

Aula 05 | PosTech | Manipulando datas e gerando novas análises

Anotações sobre a quinta aula da PosTech FIAP ✨✨

<https://on.fiap.com.br/mod/conteudoshtml/view.php?id=307799&c=8729&sesskey=mp0PcE8JII>

Temas abordados:

- Como fazer uma comparação ano a ano dos custos do SUS usando o DateTime;
- Como analisar e manipular os dados para criar análises mais avançadas;
- Saber ler um problema e atuar com eficácia.

Pré-requisitos:

- Base de dados:

<https://github.com/alura-tech/pos-datascience-introducao-a-visualizacao/archive/refs/heads/dados.zip>

- Importar essa base de dados no Colaboratory

Um modo eficaz e legal para usarmos quando queremos **converter uma coluna de data para um formato correto**, utilizando a biblioteca Pandas, é utilizar a função `to_datetime()`, que nos permite **passar dentro da função a coluna da qual queremos fazer a conversão específica**.

```
□ pd.to_datetime(mensal_aberto["dia_mes_ano"])
```

Esse exemplo foi tirado da videoaula sobre manipulação de datas. Para melhor entendimento, assistir ela. Aprenderemos também a como criar colunas desmembrando-as por ano, mês e dia, por meio do atributo '**dt**'.

Agora vamos ver, na prática, como começar a importar os nossos dados e trabalhar com eles via programação.

Parte 1 | Manipulando datas e gerando novas análises I

Continuando a aula anterior, eu não quero plotar a linha de São Paulo solta, porque fica difícil de comparar São Paulo. Então...

```
□ mensal_aberto.head()
```

```
✓ [142] 1 mensal_aberto.head()
```

	mes	uf	gasto
0	2008-02-01	São Paulo	173.06
1	2008-03-01	São Paulo	170.62
2	2008-04-01	São Paulo	170.39
3	2008-05-01	São Paulo	172.51
4	2008-06-01	São Paulo	175.56

Se eu quero mostrar uma tendência crescente, um gráfico que mostre todos os anos é legal. Agora, se quiser mostrar um crescimento desse ano em relação ao ano passado, aí precisamos de outros gráficos.

Para fazer isso, precisamos abrir esses anos e esses meses. Voltando lá em cima...

```
☐ mensal_aberto = mensal.reset_index().melt(id_vars=["index"], value_vars=mensal.columns)
☐ #mensal_aberto.columns = ["mes", "uf", "gasto"]
☐ #mensal_aberto.columns = ["dia_mes_ano", "uf", "gasto"] # editamos aqui
☐ mensal_aberto.head()
```

Editando isso também:

```
☐ plt.figure(figsize=(10,6))
```

- ☐ `#axis = sns.lineplot(data=mensal_aberto, x="mes", y="gasto", hue="uf")`
- ☐ `axis = sns.lineplot(data=mensal_aberto, x="dia_mes_ano", y="gasto", hue="uf")`
- ☐ `plt.ylim(0,600)`
- ☐ `plt.grid(linestyle="--")`
- ☐ `plt.show()`

Aí de novo...

- ☐ `mensal_aberto.head()`

```
✓ [145] 1 mensal_aberto.head()
0s
```

	dia_mes_ano	uf	gasto
0	2008-02-01	São Paulo	173.06
1	2008-03-01	São Paulo	170.62
2	2008-04-01	São Paulo	170.39
3	2008-05-01	São Paulo	172.51
4	2008-06-01	São Paulo	175.56

- ☐ `mensal_aberto["dia_mes_ano"][0]`

```
✓ [146] 1 mensal_aberto["dia_mes_ano"][0]
0s
```

```
datetime.date(2008, 2, 1)
```

Aqui nós temos um datetime. Agora, se eu quiser pegar o valor do ano desse datetime, para poder trabalhar com esses valores, nós vamos lá na documentação “datetime” do

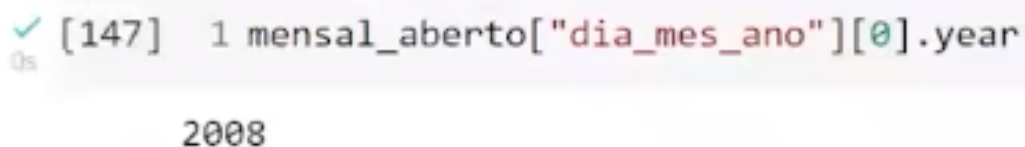
Python. Podemos representar uma diferença de tempo com ela, pode ser uma data, data e hora, pegar o tempo atual, timezone, enfim, temos várias formas de uso.

