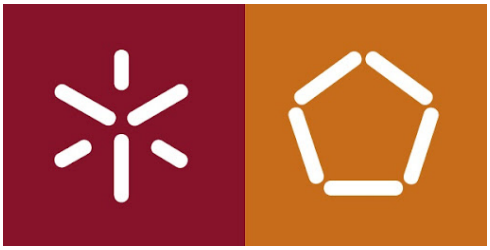

Universidade do Minho

Mestrado em Engenharia Informática

Conceção e otimização de modelos de *Machine Learning*

Dados e Aprendizagem Automática

Grupo 21

Gustavo Lourenço (pg47229)
Leonardo Marreiros (pg47398)
Martim Almeida (pg47514)
Pedro Fernandes (pg47559)

3 de janeiro de 2022

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Dataset 1 - Tráfego de veículos na cidade do Porto

1.1 Domínio

A modelação do fluxo de tráfego rodoviário é um conhecido problema de características estocásticas, não-lineares. A literatura tem apresentado um conjunto de modelos que demonstram um potencial assinalável neste tipo de previsões.

1.2 Objetivos

O objetivo principal do desenvolvimento deste modelo é prever o fluxo de tráfego rodoviário, numa determinada hora, na cidade do Porto. Com isto em mente, a única métrica que pretendemos seguir neste modelo é a *Accuracy* dado que apenas pretendemos obter o modelo com o maior número de previsões corretas.

1.3 Metodologia

A metodologia de extração de conhecimento utilizada foi a CRISP-DM. Começamos por analisar os diferentes fatores do *dataset* e como estes poderiam influenciar o fluxo de tráfego rodoviário. Após isto, definimos como objectivo obter a maior taxa de acertos nas nossas previsões. Ao analisarmos os dados verificamos algumas irregularidades como *outliers* ou valores em falta. A fase de processamento dos dados consistiu no tratamento das irregularidades encontradas na fase anterior. Passamos então para a fase de modelação onde começamos por utilizar diferentes modelos para prever o fluxo de tráfego. Ao analisar as previsões feitas pelos modelos não foram ótimas voltamos à fase inicial para melhorar os dados ou os modelos utilizados. Finalmente, após validarmos o nosso modelo como ótimo, podemos tirar conclusões sobre os resultados.

1.4 Análise dos Dados

O *dataset* utilizado contém dados referentes ao tráfego de veículos na cidade do Porto durante um período superior a 1 ano. A primeira análise a ser realizada foi visualizar os tipos dos fatores do *dataset* e se estes apresentam valores em falta. Através do comando *info* (fig 20-anexo A) conseguimos verificar que duas colunas apresentam valores em falta e seis colunas apresentam valores categóricos.

Em seguida foi realizada uma análise da distribuição dos dados numéricos do *dataset* (fig 21-anexo A).

Através da *correlation matrix* (fig 22-anexo A) conseguimos verificar a correlação entre as variáveis numéricas do *dataset*. Conseguimos realçar então a maior correlação existente entre a temperatura média e a humidade média com um valor de -0.51. Fora esta relação não conseguimos observar atributos claramente correlacionados.

1.4.1 *city_name*

Esta coluna representa o nome da cidade em causa. Ao utilizar o comando *value_counts* (fig 23-anexo A) podemos verificar que esta coluna apenas contém um valor, logo esta não foi selecionada para os dados utilizados na modelação.

1.4.2 *record_date*

Esta coluna representa o timestamp associado a cada registo do *dataset*.

1.4.3 *average_speed_diff*

Esta coluna representa a diferença entre a velocidade máxima que os carros podem atingir em cenários sem trânsito e a velocidade que realmente se verifica. Quanto mais alto o valor, maior é a diferença entre o que se está a andar no momento e o que se deveria estar a andar sem trânsito, i.e., valores altos deste atributo implicam que se está a andar mais devagar. Estes são os valores que o modelo deve prever.

1.4.4 *average_free_flow_speed*

Esta coluna representa o valor médio da velocidade máxima que os carros podem atingir em cenários sem trânsito. Consegue-se observar alguns *outliers* analisando o gráfico seguinte.

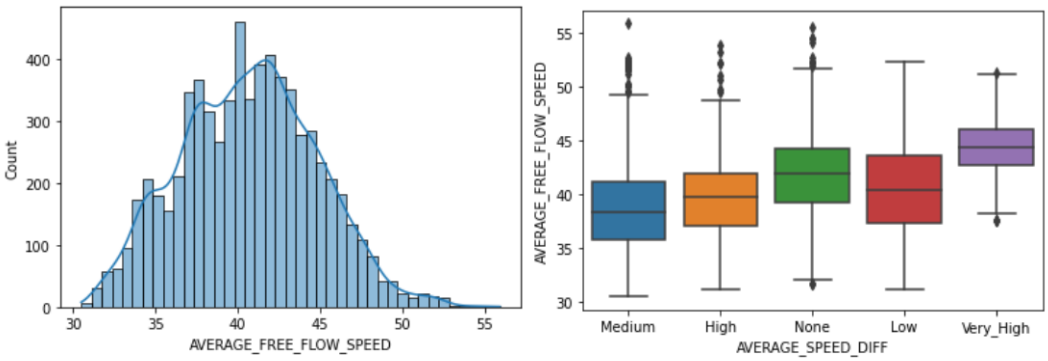

Figura 1: Visualização dos dados de *average_free_flow_speed*

1.4.5 *average_time_diff*

Esta coluna representa o valor médio da diferença do tempo que se demora a percorrer um determinado conjunto de ruas. Quanto mais alto o valor maior é a diferença entre o tempo que demora para se percorrer as ruas e o que se deveria demorar sem trânsito,

i.e., valores altos implicam que se está a demorar mais tempo a atravessar o conjunto de ruas. Consegue-se verificar que os valores variam entre 0 e 300 e maior parte dos valores encontram-se nos menores valores (entre 0 e 50). Também podemos observar alguns *outliers*.

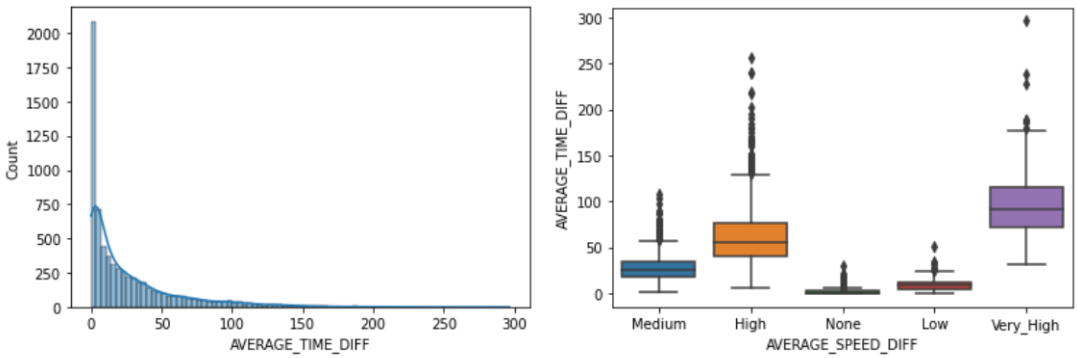

Figura 2: Visualização dos dados de *average_time_diff*

1.4.6 *average_free_flow_time*

Esta coluna representa o valor médio do tempo que demora a percorrer um determinado conjunto de ruas quando não há trânsito. Consegue-se verificar que a maior parte dos valores encontram-se entre 60 e 100. Também podemos observar alguns *outliers*.

