# Desafio de Ciencia de Dados da Gerdau - Data & Analytics


## Introdução

O time de Data Science da Gerdau usa Inteligência Artificial e Advanced Analytics para resolver problemas diversos dentro da nossa empresa. Nesse desafio você terá contato com algumas das tarefas diárias de um Cientista de Dados do time e como ele avaliaremos, como suas competências estão conectadas com nossos pilares técnicos e do negócio.

Guiaremos você ao longo do problema, e sua senioridade, de acordo com nossos padrões e pilares, será definida pela maneira como você resolverá cada item.

### O que você deve levar em consideração

- Descrever a abordagem utilizada na resolução do desafio
- Trechos de códigos deverão ter comentários essenciais
- O desafio deverá ser feito com a linguagem de programação Python + bibliotecas que você achar necessário
- Soluções utilizando Excel, R e outras linguagens serão desconsideradas
- Recomendamos utilizar Jupyter Lab ou Notebook.

### Prazo de entrega

- De acordo conforme combinado com o time de Data & Analytics

### Desclassificação Automática

- Entregar fora do prazo
- Compartilhar sua solução com outras pessoas
- Compartilhar sua solução em repositórios públicos. Exemplo: GitHub

## Objetivo

A produção de uma empresa de aço envolve a fabricação de vários tipos de componentes, incluindo bobinas e placas. As informações sobre o consumo de energia são registradas em um sistema de armazenamento em nuvem. O monitoramento do consumo energético desta indústria está disponível nos dados fornecidos. Desta forma, **gostariamos que você preveja o consumo elétrico em uma perspectiva mensal**.

---

# 0. Tratamento dos Dados

Apesar dos dados recebidos ja foram entregues tratados, pessoalmente gosto de dar uma explorada inicial e converter o tipo dos dados para otimização do espaço de memória.

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
df = pd.read_csv('../datasets/raw.csv')
df.head()
```

Out[ ]:

| | Data | Energia_usada | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 01/01/2018 00:15 | 3.17 | 2.95 | 0.0 | 0.0 | 73.21 | 100.0 | 900 | Dia da Semana | Segunda-feira | Carga leve |
| **1** | 01/01/2018 00:30 | 4.00 | 4.46 | 0.0 | 0.0 | 66.77 | 100.0 | 1800 | Dia da Semana | Segunda-feira | Carga leve |
| **2** | 01/01/2018 00:45 | 3.24 | 3.28 | 0.0 | 0.0 | 70.28 | 100.0 | 2700 | Dia da Semana | Segunda-feira | Carga leve |
| **3** | 01/01/2018 01:00 | 3.31 | 3.56 | 0.0 | 0.0 | 68.09 | 100.0 | 3600 | Dia da Semana | Segunda-feira | Carga leve |
| **4** | 01/01/2018 01:15 | 3.82 | 4.50 | 0.0 | 0.0 | 64.72 | 100.0 | 4500 | Dia da Semana | Segunda-feira | Carga leve |

In [ ]:
```
df.columns = ['data', 'consumo_energia', 'corrente_atrasada', 'corrente_p
```

In [ ]:
```
memory_before = df.memory_usage(deep=True).sum()
memory_before
```

Out[ ]:
```
11722932
```

In [ ]:
```
df.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35040 entries, 0 to 35039
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   data                         35040 non-null  object
 1   consumo_energia              35040 non-null  float64
 2   corrente_atrasada            35040 non-null  float64
 3   corrente_principal           35040 non-null  float64
 4   co2                          35040 non-null  float64
 5   potencia_atrasado            35040 non-null  float64
 6   potencia_principal           35040 non-null  float64
 7   segundos_depois_meia_noite   35040 non-null  int64
 8   estado_semana                35040 non-null  object
 9   dia_semana                   35040 non-null  object
 10  tipo_carga                   35040 non-null  object
dtypes: float64(6), int64(1), object(4)
memory usage: 11.2 MB
```

In [ ]:
```
df.nunique()
```

Out[ ]:
```
data                          35040
consumo_energia                3343
corrente_atrasada              1954
corrente_principal              768
co2                               8
potencia_atrasado              5079
potencia_principal             3366
segundos_depois_meia_noite       96
estado_semana                     2
dia_semana                        7
tipo_carga                        3
dtype: int64
```

In [ ]:
```
df.data = df.data.astype('datetime64[ns]')
df.co2 = df.co2.astype('float16')
df.segundos_depois_meia_noite = df.segundos_depois_meia_noite.astype('int
df.estado_semana = df.estado_semana.astype('category')
df.dia_semana = df.dia_semana.astype('category')
df.tipo_carga = df.tipo_carga.astype('category')
```

In [ ]:
```
memory_after = df.memory_usage(deep=True).sum()
memory_after
```

Out[ ]:
```
1998745
```

In [ ]:
```
df.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35040 entries, 0 to 35039
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype         
---  ------                      --------------  -----         
 0   data                        35040 non-null  datetime64[ns]
 1   consumo_energia             35040 non-null  float64       
 2   corrente_atrasada           35040 non-null  float64       
 3   corrente_principal          35040 non-null  float64       
 4   co2                         35040 non-null  float16       
 5   potencia_atrasado           35040 non-null  float64       
 6   potencia_principal          35040 non-null  float64       
 7   segundos_depois_meia_noite  35040 non-null  int32         
 8   estado_semana               35040 non-null  category      
 9   dia_semana                  35040 non-null  category      
 10  tipo_carga                  35040 non-null  category      
dtypes: category(3), datetime64[ns](1), float16(1), float64(5), int32(1)
memory usage: 1.9 MB
```

In [ ]:
```
print (f'{memory_after/memory_before*100: .2f}% da memória original')
```

```
 17.05% da memória original
```

In [ ]:
```
df.to_pickle('../datasets/processed.pkl')
```

# 1. Análise Exploratória e Qualidade dos Dados

Gostariamos que nesta primeira etapa do processo, use sua curiosidade e criatividade, além de experiência, para extrair informação de cada variável ou relação entre as variáveis do dataset. Nesta etapa, podem surgir insights que gerem valor para o negócio, além de ajudar ao entendimento do problema. Esperamos que você gere estatísticas, gráficos, visualizações que descrevam os dados e como eles estão conectados com o problema.

Temos acesso aos dados de consumo elétrico da empresa, referentes a um período de 1 ano, sendo que a medição é realizada em intervalos de 15 minutos.

| Nome | Description. | Tipo Variável |
|---|---|---|
| Data | Data de registro do consumo | Continua |
| Energia_usada | Consumo de energia da empresa [kWh] | Continua |
| V1 | Corrente atrasada [kVarh] | Continua |
| V2 | Corrente principal [kVarh] | Continua |
| V3 | Medições de C02 [ppm] | Continua |
| V4 | Fator de potência atual atrasado | Continua |
| V5 | Fator de potência atual principal | Continua |
| V6 | Número de segundos a partir da meia-noite [S] | Continua |
| V7 | Estado da semana (Final de semana ou dia da semana) | Categorica |
| V8 | Dia da semana (Terça-feira, Sábado, etc) | Categorica |
| V9 | Tipo de carga (Carga leve, Carga média, Carga máxima) | Categorica |

---

# Imports

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose

sns.set(style='darkgrid')
palette = 'mako'
sns.set_theme(style="ticks", rc={"axes.spines.right": False, "axes.spines
```

