**Universidade do Minho**
Escola de Engenharia
Mestrado em Engenharia Informática

# Dados e Aprendizagem Automática

Ano lectivo 2023/2024

## Conceção e otimização de modelos de Machine Learning

Grupo 14

Ana Rita Santos Poças, PG53645

Miguel Silva Pinto, PG54105

Orlando José da Cunha Palmeira, PG54123

Pedro Miguel Castilho Martins, PG54146

10 de Janeiro de 2024

DAA

# Índice


**1 Introdução** **1**

**2 Tarefa *Dataset* Grupo** **2**
2.1 *Dataset* Escolhido . . . 2
2.2 Análise dos dados . . . 3
2.2.1 Compreensão dos dados . . . 3
2.2.2 Visualização dos Dados . . . 3
2.2.3 Relação entre os Atributos . . . 4
2.3 Tratamento dos Dados . . . 6
2.3.1 Remoção de colunas . . . 6
2.3.2 Transformação de *features* categóricas em numéricas . . . 7
2.4 Modelação dos Dados . . . 7
2.4.1 Estratégias de modelação . . . 7
2.4.2 Linear Regressor . . . 7
2.4.3 Decision Tree Regressor . . . 8
2.4.4 Random Forest Regressor . . . 9
2.4.5 Multilayer Perceptron . . . 10
2.5 Resultados obtidos e análise crítica . . . 12

**3 Tarefa *Dataset* Competição** **13**
3.1 Descrição do *dataset* . . . 13
3.2 Análise dos dados . . . 14
3.3 Tratamento dos dados . . . 16
3.3.1 Limpeza de dados . . . 16
3.3.2 Merging dos datasets . . . 16
3.3.3 Feature Engineering . . . 16
3.4 Modelação dos Dados . . . 17
3.4.1 Estratégias de modelação . . . 17
3.4.2 Multilayer Perceptron . . . 18
3.4.3 Stacking . . . 18
3.4.4 Max Voting . . . 18
3.4.5 XGBoosting . . . 19
3.5 Resultados obtidos e análise crítica . . . 19

# Índice de figuras


2.1 *Header* do *dataset* de grupo . . . . . 2
2.2 Classe dos voos . . . . . 3
2.3 Preço dos voos . . . . . 4
2.4 Relação Classe-Preço . . . . . 4
2.5 Relação Classe-Preço . . . . . 5
2.6 Relação Departure Time - Price e Arrival Time - Price . . . . . 5
2.7 Relação Duration - Price . . . . . 6
2.8 Matriz de Correlação . . . . . 6
2.9 ScatterPlot do modelo LinearRegression . . . . . 8
2.10 ScatterPlot do modelo Decision Tree Regressor . . . . . 9
2.11 ScatterPlot do modelo Random Forest Regressor . . . . . 10
2.12 Performance do Modelo Multilayer Perceptron . . . . . 11

3.1 Injeção na rede vs Autoconsumo (kWh) . . . . . 14
3.2 Injeção na rede vs Normal (kWh) . . . . . 15
3.3 Injeção na rede vs Clouds_all . . . . . 15

# 1 Introdução

Este relatório surge no âmbito do trabalho prático da Unidade Curricular de Dados e Aprendizagem Automática onde se pretende que ao longo do desenvolvimento das duas tarefas propostas pela equipa docente, sejamos capazes de aprofundar os conhecimentos relativos a Machine Learning adquiridos ao longo do semestre, através do desenvolvimento de um projeto de Machine Learning.

A primeira tarefa que iremos abordar consiste numa **tarefa de *dataset* grupo**, onde nos foi permitido escolher um *dataset*, que obtivemos na plataforma *Kaggle*, de forma a concluirmos a sua exploração, análise e preparação e, assim, sermos capazes de extrair o conhecimento relevante do problema proposto no *dataset* e, desta forma, conseguirmos conceber e otimizar múltiplos modelos de Machine Learning.

A segunda tarefa consiste numa tarefa *dataset* competição, onde a equipa docente nos forneceu dois *datasets* (um de aprendizagem e um de teste) sobre os quais desenvolvemos a sua exploração, análise e o preparamos de forma a extrair o conhecimento relevante do problema em questão, de forma a podermos conceber e otimizar modelos de Machine Learning.

# 2 Tarefa *Dataset* Grupo

## 2.1 *Dataset* Escolhido

O *dataset* escolhido pelo grupo foi o ***Flight Price Prediction***, disponível no Kaggle, qye possui 12 colunas e 300153 linhas. Os dados deste *dataset* foram obtidos a partir do website "Ease My Trip" (uma plataforma que permite a compra de passagens aéreas) e o objetivo é, essencialmente, analisar os dados nele presentes e verificar de que forma o preço do voo (atributo objetivo) varia de acordo com os restantes atributos. O *dataset* possui os seguintes atributos:

- **Airline**: O nome da companhia aérea;
- **Flight**: Código do voo;
- **Source City**: Cidade de onde parte o voo;
- **Departure Time**: Período de tempo em que sai o voo (Early Morning, Morning, Afternoon, Evening, Night, Late_Night);
- **Stops**: Número de paragens entre a origem e o destino;
- **Arrival Time**: Período de tempo em que chega o voo (Early Morning, Morning, Afternoon, Evening, Night, Late_Night);
- **Destination City**: Cidade de destino do voo;
- **Class**: Informação sobre o tipo de lugar do voo (Business, Economy);
- **Duration**: Duração do voo em horas;
- **Days Left**: Dias que faltam até ao voo, calculado pela subtração ao dia da viagem do dia onde foi marcado o voo;
- **Price**: Variável objetivo que guarda o valor do bilhete;

A figura abaixo demonstra as primeiras linhas do *dataset* escolhido:

| | serial_number | airline | flight | source_city | departure_time | stops | arrival_time | destination_city | class | duration | days_left | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | SpiceJet | SG-8709 | Delhi | Evening | zero | Night | Mumbai | Economy | 2.17 | 1 | 5953 |
| 1 | 1 | SpiceJet | SG-8157 | Delhi | Early_Morning | zero | Morning | Mumbai | Economy | 2.33 | 1 | 5953 |
| 2 | 2 | AirAsia | I5-764 | Delhi | Early_Morning | zero | Early_Morning | Mumbai | Economy | 2.17 | 1 | 5956 |
| 3 | 3 | Vistara | UK-995 | Delhi | Morning | zero | Afternoon | Mumbai | Economy | 2.25 | 1 | 5955 |
| 4 | 4 | Vistara | UK-963 | Delhi | Morning | zero | Morning | Mumbai | Economy | 2.33 | 1 | 5955 |

Figura 2.1: *Header* do *dataset* de grupo

## 2.2 Análise dos dados

De forma a identificarmos qual o tratamento mais adequado para a utilização dos dados presentes no *dataset* para a construção dos modelos de aprendizagem automática, procedemos à sua análise.