Dentre elas, temos o **.year**.

<https://docs.python.org/3/library/datetime.html>

Voltando para o código...

☐ `mensal_aberto["dia_mes_ano"][0].year` # aqui, ele vai trazer 2008

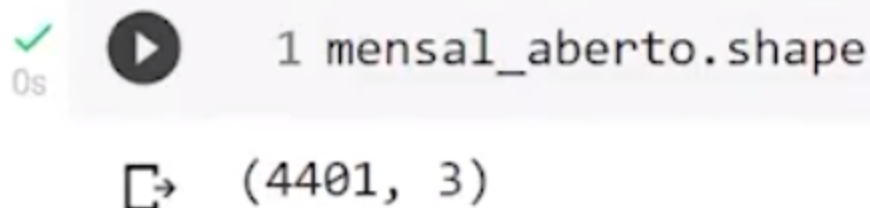


```
[147]: 1 mensal_aberto["dia_mes_ano"][0].year
```

2008

Portanto, como utilizar isso?

☐ `mensal_aberto.shape`



```
1 mensal_aberto.shape
```

(4401, 3)

Aqui verificamos que nossa tabela tem 4.401 linhas. Então se eu quiser pegar `mensal_aberto` e criar uma coluna nova, eu preciso passar 4.401 valores, certo? Um

valor para cada linha.

Vamos pegar mensal_aberto e vamos atribuir à coluna (variável) “mes” os valores e aí eu posso pegar o .year de toda essa coluna dia_mes_ano.

```
mensal_aberto["dia_mes_ano"]
```

```
0s 1 mensal_aberto["dia_mes_ano"]
```

0	2008-02-01
1	2008-03-01
2	2008-04-01
3	2008-05-01
4	2008-06-01
...	
4396	2021-04-01
4397	2021-05-01
4398	2021-06-01
4399	2021-07-01
4400	2021-08-01

Name: dia_mes_ano, Length: 4401, dtype: object

Aqui nessa coluna dia_mes_ano, temos um tamanho de justamente 4.401. O que fazer com isso? Quero pegar esse objeto (datetime) e quero pegar apenas o ano.

```
mensal_aberto["dia_mes_ano"].year
```

Dá um erro dizendo que “date series” não tem “year”.

```
1 mensal_aberto["dia_mes_ano"].year

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-150-21ff77d0cd92> in <module>()
----> 1 mensal_aberto["dia_mes_ano"].year

/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in __getattr__(self, name)
    5139         if self._info_axis._can_hold_identifiers_and_holds_name(name):
    5140             return self[name]
-> 5141         return object.__getattr__(self, name)
    5142
    5143     def __setattr__(self, name: str, value) -> None:

AttributeError: 'Series' object has no attribute 'year'
```

Quando queremos aplicar uma função de string para uma time series que tinha várias strings, não era a mesma coisa né? Quero, portanto, tratar essa time series como datetime (dt):

☐ `mensal_aberto["dia_mes_ano"].dt.year`

Erro porque ainda precisamos falar pro Pandas que essa coluna vai ser um objeto datetime.

```
1 mensal_aberto["dia_mes_ano"].dt.year

-----
AttributeError                                Traceback (most recent call last)
<ipython-input-151-cdb80a577639> in <module>()
----> 1 mensal_aberto["dia_mes_ano"].dt.year

----- 2 frames -----
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/accessors.py in __new__(cls, data)
    478         return PeriodProperties(data, orig)
    479
-> 480         raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

Se eu der um `mensal_aberto.info()`, ele vai dizer que a coluna de “dia_mes_ano” é do tipo “object”.

✓
0s



```
1 mensal_aberto.info()
```



```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4401 entries, 0 to 4400  
Data columns (total 3 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   dia_mes_ano  4401 non-null   object  
1   uf           4401 non-null   object  
2   gasto       4401 non-null   float64  
dtypes: float64(1), object(2)  
memory usage: 103.3+ KB
```

Então precisaremos trocar o tipo:



