

**Usando recursos avançados da
ORM do Django para consultas
mais eficientes**

Olar :)

Mariana Bedran Lesche

Desenvolvedora na
Labcodes Software Studio

@maribedran

twitter.com

gmail.com

github.com

[https://github.com/labcodes/
dados_brasil_io](https://github.com/labcodes/dados_brasil_io)





Motivação

Explorar as vantagens e desvantagens dos recursos avançados da ORM do Django comparados com abordagens mais simples em termos de performance e legibilidade.

Trabalhando um pouco com SQL já conseguimos escrever consultas difíceis de implementar usando Django. Como juntar esses dois conhecimentos?

Quando eu devo usar isso?

Já experimentou todas as estratégias de otimização do banco?

- ✓ Indexar tabelas
- ✓ Pagar requisições
- ✓ Usar `select_related` e `prefetch_related` nos querysets
- ✓ Usar `values`, `values_list`, `only` e `defer` nos querysets
- ✓ Configurar cache do banco
- ✓ Usar `assertNumQueries` nos tests para prevenir excesso de consultas



Metodologia

- Escolher um conjunto de dados grande
- Elaborar perguntas complexas
- Tentar respondê-las com os recursos que só conhecia da documentação e nunca tinha usado



O que falhou

- Procrastinação
- Validar os dados durante a importação
- Minha máquina não tinha memória para fazer a importação
- Carregar os dados levou mais tempo que o desenvolvimento
- Juntar o conhecimento técnico com as ideias sobre os dados é mais difícil do que parece

Os dados



Dados abertos do Brasil

A Lei nº 12.527/2011 (**Lei de Acesso à Informação**) regulamenta o direito constitucional de acesso às informações públicas. Essa norma entrou em vigor em 16 de maio de 2012 e criou mecanismos que possibilitam, a qualquer pessoa, física ou jurídica, sem necessidade de apresentar motivo, o recebimento de informações públicas dos órgãos e entidades.



Na prática a teoria é outra

Apesar dos dados estarem lá, o acesso não é simples.

- Nem todas as agências governamentais cumprem a lei ou levam muito tempo para disponibilizar os dados
- Os dados são dispersos
- Muitas vezes vêm em formatos não fechados
- A maioria das pessoas que têm interesse nos dados não têm o conhecimento técnico para processá-los

O projeto Brasil IO

Projeto criado pelo Álvaro Justen (@turicas) para coletar, limpar e disponibilizar dados públicos.

