Sabemos que um gráfico de linhas é o ideal para representação de dados temporais, seja para investigar a evolução dos dados ao longo do tempo ou para identificar padrões e tendências em nossos dados. Por isso, é muito importante escolher o tempo que estamos analisando e como vamos representá-lo, por exemplo, em valores diários, trimestrais, anuais entre outros.

Na biblioteca Matplotlib, temos dois métodos que auxiliam na construção de gráficos de linhas ao lidar com diferentes intervalos de dados: set\_major\_locator e set\_major\_formatter. Eles estão disponíveis na manipulação de objetos no eixo (**Axes**) e, de maneira resumida, possuem as seguintes características:

### **set\_major\_locator**

É utilizado para definir a localização dos principais marcadores (ticks) no eixo do gráfico. Ele permite controlar a posição dos marcadores ao longo do eixo, sendo útil tanto para dados temporais quanto para dados que não estão uniformemente espaçados.

Por exemplo, se nós estivermos plotando dados diários em um gráfico de linha com um período de um ano, o espaçamento padrão dos marcadores fica muito denso e torna o gráfico difícil de ler. Nesse caso, nós podemos utilizar o set\_major\_locator para especificar um espaçamento maior entre os marcadores, como a cada mês ou a cada ano, dependendo do intervalo dos dados.

Dentro da própria Matplotlib existe o módulo [**dates**](https://matplotlib.org/stable/api/dates_api.html) que facilita a localização dos ticks das mais variadas formas de tempo, trazendo uma série de métodos que são combinados ao set\_major\_locator, por exemplo:

* [DayLocator](https://matplotlib.org/stable/api/dates_api.html#matplotlib.dates.DayLocator): localiza o dia do mês. Podemos pontuar qual ou quais dias específicos do mês gostaríamos de avaliar ou o intervalo de exibição de dias (15 em 15 dias, por exemplo);
* [MonthLocator](https://matplotlib.org/stable/api/dates_api.html#matplotlib.dates.MonthLocator): localiza os meses pelo número do mês. Aqui também podemos escolher um ou mais meses para localizar ou o intervalo de exibição dos meses (3 em 3 meses, por exemplo);
* [YearLocator](https://matplotlib.org/stable/api/dates_api.html#matplotlib.dates.YearLocator): localiza os anos que são múltiplos do valor passado como argumento (base). Além disso é possível passar o dia exato ou mês exato que queremos exibir em cada intervalo de anos que passamos no argumento base.

### **set\_major\_formatter**

É utilizado para formatar os rótulos dos marcadores (ticks) no eixo do gráfico. Ele permite controlar a representação visual dos valores ao longo do eixo, como formatar datas, horas, números ou texto.

Por exemplo, ao plotar datas no eixo x, você pode usar o set\_major\_formatter para formatar as datas no formato desejado, como mês-ano ou dia-mês-ano. Isso torna o gráfico mais legível e compreensível para quem vai lê-lo.

Dentro da própria Matplotlib podemos utilizar um formatador de data chamado [DateFormatter](https://matplotlib.org/stable/api/dates_api.html#matplotlib.dates.DateFormatter) que utiliza a função de formatação strftime() para formatar as datas para o tipo string. A lista com os códigos de formatação de datas pode ser encontrada neste [link](https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes). Aqui abaixo, temos alguns exemplos de formatação de data:

* DateFormatter(‘%d’): retorna o dia do mês (01, 10, 31)
* DateFormatter(‘%m’): retorna o mês no formato numérico (01, para janeiro)
* DateFormatter(‘%a’): retorna o dia da semana abreviado, padrão em inglês (Sun, Mon, Tue)
* DateFormatter(‘%b’): retorna o mês com nome abreviado, padrão em inglês (Jan, Mar.. Dec)
* DateFormatter(‘%y’): retorna o ano em 2 dígitos (00, 15, 20)
* DateFormatter(‘%Y’): retorna o ano em 4 dígitos (2000, 2015, 2020)
* DateFormatter(‘%U’): retorna o número correspondente a semana no ano (00, 15, 52)

### **Exemplo**

Vamos fazer um exemplo para testar a utilização dos métodos que lemos mais acima, apresentando em um gráfico de linha as vendas no estado do Amazonas no biênio de 2018 a 2019.

