Traffic Collisions Analytics

Heitor Sabino
Gustavo Soares
Enzo Velo
Ranulfo Neto
Eduardo Oliveira

Objetivo do Projeto

- Analisar colisões de trânsito para identificar padrões e tendências.
- Aplicar técnicas de Big Data para extrair informações relevantes de grandes volumes de dados brutos.
- Utilizar inferência e reconhecimento de padrões para gerar insights.

Dados Utilizados

- Fonte: Dados Abertos da PRF.
- Período: de 2007 a 2025.
- Dois conjuntos principais:
  - Acidentes de trânsito (30 colunas, 2.150.000 registros).
  - Pessoas envolvidas (35 colunas, 4.600.000 registros).
- Total aproximado: 6,8 milhões de registros.

Principais Variáveis

Base de Pessoas:

- `data_inversa`, `idade`, `sexo`, `tipo_veiculo`, `estado_fisico`
- Indicadores de gravidade: `feridos_leves`, `feridos_graves`, `mortos`

Base de Acidentes:

- `data_inversa`, `dia_semana`, `horario`, `municipio`, `causa_acidente`
- Classificação: `tipo_acidente`, `classificacao_acidente`, `fase_dia`
- Localização: `latitude`, `longitude`, `uf`, `km`, `br`

Plano: Arquitetura e Tecnologias

- **Docker e Docker Compose:** Para criar um ambiente de desenvolvimento reprodutível e isolado, garantindo que todas as dependências funcionem de forma consistente.

Plano: Arquitetura e Tecnologias

- **Docker e Docker Compose:** Para criar um ambiente de desenvolvimento reprodutível e isolado, garantindo que todas as dependências funcionem de forma consistente.
- **Cluster Apache Spark 4.0.0:**
  - Configuração *Standalone* com 1 Master e 1 Worker para processamento distribuído dos dados.
  - Imagem base `bitnami/spark:4.0.0`.

Plano: Arquitetura e Tecnologias

- **Docker e Docker Compose:** Para criar um ambiente de desenvolvimento reprodutível e isolado, garantindo que todas as dependências funcionem de forma consistente.
- **Cluster Apache Spark 4.0.0:**
  - Configuração *Standalone* com 1 Master e 1 Worker para processamento distribuído dos dados.
  - Imagem base `bitnami/spark:4.0.0`.
- **JupyterLab com PySpark:**
  - Interface interativa para desenvolvimento e execução das análises (atuando como Spark Driver).
  - Imagem Docker customizada via `Dockerfile` para garantir que a versão do `pyspark` (4.0.0) seja idêntica à do cluster, evitando erros de compatibilidade.

Plano: Arquitetura e Tecnologias

- **Docker e Docker Compose:** Para criar um ambiente de desenvolvimento reprodutível e isolado, garantindo que todas as dependências funcionem de forma consistente.
- **Cluster Apache Spark 4.0.0:**
  - Configuração *Standalone* com 1 Master e 1 Worker para processamento distribuído dos dados.
  - Imagem base `bitnami/spark:4.0.0`.
- **JupyterLab com PySpark:**
  - Interface interativa para desenvolvimento e execução das análises (atuando como Spark Driver).
  - Imagem Docker customizada via `Dockerfile` para garantir que a versão do `pyspark` (4.0.0) seja idêntica à do cluster, evitando erros de compatibilidade.
- **Python 3.11 e Bibliotecas:**
  - `Pandas` e `Matplotlib` para agregação final e visualização dos resultados.

Plano: Execução e Fluxo de Trabalho

1. **Inicialização do Ambiente:**
   Com um único comando, todos os serviços são construídos e iniciados:

```
docker-compose up --build -d
```

Plano: Execução e Fluxo de Trabalho

1. **Inicialização do Ambiente:**
   Com um único comando, todos os serviços são construídos e iniciados:

```
docker-compose up --build -d
```

2. **Acesso e Análise:**
   O JupyterLab fica disponível no navegador para execução interativa dos notebooks.

Plano: Execução e Fluxo de Trabalho

1. **Inicialização do Ambiente:**
   Com um único comando, todos os serviços são construídos e iniciados:

```
docker-compose up --build -d
```

2. **Acesso e Análise:**
   O JupyterLab fica disponível no navegador para execução interativa dos notebooks.
3. **Conexão com o Cluster:**
   A sessão PySpark é configurada para se conectar ao Master, que distribui o trabalho para os Workers:

```
spark://spark-master:7077
```

Plano: Execução e Fluxo de Trabalho

1. **Inicialização do Ambiente:**
   Com um único comando, todos os serviços são construídos e iniciados:

```
docker-compose up --build -d
```

2. **Acesso e Análise:**
   O JupyterLab fica disponível no navegador para execução interativa dos notebooks.
3. **Conexão com o Cluster:**
   A sessão PySpark é configurada para se conectar ao Master, que distribui o trabalho para os Workers:

```
spark://spark-master:7077
```

4. **Coleta e Visualização:**
   Após o processamento distribuído, os resultados agregados são trazidos para o Jupyter e plotados com Matplotlib.