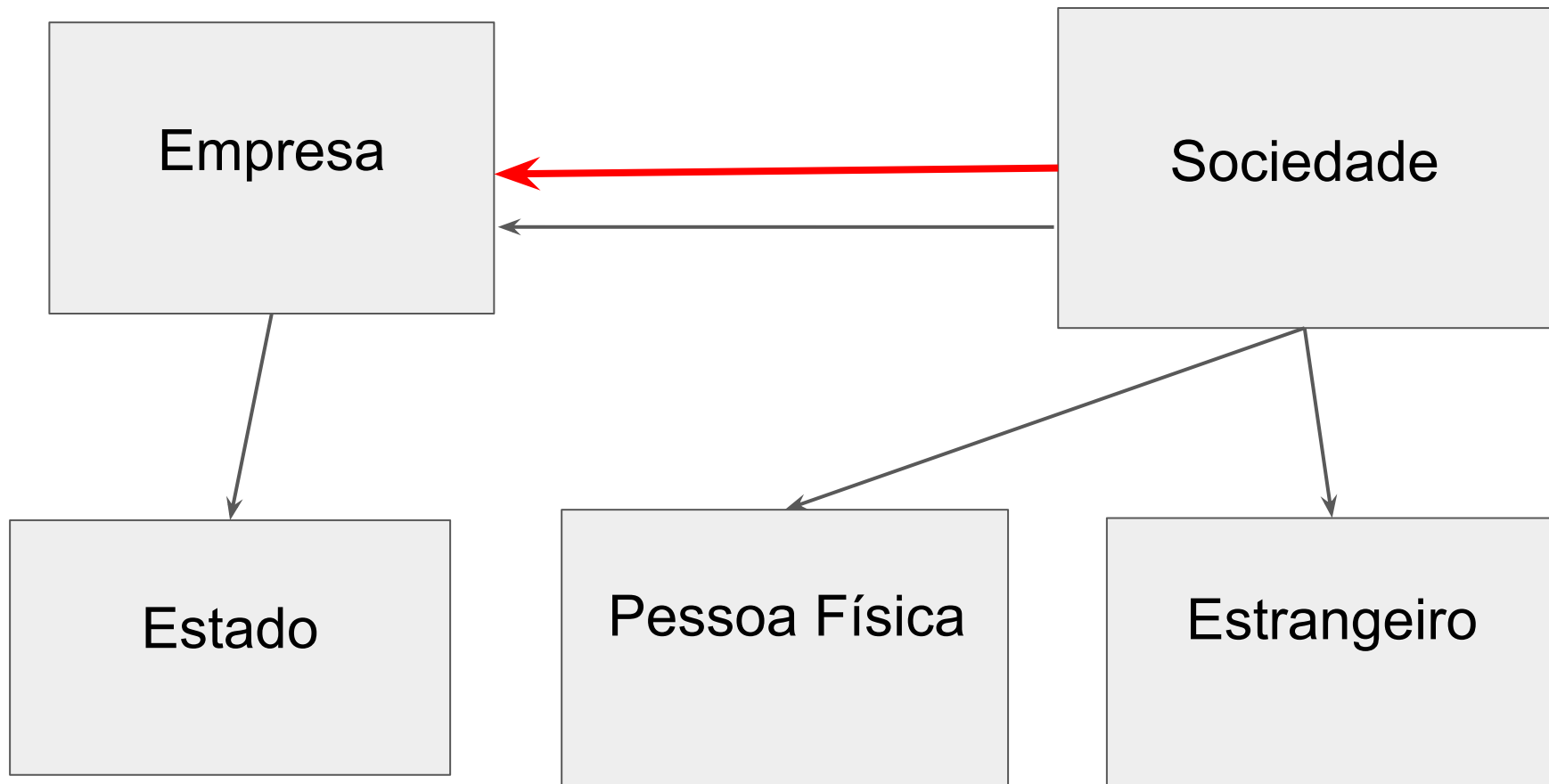
- Site: <https://brasil.io/home>
- Datasets: <https://brasil.io/datasets>
- GitHub: <https://github.com/turicas/brasil.io>
- Colabore com o projeto: <https://apoia.se/brasilio>

Os datasets



Sócios de empresas no Brasil

- **CNPJ** - Chave primária
- Nome do sócio
- Categoria do sócio
 - Pessoa Jurídica
 - Pessoa Física
 - Estrangeiro
- Categoria da sociedade
- **CNPJ** do sócio - se for PJ

