Autor: Prof. Dino Magri e Prof. João Nogueira

Contato: professor.dinomagri@gmail.com, joaonogueira@fisica.ufc.br

Parte 1 - Análise Exploratória de Dados - Exercícios

NOTA MÁXIMA: 25 pontos

NOME COMPLETO: MARCIO FERNANDES CRUZ

Acessando as bases

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Carregamento de bibliotecas
import pandas as pd
import time
import warnings
import plotly.express as px
from os.path import exists


# Fixa em duas casas decimais a visualização do float
pd.options.display.float_format = '{:.2f}'.format

warnings.filterwarnings('ignore')

#warnings.filterwarnings('always')
```

Segmentação de clientes - customer_segmentation

```
pasta_raiz = '/content/drive/MyDrive/Colab Notebooks/DataSet Data Science/ifood'

df_customer_segmentation = pd.read_csv(f'{pasta_raiz}/df_customer_segmentation.csv', sep=';')

# Conversao de Tipos
df_customer_segmentation["registration_date"] =  pd.to_datetime(df_customer_segmentation["registration_date"])
df_customer_segmentation["segmentation_month"] =  pd.to_datetime(df_customer_segmentation["segmentation_month"])
df_customer_segmentation["qtt_valid_orders"] = df_customer_segmentation["qtt_valid_orders"].astype(int)

# Normalização de Campos
df_customer_segmentation["preferred_dishes"] = df_customer_segmentation["preferred_dishes"].str.replace("\\","").replace('"','')

# Funções utilizadas no código
def recuperar_preferred_dishes(opcoes):
  pratos = str(opcoes)
  return opcoes.replace('\\', '').replace('[', '').replace(']', '').replace('"', '').split(',')
```

(0.5 ponto) `Q1` **Quantas linhas e colunas existem no conjunto de dados?**

```
print(f"Existem {df_customer_segmentation.shape[0]} linhas e {df_customer_segmentation.shape[1]} colunas no conjunto de dados de segme
```

```
Existem 210364 linhas e 50 colunas no conjunto de dados de segmentação.
```

(0.5 ponto) `Q2` Quantos clientes únicos existem?

```
print(f'Existem {df_customer_segmentation["customer_id"].nunique()} clientes no conjunto de dados de segmentação.')
```

```
Existem 30079 clientes no conjunto de dados de segmentação.
```

(1 ponto) `Q3` Quais são os 10 clientes que mais tiveram pedidos validos?

```
auxiliar = (
    df_customer_segmentation
    .groupby(by="customer_id")
    .agg(total_pedidos_validos=("qtt_valid_orders","sum"))
    .sort_values(by="total_pedidos_validos",ascending=False)
    .reset_index()
    )
auxiliar["ranking"] = auxiliar["total_pedidos_validos"].rank(ascending=False)

auxiliar = auxiliar[auxiliar["ranking"]<=10]

lista = []

def retorna_descricao(cliente):
  lista.append("O cliente na posição {} é {}".format(int(cliente["ranking"]), cliente["customer_id"]))

auxiliar.apply(retorna_descricao, axis=1)
lista
```

```
['O cliente na posição 1 é a46bb1905790aa86717354934e2b4015c4ae157ed5b08c0a433e940547733c68',
 'O cliente na posição 2 é 9f1a9d1636dff802b65c687d37583c88f263b239ff889273b08b67f6340c6877',
 'O cliente na posição 3 é 27b0b3fbf967199f757bf36f5376b20d63c6f0ad8da241755610664cf9d7125e',
 'O cliente na posição 4 é f1c5397923a646872f8385f296f9e0b66e2b0f095dcb53da7e939f0e8fc28f5a',
 'O cliente na posição 5 é c932ef421caf7076a16cb7472d4fddee9a403a304d3b1ec7100c5c32d1719140',
 'O cliente na posição 6 é 29efe6f1697579b147d7bc37c6ae2a3ebdc90546a5c926f84652c0a0e1554d88',
 'O cliente na posição 7 é 188eaac6edd1ee23f112f99c04c3c3542a56a140ea0b13351028701ff8de6116',
 'O cliente na posição 8 é 907404f63c230df86415085b09a45956fba84b973acbed6103995d6afdf0d8c3',
 'O cliente na posição 9 é 902ea307b3bcc41cf6d75162ed01d91516542ece0839f9326faff5f5b5fd2435',
 'O cliente na posição 10 é 8a7b0322a4b0f435f69e9135ccbc3560c360c4f9ec4ec3cfd7260f9a15b2f0ed']
```

(1 ponto) `Q4` Qual é o id do cliente mais antigo na plataforma? Existe apenas 1?

```
auxiliar = (
    df_customer_segmentation
  .groupby('customer_id')
  .agg(menor_registro = ('registration_date', 'min'))
  .reset_index()

)
menor_registro = auxiliar["menor_registro"].min()
auxiliar = auxiliar[auxiliar["menor_registro"]==menor_registro]

lista = []
def retorna_descricao(cliente):
  lista.append(cliente["customer_id"])

auxiliar.apply(retorna_descricao, axis=1)

print(f'Existem {auxiliar.shape[0]} registros na base com o registro do dia {menor_registro}. Segue:')
print(lista)

```

```
Existem 2 registros na base com o registro do dia 2009-12-30 22:00:00+00:00. Segue:
['07ed1b37689de6ea41fe8ca7a42129830593633ef0d058df720522b575316aea', '68180a00c9b111d93b9c7095fd545a187d871877d01080d506f7451594963
```

(1 ponto) `Q5` Quais são os três meses que contém mais clientes `Inactive`? Utilize a coluna `segmentation_month`.

