Dados e Aprendizagem Automática
Trabalho Prático

Universidade do Minho
2023/2024

Grupo 15
Inês Daniela Alves Ferreira pg53870
Inês Nogueira Ferreira pg53879
Marta da Silva e Sá pg54084
José Rafael Ferreira pg53985

11 de janeiro de 2024

Conteúdo

1 Introdução

Na Unidade Curricular de Dados e Aprendizagem Automática foi proposto para componente prática duas tarefas, onde os grupos de trabalho são desafiados a trabalhar um conjunto de datasets com vista à sua exploração, tratamento e implementação de modelos de Machine Learning usando os datasets depois de tratados. Como tarefa primeira a equipa docente deu abertura aos discentes para escolherem um conjunto de dados ao seu critério, quanto à seguinte tarefa a equipa docente desafiou os diferentes grupos a participarem na plataforma Kaggle num concurso que procura obter a melhor classificação possível de entre todos os grupos com os melhores resultados quer de grupo quer de performance do modelo usado.

Para a primeira tarefa o dataset escolhido pelo grupo foi "Suicide Rates Overview 1985 to 2016" e o objetivo é observar como os atributos presentes neste dataset e as relações entre eles têm impacto no atributo suicides_no (número de suicídios).

A segunda tarefa é referente ao tema proposto pela Equipa Docente, sendo o mesmo desenvolvido em prol de uma competição na plataforma Kaggle. Deste modo para treino e tuning dos modelos de Machine Learning foram disponibilizados dois datasets com dados energéticos nomeadamente energia_202109-202112.csv e energia_202201-202212.csv e dois datasets com dados meteorológicos nomeadamente meteo_202109-202112.csv e meteo_202201-202212.csv. Tantos os datasets energéticos como os meteorológicos estão divididos por data, i.e., um dos datasets cobre o período de 2021 enquanto que o outro cobre todo o ano de 2022. Para além disso foram disponibilizados dois datasets de teste um com dados energéticos energia_202301-202304.csv e outro com com dados meteorológicos meteo_202301-202304.csv.

Para ambas as tarefas enunciadas, adotou-se a abordagem CRISP-DM (Cross-Industry Standard Process for Data Mining) para executar o que foi proposto. Essa metodologia proporciona uma estrutura organizada e iterativa, dividida em seis fases distintas: Bussiness Understanding (neste caso, compreender a relevância dos dados e o objetivo destas tarefas), Data Understanding (compreender de que forma os dados se relacionam entre si e com o problema objetivo), Data Preparation (fase de tratamento dos dados), Modeling (criação dos modelos com as condições que se consideram mais apropriadas ao tipo de dados pós-tratamento e ao objetivo) e, por fim, no nosso caso, Evaluation (onde efetivamente tentamos perceber o efeito que o tratamento e a modelação tiveram e se cumpre o objetivo que se procura atingir). Este processo é incremental e requer que se repense muito do trabalho que foi feito nas etapas anteriores até que no final se sinta que o resultado obtido é bastante satisfatório ou excelente.

2 Tarefa 1 - *Dataset Suicide Rates Over- view 1985 to 2016*

2.1 Especificação do dataset

O modelo contém as seguintes features com os respetivos significados:

- **country**: variável do tipo string que identifica o país analisado.
- **year**: variável sexo das pessoas que cometeram suicídio.
- **age**: variavel do tipo int que representa o ano correspondente ao levantamento dos dados.
- **sex**: variável do tipo string que define o género dos individuos que cometeram suicídio.
- **suicides_no**: variável do tipo int que corresponde ao número de suicídios ocorridos.
- **population**: variável do tipo int que define a população de um determinado país num determi-

nado ano.
- **suicides/100k pop**: variável do tipo double que corresponde ao número de suicídios por cada 100 mil habitantes.
- **country-year**: variável do tipo string que representa o país e o ano correspondente ao levantamento de dados.
- **HDI for year**: variável do tipo double que corresponde ao índice de desenvolvimento humano.
- **gdp_for_year**: variável do tipo string que representa o grau de desenvolvimento de um país num determinado ano.
- **gdp_per_capita**: variável do tipo inteiro que representa o grau de desenvolvimento de um país per capita.
- **generation**: variável do tipo string que representa a geração do conjunto de pessoas que cometeram suicídio.

2.2 Modelos Utilizados

Como o objetivo com o dataset em questão era prever para determinadas condições o número de suicídios que podiam ocorrer trata-se de uma feature target "quase" contínua, isto porque obviamente, não existem, por exemplo, - 4,5 suicídios - no entanto, este valor pode ser desde 0 suicídios e estender-se até ao valor máximo de humanos que se insiram nas condições sujeitas a teste, por exemplo, - se em Portugal existirem 10 milhões de pessoas do sexo masculino com idades entre os 20 e os 24 anos, se estivermos a prever nestas condições poderíamos prever um teto máximo de 10 milhões - obviamente, sabemos que um valor destes ou ainda superior previsto por qualquer modelo seria exacerbado e notaria uma má performance do mesmo. Assim, por se tratar de uma variável contínuo e onde temos informação sobre o resultado expectável usamos modelos de regressão. Estes foram então os modelos e as condições que exploramos:

- Linear Regression
  - c/ cross-validation
  - s/ cross-validation
  - c/ *feature engineering*
  - s/ *feature engineering*
- Decision Tree Regressor
  - c/ cross-validation
- XGboost

2.3 Tratamento de Dados

Neste ponto apresentamos o tratamento que aplicamos aos dados do conjunto de dados. De salienta, que cada modelo pode ter o dataset com um diferente tratamento nos dados.

2.3.1 Renomeamos as colunas 'gdp_per_capita ($)' e ' gdp_for_year ($)'

Para tornar mais simples a interpretação das features decidimos por conveniência em todos os modelos que trabalhamos substituir 'gdp_per_capita ($)' por 'gdp_per_capita' e substituir 'gdp_for_year'.

2.3.2 Convertemos a coluna 'gdp_for_year' para valores numéricos

Nesta coluna os registos eram strings representando valores numéricos, impossibilitando assim o grupo de introduzir estes no treino dos modelos. Assim, decidiu-se remover as vírgulas que estavam presentes e logo a seguir converter para inteiro. Isto foi aplicado em todos os modelos.

2.3.3 Notamos a presença de outliers na feature 'gdp_for_year'

Ao explorar os dados detetamos a presença de outliers em 'gdp_for_year'. Inicialmente, pensamos ser útil, principalmente, na regressão linear, removê-los. Remover seria uma das opções pois tendo em conta a dimensão do dataset e a quantidade destes acreditamos que não influenciaria muito os resultados obtidos. Contudo, experimentamos em ambos os casos treinar e avaliar a performance do modelo de regressão linear e, concluímos, que não era vantajoso fazê-lo. Quanto aos restantes modelos não consideramos removê-los, como as Decision Trees, pois são mais robustos a gerir os outliers. **Nota:** Figura 1 associada.

2.3.4 Análise gráfica das colunas 'year' e 'country' registos outliers

Antes de decidirmos exatamente o que fazer com a coluna dos outliers, criamos um subdataset que contém apenas registos com outliers da coluna 'gdp_for_year'. Isto tornou-se interessante porque demos conta que ao longo dos anos registados a quantidade de registos aumentava, o que faz sentido se tivermos em conta que, historicamente, a economia mundial tende a crescer e o gdp (gross domestic product) representa a riqueza económica criada a cada ano por um país. Aproveitando a referência aos países, notamos e, uma vez mais com sentido, que as maiores potências económicas são aquelas que apareciam neste conjunto de dados quando fizemos a exposição do gráfico que relacionava cada país neste subdataset com a quantidade de vezes que aparecia num registo do mesmo. **Nota:** Figuras 2 e 3 associadas.

2.3.5 Existiam apenas missing values na feature 'HDI for year' mas que ocorriam em vários registos

Na coluna 'HDI for year' detetaram-se missing values. Como estes representavam quase 70% dos registos o grupo decidiu que a melhor opção seria mesmo eliminar esta coluna em todos os modelos que pretendesse usar.

2.3.6 Transformamos em 0's e 1's os valores da feature 'sex'

As variáveis não-numéricas não conseguem entrar nas variáveis em todos os modelos como a regressão linear, assim entre outro tratamento de outras variáveis decidimos aplicar binary encoding à feature 'sex'. O valor zero representa os valores male e o valor um representa os valores female.

2.3.7 Através de uma função geradora de valores aleatórios, atribuímos valores numéricos aos valores da feature 'country'

Os valores de 'country' para poderem ser usados nos modelos precisavam ser convertidos em valores numéricos, decidimos optar pelo label encoding porque a quantidade de valores distintos era muito grande impedindo-nos de fazer one-hot-encoding que poderia ser uma tentativa com melhor

eficácia. Para isto, usamos um função geradora de números aleatórios para que o modelo não faça uma associação entre valores pois estes como representam países não têm nenhuma relação de ordem, por exemplo.

2.3.8 Adição de uma coluna 'continent' a partir de 'country'

Decidimos testar no modelo de regressão linear criar um coluna 'continent' para entender se a performance melhorava. A adição desta coluna deve-se à quantidade de países distintos, agrupar os países em continentes poderia levar a uma melhor prestação dos modelos que estavam com maus resultados. A construção desta coluna foi feita da seguinte forma:

1. Recolher a lista de países num ficheiro;
2. Escrever um ficheiro com registos que relacionam cada país-continente;
3. Ler esse ficheiro e, a partir disso, armazenar um dicionário com esses pares
4. Fazer um mapeamento nos registos para cada país colocar no lugar de continente o valor no dicionário associado ao país.

No final, obtivemos o seguinte conjunto de valores: Europe, South America, North America, Asia, Oceania e Africa. Apesar do esforço, não houve diferença na performance do modelo.

2.3.9 Eliminação da coluna 'generation'

Verificamos que alguns valores de 'age' únicos constavam em diferentes valores da feature 'generation', decidimos então eliminar a coluna 'generation' para todos os modelos que usamos pois consideramos que existia incoerência. **Nota:** Figura 4 associada.

2.3.10 Transformamos os valores da feature 'age' em valores numéricos

Os valores registados na coluna 'age' eram uma string indicando o intervalo de idades a que pertencia cada registo, assim impossibilitava-nos de colocá-los no modelo. Decidimos pois substituir cada valor único por um número numérico representativo do seu intervalo, no caso, representam o seguinte:

- 5-14 years: 0;
- 15-24 years: 1;
- 25-34 years: 2;
- 35-54 years: 3;
- 55-74 years: 4;
- 75+ years: 5.

2.3.11 Normalização de features

Avaliamos se, de acordo, com o modelo e com os resultados experimentados seria necessário normalizar os valores de algumas colunas em que a diferença entre os valores mínimo e máximo era elevada o que podia influenciar a performance do modelo. Quanto aos modelos Decision Tree Regressor e XGBoost a normalização não é necessária pois estes modelos comportam-se, adequadamente, a valores discrepantes, no entanto, para modelos de regressão pode, em algumas situações ter impacto.

2.4 Visualização de matriz de correlação pós-tratamento

Após tratar os dados produzimos uma matriz de correlação para perceber que variáveis melhor representavam a variável target. **Nota:** Figura 5 associada.

