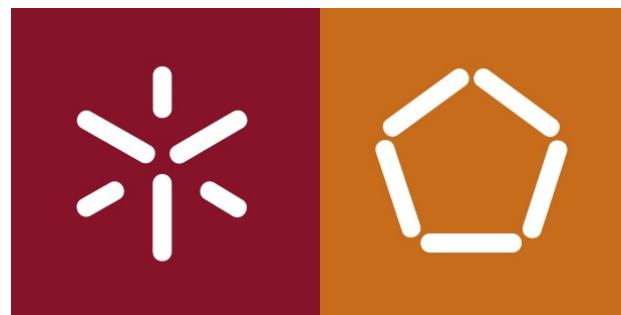


MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

CLASSIFICADORES E SISTEMAS CONEXIONISTAS
GRUPO 3



**Redes Neuronais Recorrentes para
previsão do fluxo de tráfego
rodoviário**

Bruno Manuel Macedo Nascimento, A67647
João Miguel Freitas Palmeira, A73864
Rafael Machado Silva, A74264
Ricardo Barros Pereira, A77045



Resumo

Cada vez mais a segurança rodoviária tem sido um ponto de preocupação na nossa sociedade. O número substancial diário de acidentes e mortes que se verificam nas estradas, dia após dia. Fatores climáticos, poluição atmosférica, grau de saturação que se verifica em determinadas zonas e ruas de várias cidades, desleixo dos intervenientes rodoviários são alguns dos fatores que causam estes acidentes e mortes nas estradas.

O intuito deste projeto é realizar previsões que permitam de alguma forma evitar que alguns destas calamidades aconteçam devido ao facto das pessoas não terem meios de recomendação que possam utilizar nas alturas em que circulem numa estrada.



Conteúdo

1	Introdução	6
1.1	Contextualização	6
1.2	Motivação	6
1.3	Objetivos	6
1.4	Estrutura do relatório	6
2	Metodologia	7
2.1	CRISP-DM	7
2.2	Fases	7
2.2.1	Business understanding	7
2.2.2	Data understanding	8
2.2.3	Data preparation	8
2.2.4	Modeling	9
2.2.5	Evaluation	9
2.2.6	Deployment	10
3	Problema	11
3.1	Análise e tratamento de dados	11
3.2	Construção do modelo	11
3.3	Desafios	11
3.4	Plano de desenvolvimento	11
4	Análise de dados	12
4.1	Descrição dos <i>datasets</i>	12
4.2	Análise do <i>dataset Weather Description</i>	12
4.2.1	<i>city_name</i>	13
4.2.2	<i>cloudiness</i>	13
4.2.3	<i>atmosphere</i>	13
4.2.4	<i>snow</i>	13
4.2.5	<i>thunderstorm</i>	13
4.2.6	<i>rain</i>	13
4.2.7	<i>sunrise</i>	14
4.2.8	<i>sunset</i>	14
4.2.9	<i>creation_date</i>	14
4.3	Análise do <i>dataset Traffic Incidents</i>	14
4.3.1	<i>city_name</i>	15
4.3.2	<i>description</i>	15
4.3.3	<i>cause_of_incident</i>	15
4.3.4	<i>from_road</i>	15
4.3.5	<i>to_road</i>	15
4.3.6	<i>affected_roads</i>	15
4.3.7	<i>incident_category_desc</i>	16
4.3.8	<i>magnitude_of_delay_desc</i>	16
4.3.9	<i>length_in_meters</i>	16
4.3.10	<i>delay_in_seconds</i>	16
4.3.11	<i>incident_date</i>	16
4.3.12	<i>latitude</i>	16
4.3.13	<i>longitude</i>	16
4.4	Análise do <i>dataset Traffic Flow</i>	16
4.4.1	<i>city_name</i>	18
4.4.2	<i>road_num</i>	18
4.4.3	<i>road_name</i>	18
4.4.4	<i>functional_road_class_desc</i>	18
4.4.5	<i>current_speed</i>	18
4.4.6	<i>free_flow_speed</i>	19
4.4.7	<i>speed_diff</i>	19
4.4.8	<i>current_travel_time</i>	19
4.4.9	<i>free_flow_travel_time</i>	19
4.4.10	<i>time_diff</i>	19
4.4.11	<i>creation_date</i>	19
4.5	Análise do <i>dataset Weather Until</i>	19
4.5.1	<i>city_name</i>	20
4.5.2	<i>temperature</i>	20
4.5.3	<i>atmospheric_pressure</i>	21



4.5.4	<i>humidity</i>	21
4.5.5	<i>wind_speed</i>	21
4.5.6	<i>clouds</i>	21
4.5.7	<i>precipitation</i>	21
4.5.8	<i>current_luminosity</i>	21
4.5.9	<i>sunrise</i>	21
4.5.10	<i>sunset</i>	21
4.5.11	<i>creation_date</i>	21
5	Tratamento de dados	22
5.1	Tratamento do dataset <i>Traffic Flow</i>	22
5.2	Tratamento do dataset <i>Traffic Incidents</i>	24
5.3	Tratamento do dataset <i>Weather Description</i>	28
5.4	Tratamento do dataset <i>Weather Until</i>	31
5.5	Criação dos datasets finais	32
6	Desenvolvimento de um modelo	38
6.1	Escolha das <i>features</i>	38
6.1.1	Avenida dos Aliados	39
6.1.2	Análise dos resultados	40
6.2	Criação do modelo	42
6.3	Otimizações	45
7	Análise de resultados	47
7.1	Melhores modelos	47
7.2	Previsões	49
7.2.1	Melhores previsões	50
8	Conclusão e trabalho futuro	52
9	Referências	53
A	Anexos	54
A.1	<i>Workflow</i> completo em KNIME do tratamento de dados	54
A.2	<i>Random Search</i> - Rua Conde de Vizela	55
A.3	<i>XGBoost</i> - Rua Conde de Vizela	56
A.4	<i>Random Search</i> - Avenida Gustavo Eiffel	57
A.5	<i>XGBoost</i> - Avenida Gustavo Eiffel	58
A.6	<i>Random Search</i> - Rua Nova da Alfândega	59
A.7	<i>XGBoost</i> - Rua Nova da Alfândega	60
A.8	<i>Script predict_tunning.py</i>	61
A.9	<i>Script predict.py</i>	63
A.10	<i>Script talos.py</i> (Talos)	68



Lista de Tabelas

1	Resultados - <i>Random Search</i>	41
2	Resultados - <i>XGBoost</i>	41
3	Resultados - <i>TALOS</i>	46
4	Resultados - previsão <i>Análise Ruas</i>	49



Lista de Figuras

1	Metodologia CRISP-DM [1]	7
2	Tipos de chuva	13
3	Dados numéricos	14
4	Ruas afetadas	15
5	Dados numéricos	17
6	Média dos dados em cada rua	17
7	<i>Pie Chart Functional Road Class</i>	18
8	Dados numéricos	20
9	Tipos de luminosidade	20
10	<i>Java Snippet label_day</i>	22
11	Remoção atributos indesejados	23
12	<i>Manipulation Traffic Flow</i>	23
13	Divisão dos dados por cada rua	24
14	<i>Manipulation Traffic Incidents</i>	24
15	Remoção atributos indesejados	25
16	Remoção atributos indesejados	26
17	<i>Manipulation Weather Description</i>	31
18	Remoção atributos indesejados	31
19	Remoção de informação duplicada	32
20	<i>Manipulation Weather Until</i>	32
21	Criação dos datasets finais	33
22	<i>Inner Join</i>	33
23	<i>Right Outer Join</i>	34
24	Manipulação final	34
25	Remoção atributos indesejados	35
26	Agregação por rua	35
27	Criação de horas em falta	36
28	Criação Date&Time	36
29	<i>Datasets</i> com novas horas	37
30	Avenida dos Aliados - <i>Random Search</i>	39
31	Avenida dos Aliados - <i>XGBoost</i>	40
32	Modelo da rede construída	43
33	Previsão Avenida Gustavo Eiffel	44
34	Processo até à previsão	44
35	<i>Overfitting vs Underfitting vs Ideal Balance</i>	47
36	Modelo Avenida dos Aliados	47
37	Modelo Avenida Gustavo Eiffel	48
38	Modelo Rua Conde Vizela	48
39	Modelo Rua Nova Alfândega	49
40	Previsões Avenida dos Aliados	50
41	Previsões Rua Conde de Vizela	50
42	Previsões Avenida Gustavo Eiffel	50
43	Previsões Rua Nova da Alfândega	51
44	<i>Workflow</i> completo em KNIME do tratamento de dados	54
45	Rua Conde de Vizela - <i>Random Search</i>	55
46	Rua Conde de Vizela - <i>XGBoost</i>	56
47	Avenida Gustavo Eiffel - <i>Random Search</i>	57
48	Avenida Gustavo Eiffel - <i>XGBoost</i>	58
49	Rua Nova da Alfândega - <i>Random Search</i>	59
50	Rua Nova da Alfândega - <i>XGBoost</i>	60



1 Introdução

Neste capítulo é feita uma contextualização do projeto e da motivação que levou ao desenvolvimento do mesmo. Os objetivos são expostos e é apresentada uma estrutura geral do relatório.

1.1 Contextualização

Nos dias de hoje, a segurança rodoviária é um problema cada vez mais importante e que tem vindo a aumentar ao longo do tempo com o número de acidentes e de mortes nas estradas. Estas mortes e acidentes são muitas vezes provocados por possíveis congestionamentos e distrações dos condutores nas estradas. Outro fator que promove bastante estes acontecimentos são as condições atmosféricas que se verificam no momento.

Este trabalho consiste em conceber e implementar modelos de Deep Learning baseados em Redes Neuronais Long short-term Memory (LSTM) ou MultiLayer Perceptron (MLP) com o objetivo do mesmo ser capaz de prever o fluxo de trânsito e incidentes numa cidade determinada como alvo de estudo. Neste caso, a cidade escolhida pelo grupo foi a cidade do Porto, ou seja, esta será o nosso alvo de estudo.

1.2 Motivação

O fator que mais motivou o grupo a realizar este projeto é o facto do nosso contributo poder auxiliar as diferentes entidades rodoviárias, como as autoridades, condutores, pedestres, entre outros intervenientes diretos ou indirectamente envolvidos, para que estes estejam mais protegidos nas estradas e que possam ser ajudados na prevenção de acidentes rodoviários.

A possibilidade de se poder diminuir o risco de acidentes ou evitar congestionamentos, despertou no grupo a vontade de realizar o projeto da melhor forma possível.

1.3 Objetivos

O foco principal deste projeto é a previsão do fluxo de tráfego rodoviário, que passará pela análise completa dos diferentes tipos de dados fornecidos, para que seja possível realizar o tratamento dos mesmos e posteriormente realizar a previsão definida como objetivo.

Uma análise completa e detalhada, tal como foi mencionado, será um ponto fulcral deste projeto, uma vez que é necessário averiguar todos os potenciais riscos para que a previsão seja a mais assertiva possível.

1.4 Estrutura do relatório

Este relatório aborda as várias etapas que são necessárias para a realização deste projeto. Numa primeira etapa é necessário escolher a metodologia a utilizar ao longo do projeto. Numa segunda etapa, o objetivo passa por avaliar o problema, avaliando os seus desafios e a abordagem escolhida para a resolução do mesmo.

De seguida, será necessário analisar todos os dados recebidos para na etapa seguinte ser possível realizar o tratamento dos mesmos. Numa quinta etapa, é construída a rede neuronal treinada com os dados tratados.

Para terminar, serão analisados todos os resultados obtidos e retiradas as devidas conclusões, fazendo uma breve análise para trabalho futuro.

2 Metodologia

Neste capítulo, iremos apresentar a metodologia que iremos utilizar para a planeamento deste projeto prático.

2.1 CRISP-DM

Existem várias metodologias, como SEMMA, KDD e CRISP-DM, que permitem o desenvolvimento eficiente de um projeto, no entanto a nossa escolha recaiu sobre CRISP-DM. A CRISP-DM é a abreviatura de *Cross Industry Standard Process for Data Mining* e consiste numa versão estendida da metodologia KDD e permite atualizar o modelo em qualquer estágio do mesmo. Na figura seguinte, pode-se visualizar as diferentes fases desta metodologia.

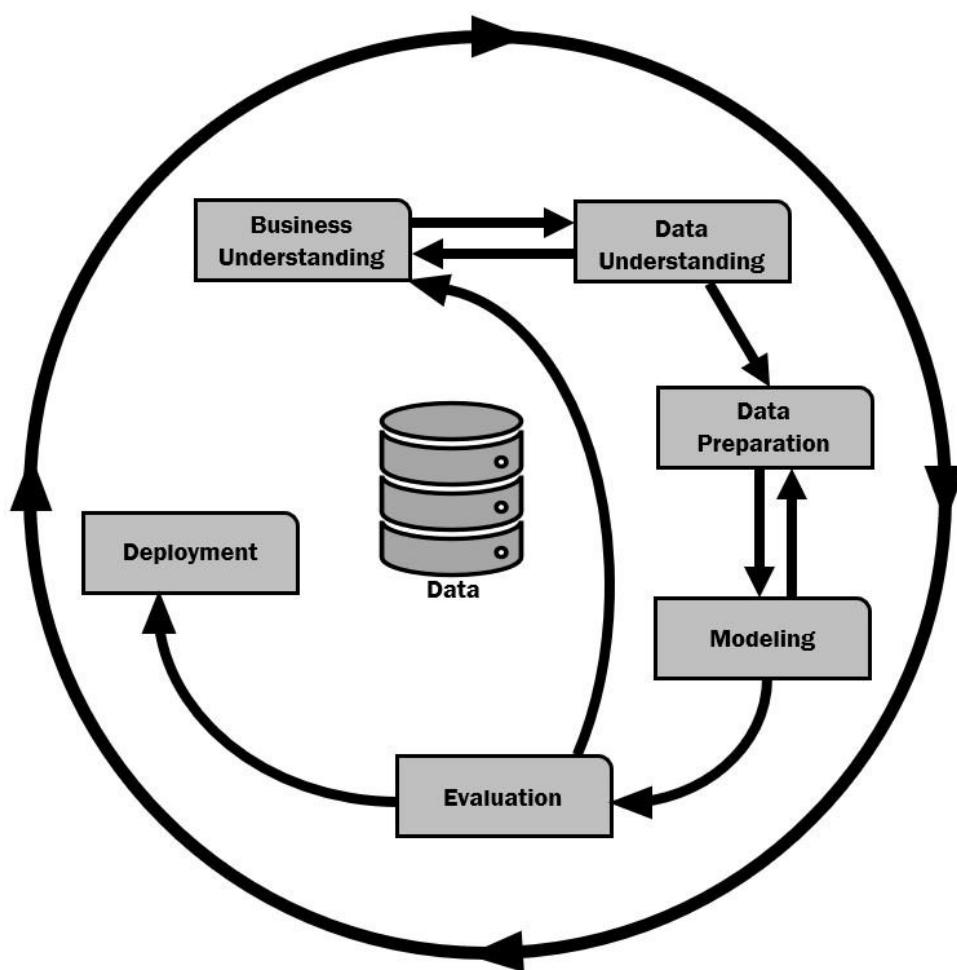


Figura 1: Metodologia CRISP-DM [1]

2.2 Fases

Nos próximos subcapítulos vamos descrever em que consiste cada fase tal como apresentado em [3], [4] e [5]

2.2.1 Business understanding

A primeira fase passa por identificar o problema a ser resolvido consoante o panorama atual em que fatores como stock de recursos, requisitos, riscos e custos e benefícios são colocados em cima de mesa para viabilização do projeto. Nesta primeira fase é necessário definir objetivos, plano de mineração de dados e plano de trabalhos.

- **Definição de objectivos:** Traduzir o problema para um cenário empresarial, estabelecer uma relação cliente/empresa. Definição de componentes a serem introduzidas no mercado, e análise e comparação de produtos provenientes de competidores.;



- **Plano Mineração de dados:** Interpretar o problema em objectivos, especificar o estilo e os critérios do problema a ser resolvido. Descrever os outputs esperados do projecto e a forma como devem ser definidos
- **Plano de trabalhos:** Definir um processo inicial, discutir a sua execução entre os intervenientes. Identificar os objectivos principais e as técnicas a serem utilizadas para a resolução desse problema. Nesta fase já são descritas as ferramentas a serem usadas para implementar o projecto. Realçar quais serão à partida os principais obstáculos na execução do projecto

2.2.2 Data understanding

Esta fase é o primeiro embate com os dados a serem trabalhados, primeiro passa por adquiri-los e passá-los para a ferramenta ideal que permita observá-los e ter uma noção que tipo de dados estão a ser coleccionados.

- **Colecção de dados:** indicar de forma transparente a origem dos dados, que métodos foram usados para obtê-los, p.e, se são de acesso privado, quais as credenciais fornecidas para ter acesso autorizado. Indicar quantitativamente os dados em cada campo e se cumprem critérios definidos anteriormente;
- **Exploração de dados:** Esta fase pode ser feita a “olho” ou de forma mais complexa, usar *queries* específicas para analisar em detalhe certas componentes, p.e, atributos chave, relações entre atributos, resultado de agregações, atributos com frequência redundante e estatísticas gerais;
- **Plano de trabalhos:** Definir um processo inicial e discutir a sua execução com os intervenientes. Identificar os objectivos principais e as técnicas a serem utilizadas para a resolução desse problema. Nesta fase já são descritas as ferramentas a serem usadas para implementar o projecto. Realçar quais serão à partida os principais obstáculos na execução do projecto;
- **Qualidade dos Dados:** Aqui são respondidas algumas questões sobre o estado dos dados, p.e, se os dados obtidos estão completos, se todos os casos estão abrangidos, se são valores admissíveis e não falsos, se há valores em falta e, se sim, se são frequentes?

2.2.3 Data preparation

Selecionam-se os dados que vão ser usados para análise. O critério de escolha será com base nos dados que melhor se adequam a minerar. Esta componente envolve gastar bastante tempo e recursos durante a implementação de um produto, derivado á passagem de uma componente de dados crua para uma pronta para ser consumida.

- **Selecionar Dados:** Escolha e análise de dados de forma racional, bem como a explicação pela selecção de alguns dados e outros não;
- **Limpar Dados:** Subsets que não interessem são descartados, inserção ou adaptação de dados pode ocorrer para melhorar a qualidade dos dados, técnicas de modelação são recomendadas para este efeito. Estas acções devem ser sempre justificadas;
- **Construção de dados :** Operações que envolvem derivações de atributos ou criação de registo;
- **Derivação de Atributos:** Através de atributos já existentes, criar novos atributos a partir desse próprio registo;
- **Criação de registo:** Registos que não aparecem nos dados iniciais mas que são necessários para criar um modelo são introduzidos de modo a explicitar determinados casos;
- **Integração de Dados:** Após modificações nos dados é necessário juntar de forma correta e coerente, e existem duas formas de o fazer: unir ou agrregar. Unir dados - Tabelas são adicionadas de forma conjunta de acordo com o atributo em questão;
- **Agregações:** Após consulta dos valores referentes a múltiplos registo e/ou tabelas, agregá-los numa só entrada;



2.2.4 Modeling

Aqui é selecionada qual a técnica de modelação a ser usada. Apesar de nesta altura de implementação já haver uma ideia prévia de qual será a mais apropriada, aqui será escolhida com maior detalhe, p.e, regressão linear, clustering hierárquico, redes neuronais etc... Mas caso seja usada mais que uma abordagem esta subfase deve ser aplicada independentemente.

A cada modelo deve ser interpretado consoante os conhecimentos na área, se ocorreu sucesso ou se ainda há parâmetros a modificar, de realçar que esta fase apenas considera modelos e não a respectiva avaliação. Os modelos testados nesta fase devem ser comparados e se possível criar um rank consoante a performance e/ou complexidade do mesmo

- **Técnica de Modelação:** Documentar qual a técnica a ser usada;
- **Afirmações do modelo:** Qualquer abordagem perante os dados e adaptações consoante o modelo a ser implementado devem ser explicadas aqui;
- **Gerar Protótipo:** Testar e avaliar a qualidade de um modelo protótipo antes de proceder a uma fase seguinte, só após testes a datasets específicos para treino e testes, e consequente validação, se constrói um modelo;
- **Definição de parâmetros:** Com a ferramenta previamente escolhida, explorar e adaptar os diferentes parâmetros oferecidos para melhor inclusão do dataset a ser usado;
- **Modelos:** Primeiro modelo criado por uma ferramenta numa componente prática e não teórica;
- **Descrição do Modelo:** Após testes, apontar os resultados obtidos e interpretá-los de acordo com expectativas e definir advertências encontradas ;
- **Revisão do modelo:** Resumir os resultados desta tarefa, através dos prós e contras através da performance observada;
- **Revisão dos parâmetros do modelo:** A cada iteração do modelo modificar os parâmetros definidos de modo a obter melhor performance do modelo;

2.2.5 Evaluation

Após os resultados obtidos por vários modelos, deve-se escolher qual o modelo que oferece melhores resultados para o dataset em questão, e analisarmeticulously cada desvantagem desse modelo, porque uma desvantagem forte pode impedir a execução do projecto, p.e, performance em tempo real, apesar de ser um modelo fiável, pode não ser eficiente em termos de performance e tempo e isto pode criar problemas de aplicação a nível real.

Caso os modelos sejam satisfatórios em todos os aspetos, deve-se rever todo o processo desde o ínicio até este ponto de modo a assegurar-se que todos os passos foram bem executados e nenhum interveniente foi ignorado.

Após todos estes checkpoints estarem analisados e validados o próximo passo será a implementação junto das entidades responsáveis e o ambiente indicado.

- **Análise dos resultados de data mining:** Comparar os resultados obtidos com os planos iniciais do projeto, se cumprem os requisitos implementados;
- **Modelos Aprovados:** Todos os modelos que estão conforme os critérios e requisitos são considerados aprovados para futuras fases;
- **Revisão do processo:** Resumir todos os processos de revisão e realçar todas actividades que possam ter sido ignoradas ou desprezadas;
- **Lista de possíveis acções:** Listar todas as potenciais acções, devidamente justificadas a aplicar nos próximos passos;
- **Decisão:** Descrever como se deve proceder para o próximo passo de forma racional;



2.2.6 Deployment

Com todas as análises dos resultados previamente obtidos, é agora altura de determinar uma estratégia de implementação no mercado, toda a documentação que servirá de suporte para manutenção e suporte do projecto, todos os aspectos empresariais tem que ser definidos junto com as entidades responsáveis, esta fase é crucial para o sucesso do projeto, porque aqui não depende da equipa que desenvolveu mas do mercado. Manutenção e suporte tem de ser acompanhados nesta fase dura de implementação, que pode ter prazo alargado.

E por fim fazer uma análise ao projeto em geral, realçar pontos que correram bem e outros nem tanto.

Plano de Implementação - Resumir a estratégia de implementação e todos os passos necessários para atingi-la.

- **Plano de Implementação:** Resumir a estratégia de implementação e todos os passos necessários para atingi-la;
- **Monitorização e Suporte:** Definir uma estratégia de suporte e manutenção;
- **Relatório Final:** Relatório escrito de acordo com o processamento de síntese de mineração de dados que inclui todos os passos do projecto descritos de forma resumida;
- **Apresentação Final:** Passo opcional, em que o projecto é apresentado ao cliente com resultados obtidos como prova de decisão;
- **Documentação:** Todos os pontos envolvidos durante o projecto, anomalias, bugs, imprevistos devem estar escritos num só sítio, para que intervenientes no futuro tenham noção da cronologia temporal;



3 Problema

Neste capítulo, começa-se por fazer uma abordagem à análise e tratamento dos dados que se irá realizar, passando pela construção do modelo, por todos os seus desafios com os quais o grupo se irá deparar e definindo o plano de desenvolvimento do projeto.

Este problema aborda os temas de Deep Learning e Redes Neuronais, o que é normal a existência de obstáculos ao longo da execução do projeto. No entanto, com os conhecimentos adquiridos na UC os desafios vão ser mais fáceis de ultrapassar.

3.1 Análise e tratamento de dados

Com os dados recebidos pretende-se construir e treinar uma rede neuronal, no entanto é necessário que todos esses dados que foram fornecidos passem por um processo detalhado de análise para que o grupo determine quais os dados são realmente necessários no conjunto de dados final. Os dados que serão colocados no *dataset* final podem estar sujeitos a algum tipo de tratamento como *parsing* dos dados ou *labelling* dos mesmos.

3.2 Construção do modelo

Na construção do modelo é necessário receber um ficheiro com todos os dados essenciais tratados que irão contribuir para o treino da rede, no entanto deverão aplicar um de vários tipos de tratamento antes como a normalização ou o *one hot encoding* para que possa ser utilizado corretamente pela rede.

3.3 Desafios

Este projeto é composto por vários desafios desde a investigação minuciosa que é necessário realizar face aos dados fornecidos até à criação do modelo de previsão final. Será necessário tomar decisões, pois muitas vezes os dados apesar de terem alguma importância, podem não ter nenhum peso no treino do modelo de previsão.

O maior desafio será, como já foi referido várias vezes, o tratamento completo dos dados, uma vez que este desafio esteja concluído, à partida o modelo a elaborar será bem treinado e validado.

3.4 Plano de desenvolvimento

De modo a resolver este problema bem como responder a estes desafios, o desenvolvimento deste projeto será composto por 3 fases principais:

- **Análise de dados**
- **Tratamento de dados**
- **Desenvolvimento de um modelo de previsão**

Numa primeira fase será realizada uma análise detalhada de todos os dados que nos são fornecidos, permitindo que na fase do tratamento seja mais fácil averiguar quais os dados que serão necessários manter, quais serão removidos e quais serão alterados.

Já com os dados tratados, espera-se obter um conjunto de dados final para se utilizar na construção e treino da rede neuronal a utilizar para a previsão do tráfego de trânsito na cidade do Porto.



4 Análise de dados

Neste capítulo, vamos analisar todos os dados fornecidos para que mais tarde seja possível realizar o tratamento dos mesmos da forma mais adequada. Como será explicado posteriormente, existem dados concretos sobre ruas específicas da cidade do Porto que irão ser o nosso foco de estudo e intervenção neste tratamento de dados.

Para realizar esta análise de dados, para além de uma pesquisa sobre cada tipo de dados, iremos utilizar a ferramente KNIME, uma ferramenta que o grupo tem alguma experiência e achou pertinente utilizar, para explorar os dados. Nas secções seguintes iremos fazer uma análise individual de cada conjunto de dados.