```
auxiliar = (
    df_customer_segmentation[df_customer_segmentation["ifood_status"]=="Inactive"]
    .groupby(by="segmentation_month")
    .agg(quantidade=("customer_id","count"))
    .reset_index()
)
auxiliar["ranking"] = auxiliar["quantidade"].rank(ascending=False)

auxiliar = auxiliar[auxiliar["ranking"]<=3]
auxiliar.sort_values(by="ranking", inplace=True)
```

```
11
12 auxiliar["ano"] = pd.DatetimeIndex(auxiliar['segmentation_month']).year
13 auxiliar["mes"] = pd.DatetimeIndex(auxiliar['segmentation_month']).month
14
15
16 lista=[]
17 def funcao(registro):
18  lista.append("Posiçào {} com {} clientes inativos está o mês {} do ano de {}".format(int(registro["ranking"]),
19                                                                                       registro["quantidade"],
20                                                                                       registro["mes"],
21                                                                                       registro["ano"]))
22 auxiliar.apply(funcao,axis=1)
23
24 lista
```

```
['Posiçào 1 com 13309 clientes inativos está o mês 10 do ano de 2019',
 'Posiçào 2 com 10414 clientes inativos está o mês 12 do ano de 2019',
 'Posiçào 3 com 10202 clientes inativos está o mês 11 do ano de 2019']
```

(3 pontos) `Q6` Quais são os 5 pratos mais preferidos do cliente do tipo Marlin?

Será necessário criar uma função chamada `recuperar_preferred_dishes` para recuperar corretamente os dados. Essa função deve retornar uma lista com os pratos preferidos.

ATENÇÃO - Considere um único tipo de prato por cliente, por exemplo, o cliente com id 1a2b3c4d5e tem 6 vezes comida brasileira, deverá ser considerado como apenas 1.

```
 1 if exists(f'{pasta_raiz}/df_lista_comida.parquet'):
 2   df_lista_comida = pd.read_parquet(f'{pasta_raiz}/df_lista_comida.parquet')
 3 else:
 4   # Passo 1 - Criar nova coluna chamada "comidas", espelho da "preferred_dishes"
 5   analitico=df_customer_segmentation[df_customer_segmentation["marlin_tag"]=="1. Marlin"]
 6
 7   def funcao(registro):
 8     return recuperar_preferred_dishes(registro["preferred_dishes"])
 9
10   analitico["comidas"] = analitico.apply(funcao, axis=1);
11
12   # Passo 2 - Separar a lista de comidas
13   df_lista_comida = pd.DataFrame(columns = ["customer_id", "comida","quantidade"])
14   df_lista_comida.set_index(["customer_id","comida"], inplace=True)
15
16   def funcao(registro):
17     for comida in registro["comidas"]:
18       comida = comida.replace("[","").replace("]","")
19
20       valor = df_lista_comida["quantidade"].get([registro["customer_id"], comida], 0)
21
22       if valor==0:
23         df_lista_comida.loc[(registro["customer_id"], comida),"quantidade"] = 1
24
25   analitico.apply(funcao,axis=1)
26
27   df_lista_comida = (
28       df_lista_comida
29       .reset_index(inplace=False)
30       .groupby(by="comida")
31       .agg(total=("quantidade","sum"))
32       .sort_values(by="total", ascending=False)
33   )
34   df_lista_comida["ranking"] = df_lista_comida["total"].rank(ascending=False)
35   df_lista_comida.reset_index(inplace=True)
36   df_lista_comida.to_parquet(f'{pasta_raiz}/df_lista_comida.parquet')
37
38 df_lista_comida = df_lista_comida[df_lista_comida["ranking"]<=5]
39
40 lista = []
41 def exibir(registro):
42   lista.append('Na posição {}, temos {} com {} preferidos por clientes do tipo Marlin.'
43                .format(int(registro["ranking"]),
44                        registro["comida"],
45                        registro["total"]))
46
47 df_lista_comida.apply(exibir, axis=1)
48
49 lista
```

```
['Na posição 1, temos Lanches com 10996 preferidos por clientes do tipo Marlin.',
 'Na posição 2, temos Pizza com 8873 preferidos por clientes do tipo Marlin.',
 'Na posição 3, temos Comida Brasileira com 7848 preferidos por clientes do tipo Marlin.',
```

```
 'Na posição 4, temos Comida Japonesa com 4278 preferidos por clientes do tipo Marlin.',
 'Na posição 5, temos Comida Saudável com 2239 preferidos por clientes do tipo Marlin.']
```

Pedidos - `orders`

```
df_orders = pd.read_csv(f'{pasta_raiz}/df_orders.csv', sep=',')
df_orders["order_timestamp_local"] = pd.to_datetime(df_orders["order_timestamp_local"])
df_orders["last_status_date_local"] = pd.to_datetime(df_orders["last_status_date_local"])
```

(2 pontos) `Q7` Quais os três bairros de São Paulo, onde o iFood mais vende pratos do tipo de Pizza?

Lembre-se de padronizar em minusculo ou maisculo a cidade, os bairros e os tipos de pratos. Utilize o DataFrame abaixo: `df_orders_exe7`.