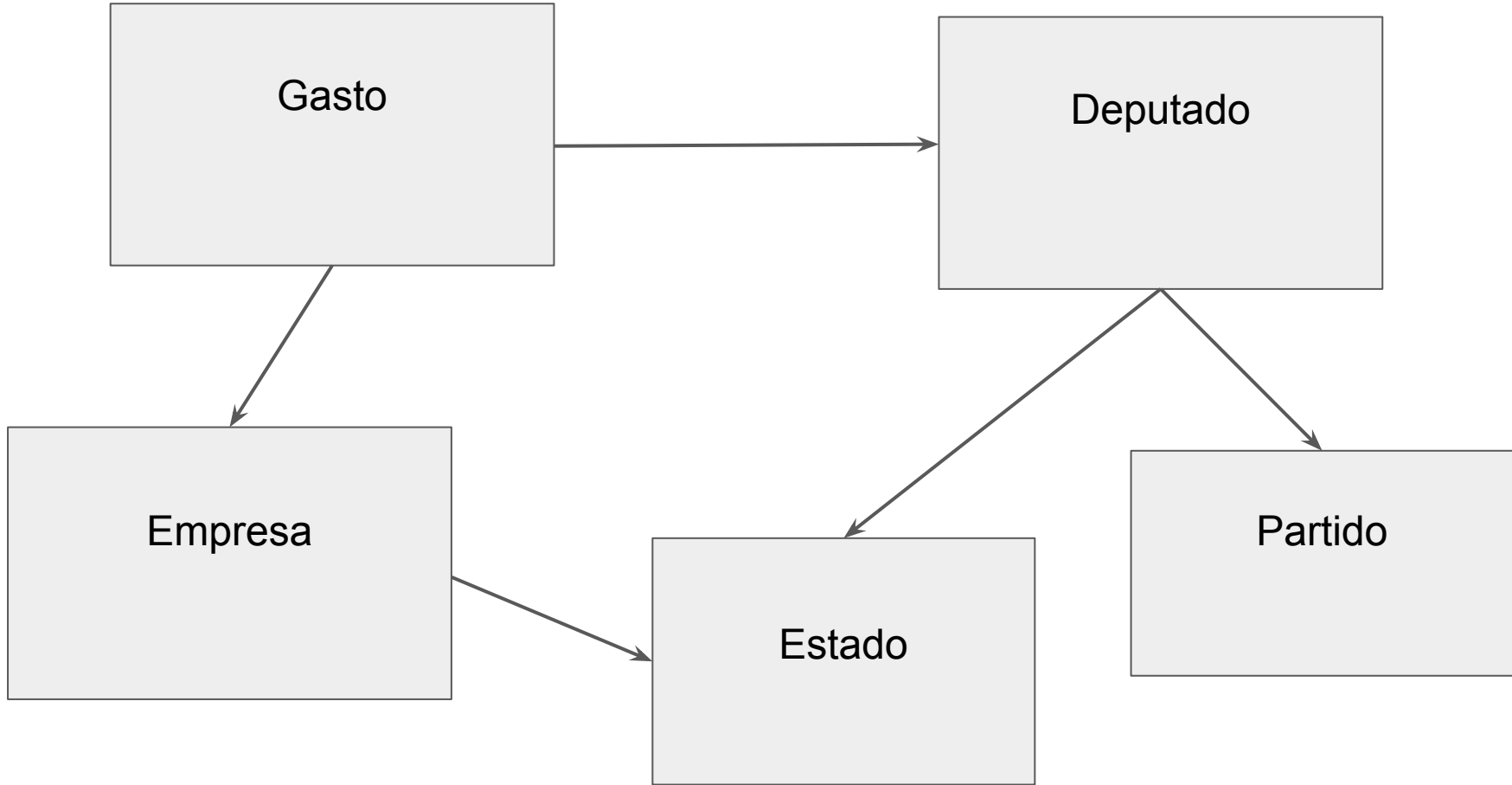




Gastos de Cota Parlamentar dos Deputados

Deputados no Congresso Nacional recebem uma cota para adquirir produtos e serviços no exercício da atividade parlamentar.

- ID do Deputado
- Partido
- Nome
- Data do gasto
- Mês de referência
- Ano de referência
- Quantia gasta
- Descrição
- CNPJ (se houver uma empresa)
- Outros campos ...



As queries

Objetos Prefetch são mara

Use para filtrar e acessar entidades relacionadas por contexto.

```
class DeputadoSerializer(serializers.ModelSerializer):
```

```
    class Meta:
```

```
        model = Deputado
```

```
        fields = [
```

```
            'id',
```

```
            'nome',
```

```
            'partido',
```

```
            'uf',
```

```
            'gastos'
```

```
        ]
```

```
        depth = 2
```

```
class DeputadoGastosListView(generics.ListAPIView):  
    serializer_class = DeputadoGastosSerializer  
  
    def get_queryset(self):  
        queryset = Deputado.objects.all().select_related(  
            'partido', 'uf'  
        ).prefetch_related(  
            'gastos'  
        )  
        # Retorna todos os gastos relacionados ao deputado
```

