Relatório de análise exploratória de dados - vendas de café -

Objetivos: Realizar estudos de análise exploratória de dados, como: Quantidade de dados faltantes, tratamento dos dados faltantes, aplicação de estatísticas básicas e análise de correlação. Estes estudos foram realizados utilizando a linguagem de programação Python e as bibliotecas: Numpy, Matplotlib, Pandas, Seaborn e Scipy.

Apresentação do data frame

O data frame trata-se de dados de transações realizadas em cafeterias. Este tem 10.000 linhas e 8 colunas, totalizando 80.000 células de dados (válidos ou inválidos).

Descrição das variáveis, classificação e valores assumidos:

- **Transaction ID:** Este é o identificador das transações. Sempre presente e único. Variável qualitativa ordinal, do tipo string.
- **Item:** Nome do item adquirido. Variável qualitativa nominal, do tipo string e assume os seguintes valores: 'Coffee', 'Cake', 'Cookie', 'Salad', 'Smoothie', 'UNKNOWN', 'Sandwich', nan, ‘'ERROR', 'Juice', 'Tea'.
- **Quantity:** A quantidade do item comprado. Variável quantitativa discreta, do tipo string e assume os valores: '2', '4', '5', '3', '1', 'ERROR', 'UNKNOWN', nan.
- **Price Per Unit:** Preço da unidade do item. Variável quantitativa contínua, do tipo string e assume os seguintes valores:'2.0', '3.0', '1.0', '5.0', '4.0', '1.5', nan, 'ERROR', 'UNKNOWN'.
- **Total Spent:** Valor total da compra. Variável quantitativa contínua, do tipo string e assume os seguintes valores: '4.0', '12.0', 'ERROR', '10.0', '20.0', '9.0', '16.0', '15.0', '25.0', '8.0', '5.0', '3.0', '6.0', nan, 'UNKNOWN', '2.0', '1.0', '7.5', '4.5', '1.5'.
- **Payment method:** Método de pagamento utilizado na compra. Variável qualitativa nominal, do tipo string e assume os seguintes valores: 'Credit Card', 'Cash', 'UNKNOWN', 'Digital Wallet', 'ERROR', nan.

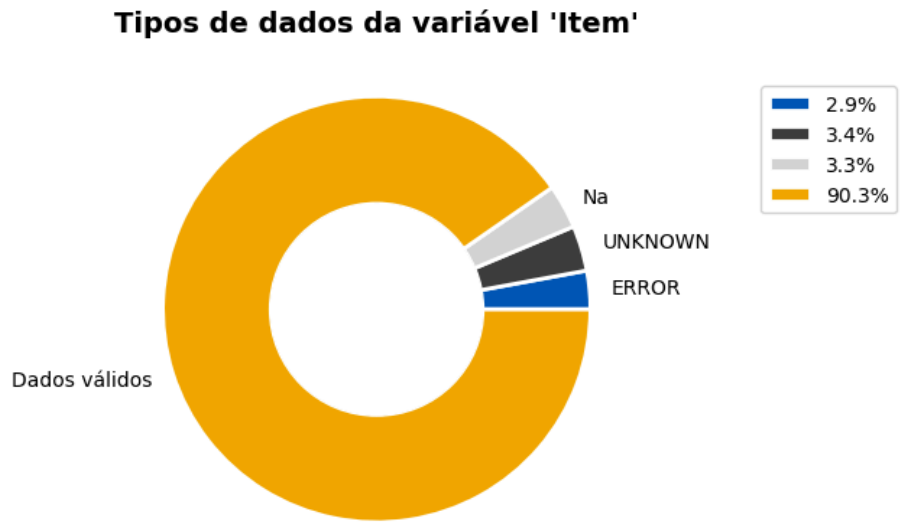
- **Location:** Local onde a transação ocorreu. Variável qualitativa nominal, do tipo string e assume os seguintes valores: 'Takeaway', 'In-store', 'UNKNOWN', nan, 'ERROR'.
- **Transaction Date:** Data em que a compra foi realizada. Variável do tipo qualitativa ordinal, do tipo string.

