



- **O que vamos aprender:**
 - utilizar a biblioteca Pandas - a mais utilizada no mundo para análise de dados
 - carregar dados de um arquivo Excel
 - fazer análise exploratória nos dados
 - gerar estatísticas das colunas quantitativas
 - gerar gráficos interativos
- **Projeto da aula:**
 - Realizar uma análise de dados sobre uma base de 70.000 linhas de uma rede de lojas de venda de Açai.

Carregando os dados do arquivo Excel

Importando a biblioteca

A biblioteca **Pandas** já vem pré-instalada no Anaconda, então só precisamos importá-la.

```
In [1]: import pandas as pd
```

```
In [2]: # lendo os dados (nesse código, o arquivo Excel precisa estar na mesma pasta)
dados = pd.read_excel("vendas.xlsx")
```

Análise Exploratória

Verificando se os dados foram carregados corretamente

Para verificar se os dados foram carregados corretamente, podemos utilizar dois métodos do Pandas:

- **head():** mostra as primeiras linhas do conjunto de dados
- **tail():** mostra as últimas linhas do conjunto de dados

```
In [3]: dados.head()
```

```
Out[3]:
```

	id_pedido	data	loja	cidade	estado	regiao	tamanho	local_consumo	preco	forma_pagamento	ano_m
0	PED1	2021-06-21	Loja 4	Santos	São Paulo	Sudeste	200ml	Delivery	5	Crédito	2021-

1	PED2	2021-04-17	Loja 2	Niterói	Rio de Janeiro	Sudeste	200ml	Consumo no local	5		Pix	2021-
2	PED3	2022-12-12	Loja 1	Fortaleza	Ceará	Nordeste	500ml	Delivery	9		Pix	2022-
3	PED4	2022-03-09	Loja 2	Niterói	Rio de Janeiro	Sudeste	300ml	Consumo no local	7		Débito	2022-
4	PED5	2022-11-28	Loja 4	Santos	São Paulo	Sudeste	700ml	Delivery	11		Débito	2022-

In [4]: `dados.tail()`

Out[4]:

	id_pedido	data	loja	cidade	estado	regiao	tamanho	local_consumo	preco	forma_pagamento
69995	PED69996	2020-12-03	Loja 5	São Paulo	São Paulo	Sudeste	300ml	Consumo no local	7	Dinheiro
69996	PED69997	2021-06-01	Loja 6	Florianópolis	Santa Catarina	Sul	700ml	Consumo no local	11	Pix
69997	PED69998	2022-07-14	Loja 6	Florianópolis	Santa Catarina	Sul	700ml	Delivery	11	Dinheiro
69998	PED69999	2021-06-01	Loja 6	Florianópolis	Santa Catarina	Sul	500ml	Consumo no local	9	Pix
69999	PED70000	2022-07-10	Loja 2	Niterói	Rio de Janeiro	Sudeste	700ml	Consumo no local	11	Dinheiro

Quantidade de linhas e colunas

- Podemos usar a propriedade **shape** para verificar a quantidade de linhas e colunas. O primeiro valor é a quantidade de **linhas** e o segundo a de **colunas**.

In [5]: `dados.shape`

Out[5]: `(70000, 11)`

Informações sobre as colunas

O Pandas tem um método muito poderoso para gerar informações importantes sobre o conjunto de dados:**info()**.

In [6]: `dados.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id_pedido              70000 non-null object
1   data                   70000 non-null datetime64[ns]
2   loja                   70000 non-null object
3   cidade                 70000 non-null object
4   estado                 70000 non-null object
5   regiao                 70000 non-null object
6   tamanho                70000 non-null object
7   local_consumo          70000 non-null object
8   preco                  70000 non-null int64
```

```
9      forma_pagamento    70000 non-null object
10     ano_mes              70000 non-null object
dtypes: datetime64[ns](1), int64(1), object(9)
memory usage: 5.9+ MB
```

Gerando estatísticas

O método **describe()** gera estatísticas sobre todas as colunas quantitativas.