### 2.2.1 Compreensão dos dados

De modo a melhor compreendermos os dados presentes no dataset, verificamos a sua estrutura, que tipo de atributos possuía (categóricos ou numéricos), a presença de *missing values* ou duplicated values e, desta forma, fomos capazes de aferir os seguintes factos:

- Não existem missing values.
- Não tem linhas duplicadas.
- Os atributos categóricos são: airline,flight,source_city,departure_time,stops,arrival_time,destination_city,class.
- Os atributos numéricos são: serial_number, duration,days_left,price.

### 2.2.2 Visualização dos Dados

Através do uso de *barplots* fomos capazes de visualizar os dados do dataset e obter algumas conclusões:

**Classe dos voos**

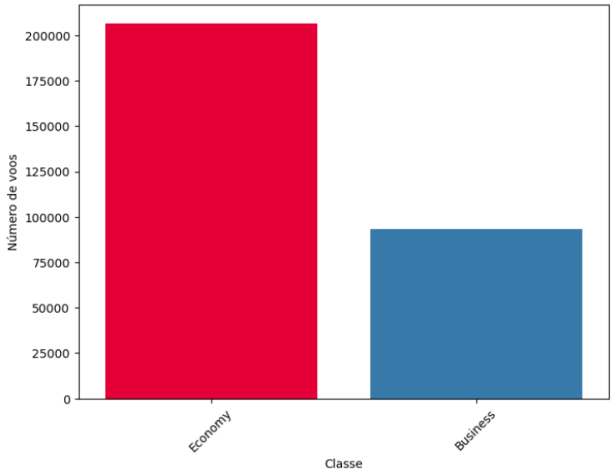


Figura 2.2: Classe dos voos

O gráfico acima permite observar que a maioria dos voos foram realizados em **classe económica**.

**Preço dos voos**

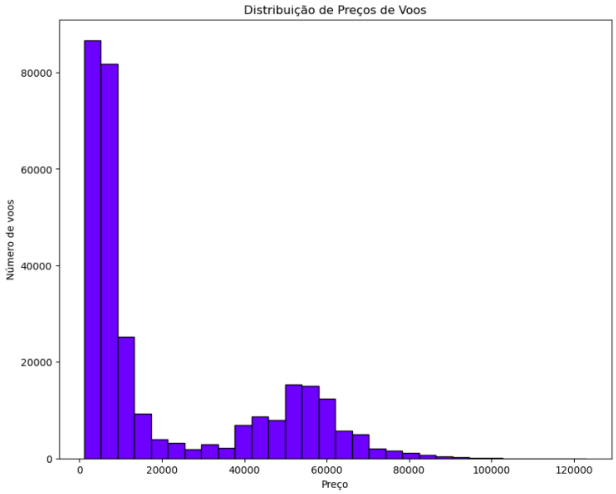


Figura 2.3: Preço dos voos

Podemos observar que a maioria dos voos realizados custou **menos de 20 000**.

### 2.2.3 Relação entre os Atributos

De modo a avaliarmos as relações presentes entre os atributos uso de *barplots*, *hisplots*, *boxplots*, *violinplots*,*lineplots* fomos capazes de obter as seguintes conclusões:

**Classe-Preço**

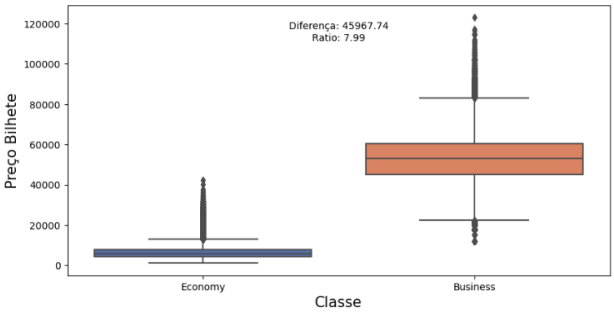


Figura 2.4: Relação Classe-Preço

Podemos observar, no gráfico acima, que há uma diferença bastante significativa entre os preços dos bilhetes ***Economy*** e ***Business***, sendo que os da classe ***Business*** são bastante **mais caros**, sendo em média **7.99 vezes mais caros** que os da classe **Economy**.

**Dias até partir - Preço - Companhia aérea**

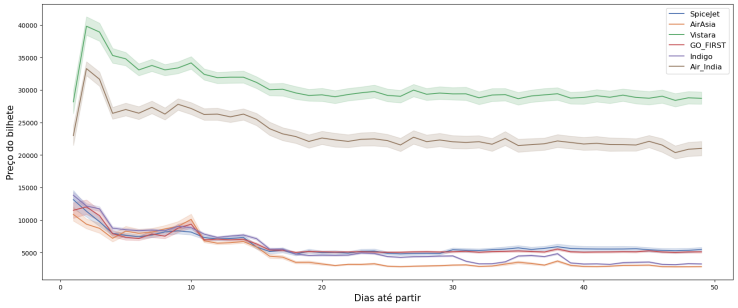


Figura 2.5: Relação Classe-Preço

Pela a análise do gráfico, podemos perceber que companhia aérea que a **Vistara** e a **Air_India** registam um **maior aumento de preços à medida que se aproxima do dia do voo**, enquanto que companhias como a **AirAsia e a GO_FIRST não registam um aumento tão grande**. Em geral, é possível observar no gráfico, que **à medida que se aproxima da data do voo**, o bilhete **fica bastante mais caro**.