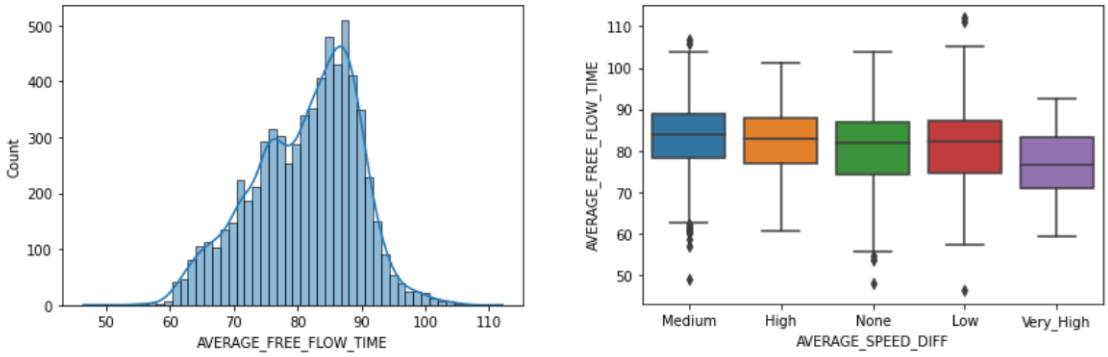

Figura 3: Visualização dos dados de *average_free_flow_time*

1.4.7 *luminosity*

Esta coluna representa o nível de luminosidade que se verificava na cidade do Porto.

1.4.8 *average_temperature*

Esta coluna representa o valor médio da temperatura para o *record_date* na cidade do Porto. Consegue-se verificar que os valores variam entre 0 e 35. Também podemos observar alguns *outliers*.

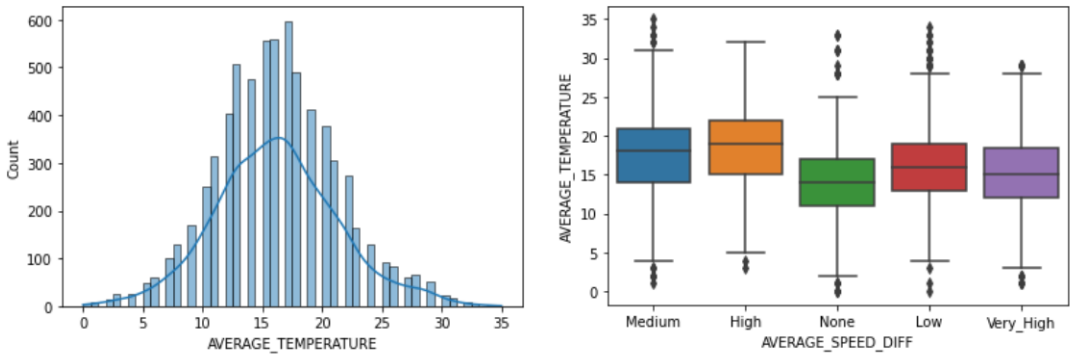

Figura 4: Visualização dos dados de *average_temperature*

1.4.9 *average_atmosp_pressure*

Esta coluna representa o valor médio da pressão atmosférica para o *record_date*. Consegue-se verificar que a maior parte dos valores variam entre 1010 e 1030. Também podemos observar bastantes *outliers*, tal como podemos verificar que esta coluna não parece apresentar uma grande correlação com a *average_speed_diff*.

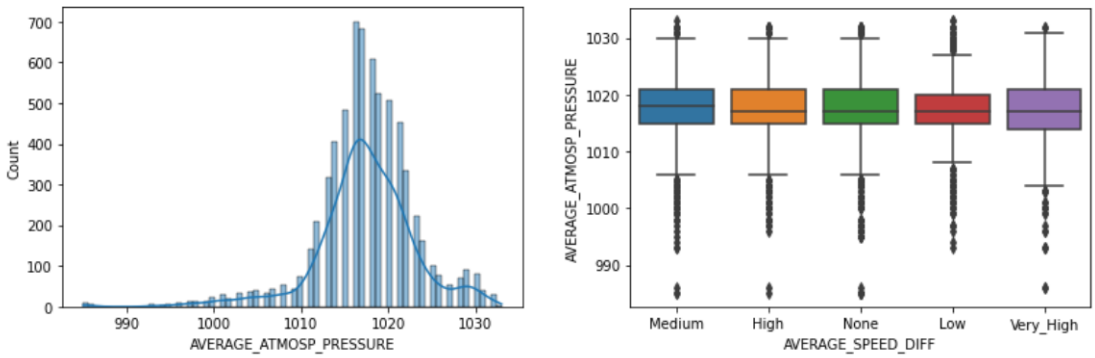

Figura 5: Visualização dos dados de *average_atmosp_pressure*

1.4.10 *average_humidity*

Esta coluna representa o valor médio da humidade para o *record_date*. Consegue-se verificar que os valores variam entre 0 e 100. Também podemos observar a exitência de *outliers*.

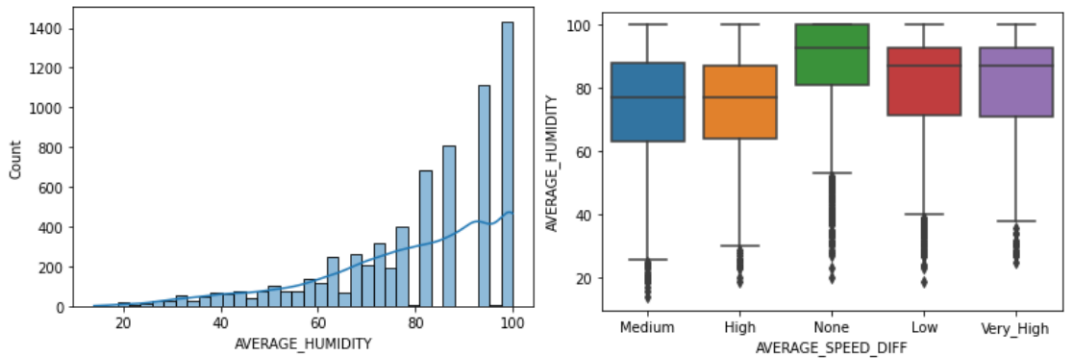

Figura 6: Visualização dos dados de *average_humidity*

1.4.11 *average_wind_speed*

Esta coluna representa o valor médio da velocidade do vento para o *record_date*. Consegue-se verificar que a maior parte dos valores variam entre 0 e 7. Também podemos observar a exitência de alguns *outliers*.

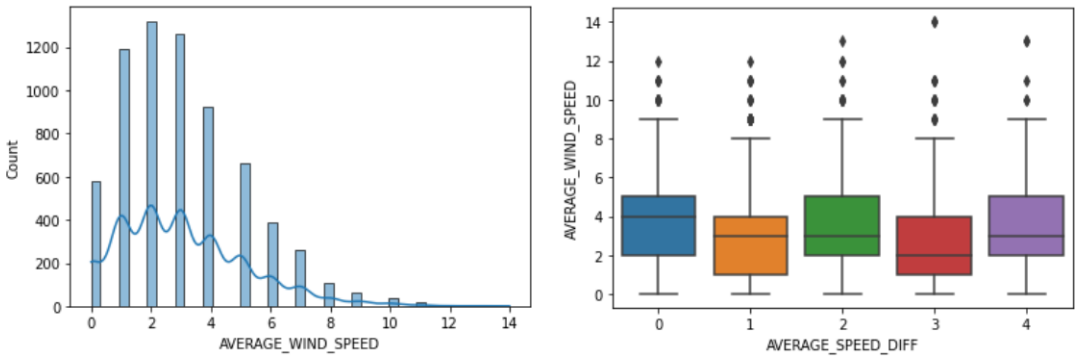

Figura 7: Visualização dos dados de *average_wind_speed*

1.4.12 *average_cloudiness*

Esta coluna representa o valor médio da percentagem de nuvens para o *record_date*. Podemos observar que podemos agrupar alguns valores como *nuvens quebradas* e *nuvens quebrados*.