Proporção dos dados inválidos

São considerados dados inválidos: “ERROR”, “UNKNOWN”, NA. Cabe destacar que, a variável “Transaction ID” não contém dados inválidos e a variável “Transaction Date” contém dos dados inválidos apenas dados NA.

Item

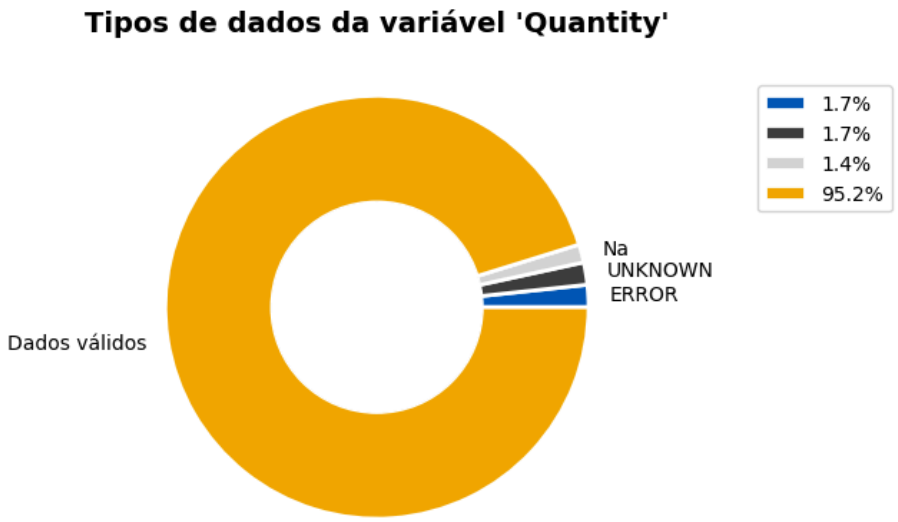
Esta variável tem aproximadamente 10% dos dados inválidos. Nenhum tratamento será aplicado a esta variável.



Todos os itens possuem grandes números de venda e o de maior venda é o suco.

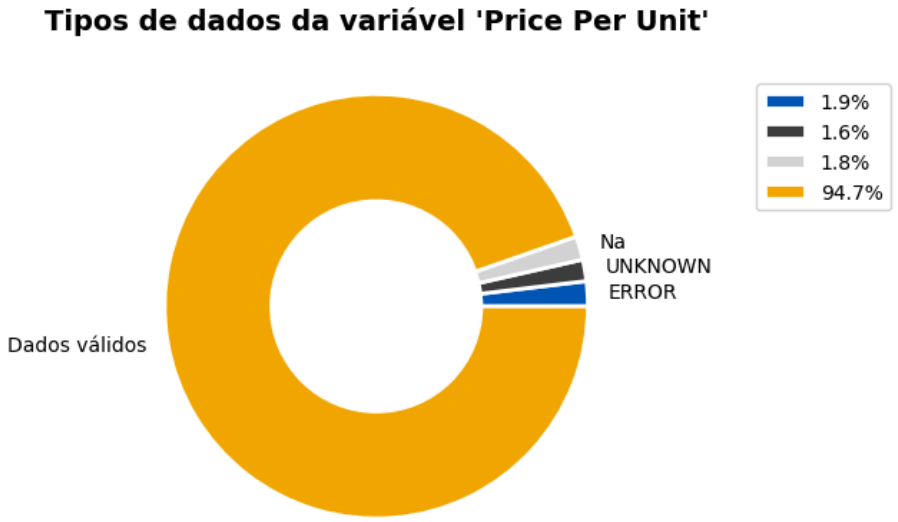
Quantity

Esta variável não é muito prejudicada pela presença dos dados inválidos, visto que somam 4,8%.