# Análise inicial

## Visão inicial dos dados

In [ ]:
```
df = pd.read_pickle('../datasets/processed.pkl')
df.head()
```

Out[ ]:

| | data | consumo_energia | corrente_atrasada | corrente_principal | co2 | potencia_atrasad |
|---|---|---|---|---|---|---|
| **0** | 2018-01-01 00:15:00 | 3.17 | 2.95 | 0.0 | 0.0 | 73.2 |
| **1** | 2018-01-01 00:30:00 | 4.00 | 4.46 | 0.0 | 0.0 | 66.7 |
| **2** | 2018-01-01 00:45:00 | 3.24 | 3.28 | 0.0 | 0.0 | 70.2 |
| **3** | 2018-01-01 01:00:00 | 3.31 | 3.56 | 0.0 | 0.0 | 68.0 |
| **4** | 2018-01-01 01:15:00 | 3.82 | 4.50 | 0.0 | 0.0 | 64.7 |

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35040 entries, 0 to 35039
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype         
---  ------                      --------------  -----         
 0   data                        35040 non-null  datetime64[ns]
 1   consumo_energia             35040 non-null  float64       
 2   corrente_atrasada           35040 non-null  float64       
 3   corrente_principal          35040 non-null  float64       
 4   co2                         35040 non-null  float16       
 5   potencia_atrasado           35040 non-null  float64       
 6   potencia_principal          35040 non-null  float64       
 7   segundos_depois_meia_noite  35040 non-null  int32         
 8   estado_semana               35040 non-null  category      
 9   dia_semana                  35040 non-null  category      
 10  tipo_carga                  35040 non-null  category      
dtypes: category(3), datetime64[ns](1), float16(1), float64(5), int32(1)
memory usage: 1.9 MB
```

Observação dos dados numéricos

Foram desconsideradas as variáveis referente a data e segundos depois da meia noite, por serem medidas de tempo

In [ ]:
```
df.drop(columns=['data', 'segundos_depois_meia_noite']).describe().T.drop
```

Out[ ]:

| | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| **consumo_energia** | 27.386892 | 33.444380 | 0.0 | 3.20 | 4.57 | 51.237500 | 157.180000 |
| **corrente_atrasada** | 13.035384 | 16.306000 | 0.0 | 2.30 | 5.00 | 22.640000 | 96.910000 |
| **corrente_principal** | 3.870949 | 7.424463 | 0.0 | 0.00 | 0.00 | 2.090000 | 27.760000 |
| **co2** | 0.011520 | 0.016144 | 0.0 | 0.00 | 0.00 | 0.020004 | 0.070007 |
| **potencia_atrasado** | 80.578056 | 18.921322 | 0.0 | 63.32 | 87.96 | 99.022500 | 100.000000 |
| **potencia_principal** | 84.367870 | 30.456535 | 0.0 | 99.70 | 100.00 | 100.000000 | 100.000000 |

Observação dos dados categóricos

In [ ]:
```
df.describe(exclude=[np.number, np.datetime64]).T.drop('count', axis=1)
```

Out[ ]:

| | unique | top | freq |
|---|---|---|---|
| **estado_semana** | 2 | Dia da Semana | 25056 |
| **dia_semana** | 7 | Segunda-feira | 5088 |
| **tipo_carga** | 3 | Carga leve | 18072 |

In [ ]:
```
df.nunique()
```

Out[ ]:
```
data                          35040
consumo_energia                3343
corrente_atrasada              1954
corrente_principal              768
co2                               8
potencia_atrasado              5079
potencia_principal             3366
segundos_depois_meia_noite       96
estado_semana                     2
dia_semana                        7
tipo_carga                        3
dtype: int64
```

A partir do site do governo foi adquirido as datas referentes as estações do ano

- Verão: 21 dezembro, às (14h28) de 2017 a 20 de março de 2018 (13h14);
- Verão: de 21 de dezembro (20h22) a 20 de março de 2019 (18h58);
- Outono: de 20 de março (13h14) a 21 de junho de 2018 (7h07);
- Inverno: de 21 de junho (7h07) a 22 de setembro de 2018 (22h53);
- Primavera: de 22 de setembro (22h53) a 21 de dezembro de 2018 (20h22).

In [ ]:
```
def estacao_do_ano(data):
    verao_2018_inicio = pd.Timestamp('2017-12-21 14:28:00')
    verao_2018_fim = pd.Timestamp('2018-03-20 13:14:00')
    verao_2019_inicio = pd.Timestamp('2018-12-21 20:22:00')
    verao_2019_fim = pd.Timestamp('2019-03-20 18:58:00')
    outono_inicio = pd.Timestamp('2018-03-20 13:14:00')
    outono_fim = pd.Timestamp('2018-06-21 07:07:00')
    inverno_inicio = pd.Timestamp('2018-06-21 07:07:00')
    inverno_fim = pd.Timestamp('2018-09-22 22:53:00')
    primavera_inicio = pd.Timestamp('2018-09-22 22:53:00')
    primavera_fim = pd.Timestamp('2018-12-21 20:22:00')

    if verao_2018_inicio <= data < verao_2018_fim or verao_2019_inicio <=
        return 'Verão'
    elif outono_inicio <= data < outono_fim:
        return 'Outono'
    elif inverno_inicio <= data < inverno_fim:
        return 'Inverno'
    elif primavera_inicio <= data < primavera_fim:
        return 'Primavera'
    else:
        return 'Fora do intervalo'

