University of Minho
School of Engineering

DADOS E APRENDIZAGEM AUTOMÁTICA

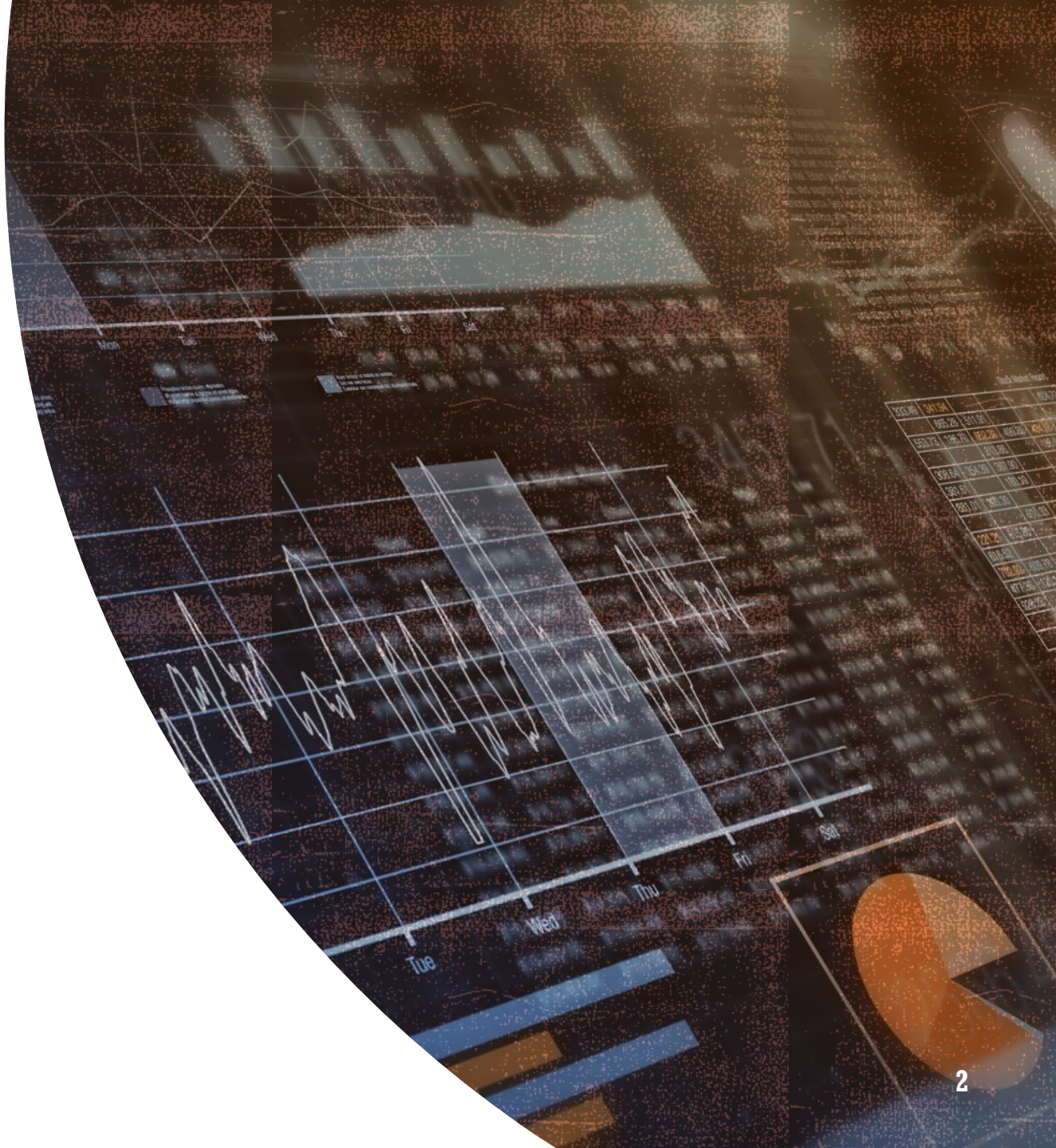
Grupo 5

- João Abreu (pg53928)
- João Faria (pg53939)
- Ricardo Sousa (pg54179)
- Rui Silva (pg54213)

PREVISÃO DA QUANTIDADE DE ENERGIA INJETADA NA REDE

Features dos *Datasets* de Energia:

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



FEATURES DOS DATASETS METEOROLÓGICOS

- **dt** - o *timestamp* associado ao registo;
- **dt_iso** - a data associada ao registo, ao segundo;
- **city_name** - o local em causa;
- **temp** - temperatura em ºC;
- **feels_like** - sensação térmica em ºC;
- **temp_min** - temperatura mínima sentida em ºC;
- **temp_max** - temperatura máxima sentida em ºC;
- **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.

TRATAMENTO PRÉVIO DOS DADOS

Concat dos dois datasets para treino da energia

```
df2021 = pd.read_csv('./datasets/competicao/energia_202109-202112.csv')
df2022 = pd.read_csv('./datasets/competicao/energia_202201-202212.csv')

df_energia_train = pd.concat([df2021, df2022], ignore_index=True)
df_energia_teste = pd.read_csv('./datasets/competicao/energia_202301-202304.csv')
```

Concat dos dois datasets para treino da meteorologia

```
dfm2021 = pd.read_csv('./datasets/competicao/meteo_202109-202112.csv')
dfm2022 = pd.read_csv('./datasets/competicao/meteo_202201-202212.csv')

df_meteo_train = pd.concat([dfm2021, dfm2022], ignore_index=True)
df_meteo_teste = pd.read_csv('./datasets/competicao/meteo_202301-202304.csv')
```

Adicionar uma coluna com o timestamp em segundos

```
df_energia_train['dt'] = pd.to_datetime(df_energia_train['Data'] + ' ' + df_energia_train['Hora'].astype(str) + ':00:00', format='%Y-%m-%d %H:%M:%S')
df_energia_teste['dt'] = pd.to_datetime(df_energia_teste['Data'] + ' ' + df_energia_teste['Hora'].astype(str) + ':00:00', format='%Y-%m-%d %H:%M:%S')

def create_timestamp(row):
    timestamp_str = str(row['dt'])
    timestamp= datetime.strptime(timestamp_str, '%Y-%m-%d %H:%M:%S')
    timestamp_utc = timestamp.replace(tzinfo=timezone.utc)
    return int(timestamp_utc.timestamp())

df_energia_train['dt'] = df_energia_train.apply(create_timestamp, axis=1)
df_energia_teste['dt'] = df_energia_teste.apply(create_timestamp, axis=1)
```

Join do dataset de energia com o de meteorologia

```
df_train = pd.merge(df_energia_train, df_meteo_train, left_on='dt', right_on='dt', how='inner')

df_teste = pd.merge(df_energia_teste, df_meteo_teste, left_on='dt', right_on='dt', how='left')
```

- Uma vez que no *dataset* de teste, para as condições meteorológicas, não continha dados referentes a todos os dias do ano de 2023, foi necessário obter estes dados a partir de fontes externas.
- A recolha de dados foi realizada a partir do website: https://www.visualcrossing.com/

COMPREENSÃO DOS DADOS

```
df_train.shape
```

```
(11016, 21)
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11016 entries, 0 to 11015
Data columns (total 21 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 
 6   dt                       11016 non-null  int64  
 7   dt_iso                   11016 non-null  object 
 8   city_name                11016 non-null  object 
 9   temp                     11016 non-null  float64
 10  feels_like               11016 non-null  float64
 11  temp_min                 11016 non-null  float64
 12  temp_max                 11016 non-null  float64
 13  pressure                 11016 non-null  int64  
 14  sea_level                0 non-null      float64
 15  grnd_level               0 non-null      float64
 16  humidity                 11016 non-null  int64  
 17  wind_speed               11016 non-null  float64
 18  rain_1h                  2284 non-null   float64
 19  clouds_all               11016 non-null  int64  
 20  weather_description      11016 non-null  object 
dtypes: float64(11), int64(5), object(5)
memory usage: 1.8+ MB
```

```
print(df_train.duplicated().sum())
```

```
0
```

```
print(df_train.isna().sum())
```

```
Data                           0
Hora                           0
Normal (kWh)                   0
Horário Económico (kWh)        0
Autoconsumo (kWh)              0
Injeção na rede (kWh)       7777
dt                             0
dt_iso                         0
city_name                      0
temp                           0
feels_like                     0
temp_min                       0
temp_max                       0
pressure                       0
sea_level                  11016
grnd_level                 11016
humidity                       0
wind_speed                     0
rain_1h                     8732
clouds_all                     0
weather_description            0
dtype: int64
```

