Programação para Ciência de Dados

Análise de Séries Temporais - Fundamentos e Manipulação

Arthur Casals

11 de Novembro de 2025

Agenda

- Introdução: O que são Séries Temporais?
- Bloco 1: DatetimeIndex e Manipulação Básica
- Bloco 2: Resample, Rolling Windows e Agregações
- Bloco 3: Decomposição e Análise de Componentes
- Bloco 4: Visualizações e Boas Práticas

O que são Séries Temporais?

Série Temporal = sequência de dados ordenados no tempo

Características principais:

- Dados coletados em intervalos regulares
- Ordem importa (temporal)
- Cada observação tem timestamp
- Passado influencia presente/futuro

Diferença de dados tradicionais:

Dados Tradicionais	Séries Temporais
Linhas independentes	Linhas dependentes
Ordem não importa	Ordem CRÍTICA
Sem timestamp	Com timestamp
i.i.d. assumido	Autocorrelação presente

Exemplos de Séries Temporais

Estão por toda parte!

Finanças:

- Preços de ações (segundo a segundo)
- Taxa de câmbio (diária)
- Volume de negociação

Negócios:

- Vendas (diária, mensal)
- Tráfego de website (horária)
- Demanda de produtos

Ciência:

- Temperatura (horária)
- Qualidade do ar (nossa aula de hoje!)
- Dados de sensores IoT

Por que Séries Temporais São Especiais?

Características únicas que exigem tratamento especial:

1. Dependência Temporal

- Valor de hoje depende de ontem
- Autocorrelação presente
- Não podemos embaralhar os dados!

2. Tendência

- Movimento de longo prazo (subindo/descendo)
- Crescimento ou declínio ao longo do tempo

Por que Séries Temporais São Especiais?

Características únicas que exigem tratamento especial:

3. Sazonalidade

- Padrões que se repetem periodicamente
- Diário, semanal, mensal, anual

4. Ciclicidade

- Flutuações de longo prazo (sem período fixo)
- Ciclos econômicos, por exemplo

Componentes de uma Série Temporal

Decomposição clássica:

$$Y_t = T_t + S_t + C_t + I_t$$

Onde:

- Y_t**:** Valor observado no tempo t
- T_t**:** Tendência (Trend) - direção de longo prazo
- S_t**:** Sazonalidade (Seasonal) - padrão repetitivo
- C_t**:** Ciclo (Cycle) - flutuações de longo prazo
- I_t**:** Irregular/Resíduo - ruído aleatório

Modelo alternativo (multiplicativo):

$$Y_t = T_t \times S_t \times C_t \times I_t$$

Objetivo: Separar componentes para entender melhor

Visualização dos Componentes

Exemplo: Vendas mensais

Tendência:

- Crescimento constante ao longo dos anos
- Linha suave subindo

Sazonalidade:

- Pico em dezembro (Natal)
- Queda em janeiro/fevereiro
- Repete todo ano

Visualização dos Componentes

Exemplo: Vendas mensais

Ciclo:

- Recessão econômica reduz vendas
- Duração: 3-5 anos
- Não tem período fixo

Irregular:

- Eventos aleatórios (promoção inesperada)
- Não previsível

Aplicações de Análise de Séries Temporais

O que podemos fazer:

1. **Descrição e Exploração**
   - Entender padrões históricos
   - Identificar anomalias
   - Visualizar tendências

2. **Previsão (Forecasting)**
   - Prever valores futuros
   - ARIMA, Prophet, LSTM
   - (Não veremos modelagem hoje - foco em manipulação)

Aplicações de Análise de Séries Temporais

O que podemos fazer:

3. Detecção de Anomalias

- Identificar comportamentos estranhos
- Fraude, falhas de sistema

4. Análise de Causa e Efeito

- Impacto de eventos
- Correlação temporal entre séries

Dataset de Hoje: Air Quality

Dados de qualidade do ar em uma cidade italiana

Origem:

- UCI Machine Learning Repository
- Sensor multivariado de qualidade do ar
- Localização: cidade industrial italiana
- Período: março 2004 - fevereiro 2005

Frequência:

- Medições HORÁRIAS
- 9358 observações (1 ano de dados)
- Timestamp completo: data + hora

Dataset de Hoje: Air Quality

Dados de qualidade do ar em uma cidade italiana

Variáveis principais:

- CO (Carbon Monoxide) - mg/m³
- NO2 (Nitrogen Dioxide) - µg/m³
- Temperature (°C)
- Relative Humidity (%)
- Absolute Humidity

Por que Air Quality é Bom Exemplo?

Dataset ideal para aprender séries temporais:

Características interessantes:

- **Frequência horária:** Alta granularidade
- **Sazonalidade clara:**
  - Diária: poluição varia durante o dia
  - Semanal: diferença fim de semana vs dias úteis
  - Anual: diferença inverno vs verão
- **Tendências:** Mudanças ao longo do ano
- **Múltiplas variáveis:** Correlações temporais
- **Missing values:** Desafio realista

Relevância prática:

- Problema de saúde pública
- Previsão para alertas
- Identificar fontes de poluição

Desafios em Séries Temporais

Problemas comuns:

1. Missing Values

- Sensores falham
- Não pode usar métodos tradicionais
- Precisa interpolação temporal

2. Outliers

- Eventos extremos vs erros de medição
- Difícil distinguir

3. Frequências Irregulares

- Dados não coletados em intervalos exatos
- Precisa resample/regularização

Desafios em Séries Temporais

Problemas comuns:

4. Múltiplas Frequências

- Combinar dados diários + mensais
- Alinhamento temporal

5. Estacionariedade

- Propriedades estatísticas mudam no tempo
- Muitos modelos assumem estacionariedade

Pandas para Séries Temporais

DatetimeIndex:

- Índice temporal especializado
- Operações temporais nativas
- Acesso por datas intuitivo

Funções temporais:

- `resample()`: Mudar frequência
- `rolling()`: Médias móveis
- `shift()`: Defasagens (lags)
- `diff()`: Diferenças temporais
- `pct_change()`: Variações percentuais

Acessores temporais:

- `.dt.year, .dt.month, .dt.day`
- `.dt.hour, .dt.dayofweek`
- `.dt.quarter, .dt.dayofyear`

Objetivos de Aprendizado

Ao final desta aula, você será capaz de:

Conhecimento:

- Entender componentes de séries temporais
- Conhecer desafios específicos
- Interpretar padrões temporais
- Distinguir tendência vs sazonalidade

Habilidades:

- Criar e manipular DatetimeIndex
- Fazer resample (mudar frequência)
- Calcular médias móveis (rolling)
- Decompor séries em componentes
- Tratar missing values em séries
- Criar visualizações temporais eficazes
- Extrair features temporais

Setup: Imports e Configurações

Python

```python
# Imports necessarios
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import zipfile
import io
import requests
```

Setup: Imports e Configurações

Python

```python

# Configuracoes
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

plt.style.use('seaborn-v0_8-darkgrid')
plt.rcParams['figure.figsize'] = (14, 6)
plt.rcParams['font.size'] = 11

# Para plots melhores
sns.set_palette("husl")

print("Setup completo!")

```

Carregar Air Quality Dataset

Python

```python
# Carregar dados
zip_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00360/AirQualityUCI.zip"
r = requests.get(zip_url)
z = zipfile.ZipFile(io.BytesIO(r.content))

# Carregar CSV
with z.open('AirQualityUCI.csv') as f:
    df = pd.read_csv(f, sep=';', decimal=',', na_values=-200)
```

Carregar Air Quality Dataset

</> Python

```python
# Combinar 'Date' e 'Time' em uma única coluna
df['timestamp'] = pd.to_datetime(
    df['Date'] + ' ' + df['Time'],
    format='%d/%m/%Y %H.%M.%S',
    dayfirst=True,
    errors='coerce'
)

print("=== DATASET CARREGADO ===")
print(f"Shape: {df.shape}")
print(f"\nPrimeiras linhas:")
print(df.head())

print(f"\nInfo:")
df.info()
```

Exploração Inicial

Python

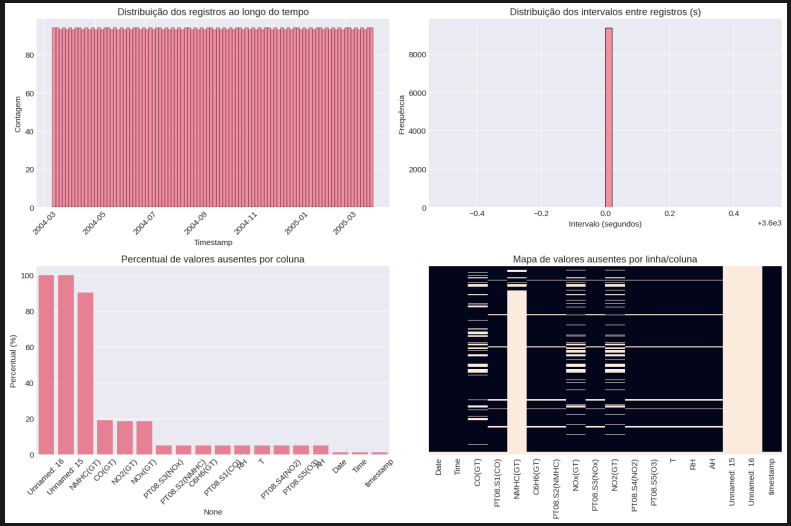
```python
# Verificar range temporal
print("=== RANGE TEMPORAL ===")
print(f"Inicio: {df['timestamp'].min()}")
print(f"Fim: {df['timestamp'].max()}")
print(f"Duracao: {df['timestamp'].max() - df['timestamp'].min()}")

# Verificar frequencia
print("\n=== FREQUENCIA ===")
time_diff = df['timestamp'].diff().mode()[0]
print(f"Intervalo mais comum: {time_diff}")
```

Exploração Inicial

Python

```python
# Valores faltantes
print("\n=== VALORES FALTANTES ===")
missing = df.isnull().sum()
missing_pct = (missing / len(df)) * 100
missing_df = pd.DataFrame({
    'Missing': missing,
    'Percent': missing_pct
}).sort_values('Percent', ascending=False)
print(missing_df[missing_df['Missing'] > 0])
```

Exploração Inicial (visual)



Fluxo de Análise de Séries Temporais

Processo sistemático:

FASE 1: Preparação

- Carregar dados
- Converter para DatetimeIndex
- Ordenar temporalmente
- Tratar missing values

FASE 2: Exploração

- Visualizar série completa
- Identificar padrões visuais
- Estatísticas descritivas
- Detectar anomalias

Fluxo de Análise de Séries Temporais

Processo sistemático:

FASE 3: Análise

- Decomposição
- Agregações temporais
- Médias móveis
- Correlações temporais

FASE 4: Feature Engineering

- Extrair componentes temporais
- Criar lags
- Calcular diferenças

Ferramentas e Bibliotecas

Manipulação:

- **Pandas:** Fundação principal
- **NumPy:** Operações numéricas
- **Matplotlib/Seaborn:** Visualizações

Análise avançada:

- **statsmodels:** Decomposição, ARIMA
- **scipy:** Análise de sinais
- **Prophet:** Forecasting por Facebook

Deep Learning:

- **TensorFlow/Keras:** LSTM, GRU
- **PyTorch:** Redes neurais recorrentes

Foco de hoje: Apenas Pandas + básicos de statsmodels

Tipos de Análise: Univariada vs Multivariada

Duas abordagens:

Univariada:

- Uma única série temporal
- Analisa apenas ela mesma
- Exemplo: apenas temperatura
- Previsão baseada em história própria

Multivariada:

- Múltiplas séries relacionadas
- Analisa relações entre elas
- Exemplo: temperatura + umidade + CO
- Uma pode ajudar a prever outra

Tipos de Análise: Univariada vs Multivariada

Air Quality:

- Dataset é multivariado
- Hoje: focaremos principalmente em univariadas
- Veremos correlações entre variáveis

Frequências Comuns

Diferentes granularidades temporais:

Frequência	Código	Exemplo
Anual	'A' ou 'Y'	Vendas anuais
Trimestral	'Q'	Relatórios financeiros
Mensal	'M'	Faturamento mensal
Semanal	'W'	Tráfego semanal
Diária	'D'	Temperatura diária
Horária	'H'	Consumo de energia
Minuto	'T' ou 'min'	Dados de sensores
Segundo	'S'	Trading de ações
Milissegundo	'L' ou 'ms'	Dados científicos

Frequências Comuns

Diferentes granularidades temporais:
Air Quality: Dados horários (‘H’)

Escolha depende:

- Natureza do fenômeno
- Disponibilidade de dados
- Objetivo da análise

Conceitos Importantes

Terminologia essencial:

Lag (Defasagem):

- Valor de t períodos atrás
- lag=1: ontem, lag=7: semana passada

Diferenciação:

- $Y_t - Y_{t-1}$
- Remove tendência
- Torna série estacionária

Conceitos Importantes

Terminologia essencial:

Autocorrelação:

- Correlação da série consigo mesma
- Em diferentes lags
- Mede dependência temporal

Estacionariedade:

- Propriedades estatísticas constantes
- Média e variância não mudam no tempo
- Importante para modelagem

Caso de Uso Motivador

Empresa de monitoramento ambiental:

Problema:

- Sensores geram dados horários
- 8760 observações por ano por sensor
- Precisa detectar padrões e anomalias
- Alertar quando qualidade do ar está ruim

Desafios:

- Volume massivo de dados
- Ruído nos dados
- Missing values ocasionais
- Múltiplas escalas (hora, dia, semana, mês)

Caso de Uso Motivador

Empresa de monitoramento ambiental:

Solução com séries temporais:

- Agregação temporal (horária $\rightarrow$ diária)
- Médias móveis para suavizar
- Decomposição para identificar tendências
- Detecção de anomalias via desvios

Expectativas Realistas

O que aprenderemos hoje:

SIM:

- ✓ Manipular séries temporais no Pandas
- ✓ Visualizar e explorar padrões
- ✓ Fazer agregações temporais
- ✓ Calcular médias móveis
- ✓ Decompor séries
- ✓ Tratar dados temporais

NÃO (ficará para o futuro):

- ✗ Modelagem preditiva (ARIMA, Prophet)
- ✗ Deep Learning para séries (LSTM)
- ✗ Análise estatística profunda
- ✗ Testes de estacionariedade formais

Expectativas Realistas

Nota Importante

Hoje é sobre MANIPULAÇÃO e EXPLORAÇÃO, não modelagem!

Estrutura da Aula

Bloco 1: DatetimeIndex

- Criar DatetimeIndex
- Acessar e filtrar por datas
- Extrair componentes temporais
- Operações básicas

Bloco 2: Agregações e Rolling

- Resample (mudar frequência)
- Rolling windows (médias móveis)
- Shifting e lags
- Diferenciação

Estrutura da Aula

Blocos 3 e 4: Decomposição + Práticas

- Decompor em componentes
- Visualizações avançadas
- Boas práticas
- Pipeline completo

Transição para Bloco 1

Fundação estabelecida!

Entendemos:

- O que são séries temporais
- Por que são especiais
- Componentes principais
- Dataset Air Quality
- Desafios comuns

Próximo: Bloco 1 - DatetimeIndex

- Coração da manipulação temporal no Pandas
- Converter strings para datas
- Definir índice temporal
- Acessar dados por período
- Extrair informações temporais

Bloco 1

DatetimeIndex e Manipulação Básica

O que é DatetimeIndex?

Índice especializado para séries temporais

Definição:

- Tipo especial de índice no Pandas
- Contém objetos datetime
- Otimizado para operações temporais
- Permite slicing por datas

Por que usar DatetimeIndex?

- Acesso intuitivo: df['2024-01-01']
- Funções temporais: resample(), rolling()
- Alinhamento automático por tempo
- Operações vetorizadas rápidas
- Componentes temporais: .dt.year, .dt.month

O que é DatetimeIndex?

Nota Importante

Sem DatetimeIndex, você não tem série temporal - só dados com datas!

DataFrame vs TimeSeries

DataFrame comum (índice numérico):

Index	Date	Value
0	2024-01-01	10
1	2024-01-02	15
2	2024-01-03	12

TimeSeries (DatetimeIndex):

Timestamp	Value
2024-01-01	10
2024-01-02	15
2024-01-03	12

Diferença principal:

- TimeSeries: data É o índice
- Permite: df['2024-01'] $\rightarrow$ todos de janeiro
- Habilita: resample, rolling, etc.

Criar DatetimeIndex: Método 1 - parse_dates

Python

```python
# Metodo 1: parse_dates no pd.read_csv
# Quando data e hora estao em colunas separadas
# (ver slides anteriores)
print("=== TIPO DA COLUNA ===")
print(type(df['timestamp'][0]))
# <class 'pandas._libs.tslibs.timestamps.Timestamp'>
# Verificar
print(df['timestamp'].head())
# Ainda NãO é indice! É apenas uma coluna
print(f"\nIndice atual: {type(df.index)}")
# Indice atual: <class 'pandas.core.indexes.range.RangeIndex'>
```

Criar DatetimeIndex: Método 2 - pd.to_datetime

Python

```python
# Metodo 2: converter coluna existente
# Se ja tem coluna de texto com datas
df['timestamp'] = pd.to_datetime(df['timestamp_string'])

# Ou especificar formato (mais rapido!)
df['timestamp'] = pd.to_datetime(df['timestamp_string'],
                                 format='%Y-%m-%d %H:%M:%S')
# Formatos comuns:
# '%Y-%m-%d' $\rightarrow$ 2024-01-15
# '%d/%m/%Y' $\rightarrow$ 15/01/2024
# '%Y-%m-%d %H:%M:%S' $\rightarrow$ 2024-01-15 10:30:00
print("=== CONVERSAO ===")
print(df['timestamp'].dtype)
# datetime64[ns]
print(df['timestamp'].head())
```

Definir DatetimeIndex

Python

```python
# Transformar coluna em indice

# Opcao 1: set_index
df = df.set_index('timestamp')

print("=== AGORA É DATETIMEINDEX ===")
print(type(df.index))
# <class 'pandas.core.indexes.datetimes.DatetimeIndex'>

print(f"\nIndice:\n{df.index}")
```

Definir DatetimeIndex (cont.)

Python

```python
# Opcao 2: Direto no read_csv
df = pd.read_csv('data.csv',
                 parse_dates=['timestamp'],
                 index_col='timestamp')

# Verificar propriedades do indice
print(f"\nFrequencia: {df.index.freq}")
print(f"Inicio: {df.index.min()}")
print(f"Fim: {df.index.max()}")
# Frequencia: None
# Inicio: 2004-03-10 18:00:00
# Fim: 2005-04-04 14:00:00
```

Verificar e Ordenar DatetimeIndex

Python

```python
# SEMPRE verificar se esta ordenado
print("=== ORDENACAO ===")
print(f"Está ordenado? {df.index.is_monotonic_increasing}")
# === ORDENACAO ===
# Está ordenado? False

# Se nao estiver, ordenar
df = df.sort_index()
# Verificar novamente
print(f"Agora esta ordenado? {df.index.is_monotonic_increasing}")
# Por que isso importa?
# - resample() e rolling() assumem ordem temporal
# - Slicing por datas precisa de ordem
# - Performance melhor
```

Verificar e Ordenar DatetimeIndex (cont.)

Python

```python
# Checar duplicatas
print(df.index.duplicated().sum())
# Remover duplicatas de timestamp
df = df[~df.index.duplicated(keep='first')]
print(f"\nShape apos remover duplicatas: {df.shape}")
# Checar NaNs
print(df.index.hasnans)
# Remover NaNs
df = df.loc[df.index.dropna()]
```

Propriedades do DatetimeIndex

Range e frequência:

- `.min()`, `.max()`: Limites
- `.freq`: Frequência (se regular)
- `.inferred_freq`: Frequência inferida

Propriedades:

- `.is_monotonic_increasing`: Ordenado?
- `.is_unique`: Sem duplicatas?
- `.has_duplicates`: Tem duplicatas?

Componentes:

- `.year`, `.month`, `.day`
- `.hour`, `.minute`, `.second`
- `.dayofweek`, `.dayofyear`
- `.quarter`, `.weekofyear`

Acessar Componentes Temporais

Python

```python
# Extrair componentes do DatetimeIndex

# Do indice direto
print("=== COMPONENTES DO INDICE ===")
print(f"Anos unicos: {df.index.year.unique()}")
print(f"Meses unicos: {df.index.month.unique()}")

# === COMPONENTES DO INDICE ===
# Anos unicos: Index([2004, 2005], dtype='int32', name='timestamp')
# Meses unicos: Index([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2], dtype='
    int32', name='timestamp')
```

Acessar Componentes Temporais (cont.)

Python

```python
# Criar colunas com componentes
df['year'] = df.index.year
df['month'] = df.index.month
df['day'] = df.index.day
df['hour'] = df.index.hour
df['dayofweek'] = df.index.dayofweek  # 0=Segunda, 6=Domingo
df['quarter'] = df.index.quarter

print("\n=== NOVAS COLUNAS ===")
print(df[['year', 'month', 'day', 'hour', 'dayofweek']].head())

# Nome do dia da semana
df['weekday_name'] = df.index.day_name()
print(f"\nDias da semana: {df['weekday_name'].unique()}")
```

Acessar Componentes Temporais (cont.)

Python

```
# === NOVAS COLUNAS ===
#                      year  month  day  hour  dayofweek
# timestamp
# 2004-03-10 18:00:00  2004      3   10    18          2
# 2004-03-10 19:00:00  2004      3   10    19          2
# 2004-03-10 20:00:00  2004      3   10    20          2
# 2004-03-10 21:00:00  2004      3   10    21          2
# 2004-03-10 22:00:00  2004      3   10    22          2
#
# Dias da semana: ['Wednesday' 'Thursday' 'Friday' 'Saturday' 'Sunday' '
    Monday' 'Tuesday']
```

Slicing por Datas: Básico

Python

```python
# Um dia especifico
jan_15 = df.loc['2004-03-15']
print(f"Dados de 15/03/2004: {len(jan_15)} registros")
# Um mes inteiro
march_2004 = df.loc['2004-03']
print(f"Dados de marco/2004: {len(march_2004)} registros")
# Um ano inteiro
year_2004 = df.loc['2004']
print(f"Dados de 2004: {len(year_2004)} registros")
# Range de datas
spring = df.loc['2004-03':'2004-05']
print(f"Primavera 2004: {len(spring)} registros")
# Até certa data
early = df.loc[:'2004-06']
print(f"Até junho/2004: {len(early)} registros")
```

Slicing por Datas: Básico (cont.)

Python

```python
# Dados de 15/03/2004: 24 registros
# Dados de marco/2004: 510 registros
# Dados de 2004: 7110 registros
# Primavera 2004: 1974 registros
# Até junho/2004: 2694 registros
```

Slicing por Datas: Avançado

Python

```python
# Horario especifico em todos os dias
morning_10am = df[df.index.hour == 10]
print(f"Todas as 10h da manha: {len(morning_10am)} registros")
# Fins de semana (sabado=5, domingo=6)
weekends = df[df.index.dayofweek >= 5]
print(f"Fins de semana: {len(weekends)} registros")
# Dias uteis
weekdays = df[df.index.dayofweek < 5]
print(f"Dias uteis: {len(weekdays)} registros")
# Horario de pico (7-9h)
rush_hour = df[df.index.hour.isin([7, 8, 9])]
print(f"Horario de pico: {len(rush_hour)} registros")
# Dezembro (mes de festas)
december = df[df.index.month == 12]
print(f"Dezembro: {len(december)} registros")
```

Slicing por Datas: Avançado (cont.)

Python

```python
# Todas as 10h da manha: 390 registros
# Fins de semana: 2688 registros
# Dias uteis: 6669 registros
# Horario de pico: 1170 registros
# Dezembro: 744 registros
```

Filtrar por Múltiplas Condições

Python

```python
# Dias uteis + horario comercial (9-18h)
business_hours = df[
    (df.index.dayofweek < 5) &  # Segunda a sexta
    (df.index.hour >= 9) &       # Depois das 9h
    (df.index.hour < 18)         # Antes das 18h
]
print(f"Horario comercial: {len(business_hours)} registros")
# Verao (junho-agosto) + dias quentes
summer_hot = df[
    (df.index.month.isin([6, 7, 8])) &
    (df['T'] > 25)  # Temperatura > 25C
]
```

Filtrar por Múltiplas Condições

Python

```python
print(f"Dias quentes de verao: {len(summer_hot)} registros")
# Madrugada (0-6h) em fins de semana
weekend_nights = df[
    (df.index.dayofweek >= 5) &
    (df.index.hour < 6)
]
print(f"Madrugadas de fim de semana: {len(weekend_nights)}")
```

Operações com Timestamps

Python

```python
from datetime import timedelta
# Adicionar/subtrair tempo
tomorrow = df.index + timedelta(days=1)
print(f"Amanha: {tomorrow[:5]}")
week_ago = df.index - timedelta(days=7)
print(f"Semana passada: {week_ago[:5]}")
# Diferenca entre timestamps
time_diff = df.index.to_series().diff()
print(f"\nDiferenca de tempo:")
print(time_diff.head())
# Tempo decorrido desde inicio
elapsed = df.index - df.index.min()
elapsed_hours = elapsed.total_seconds() / 3600
print(f"\nHoras desde inicio:")
print(f"Min: {elapsed_hours.min():.1f}h")
print(f"Max: {elapsed_hours.max():.1f}h")
```

Criar DatetimeIndex do Zero

Python

```python
# Diario
daily = pd.date_range(start='2024-01-01',
                      end='2024-12-31',
                      freq='D')
print(f"Diario: {len(daily)} dias")
# Horario
hourly = pd.date_range(start='2024-01-01',
                       periods=24*7,  # 1 semana
                       freq='H')
print(f"Horario: {len(hourly)} horas")
```

Criar DatetimeIndex do Zero (cont.)

Python

```python
# Mensal (fim do mes)
monthly = pd.date_range(start='2024-01-01',
                        periods=12,
                        freq='M')
print(f"Mensal: {len(monthly)} meses")
# Criar DataFrame com indice temporal
df_new = pd.DataFrame({
    'value': np.random.randn(len(hourly))
}, index=hourly)
```

Frequências do Pandas

Código	Descrição	Exemplo	Alias
D	Dia	2024-01-01	day
H	Hora	10:00	hour
T, min	Minuto	10:30	minute
S	Segundo	10:30:45	second
L, ms	Milissegundo	-	-
W	Semana	Domingos	week
M	Fim de mês	2024-01-31	month
MS	Início de mês	2024-01-01	-
Q	Trimestre	2024-03-31	quarter
A, Y	Ano	2024-12-31	year
B	Dias úteis	Seg-Sex	business day

Modificadores:

- '2H' = a cada 2 horas
- '15T' = a cada 15 minutos
- '3D' = a cada 3 dias

Detectar e Preencher Gaps Temporais

Python

```python
# Identificar lacunas na serie temporal

# Frequencia esperada
expected_freq = '1H'  # Horaria

# Criar range completo esperado
full_range = pd.date_range(start=df.index.min(),
                           end=df.index.max(),
                           freq=expected_freq)

print(f"Timestamps esperados: {len(full_range)}")
print(f"Timestamps reais: {len(df)}")
print(f"Missing: {len(full_range) - len(df)}")
```

Detectar e Preencher Gaps Temporais (cont.)

Python

```python
# Reindexar para preencher gaps
df_complete = df.reindex(full_range)

print(f"\nApos reindex: {df_complete.shape}")
print(f"NaN introduzidos: {df_complete.isnull().sum().sum()}")

# Agora podemos tratar missing adequadamente
```

Tratar Missing Values: Forward Fill

Python

```python
# Forward fill: propagar ultimo valor válido

# Metodo 1: ffill (forward fill)
df_filled = df_complete.fillna(method='ffill')

print("=== FORWARD FILL ===")
print(f"Missing antes: {df_complete['CO(GT)'].isnull().sum()}")
print(f"Missing depois: {df_filled['CO(GT)'].isnull().sum()}")

# Limitar quantos valores propagar
df_filled = df_complete.fillna(method='ffill', limit=3)
# Propaga no maximo 3 valores
```

Tratar Missing Values: Forward Fill (cont.)

Python

```python
# Exemplo visual
print("\nAntes:")
print(df_complete['CO(GT)'][100:110])

print("\nDepois:")
print(df_filled['CO(GT)'][100:110])

# Util quando: valor tende a permanecer constante
```

Tratar Missing Values: Interpolação

Python

```python
# Interpolacao linear (padrao)
df_interp = df_complete.interpolate(method='linear')

print("=== INTERPOLACAO ===")
print(f"Missing antes: {df_complete['CO(GT)'].isnull().sum()}")
print(f"Missing depois: {df_interp['CO(GT)'].isnull().sum()}")

# Outros metodos
# method='time': considera distancia temporal real
df_interp = df_complete.interpolate(method='time')

# method='spline': curva suave
df_interp = df_complete.interpolate(method='spline', order=2)
```

Tratar Missing Values: Interpolação (cont.)

Python

```python
# Comparar
print("\nOriginal (com gaps):")
print(df_complete['CO(GT)'][100:110])
print("\nInterpolado:")
print(df_interp['CO(GT)'][100:110])
```

Escolher Método de Imputação Temporal

Guia de decisão:

Forward Fill (ffill):

- Use: valores mudam raramente
- Exemplo: status de sensor (ligado/desligado)
- Exemplo: categoria (tipo de dia)
- Limite: poucos gaps pequenos

Backward Fill (bfill):

- Use: raramente (só em casos especiais)
- Exemplo: preencher início da série

Escolher Método de Imputação Temporal

Guia de decisão:

Interpolação Linear:

- Use: valores contínuos que mudam suavemente
- Exemplo: temperatura, umidade, poluição
- Melhor para séries temporais típicas

NUNCA usar:

- Média global (ignora contexto temporal!)

Estatísticas por Período

Python

```python
# Por hora do dia
hourly_stats = df.groupby(df.index.hour)['CO(GT)'].agg([
    'mean', 'std', 'min', 'max', 'count'
])

print("=== ESTATISTICAS POR HORA ===")
print(hourly_stats)

# Por dia da semana
weekday_stats = df.groupby(df.index.dayofweek)['CO(GT)'].mean()
weekday_stats.index = ['Seg', 'Ter', 'Qua', 'Qui', 'Sex', 'Sab', 'Dom']

print("\n=== MEDIA POR DIA DA SEMANA ===")
print(weekday_stats)
```

Estatísticas por Período (cont.)

Python

```python
# Por mes
monthly_stats = df.groupby(df.index.month)['CO(GT)'].mean()
monthly_stats.index = ['Jan', 'Fev', 'Mar', 'Abr', 'Mai', 'Jun',
                       'Jul', 'Ago', 'Set', 'Out', 'Nov', 'Dez']

print("\n=== MEDIA POR MES ===")
print(monthly_stats)
```

Comparar Períodos

Python

```python
# Comparar dias uteis vs fins de semana
weekday_co = df[df.index.dayofweek < 5]['CO(GT)'].mean()
weekend_co = df[df.index.dayofweek >= 5]['CO(GT)'].mean()

print("=== DIAS UTEIS VS FINS DE SEMANA ===")
print(f"Dias uteis: {weekday_co:.2f} mg/m³")
print(f"Fins de semana: {weekend_co:.2f} mg/m³")
print(f"Diferenca: {weekday_co - weekend_co:.2f} mg/m³")
print(f"Variacao: {((weekday_co/weekend_co - 1) * 100):.1f}%")
```

Comparar Períodos

Python

```python
# Comparar horarios de pico vs nao-pico
peak_hours = df[df.index.hour.isin([7,8,9,17,18,19])]['CO(GT)'].mean()
off_peak = df[~df.index.hour.isin([7,8,9,17,18,19])]['CO(GT)'].mean()

print("\n=== PICO VS NAO-PICO ===")
print(f"Horario de pico: {peak_hours:.2f} mg/m³")
print(f"Fora de pico: {off_peak:.2f} mg/m³")
```

Criar Features Temporais

Python

```python
# 1. Parte do dia
def get_part_of_day(hour):
    if 6 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 18:
        return 'Afternoon'
    elif 18 <= hour < 22:
        return 'Evening'
    else:
        return 'Night'

df['part_of_day'] = df.index.hour.map(get_part_of_day)
```

Criar Features Temporais

Python

```python
# 2. Fim de semana
df['is_weekend'] = (df.index.dayofweek >= 5).astype(int)

# 3. Horario de pico
df['is_rush_hour'] = df.index.hour.isin([7,8,9,17,18,19]).astype(int)

# 4. Estacao do ano (hemisferio norte)
df['season'] = df.index.month % 12 // 3 + 1
season_map = {1: 'Winter', 2: 'Spring', 3: 'Summer', 4: 'Fall'}
df['season_name'] = df['season'].map(season_map)

print(df[['part_of_day', 'is_weekend', 'season_name']].head(10))
```

Features Cíclicas

Python

```python
# Representar componentes temporais como ciclicas
# (preserva a natureza circular do tempo em analises!)

import numpy as np

# Hora do dia (0-23) como seno/cosseno
df['hour_sin'] = np.sin(2 * np.pi * df.index.hour / 24)
df['hour_cos'] = np.cos(2 * np.pi * df.index.hour / 24)

# Dia da semana (0-6) como seno/cosseno
df['dow_sin'] = np.sin(2 * np.pi * df.index.dayofweek / 7)
df['dow_cos'] = np.cos(2 * np.pi * df.index.dayofweek / 7)
```

Features Cíclicas

Python

```python
# Mes do ano (1-12) como seno/cosseno
df['month_sin'] = np.sin(2 * np.pi * df.index.month / 12)
df['month_cos'] = np.cos(2 * np.pi * df.index.month / 12)

print("=== FEATURES CICLICAS ===")
print(df[['hour_sin', 'hour_cos', 'dow_sin', 'month_sin']].head())

# Por que?
# - Preserva distancia real: 23h e 0h estao proximas
# - Dezembro (12) e Janeiro (1) sao continuos
# - Util para correlacoes e visualizacoes temporais
```

Por que Features Cíclicas?

O problema com encoding ordinal:

Hora do dia (0-23):

- Ordinal: 0, 1, 2, ..., 22, 23
- Problema: 23h e 0h codificadas como muito distantes!
- Realidade: 23h e 0h são 1 hora de diferença

Solução cíclica:

- $\sin(2\pi \times \text{hour}/24)$ e $\cos(2\pi \times \text{hour}/24)$
- 23h: sin≈-0.26, cos≈0.97
- 0h: sin≈0.0, cos≈1.0
- Distância euclidiana pequena! ✓

Por que Features Cíclicas?

Quando usar:

- Hora do dia, dia da semana, mês
- Qualquer variável cíclica/periódica
- Análises de correlação temporal
- Cálculos de distância e similaridade

Nota Importante

Representação cíclica é essencial para análises que dependem de distâncias/similaridades temporais!

Timezone Awareness

Python

```python
# Trabalhar com fusos horarios

# Verificar se tem timezone
print(f"Timezone aware? {df.index.tz is not None}")

# Adicionar timezone (se nao tiver)
df_tz = df.copy()
df_tz.index = df_tz.index.tz_localize('UTC')

print(f"Agora é timezone aware: {df_tz.index.tz}")

# Converter para outro fuso
df_sp = df_tz.copy()
df_sp.index = df_sp.index.tz_convert('America/Sao_Paulo')
```

Timezone Awareness (cont.)

Python

```python
print(f"\nUTC: {df_tz.index[0]}")
print(f"Sao Paulo: {df_sp.index[0]}")

# Importante para:
# - Dados de multiplas localizacoes
# - Horario de verao
# - Coordenacao entre sistemas
```

Encontrar Valores Extremos

Python

```python
# Maior valor de CO
max_co_idx = df['CO(GT)'].idxmax()
max_co_val = df['CO(GT)'].max()

print("=== PIOR QUALIDADE DO AR (CO) ===")
print(f"Quando: {max_co_idx}")
print(f"Valor: {max_co_val:.2f} mg/m³")
print(f"Hora: {max_co_idx.hour}h")
print(f"Dia da semana: {max_co_idx.day_name()}")
```

Encontrar Valores Extremos

Python

```python
# Melhor valor
min_co_idx = df['CO(GT)'].idxmin()
min_co_val = df['CO(GT)'].min()

print("\n=== MELHOR QUALIDADE DO AR (CO) ===")
print(f"Quando: {min_co_idx}")
print(f"Valor: {min_co_val:.2f} mg/m³")

# Top 10 piores momentos
worst_10 = df.nlargest(10, 'CO(GT)')
print("\n=== TOP 10 PIORES ===")
print(worst_10[['CO(GT)', 'T', 'RH']].to_string())
```

Exercício: Análise Temporal Básica

Exercício Prático

Explore o Air Quality dataset temporalmente:

1. Configure DatetimeIndex:

```
# - Carregar com parse_dates
# - Definir como indice
# - Ordenar e verificar
```

2. Extraia componentes temporais:
   - year, month, day, hour, dayofweek
   - Crie coluna 'season'

Exercício: Análise Temporal Básica

✋ Exercício Prático

Explore o Air Quality dataset temporalmente (cont.):

3. Análises:
   - Média de CO por hora do dia
   - Média de temperatura por mês
   - Comparar dias úteis vs fim de semana
4. Identifique:
   - 10 momentos com pior qualidade do ar
   - Padrões (quando ocorrem?)

Bloco 2

Resample, Rolling Windows e Agregações

O que é Resample?

Resample = mudar a frequência temporal dos dados

Dois tipos principais:

1. Downsampling (reduzir frequência):

- Horária $\rightarrow$ Diária
- Diária $\rightarrow$ Mensal
- Minuto $\rightarrow$ Hora
- Precisa agregar (média, soma, max, etc.)

2. Upsampling (aumentar frequência):

- Diária $\rightarrow$ Horária
- Mensal $\rightarrow$ Diária
- Hora $\rightarrow$ Minuto
- Cria gaps (precisa preencher)

O que é Resample?

Resample = mudar a frequência temporal dos dados

Por que fazer?

- Reduzir ruído (*downsampling*)
- Alinhar séries com frequências diferentes
- Visualizar tendências
- Reduzir volume de dados

Downsampling: Conceito

Exemplo: Horária → Diária

Original (horária - 24 valores por dia):

Timestamp	CO
2004-03-10 00:00	2.6
2004-03-10 01:00	2.0
...	...
2004-03-10 23:00	2.3

Após resample diário (média):

Date	CO
2004-03-10	2.2

24 valores → 1 valor (média dos 24)

Resample: Sintaxe Básica

</> Python

```python
# Sintaxe geral
# df_resampled = df.resample('frequencia').funcao_agregacao()

# Exemplo 1: Horaria -> Diaria (media)
daily_avg = df['CO(GT)'].resample('D').mean()

print("=== RESAMPLE DIARIO ===")
print(f"Original: {len(df)} registros (horarios)")
print(f"Resampled: {len(daily_avg)} registros (diarios)")
print(f"\nPrimeiros valores:")
print(daily_avg.head())
```

Resample: Sintaxe Básica

</> Python

```python
# Exemplo 2: Horaria -> Diaria (max)
daily_max = df['CO(GT)'].resample('D').max()

print("\n=== MAX DIARIO ===")
print(daily_max.head())

# Frequencias: 'D'=dia, 'W'=semana, 'M'=mes, 'Q'=trimestre, 'A'=ano
```

Resample: Múltiplas Agregações

Python

```python
# Uma coluna, multiplas funcoes
daily_stats = df['CO(GT)'].resample('D').agg([
    'mean', 'std', 'min', 'max', 'count'
])

print("=== ESTATISTICAS DIARIAS DE CO ===")
print(daily_stats.head())

# Multiplas colunas, diferentes funcoes
daily_multi = df.resample('D').agg({
    'CO(GT)': ['mean', 'max'],
    'NO2(GT)': ['mean', 'max'],
    'T': ['mean', 'min', 'max'],
    'RH': 'mean'
})
```

Resample: Múltiplas Agregações (cont.)

Python

```python
print("\n=== MULTIPLAS COLUNAS ===")
print(daily_multi.head())

# Renomear colunas se necessario
daily_multi.columns = ['_'.join(col).strip() for col in daily_multi.
    columns]

```

Resample para Diferentes Períodos

Python

```python
# Diario (media)
daily = df['CO(GT)'].resample('D').mean()
print(f"Diario: {len(daily)} registros")
# Semanal (media)
weekly = df['CO(GT)'].resample('W').mean()
print(f"Semanal: {len(weekly)} registros")
# Mensal (media)
monthly = df['CO(GT)'].resample('M').mean()
print(f"Mensal: {len(monthly)} registros")
# A cada 6 horas
six_hourly = df['CO(GT)'].resample('6H').mean()
print(f"6 em 6 horas: {len(six_hourly)} registros")
# A cada 3 dias
three_daily = df['CO(GT)'].resample('3D').mean()
print(f"A cada 3 dias: {len(three_daily)} registros")
```

Escolher Função de Agregação

Qual função usar ao fazer downsample?

mean() - Média:

- Use: valores contínuos, visão geral
- Exemplo: temperatura média diária

sum() - Soma:

- Use: totais, acumulados
- Exemplo: vendas totais do mês

max() / min():

- Use: valores extremos
- Exemplo: pico de poluição, temperatura mínima

Escolher Função de Agregação

Qual função usar ao fazer downsample?

median() - Mediana:

- Use: quando há outliers
- Mais robusta que média

count():

- Use: número de observações válidas
- Detectar missing values

Upsampling: Aumentar Frequência

Python

```python
# Criar dados mensais primeiro (downsample)
monthly = df['CO(GT)'].resample('M').mean()

print("=== DADOS MENSAIS ===")
print(f"Registros: {len(monthly)}")
print(monthly.head())

# Agora upsampling: mensal -> diario
daily_from_monthly = monthly.resample('D').asfreq()

print("\n=== APOS UPSAMPLING ===")
print(f"Registros: {len(daily_from_monthly)}")
print(daily_from_monthly.head(10))
```

Upsampling: Aumentar Frequência (cont.)

Python

```python
# Nota: Criou NaN! Precisa preencher

# Preencher com forward fill
daily_filled = monthly.resample('D').ffill()

print("\n=== COM FORWARD FILL ===")
print(daily_filled.head(10))
```

Upsampling com Interpolação

Python

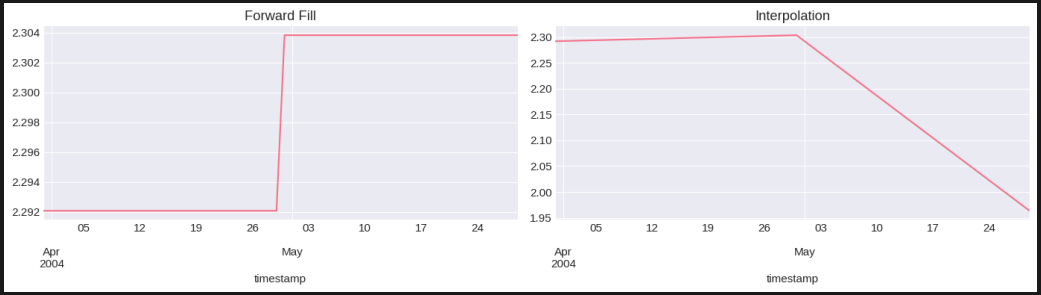
```python
# Upsampling com interpolacao (melhor!)

# Mensal -> Diario com interpolacao
daily_interp = monthly.resample('D').interpolate(method='linear')

print("=== UPSAMPLING COM INTERPOLACAO ===")
print(daily_interp.head(40))  # Primeiro mes

# Comparar metodos
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(14, 4))
```

Upsampling com Interpolação (cont.)

</> Python

```python
# Forward fill
monthly.resample('D').ffill().head(60).plot(
    ax=axes[0], title='Forward Fill'
)

# Interpolacao
monthly.resample('D').interpolate().head(60).plot(
    ax=axes[1], title='Interpolation'
)

plt.tight_layout()
plt.show()
```

Upsampling com Interpolação (cont.)



Rolling Windows: Conceito

Rolling = janela deslizante que calcula estatística

Como funciona:

`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

Janela de tamanho 3:

`[1, 2, 3]` $\rightarrow$ média = 2.0
`[2, 3, 4]` $\rightarrow$ média = 3.0
`[3, 4, 5]` $\rightarrow$ média = 4.0
...

Resultado:

`[NaN, NaN, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]`

Rolling Windows: Conceito

Rolling = janela deslizante que calcula estatística

Usos:

- Suavizar ruído (média móvel)
- Detectar tendências
- Calcular volatilidade (desvio móvel)
- Features para ML

Rolling: Média Móvel Simples

Python

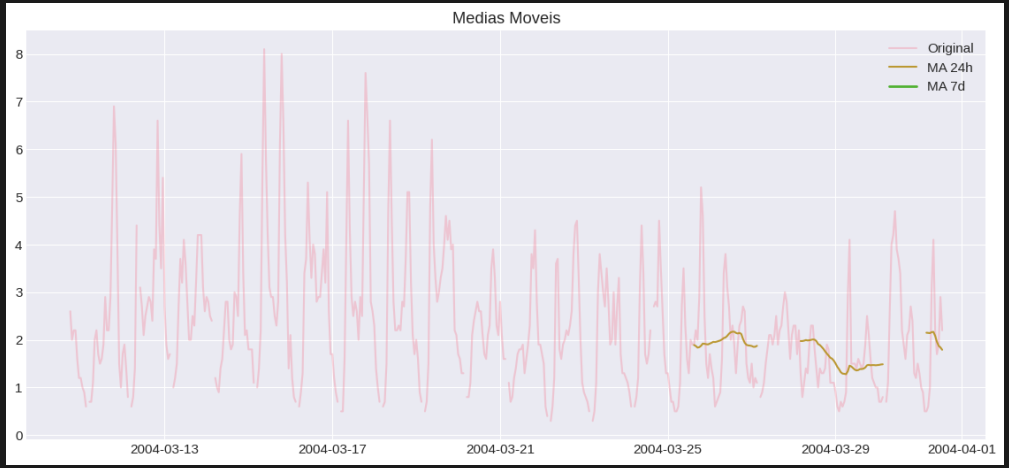
```python
# Janela de 24 horas (suavizar ruido diario)
df['CO_MA24'] = df['CO(GT)'].rolling(window=24).mean()

print("=== MEDIA MOVEL 24H ===")
print(df[['CO(GT)', 'CO_MA24']].head(30))

# Primeiras 23 serao NaN (janela incompleta)

# Janela de 7 dias (168 horas)
df['CO_MA168'] = df['CO(GT)'].rolling(window=168).mean()
```

Rolling: Média Móvel Simples (cont.)

Python

```python
# Visualizar
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 6))
plt.plot(df.index[:500], df['CO(GT)'][:500],
         alpha=0.3, label='Original')
plt.plot(df.index[:500], df['CO_MA24'][:500],
         label='MA 24h')
plt.plot(df.index[:500], df['CO_MA168'][:500],
         label='MA 7d', linewidth=2)
plt.legend()
plt.title('Medias Moveis')
plt.show()
```

Rolling: Média Móvel Simples (cont.)



Problemas no gráfico: dados faltants (NaN)

Original:

- Não possui NaN iniciais - série completa desde o primeiro timestamp
- Pode ter gaps esparsos devido a falhas no sensor (-200 convertido para NaN)

MA 24h:

- Primeiras 23 observações são NaN (janela incompleta)
- Requer 24 valores consecutivos para calcular a primeira média
- Visível apenas a partir da 24ª hora do dataset

MA 7d:

- Primeiras 167 observações são NaN (janela incompleta)
- Requer 168 valores (7 dias × 24 horas) para calcular a primeira média
- Visível apenas após 7 dias completos de dados
- Por isso não aparece no gráfico que mostra apenas [:500] (primeiras 21 dias)

Solução: usar `min_periods` (próximos slides)

Escolher Tamanho da Janela

Window size (tamanho da janela):

Janela pequena:

- Mais sensível a mudanças
- Segue dados mais de perto
- Mais ruído
- Exemplo: window=5

Janela grande:

- Mais suave
- Menos sensível
- Menos ruído
- Mais lag (atraso)
- Exemplo: window=100

Escolher Tamanho da Janela

Window size (tamanho da janela):

Regras práticas:

- Dados horários: 24 (1 dia) ou 168 (1 semana)
- Dados diários: 7 (1 semana) ou 30 (1 mês)
- Dados mensais: 12 (1 ano)
- Experimente! Visualize diferentes tamanhos

Rolling: Outras Estatísticas

Python

```python
# Desvio padrao movel (volatilidade)
df['CO_std24'] = df['CO(GT)'].rolling(window=24).std()

# Maximo movel
df['CO_max24'] = df['CO(GT)'].rolling(window=24).max()

# Minimo movel
df['CO_min24'] = df['CO(GT)'].rolling(window=24).min()

# Mediana movel (mais robusta)
df['CO_median24'] = df['CO(GT)'].rolling(window=24).median()

# Soma movel
df['CO_sum24'] = df['CO(GT)'].rolling(window=24).sum()
```

Rolling: Outras Estatísticas (cont.)

Python

```python
print("=== ROLLING STATISTICS ===")
print(df[['CO(GT)', 'CO_MA24', 'CO_std24',
          'CO_max24', 'CO_min24']].tail(10))

# Interpretacao:
# - std alta = alta variabilidade
# - max/min = range de variacao
```

Rolling: center=True

</> Python

```python
# Centrar a janela (ao inves de olhar pra tras)

# Padrao: olha para tras
df['CO_MA_backward'] = df['CO(GT)'].rolling(window=24).mean()

# Centrado: usa valores antes E depois
df['CO_MA_centered'] = df['CO(GT)'].rolling(window=24, center=True).mean()

print("=== BACKWARD VS CENTERED ===")
print(df[['CO(GT)', 'CO_MA_backward', 'CO_MA_centered']].iloc[20:30])
```

Rolling: center=True (cont.)

Python

```python
# Quando usar cada um?

# Backward (padrao): PREVISAO
# - Usa apenas passado
# - Pode usar em producao
# - Mais comum

# Centered: ANALISE/SUAVIZACAO
# - Usa passado + futuro
# - NAO pode usar para previsao!
# - Suavizacao mais precisa
```

Rolling: min_periods

Python

```python
# Define quantos valores válidos (não-NaN) são necessários no mínimo para
    calcular a estatística da janela móvel.

# Padrao: precisa janela completa
df['CO_MA24_strict'] = df['CO(GT)'].rolling(window=24).mean()
# Primeiras 23 = NaN

# Com min_periods: aceita janela parcial
df['CO_MA24_partial'] = df['CO(GT)'].rolling(
    window=24,
    min_periods=1  # Minimo de 1 valor valido
).mean()
# Todas tem valores!
print("=== MIN_PERIODS ===")
print(df[['CO(GT)', 'CO_MA24_strict', 'CO_MA24_partial']].head(25))
```

Rolling: min_periods (cont.)

Python

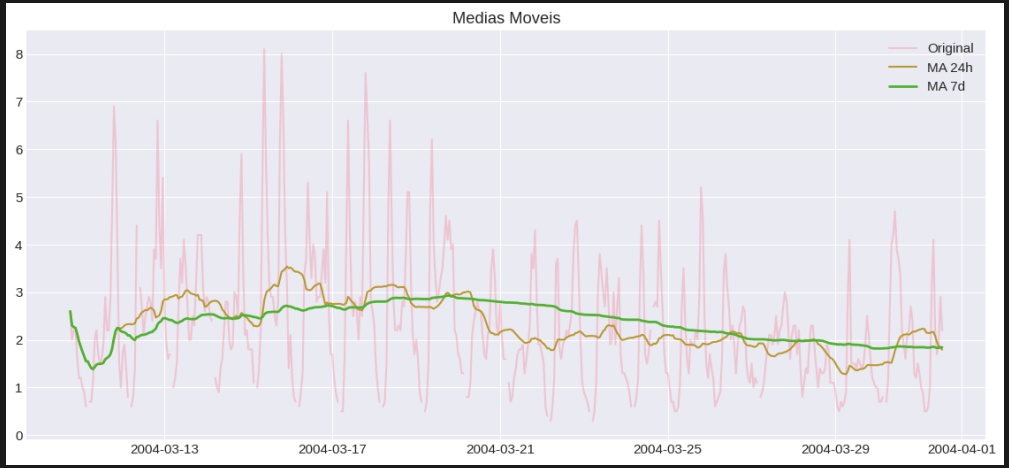
```python
# Quando usar min_periods?
# - Para evitar NaN no inicio
# - Quando poucos dados faltantes OK
# - Cuidado: janelas pequenas no inicio sao menos confiaveis
```

Rolling: Média Móvel Simples (cont.)

</> Python

```python
# Janela de 24 horas com min_periods
df['CO_MA24'] = df['CO(GT)'].rolling(window=24, min_periods=1).mean()

# Janela de 7 dias (168 horas) com min_periods
df['CO_MA168'] = df['CO(GT)'].rolling(window=168, min_periods=1).mean()

# Visualizar
...
```

Rolling: Média Móvel Simples (cont.)



Expanding: Estatísticas Cumulativas

Python

```python
# Expanding = janela que cresce desde o inicio

# Media cumulativa
df['CO_cumulative_mean'] = df['CO(GT)'].expanding().mean()

# No tempo t, usa TODOS os valores de 0 ate t

print("=== EXPANDING MEAN ===")
print(df[['CO(GT)', 'CO_cumulative_mean']].head(20))

# Outras estatisticas cumulativas
df['CO_cumulative_max'] = df['CO(GT)'].expanding().max()
df['CO_cumulative_min'] = df['CO(GT)'].expanding().min()
df['CO_cumulative_std'] = df['CO(GT)'].expanding().std()
```

Expanding: Estatísticas Cumulativas

Python

```python
# Uso:
# - Ver tendencia de longo prazo
# - Comparar valor atual com historico
# - Detectar mudancas de regime
```

Rolling vs Expanding vs Resample

Três técnicas, três propósitos:

Método	O que faz	Janela	Uso
Resample	Muda frequência	Período fixo	Agregar por dia/-mês
Rolling	Média móvel	Tamanho fixo, desliza	Suavizar, features
Expanding	Acumulativo	Cresce desde iní-cio	Tendência geral

Rolling vs Expanding vs Resample

Exemplo com 10 valores:

- **Resample('2D'):** 10 valores $\rightarrow$ 5 valores (média a cada 2 dias)
- **Rolling(3):** Janela [1,2,3], [2,3,4], ... (10 valores)
- **Expanding():** [1], [1,2], [1,2,3], ... (10 valores)

Shift: Defasagens (Lags)

</> Python

```python
# Shift: mover valores no tempo

# Lag 1 (valor de 1 hora atras)
df['CO_lag1'] = df['CO(GT)'].shift(1)

# Lag 24 (valor de 24 horas atras - ontem)
df['CO_lag24'] = df['CO(GT)'].shift(24)

# Lag 168 (7 dias atras)
df['CO_lag168'] = df['CO(GT)'].shift(168)

print("=== LAGS ===")
print(df[['CO(GT)', 'CO_lag1', 'CO_lag24', 'CO_lag168']].head(30))
```

Shift: Defasagens (Lags) (cont.)

</> Python

```python
# Uso:
# - Autocorrelacao: correlacao com proprio passado
# - Features para ML: valor de ontem ajuda prever hoje
# - Comparacoes temporais

# Shift negativo = lead (valores futuros)
df['CO_lead1'] = df['CO(GT)'].shift(-1)  # Proxima hora
# CUIDADO: data leakage se usar para previsao!
```

Diff: Diferenças Temporais

Python

```python
# Diff: diferenca entre periodos consecutivos

# Diferenca de 1 periodo (mudanca horaria)
df['CO_diff1'] = df['CO(GT)'].diff(1)
# CO_diff1[t] = CO[t] - CO[t-1]

# Diferenca de 24 periodos (mudanca dia-a-dia)
df['CO_diff24'] = df['CO(GT)'].diff(24)
# CO_diff24[t] = CO[t] - CO[t-24]

print("=== DIFFERENCES ===")
print(df[['CO(GT)', 'CO_diff1', 'CO_diff24']].head(30))
```

Diff: Diferenças Temporais (cont.)

Python

```python
# Interpretacao:
# - diff > 0: aumentou
# - diff < 0: diminuiu
# - diff = 0: estavel

# Uso:
# - Tornar serie estacionaria
# - Detectar mudancas abruptas
# - Velocidade de mudanca
```

pct_change: Variações Percentuais

Python

```python
# Variacao percentual de 1 periodo
df['CO_pct_change1'] = df['CO(GT)'].pct_change(1)
# pct_change[t] = (CO[t] - CO[t-1]) / CO[t-1]

# Variacao percentual dia-a-dia
df['CO_pct_change24'] = df['CO(GT)'].pct_change(24)

print("=== PERCENT CHANGES ===")
print(df[['CO(GT)', 'CO_pct_change1', 'CO_pct_change24']].head(30))
```

pct_change: Variações Percentuais (cont.)

</> Python

```python
# Interpretacao:
# - 0.1 = +10% de aumento
# - -0.05 = -5% de queda
# - 0.0 = sem mudanca

# Converter para percentual
df['CO_pct_change24_formatted'] = df['CO_pct_change24'] * 100

print(f"\nVariacao percentual media dia-a-dia: {df['CO_pct_change24'].mean()*100:.2f}%")
```

Autocorrelação: Correlação com o Passado

Python

```python
# Medir correlacao com valores defasados

# Criar lags
for lag in [1, 6, 12, 24, 168]:
    df[f'CO_lag{lag}'] = df['CO(GT)'].shift(lag)

# Calcular correlacoes
correlations = []
for lag in [1, 6, 12, 24, 168]:
    corr = df['CO(GT)'].corr(df[f'CO_lag{lag}'])
    correlations.append((lag, corr))
    print(f"Lag {lag:3d}h: correlacao = {corr:.3f}")
```

Autocorrelação: Correlação com o Passado (cont.)

Python

```python
# Interpretacao:
# - Lag 1h: correlacao alta (0.9+) = muito autocorrelacionado
# - Lag 24h: correlacao alta = padrao diario forte
# - Lag 168h (7d): correlacao = padrao semanal

# Usar para:
# - Identificar periodicidade
# - Escolher lags para features
```

Plotar Autocorrelação

</> Python

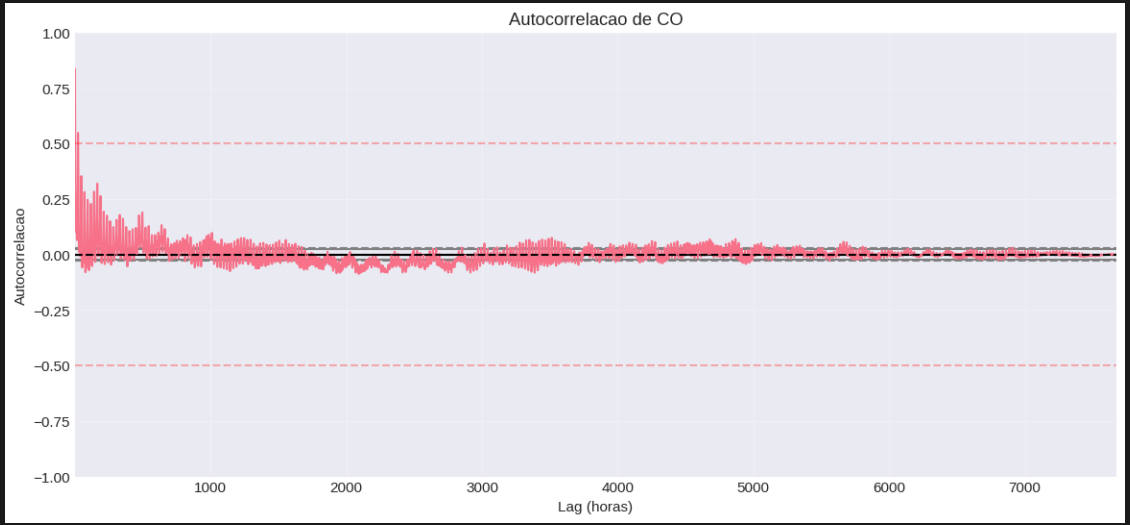
```python
from pandas.plotting import autocorrelation_plot

# Plot de autocorrelacao
plt.figure(figsize=(14, 6))
autocorrelation_plot(df['CO(GT)'].dropna())
plt.title('Autocorrelacao de CO')
plt.xlabel('Lag (horas)')
plt.ylabel('Autocorrelacao')
plt.axhline(y=0, color='k', linestyle='--')
plt.axhline(y=0.5, color='r', linestyle='--', alpha=0.3)
plt.axhline(y=-0.5, color='r', linestyle='--', alpha=0.3)
plt.grid(True, alpha=0.3)
plt.show()
```

Plotar Autocorrelação (cont.)

Python

```python
# Linhas horizontais: limites de significancia
# Se autocorr > linha: correlacao significativa
# Picos em multiplos de 24: sazonalidade diaria
# Picos em multiplos de 168: sazonalidade semanal
```

Plotar Autocorrelação (cont.)



Criar Features com Rolling e Lags

Python

```python
# Criar features uteis para modelagem

# 1. Lags importantes
df['CO_lag1'] = df['CO(GT)'].shift(1)
df['CO_lag24'] = df['CO(GT)'].shift(24)
df['CO_lag168'] = df['CO(GT)'].shift(168)

# 2. Medias moveis
df['CO_MA24'] = df['CO(GT)'].rolling(24).mean()
df['CO_MA168'] = df['CO(GT)'].rolling(168).mean()

# 3. Diferenca entre valor e media
df['CO_vs_MA24'] = df['CO(GT)'] - df['CO_MA24']
```

Criar Features com Rolling e Lags (cont.)

Python

```python
# 4. Volatilidade (desvio movel)
df['CO_volatility'] = df['CO(GT)'].rolling(24).std()

# 5. Range (max - min)
df['CO_range24'] = (df['CO(GT)'].rolling(24).max() -
                    df['CO(GT)'].rolling(24).min())

print("=== FEATURES CRIADAS ===")
print(df[['CO(GT)', 'CO_lag24', 'CO_MA24',
          'CO_vs_MA24', 'CO_volatility']].tail())
```

Features para Forecasting com Dados Históricos

Features mais úteis para previsão (criadas com valores passados):

1. **Lags (valores passados):**
   - lag1, lag24, lag168 (1h, 1d, 1sem atrás)
   - Use autocorrelação para escolher
2. **Médias móveis:**
   - MA7, MA30 (tendência de curto/longo prazo)
   - Diferença entre valor atual e MA
3. **Volatilidade:**
   - Desvio padrão móvel
   - Range móvel (max - min)

Features para Forecasting com Dados Históricos

Features mais úteis para previsão (criadas com valores passados):

4. Diferenças:

- diff1, diff24 (taxa de mudança)
- pct_change (variação percentual)

5. Componentes temporais:

- hour, dayofweek, month
- is_weekend, is_holiday

Nota Importante

Estas features são úteis para entender padrões e relações temporais nos dados!

Ewm: Exponential Weighted Moving Average

</> Python

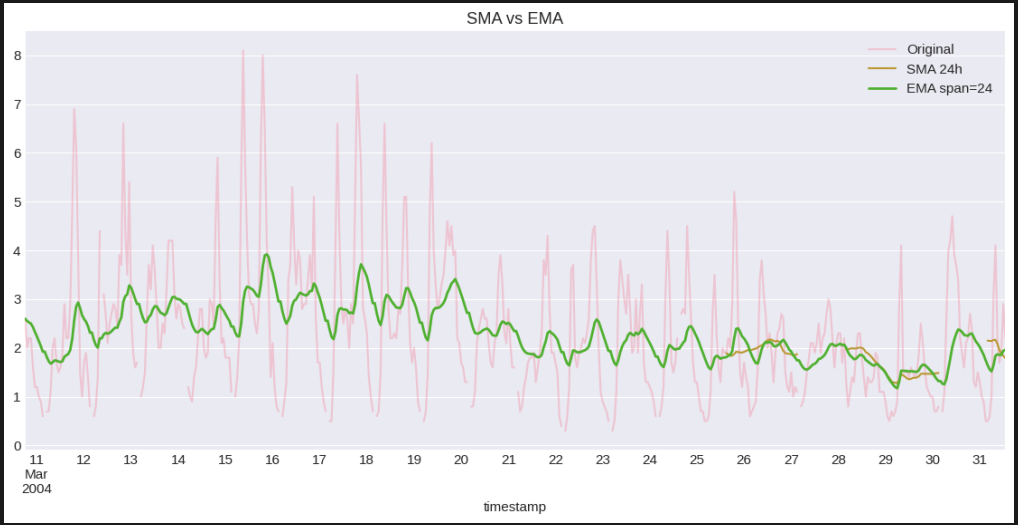
```python
# EWM: da mais peso aos valores recentes

# Media movel simples (SMA)
df['CO_SMA24'] = df['CO(GT)'].rolling(24).mean()

# Media movel exponencial (EMA/EWMA)
df['CO_EMA'] = df['CO(GT)'].ewm(span=24, adjust=False).mean()

# Comparar
print("=== SMA vs EMA ===")
print(df[['CO(GT)', 'CO_SMA24', 'CO_EMA']].tail(20))
```

Ewm: Exponential Weighted Moving Average (cont.)

</> Python

```python
# Visualizar
plt.figure(figsize=(14, 6))
df['CO(GT)'][:500].plot(alpha=0.3, label='Original')
df['CO_SMA24'][:500].plot(label='SMA 24h')
df['CO_EMA'][:500].plot(label='EMA span=24', linewidth=2)
plt.legend()
plt.title('SMA vs EMA')
plt.show()

# EMA: mais responsiva a mudancas recentes
# SMA: trata todos valores igualmente
```

Ewm: Exponential Weighted Moving Average (cont.)



SMA vs EMA: Diferenças

Simple Moving Average (SMA):

- Peso igual para todos os valores na janela
- Fórmula: $(x_1 + x_2 + ... + x_n)/n$
- Mais suave
- Lag maior

Exponential Moving Average (EMA):

- Peso decai exponencialmente
- Mais peso aos valores recentes
- Menos lag
- Mais responsiva

SMA vs EMA: Diferenças

Quando usar cada um:

- **SMA:** Análise de tendências de longo prazo
- **EMA:** Trading, detecção rápida de mudanças
- **Ambos:** Cruzamento de médias (sinais de compra/venda)

Aplicar Função Customizada em Rolling

Python

```python
# Aplicar funcao customizada

# Funcao: range (max - min)
def window_range(window):
    return window.max() - window.min()

# Aplicar
df['CO_range_custom'] = df['CO(GT)'].rolling(24).apply(window_range)

# Funcao: quantil 75%
df['CO_q75'] = df['CO(GT)'].rolling(24).quantile(0.75)
```

Aplicar Função Customizada em Rolling (cont.)

</> Python

```python
# Funcao: contagem de valores acima da media
def count_above_mean(window):
    return (window > window.mean()).sum()

df['CO_count_above'] = df['CO(GT)'].rolling(24).apply(count_above_mean)

print("=== FUNCOES CUSTOMIZADAS ===")
print(df[['CO(GT)', 'CO_range_custom',
          'CO_q75', 'CO_count_above']].tail())

# Usar .apply() para qualquer logica customizada!
```

Comparar Diferentes Janelas

Python

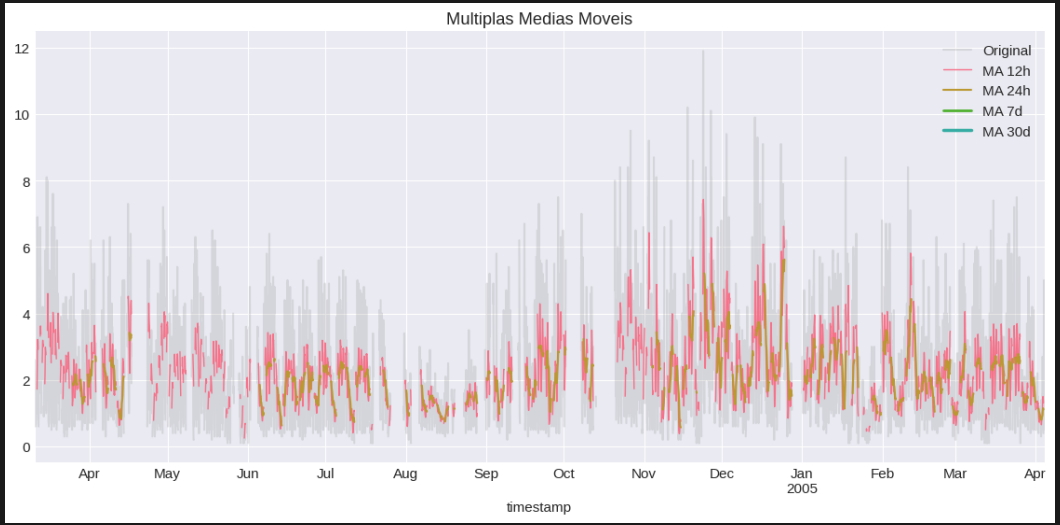
```python
# Multiplas janelas para capturar diferentes escalas

# Curto prazo (12h)
df['CO_MA12'] = df['CO(GT)'].rolling(12).mean()

# Medio prazo (24h)
df['CO_MA24'] = df['CO(GT)'].rolling(24).mean()

# Longo prazo (168h = 7 dias)
df['CO_MA168'] = df['CO(GT)'].rolling(168).mean()

# Muito longo prazo (720h = 30 dias)
df['CO_MA720'] = df['CO(GT)'].rolling(720).mean()
```

Comparar Diferentes Janelas (cont.)

Python

```python
# Visualizar todas
plt.figure(figsize=(14, 6))
df['CO(GT)'].plot(alpha=0.2, label='Original', color='gray')
df['CO_MA12'].plot(label='MA 12h', linewidth=1)
df['CO_MA24'].plot(label='MA 24h', linewidth=1.5)
df['CO_MA168'].plot(label='MA 7d', linewidth=2)
df['CO_MA720'].plot(label='MA 30d', linewidth=2.5)
plt.legend()
plt.title('Multiplas Medias Moveis')
plt.show()
```

Comparar Diferentes Janelas (cont.)



Detectar Anomalias com Rolling

Python

```python
# Detectar valores anormais usando rolling statistics

# Media e desvio movel
df['CO_rolling_mean'] = df['CO(GT)'].rolling(24).mean()
df['CO_rolling_std'] = df['CO(GT)'].rolling(24).std()

# Z-score movel
df['CO_zscore'] = (df['CO(GT)'] - df['CO_rolling_mean']) / df['
    CO_rolling_std']

# Anomalias: |z-score| > 3
df['is_anomaly'] = (df['CO_zscore'].abs() > 3).astype(int)
```

Detectar Anomalias com Rolling (cont.)

Python

```python
print("=== ANOMALIAS DETECTADAS ===")
anomalies = df[df['is_anomaly'] == 1]
print(f"Total de anomalias: {len(anomalies)}")
print(f"Percentual: {(len(anomalies)/len(df))*100:.2f}%")

print("\nExemplos:")
print(anomalies[['CO(GT)', 'CO_rolling_mean',
                 'CO_zscore']].head())

# Visualizar
# ...
```

Exercício: Análise com Rolling e Resample

✋ Exercício Prático

Explore agregações temporais no Air Quality:

1. Resample: diaria e semanal; comparar com dados horários
2. Rolling windows:
   - Calcular MA de 24h e 168h
   - Plotar original vs médias móveis
   - Calcular desvio padrão móvel
3. Features:
   - Criar lags (1h, 24h, 168h)
   - Criar diferenças (diff)
   - Calcular autocorrelação

Exercício: Análise com Rolling e Resample (cont.)

Exercício Prático

Explore agregações temporais no Air Quality:

4. Detecção de anomalias:
   - Use z-score móvel
   - Identifique top 10 anomalias
   - Quando ocorreram?

Bloco 3

Decomposição de Séries Temporais

O que é Decomposição?

Decomposição = separar série em componentes

Modelo aditivo:

$$Y_t = T_t + S_t + R_t$$

Modelo multiplicativo:

$$Y_t = T_t \times S_t \times R_t$$

Componentes:

- T_t **(Trend):** Tendência de longo prazo
- S_t **(Seasonal):** Padrão sazonal repetitivo
- R_t **(Residual):** Ruído/irregular

O que é Decomposição?

Decomposição = separar série em componentes

Modelo aditivo:

$$Y_t = T_t + S_t + R_t$$

Modelo multiplicativo:

$$Y_t = T_t \times S_t \times R_t$$

Quando usar cada modelo:

- **Aditivo:** Amplitude sazonal constante
- **Multiplicativo:** Amplitude sazonal cresce com tendência

Aditivo vs Multiplicativo

Exemplo visual:

Modelo Aditivo:

- Vendas oscilam ±100 unidades em jan/2020
- Vendas oscilam ±100 unidades em dez/2024
- Amplitude sazonal CONSTANTE
- Variação independente do nível

Modelo Multiplicativo:

- Vendas oscilam ±10% em jan/2020 (1000 ± 100)
- Vendas oscilam ±10% em dez/2024 (5000 ± 500)
- Amplitude sazonal CRESCE
- Variação proporcional ao nível

Aditivo vs Multiplicativo

Regra prática:

- Se em dúvida, comece com aditivo
- Se amplitude cresce, use multiplicativo
- Ou aplique log e use aditivo

Decomposição: Preparação dos Dados

Python

```python
# Preparar dados para decomposicao

# 1. Resample para diario (decomposicao funciona melhor em freq regular)
daily = df['CO(GT)'].resample('D').mean()

# 2. Remover NaN
daily = daily.dropna()

print("=== DADOS PARA DECOMPOSICAO ===")
print(f"Frequencia: Diaria")
print(f"Registros: {len(daily)}")
print(f"Periodo: {daily.index.min()} a {daily.index.max()}")
print(f"Missing: {daily.isnull().sum()}")
```

Decomposição: Preparação dos Dados

</> Python

```python
# Verificar
print(f"\nAmostra:")
print(daily.head())

# IMPORTANTE: Decomposicao precisa de:
# - Frequencia regular (sem gaps)
# - Sem missing values
# - Periodo suficiente (pelo menos 2 ciclos sazonais)
```

Decomposição: statsmodels

</> Python

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# Decomposicao aditiva
decomposition = seasonal_decompose(
    daily,
    model='additive',
    period=7,  # Sazonalidade semanal (7 dias)
    extrapolate_trend='freq'
)

# Extrair componentes
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

Decomposição: statsmodels (cont.)

Python

```python
print("=== COMPONENTES ===")
print(f"Trend: {trend.shape}")
print(f"Seasonal: {seasonal.shape}")
print(f"Residual: {residual.shape}")

# Verificar decomposicao
# Original = Trend + Seasonal + Residual
reconstructed = trend + seasonal + residual
print(f"\nErro de reconstrucao: {(daily - reconstructed).abs().max():.10f}
    ")
```

Visualizar Decomposição

</> Python

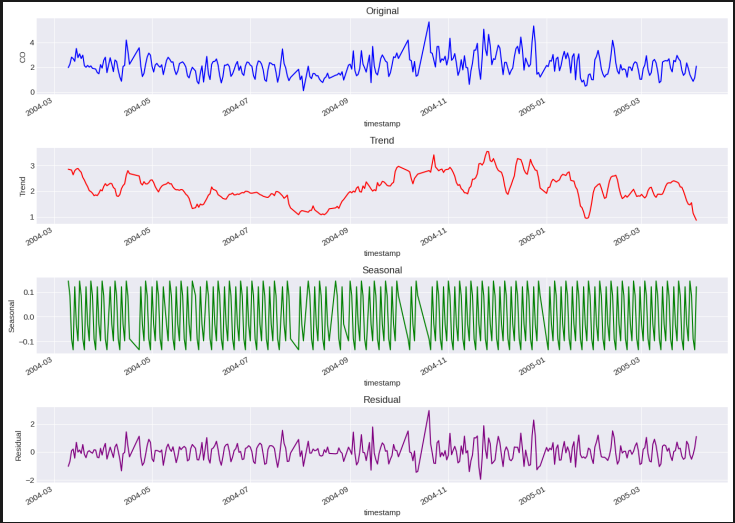
```python
import matplotlib.pyplot as plt

# Plot de decomposicao
fig = decomposition.plot()
fig.set_size_inches(14, 10)
plt.tight_layout()
plt.show()

# Ou manualmente com mais controle
fig, axes = plt.subplots(4, 1, figsize=(14, 10))

# Original
daily.plot(ax=axes[0], title='Original', color='blue')
axes[0].set_ylabel('CO')
```

Visualizar Decomposição (cont.)

Python

```python
# Tendencia
trend.plot(ax=axes[1], title='Trend', color='red')
axes[1].set_ylabel('Trend')

# Sazonalidade
seasonal.plot(ax=axes[2], title='Seasonal', color='green')
axes[2].set_ylabel('Seasonal')

# Residual
residual.plot(ax=axes[3], title='Residual', color='purple')
axes[3].set_ylabel('Residual')

plt.tight_layout()
plt.show()
```

Visualizar Decomposição (cont.)



Interpretar Componentes

O que cada componente revela:

1. Tendência (Trend):

- Direção de longo prazo
- Crescimento ou declínio
- Ciclos muito longos
- Se está subindo/descendo/estável

2. Sazonalidade (Seasonal):

- Padrão que se repete
- Picos e vales regulares
- Período fixo (diário, semanal, anual)
- Quando esperar valores altos/baixos

Interpretar Componentes

O que cada componente revela:

3. Resíduo (Residual):

- O que não foi explicado
- Ruído aleatório
- Eventos irregulares
- Deve ser aleatório (sem padrão)

Analisar Tendência

Python

```python
# Analise do componente de tendencia

print("=== ANALISE DE TENDENCIA ===")

# Direcao geral
start_value = trend.dropna().iloc[0]
end_value = trend.dropna().iloc[-1]
change = end_value - start_value
pct_change = (change / start_value) * 100

print(f"Valor inicial: {start_value:.2f}")
print(f"Valor final: {end_value:.2f}")
print(f"Mudanca: {change:.2f} ({pct_change:+.1f}%)")
```

Analisar Tendência (cont.)

Python

```python
if pct_change > 5:
    print("Tendencia de CRESCIMENTO")
elif pct_change < -5:
    print("Tendencia de DECLINIO")
else:
    print("Tendencia ESTAVEL")

# Taxa de crescimento media
growth_rate = trend.pct_change().mean() * 100
print(f"\nTaxa de crescimento diaria media: {growth_rate:.3f}%")
```

Analisar Tendência (cont.)

Python

```
# === ANALISE DE TENDENCIA ===
# Valor inicial: 2.85
# Valor final: 0.87
# Mudanca: -1.99 (-69.6%)
# Tendencia de DECLINIO
# Taxa de crescimento diaria media: -0.083%
```

Analisar Sazonalidade

Python

```python
# Analise do componente sazonal

print("=== ANALISE DE SAZONALIDADE ===")

# Amplitude sazonal
seasonal_amplitude = seasonal.max() - seasonal.min()
print(f"Amplitude sazonal: {seasonal_amplitude:.2f}")

# Percentual da variacao total
total_variation = daily.std()
seasonal_strength = (seasonal.std() / total_variation) * 100
print(f"Forca da sazonalidade: {seasonal_strength:.1f}% da variacao")
```

Analisar Sazonalidade (cont.)

</> Python

```python
# Padrao semanal (se period=7)
# Media por dia da semana
seasonal_pattern = seasonal.groupby(seasonal.index.dayofweek).mean()
seasonal_pattern.index = ['Seg', 'Ter', 'Qua', 'Qui',
                          'Sex', 'Sab', 'Dom']
print("\nPadrao semanal (media):")
print(seasonal_pattern)

# Dia com maior/menor valor sazonal
print(f"\nPior dia: {seasonal_pattern.idxmax()}")
print(f"Melhor dia: {seasonal_pattern.idxmin()}")
```

Analisar Resíduo

Python

```python
# Analise do residuo

print("=== ANALISE DE RESIDUO ===")

# Estatisticas
print(f"Media: {residual.mean():.6f}")  # Deve ser ~0
print(f"Desvio padrao: {residual.std():.3f}")
print(f"Min: {residual.min():.3f}")
print(f"Max: {residual.max():.3f}")

# Outliers no residuo (eventos incomuns)
residual_threshold = 3 * residual.std()
outliers = residual[residual.abs() > residual_threshold]
print(f"\nOutliers (|residual| > 3): {len(outliers)}")
```

Analisar Resíduo (cont.)

Python

```python
if len(outliers) > 0:
    print("\nMaiores residuos:")
    top_outliers = residual.abs().nlargest(5)
    for date, value in top_outliers.items():
        print(f"  {date.date()}: {value:.3f}")

# Teste de aleatoriedade (autocorrelacao do residuo)
residual_autocorr = residual.autocorr(lag=1)
print(f"\nAutocorrelacao lag-1: {residual_autocorr:.3f}")
print("(Deve ser proximo de 0 se residuo é aleatorio)")
```

Comparar Aditivo vs Multiplicativo

Python

```python
# Decompor com ambos os modelos

# Aditivo
decomp_add = seasonal_decompose(daily, model='additive',
                                period=7, extrapolate_trend='freq')

# Multiplicativo
decomp_mult = seasonal_decompose(daily, model='multiplicative',
                                 period=7, extrapolate_trend='freq')

# Comparar residuos (qual modelo se ajusta melhor?)
residual_add = decomp_add.resid
residual_mult = decomp_mult.resid
```

Comparar Aditivo vs Multiplicativo (cont.)

</> Python

```python
print("=== COMPARACAO DE MODELOS ===")
print(f"Aditivo - Residuo std: {residual_add.std():.3f}")
print(f"Multiplicativo - Residuo std: {residual_mult.std():.3f}")

# Menor desvio = melhor ajuste
if residual_add.std() < residual_mult.std():
    print("\nModelo ADITIVO se ajusta melhor")
else:
    print("\nModelo MULTIPLICATIVO se ajusta melhor") # É o caso aqui
```

Escolher Período de Sazonalidade

Python

```python
# Como escolher o periodo correto?

# Para dados diarios:
# - period=7: sazonalidade SEMANAL
# - period=30: sazonalidade MENSAL
# - period=365: sazonalidade ANUAL

# Para dados horarios:
# - period=24: sazonalidade DIARIA
# - period=168: sazonalidade SEMANAL (7*24)

# Testar diferentes periodos
```

Escolher Período de Sazonalidade

</> Python

```python
periods_to_test = [7, 14, 30]

for period in periods_to_test:
    decomp = seasonal_decompose(daily, model='additive',
                                period=period,
                                extrapolate_trend='freq')
    residual_std = decomp.resid.std()
    print(f"Period={period:2d}: Residual std = {residual_std:.4f}")

print("\nMenor residual std = melhor periodo")

# Ou use conhecimento do dominio:
# - Qualidade do ar: padrao semanal (trabalho vs fim de semana)
```

Remover Tendência (Detrending)

Python

```python
# Remover tendencia da serie original

# Opcao 1: Subtrair tendencia (aditivo)
detrended = daily - trend

print("=== SERIE SEM TENDENCIA ===")
print(detrended.head())

# Opcao 2: Dividir por tendencia (multiplicativo)
detrended_mult = daily / trend

# Visualizar
fig, axes = plt.subplots(2, 1, figsize=(14, 8))
```

Remover Tendência (Detrending)

Python

```python
# Original
daily.plot(ax=axes[0], title='Original', color='blue')

# Sem tendencia (destaca sazonalidade)
detrended.plot(ax=axes[1], title='Detrended', color='green')

plt.tight_layout()
plt.show()

# Por que remover tendencia?
# - Analise de sazonalidade mais clara
# - Tornar serie estacionaria
# - Pre-processamento para modelagem
```

Remover Sazonalidade (Seasonal Adjustment)

</> Python

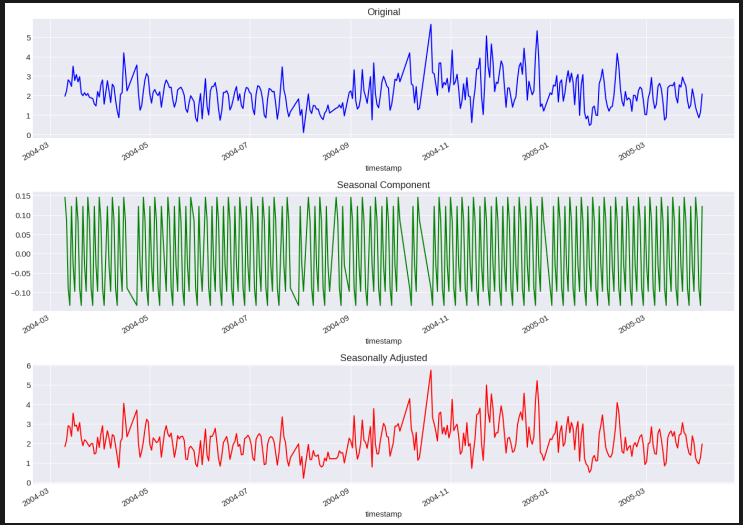
```python
# Remover sazonalidade (dessazonalizar)

# Serie ajustada = Original - Sazonal (aditivo)
seasonally_adjusted = daily - seasonal

print("=== SERIE DESSAZONALIZADA ===")
print(seasonally_adjusted.head())

# Visualizar
fig, axes = plt.subplots(3, 1, figsize=(14, 10))

# Original
daily.plot(ax=axes[0], title='Original', color='blue')
```

Remover Sazonalidade (Seasonal Adjustment) (cont.)

</> Python

```python
# Sazonalidade
seasonal.plot(ax=axes[1], title='Seasonal Component', color='green')

# Ajustada (sem sazonalidade)
seasonally_adjusted.plot(ax=axes[2],
                         title='Seasonally Adjusted',
                         color='red')
plt.tight_layout()
plt.show()
# Uso: ver tendencia sem oscilacoes sazonais
```

Remover Sazonalidade (Seasonal Adjustment) (cont.)



Aplicações da Decomposição

Por que decompor séries temporais?

1. Entendimento:

- Identificar padrões ocultos
- Separar efeitos de curto vs longo prazo
- Comunicar insights ao negócio

2. Análise:

- Medir força da sazonalidade
- Detectar mudanças de tendência
- Encontrar eventos anômalos (resíduo)

Aplicações da Decomposição

Por que decompor séries temporais?

3. Previsão:

- Prever cada componente separadamente
- Modelos específicos para cada parte
- Combinar previsões

4. Pré-processamento:

- Remover tendência $\rightarrow$ estacionariedade
- Ajuste sazonal $\rightarrow$ comparações justas
- Isolar ruído $\rightarrow$ detecção de anomalias

Limitações da Decomposição

Cuidados importantes:

1. Assume periodicidade constante:

- Sazonalidade deve ter período fixo
- Não funciona se período muda

2. Perda de dados nas pontas:

- Tendência usa média móvel
- NaN no início/fim da série
- Use extrapolate_trend='freq'

Limitações da Decomposição

Cuidados importantes:

3. Escolha do período é crítica:

- Período errado $\rightarrow$ decomposição sem sentido
- Precisa conhecimento do domínio

4. Múltiplas sazonalidades:

- Decomposição clássica captura apenas UMA
- Exemplo: diária E semanal E anual
- Precisa métodos avançados (STL, TBATS)

Bloco 4

Visualizações e Boas Práticas

Visualização 1: Line Plot Básico

Python

```python
# Plot de linha: padrao para series temporais

plt.figure(figsize=(14, 6))
plt.plot(daily.index, daily.values, linewidth=1, color='steelblue')
plt.title('CO (daily average)', fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('CO (mg/m³)', fontsize=12)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Visualização 1: Line Plot Básico (cont.)

Python

```python
# Boas praticas:
# - Sempre rotular eixos com unidades
# - Titulo descritivo
# - Grid para facilitar leitura
# - Tamanho adequado (14x6 ou 16x6)
# - Linha nao muito grossa (linewidth=1 ou 1.5)
```

Visualização 2: Comparar Múltiplas Séries

Python

```python
# Comparar varias series no mesmo plot

daily_multi = df[['CO(GT)', 'NO2(GT)', 'T']].resample('D').mean()

fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# CO
axes[0].plot(daily_multi.index, daily_multi['CO(GT)'],
             color='red', linewidth=1)
axes[0].set_ylabel('CO (mg/m³)')
axes[0].set_title('Carbon Monoxide')
axes[0].grid(True, alpha=0.3)
```

Visualização 2: Comparar Múltiplas Séries (cont.)

Python

```python
# NO2
axes[1].plot(daily_multi.index, daily_multi['NO2(GT)'],
             color='blue', linewidth=1)
axes[1].set_ylabel('NO2 (µg/m³)')
axes[1].set_title('Nitrogen Dioxide')
axes[1].grid(True, alpha=0.3)
# Temperatura
axes[2].plot(daily_multi.index, daily_multi['T'],
             color='green', linewidth=1)
axes[2].set_ylabel('Temperature (°C)')
axes[2].set_title('Temperature')
axes[2].grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Visualização 3: Heatmap Temporal

Python

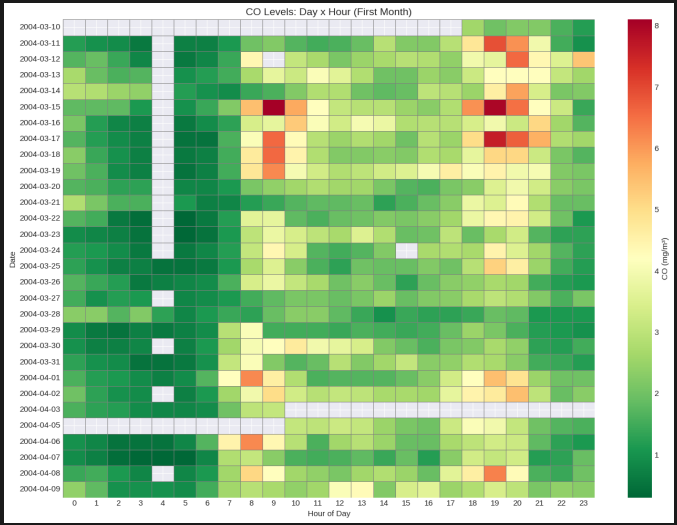
```python
import seaborn as sns

# Heatmap: dia x hora

# Criar matriz: linhas=dias, colunas=horas
pivot = df.pivot_table(
    values='CO(GT)',
    index=df.index.date,
    columns=df.index.hour,
    aggfunc='mean'
)
```

Visualização 3: Heatmap Temporal (cont.)

Python

```python
# Plot
plt.figure(figsize=(14, 10))
sns.heatmap(pivot[:30],  # Primeiro mes
            cmap='RdYlGn_r',  # Vermelho=alto, Verde=baixo
            cbar_kws={'label': 'CO (mg/m³)'},
            linewidths=0.5,
            linecolor='gray')
plt.title('CO Levels: Day x Hour (First Month)', fontsize=14)
plt.xlabel('Hour of Day')
plt.ylabel('Date')
plt.tight_layout()
plt.show()
# Revela: padroes diarios e semanais juntos!
```

Visualização 3: Heatmap Temporal (cont.)



Visualização 4: Box Plot por Período

Python

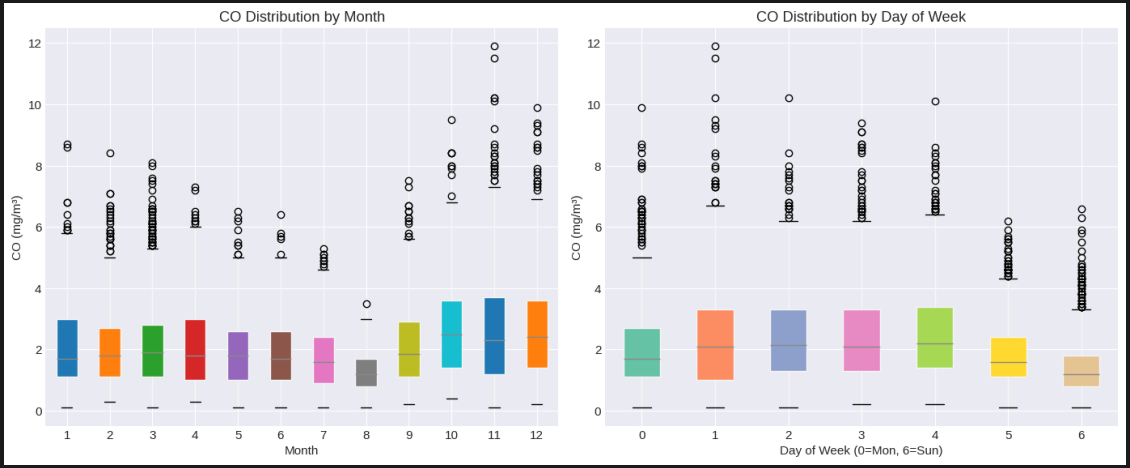
```python
# Box plots para ver distribuicao por periodo

# Por mes
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Box plot por mes
df['month'] = df.index.month
df.boxplot(column='CO(GT)', by='month', ax=axes[0])
axes[0].set_xlabel('Month')
axes[0].set_ylabel('CO (mg/m³)')
axes[0].set_title('CO Distribution by Month')
axes[0].get_figure().suptitle('')  # Remove titulo auto
```

Visualização 4: Box Plot por Período (cont.)

Python

```python
# Box plot por dia da semana
df['dayofweek'] = df.index.dayofweek
df.boxplot(column='CO(GT)', by='dayofweek', ax=axes[1])
axes[1].set_xlabel('Day of Week (0=Mon, 6=Sun)')
axes[1].set_ylabel('CO (mg/m³)')
axes[1].set_title('CO Distribution by Day of Week')
axes[1].get_figure().suptitle('')

plt.tight_layout()
plt.show()
```

Visualização 4: Box Plot por Período (cont.)



Visualização 5: Lag Scatter Plot

Python

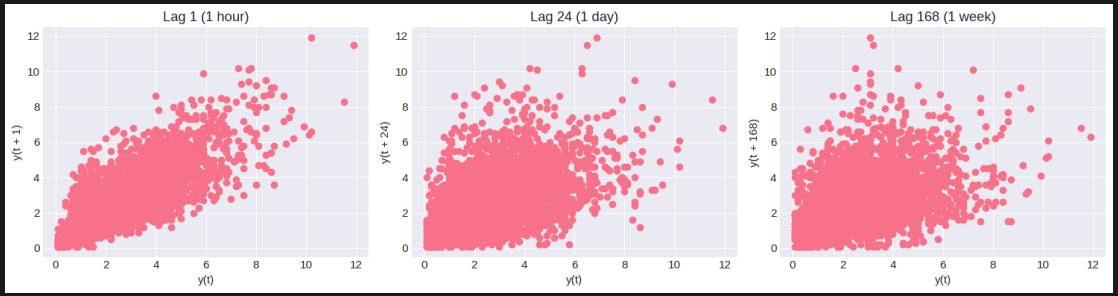
```python
from pandas.plotting import lag_plot

# Lag plot: hoje vs ontem

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Lag 1 (1 hora atras)
lag_plot(df['CO(GT)'], lag=1, ax=axes[0])
axes[0].set_title('Lag 1 (1 hour)')

# Lag 24 (1 dia atras)
lag_plot(df['CO(GT)'], lag=24, ax=axes[1])
axes[1].set_title('Lag 24 (1 day)')
```

Visualização 5: Lag Scatter Plot (cont.)

Python

```python
# Lag 168 (1 semana atras)
lag_plot(df['CO(GT)'], lag=168, ax=axes[2])
axes[2].set_title('Lag 168 (1 week)')
plt.tight_layout()
plt.show()

# Interpretacao:
# - Pontos em linha diagonal = alta correlacao
# - Pontos dispersos = baixa correlacao
```

Visualização 5: Lag Scatter Plot (cont.)



Boas Práticas: Pipeline Completo

Python

```python
def analyze_timeseries(df, col, freq='D'):
    """Pipeline completo de analise temporal"""

    # 1. Preparar
    series = df[col].resample(freq).mean().dropna()

    # 2. Estatisticas basicas
    print("=== ESTATISTICAS ===")
    print(series.describe())

    # 3. Decomposicao
    from statsmodels.tsa.seasonal import seasonal_decompose
    decomp = seasonal_decompose(series, model='additive',
                                period=7, extrapolate_trend='freq')
```

Boas Práticas: Pipeline Completo (cont.)

Python

```python
    # 4. Visualizacoes
    fig, axes = plt.subplots(5, 1, figsize=(14, 12))
    series.plot(ax=axes[0], title='Original')
    decomp.trend.plot(ax=axes[1], title='Trend')
    decomp.seasonal.plot(ax=axes[2], title='Seasonal')
    decomp.resid.plot(ax=axes[3], title='Residual')
    # Autocorrelacao
    from pandas.plotting import autocorrelation_plot
    autocorrelation_plot(series, ax=axes[4])
    plt.tight_layout()
    plt.show()
    return decomp
# Usar
decomp = analyze_timeseries(df, 'CO(GT)')
```

Antes de analisar séries temporais: Checklist

1. ☐ **DatetimeIndex configurado**
   - Coluna datetime como índice
   - Ordenado temporalmente
2. ☐ **Dados limpos**
   - Missing values tratados
   - Outliers investigados
   - Duplicatas removidas
3. ☐ **Frequência regular**
   - Sem gaps temporais (ou preenchidos)
   - Frequência consistente
4. ☐ **Exploração inicial**
   - Visualizar série completa
   - Estatísticas descritivas
   - Identificar padrões visuais
5. ☐ **Componentes identificados**
   - Tendência analisada
   - Sazonalidade detectada
   - Período determinado

Erros Comuns - Evite!

1. **Não definir DatetimeIndex**
   - ✗ Manter data como coluna comum
   - ✓ Usar set_index()
2. **Ignorar ordem temporal**
   - ✗ Shuffle ou embaralhar dados
   - ✓ Sempre manter ordem cronológica
3. **Validação em períodos aleatórios**
   - ✗ Embaralhar dados e testar em períodos passados
   - ✓ Sempre validar em períodos futuros (cronológico)
   - ✓ Exemplo: 80% inicial para análise, 20% final para validação

Erros Comuns - Evite!

4. **Usar média global para missing**
   - ✗ fillna(mean())
   - ✓ interpolate() ou ffill()
5. **Ignorar sazonalidade**
   - ✗ Não considerar padrões periódicos
   - ✓ Decompor e analisar
6. **Escala de tempo errada**
   - ✗ Analisar horária quando padrão é mensal
   - ✓ Resample para frequência adequada

Salvar Análise para Reutilização

Python

```python
# Salvar objetos importantes

import joblib

# 1. Serie processada
daily.to_csv('co_daily.csv')

# 2. Componentes da decomposicao
decomposition.trend.to_csv('trend.csv')
decomposition.seasonal.to_csv('seasonal.csv')
```

Salvar Análise para Reutilização (cont.)

Python

```python
# 3. Features criadas
features = df[['CO_MA24', 'CO_MA168', 'CO_lag24',
               'hour_sin', 'hour_cos']].copy()
features.to_csv('temporal_features.csv')
# 4. Scaler (se usado)
# scaler = StandardScaler()
# scaler.fit(features)
# joblib.dump(scaler, 'scaler.pkl')
print("Analise salva!")
print("Para recarregar:")
print("daily = pd.read_csv('co_daily.csv', index_col=0, parse_dates=True)"
    )
```

Exercício Final: Análise Completa

Exercício Prático

Pipeline completo no Air Quality:

1. Setup:

```
# - Carregar dados
# - Configurar DatetimeIndex
# - Verificar e tratar missing
```

2. Exploração:
   - Visualizar série completa
   - Estatísticas por período
   - Comparar dias úteis vs fim de semana

Exercício Final: Análise Completa (cont.)

Exercício Prático

Pipeline completo no Air Quality:

3. Agregações:
   - Resample para diário
   - Calcular médias móveis (24h, 7d)
   - Criar features temporais
4. Decomposição:
   - Decompor série diária
   - Analisar cada componente
   - Visualizar decomposição
5. Insights:
   - Quando poluição é pior?
   - Há tendência de melhora/piora?
   - Qual a força da sazonalidade?

Recap: Aula 11 Completa

Jornada de hoje:

Bloco 1: DatetimeIndex

- Criar e manipular índice temporal
- Filtrar e acessar por datas
- Extrair componentes temporais
- Criar features básicas

Bloco 2: Agregações

- Resample (mudar frequência)
- Rolling windows (médias móveis)
- Shift, diff, pct_change
- Autocorrelação

Recap: Aula 11 Completa

Jornada de hoje:

Bloco 3: Decomposição

- Separar tendência, sazonalidade, resíduo
- Analisar cada componente
- Aditivo vs multiplicativo

Bloco 4: Visualizações

- Plots temporais eficazes
- Heatmaps, box plots
- Boas práticas

Próximos Passos

Para aprofundar em séries temporais:

Modelagem (futuro):

- ARIMA, SARIMA (modelos clássicos)
- Prophet (Facebook)
- LSTM, GRU (deep learning)
- Exponential Smoothing

Análise avançada:

- Testes de estacionariedade
- Cointegração
- VAR (Vector Autoregression)
- Análise espectral

Próximos Passos

Para aprofundar em séries temporais:

Bibliotecas:

- statsmodels: análise estatística
- Prophet: forecasting fácil
- sktime: ML para séries
- pmdarima: auto-ARIMA

A Seguir: Aula Prática

Exercício Prático

Tempo: 60 minutos
Entrega: via Moodle (notebook)

Tarefas:

1. (atualizado durante a aula)

Notebook: Disponível no Moodle

Obrigado!