4.1 Descrição dos *datasets*

Como referido no capítulo anterior, iremos utilizar os quatro *datasets* referentes à cidade do Porto. Cada um destes *datasets* contém informações diferentes que iremos analisar detalhadamente.

O primeiro *dataset*, **Weather Description**, contém informação referente ao estado do tempo na cidade do Porto.

O segundo *dataset*, **Traffic Incidents**, contém informação sobre os incidentes que ocorreram nas estradas da cidade, bem como as estradas que cada incidente poderá afectar e a caracterização do incidente.

O terceiro *dataset*, **Traffic Flow**, contém informação detalhada sobre o fluxo de trânsito em quatro estradas da cidade do Porto: Avenida dos Aliados, Rua Conde de Vizela, Avenida Gustavo Eiffel e Rua Nova da Alfândega.

O quarto *dataset*, **Weather Until**, contém informação complementar ao *dataset* **Weather Description** sobre o estado do tempo na cidade do Porto.

Todos os *datasets* serão explicados e analisados detalhadamente nos próximos capítulos.

4.2 Análise do *dataset Weather Description*

Começando pelo conjunto de dados *Weather Description*, verificamos que neste *dataset* existem os seguintes atributos:

- *city_name* - Nome da Cidade (*String*)
- *cloudiness* - Nebulosidade (*String*)
- *atmosphere* - Nebulosidade (*String*)
- *snow* - Neve (N/A)
- *thunderstorm* - Intensidade de Trovoada (*String*)
- *rain* - Intensidade da Chuva (*String*)
- *sunrise* - Data do Nascer do Sol (*String*)
- *sunset* - Data do Pôr do Sol (*String*)
- *creation_date* - Data de criação do registo (*String*)

Verificados todos os dados disponíveis neste *dataset*, decidimos avaliar a quantidade de variáveis que contém cada tipo de dados através de um *Data Explorer* e alguns *Pie Charts*.

No caso do exemplo seguinte, onde podemos observar que grande parte dos valores são *missing values*, ou seja, é o mesmo caso de "N/A", não ocorreu qualquer tipo de chuva. Já o resto dos valores são respetivos aos tipos de chuva em concreto.

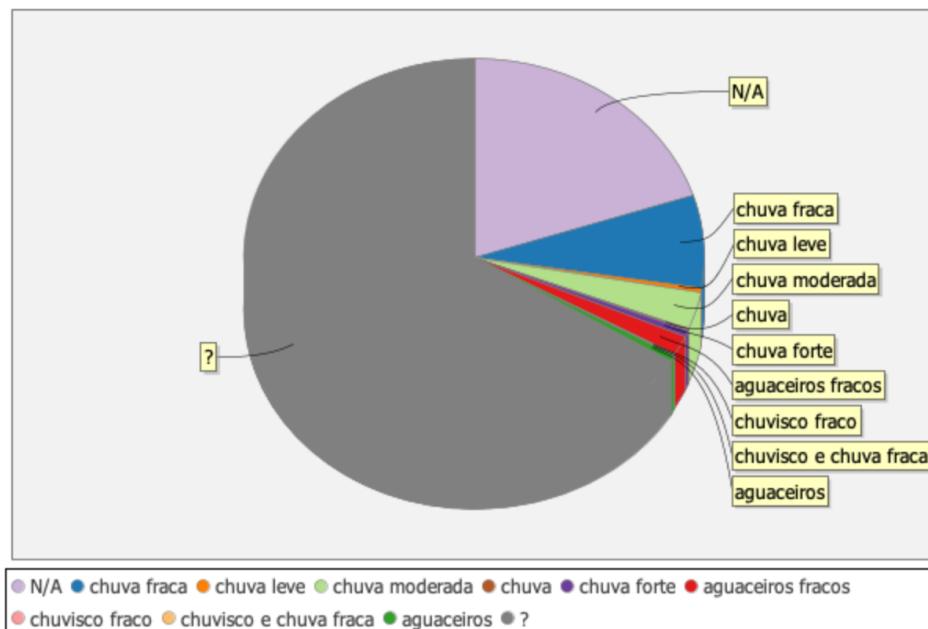


Figura 2: Tipos de chuva

Posto isto, é necessário fazer uma análise detalhada de cada atributo para que seja possível saber que decisões tomar face a cada um deles.

4.2.1 *city_name*

Para este atributo a decisão a tomar passa pela sua remoção, uma vez que já sabemos qual é a cidade que estamos a trabalhar e é sempre a mesma, logo não é necessária a informação.

4.2.2 *cloudiness*

Quanto a este atributo pode-se afirmar que pode ter influência no trânsito, pois quanto mais nuvens tiver no céu mais maior a probabilidade de chuva e pior será a visibilidade.

4.2.3 *atmosphere*

Este atributo irá ter bastante peso numa previsão de trânsito, uma vez que a existência de nevoeiro na estrada pode condicionar o trânsito. Existem vários níveis de intensidade de nevoeiro, quanto mais denso for, menor será a visibilidade da estrada.

4.2.4 *snow*

A neve é outro fator que pode condicionar bastante o trânsito rodoviário. Caso exista queda de neve, o trânsito pode ser cortado, existe grande probabilidade de existência de acidentes rodoviários, logo será um fator a ter em conta.

Neste caso, não existe nenhum dia em que houve queda de neve, logo à partida será um dado a não considerar apesar da importância do mesmo.

4.2.5 *thunderstorm*

Relativamente à trovoada, pode ser um fator a ter em consideração no trânsito das estradas, uma vez que pode causar problemas em veículos ou problemas elétricos relacionados com os semáforos, como aconteceu em 2019 na cidade do Porto.[2] Nessa altura a trovoada afetou 127 semáforos no Porto, o que poderia ter originado vários acidentes de viação.

4.2.6 *rain*

Quanto à existência ou não de chuva, é importante referir que este é um atributo bastante importante. Esta causa natural pode levar à criação de vários acidentes de viação provocando trânsito nas estradas.



Sabemos que o aumento da nível da chuva, ou seja, quanto mais chuva, maior irá ser a probabilidade de existirem colisões nas estradas, acidentes, aumentando o trânsito nas mesmas.

4.2.7 sunrise

O nascer do sol é um fator da luminosidade, não tendo um contributo benéfico nem prejudicial para os intervenientes rodoviários, uma vez que os condutores com a falta de luminosidade podem utilizar os faróis para aumentar a luminosidade ou vice-versa.

4.2.8 sunset

Semelhante ao nascer do sol, o pôr-do-sol é um fator da luminosidade, não tendo mais uma vez um contributo benéfico nem prejudicial para os intervenientes rodoviários.

4.2.9 creation_date

Face à data, esta vai ser importante para saber a altura a que foram captados os dados e importante para se realizar o agrupamento dos dados com os restantes conjuntos de dados.

4.3 Análise do dataset *Traffic Incidents*

Analisando agora este segundo *dataset*, verificamos que existem os seguintes atributos:

- *city_name* - Nome da Cidade (*String*)
- *description* - Descrição do incidente (*String*)
- *cause_of_incident* - Causa do incidente (*String*)
- *from_road* - Rua do inicio do incidente (*String*)
- *to_road* - Rua do fim do incidente (*String*)
- *affected_roads* - Ruas afetadas pelo incidente (*String*)
- *incident_category_desc* - Descrição da categoria do incidente (*String*)
- *magnitude_of_delay_desc* - Descrição do atraso que o incidente causou (*String*)
- *length_in_meters* - Tamanho da fila em metros causada pelo incidente (*Numeric*)
- *delay_in_seconds* - Atraso em segundos causado pelo incidente (*Numeric*)
- *incident_date* - Data em que foi registo o incidente (*String*)
- *latitude* - Latitude da localização do incidente (*Numeric*)
- *longitude* - Longitude da localização do incidente (*Numeric*)

Mais uma vez decidiu-se verificar todos os dados disponíveis neste *dataset*, decidimos avaliar a quantidade de variáveis que contém cada tipo de dados através de um *Data Explorer* e alguns *Pie Charts*.

Começando por verificar os dados numéricos, destes dados os que à partida serão para manter são o *length_in_meters* e o *delay_in_seconds*, uma vez que são fatores que podem indicar a existência de trânsito, pois quanto maior for uma fila de trânsito ou quanto maior for o atraso, consequentemente maior é a intensidade de trânsito rodoviário.

Column	Exclude Column	Minimum	Maximum	Mean	Standard Deviation	Variance	Skewness	Kurtosis
length_in_meters	<input type="checkbox"/>	0	19670230	1148.681	103731.528	10760229863.670	189.595	35947.377
delay_in_seconds	<input type="checkbox"/>	0	9177	231.782	227.966	51968.430	5.691	88.360
latitude	<input type="checkbox"/>	41.063	42.373	41.161	0.018	0.000	13.796	915.768
longitude	<input type="checkbox"/>	-87.905	-8.555	-8.629	0.553	0.306	-142.985	20476.060

Figura 3: Dados numéricos

Optou-se por analisar um *Pie Chart* dos dados do atributo *affected_roads*. Como se pode verificar pela figura seguinte existem bastantes *missing values* (são a maioria dos dados) e os restantes dados à partida não serão utilizados, visto que nenhum dos nomes das ruas corresponde às ruas que se irão tratar.

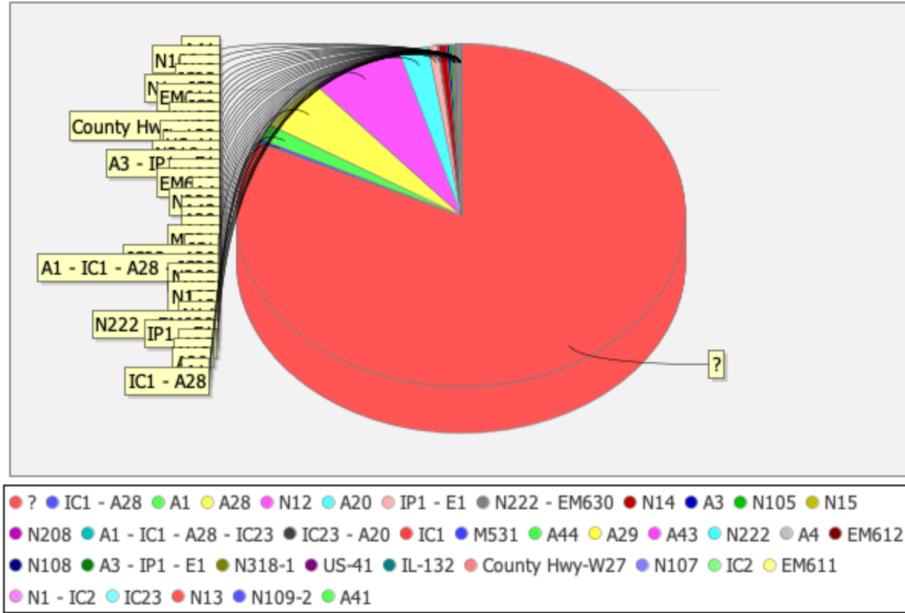


Figura 4: Ruas afetadas

À semelhança do que aconteceu no conjunto de dados anterior, iremos analisar cada atributo individualmente.

4.3.1 *city_name*

Começando pelo nome da cidade, tal como aconteceu no conjunto de dados anterior, vai ser um atributo a não considerar, pois é sempre a mesma cidade e não há necessidade de ter em consideração.

4.3.2 *description*

Quanto à descrição podemos dizer que pode ser um atributo a ter em consideração, uma vez que é possível retirar o nível de trânsito através da sua descrição, ou seja, é possível saber se há mais trânsito ou menos trânsito a determinada hora.

4.3.3 *cause_of_incident*

Relativamente ao atributo da causa do incidente podemos informar que não será um fator a ter em consideração, pois existem bastante *missing values* e poucos dados que podem ser utilizados.

4.3.4 *from_road*

Quanto a este atributo podemos afirmar que é um atributo a ter em consideração. A partir da informação que este atributo nos fornece, é possível determinar se o incidente foi numa das ruas que estão a ser tratadas.

Assim, caso alguma das ruas que estão a ser tratadas estiverem presentes neste atributo é indicativo da existência de trânsito nessa zona, logo é importante para a previsão final.

4.3.5 *to_road*

A importância deste atributo é semelhante à do atributo anterior, pois um acidente pode ter começado num local e estender-se até outro local. Logo também se pode e deve utilizar este atributo para verificar se alguma das ruas corresponde a alguma das que estão a ser tratadas.

4.3.6 *affected_roads*

Em relação às ruas afetadas e tendo em consideração o que foi referido anteriormente, poderia ser um atributo de alguma importância, pois deveria ser possível verificar se alguma das ruas afetadas foi uma das que estão a ser analisadas. No entanto, existem inúmeros *missing values* e as ruas que são mencionadas não são as que estão a ser tratadas.



4.3.7 *incident_category_desc*

Avançando para a categoria de cada incidente podemos afirmar que será um atributo que não iremos ter em consideração, pois verificando os dados deste atributo é possível concluir que a maioria destes são iguais (têm o valor "jam") e não terão relevância numa futura previsão devido a isso.

4.3.8 *magnitude_of_delay_desc*

Em relação à magnitude do incidente é possível retirar informação que nos permitirá perceber a gravidade do incidente, logo a partir disto é possível saber se existe mais ou menos trânsito.

4.3.9 *length_in_meters*

Este atributo é um forte indicativo da existência de trânsito rodoviário, uma vez que quanto maior for a distância maior será a fila ou o engarrafamento de trânsito afetando não só a rua onde aconteceu o acidente como as localidades em redor.

4.3.10 *delay_in_seconds*

O atributo *delay_in_seconds* é uma atributo que podemos comparar com o anterior, *length_in_meters*, pois se o *delay* for significativo significa que existem filas ou engarrafamentos, o que leva a concluir que existe trânsito na zona do incidente. O contrário também pode ser inferido caso o *delay* for baixo ou inexistente.

4.3.11 *incident_date*

Quanto à data do incidente podemos afirmar que será importante para o tratamento de dados, pois terá a mesma função que o atributo *creation_date* do *dataset* anterior, auxiliar a agrupar todos os dados.

4.3.12 *latitude*

A latitude será um atributo a não ter em consideração, uma vez que já existe informação relativa à rua onde iniciou o incidente, onde terminou o incidente e as ruas que ficaram afetadas.

4.3.13 *longitude*

A longitude será um atributo a não ter em consideração tal como a latitude.

4.4 Análise do *dataset Traffic Flow*

Avançando para a análise do terceiro conjunto de dados, verificamos mais uma vez os atributos que constituem este conjunto:

- *city_name* - Nome da Cidade (*String*)
- *road_num* - Número da Rua (*Numreic*)
- *road_name* - Nome da Rua (*String*)
- *functional_road_class_desc* - Descrição do tipo de rua (*String*)
- *current_speed* - Velocidade atual de circulação na rua (*Numreic*)
- *free_flow_speed* - Velocida de circulação na rua sem trânsito (*Numreic*)
- *speed_diff* - Diferença entre a velocidade máxima na rua e a velocidade atual (*Numreic*)
- *current_travel_time* - Tempo atual que demora a percorrer a rua (*Numreic*)
- *free_flow_travel_time* - Tempo que demora a percorrer a rua sem trânsito (*Numreic*)
- *time_diff* - Diferença entre o tempo mínimo e o atual que demora a percorrer a rua (*Numreic*)
- *creation_date* - Data em que foram recolhidos os dados (*String*)



Tal como no *dataset* anterior, verificados os atributos existentes, utilizou-se um *Data Explorer* e alguns *Pie Charts* para ter uma noção mais detalhada de cada um deles.

Novamente verificou-se os dados numéricos e pode-se concluir que, para além deste conjunto de dados conter dois atributos que nos interessa prever, o *speed_diff* e o *time_diff*, os restantes dados numéricos, com exceção do *road_num*, terão um papel importante na previsão e cada um deles está disperso numa gama de valores que nos permite executar um bom tratamento neles.

É importante mencionar que a partir dos atributos *current_speed*, *free_flow_speed*, *current_travel_time* e *free_flow_travel_time* é possível calcular o *speed_diff* e o *time_diff*, logo são dados que definitivamente devem ser incluídos na previsão.

Column	Exclude Column	Minimum	Maximum	Mean	Standard Deviation	Variance	Skewness	Kurtosis
road_num	□	1	4	2.506	1.119	1.253	-0.006	-1.363
current_speed	□	0	39	20.841	10.136	102.734	0.071	-1.335
free_flow_speed	□	0	45	24.037	9.532	90.865	-0.179	-1.517
speed_diff	□	0	43	3.197	6.188	38.294	2.488	6.676
current_travel_time	□	0	1859	117.512	103.706	10755.026	2.938	16.872
free_flow_travel_time	□	0	214	88.389	55.767	3110.013	0.610	-1.162
time_diff	□	0	1720	29.123	84.552	7149.036	5.325	40.220

Figura 5: Dados numéricos

Ainda relativo aos dados anteriores, optou-se por construir um gráfico com a média dos valores numéricos para que fosse possível compreender a dispersão desses mesmos valores.

Como se pode observar pelo gráfico, não importa que estes sejam muito altos ou muito baixos, mas sim a diferença entre eles, pois o *time_diff* corresponde à diferença entre o *current_travel_time* e o *free_flow_travel_time*, já o *speed_diff* corresponde à diferença entre o *current_speed* e o *free_flow_speed*. Percebemos então que quanto maior for a diferença entre cada par de valores, maior será os respetivos resultados, ou seja, o *time_diff* e o *speed_diff*.

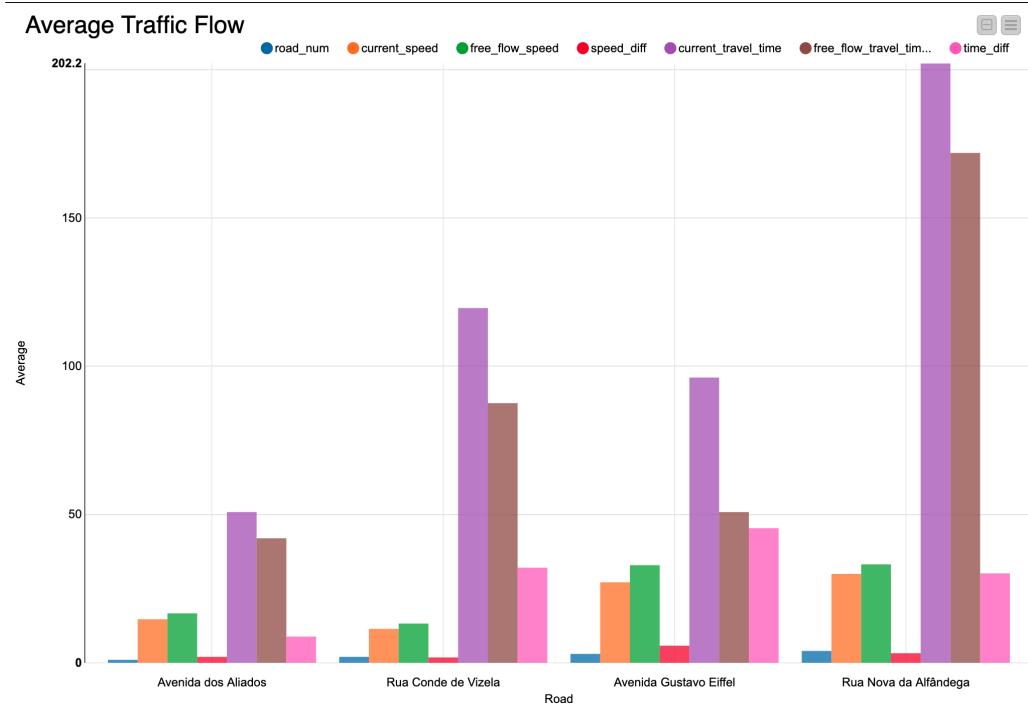


Figura 6: Média dos dados em cada rua

Depois de analisar os dados numéricos, verificou-se alguns gráficos dos atributos disponíveis, como o exemplo da figura seguinte.

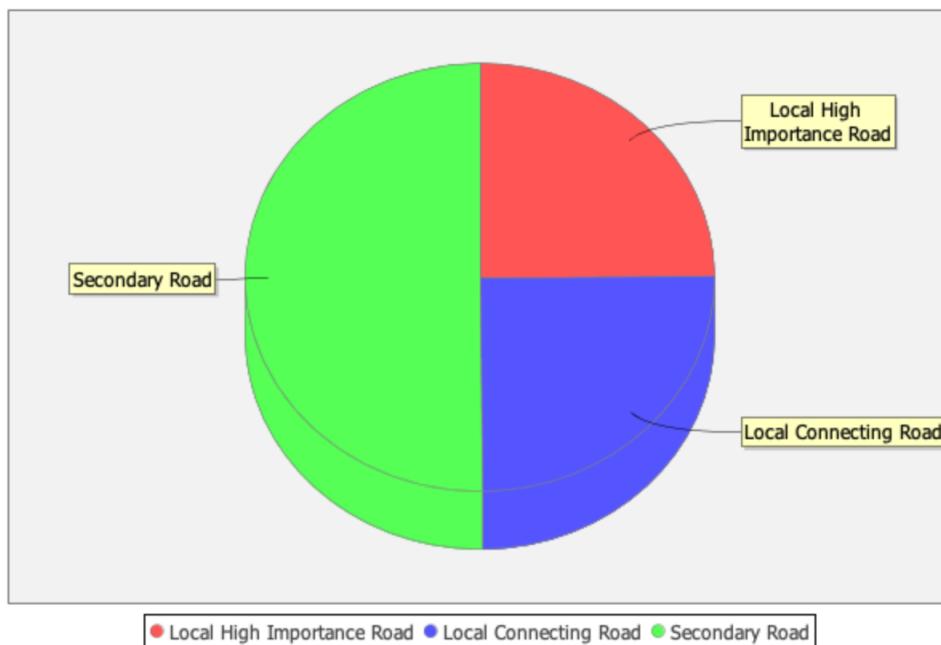


Figura 7: Pie Chart Functional Road Class

Analisando os dados do atributo relativo à classe da rua, podemos constatar que existem três diferentes tipos de ruas: *Secondary Road*, *Local High Importance Road* e *Local Connecting Road*. Os três diferentes tipos de rua têm importâncias diferentes, o que nos pode ajudar, uma vez que se existir mais trânsito numa rua secundária não é tão significativo como se existisse mais trânsito numa *Local High Importance Road*.

No entanto, não podemos afirmar se será um dado a utilizar numa futura previsão, pois como só existem quatro ruas, este atributo só nos fornece informação dessas mesmas quatro ruas, logo será o mesmo que manter o atributo *road_name* que poderá ser mais útil ainda para o agrupamento final de dados.

Mais uma vez, vamos realizar uma análise individual de cada atributo presente neste *dataset*.

4.4.1 *city_name*

Começando pelo atributo correspondente ao nome da cidade, mais uma vez será um atributo a descartar, uma vez que só estamos a analisar uma cidade, a cidade do Porto.

4.4.2 *road_num*

Quanto ao número da rua, podemos afirmar que também será um atributo a descartar, visto que existem apenas quatro ruas e este atributo apenas serve para numerá-las. No entanto, o facto de ser um atributo numérico pode facilitar na previsão, uma vez que é recomendada a utilização de dados numéricos e este poderia substituir perfeitamente o atributo seguinte, embora no caso deste projeto serão construídos conjuntos de dados finais para cada rua.

4.4.3 *road_name*

O nome da rua é uma atributo que indica o nome das quatro diferentes ruas. Pode ser utilizado para facilitar o agrupamento dos dados para cada rua, assim como o atributo anterior.

4.4.4 *functional_road_class_desc*

Quanto a este atributo, tal como foi referido anteriormente, será um atributo a não ter importância, uma vez que apenas identifica cada tipo de cada uma das quatro ruas.

4.4.5 *current_speed*

A velocidade atual é um fator importante a ter em consideração, uma vez que é um indicativo de que este pode estar na velocidade máxima permitida ou não e, consoante esse valor, pode-se retirar a conclusão da existência de trânsito ou não. Se o valor for igual



à velocidade máxima permitida não existe trânsito, no entanto se for bastante inferior é um dado indicativo da existência de trânsito naquela via.

4.4.6 *free_flow_speed*

Tal como foi referido anteriormente, este é um atributo a ter em consideração, pois é um indicativo do valor máximo que se pode atingir num cenário sem trânsito naquele momento e pelos factos que já foram referidos naquele atrás.

4.4.7 *speed_diff*

A diferença de velocidade é um atributo que tem de ser mantido, visto que é um dos dados que tencionamos prever.

Este atributo corresponde à diferença de velocidade entre a velocidade que os carros podem atingir num cenário sem trânsito e a velocidade que realmente se verifica.

4.4.8 *current_travel_time*

Relativamente ao tempo de viagem, podemos afirmar que é um atributo que se tem de ter em consideração para a previsão. Este atributo fornece a informação do tempo que durou a percorrer a rua, ou seja, se o valor for superior ao que deveria ser, pode considerar-se que existe trânsito naquela via e vice-versa.

4.4.9 *free_flow_travel_time*

É necessário ter em consideração este atributo, pois este indica-nos o valor que se demora a percorrer a rua quando não existe trânsito e pode ser utilizado como termo de comparação com o atributo anterior.

4.4.10 *time_diff*

Tal como o *speed_diff*, este é um dos atributos que pretendemos prever, logo é necessário mantê-lo no conjunto de dados final.

Este atributo corresponde à diferença do tempo que se devia demorar a percorrer a rua e o tempo que realmente demorou a percorrer a rua.

4.4.11 *creation_date*

Por último, é necessário manter a data de criação, pois será utilizar para agrupar os dados tal como já foi referido nos conjuntos de dados anteriores.

4.5 Análise do dataset *Weather Until*

Avançando agora para a análise do último conjunto de dados, verificamos a existência dos seguintes atributos:

- *city_name* - Nome da Cidade (*String*)
- *temperature* - Temperatura (*Numeric*)
- *atmospheric_pressure* - Pressão Atmosférica (*Numeric*)
- *humidity* - Humidade (*Numeric*)
- *wind_speed* - Velocidade do Vento (*Numeric*)
- *clouds* - Densidade da Cobertura de Nuvens (*Numeric*)
- *precipitation* - Nível de Precipitação (*Numeric*)
- *current_luminosity* - Luminosidade (*String*)
- *sunrise* - Data do Nascer do Sol (*String*)
- *sunset* - Data do Pôr do Sol (*String*)
- *creation_date* - Data em que foram recolhidos os dados (*String*)

Novamente optou-se por utilizar um *Data Explorer* e recorrer a análise de alguns atributos através de *Pie Charts*.