**Departure Time - Price e Arrival Time - Price**

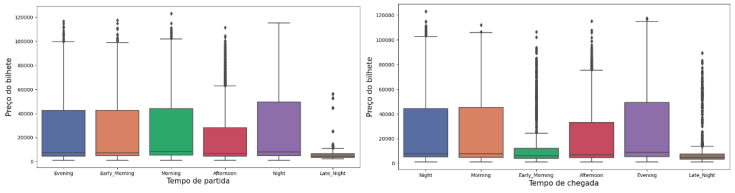


Figura 2.6: Relação Departure Time - Price e Arrival Time - Price

Verifica-se, no gráfico da esquerda, que o voo é **mais caro** para quando a partida é feita durante a **noite**(Night) e **mais barato** quando é feito durante a **madrugada** (Late Night).

No gráfico da direita, verifica-se que o voo é **mais caro** quando chega de **noite** (Night) e **mais barato** quando chega de **madrugada**(Late_Night).

### Duration - Price

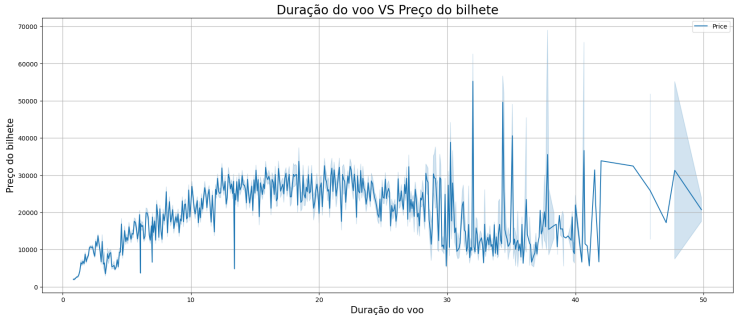


Figura 2.7: Relação Duration - Price

Como é possível observar, por norma, **os preços ficam mais altos entre as 20 e as 30 horas**.

### Matriz de Correlação

Obtivemos a seguinte matriz de correlação:

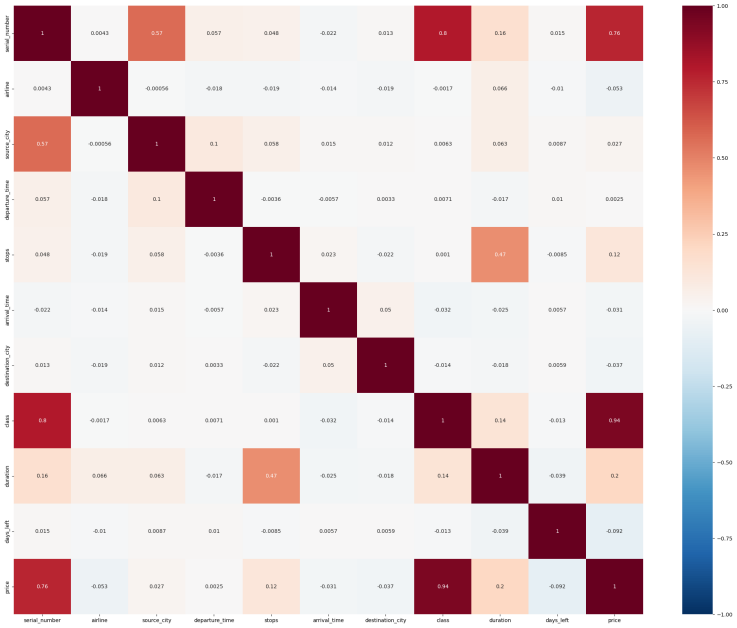


Figura 2.8: Matriz de Correlação

## 2.3 Tratamento dos Dados

### 2.3.1 Remoção de colunas

Após a realização da análise dos dados presentes no dataset, prosseguimos para o tratamento e limpeza dos dados. Para esta fase aplicamos técnicas de *Feature Engineering*, como *Label Encoding* e *One Hot*

*Encoding*, técnicas de *Feature Selection* que levou à remoção das colunas de *serial_number* e *flight*, por não apresentarem informação relevante ao treinamento de modelos de previsão.

### 2.3.2 Transformação de *features* categóricas em numéricas

**Na matriz de correlação**

Para obtermos a matriz de correlação, efetuamos o seguinte tratamento em:

- Convertemos os 3 valores categóricos únicos do atributo ***stops*** (zero, one, two_or_more) em numéricos (0,1,2).
- Convertemos os 2 valores categóricos únicos do atributo ***class*** (*Economy* e *Business*) em numéricos (1,2).
- Nos restantes atributos categóricos, decidimos fatorizá-los de forma a torná-los numéricos.

**Para a modelação dos dados**

As técnicas de *Label Encoding* e *One Hot Encoding* foram aplicadas às colunas *airline*, *source_city*, *departure_time*, *stops*, *arrival_time*, *destination_city* e *class*. Este tratamento foi necessário para converter estas *features* de categóricas para *features* numéricas. Foram utilizadas estas 2 técnicas para avaliar qual teria o melhor resultado para o nosso caso de estudo.

Outro tratamento aplicado foi a remoção de *outliers* da *feature* objetivo ***Price***, onde utilizamos a estratégia de intervalos interquartis para remover entradas com valores consideravelmente elevados.

Por fim aplicamos uma normalização aos dados num intervalo entre 0 e 1.

## 2.4 Modelação dos Dados

### 2.4.1 Estratégias de modelação

Para desenvolver modelos capazes de fazer a previsão dos preços dos voos começamos por explorar os algoritmos de regressão estudados nas aulas. Os algoritmos escolhidos foram os seguintes:

- Linear Regressor
- Decision Tree Regressor
- Random Forest Regressor
- Multilayer Perceptron

Estes modelos foram escolhidos porque permitem uma abordagem diferente ao problema e desta forma podemos avaliar qual abordagem é a mais indicada ao problema.

### 2.4.2 Linear Regressor

O primeiro modelo que testamos foi o **Linear Regressor**, que é um modelo com o propósito de resolver problemas lineares. Porém após aplicar o modelo ao nosso dataset ficou claro que este modelo não era capaz de resolver o nosso problema, visto que a relação entre as variáveis independentes e a variável objetivo não é linear.