2.5 Análise dos modelos e Reflexão crítica

2.5.1 Linear Regression

Neste modelo experimentamos também cross-validation e observando os resultados obtidos com e sem cross-validation percebeu-se que não melhorava a má performance do modelo. Tanto que experimentando diferentes valores para o parâmetro cv apenas para um valor de 'cv = 2' se verificou um resultado de r^2 positivo, mas tão baixo que revelava fraco rendimento para este dataset. Além de cross-validation treinamos com diferentes condições, quer com normalização de algumas ou todas variáveis como o 'population', 'suicides_100k pop', 'suicides_no', 'gdp_for_year' e 'gdp_per_capita', quer com a criação da coluna 'continent'. Testamos também, por exemplo, com a presença e ausência dos outliers mencionados, anteriormente. Todo o resto de tratamento foi aplicado de forma global ao dataset para todos os modelos.

Com diferentes combinações no que diz respeito ao tratamento de dados e sempre com a presença de uma má prestação deste modelo concluímos que os dados não seguem, pelo menos alguns, totalmente uma relação linear com a componente que pretendemos prever. **Nota:** Figuras 6 e 7 associadas.

2.5.2 Decision Tree Regressor

No tratamento de dados para o Decision Tree Regressor não foi necessário criar a coluna 'continent' pois mesmo sem ela conseguimos obter bons resultados. No entanto, todo o restante tratamento comum a todos os modelos foi mantidos. Relativamente, aos outliers, como as árvores de decisão são robustas decidimos mantê-los sem aplicar qualquer alteração sobre os mesmos. Não foi necessário proceder a normalização de variáveis pela mesma razão da manutenção dos outliers para treino e teste.

Neste modelo utilizamos GridSearchCV para procurar uma melhor configuração que nos fornecesse bons resultados. Como parâmetros, na métrica de critério de avaliação usamos o r^2 e uma profundidade entre 1 e 10, no cv invocamos um KFold com parâmetros de split para dividir o dataset de treino em 7 e um shuffle para os dados serem misturados.

Após experimentar várias combinações para os diferentes parâmetros o grupo conseguiu obter um excelente resultado. **Nota:** Figura 8 associada.

2.5.3 XGBoost

Tal como no modelo Decision Tree Regressor o tratamento de dados que aplicamos ao XGBoost é o mesmo.

Neste modelo usamos também GridSearchCV, mudam os parâmetros, como estimador usamos XGBRegressor, como parâmetros de param aplicamos uma eta (taxa de aprendizagem de cada árvore) de 0.001, um total de 8000 estimadores e uma profundidade entre 1 e 3. Novamente, para o cv usamos um KFold que divide em 3 o dataset e aplica um shuffle para misturar os dados. Conseguimos também com este modelo obter bons resultados, surpreendentemente não tanto quanto aqueles que observamos na Decision Tree Regressor. **Nota:** Figura 9 associada.

3 Tarefa 2 - Dataset Competição

3.1 Descrição da Competição

A energia solar é uma das principais fontes de energias renováveis, desempenhando não só um papel fundamental na transição para fontes de energia limpa e renovável, mas também na promoção da sustentabilidade ambiental. Neste contexto, além de ser crucial otimizar o uso da energia solar, a relação entre o gasto e a produção energética é essencial para permitir um planeamento eficaz do consumo energético e a integração de sistemas de energia solar em redes elétricas existentes.

Assim, a competição consiste no desenvolvimento de modelos de **Machine Learning** capazes de prever, com precisão, a quantidade de energia elétrica, em kWh, gerada por painéis solares e injetada na rede elétrica existente a cada hora do dia com base numa ampla gama de atributos.

Este é, portanto, um problema de previsão de energia com impacto significativo na eficiência energética, mas também na redução das emissões de gases com efeito estufa e na promoção da sustentabilidade.

3.2 Objetivos

Relativamente ao presente estudo, explica-se de seguida os principais objetivos:

- desenvolver o melhor modelo possível capaz de prever a quantidade de energia, em kWh, produzida por painéis solares e injetada, em redes elétricas existentes, a cada hora do dia. Como métrica será utilizada a percentagem de acerto, i.e., a accuracy.
- prever, para cada registo do dataset de teste, a quantidade de energia que se injectou na rede elétrica num determinado ponto temporal, utilizando a escala **None**, **Low**, **Medium**, **High** e **Very_High**. É de notar que esta previsão deve ser feita a partir de um modelo de **Classificação** dado o tipo de dados da variável *target*.

Assim, ao longo do projeto, utilizámos um Jupiter Notebook de Python, junto com as várias técnicas de preparação, tratamento e visualização como mecanismo para alcançar os objetivos apresentados.

3.3 Descrição do Dataset

O dataset utilizado nesta competição contém um conjunto de features sendo de destacar a feature Injeção na rede (kWh) . Esta feature indica, numa escala qualitativa (None, Low, Medium, High e Very_High), a quantidade de energia que se injectou na rede elétrica num determinado ponto temporal (i.e., numa determinada hora de um determinado dia). Se o valor desta feature for None, significa que não foi injetada nenhuma energia na rede (ou porque nada foi obtido dos painéis solares, ou porque toda a energia produzida pelos painéis foi consumida localmente). Um valor de Very_High implica a existência de uma quantidade muito alta de energia injetada na rede elétrica naquele ponto temporal. No que toca aos datasets relativos à energia de 2021 e 2022 têm 2256 linhas e 6 colunas e 8760 linhas e 6 colunas respetivamente. Relativamente aos features presentes nos datasets ambos pussuem as mesmas fetures que são nomeadamente:

- Data - o timestamp associado ao registo, ao dia;
- Hora - a hora associada ao registo;
- Normal (kWh) - quantidade de energia eléctrica consumida, em kWh e proveniente da rede eléctrica, num período considerado normal em ciclos bi-horário diários (horas fora de vazio);
- Horário Económico (kWh) - quantidade de energia eléctrica consumida, em kWh e proveniente da rede eléctrica, num período considerado económico em ciclos bi-horário diários (horas de vazio);
- Autoconsumo (kWh) - quantidade de energia eléctrica consumida, em kWh, proveniente dos painéis solares;
- Injeção na rede (kWh) - quantidade de energia eléctrica injectada na rede eléctrica, em kWh, proveniente dos painéis solares.

No que toca aos datasets relativos à meteorologia de 2021 e 2022 têm 2928 linhas e 15 colunas e 8760 linhas e 15 colunas respetivamente. Relativamente aos features presentes nos datasets ambos possuem as mesmas features que são nomeadamente:

- dt - o timestamp associado ao registo;
- dt_iso - a data associada ao registo, ao segundo;
- city_name - o local em causa;
- temp - temperatura em ^{o}C;
- feels_like - sensação térmica em ^{o}C;
- temp_min - temperatura mínima sentida em ^{o}C;
- temp_max - temperatura máxima sentida em ^{o}C;
- pressure - pressão atmosférica sentida em atm;
- sea_level - pressão atmosférica sentida ao nível do mar em atm;
- grnd_level - pressão atmosférica sentida à altitude local em atm;

- humidity - humidade em percentagem;
- wind_speed - velocidade do vento em metros por segundo;
- rain_1h - valor médio de precipitação;
- clouds_all - nível de nebulosidade em percentagem;
- weather_description - avaliação qualitativa do estado do tempo.

3.4 Análise dos Dados

A análise deste dataset passa pela elaboração de um estudo geral do estado inicial dos dados, de modo a determinar e a aplicar o tratamento necessário antes da criação de modelos de aprendizagem automática.

3.4.1 Treino

Assim, as Figuras 10 e 11 apresentam o output da função *info()*, que permitiu obter informação estatísticas acerca do tipo de dados, do número e nome das colunas, do número de entradas e da quantidade de missing values presentes nos dataset de treino.

Dos resultados obtido, podemos ter uma ideia acerca do conjunto de features que deverão ser tratadas posteriormente, ou seja, as features com o tipo de dados *object* e as features com uma grande quantidade de missing values.

Posteriormente, de forma a facilitar a continuação do projeto, decidimos juntar os datasets correspondentes aos valores de energia nos anos de 2021 e 2022 e os datasets correspondentes aos valores de meteorologia nos anos de 2021 e 2022, ambos a partir de **full outer joins**. Desta abordagem surgiram os dataframes energia e meteo cujas informações estão representadas na Figura 12.

De seguida, tratamos de juntar ambos os dataframes energia e meteo num único dataframe e_m a partir de um **inner join**, dado que o intervalo de datas no dataframe meteo era maior do que no dataframa energia. Para aplicar esta abordagem, tivemos primeiro que criar uma coluna comum a ambos os dataframes. No caso, criámos a coluna **dt**, correspondente ao timestamp associado ao registo, no dataframe energia a partir das features Data e Hora.

A análise do novo dataframe pode ser observada com mais detalhe nas Figuras 16, 17, 18 e 19. Em comparação à análise individual de cada dataset, desta análise concluímos que existem features cujos valores únicos não são maiores do que 1, pelo que poderão ser removidas, e que há alguma discrepância entre valores das features, o que sugere uma eventual normalização dos dados. Para além disso, verificamos que os missing values da feature target correspondem à classe **None**, a mais representativa, pelo que deve ser transformada num valor numérico.

3.4.2 Teste

Os dados de teste, como seria de esperar, passaram por um processo semelhante àquele aplicado nos dados de treino o que podemos verificar nas Figuras 20, 21, 22 e 23.

Da análise destas figuras, percebemos que as features relacionadas com a meteorologia contém missing values, dado que o intervalo de datas do dataset de teste da meteorologia é mais pequeno do

que aquele que abrange o dataset de teste da energia. As restantes observações são iguais àquelas feitas no dataset de treino, nomeadamente, a existência de colunas com 0 ou 1 valores únicos, variáveis categóricas e colunas com intervalos de valores muito discrepantes entre si.

3.5 Tratamento de Dados

Após analisar exploratória do conjunto de dados, passamos ao trabalho desenvolvido em termos de tratamento e de limpeza inicial do dataset. A preparação de um dataset para a criação de modelos de Machine Learning não é tem uma solução única, dado que alguns modelos de podem impor a implementação de determinados passos de tratamento de dados pouco relevantes para outros modelos. Assim, o foco deste tratamento inicial passa por efetuar um conjunto de operações transversais a vários modelos de Machine Learning, pelo que, numa fase posterior, aplicamos uma preparação adicional dos dados.

3.5.1 Simplicação do Nome das Features

Para facilitar orientação ao longo do trabalho e a compreensão do significado de cada feature, começamos o tratamento de dados pela simplificação dos nomes das features. As alterações de nomes que realizamos nos datasets de treino e de teste encontram-se na Figura 14.

3.5.2 Remoção de Features

Em termos de remoção de features, nós removemos as features *sea_level* e *grnd_level*, uma vez que as são constituídas apenas por missing values. De seguida, removemos a feature *city_name*, dado que apresenta apenas um valor único. Por fim, removemos as features *Data_Hora*, *Data* e *Hora*, uma vez que a informação contida nos dados destas colunas encontra-se contida na coluna *Timestamp*.

As colunas resultantes podem ser observadas na Figura 15.