Analisando os dados numéricos disponíveis neste conjunto de dados verificamos que existem dados que potencialmente podem ser utilizados e que a gama de valores deles é significativa o que leva a crer que necessitarão de posterior tratamento.



Column	Exclude Column	Minimum	Maximum	Mean	Standard Deviation	Variance	Skewness	Kurtosis
temperature	<input type="checkbox"/>	0	32	15.173	5.144	26.460	0.241	0.151
atmospheric_pressure	<input type="checkbox"/>	990	1033	1017.864	5.956	35.472	-0.713	2.338
humidity	<input type="checkbox"/>	19	100	82.137	16.817	282.800	-1.078	0.804
wind_speed	<input type="checkbox"/>	0	14	3.342	2.174	4.728	0.930	1.128
clouds	<input type="checkbox"/>	0	100	34.747	34.740	1206.870	0.395	-1.492
precipitation	<input type="checkbox"/>	0	2	0.000	0.024	0.001	82.589	6821.000

Figura 8: Dados numéricos

Na figura seguinte é apresentado um gráfico dos dados do atributo *current_luminosity* e como é possível constatar existem três de luminosidade: luz, pouca luz e escuro. Estes três tipo de luminosidade podem ter influência no trânsito rodoviário, uma vez que os condutores, por norma, têm maior precaução na condução noturna, logo o risco da ocorrência de acidentes é menor o que leva a querer que existirá menos trânsito de noite, já de dia quando há luz estes estão mais desocupados o que pode levar a uma maior taxa de acidentes e, consequentemente, maior trânsito nas estradas. Pode-se verificar ainda que em grande parte dos dados o tipo de luminosidade é escuro e noutra grande parte dos dados o tipo de luminosidade é luz, apenas uma pequena percentagem dos dados diz respeito a pouca luz.

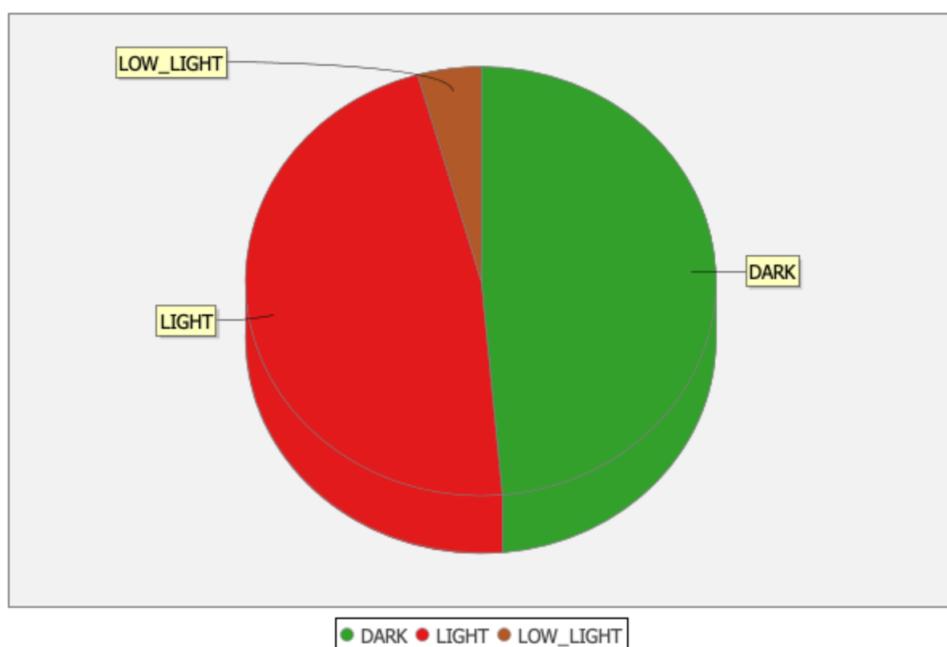


Figura 9: Tipos de luminosidade

Posto isto, vamos analisar individualmente cada um dos atributos tal como se fez nos restantes conjuntos de dados.

4.5.1 *city_name*

Novamente, este *dataset* é composto pelo atributo do nome da cidade, visto que esta é sempre a mesma, será um atributo a não ter em consideração.

4.5.2 *temperature*

Relativamente ao atributo da temperatura pode-se afirmar que é um fator que condiciona os trânsitos nas estradas. Nos dias de maior calor, o cansaço e a falta de atenção podem tomar conta das atitudes do condutor o que pode levar a acidentes rodoviários provocando trânsitos nas estradas. Para além de afetar o condutor, as diferentes temperaturas podem provocar problemas nos veículos como a falta de água nos dias mais quentes ou o piso molhado e escorregadio nos dias mais frios, que podem levar a acidentes ou congestionamentos nas estradas.[7]



4.5.3 *atmospheric_pressure*

Quanto à pressão atmosférica podemos afirmar que será um atributo a não ter em consideração, uma vez que os valores são constantes e não existem grandes variações, logo à partida será um atributo que não terá influência numa futura previsão.

4.5.4 *humidity*

Em relação à humidade é possível afirmar que é um dos atributos a ter em conta para uma possível previsão. A humidade à semelhança de outros atributos como a chuva ou a temperatura, é um fator que pode causar acidentes rodoviários. Num dia bastante húmido, as estradas podem provocar acidentes rodoviários e consequentemente trânsito nas estradas.[8]

4.5.5 *wind_speed*

No que diz respeito à velocidade do vento, podemos afirmar que será um fator a desconsiderar, uma vez que, apesar de o vento poder ser perigoso para os diferentes intervenientes rodoviários, se os valores da velocidade do vento não forem significativos, como é o caso, não é um fator que tenha importância para que seja capaz de causa trânsito nas estradas.

4.5.6 *clouds*

Acerca das nuvens, este é um atributo numérico que tem uma gama de valores de 0 a 100, ou seja, se o valor for 0 não existem nuvens, mas se o valor for 100, o céu está coberto de nuvens.

Quanto maior for o número de nuvens no céu, pior estará o tempo e maior será a probabilidade de chuva, que, consequentemente, será mais provável da ocorrência de incidentes nas vias. Com isto, achamos que será um atributo a ter em consideração.

4.5.7 *precipitation*

Em relação à precipitação é possível afirmar que é um atributo a não considerar, uma vez que todos os valores são 0 com a exceção de um único valor que é 2, ou seja, não irá ter interferência numa futura previsão.

4.5.8 *current_luminosity*

Quanto à *current luminosity* e tal como foi mencionado atrás, este atributo porventura terá influência numa futura previsão devido aos fatores que foram mencionados, o que nos leva a concluir que irá ser um atributo a considerar no conjunto de dados final.

4.5.9 *sunrise*

O nascer do sol é um fator de luminosidade que em pouco afeta os utilizadores na estrada, pois estes em caso da sua presença adequam-se com o uso de palas de sol ou óculos de sol para minimizar o impacto.

4.5.10 *sunset*

No mesmo cenário que *sunrise*, mas apesar num espaço temporal oposto, os utilizadores utilizam os mesmos procedimentos para reduzir o impacto criado por este fenómeno natural.

4.5.11 *creation_date*

Este atributo, na área dos valores relacionados com datas, será importante, pois descreve o espaço temporal em que foram recolhidos os dados e estes serão utilizados para que os dados retirados deste conjunto de dados sejam agrupados aos restantes para construir o *dataset* final.



5 Tratamento de dados

Neste capítulo é realizada a explicação do processo detalhado de tratamento de dados, que foi elaborado com base no que foi analisado no capítulo anterior.

Mais uma vez a ferramenta utilizada para realizar este tratamento de dados foi o KNIME e nas secções seguintes será detalhado todo o processo realizado no KNIME aos 4 conjuntos de dados. Em anexo será colocado uma figura do *workflow* completo construído no KNIME.

5.1 Tratamento do dataset *Traffic Flow*

O tratamento de dados foi iniciado pelo conjunto de dados *Traffic Flow* e o primeiro passo foi realizar o tratamento do atributo *creation_date*. Este atributo era do tipo *string* "yyyy-mm-dd hh:mm:ss:mmm" e o primeiro passo foi alterar este tipo para "yyyy-mm-dd hh:mm:ss" através de um *String Manipulation*. Com esta alteração foi possível transformar esta *string* num *Date&Time* através de um *String to Date&Time Fields*. Agora que temos um *Date&Time* é possível dividir os suas componentes, ou seja, é possível obter novos atributos como a hora, o mês, o dia e o ano. Para tal foi utilizado um *Extract Date&Time Fields* e definiu-se os novos atributos: *Month (number)*, *Day of month*, *Day of week (name)* e *hour*.

Através do novo atributo *Day of week (name)* foi possível criar um novo atributo ao qual chamamos *label_day*, o objetivo deste atributo é saber se o dia em questão é um dia da semana, é sábado ou domingo. Foi necessário recorrer a um *Java Snippet* para que fosse possível fazer a divisão tal como está representado na figura seguinte.

```
1 // system imports
13 // Your custom imports:
14
15 // system variables
25 // Your custom variables:
26 int label_day;
27 // expression start
28 // Enter your code here:
29
30 if (c_Dayofweekname.equals("Segunda-feira") ||
31     c_Dayofweekname.equals("Terça-feira") ||
32     c_Dayofweekname.equals("Quarta-feira") ||
33     c_Dayofweekname.equals("Quinta-feira") ||
34     c_Dayofweekname.equals("Sexta-feira")) {
35     label_day = 0;
36 } else if (c_Dayofweekname.equals("Sábado")) {
37     label_day = 1;
38 } else if (c_Dayofweekname.equals("Domingo")) {
39     label_day = 2;
40 }
41
42
43
44
45 out_label_day = label_day;
46
47
48 // expression end
51
```

Figura 10: *Java Snippet label_day*

Como se pode observar na figura anterior, o algoritmo definido verifica se o dia da semana é "Segunda-feira", "Terça-feira", "Quarta-feira", "Quinta-feira" ou "Sexta-feira" e atribui o valor zero no *label_day*, caso seja "Sábado" atribui o valor um e caso seja "Domingo" atribui o valor dois.

Definidos os novos atributos, decidiu-se remover todos os atributos que achamos não serem necessários. Utilizou-se um *Column Filter* para remover os seguintes atributos: *city_name*, *road_num*, *functional_road_class_desc*, *creation_date* e *Day of week (name)*. A figura seguinte apresenta a exclusão destes dados do conjunto de dados.

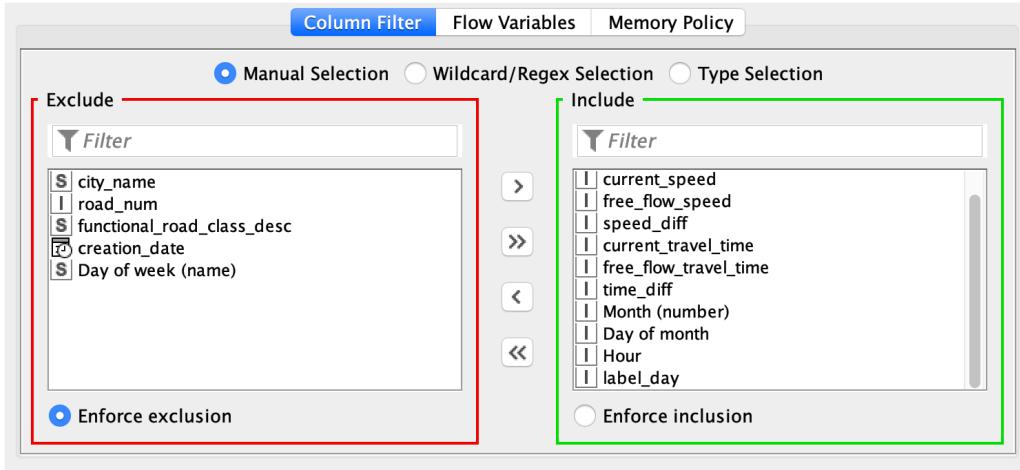


Figura 11: Remoção atributos indesejados

A decisão de remover estes atributos foi tomada com base no que foi analisado anteriormente.

Posto isto, optou-se por organizar os dados através da hora, do dia, do mês e pelo nome da rua, permitindo assim visualizar os dados consoante cada rua diferente. Para isso, colocou-se um *GroupBy* para agrupar os dados por esses parâmetros. Observando os dados disponíveis verificamos que existem mais que um valor de diferentes atributos, tais como o *current_speed*, *free_flow_speed*, entre outros, para a mesma hora. Desta forma utilizou-se o mesmo *GroupBy* para transformar esses vários valores na sua média para que fosse possível manter um registo por hora para cada rua.

De seguida, por intermédio de um *Double to Int*, todos as médias dos atributos calculados foram transformados de *doubles* para inteiros pelo facto de se trabalhar com inteiros posteriormente na rede neuronal que se irá desenvolver. Por último, utilizou-se um *Column Rename* devido ao facto de, com a alteração realizada no *GroupBy* para que fosse possível utilizar as médias dos atributos, as colunas dos atributos que contém valores das médicas ficaram com o seu nome alterado, isto é, foi anexado um "Mean()" ao nome onde o seu nome original foi colocado dentro dos parêntesis, logo foi necessário renomear dessas mesmas colunas.

Todo este processo descrito está dentro de um Metanodo ao qual denominamos *Manipulation Traffic Flow* e é possível visualizá-lo na figura seguinte.

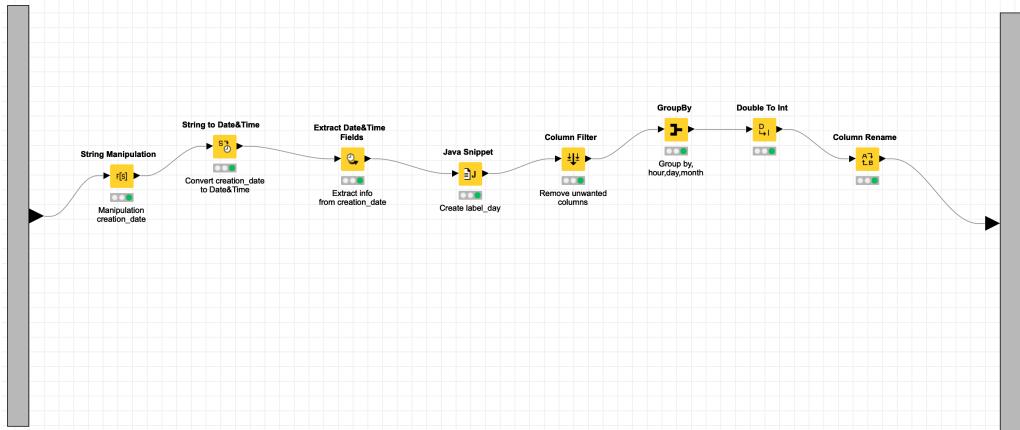


Figura 12: *Manipulation Traffic Flow*

De modo a concluir o tratamento deste *dataset*, foi necessário realizar a divisão dos dados por ruas. Como anteriormente os dados foram organizados por ruas, a divisão destes dados em quatro grupos diferentes será mais fácil e, para tal, utilizaram-se quatro *Row Filters* que agruparam todas as *rows* por cada rua diferente. Isto é, um *Row Filter* recolhe todas as *rows* referentes à rua Avenida dos Aliados, outro todas as *rows* referentes à Rua Conde de Vizela, outro todas as *rows* referentes à rua Avenida Gustavo Eiffel e outro todas as *rows* referentes à rua Nova da Alfândega. Estes filtros são apresentados a seguir.

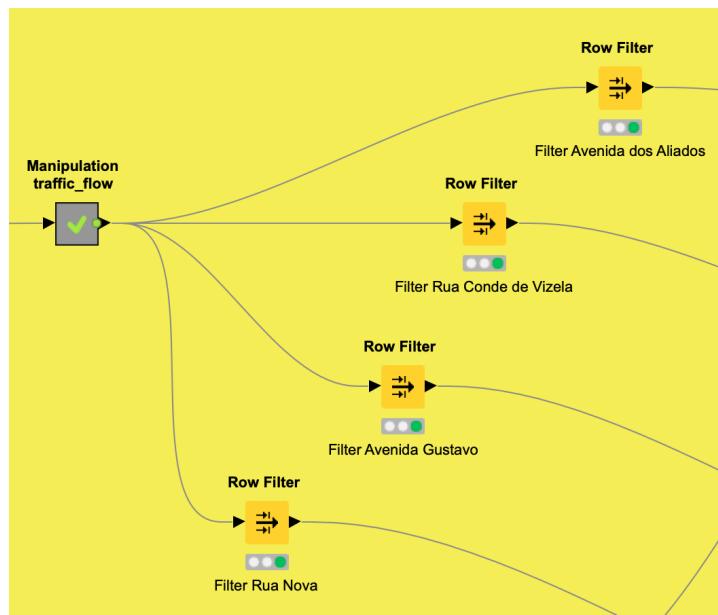


Figura 13: Divisão dos dados por cada rua

5.2 Tratamento do dataset Traffic Incidents

Avançando agora para o tratamento de dados do *dataset Traffic Incidents*, podemos observar na figura seguinte o processo completo da manipulação desse conjunto de dados no KNIME.

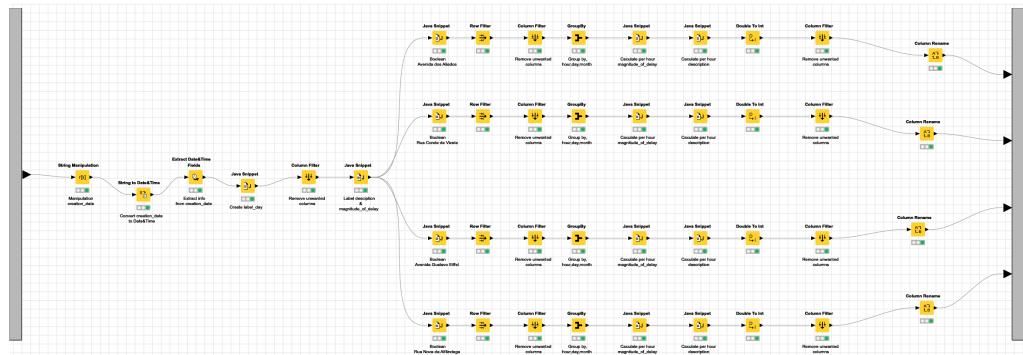


Figura 14: Manipulation Traffic Incidents

Inicialmente optou-se por repetir o mesmo processo aplicado no conjunto de dados anterior, mais concretamente ao atributo *creation_date*, e realizou-se o tratamento do atributo *incident_date* através de um *String Manipulation*, de um *String to Date&Time* e de um *Extract Date&Time Fields*, obtendo-se os novos atributos: *Month (number)*, *Day of month*, *Day of week (name)* e *hour*. Criou-se novamente um novo atributo chamado *label_day* através de um *Java Snippet* tal como no *dataset* anterior e, por último, removeram-se as colunas que se acho que não seriam pertinentes utilizar através de um *Column Filter*, tal como se pode verificar na figura seguinte. Neste caso, os atributos que foram nomeados para serem excluídos são *city_name*, *cause_of_incident*, *affected_roads*, *incident_category_desc*, *incident_date*, *latitude*, *longitude* e o *Day of week (name)* tendo por base a análise realizada no capítulo anterior.

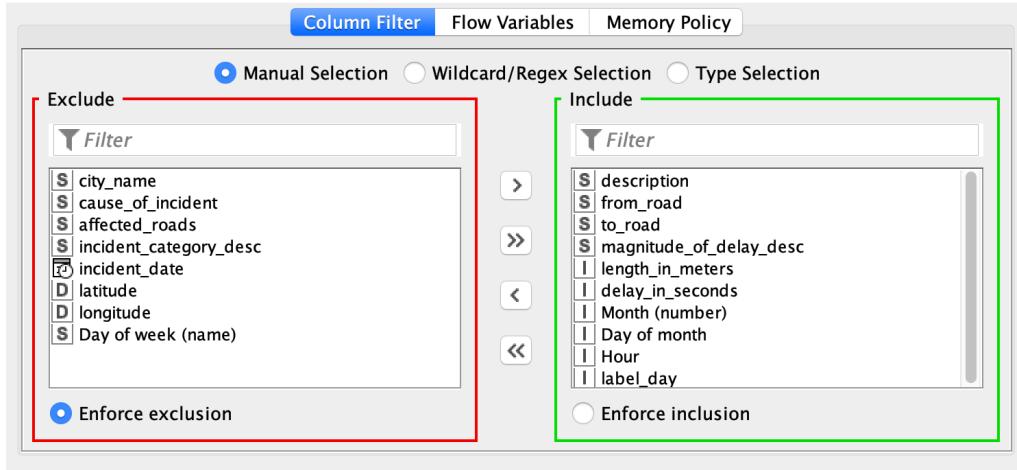


Figura 15: Remoção atributos indesejados

De seguida, optou-se por tratar os dados relativos aos atributos *description* e *magnitude_of_delay_desc*, decidiu-se transformar estes dados que são *strings* em dados numéricos. Para tal utilizou-se um *Java Snippet* onde se introduziu o seguinte algoritmo:

```
// Your custom variables :
int labelMagnitude;
int labelDesc;

// Enter your code here :

labelMagnitude = -1;

if (c_magnitude_of_delay_desc.equals("Unknown_Delay") ||
    c_magnitude_of_delay_desc.equals("Undefined")) {
    labelMagnitude = 0;
} else if (c_magnitude_of_delay_desc.equals("Minor")) {
    labelMagnitude = 1;
} else if (c_magnitude_of_delay_desc.equals("Moderate")) {
    labelMagnitude = 2;
} else if (c_magnitude_of_delay_desc.equals("Major")) {
    labelMagnitude = 3;
}

out_magnitude_of_delay_desc = labelMagnitude;

labelDesc = -1;

if (c_description.equals("tunnel_closed") ||
    c_description.equals("bridge_closed") ||
    c_description.equals("lane_closed")) {
    labelDesc = 0;
} else if (c_description.equals("roadworks") ||
    c_description.equals("narrow_lanes") ||
    c_description.equals("incident")) {
    labelDesc = 1;
} else if (c_description.equals("queuing_traffic") ||
    c_description.equals("slow_traffic")) {
    labelDesc = 2;
} else if (c_description.equals("stationary_traffic")) {
    labelDesc = 3;
}

out_description = labelDesc;
```

Como é possível verificar, no caso da magnitude é transformar o "Undefined" e o "Unknown_Delay" no valor zero, o "Minor" no valor 1, o "Moderate" no valor 2 e o "Major" no valor 3. Já no caso da descrição pertende-se transformar os casos de "tunned_closed", "bridge_closed" e "lane_closed" no valor zero, os casos "roadworks", "narrow_lanes" e "incident" no valor 1, os casos de "queuing_traffic" e "slow_traffic" no valor 2 e, por fim, o caso



de "stationary_traffic" no valor 3.

De seguida, decidiu-se dividir este conjunto de dados em quatro distintos onde cada um deles contém os dados de cada rua. Foi aplicado o mesmo processo para as quatro ruas e consiste em criar uma nova coluna onde em cada um dos quatro novos conjuntos de dados estiver com o valor zero a rua que se pretende para aquele conjunto, com o valor 1 as restantes ruas. Através de um *Java Snippet* criou-se a nova coluna denominada de *street_flag* e foi definido o seguinte algoritmo.

```
int street_flag = 0;

if (c_from_road.contains("Avenida_Gustavo_Eiffel") ||
    c_to_road.contains("Avenida_Gustavo_Eiffel") ||
    c_from_road.contains("Rua_Conde_de_Vizela") ||
    c_to_road.contains("Rua_Conde_de_Vizela") ||
    c_from_road.contains("Rua_Nova_da_Alf_ndega") ||
    c_to_road.contains("Rua_Nova_da_Alf_ndega")) {
    street_flag = 1;
}

out_street_flag = street_flag;
```

O caso do algoritmo a cima diz respeito ao utilizado para a rua Avenida dos Aliados, no entanto cada rua tinha o seu algoritmo específico consoante o que se cria identificar. Como se pode observar, o processo utilizado pelo algoritmo consiste em verificar se na coluna *to_road* ou na coluna *from_road* existem uma das 3 ruas que não pretendemos naquele conjunto de dados. Determinados todos os valores da nova coluna *street_flag*, utilizou-se um *Row Filter* para remover as linhas onde o valor da *street_flag* é igual a 1, mantendo todos os dados da rua a que diz respeito aquele conjunto.

Realizada a divisão do conjunto de dados, removeu-se mais alguns atributos que não se pretendia manter através de um *Column Filter*. As colunas removidas são a *from_road*, *to_road* e a *street_flag* e foram removidas, pois não teriam mais influência devido ao tratamento que se realizou atrás.