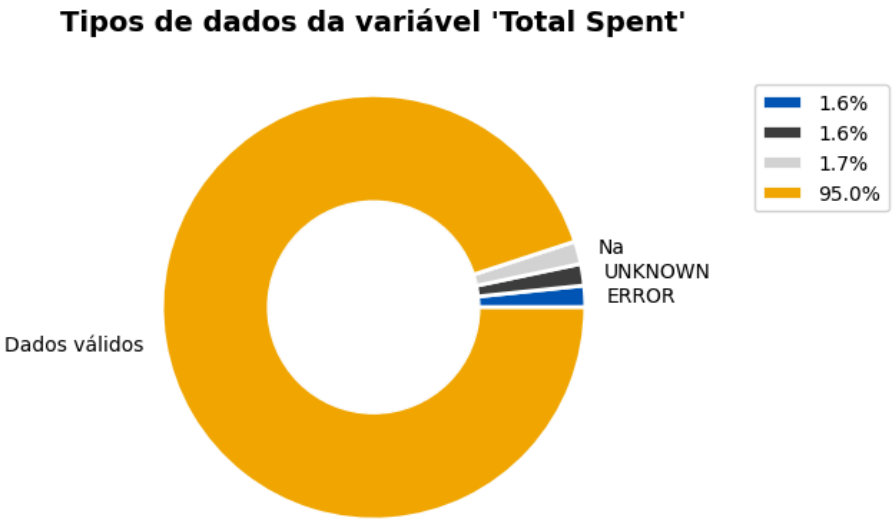
Price per Unit

Variável também pouco prejudicada, os dados inválidos somam 5,3%.



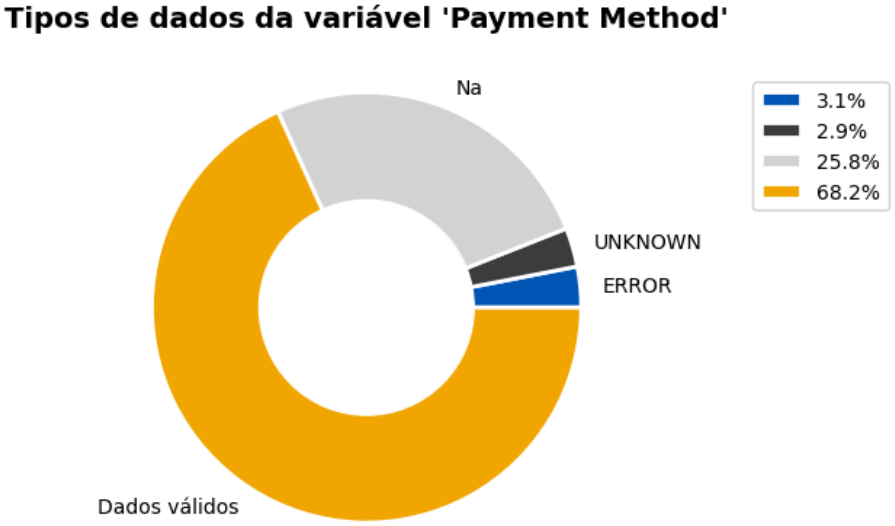
Total Spent

Variável também pouco prejudicada, os dados inválidos somam 4,9%.