df['estacao'] = df['data'].apply(estacao_do_ano)
del estacao_do_ano
```

## Observação da variável target em ordem crescente e decrescente

Apesar dos dados não possuirem valores null, foi observado uma linha completamente nula, podendo ser considerado um outlier, ou um erro na medição

In [ ]:
```
df.sort_values('consumo_energia', ascending=True).head(10)
```

Out[ ]:

| | data | consumo_energia | corrente_atrasada | corrente_principal | co2 | potencia_atr |
|---|---|---|---|---|---|---|
| **29855** | 2018-07-11 00:00:00 | 0.00 | 0.00 | 0.0 | 0.0 | |
| **25848** | 2018-09-27 06:15:00 | 2.45 | 4.93 | 0.0 | 0.0 | |
| **25847** | 2018-09-27 06:00:00 | 2.45 | 4.97 | 0.0 | 0.0 | |
| **26234** | 2018-01-10 06:45:00 | 2.45 | 4.36 | 0.0 | 0.0 | |
| **25851** | 2018-09-27 07:00:00 | 2.45 | 5.08 | 0.0 | 0.0 | |
| **26018** | 2018-09-29 00:45:00 | 2.45 | 4.00 | 0.0 | 0.0 | |
| **25820** | 2018-09-26 23:15:00 | 2.45 | 4.64 | 0.0 | 0.0 | |
| **25822** | 2018-09-26 23:45:00 | 2.45 | 4.61 | 0.0 | 0.0 | |
| **25821** | 2018-09-26 23:30:00 | 2.45 | 4.61 | 0.0 | 0.0 | |
| **23733** | 2018-05-09 05:30:00 | 2.48 | 5.00 | 0.0 | 0.0 | |

In [ ]:
```
df.sort_values('consumo_energia', ascending=False).head(10)
```

Out[ ]:

| | data | consumo_energia | corrente_atrasada | corrente_principal | co2 | potenci |
|---|---|---|---|---|---|---|
| **31238** | 2018-11-22 09:45:00 | 157.18 | 77.72 | 0.0 | 0.070007 | |
| **1398** | 2018-01-15 13:45:00 | 153.14 | 70.45 | 0.0 | 0.070007 | |
| **31723** | 2018-11-27 11:00:00 | 151.67 | 69.73 | 0.0 | 0.070007 | |
| **7812** | 2018-03-23 09:15:00 | 151.31 | 65.20 | 0.0 | 0.070007 | |
| **1701** | 2018-01-18 17:30:00 | 149.65 | 64.87 | 0.0 | 0.070007 | |
| **33848** | 2018-12-19 14:15:00 | 149.18 | 74.56 | 0.0 | 0.070007 | |
| **162** | 2018-02-01 16:45:00 | 147.46 | 65.27 | 0.0 | 0.000000 | |
| **6111** | 2018-05-03 16:00:00 | 146.88 | 70.49 | 0.0 | 0.070007 | |
| **1679** | 2018-01-18 12:00:00 | 146.48 | 84.89 | 0.0 | 0.070007 | |
| **447** | 2018-05-01 16:00:00 | 146.34 | 66.92 | 0.0 | 0.070007 | |

## Análise referente aos meses

In [ ]:
```
def get_mode(x: pd.Series) -> str:
    return x.mode()[0]

df_groupby_month = df.groupby(df.data.dt.month_name(), sort=False).agg(
    {
        'consumo_energia': ['mean', 'median'],
        'corrente_atrasada': ['mean', 'median'],
        'corrente_principal': ['mean', 'median'],
        'co2': ['mean', 'median'],
        'potencia_atrasado': ['mean', 'median'],
        'potencia_principal': ['mean', 'median'],
        'dia_semana': get_mode,
        'tipo_carga': get_mode,
    }
)

del get_mode

df_groupby_month
```

Out[ ]:

| | consumo_energia | | corrente_atrasada | | corrente_principal | | co2 | | p |
|---|---|---|---|---|---|---|---|---|---|
| | mean | median | mean | median | mean | median | mean | median | |
| data | | | | | | | | | |
| January | 33.876300 | 13.68 | 15.136035 | 5.18 | 4.520719 | 0.00 | 0.014759 | 0.010002 | 85 |
| February | 29.330588 | 5.15 | 12.377035 | 4.86 | 4.434911 | 0.00 | 0.011741 | 0.000000 | 83 |
| March | 27.107282 | 4.46 | 11.852571 | 4.57 | 3.993380 | 0.00 | 0.011476 | 0.000000 | 81 |