GastoCotaParlamentar



Deputado

```
class DeputadoListView(generics.ListAPIView):
    serializer_class = DeputadoSerializer

    def get_queryset(self):
        queryset = Deputado.objects.all().select_related(
            'partido', 'uf'
        )
        today = date.today()
        filtros = self.request.query_params.dict()
        filtros.setdefault('gastos_mes', today.month)
        filtros.setdefault('gastos_ano', today.year)

        return queryset.prefetch_gastos(**{
            campo.replace('gastos_', ''): valor
            for campo, valor in filtros.items()
            if campo.startswith('gastos_')
        })
```

```
from django.db.models import Prefetch
```

```
class DeputadoQuerySet(models.QuerySet):
```

```
    def prefetch_gastos(self, **kwargs):
        gastos_qs = GastoCotaParlamentar.objects.select_related(
            'empresa'
        ).filter(
            **kwargs      # {'ano': 2018, 'mes': 1}
        ) # Nunca faça isso na vida real
        prefetch = Prefetch('gastos', queryset=gastos_qs)
        return self.prefetch_related(prefetch)
```

GastoCotaParlamentar



Deputado

mat=json&gastos_mes=3&gastos_ano=2018&gastos_valor_liquido__gt=100

Search

JSONRaw DataHeaders

SaveCopy

Filter JSON

0:

id:133374

nome:"ADEMIR CAMILO"

partido:{...}

uf:{...}

id_legislatura:55

gastos:

0:{...}

1:{...}

2:{...}

3:{...}

4:{...}

1:

id:74103

nome:"ARIOSTO HOLANDA"

partido:{...}

uf:{...}

id_legislatura:55

gastos:

0:{...}

1:{...}

2:{...}

3:{...}

2:

id:74291

nome:"ARNON BEZERRA"

partido:{...}

uf:{...}

id_legislatura:55

gastos:[]

3:{...}

4:{...}

Filters

- mês 3
gastos_mes=3
- ano 2018
gastos_ano=2018
- valor líquido maior que 100
gastos_valor_liquido__gt=100

Aggregates também podem ser filtrados

```
from django.db.models import Q, Sum
```

```
class DeputadoQuerySet(models.QuerySet):
```

```
    def annotate_gasto_no_mes_por_deputado(self, mes, ano):
        annotation = {
            f'gastos_{ano}_{mes:02}': Sum(    # 'gastos_2018_01'
                'gastos__valor_liquido',
                filter=Q(
                    gastos__mes=mes,
                    gastos__ano=ano
                )
            )
        }
        return self.annotate(**annotation)
```



```
SELECT
    "políticos_deputado"."id",
    "políticos_deputado"."nome",
    "políticos_deputado"."partido_id",
    "políticos_deputado"."uf_id",
    "políticos_deputado"."id_legislatura",
    "políticos_deputado"."carteira_parlamentar",
    SUM("políticos_gastocotaparlamentar"."valor_liquido") FILTER (
        WHERE
            (
                "políticos_gastocotaparlamentar"."ano" = 2018
                AND "políticos_gastocotaparlamentar"."mes" = 1
            )
        ) AS "gastos_2018_01"
FROM
    "políticos_deputado"
    LEFT OUTER JOIN "políticos_gastocotaparlamentar" ON (
        "políticos_deputado"."id" = "políticos_gastocotaparlamentar"."deputado_id"
    )
GROUP BY
    "políticos_deputado"."id"
ORDER BY
    "políticos_deputado"."id" ASC
LIMIT
    1;
```