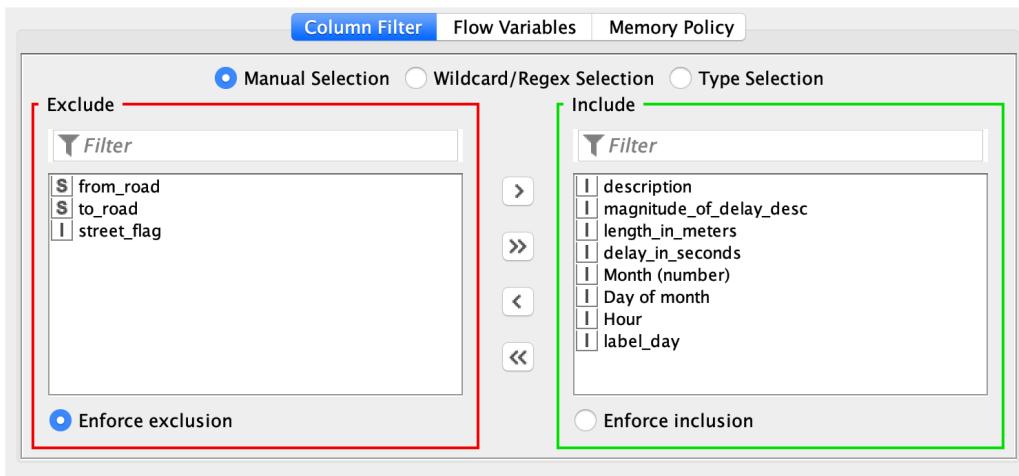


Figura 16: Remoção atributos indesejados

Logo depois foi utilizado um *GroupBy* para agrupar os dados por hora, dia e mês como se fez no *dataset traffic flow*. Depois de se agrupar os dados, os atributos *magnitude_of_delay_desc* e *description* tornaram-se numa lista de valores por hora. Assim sendo, foi necessário tratar destas duas listas de valores e, para isso, utilizou-se um *Java Snippet* para tratar o *magnitude_of_delay_desc*. O algoritmo utilizado neste *Java Snippet* irá verificar quantos elementos de cada tipo existem em cada lista e multiplica cada soma de cada tipo por 10, 25, 30 ou 35, respetivamente. No fim, soma os valores das diferentes multiplicações e verificada entre que valores está e consoante o resultado é definido um valor final para a hora a que dizia respeito a lista inicial, e esse resultado varia entre 0 e 3.

```
// Your custom variables:
int countUndefined;
int countMinor;
int countModerate;
int countMajor;
```



```
int countTotal;
int countTemp;
int magnitude_incident;

// Enter your code here:

countUndefined = 0;
countMinor = 0;
countModerate = 0;
countMajor = 0;
countTotal = 0;
countTemp = 0;
magnitude_incident = -1;

for(int i=0; i < c_Listmagnitude_of_delay_desc.length; i++) {
    if (c_Listmagnitude_of_delay_desc[i] == 0) {
        countUndefined += 1;
    } else if (c_Listmagnitude_of_delay_desc[i] == 1) {
        countMinor += 1;
    } else if (c_Listmagnitude_of_delay_desc[i] == 2) {
        countModerate += 1;
    } else if (c_Listmagnitude_of_delay_desc[i] == 3) {
        countMajor += 1;
    }
    countTotal += 1;
}

countUndefined = countUndefined * 10;
countMinor = countMinor * 25;
countModerate = countModerate * 30;
countMajor = countMajor * 35;

countTemp = countUndefined + countMinor + countModerate + countMajor;

if(((countTemp/countTotal) >= 10) && ((countTemp/countTotal) < 25)) {
    magnitude_incident = 0;
} else if(((countTemp/countTotal) >= 25) && ((countTemp/countTotal) < 30)) {
    magnitude_incident = 1;
} else if(((countTemp/countTotal) >= 30) && ((countTemp/countTotal) < 35)) {
    magnitude_incident = 2;
} else if(((countTemp/countTotal) >= 35)) {
    magnitude_incident = 3;
}

out_magnitude_criteria = magnitude_incident;
```

Já no caso do atributo *description*, o processo foi o mesmo que o anterior, ou seja, também se utilizou um *Java Snippet*. O algoritmo é o mesmo como se pode verificar no código abaixo, este verifica em cada lista o tipo de valor e soma cada ocorrência de cada valor. Estes valores são multiplicados por diferentes valores, mais concretamente 10, 25, 30 ou 35, respetivamente. No fim estas multiplicações são somadas, a soma é verificada para que intervalo de valores corresponde e consoante o resultado é definido um valor final para a hora a que dizia respeito a lista inicial, e esse resultado varia entre 0 e 3.

```
// Your custom variables:
int countClosed;
int countWorks;
int countSlow;
int countTraffic;
int countTotal;
int countTemp;
int description_incident;

// Enter your code here:

countClosed = 0;
```



```
countWorks = 0;
countSlow = 0;
countTraffic = 0;
countTotal = 0;
countTemp = 0;
description_incident = -1;

for(int i=0; i < c_Listdescription.length; i++) {
    if (c_Listdescription[i] == 0) {
        countClosed += 1;
    } else if (c_Listdescription[i] == 1) {
        countWorks += 1;
    } else if (c_Listdescription[i] == 2) {
        countSlow += 1;
    } else if (c_Listdescription[i] == 3) {
        countTraffic += 1;
    }
    countTotal += 1;
}

countClosed = countClosed * 15;
countWorks = countWorks * 20;
countSlow = countSlow * 30;
countTraffic = countTraffic * 35;

countTemp = countClosed + countWorks + countSlow + countTraffic;

if(((countTemp/countTotal) >= 15) && ((countTemp/countTotal) < 20)) {
    description_incident = 0;
} else if(((countTemp/countTotal) >= 20) && ((countTemp/countTotal) < 30)) {
    description_incident = 1;
} else if(((countTemp/countTotal) >= 30) && ((countTemp/countTotal) < 35)) {
    description_incident = 2;
} else if(((countTemp/countTotal) >= 35)) {
    description_incident = 3;
}

out_description_incident = description_incident;
```

O resultado destes dois *Java Snippets* anteriores corresponde a duas novas colunas construídas que denominamos de *magnitude_criteria* e *description_criteria*. A estas duas novas colunas foi aplicado um *Double to Int*, ou seja, transformou estes dois novos atributos em números inteiros. Feito isto, é necessário remover as colunas que estão a mais nestes conjuntos de dados e, como tal, utilizou-se um *Column Filter* e removeu-se as duas listas que foram tratadas anteriormente.

Por último, foi utilizado um *Column Rename* para alterar o nome das colunas que contém informação relativa aos atributos *delay_in_seconds* e *length_in_meters*, pois como também foi utilizado a média desses valores, as respetivas colunas ficaram com o seu nome a "Mean(delay_in_seconds)" e "Mean(length_in_meters)", respetivamente. Com este renome, as colunas ficaram com o nome *delay_in_seconds* e *length_in_meters*, respetivamente.

5.3 Tratamento do dataset *Weather Description*

O tratamento de dados em relação do *dataset* que contém todos parâmetros meteorológicos, começa por implementar uma *String Manipulation* ao atributo *creation_date*, que através de uma Expressão Regular `[0-9]{6}`, os valores são eliminados, p.e, de um atributo **2019-01-15 19:05:00.000000** obtemos **2019-01-15 19:05:00**.

De seguida aplica-se *String to Date&Time* para converter os valores do atributo *creation_date* em formato string para *Date&Time*.

Dentro dos parametros do mesmo tipo aplica-se o nodo *Extract Date&Time Fields* para extrair cada componente de forma isolada e para tal divisão da data é efetuada com foco nas seguintes componentes *número do Mês*, *dia do mês*, *dia da semana por extenso*. para melhor interpretação dos valores de data e para mais tarde serem usados separadamente.



Após todas alterações relacionadas com dados cronológicos, agora vai-se proceder a uma nova adição de componentes através dos nodos *Java Snippet*, será aplicado ao atributo *create_label*, que por sua vez vai ser criado um novo atributo associado com o nome *label_day* que vai indicar se o dia associado corresponde a um dia da semana útil ou fim de semana. O código para este efeito está demonstrado abaixo.

```
// Your custom variables:  
int label_day;  
// expression start  
public void snippet() throws TypeException, ColumnException, Abort {  
// Enter your code here:  
  
if (c_Dayofweekname.equals("Segunda-feira") ||  
    c_Dayofweekname.equals("Terça-feira") ||  
    c_Dayofweekname.equals("Quarta-feira") ||  
    c_Dayofweekname.equals("Quinta-feira") ||  
    c_Dayofweekname.equals("Sexta-feira")) {  
    label_day = 0;  
} else if (c_Dayofweekname.equals("Sábado")) {  
    label_day = 1;  
} else if (c_Dayofweekname.equals("Domingo")) {  
    label_day = 2;  
}  
  
out_label_day = label_day;
```

O seguinte passo será apagar colunas desnecessárias, e para tal é usado o nodo *Column Filter*, filtrando apenas as que interessam.

Na coluna *rain* ocorrem missing values, e o tratamento a fazer será fazer a substituição desses valores.

Após estes nodos todos, vamos aplicar mais *Java Snippet* a várias colunas para poder extraír mais informação para melhor tratamento e individualização de dados. A primeira coluna será *Atmosphere* que vai ver os dados exclusivos e vai associar um número a cada valor e armazenar esses valores na própria coluna.

```
// system variables  
public class JSnippet extends AbstractJSnippet {  
    // Fields for input columns  
    /** Input column: "atmosphere" */  
    public String c_atmosphere;  
  
    // Fields for output columns  
    /** Output column: "atmosphere" */  
    public Integer out_atmosphere;  
  
    // Your custom variables:  
    int label;  
    // expression start  
    public void snippet() throws TypeException, ColumnException, Abort {  
        // Enter your code here:  
  
        label = -1;  
  
        if (c_atmosphere.equals("N/A")){  
            label = 0;  
        } else if (c_atmosphere.equals("céu limpo")){  
            label = 1;  
        } else if (c_atmosphere.equals("nublado") || c_atmosphere.equals("garoa fraca")){  
            label = 2;  
        } else if (c_atmosphere.equals("névoa") || c_atmosphere.equals("neblina") ||  
                   c_atmosphere.equals("nevoeiro")){  
            label = 3;  
        }  
  
        out_atmosphere = label;
```

Para missing values o valor será de 0, "céu limpo" será valor de 1, "nublado" e "fraca" o valor será de 2 e por fim "nevoeiro", "névoa", "neblina" assumem o valor 3.



Para a coluna *rain* vamos repetir o processo, para tal segue a seguinte implementação

```
// Your custom variables:  
int label;  
// expression start  
public void snippet() throws TypeException, ColumnException, Abort {  
// Enter your code here:  
  
label = -1;  
  
if (c_rain.equals("N/A")) {  
    label = 0;  
} else if (c_rain.equals("chuva_fraca") || c_rain.equals("chuva_leve") ||  
    c_rain.equals("aguaceiros_fracos") || c_rain.equals("chuvisco_fraco") ||  
    c_rain.equals("chuvisco_e_chuva_fraca")) {  
    label = 1;  
} else if (c_rain.equals("chuva") || c_rain.equals("aguaceiros")) {  
    label = 2;  
} else if (c_rain.equals("chuva_moderada")) {  
    label = 3;  
} else if (c_rain.equals("chuva_forte")) {  
    label = 4;  
}  
  
out_rain = label;
```

Atributo "N/A" assume o valor 0. Atributos como "chuva fraca", "chuva leve", "aguaceiros fracos", "chuvisco fraco" e "chuvisco e chuva fraca" ficam com o valor 1. Atributos "chuva", "aguaceiros" assumem o valor 2. Atributo "chuva moderada" e "chuva forte" assumem 3 e 4 respectivamente.

A próxima coluna a ser afetada com *Java Snippet* é a coluna *cloudiness* e o seguinte algoritmo descreve essa transformação.

```
// Your custom variables:  
int label;  
// expression start  
public void snippet() throws TypeException, ColumnException, Abort {  
// Enter your code here:  
  
label = -1;  
  
if (c_cloudiness.equals("N/A")) {  
    label = 0;  
} else if (c_cloudiness.equals("ceu_claro") || c_cloudiness.equals("ceu_limpo")) {  
    label = 1;  
} else if (c_cloudiness.equals("algumas_nuvens") ||  
    c_cloudiness.equals("ceu_pouco_nublado") ||  
    c_cloudiness.equals("nuvens_quebradas") ||  
    c_cloudiness.equals("nuvens_quebrados") ||  
    c_cloudiness.equals("nuvens_dispersas")) {  
    label = 2;  
} else if (c_cloudiness.equals("tempo_nublado") || c_cloudiness.equals("nublado")) {  
    label = 3;  
}  
  
out_cloudiness = label;
```

Atributo "N/A" assume o valor 0. Atributos como "céu claro", "céu limpo", "algumas nuvens", "céu pouco nublado" ficam com o valor 1. Atributos "algumas nuvens", "céu pouco nublado", "nuvens quebradas", "nuvens quebrados" e "nuvens dispersas" assumem o valor 2. Atributo "tempo nublado" e "nublado" assumem 3.

Por fim o último nodo a ser implementado será *Duplicate Row Filter* que irá remover dados em duplicado das seguintes colunas *Número do mês, dia do mês e Hora*.

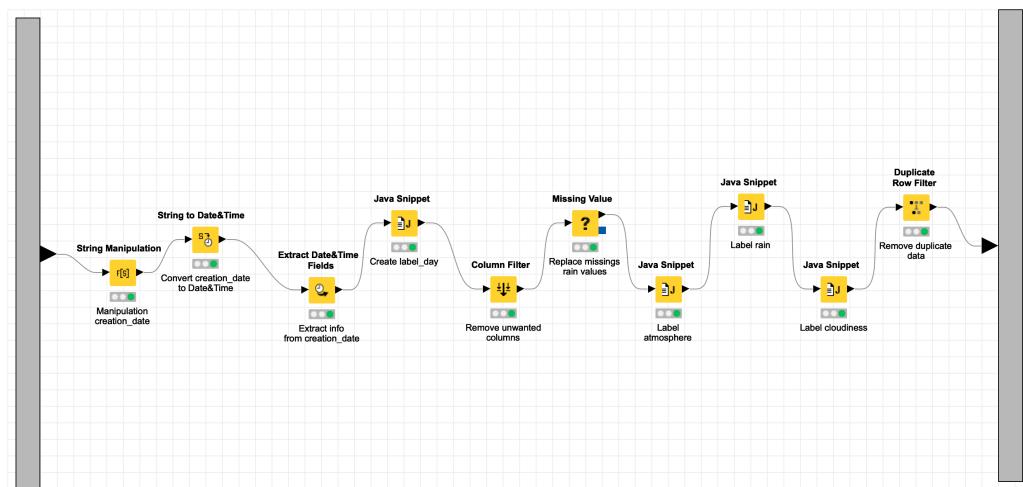


Figura 17: *Manipulation Weather Description*

5.4 Tratamento do dataset *Weather Until*

Face a este conjunto de dados, o procedimento inicial é igual ao que foi realizado nos *datasets* anteriores em que se realizou o tratamento do atributo *creation_date* através de um *String Manipulation*, um *String to Date&Time* e um *Extract Date&Time Fields*. Mais uma vez foi criada uma nova coluna chamada *label_day* através de um *Java Snippet* tal como nos *datasets* anteriores.

Posto isto, foi altura de remover algumas colunas que não se pretendia utilizar. Utilizou-se um *Column Filter* e removeu-se o *city_name*, *atmospheric_pressure*, *wind_speed*, *precipitation*, *sunrise*, *sunset*, *creation_date* e *Day of the week (name)*.

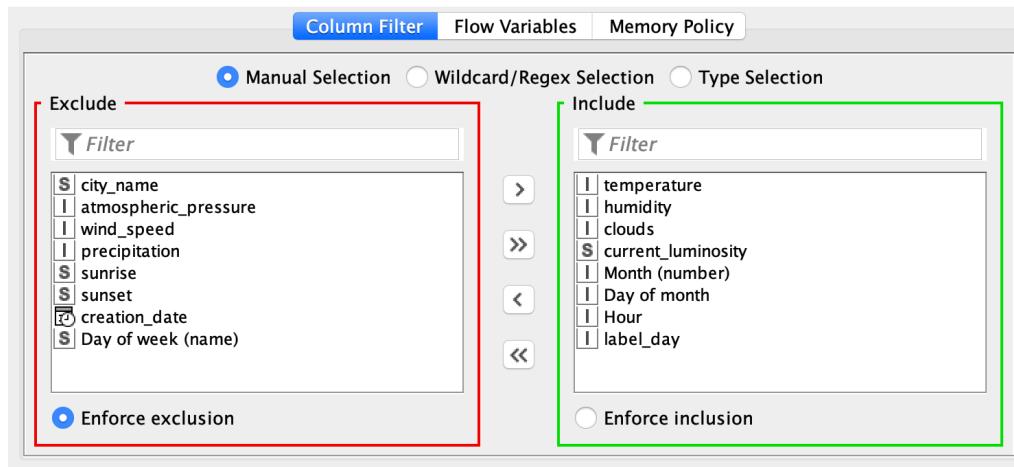


Figura 18: Remoção atributos indesejados

De seguida, optou-se por fazer um tratamento ao atributo *current_luminosity*, uma vez que apenas tinha três tipos de dados no formato *string*, mais concretamente "LIGHT", "LOW_LIGHT" e "DARK", tal como foi explicado no capítulo anterior, e pretende-se passar esses dados para valores numéricos. Para tal, foi utilizado um *Java Snippet* onde foi colocado o seguinte algoritmo:

```
// Your custom variables:  
int label;  
  
// Enter your code here:  
label = -1;  
  
if (c_current_luminosity.equals("LIGHT")) {  
    label = 0;  
} else if (c_current_luminosity.equals("LOW_LIGHT")) {  
    label = 1;  
} else if (c_current_luminosity.equals("DARK")) {  
    label = 2;  
}
```



```
out_current_luminosity = label;
```

Como se pode observar pelo algoritmo, este verifica qual é o tipo de dado de cada *row* e caso seja "LIGHT" o seu valor será zero, caso seja "LOW_LIGHT" o seu valor passa a ser 1 e caso seja "DARK" o seu valor passa a ser 2.

Por último, é aplicado um *Duplicate Row Filter* com o objetivo de remover toda a informação que esteja duplicada neste conjunto de dados.

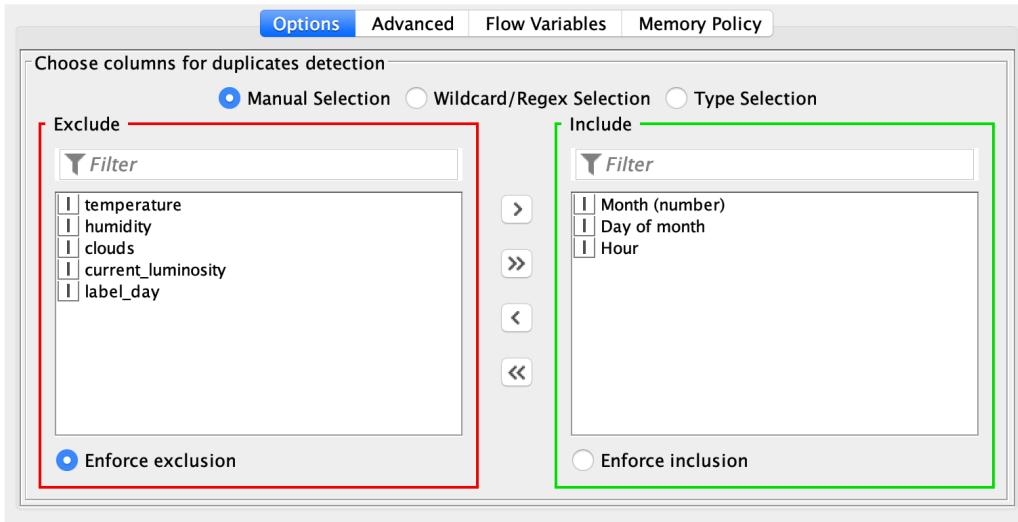


Figura 19: Remoção de informação duplicada

Todo o processo descrito atrás está representado na figura seguinte com os diferentes nodos utilizados no KNIME.

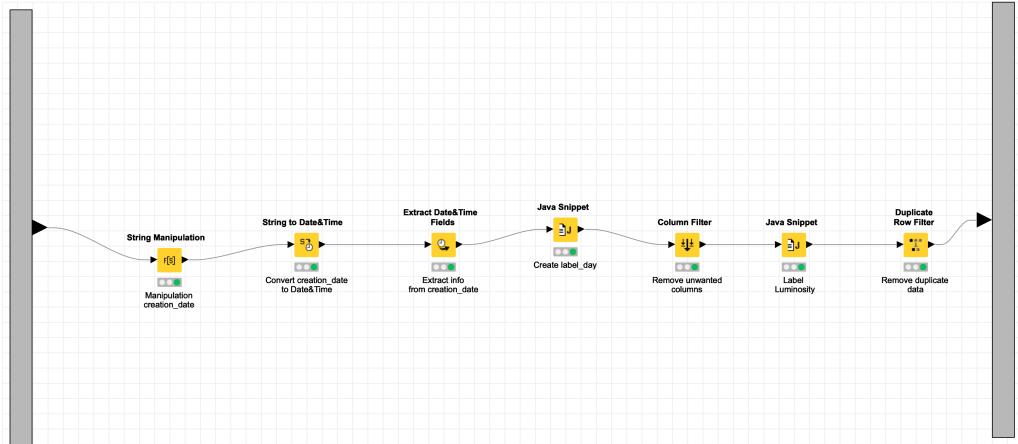
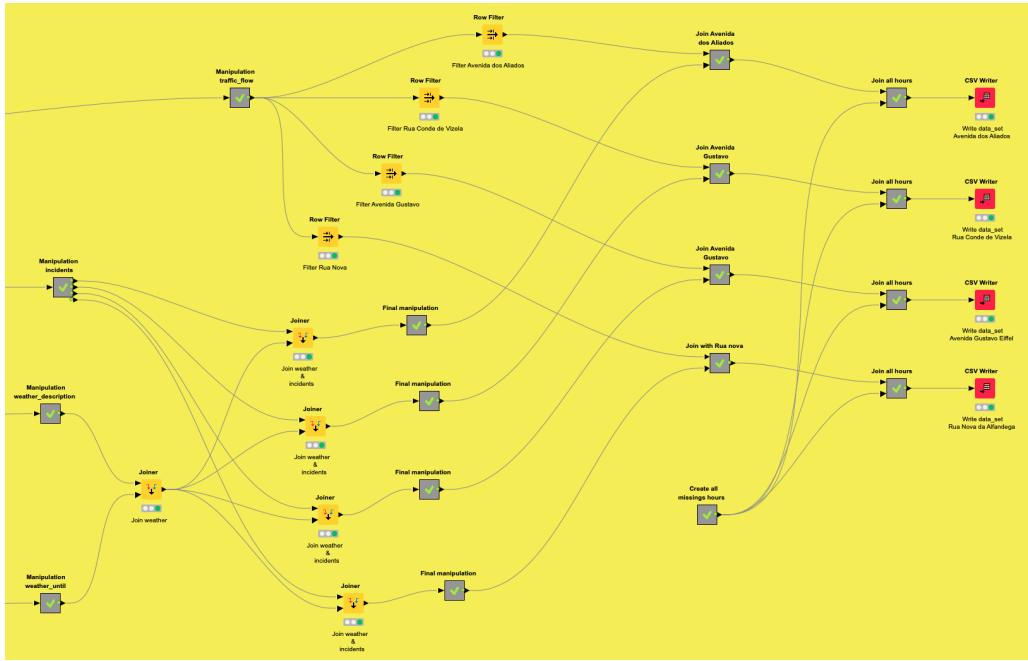


Figura 20: *Manipulation Weather Until*

5.5 Criação dos datasets finais

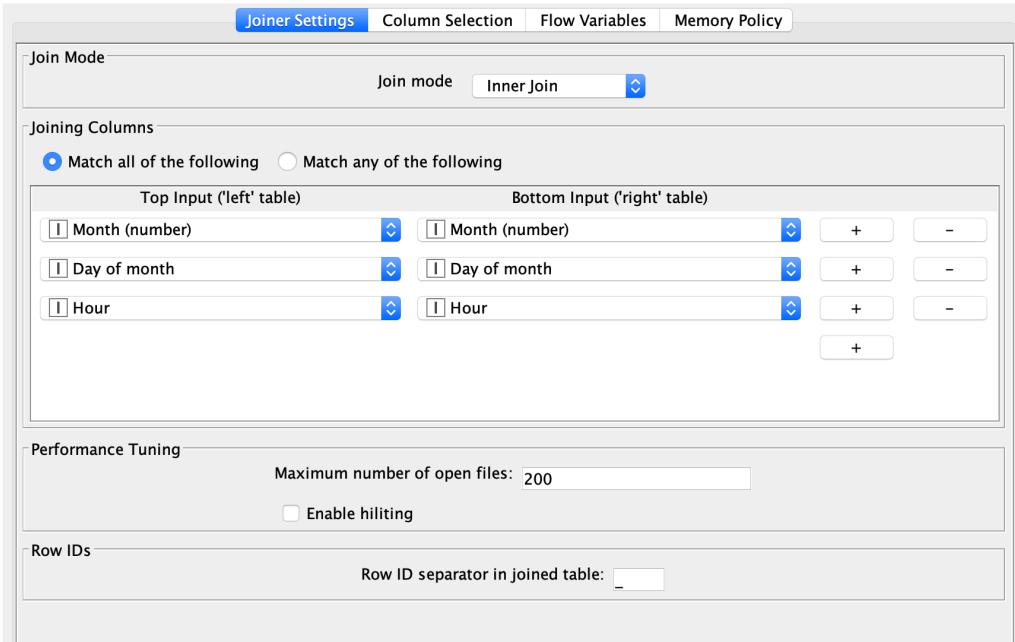
Realizado o tratamento de dados de cada conjunto de dados inicial, é necessário agrupá-los consoante cada rua, ou seja, será realizado um agrupamento de todos os *datasets* e posteriormente uma divisão em novos quatro *datasets* com os mesmos atributos, mas com dados relativos a cada rua diferente.

A figura seguinte apresenta todo o processo que nos levou a construção destes quatro conjuntos de dados finais incluindo as manipulações já realizadas atrás.


 Figura 21: Criação dos *datasets* finais

Como se pode ver na figura, parte do tratamento dos quatro novos *datasets* já foi realizado antes, para além das manipulações realizadas anteriormente, quando se apresentou o tratamento do *dataset Traffic Flow*. Neste caso, associar os dados provenientes do *Traffic Flow* por cada rua será mais fácil, pois os dados já estão separados.

Começando agora por agregar os dados do *dataset Weather Description* com os dados do *dataset Weather Until*, foi necessário utilizar um *Joiner* onde se aplicou um *Inner Join* segundo os atributos "Month (number)", "Day of month" e "Hour" dos dois conjuntos de dados.


 Figura 22: *Inner Join*

De seguida, agregou-se o resultado deste *Joiner* com os diferentes conjuntos de dados obtidos no *Manipulation incidents*. Utilizou-se quatro *Joiners*, uma vez que os dados provenientes do *Manipulation incidents* estão separados pelas diferentes ruas. Neste caso aplicou-se um *Right Outer Join* em cada *Joiner* segundo os atributos "Month (number)", "Day of month" e "Hour" dos dois conjuntos de dados.