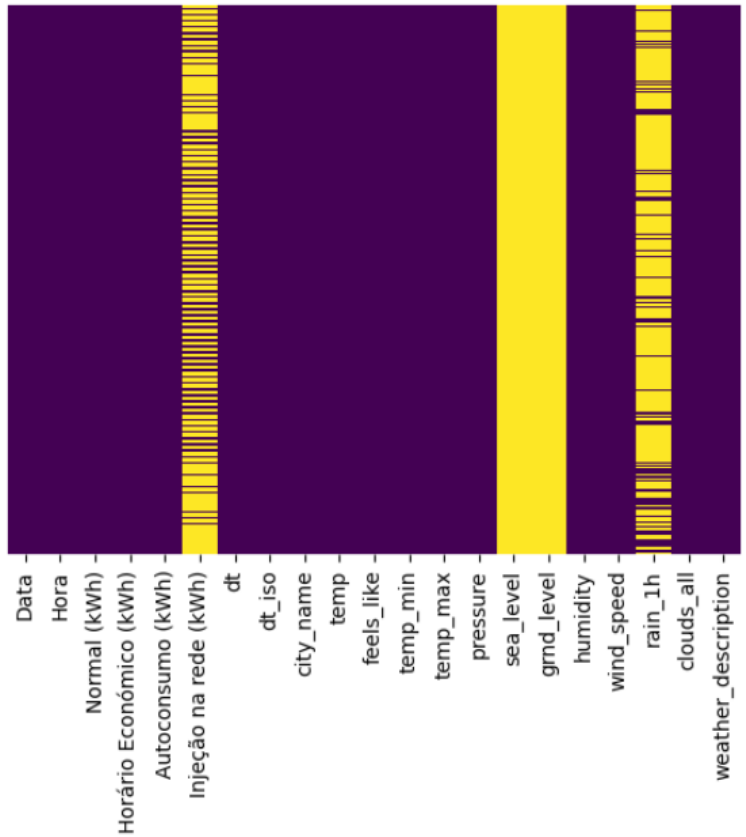
PREPARAÇÃO E EXPLORAÇÃO DE DADOS

1. Missing values

Verificar *missing values* em ambos os *dataframes*

```
sns.heatmap(df_train.isnull(),yticklabels=False, cbar=False, cmap='viridis')
```

```
<Axes: >
```



- Para a *feature* **Injeção na rede (kWh)** e **rain_1h** o grupo considerou que a falta de valores nas colunas deve-se ao facto de não ter havido injeção na rede ou chuva, preenchendo com o valor 0.
- Para as *features* sea level, grnd level, uma vez que toda a coluna apresentava *missing values* decidimos remover ambas as colunas.

```
df_train.drop(['sea_level'], axis=1, inplace=True)
df_train.drop(['grnd_level'], axis=1, inplace=True)
df_train.drop(['dt'], axis=1, inplace=True)
df_train.drop(['dt_iso'], axis=1, inplace=True)
df_train.head()
```

- Removemos a *feature* **city_name** por ser um valor único para todas as linhas.

```
df_train.drop('city_name',axis=1,inplace=True)
```

```
df_train['Data'].head()
```

```
0    2021-09-29
1    2021-09-29
2    2021-09-29
3    2021-09-29
4    2021-09-29
Name: Data, dtype: object
```

```
df_train['Data'] = pd.to_datetime(df_train['Data'], format='%Y-%m-%d', errors='coe
```

```
df_teste_v1['Data'] = pd.to_datetime(df_teste_v1['Data'], format='%Y-%m-%d', error
```

```
df_train['date_year'] = df_train['Data'].dt.year
df_train['date_month'] = df_train['Data'].dt.month
df_train['date_day'] = df_train['Data'].dt.day
df_train['date_hour'] = df_train['Hora']
```

```
df_teste_v1['date_year'] = df_teste_v1['Data'].dt.year
df_teste_v1['date_month'] = df_teste_v1['Data'].dt.month
df_teste_v1['date_day'] = df_teste_v1['Data'].dt.day
df_teste_v1['date_hour'] = df_teste_v1['Hora']
```

```
df_train.drop('Data', axis=1, inplace=True)
df_train.drop('Hora', axis=1, inplace=True)
```

```
df_teste_v1.drop('Data', axis=1, inplace=True)
df_teste_v1.drop('Hora', axis=1, inplace=True)
```

PREPARAÇÃO E EXPLORAÇÃO DE DADOS

2. Tratamento de datas

- A *feature* **data** foi dividida em várias colunas, de forma a representar informação mais relevante para o nosso modelo.
- Remoção das colunas **dt** e **dt_iso** devido ao facto de apresentarem um valor único para cada alínea.

PREPARAÇÃO E EXPLORAÇÃO DE DADOS

3. Dados categóricos

Para as *features* **weather description** e **Injeção na rede (kWh),** visto que ambas as colunas têm uma certa ordem (categóricos ordinais), os valores de ambas as colunas foram substituídos por números inteiros relacionados com a sua ordem prévia.

```
replace_map = {'weather_description': {
'sky is clear': 1,
'few clouds': 2,
'scattered clouds': 3,
'broken clouds': 4,
'overcast clouds': 5,
'light rain': 6,
'moderate rain': 7,
'heavy intensity rain': 8}}
```

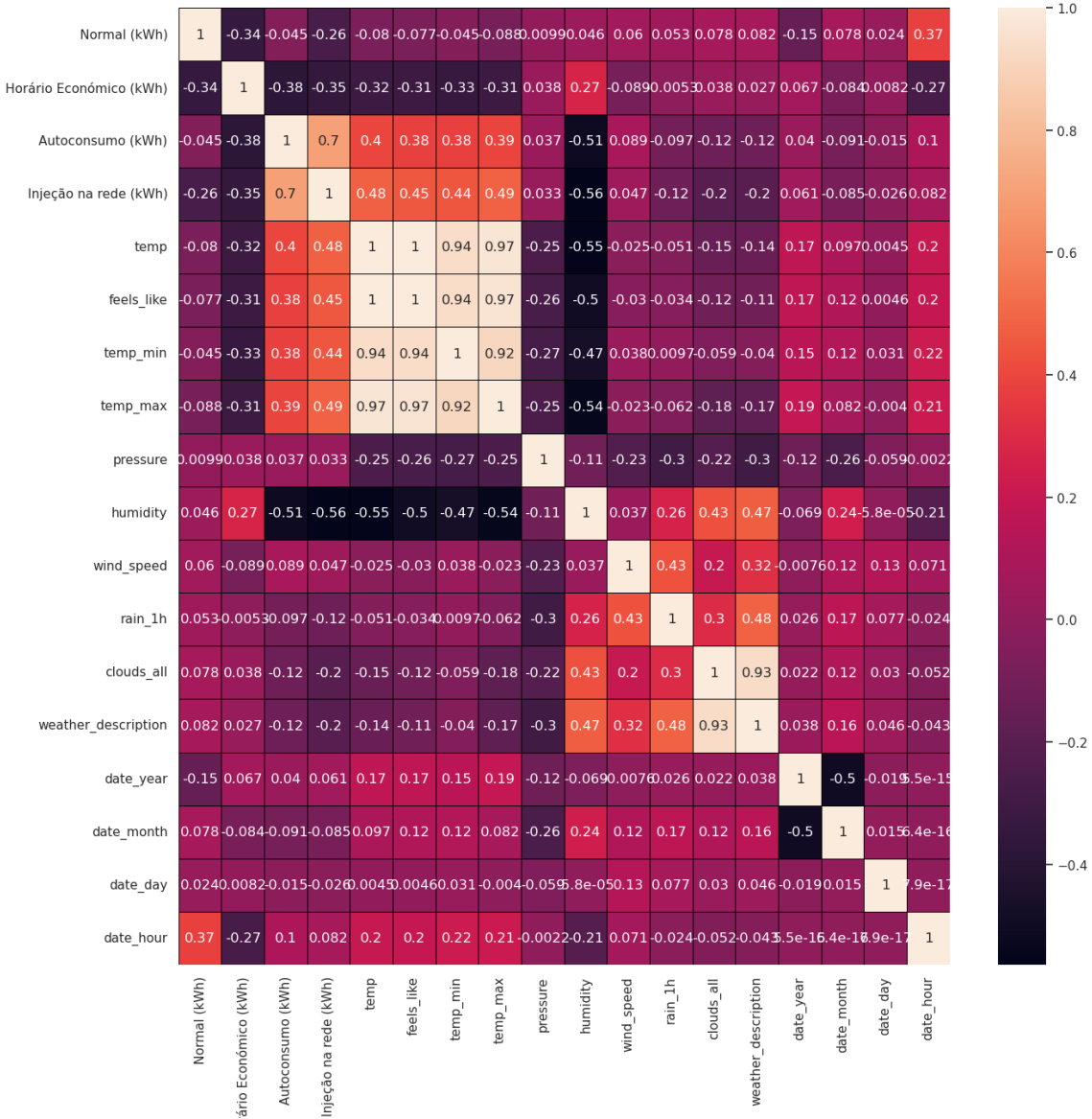
```
df_train.replace(replace_map, inplace=True)
```

```
df_teste_v1.replace(replace_map, inplace=True)
```

```
replace_map = {'Injeção na rede (kWh)': {
    'None': 0,
    'Low': 1,
    'Medium': 2,
    'High': 3,
    'Very High': 4}}
```

```
df_train.replace(replace_map, inplace=True)
```

```
df_train['Injeção na rede (kWh)'].fillna(0, inplace=True)
```

