IESB

Big Data e Inteligência Analítica

Projeto Integrador em Big Data e Inteligência Analítica

Lucas Siqueira Rodrigues

Brasília - DF
Junho de 2025

1 Obtenção da base de dados

1.1 Introdução

O trabalho tem como objetivo explorar os dados públicos disponibilizados pela Câmara dos Deputados para analisar e compreender os gastos realizados pelos parlamentares. Para isso, os dados serão obtidos por meio do portal dos dados abertos da câmara[1], armazenados em um banco de dados relacional na nuvem e, posteriormente, analisados.

1.2 Motivação

A transparência pública é essencial para garantir a confiança da população nas instituições governamentais. Esse projeto busca realizar um ciclo completo de extração, transformação, armazenamento e análise dos dados de gastos públicos. Além disso, ao construir dashboards, espera-se fornecer ferramentas que possam auxiliar na identificação de possíveis irregularidades e na fiscalização das despesas parlamentares.

1.3 Script e Banco de Dados

Para o ano de 2022, a obtenção dos dados por meio da API[1] retornou apenas 31 registros.

```python
import io
import json
import zipfile

import httpx
from tqdm import tqdm

class CamaraAPI:
  def __init__(self) -> None:
    self.base_url = "https://dadosabertos.camara.leg.br/api/v2"

  def request(self, endpoint: str) -> dict:
    response = httpx.get(f"{self.base_url}/{endpoint}")
    return response.json()

  def get_deputados(self) -> dict:
    return self.request("deputados").get("dados", {})

  def get_despesas(self, id_: int, year: int = 2022) -> dict:
    return self.request(f"deputados/{id_}/despesas?ano={year}")

despesas = []

api = CamaraAPI()

deputados = api.get_deputados()
for deputado in tqdm(deputados):
  id_ = deputado["id"]
  despesas_deputado = api.get_despesas(id_=id_, year=2022)
  despesas.extend(despesas_deputado["dados"])

print(len(despesas))  # output: 31
```

Por isso, apenas para esse ano a coleta de dados foi por meio de um arquivo no formato JSON, que também é fornecido no portal de dados abertos da câmara na aba “Arquivos”.

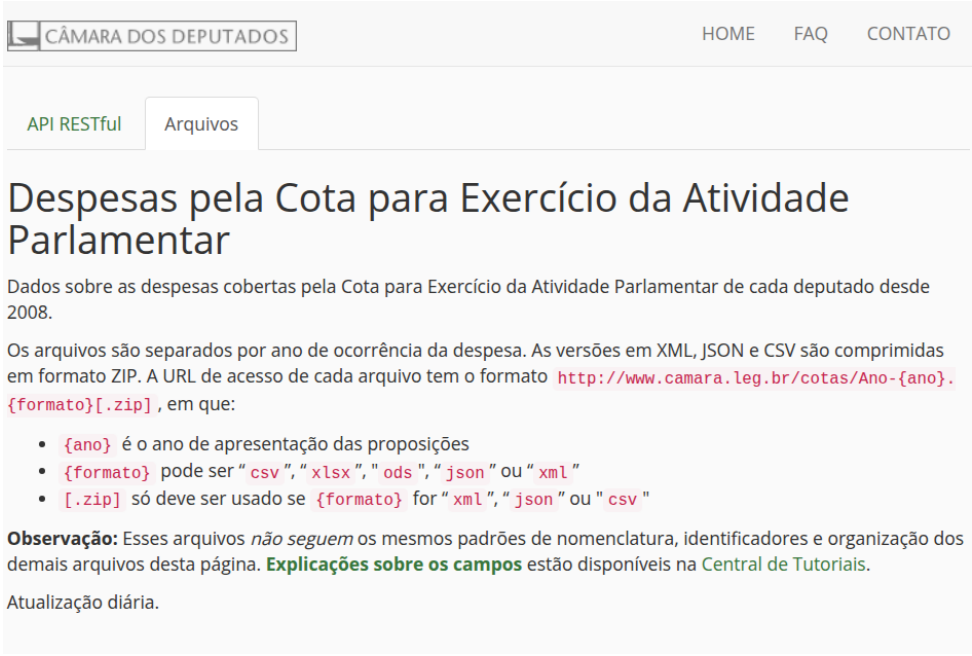

Figura 1: Coleta de dados por meio da aba de arquivos.

Para os anos de 2023 e 2024 a obtenção dos dados foi realizada por meio da API[1] da Câmara dos Deputados, explorando dois principais endpoints:

- **/deputados**: Retorna informações gerais sobre os parlamentares, como seus nomes, partidos, estados e e-mails.
- **/deputados/{id}/despesas**: Fornece detalhes sobre as despesas realizadas pelos parlamentares, incluindo valores, fornecedores, tipos de despesa e datas, para se ter uma consulta mais completa, devemos adicionar filtros como mês, ano, legislatura, CNPJ ou CPF de um fornecedor, no nosso caso adicionaremos filtros para os anos, ficando com a url: /deputados/{id}/despesas?ano={ano}.

Para organizar os dados de forma eficiente e integrar os dados obtidos por meio da API e por meio do JSON, foi criado um modelo de banco de dados relacional com tabelas normalizadas para representar as informações de deputados, despesas e fornecedores[2].

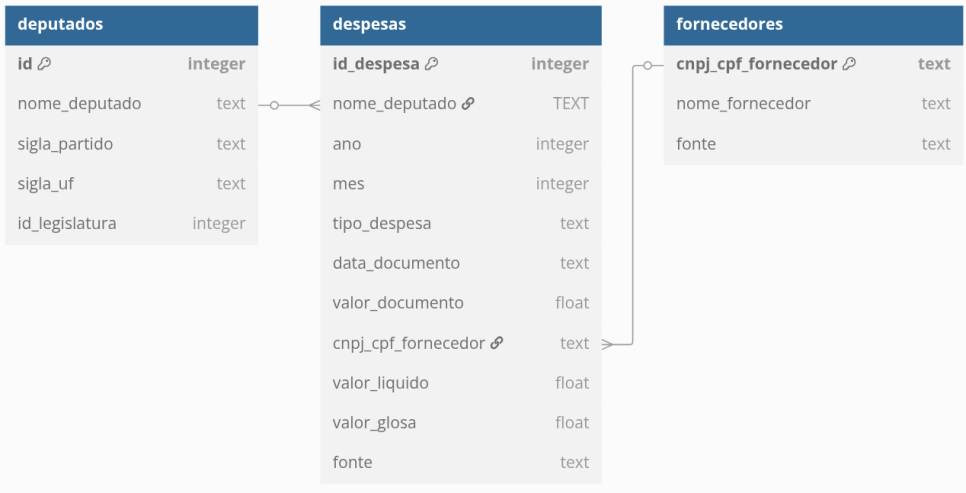

Figura 2: Diagrama relacional.