```
mensal_aberto["dia_mes_ano"].astype("datetime64").dt.year
```

✓
0s



```
1 mensal_aberto["dia_mes_ano"].astype("datetime64").dt.year
```



```
0      2008  
1      2008  
2      2008  
3      2008  
4      2008  
...  
4396   2021  
4397   2021  
4398   2021  
4399   2021  
4400   2021  
Name: dia_mes_ano, Length: 4401, dtype: int64
```


Outra forma de fazer isso é:

- ☐ `mensal_aberto["dia_mes_ano"] = pd.to_datetime(mensal_aberto["dia_mes_ano"])`
- ☐ `mensal_aberto["ano"] = mensal_aberto["dia_mes_ano"].dt.year`
- ☐ `mensal_aberto["mes"] = mensal_aberto["dia_mes_ano"].dt.month`
- ☐ `mensal_aberto.head()`

```
1 mensal_aberto["ano"] = mensal_aberto["dia_mes_ano"].dt.year
2 mensal_aberto["mes"] = mensal_aberto["dia_mes_ano"].dt.month
3 mensal_aberto.head()
```

	dia_mes_ano	uf	gasto	ano	mes
0	2008-02-01	São Paulo	173.06	2008	2
1	2008-03-01	São Paulo	170.62	2008	3
2	2008-04-01	São Paulo	170.39	2008	4
3	2008-05-01	São Paulo	172.51	2008	5
4	2008-06-01	São Paulo	175.56	2008	6

Podemos fazer uma pergunta (query) usando o Pandas, queremos que a coluna uf seja igual à São Paulo. Isso vai passar para cada coluna e aplicar esse “if” (se for SP ou não).

✓ 0s

```
1 mensal_aberto.query("uf=='São Paulo'")
```

	dia_mes_ano	uf	gasto	ano	mes
0	2008-02-01	São Paulo	173.06	2008	2
1	2008-03-01	São Paulo	170.62	2008	3
2	2008-04-01	São Paulo	170.39	2008	4
3	2008-05-01	São Paulo	172.51	2008	5
4	2008-06-01	São Paulo	175.56	2008	6
...
158	2021-04-01	São Paulo	504.43	2021	4
159	2021-05-01	São Paulo	548.11	2021	5
160	2021-06-01	São Paulo	503.16	2021	6
161	2021-07-01	São Paulo	404.37	2021	7
162	2021-08-01	São Paulo	301.99	2021	8

☐ `plt.figure(figsize=(10,6))`

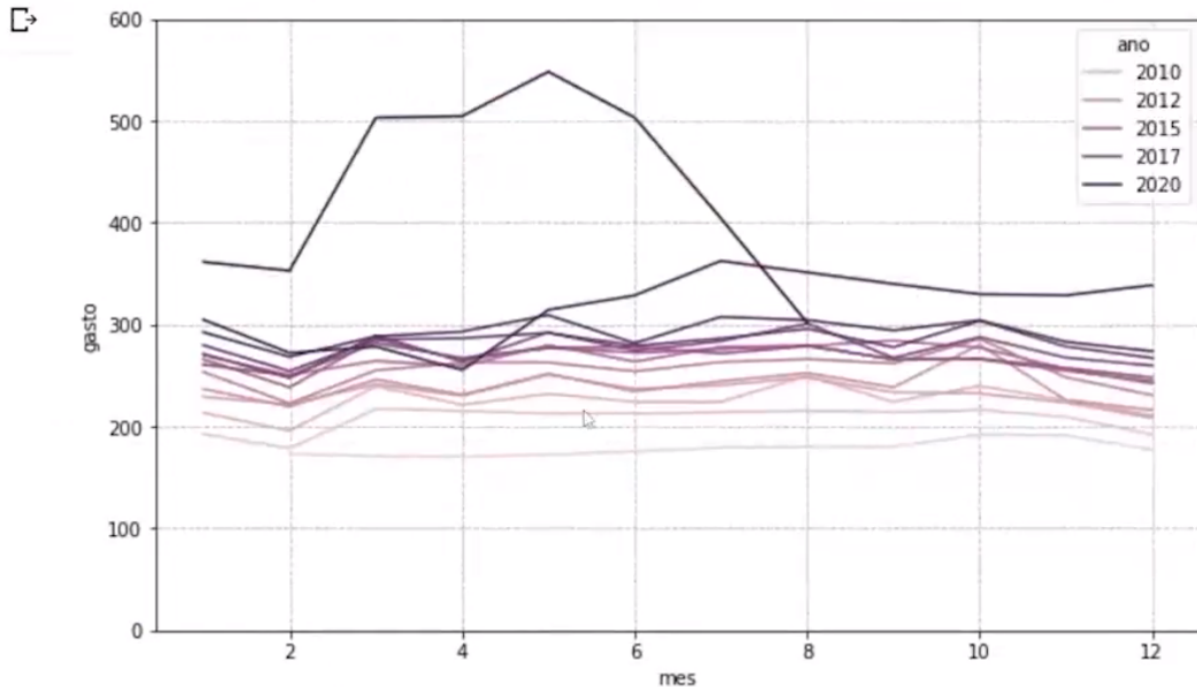
☐ `axis = sns.lineplot(data=mensal_aberto.query("uf=='São Paulo'"), x="mes", y="gasto", hue="ano")`

☐ `plt.ylim(0, 600)`

☐ `plt.grid(linestyle="--")`

☐ `plt.show()`

