- **O que vamos aprender:**
  - utilizar a bilioteca 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 Açaí.

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	ar
0	PED1994	2020-01-01	Loja 4	Santos	São Paulo	Sudeste	300ml	Consumo no local	5	Dinheiro	2

1	PED2246	2020-01-01	Loja 6	Florianópolis	Santa Catarina	Sul	500ml	Consumo no local	11	Débito	2
2	PED3876	2020-01-01	Loja 3	Rio de Janeiro	Rio de Janeiro	Sudeste	300ml	Delivery	7	Crédito	2
3	PED4352	2020-01-01	Loja 1	Fortaleza	Ceará	Nordeste	1000ml	Consumo no local	7	Débito	2
4	PED8633	2020-01-01	Loja 5	São Paulo	São Paulo	Sudeste	200ml	Delivery	9	Crédito	2

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

Out[4]:

	id_pedido	data	loja	cidade	estado	regiao	tamanho	local_consumo	preco	forma_pagament
69995	PED67084	2022-12-31	Loja 6	Florianópolis	Santa Catarina	Sul	500ml	Consumo no local	11	Crédit
69996	PED67857	2022-12-31	Loja 3	Rio de Janeiro	Rio de Janeiro	Sudeste	200ml	Consumo no local	7	Pi
69997	PED69171	2022-12-31	Loja 4	Santos	São Paulo	Sudeste	500ml	Consumo no local	5	Dinheir
69998	PED69229	2022-12-31	Loja 4	Santos	São Paulo	Sudeste	300ml	Consumo no local	9	Pi
69999	PED69356	2022-12-31	Loja 1	Fortaleza	Ceará	Nordeste	300ml	Delivery	9	Pi

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	8.355200
std	2.653061
min	5.000000
25%	7.000000
50%	7.000000
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 6
2         Loja 3
3         Loja 1
4         Loja 5
           ...
69995     Loja 6
69996     Loja 3
69997     Loja 4
69998     Loja 4
69999     Loja 1
Name: loja, Length: 70000, dtype: object
```

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

Out[9]:
```
0         Loja 4
1         Loja 6
2         Loja 3
3         Loja 1
4         Loja 5
           ...