Plano: Pré-processamento e Limpeza (Parte 1)

A qualidade dos dados brutos apresentou desafios que foram tratados com as seguintes técnicas em PySpark:

- **Tratamento de Dados Inválidos (Idade):**
  - A coluna `idade` continha valores não numéricos (ex: 'NA').
  - **Solução:** Foi utilizada uma conversão segura para tipo numérico. Valores inválidos se tornaram `NULL` e foram posteriormente classificados como "Desconhecida", evitando quebras na execução.

Plano: Pré-processamento e Limpeza (Parte 1)

A qualidade dos dados brutos apresentou desafios que foram tratados com as seguintes técnicas em PySpark:

- **Tratamento de Dados Inválidos (Idade):**
  - A coluna `idade` continha valores não numéricos (ex: 'NA').
  - **Solução:** Foi utilizada uma conversão segura para tipo numérico. Valores inválidos se tornaram `NULL` e foram posteriormente classificados como "Desconhecida", evitando quebras na execução.
- **Tratamento de Dados Inválidos (Gênero):**
  - A coluna `sexo` continha valores não numéricos (ex: 'NA', 'M', 'F', 'Não Informado') e numéricos.
  - **Solução:** Foi utilizada uma conversão segura para padronizar a categorização. Valores inválidos foram classificados como "Desconhecida", evitando quebras na execução.

Plano: Pré-processamento e Limpeza (Parte 2)

Continuando o processo, os dados foram refinados com as seguintes técnicas:

- **Padronização de Categorias (Gênero):**
  - A coluna `sexo` possuía múltiplas representações para a mesma categoria (ex: "M", "Masculino", "Numeros Aleatórios").
  - **Solução:** A função `when` foi usada para unificar os valores em categorias consistentes ("Masculino", "Feminino", "Não Informado").

Plano: Pré-processamento e Limpeza (Parte 2)

Continuando o processo, os dados foram refinados com as seguintes técnicas:

- **Padronização de Categorias (Gênero):**
  - A coluna `sexo` possuía múltiplas representações para a mesma categoria (ex: "M", "Masculino", "Numeros Aleatórios").
  - **Solução:** A função `when` foi usada para unificar os valores em categorias consistentes ("Masculino", "Feminino", "Não Informado").
- **Extração de Componentes Temporais:**
  - Para análises temporais, foram extraídos componentes específicos das colunas de data/hora usando funções nativas do Spark como `hour()` e `year()`.

Plano: Ingestão e Integração de Dados

A fase inicial do projeto constitui a ingestão e a subsequente integração dos conjuntos de dados heterogêneos da PRF.

1. **Ingestão dos Dados Primários:**
   Uma classe customizada, `DataLoader`, foi implementada para realizar a leitura programática dos múltiplos arquivos CSV de cada diretório. Este processo agrega os registros anuais em dois DataFrames Spark distintos: um para ocorrências e outro para pessoas.

Plano: Ingestão e Integração de Dados

A fase inicial do projeto constitui a ingestão e a subsequente integração dos conjuntos de dados heterogêneos da PRF.

1. **Ingestão dos Dados Primários:**
   Uma classe customizada, `DataLoader`, foi implementada para realizar a leitura programática dos múltiplos arquivos CSV de cada diretório. Este processo agrega os registros anuais em dois DataFrames Spark distintos: um para ocorrências e outro para pessoas.
2. **Unificação dos DataFrames:**
   Os DataFrames de ocorrências e pessoas são unificados por meio de uma operação de junção interna (`inner join`), utilizando a coluna `id` como chave de junção. Essa operação correlaciona as informações de cada acidente com os respectivos indivíduos envolvidos.