Os dados são obtidos e unificados por meio da classe DataService.

```python
import io
import json
import re
import zipfile

import httpx
from tqdm import tqdm

from data_ingestion.services.log_service import logger


class DataService:
  def __init__(self) -> None:
  self.api_base_url = "https://dadosabertos.camara.leg.br/api/v2"

  def get_deputados(self) -> list:
    response = httpx.get(f"{self.api_base_url}/deputados")
    data = response.json().get("dados")
    selected_data = [
      {
        "id": i["id"],
        "nome": i["nome"],
        "sigla_partido": i["siglaPartido"],
        "id_legislatura": i["idLegislatura"],
        "sigla_uf": i["siglaUf"],
      }
      for i in data
      ]

    return selected_data

  def get_data_from_api(self, anos: list[int]) -> tuple[list[dict], list
                                    [dict], list[dict]]:
    deputados = self.get_deputados()
    despesas = []
    fornecedores = []
    for ano in anos:
```

```
37        logger.info(f"Getting data from API for year {ano}")
38        for deputado in tqdm(deputados):
39          response = httpx.get(f"{self.api_base_url}/deputados/{deputado['
                                    id']}/despesas?ano={ano}")
40          despesas_deputado = response.json().get("dados")
41
42          for item in despesas_deputado:
43            despesas.append({
44              "nome_deputado": deputado.get("nome"),
45              "ano": item.get("ano"),
46              "mes": item.get("mes"),
47              "tipo_despesa": item.get("tipoDespesa"),
48              "data_documento": item.get("dataDocumento"),
49              "valor_documento": item.get("valorDocumento"),
50              "cnpj_cpf_fornecedor": re.sub(r"[^0-9]", "", item.get("
                                    cnpjCpfFornecedor")),
51              "valor_liquido": item.get("valorLiquido"),
52              "valor_glosa": item.get("valorGlosa"),
53              "fonte": "api",
54            })
55
56            fornecedores.append({
57              "nome_fornecedor": item.get("nomeFornecedor"),
58              "cnpj_cpf_fornecedor": re.sub(r"[^0-9]", "", item.get("
                                    cnpjCpfFornecedor")),
59              "fonte": "api",
60            })
61
62      logger.info(f"Total number of despesas: {len(despesas)}")
63
64      return deputados, despesas, fornecedores
65
66    @staticmethod
67    def get_data_from_url(url: str) -> tuple[list[dict], list[dict], list[
                                    dict]]:
68      logger.info(f"Getting data from url: {url}")
69      response = httpx.get(url=url)
70      zip_content = response.content
71
72      with zipfile.ZipFile(io.BytesIO(zip_content)) as zip_file:
73        with zip_file.open(zip_file.namelist()[0]) as json_file:
74          json_content = json_file.read().decode("utf-8")
75          json_data = json.loads(json_content).get("dados")
76
77      deputados = []
78      despesas = []
79      fornecedores = []
80
81      for item in json_data:
82        deputados.append({
83          "id": item["numeroDeputadoID"],
84          "nome": item["nomeParlamentar"],
85          "sigla_partido": item["siglaPartido"],
86          "id_legislatura": item["codigoLegislatura"],
87          "sigla_uf": item["siglaUF"],
88        })
89
90        despesas.append({
```

```
        "nome_deputado": item.get("nomeParlamentar"),
        "ano": item.get("ano"),
        "mes": item.get("mes"),
        "tipo_despesa": item.get("descricao"),
        "data_documento": item.get("dataEmissao"),
        "valor_documento": item.get("valorDocumento"),
        "cnpj_cpf_fornecedor": re.sub(r"[^0-9]", "", item.get("cnpjCPF")
                                      ),
        "valor_liquido": item.get("valorLiquido"),
        "valor_glosa": item.get("valorGlosa"),
        "fonte": "url",
      })

      fornecedores.append({
        "nome_fornecedor": item.get("fornecedor"),
        "cnpj_cpf_fornecedor": re.sub(r"[^0-9]", "", item.get("cnpjCPF")
                                      ),
        "fonte": "url",
      })

    logger.info(f"Total number of despesas: {len(despesas)}")

    return deputados, despesas, fornecedores
```

A persistência dos dados na base de dados é feita por meio da classe DBService. Ela permite a ingestão dos dados em uma tabela SQLite e MySQL, a tabela do SQLite foi criada e usada apenas para o proprósito de desenvolvimento.

```
from sqlalchemy import Column, Float, Integer, String, create_engine,
                                    insert
from sqlalchemy.orm import declarative_base, sessionmaker

from data_ingestion.services.log_service import logger

Base = declarative_base()


class Despesas(Base):
  tablename__ = "despesas"
  id = Column(Integer, primary_key=True, autoincrement=True)
  nome_deputado = Column(String(100))
  ano = Column(Integer)
  mes = Column(Integer)
  tipo_despesa = Column(String(100))
  data_documento = Column(String(100))
  valor_documento = Column(Float)
  cnpj_cpf_fornecedor = Column(String(100))
  valor_liquido = Column(Float)
  valor_glosa = Column(Float)
  fonte = Column(String(100))


class Fornecedores(Base):
  tablename__ = "fornecedores"
  id = Column(Integer, primary_key=True, autoincrement=True)
  nome_fornecedor = Column(String(100))
  cnpj_cpf_fornecedor = Column(String(100), unique=True)
  fonte = Column(String(100))
```

```
30
31
32  class Deputados(Base):
33    tablename__ = "deputados"
34    id = Column(Integer, primary_key=True)
35    nome = Column(String(100), unique=True)
36    sigla_partido = Column(String(100))
37    id_legislatura = Column(Integer)
38    sigla_uf = Column(String(100))
39
40
41  class DBService:
42    def __init__(
43      self,
44      local: bool = False,
45      user: str | None = None,
46      password: str | None = None,
47      host: str | None = None,
48      port: int | None = None,
49      dbname: str | None = None,
50    ) -> None:
51      if local:
52        self.engine = create_engine("sqlite:///database.db")
53        self.dialect = "sqlite"
54      else:
55        self.engine = create_engine(f"mysql+pymysql://{user}:{password}@{
                                      host}:{port}/{dbname}", pool_recycle
                                      =3600)
56        self.dialect = "mysql"
57
58      self.Session = sessionmaker(bind=self.engine)
59
60    def insert_data(self, deputados: list, despesas: list, fornecedores:
                                      list) -> None:
61      Base.metadata.create_all(self.engine)
62      session = self.Session()
63
64      try:
65        logger.info("Inserindo deputados")
66        if deputados:
67          stmt = insert(Deputados)
68          if self.dialect == "sqlite":
69            stmt = stmt.prefix_with("OR IGNORE")
70          elif self.dialect == "mysql":
71            stmt = stmt.prefix_with("IGNORE")
72          session.execute(stmt, deputados)
73
74        logger.info("Inserindo fornecedores")
75        if fornecedores:
76          stmt = insert(Fornecedores)
77          if self.dialect == "sqlite":
78            stmt = stmt.prefix_with("OR IGNORE")
79          elif self.dialect == "mysql":
80            stmt = stmt.prefix_with("IGNORE")
81          session.execute(stmt, fornecedores)
82
83        logger.info("Inserindo despesas")
84        if despesas:
```

```
85          stmt = insert(Despesas)
86          if self.dialect == "sqlite":
87            stmt = stmt.prefix_with("OR IGNORE")
88          elif self.dialect == "mysql":
89            stmt = stmt.prefix_with("IGNORE")
90          session.execute(stmt, despesas)
91
92        session.commit()
93      finally:
94        session.close()
```