Este foi o resultado que obtivemos com este modelo:

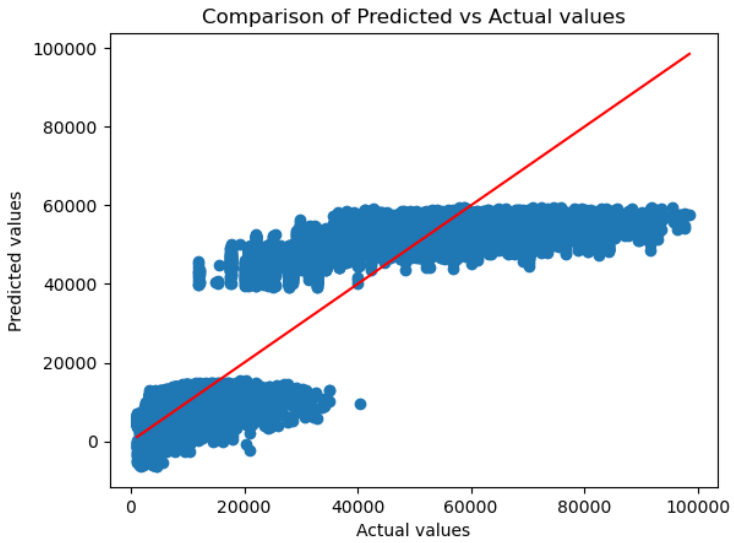


Figura 2.9: ScatterPlot do modelo LinearRegression

Como podemos ver o modelo não conseguiu obter bons resultados para o nosso dataset.

### 2.4.3 Decision Tree Regressor

Para o modelo **Decision Tree Regressor** nós primeiramente avaliamos o modelo sem quaisquer hiperparâmetros para ganhar uma noção inicial dos resultados. Para isso calculamos o valor de **MAE**, **MSE** e **RMSE** com a técnica de **Label Encoding** e de **One Hot Encoding** para avaliar qual seria a técnica de *encoding* dos dados mais apropriada ao modelo.

| | Label Encoding | One Hot Encoding |
|---|---|---|
| MAE | 1170.22 | 2293.87 |
| MSE | 11781314.31 | 26823025.35 |
| RMSE | 3432.39 | 5179.10 |

Como podemos ver o **Label Encoding** teve uma melhor performance em todas as métricas. Isto deve-se ao facto do **One Hot Encoding** aumentar demasiado a complexidade do problema o que afeta a performance das árvores de decisão.

Com o *encoding* escolhido passamos para a exploração dos melhores hiperparâmetros para o nosso modelo. Para isso recorremos a um **GridSearch** e com ele avaliamos diferentes valores para os parâmetros *criterion*, *max_depth*, *min_samples_split* e *min_samples_leaf* por forma a obter os valores mais adequados para o nosso problema. O *scoring* usado no **GridSearch** foi "*neg_mean_squared_error*", porque consideramos que os erros de maior escala deveriam ser mais penalizados. Os melhores valores dos hiperparâmetros foram os seguintes:

- ***criterion*** : "*friedman_mse*"
- ***max_depth*** : 30
- ***min_samples_split*** : 5
- ***min_samples_leaf*** : 7

Com estes hiperparâmetros obtivemos estes resultados:

| | Modelo Base | Modelo com Hiperparâmetros |
|---|---|---|
| MAE | 1170.22 | 1264.48 |
| MSE | 11781314.31 | 8484109.77 |
| RMSE | 3432.39 | 2912.75 |

Ao comparar os resultados podemos ver que os hiperparâmetros conseguiram ajustar o modelo para que os erros de grande escala não sejam tão frequentes porque houve uma diminuição significativa nos valores de **MSE** e **RMSE**. Apesar do modelo possuir um erro em média (**MAE**) maior do que o inicial, achamos que para este caso de estudo achamos que esta métrica é menos importante.

Este foi o **scatter plot** obtido das previsões do modelo:

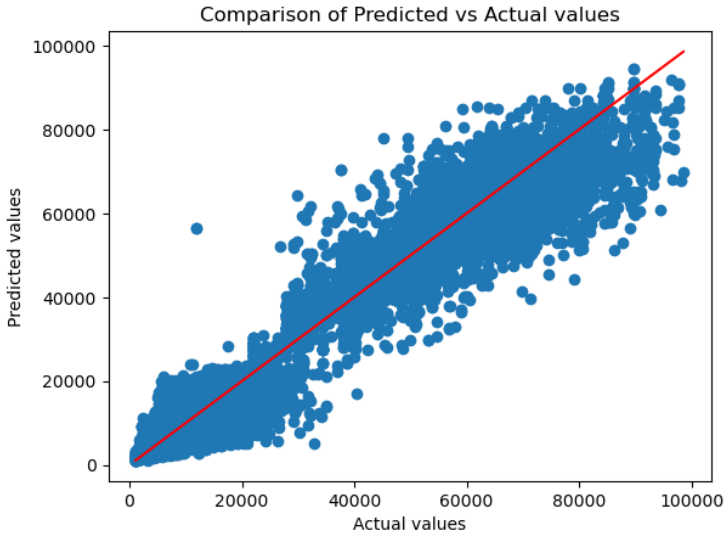


Figura 2.10: ScatterPlot do modelo Decision Tree Regressor

Pelo **scatter plot** conseguimos ver uma melhoria substancial na previsão deste modelo quando comparado ao **Linear Regression**, no entanto ainda conseguimos detetar vários casos onde a previsão está bastante distante do valor real.

### 2.4.4 Random Forest Regressor

O modelo **Random Forest Regressor** é um modelo de *ensemble* constituído por várias árvores de decisão treinadas com diferentes *subsets* dos dados de treino.

Para este modelo utilizamos o mesmo tratamento feito para o **Decision Tree Regressor**, porque são ambas baseadas em árvores por isso não foi necessário um tratamento especial.

A avaliação deste modelo foi semelhante à do **Decision Tree Regressor**, onde começamos por avaliar o modelo sem quais quer hiperparâmetros e depois aplicamos o **GridSearch** para descobrir os melhores hiperparâmetros para o modelo. Os melhores hiperparâmetros obtidos foram os seguintes:

- ***n_estimators*** : 725
- ***max_depth*** : 35

- ***min_samples_split*** : 5
- ***min_samples_leaf*** : 7

Os resultados obtidos foram os seguintes:

| | Modelo Base | Modelo com Hiperparâmetros |
|---|---|---|
| MAE | 1099.99 | 1255.31 |
| MSE | 7550163.61 | 7420787.74 |
| RMSE | 2747.76 | 2724.11 |

Como podemos ver o modelo com hiperparâmetros voltou a diminuir o valor de **RMSE**, aumentando no entanto o valor de **MAE**.