```
# Passo 1 - Deixar features em minusculo
auxiliar = (
  df_orders[['customer_city', 'customer_district', 'merchant_dish_type']]
  .copy()
  .apply(lambda x: x.astype(str).str.lower())
)

# Passo 2 - Filtrar pizza e cidade de São Paulo
filtro = (auxiliar["merchant_dish_type"]=="pizza") & (auxiliar["customer_city"]=="sao paulo")
auxiliar = auxiliar[filtro]

# Passo 3 - Eliminar features desnecessárias
auxiliar.drop(columns=["merchant_dish_type","customer_city"], inplace=True)

# Passo 4 - Contabilizar total por bairro
auxiliar = (auxiliar
 .groupby(by="customer_district")
 .agg(total=("customer_district", "count"))
 .reset_index()
 .sort_values(by="total", ascending=False)
  )

auxiliar["ranking"] = auxiliar["total"].rank(ascending=False)
auxiliar = auxiliar[auxiliar["ranking"]<=3]

# Passo 5 - Trazer o resultado dos 3 principais bairros
lista=[]
def funcao(registro):
  lista.append("O bairro {} está na posição {} com {} pedidos."
               .format(registro["customer_district"],
                       registro["ranking"],
                       registro["total"]

                       )
               )


auxiliar.apply(funcao, axis=1)
lista

```

```
['O bairro bela vista está na posição 1.0 com 590 pedidos.',
 'O bairro perdizes está na posição 2.0 com 424 pedidos.',
 'O bairro vila mariana está na posição 3.0 com 343 pedidos.']
```

(1 ponto) `Q8` Qual é o período do dia em que os pedidos são enviados com mais frequencia?

Considere o período sendo dinner, lunch, snack, dawn ou breakfast. Não importa se for dia de semana ou final de semana. Por exemplo: weekend dinner e weekday dinner devem ser considerados como dinner.

```
# Passo 1 - Filtrar apenas tipo de entrega Delivery
auxiliar =  df_orders[df_orders["delivery_type"]=="DELIVERY"]

auxiliar["periodo"] =  auxiliar["order_shift"]
auxiliar["periodo"] = auxiliar["periodo"].str.replace("weekend","")
auxiliar["periodo"] = auxiliar["periodo"].str.replace("weekday","")
auxiliar["periodo"] = auxiliar["periodo"].str.replace(" ","")

auxiliar = (
  auxiliar
  .groupby(by="periodo")
  .agg(total=("periodo","count"))
  .sort_values(by="total", ascending=False)
  .reset_index()
```

```
  .nlargest(1, "total")
)


#Passo 2 Imprimir Resultados
print("O período do dia que é enviado (status = Delivery) mais pedidos é {} com {} pedidos."
      .format(auxiliar.iloc[0]["periodo"],
              auxiliar.iloc[0]["total"]))
```

```
O período do dia que é enviado (status = Delivery) mais pedidos é dinner com 298422 pedidos.
```

(1 pontos) `Q9` Qual é a média de gastos, total de gastos e total de pedidos que os usuários de Android e do iOS tiveram? Os calculos devem ser feitos separadamente para Android e iOS.

```

# Passo 1 - Filtrar DataSet
auxiliar = df_orders[df_orders["device_platform"].isin(["ANDROID", "IOS"])]

(
  auxiliar
 .groupby(by="device_platform")
 .agg(media_order_total=("order_total",  "mean"),
      total_order_total=("order_total", "sum"),
      qtde_pedidos=("order_id", "nunique"))
 .reset_index()
)

```

	device_platform	media_order_total	total_order_total	qtde_pedidos
0	ANDROID	51.85	14250281.49	274822
1	IOS	60.87	12626553.58	207432

(3 pontos) `Q10` Quais foram os 10 pedidos que mais demoraram para serem entregues? Utilize as variáveis `order_timestamp_local` e `last_status_date_local`.

Lembre-se de converter as datas e horários corretamente.

```
# Passo 1 - separar features de interesse
auxiliar = (
  df_orders[['order_number', 'order_timestamp_local', 'last_status_date_local']]
  .copy()
)

# Passo 2 - Criar nova coluna para armazenar campo
auxiliar["tempo_entrega_minutos"] = (auxiliar["last_status_date_local"]-auxiliar["order_timestamp_local"]) / pd.Timedelta(minutes=1)

auxiliar.drop(columns=["order_timestamp_local", "last_status_date_local"], inplace=True)
auxiliar.sort_values(by="tempo_entrega_minutos", ascending=False, inplace=True)
auxiliar["ranking"] = auxiliar["tempo_entrega_minutos"].rank(ascending=False)
auxiliar = auxiliar[auxiliar["ranking"]<=10]