69995     Loja 6
69996     Loja 3
69997     Loja 4
69998     Loja 4
```

```
69999    Loja 1
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 6', 'Loja 3', 'Loja 1', 'Loja 5', 'Loja 2'],
      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 4    13483
Loja 6    13075
Loja 1    12344
Loja 5    12177
Loja 3    10603
Loja 2     8318
Name: loja, dtype: int64
```

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

Out[12]:
```
Loja 4    0.192614
Loja 6    0.186786
Loja 1    0.176343
Loja 5    0.173957
Loja 3    0.151471
Loja 2    0.118829
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').preco.sum()
```

Out[13]:
```
loja
Loja 1    103162
Loja 2     69592
Loja 3     88357
Loja 4    112379
Loja 5    102189
Loja 6    109185
Name: preco, dtype: int64
```

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

Out[14]:
```
loja
Loja 1    8.357259
Loja 2    8.366434
```

```
Loja 3    8.333208
Loja 4    8.334866
Loja 5    8.391968
Loja 6    8.350669
Name: preco, dtype: float64
```

Gráficos

Instalando e importando a biblioteca de gráficos

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

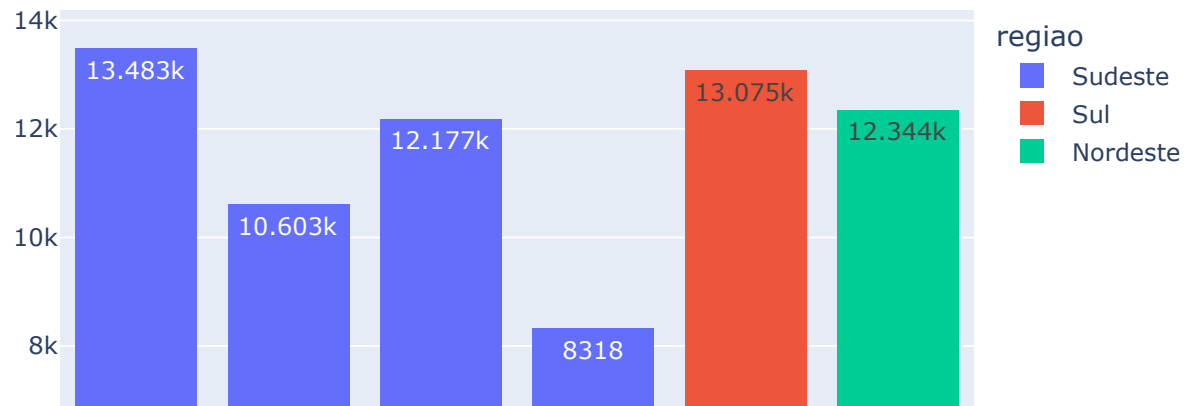
In [18]:
```
!pip install plotly_express
```

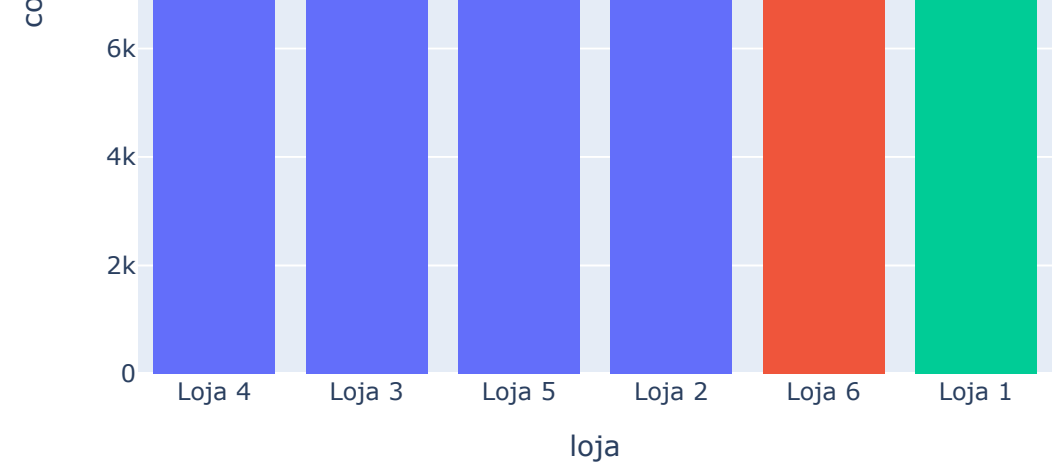
```
Requirement already satisfied: plotly_express in c:\users\pietro sales\anaconda3\lib\site-packages (0.4.1)
Requirement already satisfied: plotly>=4.1.0 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (5.9.0)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (0.13.5)
Requirement already satisfied: numpy>=1.11 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (1.23.5)
Requirement already satisfied: pandas>=0.20.0 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (1.5.3)