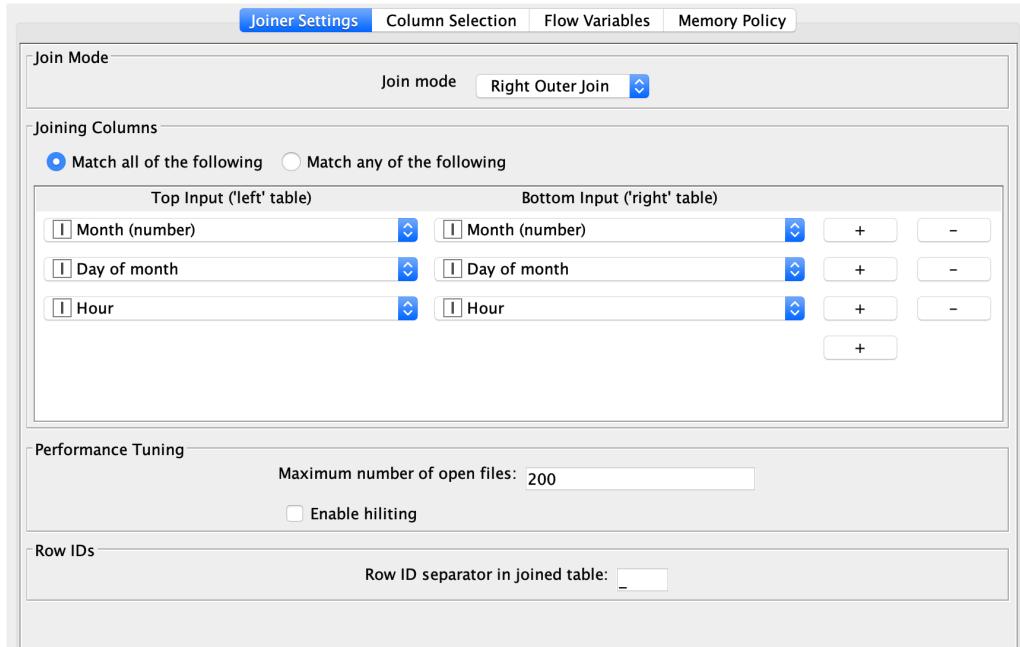


Figura 23: Right Outer Join

A cada um destes quatro resultados diferentes realizou-se uma última manipulação, onde o primeiro passo foi remover colunas que não eram precisas após a agregação. Por fim, aplicou-se um *Duplicate Row Filter* para remover os dados duplicados consoante a hora, o dia e o mês.

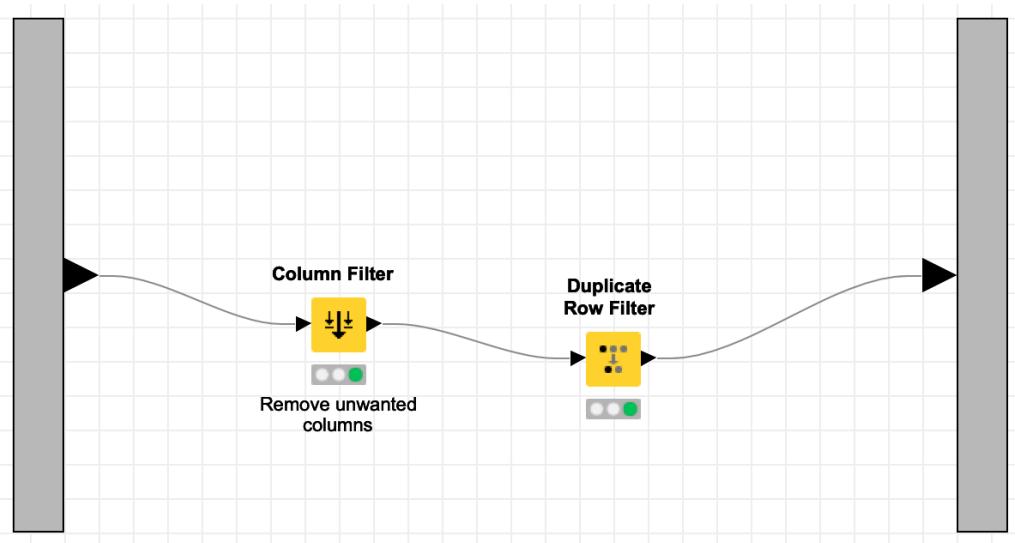


Figura 24: Manipulação final

Após isto, ficaram disponíveis 4 conjuntos de dados com as agregações dos *datasets* *Traffic Incidents*, *Weather Description* e *Weather Until* e é necessário agregá-los aos quatro conjuntos de dados retirados do *dataset* *Traffic Flow*. O primeiro passo foi aplicar um *Joiner* onde se aplicou um *Right Outer Join* consoante a hora, o dia e o mês. De seguida, utilizou-se um *Column filter* com o intuito de remover colunas de atributos que já se havia removido numa das partes da agregação e na outra não. Assim sendo, as colunas removidas foram a *road_name* e a *Frist*(label_day)*, tal como se comprova na figura seguinte.

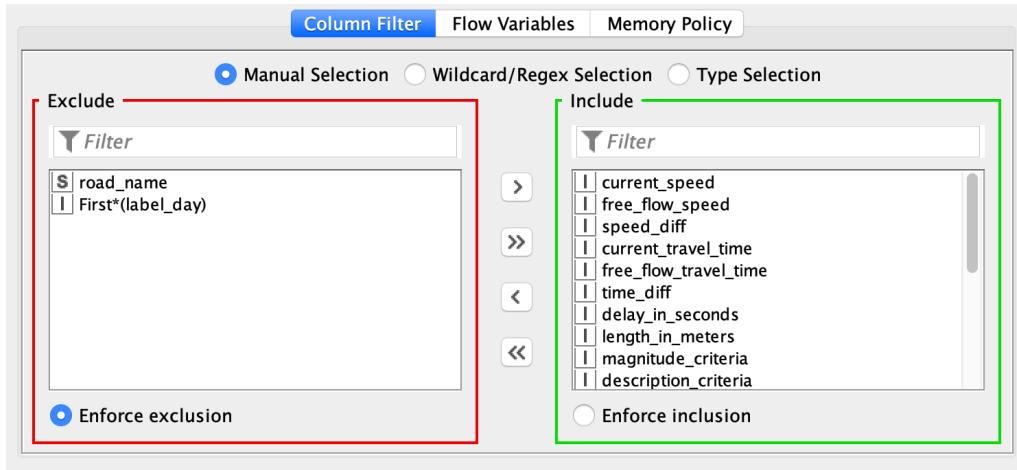


Figura 25: Remoção atributos indesejados

Nesta fase, o grupo optou por aplicar um nodo *Sorter* com o intuito de ordenar os dados consoante o mês, o dia e a hora de forma ascendente. Feito isto, recorreu-se a *Java Snippet* para alterar os dados da coluna *RowID*, ou seja, alterar o identificador de cada linha. O objetivo passa por colocar como identificador de cada linha a hora, o dia e o mês da forma "hora/dia/mês", para esse fim definiu-se o seguinte algoritmo.

```
int row = 0;

String row_out = ""+ c_Hour + "/" + c_Dayofmonth + "/" + c_Monthnumber + "";
row += 1;

out_RowId = row_out;
```

Este algoritmo cria uma nova coluna com todos os novos identificadores e através de um nodo *RowID* definiu-se que esta nova coluna seria o *RowID*. Todo o processo descrito atrás é visível na figura seguinte.

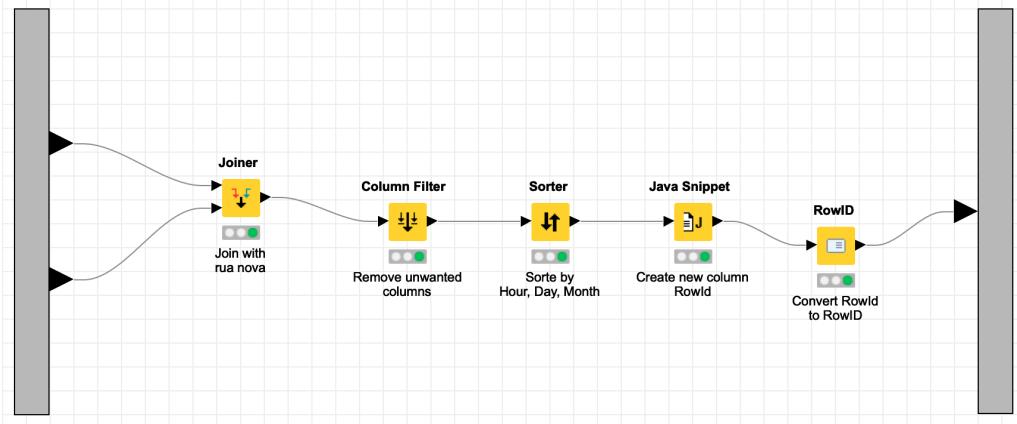


Figura 26: Agregação por rua

Neste momento, estão definidos os quatro conjuntos de dados com os dados de cada rua respetivamente. Após uma breve análise por cada um dos conjuntos, verificou-se a inexistência de horas que completassem um ano completo, uma vez que um 365 dias dá um total de 8760 horas, no entanto cada *dataset* continha apenas 6821 horas. Para resolver esta questão da falta de horas para perfazer as 8760 horas do ano completo, criou-se um metanodo onde se introduziu vários nodos com o intuito de gerar as horas que faltam para posteriormente juntar a cada um dos *datasets*, tal como se pode ver na figura seguinte.

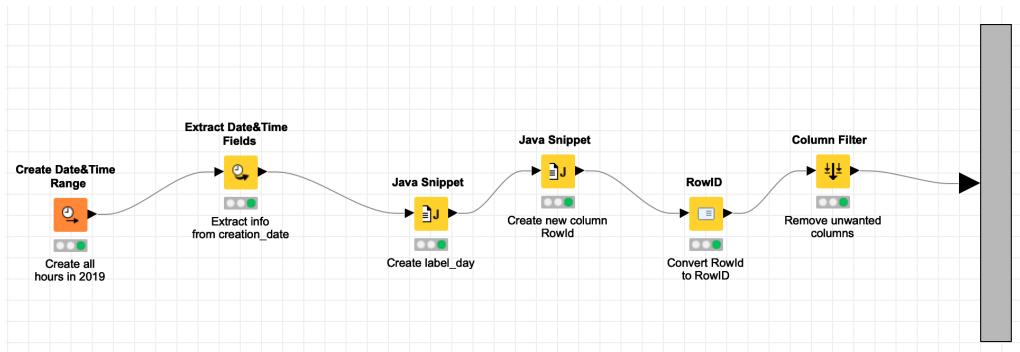


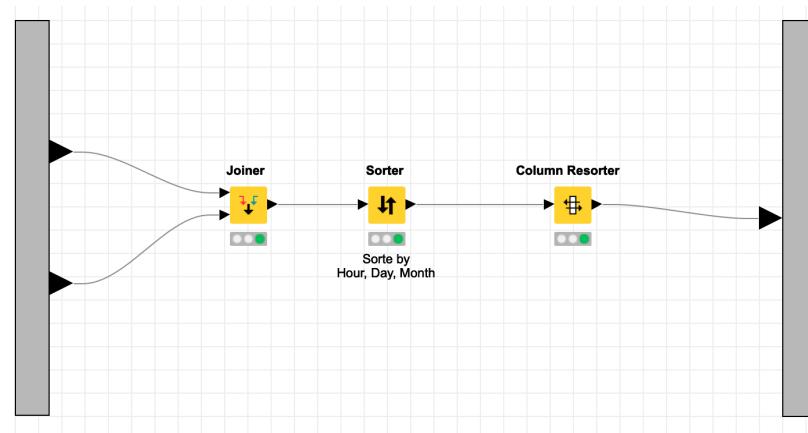
Figura 27: Criação de horas em falta

A estratégia foi utilizar um *Create Date&Time Range* para criar um *Date&Time* onde tivesse todas as horas desde o dia 1 de Janeiro de 2019 às zero horas zero minutos e zero segundos até ao dia 31 de Dezembro de 2019 às 23 horas, 59 minutos e 59 segundos, tal como se pode verificar na figura seguinte.

Figura 28: Criação Date&Time

Feito isto, aplicou-se um processo semelhante ao que já se tinha realizado com outros atributos como o *creation_date*, aplicou-se um *Extract Date&Time Fields* à nova coluna *Date&Time* para extraír o mês, o dia e a hora. De seguida, foi necessário utilizar um *Java Snippet* para criar o atributo *label_day* tal como já se tinha feito anteriormente. Com estes novos dados, foi possível definir os identificadores de cada coluna seguindo o mesmo processo que foi aplicado aos quatro conjuntos de dados, tal como foi explicado anteriormente, através de um *Java Snippet* e um *RowID*, fazendo com que o identificador de cada linha seja composto por um valor do tipo "hora/dia/mês". Por último, foi necessário remover as colunas que eram desnecessárias com o auxílio de um *Column Filter*, mais concretamente foram removidas as colunas *Date&Time* e *Day of week (name)*.

Criadas as horas em falta, é altura de adicioná-las aos quatro conjuntos de dados construídos anteriormente. Visto que tanto o conjunto com as novas horas como cada conjunto com os dados rodoviários tem o mesmo tipo de *RowID*, foi aplicado um *Joiner* que recebe o *dataset* com os dados de todas as horas e um dos *datasets* das quatro ruas agrupando-os segundo o *RowID*. Logo após aplica-se um *Sorter* com o intuito de ordenar os dados consoante o mês, o dia e a hora de forma ascendente. Agora que os dados estão todos agregados, o último passo deste agrupamento foi utilizar um *Column Resorter* para dispor as colunas de cada um dos quatro conjuntos de dados na ordem que pretendíamos. Este tratamento está representado na figura seguinte.

Figura 29: *Datasets* com novas horas

Com todos os dados tratados bem como os quatro novos *datasets* prontos é altura de exportá-los em ficheiros CSV através de um *CSV Writer*. Nesta fase, podemos prosseguir para a criação de um modelo para efetuar as previsões que pretendemos. No entanto, é importante referir que ainda encontramos alguns vários valores em falta nestes quatro conjuntos de dados, uma vez que não tinham sido recolhidos dados para diversas horas. O grupo decidiu realizar o tratamento desses *missing values* através de um algoritmo em *Python* que iremos incluir no *script* da criação do modelo e de *forecasting*.



6 Desenvolvimento de um modelo

Neste capítulo é realizada a explicação do processo detalhado de desenvolvimento de um modelo. Para isso, o primeiro passo foi identificar as as *features* mais importantes.

6.1 Escolha das *features*

Para a escolha das melhores *features* para cada rua analisada, utilizamos dois métodos: *Random Search* e *XGBoost*.

O método *Random Search* identifica as N melhores *features* sem qualquer grau de importância, sendo N um parâmetro a ser escolhido por nós. Uma vez que este método apenas identifica as *features*, mas não lhes atribui qualquer tipo de classificação não conseguimos apenas com isto comparar as *features*, logo decidimos complementar esta escolha com um segundo método.

O método de *XGBoost* identifica não só as melhores *features*, como também as classifica dando-nos uma classificação baseada em importância para cada uma. Desta forma poderemos identificar as *features* mais importantes mas também saber que grau de influência terão nas decisões finais.

Explicados os dois métodos, criou-se um *script* para aplicar os dois conceitos. Primeiramente criou-se uma função para fazer *load* do *dataset* que quisermos utilizar e que denominamos de *load_data*. Esta recebe um ficheiro, lê o ficheiro CSV (*read_csv*) e transforma-o num *dataframe*. De seguida, definiu-se uma outra função para realizar a preparação dos dados à qual chamamos de *prepare_data*. Esta função normaliza os dados do *dataframe* com um *Scaler* criado através da função *MinMaxScaler* e é definido as *features* e o *target*.

Avançando agora para a *main*, importa-se o ficheiro que se pretende utilizar e aplica-se a função *load_data* a esse ficheiro. A seguir, aplica-se a função *prepare_data* e obtém-se o X (*features*), o Y (*target*) e o *Scaler*. O primeiro método que se vai recorrer ao *Random Search*, através do algoritmo que construímos. O algoritmo construído numa primeira fase utiliza uma *Decision Tree Regressor* com um *RandomizedSearchCV* de maneira a obter o número máximo de *features* que se deve utilizar. Na segunda parte deste método, já com o número máximo de *features* a utilizar, defini-se uma nova *Decision Tree Regressor* para utilizar com uma função *RFE*, isto é, para que seja possível aplicar uma eliminação recursiva de recursos (RFE), que tem como objetivo selecionar *features* considerando recursivamente conjuntos de *features* cada vez menores. Para a função *RFE* é fornecida a *Decision Tree Regressor* e o número máximo de *features* a utilizar, que neste caso são 13. O resultado é apresentado na forma de um gráfico com todas as *features* disponíveis e com as 13 *features* que se devem utilizar devidamente identificadas.

Por último, aplicou-se o segundo método, o *XGBoost*. Neste caso, começou-se por definir os parâmetros a utilizar por este método.

```
xgb_params = {  
    'eta': 0.05,  
    'max_depth': 10,  
    'subsample': 1.0,  
    'colsample_bytree': 0.7,  
    'objective': 'reg:squarederror',  
    'eval_metric': 'rmse',  
    'silent': 1  
}
```

Definidos os parâmetros, utilizou-se a função *DMatrix*, que é uma estrutura de dados interna usada pelo *XGBoost*, otimizada para eficiência de memória e velocidade de treino. Para a *DMatrix* é-lhe fornecido as *features* e o *target*, o resultado é fornecido à função *train*. O resultado final é apresentado na forma de gráfico tal como no método anterior, com todas as *features* e o *score* de cada uma delas por ordem.

De seguida apresentaremos os gráficos obtidos da aplicação dos dois métodos previamente enunciados, apresentamos os obtidos para a rua Avenida dos Aliados e os restantes serão colocados em anexo. Poderemos assim verificar quais as *features* mais importantes e quais podemos descartar.



6.1.1 Avenida dos Aliados

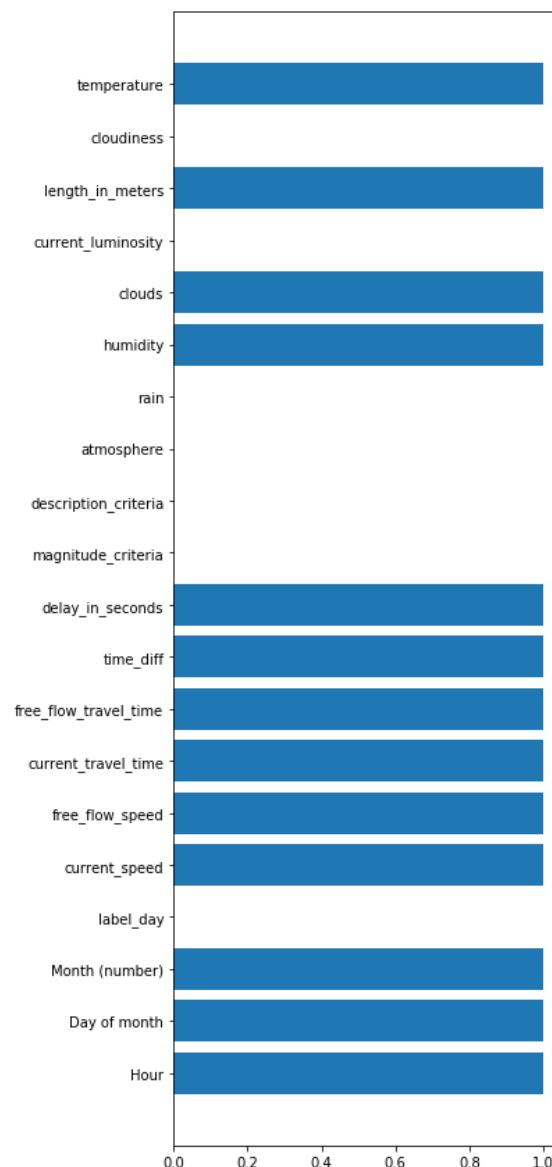
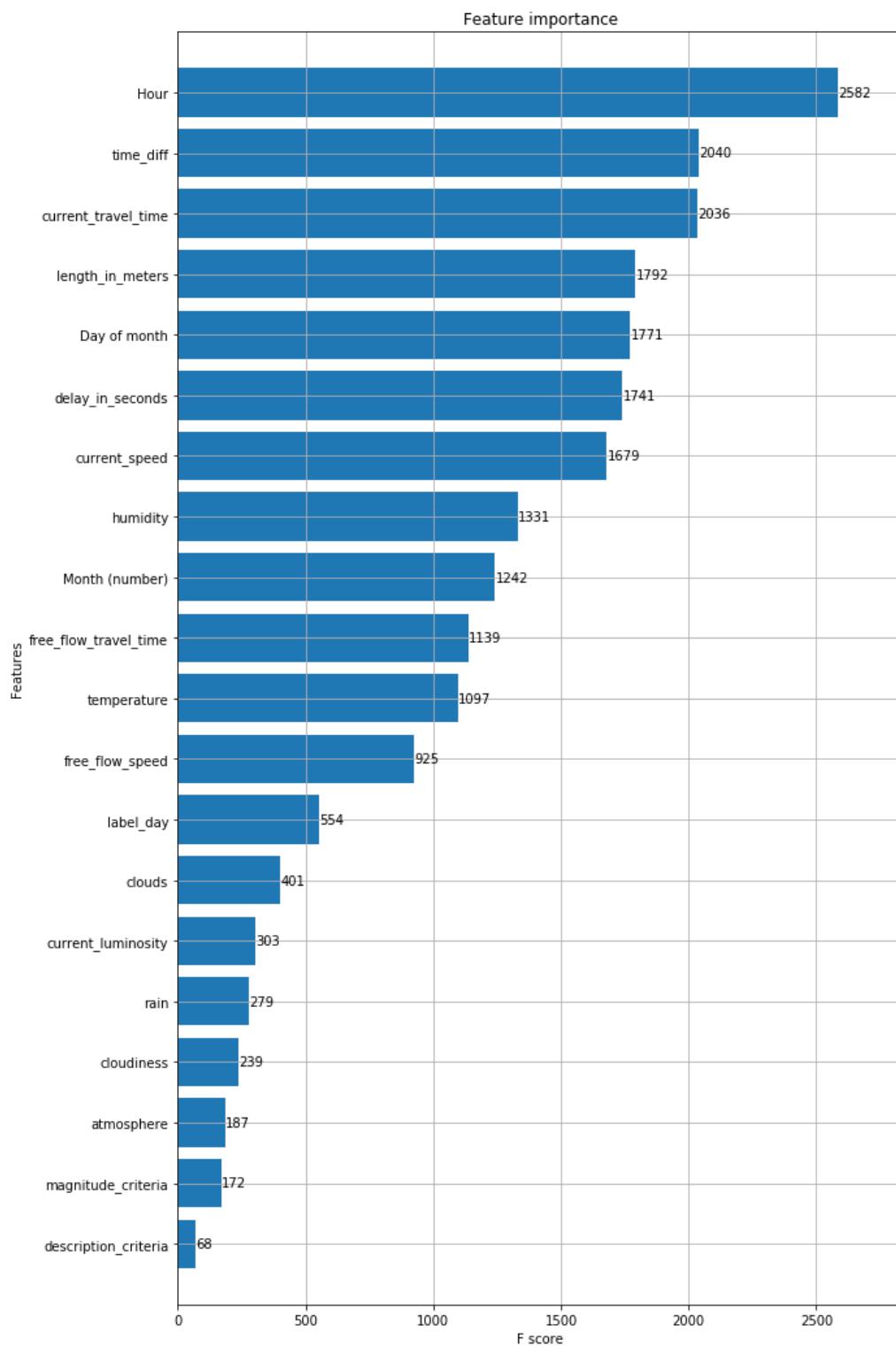


Figura 30: Avenida dos Aliados - *Random Search*

Figura 31: Avenida dos Aliados - *XGBoost*

6.1.2 Análise dos resultados

De forma a visualizar os resultados obtidos nos gráficos, optou-se por elaborar duas tabelas: resultados do *Random Search* e resultados do *XGBoost*. Como se pode verificar, os resultados nas duas pesquisas indicam que as *features* a utilizar nas diferentes ruas são praticamente os mesmos, o que nos leva a concluir que o modelo que se irá definir, poderá ser utilizado nas diferentes ruas.



Rua\Feature	Avenida dos Aliados	Rua Conde de Vizela	Avenida Gustavo Eiffel	Rua Nova da Alfândega
<i>temperature</i>	S	S	S	S
<i>cloudiness</i>	N	N	N	N
<i>length_in_meters</i>	S	S	S	S
<i>current_luminosity</i>	N	N	N	N
<i>clouds</i>	S	N	S	S
<i>humidity</i>	S	S	S	S
<i>rain</i>	N	N	N	N
<i>atmosphere</i>	N	N	N	N
<i>description_criteria</i>	N	N	N	N
<i>magnitude_criteria</i>	N	N	N	N
<i>delay_in_seconds</i>	S	S	S	S
<i>time_diff</i>	S	S	S	S
<i>free_flow_travel_time</i>	S	S	S	S
<i>current_travel_time</i>	S	S	S	S
<i>free_flow_speed</i>	S	S	S	S
<i>current_speed</i>	S	S	S	S
<i>label_day</i>	N	S	N	N
<i>Month (number)</i>	S	S	S	S
<i>Day of Month</i>	S	S	S	S
<i>Hour</i>	S	S	S	S

Tabela 1: Resultados - *Random Search*

Rua\Feature	Avenida dos Aliados	Rua Conde de Vizela	Avenida Gustavo Eiffel	Rua Nova da Alfândega
<i>temperature</i>	1097	860	1259	733
<i>cloudiness</i>	239	133	223	98
<i>length_in_meters</i>	1792	1181	1897	1022
<i>current_luminosity</i>	303	117	208	142
<i>clouds</i>	401	341	341	227
<i>humidity</i>	1331	1000	1341	776
<i>rain</i>	279	153	235	123
<i>atmosphere</i>	187	176	259	114
<i>description_criteria</i>	68	72	84	29
<i>magnitude_criteria</i>	172	163	233	120
<i>delay_in_seconds</i>	1741	1214	1911	1117
<i>time_diff</i>	2040	2154	1734	1645
<i>free_flow_travel_time</i>	1139	1752	1237	959
<i>current_travel_time</i>	2036	2085	1905	1262
<i>free_flow_speed</i>	925	850	1302	969
<i>current_speed</i>	1679	1714	2158	1420
<i>label_day</i>	554	301	435	309
<i>Month (number)</i>	1242	880	1319	783
<i>Day of Month</i>	1771	1695	1962	1002
<i>Hour</i>	2582	1859	2292	1592

Tabela 2: Resultados - *XGBoost*

Analisando as duas tabelas, verificamos que as colunas que não se irão utilizar são as seguintes: *label_day*, *magnitude_criteria*, *atmosphere*, *description_criteria*, *rain*, *clouds*, *current_luminosity* e *cloudiness*.



