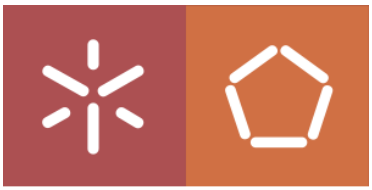


Universidade do Minho
Escola de Engenharia

MESTRADO EM ENGENHARIA INFORMÁTICA

DADOS E APRENDIZAGEM AUTOMÁTICA

TRABALHO PRÁTICO

CONCEÇÃO E OTIMIZAÇÃO DE MODELOS DE *MACHINE LEARNING*

Grupo 07

ANA FILIPA PEREIRA - PG46978

BRUNO SOUSA - PG45577

CAROLINA SANTEJO - PG47102

RAQUEL COSTA - PG47600

RESUMO

O relatório que se segue tem como objetivo relatar o desenvolvimento do projeto da unidade curricular de Dados e Aprendizagem Automática.

Primeiramente, são apresentados os objetivos que a equipa de trabalho se propõe a atingir, e a metodologia seguida ao longo do trabalho prático, bem como a sua aplicação.

De seguida, é feito um relato de todo o processo realizado para cada caso de estudo que a equipa se propôs a analisar durante o projeto. Isto inclui a descrição e exploração detalhada do *dataset* de cada caso de estudo, tal como o tratamento de dados efetuado ao mesmo. Além disso, são apresentados também os modelos preditivos de *Machine Learning* desenvolvidos, sendo que no final é feito um sumário dos resultados obtidos e uma análise crítica aos mesmos, tendo em conta vários aspetos, como por exemplo, a performance de um modelo.

No final, é elaborada uma análise geral sobre o projeto realizado, onde são referidas sugestões e possíveis alterações que melhorassem os resultados obtidos pelo grupo em ambos os casos de estudo.

ÍNDICE

I. INTRODUÇÃO

Os modelos de *Machine Learning* permitem prever acontecimentos futuros através do uso de algoritmos baseados em matemática e estatística. Estes surgem da necessidade de processar e obter conhecimento a partir de dados fornecidos, permitindo, deste modo, a aprendizagem de modo autónomo e independente. Assim sendo, o paradigma de *Machine Learning* através da utilização de diversos algoritmos permite extrair informações a partir de dados brutos e representá-los por meio de um modelo matemático, que mais tarde irá permitir fazer inferências ou previsões a partir de um novo conjunto de dados.

A conceção e o desenvolvimento de um projeto de *Machine Learning* trata-se de um processo que inclui diversas etapas, dependendo da metodologia adotada consoante o caso de estudo que se esteja a analisar. Uma das etapas fundamentais é a análise e tratamento de dados do *dataset* em estudo, uma vez que a qualidade dos dados influencia a qualidade do modelo que será desenvolvido. Para tal é importante visualizar e explorar a distribuição dos vários atributos presentes no *dataset,* de modo a perceber a relação entre eles e o atributo *target*. Deste modo, é então necessário aplicar diversas técnicas de tratamento de dados, com o intuito de obter a melhor performance possível no modelo preditivo a ser desenvolvido.

Por fim, é importante referir que todos os modelos preditivos desenvolvidos ao longo deste processo são testados e analisados através de diversas métricas de avaliação, sendo uma delas a *Accuracy,* permitindo assim testar a precisão preditiva que o modelo desenvolvido tem quando exposto a um certo conjunto de dados. É importante que o modelo consiga reconhecer padrões nos dados não vistos, em vez de apenas memoriar os dados vistos durante o treino.

II. PRINCIPAIS OBJETIVOS

O objetivo principal do presente trabalho prático é elaborar um projeto de *Machine Learning* utilizando os modelos de aprendizagem abordados ao longo do semestre.

Ao longo do projeto foram estudados dois problemas, o primeiro trata-se da previsão do fluxo de tráfego rodoviário na cidade do Porto. Já o segundo problema, escolhido pela equipa, tem como objetivo analisar e prever a taxa de "burnout" dos funcionários com base nas condições que lhes são proporcionadas. Em relação ao primeiro caso de estudo, o grupo fez parte da competição criada pela equipa docente na plataforma Kaggle, o que permitiu que a equipa permanecesse sempre motivada ao longo do desenvolvimento do projeto.

Numa 1ª fase, um dos objetivos da equipa foi realizar um bom tratamento de dados, procurando extrair conhecimento relevante no contexto dos problemas em questão, para que consequentemente os modelos de ML desenvolvidos obtivessem uma melhor performance. Para tal, recorreu-se a várias técnicas de tratamento de dados, tais como a deteção de *outliers*, a verificação de registos duplicados ou de *missing values*, a categorização de *features*, e ainda a *features extraction*.

Já referente a uma 2ª fase, a equipa teve como objetivo desenvolver e otimizar modelos de ML para ambos os problemas estudados, aplicando ao longo do processo diversas métricas de avaliação. Deste modo, é possível interpretar os resultados obtidos e definir a sua utilidade no contexto dos problemas subjacentes aos *datasets* trabalhados.

III. METODOLOGIA

FIGURA 1 - METODOLOGIA PARA EXTRAÇÃO DE CONHECIMENTO

Para desenvolver este projeto de extração de conhecimento foi aplicada uma metodologia baseada naquela que foi utilizada ao longo das aulas da unidade curricular de DAA. Deste modo é importante destacar as 5 diferentes fases representadas na Figura 1:

- **Compreensão do problema**

Nesta primeira fase é necessário perceber os objetivos do projeto e definir o problema de extração de conhecimento.

- **Análise e exploração dos dados**

Esta fase tem como objetivo analisar os dados do *dataset* fornecido, ou seja, é nesta etapa que se pretende encontrar valores inconsistentes ou em falta, relações entre atributos, identificar *outliers*, avaliar a qualidade dos dados, etc.

- **Tratamento dos dados**

Após a análise e exploração dos dados é necessário efetuar o tratamento dos mesmos. Nesta fase são escolhidos os atributos relevantes para o modelo, serão completadas ou retiradas entradas constituídas por valores em falta e, caso haja valores inconsistentes é necessário corrigi-los. Nesta etapa é também fundamental transformar todos os atributos com tipos de dados incompatíveis com o modelo de aprendizagem para tipos compatíveis, utilizando as técnicas correspondentes a cada um.

- **Modelo de extração de conhecimento**

Para extrair o conhecimento dos dados já tratados e analisados, é selecionado e aplicado o modelo de extração que se considere mais adequado para a situação.

- **Avaliação do modelo**

Por fim, para saber se houve sucesso na extração do conhecimento é feita a comparação do modelo obtido com os resultados esperados e assim é possível fazer uma avaliação.

IV. **DATASET 1:** PREVISÃO DO FLUXO DE TRÁFEGO RODOVIÁRIO

A. CONTEXTUALIZAÇÃO

O presente caso de estudo trata-se de um conhecido problema de características estocásticas, não-lineares. Assim sendo, a equipa docente da UC disponibilizou um dataset que contém dados referentes ao tráfego rodoviário na cidade do Porto durante um período superior a um ano.

Além disso, é de salientar que o *dataset* disponibilizado contém um conjunto de *features*, porém, é importante destacar a *average_speed_diff*, que irá indicar através de uma escala, se efetivamente existe trânsito ou não.

Deste modo, o objetivo é desenvolver um modelo que seja capaz de prever o fluxo de trânsito, numa determinada hora, na referida cidade.

B. ANÁLISE E EXPLORAÇÃO DE DADOS

Na fase de análise e exploração dos dados foi feita uma análise detalhada das *features* do *dataset* fornecido e dos seus valores.

1. VALORES EM FALTA

```
data.isna().any()
```

```
record_date                False
AVERAGE_SPEED_DIFF         False
AVERAGE_FREE_FLOW_SPEED    False
AVERAGE_TIME_DIFF          False
AVERAGE_FREE_FLOW_TIME     False
LUMINOSITY                 False
AVERAGE_TEMPERATURE        False
AVERAGE_ATMOSP_PRESSURE    False
AVERAGE_HUMIDITY           False
AVERAGE_WIND_SPEED         False
AVERAGE_CLOUDINESS          True
AVERAGE_PRECIPITATION      False
AVERAGE_RAIN                True
dtype: bool
```

```
percent_missing = data.isna().sum() * 100 / len(data)
missing_value_df = pd.DataFrame({'percent_missing': percent_missing})
missing_value_df
```

	percent_missing
record_date	0.000000
AVERAGE_SPEED_DIFF	0.000000
AVERAGE_FREE_FLOW_SPEED	0.000000
AVERAGE_TIME_DIFF	0.000000
AVERAGE_FREE_FLOW_TIME	0.000000
LUMINOSITY	0.000000
AVERAGE_TEMPERATURE	0.000000
AVERAGE_ATMOSP_PRESSURE	0.000000
AVERAGE_HUMIDITY	0.000000
AVERAGE_WIND_SPEED	0.000000
AVERAGE_CLOUDINESS	39.371697
AVERAGE_PRECIPITATION	0.000000
AVERAGE_RAIN	91.735173

FIGURA 2 - "MISSING VALUES"

Pela análise dos dados foi possível identificar valores em falta nas colunas 'AVERAGE_CLOUDINESS' e 'AVERAGE_RAIN' e a percentagem dos mesmos dentro do atributo em questão.