Este foi o **scatter plot** obtido das previsões do modelo:

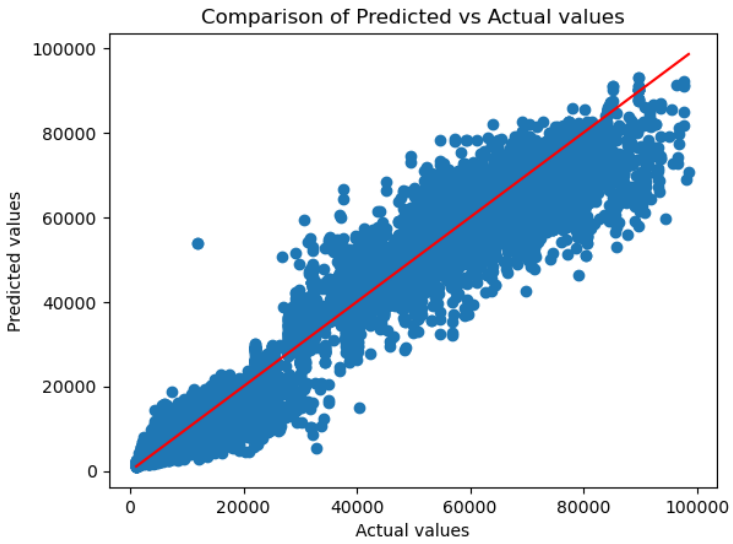


Figura 2.11: ScatterPlot do modelo Random Forest Regressor

Através deste **scatter plot** conseguimos ver a sua semelhança ao do modelo **Decision Tree Regression** porém um pouco mais justo à linha linear ótima.

### 2.4.5 Multilayer Perceptron

O modelo **Multilayer Perceptron** é um tipo de rede neuronal artificial capaz de reconhecer e aprender padrões existentes nos dados.