6.2 Criação do modelo

Sabendo agora que *features* utilizar, é altura de criar o modelo para efetuar a previsão de trânsito, mais concretamente a previsão do *speed diff*. O objetivo passará por construir um modelo baseado numa rede LSTM para realizar a previsão que pretendemos. Para isso é necessário definir os parâmetros a utilizar para a criação da rede, tais como o número de *features* a utilizar, o número de *timesteps*, o número de *multisteps*, o número de épocas, entre outros.

```
nr_features = 13
timesteps = 24
multisteps = 24
epochs = 100
batch_size = 32
verbose = 2
patience = 10
n_splits = 5
now = datetime.datetime.now()
```

Começamos pela recolha dos dados, ou seja, importou-se um dos quatro ficheiros CSV com os dados de cada rua, respetivamente. Como os dados de cada ficheiro CSV contém dados semelhantes, isto é, com dados das mesmas *features*, o modelo que se irá construir poderá ser utilizado com os quatro ficheiros diferentes. Utilizou-se a mesma função utilizada no *script* anterior para ler os ficheiros CSV, a *load_data*. Feito a leitura do ficheiro, avançamos para a preparação dos dados tal como se fez no *script* anterior e construímos a função *prepare_data*. O primeiro passo da preparação dos dados é a remoção das *features* que não irão ser utilizadas. Os dados do *dataframe* são convertidos em *float*. Posto isto, criou-se um novo *dataframe* onde se aplicou a função desenvolvida à qual denominamos de *fill_missing*, que recebe o *dataframe* anterior e preenche os valores em falta com os valores do mesmo dia da semana à mesma hora na semana anterior, com exceção das colunas "delay_in_seconds" e "length_in_meters" em que achamos que faria mais sentido preencher com o valor fixo de 0 e as colunas "temperature" e "humidity" pelos valores correspondentes de há uma hora atrás. Com isto, é construído este novo *dataframe* sem *missing values*, os dados são normalizados e obtém-se um *scaler*.

De seguida, definiu-se uma variável com o valor atual da hora, dia e mês que coincidisse com o *index* dos dados, ou seja, do tipo "hora/dia/mês". Com esta variável, foi possível criar um *dataframe* com os dados até à hora atual e serão estes os dados a utilizar para realizar a previsão e treino da rede. Depois se realizar a preparação dos dados, é altura de aplicar uma função de *supervised learning* à qual chamamos de *to_supervised*. Esta função tem o objetivo de mapear uma entrada para uma saída com base em exemplos de pares entrada-saída. Esta função recebe o *dataframe* que se está a utilizar, os *multisteps*, os *timesteps* e o número de *features* a utilizar. O resultado desta função será utilizado para realizar uma *Time Series Split*.

O *Time Series Split* consiste em fornecer dados de treino e de teste para dividir amostras de dados em várias séries temporais, que são observadas em intervalos de tempo fixos em conjuntos de treino/teste. Em cada divisão, os índices de teste devem ser mais altos que os anteriores. No nosso caso, começou-se por recolher os valores para o *Time Series Split* a cada iteração e por construir o modelo através da função *build_model*. Esta função recebe como parâmetros o *timesteps*, o *multisteps* e o número de *features*, e tem os parâmetros número de neurónios (32 neurónios), ativação (*tanh*) e o *dropout rate* (0.4) pré-definidos na função. O modelo que a função *build_model* irá construir é composto por quatro camadas no total, duas camadas LSTM e duas camadas Dense, em que uma delas utiliza uma ativação diferente, utiliza *relu*. O *loss* utilizado é o mse, o otimizador utilizado é o Nadam e a métrica utilizada é a mae. No final é retornado o modelo construído. A figura seguinte é a representação do melhor modelo retirar para uma das quatro ruas disponíveis, neste caso diz respeito à Avenida dos Aliados.

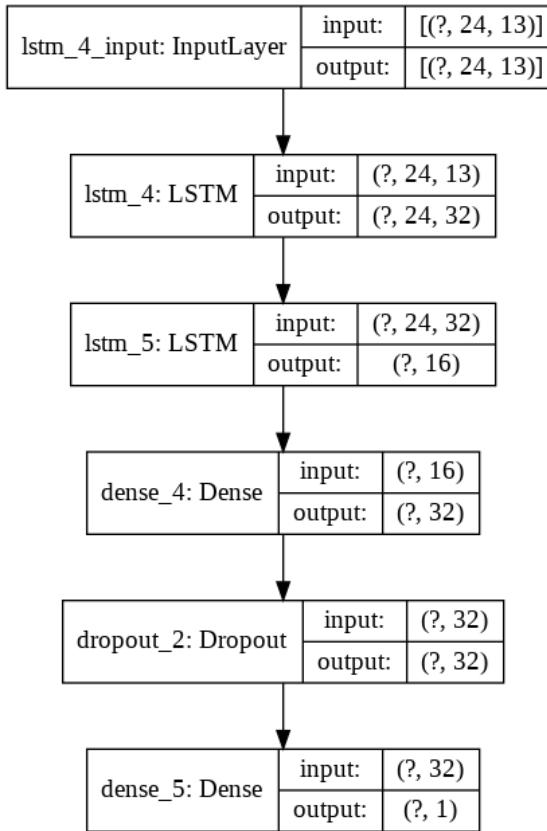


Figura 32: Modelo da rede construída

Com o modelo construído, foi necessário a criação de duas *callbacks* para utilizar no treino do modelo. Com o auxílio destas *callbacks* será possível evitar o *overfitting*, uma vez que o treino do modelo será encerrado atempadamente, ou seja, quando este não precisar de percorrer as épocas todas, irá parar o treino.

```

lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='mae',
                                            factor=0.5,
                                            patience=patience,
                                            min_lr=0.00005)

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_mae',
                                              min_delta=0,
                                              patience=20,
                                              verbose=verbose,
                                              mode='auto')

```

Depois de se definir as *callbacks*, é necessário realizar o *fit* do modelo e, para isso, utilizaram-se 100 épocas com os respetivos dados de validação, um *batch size* de 32 e as *callbacks* definidas. É importante referir que se utilizou os dados todos do conjunto de dados para treinar os modelos. Inicialmente testou-se com os dados apenas até ao dia e hora atual, no entanto os resultados que se irão visualizar no capítulo seguinte são melhores do que a primeira opção. No final do fit, são apresentados os gráficos tanto do valor do *loss* como do valor de MAE e é verificado se este é o melhor modelo deste *Time Series Split* ou não. Caso seja o melhor modelo, é guardado para ser utilizado na previsão. Caso contrário é descartado.

Concluído o *Time Series Split*, é apresentado o melhor modelo com a respetiva percentagem de mae e também é apresentado a média dos vários modelos gerados com o valor da média do MAE. Neste momento, está tudo preparado para se realizar a previsão. Para esse fim, invoca-se a função que se denominou de *forecast* fornecendo-lhe o melhor modelo, os *timesteps*, os *multisteps*, os *dataframes* necessários para realizar a previsão. A função *forecast* irá fazer a previsão consoante os *multisteps* que pretendemos, ou seja, se se pretende prever apenas uma hora, doze horas, 24 horas. Foi construído um ciclo que irá iterar até ao final das *multisteps* e utilizará a função auxiliar *predict_multistep* para gerar a previsão de cada hora (*multistep*). O conceito desta função auxiliar diz respeito à realização da previsão com base nas 24 horas anteriores e feita a previsão, o *dataframe* que contém as 24 horas anteriores, ou seja, os 24 *multisteps* anteriores é atualizado, isto é,



é-lhe removido o *multistep* mais antigo e adicionado o mais recente para que seja possível prever o próximo *multistep* do ciclo da função *forecast*. No final da execução é gerada uma lista com as previsão todas efetuadas.

Posto isto, são apresentadas todas as previsões e apresentado um gráfico final onde estão os valores das últimas 24 horas do *speed_diff* e as próximas 24 horas e ainda as 24 horas previstas pela função *forecast*, tal como é possível visualizar na figura seguinte.

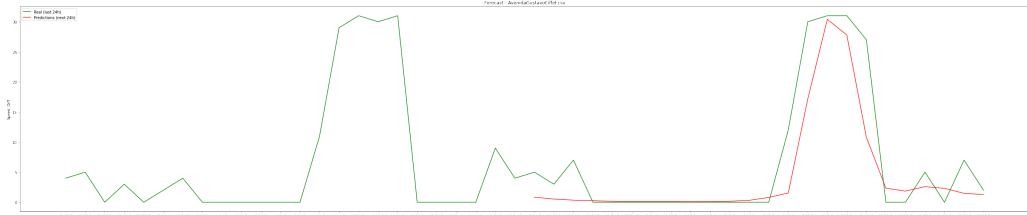


Figura 33: Previsão Avenida Gustavo Eiffel

Por último, foi elaborado um processo para retornar o valor da previsão e o valor da diferença entre os dados previstos e os dados reais para aquela previsão, sendo estes apresentados como o exemplo seguinte:

Real error (Km/h) : 7.67Km/h

Final error (Km/h) : 2.40Km/h (+/- 2.65Km/h)

O processo anteriormente descrito está representado na figura seguinte e, como podemos observar, são lidos um de quatro ficheiros CSV, são preparados os dados, é aplicada a função de super-visionamento, é utilizado o *Time Series Split* para construir os diferentes modelos e, por fim, é realizado o *forecast* que retorna o gráfico das previsões.

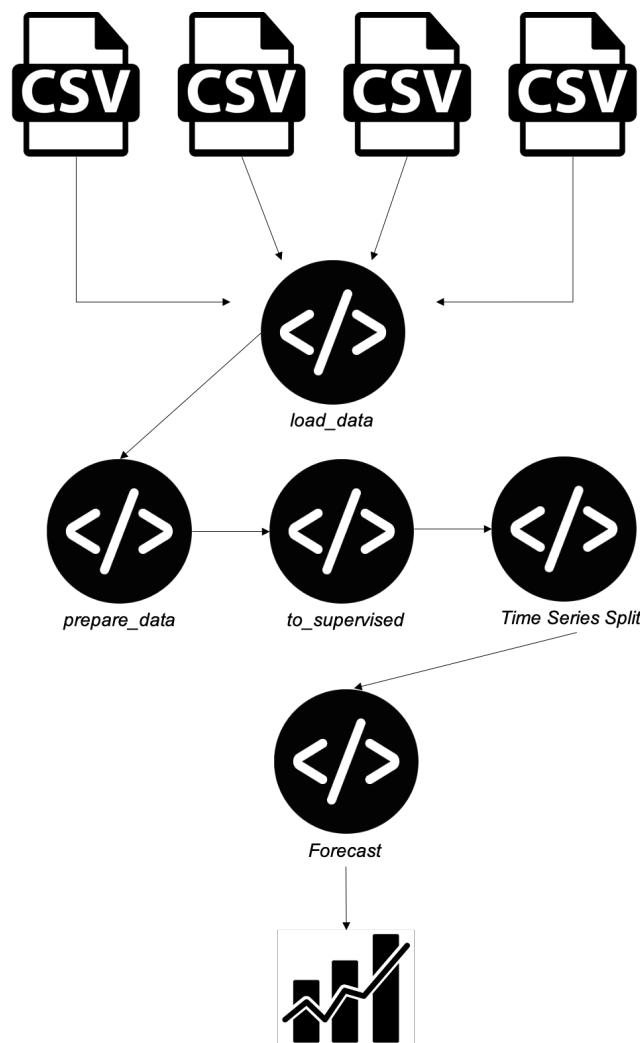


Figura 34: Processo até à previsão



6.3 Otimizações

No desenrolar do desenvolvimento do modelo, os variados resultados obtidos encorajaram o grupo a chegar uma fórmula de calcular quais os hiper parâmetros que retornam melhor valor. Este processo é executado de forma manual, procedimento normal face a este tipo de projetos, no entanto o grupo debruçou-se que possibilidades existem para automatizar e aumentar a eficiência, accuracy e diminuir loss

Para tal o grupo começou a investigar quais os algoritmos de otimização de hiper parâmetros disponíveis para auxiliar e descobriu duas propostas, *scikit-learn hyper-parameter tuning* e *Talos*, ambas ferramentas tem como funcionalidade dar um modelo e os parâmetros a serem implementados. O algoritmo escolhido para executar esta tarefa foi *Talos*, que pertence á biblioteca do keras, este algoritmo foi lançado em 2018[9] e segue a estratégia **POD**,(Prepare, Optimize, Deploy). Através da funcionalidade *scan* todas as combinações possíveis são testadas numa só iteração. De seguida são retornados os modelos com melhores resultados e os respectivos parâmetros que contribuíram para tal.

No que diz respeito á implementação do algoritmo no modelo foi preciso definir funcionalidades. Primeiro deve-se definir os parâmetros em que o algoritmo se vai basear para executar.

```
# Tuning
p = {
    'first_neuron': [32, 64],
    'batch_size': [32, 64],
    'epochs': [50, 100],
    'dropout': [0.2, 0.4],
    'optimizer': ['Adam', 'Nadam', 'RMSprop'],
    'losses': ['mse', 'mae'],
    'activation': ['relu', 'elu', 'tanh'],
    'last_activation': ['tanh', 'softmax', 'linear'],
}
```

A cada parâmetro acima mencionado, são definidos os valores em que cada um terá que ser testado, e no final a combinação com melhores resultados é explicita com a melhor combinação. A função *tunning* é adaptada para incluir todas as funcionalidades de uma só vez.

```
def tuning(x_train, y_train, x_val, y_val, params):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                    return_sequences=True,
                                    input_shape=(x_train.shape[1], x_train.shape[2]),
                                    activation=params['activation']))

    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                    return_sequences=False,
                                    dropout=params['dropout'],
                                    activation=params['activation']))

    model.add(tf.keras.layers.Dense(params['first_neuron'],
                                    activation=params['activation']))
    model.add(tf.keras.layers.Dropout(params['dropout']))

    model.add(tf.keras.layers.Dense(1, activation=params['last_activation']))

    model.compile(
        loss=params['losses'],
        optimizer=params['optimizer'],
        metrics=['mae']
    )

    history = model.fit(x_train, y_train,
                        validation_data=[x_val, y_val],
                        batch_size=params['batch_size'],
                        epochs=params['epochs'],
                        verbose=0)
```



```
return history, model
```

Os valores das componentes da função tunning foram alterados para incorporar as várias possibilidades de valor a serem testadas, p.e, em vez de possuírmos $activation=tahn$, temos agora $activation=params['activation']$, que possui agora a possiblidade de testar o modelo para as seguintes ativações '*relu*', '*elu*', '*tanh*', isto também foi aplicada ás restantes componentes.

Para executar o algoritmo utiliza-se a seguinte função.

```
scan_object = talos.Scan(X, Y,
                           model=tunning,
                           params=p,
                           experiment_name='speed_diff',
                           fraction_limit=0.1)
```

O tempo de execução do algoritmo é extenso, horas, e no final ao aceder ao conteúdo através do comando

```
analyze_object = talos.Analyze(scan_object)
analyze_object.best_params('val_mean_absolute_error', ['mean_absolute_error', 'loss',
                                                       'val_loss'])
```

São retornadas combinações com os melhores valores de *erro*.

Dropout	batch size	epochs	activation	last_activation	optimizer	first_neuron	losses
0.2	32	100	relu	softmax	RMSprop	32	mse
0.2	64	100	tanh	softmax	RMSprop	64	mse
0.4	32	100	relu	softmax	RMSprop	32	mae
0.4	64	50	relu	softmax	Nadam	32	mse
0.2	64	100	elu	softmax	Adam	64	mse
0.4	64	50	tanh	softmax	RMSprop	32	mse
0.4	32	100	relu	softmax	Adam	32	mse
0.2	64	100	elu	softmax	Nadam	64	mse
0.4	32	100	tanh	softmax	RMSprop	32	mae
0.2	64	50	tanh	softmax	Nadam	64	mae

Tabela 3: Resultados - *TALOS*



7 Análise de resultados

Concluído o processo de criação do modelo de *Deep Learning* para a previsão da diferença de velocidades (*speed diff*) num intervalo de tempo, foi possível obter vários resultados e proceder à análise dos mesmos.

Como foi explicado anteriormente, durante o processo de otimização do modelo foram feitas várias alterações nos parâmetros tais como, *batch size*, número de neurónios, número de camadas, quantidade de épocas de treino, tipo de otimizador usado, entre outras. Com base nos resultados obtidos no *Talos*, conclui-se que para o *batch size* os melhores valores eram 32 e 64, pois era com estes valores que o modelo se comportava melhor. Em relação ao numero de épocas de treino, pudemos afirmar que quanto maior, melhor é para a aprendizagem do modelo, no entanto limitamos este valor a 100 para não se torne muito custoso em termos de tempo. Quanto ao *patience* pode-se afirmar que o valor 10 foi o mais adequado dado que não se pretendia um valor baixo nem muito alto, no entanto pretendia-se um valor que nos fornece-se segurança nos resultados. Face ao resto dos parâmetros, as escolhas foram justificadas com o que foi dito anteriormente.

O resultados obtidos baseiam-se no cálculo do MAE (*mean absolute error*) para a previsão do *speed_diff* para cada uma das quatro ruas disponíveis da cidade do Porto. Relembrando que estes ficheiros contêm informação das condições atmosféricas e do fluxo de trânsito. Na secção seguinte serão apresentados os gráficos dos melhores modelos que foram obtidos a partir da execução do *Time Series Split* para cada rua.

7.1 Melhores modelos

Os modelos desenvolvidos ajustam-se muito bem ao conjunto de dados fornecido para treino e teste, mostram-se ineficazes para prever novos resultados, ou seja, não existe *overfitting* nem *underfitting*. No exemplo seguinte vemos as diferenças entre *overfitting*, *underfitting* e o que seria o balanço ideal.[6]

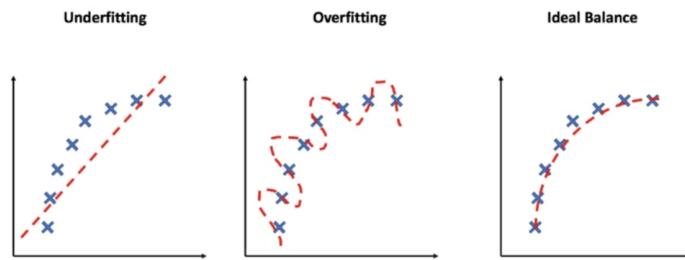


Figura 35: *Overfitting* vs *Underfitting* vs *Ideal Balance*

Como se pode comparar entre o que foi apresentado na figura anterior e os gráficos seguintes obtidos para cada rua, leva-nos a concluir que os modelos alcançados são balanceados e vão de encontro aos resultados que pretendíamos obter.

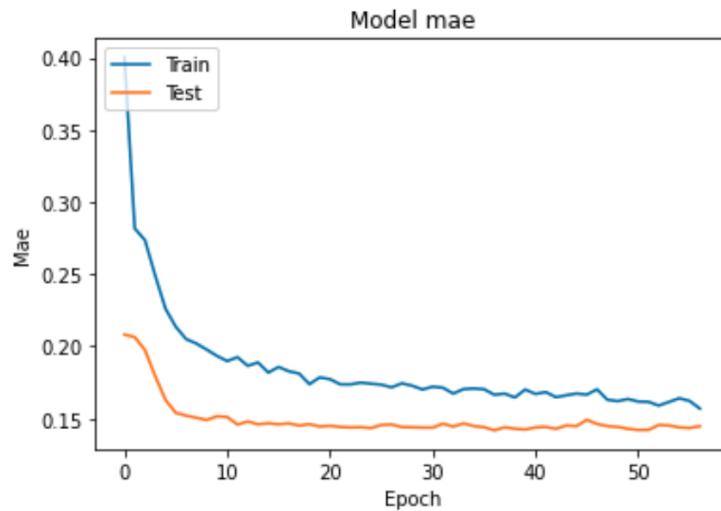
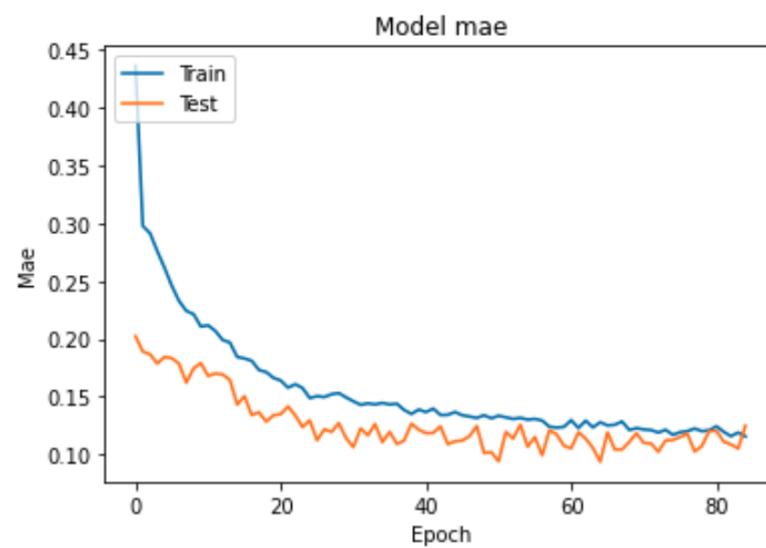
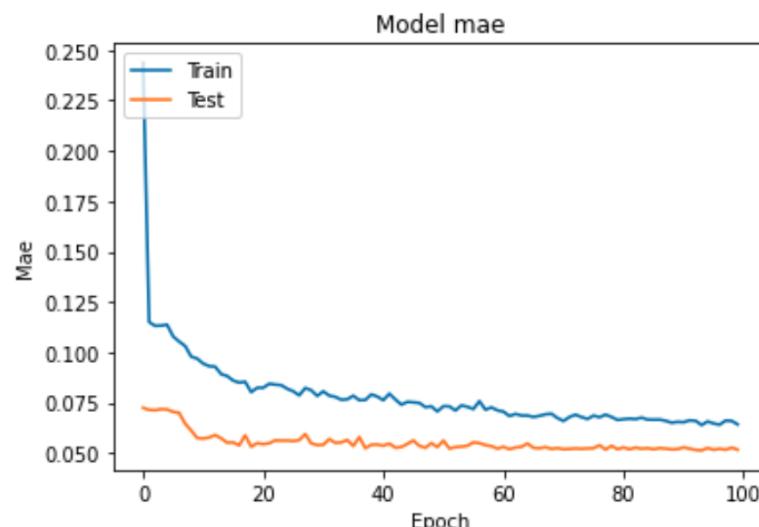


Figura 36: Modelo Avenida dos Aliados



17/17 - 0s - loss: 0.0615 - mae: 0.1250
mae: 12.50%

Figura 37: Modelo Avenida Gustavo Eiffel



17/17 - 0s - loss: 0.0079 - mae: 0.0517
mae: 5.17%

Figura 38: Modelo Rua Conde Vizela

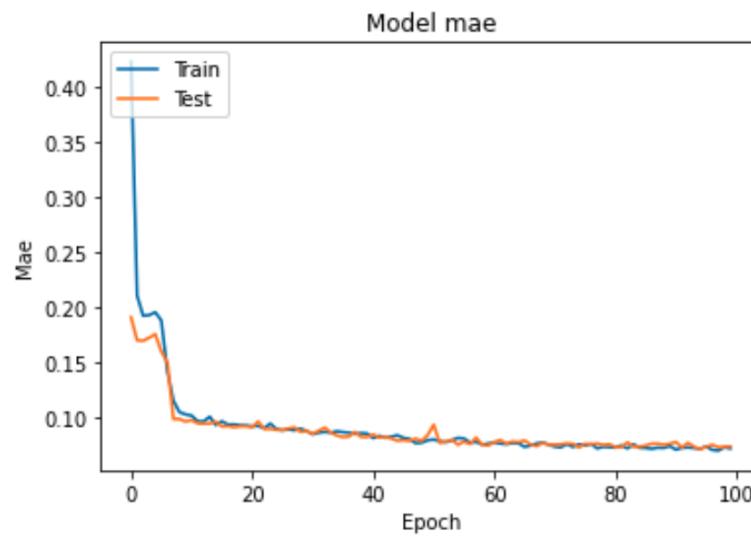


Figura 39: Modelo Rua Nova Alfândega

Os diferentes valores de MAE obtidos nos diferentes modelos dizem respeito aos diferentes dados, uma vez que cada modelo corresponde a uma rua diferente e cada uma tem dados diferentes para cada um dos seus atributos (colunas) do *dataset*. No entanto os valores obtidos são bastante satisfatórios em que o pior valor de MAE obtido diz respeito a 14.47% na rua Avenida dos Aliados e o melhor valor de MAE obtido diz respeito a 5.17% na rua Conde Vizela.

7.2 Previsões

Com os modelos gerados, pode-se finalmente realizar as previsões do "speed_diff". Foram realizadas várias tentativas para prever essa variável, no entanto foram aplicadas duas abordagens para que fosse possível verificar qual nos permitia obter resultados mais reais. As primeiras previsões realizadas utilizar uma normalização dos dados até à hora atual, ou seja, o *scaler* utilizado continha apenas os dados até à hora atual normalizados. Já numa fase mais avançada utilizou-se todos os dados normalizados, isto é, o *scaler* utilizado continha os dados do *dataset* todos normalizados. O resultados obtidos com a utilização do *scaler* com os dados todos normalizados foram ligeiramente melhores, o que nos leva a concluir que é melhor utilizar este método.

Dito isto, foram recolhidas várias previsões do dia 15 de Maio de 2020 num intervalo de horas para as diferentes ruas e os resultados estão representados na tabela seguinte. A tabela contém dois tipos de informação que, tal como foi apresentado no final da secção 6.2, são o valor da previsão (RR) e o valor da diferença entre os dados previstos e os dados reais para aquela previsão (FR).

Rua/Hora	Avenida dos Aliados	Rua Conde de Vizela	Avenida Gustavo Eiffel	Rua Nova da Alfândega
13h	RR: 8.66km/h	RR: 19.95km/h	RR: 8.97km/h	RR: 17.61km/h
	FR: 1.53km/h	FR: 3.67km/h	FR: 4.60km/h	FR: 1.70km/h
14h	RR: 8.58km/h	RR: 20.62km/h	RR: 8.96km/h	RR: 17.61km/h
	FR: 1.71km/h	FR: 3.50km/h	FR: 4.36km/h	FR: 1.88km/h
15h	RR: 8.61km/h	RR: 19.61km/h	RR: 8.98km/h	RR: 17.54km/h
	FR: 1.73km/h	FR: 3.47km/h	FR: 4.61km/h	FR: 1.83km/h
16h	RR: 8.61km/h	RR: 20.10km/h	RR: 8.97km/h	RR: 17.63km/h
	FR: 1.63km/h	FR: 3.40km/h	FR: 4.89km/h	FR: 1.75km/h
17h	RR: 8.60km/h	RR: 20.66km/h	RR: 8.96km/h	RR: 17.58km/h
	FR: 1.47km/h	FR: 4.68km/h	FR: 4.53km/h	FR: 1.75km/h

Tabela 4: Resultados - previsão Análise Ruas

Observando os resultados que estão na tabela anterior, é possível inferir os seguintes conclusões:

1. Ambos os erros sofrem pouca alteração ao longo das horas, o que nos leva a crer que os modelos estão bem construídos, tal como foi explicado anteriormente.
2. Obtiveram-se valores bastante bons para diferença entre os dados previstos e os dados reais para aquela previsão (FR), obtendo-se valores entre os 2km/h e os 5km/h dependendo da rua, o que consideramos um erro bastante baixo.
3. Em termos de previsão, não existe grandes atrasos na previsão com excessão das ruas com mais dados onde se nota mais o atraso na previsão e onde esta poderia ainda ser melhor.