2. Tipo de Dados dos Atributos

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6812 entries, 0 to 6811
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   record_date              6812 non-null   object
 1   AVERAGE_SPEED_DIFF       6812 non-null   object
 2   AVERAGE_FREE_FLOW_SPEED  6812 non-null   float64
 3   AVERAGE_TIME_DIFF        6812 non-null   float64
 4   AVERAGE_FREE_FLOW_TIME   6812 non-null   float64
 5   LUMINOSITY               6812 non-null   object
 6   AVERAGE_TEMPERATURE      6812 non-null   float64
 7   AVERAGE_ATMOSP_PRESSURE  6812 non-null   float64
 8   AVERAGE_HUMIDITY         6812 non-null   float64
 9   AVERAGE_WIND_SPEED       6812 non-null   float64
 10  AVERAGE_CLOUDINESS       4130 non-null   object
 11  AVERAGE_PRECIPITATION    6812 non-null   float64
 12  AVERAGE_RAIN             563 non-null    object
dtypes: float64(8), object(5)
memory usage: 745.1+ KB
```

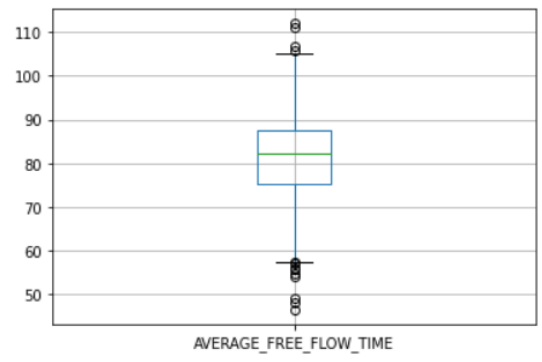
FIGURA 3 - TIPOS DE DADOS DAS COLUNAS

Com recurso ao método *info* foi possível identificar 5 colunas (record_date, AVERAGE_SPEED_DIFF, LUMINOSITY, AVERAGE_CLOUDINESS e AVERAGE_RAIN) com o tipo de dados *object,* que será necessário tratar uma vez que não podem ser utilizadas pelos modelos de aprendizagem.

3. Outliers

```
data.boxplot('AVERAGE_FREE_FLOW_TIME')
```

```
<AxesSubplot:>
```




```
data.boxplot('AVERAGE_FREE_FLOW_SPEED')
```

```
<AxesSubplot:>
```

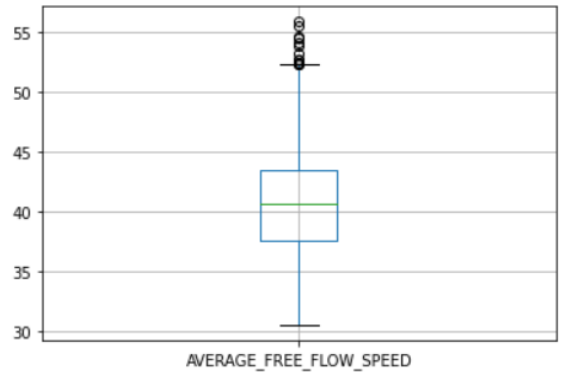


FIGURA 4 – BOXPLOT DO ATRIBUTO AVERAGE_FREE_FLOW_TIME E AVERAGE_FREE_FLOW_SPEED

Gerando o *boxplot* de cada um dos atributos foi possível identificar alguns dos seus *outliers.*

4. RELAÇÕES ENTRE ATRIBUTOS

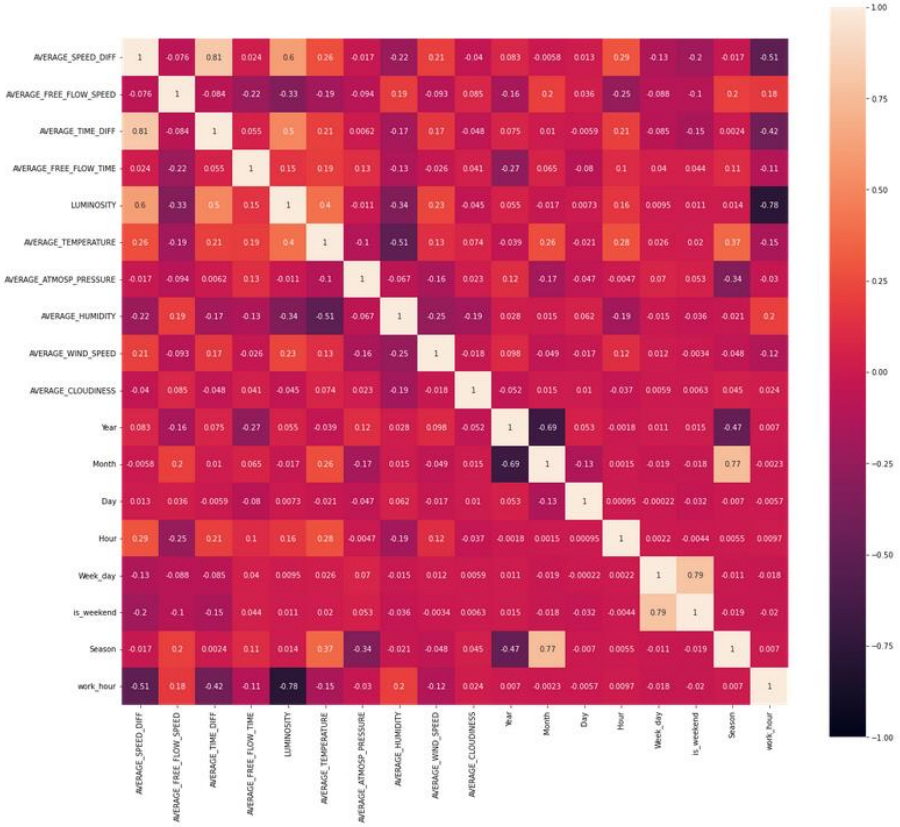


FIGURA 5 - MATRIZ DE CORRELAÇÃO

Gerando uma matriz de correlação foi possível identificar quais os atributos que se relacionam mais entre si. Por exemplo, os pares de colunas 'is_weekend' e 'week_day', 'LUMINOSITY' e 'work_hour', 'Year' e 'Month' têm um alto coeficiente de correlação.

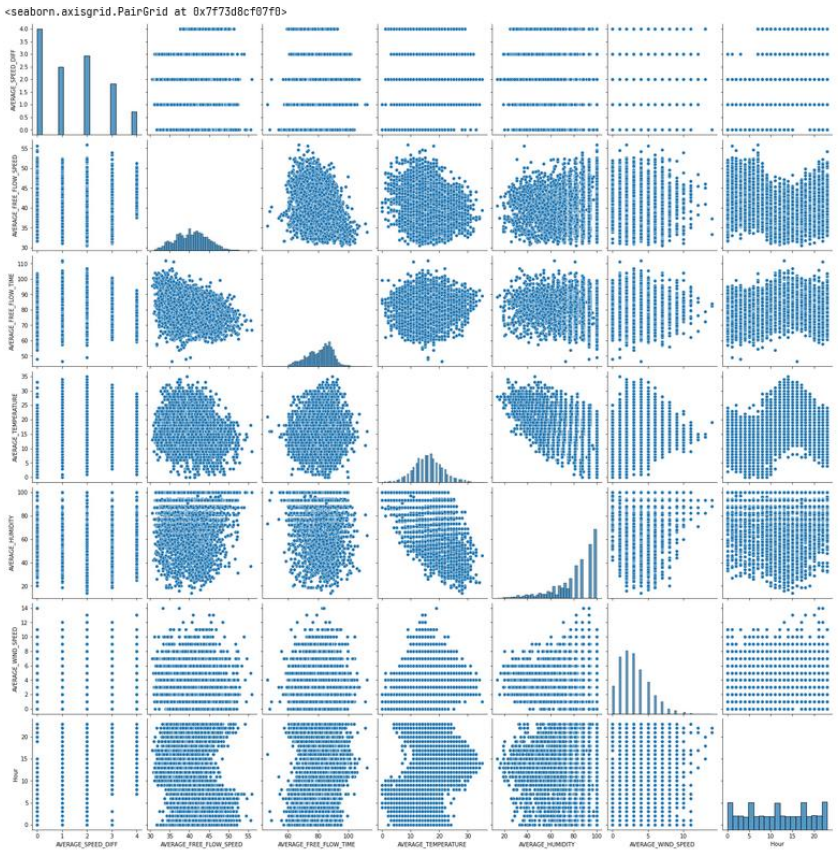


FIGURA 6 - PAIRPLOT

Do mesmo modo, utilizando método *pairplot* da biblioteca seaborn foi possível observar de uma forma ainda mais detalhada a relação entre o comportamento de dois atributos.

5. RELAÇÃO ENTRE ATRIBUTOS E O ATRIBUTO TARGET

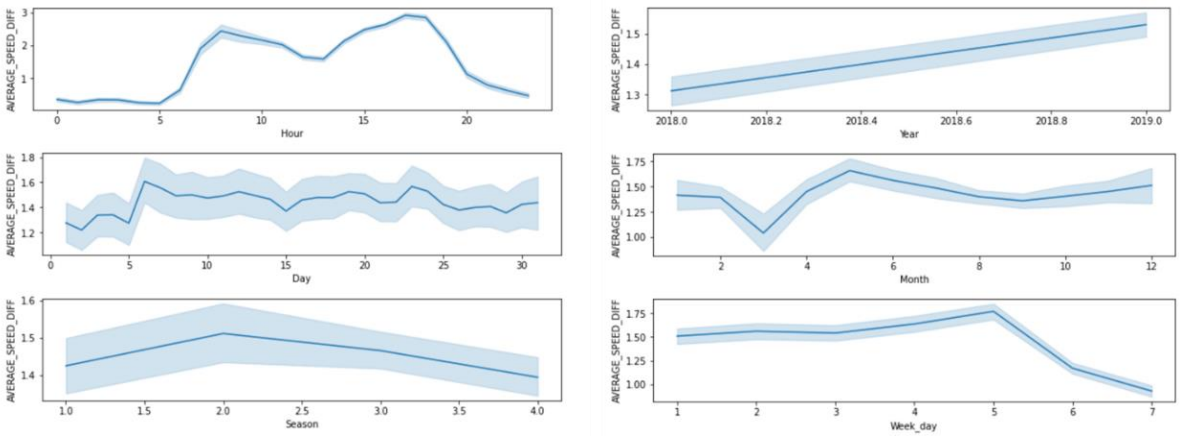


FIGURA 7 - RELAÇÕES ENTRE ATRIBUTOS

Neste *dataset* foi também importante observar o comportamento de certos atributos em relação ao valor do *target*. Gerando um 'lineplot' foi possível concluir que, por exemplo, aos fins de semana os valores do *target* sofrem uma queda em relação aos restantes dias. Do mesmo modo, também se conseguiu identificar as horas do dia em que há mais e menos trânsito.

C. TRATAMENTO DOS DADOS

1. TRATAMENTO DE DADOS "RUIDOSOS" E REDUNDANTES

No *dataset* original foram detetados alguns erros, entre eles um conjunto de caracteres inválidos, para tal foi necessário desenvolver um script em *python* para tratar os mesmos. No seguinte excerto de código é possível visualizar alguns dos caracteres inválidos que foram substituídos pelos corretos.