# Passo 3 - Preparar Listagem
lista = []

def funcao(registro):
  lista.append("Posição {} está o pedido {} com o tempo de {} minutos"
               .format(int(registro["ranking"]),
                       int(registro["order_number"]),
                       registro["tempo_entrega_minutos"]))

auxiliar.apply(funcao, axis=1)

lista
```

```
['Posição 1 está o pedido 907765890 com o tempo de 1719.2011833333333 minutos',
 'Posição 2 está o pedido 907766642 com o tempo de 1709.7536833333334 minutos',
 'Posição 3 está o pedido 1349842776 com o tempo de 1691.2660666666666 minutos',
 'Posição 4 está o pedido 1349845810 com o tempo de 1661.8333 minutos',
 'Posição 5 está o pedido 907774852 com o tempo de 1631.5046333333332 minutos',
 'Posição 6 está o pedido 907776818 com o tempo de 1617.3615333333332 minutos',
 'Posição 7 está o pedido 907780281 com o tempo de 1593.5330833333333 minutos',
 'Posição 8 está o pedido 828853235 com o tempo de 1576.49035 minutos',
 'Posição 9 está o pedido 1349857406 com o tempo de 1573.1572666666666 minutos',
 'Posição 10 está o pedido 1349858746 com o tempo de 1566.3488 minutos']
```

(3 pontos) `Q11` Quais foram os valores minimo, máximo e médio gasto pelos clientes durante os meses disponíveis na base?

Utilize a variável `order_timestamp_local` para recuperar o mês.

Utilize a variável `order_total` para computar as estatísticas básicas para cada mês.

É obrigatório o uso do `groupby`.

```
# a variável `order_timestamp_local` deve ter sido convertido já no exercício anterior
auxiliar = df_orders.copy()
auxiliar["ano"] = pd.DatetimeIndex(auxiliar['order_timestamp_local']).year
auxiliar["mes"] = pd.DatetimeIndex(auxiliar['order_timestamp_local']).month

(
  auxiliar
 .groupby(by=["ano", "mes"])
 .agg(valor_minimo=("order_total", "min"),
      valor_maximo=("order_total", "max"),
      valor_medio=("order_total", "mean"),
      )
 .reset_index()
)

```

	ano	mes	valor_minimo	valor_maximo	valor_medio
0	2019	6	13.05	609.10	55.43
1	2019	7	13.50	603.60	54.82
2	2019	8	13.00	1058.00	55.77
3	2019	9	16.50	759.50	55.62
4	2019	10	15.00	807.30	55.43
5	2019	11	13.79	667.70	56.10
6	2019	12	14.00	1641.00	57.76

Sessões das visitas realizadas - `sessions_visits`

```
df_session_visits = pd.read_csv(f'{pasta_raiz}/df_sessions_visits.csv', sep=',')

df_session_visits.insert(0,"unique_id", df_session_visits["user_identifier"].astype(str)+df_session_visits["session_id"].astype(str))

df_session_visits["session_started_at_utc0"] = pd.to_datetime(df_session_visits["session_started_at_utc0"])
df_session_visits["session_ended_at_utc0"] = pd.to_datetime(df_session_visits["session_ended_at_utc0"])
df_session_visits["tempo_sessao"] = (df_session_visits["session_ended_at_utc0"]-df_session_visits["session_started_at_utc0"]) / pd.Tim


def recuperar_n_mais_usuarios_ativos(df_session, ranking_inicial, ranking_final):
  df_auxiliar = (
      df_session_visits
    .copy()
    .sort_values(by="tempo_sessao", ascending=False)
  )

  df_auxiliar.insert(1,"ranking", df_auxiliar["tempo_sessao"].rank(ascending=False))

  filtro = (df_auxiliar["ranking"]>=ranking_inicial) & (df_auxiliar["ranking"]<=ranking_final)
  return df_auxiliar[filtro]

```

(3 pontos) `Q12` Realize o passo a passo abaixo:

- Faça concatenação das variáveis `user_identifier` e `session_id`, nesta ordem, para criar um novo identidicador único, chamado `unique_id`
- Insira a coluna `unique_id` na posição 0 do DataFrame `df_session_visits`. Utilize uma variável auxiliar para facilitar.
- Crie uma nova coluna com a diferença entre as variáveis `session_started_at_utc0` e `session_ended_at_utc0`. O nome da coluna deve ser `tempo_sessao`.
- Crie uma função que irá receber dois parâmetros, uma DataFrame e outra a quantidade de usuário mais ativos dentro da plataforma.

O objetivo dessa função é verificar os n usuários que ficaram mais tempo dentro da plataforma. NÃO precisa fazer a agregação, considere cada linha um usuário único.

O retorno dessa função deve ser um DataFrame que irá conter todas as colunas filtradas para os n usuários mais ativos, incluindo as duas novas colunas criadas anteriormente (`unique_id` e `tempo_sessao`)

Teste a função com os seguintes comandos:

```
recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, n_mais_usuarios=0)
recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, n_mais_usuarios=10)
```

Lembre-se de tratar possíveis erros

- Por fim, copie e cole o seguinte comando na última célula do exercício.