Requirement already satisfied: scipy>=0.18 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (1.10.0)
Requirement already satisfied: patsy>=0.5 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly_express) (0.5.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\pietro sales\anaconda3\lib\site-packages (from pandas>=0.20.0->plotly_express) (2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\pietro sales\anaconda3\lib\site-packages (from pandas>=0.20.0->plotly_express) (2.8.2)
Requirement already satisfied: six in c:\users\pietro sales\anaconda3\lib\site-packages (from patsy>=0.5->plotly_express) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\pietro sales\anaconda3\lib\site-packages (from plotly>=4.1.0->plotly_express) (8.0.1)
Requirement already satisfied: packaging>=21.3 in c:\users\pietro sales\anaconda3\lib\site-packages (from statsmodels>=0.9.0->plotly_express) (22.0)
```

In [19]:
```
import plotly_express as px
```

Contagem de pedidos por loja

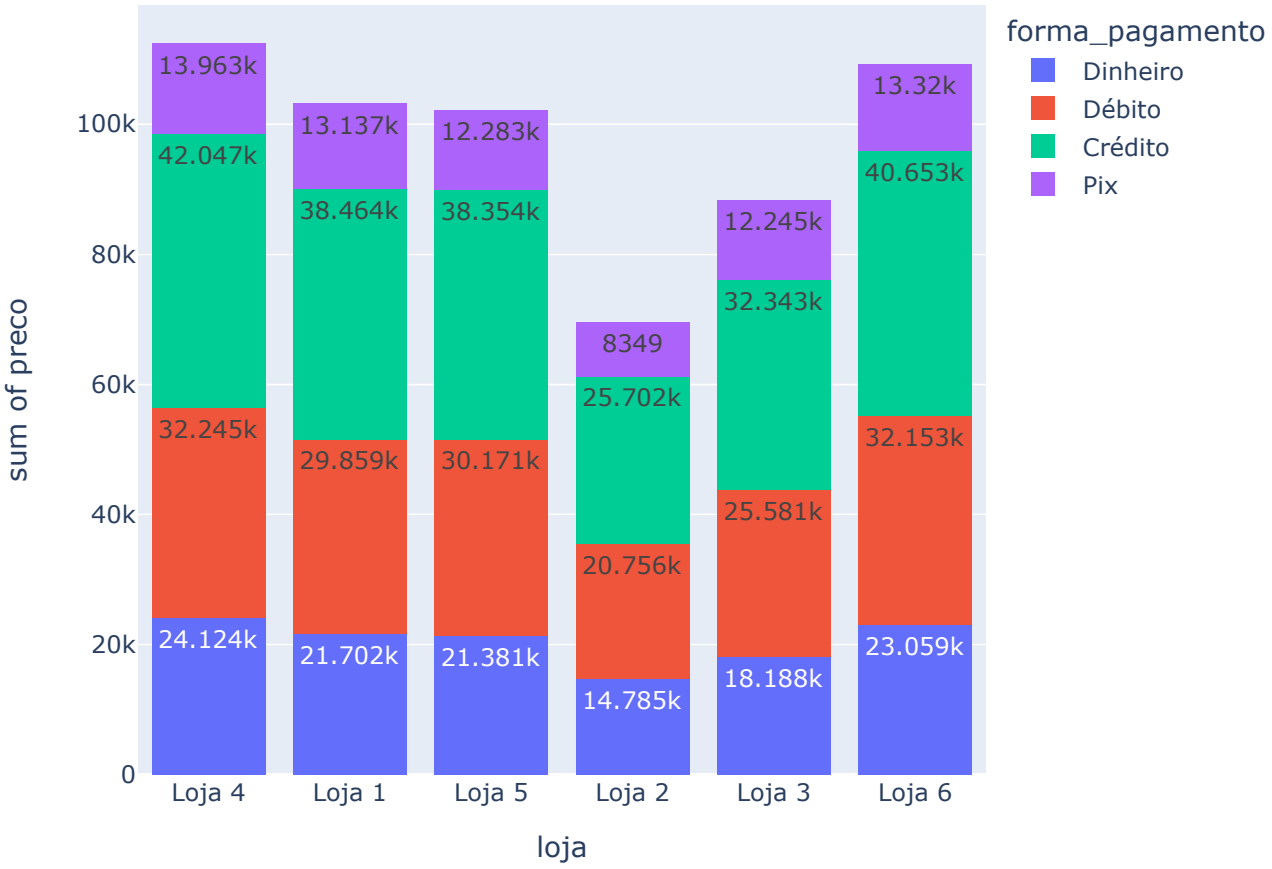
In [20]:
```
px.histogram(dados, x="loja", color="regiao", text_auto=True)
```

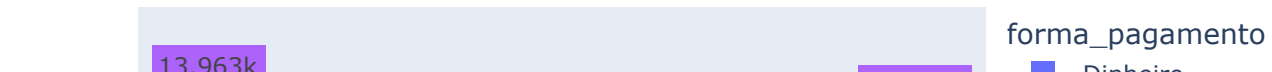


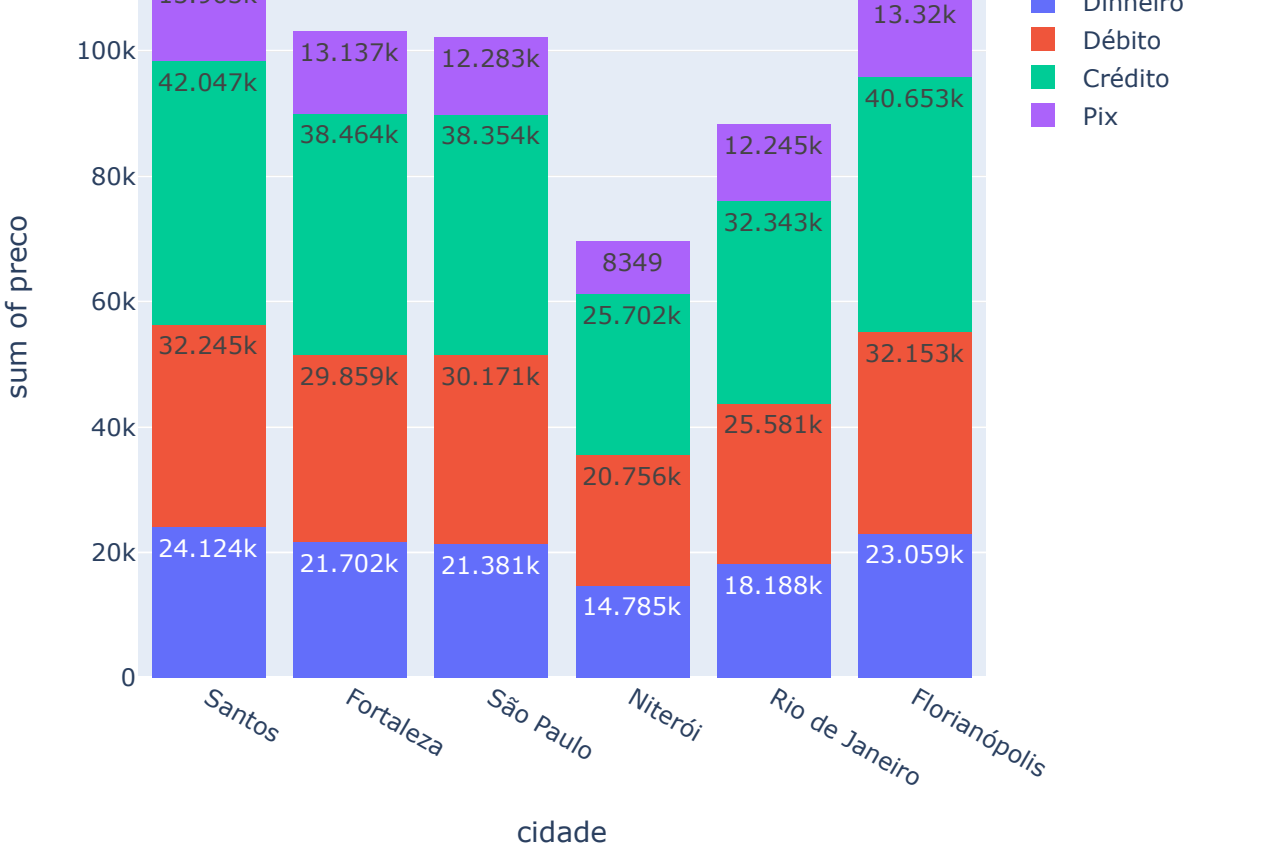


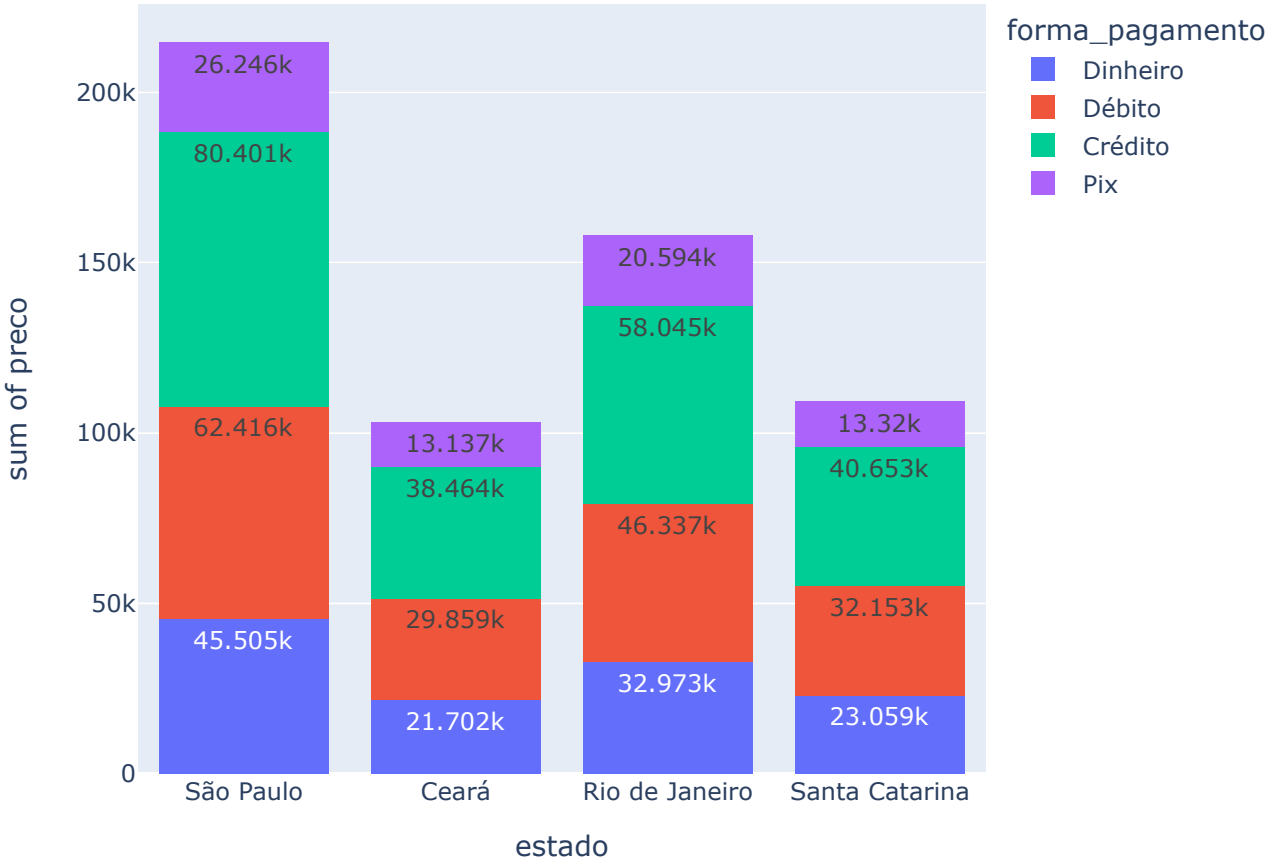
Criando múltiplos gráficos

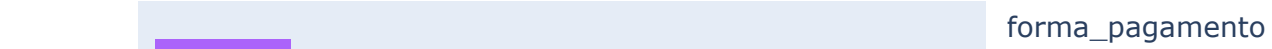
In [22]:
```
colunas = ['loja', 'cidade', 'estado', 'tamanho', 'local_consumo']

for coluna in colunas:
    fig = px.histogram(dados, x=coluna, y='preco', color='forma_pagamento', text_auto=Tr
    fig.show()
```

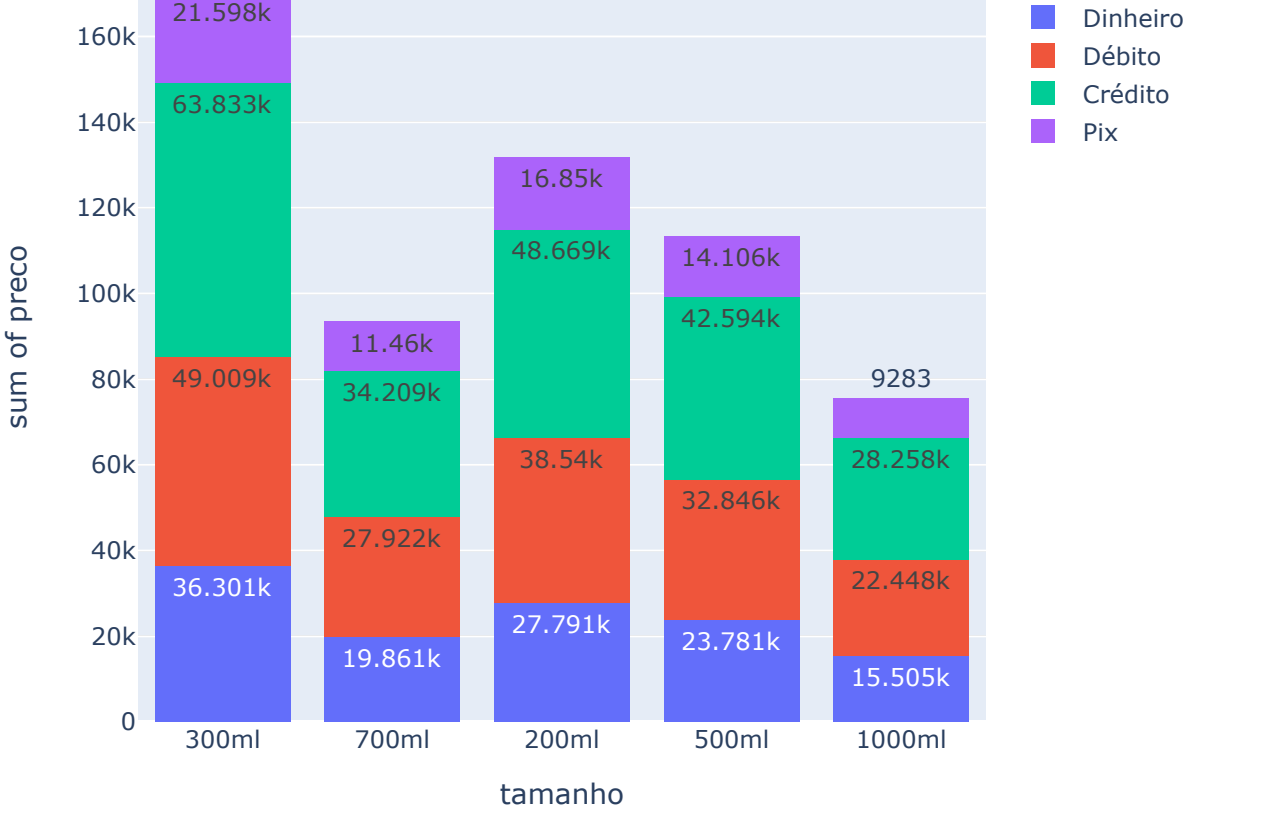


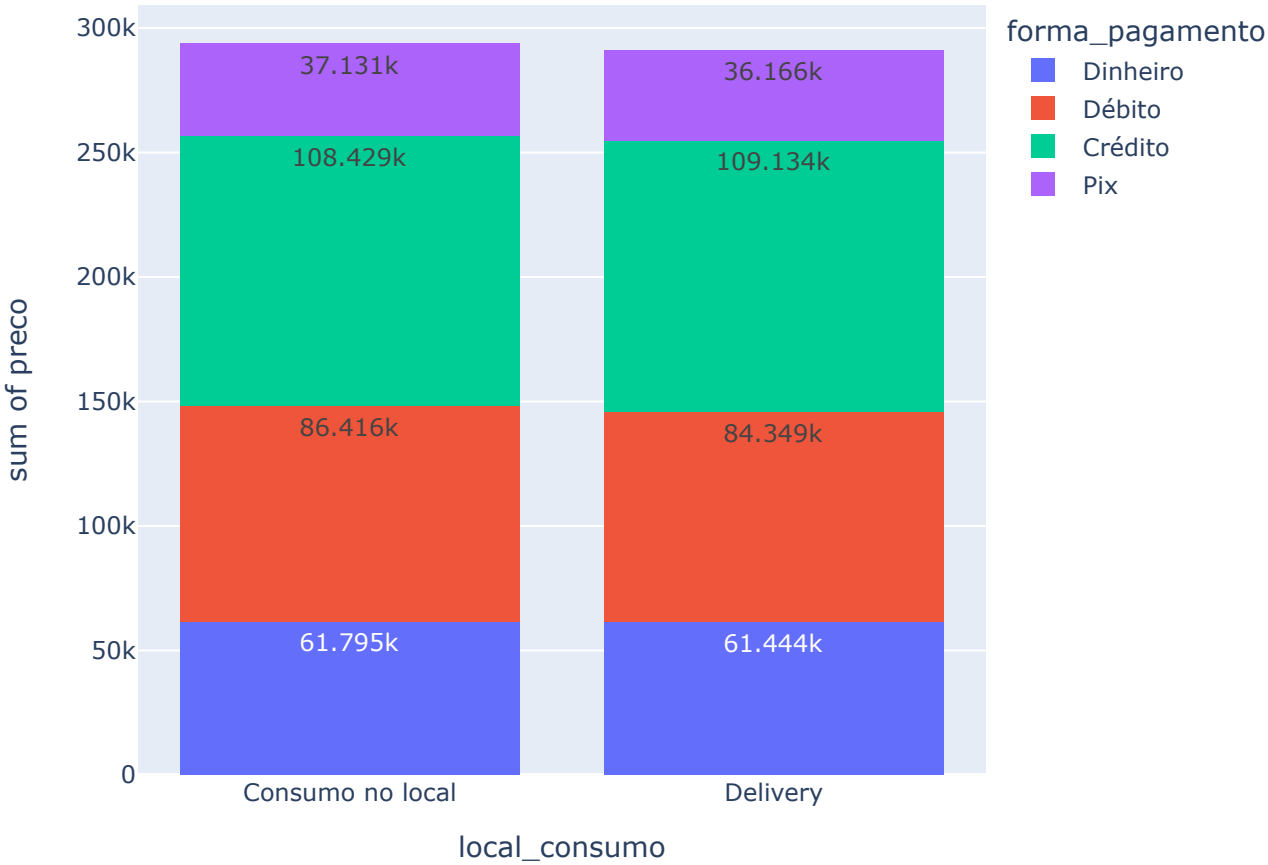



forma_pagamento
Dinheiro
Débito
Crédito
Pix
sum of preco
0
20k
40k
60k
80k
100k
24.124k
32.245k
42.047k
13.137k
21.702k
29.859k
38.464k
13.137k
21.381k
30.171k
38.354k
12.283k
14.785k
20.756k
25.702k