| April | 25.923153 | 4.12 | 12.074792 | 4.28 | 4.427858 | 0.00 | 0.010879 | 0.000000 | 81 |
| May | 28.636166 | 4.21 | 13.803901 | 5.15 | 3.579778 | 0.00 | 0.012218 | 0.000000 | 78 |
| June | 25.909760 | 4.21 | 12.484104 | 5.18 | 3.726187 | 0.00 | 0.010903 | 0.000000 | 78 |
| July | 27.497762 | 4.05 | 12.963784 | 5.08 | 3.508673 | 0.00 | 0.011698 | 0.000000 | 80 |
| August | 28.021788 | 4.43 | 14.766176 | 5.47 | 2.888901 | 0.00 | 0.011912 | 0.000000 | 76 |
| September | 20.581271 | 3.42 | 11.048615 | 5.11 | 3.855295 | 0.00 | 0.008431 | 0.000000 | 76 |
| October | 27.564022 | 4.46 | 14.895481 | 5.98 | 3.486727 | 0.00 | 0.011573 | 0.000000 | 76 |
| November | 30.867705 | 5.11 | 14.681601 | 5.15 | 3.175757 | 0.00 | 0.013087 | 0.000000 | 81 |
| December | 23.312893 | 4.25 | 10.217043 | 3.31 | 4.898138 | 0.04 | 0.009550 | 0.000000 | 85 |

Apenas com esse agrupamento foi observada uma forte correlação referente ao consumo de energia e a corrente atrasada, então ordenamos os meses referentes as duas variaveis e comparamos

In [ ]:
```
consumo_energia = df_groupby_month.consumo_energia.sort_values('mean', as
corrente_atrasada = df_groupby_month.corrente_atrasada.sort_values('mean'
```

In [ ]:
```
# Mean
pd.DataFrame(data=[consumo_energia[0], corrente_atrasada[0]], index=['con
```

Out[ ]:

| | consumo_energia | corrente_atrasada |
|---|---|---|
| 0 | January | January |
| 1 | November | October |
| 2 | February | August |
| 3 | May | November |
| 4 | August | May |
| 5 | October | July |
| 6 | July | June |
| 7 | March | February |
| 8 | April | April |
| 9 | June | March |
| 10 | December | September |
| 11 | September | December |

In [ ]:
```
# Median
pd.DataFrame(data=[consumo_energia[1], corrente_atrasada[1]], index=['con
```

Out[ ]:

| | consumo_energia | corrente_atrasada |
|---|---|---|
| 0 | January | October |
| 1 | February | August |
| 2 | November | January |
| 3 | March | June |
| 4 | October | May |
| 5 | August | November |
| 6 | December | September |
| 7 | May | July |
| 8 | June | February |
| 9 | April | March |
| 10 | July | April |
| 11 | September | December |

# Análise gráfica

## Histograma

Inicialmente vamos observar o histograma desses dados, assim observando como sua distribuição se comporta.

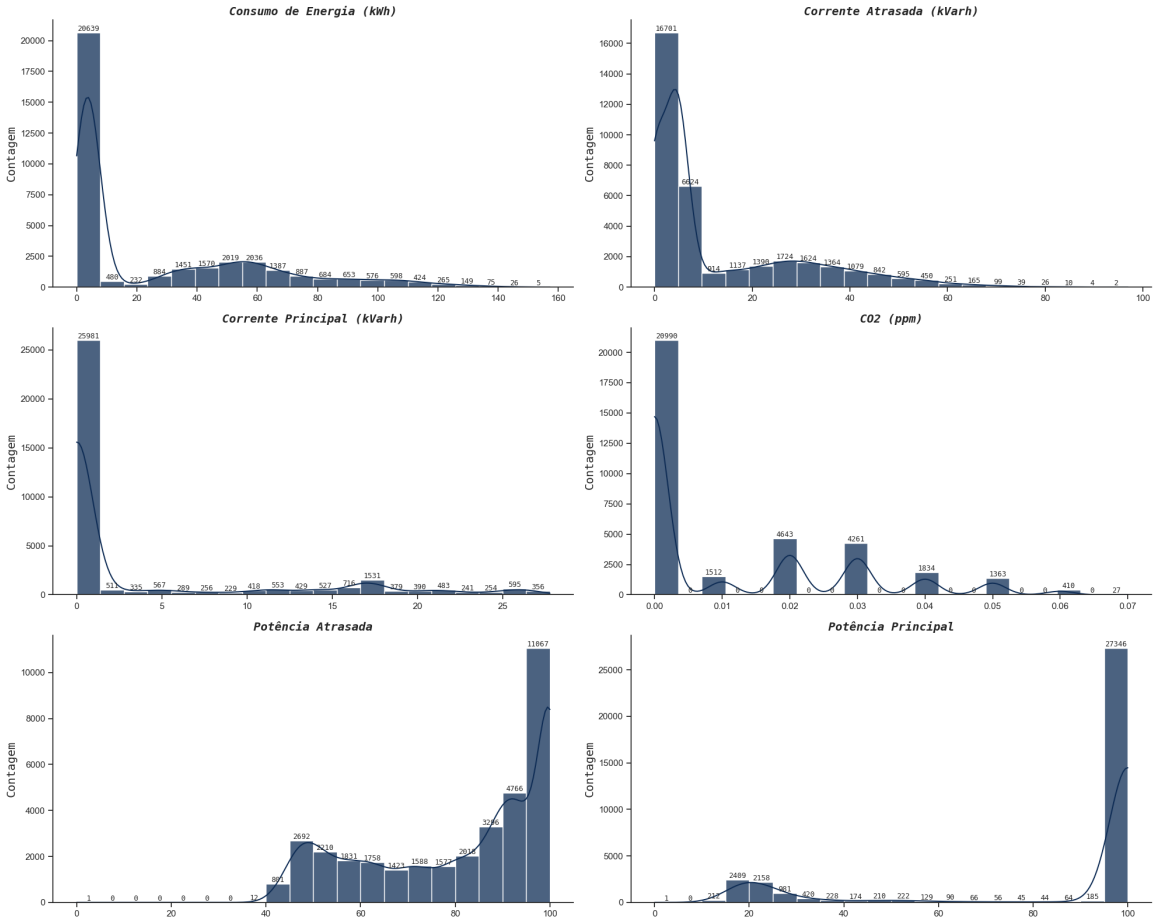
Para tal foram desconsiderados elementos:

- categóricos
  - Estado da semana
  - Dia da semana
  - Tipo de carga
- temporais
  - Data
  - Estação
  - Segundos após a meia noite

Foi observado que apesar do seu valor flutuante a variável co2 se comporta de maneira categórica, por possuir apenas 8 elementos únicos.

In [ ]:
```
def plot_histograms(df: pd.DataFrame, columns: list, n_rows: int, n_cols:
    fig = plt.figure(figsize=(20, 16))
    for i, column in enumerate(columns):
        ax = fig.add_subplot(n_rows, n_cols, i + 1)
        sns.histplot(data=df, x=column, ax=ax, bins=20, kde=True, color='
        for value in ax.containers:
            ax.bar_label(value, label_type='edge', fontsize=9, family='mo
        ax.set_xlabel(None)
        ax.set_ylabel('Contagem', fontsize='large', family='monospace')
        ax.set_title(title[i], fontsize='large', fontweight='bold', style
    fig.tight_layout()