```
In [7]: dados.describe()
```

```
Out[7]:
```

	preco
count	70000.000000
mean	9.009571
std	2.831874
min	5.000000
25%	7.000000
50%	9.000000
75%	11.000000
max	13.000000

Acessando uma coluna

Para acessar uma coluna, podemos utilizar a notação de colchetes, passando o nome da coluna desejada.

Caso o nome da coluna não possua espaços em branco de nem caracteres especiais, podemos acessar também com a notação de ponto.

```
In [8]: dados['loja']
```

```
Out[8]:
```

0	Loja 4
1	Loja 2
2	Loja 1
3	Loja 2
4	Loja 4
...	
69995	Loja 5
69996	Loja 6
69997	Loja 6
69998	Loja 6
69999	Loja 2

Name: loja, Length: 70000, dtype: object

```
In [9]: dados.loja
```

```
Out[9]:
```

0	Loja 4
1	Loja 2
2	Loja 1
3	Loja 2
4	Loja 4
...	
69995	Loja 5
69996	Loja 6
69997	Loja 6
69998	Loja 6

```
69999      Loja 2
Name: loja, Length: 70000, dtype: object
```

Obtendo os únicos de uma coluna

Para obter os valores únicos de uma coluna, utilizamos o método **unique()**.

```
In [10]: dados['loja'].unique()
```

```
Out[10]: array(['Loja 4', 'Loja 2', 'Loja 1', 'Loja 6', 'Loja 3', 'Loja 5'],
          dtype=object)
```

Contagem de valores

Para fazer a contagem de valores de uma coluna, podemos utilizar o método **value_counts()**.

Podemos obter também o valor relativo, utilizando o parâmetro **normalize=True**.

```
In [11]: dados['loja'].value_counts()
```

```
Out[11]: Loja 6      16648
          Loja 3      12367
          Loja 1      12344
          Loja 5      12177
          Loja 2       8318
          Loja 4       8146
          Name: loja, dtype: int64
```

```
In [12]: dados['loja'].value_counts(normalize=True)
```

```
Out[12]: Loja 6      0.237829
          Loja 3      0.176671
          Loja 1      0.176343
          Loja 5      0.173957
          Loja 2      0.118829
          Loja 4      0.116371
          Name: loja, dtype: float64
```

Agrupando dados

O método **groupby()** realiza o agrupamento de dados por determinada coluna.

Sempre que utilizarmos o **groupby()**, precisamos definir o **método de agregação** que será usado.

```
In [13]: # faturamento por loja
          dados.groupby('loja').sum()
```

```
Out[13]:
```

	preco
loja	
Loja 1	111042
Loja 2	74700
Loja 3	111709
Loja 4	73598
Loja 5	109393
Loja 6	150228

```
In [14]: # média de faturament por loja (ticket médio)
dados.groupby('loja').mean()
```

```
Out[14]:
```

	preco
loja	
Loja 1	8.995625
Loja 2	8.980524
Loja 3	9.032829
Loja 4	9.034864
Loja 5	8.983576
Loja 6	9.023787

Gráficos

Instalando e importando a biblioteca de gráficos

Para gerar os gráficos vamos utilizar a biblioteca **Plotly Express**.