E por fim, temos a função main(), que realiza todo o ciclo de extração, transformação e carregamento dos dados.

```
1  import os
2
3  from dotenv import load_dotenv
4
5  from data_ingestion.services.data_service import DataService
6  from data_ingestion.services.db_service import DBService
7
8  _ = load_dotenv()
9
10
11 def main() -> None:
12   url = "https://www.camara.leg.br/cotas/Ano-2022.json.zip"
13   anos = [2023, 2024]
14
15   data_service = DataService()
16   api_deputados, api_despesas, api_fornecedores = data_service.
                                        get_data_from_api(anos=anos)
17   url_deputados, url_despesas, url_fornecedores = data_service.
                                        get_data_from_url(url=url)
18
19   deputados = [*api_deputados, *url_deputados]
20   despesas = [*api_despesas, *url_despesas]
21   fornecedores = [*api_fornecedores, *url_fornecedores]
22
23   db_service = DBService(
24     user=os.environ["DB_USER"],
25     password=os.environ["DB_PASSWORD"],
26     host=os.environ["DB_HOST"],
27     port=int(os.environ["DB_PORT"]),
28     dbname=os.environ["DB_NAME"],
29   )
30   db_service.insert_data(deputados=deputados, despesas=despesas,
                                        fornecedores=fornecedores)
31
32
33 if __name__ == "__main__":
34   main()
```

O programa, ao ser executado, faz a extração, transformação e carga de 223.982 registros de despesas dos anos de 2022, 2023 e 2024.

Logo após serem obtidos, os dados são inseridos em um banco de dados MySQL, que foi criado usando o serviço Amazon RDS (Relational Database Service).

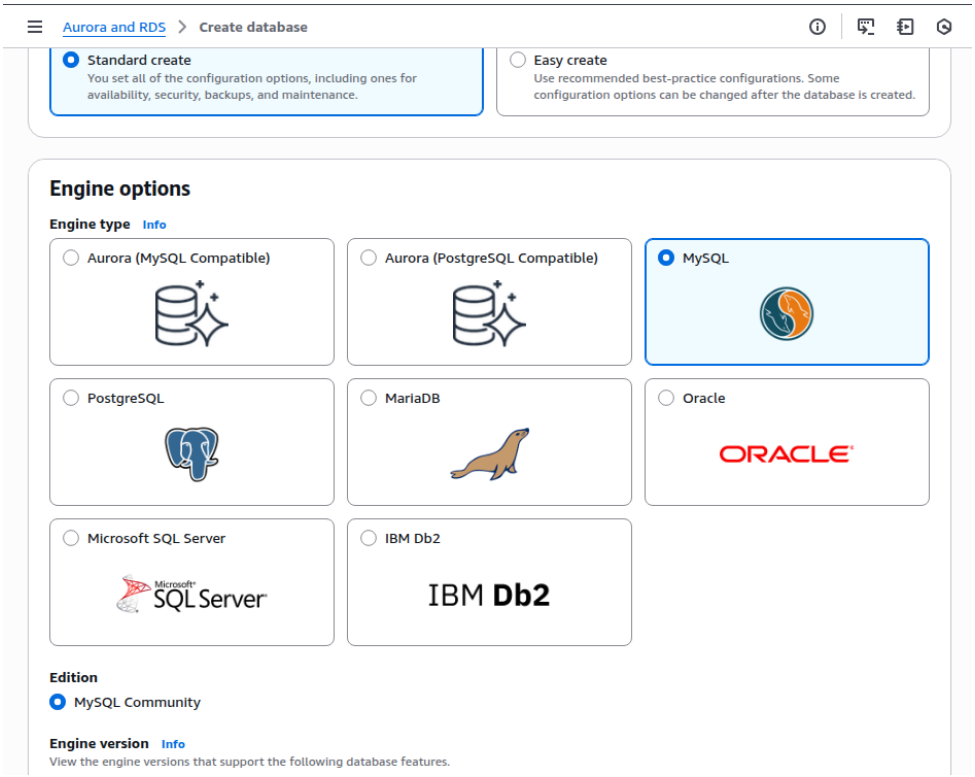

Figura 3: Criação do MySQL na AWS.

Após a base de dados ser criada, podemos nos conectar a ela usando o DBeaver[3].

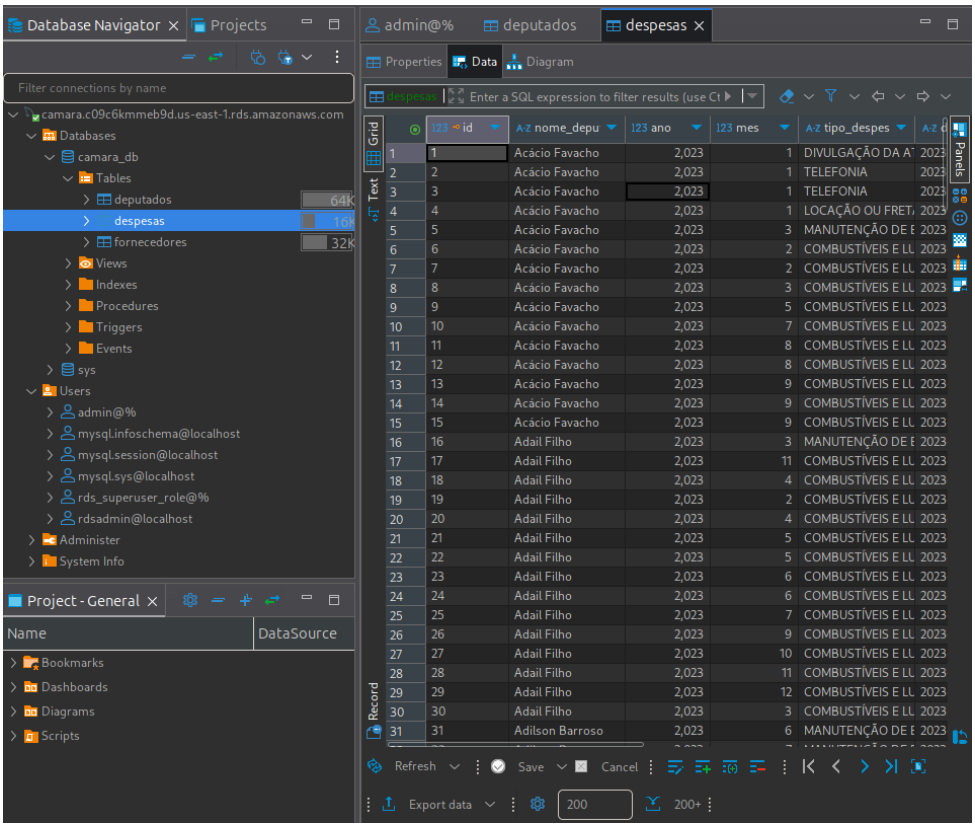

Figura 4: Base de dados conectada no DBeaver.

Todo o código relacionado ao projeto está no Github[4].

1.4 Considerações finais

A combinação de tecnologias como Python, AWS e MySQL foi fundamental para realizar as etapas de extração, transformação e carregamento dos dados. A API da Câmara revelou-se limitada em relação à quantidade de dados retornados para o ano de 2022, mas a extração dos dados do JSON da aba Arquivos permitiu superar essa restrição e criar uma base robusta com uma quantidade considerável de dados.

2 Relatório Analítico

2.1 Introdução

A análise de dados públicos desempenha um papel crucial na promoção da transparência governamental e no combate a irregularidades. Utilizando o Streamlit para a criação de dashboards, é possível transformar grandes volumes de dados em informações compreensíveis e acessíveis para a população e órgãos de fiscalização.