3.5.3 Alteração de datas

Dado que o datasets de treino e de teste têm duas colunas relacionadas com datas, nesta etapa do tratamento de dados decidimos verificar se ambas as colunas continham a mesma informação. Para isto, começamos por converter os tipos de dados das colunas para o tipo *datetime64[ns, UTC]*. De seguida, utilizamos a função *equals()* para verificar se todos os registos de cada coluna são iguais. Concluindo que ambas são iguais tratamos de remover a coluna *dt_iso*.

3.5.4 Converter tipos de dados categóricos para numéricos

Para o tipo de dados ser compatível com a maioria dos modelos de Machine Learning, convertemos as features categóricas *Injeção na Rede* e *Estado do Tempo* para dados númericos. Para isso, para cada uma destas features, testamos diferentes abordagens com o objetivo de tentar determinar qual delas revelava melhores resultados.

Entre os métodos de conversão que testamos estão *Label Encoding*, *One-Hot Encoding*, *Binary Encoding*, *Backward Difference Encoding* e *Factorize* para a features *Estado do Tempo* e *Label Encoding* e *Factorize* para a features *Injeção de Rede*. É de notar que, para a *Injeção de Rede* não testamos

One-Hot Encoding, *Binary Encoding*, *Backward Difference Encoding* por uma questão de simplicidade, já que estes métodos criam mais colunas.

3.5.5 Normalização

Como verificamos na análise dos dados, existem algumas features que apresentam um intervalo de dados muito discrepante em relação a outras, pelo que tratamos de normalizá-las com o objetivo de evitar que durante o processo de criação do modelo as features com valores maiores influenciem o modelo. Os resultados obtidos podem ser observados na Figura 24.

3.5.6 Tratamento dos Outliers

No que diz respeito ao tratamento de outliers, nós tentamos efetuar o tratamento de cada feature de várias formas diferentes. As formas que usamos foram *dropping outliers*, substituir pela mediana, *winsorize* e *log transformation*, pelo que a explicação de cada um deles é a seguinte:

- Dropping dos outliers - consiste na remoção de todas as linhas que contêm outliers a partir de limites definidos por nós.
- Winsorize - é um processo que envolve a definição de um limite superior e um limite inferior, e consiste em substituir os valores que ultrapassam esses limites pelos próprios limites. Isso significa que, se um valor estiver acima do limite superior, ele é substituído pelo valor do limite superior; se estiver abaixo do limite inferior, é substituído pelo valor do limite inferior. Por exemplo, se você escolhermos winsorizar os 5% superiores e inferiores de um conjunto de dados, os valores que estão nos 2,5% mais altos serão substituídos pelo valor no percentil 97,5, e os valores nos 2,5% mais baixos serão substituídos pelo valor no percentil 2,5.
- Log transformation - consiste na aplicação de do logaritmo a todos os outliers

Os box plots analisados podem ser verificados na Figura 40.

3.5.7 Missing Values

Para efetuar o tratamento dos missing values começamos por identificar as features que apresentam missing values que no caso do dataset de treino era apenas a *Precipitação* e no caso do dataset de teste eram *Temperatura*, *Sensação Térmica*, *Temperatura Mínima*, *Temperatura Máxima*, *Pressão Atmosférica*, *Humidade*, *Velocidade Vento*, *Precipitação Média*, *Nebulosidade*. Para cada um destes features recorremos a diferentes estratégias e avaliamos o impacto delas no modelo final para assim escolher qual das estratégias era a mais eficiente no tratamento dos missing values de cada feature. Entre as estratégias de tratamento que utilizamos estão substituição dos missing values pela média, pela mediana, interpolação e remoção da coluna (*Precipitação Média*), dado que a percentagem de missing values era muito elevada.

3.5.8 Exploratory Data Analysis

Observando a Figura 25 podemos ver a relação que algumas classes poderão ter entre si, ou melhor designando, quanto o aumento do valor de uma implica o aumento ou o decréscimo do valor de

outra. Se nos focarmos mais nos valores menores que -0.5 e os valores maiores que 0.5 notamos que: quer a *Temperatura Máxima* quer a *Temperatura Mínima* têm uma relação fortemente positiva com a *Temperatura* e com a *Sensação Térmica*, significando isto que quando um aumenta a outra também e, reparando nesta situação faz bastante sentido porque todas são classes que têm relação no contexto real com a temperatura ambiente. Uma vez mais pelo motivo anterior, a classe *Sensação Térmica* tem uma relação de 1 para 1 com a *Temperatura* e, fortemente positivo para as classes *Temperatura Mínima* e *Temperatura Máxima*. Para além disto, de notar que as classes *Temperatura*, *Sensação Térmica*, *Temperatura Mínima* e *Temperatura Máxima* têm uma relação satisfatoriamente forte negativa com a classe *Humidade*, significa isto que quando, por exemplo, os valores de *Humidade* diminuem os valores das outras classes aumentam e vice-versa. Além disso, também a classe *Humidade* tem uma relação satisfatoriamente forte negativa como o *Autoconsumo*, ou seja, quando o valor desta última diminui o valor da outra aumenta e vice-versa.

3.6 Análise dos modelos e Reflexão crítica

De forma a tentar obter os melhores resultados, decidimos utilizar modelos baseados em árvores, nomeadamente, o *Random Forest* e o *XGBoost*. Desta forma, as próximas secções tratam de explicar o tratamento mais aprofundado aplicado a cada modelo bem como as justificações das decisões tomadas.

Inicialmente, a análise de dados é feita de igual forma para ambos os modelos. No entanto, uma vez que concluímos que o modelo criado pelo *XGBoost* teve um melhor desempenho a nível de accuracy, decidimos tentar optimizá-lo. Desta forma, recorremos a uma API [1] para obter valores que permitem preencher os missing values que inicialmente encontrámos no dataset da meteorologia, nomeadamente, nas colunas *sea_level*, *grnd_level* e nas colunas correspondentes aos dados meteorológicos do dataset de teste já que o intervalo de datas era menor comparativamente ao de treino.

3.7 Random Forest

No modelo do Random Forest a abordagem seguida para a análise de dados foi a mesma que foi descrita em cima. No que diz respeito ao tratamento para este modelo fizemos simplificação e remoção tal como foi explicado, a converção as features *Injeção de Rede* e *Estado do tempo* foi realizada através do *Label Encoding* e a normalização foi feita usando o z score. Para além disso, adicionamos colunas a partir dos dados que já possuiamos(Feature Engineering), para isso convertemos o tipo de dados da coluna *Timestamp* para datetime e criamos as colunas *Dia* e *Mês* com partes extraídas do *Timestamp*.No tratamento dos outliers os features tratados foram *Temperatura Mínima* com o winsorize, a *Temperatura Máxima* e o *Autoconsumo* através da remoção das linhas com outliers, a *Velocidade do Vento* e o *Horário Económico* através da substituição dos outliers pela mediana. Relativamente aos missing values a features que tratamos foram a *Humidade* e a *Nebolusidade* atráves da substituição dos outliers pela mediana e a *Precipitação média* atráves da remoção da coluna. A accuracy que obtemos com o Random Forest testando no nosso computador foi **0.8990** e na competição pública foi **0.8417**.

3.8 XGBoost

Como foi referido, o modelo XGBoost seguiu inicialmente a mesma análise de dados explicada anteriormente. No entanto, apesar de ter obtido melhores resultados relativamente ao modelo Ranfom Forest, decidimos aplicar novas estratégias de tratamento de dados, nomeadamente, a partir de novos

dados obtidos a partir da API, pelo que achamos relevante continuar este relatório segundo esta abordagem. É de notar que muitos dos outros tratamentos aplicados aos dados foram inicialmente aplicados no dataset original referente à meteorologia, pelo que decidimos manter, dado que mostrou trazer vantagens, em termos de accuracy, para o novo dataset. Durante o desenvolvimento deste tema vamos apenas expor as novas mudanças.

3.8.1 Descrição do Dataset

- dt - timestamp associado ao registo;
- temp - temperatura em ºC;
- humidity - humidade em percentagem;
- dew_point - Temperatura do ponto de orvalho acima do solo
- feels_like - sensação térmica em ºC;
- rain_1h - valor médio de precipitação;
- sea_level - pressão atmosférica sentida ao nível do mar em atm;
- grnd_level - pressão atmosférica sentida à altitude local em atm;
- clouds_all - nível de nebulosidade em percentagem;
- evapotranspiration - soma diária da taxa de evapotranspiração de referência de um campo de erva;
- vapour_pressure_deficit (kPa) - déficit de pressão de vapor(VDP) em quilopascal. Para VPD elevado, a transpiração de água das plantas aumenta. Para VPD baixo, a transpiração diminui.
- wind_speed - velocidade do vento em metros por segundo;
- wind_direction - a direção do vento acima do solo.
- wind_gusts - Rajadas de vento no solo numa hora indicada. As rajadas de vento são definidas como as rajadas de vento máximas da hora anterior.
- soil_temperature - Temperatura média de diferentes níveis de solo abaixo do solo.
- soil_moisture - Conteúdo médio de água no solo como razão de mistura volumétrica em profundidades.
- is_day - Indicador booleano que informa se é dia ou noite.
- sunshine_duration (s) - Número de segundos de luz solar da hora anterior por hora, calculado por irradiância direta normalizada superior a 120 W/m², de acordo com a definição da OMM.
- direct_radiation_instant - Radiação solar direta como média da hora anterior no plano horizontal e no plano normal (perpendicular ao sol)

- direct_normal_irradiance_instant - Irradiância solar direta instantânea no plano normal (perpendicular ao sol). Essa medida representa a quantidade de energia solar direta que atinge uma superfície por unidade de área, considerando a posição do sol em relação à superfície;
- weather_description - avaliação qualitativa do estado do tempo.

3.8.2 Análise de Dados

Relativamente à análise de dados, destacamos a existência de missing values apenas na feature *Injeção na rede (kWh)*, o que não é relevante, dado que estes valores correspondem ao nível **None** da escala de valores desta feature. (Figura 28)

Para além disso, podemos verificar na Figura 27 o tipo de dados e a quantidade de colunas e linhas no dataset.

3.8.3 Tratamento de Dados

Simplificação das features

Tal como no anterior dataset efetuamos uma simplificação das variáveis de modo a tentar representar melhor o seu significado real e, por isso, melhorando a sua legibilidade. Na figura 26 consta a simplificação que foi efetuada nas features.

Normalização

Dado que a normalização implementada inicialmente não mostrou melhorias nos resultados do modelo, tentamos aplicar a normalização z-score. No entanto esta abordagem, também não mostrou melhorias. Após alguma pesquisa, chegamos à conclusão que o modelo não é afetado por diferentes escalas nos dados de *input*, pelo que decidimos remover.

Tratamento dos Outliers

No que toca ao tratamento dos outliers, numa tentativa de melhorar a accuracy do modelo tentamos aplicar outros métodos ao novo dataset, nomeadamente, a substituição por missing values. Assim, a partir da função evidenciada na Figura 29, fomos capazes de selecionar o melhor limite superior e inferior de cada feature que consideramos relevante e de substituir os respetivos outliers por missing values.