```
In [15]: !pip install plotly_express
```

```
Requirement already satisfied: plotly_express in c:\users\vinil\anaconda3\envs\datascien
ce\lib\site-packages (0.4.1)
Requirement already satisfied: numpy>=1.11 in c:\users\vinil\anaconda3\envs\datascience
\lib\site-packages (from plotly_express) (1.23.3)
Requirement already satisfied: pandas>=0.20.0 in c:\users\vinil\anaconda3\envs\datascien
ce\lib\site-packages (from plotly_express) (1.4.4)
Requirement already satisfied: scipy>=0.18 in c:\users\vinil\anaconda3\envs\datascience
\lib\site-packages (from plotly_express) (1.9.3)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\vinil\anaconda3\envs\datas
cience\lib\site-packages (from plotly_express) (0.13.5)
Requirement already satisfied: patsy>=0.5 in c:\users\vinil\anaconda3\envs\datascience\l
ib\site-packages (from plotly_express) (0.5.3)
Requirement already satisfied: plotly>=4.1.0 in c:\users\vinil\anaconda3\envs\datascienc
e\lib\site-packages (from plotly_express) (5.11.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\vinil\anaconda3\envs\d
atascience\lib\site-packages (from pandas>=0.20.0->plotly_express) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\vinil\anaconda3\envs\datascience
\lib\site-packages (from pandas>=0.20.0->plotly_express) (2022.1)
Requirement already satisfied: six in c:\users\vinil\anaconda3\envs\datascience\lib\site
-packages (from patsy>=0.5->plotly_express) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\vinil\anaconda3\envs\datascie
nce\lib\site-packages (from plotly>=4.1.0->plotly_express) (8.1.0)
Requirement already satisfied: packaging>=21.3 in c:\users\vinil\anaconda3\envs\datascie
nce\lib\site-packages (from statsmodels>=0.9.0->plotly_express) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\vinil\anaconda3\envs
\datascience\lib\site-packages (from packaging>=21.3->statsmodels>=0.9.0->plotly_expres
s) (3.0.9)
```