2.2 Demonstração

O Streamlit foi integrado ao banco de dados MySQL, permitindo consultas em tempo real e construção de dashboards dinâmicos.

Na nossa base de dados foram construídas 3 tabelas, e para facilitar o processo de construção de gráficos uma query foi feita realizando o join das tabelas.

```
SELECT
 despesas.nome_deputado,
 despesas.ano,
 despesas.mes,
 despesas.tipo_despesa,
 despesas.valor_documento,
 despesas.cnpj_cpf_fornecedor,
 deputados.sigla_partido,
 deputados.id_legislatura,
 deputados.sigla_uf,
 fornecedores.nome_fornecedor
FROM
 despesas
LEFT JOIN
 deputados on despesas.nome_deputado = deputados.nome
LEFT JOIN
 fornecedores ON despesas.cnpj_cpf_fornecedor = fornecedores.cnpj_cpf_fornecedor;
```

A partir dos dados recuperados da base de dados criamos um dataframe do Pandas[5], nele podemos aplicar filtros e realizar agrupamentos nos dados para extrair informações e criar gráficos.

```
1 grouped_df = df[["tipo_despesa", "valor_documento"]].groupby("
                                    tipo_despesa").sum()
2 st.bar_chart(grouped_df, horizontal=True, x_label="Valor total em R$")
```

Figura 5: Gráfico mostrando a quantidade de gastos por tipo de despesa.

2.3 Relatórios e tabelas

Aqui podemos analisar o gastos dos anos de 2022, 2023 e 2024 de forma combinada ou isolada. Temos que o total de gastos são:

Ano	Total de Gastos	Quantidade de Deputados
Combinados	R$ 249 mi	795
2022	R$ 221 mi	572
2023	R$ 14 mi	494
2024	R$ 14 mi	496

Tabela 1: Resumo de gastos.

Para os anos combinados, temos que DIVULGAÇÃO DA ATIVIDADE PARLAMENTAR foi a categoria de gastos com o maior valor, seguido de PASSAGEM AÉREA - SIGEPA e MANUTENÇÃO DE ESCRITÓRIO DE APOIO À ATIVIDADE PARLAMENTAR.

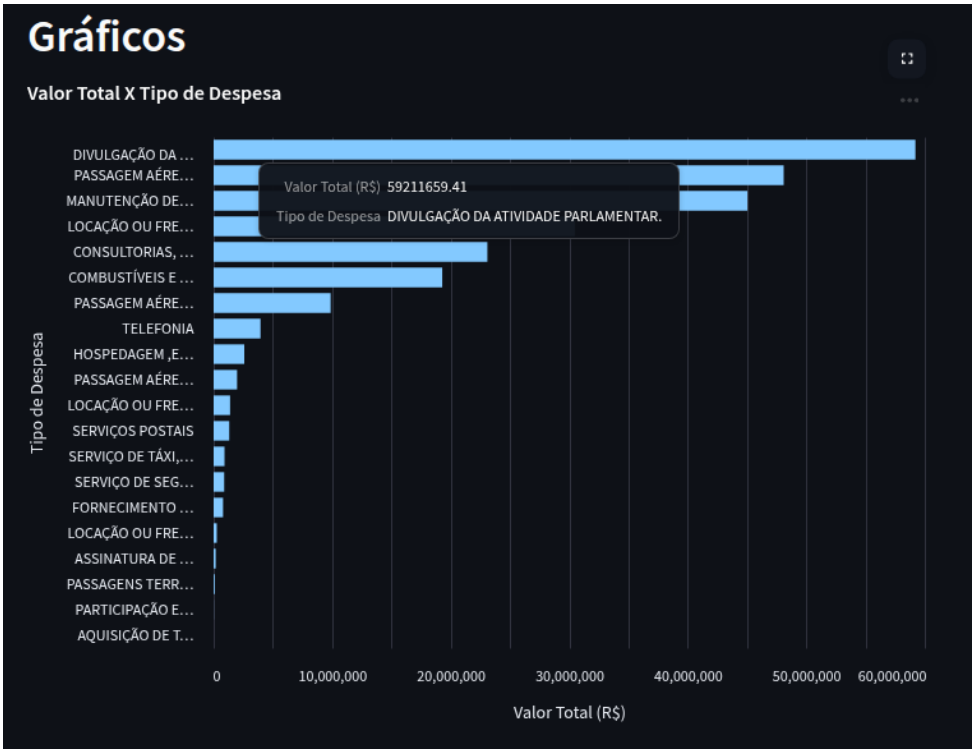

Figura 6: Valor dos gastos por tipo de despesa.

Giacobo foi o parlamentar com a maior quantidade de gastos, totalizando R$ 1.289.809,76 em gastos, seguido por Silas Câmara com R$ 1.177.321,09 e Gustinho Ribeiro com R$ 1.061.509,11.

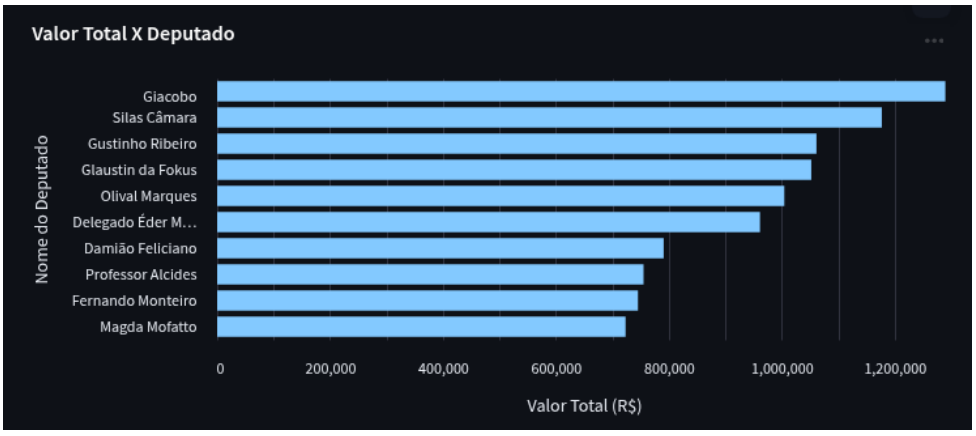

Figura 7: Valor dos gastos por deputado.

Com relação aos fornecedores, as 3 maiores são empresas de companhias aéreas. A TAM foi disparadamente a maior, com gastos totalizando R$ 48 milhões, seguida por Latam Linhas Aéreas com R$ 5 milhões e Gol Linhas Aéreas com quase R$ 4 milhões.

Usando o streamlit também podemos criar gráficos dinâmicos para analisar os dados separados por categorias, assim podemos selecionar a variável de interesse e a função de agregação, para assim poder analisar o total de gastos, média, valor máximo e muito mais.