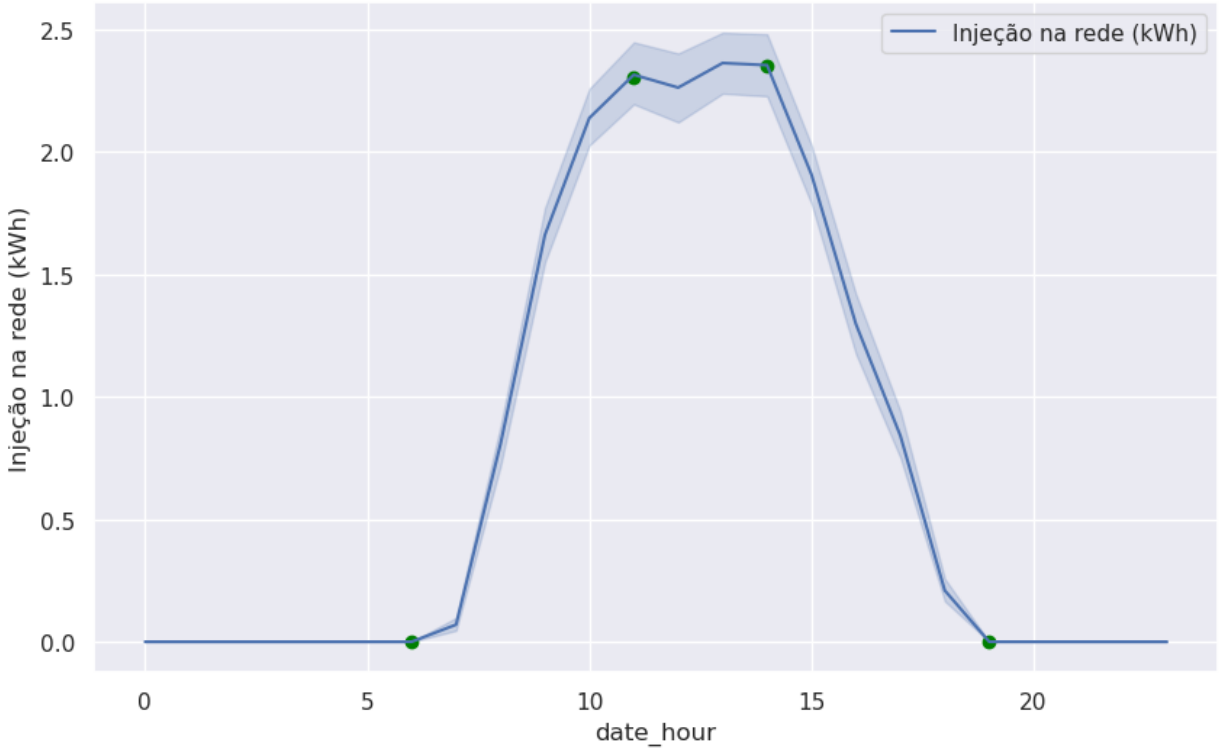


ANÁLISE EXPLORATÓRIA DE DADOS

- Podemos observar que os atributos **temp_min, temp_max** e **feels_like** têm uma grande correlação com a *feature* **temp**, e devido a tal decidimos remover estas três *features.*
- Entretanto, notamos que os atributos **date_day** e **date_year** seguintes têm uma correlação próxima de zero com os demais atributos, devido a tal optamos por removê-los.
- Já os atributos **weather desciption** e **clouds_all** apresentam valores muito semelhantes. Decidimos, assim, remover a *feature* **weather_description**, uma vez que é uma *feature* que sofreu da adição de valores externos e poderia influenciar negativamente os resultados do nosso modelo.

ANÁLISE EXPLORATÓRIA DE DADOS

Através do *feature engineering* que realizamos à hora com a nossa variável target, o grupo decidiu criar um novo atributo que representasse a fase do dia vez que a injeção na rede varia consoante a hora do dia. Esta nova coluna foi criada analisando o seguinte gráfico:




```python
def fase_do_dia(hour):

    if hour<=6 and hour>=19:
        return 1

    elif hour<=11 and hour>=6:
        return 2

    elif hour<=14 and hour>=11:
        return 3
    else:
        return 4
```

```python
df_train['day_fase']=df_train['date_hour'].apply(fase_do_dia)
```

```python
df_teste_v1['day_fase']=df_teste_v1['date_hour'].apply(fase_do_dia)
```

DADOS DE TREINO E TESTE

```
X = df_train.drop(['Injeção na rede (kWh)'], axis=1)
y = df_train['Injeção na rede (kWh)']
```

```
X_train_p, X_test_p, y_train_p, y_test_p = train_test_split(X, y, test_size=0.2, random_state=2023)
```

Decision Tree

```
dt_model.fit(X_train, y_train)
```

```
dt_predictions = dt_model.predict(X_teste)
```

```
param_grid_dt={'criterion':['gini','entropy','log_loss'],'max_depth':[5,6,7,8],'min_samples_split': [2,3,4],'min_samples_leaf': [1,2,3, 4]}
estimator_dt=DecisionTreeClassifier(random_state=2021)
grid_dt=GridSearchCV(estimator_dt,param_grid_dt,refit=True,verbose=0)
```

```
grid_dt.fit(X_train_p,y_train_p)
```

```
GridSearchCV(estimator=DecisionTreeClassifier(random_state=2021),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [5, 6, 7, 8],
                         'min_samples_leaf': [1, 2, 3, 4],
                         'min_samples_split': [2, 3, 4]})
```

```
previsao_dt = dt_model.predict(X_test_p)
```

```
accuracy_score(y_test_p,previsao_dt)
```

```
0.8666061705989111
```

```
print(classification_report(y_test_p,previsao_dt))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.98      0.97      1568
         1.0       0.40      0.19      0.26        91
         2.0       0.61      0.66      0.63       214
         3.0       0.63      0.67      0.65       215
         4.0       0.68      0.59      0.63       116

    accuracy                           0.87      2204
   macro avg       0.66      0.62      0.63      2204
weighted avg       0.86      0.87      0.86      2204
```

DADOS DE TREINO E TESTE

Random Forest

```
rf_model.fit(X_train, y_train)
```

```
rf_predictions = rf_model.predict(X_teste)
```

```
rf_model = grid_dt2.best_estimator_
```

```
previsao_rf = rf_model.predict(X_test_p)
```

```
accuracy_score(y_test_p,previsao_rf)
```

```
0.8861161524500908
```

```
param_grid_dt2={'n_estimators':[150,200,250],'max_depth':[7,8,9],'min_samples_split': [2,3,4],'min_samples_leaf': [1,2,3]}
estimator_dt2=RandomForestClassifier(bootstrap=False ,random_state=2022)
grid_dt2=GridSearchCV(estimator_dt2,param_grid_dt2,refit=True,verbose=0)
```

```
print(classification_report(y_test_p,previsao_rf))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.97      1568
         1.0       0.59      0.11      0.19        91
         2.0       0.65      0.71      0.68       214
         3.0       0.67      0.77      0.72       215
         4.0       0.79      0.68      0.73       116

    accuracy                           0.89      2204
   macro avg       0.73      0.65      0.66      2204
weighted avg       0.88      0.89      0.88      2204
```

SVM

```
svm_model.fit(X_train, y_train)
```

```
svm_predictions = svm_model.predict(X_teste)
```

```
param_grid={'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001],'kernel':['rbf','linear']}
```

```
grid_svc = GridSearchCV(SVC(random_state=2023),param_grid,refit=True,verbose=0)
```

```
grid_svc.fit(X_train_p,y_train_p)
```

```
GridSearchCV(estimator=SVC(random_state=2023),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'linear']})
```

```
accuracy_score(y_test_p,previsao_svm)
```

```
0.8566243194192378
```

```
svm_model = grid_svc.best_estimator_
```

```
previsao_svm = svm_model.predict(X_test_p)
```

```
print(classification_report(y_test_p,previsao_svm))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97      1568
         1.0       0.25      0.14      0.18        91
         2.0       0.57      0.59      0.58       214
         3.0       0.64      0.64      0.64       215
         4.0       0.72      0.72      0.72       116

    accuracy                           0.86      2204
   macro avg       0.63      0.61      0.62      2204
weighted avg       0.85      0.86      0.85      2204
```

DADOS DE TREINO E TESTE

XGBoost

```
xgb_model.fit(X_train, y_train)
```

```
xgb_predictions = xgb_model.predict(X_teste)
```

```
param_xgb={'n_estimators':[200,225,250],'learning_rate':[0.05,0.1,0.2],'max_depth':[3,4,5]}
estimator_xgb=XGBClassifier()
grid_xgb=GridSearchCV(estimator_xgb,param_xgb,scoring='accuracy',refit=True,verbose=0)
```

```
grid_xgb.fit(X_train_p,y_train_p)
```

```
GridSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, device=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     grow_policy=None, importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max_b...
                                     max_cat_threshold=None,
                                     max_cat_to_onehot=None,
                                     max_delta_step=None, max_depth=None,
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     multi_strategy=None, n_estimators=None,
                                     n_jobs=None, num_parallel_tree=None,
                                     random_state=None, ...),
             param_grid={'learning_rate': [0.05, 0.1, 0.2],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [200, 225, 250]},
             scoring='accuracy')
```

```
xgb_score = grid_xgb.score(X_test_p,y_test_p)
```

```
print("Accuracy: %.2f%%" % (xgb_score * 100))
```

```
Accuracy: 89.11%
```

```
xgb_model = grid_xgb.best_estimator_
xgb_predictions = xgb_model.predict(X_test_p)
```

```
print(classification_report(y_test_p,xgb_predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.98      0.98      1568
         1.0       0.40      0.19      0.26        91
         2.0       0.66      0.64      0.65       214
         3.0       0.72      0.77      0.74       215
         4.0       0.77      0.86      0.81       116

    accuracy                           0.89      2204
   macro avg       0.70      0.69      0.69      2204
weighted avg       0.88      0.89      0.88      2204
```