Mas tem também o `FilteredRelations` pra isso

```
from django.db.models import FilteredRelation, Q, Sum


class DeputadoQuerySet(models.QuerySet):

    def annotate_gasto_no_mes_por_deputado2(self, mes, ano):
        return self.annotate(
            gastos_filtrados=FilteredRelation(
                'gastos',
                condition=Q(
                    gastos__mes=mes,
                    gastos__ano=ano,
                )
            )
        ).annotate(**{
            f'gastos_{ano}_{mes:02}': Sum(
                'gastos_filtrados__valor_liquido'
            )
        })
```

```
SELECT
    "políticos_deputado"."id",
    "políticos_deputado"."nome",
    "políticos_deputado"."partido_id",
    "políticos_deputado"."uf_id",
    "políticos_deputado"."id_legislatura",
    "políticos_deputado"."carteira_parlamentar",
    SUM(gastos_filtrados."valor_liquido") AS "gastos_2018_01"
FROM
    "políticos_deputado"
    LEFT OUTER JOIN "políticos_gastocotaparlamentar" gastos_filtrados ON (
        "políticos_deputado"."id" = gastos_filtrados."deputado_id"
        AND (
            (
                gastos_filtrados."ano" = 2018
                AND gastos_filtrados."mes" = 1
            )
        )
    )
GROUP BY
    "políticos_deputado"."id"
ORDER BY
    "políticos_deputado"."id" ASC
LIMIT
    1;
```



QUERY PLAN



```
Sort (cost=130530.24..130531.75 rows=604 width=48) (actual time=1740.328..1740.354 rows=608 loops=1)
  Output: politicos_deputado.id, politicos_deputado.nome, politicos_deputado.partido_id, politicos_deputado.uf_id, politicos_deputado.id_legislatura, politicos_deputado.
  carteira_parlamentar, (sum(politicos_gastocotaparlamentar.valor_liquido) FILTER (WHERE ((politicos_gastocotaparlamentar.ano = 2018) AND (politicos_gastocotaparlamentar.mes = 1))))
  Sort Key: politicos_deputado.id
  Sort Method: quicksort Memory: 74kB
  -> HashAggregate (cost=130494.79..130502.34 rows=604 width=48) (actual time=1740.000..1740.159 rows=608 loops=1)
    Output: politicos_deputado.id, politicos_deputado.nome, politicos_deputado.partido_id, politicos_deputado.uf_id, politicos_deputado.id_legislatura, politicos_deputado.
    carteira_parlamentar, sum(politicos_gastocotaparlamentar.valor_liquido) FILTER (WHERE ((politicos_gastocotaparlamentar.ano = 2018) AND (politicos_gastocotaparlamentar.mes = 1)))
    Group Key: politicos_deputado.id
    -> Hash Right Join (cost=22.59..108654.73 rows=2184006 width=48) (actual time=16.266..1277.289 rows=2184008 loops=1)
      Output: politicos_deputado.id, politicos_deputado.nome, politicos_deputado.partido_id, politicos_deputado.uf_id, politicos_deputado.id_legislatura, politicos_deputado.
      carteira_parlamentar, politicos_gastocotaparlamentar.valor_liquido, politicos_gastocotaparlamentar.ano, politicos_gastocotaparlamentar.mes
      Hash Cond: (politicos_gastocotaparlamentar.deputado_id = politicos_deputado.id)
      -> Seq Scan on public.politicos_gastocotaparlamentar (cost=0.00..78602.06 rows=2184006 width=18) (actual time=15.813..404.691 rows=2184006 loops=1)
        Output: politicos_gastocotaparlamentar.id, politicos_gastocotaparlamentar.legislatura, politicos_gastocotaparlamentar.data_emissao, politicos_gastocotaparlamentar.
        documento, politicos_gastocotaparlamentar.tipo_documento, politicos_gastocotaparlamentar.ano, politicos_gastocotaparlamentar.especificacao_subcota, politicos_gastocotaparlamentar.
        lote, politicos_gastocotaparlamentar.mes, politicos_gastocotaparlamentar.parcela, politicos_gastocotaparlamentar.resscamento, politicos_gastocotaparlamentar.subcota,
        politicos_gastocotaparlamentar.cpf, politicos_gastocotaparlamentar.descricao, politicos_gastocotaparlamentar.descricao_especificacao, politicos_gastocotaparlamentar.fornecedor,
        politicos_gastocotaparlamentar.numero_documento, politicos_gastocotaparlamentar.nome_passageiro, politicos_gastocotaparlamentar.trecho_viagem, politicos_gastocotaparlamentar.
        valor_documento, politicos_gastocotaparlamentar.valor_glosa, politicos_gastocotaparlamentar.valor_liquido, politicos_gastocotaparlamentar.valor_restituicao, politicos_gastocotaparlamentar.
        deputado_id, politicos_gastocotaparlamentar.empresa_id
      -> Hash (cost=15.04..15.04 rows=604 width=34) (actual time=0.434..0.434 rows=608 loops=1)
        Output: politicos_deputado.id, politicos_deputado.nome, politicos_deputado.partido_id, politicos_deputado.uf_id, politicos_deputado.id_legislatura, politicos_deputado.
        carteira_parlamentar
        Buckets: 1024 Batches: 1 Memory Usage: 47kB
        -> Seq Scan on public.politicos_deputado (cost=0.00..15.04 rows=604 width=34) (actual time=0.008..0.194 rows=608 loops=1)
          Output: politicos_deputado.id, politicos_deputado.nome, politicos_deputado.partido_id, politicos_deputado.uf_id, politicos_deputado.id_legislatura, politicos_deputado.
          carteira_parlamentar
        Planning time: 0.391 ms
        Execution time: 1740.445 ms
(19 rows)
```