columns = df.drop(['data', 'estado_semana', 'dia_semana', 'tipo_carga', '
title=['Consumo de Energia (kWh)', 'Corrente Atrasada (kVarh)', 'Corrente
plot_histograms(df=df, columns=columns, n_rows=3, n_cols=2, title=title)
plt.show()
```




## Gráficos de Barra

Iremos agora analisar o comportamento das variáveis referentes a média de consumo energético

In [ ]:
```
def create_bar(df_: pd, columns: list, n_rows: int, n_cols: int, rot=45):
    y_ = 'consumo_energia'
    fig = plt.figure(figsize=(20, 30))
    for i, column in enumerate(columns):
        for j in range (0, 2):
            ax = fig.add_subplot(n_rows, n_cols, 2*i + j + 1)
            if j:
                df = df_.groupby(by=column, as_index=False, sort=False)[y
            else:
                df = df_.groupby(by=column, as_index=False, sort=False)[y
            try:
                df[column] = df[column].str.capitalize()
            except:
                pass

            sns.barplot(data=df, x=column, y=y_, palette=palette, ax=ax)
            for value in ax.containers:
                ax.bar_label(value)
            ax.tick_params(axis='x', labelrotation=rot, size=12)
            ax.tick_params(axis='y', size=12)
            ax.set_xlabel(column.capitalize(), fontsize='large', family='
            ax.set_ylabel("Consumo de Energia (kWh)", fontsize='large', f
            if j:
                ax.set_title(f"Mediana de Consumo de Energia por {column.
                             style='italic', family='monospace')
            else:
                ax.set_title(f"Média de Consumo de Energia por {column.ca
                            style='italic', family='monospace')
    fig.tight_layout()
```

Para tal observação foi considerada as variáveis categóricas

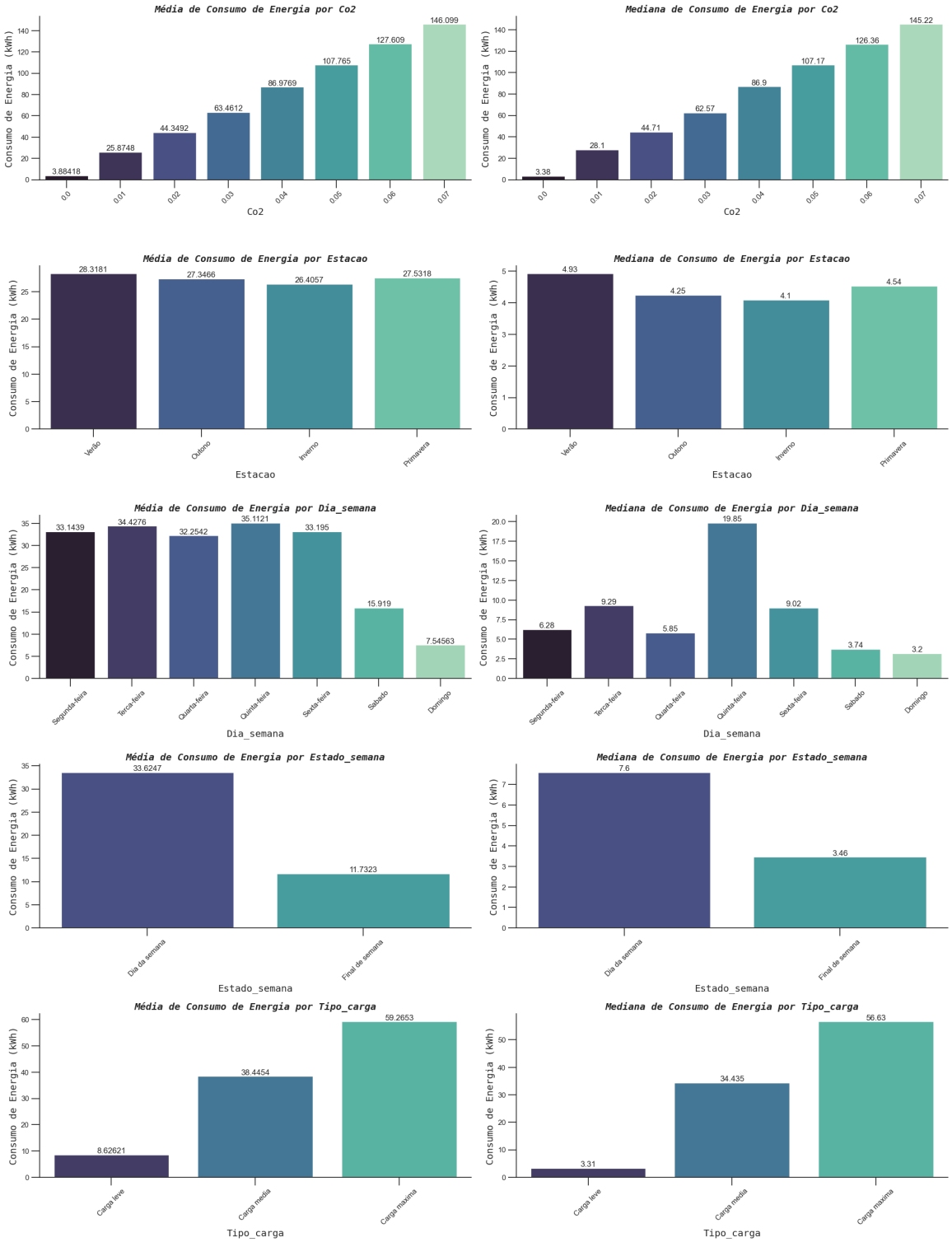
- Co2 (Por possuir comportamento categórico)
- Estação do ano
- Dia da semana
- Estado da semana
- Tipo de carga

Algumas observações com os gráficos de barra

Em ordem de plotagem

1. Co2
    - O consumo de energia cresce conforme a emissão de Co2
2. Estações do ano
    - O agrupamento a partir das estações do ano deu-se por conta da hipótese de que em épocas mais quentes (verão) seria necessário mais energia que em épocas mais frias (inverno);
    - Apesar dessa hipótese se mostrar verdadeira, a diferença representada pela média é baixa.