DADOS DE TREINO E TESTE

Stacking

```
st_model.fit(X_train, y_train)
```

```
estimators = [("dt", dt_model), ("svm", svm_model), ("rf", rf_model),("xgb",xgb_model)]
```

```
st_model = StackingClassifier(estimators=estimators,final_estimator=LogisticRegression(max_iter=1000))
```

```
st_model.fit(X_train_p,y_train_p)
```

```
StackingClassifier(estimators=[('dt',
                                DecisionTreeClassifier(max_depth=6,
                                                       min_samples_leaf=2,
                                                       random_state=2021)),
                               ('svm',
                                SVC(C=1000, gamma=0.001, random_state=2023)),
                               ('rf',
                                RandomForestClassifier(bootstrap=False,
                                                       max_depth=9,
                                                       min_samples_split=3,
                                                       n_estimators=150,
                                                       random_state=2022)),
                               ('xgb',
                                XGBClassifier(base_score=None, booster=None,
                                              callbacks=None,
                                              colsample_...
                                              interaction_constraints=None,
                                              learning_rate=0.05, max_bin=None,
                                              max_cat_threshold=None,
                                              max_cat_to_onehot=None,
                                              max_delta_step=None, max_depth=5,
                                              max_leaves=None,
                                              min_child_weight=None,
                                              missing=nan,
                                              monotone_constraints=None,
                                              multi_strategy=None,
                                              n_estimators=225, n_jobs=None,
                                              num_parallel_tree=None,
                                              objective='multi:softprob', ...))],
                   final_estimator=LogisticRegression(max_iter=1000))
```

```
print("Accuracy: %.2f%%" % (st_score * 100))
```

```
Accuracy: 88.97%
```

DESCRIÇÃO DO PROBLEMA

- No cenário acelerado do trabalho moderno, o burnout tem vindo a ganhar destaque como um grande desafio à saúde mental e o bem-estar dos trabalhadores.
- Esse estado de exaustão emocional e mental, alimentado por pressões constantes, afeta o desempenho dos trabalhadores. Deste modo, com este dataset, o nosso grupo têm como objetivo, numa perspetiva empresarial, determinar o nível de burnout, a partir de outros fatores, para assim tirar o melhor desempenho dos seus funcionários.

PREVISÃO DE *BURNOUT* EM TRABALHADORES

Problema: Prever o *burnout* em trabalhadores.

***Features* do *dataset*:**

- **Employee ID** - Identificador do funcionário (employee).
- **Date of Joining** - Data em que o funcionário começou a trabalhar.
- **Gender** - Género do funcionário.
- **Company Type** - Tipo de empresa do funcionário.
- **WFH Setup Available** - Indica se existem sistemas que permitem o trabalho a partir de casa.
- **Designation** - Escalão do funcionário.
- **Resource Allocation** - Número de horas de trabalho por dia.
- **Mental Fatigue Score** - Nível de fadiga mental do funcionário.
- **Burn Rate** - Taxa de *burnout* de um funcionário.

COMPREENSÃO DOS DADOS

```
df = pd.read_excel('./datasets/grupo/Burnout.xlsx')
df
```

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2	3.0	3.8	0.16
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1	2.0	5.0	0.36
2	fffe31003300320037003900	2008-03-10	Female	Product	Yes	2	NaN	5.8	0.49
3	fffe32003400380032003900	2008-11-03	Male	Service	Yes	1	1.0	2.6	0.20
4	fffe31003900340031003600	2008-07-24	Female	Service	No	3	7.0	6.9	0.52
...	...	...	...	...	...	...	...	...	...
22745	fffe31003500370039003100	2008-12-30	Female	Service	No	1	3.0	NaN	0.41
22746	fffe33003000350031003800	2008-01-19	Female	Product	Yes	3	6.0	6.7	0.59
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3	7.0	NaN	0.72
22748	fffe33003300320036003900	2008-01-10	Female	Service	No	2	5.0	5.9	0.52
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3	6.0	7.8	0.61

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Employee ID           22750 non-null  object
 1   Date of Joining       22750 non-null  datetime64[ns]
 2   Gender                22750 non-null  object
 3   Company Type          22750 non-null  object
 4   WFH Setup Available   22750 non-null  object
 5   Designation           22750 non-null  int64
 6   Resource Allocation   21369 non-null  float64
 7   Mental Fatigue Score  20633 non-null  float64
 8   Burn Rate             21626 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(4)
memory usage: 1.6+ MB
```

```
df.isnull().sum()
```

```
Employee ID              0
Date of Joining          0
Gender                   0
Company Type             0
WFH Setup Available      0
Designation              0
Resource Allocation     1381
Mental Fatigue Score    2117
Burn Rate               1124
dtype: int64
```

Podemos verificar que este *dataset* contêm 22750 entradas e 9 atributos.

```
df.shape
```

```
(22750, 9)
```

O *dataset* possui 4 atributos numéricos.

É possível identificar a presença de valores nulos em:

- Resource Allocation
- Mental Fatigue Score
- Burn Rate

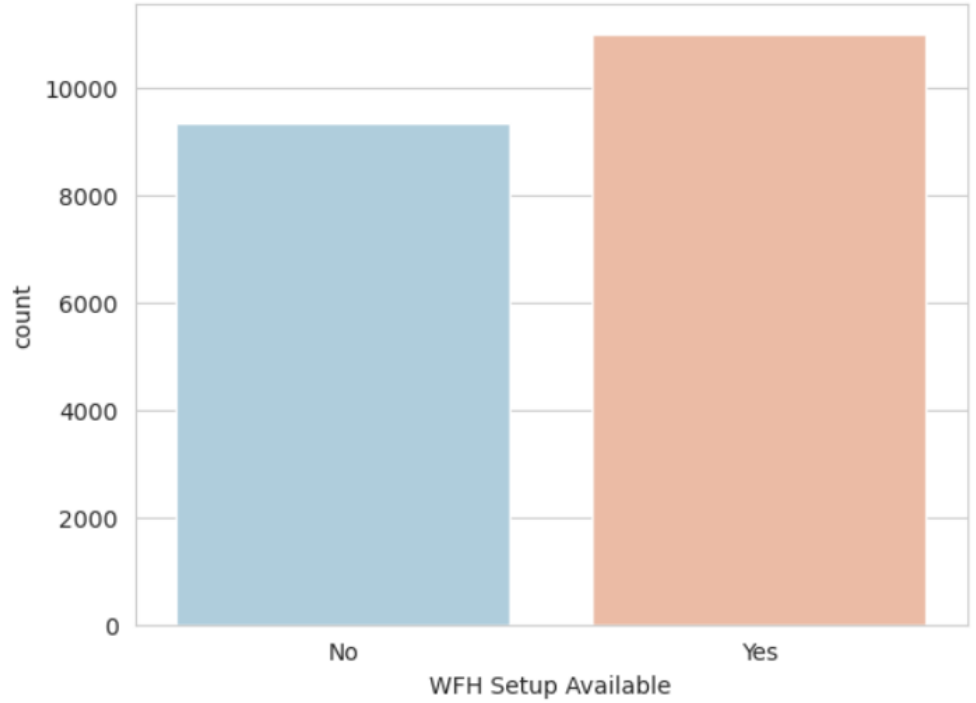
- O objetivo de trabalhar com este *dataset* é identificar o *burn rate*, sendo a label um valor continuo compreendido entre 0 e 1.

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

- Verificamos que a maioria dos trabalhadores do *dataset* trabalha a partir de casa, prática mais recorrente a partir da época da pandemia COVID-19.

```
sns.set_style('whitegrid')
sns.countplot(x='WFH Setup Available',data=df,palette='RdBu_r')
```

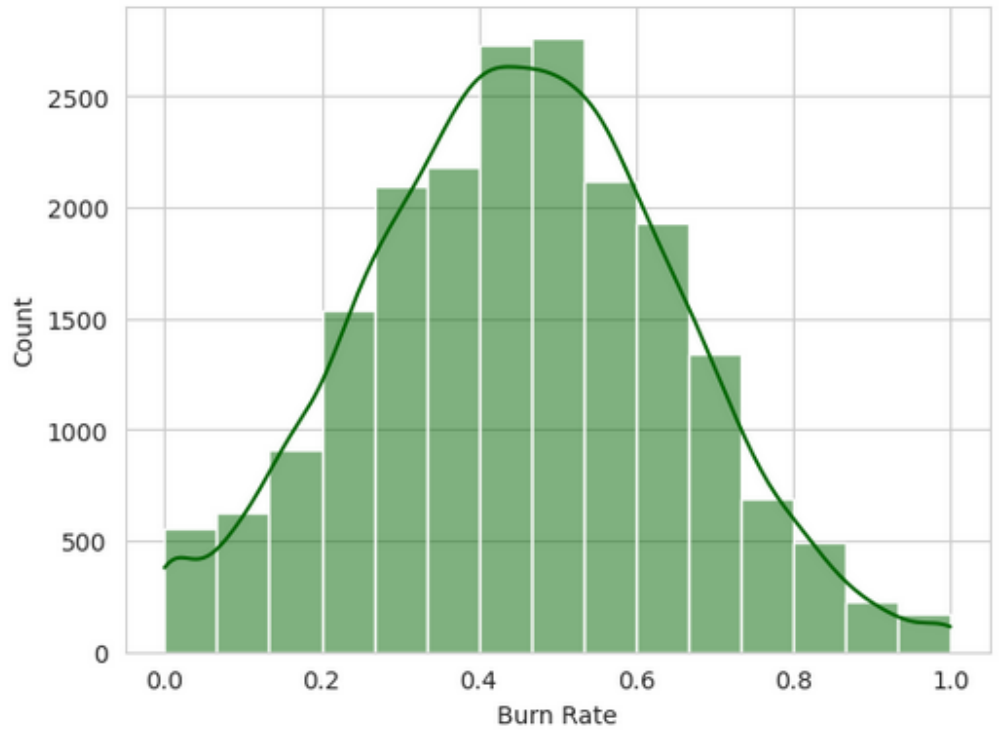
```
<Axes: xlabel='WFH Setup Available', ylabel='count'>
```



- Através do seguinte histograma podemos observar que a nossa variável target segue uma distribuição normal.