(END)

Dá pra fazer aggregate em annotations

```
from django.db.models import Avg, Sum, Q

class DeputadoQuerySet(models.QuerySet):

    def annotate_gasto_mensal_por_deputado(self):
        meses = range(1, 13)
        anos = range(2009, 2019)
        annotations = {
            f'gastos_{ano}_{mes:02}': Sum( # 'gastos_2018_01'
                'gastos__valor_liquido',
                filter=Q(gastos__mes=mes, gastos__ano=ano)
            )
            for ano in anos for mes in meses
        }
        return self.annotate(**annotations)
```

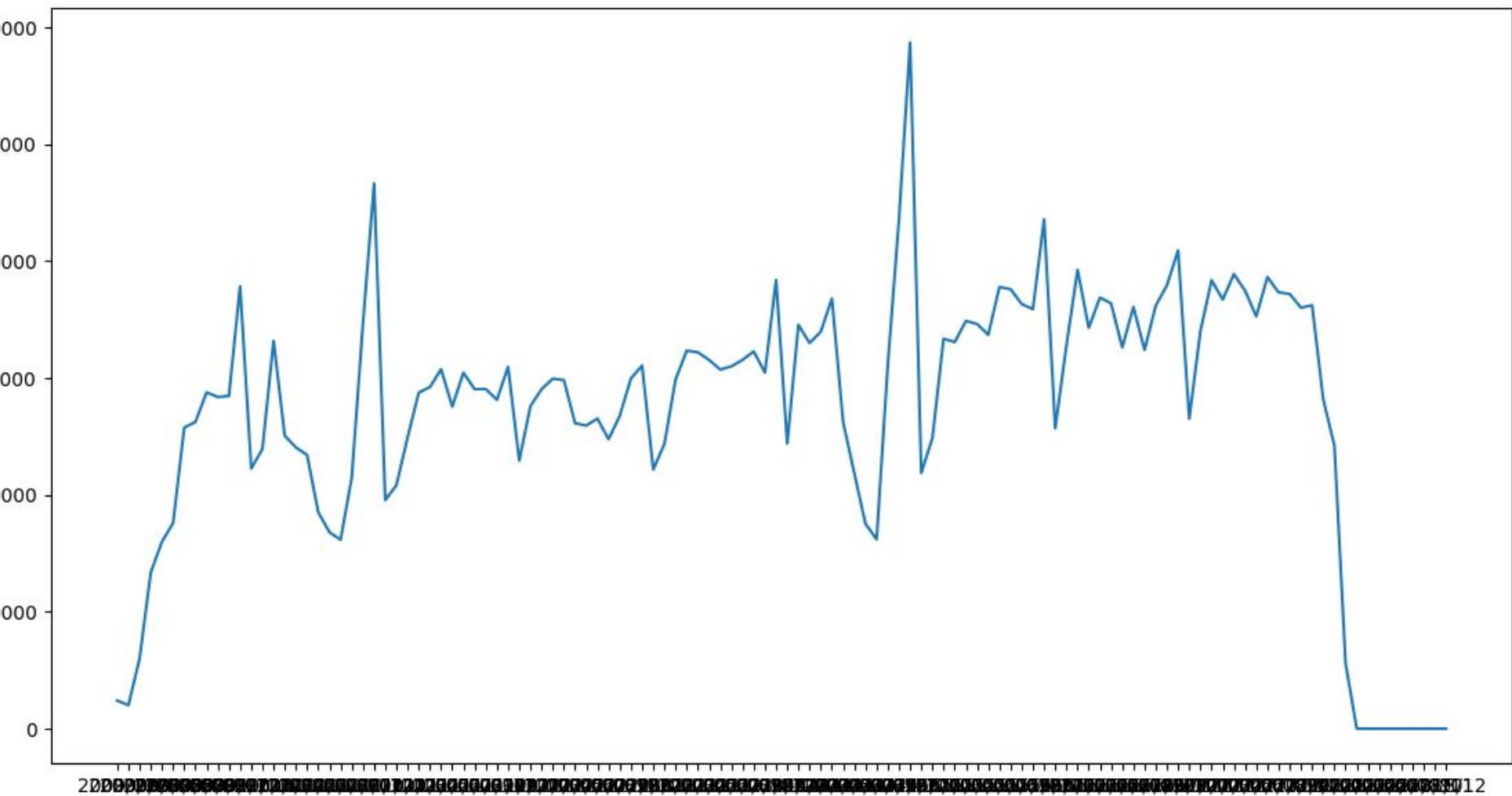


```
from django.db.models import Avg

class DeputadoQuerySet(models.QuerySet):

    def get_media_mensal(self):
        meses = range(1, 13)
        anos = range(2009, 2019)
        aggregations = {
            f'media_{ano}_{mes:02}': Avg( # 'media_2018_01'
                f'gastos_{ano}_{mes:02}' # gastos_2018_01
            )
            for ano in anos for mes in meses
        }

    return self.annotate_gasto_mensal_por_deputado() \
        .aggregate(**aggregations)
```



Subqueries

```
from django.db.models import Exists, OuterRef

class DeputadoQuerySet(models.QuerySet):