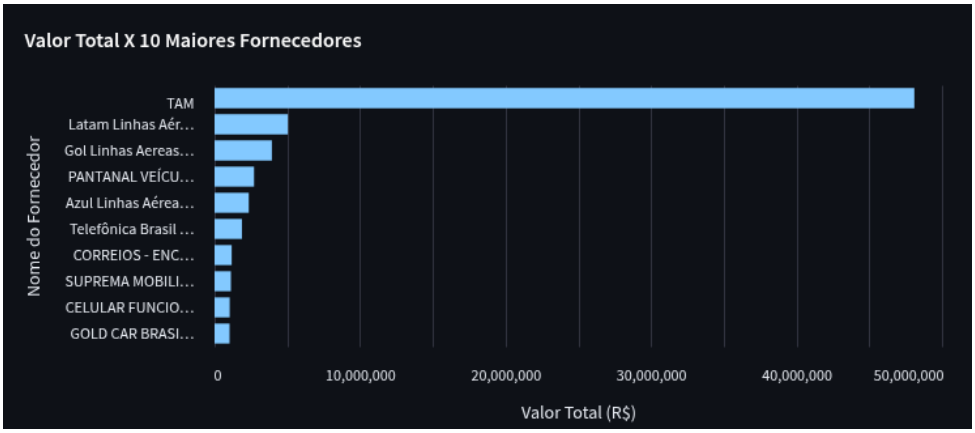

Figura 8: Valor dos gastos por fornecedor.

Figura 9: Gráfico dinâmico.

Ao selecionar a categoria sigla_partido, podemos ver que os partidos com o maior número de gastos foi o PL com R$ 34 milhões, seguido pelo PP com R$ 28 milhões e o PT com R$ 27 milhões.

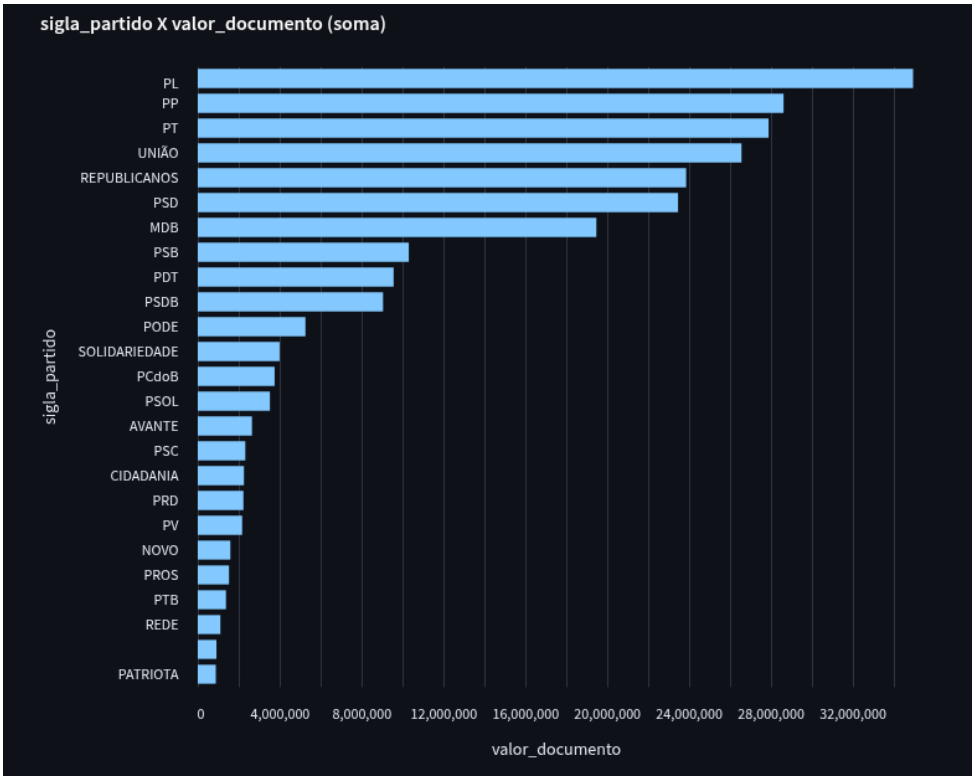

Figura 10: Valor dos gastos por partido.

Podemos filtrar os gastos por UF, calculando o valor total dos gastos nesse estado.

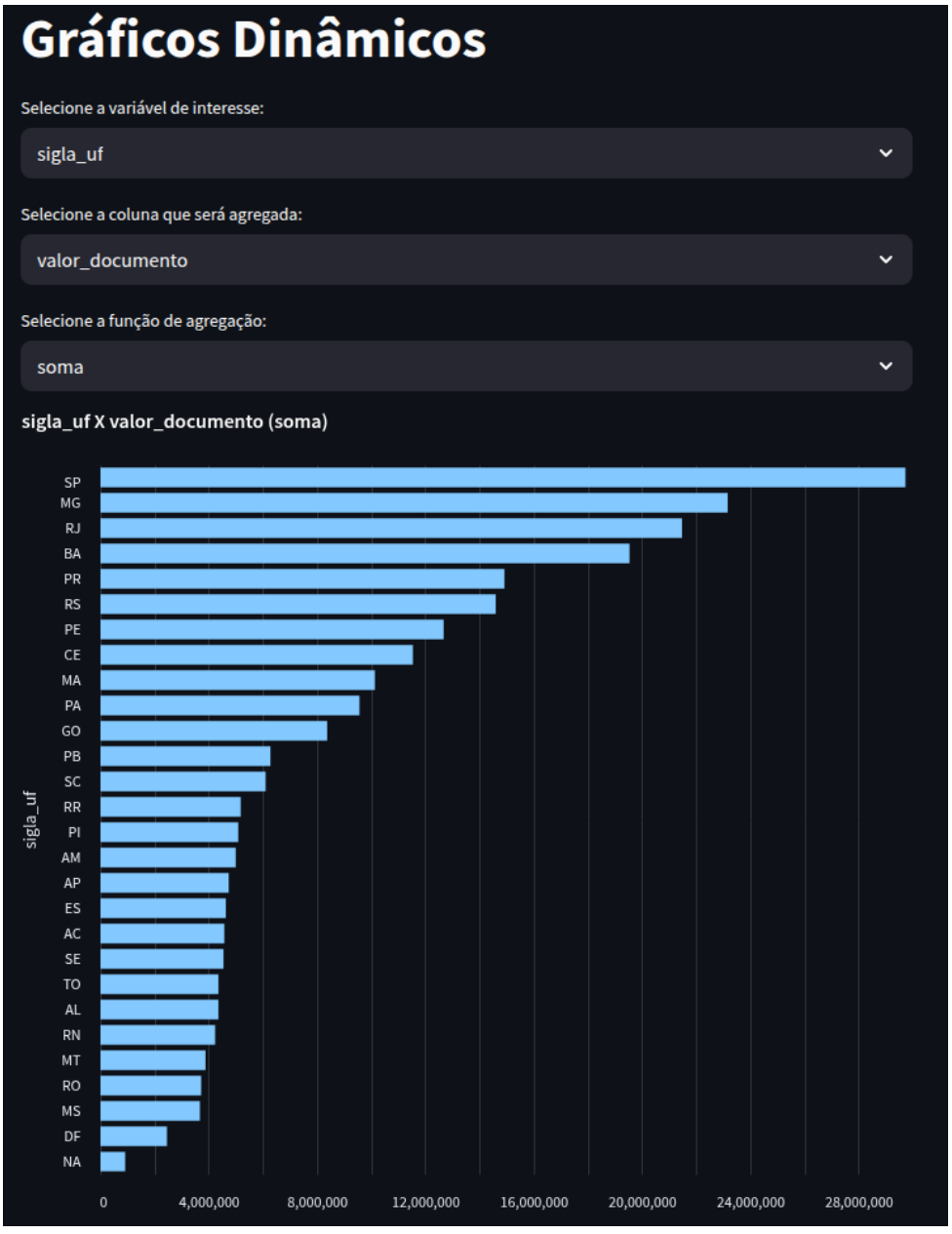

Figura 11: Valor dos gastos por UF.