Exploratory Data Analysis

Em seguida, foi realizada uma análise da correlação entre as variáveis com o objetivo de concluir quais features devíamos incorporar nos modelos. Como podemos ver pela figura 30 e considerando, principalmente, os valores com uma relação forte negativa, ou seja, inferiores a -0.7, por exemplo, ou com uma relação forte positiva, ou seja, inferiores a 0.7, a classe *Temperatura* tem uam forte relação negativa com as classes *Evapotranspiração* e o *Déficit de Vapor* e a classe *injeção da Rede* tem o mesmo tipo de relação com as classes *Evapotranspiração* e *Autoconsumo*, significa que quando o valor numa aumenta na outra com a qual se relaciona o valor diminui. Para além disso, existem muitas classes que têm uma relação forte positiva, ou seja, quando o valor de uma aumenta o da outra aumenta também, por exemplo, com valor absoluto superior 80 principalmente, isto ocore entre a classe *Temperatura Solo* com *Temperatura*, *Ponto de Orvalho* e *Sensação Térmica*, entre a classe *Evapotranspiração*, e as classes *Duração Solar*, *Radiação Solar* e a *Irradiação Solar* e a *Autoconsumo*, e a classe *Duração Solar* com as classes *Radiação Solar* e a *Irradiação Solar*, e, por fim a classe *Radiação Solar* com a classe

Irradiância Solar.

Depois de várias tentativas de remoção de features, chegamos à conclusão que o melhor desempenho foi obtido com a remoção das features *Ponto de Orvalho*, *Timestamp*, *Humidade*, *Evapotranspiração*, *Déficit de Vapor*, *Radiação Solar*, *é_de_Dia*, *Irradiância Solar*, *Duração Solar*, *Temperatura Solo* e *Sensação Térmica*. Estas conclusões foram também obtidas a partir da análise de relatórios de *feature importance* onde podemos verificar quais as features que mais influenciam a execução do modelo.

Abordagens Escolhidas

Entre as abordagens escolhidas para tentar melhorar o desempenho do dataset, houve algumas que se destacaram positivamente, entre elas o tunning de hiperparâmetros a partir do *GridSearchCV*. Após várias experiências, verificamos na Figura 31 os melhores valores para cada parâmetro.

Relativamente aos teste que fizemos mas que não melhoraram significativamente o resultado do modelo, podemos dizer que tentamos o seguinte:

- aplicar outras de cross validation, nomeadamente o StratifiedKFold e o RepeatedStratifiedKFold.
- balanceamento de classes, dado que segundo a análise de um *Classification Report*, a classe 1 (Low) da feature target tinha pouca representatividade relativamente às restantes classes. Para tentar resolver este problema tentamos duas abordagens: utilizar a função *compute_class_weight()* do XGBoost e os métodos de *Undersampling* e *Oversampling*.

Para além disso, fizemos uma análise de overfit comparando a accuracy dos dados de treino com a accuracy dos dados de teste e comparando a função loss entre os mesmos conjuntos de dados. A partir da Figura 32, podemos verificar que ainda existe algum overfit que tentamos resolver aplicando as abordagens anteriormente referidas e ajustando hiperparâmetros, nomeadamente, a L1 e a L2 regularization, o número de iterações e a *learning rate*. Para além disso, implementamos um método de *early stopping*, isto é, um método que termina a execução das iterações do modelo quando verifica que este começa a ter um aumento na função loss.

Desta forma, podemos concluir que o XGBoost obteve uma classificação de 0.86329 na classificação privada do kaggle.

4 Conclusão

O relatório apresenta uma abordagem estruturada na exploração e modelagem de dois conjuntos de dados distintos: "Suicide Rates Overview 1985 to 2016"e conjuntos de dados energéticos e meteorológicos. O uso da metodologia CRISP-DM proporcionou uma análise organizada, desde a compreensão dos dados até a avaliação dos modelos.

Embora que a Regressão Linear tenha mostrado uma má performance no dataset de suicídios, os modelos Decision Tree Regressor e XGBoost apresentaram resultados notáveis.

No segundo dataset de estudo modelos como Decision Tree Regressor e XGBoost demonstraram um bom desempenho, destacando a importância da qualidade dos dados e do tratamento adequado. Para melhorar a exploração de dados no primeiro dataser poderia se ter usado técnicas mais avançadas, como redes neurais, e aprimorar o tratamento de outliers. Além disso, poderia ter sido feito uma avaliação mais profunda do impacto das variáveis podem melhorar a robustez dos modelos. Já conjunto de dados energéticos e meteorológicos a validação implementado o Ensemble Learning a partir de vários modelos, criar mais modelos, nomeadamente, Redes Neuronais e Support Vector Machine . Ainda há espaço para refinamentos, mas os resultados atuais oferecem uma base sólida para futuras melhorias na predição de padrões energéticos e meteorológicos.

4.1 Bibliografia

[1] API Open Meteo- https://open-meteo.com/en/docs/historical-weather-api#latitude=41.5503&longitude=-8.42&hourly=dew_point_2m,rain,direct_radiation_instant,direct_normal_irradiance_instant&daily=&wind_speed_unit=ms&timeformat=unixtime&timezone=Europe%2FLondon.

5 Anexos

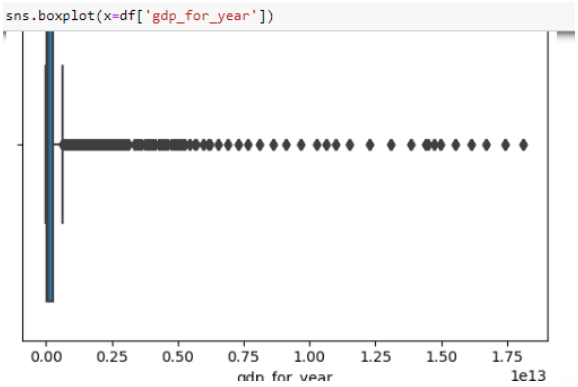

Figura 1: Outliers na coluna gdp_for_year

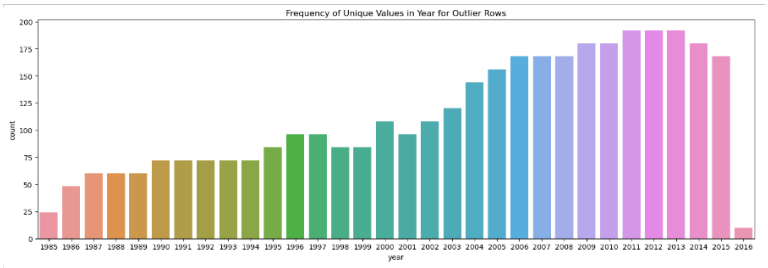

Figura 2: Frequência de cada ano nos valores que são outliers

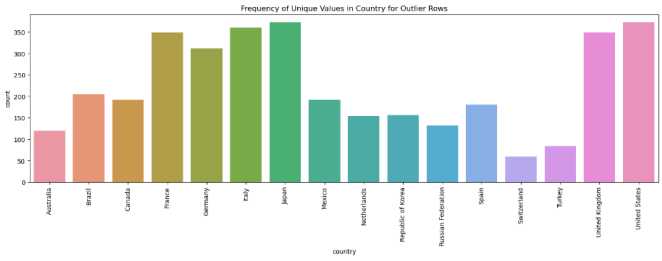

Figura 3: Frequência de cada país nos valores que são outliers

```
gens = df['generation'].unique()
for g in gens:
    print(f'For the {g} generation we have this ages: \n')
    ages = df[df['generation'] == g]['age'].unique()
    for idade in ages:
        print(f"  {idade}\n")
```

```
For the Generation X generation we have this ages:

  15-24 years

  5-14 years

  25-34 years

  35-54 years

For the Silent generation we have this ages:

  35-54 years

  55-74 years

  75+ years

For the G.I. Generation generation we have this ages:
```

```
"""
Remove a coluna generation
"""
df = df.drop('generation',axis=1)
```