```
sns.histplot(df_v1['Burn Rate'],bins=15,color='darkgreen',kde=True)
```

```
<Axes: xlabel='Burn Rate', ylabel='Count'>
```

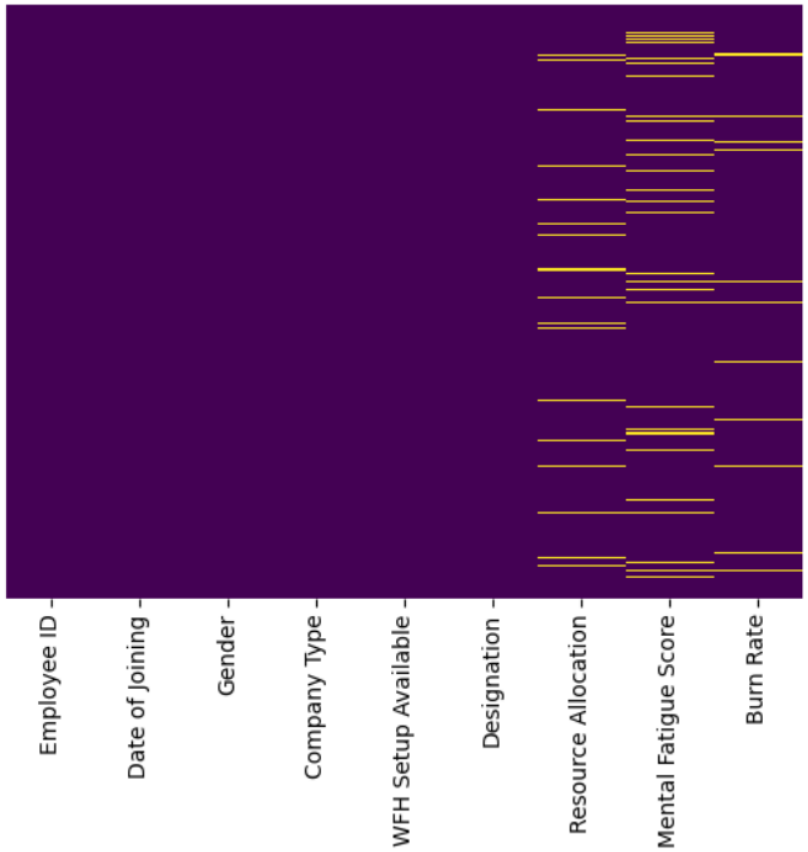


EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

1. Missing values

Começamos o nosso processo de preparação de dados pela exploração dos *missing values* do nosso *dataset*. Para obter uma visualização gráfica dos *missing values* utilizamos o seguinte *heatmap*.

```
sns.heatmap(df.isnull(),yticklabels=False, cbar=False, cmap='viridis')
```




```
def fill_with_zeros(data,columns):
    data = df

    data[columns] = data[columns].fillna(0)

    return data
```

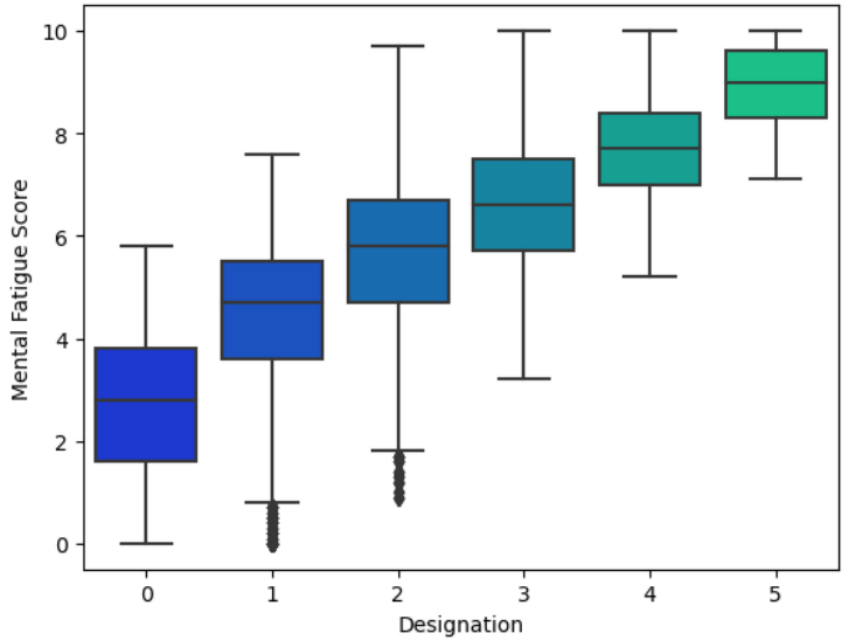
```
def remove_lines(data,columns):
    data = df

    data.dropna(subset=columns,inplace=True)

    return data
```

```
def fill_with_mean(data,columns):
    data = df

    data[columns] = data[columns].fillna(data[columns].mean())

    return data
```

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

```
sns.boxplot(x='Designation',y='Mental Fatigue Score',data=df,palette='winter')
```

```
<Axes: xlabel='Designation', ylabel='Mental Fatigue Score'>
```




```
medias = df.groupby('Designation')['Mental Fatigue Score'].mean()
print(medias)
```

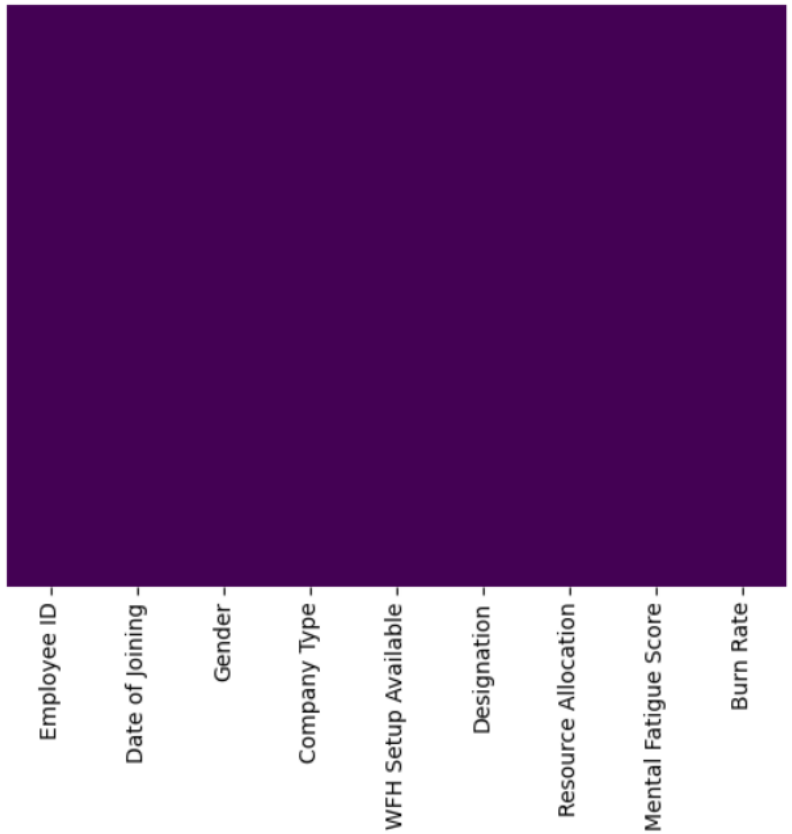
```
Designation
0    2.637157
1    4.443171
2    5.672385
3    6.623577
4    7.735645
5    8.929022
Name: Mental Fatigue Score, dtype: float64
```

```
def impute_mental(cols):
    Mental_Fatigue_Score = cols[0]
    Designation = cols[1]

    if pd.isnull(Mental_Fatigue_Score):

        if Designation == 1:
            return 4.564954

        elif Designation == 2:
            return 5.673261

        elif Designation == 3:
            return 6.541166

        elif Designation == 4:
            return 7.550271

        elif Designation == 5:
            return 8.591660

        else:
            return 2.924587

    else:
        return Mental_Fatigue_Score
```

```
df['Mental Fatigue Score'] = df[['Mental Fatigue Score','Designation']].apply(impute_mental,axis=1)
```

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

```
sns.heatmap(df.isnull(),yticklabels=False, cbar=False, cmap='viridis')
```



2. Duplicados

O *dataset* não possui valores duplicados, portanto não foi necessário fazer nada a este respeito.