```
df_final = recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, n_mais_usuarios=1000)
```

```
# 10 primeiros
df_final_10_primeiros = recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, ranking_inicial=1, ranking_final=10)
df_final_10_primeiros.head(100)
```

	unique_id	ranking	session_id	dau	platform	
351470	503d295dbeef80d0169bc59bd4ea87f9f9a7689b9cb4b3...	1.00	3b42a14d-b19b-40fe-9b0b-e0dbc4689197	2019-10-17_9f643ee2-35b0-3d16-a71c-c403417f77b2	ANDROID	503d295dbeef80d0169bc59bd4
350512	de14cef1f693ed124d1ac58b38a88ebe835b7dd1752cf2...	2.00	2d10f2ac-7883-42aa-8d30-d9c4c548cdca	2019-09-01_2305c8c3-3784-386f-8960-6a129ca1716d	ANDROID	de14cef1f693ed124d1ac58b38a
137922	741e8ab0-47ab-49e0-b00a-135d80d6614f176e1dd5-d...	3.00	176e1dd5-d9a1-4508-a7a3-b95cbaa6eeeb	2019-06-01_f395407f-0eda-4fa6-b8bc-b5510dab53d3	IOS	741e8ab0-47ab-4
99004	941fd963eeed5e6863eb26140413e4fd12ec83d4dac1a9...	4.00	4e8ee134-642b-418a-8273-bdbd31e358dd	2019-09-11_1d2e04c2-dbec-4fce-8dff-58e26105b39c	ANDROID	941fd963eeed5e6863eb2614041
224938	b28b3fba-6ce4-4ef7-a276-e33aeadfbd4f87976490-0...	5.00	87976490-01a4-4f11-9d57-9f8e785c9def	2019-06-19_b5652d5a-c7bc-4dc3-9e7e-c45e52676566	ANDROID	b28b3fba-6ce4
78067	d732b2a62f422885550a46128af471cc98ebe3d7c52404...	6.00	8a6089c3-6bc3-4f3d-8a7f-107da53b3c1f	2019-08-24_20fdb364-ddbb-4f44-b615-020ca22e3d98	ANDROID	d732b2a62f422885550a46128af
262062	708c6f45-d4bd-423d-8081-305dd773fd08d1ca9e7d-9...	7.00	d1ca9e7d-986e-4944-8b4a-eec2f9da45b4	2019-07-25_5de0588c-7fd1-4e5f-86cd-dc65b4219400	ANDROID	708c6f45-d4bd-4
71112	a53df1a9-df6c-4fb8-909a-3881d1e741a2b8d7985b-6...	8.00	b8d7985b-67c3-4b38-ba4d-67e129b60207	2019-06-14_db92eba2-e00e-4719-abf4-448bd89fb2b5	ANDROID	a53df1a9-df6c-4
57049	8873e87a-61a7-41da-8380-ac8e6981c45c71f7dca9-9...	9.00	71f7dca9-9eb5-4171-901b-23bf0f9a45d9	2019-06-11_81bcb38d-e6f3-4593-9e85-924a7154d661	ANDROID	8873e87a-61a7-4
73269	2039e3cf-b93e-4a11-bd95-010414a458b99ece17d8-3...	10.00	9ece17d8-324d-4da3-9788-848209158dad	2019-06-21_52866b5f-f3c4-45b5-b480-96ea111c523e	ANDROID	2039e3cf-b93e-4

```
# 30.o ao 50.o no ranking
df_final_30_ao_50 = recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, ranking_inicial=30, ranking_final=50)
df_final_30_ao_50.head(30)

df_final = recuperar_n_mais_usuarios_ativos(df_session=df_session_visits, ranking_inicial=1, ranking_final=1000)
```

(5 pontos) `Q13` Utilizando qualquer um dos quatro DataFrames (`df_customer_segmentation`, `df_orders`, `df_session_visits` ou `df_final`), crie os seguintes gráficos:

- Histograma
- Gráfico de dispersão
- Gráfico de barras
- Box-plot

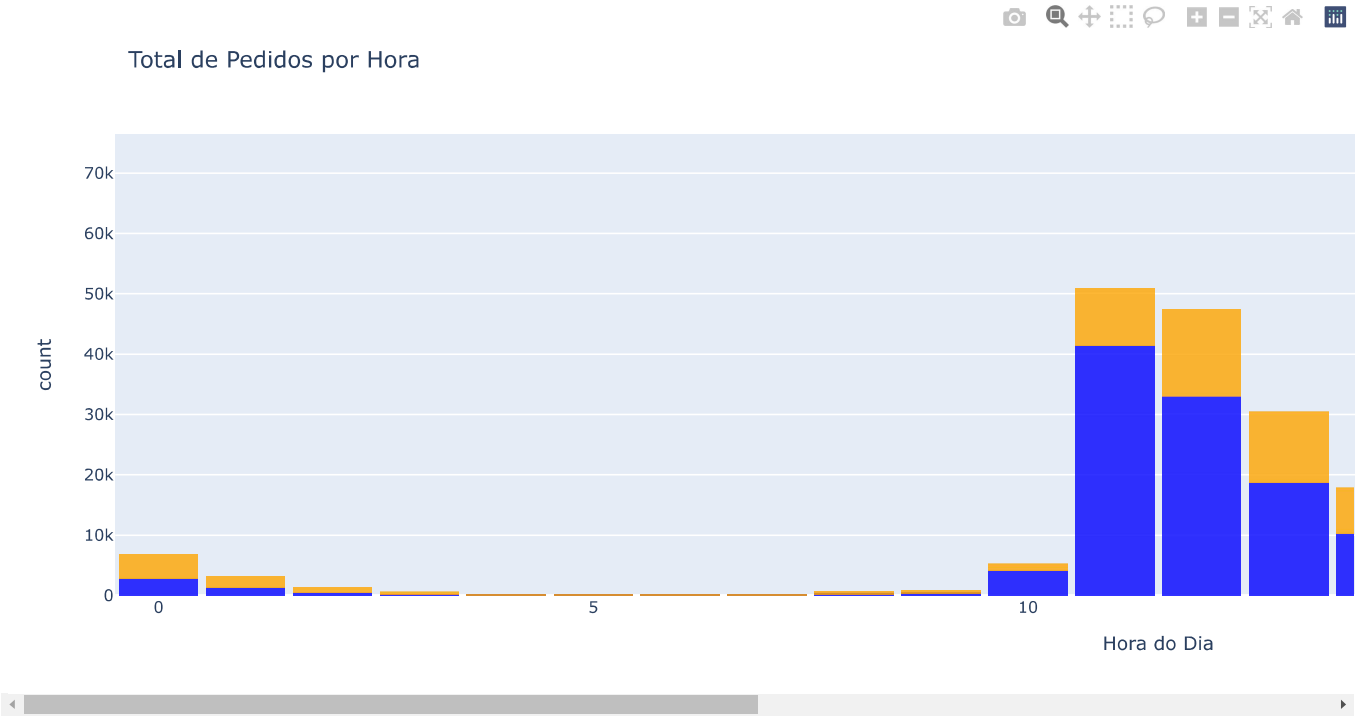
Para cada um dos gráficos, faça uma análise da visualização obtida.

Histograma