    def annotate_empresas(self):
        empresas_qs = Empresa.objects.filter(
            sociedades__socio_pessoa_fisica__nome=OuterRef('nome'),
            uf=OuterRef('uf')
        )
        return self.annotate(
            empresas=Exists(empresas_qs)
        )
```

```
SELECT
    "políticos_deputado"."id",
    "políticos_deputado"."nome",
    "políticos_deputado"."partido_id",
    "políticos_deputado"."uf_id",
    "políticos_deputado"."id_legislatura",
    "políticos_deputado"."carteira_parlamentar",
    EXISTS (
        SELECT
            U0."cnpj",
            U0."nome",
            U0."uf_id"
        FROM
            "empresas_empresa" U0
        INNER JOIN "empresas_sociedade" U1 ON (U0."cnpj" = U1."empresa_id")
        INNER JOIN "empresas_pessoafisica" U2 ON (U1."socio_pessoa_fisica_id" = U2."id")
        WHERE
            (
                U2."nome" = ("políticos_deputado"."nome")
                AND U0."uf_id" = ("políticos_deputado"."uf_id")
            )
    ) AS "empresas"
FROM
    "políticos_deputado"
;
```



Obrigada!

@maribedran

www.labcodes.com.br
github.com/labcodes
twitter.com/labcodes
speakerdeck.com/labcodes