```
df.duplicated().sum()
```

```
0
```

3. Drop

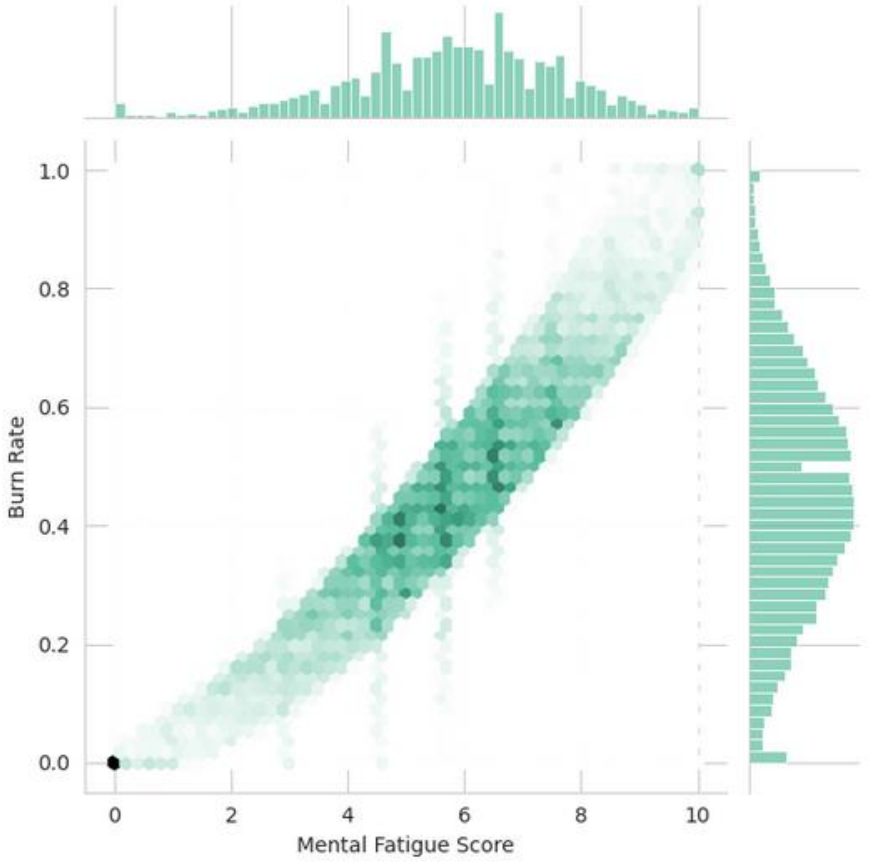
Uma vez que a coluna **Employee ID** possui um valor diferente para todos os indivíduos decidimos removê-la, uma vez que em nada contribui para o treino do modelo.

```
df.drop(['Employee ID'], axis = 1, inplace = True)
```

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

- A relação que mais chamou atenção foi entre a label *Burn rate* e a *feature Mental Fatigue Score*.

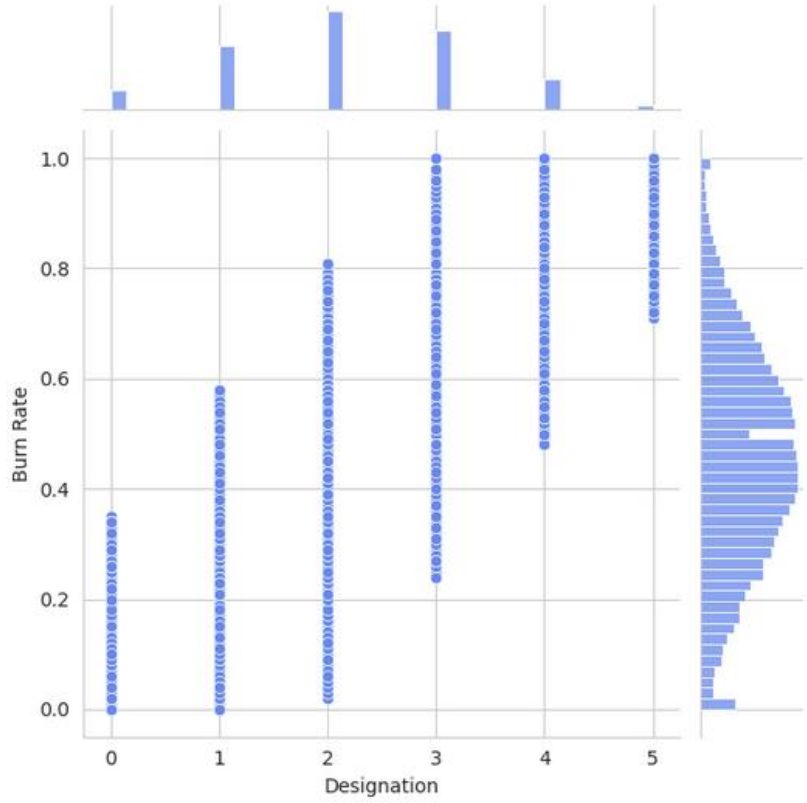
```
sns.set_style('whitegrid')
sns.set_palette('Set2')
sns.jointplot(x='Mental Fatigue Score',y='Burn Rate',data=df_v1,kind="hex")
```



- Uma outra relação identificada tanto no *jointplot* quanto na matriz de correlação é aquela existente entre a variável alvo (*Burn Rate*) e a *feature* de *Designation*, indicando uma possível associação ou influência mútua entre elas.

```
sns.set_style('whitegrid')
sns.set_palette('coolwarm')
sns.jointplot(x='Designation',y='Burn Rate',data=df_v1)
```

```
<seaborn.axisgrid.JointGrid at 0x7f8e44a4bf40>
```



EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

4. Dados Categóricos

- Em seguida, começamos por tratar os nossos dados categóricos, de todas as técnicas de *encoding* decidimos utilizar a técnica de *One-Hot encoding* em todos os atributos categóricos.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Employee ID           22750 non-null  object
 1   Date of Joining       22750 non-null  datetime64[ns]
 2   Gender                22750 non-null  object
 3   Company Type          22750 non-null  object
 4   WFH Setup Available   22750 non-null  object
 5   Designation           22750 non-null  int64
 6   Resource Allocation   21369 non-null  float64
 7   Mental Fatigue Score  20633 non-null  float64
 8   Burn Rate             21626 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(1), object(4)
memory usage: 1.6+ MB
```

- Gender One-Hot encoding

```
df['Gender'].value_counts()
```

```
Gender
Female    10691
Male       9657
Name: count, dtype: int64
```

```
df_v1 = df.copy()
gender_mapper = {'Male':0, 'Female':1}
df_v1['Gender'] = df_v1['Gender'].replace(gender_mapper)
df_v1 = df_v1.join(pd.get_dummies(df_v1['Gender'], prefix='Gender').astype(int))
df_v1.rename(columns={'Gender_0':'Male','Gender_1':'Female'},inplace=True)
df_v1.drop(['Gender'], axis = 1, inplace = True)
df_v1.head()
```

	Date of Joining	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate	Male	Female
0	2008-09-30	Service	No	2	3.0	3.8	0.16	0	1
1	2008-11-30	Service	Yes	1	2.0	5.0	0.36	1	0
3	2008-11-03	Service	Yes	1	1.0	2.6	0.20	1	0
4	2008-07-24	Service	No	3	7.0	6.9	0.52	0	1
5	2008-11-26	Product	Yes	2	4.0	3.6	0.29	1	0

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

- WFH Setup Available One-Hot encoding

```
df_v1['WFH Setup Available'].value_counts()
```

```
WFH Setup Available
Yes    11000
No      9348
Name: count, dtype: int64
```

```
workplace_mapper = {'No':0, 'Yes':1}
df_v1['WFH Setup Available'] = df_v1['WFH Setup Available'].replace(workplace_mapper)
df_v1 = df_v1.join(pd.get_dummies(df_v1['WFH Setup Available'], prefix='WFH Setup Available').astype(int))
df_v1.rename(columns={'WFH Setup Available_0':'Office','WFH Setup Available_1':'Home'},inplace=True)
df_v1.drop(['WFH Setup Available'], axis = 1, inplace = True)
df_v1.head()
```

	Date of Joining	Company Type	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate	Male	Female	Office	Home
0	2008-09-30	Service	2	3.0	3.8	0.16	0	1	1	0
1	2008-11-30	Service	1	2.0	5.0	0.36	1	0	0	1
3	2008-11-03	Service	1	1.0	2.6	0.20	1	0	0	1
4	2008-07-24	Service	3	7.0	6.9	0.52	0	1	1	0
5	2008-11-26	Product	2	4.0	3.6	0.29	1	0	0	1

- Company Type One-Hot encoding

```
df_v1['Company Type'].nunique()
```

```
2
```

```
df_v1['Company Type'].value_counts()
```

```
Company Type
Service    13316
Product     7032
Name: count, dtype: int64
```

```
company_mapper = {'Service':0, 'Product':1}
df_v1['Company Type'] = df_v1['Company Type'].replace(company_mapper)
df_v1 = df_v1.join(pd.get_dummies(df_v1['Company Type'], prefix='Company Type').astype(int))
df_v1.rename(columns={'Company Type_0':'Service','Company Type_1':'Product'},inplace=True)
df_v1.drop(['Company Type'], axis = 1, inplace = True)
df_v1.head()
```

	Date of Joining	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate	Male	Female	Office	Home	Service	Product
0	2008-09-30	2	3.0	3.8	0.16	0	1	1	0	1	0
1	2008-11-30	1	2.0	5.0	0.36	1	0	0	1	1	0
3	2008-11-03	1	1.0	2.6	0.20	1	0	0	1	1	0
4	2008-07-24	3	7.0	6.9	0.52	0	1	1	0	1	0
5	2008-11-26	2	4.0	3.6	0.29	1	0	0	1	0	1

EXPLORAÇÃO E PREPARAÇÃO DOS DADOS

5. Tratamento de datas

Para o tratamento das datas decidimos extrair o dia, mês e ano da data, para colunas em separado.

```
df_v1['Date of Joining'].head()
```

```
0   2008-09-30
1   2008-11-30
3   2008-11-03
4   2008-07-24
5   2008-11-26
Name: Date of Joining, dtype: datetime64[ns]
```

```
df_v1['year_of_joining'] = df_v1['Date of Joining'].dt.year
df_v1['month_of_joining'] = df_v1['Date of Joining'].dt.month
df_v1['day_of_joining'] = df_v1['Date of Joining'].dt.day
```