```
# Preparar DataSet Auxiliar
df_orders_hora = df_orders.copy()

def eh_fim_de_semana(registro):
  return registro["order_shift"].find("weekend ")==0

df_orders_hora["eh_fim_de_semana"] = df_orders_hora.apply(eh_fim_de_semana, axis=1)
df_orders_hora["hora"] = pd.DatetimeIndex(df_orders_hora['order_timestamp_local']).hour
fig = px.histogram(df_orders_hora,
                   x="hora",
                   title='Total de Pedidos por Hora',
                   labels={'hora':'Hora do Dia', 'eh_fim_de_semana':'Fim de Semana'},
#                   marginal="box",
                   color="eh_fim_de_semana",
                   color_discrete_sequence=["blue","orange"],
                   opacity=0.8,
                   nbins=24)
fig.update_layout(bargap=0.1)
fig.show()
```



Análise do Histograma:

- A maioria dos pedidos são feitos final de semana;
- No dia, começa na faixa das 11h e vai descrescendo até as 17h;
- E, na parte da noite, inicia a partir das 19h, com pico entre 20h.

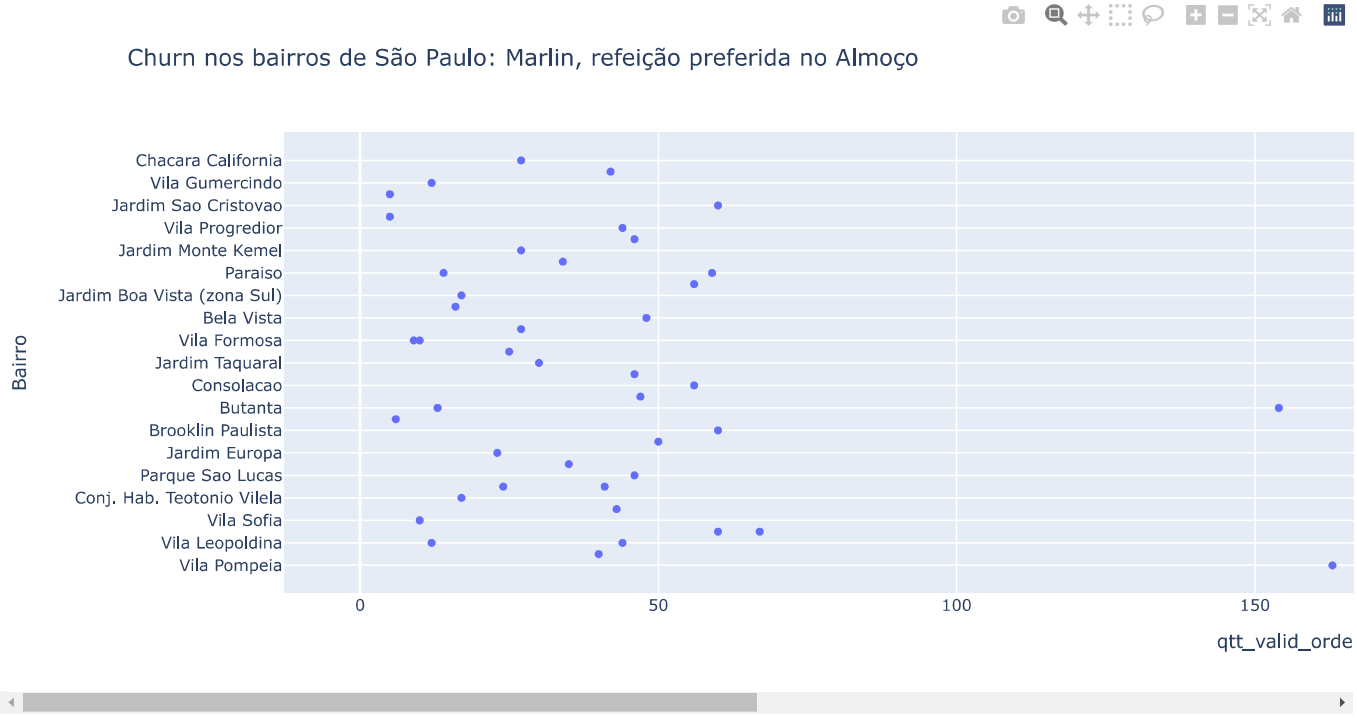
Conclusão sobre o Histograma de Total de Pedidos por Hora

- Conseguimos demandar a partir deste gráfico os picos de uso do Serviço como um todo;
- Pode-se infomar os entregadores para que se preparem melhor nestes horários;
- Pode-se infomar a TI para demandar recursos de Cloud e Suporte de Pessoal para estes horários;
- Pode-se avisar os restaurantes dos principais horários de demanda para que se preparem também, porém, este último seria melhor análises adicionais para saber qual tipo de comida mais solicitada por horário.

Gráfico de dispersão