1.4.13 *average_precipitation*

Esta coluna representa o valor médio de precipitação para o *record_date*. Ao utilizar o comando *value_counts* (fig 28-anexo A) podemos verificar que esta coluna apenas contém um valor, logo esta não foi selecionada para os dados utilizados na modelação.

1.4.14 *average_rain*

Esta coluna representa uma avaliação qualitativa da precipitação para o *record_date*. Através da Figura 20 podemos verificar que esta coluna só possui 563 valores não nulos, logo esta não foi selecionada para os dados utilizados na modelação.

1.5 Pré-Processamento dos Dados

1.5.1 *Encodings*

Foram utilizadas duas estratégias distintas para codificar os valores categóricos presentes no *dataset*, algumas colunas foram codificadas com *Label Encoding* enquanto outras beneficiaram do uso de *One Hot Encoding*.

Average_cloudiness
Para esta coluna começamos por agrupar valores diferentes ao substituí-los por valores já existentes na coluna. Em seguida utilizamos o método de *One Hot Encoding* para codificar os valores desta coluna, pois foi o método que apresentou melhor *accuracy*.

Luminosity
Utilizamos o método de *Label Encoding* para codificar os valores desta coluna, pois foi o método que apresentou melhor *accuracy*.

Average_speed_diff
Utilizamos o método de *Label Encoding* para codificar os valores desta coluna, pois foi o método que apresentou melhor *accuracy*.

1.5.2 *Missing values*

Para corrigir os valores em falta utilizamos duas estratégias: substituir pelo valor mais comum e substituir por um novo valor. Obtivemos uma *accuracy* melhor através do segundo método.

1.5.3 *Outliers*

Para corrigir os *outliers* tentamos substituí-los pelo valor médio da coluna. Como a *accuracy* diminuiu após o tratar os *outliers* de cada coluna optamos por deixar estes valores inalterados.

1.5.4 *Tratamento de datas*

Como foi referido anteriormente a coluna *record_date* representa o timestamp associado a cada registo do *dataset*, por isso a partir desta podemos retirar muitos valores e para tal

foram utilizadas diversas abordagens.

Variáveis diretas

Através destas datas podemos retirar os seguintes valores diretamente: ano, mês, semana, dia, dia da semana e hora, destes valores apenas mantivemos mês, semana, dia da semana e hora pois foram os únicos que beneficiaram a *accuracy*.

Feriados

Criamos uma nova coluna que indicaria se a data em questão representava um feriado, mas como esta piorou a *accuracy* acabamos por removê-la.

Partes do dia

Criamos e optimizamos uma função que indica a parte do dia que a *record_date* refere através da hora.

Após a utilização de *One Hot Encoding* e da função referida previamente podemos criar novas colunas para cada parte do dia.

Classificações adicionais

Também tentamos criar novas colunas como a separação entre estações do ano, indicar se uma data encontra-se no verão ou se esta encontra-se no fim de semana, mas após não obtermos uma *accuracy* melhor com estas colunas, decidimos não incluí-las no modelo final.

1.6 Modelos *Machine Learning*

1.6.1 Árvore de Decisão

Uma árvore de decisão é um algoritmo de aprendizagem supervisionado perfeito para problemas de classificação, pois é capaz de ordenar as classes a um nível preciso. Funciona como um diagrama de fluxo, separando os dados em duas categorias semelhantes ao mesmo tempo, do "tronco da árvore" aos "ramos" e às "folhas", onde as categorias se tornam mais finitamente semelhantes. Isso cria categorias dentro de categorias, permitindo a classificação orgânica com supervisão humana limitada.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params= {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
```

```
    'criterion': ["gini", "entropy"]
}
```

Após testar estes parâmetros podemos utilizar os que obtiveram os melhores resultados e criar o melhor modelo *Decision Tree Classifier*:

```
treemodel = DecisionTreeClassifier(criterion= 'gini',
                                   max_depth= 10,
                                   min_samples_leaf= 20)
```

1.6.2 SVM

O modelo *support vector machine* (SVM) usa algoritmos para treinar e classificar os dados em graus de polaridade, levando-os a um grau além da previsão X/Y.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params = {'C': [0.1,1, 10, 100, 200],
          'gamma': [1,0.1,0.01,0.001,0.0001],
          'kernel': ['rbf', 'linear']}
```

Após testar estes parâmetros podemos utilizar os que obtiveram os melhores resultados e criar o melhor modelo *Support Vector Classification*:

```
svcmodel = SVC(C = 10,
               gamma = 0.001,
               kernel = 'rbf')
```

1.6.3 XGBoost

Ao contrário de outros algoritmos, XGBoost é um algoritmo de aprendizagem conjunto, isto é, combina os resultados de vários modelos, chamados *base learners*, para fazer uma previsão. Tal como nas Florestas aleatórias o XGBoost usa Árvores de decisão como base.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params = {
  'learning_rate':[0.1,0.2,0.25,0.3],
  'n_estimators':[100,200,500,1000],
  'max_depth':range(3,10,2),
  'min_child_weight':range(1,6,2),
  'gamma':[i/10.0 for i in range(0,5)],
  'subsample':[i/10.0 for i in range(6,10)],
  'colsample_bytree':[i/10.0 for i in range(6,10)],
  'min_child_weight':[6,8,10,12],
  'reg_alpha':[1e-5, 1e-2, 0.1, 1, 100],
  'scoring':['accuracy']
}
```

Após testar estes parâmetros podemos utilizar os que obtiveram os melhores resultados e criar o melhor modelo *XGBClassifier*:

```
modelxgb = XGBClassifier(learning_rate =0.1,
                         n_estimators=100,
                         max_depth = 9,
                         min_child_weight = 13,
                         gamma=0.4,
                         subsample=0.6,
                         colsample_bytree=0.7,
                         reg_alpha=0.0001,
                         scale_pos_weight=1)
```

1.6.4 Outros Modelos

Também utilizamos diversos modelos como Redes Neuronais Artificiais, Florestas Aleatórias ou *Gradient Boosting*, mas, como não obtivemos resultados satisfatórios com estes modelos optamos por abandoná-los.

1.7 Análise dos Resultados

Para validar os resultados obtidos com os diferentes modelos utilizamos a *cross_val_score* com 10 *folds*.

	ÁRVORE DE DECISÃO	SUPPORT VECTOR MACHINE	XGBOOST
Accuracy	0.775	0.795	0.814
Standard deviation	0.01	0.008	0.007

Figura 8: Resultados obtidos com *cross_val_score*

Analisando a tabela conseguimos observar que o modelo cujo resultado foi mais favorável(assinalado a amarelo) para o problema em questão, foi o uso do *XGBoost*.

1.8 Considerações Finais

Consideramos ter atingido os objetivos inicialmente propostos, porém acreditamos que conseguiríamos obter um resultado mais satisfatório através da otimização de outros modelos (principalmente Redes Neuronais).

2 Dataset 2 - Previsão de insuficiência cardíaca

2.1 Domínio

Doenças cardiovasculares são a principal causa de morte em todo o mundo, causando cerca de 17.9 milhões de mortes todos os anos, o que representa 31% de todas as mortes mundialmente. Quatro em cada cinco mortes por doenças cardiovasculares são devido a ataques cardíacos e derrames, e um terço destas mortes ocorrem prematuramente em pessoas com menos de 70 anos. A insuficiência cardíaca é um comum causado por doenças cardiovasculares - este *dataset* contém 11 características que podem ser usadas para prever uma possível doença cardíaca.

Pessoas com doenças cardiovasculares ou com alto risco cardiovascular (devido à presença de um ou mais fatores de risco como hipertensão, diabetes, hiperlipidemia ou outra doença já estabelecida) precisam de deteção e gestão precoce, em que um modelo de aprendizagem *machine learning* pode ser de grande ajuda.

2.2 Objetivos

O objetivo principal do desenvolvimento deste modelo é prever se alguém tem alto risco de ser diagnosticado como cardiopata. Quanto mais cedo a identificação deste risco seja detetada, mais rápido podem ser prescritos exames de cálcio coronário e angiotomografias que fornecem fotos detalhadas do coração e dos vasos sanguíneos utilizadas para procurar doenças cardíacas.

Com isto em mente, a métrica com a qual estaremos mais preocupados será o *Recall* dada a extrema importância de uma taxa de falsos negativos mínima neste caso em partícular. A presença de falsos negativos implica que o paciente tenha uma doença cardíaca, apesar do modelo prever que a pessoa está segura e sem risco de doença colocando em risco a sua vida. O desenvolvimento deste *dataset* visa a construção de modelos com taxas de falsos negativos mínimas.

2.3 Metodologia

A metodologia de extração de conhecimento utilizada foi a CRISP-DM. Começamos por reconhecer como fatores como colesterol, idade e pressão sanguínea afetam a possibilidade de insuficiência cardíaca e os valores que seriam de esperar (valores altos de colesterol, idade mais avançada, pressão sanguínea alta, etc) para os pacientes que de facto tiveram doenças deste tipo. Reconhecemos então a importância de uma taxa de falsos negativos o mais baixa possível. Ao visualizarmos os dados verificamos que apesar de não haverem muitos *outliers*, havia muitos dados em falta ou que foram medidos erroneamente. Com isto, passamos para a fase de processamento dos dados onde inicialmente removemos os *outliers*, removemos os valores incorretos e aplicamos técnicas de *encoding* aos valores discretos. Passamos então para a modelação onde reparamos que talvez em vez de remover

os valores errados, trocar o seu valor pela média ou moda geraria melhores resultados e voltamos a modelar os modelos. Ao analisar os resultados que obtemos apercebemo-nos que diferentes técnicas de *encoding* e escalamento dos dados benificiam melhor certos modelos pelo que voltamos a processar os dados e a testar os modelos com estas novas características. Finalmente, antes de tirar conclusões sobre os resultados, voltamos a rever o contexto do problema e confirmar que de facto os objetivos que tínhamos definidos foram alcançados.

2.4 Análise dos Dados

O *dataset* em questão é referente a um diagnóstico de doenças cardiovasculares contendo diversas colunas que ajudam a prever o risco de doenças deste género. A primeira análise a ser realizada foi visualizar os tipos de dados assim como verificar a existência de *missing values*. Através do comando `df.info()`, conseguimos verificar que em princípio não existem valores em falta (fig 39-anexo B).

Em seguida foi realizada uma análise dos dados numéricos, sendo que, a distribuição inicial dos dados é a que se encontra nos anexos (fig 40-anexo B).

Através da matriz de confusão conseguimos verificar a correlação entre todas as variáveis numéricas do *dataset*. Conseguimos observar que não existem atributos claramente correlacionados, sendo que é de realçar que as maiores correlações são relativas a *MaxHR* e *HeartDisease* que apresentam um índice de correlação de -0.4, o que significa que à partida quanto maior for o *MaxHR* menor a probabilidade de *HeartDisease*, e a correlação entre *HeartDisease* com *Oldpeak* com valor de 0.4 o que revela que crescem os seus valores uma com a outra. Existe ainda com valor de -0.38 uma certa correlação negativa entre *MaxHR* e *Age* (fig 41-anexo B).

2.4.1 *Age*

Esta coluna representada por um dado discreto, é referente à idade de cada pessoa presente no *dataset*. Através da visualização do histograma e do *boxplot* parece que a idade está de acordo com o esperado, isto é, pessoas mais velhas são mais prováveis de ter doenças cardiovasculares. Mesmo assim parecem existir alguns *outliers* que contém doenças deste género com idades bastante baixas.

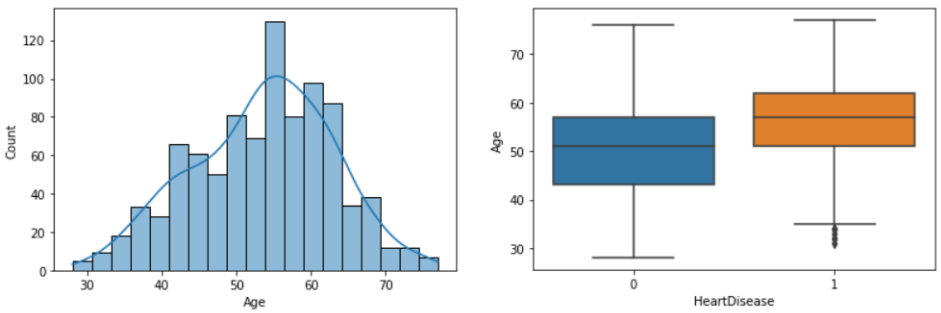

Figura 9: Visualização dos dados de *Age*

2.4.2 *RestingBP*