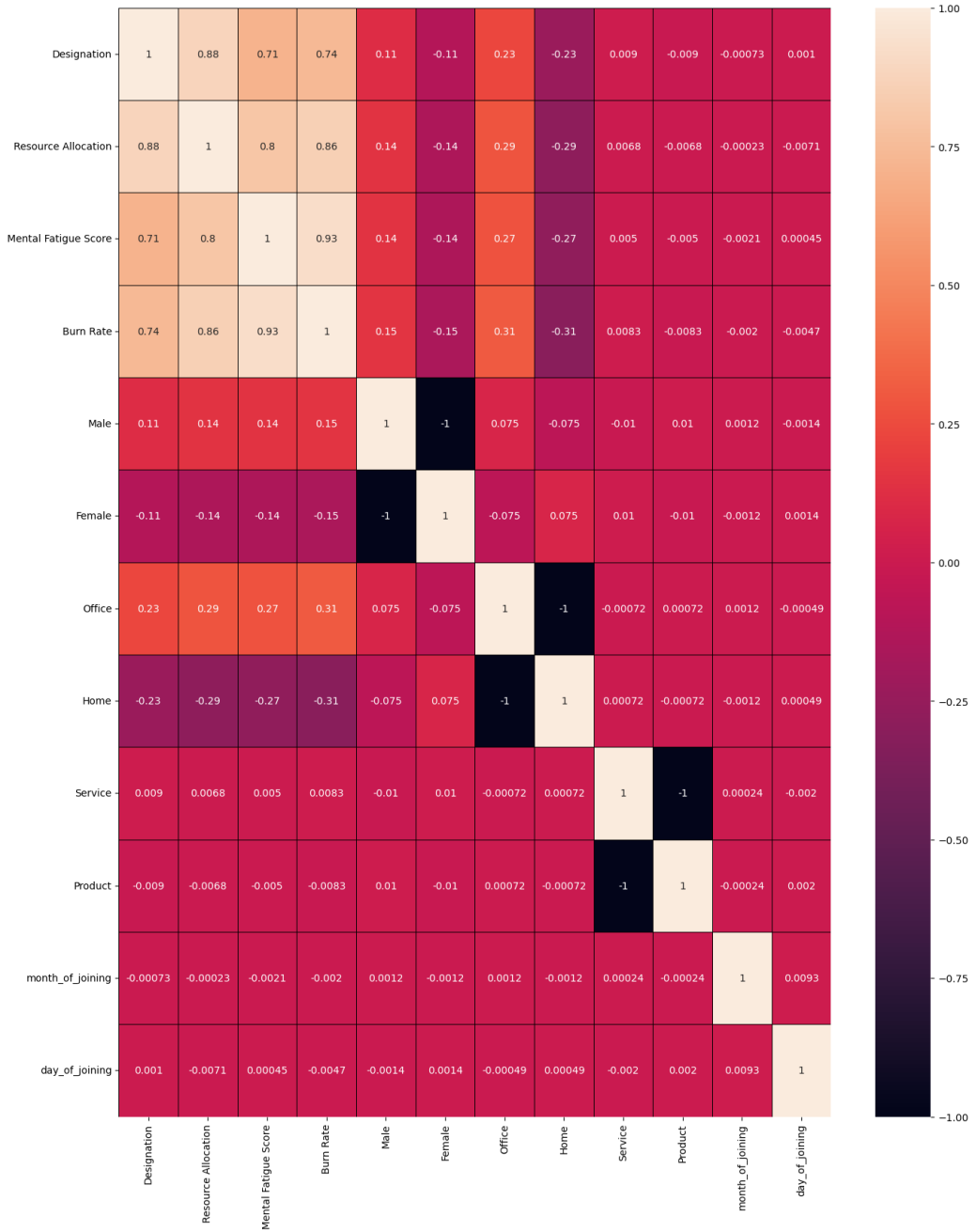
Como podemos constatar, a coluna referente ao ano de junção do funcionário exibe apenas um valor (2008).

```
df_v1.nunique()
```

```
Date of Joining          366
Designation                6
Resource Allocation       10
Mental Fatigue Score     107
Burn Rate                101
Male                       2
Female                     2
Office                     2
Home                       2
Service                    2
Product                    2
year_of_joining            1
month_of_joining          12
day_of_joining            31
dtype: int64
```

Assim, retiramos as colunas *Date of Joining* e *year_of_joining*

```
df_v1.drop(['Date of Joining','year_of_joining'], axis = 1, inplace = True)
```



ANÁLISE EXPLORATÓRIA DE DADOS

Podemos constatar que existem relações fortes entre os seguintes atributos:

- *burn rate* e *designation*
- *burn rate* e *Resource Allocation*
- *burn rate* e *mental fatigue score*

Entretanto, notamos que os atributos seguintes têm uma correlação próxima de zero com os demais atributos:

- *month_of_joining*
- *day_of_joining*

Devido à fraca correlação destes atributos, optamos por removê-los, igualmente ao que havia acontecido com o atributo *Employee ID*.

```
df_v1.drop(['month_of_joining','day_of_joining'], axis = 1, inplace = True)
```

```
df_v1.drop(['Product','Service'], axis = 1, inplace = True)
```

DADOS DE TREINO E TESTE

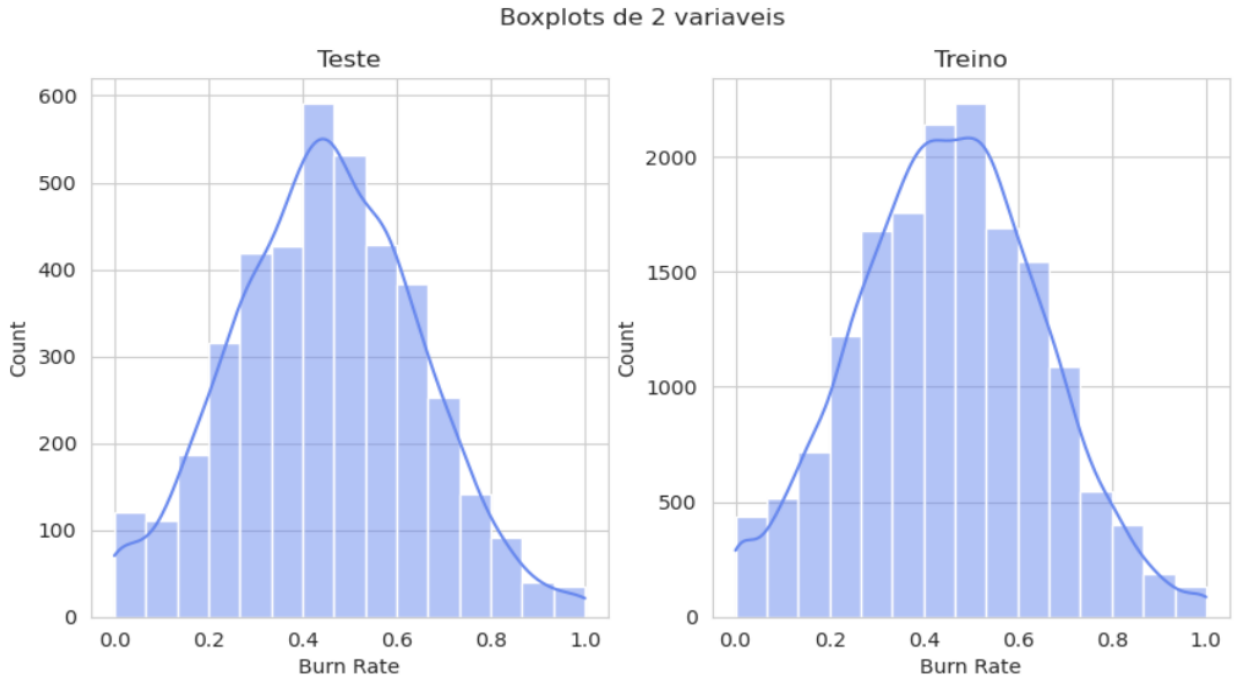
- Antes de treinar os modelos dividimos as ***features*** e a ***label*** em duas variaveis diferentes, sendo elas **X** e **y**, respetivamente.

```
X = df_v1.drop(['Burn Rate'], axis=1)
y = df_v1['Burn Rate']
```