```
4 plt.grid(linestyle="--")
5 plt.show()
```



Aqui temos um resultado de diversos anos com seus gastos por meses.

Agora, fica mais fácil de comparar os anos.

Desafio 01: escolher um ano e plotar o mesmo ano para dois estados diferentes no mesmo gráfico. Usar estados de regiões diferentes do país. Comparar e tentar concluir algo.

Desafio 02: melhorar a paleta de cores, as labels, o espaçamento dos ticks (de 2 em 2 meses está meio estranho).

Desafio 03: nomes dos meses no eixo x.

A função 'map' pode funcionar não só como função mas também como dicionário, isso é legal de saber:

```
dias_por_mes = {  
  1 : 31,  
  2 : 28,  
  3 : 31,  
  4 : 30,  
  5 : 31,  
  6 : 30,  
  7 : 31,  
  8 : 31,  
  9 : 30,  
 10 : 31,  
 11 : 30,  
 12 : 31  
}
```

```
mensal_aberto["gasto_diario"] = mensal_aberto["gasto"] /  
mensal_aberto["mes"].map(dias_por_mes)
```

```
mensal_aberto.head()
```

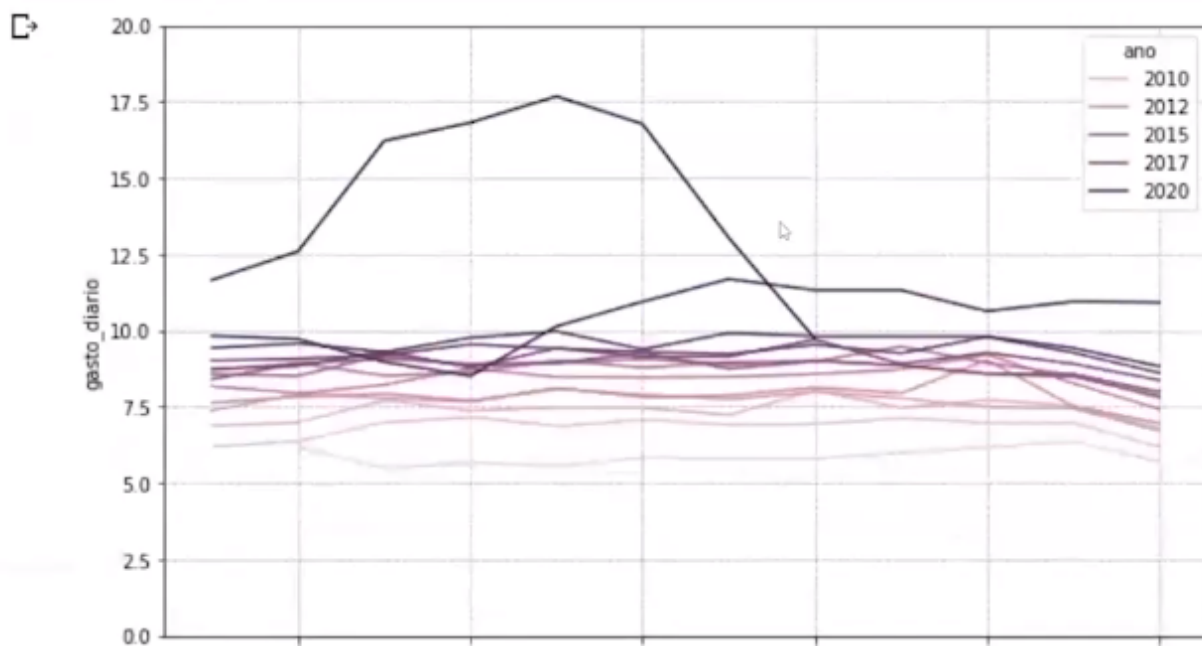
```
1 mensal_aberto["gasto_diario"] = mensal_aberto["gasto"] / mensal_aberto["mes"].map(dias_por_mes)  
2 mensal_aberto.head()
```

	dia_mes_ano	uf	gasto	ano	mes	gasto_diario
0	2008-02-01	São Paulo	173.06	2008	2	6.18
1	2008-03-01	São Paulo	170.62	2008	3	5.50
2	2008-04-01	São Paulo	170.39	2008	4	5.68
3	2008-05-01	São Paulo	172.51	2008	5	5.56
4	2008-06-01	São Paulo	175.56	2008	6	5.85

Agora nós podemos plotar os gastos diários:

- ☐ `plt.figure(figsize=(10,6))`
- ☐ `axis = sns.lineplot(data=mensal_aberto.query("uf=='São Paulo'"), x="mes",
y="gasto_diario", hue="ano") # mudei apenas o y`
- ☐ `plt.ylim(0, 20) # mudei de 600 para 20`
- ☐ `plt.grid(linestyle="--")`
- ☐ `plt.show()`