Este atributo discreto mede a pressão arterial em repouso na unidade mm Hg. Sendo que valores normais residem abaixo de 120, valores de risco entre 120 e 139, e acima de 140 revela é sinal de hipertensão os dados apresentados parecem bastante realistas pensando nas pessoas que se encontram em estudo. Ainda assim existem uns certos *outliers* como por exemplo aqueles valores perto de 0 que devem ser resultado de um erro de medição.

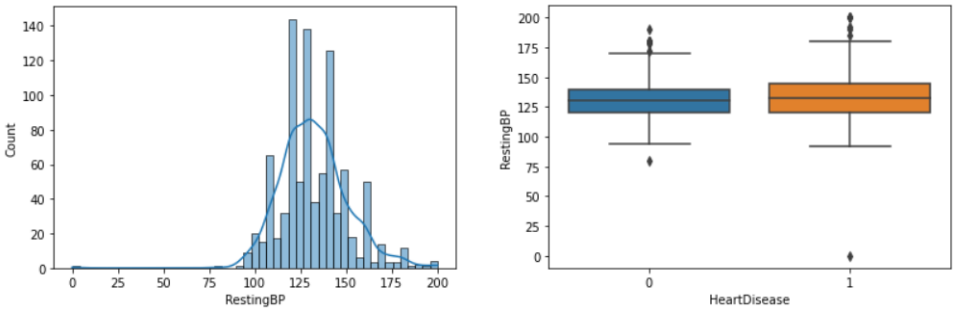

Figura 10: Visualização dos dados de *RestingBP*

2.4.3 *Cholesterol*

O colesterol, representado nesta coluna sob a forma de um dado discreto, é uma medida bem conhecida de probabilidade da existência de doenças cardiovasculares sendo medido em mm/dl. Valores ideais residem abaixo de 200 e valores indesejáveis acima dos 240. Visualizando os gráficos em seguida verifica-se que algo de errado não está certo. Existem imensos valores de colesterol a 0, o que definitivamente está incorreto, o que vai levar a um obrigatório tratamente destes dados.

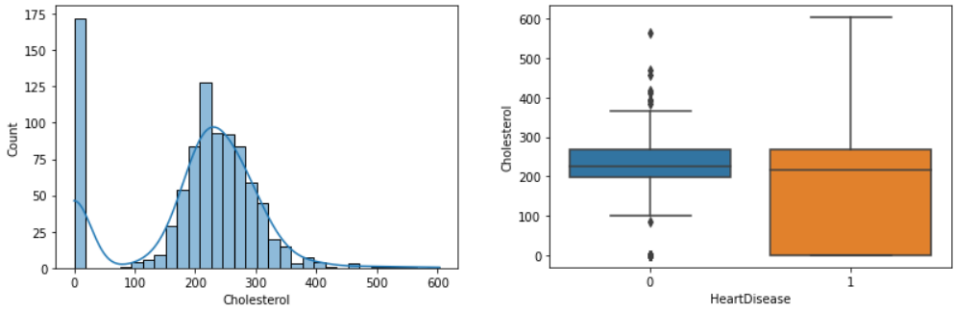

Figura 11: Visualização dos dados de *Cholesterol*

2.4.4 *MaxHR*

Por sua vez o *MaxHR* representa o valor máximo de batimentos cardíacos atingido sendo que, os valores variam de 60 a 202 bpm (dados discretos). Visualizando os dados parece estar tudo dentro da normalidade (tirando a existência de alguns *outliers*) sendo

que, um menor número de batimentos máximo parece favorecer a existência de doenças cardiovasculares.

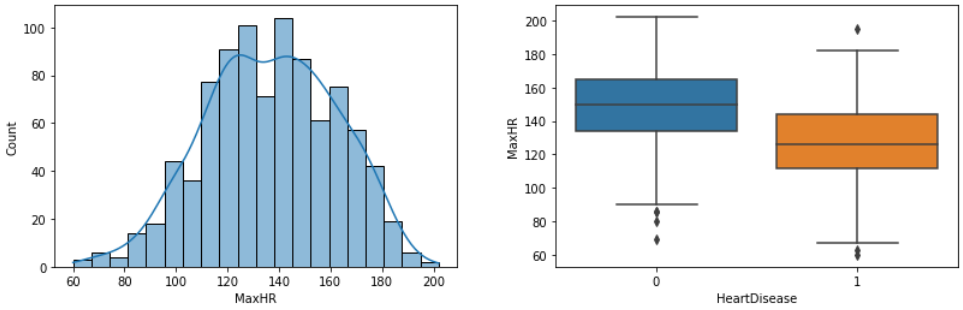

Figura 12: Visualização dos dados de *MaxHR*

2.4.5 *Oldpeak*

O *Oldpeak* é uma métrica comum para diagnósticos de deteção destas doenças, sendo este a inclinação do pico atingindo do exercício em relação ao repouso. Segundo pesquisa realiazada, valores inferiores a 2 representam baixo risco, de 1.5 a 4.2 já são considerados perigosos e maior que 2.55 é pior ainda. Consegue-se observar alguns *outliers* analisando o gráfico seguinte, sendo que irão ser retirados posteriormente.

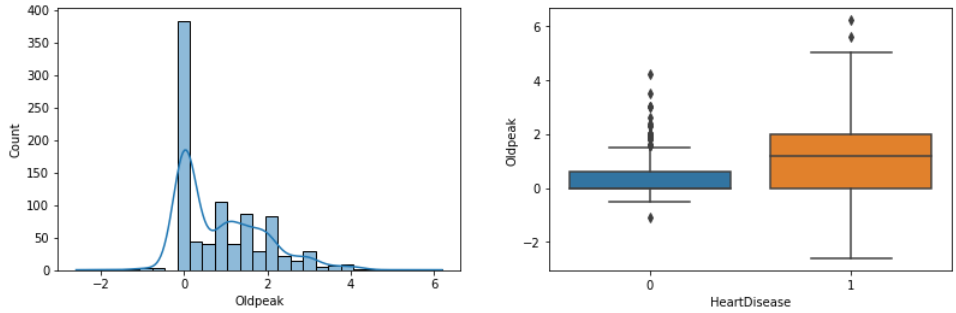

Figura 13: Visualização dos dados de *Oldpeak*

2.5 Pré-Processamento dos Dados

Nesta fase foram preparados os dados tendo como base a análise dos dados da etapa anterior. Esta parte é fulcral para que os modelos consigam obter os melhores resultados possíveis. Desta forma dependendo do modelo utilizado foram usadas algumas variantes, de modo que estes fossem otimizados.

2.5.1 Encodings

Foram criados duas manipulações distintas do dataset original de modo a beneficiar cada modelo. Modelos baseados em árvores preferem dados categóricos codificados com *Label*

Encoding, isto é, cada valor de atributo passar a ser um número representativo. Os outros modelos preferem por sua vez *One Hot Encoding* que é uma codificação que adiciona uma coluna por cada valor nos atributos categóricos, sendo o valor destas novas colunas 0 ou 1, caso se verifica ou não. Em seguida apresenta-se o resultado do comando head() tanto para a versão de *Label Encoding* em primeiro e a de *One Hot Encoding* em seguida.

	Age	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Oldpeak	ST_Slope	HeartDisease	Gender	Exerc
0	40	2	140	289	0	0	172	0.0	2	0	1	0
1	49	1	160	180	0	0	156	1.0	1	1	0	0
2	37	2	130	283	0	1	98	0.0	2	0	1	0
3	48	0	138	214	0	0	108	1.5	1	1	0	1
4	54	1	150	195	0	0	122	0.0	2	0	1	0

Figura 14: *Dataset* df, *Label Encoding* das variáveis categóricas

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	F	M	ASY	...	NAP	TA	LVH	Normal	ST	N	Y	Down	Flat	Up
0	40	140	289	0	172	0.0	0	0	1	0	...	0	0	0	1	0	1	0	0	0	1
1	49	160	180	0	156	1.0	1	1	0	0	...	1	0	0	1	0	1	0	0	1	0
2	37	130	283	0	98	0.0	0	0	1	0	...	0	0	0	0	1	1	0	0	0	1
3	48	138	214	0	108	1.5	1	1	0	1	...	0	0	0	1	0	0	1	0	1	0
4	54	150	195	0	122	0.0	0	0	1	0	...	1	0	0	1	0	1	0	0	0	1

5 rows × 21 columns

Figura 15: *Dataset* df1he, *One Hot Encoding* das variáveis categóricas

2.5.2 Outliers

Para todos os dados numéricos foram retirados os *outliers* com base na distância dos pontos ao primeiro e terceiro quartil. Se estes se encotrarem a uma distância superior a 1.5 vezes a distância inter quartil 1 e 3, serão removidos. No anexo B encontra-se um exemplo com o código relativo a este método para a *feature RestingBPf* (Excerto 1).

Na imagem em seguida podemos ver o resultado da aplicação do método anteriormente referido à coluna *RestingBPf*, sendo o *boxplot* da esquerda o estado inicial e o da direita o estado processado.

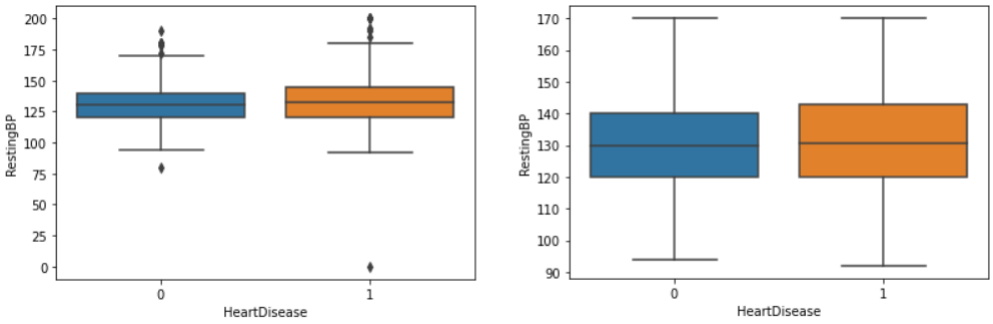

Figura 16: *RestingBPf* antes e depois da remoção dos *outliers*

2.5.3 Valores Errados

Durante a análise dos valores de Colesterol verificaram-se imensos dados com o valor 0 (161 casos). Este valor nulo é claramente um erro ou um *missing value*, de modo que tem que ser tratado. Para resolver o problema foram realizadas duas abordagens, uma em que todos os registos de valores de colesterol 0 foram removidos, e uma outra que pegou na média dos valores de colesterol (sem contar com os valores a 0) e substitui os valores a 0 pelo resultado obtido.

Assim surgiram mais duas variações de *datasets*, o dfaux (e respetivo dfaux1h para a versão com *one hot encoding*) e df (e respetivo df1he para a versão com *one hot encoding*), sendo respectivamente a versão com a remoção das colunas a 0, e a versão de alteração do valor do colesterol pela média.

Finalmente verificou-se o balanceamento de ambas as variações relativamente à contagem de *HeartDisease* e em ambos os casos não existe uma variação muito grande para os valores a 0 e a 1. Na imagem seguinte pode-se ver a distribuição para o caso da remoção dos valores nulos à esquerda, e a substituição dos valores à direita:

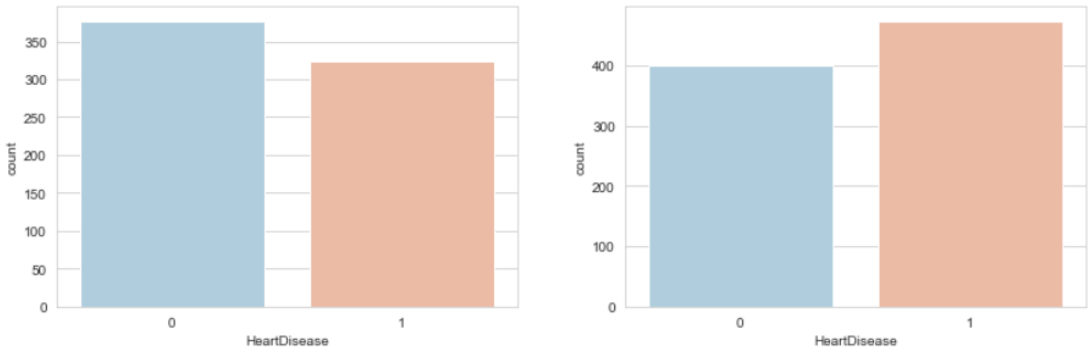

Figura 17: Balanceamento dos *Datasets*

2.6 Modelos *Machine Learning*

Para cada modelo de aprendizagem foi aplicado o *GridSearchCV* com *KFold* e um grupo de parâmetros específico. Foi também tendo em conta a normalização de dados assim como o tipo de *encoding* dos dados categóricos preferido por cada modelo.

2.6.1 Separação Treino e Teste

De modo a customizar qual o método de eliminação dos valores nulos foi realizada a seguinte divisão em treino e teste que comentando a versão que não se pretende usar, permite selecionar o método pretendido. Uma vez que a variante de *label encoding* será usado apenas para modelos de árvores não se recorreu à normalização de dados, enquanto que para a outra, esta técnica já foi realizada. O código apresentado em seguida mostra como esta parte foi realizada, sendo que neste caso encontra-se seleccionado a opção de remoção dos dados nulos.