- Dividimos então os dados em dados de treino e teste utilizando uma técnica de Hold-out Validation.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2023)
```

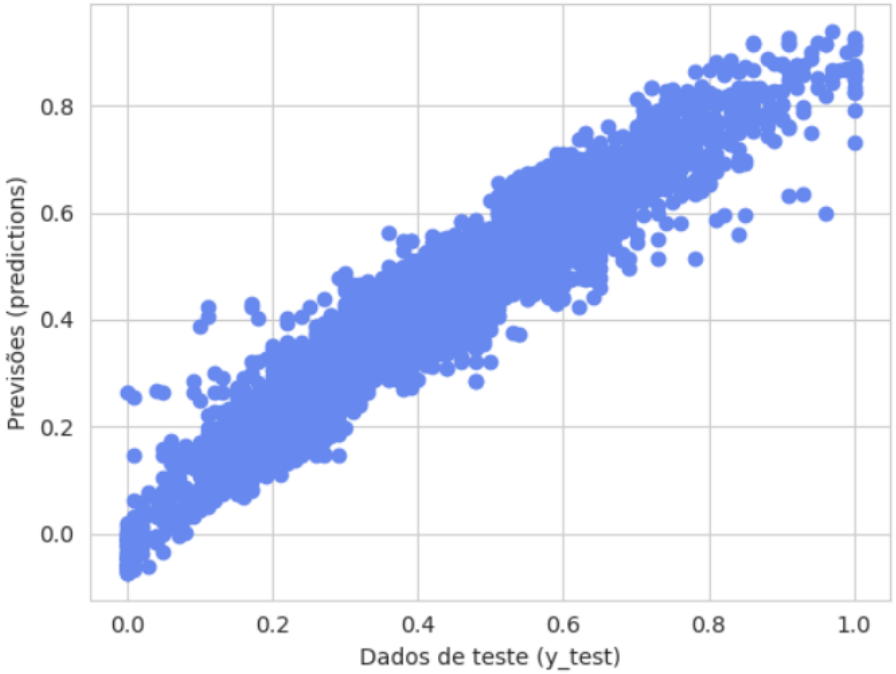
- De forma, a verificar se os dados de teste estão balanceados de forma semelhante aos dados de treino criamos os dois seguintes diagramas:



CRIAÇÃO E TREINO DE MODELOS

- Linear Regression

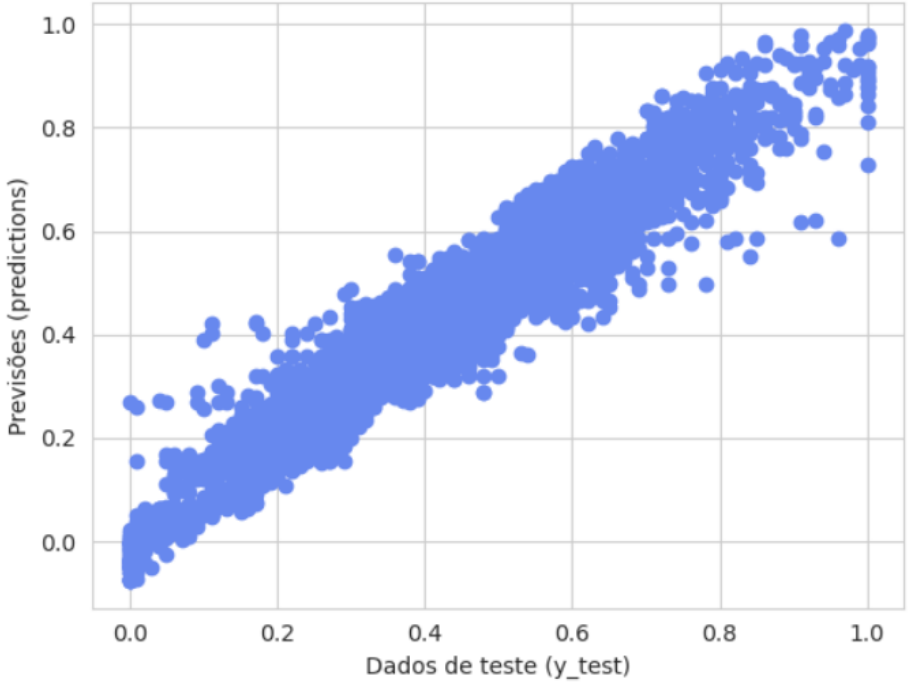
```
lm = LinearRegression()
lm.fit(X_train,y_train)
```




```
MAE: 0.04872399285439895
MSE: 0.0038600687723920513
RMSE: 0.062129451730979016
R-squared score: 0.9011000003884794
```

- SVR

```
svr = GridSearchCV(
    SVR(kernel="rbf"),
    param_grid={"C": [400,500,600], "gamma":[1,0.01,0.0001]},refit=True,verbose=3,scoring='neg_mean_absolute_error'
)
```

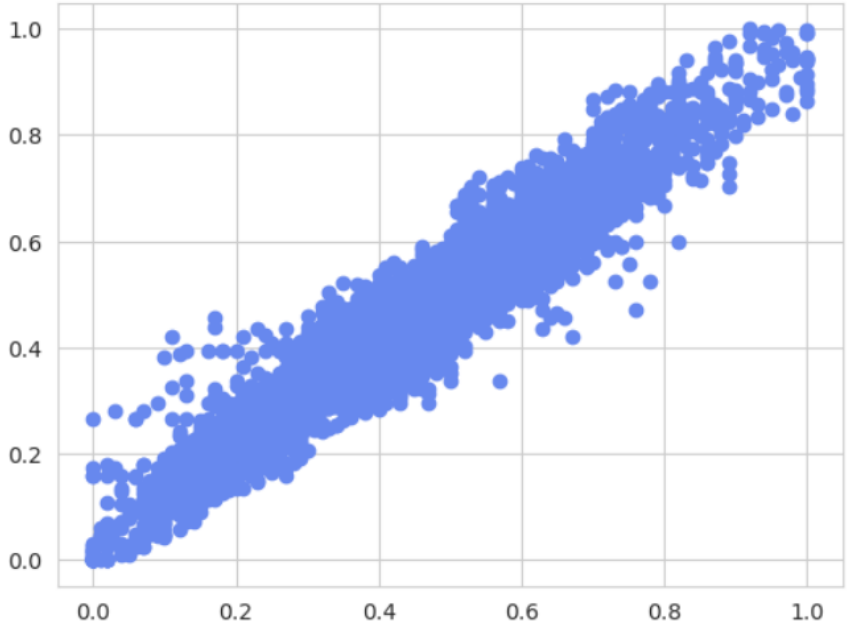



```
MAE: 0.04954229400349311
MSE: 0.0039086508165031226
RMSE: 0.06251920358180454
R-squared score: 0.9011000003884794
```

Artificial Neural Network

```
def build_model(activation='relu',learning_rate=0.01):
    model=Sequential()
    model.add(Dense(16,input_dim=7,activation=activation))
    model.add(Dense(9,activation=activation))
    model.add(Dense(1,activation=activation))

    model.compile(
        loss='mae',
        optimizer=tf.optimizers.Adam(learning_rate),
        metrics=['mae','mse'])
    return model
```

```
optimizer=['SGD','RMSprop','Adagrad']
activation=['sigmoid','relu']
param_grid =dict(optimizer=optimizer)
```




```
MAE: 0.0489675449763999
MSE: 0.0038331474462906493
RMSE: 0.06191241754519564
```

Gradient Boosting

```
param_boost = {'learning_rate': [0.01,0.02,0.03],'subsample': [0.1, 0.2, 0.4],'n_estimators' : [128,256,512],'max_depth': [4,8,12]}
```

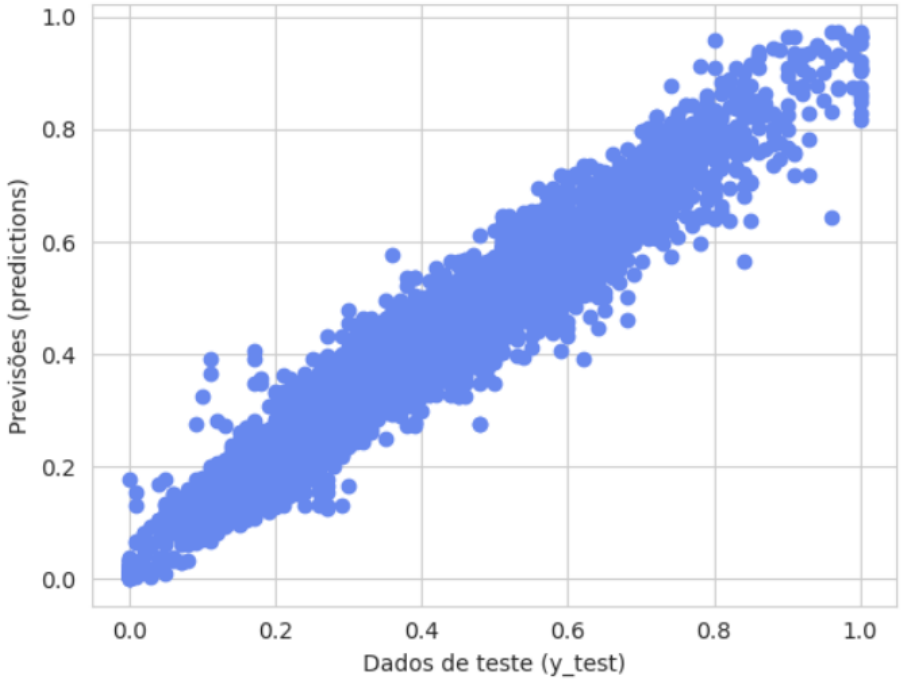
```
estimator_boost=GradientBoostingRegressor()
grid_boost=GridSearchCV(estimator_boost,param_boost,refit=True,verbose=0)
```

```
grid_boost.fit(X_train, y_train)
```

```
GridSearchCV(estimator=GradientBoostingRegressor(),
             param_grid={'learning_rate': [0.01, 0.02, 0.03],
                         'max_depth': [4, 8, 12],
                         'n_estimators': [128, 256, 512],
                         'subsample': [0.1, 0.2, 0.4]})
```

```
grid_boost.best_params_
```

```
{'learning_rate': 0.03, 'max_depth': 4, 'n_estimators': 512, 'subsample': 0.4}
```




```
MAE: 0.045492293029560094
MSE: 0.003299095791622502
RMSE: 0.05743775580245543
```

ANÁLISE FINAL

- Como é possível observar, o modelo que apresentou melhor resultados foi o **Gradient Boosting**.
- A regressão linear apresenta-se como sendo o algoritmo que apresenta menor tempo de execução, por ser mais simples.
- Na necessidade de efetuar uma previsão o mais acertada possível, sem restrições a nível de tempo, o modelo a ser utilizado deve ser o **Gradient Boosting**, sendo o que apresentou melhores resultados.