```
for coluna in lista_colunas:
    df[coluna] = df[coluna].str.replace('Ã£','ã')
    df[coluna] = df[coluna].str.replace('Ã³','ó')
    df[coluna] = df[coluna].str.replace('Ã©','e')
    df[coluna] = df[coluna].str.replace('Ã¢','â')
    df[coluna] = df[coluna].str.replace('Ã§','ç')
    df[coluna] = df[coluna].str.replace('Ãª','ê')
    df[coluna] = df[coluna].str.replace('Ãº','ú')
    df[coluna] = df[coluna].str.replace('Ã¡','á')
    df[coluna] = df[coluna].str.replace('Ã•','Á')
    df[coluna] = df[coluna].str.replace('Ã','í')

return df
```

FIGURA 8 - PEDAÇO DE CÓDIGO DO SCRIPT EM PYTHON

Além disso, foram também encontrados valores, mais precisamente na coluna "AVERAGE_CLOUDINESS" que estavam escritos de forma diferentes apesar de terem o mesmo significado. Exemplos disso são: "Nuvens quebrados" que significa o mesmo que "Nuvens quebradas", neste caso o grupo concluiu que muito provavelmente pode ter sido um erro ortográfico. E, além disso, também foi considerado o seguinte caso: "Céu nublado" que significa o mesmo que "Tempo nublado". Para corrigir isto, o grupo substituiu todos os valores denominados de "Nuvens quebrados" por "Nuvens quebradas" e os valores "Tempo nublado" por "Céu nublado". É importante ainda realçar que de modo a não haver erros em relação a letras maiúsculas ou minúsculas, o grupo colocou todos os valores em letra minúscula.

2. SELEÇÃO DE *FEATURES*

O grupo decidiu, inicialmente, remover 3 colunas do *dataset* original de forma a otimizar a qualidade dos dados, essas são: a AVERAGE_RAIN, a AVERAGE_PRECIPITATION e a CITY_NAME. A coluna AVERAGE_RAIN foi retirada uma vez que grande parte das linhas desta coluna têm valores em falta, mais concretamente existem cerca de 91,7% valores *NaN*, tal como foi possível observar na etapa de exploração e visualização de dados. Já em relação às colunas AVERAGE_PRECIPITATION e CITY_NAME, todas as linhas de cada um destes atributos possuem o mesmo valor, portanto, não há necessidade em manter estas colunas.

```
data = data.drop('AVERAGE_RAIN', axis=1)
data = data.drop('AVERAGE_PRECIPITATION',axis=1)
data = data.drop('city_name',axis=1)
```

FIGURA 9 - CÓDIGO DE SELEÇÃO DE *FEATURES*

3. TRATAMENTO DE *MISSING VALUES*

Além da coluna AVERAGE_RAIN, verificou-se também, pela análise dos dados, que ainda havia outra coluna com uma percentagem superior a 0 de *missing values*, sendo essa coluna a AVERAGE_CLOUDINESS. Esta coluna apresenta cerca de 39% de valores em falta, o grupo considerou que não consistia num valor significativamente alto que justificasse a sua remoção. Portanto, de forma a tratar os dados em falta, foi calculado o valor mais frequente da coluna, substituindo os *NaN* pelo valor obtido, desta forma, os valores adicionados não terão grande influência no comportamento de dados.

```
most_common = data['AVERAGE_CLOUDINESS'].mode()
data["AVERAGE_CLOUDINESS"].fillna(value=most_common[0] , inplace = True)
```

FIGURA 10 - TRATAMENTO DE MISSING VALUES NA AVERAGE_CLOUDINESS

4. *NOMINAL VALUE DISCRETIZATION*

Tal como foi possível observar pela análise e exploração dos dados, existem várias colunas com o tipo de dados *object,* entre elas as colunas AVERAGE_SPEED_DIFF, LUMINOSITY e AVERAGE_CLOUDINESS.

Nas primeiras duas colunas, o grupo decidiu transformar o tipo de dados em categórico ordinal, considerando, desta forma, que estes números inteiros têm uma relação de ordem natural entre si. Para tal, foi criado o seguinte excerto de código:

```
luminosity_mapper = {'DARK': 0, 'LOW_LIGHT': 1, 'LIGHT': 2}
data['LUMINOSITY'] = data["LUMINOSITY"].replace(luminosity_mapper)
```

```
avg_speed_diff_mapper = {'None': 0, 'Low': 1, 'Medium': 2, 'High': 3, 'Very_High': 4}
data['AVERAGE_SPEED_DIFF'] = data["AVERAGE_SPEED_DIFF"].replace(avg_speed_diff_mapper)
```

FIGURA 11 - CÓDIGO DE TRATAMENTO DO TIPO DE DADOS

Já na coluna AVERAGE_CLOUDINESS, foi necessário também proceder a alterações. Com base numa pesquisa efetuada pelo grupo de trabalho, verificou-se a existência da escala de nebulosidade. Sendo que, segundo as normas meteorológicas atuais, a nebulosidade pode ser dividida em: **Céu limpo**, **Céu quase limpo**, **Céu pouco nublado**, **Céu parcialmente nublado**, **Céu quase nublado**, e **Céu nublado**.

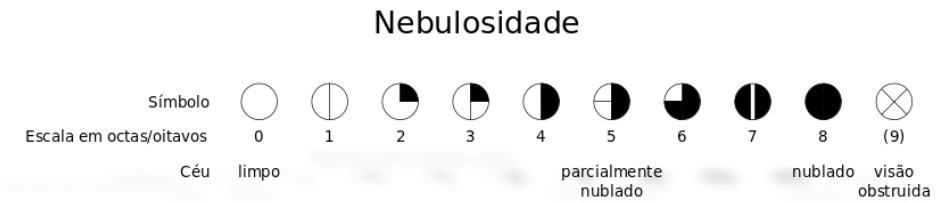


FIGURA 12 - ESCALA DE NEBULOSIDADE

Assim sendo, foi decidido agrupar os valores da coluna AVERAGE_CLOUDINESS em três tipos: 'céu limpo' (3), 'parcialmente nublado' (2) e 'nublado' (1).