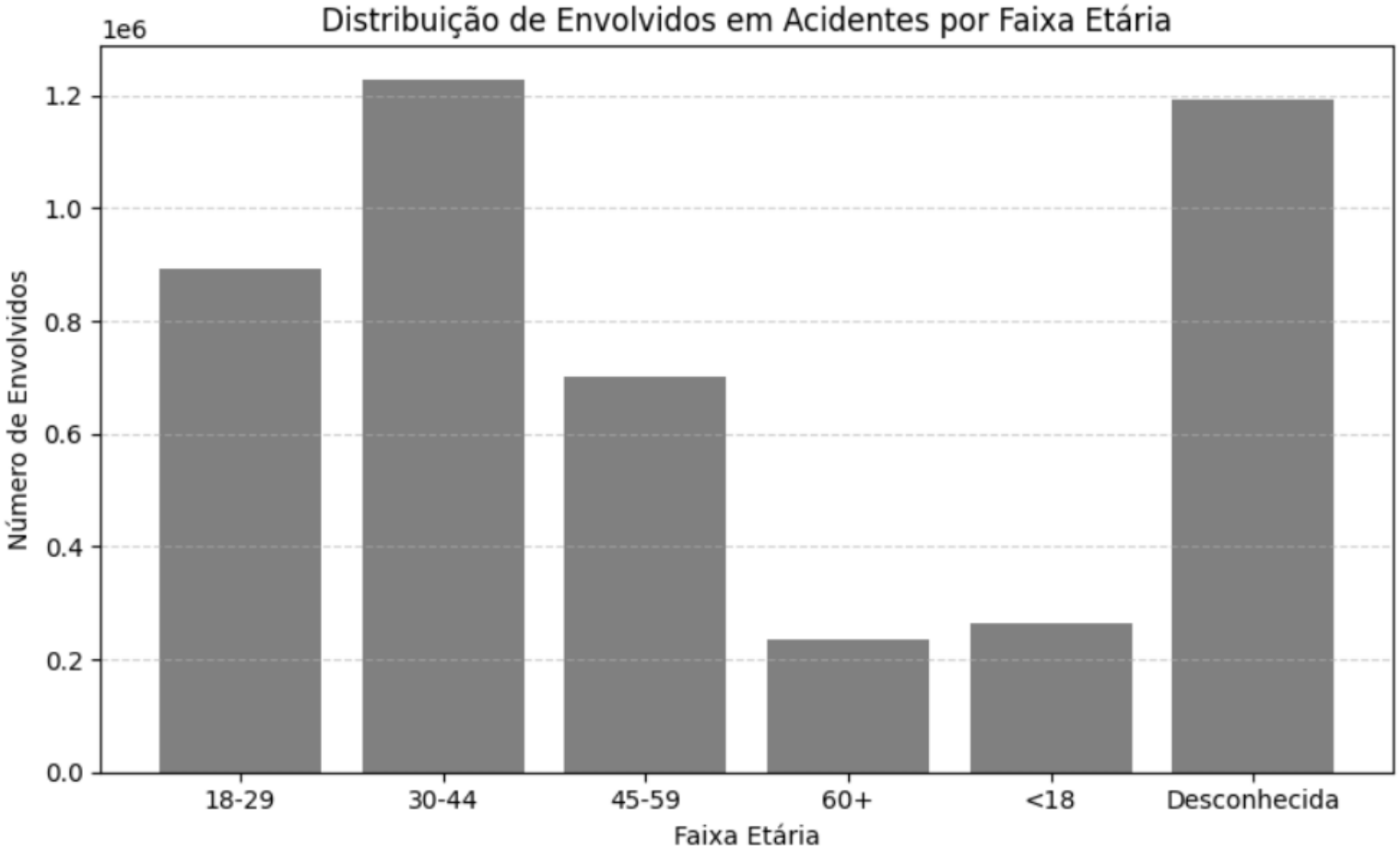
Plano: Ingestão e Integração de Dados

A fase inicial do projeto constitui a ingestão e a subsequente integração dos conjuntos de dados heterogêneos da PRF.

1. **Ingestão dos Dados Primários:**
   Uma classe customizada, `DataLoader`, foi implementada para realizar a leitura programática dos múltiplos arquivos CSV de cada diretório. Este processo agrega os registros anuais em dois DataFrames Spark distintos: um para ocorrências e outro para pessoas.
2. **Unificação dos DataFrames:**
   Os DataFrames de ocorrências e pessoas são unificados por meio de uma operação de junção interna (`inner join`), utilizando a coluna `id` como chave de junção. Essa operação correlaciona as informações de cada acidente com os respectivos indivíduos envolvidos.
3. **Geração do DataFrame Consolidado:**
   O resultado é um DataFrame único e enriquecido, `df_joined`, que constitui a base analítica para todas as investigações subsequentes do projeto.

Análise Demográfica: Distribuição por Faixa Etária

Objetivo: Identificar a distribuição dos indivíduos envolvidos em acidentes por faixas etárias, para entender quais grupos são mais vulneráveis ou frequentemente registrados.



Análise Demográfica: Observações sobre Faixa Etária

Principais Observações:

- Jovens e Adultos: A faixa de **18 a 44 anos** concentra a maior parte dos envolvidos, coincidindo com a população economicamente ativa e com maior mobilidade.