```
# ALTERAÇÃO DOS MISSING VALUE
#X = df.drop(['HeartDisease'], axis=1)
#y = df['HeartDisease'].to_frame()

# ALTERAÇÃO DOS MISSING VALUE
#X1h = df1he.drop(['HeartDisease'], axis=1)
#y1h = df1he['HeartDisease'].to_frame()

# REMOÇÃO DOS MISSING VALUE
X = dfaux.drop(['HeartDisease'], axis=1)
y = dfaux['HeartDisease'].to_frame()

# REMOÇÃO DOS MISSING VALUE
X1h = dfaux1h.drop(['HeartDisease'], axis=1)
y1h = dfaux1h['HeartDisease'].to_frame()


X_train, X_test, y_train, y_test =
                train_test_split(X,y, test_size=0.25, random_state=2021)
X1h_train, X1h_test, y1h_train, y1h_test =
                train_test_split(X1h,y1h, test_size=0.25, random_state=2021)

ro_scaler = MinMaxScaler()
X1h_train = ro_scaler.fit_transform(X1h_train)
X1h_test = ro_scaler.transform(X1h_test)
```

2.6.2 Regressão Logística

A regressão logística é um cálculo usado para prever um resultado binário: ou algo acontece ou não. Variáveis independentes são analisadas para determinar o resultado binário com resultados em uma de duas categorias. As variáveis independentes podem ser categóricas ou numéricas, mas a variável dependente é sempre categórica.

Parâmetros

Foi utilizado o *solver saga* que converge rapidamente com atributos com aproximadamente a mesma escala pelo que também foi feito um escalamento dos dados antes. Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params= {'logisticregression__C':[0.001,.009,0.01,.09,1,5,10,25],
        'logisticregression__penalty':['l1', 'l2']}
```

2.6.3 Árvore de Decisão

Uma árvore de decisão é um algoritmo de aprendizagem supervisionado perfeito para problemas de classificação, pois é capaz de ordenar as classes a um nível preciso. Funciona como um diagrama de fluxo, separando os dados em duas categorias semelhantes ao mesmo tempo, do "tronco da árvore" aos "ramos" e às "folhas", onde as categorias se tornam mais finitamente semelhantes. Isso cria categorias dentro de categorias, permitindo a classificação orgânica com supervisão humana limitada.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params= {'criterion': ['gini', 'entropy'], 'max_depth': range(1,10),
             'min_samples_leaf': range(1,5)}
```

2.6.4 SVM

O modelo *support vector machine* (SVM) usa algoritmos para treinar e classificar os dados em graus de polaridade, levando-os a um grau além da previsão X/Y.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

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

2.6.5 Florestas Aleatórias

Uma floresta aleatória, como o próprio nome indica, consiste num grande número de árvores de decisão individuais que operam como um conjunto. Cada árvore individual

na floresta aleatória executa uma previsão de classe e a classe com mais votos torna-se a previsão do nosso modelo.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
```

2.6.6 XGBoost

Ao contrário de outros algoritmos, XGBoost é um algoritmo de aprendizagem conjunto, isto é, combina os resultados de vários modelos, chamados *base learners*, para fazer uma previsão. Tal como nas Florestas aleatórias o XGBoost usa Árvores de decisão como base.