```
filtro =  (df_customer_segmentation["top_city"]=="SAO PAULO") & (df_customer_segmentation["ifood_status"]=="Churn") & (df_customer_seg
df_auxiliar = df_customer_segmentation[filtro]

fig = px.scatter(df_auxiliar,
           x="qtt_valid_orders",
           title="Churn nos bairros de São Paulo: Marlin, refeição preferida no Almoço",
           y="top_district",
           labels={'merchant_offer':'Cobertura Restaurante', 'top_district':'Bairro'})
fig.show()
```



Análise do Gráfico de Dispersão:

- Análise: Churn nos bairros de São Paulo: Marlin, refeição preferida no Almoço;
- O objetivo é tentar descobrir se há diferença no comportamento que evolui a Churn.

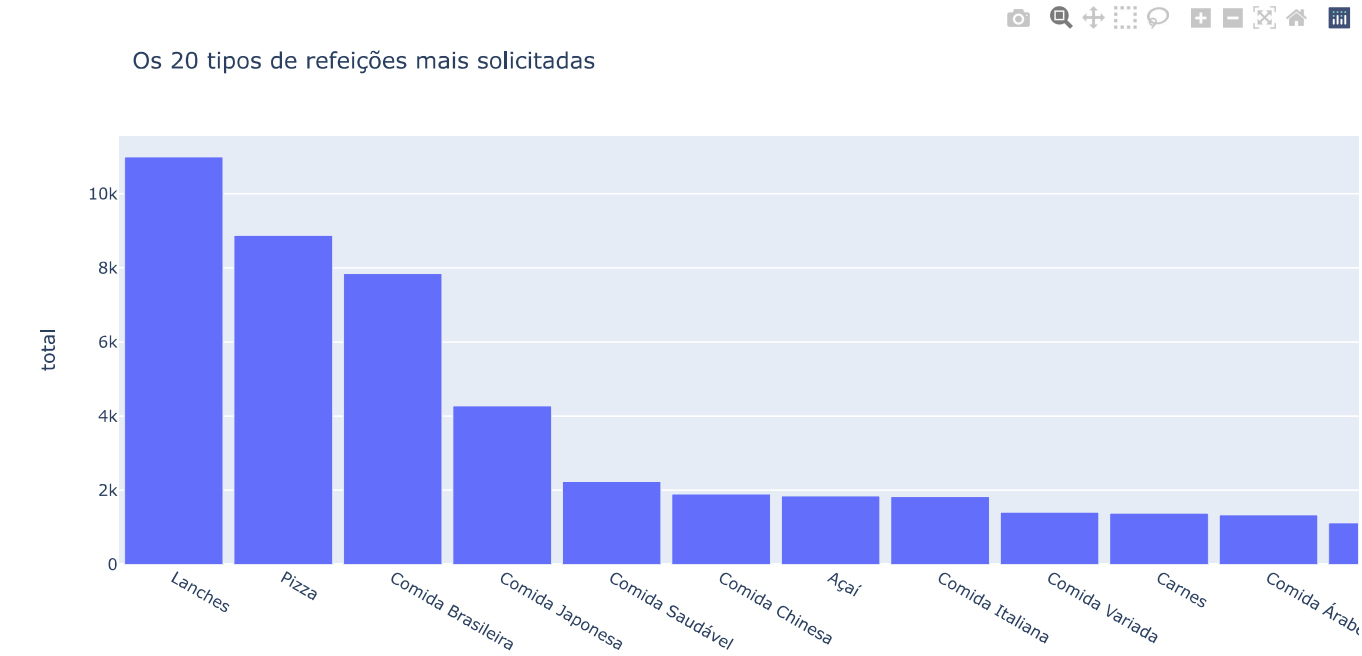
Conclusão sobre o Gráfico de Dispersão

- Peguei a cidade e verifiquei o comportamento com outras grandes cidade e, o padrão de comportamento é o mesmo;
- Achei que alguns bairros poderiam ter comportamento diferente quanto ao Churn para pedidos preferidos no horário do Almoço;
- A maioria de Churn ocorre com clientes que fazem até 50 pedidos, independente do bairro.

Gráfico de barras

```
 df_auxiliar = pd.read_parquet(f'{pasta_raiz}/df_lista_comida.parquet')
# Preparar DataSet Auxiliar
df_auxiliar = (df_auxiliar
 .groupby(by="comida")
 .agg(total=("total", "sum"))
 .reset_index()
 .nlargest(20, "total")
 )