8349
18.188k
25.581k
32.343k
12.245k
23.059k
32.153k
40.653k
13.32k
Santos
Fortaleza
São Paulo
Niterói
Rio de Janeiro
Florianópolis
cidade


forma_pagamento
Dinheiro
Débito
Crédito
Pix
sum of preco
0
50k
100k
150k
200k
45.505k
62.416k
80.401k
26.246k
21.702k
29.859k
38.464k
13.137k
32.973k
46.337k
58.045k
20.594k
23.059k
32.153k
40.653k
13.32k
São Paulo
Ceará
Rio de Janeiro
Santa Catarina
estado


forma_pagamento


sum of preco
tamanho
300ml
700ml
200ml
500ml
1000ml
0
20k
40k
60k
80k
100k
120k
140k
160k
21.598k
63.833k
49.009k
36.301k
11.46k
34.209k
27.922k
19.861k
16.85k
48.669k
38.54k
27.791k
14.106k
42.594k
32.846k
23.781k
9283
28.258k
22.448k
15.505k
Dinheiro
Débito
Crédito
Pix


sum of preco
local_consumo
Consumo no local
Delivery
0
50k
100k
150k
200k
250k
300k
37.131k
108.429k
86.416k
61.795k
36.166k
109.134k
84.349k
61.444k
forma_pagamento
Dinheiro
Débito
Crédito
Pix