Figura 4: Eliminação da coluna generation

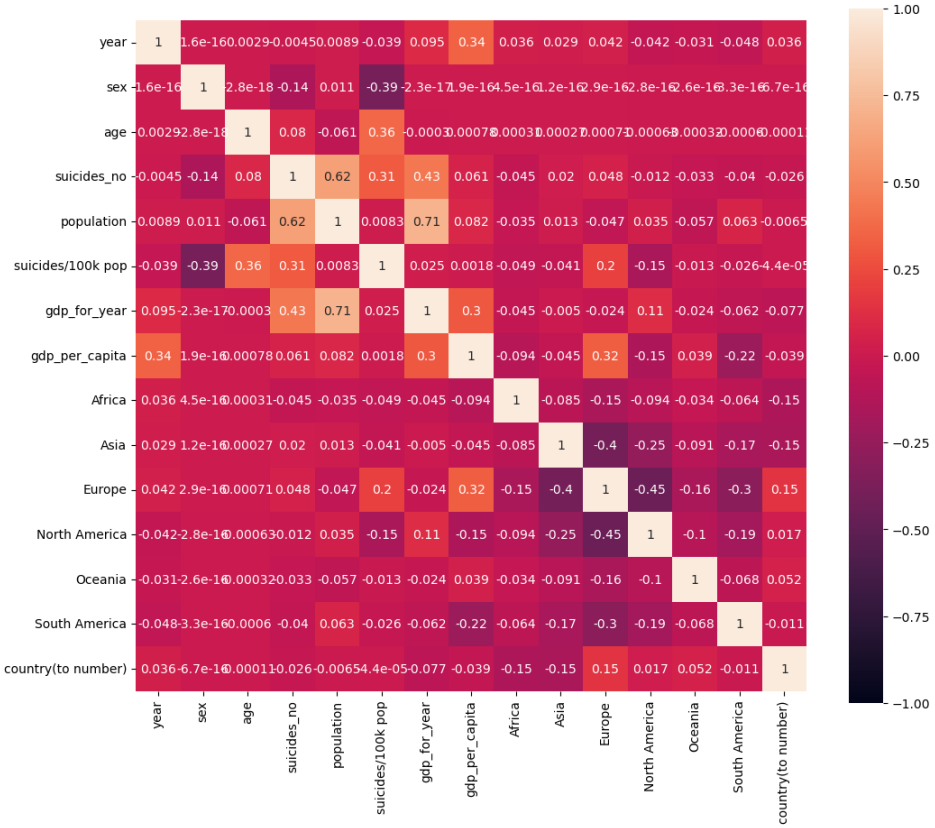

Figura 5: Matriz de Correlação após tratamento de dados

Figura 6: Treino e avaliação do modelo de regressão linear sem cross-validation

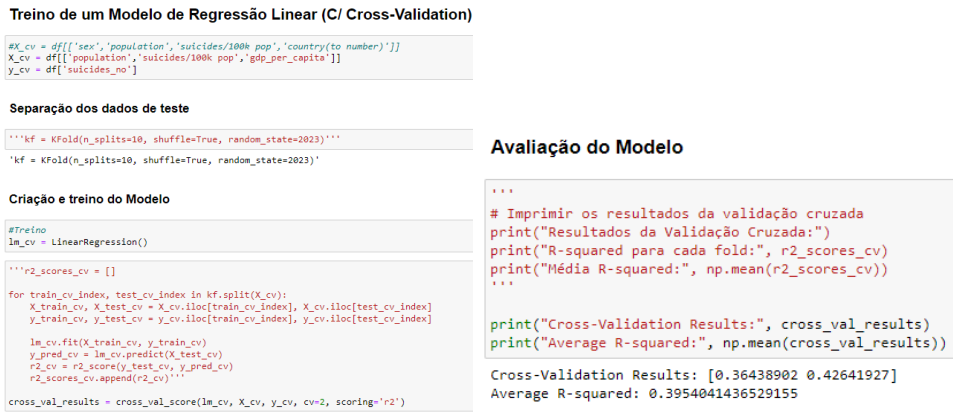

Figura 7: Treino e avaliação do modelo de regressão linear com cross-validation

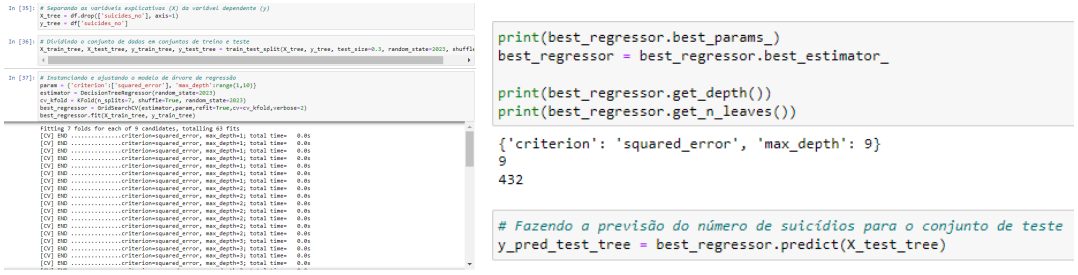

Figura 8: Treino e avaliação do modelo da árvore de regressão

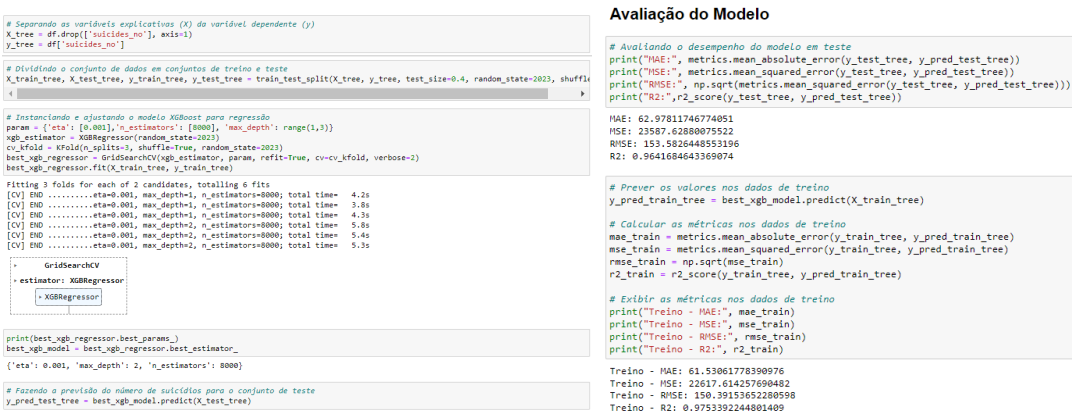

Figura 9: Treino e avaliação do modelo XGBoost

Figura 10: Função *info()* aplicada aos datasets de treino referentes à energia.

```
meteo_2021.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2928 entries, 0 to 2927
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   dt                   2928 non-null   int64
 1   dt_iso               2928 non-null   object
 2   city_name            2928 non-null   object
 3   temp                 2928 non-null   float64
 4   feels_like           2928 non-null   float64
 5   temp_min             2928 non-null   float64
 6   temp_max             2928 non-null   float64
 7   pressure             2928 non-null   int64
 8   sea_level            0 non-null      float64
 9   grnd_level           0 non-null      float64
 10  humidity             2928 non-null   int64
 11  wind_speed           2928 non-null   float64
 12  rain_1h              537 non-null    float64
 13  clouds_all           2928 non-null   int64
 14  weather_description  2928 non-null   object
dtypes: float64(8), int64(4), object(3)
memory usage: 343.2+ KB
```

```
meteo_2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   dt                   8760 non-null   int64
 1   dt_iso               8760 non-null   object
 2   city_name            8760 non-null   object
 3   temp                 8760 non-null   float64
 4   feels_like           8760 non-null   float64
 5   temp_min             8760 non-null   float64
 6   temp_max             8760 non-null   float64
 7   pressure             8760 non-null   int64
 8   sea_level            0 non-null      float64
 9   grnd_level           0 non-null      float64
 10  humidity             8760 non-null   int64
 11  wind_speed           8760 non-null   float64
 12  rain_1h              1898 non-null   float64
 13  clouds_all           8760 non-null   int64
 14  weather_description  8760 non-null   object
dtypes: float64(8), int64(4), object(3)
memory usage: 1.0+ MB
```

Figura 11: Função *info()* aplicada aos datasets de treino referentes à meteorologia.

```
energia.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11016 entries, 0 to 11015
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Data                      11016 non-null  object
 1   Hora                      11016 non-null  int64
 2   Normal (kWh)              11016 non-null  float64
 3   Horário Económico (kWh)   11016 non-null  float64
 4   Autoconsumo (kWh)         11016 non-null  float64
 5   Injeção na rede (kWh)     3239 non-null   object
dtypes: float64(3), int64(1), object(2)
memory usage: 516.5+ KB
```

```
meteo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11688 entries, 0 to 11687
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   dt                   11688 non-null  int64
 1   dt_iso               11688 non-null  object
 2   city_name            11688 non-null  object
 3   temp                 11688 non-null  float64
 4   feels_like           11688 non-null  float64
 5   temp_min             11688 non-null  float64
 6   temp_max             11688 non-null  float64
 7   pressure             11688 non-null  int64
 8   sea_level            0 non-null      float64
 9   grnd_level           0 non-null      float64
 10  humidity             11688 non-null  int64
 11  wind_speed           11688 non-null  float64
 12  rain_1h              2435 non-null   float64
 13  clouds_all           11688 non-null  int64
 14  weather_description  11688 non-null  object
dtypes: float64(8), int64(4), object(3)
memory usage: 1.3+ MB
```

Figura 12: Informação referente aos novos conjuntos de dados.

```
# Junção das colunas 'Data' e 'Hora' para criar a coluna 'Data_Hora'
energia['Data_Hora'] = energia['Data'] + ":" + energia['Hora'].astype(str)

# Criação da coluna 'dt' com timetamps a partir da coluna 'Data_Hora'
energia['dt'] = energia['Data_Hora'].apply(lambda x: pd.to_datetime(x,format='%Y-%m-%d:%H').timestamp()).astype('int64')
energia.head()
```

	Data	Hora	Normal (kWh)	Horário Económico (kWh)	Autoconsumo (kWh)	Injeção na rede (kWh)	Data_Hora	dt
0	2021-09-29	0	0.0	0.0	0.0	NaN	2021-09-29:0	1632873600
1	2021-09-29	1	0.0	0.0	0.0	NaN	2021-09-29:1	1632877200
2	2021-09-29	2	0.0	0.0	0.0	NaN	2021-09-29:2	1632880800
3	2021-09-29	3	0.0	0.0	0.0	NaN	2021-09-29:3	1632884400
4	2021-09-29	4	0.0	0.0	0.0	NaN	2021-09-29:4	1632888000

Figura 13: Criação da coluna **dt** no dataframe energia.

```python
e_m.rename(columns={"dt": "Timestamp"}, inplace=True)
e_m.rename(columns={"temp": "Temperatura"}, inplace=True)
e_m.rename(columns={"feels_like": "Sensação Térmica"}, inplace=True)
e_m.rename(columns={"temp_min": "Temperatura Mínima"}, inplace=True)
e_m.rename(columns={"temp_max": "Temperatura Máxima"}, inplace=True)
e_m.rename(columns={"pressure": "Pressão Atmosférica"}, inplace=True)
e_m.rename(columns={"humidity": "Humidade"}, inplace=True)
e_m.rename(columns={"wind_speed": "Velocidade Vento"}, inplace=True)
e_m.rename(columns={"rain_1h": "Precipitação Média"}, inplace=True)
e_m.rename(columns={"clouds_all": "Nebulosidade"}, inplace=True)
e_m.rename(columns={"weather_description": "Estado do Tempo"}, inplace=True)
e_m.rename(columns={"Normal (kWh)": "Normal"}, inplace=True)
e_m.rename(columns={"Horário Económico (kWh)": "Horário Económico"}, inplace=True)
e_m.rename(columns={"Autoconsumo (kWh)": "Autoconsumo"}, inplace=True)
e_m.rename(columns={"Injeção na rede (kWh)": "Injeção na Rede"}, inplace=True)
```

Figura 14: Função *rename()* aplicada aos datasets.

```python
teste.columns
```

✓ 0.0s

```
Index(['Timestamp', 'dt_iso', 'Temperatura', 'Sensação Térmica',
       'Temperatura Mínima', 'Temperatura Máxima', 'Pressão Atmosférica',
       'Humidade', 'Velocidade Vento', 'Precipitação Média', 'Nebulosidade',
       'Estado do Tempo', 'Normal', 'Horário Económico', 'Autoconsumo'],
      dtype='object')
```

Figura 15: Drop das features irrelevantes no contexto do dataframe e_m.

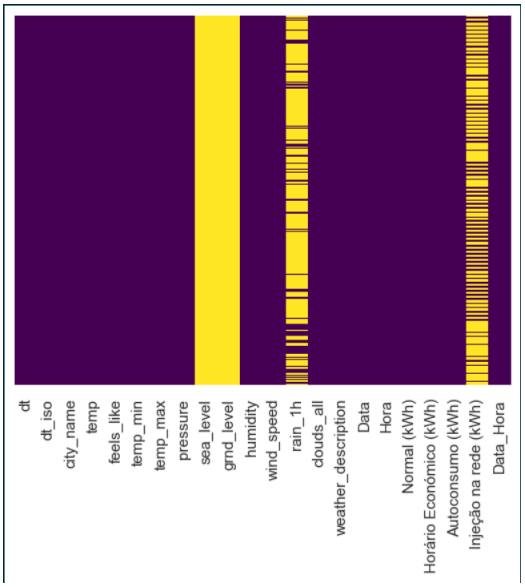

Figura 16: Heatmap com os missing values do dataframe e_m.

```
e_m.nunique()
```

```
dt                          11016
dt_iso                      11016
city_name                       1
temp                         2382
feels_like                   2670
temp_min                      481
temp_max                      545
pressure                       41
sea_level                       0
grnd_level                      0
humidity                       82
wind_speed                    771
rain_1h                       367
clouds_all                    101
weather_description             8
Data                          459
Hora                           24
Normal (kWh)                 1282
Horário Económico (kWh)       851
Autoconsumo (kWh)             752
Injeção na rede (kWh)           4
Data_Hora                   11016
dtype: int64
```

Figura 17: Unique values presentes no dataframe e_m.

```
e_m_count = e_m["Injeção na rede (kWh)"].value_counts()
sns.set(style="darkgrid")
sns.barplot(x=e_m_count.index, y=e_m_count.values)
plt.title('')
plt.ylabel('Número de Ocorrências', fontsize=12)
plt.xlabel('Energia Injetada na Rede', fontsize=12)
plt.show()
```