A primeira decisão da tomar é a construção do modelo. O modelo foi escolhido após vários testes feitos com os dados de treino. O melhor modelo encontrado foi o seguinte:

```
def build_model(activation = 'sigmoid', learning_rate = 0.001, input_dim=9):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation=activation))
    model.add(Dense(32, activation=activation))
```

```
model.add(Dense(16, activation=activation))
model.add(Dense(8, activation=activation))
model.add(Dense(1, activation=activation)) #output

# Compile the model

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

Este modelo apresenta uma estrutura de 4 *hidden layers* onde a primeira tem 64 neurónios, a segunda 32 neurónios, a terceira 16 neurónios e a ultima 8 neurónios.
A função de ativação escolhida foi o ***sigmoid***, com um ***learning rate*** de 0.001.
Para compilação utilizamos o **MSE** como métrica de **Loss** porque queremos que os erros de maior escala sejam mais penalizados como já foi explicado anteriormente. O ***optimizer*** que obteve os melhores resultados foi o ***Adam***.

Para a análise dos modelos foi usado o *k-fold cross validation* com 3 *folds*. O modelo estava a utilizar um ***batch_size*** de 128 devido à dimensão do dataset e na validação foram usados apenas 10 ***epochs*** devido ao tempo de treinamento.

Este foi a performance do modelo obtida. Como podemos ver o modelo teve uma curva de aprendizagem quase perfeita entre os dados de treino e os dados de validação, o que indica que o modelo está com uns parâmetros bons para o caso de estudo em questão.

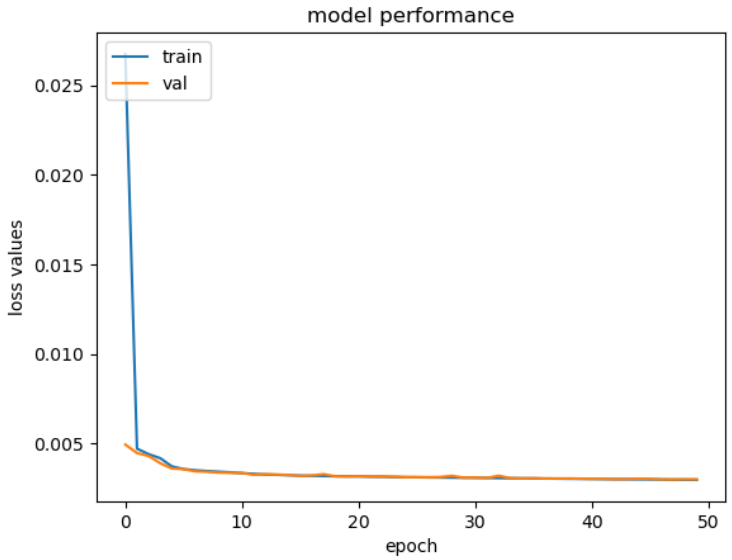


Figura 2.12: Performance do Modelo Multilayer Perceptron

Os valores obtidos com este modelo foram os seguintes:

| | Label Encoding | One-Hot Encoding |
|---|---|---|
| MAE | 2979.43 | 3325.44 |
| MSE | 24661669.81 | 28830206.0 |
| RMSE | 4966.05 | 5369.38 |

Como foi feito para os outros modelos também decidimos testar a diferenças dos *encoding* dos dados categóricos. E mais uma vez o ***label encoding*** mostrou-se uma opção melhor para o nosso caso tento uma melhor performance em todas as métricas calculadas.

## 2.5 Resultados obtidos e análise crítica

Para terminar esta tarefa iremos fazer uma análise ao trabalho desenvolvido em relação ao dataset escolhido pelo grupo.

Em relação ao tratamento de dados conseguimos perceber pelos diversos modelos desenvolvidos e testes realizado de que o ***Label Encoding*** dos atributos categóricos era uma melhor escolha quando comparada ao ***One-Hot Encoding***. Isto em parte já seria o esperado porque o ***One-Hot Encoding*** elevava bastante a complexidade do problema, devido ao aumento de colunas presentes no dataset, o que poderia afetar a performance dos modelos.

Já quanto aos modelos explorados conseguimos perceber que os que obtiveram os melhores resultados foram os modelos baseados em árvores (**Decision Tree** & **Random Forest**).

Estes modelos conseguiram obter um valor de **RMSE** por volta dos 2700-2900, enquanto que os modelos **Linear Regression** e **Multilayer Perceptron** demonstraram não serem capazes de obter bons valores. Enquanto que o modelo **Linear Regression** já seria esperado devido ao facto do problema em mãos não ser um problema linear, no entanto o modelo de redes neuronais surpreendeu-nos porque estávamos à espera de conseguir obter resultados semelhantes ou até melhores aos modelos explorados anteriormente. Isto claro pode ser explicado pela construção do modelo, no entanto nós exploramos várias alternativas e não conseguimos bons resultados.

Por fim, nós consideramos que o nosso modelo apresenta um bom resultado tendo em conta os valores muito dispersos presente no dataset. Como trabalho futuro pensamos que poderíamos trabalhar um pouco mais nos dados como por exemplo fazer uma divisão no atributo objetivo de forma a ter os valores mais contidos numa gama e talvez adicionar mais dados de voos de outras fontes ou até de voos entre outras cidades.

# 3 Tarefa *Dataset* Competição

## 3.1 Descrição do *dataset*

Para a competição criada pela equipa docente foram disponibilizados dois tipos de *datasets* - um com dados energéticos outro com dados meteorológicos. Decidimos usar os *datasets* dos anos de 2021,2022 e 2023. O *dataset* de energia possui os seguintes atributos:

- **Data**: o dia associado ao registo.
- **Hora**: a hora associada ao registo.
- **Normal (kWh)**: quantidade de energia elétrica consumida, em kWh, proveniente da rede elétrica, num período considerado normal em termos de ciclo bi-horário diários (horas fora de vazio).
- **Horário Económico**: quantidade de energia elétrica consumida, em kWh, proveniente da rede elétrica, num período considerado económico em termos de ciclo 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.

O dataset de metereologia possui os seguintes atributos:

- **dt**: o timestamp associado ao registo.
- **dt_iso**: a data associada ao registo.
- **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.

## 3.2 Análise dos dados

Através do uso de *barplots* e *scatterplots* fomos capazes de visualizar a relação existente entre os dados que consideramos mais relevantes no dataset e obter algumas conclusões:

**Relação entre o Autoconsumo e a Injeção de energia na Rede**

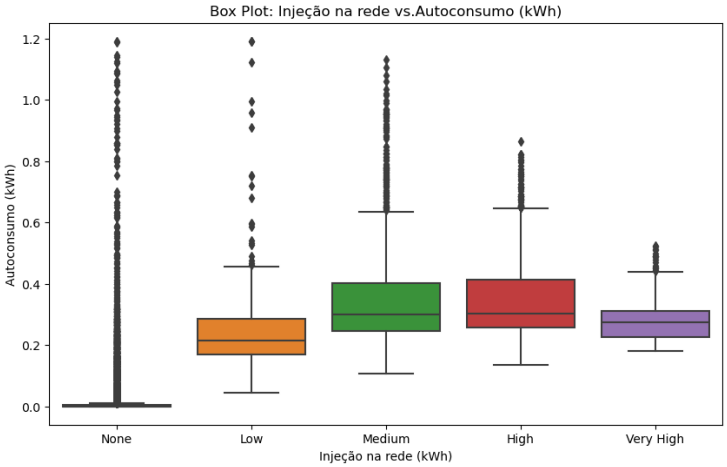


Figura 3.1: Injeção na rede vs Autoconsumo (kWh)

Podemos ver através deste *boxplot* que o auto consumo para os casos onde não há injeção na rede é, na maior parte dos casos, 0. Isto deve-se ao facto de que quando não há consumo de energia proveniente dos painéis solares é porque os painéis solares não estão em funcionamento, quer seja por estar muito nublado ou por estar de noite.

**Relação entre o consumo Normal e a Injeção de energia na Rede**

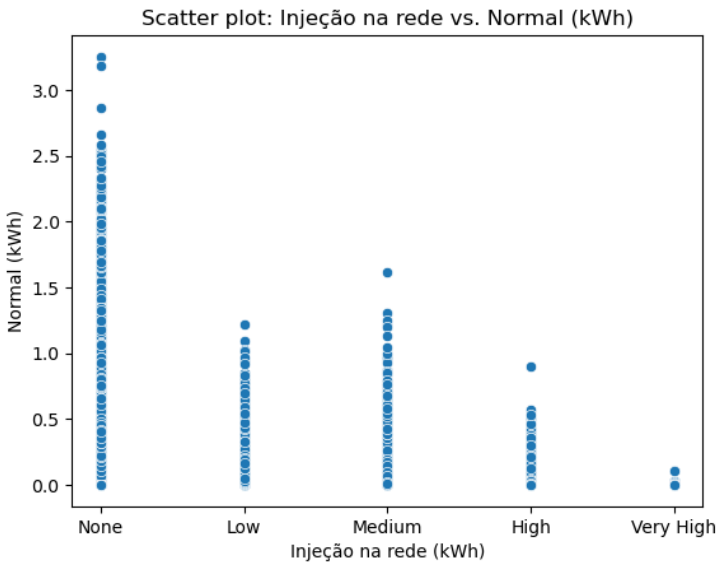


Figura 3.2: Injeção na rede vs Normal (kWh)