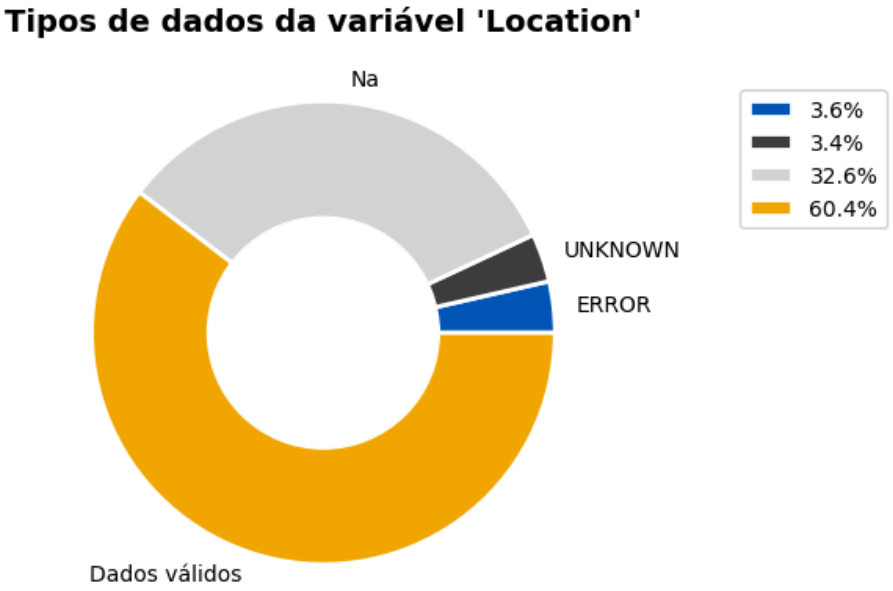
Payment Method

Uma das variáveis mais prejudicadas, visto que a porcentagem dos dados inválidos somam 32%. Isto pode prejudicar análises futuras.



Location

É a variável mais prejudicada, seus dados inválidos somam 40%.



Os locais da transação têm quantidades quase equivalentes.

Tratamento e transformação dos dados

Percebe-se que a natureza das variáveis ‘Quantity’, ‘Price per unit’ e ‘Total Spent’ no data frame, é do tipo string, contudo, são variáveis numéricas. Por isso, houve a transformação para o tipo numérico. Não há nenhum dado duplicado.

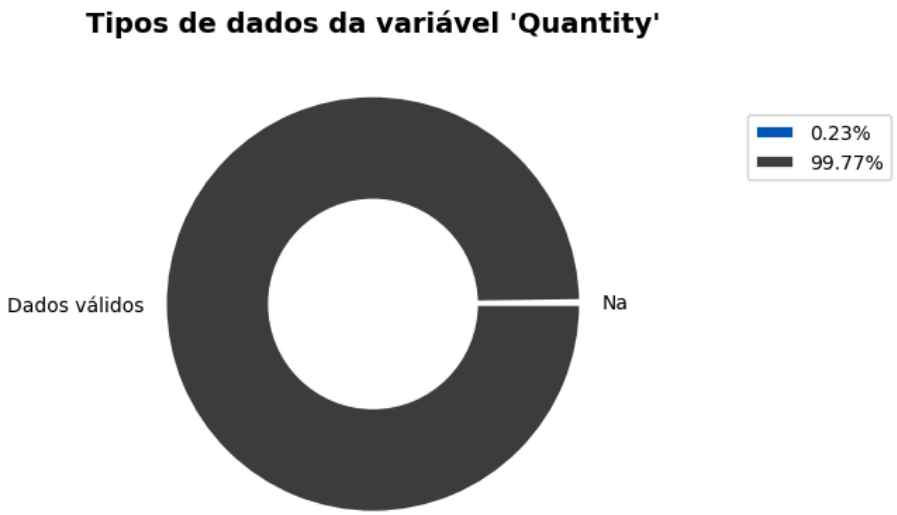
Para o tratamento dos dados, nota-se que entre as variáveis citadas anteriormente, há uma relação matemática, isto é, “Total Spent” é a multiplicação entre “Quantity” e “Price Per Unit”. Através desta relação, foi possível restaurar a maioria dos dados faltantes destas colunas. Além disso, podemos usufruir de outra relação, entre o preço unitário e o item. Isto é, numa transação onde possui-se o nome do item, então automaticamente saberemos o seu preço unitário e podemos reparar este dado caso esteja ausente. Por outro lado, se temos o preço unitário, podemos saber o nome do item, a não ser que o valor do preço unitário esteja associado a mais de um item, como nos casos dos itens: “Sandwich” e “Smoothie”, “Cake” e “Juice. Nestes casos, nada foi feito.

Para os dados inválidos que não foram possíveis serem reparados, apenas foram retirados da análise.