Parâmetros

Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params = {'max_depth': range (2, 10, 1),
          'n_estimators': [50, 100, 250, 400, 500, 600, 750, 1000],
          'learning_rate': [0.001, 0.0025, 0.005, 0.0075, 0.01],
          'objective': ['binary:hinge', 'binary:logistic',
          'binary:logitraw']
}
```

2.6.7 Redes Neuronais Artificiais

As redes neurais artificiais (ANNs) são compostas por camadas de nodos, contendo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada nodo, ou neurónio artificial, está conectado a outro e tem um peso e *threshold* associados. Se o *output* de qualquer nó individual estiver acima do *threshold* especificado, esse nó será ativado, enviando dados para a próxima camada da rede. Caso contrário, nenhum dado é passado para a próxima camada da rede.

Parâmetros

Primeiramente os dados foram escalados. Utilizando o *one hot encoding*, a dimensão do input é igual a 20. Os parâmetros a testar pelo *GridSearchCV* foram os seguintes:

```
params = {
    'batch_size': [30, 60, 90],
    'activation': ['relu', 'tanh', 'sigmoid'],
    'kernel_initializer': ['HeNormal', 'GlorotNormal'],
    'neurons': [12,13,14],
    'epochs': [500] ,
    'learning_rate' : [0.001, 0.01]
}
```

2.7 Análise dos Resultados

Os resultados obtidos para todos os modelos quer na variante da remoção dos valores nulos, quer na de substituição encontram-se nas tabelas abaixo. Visto o nosso problema ser relacionado com diagnóstico de doenças, é importante ter uma *accuracy* alta, no entanto o mais importante é evitar os falsos negativos, daí resulta que a métrica mais importante é o *recall*.

	REMOÇÃO DOS MISSING VALUE					
	REGRESSÃO LOGISTICA	ÁRVORE DE DECISÃO	SUPPORT VECTOR MACHINE	RANDOM FORESTS	REDES NEURONAIS	XGBOOST
Accuracy	0,875	0,869	0,886	0,892	0,926	0,886
Recall	0,899	0,91	0,91	0,933	0,955	0,899
F1 Score	0,879	0,876	0,89	0,897	0,929	0,889

Figura 18: Resultados da variante que remove os valores 0

	ALTERAÇÃO DOS MISSING VALUE					
	REGRESSÃO LOGISTICA	ÁRVORE DE DECISÃO	SUPPORT VECTOR MACHINE	RANDOM FORESTS	REDES NEURONAIS	XGBOOST
Accuracy	0,877	0,826	0,881	0,868	0,872	0,872
Recall	0,921	0,851	0,93	0,921	0,93	0,912
F1 Score	0,886	0,836	0,891	0,879	0,883	0,881

Figura 19: Resultados da variante que altera os valores 0

Analisando as tabelas conseguimos observar que o modelo cujo resultado foi mais favorável(assinalado a amarelo) para o problema em questão, foi o uso de redes neuronais com a remoção dos registos com valores a 0 de colesterol.

2.8 Considerações Finais

Dada por concluída a análise deste *dataset*, é importante referir que, apesar da dimensão deste conjunto de dados ser baixa, escolhemos este problema por ser interessante e com um propósito real e compensamos este "problema"com o desenvolvimento de vários modelos com vários pré-processamentos de dados diferentes, daí termos conseguido resultados bastante satisfatórios.

Consideramos ter atingido os objetivos inicialmente propostos com uma taxa de falsos negativos mínima. Dada a dimensão do *dataset*, remover os valores a 0 não é o cenário ideal pois pode levar a algum *overfitting* do modelo. Ainda assim, os resultados conseguidos com a alteração dos valores a 0 foram igualmente satisfatórios.

A Anexos Dataset 1

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6812 entries, 0 to 6811
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   city_name                6812 non-null   object
 1   record_date              6812 non-null   object
 2   AVERAGE_SPEED_DIFF       6812 non-null   object
 3   AVERAGE_FREE_FLOW_SPEED  6812 non-null   float64
 4   AVERAGE_TIME_DIFF        6812 non-null   float64
 5   AVERAGE_FREE_FLOW_TIME   6812 non-null   float64
 6   LUMINOSITY               6812 non-null   object
 7   AVERAGE_TEMPERATURE      6812 non-null   float64
 8   AVERAGE_ATMOSP_PRESSURE  6812 non-null   float64
 9   AVERAGE_HUMIDITY         6812 non-null   float64
 10  AVERAGE_WIND_SPEED       6812 non-null   float64
 11  AVERAGE_CLOUDINESS       4130 non-null   object
 12  AVERAGE_PRECIPITATION    6812 non-null   float64
 13  AVERAGE_RAIN             563 non-null    object
dtypes: float64(8), object(6)
memory usage: 745.2+ KB
```

Figura 20: Visualização dos tipos de dados e dos valores em falta

```
df.describe()
```

	AVERAGE_FREE_FLOW_SPEED	AVERAGE_TIME_DIFF	AVERAGE_FREE_FLOW_TIME	AVERAGE_TEMPERATURE	AVERAGE_ATMOSP_PRESSURE	AVERAGE_HUMIDITY	AVERAGE_WIND_SPEED	AVERAGE_PRECIPITATION
count	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.000000	6812.0
mean	40.661010	25.637111	81.143952	16.193482	1017.388139	80.084190	3.058573	0.0
std	4.119023	33.510507	8.294401	5.163492	5.751061	18.238863	2.138421	0.0
min	30.500000	0.000000	46.400000	0.000000	985.000000	14.000000	0.000000	0.0
25%	37.600000	2.275000	75.400000	13.000000	1015.000000	69.750000	1.000000	0.0
50%	40.700000	12.200000	82.400000	16.000000	1017.000000	83.000000	3.000000	0.0
75%	43.500000	36.200000	87.400000	19.000000	1021.000000	93.000000	4.000000	0.0
max	55.900000	296.500000	112.000000	35.000000	1033.000000	100.000000	14.000000	0.0