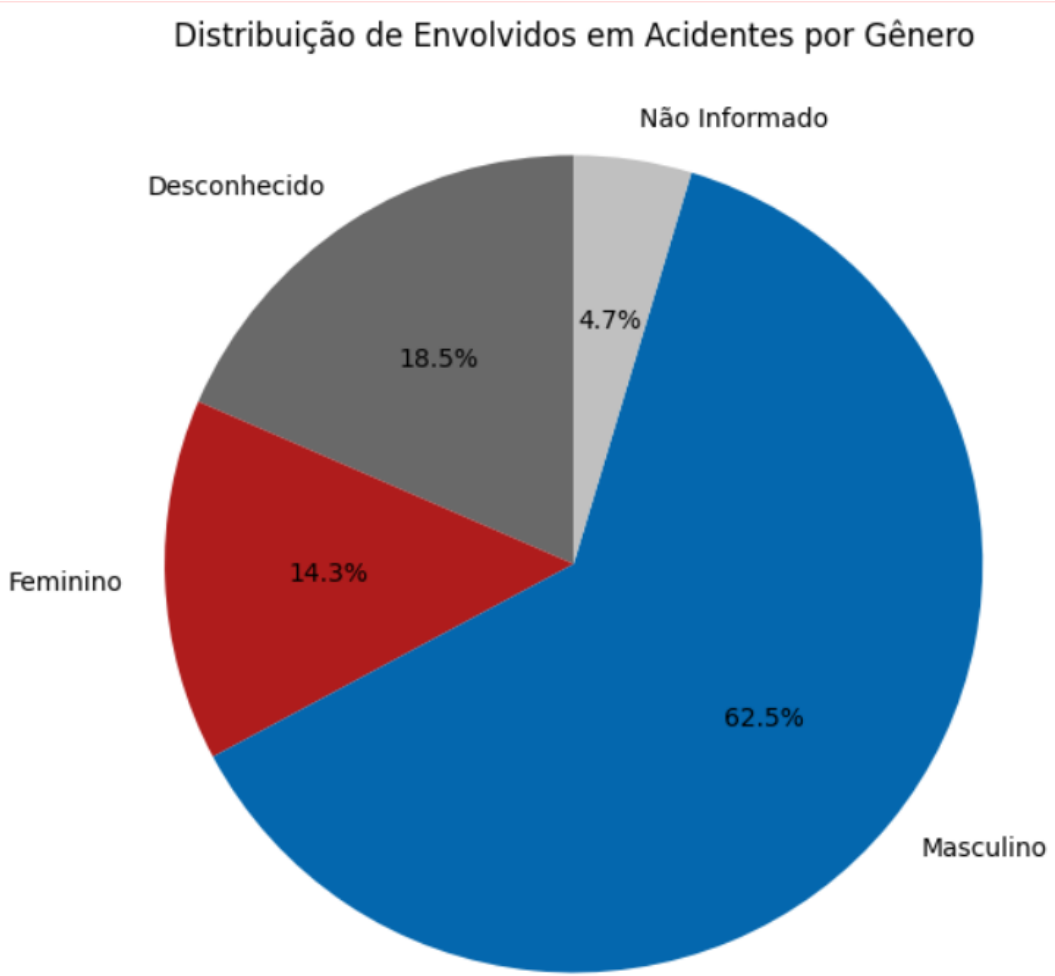
Análise Demográfica: Observações sobre Faixa Etária

Principais Observações:

- Jovens e Adultos: A faixa de **18 a 44 anos** concentra a maior parte dos envolvidos, coincidindo com a população economicamente ativa e com maior mobilidade.
- Dados Ausentes: Uma parcela significativa é classificada como "Desconhecida", indicando desafios na coleta de dados de idade no momento da ocorrência.

Análise Demográfica: Distribuição por Gênero

Objetivo: Analisar a proporção de envolvidos em acidentes por gênero, para verificar se existem disparidades significativas.



Análise Demográfica: Observações sobre Gênero

Principais Observações:

- Disparidade de Gênero: Há uma predominância expressiva de envolvidos do gênero **masculino** nos registros de acidentes em rodovias federais.

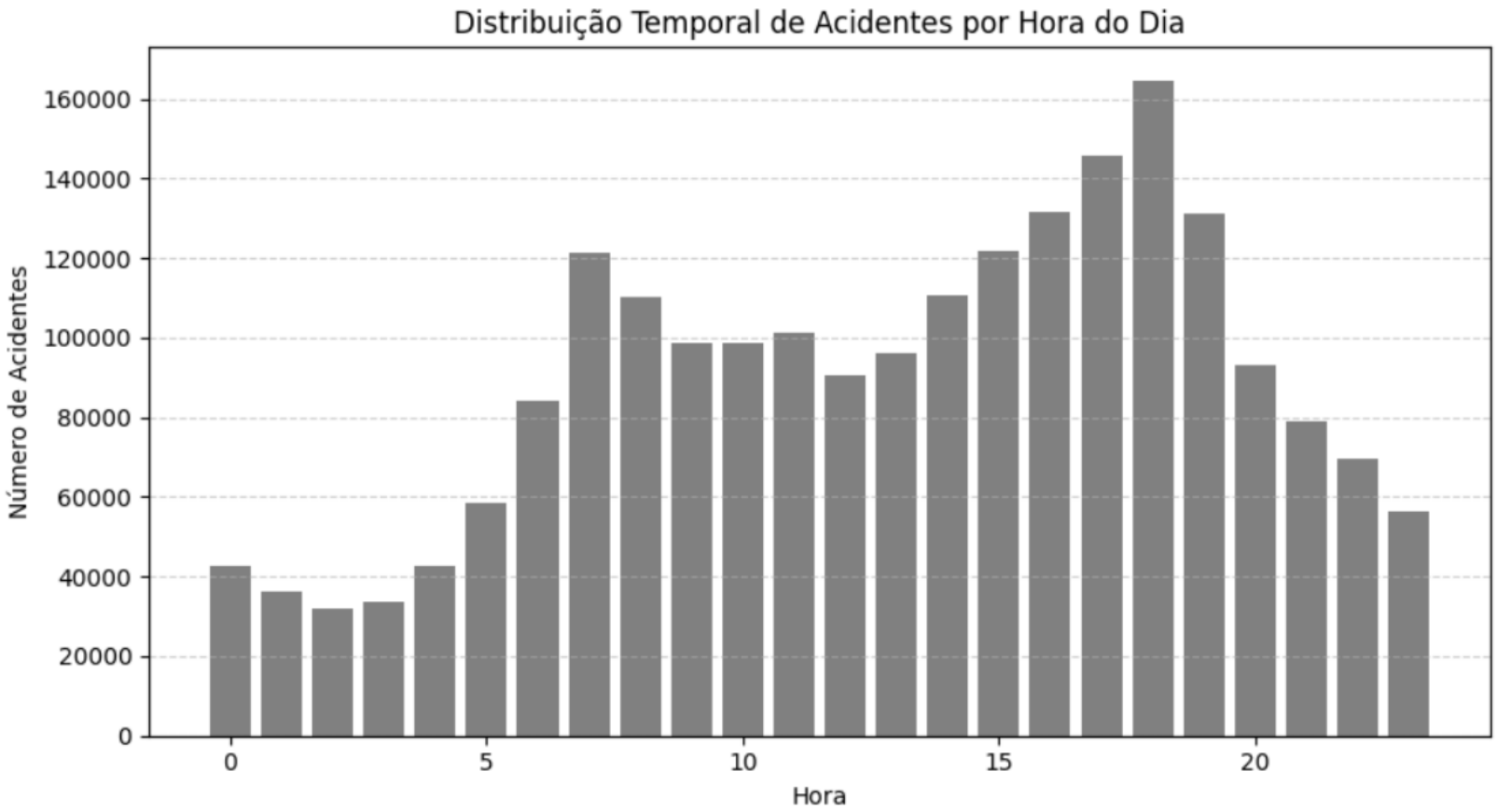
Análise Demográfica: Observações sobre Gênero

Principais Observações:

- Disparidade de Gênero: Há uma predominância expressiva de envolvidos do gênero **masculino** nos registros de acidentes em rodovias federais.
- Qualidade do Dado: A categoria "Não Informado" possui uma porcentagem relevante, o que sugere a necessidade de políticas para aprimorar o preenchimento dos boletins de ocorrência.

Análise Temporal: Distribuição por Hora do Dia

Objetivo: Identificar os horários de pico de acidentes ao longo do dia, visando orientar operações de fiscalização e campanhas de conscientização.



Análise Temporal: Observações sobre Hora do Dia

Principais Observações:

- Picos de Ocorrências: Os acidentes se concentram no início do dia (7h) e da noite (entre **17h e 19h**), período de maior fluxo de ida e retorno do trabalho.

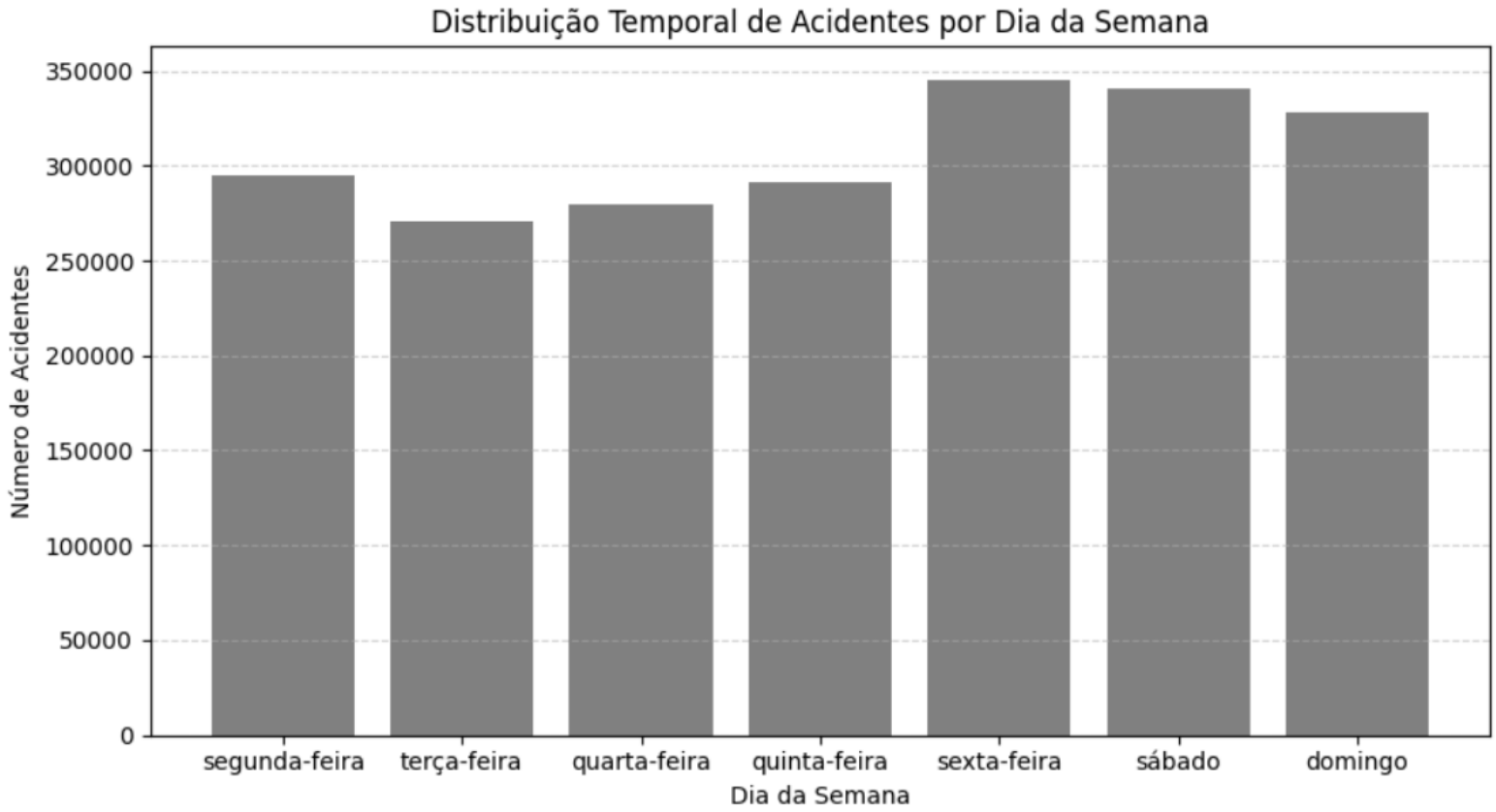
Análise Temporal: Observações sobre Hora do Dia

Principais Observações:

- Picos de Ocorrências: Os acidentes se concentram no início do dia (7h) e da noite (entre **17h e 19h**), período de maior fluxo de ida e retorno do trabalho.
- Período da Madrugada: As horas da madrugada (entre 2h e 5h) apresentam a menor frequência de acidentes, coincidindo com o menor volume de tráfego.

Análise Temporal: Distribuição por Dia da Semana

Objetivo: Comparar a frequência de acidentes entre os dias da semana para identificar padrões relacionados a dias úteis e finais de semana.



Análise Temporal: Observações sobre Dia da Semana

Principais Observações:

- Finais de Semana: **Sexta-feira, Sábado e Domingo** são os dias com maior número de registros, possivelmente associados ao aumento do fluxo de veículos para lazer e viagens.

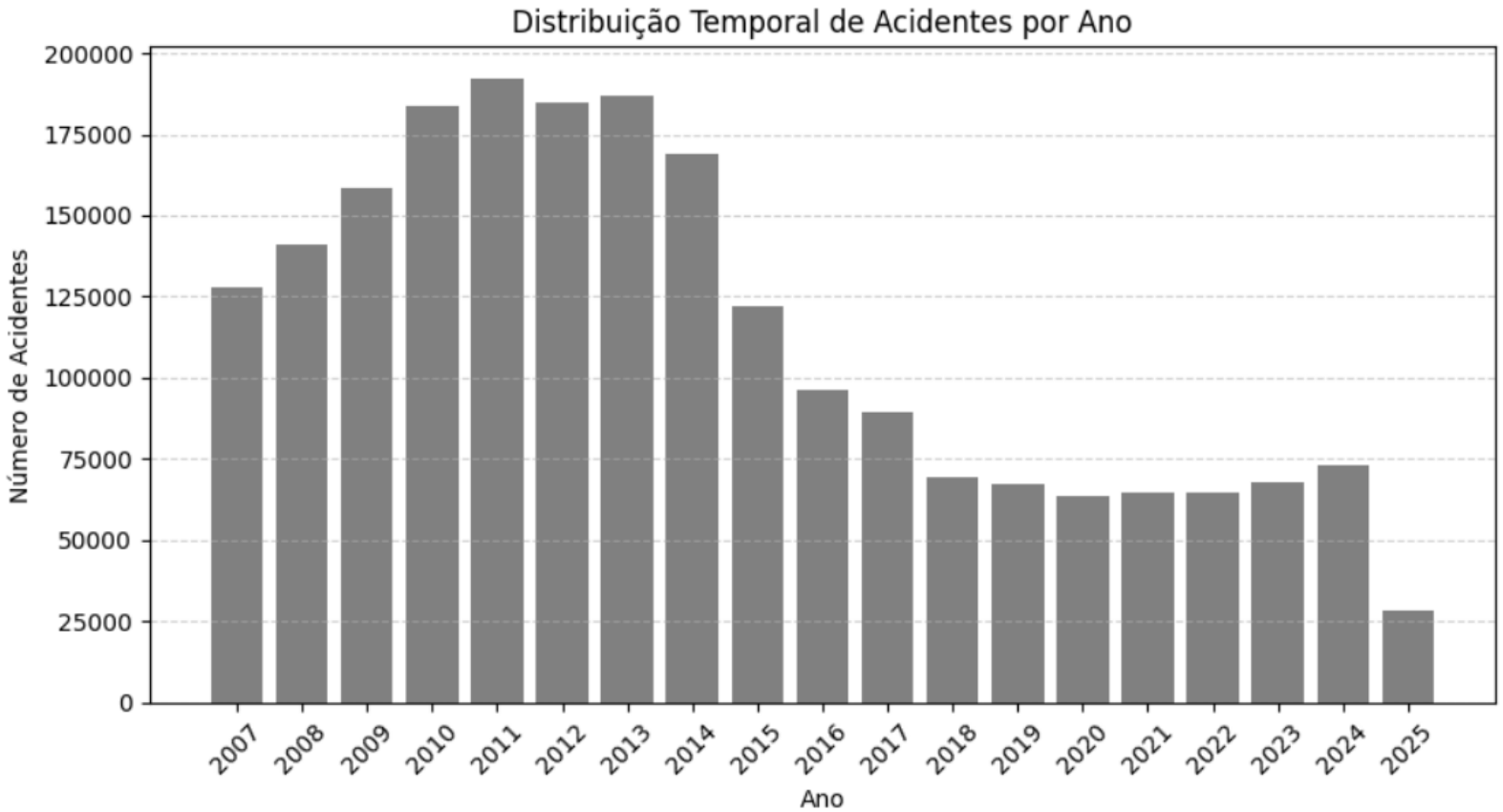
Análise Temporal: Observações sobre Dia da Semana

Principais Observações:

- Finais de Semana: **Sexta-feira, Sábado e Domingo** são os dias com maior número de registros, possivelmente associados ao aumento do fluxo de veículos para lazer e viagens.
- Meio da Semana: Terça e quarta-feira apresentam os menores índices de acidentes, representando um período de tráfego mais rotineiro e estável.

Análise Temporal: Evolução Anual dos Acidentes

Objetivo: Observar a tendência de longo prazo no número de acidentes registrados, permitindo avaliar o impacto de políticas públicas e outros fatores macro.



Análise Temporal: Observações sobre a Evolução Anual

Principais Observações:

- Tendência de Queda: Observa-se uma tendência geral de **redução no número de acidentes** a partir de meados da década de 2010, o que pode indicar melhorias em segurança viária e fiscalização.

Análise Temporal: Observações sobre a Evolução Anual

Principais Observações:

- Tendência de Queda: Observa-se uma tendência geral de **redução no número de acidentes** a partir de meados da década de 2010, o que pode indicar melhorias em segurança viária e fiscalização.
- Anos Atípicos: O ano de **2020** apresenta uma queda acentuada, possivelmente um reflexo direto das restrições de mobilidade impostas durante a pandemia de COVID-19.

Métricas: Comparativo de Formatos

Para avaliar a eficiência da arquitetura e do formato dos dados, medimos o tempo de execução (em segundos) de um pipeline completo, comparando o uso de arquivos **CSV** contra o formato colunar otimizado **Parquet**.

- **Métrica Coletada:** Tempo de execução em segundos, medido com `time.perf_counter`.

Métricas: Comparativo de Formatos

Para avaliar a eficiência da arquitetura e do formato dos dados, medimos o tempo de execução (em segundos) de um pipeline completo, comparando o uso de arquivos **CSV** contra o formato colunar otimizado **Parquet**.

- **Métrica Coletada:** Tempo de execução em segundos, medido com `time.perf_counter`.
- **Etapas Medidas:**
  1. **Carregamento:** Tempo para ler a base de dados completa do disco para um DataFrame Spark.
  2. **Processamento:** Tempo para executar a análise de faixa etária sobre o DataFrame carregado.
  3. **Escrita:** Tempo para salvar o resultado processado de volta no disco.

Métricas: Comparativo de Formatos

Para avaliar a eficiência da arquitetura e do formato dos dados, medimos o tempo de execução (em segundos) de um pipeline completo, comparando o uso de arquivos **CSV** contra o formato colunar otimizado **Parquet**.

- **Métrica Coletada:** Tempo de execução em segundos, medido com `time.perf_counter`.
- **Etapas Medidas:**
  1. **Carregamento:** Tempo para ler a base de dados completa do disco para um DataFrame Spark.
  2. **Processamento:** Tempo para executar a análise de faixa etária sobre o DataFrame carregado.
  3. **Escrita:** Tempo para salvar o resultado processado de volta no disco.
- **Objetivo da Medição:** Demonstrar quantitativamente os ganhos de performance ao utilizar formatos de arquivo otimizados para Big Data (Parquet) em vez de formatos de texto simples (CSV).

Resultados de Performance: Docker Compose

A tabela a seguir apresenta a média de 5 execuções do pipeline no ambiente single-node com `docker-compose`.

Tabela: Tempo Médio de Execução (em segundos) - Docker Compose.

Etapa do Pipeline	**CSV (s)**	**Parquet (s)**	**Ganho (Parquet)**
1. Carregamento de Dados	5.345s	0.048s	**111.3x**
2. Processamento	0.038s	0.031s	**1.2x**
3. Escrita de Resultados	4.679s	0.679s	**6.9x**

Conclusão

- **Conclusão Estratégica:**
  A adoção do Parquet como formato padrão é crucial em um pipeline de dados, pois acelera drasticamente o ciclo de análise e viabiliza um trabalho mais interativo e eficiente.

Introdução à Simulação de Ambiente Produtivo

Para avaliar o desempenho da aplicação em um cenário mais próximo de um ambiente de produção, utilizamos o **Docker Swarm**, a ferramenta nativa do Docker para orquestração de contêineres.

- **O que é?** O Swarm permite gerenciar um cluster de múltiplas máquinas Docker como se fossem uma só, atuando como um "maestro"que distribui os serviços (contêineres).

Introdução à Simulação de Ambiente Produtivo

Para avaliar o desempenho da aplicação em um cenário mais próximo de um ambiente de produção, utilizamos o **Docker Swarm**, a ferramenta nativa do Docker para orquestração de contêineres.