Trocando a função de agregação, podemos obter outras informações como a média dos gastos por UF.

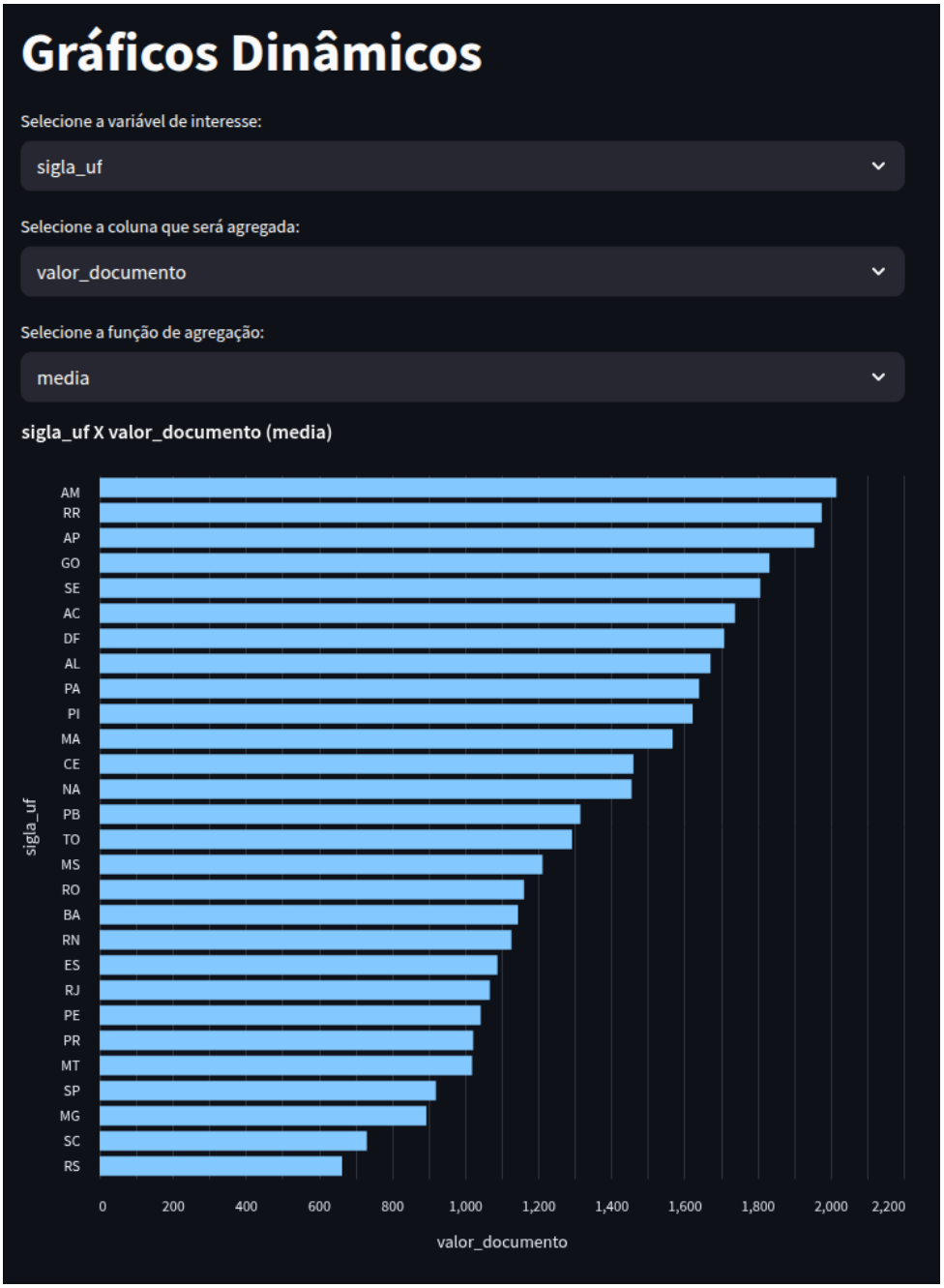

Figura 12: Valor médio dos gastos por UF.

Outra informação é a tendência de gastos mensais, aqui temos a tendência para os anos 2022, 2023 e 2024, mas como para o ano de 2022 temos uma quantidade bem maior de gastos, fica um pouco difícil analisar, por isso podemos selecionar os anos na barra lateral para então analisar cada ano individualmente.

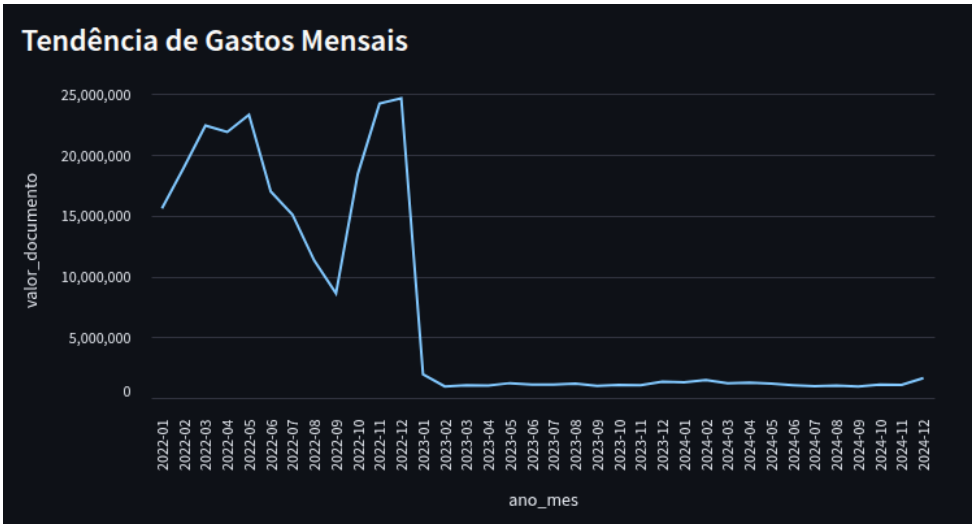

Figura 13: Tendência de gastos para os anos de 2022, 2023 e 2024.

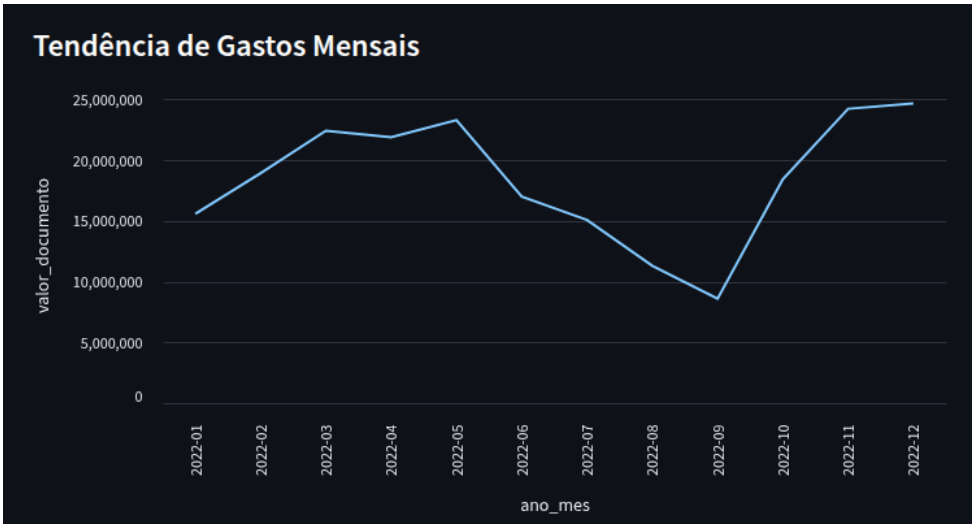

Figura 14: Tendência de gastos para o ano de 2022.