Neste gráfico conseguimos analisar o consumo de energia num horário normal e como isso afeta a injeção de energia na rede. Podemos claramente ver que para os casos de maior injeção o consumo tende a ser menor.

**Relação entre o Clouds_all e a Injeção de energia na Rede**

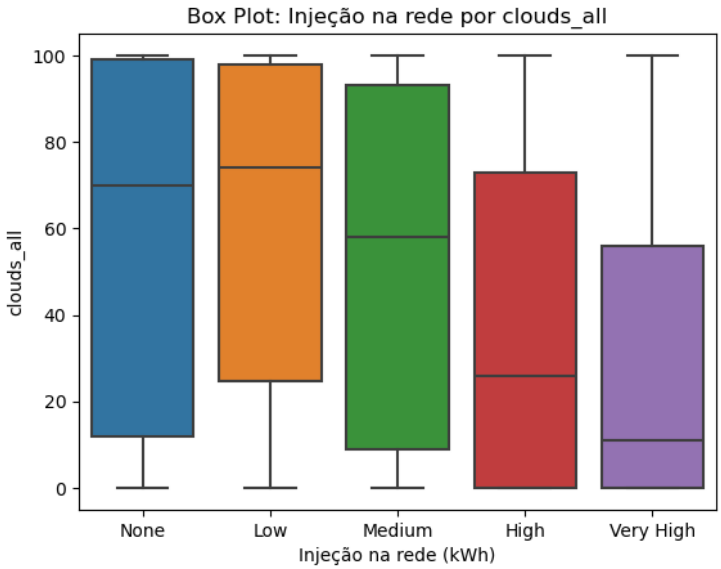


Figura 3.3: Injeção na rede vs Clouds_all

Este gráfico demonstra como as nuvens afetam a injeção de energia na rede. Conseguimos perceber que para os níveis maiores de injeção a quantidade de nuvens no céu tem tendência a diminuir pois os painéis solares funcionam melhor quando têm um contacto direto com os raios solares.

## 3.3 Tratamento dos dados

### 3.3.1 Limpeza de dados

A limpeza dos dados começou com a concatenação dos *datasets* de energia dos anos de 2021 e 2022. Estes dados irão ser os dados de teste usados mais tarde pois possuem o atributo objetivo "Injeção na rede (kWh)". Dessa forma conseguimos avaliar os modelos antes de fazer a previsão com os dados de 2023. A mesma concatenação foi feita aos dados meteorológicos de 2021 e 2022.

Após esta junção começamos a explorar os *datasets* de forma a encontrar problemas como *missing values*, atributos irrelevantes e qualquer tratamento que seja necessário nos atributos.

O primeiro problema que encontramos nos dados meteorológicos foi a existência de atributos com *missing values*. Essas atributos eram *sea_level*, *grnd_level* e *rain_1h*. No caso dos atributos *sea_level* e *grnd_level* todos os valores estavam vazios o que levou à remoção dos mesmos. Já o atributo *rain_1h* possuía apenas algumas linhas com um valor nulo e após uma análise do atributo percebemos que para os casos onde não existia um valor deveríamos atribuir o valor 0 de forma a indicar que nessa entrada do dataset não houve precipitação.

Os atributos *city_name* e *dt* também foram removidos por possuírem um único valor para cada entrada no caso do *city_name* e por possuir um valor único para cada entrada do *dataset*.

Para finalizar a limpeza dos dados meteorológicos convertemos o atributo *dt_iso* para *Datetime*. Esta transformação irá ser necessária durante o *merge* dos *datasets* de meteorologia e energia.

A limpeza feita no *dataset* de energia baseou-se apenas na conversão do atributo *Data* e *Hora* para um único atributo no formato *Datetime*.

### 3.3.2 Merging dos datasets

O *merging* dos *datasets* de energia e meteorologia foi feito através do atributo *Data* de cada *dataset*. Desta forma conseguimos associar o estado meteorológico com a informação energética daquele dia e hora.

Ao fazer o *merge* deparamo-nos com alguns horários onde a informação meteorológica não estava disponível nos *datasets* fornecidos pelos docentes. Os dias eram entre 2023-03-15 até 2023-04-04, onde seria necessário fazer a previsão da Injeção na Rede para a competição, no entanto sem os dados meteorológicos essas previsões poderiam perder precisão levando a um pior desempenho dos modelos.

Para resolver este problema recorremos à OpenWeather API para obter os dados meteorológicos para esses dias.

### 3.3.3 Feature Engineering

De forma a melhorarmos o desempenho geral do nosso modelo, decidimos criar as seguintes variáveis:

- **Ano**, obtido a partir da coluna Data.
- **Mês**, obtido a partir da coluna Data.
- **Dia**, obtido a partir da coluna Data.
- **Hora**, obtido a partir da coluna Data.

- **pressure_level**, onde caracterizamos os valores da coluna *pressure* (pressão atmosférica) em níveis específicos: *low*, *medium*, *high*.
- **rolling_avg_temp**: representa a "média móvel" da temperatura máxima ao longo de uma janela de 3 dias.
- **temp_humidity_interaction**: representa a interação entre a temperatura máxima e a humidade.
- **wind_rain_interaction**: representa a interação entre a velocidade do vento e a quantidade de chuva.
- **temp_wind_interaction**: representa a interação entre a temperatura máxima e a velocidade do vento.
- **wind_humidity_interaction**: representa a interação entre a velocidade do vento e a humidade.
- **humidity_rain_interaction**: representa a interação entre a humidade e a quantidade de chuva.
- **month_mean_consumption**: representa a média de consumo de energia por mês.
- **daily_mean_consumption**: representa a média de consumo de energia por dia.
- **relative_humidity**: representa a humidade relativa em termos percentuais.
- **wind_speed_category**: categoriza a velocidade do vento em *calm*, *light*, *moderate* e *strong*.
- **lag_temp_max**: representa a temperatura máxima defasada num dia.
- **temp_change**: representa a mudança na temperatura máxima em relação ao dia anterior.
- **temp_deviation_from_month_mean**: representa a diferença entre a temperatura máxima e a média mensal da temperatura máxima.
- **wind_speed_deviation_from_daily_mean**: representa a diferença entre a velocidade do vento e a média diária da velocidade do vento.
- **humidity_deviation_from_month_mean**: representa a diferença entre a humidade e a média mensal da humidade.
- **rolling_avg_temp_humidity_interaction**: representa a interação entre a média móvel da temperatura máxima e a humidade.
- **wind_humidity_rain_interaction**: representa a interação entre a velocidade do vento, humidade e a quantidade de chuva.

## 3.4 Modelação dos Dados

### 3.4.1 Estratégias de modelação

Para o desenvolvimento de modelos capazes de preverem a quantidade de energia injetada na rede elétrica de uma determinada localidade recorremos a vários algoritmos de previsão. Os algoritmos explorados foram: *Decision Trees*, *Random Forest*, *Support Vector*, *Multilayer Perceptron*, *Gradient Boosted*, *Stacking*, *Max Voting* e *XGBoosting*.

Neste relatório iremos explorar em mais detalhe os modelos com os algoritmos ***Multilayer Perceptron***, ***Stacking***, ***Max Voting*** e ***XGBoosting*** pois foram aqueles que obtiveram os melhores resultados durante a competição.