```
cloudiness_mapper = {'céu claro': 'céu limpo', 'nuvens quebradas' : 'parcialmente nublado',
'algumas nuvens' : 'parcialmente nublado', 'céu pouco nublado' : 'parcialmente nublado',
'nuvens dispersas':'parcialmente nublado', 'tempo nublado': 'nublado'}
data['AVERAGE_CLOUDINESS'] = data['AVERAGE_CLOUDINESS'].replace(cloudiness_mapper)
```

```
cloudiness_mapper = {'céu limpo': 3, 'parcialmente nublado': 2, 'nublado': 1}
data['AVERAGE_CLOUDINESS'] = data["AVERAGE_CLOUDINESS"].replace(cloudiness_mapper)
```

FIGURA 13 - TRATAMENTO DOS DADOS DA COLUNA "AVERAGE_CLOUDINESS"

Desta forma, foi possível estabelecer uma relação entre os valores deste atributo.

5. *FEATURE ENGINEERING*

Além das colunas vistas anteriormente, existem outras duas colunas cujo tipo de dados é *object,* sendo essas a coluna record_date e AVERAGE_CLOUDINESS.

Em relação à coluna record_date, através desta mesma, o grupo conseguiu criar novos *features* (atributos). Sendo assim, foi então possível extrair novos atributos tais como o Ano, o Mês, o Dia, a Hora, o Dia da Semana, e entre outros. Foi ainda criado um atributo que explicitava se o dia em questão tratava-se de um dia que fazia parte do fim de semana ou não ("is_weekend"), uma vez que, tal como observamos na etapa de exploração dos dados, o trânsito era superior em "work days" ao contrário do que se passava durante o fim de semana. Além disso, foram também extraídos os atributos "Season" e "is_holiday", desta forma conseguimos explorar se a estação do ano tinha grande impacto no tráfego rodoviário, bem como explorar a influência dos dias correspondentes a feriados nacionais.

```
data.record_date = pd.to_datetime(data.record_date)
data['Year'] = data.record_date.dt.year
data['Month'] = data.record_date.dt.month
data['Day'] = data.record_date.dt.day
data['Hour'] = data.record_date.dt.hour
data['Week_day'] = data.record_date.dt.strftime("%A")
data["is_weekend"] = data.record_date.dt.day_name().apply(lambda x : 1 if x in ['Saturday','Sunday'] else 0)


def season(month):
    if month in [12,1,2] :
        return 1
    elif month in [3,4,5]:
        return 2
    elif month in [6,7,8]:
        return 3
    else:
        return 4

data["Season"] = data["Month"].apply(season)
```

```
import holidays

# Select country
pt_holidays = holidays.Portugal()

def holiday(dateTime):
    if dateTime in pt_holidays :
        return 1
    else:
        return 0

data["is_holiday"] = data["record_date"].apply(holiday)

data.drop('record_date', axis=1, inplace=True)
```

FIGURA 14 - TRATAMENTO DA COLUNA "RECORD_DATE"

De acordo o '*lineplot*' apresentado entre a relação entre o atributo target e o atributo 'Hour', o grupo decidiu extrair um atributo que fosse capaz de dividir o dia em partes consoante as horas, uma vez que, foi possível observar que em horas de ponta o trânsito era maior. Para tal, fez-se o seguinte:

```
def daypart(hour):
    if hour>=20 and hour<7 :
        return 1
    elif hour >= 7 and hour <= 11:
        return 2
    elif hour > 11 and hour < 15:
        return 3
    elif hour >= 15 and hour < 20:
        return 4
    else :
        return 5

data["work_hour"] = data["Hour"].apply(daypart)
```

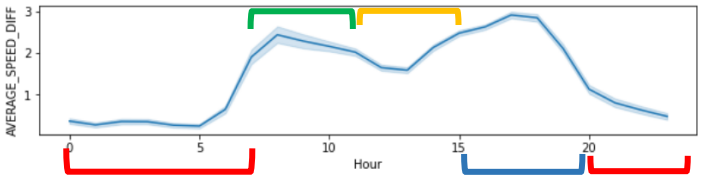


FIGURA 15 - EXTRAÇÃO DO ATRIBUTO 'WORK_HOUR'

Por fim, o grupo voltou a explorar e a analisar os dados, agora com os novos atributos extraídos, verificou-se então que as features 'Season', 'Year' e 'is_holiday', não tinham grande impacto em relação ao atributo target, tal como se pode verificar na figura seguinte, não existe grande variação nos valores de AVERAGE_SPEED_DIFF.

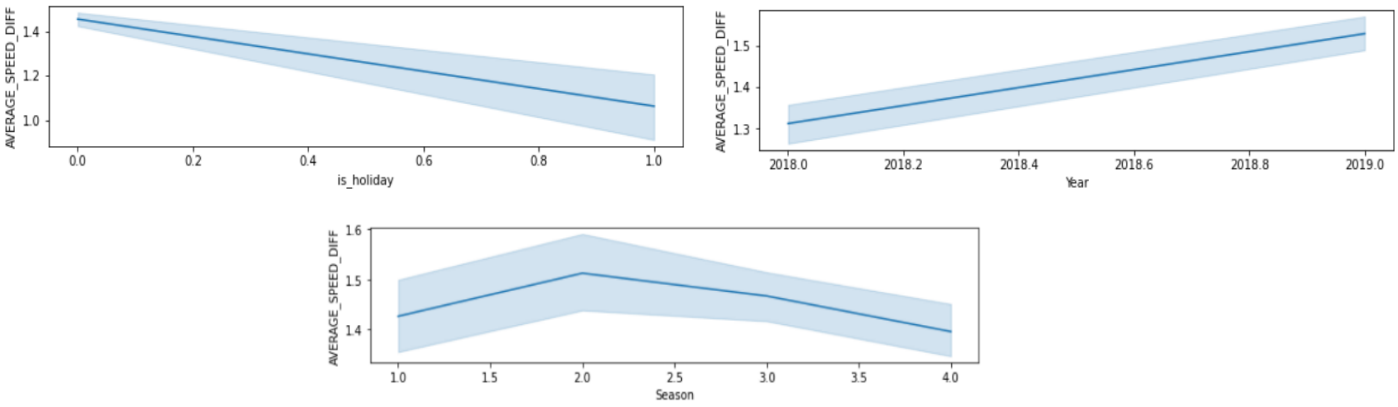


FIGURA 16 - VISUALIZAÇÃO DOS NOVOS ATRIBUTOS

Assim sendo, foi então decidido pelo grupo retirar estas três colunas.

```
data.drop(['is_holiday','Year','Season'],axis = 1,inplace = True)
```

FIGURA 17 - REMOÇÃO DE COLUNAS

6. DETEÇÃO DE *OUTLIERS*

Através da análise dos dados e de gráficos *boxplot* verificou-se que existiam *outliers* em atributos tais como *Average_Free_Flow_Time, Average_Temperature*, entre outros.

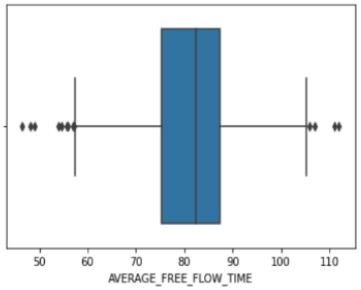


FIGURA 19 - BOXPLOT DO ATRIBUTO AVERAGE_FREE_FLOW_TIME

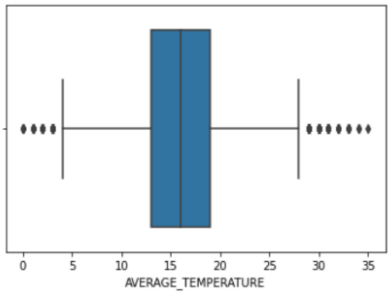


FIGURA 18 - BOXPLOT DO ATRIBUTO AVERAGE_TEMPERATURE

O tratamento destes *outliers*, poderia ser feito recorrendo a várias metodologias como por exemplo a sua remoção. No entanto, e visto que isto leva à perda de informação, o grupo optou por utilizar outro método designado por *percentile capping.* Esta metodologia, consiste em mudar o valor dos *outliers* mais extremos para valores mais próximos dos verificados no resto do *dataset* de forma a atenuar o seu impacto. Os valores para os quais os *outliers* são mudados são os dos percentis, por exemplo 1% e 99%. No pedaço de código abaixo, mostra-se como se efetuou o flooring e capping dos *outliers* do atributo *Average_Free_Flow_Time* utilizando o percentil 99% e 1%.

```
lower = data['AVERAGE_FREE_FLOW_TIME'].quantile(0.01)
upper = data['AVERAGE_FREE_FLOW_TIME'].quantile(0.99)


data['AVERAGE_FREE_FLOW_TIME'] = np.where(data['AVERAGE_FREE_FLOW_TIME'] <lower, lower,data['AVERAGE_FREE_FLOW_TIME'])
data['AVERAGE_FREE_FLOW_TIME'] = np.where(data['AVERAGE_FREE_FLOW_TIME'] >upper, upper,data['AVERAGE_FREE_FLOW_TIME'])
```

FIGURA 20-FLOORING E CAPPING DO AVERAGE_FREE_FLOW_TIME

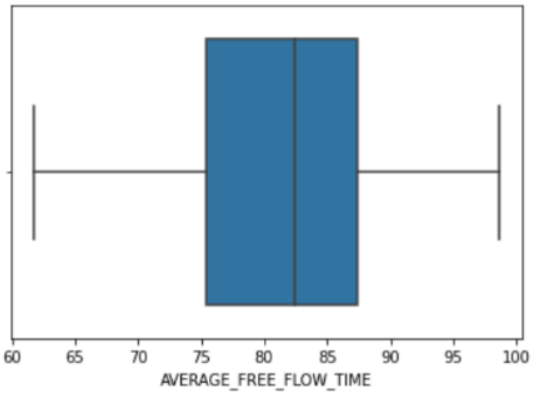


FIGURA 21-BOXPLOT DO ATRIBUTO AVERAGE_FREE_FLOW_TIME APÓS FLOORING E CAPPING

D. MODELOS DESENVOLVIDOS

Após ser feita a análise detalhada dos dados bem como o seu tratamento, passa-se à fase de desenvolvimento do modelo de previsão. Esta fase é bastante importante uma vez que não só se pretende obter um modelo que seja capaz de representar o *dataset* em questão, mas, que também possa fazer previsões corretas a partir de dados desconhecidos. Além disto, o modelo a desenvolver não pode ser demasiado simples de forma a evitar *underfitting*, mas também não deverá ser demasiado "perfeito", ou seja, funciona muito bem para o *dataset* de treino, mas falha ao fazer previsões com dados que nunca viu. A isto designa-se *overfitting*.

Assim sendo, o grupo começou por efetuar a divisão dos dados em dados de treino e de teste, recorrendo-se ao *train_test_split*. Decidiu-se que 20% do *dataset* seria usado para testar o modelo e os restantes 80% para o treinar.