```
In [16]: import plotly_express as px
```

Contagem de pedidos por loja

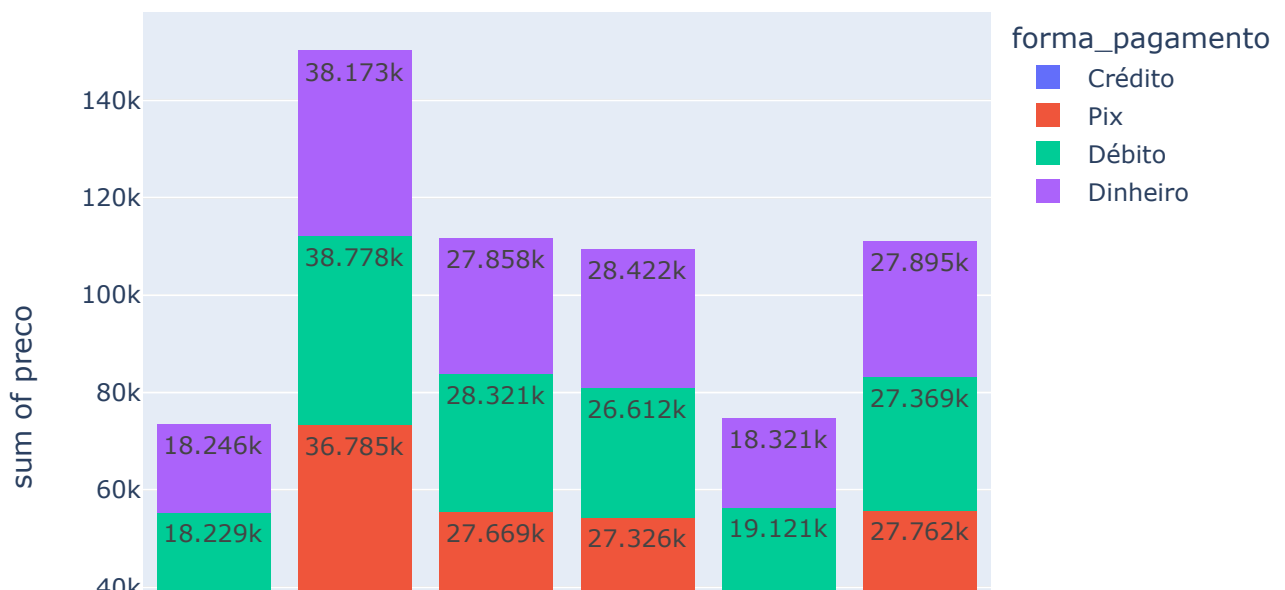
```