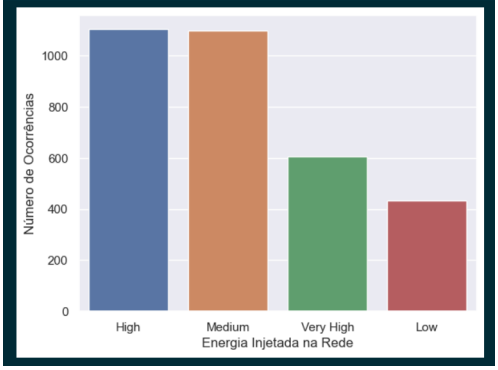

Figura 18: Número de ocorrências do target no dataframe e_m.

	dt	temp	feels_like	temp_min	temp_max	pressure	sea_level	grnd_level
count	1.101600e+04	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000	0.0	0.0
mean	1.652701e+09	16.182991	15.826868	14.195064	17.288098	1018.396605	NaN	NaN
std	1.144868e+07	5.756972	6.264860	4.956014	6.164689	6.203379	NaN	NaN
min	1.632874e+09	0.320000	-2.190000	-0.640000	1.330000	994.000000	NaN	NaN
25%	1.642787e+09	12.080000	11.507500	10.720000	12.890000	1015.000000	NaN	NaN
50%	1.652701e+09	15.625000	15.230000	14.230000	16.260000	1018.000000	NaN	NaN
75%	1.662614e+09	19.270000	19.260000	17.490000	20.340000	1022.000000	NaN	NaN
max	1.672528e+09	40.850000	41.330000	36.720000	41.450000	1034.000000	NaN	NaN

humidity	wind_speed	rain_1h	clouds_all	Hora	Normal (kWh)	Horário Económico (kWh)	Autoconsumo (kWh)
11016.000000	11016.000000	2284.000000	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000
76.570715	2.672039	0.896642	54.000908	11.500000	0.202278	0.159714	0.117314
16.268260	1.476158	1.079381	40.553002	6.922501	0.349478	0.271792	0.176762
19.000000	0.060000	0.100000	0.000000	0.000000	0.000000	0.000000	0.000000
66.000000	1.620000	0.210000	7.000000	5.750000	0.000000	0.000000	0.000000
81.000000	2.400000	0.460000	60.000000	11.500000	0.000000	0.000000	0.000000
91.000000	3.420000	1.092500	98.000000	17.250000	0.314000	0.288000	0.227000
100.000000	11.100000	7.450000	100.000000	23.000000	3.251000	6.978000	1.192000

Figura 19: Função *describe* aplicada ao dataframe e_m.

```
teste.info()
✓ 0.0s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2256 entries, 0 to 2255
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   dt                        2256 non-null   int64
 1   dt_iso                    1752 non-null   object
 2   city_name                 1752 non-null   object
 3   temp                      1752 non-null   float64
 4   feels_like                1752 non-null   float64
 5   temp_min                  1752 non-null   float64
 6   temp_max                  1752 non-null   float64
 7   pressure                  1752 non-null   float64
 8   sea_level                 0 non-null      float64
 9   grnd_level                0 non-null      float64
 10  humidity                  1752 non-null   float64
 11  wind_speed                1752 non-null   float64
 12  rain_1h                   206 non-null    float64
 13  clouds_all                1752 non-null   float64
 14  weather_description       1752 non-null   object
 15  Data                      2256 non-null   object
 16  Hora                      2256 non-null   int64
 17  Normal (kWh)              2256 non-null   float64
 18  Horário Económico (kWh)   2256 non-null   float64
 19  Autoconsumo (kWh)         2256 non-null   float64
 20  Data_Hora                 2256 non-null   object
dtypes: float64(14), int64(2), object(5)
memory usage: 370.2+ KB
```

Figura 20: Informação referente ao conjunto de dados de teste (após o full outer join).

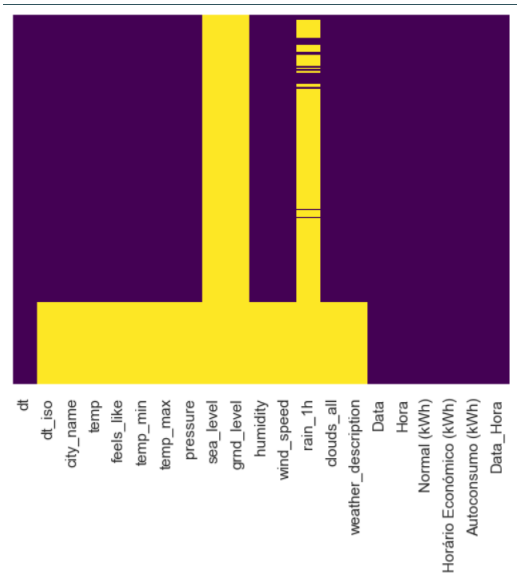

Figura 21: Heatmap com os missing values do dataframe teste.

```
teste.nunique()
✓ 0.0s
```

dt	2256
dt_iso	1752
city_name	1
temp	889
feels_like	1007
temp_min	169
temp_max	174
pressure	31
sea_level	0
grnd_level	0
humidity	71
wind_speed	512
rain_1h	120
clouds_all	99
weather_description	8
Data	94
Hora	24
Normal (kWh)	709
Horário Económico (kWh)	490
Autoconsumo (kWh)	485
Data_Hora	2256

dtype: int64

Figura 22: Unique values presentes no dataframe teste.

	dt	temp	feels_like	temp_min	temp_max	pressure	sea_level	grnd_level
count	2.256000e+03	1752.000000	1752.000000	1752.000000	1752.000000	1752.000000	0.0	0.0
mean	1.676590e+09	10.599606	9.543265	9.177917	11.643813	1023.493721	NaN	NaN
std	2.345024e+06	3.715156	4.191224	4.021176	3.596882	6.650873	NaN	NaN
min	1.672531e+09	0.930000	-1.790000	-0.850000	3.340000	1006.000000	NaN	NaN
25%	1.674561e+09	7.877500	6.217500	6.247500	8.900000	1020.000000	NaN	NaN
50%	1.676590e+09	10.950000	10.135000	9.710000	11.820000	1024.000000	NaN	NaN
75%	1.678620e+09	13.292500	12.742500	12.320000	14.040000	1029.000000	NaN	NaN
max	1.680649e+09	20.610000	19.790000	20.010000	22.010000	1036.000000	NaN	NaN

	humidity	wind_speed	rain_1h	clouds_all	Hora	Normal (kWh)	Horário Económico (kWh)	Autoconsumo (kWh)
count	1752.000000	1752.000000	206.000000	1752.000000	2256.000000	2256.000000	2256.000000	2256.000000
mean	75.639840	2.873613	1.025097	44.489726	11.500000	0.268060	0.226809	0.119439
std	17.415468	1.620450	1.209617	42.972438	6.923721	0.464323	0.375752	0.190601
min	23.000000	0.120000	0.110000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	62.000000	1.800000	0.250000	0.000000	5.750000	0.000000	0.000000	0.000000
50%	81.000000	2.490000	0.565000	35.000000	11.500000	0.000000	0.000000	0.000000
75%	91.000000	3.500000	1.245000	96.000000	17.250000	0.398750	0.336000	0.237250
max	97.000000	10.320000	6.380000	100.000000	23.000000	3.381000	2.771000	1.161000

Figura 23: Função *describe* aplicada ao dataframe teste.

```
e_m_norm.describe()
```
✓ 0.2s Python

	Timestamp	Temperatura	Sensação Térmica	Temperatura Mínima	Temperatura Máxima	Pressão Atmosférica	Humidade	Velocidade Vento	Precipitação Média	Nebulosidade	Normal	Horário Económico	Autoconsumo
count	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000	11016.000000	2284.000000	11016.000000	11016.000000	11016.000000	11016.000000
mean	0.500000	0.391389	0.413991	0.397084	0.397759	0.609915	0.710750	2.672039	0.896642	0.540009	0.202278	0.159714	0.117314
std	0.288714	0.142042	0.143954	0.132656	0.153656	0.155084	0.200843	1.476158	1.079381	0.405530	0.349478	0.271792	0.176762
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.060000	0.100000	0.000000	0.000000	0.000000	0.000000
25%	0.250000	0.290155	0.314740	0.304069	0.288136	0.525000	0.580247	1.620000	0.210000	0.070000	0.000000	0.000000	0.000000
50%	0.500000	0.377622	0.400276	0.398019	0.372134	0.600000	0.765432	2.400000	0.460000	0.600000	0.000000	0.000000	0.000000
75%	0.750000	0.467555	0.492877	0.485278	0.473829	0.700000	0.888889	3.420000	1.092500	0.980000	0.314000	0.288000	0.227000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	11.100000	7.450000	1.000000	3.251000	6.978000	1.192000

```
teste_norm.describe()
```
✓ 0.2s Python

	Timestamp	Temperatura	Sensação Térmica	Temperatura Mínima	Temperatura Máxima	Pressão Atmosférica	Humidade	Velocidade Vento	Precipitação Média	Nebulosidade	Estado do Tempo	Normal	Horário Económico	Autoconsumo
count	2256.000000	1752.000000	1752.000000	1752.000000	1752.000000	1752.000000	1752.000000	1752.000000	206.000000	1752.000000	2256.000000	2256.000000	2256.000000	2256.000000
mean	0.500000	0.491342	0.525174	0.480725	0.444768	0.583124	0.711349	2.873613	1.025097	0.444897	4.242465	0.268060	0.226809	0.119439
std	0.288867	0.188778	0.194218	0.192770	0.192656	0.221696	0.235344	1.620450	1.209617	0.429724	2.846897	0.464323	0.375752	0.190601
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.120000	0.110000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.250000	0.353023	0.371061	0.340244	0.297804	0.466667	0.527027	1.800000	0.250000	0.000000	1.000000	0.000000	0.000000	0.000000
50%	0.500000	0.509146	0.552595	0.506232	0.454205	0.600000	0.783784	2.490000	0.565000	0.350000	4.000000	0.000000	0.000000	0.000000
75%	0.750000	0.628176	0.673424	0.631352	0.573112	0.766667	0.918919	3.500000	1.245000	0.960000	7.000000	0.398750	0.336000	0.237250
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	10.320000	6.380000	1.000000	8.000000	3.381000	2.771000	1.161000

Figura 24: Normalização dos dados.