```
X = data.drop('AVERAGE_SPEED_DIFF',axis=1)
y = data['AVERAGE_SPEED_DIFF']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2021)
```

Desta forma, e tendo em conta que o problema em questão se trata de um problema de classificação e que nos encontramos no âmbito da aprendizagem supervisionada, o grupo começou por testar vários modelos e avaliar o seu desempenho.

1. DECISION TREE

Inicialmente, o grupo decidiu recorrer a uma *Decision Tree* para resolver o problema, visto que este modelo foi bastante abordado nas aulas da unidade curricular.

As árvores de decisão estão entre os métodos mais usados em *Machine Learning*. Estas árvores são treinadas de acordo com um conjunto de dados de treino e posteriormente, outros exemplos serão classificados de acordo com essa mesma árvore.

Em *Python*, foi usado o modelo *DecisionTreeClassifier* da biblioteca *Sklearn*.

Assim sendo, definiu-se um modelo ao qual se deu o nome de *clf*. É de realçar que o grupo testou este modelo com vários valores para o hiperparâmetro *max_depth*, e verificou que o melhor resultado era quando este tinha o valor 6.

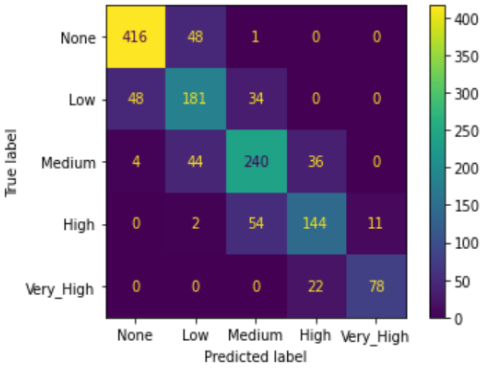
Desta forma, foi feita a previsão da árvore para os dados de teste.

```
rfc=RandomForestClassifier(n_estimators = 265, n_jobs = 2,random_state=2021)
rfc.fit(X_train,y_train)
pred_dtc = clf.predict(X_test)
```

Posto isto, procedeu-se à análise da avaliação do modelo desenvolvido.

Através da análise da matriz de confusão abaixo, podemos ver que valores foram corretamente e erradamente classificados pela árvore de decisão. Além disto, o *classification report* indica o valor de outras métricas tais como a precisão ou *f1-score* para cada valor.

Por outro lado, em termos de *accuracy* foi obtido um valor de 77.67%, sendo este um resultado razoável.




```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       465
           1       0.68      0.67      0.68       263
           2       0.77      0.76      0.76       324
           3       0.72      0.77      0.74       211
           4       0.95      0.75      0.84       100

    accuracy                           0.79      1363
   macro avg       0.80      0.77      0.78      1363
weighted avg       0.80      0.79      0.79      1363
```

FIGURA 22 - ANÁLISE FINAL DECISION TREE

É importante realçar que, embora o algoritmo das *decision trees* seja simples e flexível, este também é *greedy*, não se preocupando com o impacto que um dado *split* pode ter em toda a árvore. Isto leva a que *decision trees* sejam propícias a *overfitting,* ou seja, obtém-se modelos de aprendizagem com alta variância.

2. RANDOM FOREST

Em alternativa às árvores de decisão, o grupo decidiu recorrer a métodos ensemble nos quais está incluído o algoritmo *randomForest*. Este algoritmo consiste em criar várias *"mini" decision trees* combinando os seus resultados de forma a efetuar uma previsão final.

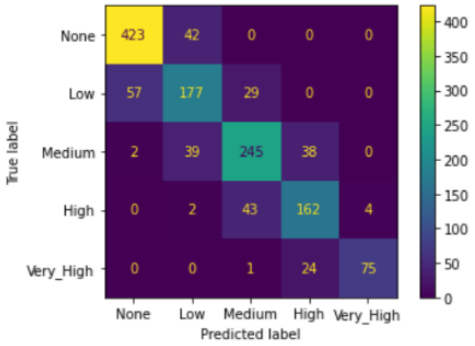
Uma das vantagens das *random forests* é que elas atenuam um grande problema das árvores de decisão e que foi referido anteriormente: a variância.

Assim sendo, começou-se por definir um modelo denominado *rfc*. É de realçar que o grupo testou este modelo com vários valores para o parâmetro *n_estimators*, e verificou que o melhor resultado era quando este tinha o valor 265.

```
rfc=RandomForestClassifier(n_estimators = 265, n_jobs = 2,random_state=2021)
rfc.fit(X_train,y_train)
red_rfc = rfc.predict(X_test)
```

Posto isto, procedeu-se à análise da avaliação do modelo desenvolvido. Com a *randomForest* foi possível obter uma accuracy de 79.3%.

Além disto, e assim como foi feito no tópico das *decision trees*, recorreu-se à matriz de confusão e ao *classification report*.




```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       465
           1       0.68      0.67      0.68       263
           2       0.77      0.76      0.76       324
           3       0.72      0.77      0.74       211
           4       0.95      0.75      0.84       100

    accuracy                           0.79      1363
   macro avg       0.80      0.77      0.78      1363
weighted avg       0.80      0.79      0.79      1363
```

FIGURA 23 - ANÁLISE FINAL RANDOM FOREST

Após serem analisados os valores obtidos, é possível afirmar que, para este problema em concreto, a *random forest* produz melhores resultados que a *decision tree*. Aliás, a *random forest* consegue generalizar melhor que as árvores de decisão. No entanto, é importante salientar que apesar destas vantagens, esta metodologia exige mais recursos computacionais.

3. MODEL TUNING COM GRIDSEARCHCV

Uma vez que se verificou que a *random forest* produz melhores resultados que a árvore de decisão, o grupo decidiu proceder à otimização dos seus hiperparâmetros recorrendo ao *GridSearchCV*, metodologia esta, abordada nas aulas da unidade curricular. Em primeiro lugar, é necessário definir o modelo que onde queremos fazer a otimização. Como já foi dito, este será uma *random forest.* Após isto, e dado que o GridSearchCV recorre ao *cross validation,* é necessário definir também um *cross validator* que vai dividir o *dataset* em K *folds* (no nosso caso foram definidos 3 *folds*). O *cross validator* escolhido foi o *StratifiedKFold* uma vez que este preserva tem a distribuição original dos atributos do *dataset*. Após isto é definido um dicionário designado *space* com os hiperparâmetros que se pretende otimizar bem como os valores que se pretende testar. Os hiperparamentros selecionados foram o *n_estimators* que define o número de árvores de decisão que a *random forest* vai usar, e o *max_features* que define o número máximo de *features* que serão usados em cada *decision tree*. Posto isto, efetua-se o *gridSearchCV*, no qual se passa se define também a métrica que pretendemos otimizar, sendo esta no nosso caso, a *accuracy*.

Após isto, utiliza-se o atributo *best_estimator_* para obtermos a melhor combinação de *hiperparametros* calculados.

```
model = RandomForestClassifier(random_state=2021)
cv_inner = StratifiedKFold(n_splits = 3, shuffle=True,random_state=2021)
```

```
space = dict()
space['n_estimators'] = [205,220,265,267,270]
space['max_features'] = [2,3,4,5,6,7,8,9,10,12]
```

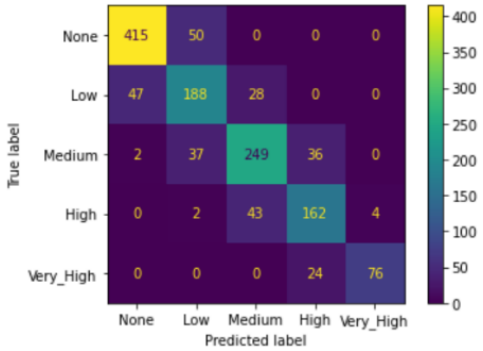
```
search = GridSearchCV(model,space,scoring='accuracy',n_jobs=2, cv=cv_inner, refit=True)
result = search.fit(X_train, y_train)
```

Tal como é possível verificar, a melhor combinação calculada foi *max_features* = 9 e *n_estimators* = 205.