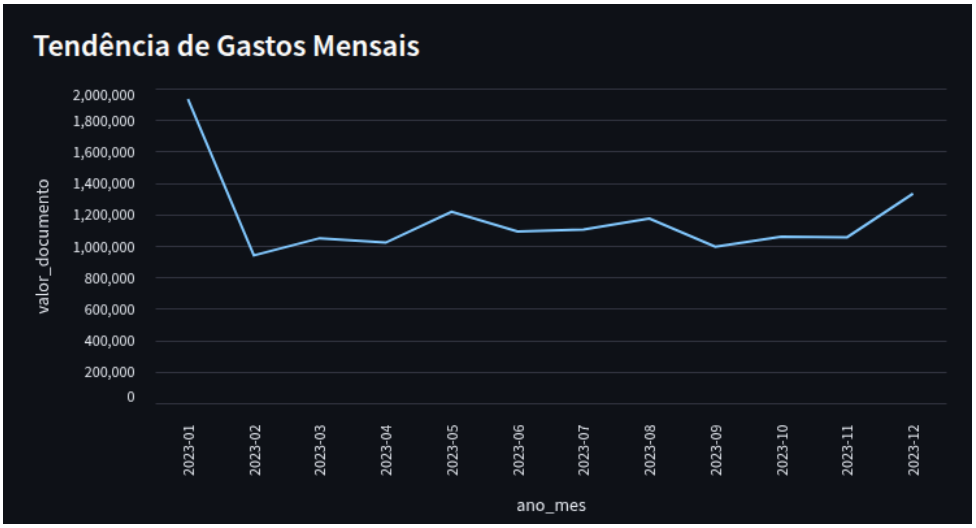

Figura 15: Tendência de gastos para o ano de 2023.

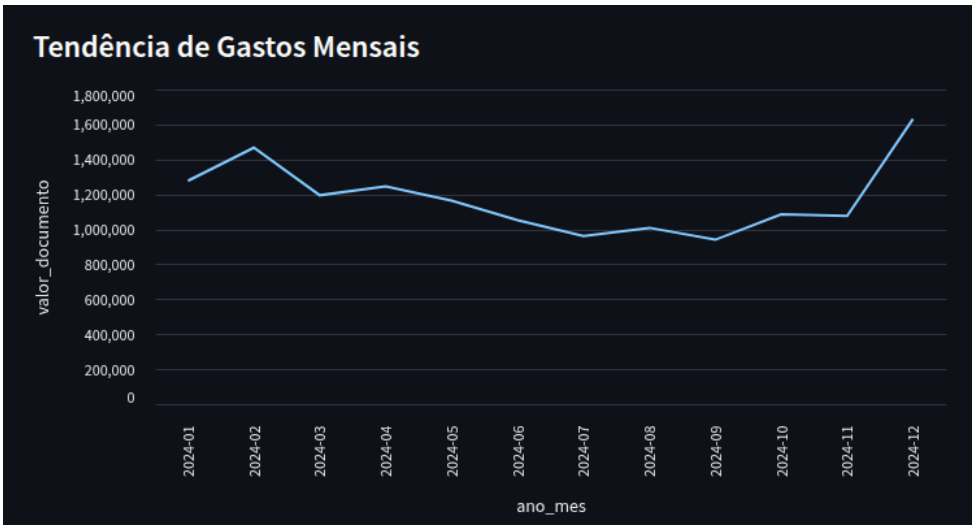

Figura 16: Tendência de gastos para o ano de 2024.

2.4 Considerações finais

O Streamlit é uma ferramenta robusta para análise de dados, permitindo a criação de gráficos interativos que facilitam a compreensão e a extração de insights valiosos. Diferente de plataformas de BI como o Power BI, o Streamlit oferece maior controle e personalização sobre suas visualizações, embora exija conhecimento em programação e um tempo de desenvolvimento inicial maior.

3 Machine Learning

3.1 Introdução

Machine Learning (Aprendizado de Máquina) é um campo da inteligência artificial que permite aos sistemas aprender com dados, identificar padrões, tomar decisões e fazer

predições futuras com mínima intervenção humana. Sua aplicação vai desde a detecção de anomalias e o reconhecimento de padrões em grandes volumes de dados até a otimização de processos e a personalização de experiências. É uma ferramenta poderosa para transformar dados brutos em insights acionáveis.

3.2 Dicionário de dados

No contexto do nosso projeto, as seguintes variáveis são selecionadas para a construção do modelo de machine learning:

- **nome_deputado**: Nome completo do deputado.
- **ano**: Ano em que a despesa foi realizada.
- **mes**: Mês em que a despesa foi realizada.
- **tipo_despesa**: Categoria da despesa, como "DIVULGAÇÃO DA ATIVIDADE PARLAMENTAR" ou "PASSAGEM AÉREA - SIGEPA".
- **valor_documento**: Valor da despesa conforme registrado no documento.
- **nome_fornecedor**: Nome completo do fornecedor do bem ou serviço.
- **sigla_partido**: Sigla do partido político ao qual o deputado é afiliado.
- **sigla_uf**: Sigla da Unidade Federativa (estado) pela qual o deputado foi eleito.

Com o propósito de criar um modelo de identificação de anomalias nos dados, utilizamos o algoritmo Isolation Forest[6].

O Isolation Forest é um algoritmo de Machine Learning não supervisionado amplamente utilizado para detecção de anomalias. Diferente de outros métodos que tentam "encontrar"a normalidade para depois identificar o que é diferente, o Isolation Forest adota uma abordagem mais direta, ele isola as anomalias. Ele funciona da seguinte forma[7]:

1. **Construção de Árvores Aleatórias**: O algoritmo constrói uma floresta de árvores de decisão. Para cada árvore, ele seleciona aleatoriamente um atributo e, em seguida, seleciona aleatoriamente um ponto de divisão dentro do intervalo de valores desse atributo.
2. **Isolamento de Pontos**: Esse processo de divisão continua até que cada ponto de dado esteja isolado ou um limite de profundidade da árvore seja atingido.
3. **Identificação de Anomalias**: O princípio é que as anomalias, por serem pontos mais raros e diferentes, geralmente exigem menos divisões para serem isoladas nas árvores. Pontos normais, por outro lado, são mais densos e requerem mais divisões para serem separados uns dos outros.

Podemos transformar os dados em um dataframe do Pandas[5], e usá-lo para criar um modelo usando a classe IsolatioForest presente no Scikit-Learn[6].