3. Dia da semana e Estado da semana
    - Uma grande fábrica produtora de aço necessita de funcionamento diário, porém a carga dos funcionários aos fins de semana seria diminuida.
4. Tipo de carga
    - Temos um ponto importante a salientar, apesar da baixa demanda energética para objetos com carga leve, a quantidade de elementos contabilizados como carga leve são exponencialmente maiores.
    - Um dos fatores pode ser que o tempo para processar uma carga seja proporcional ao seu tamanho

In [ ]:
```
columns = ['co2', 'estacao', 'dia_semana', 'estado_semana', 'tipo_carga']
create_bar(df, columns, 6, 2)
plt.show()
```




## Gráficos de caixa

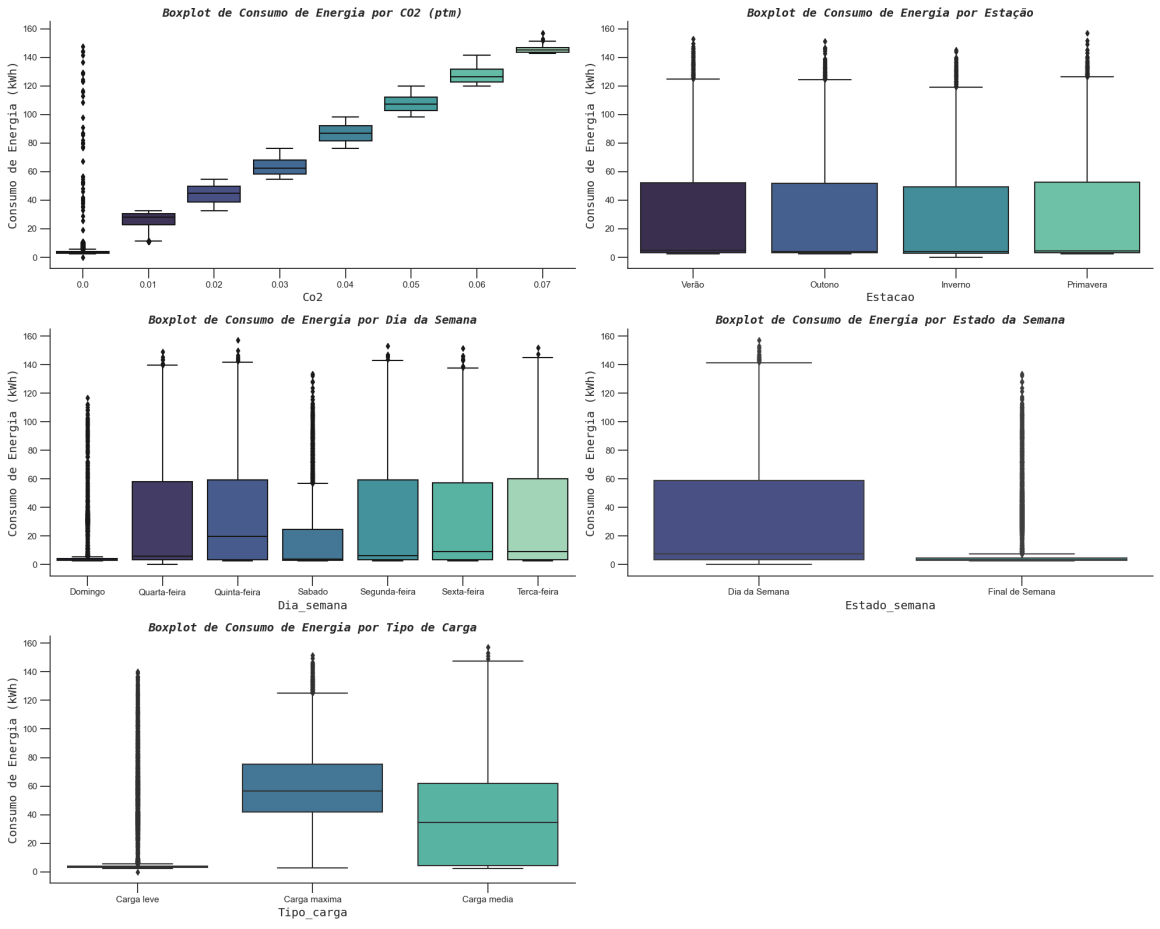
In [ ]:
```
def create_box_plot(df_: pd, columns: list, title: list, n_rows: int, n_c
    y_ = 'consumo_energia'
    fig = plt.figure(figsize=(20, 16))
    for i, column in enumerate(columns):
        ax = fig.add_subplot(n_rows, n_cols, i + 1)
        sns.boxplot(data=df_, x=column, y=y_, ax=ax, palette=palette)
        ax.tick_params(axis='x', size=12)
        ax.tick_params(axis='y', size=12)
        ax.set_xlabel(column.capitalize(), fontsize='large', family='mono
        ax.set_ylabel("Consumo de Energia (kWh)", fontsize='large', famil
        ax.set_title(f"Boxplot de Consumo de Energia por {title[i]}", fon
                style='italic', family='monospace')
    fig.tight_layout()
```

Novamente iremos observar as variáveis categóricas

Algumas observações com os gráficos de caixa

Conforme aumentamos as observações temos maiores outliers, também podemos notar as diferenças entre os ranges de consumo energético.

In [ ]:
```
columns = ['co2', 'estacao', 'dia_semana', 'estado_semana', 'tipo_carga']
title = ['CO2 (ptm)', 'Estação', 'Dia da Semana', 'Estado da Semana', 'Ti
create_box_plot(df, columns, title, 3, 2)
plt.show()
```




Agora iremos ver na prática a quantidade de outliers a partir do intervalo interquantil.

Nos casos analisados existe apenas outlier em um dos lados (superior ou inferior).

In [ ]:
```
def calculate_outliers(x: pd.Series) -> list:
    q1 = x.quantile(.25)
    q3 = x.quantile(.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return x[(x < lower_bound)].index.tolist(), x[(x > upper_bound)].inde