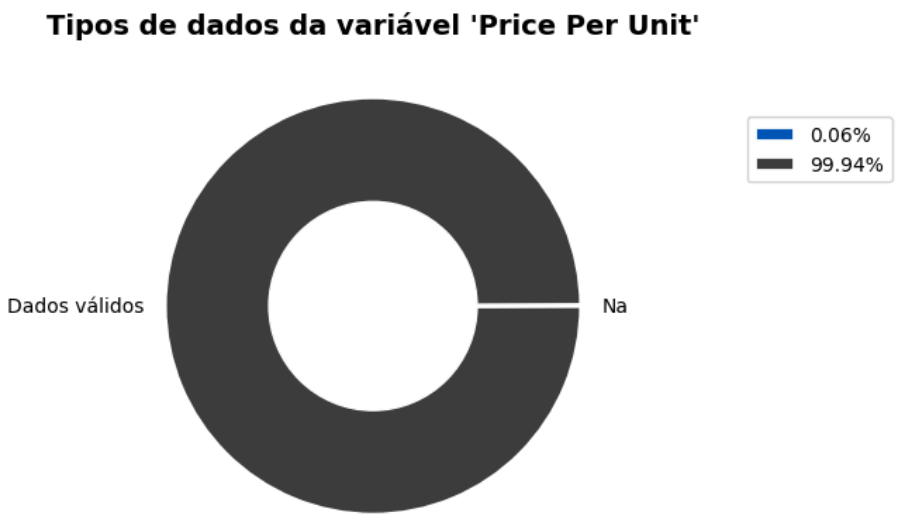
Proporção dos dados inválidos pós tratamento