Primeiro, vamos importar os dados e as bibliotecas que iremos utilizar, selecionando os dados de interesse e agrupando-os por mês com a função resample(“MS”). Isto agrupa os dados pelo início de cada mês.

# Importando as bibliotecas

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

# Importando o relatório de vendas e atualizando a colunas de pedido para o tipo data

vendas = pd.read\_csv("https://raw.githubusercontent.com/alura-cursos/dataviz-graficos/master/dados/relatorio\_vendas.csv")

vendas["data\_pedido"] = pd.to\_datetime(vendas["data\_pedido"], format="%Y-%m-%d")

# Criando um df com os dados desejados

df\_am = vendas.copy()

bienio = [2018, 2019]

df\_am = df\_am.query('estado == "Amazonas" and data\_pedido.dt.year == @bienio')[["data\_pedido", "vendas"]]

# Agrupando as vendas por início do mês (MS)

df\_am.set\_index("data\_pedido", inplace = True)

df\_am = df\_am.resample("MS").agg("sum")

df\_am = df\_am.reset\_index()

df\_am.head()

**Saída**:

|  | **data\_pedido** | **vendas** |
| --- | --- | --- |
| 0 | 2018-01-01 | 532.16 |
| 1 | 2018-02-01 | 568.42 |
| 2 | 2018-03-01 | 8691.91 |
| 3 | 2018-04-01 | 712.42 |
| 4 | 2018-05-01 | 11791.93 |

Agora, vamos criar o nosso primeiro gráfico. Neste vamos focar em apresentar os dados de tempo em intervalos de 4 em 4 meses aplicando a função MonthLocator() no set\_major\_locator() e trazendo para o formato mês-ano aplicando a função DateFormatter()no set\_major\_formatter():

# Área do gráfico e tema da visualização

fig, ax = plt.subplots(figsize=(16,4))

# Criando o gráfico de linha das vendas pelo biênio 2018-2019

ax.plot(df\_am["data\_pedido"], df\_am["vendas"], marker = "o")

## Personalizando o gráfico

ax.set\_title('Vendas no estado do Amazonas no biênio 2018-2019', fontsize = 16, loc='left', pad = 20)

ax.set\_xlabel('Meses', fontsize = 14)

ax.set\_ylabel('')

ax.grid(axis = "y", linestyle="--")

ax.set\_frame\_on(False)

# remover todos os ticks do eixo x e y

ax.tick\_params(axis='both', which='both', length=0)

# Descrevendo o limite mínimo e máximo do eixo y

plt.ylim(0, 15000)

# Definindo o intervalo de 4 em 4 meses (quadrimestre)

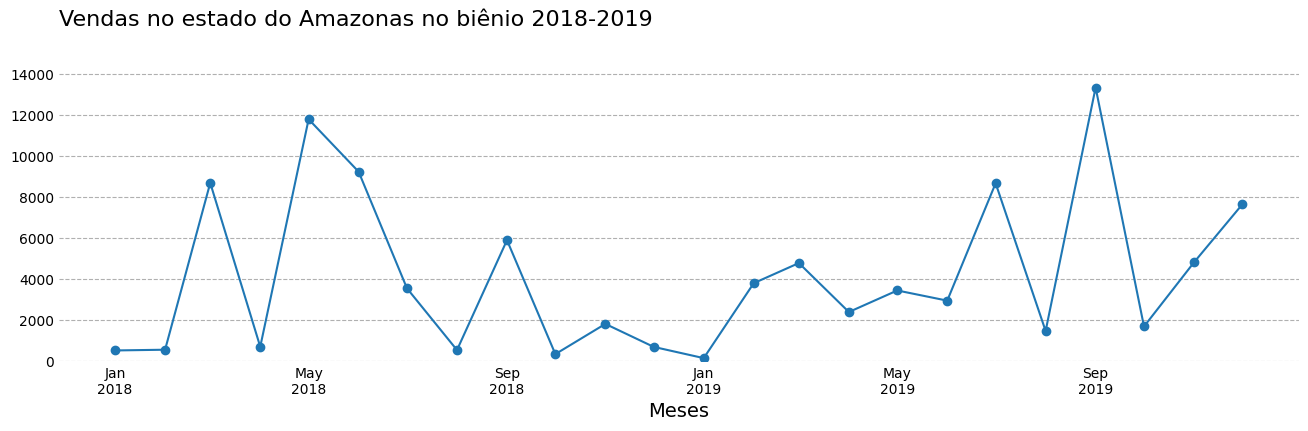
ax.xaxis.set\_major\_locator(mdates.MonthLocator(bymonth = range(1,12,4)))

ax.xaxis.set\_major\_formatter(mdates.DateFormatter("%b\n%Y"))

ax.set\_xticks(ax.get\_xticks()[:-1]) # retirando o último tick que não possui valores (Jan/2020)

plt.show()