In [17]: px.histogram(dados, x="loja", color="regiao", text_auto=True)
```

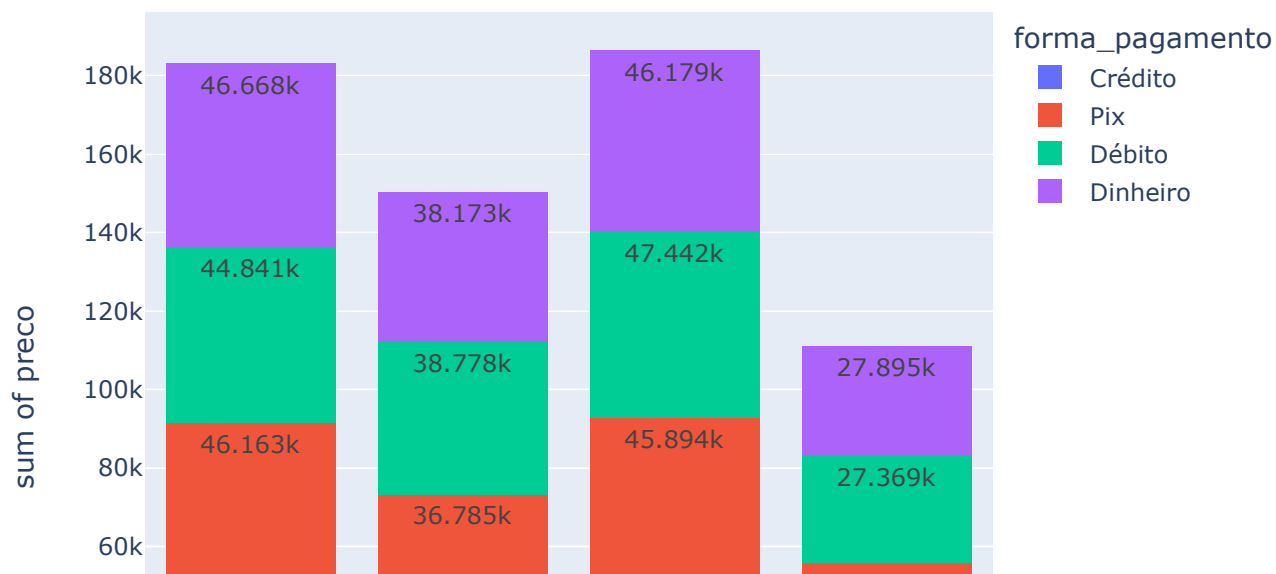