Item

Quantity



Price Per Unit



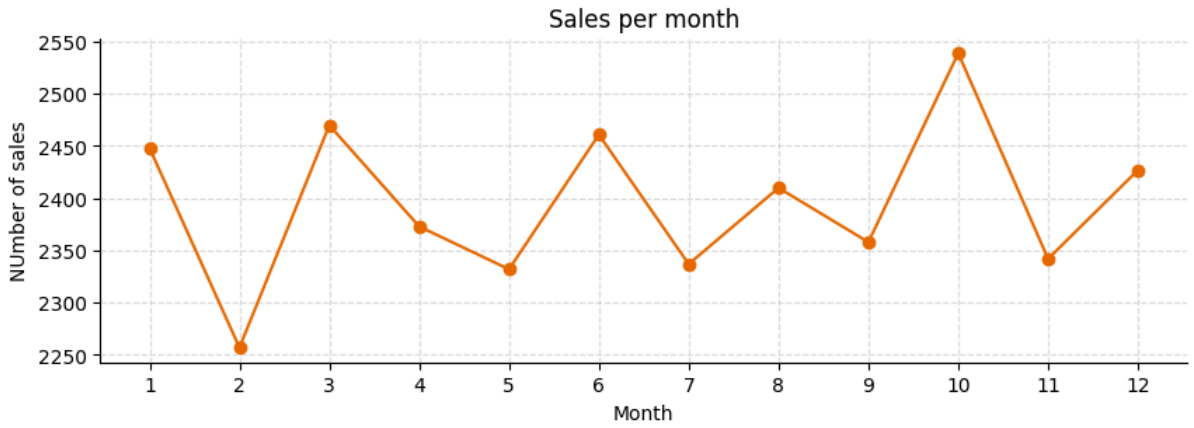
Total Spent

Análises Unidimensionais

Estatísticas descritivas

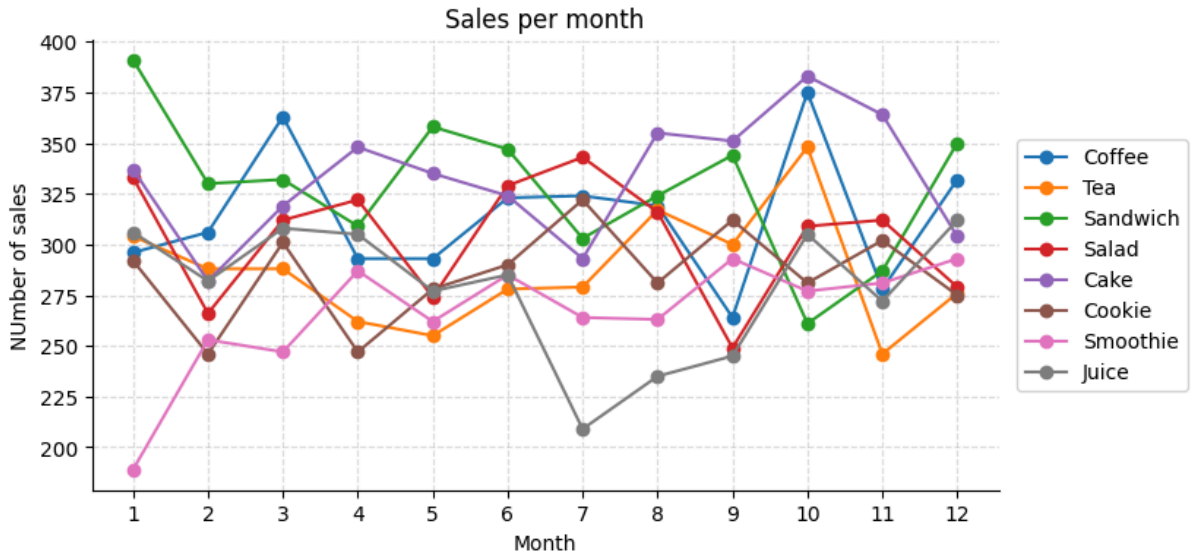
	Quantity	Price Per Unit	Total Spent	Transaction Date
count	9977.000000	9994.000000	9977.000000	9540
mean	3.024957	2.947018	8.930139	2023-07-01 23:00:31.698113280
min	1.000000	1.000000	1.000000	2023-01-01 00:00:00
25%	2.000000	2.000000	4.000000	2023-04-01 00:00:00
50%	3.000000	3.000000	8.000000	2023-07-02 00:00:00
75%	4.000000	4.000000	12.000000	2023-10-02 00:00:00
max	5.000000	5.000000	25.000000	2023-12-31 00:00:00
std	1.420395	1.280006	6.004921	NaN

Vendas totais por mês



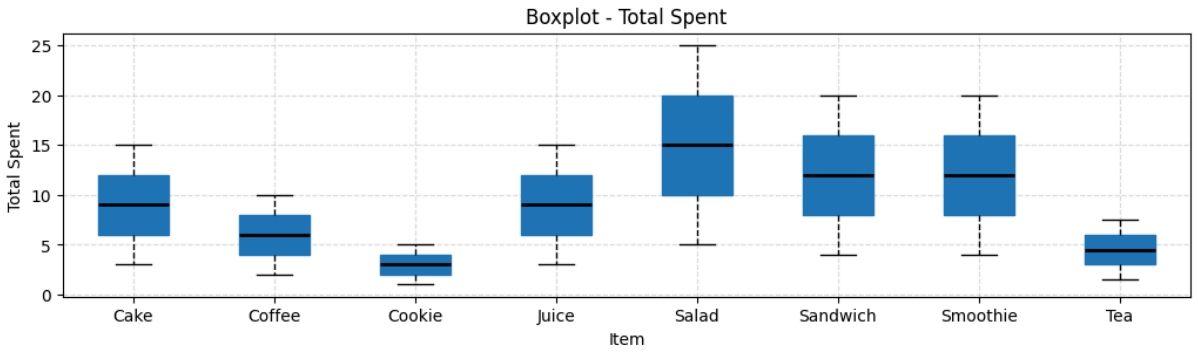
Nota-se que os meses de Outubro e Março tiveram a maior quantidade de vendas, o que pode dar um direcionamento para um maior investimento nestas temporadas. Além disso, cria-se um ponto de alerta para a motivação de baixas vendas no mês de fevereiro.

Vendas totais por mês e item



Este gráfico nos ajuda a entender qual o item mais ou menos vendido em determinado mês. Observa-se o comportamento do chá, que aparenta ter aumento em vendas no início do inverno. Por outro lado, pode-se investigar a motivação de uma queda de vendas do bolo no mês de Julho.