# Preparar o Gráfico
fig = px.bar(df_auxiliar,
             x='comida',
             y='total',
             title='Os 20 tipos de refeições mais solicitadas'
             )
fig.update_layout(bargap=0.1)
fig.show()
```



Análise do Gráfico de Barras:

- A maioria dos pedidos são de Lanches, Pizza e culinária Brasileira.

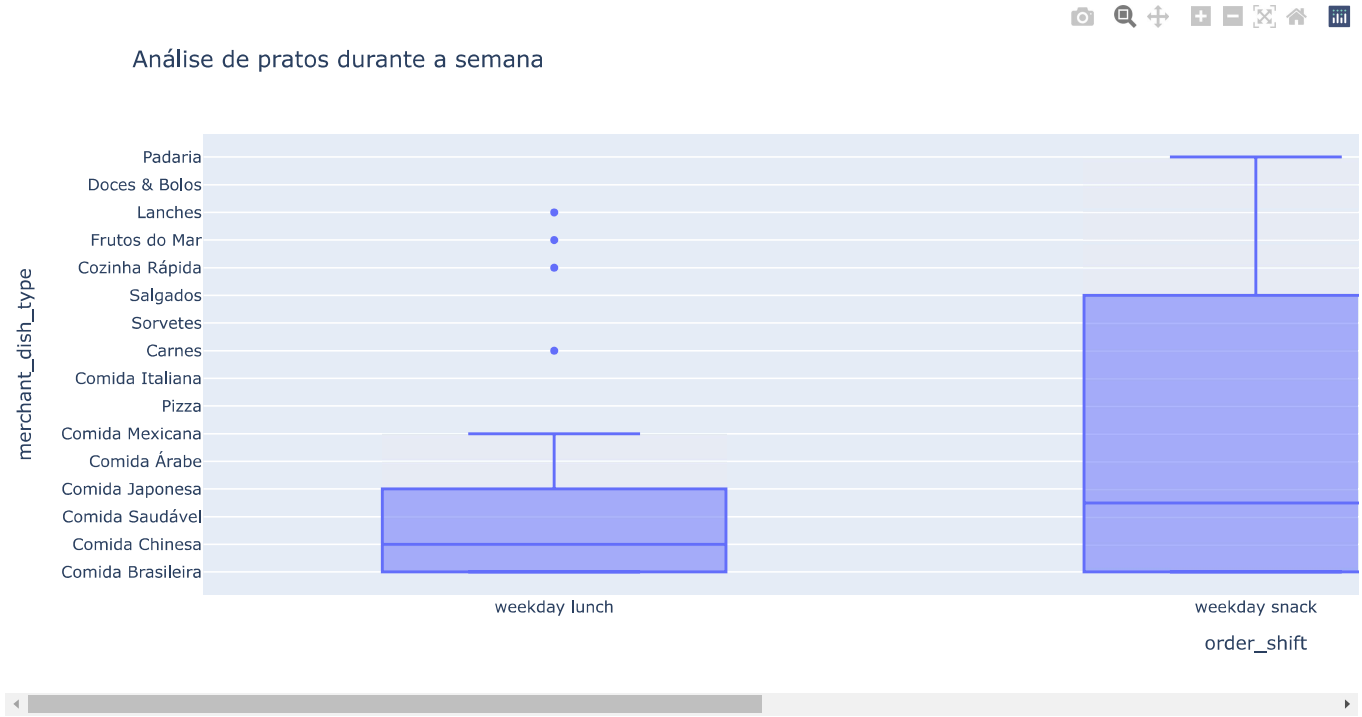
Conclusão sobre o o Gráfico de Barras

- A comida saudável está apenas na 5.a posição, metade que a Japonesa;
- Pode ser divulgado esta análise para que empreendedores criem negócios direcionados a certos tipos de pratos;
- Este gráfico traz uma visão geral dos pratos mas, seria mais interessante fazer análises regionais e por horário para trazer dados mais assertivos para eventuais tomadas de decisão, caso estas utilizem o prato como referência.

Box-plot

```
df_auxiliar = df_orders_hora.copy()
filtro =  (df_auxiliar["eh_fim_de_semana"]==False) & (df_auxiliar["customer_city"]=="SAO PAULO") & (df_auxiliar["customer_district"]==
df_auxiliar = df_auxiliar[filtro]

fig = px.box(df_auxiliar,
             x="order_shift",
             y="merchant_dish_type",
             title="Análise de pratos durante a semana")
fig.show()
```



Análise do BoxPlot dos Pedidos:

- Coloquei uma suposição de um empreendedor que quer montar um disque entrega de comida no Centro da Cidade e, funcionará até o inicio da noite.

Conclusão sobre o o Gráfico

- Das 10h as 14h59, deve-se direcionar sua cozinha para Comida Brasileira, Chinesa, Japonesa ou Saudável;
- Das 15h as 16h59, o espectro da comida é ainda mior mas o foco vai para a comida Saudável;
- A partir das 17h, o foco do gráfico fica entre Japonesa e Pizza, ainda num espectro grande mas, de acordo com a o dataframe, a media para a noite fica com a Pizza na região dos pedidos feitos no centro de São Paulo.