O gráfico gerado pelo código é:



Note que o gráfico apresenta todos os dados do df\_am, mas o eixo x só apresenta o intervalo das datas que escolhemos.

Agora, vamos modificar o eixo x novamente com outro intervalo de tempo. Vamos apresentar os dados de tempo pela semana do ano que aquele dado é reconhecido aplicando a função DayLocator() no set\_major\_locator() e trazendo para o formato semana-ano aplicando a função DateFormatter()no set\_major\_formatter():

# Atualizando o gráfico para dados semanais

ax.set\_xlabel('Semana do Ano', fontsize = 14)

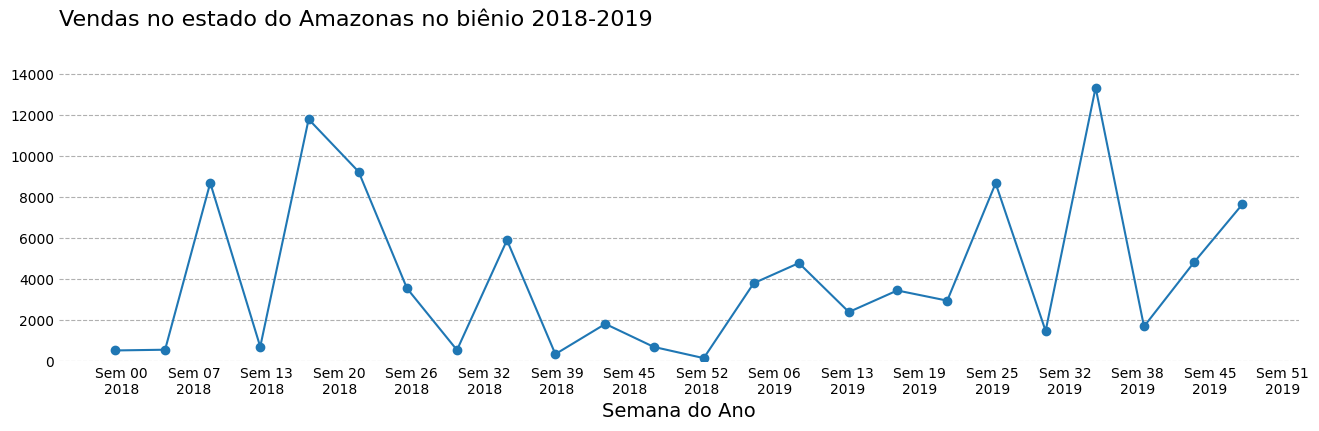
# Definindo o intervalo em 45 dias e passando os valores de ticks como Semana e ano

ax.xaxis.set\_major\_locator(mdates.DayLocator(interval = 45))

ax.xaxis.set\_major\_formatter(mdates.DateFormatter("Sem %U\n%Y"))

ax.get\_figure()

Por sua vez, o novo gráfico é:



Note que a linha do gráfico não sofre alteração, pois ainda apresentamos todos os dados do df\_am, mas o eixo x alterou o intervalo do tempo para o formato que definimos em DateFormatter().

Resumindo, os métodos set\_major\_locator e set\_major\_formatter da biblioteca Matplotlib desempenham um papel crucial na construção de gráficos de linhas ao lidar com diferentes intervalos de dados, permitindo selecionar e formatar adequadamente os rótulos dos eixos temporais, proporcionando uma representação clara e significativa dos dados.

Esses métodos auxiliam as pessoas profissionais de dados a apresentar informações precisas e convincentes, facilitando a compreensão e a tomada de decisões. Portanto, dominar o uso desses métodos é essencial para criar gráficos de linhas eficazes e informativos, contribuindo para uma análise mais completa e robusta dos dados temporais.