Dito isto, iremos apresentar alguns dos melhores resultados obtidos em termos de previsões colocando na secção seguinte um grande obtido para cada uma das ruas testadas.

7.2.1 Melhores previsões

As quatro figuras seguintes representam a previsão obtida para cada rua numa determinada hora. A linha a verde representa 48 horas, isto é, as 24 horas anteriores à hora atual e as 24 horas seguintes à hora atual. A linha vermelha representa a previsão obtida para a hora seguinte à hora atual até as 23 horas seguintes, ou seja, corresponde a uma previsão de 24 horas desde a hora atual. Como podemos verificar, em todos os gráficos a previsão tenta acompanhar os dados reais, a linha vermelha em grande parte dos casos tem o mesmo percurso que os dados reais, o que é bastante satisfatório para nós.

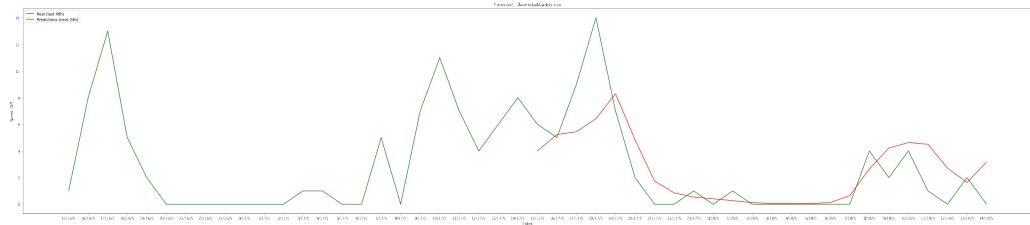


Figura 40: Previsões Avenida dos Aliados

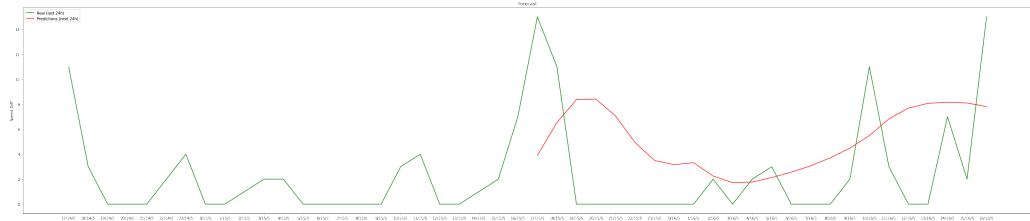


Figura 41: Previsões Rua Conde de Vizela

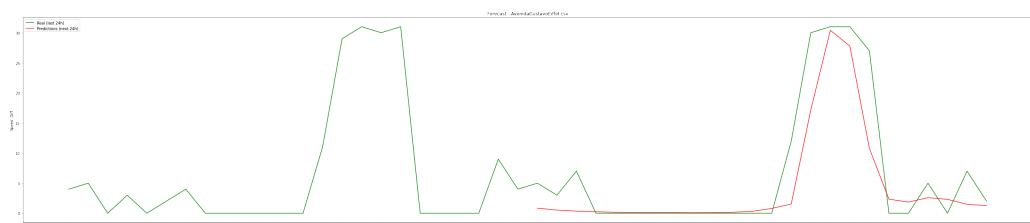


Figura 42: Previsões Avenida Gustavo Eiffel



7 ANÁLISE DE RESULTADOS

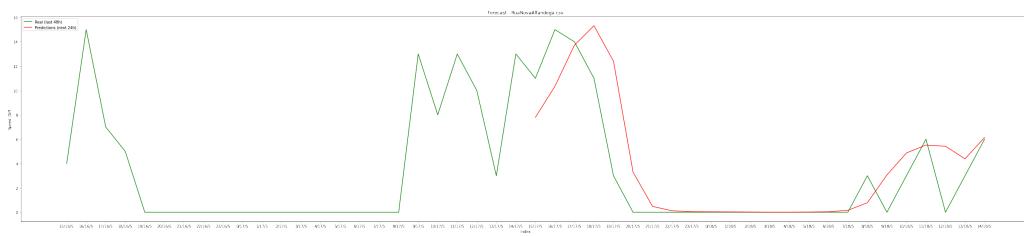


Figura 43: Previsões Rua Nova da Alfândega



8 Conclusão e trabalho futuro

Após a análise dos resultados, podemos afirmar que o modelo elaborado com as redes neuronais LSTM é robusto e capaz de prever com exatidão o que lhe é proposto. Em termos de metodologia e fazendo uma retrospectiva a tudo o que foi realizado, é possível afirmar da nossa parte que cumprimos o modelo que nos propusemos a seguir e o resultado obtido corresponde aos objetivos que nos propusemos a cumprir.

Ao longo do projeto, tivemos vários percalços que o grupo tentou ultrapassar da melhor forma. É importante dar ênfase às tentativas que o grupo precisou para alcançar o melhor modelo possível para cada rua disponível e ainda à previsão que se realizou para cada rua. O algoritmo construído para o *forecast* foi um grande obstáculo para nós, no entanto, e com base nos resultado apresentados no capítulo anterior, foi bem construído o que nos deixa satisfeitos.

Contudo, sabemos que há aspectos que poderiam ser melhorados e que poderíamos tentar uma abordagem diferente, onde, por exemplo, ao invés de separar os dados por ruas e criar quatro conjuntos de dados diferentes, aglomerar toda a informação num conjunto de dados e construir o modelo em volta desse conjunto, mantendo o mesmo objetivo, ou seja, que o modelo seja capaz de calcular todas as necessidades que se sejam visíveis para a redução da sinistralidade nas estradas da cidade do Porto.



9 Referências

Referências

- [1] Akshay Mungekar, Nikita Parab, Prateek Nima and Sanchit Pereira. Quora Insincere Questions Classification, 2019. https://www.researchgate.net/publication/334549103_Quora_Insincere_Questions_Classification
- [2] Trovoada afeta 127 semáforos no Porto. <https://www.jn.pt/local/noticias/porto/porto/trovoada-afeta-127-semaforos-no-porto-11240870.html>
- [3] CRISP-DM methodology leader in data mining and big data, <https://towardsdatascience.com/crisp-dm-methodology-leader-in-data-mining-and-big-data-467efd3d3781>
- [4] Cross-Industry Standard Process for Data Mining FEUP, https://paginas.fe.up.pt/~ec/files_0405/slides/02%20CRISP.pdf?fbclid=IwAR26NsvrzD10z_-jF3QKwiTUZKFvtGJrr28gW_WNeIFX-U-Ofmv0ZKImx1Y
- [5] What is the CRISP-DM methodology?, https://www.sv-europe.com/crisp-dm-methodology/?fbclid=IwAR2y1dQ_JkikLr1eA10D5u4bFjij24p6c8CRAFIWdYVR0SQgHwARCeCy5UI#modeling
- [6] Overfitting and Underfitting, <https://mc.ai/overfitting-and-underfitting-bug-in-ml-models/>
- [7] Gelo na estrada em dias de frio extremo, <https://www.circulaseguro.pt/gelo-na-estrada-em-dias-de-frio-extremo/>
- [8] Investigação de acidentes rodoviários: recolha de informação, <https://bit.ly/2Z6s1w5>
- [9] Using Talos for Feature Hyperparameter Optimization? <https://neurosphere.io/blog/2019/04/using-talos-for-feature-hyperparameter-optimization/>
- [10] Towards Better Keras Modeling https://alphascientist.com/hyperparameter_optimization_with_talos.html
- [11] Talos Tutorial https://autonomio.github.io/docs_talos/#introduction
- [12] Talos Jupyter <https://nbviewer.jupyter.org/github/autonomio/talos/blob/master/examples/Hyperparameter%20Optimization%20with%20Keras%20for%20the%20Iris%20Prediction.ipynb>
- [13] Hyperparameter Optimization with Keras <https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53>

A Anexos

Nesta secção, serão anexados alguns modelos, *outputs* e os *scripts* elaborados.

A.1 Workflow completo em KNIME do tratamento de dados

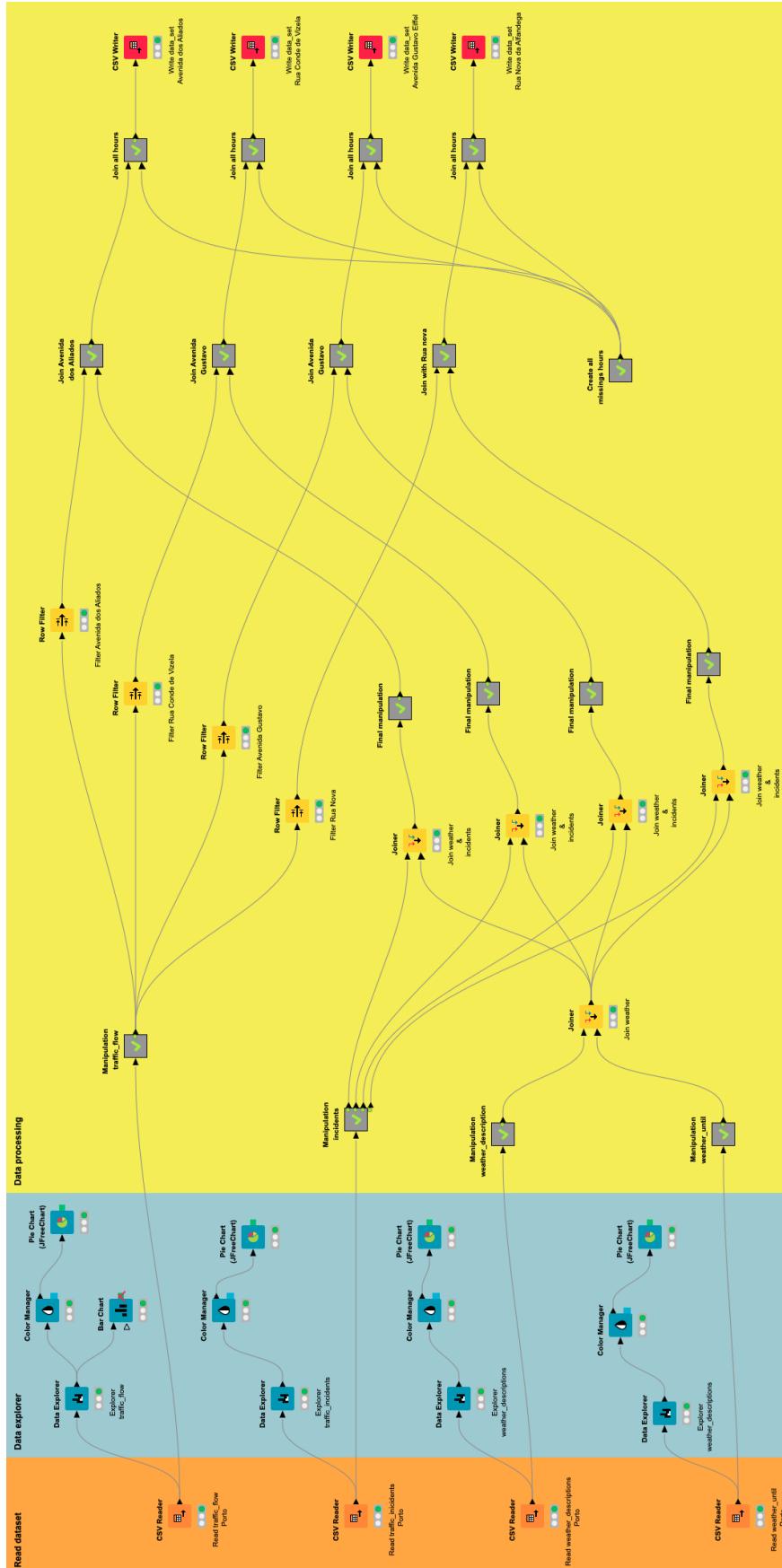


Figura 44: Workflow completo em KNIME do tratamento de dados

A.2 Random Search - Rua Conde de Vizela

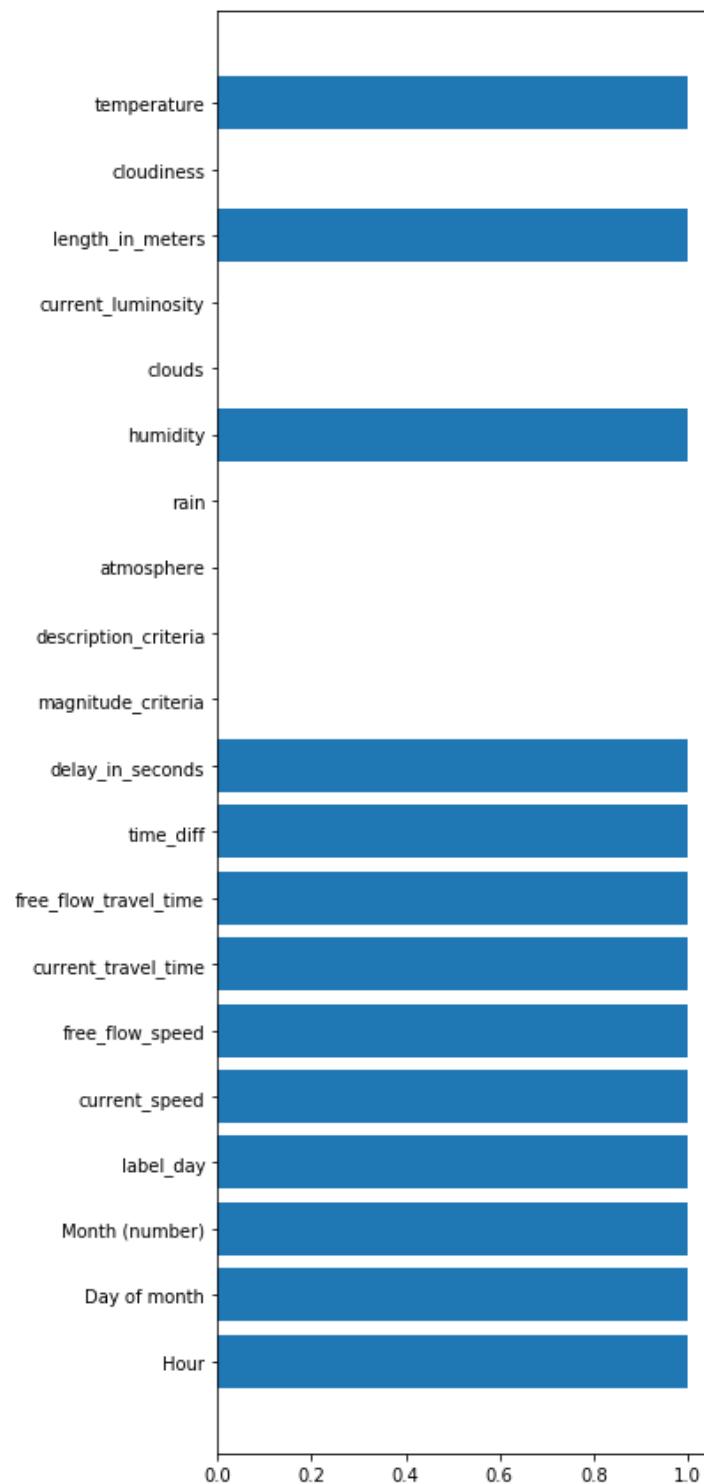


Figura 45: Rua Conde de Vizela - *Random Search*

A.3 XGBoost - Rua Conde de Vizela

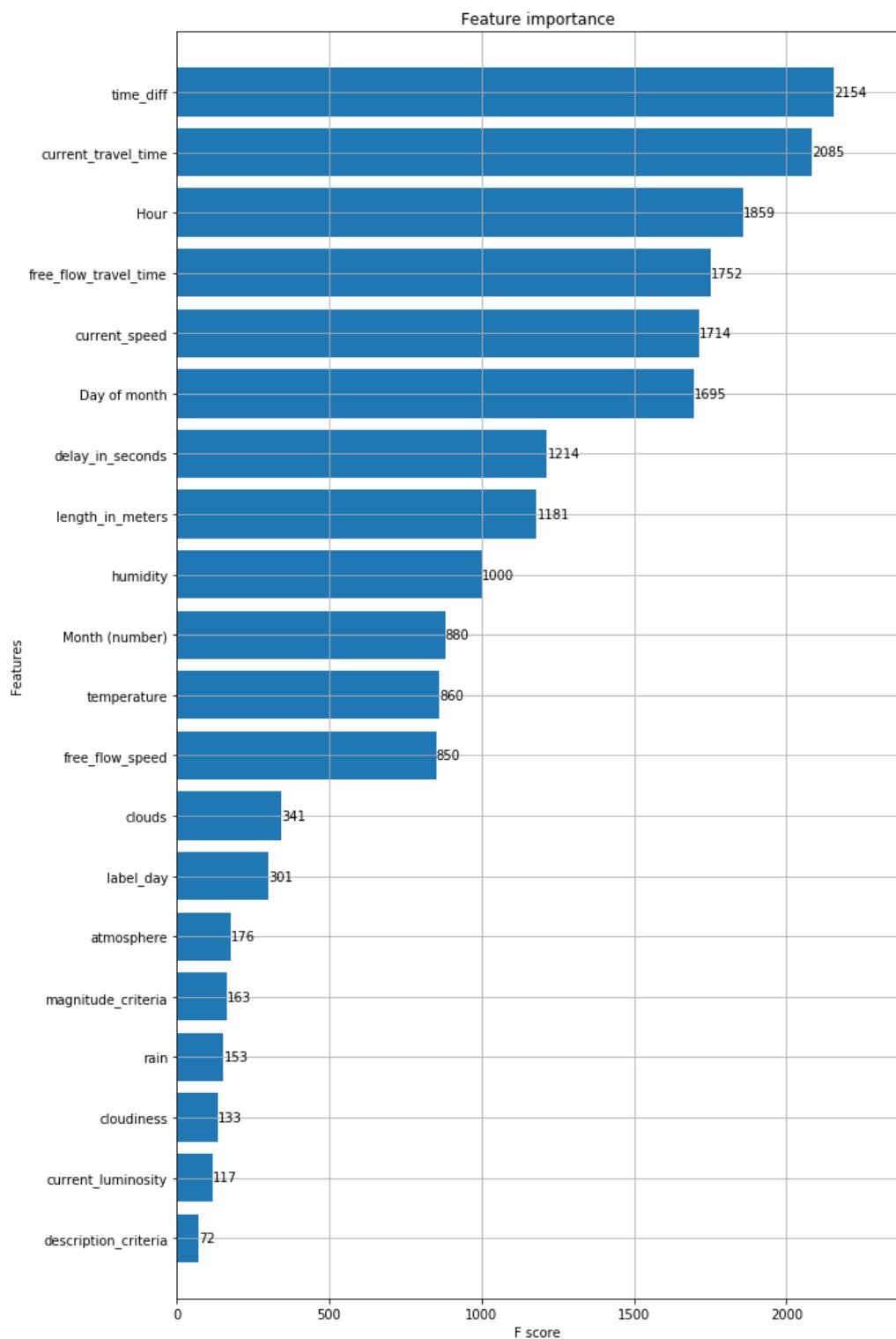


Figura 46: Rua Conde de Vizela - XGBoost

A.4 *Random Search* - Avenida Gustavo Eiffel

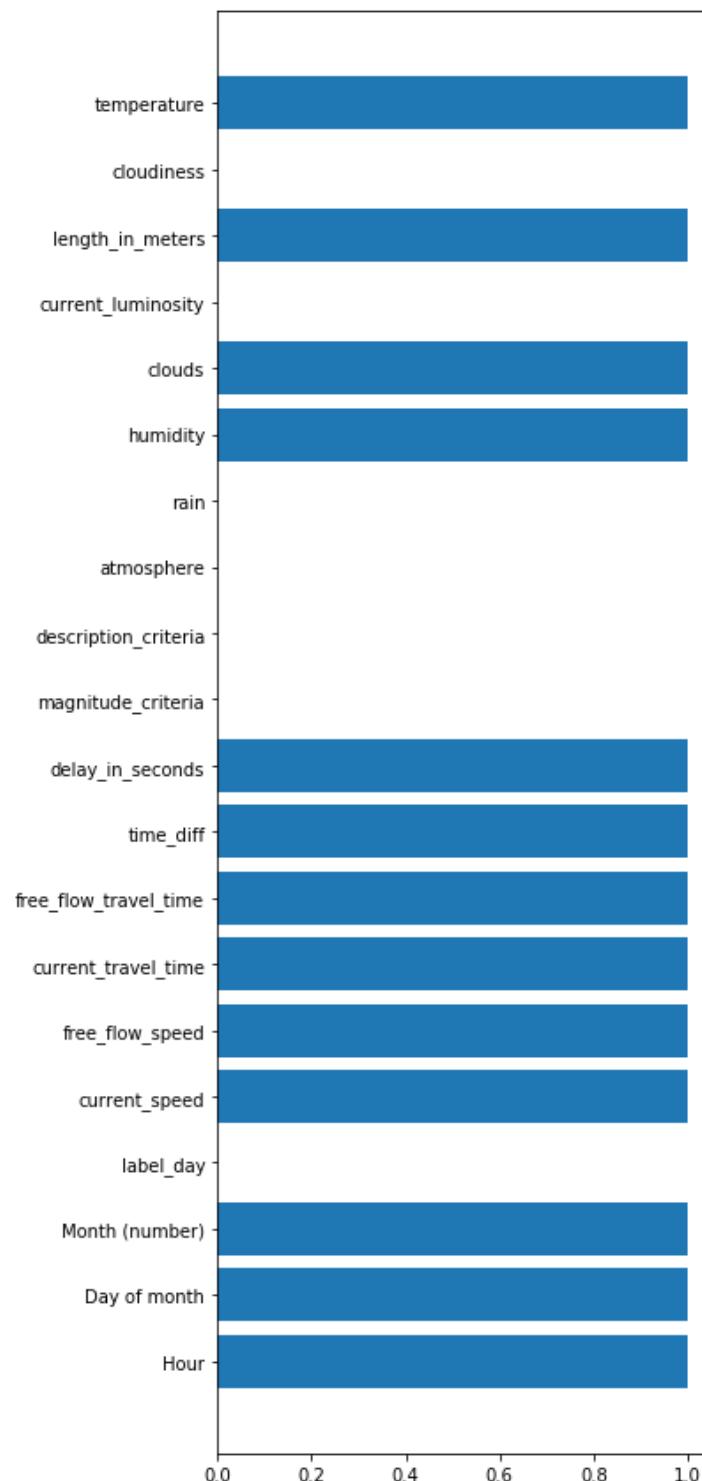


Figura 47: Avenida Gustavo Eiffel - *Random Search*

A.5 XGBoost - Avenida Gustavo Eiffel

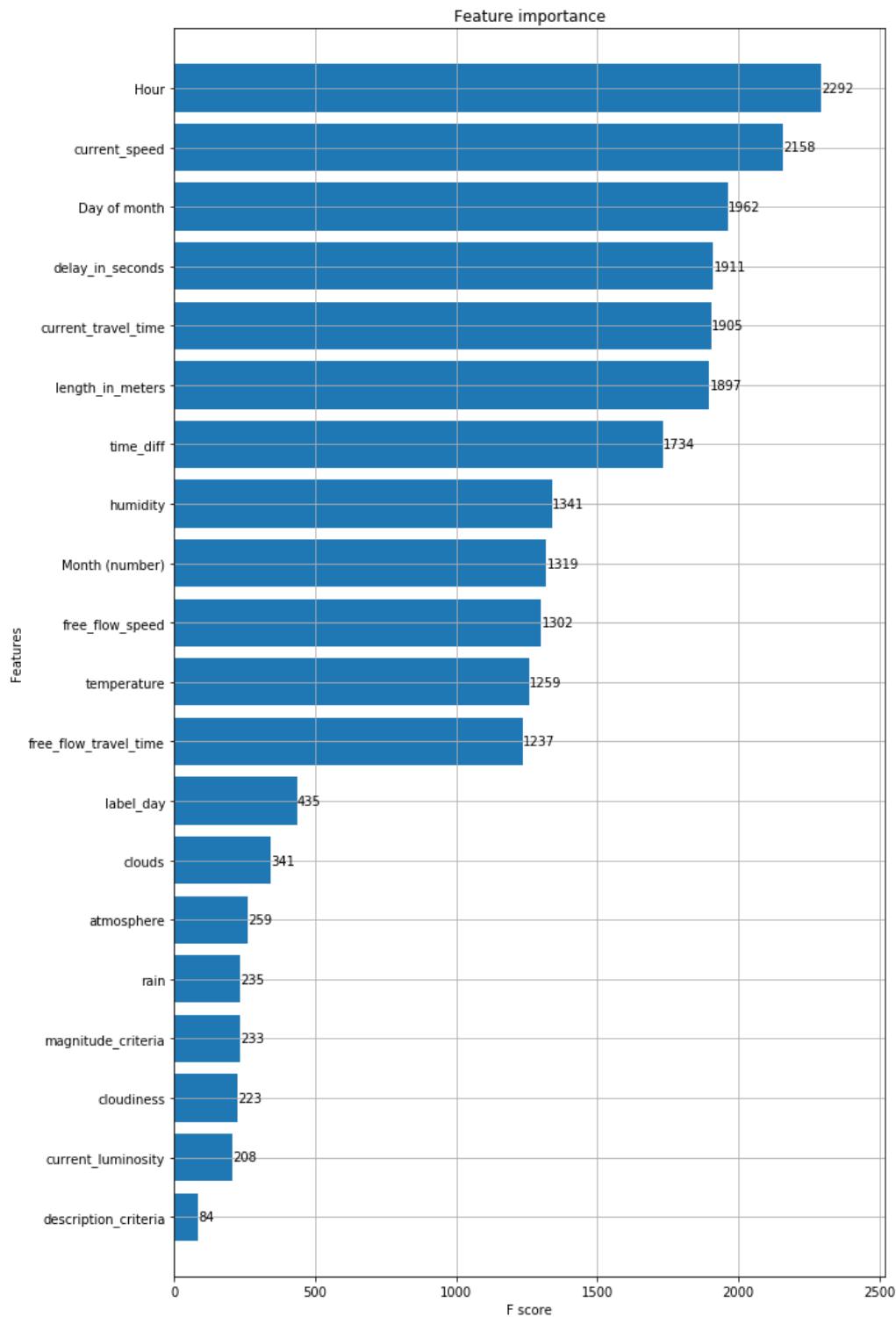


Figura 48: Avenida Gustavo Eiffel - XGBoost

A.6 Random Search - Rua Nova da Alfândega

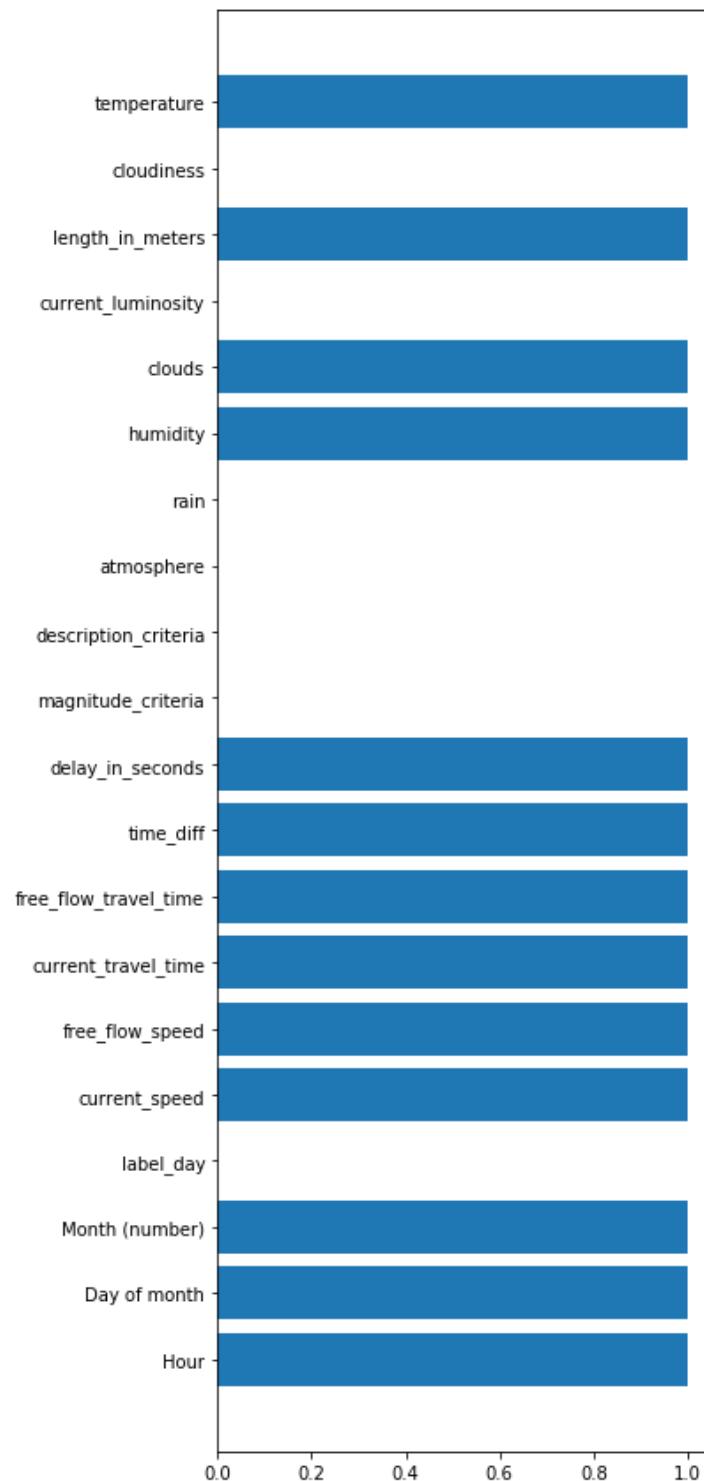


Figura 49: Rua Nova da Alfândega - *Random Search*

A.7 XGBoost - Rua Nova da Alfândega

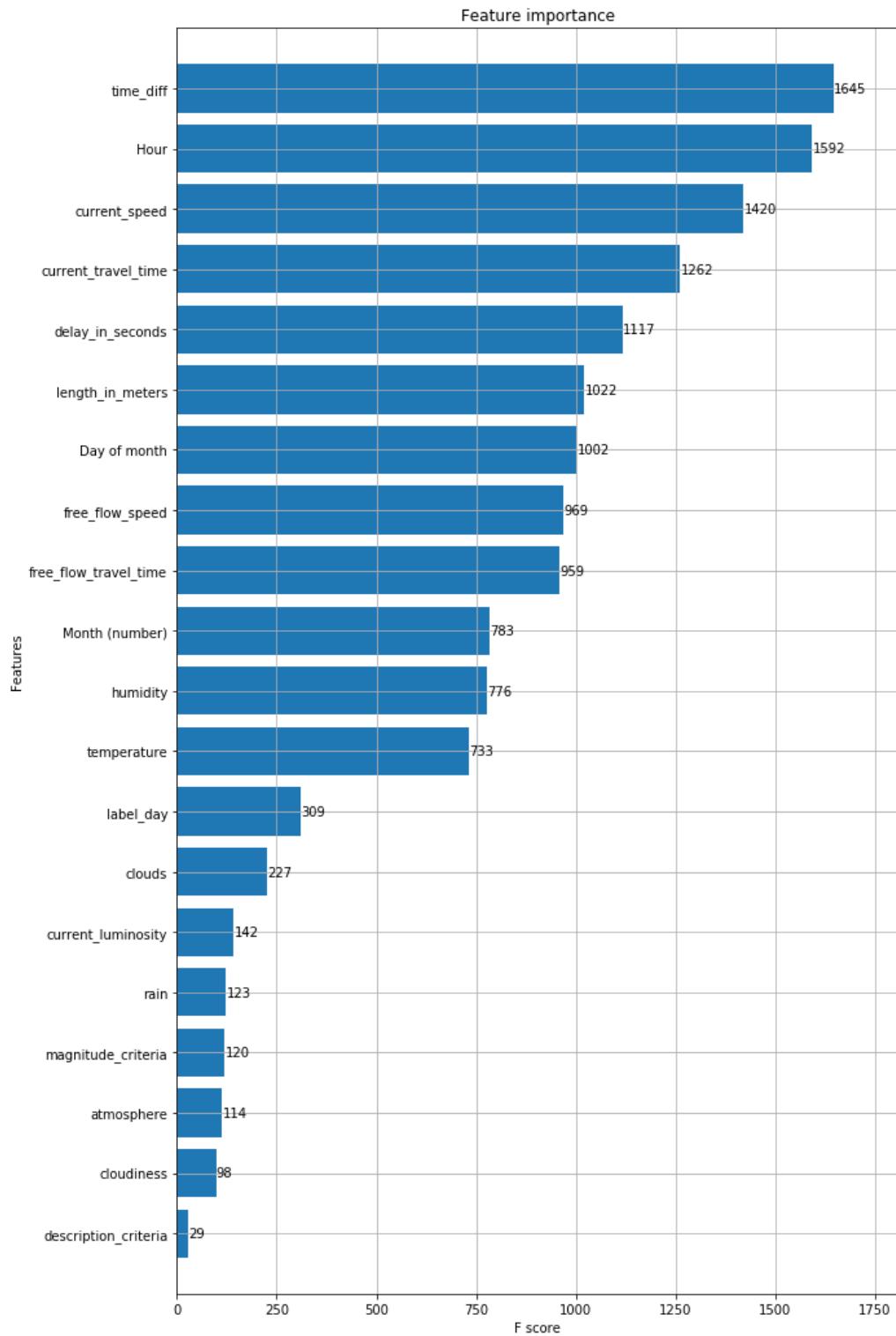


Figura 50: Rua Nova da Alfândega - XGBoost



A.8 Script predict_tunning.py

```
import pandas as pd
import numpy as np
import talos
from talos.utils import hidden_layers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import tensorflow as tf
import datetime
from sklearn.model_selection import TimeSeriesSplit
from numpy import nan
from numpy import isnan
from numpy import array

#tf.random.set_seed(91195003)
#np.random.seed(91195003)
#for an easy reset backend session state
tf.keras.backend.clear_session()

,,,
Read csv file
,,,
def load_data(file):
    df = pd.read_csv(file, encoding='utf-8', index_col='row_ID')
    df.index.name = 'Index'
    return df

,,,
Fill missing values with a value at the same time one day ago
,,,
def fill_missing(df):
    df['delay_in_seconds'].fillna(0, inplace=True)
    df['length_in_meters'].fillna(0, inplace=True)
    values = df.values
    time = 24 * 7
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if col == 9 or col == 12:
                time = 1
            else:
                time = 24 * 7
            if isnan(values[row, col]):
                values[row, col] = values[row - time, col]
    new_df = pd.DataFrame(values, columns=df.columns, index=df.index)
    return new_df

,,,
Prepare dataset
,,,
def prepare_data(df, norm_range=(-1,1)):
    # Drop columns
    columns = ['label_day', 'magnitude_criteria', 'atmosphere',
               'description_criteria', 'rain', 'clouds',
               'current_luminosity', 'cloudiness']
    data = df.drop(columns=columns)
    # Make dataset numeric
    data = data.astype('float32')
    # Missing Value
    df_temp = fill_missing(data)
    print(df_temp.isnull().sum())
```



```
# Normalized data
scaler = MinMaxScaler(feature_range=norm_range)
data_scaled = scaler.fit_transform(df_temp.values)
new_df = pd.DataFrame(data_scaled, columns=df_temp.columns, index=df_temp.index)

return new_df, scaler, df_temp, columns

,,,
Creating a supervised problem
,,,

def to_supervised(df, timesteps, multisteps, nr_features):
    data = df.values
    X, y = list(), list()
    data_size = len(data)
    for curr_pos in range(data_size):
        end_timesteps = curr_pos + timesteps
        begin_prev = end_timesteps
        end_prev = end_timesteps + 1
        if end_prev < data_size:
            X.append(data[curr_pos:end_timesteps,:])
            y.append(data[begin_prev:end_prev,5])

    X = np.reshape(np.array(X), (len(X), timesteps, nr_features))
    y = np.reshape(np.array(y), (len(y), 1))

return X, y

,,,
Tunning
,,,

def tunnig(x_train, y_train, x_val, y_val, params):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                   return_sequences=True,
                                   input_shape=(x_train.shape[1],
                                               x_train.shape[2]),
                                   activation=params['activation']))

    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                   return_sequences=False,
                                   dropout=params['dropout'],
                                   activation=params['activation']))

    model.add(tf.keras.layers.Dense(params['first_neuron'],
                                   activation=params['activation']))
    model.add(tf.keras.layers.Dropout(params['dropout']))

    model.add(tf.keras.layers.Dense(1, activation=params['last_activation']))

    model.compile(
        loss=params['losses'],
        optimizer=params['optimizer'],
        metrics=['mae']
    )

    history = model.fit(x_train, y_train,
                         validation_data=[x_val, y_val],
                         batch_size=params['batch_size'],
                         epochs=params['epochs'],
                         verbose=0)

return history, model
```



A.9 *Script predict.py*

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import tensorflow as tf
import datetime
from sklearn.model_selection import TimeSeriesSplit
from numpy import nan
from numpy import isnan
from numpy import array

tf.random.set_seed(91195003)
np.random.seed(91195003)
#for an easy reset backend session state
tf.keras.backend.clear_session()

,,
Read csv file
,,
def load_data(file):
    df = pd.read_csv(file, encoding='utf-8', index_col='row_ID')
    df.index.name = 'Index'
    return df

,,
Fill missing values with a value at the same time one day ago
,,
def fill_missing(df):
    df['delay_in_seconds'].fillna(0, inplace=True)
    df['length_in_meters'].fillna(0, inplace=True)
    values = df.values
    time = 24 * 7
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if col == 9 or col == 12:
                time = 1
            else:
                time = 24 * 7
            if isnan(values[row, col]):
                values[row, col] = values[row - time, col]
    new_df = pd.DataFrame(values, columns=df.columns, index=df.index)
    return new_df

,,
Prepare dataset
,,
def prepare_data(df, norm_range=(-1,1)):
    # Drop columns
    columns = ['label_day', 'magnitude_criteria', 'atmosphere',
               'description_criteria', 'rain', 'clouds',
               'current_luminosity', 'cloudiness']
    data = df.drop(columns=columns)

    # Make dataset numeric
    data = data.astype('float32')

    # Missing Value
    data_temp = fill_missing(data)
    #data_temp = data_temp.interpolate(method='linear',
```



```
        limit_direction='forward' , axis=0)
#print( data_temp . isnull () . sum () )

# Normalized data
scaler = MinMaxScaler( feature_range=norm_range)
data_scaled = scaler . fit_transform( data_temp . values)
new_data = pd . DataFrame( data_scaled , columns= data . columns , index= data . index)

# Data until now
index = str( now . hour ) + ' / ' + str( now . day ) + ' / ' + str( now . month )
df_until_now = new_data . loc [ : index]

return df_until_now , scaler , columns , new_data

, ,
Creating a supervised problem
, ,

def to_supervised( df , timesteps , multisteps , nr_features ):
    data = df . values
    X, y = list () , list ()
    data_size = len ( data )
    for curr_pos in range( data_size ):
        end_timesteps = curr_pos + timesteps
        begin_prev = end_timesteps
        end_prev = end_timesteps + 1
        if end_prev < data_size:
            X . append( data [ curr_pos : end_timesteps , : ] )
            y . append( data [ begin_prev : end_prev , 5 ] )

    X = np . reshape( np . array( X ) , ( len ( X ) , timesteps , nr_features ) )
    y = np . reshape( np . array( y ) , ( len ( y ) , 1 ) )

return X, y

, ,
Buid the model
, ,

def build_model( timesteps , features , multisteps , n_neurons=32 ,
                 activation='tanh' , dropout_rate=0.4):
    model = tf . keras . models . Sequential()
    model . add( tf . keras . layers . LSTM( int ( n_neurons ) ,
                                             activation=activation ,
                                             input_shape=( timesteps , features ) ,
                                             return_sequences=True))

    model . add( tf . keras . layers . LSTM( int ( n_neurons / 2 ) ,
                                             activation=activation ,
                                             dropout=dropout_rate ,
                                             return_sequences=False))

    model . add( tf . keras . layers . Dense( int ( n_neurons ) , activation='relu' ))
    model . add( tf . keras . layers . Dropout( dropout_rate ))

    model . add( tf . keras . layers . Dense( 1 , activation='tanh' ))

    model . compile(
        loss=tf . keras . losses . mse ,
        optimizer=tf . keras . optimizers . Nadam( 0.001 ) ,
        metrics=[ 'mae' ]
    )

return model
```



```
, , ,  
Forecast  
, , ,  
def forecast(best_model, X, timesteps, multisteps, scaler, columns,  
nr_features, df_scaled):  
    input_seq = X[-1:]  
    inp = input_seq  
    predictions = list()  
    for step in range(1, multisteps+1):  
        yhat = best_model.predict(inp, verbose=verbose)[0]  
        yhat_inversed, inp = predict_multistep(yhat, scaler, columns, inp,  
                                                timesteps, step, df_scaled)  
        predictions.append(yhat_inversed)  
  
    return predictions  
  
, , ,  
Predict multistep  
, , ,  
def predict_multistep(yhat, scaler, columns, inp, timesteps, step, df_scaled):  
    # Save prediction(speed_diff) on data frame  
    df_aux = pd.DataFrame(np.nan, index=range(0,1), columns=columns)  
    df_aux['speed_diff'] = yhat[0]  
  
    # Inverse scale of prediction  
    yhat_inversed = scaler.inverse_transform(df_aux.values)[0][5]  
    yhat_inversed = round(yhat_inversed, 2)  
  
    # Concatenate data from last timestep and prediction, save on data frame  
    inp = np.concatenate((inp, np.array(df_aux.values)[:,None]), axis=1)  
    df_temp = pd.DataFrame(inp[0], index=range(0,timesteps+1), columns=columns)  
  
    # Fill last multi_step with values df_scaled and prediction speed_diff  
    next_multi_step = datetime.datetime.today() + datetime.timedelta(hours=step)  
    index = str(next_multi_step.hour) + '/' + str(next_multi_step.day) + '/' +  
           str(next_multi_step.month)  
    df_temp.iloc[-1:] = df_scaled.loc[[index]].values  
    df_temp.iloc[-1, df_temp.columns.get_loc('speed_diff')] = yhat[0]  
  
    # Drop last timestep  
    inp_temp = list()  
    inp_temp.append(df_temp.values)  
    inp = np.reshape(np.array(inp_temp), (len(inp_temp), timesteps+1, nr_features))  
    inp = np.delete(inp, (0), axis=1)  
  
    return yhat_inversed, inp  
  
, , ,  
Plot Loss  
, , ,  
def plot_loss(history, count):  
    plt.plot(history.history['loss'])  
    plt.plot(history.history['val_loss'])  
    plt.title('Model loss')  
    plt.ylabel('Loss')  
    plt.xlabel('Epoch')  
    plt.legend(['Train', 'Test'], loc='upper left')  
    plt.show()  
  
, , ,  
Plot Mae
```



```
,,,  
def plot_mae(history , count):  
    plt.plot(history.history [ 'mae' ])  
    plt.plot(history.history [ 'val_mae' ])  
    plt.title('Model_mae')  
    plt.ylabel('Mae')  
    plt.xlabel('Epoch')  
    plt.legend([ 'Train' , 'Test' ] , loc='upper_left')  
    plt.show()  
  
,,,  
Plot Forecast  
,,  
def plot_forecast(index , real , predictions , multisteps , title):  
    plt.figure(figsize=(48, 10))  
    plt.plot(index , real , color='green' , label='Real')  
    plt.plot(range(len(real)-multisteps , len(real)+len(predictions)-multisteps) ,  
             predictions , color='red' , label='Prediction')  
    plt.title('Forecast--' + title)  
    plt.ylabel('Speed_Diff')  
    plt.xlabel('Index')  
    plt.legend([ 'Real_(last_48h)' , 'Predictions_(next_-' + str(multisteps) + 'h)' ] ,  
              loc='upper_left')  
    plt.savefig('forecast.png')  
    plt.show()  
  
##### MAIN #####  
  
,,,  
Vars  
,,  
nr_features = 13  
timesteps = 24  
multisteps = 24  
epochs = 100  
batch_size = 32  
verbose = 0  
patience = 10  
n_splits = 5  
now = datetime.datetime.now()  
  
# Files  
#file_path = 'AvenidaAliados.csv'  
#file_path = 'AvenidaGustavoEiffel.csv'  
file_path = 'RuaCondeVizela.csv'  
#file_path = 'RuaNovaAlfandega.csv'  
  
# Load data  
df = load_data(file_path)  
  
# Prepare data  
df_until_now , scaler , drop_columns , df_scaled = prepare_data(df)  
  
df_copy = df.copy()  
df_complete = df_copy.drop(columns=drop_columns)  
df_complete = fill_missing(df_complete)  
  
# Num features to use  
print("Nr_Features:" , df_until_now.shape[1])
```



```
nr_features = df_until_now.shape[1]
columns = df_until_now.columns

# To supervisioned
X, Y = to_supervised(df_scaled, timesteps, multisteps, nr_features)

print("Shape_X:", X.shape)
print("Shape_Y:", Y.shape)

# TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits)
cvscores = list()
split_num = 1
current_mae = 100
best_model = ''
for train_index, test_index in tscv.split(X):
    print(10*'-' + '_Begin_Time_Series_Split_N' + str(split_num) + '-' + 10*'-')
    # Get values form time series split
    x_train, x_test = X[train_index], X[test_index]
    y_train, y_test = Y[train_index], Y[test_index]

    # Create model
    model = build_model(timesteps, nr_features, multisteps)

    # Experiment the model
    lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='mae', factor=0.5,
                                                patience=patience, min_lr=0.00005)
    early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_mae',
                                                min_delta=0, patience=25, verbose=verbose, mode='auto')

    # Fit the model
    print('>_Fit_the_model,_please_wait... ')
    history = model.fit(x_train, y_train,
                         epochs=epochs,
                         validation_data=(x_test, y_test),
                         batch_size=batch_size,
                         verbose=verbose,
                         shuffle=False,
                         callbacks=[early_stop, lr])

    # Plot loss
    plot_loss(history, str(split_num))

    # Plot mae
    plot_mae(history, str(split_num))

    # Evaluate the model
    scores = model.evaluate(x_test, y_test, verbose=verbose)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1]*100)

    # Save best model
    if scores[1] < current_mae:
        current_mae = scores[1]
        best_model = model

    split_num += 1
    print(49*'-')
    print('\n')

# Final score
print(best_model.summary())
tf.keras.utils.plot_model(best_model, 'best_model.png', show_shapes=True)
print('Best_(mae): %.2f%%' % (current_mae*100))
print('Final_score(mae): %.2f%% (+/- %.2f%%)\n' %
```



```
(np.mean(cvscores), np.std(cvscores)))
```

```
# Forecast
X_now, y_now = to_supervised(df_until_now, timesteps, multisteps, nr_features)
predictions = forecast(best_model, X_now, timesteps, multisteps, scaler, columns,
                       nr_features, df_scaled)
step = 1
print('\n')
print(10*( '-') + ' ' + file_path + ' ' + 10*( '-'))
df_until_now_normal = scaler.inverse_transform(df_until_now)
aux_df = pd.DataFrame(df_until_now_normal, columns=df_until_now.columns,
                       index=df_until_now.index)
print('\n> Prediction for the next %d MultiSteps\n' % multisteps)
print('> speed_diff(max): %d' % aux_df['speed_diff'].max())
print('> speed_diff(min): %d\n' % aux_df['speed_diff'].min())
for pred in predictions:
    next_multi_step = datetime.datetime.today() + datetime.timedelta(hours=step)
    print('> MultiStep N %d(%d(horas)---%d(dia)---%d(mes)): %.2f' % (step,
                                                                      next_multi_step.hour,
                                                                      next_multi_step.day,
                                                                      next_multi_step.month))
    step += 1
```

```
# Plot results / forecast
next_hours = datetime.datetime.today() + datetime.timedelta(hours=multisteps)
index = str(next_hours.hour) + '/' + str(next_hours.day) + '/' +
        str(next_hours.month)
df_complete_until_now = df_complete.loc[:index]
df_last_timesteps = df_complete_until_now.tail(timesteps*2)
plot_forecast(df_last_timesteps.index.values,
              df_last_timesteps['speed_diff'].values,
              predictions, multisteps, file_path)
```

```
# Evaluate model
df_aux = pd.DataFrame(np.nan, index=range(0,1), columns=columns)
df_aux['speed_diff'] = current_mae
real_error = scaler.inverse_transform(df_aux.values)[0][5]
print("Real error (Km/h): %.2f Km/h\n" % round(real_error,2))

df_last_timesteps_aux = df_complete_until_now.tail(timesteps)['speed_diff'].values
errors = []
for step in range(len(predictions)):
    error = abs(predictions[step]-df_last_timesteps_aux[step])
    errors.append(error)
print('Final error (Km/h): %.2f Km/h (+/- %.2f Km/h)\n' % (np.mean(errors),
                                                               np.std(errors)))
```

A.10 Script talos.py (Talos)

```
import pandas as pd
import numpy as np
import talos
from talos.utils import hidden_layers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import tensorflow as tf
import datetime
from sklearn.model_selection import TimeSeriesSplit
from numpy import nan
from numpy import isnan
from numpy import array

#tf.random.set_seed(91195003)
#np.random.seed(91195003)
```



```
#for an easy reset backend session state
tf.keras.backend.clear_session()

,,,
Read csv file
,,,

def load_data(file):
    df = pd.read_csv(file, encoding='utf-8', index_col='row_ID')
    df.index.name = 'Index'
    return df

,,,
Fill missing values with a value at the same time one day ago
,,,

def fill_missing(df):
    df['delay_in_seconds'].fillna(0, inplace=True)
    df['length_in_meters'].fillna(0, inplace=True)
    values = df.values
    time = 24 * 7
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if col == 9 or col == 12:
                time = 1
            else:
                time = 24 * 7
            if isnan(values[row, col]):
                values[row, col] = values[row - time, col]
    new_df = pd.DataFrame(values, columns=df.columns, index=df.index)
    return new_df

,,,
Prepare dataset
,,,

def prepare_data(df, norm_range=(-1,1)):
    # Drop columns
    columns = ['label_day', 'magnitude_criteria', 'atmosphere',
               'description_criteria', 'rain', 'clouds',
               'current_luminosity', 'cloudiness']
    data = df.drop(columns=columns)

    # Make dataset numeric
    data = data.astype('float32')

    # Missing Value
    df_temp = fill_missing(data)
    print(df_temp.isnull().sum())

    # Normalized data
    scaler = MinMaxScaler(feature_range=norm_range)
    data_scaled = scaler.fit_transform(df_temp.values)
    new_df = pd.DataFrame(data_scaled, columns=df_temp.columns, index=df_temp.index)

    return new_df, scaler, df_temp, columns

,,,
Creating a supervised problem
,,,

def to_supervised(df, timesteps, multisteps, nr_features):
    data = df.values
    X, y = list(), list()
    data_size = len(data)
```



```
for curr_pos in range(data_size):
    end_timesteps = curr_pos + timesteps
    begin_prev = end_timesteps
    end_prev = end_timesteps + 1
    if end_prev < data_size:
        X.append(data[curr_pos:end_timesteps,:])
        y.append(data[begin_prev:end_prev,5])

X = np.reshape(np.array(X), (len(X), timesteps, nr_features))
y = np.reshape(np.array(y), (len(y), 1))

return X, y

,,,
Tunning
,,,

def tunning(x_train, y_train, x_val, y_val, params):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                   return_sequences=True,
                                   input_shape=(x_train.shape[1],
                                               x_train.shape[2]),
                                   activation=params['activation']))

    model.add(tf.keras.layers.LSTM(params['first_neuron'],
                                   return_sequences=False,
                                   dropout=params['dropout'],
                                   activation=params['activation']))

    model.add(tf.keras.layers.Dense(params['first_neuron'],
                                   activation=params['activation']))
    model.add(tf.keras.layers.Dropout(params['dropout']))

    model.add(tf.keras.layers.Dense(1, activation=params['last_activation']))

    model.compile(
        loss=params['losses'],
        optimizer=params['optimizer'],
        metrics=['mae']
    )

    history = model.fit(x_train, y_train,
                         validation_data=[x_val, y_val],
                         batch_size=params['batch_size'],
                         epochs=params['epochs'],
                         verbose=0)

return history, model
```