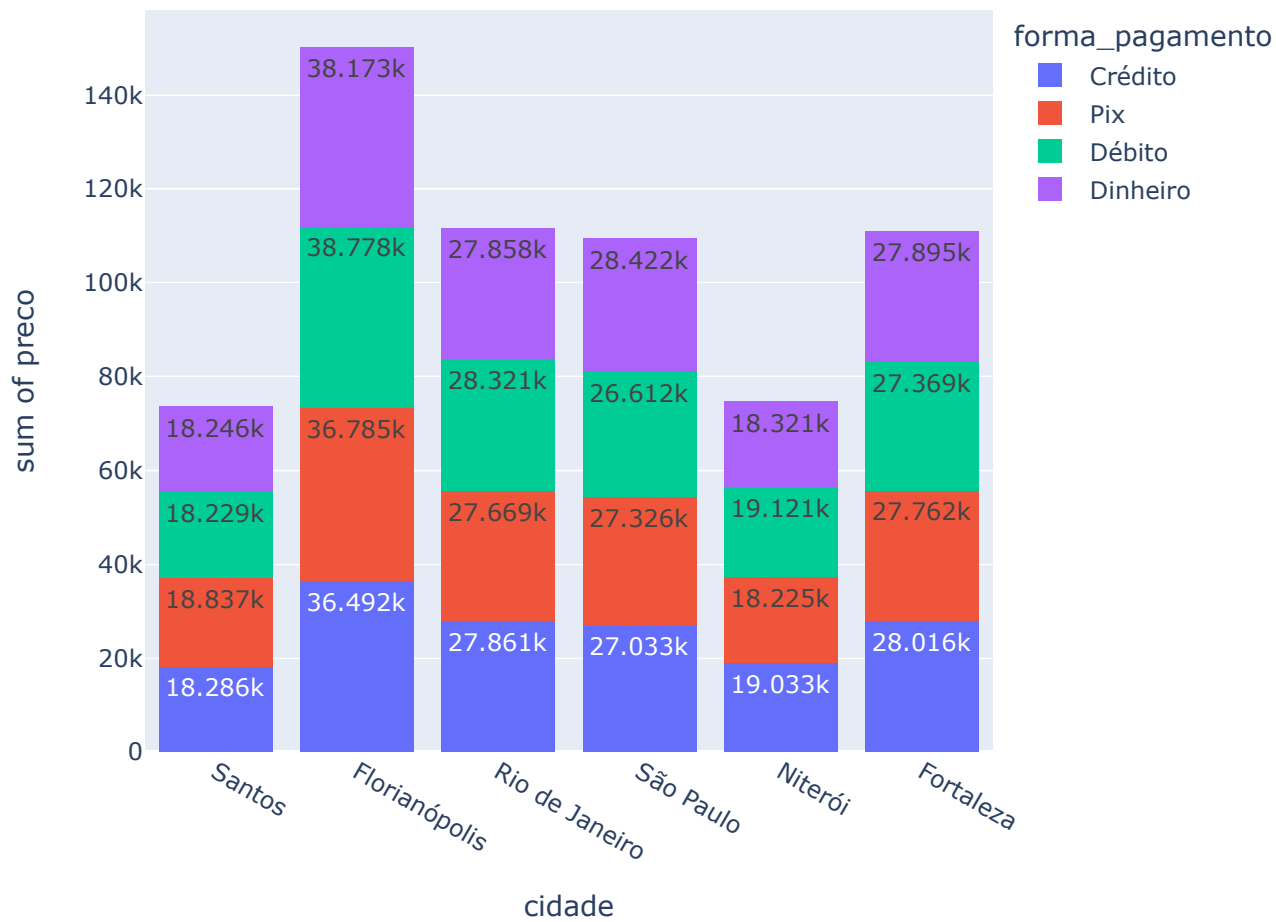
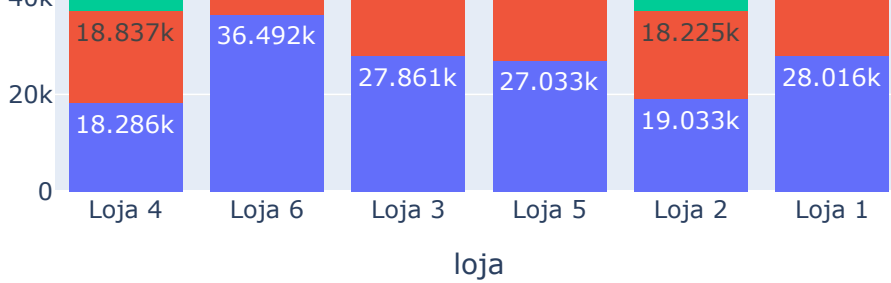


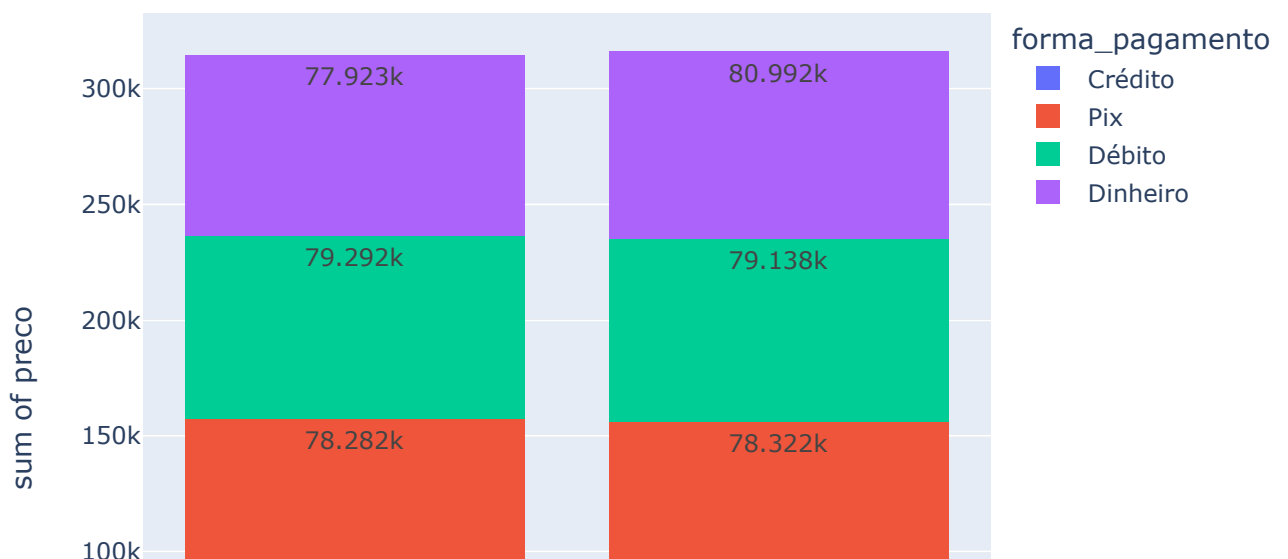
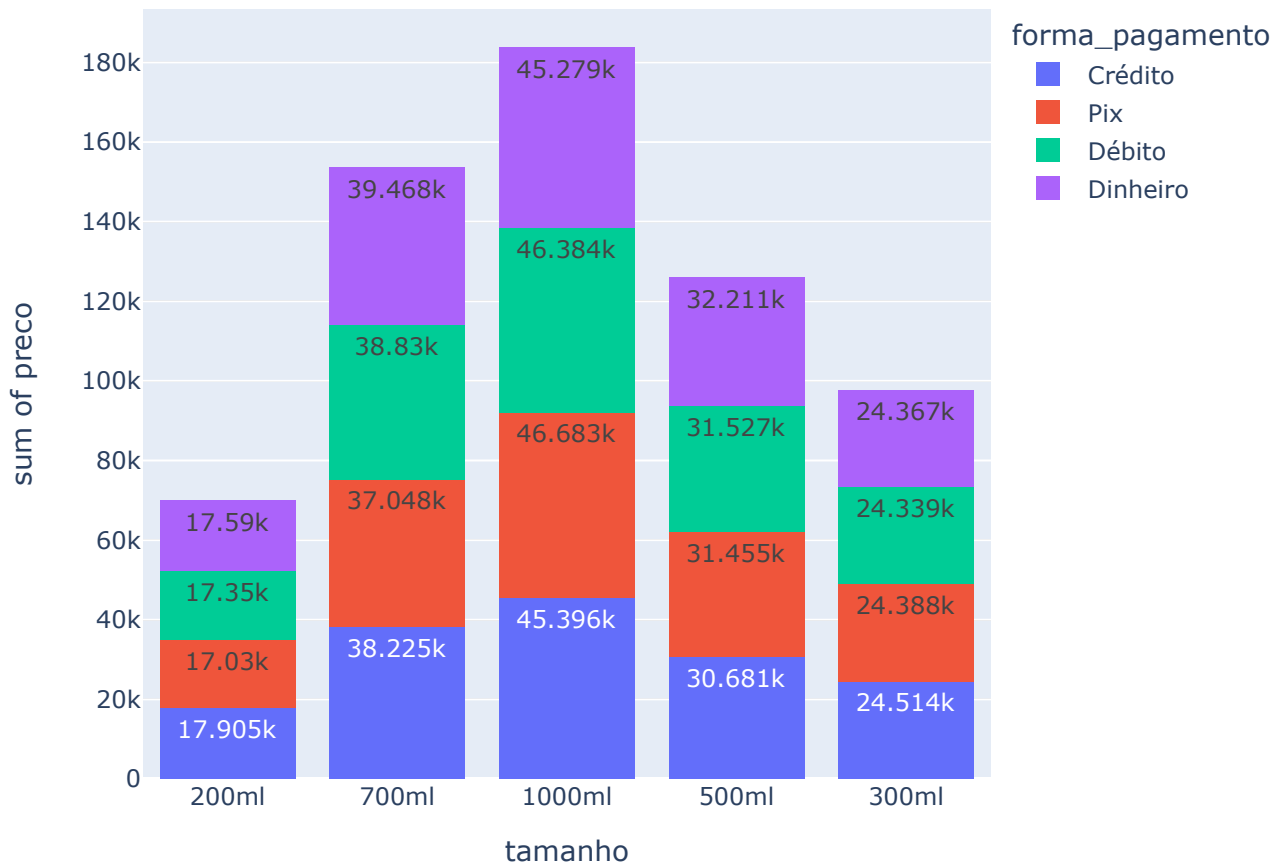
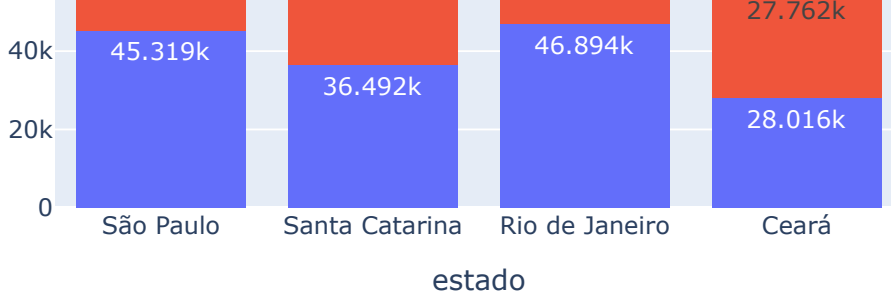
Criando múltiplos gráficos e gerando seus respectivos arquivos HTML