Figura 21: Distribuição dos dados numéricos iniciais

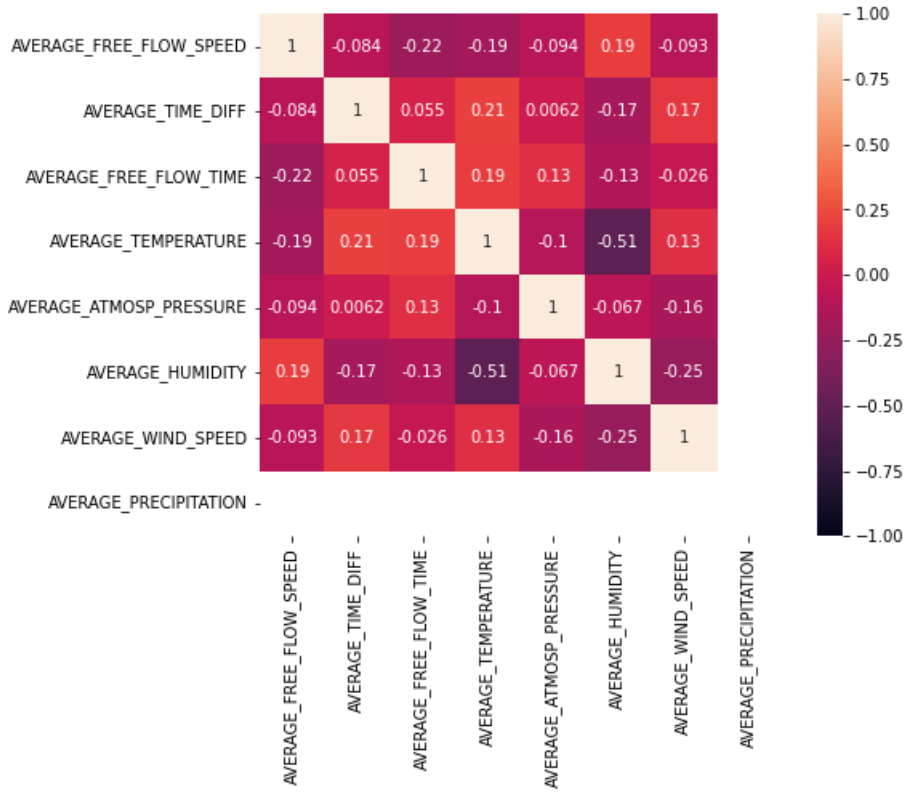

Figura 22: Correlation matrix

```
print(df['city_name'].value_counts())
print(dfTest['city_name'].value_counts())
```

```
Porto    6812
Name: city_name, dtype: int64
Porto    1500
Name: city_name, dtype: int64
```

Figura 23: Frequência dos diferentes valores presentes em *city_name*

```
print(df["AVERAGE_SPEED_DIFF"].value_counts())
```

```
None         2200
Medium       1651
Low          1419
High         1063
Very_High     479
Name: AVERAGE_SPEED_DIFF, dtype: int64
```

Figura 24: Frequência dos diferentes valores presentes em *average_speed_diff*

```
print(df["LUMINOSITY"].value_counts())
```

```
LIGHT        3293
DARK         3253
LOW_LIGHT     266
Name: LUMINOSITY, dtype: int64
```

Figura 25: Frequência dos diferentes valores presentes em *luminosity*

```
print(df['AVERAGE_CLOUDINESS'].value_counts())
```

```
céu claro            1582
céu pouco nublado     516
nuvens dispersas      459
nuvens quebrados      448
algumas nuvens        422
nuvens quebradas      416
céu limpo             153
nublado                67
tempo nublado          67
Name: AVERAGE_CLOUDINESS, dtype: int64
```

Figura 26: Frequência dos diferentes valores presentes em *average_cloudiness*

```
df['AVERAGE_CLOUDINESS'] = df['AVERAGE_CLOUDINESS'].replace(['nublado'],'tempo nublado')
df['AVERAGE_CLOUDINESS'] = df['AVERAGE_CLOUDINESS'].replace(['nuvens quebrados'],'nuvens quebradas')
df['AVERAGE_CLOUDINESS'] = df['AVERAGE_CLOUDINESS'].replace(['céu limpo'],'céu claro')
df['AVERAGE_CLOUDINESS'] = df['AVERAGE_CLOUDINESS'].replace(['nuvens dispersas'],'nuvens quebradas')
df['AVERAGE_CLOUDINESS'] = df['AVERAGE_CLOUDINESS'].replace(['céu pouco nublado'],'algumas nuvens')
```

Figura 27: Substituição dos valores em *average_cloudiness*

```
print(df["AVERAGE_PRECIPITATION"].value_counts())
```

```
0.0    6812
Name: AVERAGE_PRECIPITATION, dtype: int64
```

Figura 28: Frequência dos diferentes valores presentes em *average_precipitation*

algumas nuvens	céu claro	desconhecido	nuvens quebradas	tempo nublado
0	0	1	0	0
0	1	0	0	0
0	0	1	0	0
0	1	0	0	0
0	0	1	0	0

Figura 29: colunas criadas por *One Hot Encoding* em *average_cloudiness*

LUMINOSITY
2
2
2
2
2

Figura 30: *Luminosity* depois da utilização de *Label Encoding*

AVERAGE_SPEED_DIFF
2
0
0
0
2

Figura 31: *Average_speed_diff* depois da utilização de *Label Encoding*

```
df['AVERAGE_CLOUDINESS'].fillna(value = 'desconhecido', inplace=True)
dfTest['AVERAGE_CLOUDINESS'].fillna(value = 'desconhecido', inplace=True)
```

Figura 32: Substituição dos valores em falta em *average_cloudiness*

```
def mean_imputation_outliers(df,field):
  q1 = df[field].quantile(0.25)
  q3 = df[field].quantile(0.75)
  iqr = q3 - q1
  df.replace(df[df[field] > (iqr + np.percentile(df[field],75))].index,np.mean(df[field]), inplace=True)
  df.replace(df[df[field] < (np.percentile(df[field],25)- iqr)].index,np.mean(df[field]), inplace=True)
```

Figura 33: Função para subtituir *outliers* pelo valor médio da coluna

```
df['Year'] = df['record_date'].dt.year
df['Month'] = df['record_date'].dt.month
df['Week'] = df['record_date'].dt.week
df['Day'] = df['record_date'].dt.day
df['Weekday'] = df['record_date'].dt.dayofweek
df['Hour'] = df['record_date'].dt.hour
```

Figura 34: Criação das novas colunas através da *record_date*

```
import holidays
days = holidays.CountryHoliday('PT')
df['Is_Holiday'] = df['record_date'].apply(lambda x : 1 if x in days else 0)
```

Figura 35: Criação da coluna indicativa de feriados

```
def daypart(hour):
  if hour >= 0 and hour < 8:
    return "dawn"
  if hour >= 8 and hour < 9:
    return "morning commute"
  if hour >= 9 and hour < 12:
    return "working hours"
  if hour >= 12 and hour < 14:
    return "lunch break"
  if hour >= 14 and hour < 16:
    return "working hours"
  if hour >= 16 and hour < 18:
    return "end of work"
  if hour >= 18 and hour < 21:
    return "evening"
  if hour >= 21 and hour < 24:
    return "night"
```

Figura 36: Função de classificação da parte do dia

dawn	end of work	evening	lunch break	morning commute	night	working hours
1	0	0	0	0	0	0
0	0	0	0	0	0	1
0	1	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	1	0	0	0

Figura 37: colunas obtidas através de *One Hot Encoding*

```
df['Is_Summer'] = df['Month'].apply(lambda x : 1 if x in range(6,9) else 0)
dfTest['Is_Summer'] = dfTest['Month'].apply(lambda x : 1 if x in range(6,9) else 0)
df['Is_Weekend'] = df['Weekday'].apply(lambda x : 1 if x in [5,6] else 0)
dfTest['Is_Weekend'] = dfTest['Weekday'].apply(lambda x : 1 if x in [5,6] else 0)
```

Figura 38: Criação de novas colunas que indicam se a data está no verão ou num fim de semana

B Anexos Dataset 2

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Figura 39: Visualização dos tipos de dados e *missing values*

```
df.describe()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

Figura 40: Distribuição dos dados iniciais

Figura 41: Matriz de confusão

```
# Cálculo da distância inter quartil

Q1 = np.percentile(df['RestingBP'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(df['RestingBP'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

# Remoção Dos outliers em ambos os datasets

df = df[ (df['RestingBP'] <= (Q3+1.5*IQR)) &
         (df['RestingBP'] >= (Q1-1.5*IQR))]

df1he = df1he[ (df1he['RestingBP'] <= (Q3+1.5*IQR)) &
               (df1he['RestingBP'] >= (Q1-1.5*IQR))]
```

Listing 1: Remoção de outliers - exemplo