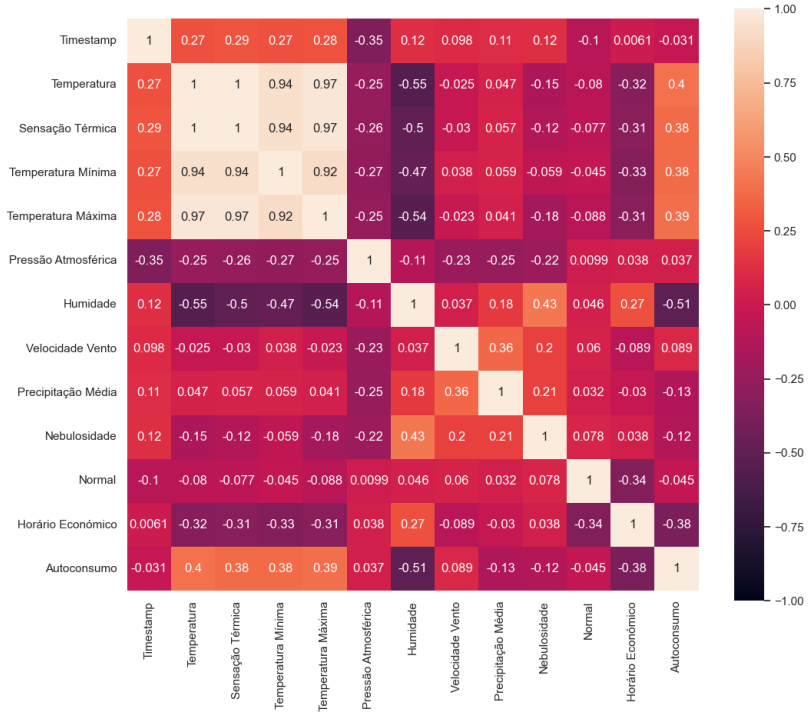

Figura 25: Matriz de correlação do dataframe e_m.

```python
e_m.rename(columns={"dt": "Timestamp"}, inplace=True)
e_m.rename(columns={"temp": "Temperatura"}, inplace=True)
e_m.rename(columns={"dew_point": "Ponto de Orvalho"}, inplace=True)
e_m.rename(columns={"feels_like": "Sensação Térmica"}, inplace=True)
e_m.rename(columns={"humidity": "Humidade"}, inplace=True)
e_m.rename(columns={"wind_speed": "Velocidade Vento"}, inplace=True)
e_m.rename(columns={"rain_1h": "Precipitação Média"}, inplace=True)
e_m.rename(columns={"sea_level": "Pressão Mar"}, inplace=True)
e_m.rename(columns={"grnd_level": "Pressão Local"}, inplace=True)
e_m.rename(columns={"clouds_all": "Nebulosidade"}, inplace=True)
e_m.rename(columns={"evapotranspiration": "Evapotranspiração"}, inplace=True)
e_m.rename(columns={"vapour_pressure_deficit (kPa)": "Déficit de Vapor"}, inplace=True)
e_m.rename(columns={"wind_direction": "Direção Vento"}, inplace=True)
e_m.rename(columns={"wind_gusts": "Rajadas Vento"}, inplace=True)
e_m.rename(columns={"soil_temperature": "Temperatura Solo"}, inplace=True)
e_m.rename(columns={"soil_moisture": "Humidade Solo"}, inplace=True)
e_m.rename(columns={"is_day": "é_de_Dia"}, inplace=True)
e_m.rename(columns={"sunshine_duration (s)": "Duração Solar"}, inplace=True)
e_m.rename(columns={"direct_radiation_instant": "Radiação Solar"}, inplace=True)
e_m.rename(columns={"direct_normal_irradiance_instant": "Irradiância Solar"}, inplace=True)
e_m.rename(columns={"weather_description": "Estado do Tempo"}, inplace=True)
e_m.rename(columns={"Normal (kWh)": "Normal"}, inplace=True)
e_m.rename(columns={"Horário Económico (kWh)": "Horário Económico"}, inplace=True)
e_m.rename(columns={"Autoconsumo (kWh)": "Autoconsumo"}, inplace=True)
e_m.rename(columns={"Injeção na rede (kWh)": "Injeção na Rede"}, inplace=True)
```

```python
teste.rename(columns={"dt": "Timestamp"}, inplace=True)
teste.rename(columns={"temp": "Temperatura"}, inplace=True)
teste.rename(columns={"dew_point": "Ponto de Orvalho"}, inplace=True)
teste.rename(columns={"feels_like": "Sensação Térmica"}, inplace=True)
teste.rename(columns={"humidity": "Humidade"}, inplace=True)
teste.rename(columns={"wind_speed": "Velocidade Vento"}, inplace=True)
teste.rename(columns={"rain_1h": "Precipitação Média"}, inplace=True)
teste.rename(columns={"sea_level": "Pressão Mar"}, inplace=True)
teste.rename(columns={"grnd_level": "Pressão Local"}, inplace=True)
teste.rename(columns={"clouds_all": "Nebulosidade"}, inplace=True)
teste.rename(columns={"evapotranspiration": "Evapotranspiração"}, inplace=True)
teste.rename(columns={"vapour_pressure_deficit (kPa)": "Déficit de Vapor"}, inplace=True)
teste.rename(columns={"wind_direction": "Direção Vento"}, inplace=True)
teste.rename(columns={"wind_gusts": "Rajadas Vento"}, inplace=True)
teste.rename(columns={"soil_temperature": "Temperatura Solo"}, inplace=True)
teste.rename(columns={"soil_moisture": "Humidade Solo"}, inplace=True)
teste.rename(columns={"is_day": "é_de_Dia"}, inplace=True)
teste.rename(columns={"sunshine_duration (s)": "Duração Solar"}, inplace=True)
teste.rename(columns={"direct_radiation_instant": "Radiação Solar"}, inplace=True)
teste.rename(columns={"direct_normal_irradiance_instant": "Irradiância Solar"}, inplace=True)
teste.rename(columns={"weather_description": "Estado do Tempo"}, inplace=True)
teste.rename(columns={"Normal (kWh)": "Normal"}, inplace=True)
teste.rename(columns={"Horário Económico (kWh)": "Horário Económico"}, inplace=True)
teste.rename(columns={"Autoconsumo (kWh)": "Autoconsumo"}, inplace=True)
teste.rename(columns={"Injeção na rede (kWh)": "Injeção na Rede"}, inplace=True)
```

Figura 26: Simplificação das features nos dados de treino e no dataset e_m

```python
e_m.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11016 entries, 0 to 11015
Data columns (total 28 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   dt                              11016 non-null  int64
 1   temp                            11016 non-null  float64
 2   humidity                        11016 non-null  int64
 3   dew_point                       11016 non-null  float64
 4   feels_like                      11016 non-null  float64
 5   rain_1h                         11016 non-null  float64
 6   sea_level                       11016 non-null  float64
 7   grnd_level                      11016 non-null  float64
 8   clouds_all                      11016 non-null  int64
 9   evapotranspiration              11016 non-null  float64
 10  vapour_pressure_deficit (kPa)   11016 non-null  float64
 11  wind_speed                      11016 non-null  float64
 12  wind_direction                  11016 non-null  int64
 13  wind_gusts                      11016 non-null  float64
 14  soil_temperature                11016 non-null  float64
 15  soil_moisture                   11016 non-null  float64
 16  is_day                          11016 non-null  int64
 17  sunshine_duration (s)           11016 non-null  float64
 18  direct_radiation_instant        11016 non-null  float64
 19  direct_normal_irradiance_instant  11016 non-null  float64
...
 26  Injeção na rede (kWh)           3239 non-null   object
 27  Data_Hora                       11016 non-null  object
dtypes: float64(18), int64(6), object(4)
memory usage: 2.4+ MB
```

```python
teste.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2256 entries, 0 to 2255
Data columns (total 27 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   dt                              2256 non-null   int64
 1   temp                            2256 non-null   float64
 2   humidity                        2256 non-null   int64
 3   dew_point                       2256 non-null   float64
 4   feels_like                      2256 non-null   float64
 5   rain_1h                         2256 non-null   float64
 6   sea_level                       2256 non-null   float64
 7   grnd_level                      2256 non-null   float64
 8   clouds_all                      2256 non-null   int64
 9   evapotranspiration              2256 non-null   float64
 10  vapour_pressure_deficit (kPa)   2256 non-null   float64
 11  wind_speed                      2256 non-null   float64
 12  wind_direction                  2256 non-null   int64
 13  wind_gusts                      2256 non-null   float64
 14  soil_temperature                2256 non-null   float64
 15  soil_moisture                   2256 non-null   float64
 16  is_day                          2256 non-null   int64
 17  sunshine_duration (s)           2256 non-null   float64
 18  direct_radiation_instant        2256 non-null   float64
 19  direct_normal_irradiance_instant  2256 non-null   float64
...
 25  Autoconsumo (kWh)               2256 non-null   float64
 26  Data_Hora                       2256 non-null   object
dtypes: float64(18), int64(6), object(3)
memory usage: 476.0+ KB
```

Figura 27: Informação referente ao conjunto de dados do novo dataset de teste e treino.

```
e_m.isnull().sum() # 7777 None na colu
```

```
dt                                    0
temp                                  0
humidity                              0
dew_point                             0
feels_like                            0
rain_1h                               0
sea_level                             0
grnd_level                            0
clouds_all                            0
evapotranspiration                    0
vapour_pressure_deficit (kPa)         0
wind_speed                            0
wind_direction                        0
wind_gusts                            0
soil_temperature                      0
soil_moisture                         0
is_day                                0
sunshine_duration (s)                 0
direct_radiation_instant              0
direct_normal_irradiance_instant      0
weather_description                   0
Data                                  0
Hora                                  0
Normal (kWh)                          0
Horário Económico (kWh)               0
Autoconsumo (kWh)                     0
Injeção na rede (kWh)              7777
Data_Hora                             0
dtype: int64
```

```
teste.isnull().sum()
```

```
dt                                    0
temp                                  0
humidity                              0
dew_point                             0
feels_like                            0
rain_1h                               0
sea_level                             0
grnd_level                            0
clouds_all                            0
evapotranspiration                    0
vapour_pressure_deficit (kPa)         0
wind_speed                            0
wind_direction                        0
wind_gusts                            0
soil_temperature                      0
soil_moisture                         0
is_day                                0
sunshine_duration (s)                 0
direct_radiation_instant              0
direct_normal_irradiance_instant      0
weather_description                   0
Data                                  0
Hora                                  0
Normal (kWh)                          0
Horário Económico (kWh)               0
Autoconsumo (kWh)                     0
Data_Hora                             0
dtype: int64
```

Figura 28: Missing values no novo dataset de teste e treino.

```
def get_outliers(lower,upper,field):
    Q1 = e_m[field].quantile(0.25)
    Q3 = e_m[field].quantile(0.75)
    IQR = Q3-Q1

    lower_lim = Q1 - lower * IQR
    upper_lim = Q3 + upper * IQR

    out_low = (e_m[field]<lower_lim)
    out_up = (e_m[field]>upper_lim)

    return [out_low,out_up]
```

Figura 29: Função outliers.