O pré-processamento dos dados é etapa crucial para criar bons modelos de machine learning, iniciaremos essa análise usando a biblioteca missingno para encontrar os valores ausentes no nosso dataset.

```
msno.matrix(df, figsize=(12, 5), fontsize=12)

import missingno as msno
```

Podemos ver que não há nenhuma linha com dado faltante no nosso dataset, por isso a parte de inputação de dados ou remoção de linhas com valores nulos não será necessária.

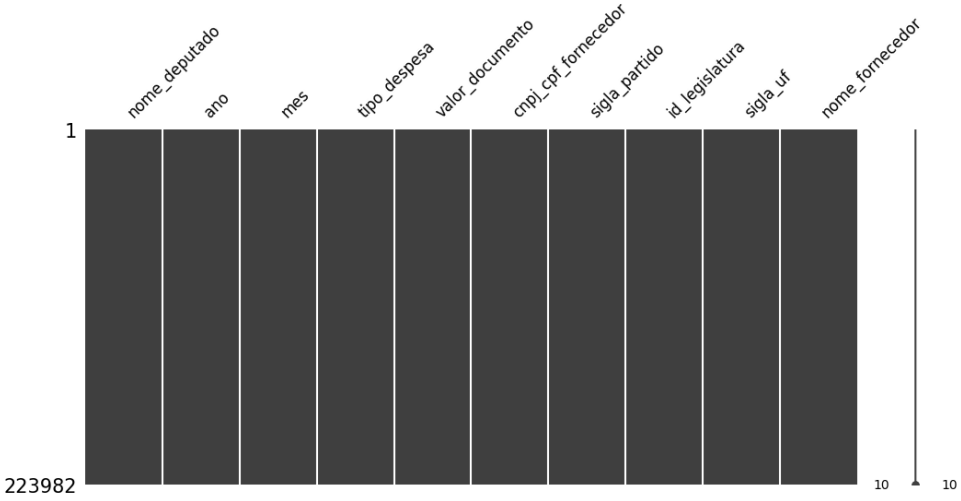

Figura 17: Dados faltantes.

Logo após podemos selecionar as features e usar o Label Encoder, com ele convertemos variáveis categóricas em numéricas para que o nosso modelo de machine learning possa computar.

```
le_df = df[
  [
    "nome_deputado",
    "ano",
    "mes",
    "tipo_despesa",
    "valor_documento",
    "sigla_partido",
    "sigla_uf",
    "nome_fornecedor",
  ]
]

for col in le_df.select_dtypes(include=["object"]).columns:
  label_encoder = LabelEncoder()
  le_df[col] = label_encoder.fit_transform(le_df[col].astype(str))
```

E por fim temos o nosso dataframe após o Label Encoder.

	nome_deputado	ano	mes	tipo_despesa	valor_documento	sigla_partido	sigla_uf	nome_fornecedor
0	3	2023	1	4	25000.00	3	3	9828
1	3	2023	1	19	353.70	3	3	5123
2	3	2023	1	19	116.43	3	3	5123
3	3	2023	1	9	10000.00	3	3	7176
4	3	2023	3	10	809.60	3	3	7384
5	3	2023	2	2	221.98	3	3	26
6	3	2023	2	2	283.97	3	3	26
7	3	2023	3	2	299.98	3	3	26
8	3	2023	5	2	100.04	3	3	26
9	3	2023	7	2	150.00	3	3	26

Figura 18: Label Encoder.

Após o pré-processamento dos dados podemos dar início ao processo de construção, treinamento e inferência do nosso modelo.

```
n_estimators = 100
contamination = 0.01
sample_size = 256

model = IsolationForest(
  n_estimators=n_estimators,
  contamination=contamination,
  max_samples=sample_size,
  random_state=42
)
model.fit(df)

df["anomaly"] = model.predict(le_df)

# -1 para anomalias, 1 para dados normais
anomalies = df[df["anomaly"] == -1]
```

E então, temos 2240 anomalias no dataframe contendo 223982 registros, isso representa 1.0000803636006466% dos registros.

Aqui temos alguns exemplos de dados que foram detectados como anomalias.

	nome_deputado	ano	mes	tipo_despesa	valor_documento	cnpj_cpf_fornecedor	sigla_partido	id_legislatura	sigla_uf	nome_fornecedor	anomaly
9662	Dra. Alessandra Haber	2024	11	MANUTENÇÃO DE ESCRITÓRIO DE APOIO À ATIVIDADE PARLAMENTAR	720	08532429000131	MDB	57	PA	AMORETTO CAFES EXPRESSO LTDA	-1
801	Bacelar	2023	5	MANUTENÇÃO DE ESCRITÓRIO DE APOIO À ATIVIDADE PARLAMENTAR	9124.9	89982185500	PV	57	BA	AFRANIO CEZAR OLIVA DE MATTOS FILHO	-1
7436	Aécio Neves	2024	7	MANUTENÇÃO DE ESCRITÓRIO DE APOIO À ATIVIDADE PARLAMENTAR	7723.9	04292201000160	PSDB	57	MG	ANUAR DONATO CONSULT. IMOBILIARIA	-1
8679	Clarissa Tércio	2024	1	MANUTENÇÃO DE ESCRITÓRIO DE APOIO À ATIVIDADE PARLAMENTAR	7500	61601152434	PP	57	PE	ILA DE SOUZA CABRAL	-1
13952	Ruy Carneiro	2024	10	COMBUSTÍVEIS E LUBRIFICANTES.	1129.15	01420327000185	PODE	57	PB	MOTOGAS IND.DE COMPRESSAO E COM.DE GAS NATURAL LTD	-1
190457	Alencar Santana	2022	12	DIVULGAÇÃO DA ATIVIDADE PARLAMENTAR.	62000	03188474000105	PT	57	SP	DIGRAF GRAFICA E EDITORA LTDA	-1
9378	Diego Coronel	2024	11	COMBUSTÍVEIS E LUBRIFICANTES.	100	00306597007614	PSD	57	BA	076 - MELHOR 10 - CASCOL COMBUSTIVEIS PARA VEICULOS LTDA	-1

Figura 19: Anomalias nos gastos.

3.3 Considerações finais

Para identificar comportamentos ou eventos incomuns que se desviam do padrão em grandes volumes de dados, o Isolation Forest é uma escolha excepcionalmente robusta. Essa ferramenta se mostra muito eficaz na detecção de problemas em gastos e como suporte crucial em auditorias, ajudando a detectar o que realmente importa.

4 Vídeo

O vídeo de 10 minutos mostrando todo o projeto foi gravado e disponibilizado por meio do google drive através do link: https://drive.google.com/file/d/14FIFpgSWjTtZ3jbXWYxiaxWnMd5

Referências

[1] Portal de Dados Abertos da Câmara dos Deputados: https://dadosabertos.camara.leg.br/swagger/api.html

[2] dbdiagram: https://dbdiagram.io/

[3] DBeaver: https://dbeaver.io/

[4] Repositório do Projeto no Github: https://github.com/lucassiro/pi

[5] Pandas: https://pandas.pydata.org/

[6] Isolation Forest no Scikit-Learn: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[7] Guia da floresta de isolamento: Explicação e implementação em Python: https://www.datacamp.com/pt/tutorial/isolation-forest

[8] Vídeo do projeto: https://drive.google.com/file/d/14FIFpgSWjTtZ3jbXWYxiaxWnMd5L5SLw/v