```
In [18]: columnas = ['loja', 'cidade', 'estado', 'tamanho', 'local_consumo']

for coluna in columnas:
    fig = px.histogram(dados, x=coluna, y='preco', color='forma_pagamento', text_auto=True)
    fig.write_html(f"faturamento por {coluna}.html")
    fig.show()
```







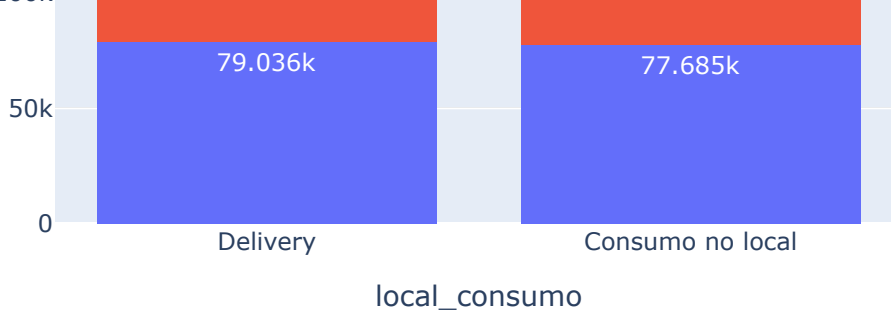


Gráfico animado

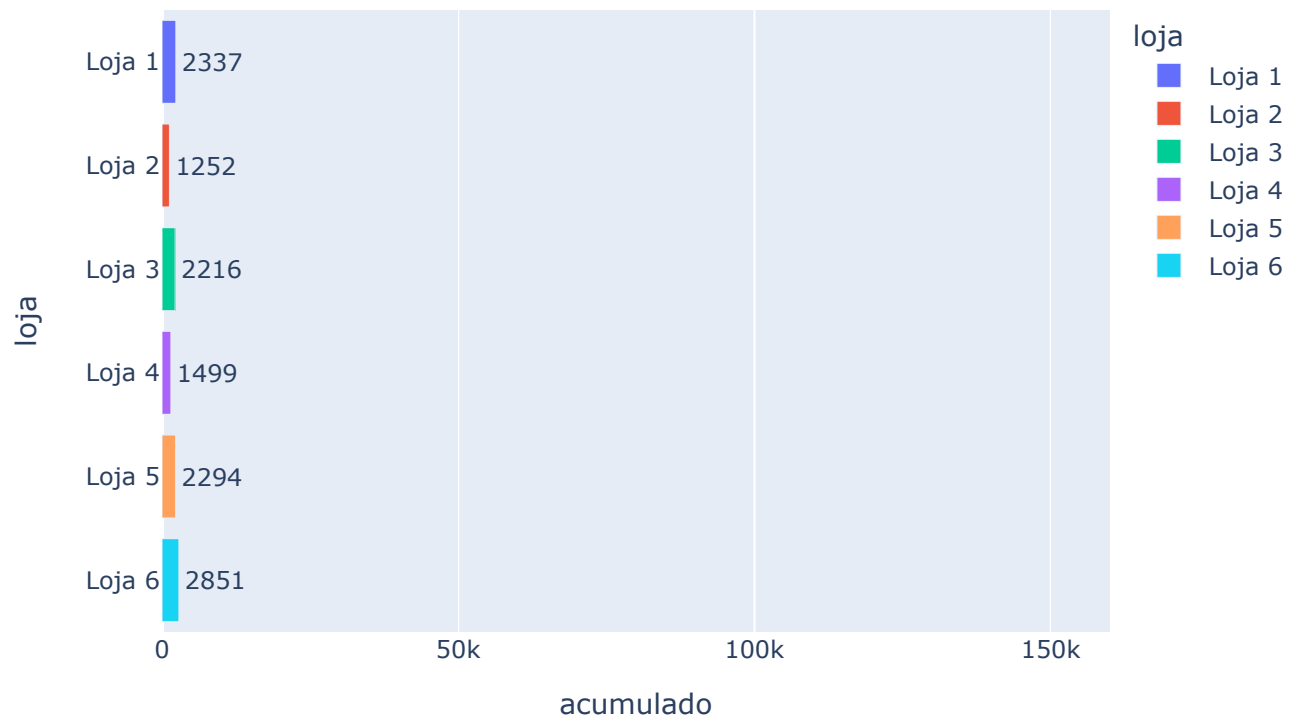
```
In [19]: # agrupando os dados
agrupado = dados.groupby(['loja', 'ano_mes']).sum()

#resetando os índices
agrupado.reset_index(inplace=True)

# criando uma coluna com o valor acumulado
agrupado['acumulado'] = agrupado.groupby('loja').cumsum()
```

```
In [20]: # gerando o gráfico
fig = px.bar(agrupado,
             x='acumulado',
             y="loja",
             color='loja',
             text_auto=True,
             range_x=[0,160000],
             animation_frame='ano_mes')

fig.show()
```



```
In [21]: # exportando o gráfico para um arquivo  
fig.write_html('grafico_animado.html')
```

PARABÉNS!

Mais um projeto concluído!

QUER IR ALÉM?

As inscrições para a **Formação Expert em Python** da Empowerdata estão abertas!