- **O que é?** O Swarm permite gerenciar um cluster de múltiplas máquinas Docker como se fossem uma só, atuando como um "maestro"que distribui os serviços (contêineres).
- **Principais Vantagens:**
  - **Escalabilidade:** Permite aumentar ou diminuir o número de contêineres de um serviço com um único comando.
  - **Alta Disponibilidade:** Se uma máquina falha, o Swarm pode reiniciar os contêineres em outra máquina saudável.
  - **Gerenciamento Simplificado:** Implantação de aplicações complexas através de um arquivo de "stack".

Ambiente de Teste em Docker Swarm

A análise de performance foi conduzida em um cluster Swarm com a seguinte configuração, implantada via `docker stack deploy`:

- 1x Contêiner **Spark Master**: Responsável por coordenar o cluster.
- 2x Contêineres **Spark Worker**: Réplicas responsáveis pela execução das tarefas de processamento.
- 1x Contêiner **JupyterLab**: Interface interativa para submeter os trabalhos ao cluster (Spark Driver).

Viés do Ambiente Single-Node

É crucial notar que, para este teste, todos os serviços (manager e workers) foram executados na mesma máquina física. Embora o Swarm gerencie os **2 workers** como entidades separadas, eles competem pelos mesmos recursos de CPU e memória. Portanto, os resultados de performance do processamento distribuído devem ser interpretados como uma simulação da arquitetura, e não como uma medida de ganho de distribuição real.

Resultados de Performance no Ambiente Swarm (Média de 5 Execuções) para 2 Workers

A tabela a seguir apresenta o tempo médio de execução (em segundos) para cada etapa do pipeline, comparando os formatos CSV e Parquet.

Tabela: Tempo Médio de Execução (em segundos) por Etapa e Formato.

Etapa do Pipeline	**CSV (s)**	**Parquet (s)**	**Ganho (Parquet)**
1. Carregamento de Dados	3.127s	0.071s	**44x mais rápido**
2. Processamento	0.043s	0.031s	**1.4x mais rápido**
3. Escrita do Resultado	2.708s	0.730s	**3.7x mais rápido**

Análise Comparativa: Docker Compose vs. Docker Swarm

Comparativo de performance entre a execução **single-node** (`Compose`) e **orquestrada** (`Swarm`), ambas na mesma máquina física.

Tabela: Comparativo de Tempo Médio de Execução (em segundos).

Etapa (Formato)	**Compose (s)**	**Swarm (s)**
Carregamento de Dados (CSV)	5.345s	3.127s
Carregamento de Dados (Parquet)	0.048s	0.071s
Processamento (CSV)	0.038s	0.043s
Processamento (Parquet)	0.031s	0.031s
Escrita do Resultado (CSV)	4.679s	2.708s
Escrita do Resultado (Parquet)	0.679s	0.730s

Análise Comparativa: Interpretação dos Resultados

Principais Observações:

- **Overhead vs. Otimização:**
  Para operações muito rápidas (ex: carga de Parquet), o `Compose` foi ligeiramente mais rápido. Isso sugere que a camada de gerenciamento e rede do Swarm introduz um pequeno *overhead* (sobrecarga) perceptível em tarefas de milissegundos.

Análise Comparativa: Interpretação dos Resultados

Principais Observações:

- **Overhead vs. Otimização:**
  Para operações muito rápidas (ex: carga de Parquet), o `Compose` foi ligeiramente mais rápido. Isso sugere que a camada de gerenciamento e rede do Swarm introduz um pequeno *overhead* (sobrecarga) perceptível em tarefas de milissegundos.
- **Gerenciamento de I/O:**
  Curiosamente, para a tarefa mais lenta (carregamento de CSV), o ambiente `Swarm` foi significativamente mais rápido. Isso pode indicar que o scheduler do Swarm otimiza a alocação de recursos de I/O de forma mais eficiente.

Análise Comparativa: Interpretação dos Resultados

Principais Observações:

- **Overhead vs. Otimização:**
  Para operações muito rápidas (ex: carga de Parquet), o `Compose` foi ligeiramente mais rápido. Isso sugere que a camada de gerenciamento e rede do Swarm introduz um pequeno *overhead* (sobrecarga) perceptível em tarefas de milissegundos.
- **Gerenciamento de I/O:**
  Curiosamente, para a tarefa mais lenta (carregamento de CSV), o ambiente `Swarm` foi significativamente mais rápido. Isso pode indicar que o scheduler do Swarm otimiza a alocação de recursos de I/O de forma mais eficiente.
- **Conclusão:**
  O teste valida a viabilidade da arquitetura Swarm. A verdadeira vantagem de performance do Swarm se manifestaria em um cluster com múltiplos nós, onde o paralelismo real superaria qualquer *overhead* inicial.

Resultados de Performance: Swarm com 4 Workers

A tabela a seguir apresenta a média de 5 execuções do pipeline no ambiente orquestrado com **4 nós workers**.

Tabela: Tempo Médio de Execução (em segundos) - Swarm com 4 Workers.

Etapa do Pipeline	**CSV (s)**	**Parquet (s)**	**Ganho (Parquet)**
1. Carregamento de Dados	2.364s	0.079s	**29.9x mais rápido**
2. Processamento	0.038s	0.029s	**1.3x mais rápido**
3. Escrita do Resultado	1.890s	0.877s	**2.2x mais rápido**