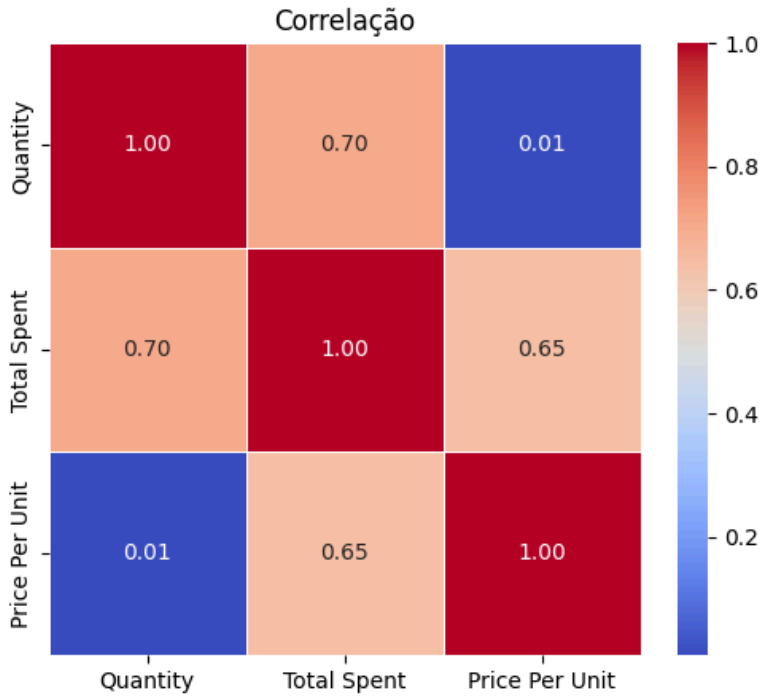
Box plot dos itens



Usando o *boxplot,* podemos notar a abrangência dos valores das transações de cada item. Averigua-se que a salada é o item que possui maior faixa de valor em sua compra. Além disso, observa-se que o cookie e o chá são itens que não tem uma grande faixa de valores, o que pode indicar que quando comprados, são comprados em poucas quantidades.

Análises Bidimensionais

Associação entre variáveis quantitativas - Correlação entre “Quantity”, “Total Spent”, “Price Per Unit”.



Como citado anteriormente, há uma relação matemática entre as três variáveis. Então, já era de se esperar uma alta correlação entre: “Quantity” e “Total Spent”, e “Price Per Unit” e “Total Spent”. A baixa correlação entre “Quantity” e “Price Per Unit” é explicada pela inexistência de relação entre as duas variáveis, visto que “Price Per Unit” não é determinada pela quantidade comprada na transação, ou seja, pela variável “Quantity”.

Associação entre variáveis qualitativas - Qui-quadrado de Pearson

Item ~ Payment method

```
Qui-Quadrado: 11.1592
p-valor: 0.6735
Graus de liberdade: 14
```

Location ~ Payment Method

```
Qui-Quadrado: 2.8320
p-valor: 0.2427
Graus de liberdade: 2
```

Location ~ Item

```
Qui-Quadrado: 6.5343
p-valor: 0.4789
Graus de liberdade: 7
```

Todas análises de associação entre as variáveis qualitativas, com uso do método de qui-quadrado de Pearson, obtiveram resposta negativa. Isto é, não há relação entre as variáveis, visto que o valor qui-quadrado é pequeno e o p-valor é grande em relação ao valor de confiança geralmente adotado, p = 0.05.

Tópicos sobre a transformação dos dados e reparo dos dados

Para a transformação dos dados para numéricos foi utilizada a função nativa da biblioteca Pandas, “to_numeric”. Cabe ressaltar que, caso haja erro durante a transformação, então a própria biblioteca põe como valor NA. Portanto, os dados inválidos “UNKNOWN” e “ERROR”, são transformados em NA.

```python
# Alteração dos dados de natureza numérica str --> int or float
df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')
df['Price Per Unit'] = pd.to_numeric(df['Price Per Unit'], errors='coerce')
df['Total Spent'] = pd.to_numeric(df['Total Spent'], errors='coerce')
# Mudança do tipo de dado da data da transação
df['Transaction Date'] = pd.to_datetime(df['Transaction Date'], errors='coerce')
print("Dados tranformados.")
```

Além disso, as datas foram transformadas para o objeto de data do pandas, para que permitisse melhor manipulação e análises sobre estes dados.