for column in df.drop(['data', 'estado_semana', 'dia_semana', 'segundos_d
    lower, upper = calculate_outliers(df[column])
    column = column.replace('_', ' ')
    column = column.capitalize()
    print(f"{column}: Lower = {len(lower)}, Upper = {len(upper)}, Total =

del calculate_outliers
```

```
Consumo energia: Lower = 0, Upper = 328, Total = 328
Corrente atrasada: Lower = 0, Upper = 1059, Total = 1059
Corrente principal: Lower = 0, Upper = 7759, Total = 7759
Co2: Lower = 0, Upper = 437, Total = 437
Potencia atrasado: Lower = 1, Upper = 0, Total = 1
Potencia principal: Lower = 8327, Upper = 0, Total = 8327
```

In [ ]:
```
def calculate_amplitude(x: pd.Series) -> float:
    return x.max() - x.min()

for column in df.drop(['data', 'estado_semana', 'dia_semana', 'segundos_d
    amplitude = calculate_amplitude(df[column])
    column = column.replace('_', ' ')
    column = column.capitalize()
    print(f"{column}: Amplitude = {amplitude:.2f}")

del calculate_amplitude
```

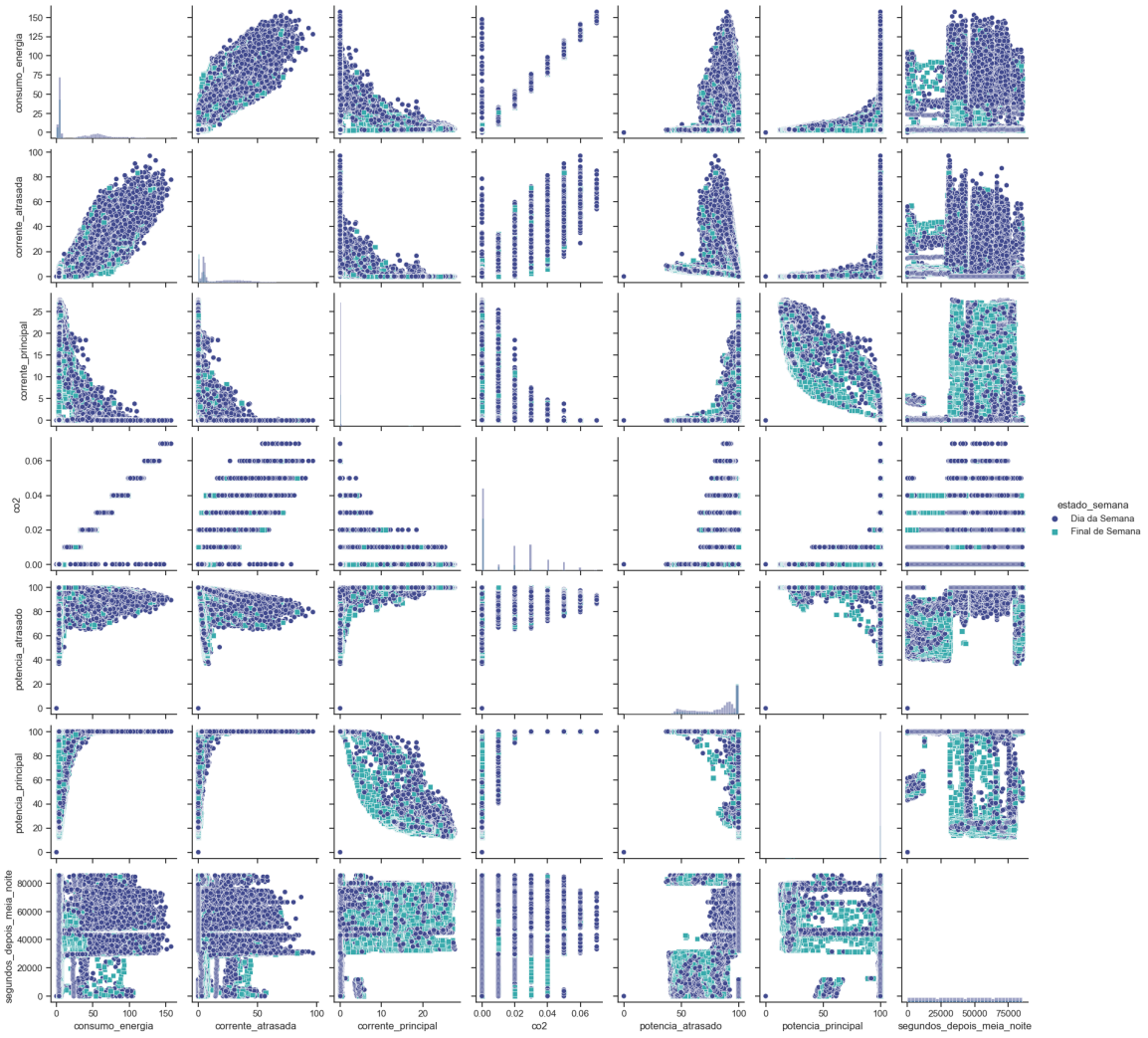
```
Consumo energia: Amplitude = 157.18
Corrente atrasada: Amplitude = 96.91
Corrente principal: Amplitude = 27.76
Co2: Amplitude = 0.07
Potencia atrasado: Amplitude = 100.00
Potencia principal: Amplitude = 100.00
```

## Gráfico de dispersão

Algumas observações

1. Confirmação da alta correlação (linear) positiva entre o consumo de energia e a corrente atrasada
2. Observada uma correlação (linear) negativa entre corrente principal e potencia principal
3. Correlação (não linear) positiva entre potencia principal e corrente atrasada

In [ ]:
```
sns.pairplot(df, hue="estado_semana", palette=palette, markers=["o", "s"]
plt.show()
```




## Matriz de correlação

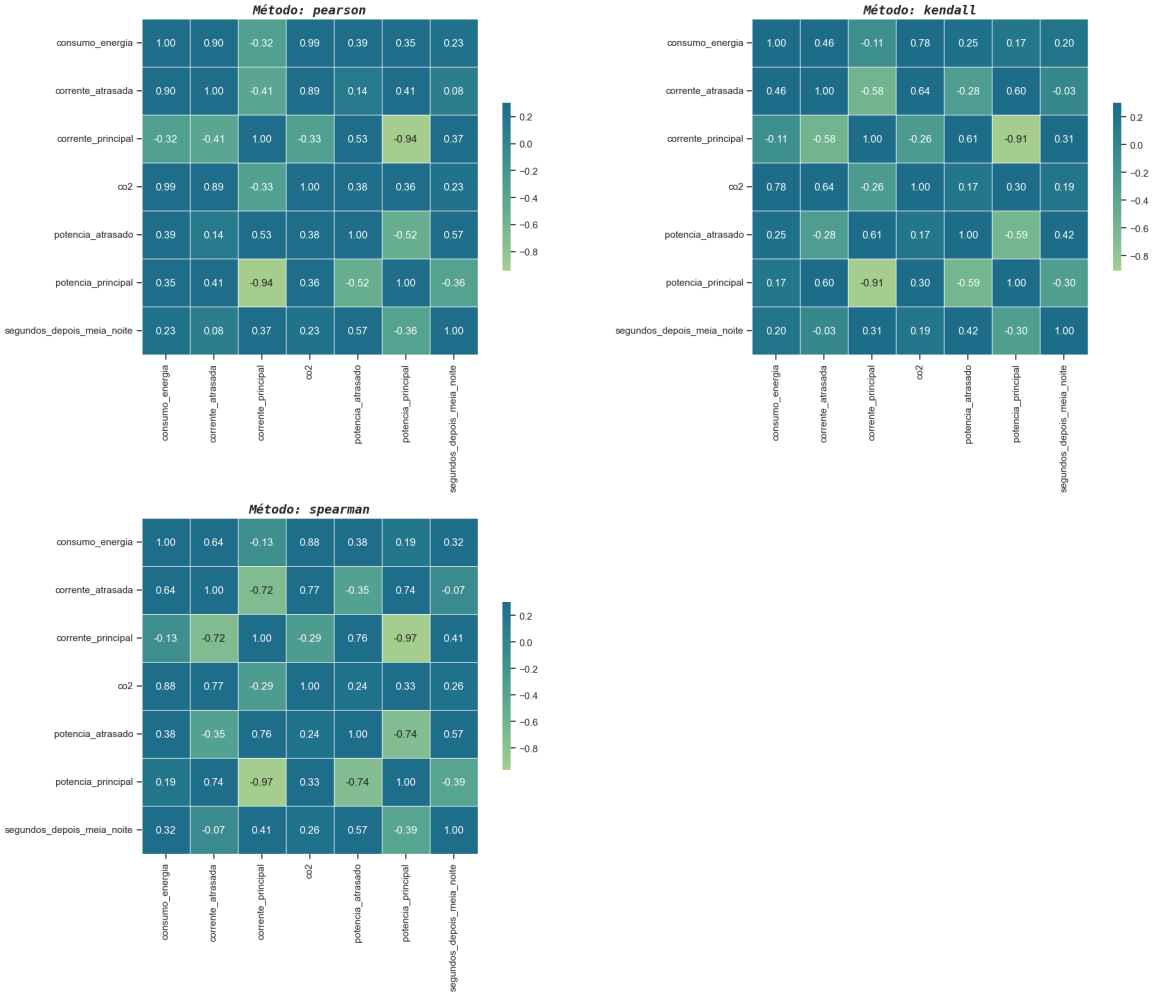
Agora para confirmar as observações das correlações lineares foi plotado a matriz de correlação utilizando os três métodos:

- Pearson (linear)
- Kendall (não linear)
- Spearman (não linear)

In [ ]:
```
fig = plt.figure(figsize=(20, 16))
for i, method in enumerate(['pearson', 'kendall', 'spearman']):
    ax = fig.add_subplot(2, 2, i + 1)
    corr = df.corr(numeric_only=True, method=method)
    sns.heatmap(corr, cmap="crest", vmax=.3, center=0,
        square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,
    ax.set_title(f'Método: {method}', fontsize='large', fontweight='bold'

fig.tight_layout()