```
best_model = result.best_estimator_
best_model
```

```
RandomForestClassifier(max_features=9, n_estimators=205, random_state=2021)
```

Por fim, é feita a avaliação do modelo obtido. Verificou-se que se obteve uma *accuracy* melhor, esta com um valor de 79.9%. Além disto, e assim como foi feito anteriormente, recorremos também à matriz de confusão e ao *classification report*.




```
=== Classification Report ===
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       465
           1       0.68      0.71      0.70       263
           2       0.78      0.77      0.77       324
           3       0.73      0.77      0.75       211
           4       0.95      0.76      0.84       100

    accuracy                           0.80      1363
   macro avg       0.81      0.78      0.79      1363
weighted avg       0.80      0.80      0.80      1363
```

FIGURA 24 - ANÁLISE FINAL GRIDSEARCHCV

Desta forma, conclui-se que com a otimização de hiperparâmetros foi possível na generalidade, obter melhores resultados.

E. RESULTADOS E ANÁLISE

Tal como já foi referido, foram testados vários modelos de forma a determinar aquele que produziria melhores resultados. Após se ter procedido às suas respetivas análises determinou-se que o algoritmo *random forest* com otimização de parâmetros obteve na generalidade os resultados mais promissores. No entanto, e dado que decorreu uma competição no *Kaggle*, foi possível testar o modelo desenvolvido com dados aos quais nunca teve acesso. Estes dados encontravam-se disponíveis no ficheiro de teste fornecido no início da competição.

Após ter-se submetido no *Kaggle* o ficheiro com as previsões feitas pelo modelo, o grupo obteve uma *accuracy* de 86.4% no *leaderboard* público e 80.380% no *leaderboard* privado tendo conquistado no público o segundo lugar e no privado o quarto lugar. Estes resultados permitem afirmar que o modelo desenvolvido consegue prever de forma correta valores a partir de dados que nunca viu e o facto de que a diferença entre a *accuracy* do *leaderboard* publico e do *leaderboard* privado não ser muito grande indica que o modelo não tem muita tendência a *overfitting*.

Assim sendo, o grupo considera que fez um bom trabalho neste *dataset*, no entanto considera também que poderão se efetuadas melhorias, nomeadamente serem testados novos modelos, otimizar mais hiperparâmetros ou tentar novas metodologias na fase de tratamento de dados.

V. DATASET 2: ANÁLISE DA TAXA DE "BURNOUT" DOS FUNCIONÁRIOS

A. CONTEXTUALIZAÇÃO

Nos últimos tempos, a saúde mental no trabalho tornou-se num tema de interesse para muitos profissionais e empresas, sendo que de acordo com os dados da OMS (Organização Mundial da Saúde) os transtornos mentais e comportamentais são uma das principais causas para o afastamento do trabalho. Isto demonstra que a nossa sociedade está a evoluir, mostrando-se predisposta a discutir este tema abertamente, e a valorizar a saúde mental de cada pessoa.

Durante a situação pandémica que ainda está a ser atravessada, muitas pessoas viram-se obrigadas a trabalhar remotamente, isto fez com que muitas delas tivessem que mudar completamente a sua rotina e a adotar outro tipo de estilo de vida. De acordo com os resultados do estudo "Saúde Mental em Tempos de Pandemia (SM-COVID19)", uma grande percentagem de trabalhadores apresenta sintomas moderados a graves de ansiedade, depressão e *burnout*. Além disso, verificou-se também que entre os profissionais de saúde, estes são aqueles cujos níveis de *burnout* (exaustão física e emocional) são mais elevados. Por outro lado, outros resultados deste estudo sugerem que algumas novas formas de trabalho, nomeadamente o teletrabalho, são vistas de forma positiva por alguns trabalhadores. Será então o trabalho em casa vantajoso ou prejudicial?

Assim sendo, o objetivo do grupo, com este caso de estudo, é compreender como é que a taxa de *burnout* dos funcionários é afetada com base nas várias condições fornecidas, tendo em conta a situação pandémica destes últimos tempos. Deste modo, o objetivo é desenvolver um modelo ML que consiga prever qual será a taxa de *burnout* de um funcionário com base nas condições colocadas pela atual pandemia.

B. ATRIBUTOS/FEATURES DO DATASET ORIGINAL

	Tipo	Exemplo(s)	Descrição
Employee ID	Hexadecimal	Ex: fffe390032003000	ID único atribuído a cada funcionário
Date of Joining	Data	Ex: 2008-12-30	Data em que o funcionário se juntou à empresa
Gender	Categórico Nominal	Male/Female	Género
Company Type	Categórico Nominal	Service/Product	Tipo da empresa do funcionário
WFH Setup Available	Categórico Nominal	Yes/No	Existem sistemas que permitam o trabalho a partir de casa?
Designation	Numérico Discreto	{0, 1, 2, 3, 4, 5}	Escalão do funcionário dentro da sua empresa

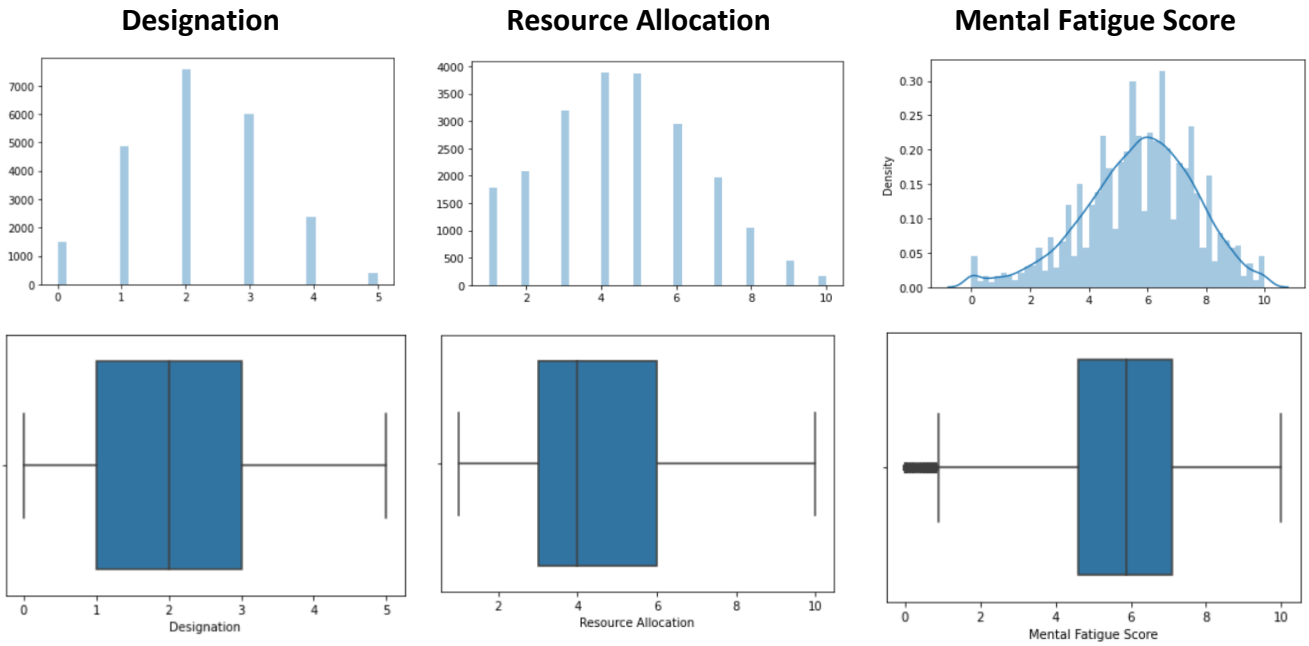
Resource Allocation	Numérico Discreto	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10)	Número de horas de trabalho por dia alocadas
Mental Fatigue Score	Numérico Contínuo	[0, 10]	Nível de fadiga mental que o funcionário está a enfrentar
Burn Rate	Numérico Continuo	[0, 1]	Atributo alvo: taxa de *burn out* associada ao funcionário

C. ANÁLISE E EXPLORAÇÃO DE DADOS

A exploração e visualização recorrentes das diferentes relações entre os atributos e a análise dos dados registados permitiram aplicar subsequentemente determinadas transformações/alterações de tratamento de dados que tiveram um impacto bastante satisfatório e notável ao nível dos resultados produzidos. Esta fase de análise permitiu-nos ainda chegar a algumas conclusões interessantes quanto à carga de trabalho e ao escalão de cada trabalhador e as implicações destas ao nível da taxa de *burnout* esperado.

1. VISUALIZAÇÃO DOS DADOS

TABLE 1- EXEMPLOS DE GRÁFICOS ASSOCIADOS À VISUALIZAÇÃO DA DISTRIBUIÇÃO DOS DADOS RELATIVOS, RESPECTIVAMENTE, AOS ATRIBUTOS DESIGNATION, RESOURCE ALLOCATION E MENTAL FATIGUE SCORE



Os registos dos três atributos apresentados acima, assim como de outras *features* não mencionadas, adoptam consistentemente uma distribuição normal. A densidade dos dados é tendencialmente superior nos valores perto da média.

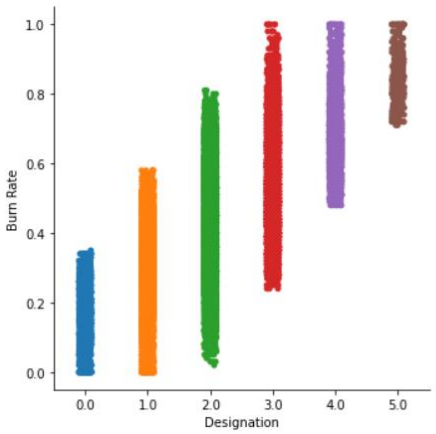


FIGURA 25- BURN RATE EM FUNÇÃO DE DESIGNATION

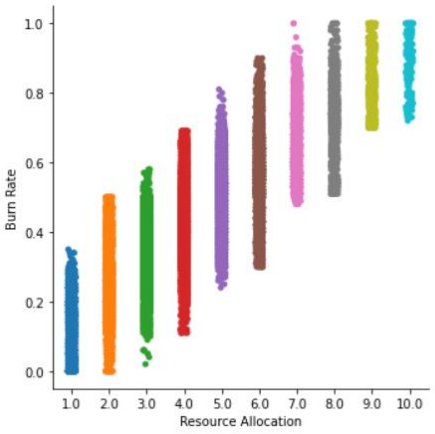


FIGURA 26- BURN RATE EM FUNÇÃO DE RESOURCE ALLOCATION

2. RELAÇÕES ENTRE ATRIBUTOS

A visualização de certas relações entre alguns pares de atributos mostrou-se também interessante e proveitosa, permitindo-nos chegar a algumas asserções pertinentes ao contexto em estudo:

As interpretações mais evidentes daqui retiradas serão:

1. Funcionários/Trabalhadores com designações/cargos/escalões superiores tendem a apresentar uma taxa de *burnout* mais elevada em comparação com os de designação inferior.
2. Funcionários/Trabalhadores com mais horas de trabalho tendem a apresentar uma taxa de *burnout* superior quando comparados com trabalhadores que perfazem menos horas de trabalho.

Existe outra relação que nos permite verificar e confirmar a ocorrência de outro fenómeno:

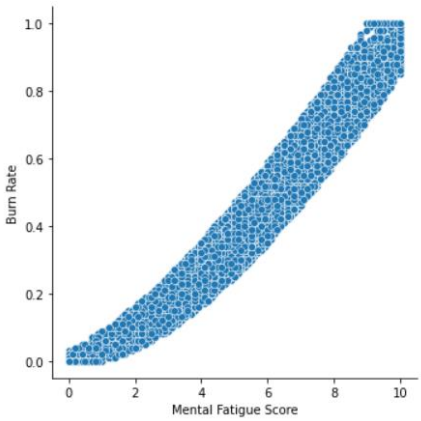


FIGURA 27 - BURN RATE EM FUNÇÃO DE MENTAL FATIGUE SCORE

Verifica-se que o grau de fadiga/cansaço mental indicado por cada funcionário não deve/pode ser utilizado consistentemente como um indicador objetivo e preciso do nível ou taxa de *burnout*. De facto, percebe-se que muitos poucos funcionários foram realmente capazes de aferir, com exatidão, o valor mais correto correspondente ao seu estado mental.

Esta exatidão na determinação do Mental Fatigue Score deverá, previsivelmente, ter grande influência na previsão correta do atributo alvo, tais como os dois atributos anteriormente mencionados *(Resource Allocation* e *Designation*). Podemos suportar esta intuição recorrendo também à visualização de uma matriz de correlação:

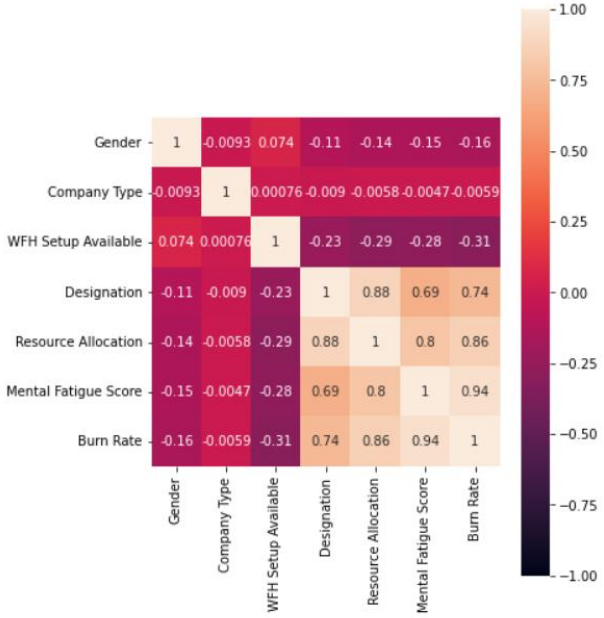


FIGURA 28 - MATRIZ DE CORRELAÇÃO ASSOCIADA AOS DADOS

D. TRATAMENTO DOS DADOS

1. SELECÇÃO DE *FEATURES*

O *dataset* original continha uma coluna com informação redundante. Nomeadamente, existia uma coluna denominada Employee ID, que foi removida por conter os identificadores associados a cada trabalhador, e que isso não continha qualquer significado estatisticamente relevante.

Além disto, verificou-se que existiam entradas incompletas: as colunas Burn Rate, *Resource Allocation* e *Mental Fatigue Score* continham registos sem valor corretamente definido (NaN). Tendo em conta que, em cada coluna, o número de ocorrências desses registos perfazia menos de 1% da totalidade dos registos, decidiu-se enveredar por um processo de tratamento mais simples, procedendo apenas à remoção destas entradas.

Este processo de seleção terminou com a decisão de remover a coluna Company Type. Partindo da observação da matriz de correlação anterior, e intuindo que que o atributo Company Type poderia não ser efetivamente relevante para os modelos de aprendizagem (grau de correlação baixo com todas as outras *features*), procedeu-se à formulação de um modelo de aprendizagem (regressão linear) que não fizesse recurso a esta coluna. Os resultados obtidos confirmaram a intuição inicial: o modelo obtido é capaz de obter os mesmos resultados (erro de previsão semelhante) sem recorrer a este atributo, revelando-se mais eficiente ao nível da utilização de informação.

2. Conversão e Transformação

Inicialmente, os registos associados aos atributos Gender, Company Type e WFH Setup Available eram de natureza nominal. Tendo em vista utilizar estes atributos em modelos de aprendizagem, procedeu-se à conversão dos dados através de uma codificação, fazendo corresponder cada elemento das classes binárias a um elemento em {0, 1}:

```
gender_mapper = {'Male':0, 'Female':1}
data['Gender'].replace(gender_mapper, inplace=True)

company_type_mapper = {'Service': 0, 'Product': 1}
data['Company Type'].replace(company_type_mapper, inplace=True)

wfh_mapper = {'No': 0, 'Yes': 1}
data['WFH Setup Available'].replace(wfh_mapper, inplace=True)
```

No entanto, reconhecendo que esta codificação poderia influenciar o processo de aprendizagem, possivelmente levando os modelos a inferir, erradamente, relações de ordem entre os dados, decidiu-se recorrer a *one-hot encoding:*

Gender_Female	Gender_Male	WFH Setup_No	WFH Setup_Yes
1	0	1	0
0	1	0	1
0	1	0	1
1	0	1	0

Figura 29 - Colunas resultantes da aplicação de codificação one-hot nos atributos Gender e WFH Setup Available

Verificou-se mais tarde que esta técnica de codificação permite obter previsões ligeiramente mais corretas.

E. Modelos Desenvolvidos

1. Regressão Linear

O primeiro modelo formulado recorre a 7 atributo de input (sendo que 4 colunas dizem respeito às *features* Gender e WFH Setup Available), e foi treinado de forma a poder produzir previsões quanto à taxa de burnout Burn Rate.

	Designation	Resource Allocation	Mental Fatigue Score	Gender_Female	Gender_Male	WFH Setup_No	WFH Setup_Yes
0	2.0	3.0	3.8	1.0	0.0	1.0	0.0
1	1.0	2.0	5.0	0.0	1.0	0.0	1.0
3	1.0	1.0	2.6	0.0	1.0	0.0	1.0
4	3.0	7.0	6.9	1.0	0.0	1.0	0.0
5	2.0	4.0	3.6	0.0	1.0	0.0	1.0

Figura 30 - Exemplo de dados de treino/teste do modelo de regressão linear

O processo de implementação, treino e de obtenção de previsões associado ao modelo é extremamente simples:

```
lrm = LinearRegression()        # Criar instância modelo de regressão linear
lrm.fit(X_train, y_train)       # Treinar o modelo
predictions = lrm.predict(X_test) # Obter previsões para os dados de teste
```

2. Gradient Boosting

Tendo em vista experimentar outros modelos de regressão, e na tentativa de determinar uma combinação de hiperparâmetros que permitissem optimizar o processo de aprendizagem, decidiu-se também implementar um modelo de **Gradient Boosting** parametrizado através de **Grid Search**.

Utilizando os mesmos dados do modelo anterior, começa-se pela criação do modelo e a definição do intervalo de valores a testar em cada parâmetro:

```
1.  model = GradientBoostingRegressor()
2.  params = {
3.      'learning_rate': [0.01,0.02,0.03,0.04],
4.      'subsample'    : [0.1, 0.2, 0.4, 0.8],
5.      'n_estimators' : [128,256,512],
6.      'max_depth'    : [4,8,16]
7.  }
```

O processo de treino recorre à procura exaustiva pela melhor combinação de hiperparâmetros (com *5-fold cross validation*, por omissão):

```
1.  gridCV =  GridSearchCV(estimator=model, param_grid = params, n_jobs=-1)
2.  gridCV.fit(X_train, y_train)
```

3. Rede Neuronal

Seguindo um raciocínio idêntico ao aplicado durante as aulas pratico-laboratoriais, optou-se por construir um modelo de redes neuronais sequencial composto por três camadas (uma camada de input com 16 nodos e outra camada de output com apenas um nodo). As instruções de construção deste modelo são:

```
1.  def build_model(activation='relu', learning_rate=0.01):
2.      model = Sequential()
3.      model.add(Dense(16, input_dim=7, activation=activation))
4.      model.add(Dense(8, activation=activation))
5.      model.add(Dense(1, activation=activation))
6.
7.      model.compile(
8.          loss='mae',
9.          optimizer=tf.optimizers.Adam(learning_rate),
10.         metrics=['mae', 'mse']
11.     )
12.
13.     return model
```

Cada nodo da camada intermédia está ligado a todos os 7 nodos da camada de input

O objectivo principal é a minimização do erro médio absoluto (mean absolute error)

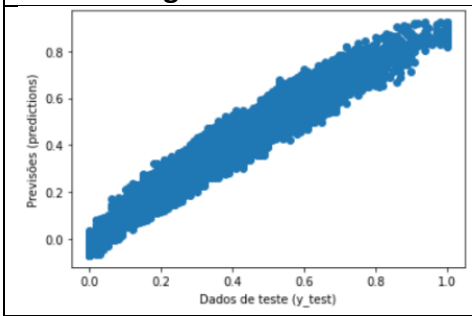
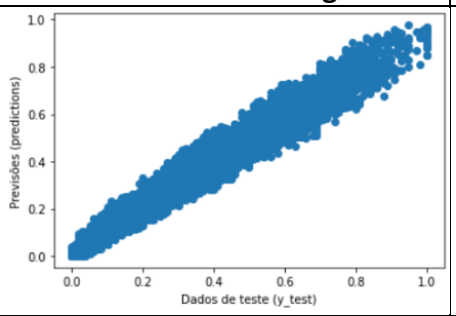
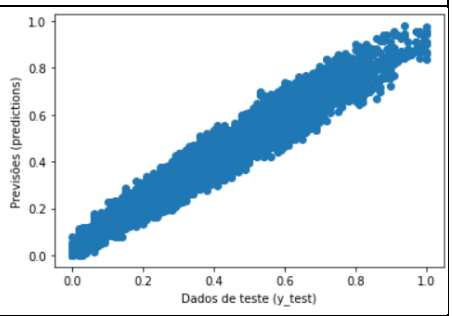
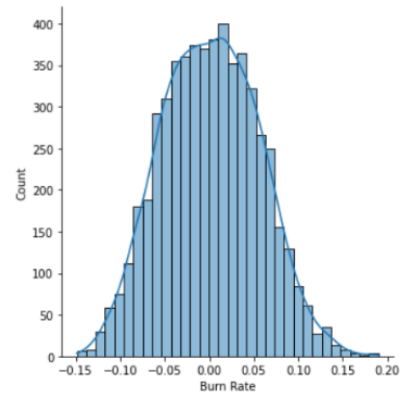
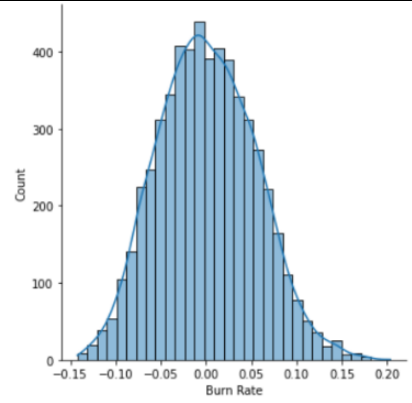
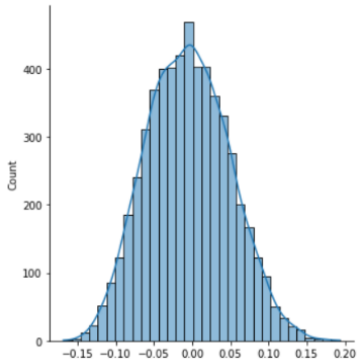
Finalmente, recorre-se novamente à procura exaustiva pela configuração óptima de parâmetros:

```
1.  kfold = KFold(n_splits=5, shuffle=True, random_state=2021)
2.  paramGrid = {
3.      'activation': ['relu', 'sigmoid'],
4.      'learning_rate': [0.01, 0.02, 0.03]
5.  }
6.
7.  model = KerasRegressor(build_fn=build_model, epochs=100, batch_size=128)
8.  grid_search = GridSearchCV(estimator=model, param_grid=paramGrid, cv=kfold,
    scoring='neg_mean_absolute_error', refit=True, verbose=1)
9.
10. grid_search_result = grid_search.fit(X_train, y_train, validation_split=0.3,
    verbose=1)
```

Permite-se a utilização de uma função não linear: a rede neuronal tem apenas três camadas, por isso não existe grande risco de retropropagação de erros

F. RESULTADOS E ANÁLISE

Os três modelos implementados tendem a produzir previsões com grau aproximado de exatidão.

Regressão Linear	Gradient Boosting	Rede Neuronal
		
		
MAE: 0.0447 MSE: 0.0030 RMSE: 0.0546	MAE: 0.0435 MSE: 0.0029 RMSE: 0.0536	MAE: 0.0440 MSE: 0.0029 RMSE: 0.0541

No entanto, o tempo de execução necessário para treinar cada modelo é, neste exemplo, o factor determinante na avaliação do desempenho global de cada um.

O modelo de **regressão linear** foi capaz de produzir estas previsões através de um processo de treino que necessitou menos de 1 segundo de execução. Tendo isto em conta, e sabendo que as outras duas técnicas de aprendizagem necessitaram de 3 a 10 minutos de execução

(aproximadamente), percebe-se claramente que o modelo de regressão linear será a escolha mais razoável.

Este será o modelo a utilizar em contextos onde o tempo de execução, utilização de recursos computacionais deve ser e utilização de recursos computacionais devem ser minimizados. Além disto, a implementação desse modelo é extremamente simples, enquanto as dos outros modelos necessitaram a especificação de parâmetros adicionais (e procura exaustiva pela melhor combinação destes) para obter resultados semelhantes.

Nas situações em que a exatidão/precisão das previsões é mais importante, e não existindo muitas ao nível da utilização de recursos, será razoável optar pela utilização de Gradient Boosting, que tende a produzir melhores resultados.

Verifica-se também que contexto não existe grande vantagem na utilização de uma rede neuronal. A relativa simplicidade dos dados e das relações entre os atributos não justificam a aplicação desta técnica, que acaba por produzir resultados semelhantes, necessitando, no entanto, de muitos mais recursos, mais parametrização e maior complexidade ao nível da implementação. Esta técnica adequar-se-á melhor a conjuntos de dados que apresentem problemas mais interessantes, onde exista a necessidade de assimilação de padrões mais complexos, e que não são tão facilmente modelados por técnicas de regressão (linear, em particular).

VI. CONCLUSÕES E TRABALHOS FUTUROS

A realização deste trabalho prático permitiu consolidar os conceitos relativos ao desenvolvimento de um projeto de *Machine Learning* utilizando modelos de aprendizagem, lecionados ao longo da unidade curricular. Este projeto proporcionou ao grupo a familiarização, não só com diferentes técnicas de exploração de dados, como também com diversos modelos de extração de conhecimento.

Durante o desenvolvimento do projeto, o grupo sentiu algumas dificuldades das quais se pode realçar o tratamento de colunas como a 'AVERAGE_CLOUDINESS' constituída por alguns valores inconsistentes e outros em falta, a escolha do melhor modelo de aprendizagem e consequentemente os seus parâmetros ótimos.

Outro aspecto relevante, relativo ao segundo *dataset*, é o nível de adequação da aplicação de diferentes técnicas de aprendizagem a um dado problema. Verificou-se, neste caso, que a utilização de regressão linear é muito mais pertinente. Isto evidenciou, principalmente, que técnicas mais simples não produzem necessariamente resultados inferiores, e que modelos de *deep learning* nem sempre são os candidatos mais ajustados. Percebeu-se que, em certos casos, é possível obter resultados mais satisfatórios recorrendo a técnicas de extração de conhecimento menos complexas e que se adequam melhor ao contexto (recursos de hardware/software, restrições ao nível do tempo de resposta/execução, *etc*…).

Assim sendo, o grupo considera que fez um bom trabalho tendo implementado todas as tarefas definidas no enunciado. No entanto, uns dos aspetos a melhorar seria a aplicação de um modelo de extração de conhecimento ainda melhor no primeiro *dataset,* como por exemplo as redes neuronais.