Usando as duas relações para reparar as células com dados inválidos, item e preço por unidade, e a relação matemática entre **preço por unidade**, **quantidade** e **preço total**, obtém-se o seguinte problema: Imagine que haja dois blocos de código, o primeiro, repara as linhas usando a primeira relação e o segundo bloco, repara as linhas usando a segunda relação. Suponha que exista no data frame uma linha com as variáveis **item** e **preço por unidade**, com os dados inválidos/ausentes. Ao executar o primeiro bloco de código, nada será feito, visto que ambas variáveis estarão ausentes, logo não podemos associá-las. Na segunda execução, como as variáveis **quantidade** e **preço total** estarão presentes, poderemos usar a relação matemática para encontrar o **preço por unidade**. Tendo reparado este dado, esta linha entra em condição para ser reparada pelo primeiro bloco de código, entretanto este bloco já foi executado. Portanto, a segunda execução gera linhas que podem ser reparadas na primeira execução, e vice-versa. Este é um problema de **imputação cíclica de dados,** em que o reparo de uma linha depende de um reparo aplicado anteriormente e vice-versa. Solução:

Função que repara os dados com o uso da relação entre **item e preço por unidade**.

```
[ ]
def repair_data_item_price(df):
  """
  Função que realiza o reparo dos dados através da relação item-preço_unitário e
  preço_unitário-item.

  Args:
    df (DataFrame): Dataframe.
  """
  # Itens que possuem preços unitários iguais
  itens_reapts = ['Sandwich', 'Cake', 'Smoothie', 'Juice']

  # Reparando os dados via relação item-preço_unitário
  for price, item in zip(prices, itens):

    # Definindo os preços pelos nomes
    df.loc[(df['Item'] == item), 'Price Per Unit'] = price

    # Definindo os nomes pelo preço
    df.loc[(df['Price Per Unit'] == price) & (~df['Item'].isin(itens_reapts)), 'Item'] = item
```

Função que repara os dados com o uso da relação matemática entre quantidade, preço por unidade e preço total.

```
def repair_data_quantity_price_total(df):
  """
  Função que realiza o reparo dos dados através da relação matemática entre quantidade
  preço unitário e total gasto.

  Args:
    df (DataFrame): Dataframe.
  """

  # Função que busca linhas no data frame, das quais duas das três colunas analisadas
  # possuam dados válidos e a restante, um dado inválido, conforme as ordem dos parâmetros.
  f = lambda x, y, z, colname: df.loc[(df[x].notna()) & (df[y].isna()) & (df[z].notna()), colname]

  # Neste loop, caminhamos por toda combinações da função 'f', preenchendo os dados conforme
  # a relação matemática.
  for x, y, z in zip(
      ['Total Spent', 'Total Spent', 'Price Per Unit'],
      ['Quantity', 'Price Per Unit', 'Total Spent'],
      ['Price Per Unit', 'Quantity', 'Quantity']):

    if y == 'Total Spent':
      df.loc[(df[x].notna()) & (df[y].isna()) & (df[z].notna()), y] = f(x, y, z, x)*f(x, y, z, z)
    elif y == 'Quantity':
      df.loc[(df[x].notna()) & (df[y].isna()) & (df[z].notna()), y] = f(x, y, z, x)/f(x, y, z, z)
    elif y == 'Price Per Unit':
      df.loc[(df[x].notna()) & (df[y].isna()) & (df[z].notna()), y] = f(x, y, z, x)/f(x, y, z, z)
```

A ideia principal foi executar estas duas funções em um loop while, enquanto existirem linhas que se encaixam para serem reparadas por uma das relações o loop será executado, chamando estas funções de reparo.