```
3 plt.ylim(0, 20)
4 plt.grid(linestyle="--")
5 plt.show()
```



Sempre quando vamos fazer comparações, existe a comparação absoluta e a comparação relativa.

Vamos mudar agora o uf para Pará:

☐ `plt.figure(figsize(10,6))`

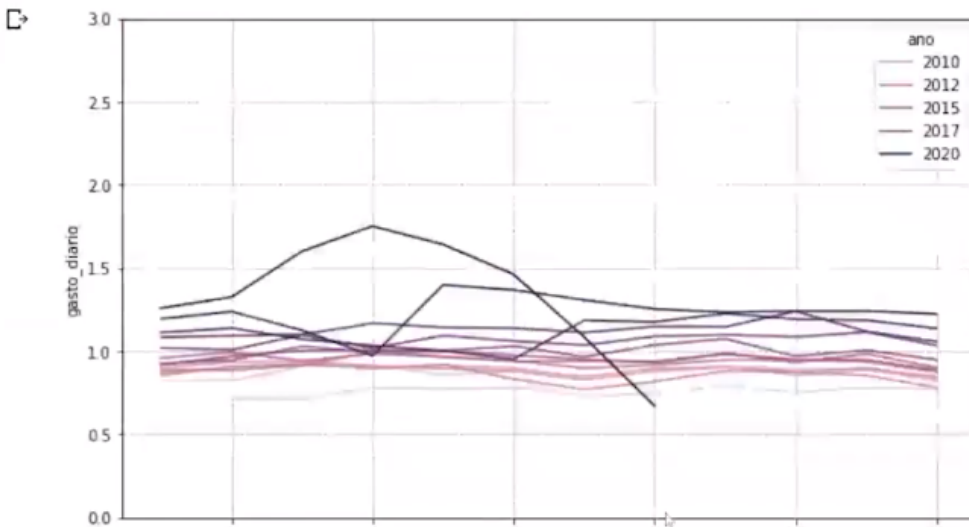
☐ `axis = sns.lineplot(data=mensal_aberto.query("uf=='Pará'"), x="mes", y="gasto_diario", hue="ano")` # mudei apenas o uf

☐ `plt.ylim(0, 3)` # mudei de 20 para 3

☐ `plt.grid(linestyle="--")`

☐ `plt.show()`

```
2 axis = sns.lineplot(data=mensal_aberto.query("uf=='Pará'"), x="mes", y="gasto_diario", hue="ano")
3 plt.ylim(0, 3)
4 plt.grid(linestyle="--")
5 plt.show()
```



Agora consigo ver melhor o Pará, mas fica muito difícil de comparar com São Paulo porque são escalas diferentes, do jeito que está feito não dá para comparar um gráfico com o outro.

Se colocarmos o limite em y para 20 tanto para SP quanto para Pará, aí sim ok. 😊

Essa comparação mostra o gasto total, não o per capita de um estado e de outro, o que já era de se esperar. A questão da escala faz toda a diferença de acordo com o que queremos usar / verificar.

Desafio 04: um único gráfico com um único ano do Pará e de São Paulo. Cuidado com as cores!

Desafio 05: escolha dois estados. pegue a população desses dois estados nos últimos 12 meses e calcule o gasto por pessoa nesses estados. Faça o gráfico de linhas com uma linha para cada 12 meses desses estados com o seu gasto por pessoa.