
      }
      if (item.vendas && typeof item.vendas === "string") {
        item.vendas = Number(item.vendas);
      }
      return item;
    });
    await fs.promises.writeFile(
      "arquivos corrigidos/broken_database_1_corrigido.json",
      JSON.stringify(json_nome_veiculo_corrigido_1),
      "utf-8"
    );
  }
}
corrigir_json();
```

Na segunda função, corrigir_json, recebo o arquivo JSON carregado (chamando lendo_Database2). Utilizo uma estrutura condicional (if) para verificar se o arquivo existe. Se existir, a função corrige o campo "marca" em cada item do JSON, substituindo caracteres específicos por "a" e "o". Isso é feito utilizando o método map, que cria um novo array aplicando uma determinada função a cada elemento do array original. Vale ressaltar que essa operação não altera o array original, mas retorna um novo array contendo os resultados das operações aplicadas a cada elemento (item).Em seguida, os dados corrigidos são escritos em um novo arquivo chamado "broken_database_2_corrigido.json". Por fim, chamo a função para garantir a sua execução correta.No script 2 foi aplicado a mesma lógica.

Implementei uma função(que não foi solicitada no desafio), mas basicamente eu recebo o json com as devidas correções, faço um if(se o arquivo existe) ele vai receber esse arquivo, com várias promessas simultâneas, depois disso eu faço um map para percorrer cada item no array, depois de percorrer ele vai usar o método DadosVendas.create para criar e salvar os dados dentro do banco de dados e retorna um array de promessas que são resolvidas quando todas as operações assíncronas concluem. Esse array é atribuído à variável dadosRecebidos.

```
;(async () => {
  try {
    const dadosJson = await corrigirJson()
    if (dadosJson) {
      let dadosRecebidos = await Promise.all(
        dadosJson.map(async (item) => {
          const criarDados = await DadosVendasVeiculos.create({
            data: item.data,
            vendas: item.vendas,
            valor_do_veiculo: item.valor_do_veiculo,
            nome: item.nome,
          })
          return criarDados
        })
      )
      return dadosRecebidos
    }
  } catch (error) {
    console.error("Erro geral:", error)
  }
})()
```

Para essa função, utilizei o Sequelize para criar as Models(representa em código de uma tabela no banco de dados) e as Migrations(define como as tabelas devem ser criadas), assim como o banco de dados MySQL para armazenar as informações. Como boa prática, optei por utilizar o MySQL dentro de um container Docker.