Durante a criação destes modelos os mesmos foram testados a partir dos dados de 2021 e 2022 pois conseguíamos fazer uma avaliação dos resultados devido à presença do atributo objetivo. Sempre que

possível também utilizávamos as submissões presentes no *Kaggle* para ter uma avaliação mais concreta do modelo em relação ao verdadeiro objetivo da competição que era a previsão da Injeção na Rede durante o ano de 2023.

### 3.4.2 Multilayer Perceptron

Para o modelo de redes neuronais começamos por fazer uma normalização dos dados do dataset para que os seus valores fiquem contidos entre 0 e 1. Isto é bastante vantajoso porque este tipo de modelos sofrem bastante na sua capacidade de previsão com dados em diferentes escalas.

O modelo usado foi o **MLPClassifier** presente na biblioteca *sklearn.neural_network*. Este modelo permite a construção de modelos de redes neuronais com diferentes números de *layers* e neurónios.

Para facilitar a escolha da estrutura da rede neuronal e de outros parâmetros utilizamos o *Gridsearch*. Após vários testes estes foram os parâmetros explorados juntamente com os seus valores:

- ***hidden_layer_sizes*** : (50, 50, 25)
- ***activation*** : *tanh*
- ***solver*** : *adam*
- ***alpha*** : 0.001

Com este modelo e parâmetros conseguimos obter um *score* público na competição de 0.86686.

### 3.4.3 Stacking

O modelo de ***stacking*** é um tipo de *ensemble* onde são utilizados vários modelos para fazerem previsões nos dados de treino e de teste. Essas previsões são depois usadas como dados de treino e teste para um ultimo modelo de previsão.

Os modelos utilizados no ***stacking*** foram os seguintes:

- *DecisionTreeClassifier*
- *SupportVectorClassifier* : (C=1, *gamma*=0.01, *kernel*='*linear*')
- *RandomForestClassifier*
- *MLPClassifier* : (*activation*='*tanh*', *alpha*=0.001, *hidden_layer_sizes*=(50, 50, 25))

Estes modelos foram escolhidos por representarem estratégias diversificadas de previsão o que permite ao modelo de *stacking* capturar padrões e relações nos dados mais diversificados, e também é mais resiliente a causar um *overfitting* nos dados de treino.

O modelo utilizado na previsão final foi o *LogisticRegression*. Fizemos testes com outros modelos mas este foi aquele que obteve os melhores resultados para o nosso caso de estudo.

Nesta secção especificamos o melhor modelo de *stacking* que obtivemos a partir dos testes e submissões feitas no *Kaggle*. Foram testadas outras possibilidades de modelos, como por exemplo, 5 *MLPClassifiers* na *stack* ou outros modelos para o modelo final de previsão. No entanto este foi o que obteve um melhor resultado na competição com um *score* público de 0.84467.

### 3.4.4 Max Voting

O modelo de ***Max Voting*** é similar ao ***Stacking*** no aspeto em que possibilita a utilização de vários tipos de modelos, no entanto o seu funcionamento é diferente. O ***Max Voting*** faz as suas previsões através dos

votos dos modelos que está a ser utilizados. Esses votos podem ser contabilizados de 2 formas diferentes. *Hard voting* onde a previsão com o maior número de votos é a previsão escolhida para aquele caso de teste e *Soft Voting* onde a os modelos atribuem as classes de previsão um *score* de confiança (ou probabilidade) e o modelo escolhe a classe que contém a maior probabilidade em média de todos os modelos.

Semelhante ao modelo de ***Stacking*** vários conjuntos de modelos foram utilizados na modelação com ***Max Voting***. Nesta secção apenas vamos apresentar o modelo que resultou no melhor *score* na competição.

Este foram os modelos utilizados:

- *DecisionTreeClassifier*
- *SupportVectorClassifier* : (C=1, *gamma*=0.01, *kernel*='*linear*')
- *RandomForestClassifier*
- *MLPClassifier* : (*activation*='*tanh*', *alpha*=0.001, *hidden_layer_sizes*=(50, 50, 25))

Após alguns testes com estes modelos percebemos que para o nosso caso de estudo o uso de *hard voting* conseguia resultados um pouco melhores do que o *soft voting*. Também exploramos o parâmetro ***weight*** que permite atribuir pesos aos modelos por forma a que os seus votos tenham uma maior importância na decisão final da previsão. Os modelos de *Random Forest* e *MLPClassifier* quando possuíam um maior peso na votação causavam um melhor *score* na competição do que quando o seu peso no voto era igual aos outros modelos.

O melhor *score* público obtido através deste modelo foi de 0.86242, em que os modelos *Random Forest* e *MLPClassifier* tinham um peso 2 vezes superior aos outros modelos e a estratégia de voto foi *hard*.

### 3.4.5 XGBoosting

O modelo de ***XGBoosting*** foi o modelo que conseguiu os melhores resultados para o nosso caso de estudo. Este modelo é um modelo de *boosting* que utiliza o gradiente descendente para otimizar os parâmetros de novos estimadores que são adicionados ao processo de *boosting*.

Para obter os melhores resultados com este modelo foi necessário fazer *hypertunning* dos hiperparâmetros do modelo. Novamente, foi utilizado o *Gridsearh* para encontar os valores ótimos para o nosso caso de estudo. Os parâmetros explorados foram:

- ***learning_rate*** : 0.01
- ***n_estimators*** : 600
- ***max_depth*** : 5
- ***gamma*** : 0.1
- ***min_child_weight*** : 2
- ***colsample_bytree*** : 0.8

Com este valores conseguimos atingir um valor de *score* público de 0.89053.

## 3.5 Resultados obtidos e análise crítica

Para finalizar esta tarefa vamos analisar todo o trabalho feito em relação a esta competição.

Em relação ao tratamento dos dados conseguimos tirar partido de todos os dados fornecidos pelos

docente e através de *feature engineering* conseguimos criar novos atributos para melhorar o *score* dos nossos modelos.

Em relação à nossa performance na competição não achamos que foi o ideal visto que sofremos uma queda na *leaderboard* após a revelação do *score* final. Apesar de termos conseguido um bom resultado com o modelo ***XGBoosting*** com um *score* publico de 0.89053 acabamos por perder algumas posições e baixar o *score* para 0.85949.