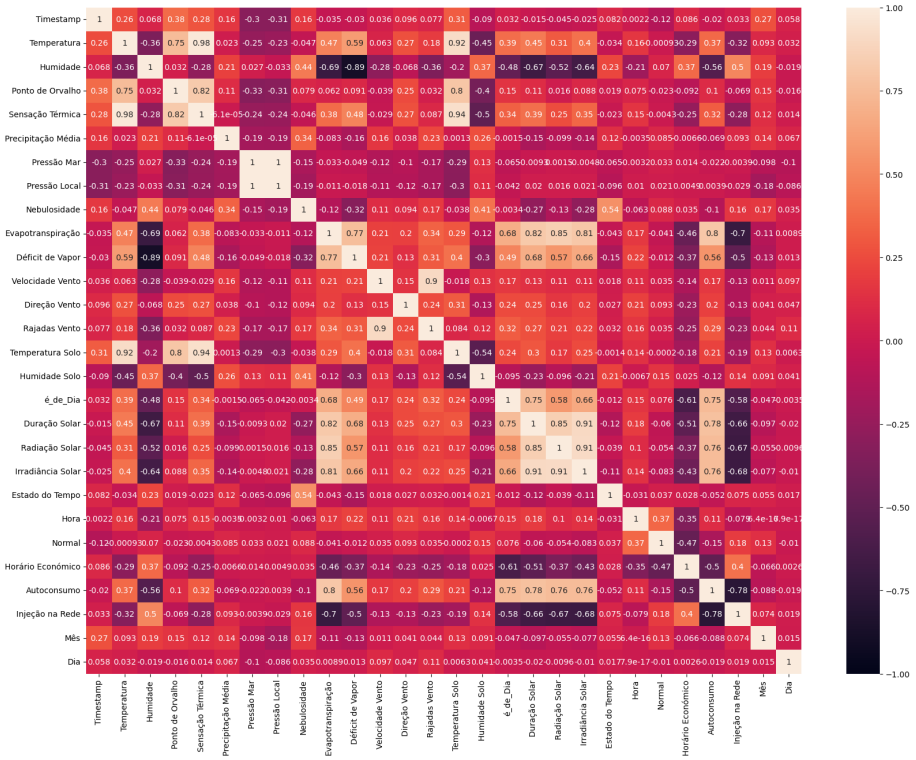

Figura 30: Matriz de correlação do novo dataset.

```
param_grid = {
    'learning_rate': [0.01],
    'n_estimators': [1000],
    'max_depth': [4],
    'min_child_weight': [5],
    'subsample': [0.6],
    'colsample_bytree': [0.5],
    'gamma': [0.1],
    'lambda': [1.5],
    'alpha': [0.5]
}
```

Figura 31: Hiperpârametros XGBoost.

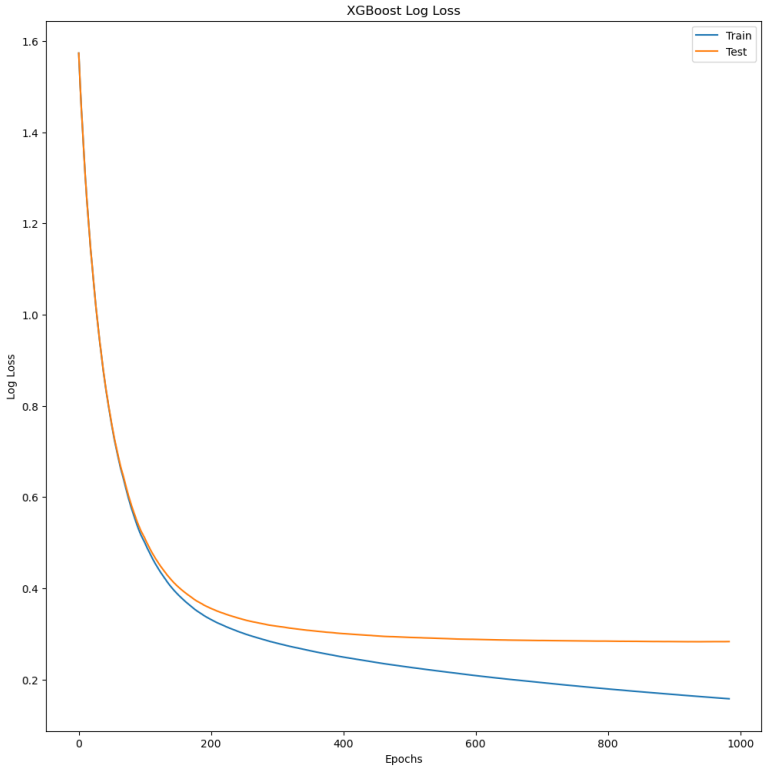

Figura 32: Gráfico para análise de Overfit.

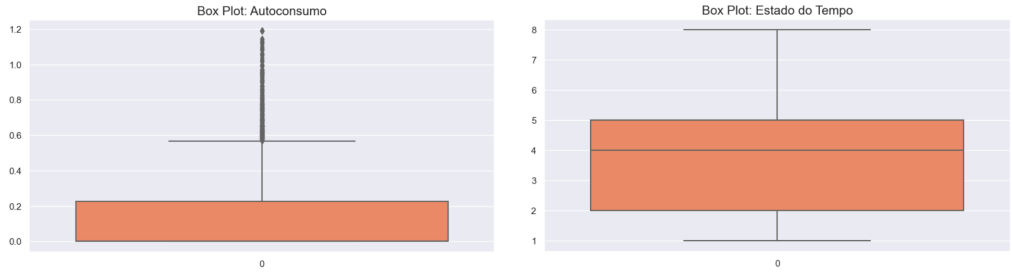

Figura 33: Box plot do Autoconsumo e do Estado do Tempo.

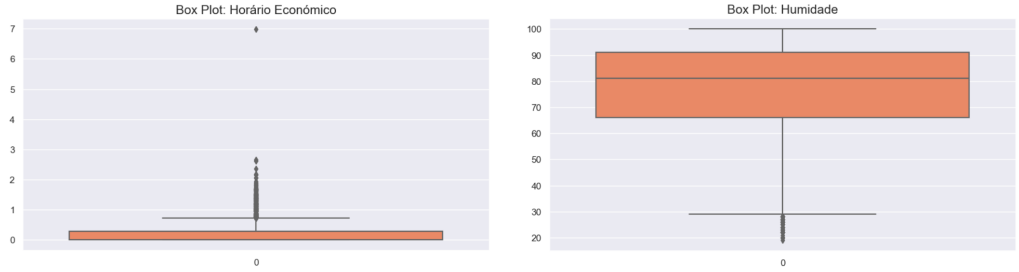

Figura 34: Box plot do Horário Económico e da Humidade.

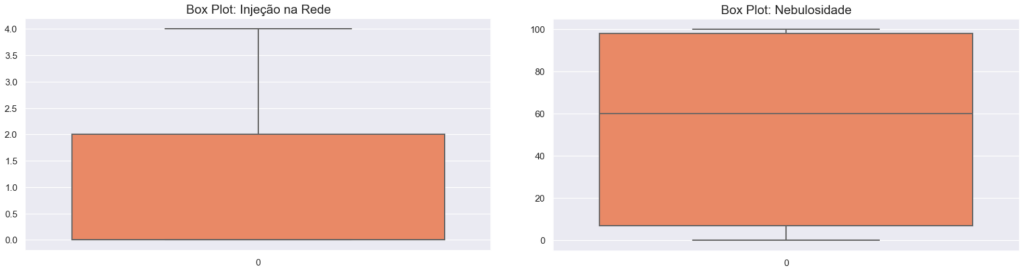

Figura 35: Box plot da Injeção de Rede e da Nebulosidade.

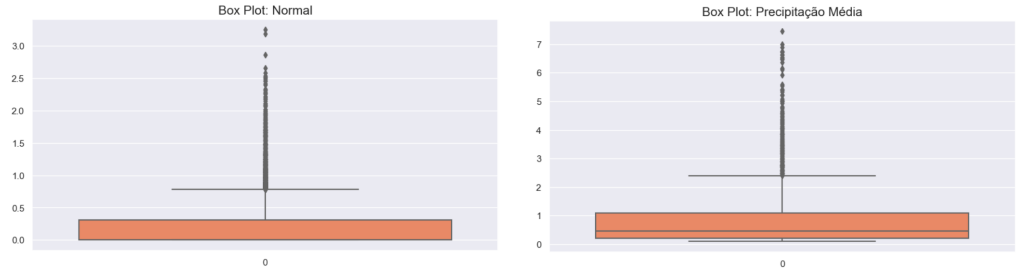

Figura 36: Box plot da Normal e da Precipitação Média.

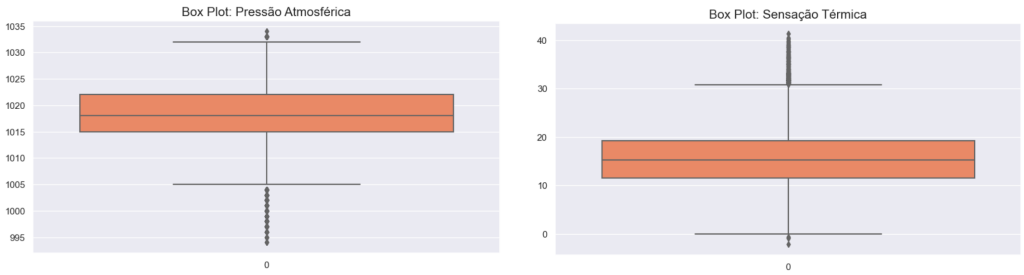

Figura 37: Box plot da Pressão Atmosférica e da Humidade.

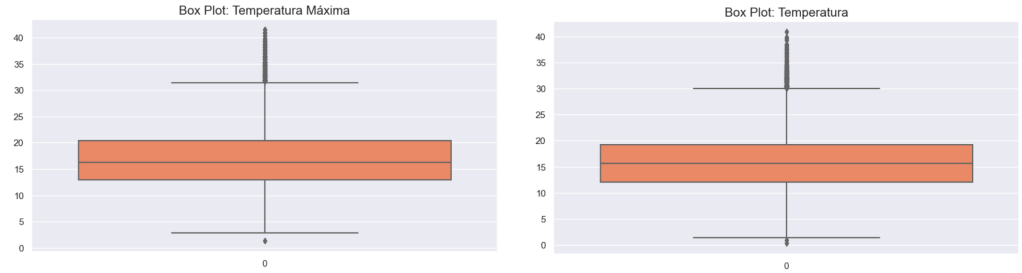

Figura 38: Box plot da Temperatura Máxima e da Temperatura.

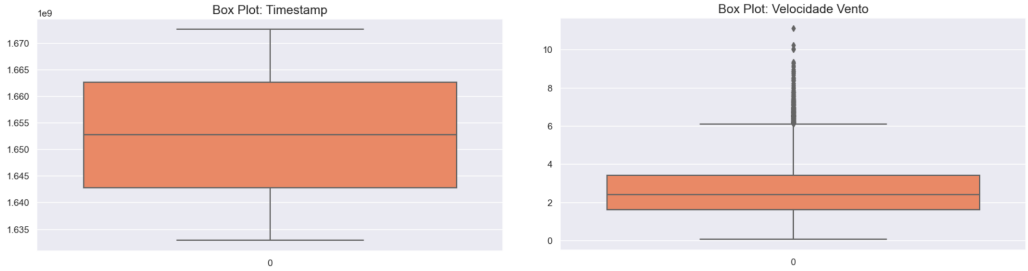

Figura 39: Box plot do Timestamp e da Velocidade do Vento.

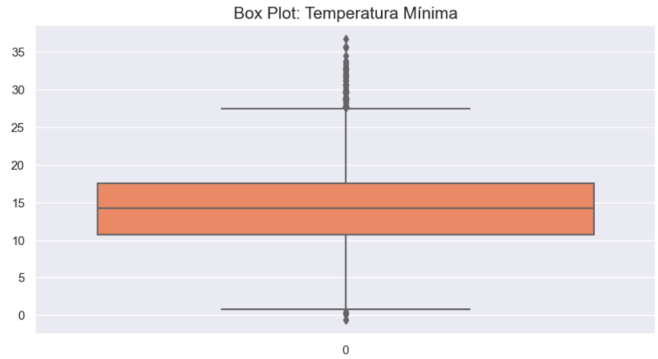

Figura 40: Box plot Temperatura Mínima.