plt.show()
```



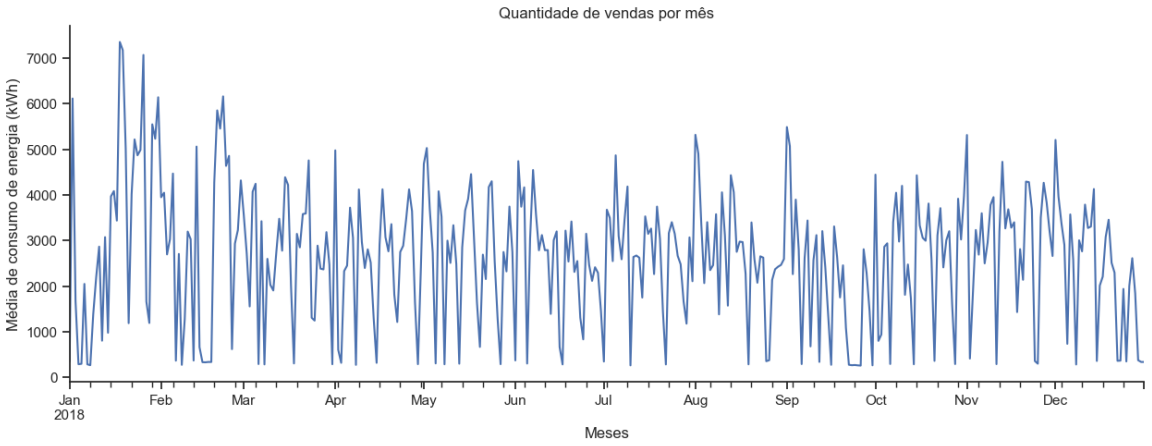

## Análise temporal

Apenas por curiosidade foi feita uma análise básica temporal

In [ ]:
```
df_groupby_day = df.set_index('data').resample('D').agg(
    {
        'consumo_energia': 'sum',
        'corrente_atrasada': 'sum',
        'corrente_principal': 'sum',
        'co2': 'mean',
        'potencia_atrasado': 'mean',
        'potencia_principal': 'mean',
        'dia_semana': lambda x: x.mode()[0],
        'tipo_carga': lambda x: x.mode()[0],
        'estacao': lambda x: x.mode()[0],
    }
).copy()
```

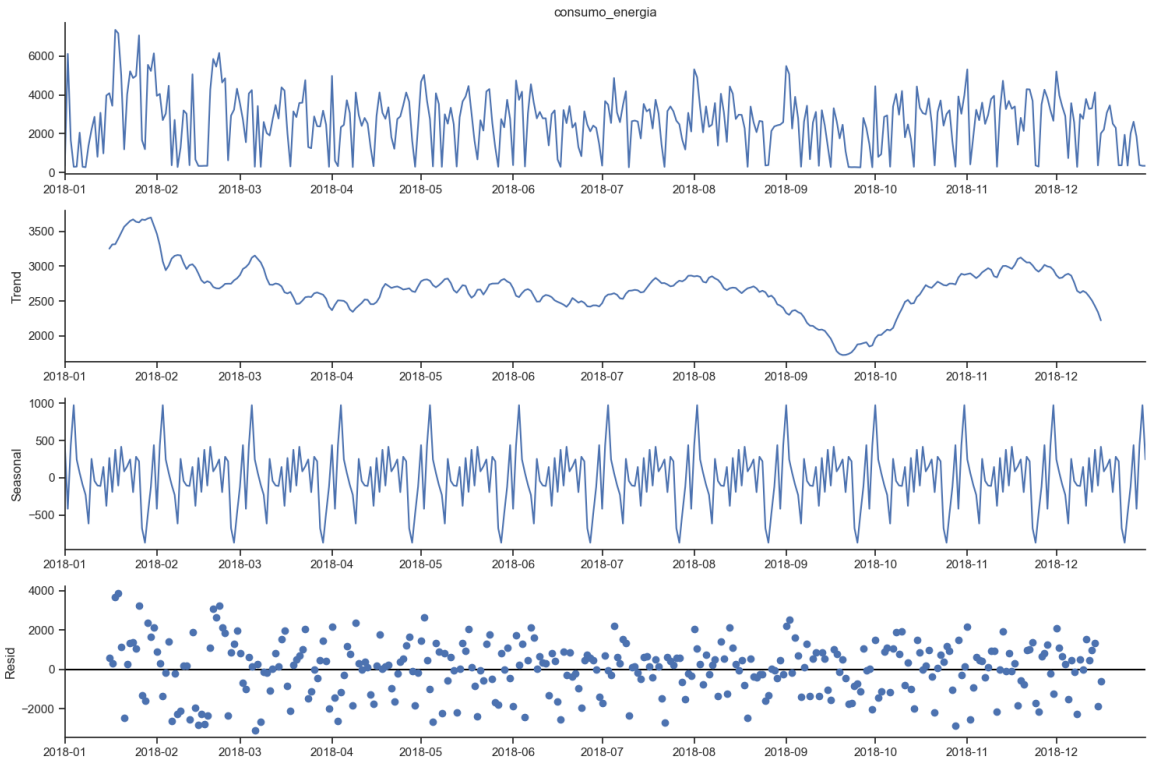
In [ ]:
```
plt.figure(figsize=(15, 5))

df_groupby_day['consumo_energia'].plot()
plt.xlabel('Meses')
plt.ylabel('Média de consumo de energia (kWh)')
plt.title('Quantidade de vendas por mês')
plt.show()
```




In [ ]:
```
decomposed = seasonal_decompose(df_groupby_day['consumo_energia'], model=

fig = decomposed.plot()
fig.set_size_inches((15, 10))
fig.tight_layout()
plt.show()
```




## Conclusão do EDA

Foram realizadas diversas análises para compreender melhor a distribuição dos dados, identificar possíveis anomalias e outliers. Além disso, foram exploradas as relações entre as variáveis e como elas se correlacionam com a variável alvo.

Durante o processo de análise, foi possível identificar algumas questões importantes que precisam ser levadas em consideração durante a criação do modelo. Se duas variáveis estão altamente correlacionadas, isso pode indicar que elas estão medindo essencialmente a mesma coisa.

Correlações fortes (>=0.9)

| Variável 1 | Variável 2 | Tipo de Correlação |
|---|---|---|
| Consumo de Energia | Corrente atrasada | Linear Positiva |
| Consumo de Energia | Co2 | Linear Positiva |
| Co2 | Corrente atrasada | Linear Positiva |
| Potência principal | Corrente atrasada | Não Linear Positiva |
| Corrente principal | Potência principal | Não Linear Negativa |

Portanto as variáveis que não possuem uma alta correlação com a variável target podem ser eliminadas para criação do modelo, deve-se entretanto observar a performance por conta do problema de multicolinearidade.

- Potência principal

Outro ponto importante é o outlier presente no index = 29855, deve-se entender o motivo de sua aparição, afinal caso o medidor esteja enviesado será necessário avaliar melhor os dados.

É importante levar em consideração todos os insights e padrões identificados, bem como as questões que precisam ser tratadas, a fim de desenvolver um modelo preciso e eficiente.

# 2. Modelo de aprendizado de máquina

Finalmente a etapa de modelagem. Mostre o processo de desenvolvimento, as técnicas utilizadas, os algoritmos, sempre justificando suas escolhas. O modelo final é aquele que você teria confiança para usar na vida real.

In [ ]:

---

## 2.2 Avaliação e escolha do modelo

Descreva a(s) métrica(s) que você usou para avaliar o modelo. Lembre-se do objetivo do desafio, **Prever o consumo elétrico em uma perspectiva mensal**

In [ ]:

---

# 3. Uso do modelo

Nesse ponto não há margem para mudança, é de fato o modelo final. Se bem apresentado e trazendo bons resultados, certamente o cliente da área de negócio perguntará: **"Ok, como a gente pode usar esse modelo?"** (Não precisa implementar)

1. Como será utilizado o modelo: entra base, sai status?
2. Qual é a entrega de valor deste modelo
3. Quais tecnologias você usaria
4. Como será a implementação do modelo? API, Docker, etc

In [ ]: