Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de
Dados e Aprendizagem Automática

Ano Letivo de 2022/2023

Relatório
Conceção e otimização de modelos de Machine Learning

Grupo 01

Rita Gomes	pg50723
Joana Oliveira	pg50460
Pedro Araújo	pg50684

15 de Janeiro de 2023

Índice

1 Introdução

O presente relatório serve de suporte ao trabalho realizado no âmbito da unidade curricular de Dados e Aprendizagem Automática.

Para a elaboração deste trabalho foi utilizada a ferramenta Jupyter Notebook, que é uma plataforma de computação interativa baseada na web que permite aos utilizadores criar e partilhar códigos, equações, visualizações e texto. Jupyter é um acrónimo indireto de três linguagens de programação (Julia, Python e R). Neste caso em específico, o grupo utilizou a linguagem Python.

O objetivo deste trabalho consiste em analisar, explorar e procurar extrair conhecimento de dois datasets. O primeiro dataset ficou ao cargo do grupo e o segundo dataset foi-nos atribuído através de uma competição no link disponibilizado no enunciado.

Para a escolha do primeiro dataset, o grupo consultou várias fontes para encontrar um que fosse interessante. Após analisar vários datasets, o grupo optou por escolher um relacionado com os preços das casas no Dubai. Esta escolha baseou-se no facto de este ser um dataset que contém diferentes tipos de informação relevante para fazer uma análise completa dos dados. Sendo assim, o dataset que o grupo escolheu intitula-se de *properties data* e foi escolhido através da plataforma Kaggle.

O dataset que nos foi atribuído através da competição referida acima foi um dataset que contém dados referentes à quantidade e características dos incidentes rodoviários que ocorreram na cidade de Guimarães em 2021.

Neste documento vamos fazer uma análise detalhada do tratamento dos dados dos dois datasets.

2 Principais objetivos

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

Numa primeira fase, o grupo empenhou-se em realizar um bom tratamento de dados, procurando extrair conhecimento relevante no contexto dos problemas eme questão, para que consequentemente os modelos de Machine Learning desenvolvidos obtivessem uma boa perfomance. Para tal, recorremos a várias técnicas de tratamento de dados, tais como deteção e remoção de outliers, verificação de registos duplicados ou nulos (missing values), categorização de features e ainda extração/remoção de features.

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

De uma forma simplificada, para desenvolver este projeto de extração de conhecimento foi aplicada uma metodologia baseada em cinco diferentes fases:

- Compreensão do problema
- Análise e exploração dos dados
- Tratamento dos dados
- Modelo de extração do conhecimento
- Avaliação do modelo

3 Formalização dos problemas

3.1 Dataset Grupo (*properties data*)

Link Dataset Properties

3.1.1 Análise e exploração dos dados

A análise deste dataset teve como objetivo prever os preços dos imóveis do Dubai, trata-se de um problema adequado a utilizar um modelo supervisionado de regressão. Este dataset contém 38 colunas: id, neighborhood, latitude, longitude, price, size in sqft, price per sqft, no of bedrooms, no of bathrooms, quality, maid room, unfurnished, balcony, barbecue area, built in wardrobes, central ac, childrensplay area, childrens pool, concierge, covered parking, kitchen appliances, lobby in building, maid service, networked, pets allowed, private garden, private gym, private jacuzzi, private pool, security, shared gym,shared pool, shared spa, study, vastu compliant, view of landmark, view of water, walk in closet. É de salientar que a feature 'price' será a target feature.

Valores em falta

Como podemos observar pelas duas imagens abaixo, este dataset não tinha missing values.

```
# Check for missing values or duplicated data
print(df.isna().sum())
print(df.duplicated().sum())
```

```
Output exceeds the size limit. Open the full output data in a text editor
id                  0
neighborhood        0
latitude            0
longitude           0
price               0
size_in_sqft        0
price_per_sqft      0
no_of_bedrooms      0
no_of_bathrooms     0
quality             0
maid_room           0
```

Figura 3.1: Missing values

```
unfurnished            0
balcony                0
barbecue_area          0
built_in_wardrobes     0
central_ac             0
childrens_play_area    0
childrens_pool         0
concierge              0
covered_parking        0
kitchen_appliances     0
lobby_in_building      0
maid_service           0
networked              0
pets_allowed           0
...
view_of_water          0
walk_in_closet         0
```

Figura 3.2: Missing values

Tipos de dados dos atributos

Com recurso ao método **info()** foi possível identificar 3 colunas do tipo ***float*** ('latitude', 'longitude' e 'price_per_sqft'), 5 do tipo ***int*** ('id', 'price', 'size_in_sqft', 'no_of_bedrooms', 'no_of_bathrooms'), 2 do tipo ***object*** ('neighborhood' e 'quality') e as restantes(28) do tipo ***bool***.

```
Data columns (total 38 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   1905 non-null   int64
 1   neighborhood         1905 non-null   object
 2   latitude             1905 non-null   float64
 3   longitude            1905 non-null   float64
 4   price                1905 non-null   int64
 5   size_in_sqft         1905 non-null   int64
 6   price_per_sqft       1905 non-null   float64
 7   no_of_bedrooms       1905 non-null   int64
 8   no_of_bathrooms      1905 non-null   int64
 9   quality              1905 non-null   object
 10  maid_room            1905 non-null   bool
 11  unfurnished          1905 non-null   bool
 12  balcony              1905 non-null   bool
 13  barbecue_area        1905 non-null   bool
 14  built_in_wardrobes   1905 non-null   bool
 15  central_ac           1905 non-null   bool
 16  childrens_play_area  1905 non-null   bool
 17  childrens_pool       1905 non-null   bool
 18  concierge            1905 non-null   bool
 19  covered_parking      1905 non-null   bool
 20  kitchen_appliances   1905 non-null   bool
 21  lobby_in_building    1905 non-null   bool
 22  maid_service         1905 non-null   bool
 23  networked            1905 non-null   bool
 24  pets_allowed         1905 non-null   bool
 25  private_garden       1905 non-null   bool
 26  private_gym          1905 non-null   bool
 27  private_jacuzzi      1905 non-null   bool
 28  private_pool         1905 non-null   bool
 29  security             1905 non-null   bool
 30  shared_gym           1905 non-null   bool
 31  shared_pool          1905 non-null   bool
 32  shared_spa           1905 non-null   bool
 33  study                1905 non-null   bool
 34  vastu_compliant      1905 non-null   bool
 35  view_of_landmark     1905 non-null   bool
 36  view_of_water        1905 non-null   bool
 37  walk_in_closet       1905 non-null   bool
```

Figura 3.3: Tipos de dados dos atributos

Relações entre atributos

Figura 3.4: Relação entre a feature preço e vizinhança

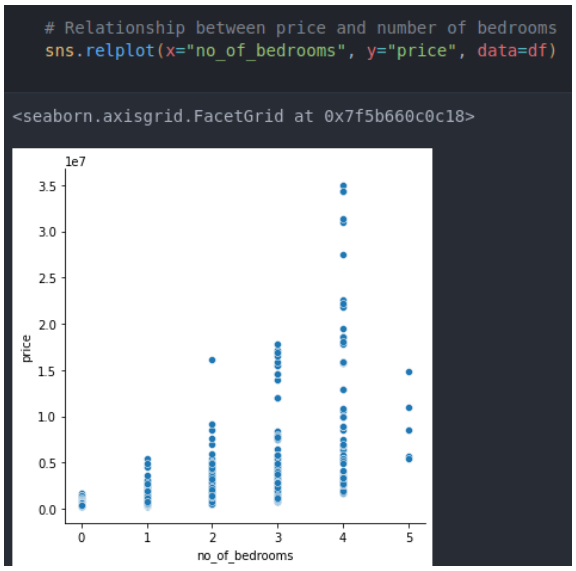

Figura 3.5: Relação entre a feature preço e número de quartos

Gerando o *catplot* para analisar a relação entre a feature 'price' e 'neighborhood', vemos que existem alguns outliers que devem ser tratados. Para além disso, vemos que a zona de Jumeirah, Palm Jumeirah e Culture Village são as vizinhanças onde o preço tem um valor mais elevado.

Quando geramos o *relplot* para analisar a relação da feature 'price' e 'no_of_bedrooms'

conseguimos perceber que a quantidade de quartos numa casa influencia muito o preço da casa. O mesmo acontece com o número de casas de banho.

Feature price

```
# See Stats of attribute Price
df['price'].describe()
```

```
count    1.905000e+03
mean     2.085830e+06
std      2.913200e+06
min      2.200000e+05
25%      8.900000e+05
50%      1.400000e+06
75%      2.200000e+06
max      3.500000e+07
Name: price, dtype: float64
```

Figura 3.6: Análise dos valores na feature price

```
# See how prices are distributed
sns.displot(df,x="price")
sns.displot(df,x="price",kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5b67630828>
```

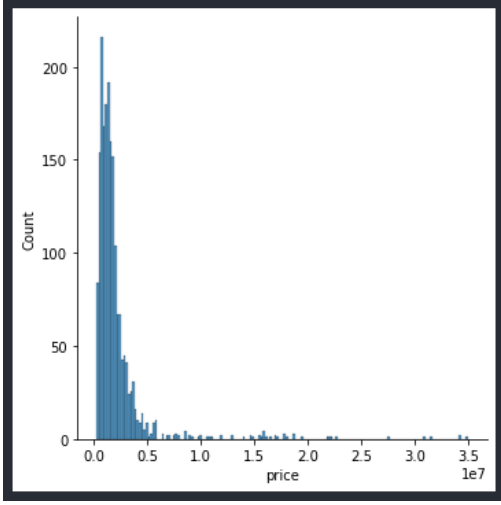

Figura 3.7: Distribuição dos valores na feature price

A partir das imagens acima, conseguimos perceber de que forma os valores na feature

'price' se distribuem. Vemos que o preço de uma casa no Dubai varia desde os 220 mil de euros até aos 35 milhões de euros, sendo a média de preços de 2 milhões de euros.

3.1.2 Tratamento e análise dos dados

Feita toda a análise que achámos necessária passámos ao tratamento dos dados.

Seleção de features

O grupo decidiu remover algumas colunas do dataset original de forma a otimizar a qualidade dos dados, essas são: id, maid room, unfurnished, balcony, barbecue area, built in wardrobes, central ac, childrensplay area, childrens pool, concierge, covered parking, kitchen appliances, lobby in building, maid service, networked, pets allowed, private garden, private gym, private jacuzzi, private pool, security, shared gym,shared pool, shared spa, study, vastu compliant, view of landmark, view of water, walk in closet. O grupo achou que a feature 'id' não era necessária para a análise do problema, uma vez que não representa qualquer tipo de informação relevante quanto ao dataset. Quanto às outras features, percebemos que estas poderiam ser substituídas apenas pela feature 'quality', uma vez que esta é calculada com base nos restantes atributos.

```
# Rename columns

# Drop columns
df.drop(['id'], axis=1,inplace=True)
df = df.iloc[:,:9]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1905 entries, 0 to 1904
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   neighborhood     1905 non-null   object
 1   latitude         1905 non-null   float64
 2   longitude        1905 non-null   float64
 3   price            1905 non-null   int64
 4   size_in_sqft     1905 non-null   int64
 5   price_per_sqft   1905 non-null   float64
 6   no_of_bedrooms   1905 non-null   int64
 7   no_of_bathrooms  1905 non-null   int64
 8   quality          1905 non-null   object
dtypes: float64(3), int64(4), object(2)
```

Figura 3.8: Seleção de features

Label Encoding

O Label Encoding é um método usado para converter variáveis categóricas em valores numéricos. A cada categoria atribui um valor inteiro exclusivo, tornando-a utilizável no modelo.

```
# Label Enconding in Neighborhood
label_encoder = preprocessing.LabelEncoder()
df['neighborhood']= label_encoder.fit_transform(df['neighborhood'])
```

```
df['neighborhood'].unique()
```

```
array([46, 36, 11, 15, 22,  9, 45,  3, 39,  6, 34, 16, 30, 10,  2, 12, 35,
       33, 20, 51, 43, 13, 37,  7,  0, 23, 48, 49, 27, 25, 38, 42, 18,  8,
       31, 28, 40, 50, 29,  1,  5, 17, 32, 21, 52, 44, 14, 47, 41, 26, 19,
       53, 24,  4])
```

Figura 3.9: Label Encoding na feature neighborhood

O mesmo processo foi aplicado à feature 'quality'.

Normalização de valores

Devido aos diferentes intervalos existentes entre os diferentes atributos do dataset e de modo a minimizar o impacto que essas diferenças poderiam provocar na análise dos dados, normalizamos os valores de diversas colunas para um intervalo entre 0 e 1. Na imagem seguinte é possível observar qual a estratégia seguida, utilizando como exemplo a coluna 'neighborhood'.

```
#Normalize Neighboorhood values
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
df[['neighborhood']] = min_max_scaler.fit_transform(df[['neighborhood']])
df['neighborhood'].describe()
```

Figura 3.10: Normalização dos valores da feature neighborhood

O mesmo processo demonstrado em cima foi aplicado às features 'size_in_sqft', 'price_per_sqft', 'no_of_bedrooms', 'no_of_bathrooms'.

Tratamento dos outliers

```
# Calculate IQR (Inter Quartile Range)
Q1 = np.percentile(df['price'], 25, interpolation = 'midpoint')
Q3 = np.percentile(df['price'], 75, interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Dimensions: ", df.shape)

# Define upper bound
upper = np.where(df['price'] >= (Q3+1.5*IQR))
# Define lower bound
lower = np.where(df['price'] <= (Q1-1.5*IQR))

#Remove outliers
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Dimensions: ", df.shape)
```

```
Old Dimensions:  (1905, 33)
New Dimensions:  (1777, 33)
```

Figura 3.11: Intervalo interquartil (IQR)

Para tratar os outliers, o grupo decidiu usar a estratégia dos intervalos interquartis. Assim sendo, qualquer entrada com um valor na coluna 'price' fora de um intervalo definido foi removido.

3.1.3 Modelos desenvolvidos

Após a análise detalhada dos dados, bem como o seu tratamento, passamos à fase de desenvolvimento do modelo de previsão. Esta fase pretende obter um modelo que seja capaz de representar o dataset em questão, bem como fazer previsões corretas a partir de dados desconhecidos. Assim sendo, o grupo começou por efetuar a divisão dos dados em dados de treino e dados de teste, recorrendo-se ao ***train_test_split***. Decidiu-se que 25% do dataset seria usado para teste e 75% para treino.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=RANDOM_SEED)
```

Figura 3.12: Divisão do dataset em dados de treino e teste

Apresentamos de seguida, os vários modelos adotados e a avaliação do desempenho de cada um dos modelos.

Linear Regression

A regressão linear é um método usado para modelar a relação entre uma variável dependente (price) e uma ou mais variáveis independentes. O objetivo da regressão linear é encontrar a linha reta mais adequada através dos pontos de dados. A linha de melhor ajuste é aquela que minimiza a soma das diferenças ao quadrado entre os valores previstos e os valores reais.

```
lm = LinearRegression()
lm.fit(x_train,y_train)
predictions = lm.predict(x_test)
```

Figura 3.13: Linear Regression

```
plt.scatter(y_test, predictions)
```

```
<matplotlib.collections.PathCollection at 0x
```

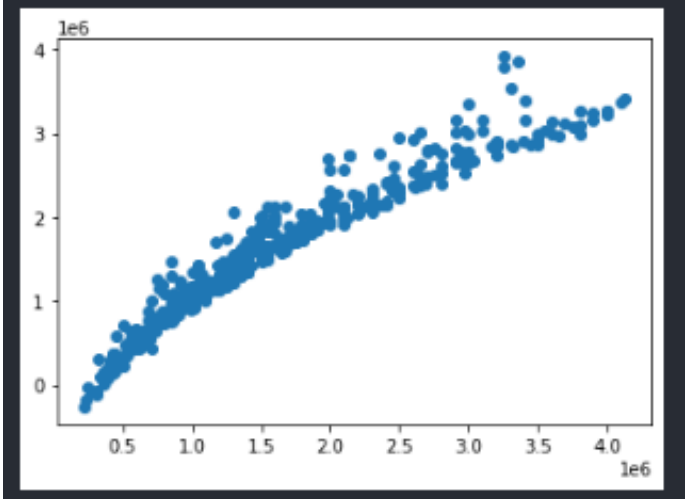

Figura 3.14: Previsões com Linear Regression

Decision Tree Regression

Este algoritmo funciona dividindo recursivamente o conjunto de dados em subconjuntos, com base nos valores dos recursos de entrada. A cada divisão o algoritmo escolhe o recurso e o ponto de divisão que resultam na maior redução de impurezas. A saída final da árvore de decisão é um conjunto de nós folha, cada um representando um valor para a variável de destino.

```
dtr = DecisionTreeRegressor(random_state=RANDOM_SEED)
dtr.fit(x_train, np.ravel(y_train))
```

Figura 3.15: Decision Tree Regressor

```
predictionsDT = dtr.predict(x_test)
predictionsDT
```

Figura 3.16: Predictions in Decision Tree Regressor

Neste caso, a validação cruzada foi usada para avaliar o desempenho do modelo dividindo o conjunto de dados em conjuntos de treino e validação.

```
#Cross Validation
scores = cross_val_score(dtr,x,y,cv=10)
print(scores)
print("RESULT: %0.2f accuracy with standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
[0.97906043 0.97759194 0.98372432 0.98236922 0.99048409 0.96578953
 0.94419076 0.98575658 0.98946377 0.98908868]
RESULT: 0.98 accuracy with standard deviation of 0.01
```

Figura 3.17: Cross Validation

MultiLayer Perceptron (MLP)

Um Multilayer Perceptron (MLP) é um tipo de rede neural artificial (ANN) composta por várias camadas de "neurónios"interligados, usados para modelar relações complexas entre dados de entrada e saída. Um MLP é uma rede neuronal feedforward, o que significa que os dados fluem numa direção através da rede, da entrada para a saída, sem formar um loop.

Os MLPs são treinados usando o algoritmo backpropagation, que atualiza os pesos da rede para minimizar o erro entre a saída prevista e a saída real.

```
#Scale the features between [0-1]
scaler_x = MinMaxScaler(feature_range=(0,1)).fit(x)
scaler_y = MinMaxScaler(feature_range=(0,1)).fit(y)
x_scaled = pd.DataFrame(scaler_x.transform(x[x.columns]), columns=x.columns)
y_scaled = pd.DataFrame(scaler_y.transform(y[y.columns]), columns=y.columns)
```

Figura 3.18: Normalização das features para o intervalo [0,1]

```python
#em vez do relu usar o softmax - sugestao da prof
#modelo MLP com 7 camadas
def build_model(activation='relu', learning_rate=0.01):
    model=Sequential()
    model.add(Dense(16,input_dim=8,activation=activation))
    model.add(Dense(8,activation=activation))
    model.add(Dense(4,activation=activation))
    model.add(Dense(2,activation=activation))
    model.add(Dense(1,activation='relu'))

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

Figura 3.19: Modelo MLP com 5 camadas

```python
model = build_model()
model.summary()
```

```
Model: "sequential_375"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2309 (Dense)           (None, 16)                144
_________________________________________________________________
dense_2310 (Dense)           (None, 8)                 136
_________________________________________________________________
dense_2311 (Dense)           (None, 4)                 36
_________________________________________________________________
dense_2312 (Dense)           (None, 2)                 10
_________________________________________________________________
dense_2313 (Dense)           (None, 1)                 3
=================================================================
Total params: 329
Trainable params: 329
Non-trainable params: 0
```

Figura 3.20: Resultado da construçao do modelo MLP

```python
#encontrar o melhor MLP possivel
TURNING_DICT = {
    'activation': ['relu','sigmoid'],
    'learning_rate':[0.01,0.001]
}
```

Figura 3.21: Encontrar o melhor MLP possível

A técnica de Grid Search é uma técnica usada para ajustar os hiperparâmetros de um modelo de machine learning. Esta funciona especificando um intervalo de valores para cada hiperparâmetro e, em seguida, treinando o modelo usando todas as combinações

possíveis desses valores. O desempenho do modelo é então avaliado usando uma métrica de desempenho, neste caso, a média.

```
kf = KFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)

model = KerasRegressor(build_fn=build_model, epochs=20, batch_size=32)
grid_search = GridSearchCV(estimator=model, param_grid=TURNING_DICT, cv=kf, scoring='neg_mean_absolute_error', refit='True', verbose=1)
grid_search.fit(x_train, y_train, validation_split=0.2)
```

Figura 3.22: Grid Search aplicado no modelo MLP

```
print("Best: %f using %s" % (grid_search.best_score_,grid_search.best_params_))
means = grid_search.cv_results_['mean_test_score']
stds = grid_search.cv_results_['std_test_score']
params = grid_search.cv_results_['params']
for mean,stdev,param in zip(means,stds,params):
    print("%f (%f) with: %r" % (mean,stdev,param))
```

```
Best: -0.147123 using {'activation': 'relu', 'learning_rate': 0.01}
-0.147123 (0.145926) with: {'activation': 'relu', 'learning_rate': 0.01}
-0.214952 (0.140457) with: {'activation': 'relu', 'learning_rate': 0.001}
-0.270927 (0.117853) with: {'activation': 'sigmoid', 'learning_rate': 0.01}
-0.194213 (0.064252) with: {'activation': 'sigmoid', 'learning_rate': 0.001}
```

Figura 3.23: Encontrar o melhor resultado

```
best_mlp_model = grid_search.best_estimator_
best_mlp_model.fit(x_train,y_train,epochs=20,validation_data=(x_test,y_test),callbacks=[PlotLossesKerasTF()],verbose=1)
```

Figura 3.24: Resultado

De forma a perceber se modelo estava overfitted o grupo decidiu usar o ***livelossplot*** da biblioteca *PlotLossesKerasTF*.

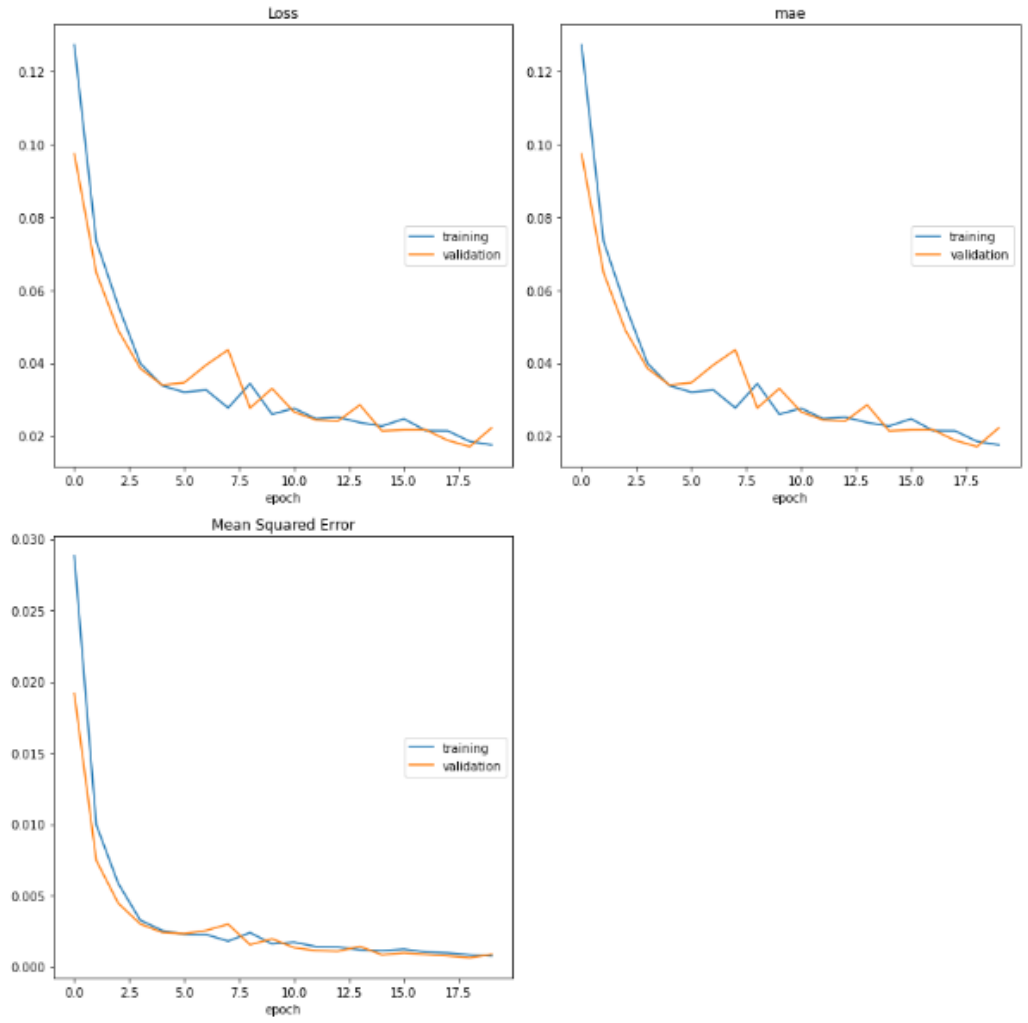

Figura 3.25: Resultado das predictions do modelo MLP

Da imagem acima, conseguimos perceber que o modelo náo está overfitted.

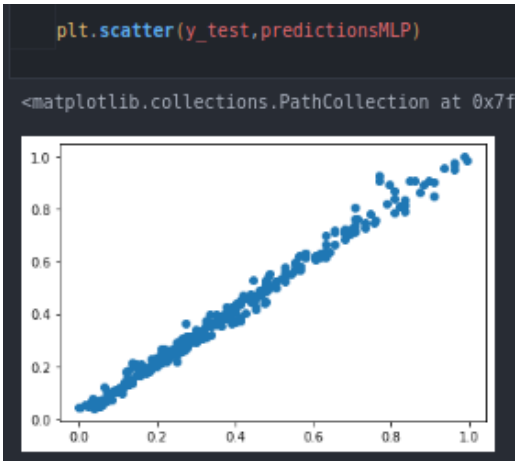

Figura 3.26: Diagrama de dispersão

Figura 3.27: Real vs Predicted

Das imagens acima, conseguimos perceber que os valores previstos estão muito próximos dos valores reais, pelo que foi feita uma boa análise da previsão dos preços das casas do Dubai.

3.1.4 Análise dos resultados

Tal como já foi referido, foram testados vários modelos de forma a determinar aquele que produziria melhores resultados. Após se ter procedido às respetivas análises dos vários modelos implementados, determinou-se que o algoritmo **Decision Tree Regression**

obteve os resultados mais promissores, como podemos verificar na imagem a seguir.

```
# Compare model results
print("MLP results\n")
print('MAE:',metrics.mean_absolute_error(y_test_unscaled,predictionsMLP_unscaled))
print('MSE:',metrics.mean_squared_error(y_test_unscaled,predictionsMLP_unscaled))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test_unscaled,predictionsMLP_unscaled)))
print('------------------------------------------------------------')
print('DecisionTreeRegressor results:\n')
print('MAE:',metrics.mean_absolute_error(y_test1,predictionsDT))
print('MSE:',metrics.mean_squared_error(y_test1, predictionsDT, squared=True))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test1, predictionsDT, squared=False)))
print('------------------------------------------------------------')
print('LinearRegressor results:\n')
print('MAE:',metrics.mean_absolute_error(y_test1,predictionsLR))
print('MSE:',metrics.mean_squared_error(y_test1, predictionsLR))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test1, predictionsLR)))
```

```
MLP results

MAE: 87422.87210323036
MSE: 13523977150.714931
RMSE: 116292.63584043029
------------------------------------------------------------
DecisionTreeRegressor results:

MAE: 68293.60898876404
MSE: 14619723784.026966
RMSE: 347.7241000971561
------------------------------------------------------------
LinearRegressor results:

MAE: 178092.749069489
MSE: 62995077155.87196
RMSE: 250988.20122840826
```

Figura 3.28: Comparação dos vários modelos

3.2 Dataset Competição

Retirado de: **Link Dataset Competição**

3.2.1 Análise e exploração dos dados

A análise deste dataset passa por explorar dados referentes à quantidade e características dos incidentes rodoviários que ocorreram na cidade de Guimarães em 2021. Este dataset foi disponibilizado aos alunos, divido em 2 datasets: um dataset de aprendizagem (treino) e um dataset de teste. o dataset contém 13 colunas: city_name, magnitude_of_delay, delay_in_seconds, affected_roads, record_date, luminosity, avg_temperature, avg_atm_pressure, avg_humidity, avg_wind_speed, avg_precipitation, avg_rain, incidents.

O dataset de aprendizagem foi utilizado para desenvolver e treinar o modelo de Machine Learning. Neste dataset é fornecida informação referente à quantidade de incidentes rodoviários para cada registo (incidents). O modelo a desenvolver teve, na sua base, features como a magnitude do atraso que se verifica numa determinada hora, o tempo de atraso provocado pelos incidentes, a temperatura, pressão atmosférica e a velocidade do vento, entre outras features que caracterizam um determinado ponto temporal.

Por sua vez, o dataset de teste foi utilizado para validar a accuracy do modelo em dados ainda não vistos pelo mesmo. Neste dataset é necessário prever a class/target/label referente à quantidade de incidentes rodoviários. Para isso utilizamos o modelo desenvolvido para prever, para cada registo do dataset de teste, o nível de incidentes rodoviários esperados.

Valores em falta

```
#Check duplicates
db.duplicated().sum()
```

```
0
```

Figura 3.29: Valores duplicados

```
city_name              0
magnitude_of_delay     0
delay_in_seconds       0
affected_roads        85
record_date            0
luminosity             0
avg_temperature        0
avg_atm_pressure       0
avg_humidity           0
avg_wind_speed         0
avg_precipitation      0
avg_rain               0
incidents              0
dtype: int64
```

Figura 3.30: Missing values

Como podemos observar pela análise da imagem acima existem 85 valores nulos no dataset de treino, na feature 'affected_roads', e cujo tratamento será falado mais tarde.

Tipos de dados dos atributos

```
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   city_name           5000 non-null   object 
 1   magnitude_of_delay  5000 non-null   object 
 2   delay_in_seconds    5000 non-null   int64  
 3   affected_roads      4915 non-null   object 
 4   record_date         5000 non-null   object 
 5   luminosity          5000 non-null   object 
 6   avg_temperature     5000 non-null   float64
 7   avg_atm_pressure    5000 non-null   float64
 8   avg_humidity        5000 non-null   float64
 9   avg_wind_speed      5000 non-null   float64
 10  avg_precipitation   5000 non-null   float64
 11  avg_rain            5000 non-null   object 
 12  incidents           5000 non-null   object 
dtypes: float64(5), int64(1), object(7)
```

Figura 3.31: Tipos de dados

```
# See number of different values of each attribute
db.nunique()
```

```
city_name               1
magnitude_of_delay      3
delay_in_seconds     1186
affected_roads        678
record_date          5000
luminosity              3
avg_temperature        35
avg_atm_pressure       36
avg_humidity           83
avg_wind_speed         11
avg_precipitation       1
avg_rain                4
incidents               5
dtype: int64
```

Figura 3.32: Análise dos diferentes valores para cada atributo

Com recurso ao método **info()** foi possível identificar 5 colunas com o tipo ***float*** (avg_temperature, avg_atm_pressure, avg_humidity, avg_wind_speed, avg_precipitation), 1 coluna com o tipo ***int*** (delay_in_seconds) e as restantes colunas (7) com o tipo ***object***.
Com recurso ao método **nunique()** foi possível observer quantos valores diferentes para cada atributo temos presente no dataset.

Distribuição dos dados dos atributos

Os gráficos seguintes permitem-nos perceber como estão distríbuidos os diferentes valores de cada atributo, ou seja, que intervalos de valores existem e em que quantidade existe cada um desses valores.

Figura 3.33: Distribuição do nível de incidentes

Figura 3.34: Distribuição do nível de luminosidade

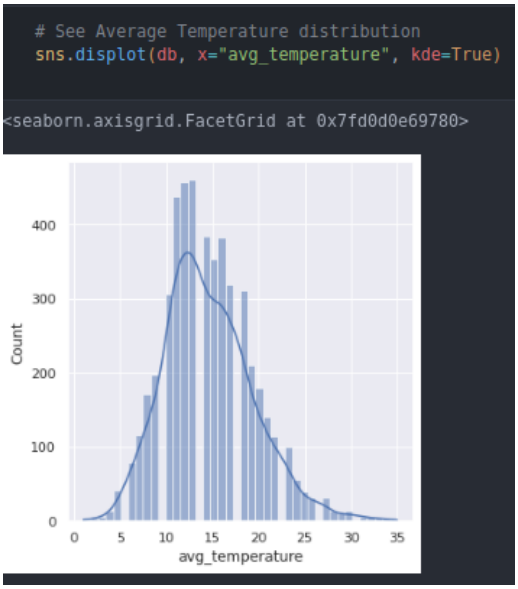

Figura 3.35: Distribuição da temperatura

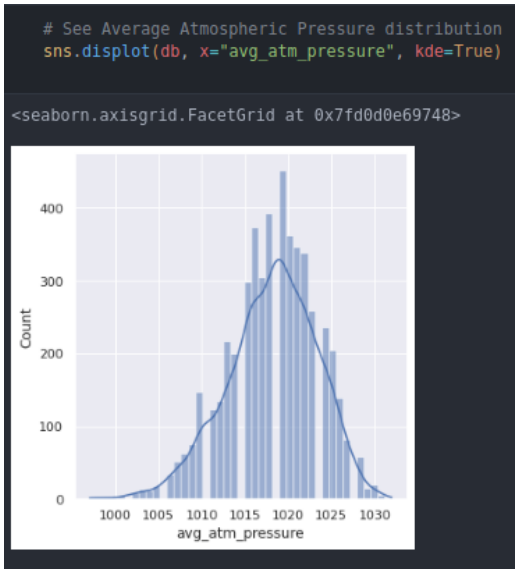

Figura 3.36: Distribuição do nível de pressão atmosférica

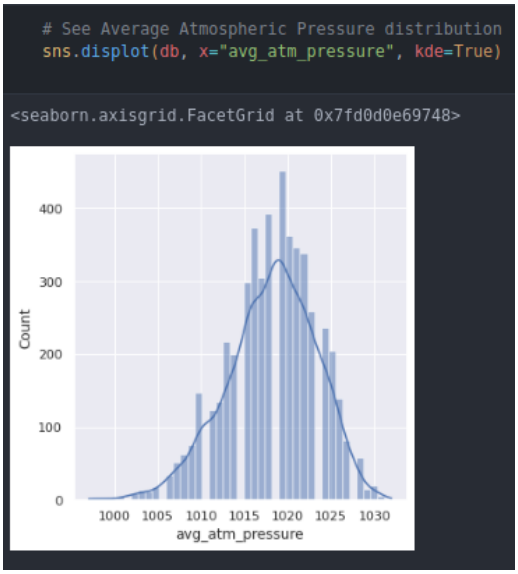

Figura 3.37: Distribuição do nível de pressão atmosférica

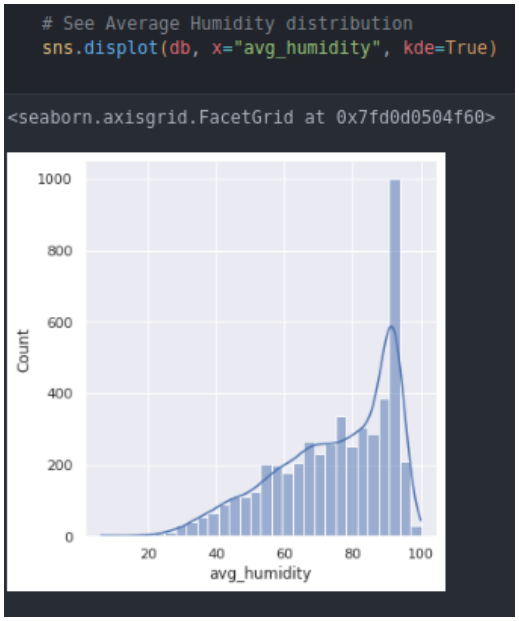

Figura 3.38: Distribuição da humidade média

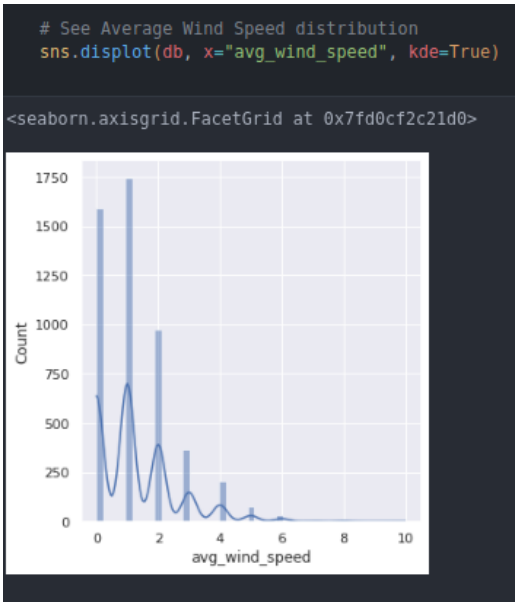

Figura 3.39: Distribuição da velocidade média do vento

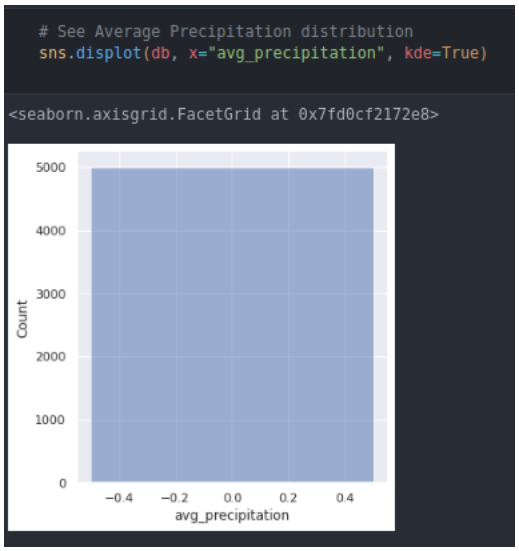

Figura 3.40: Distribuição da precipitação média

Relações entre atributos

Através da análise de gráficos *catplot* conseguimos perceber como se relacionam os diferentes atributos uns com os outros. Isto permitiu-nos, por exemplo, verificar a existência de outliers em várias features.

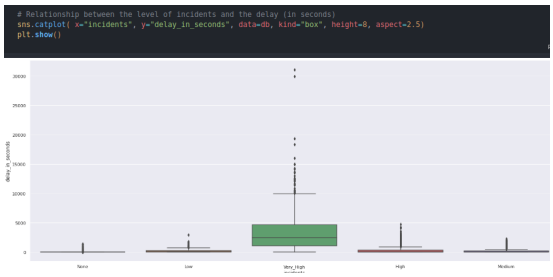

Figura 3.41: Relação entre o nível de incidents e o delay

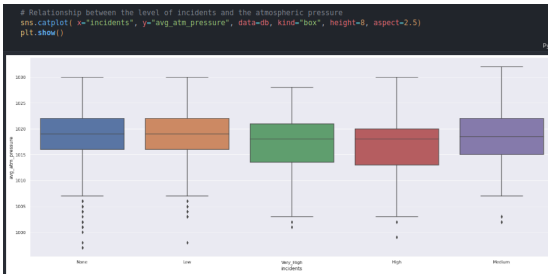

Figura 3.42: Relação entre o nível de incidents e a pressão atmosférica

Figura 3.43: Relação entre o nível de incidents e a humidade

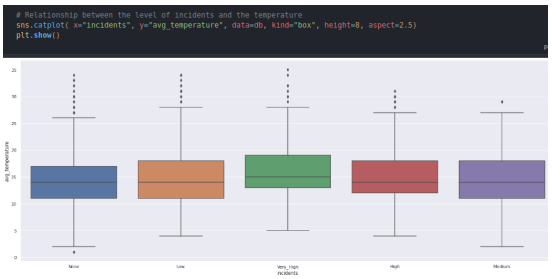

Figura 3.44: Relação entre o nível de incidents e a temperatura

Matriz de correlação

Figura 3.45: Matriz de correlação

Gerando uma **matriz de correlação**, foi possível identificar quais atributos se relacionam mais entre si. Neste caso, vemos que as colunas 'avg_humidity' e 'avg_temperature' têm um coeficiente negativo alto em termos de correlação. Isto significa que, quando uma variável aumenta, a outra tende a diminuir e vice-versa. Os restantes valores como têm um coeficiente de correlação próximo de 0, indicam uma correlação fraca ou nula.

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

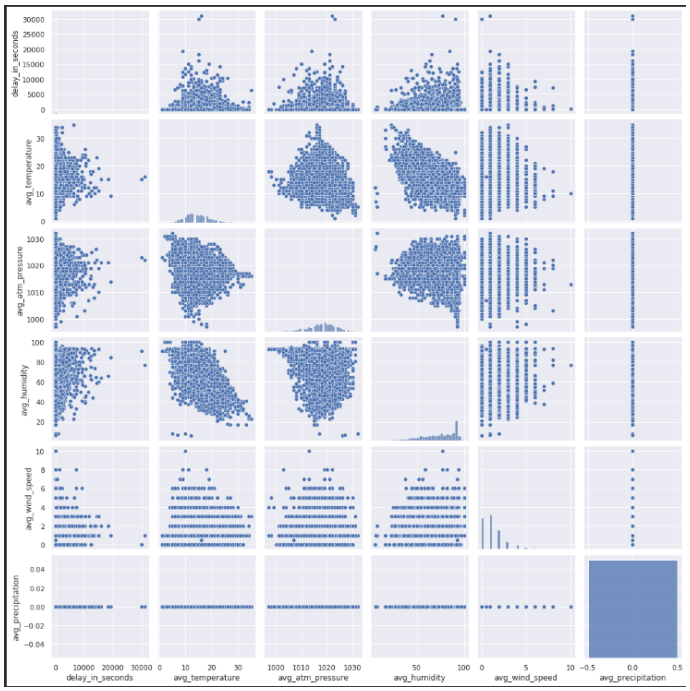

Figura 3.46: Relação entre todos os atributos

3.2.2 Tratamento dos dados

Finda toda a análise que achámos necessária passámos ao tratamento dos dados.

Tratamento dos outliers

Para fazer o tratamento e remoção dos outliers, o grupo optou por usar o conhecido método**z-score**. Este método é usado para comparar um valor específico a uma distribuição de dados e determinar se esse valor é anormal (outlier) ou não. É possível observar na imagem abaixo qual a estratégia adotada usando como exemplo o atributo 'avg_temperature'.

```
# Calculate IQR (Inter Quartile Range)
print("Old Dimensions: ", db.shape)
db.loc[np.abs(stats.zscore(db['avg_temperature'])) >= 3, 'avg_temperature'] = np.nan
db['avg_temperature'].fillna(db['avg_temperature'].median(),inplace=True)
#db = db[np.abs(stats.zscore(db['avg_temperature'])) < 3]
print("New Dimensions: ", db.shape)
```

Figura 3.47: Remoção de outliers na feature avg_temperature

A mesma estratégia foi aplicada às features avg_atm_pressure, avg_humidity e delay_in_seconds.

Seleção de features

O grupo decidiu remover 3 colunas (avg_precipitation, magnitude_of_delay, city_name). Esta decisão tomada pelo grupo recai sobre o facto de, tanto o atributo 'city_name' como o 'avg_precipitation' conterem apenas um único valor, tal como foi demonstrado na análise e exploração de dados feita acima, com o método nunique(). Para além disso, sabemos desde o início que o dataset é relativo apenas à cidade de Guimarães, pelo que não faz sentido ter essa referência repetida nas várias entradas do dataset.

```
#Drop columns in Training Data
db.drop(['city_name'],axis=1,inplace=True)
db.drop(['avg_precipitation'],axis=1,inplace=True)
db.drop(['magnitude_of_delay'],axis=1,inplace=True)
db.dropna(inplace=True)
```

Figura 3.48: Remoção de features

Label Encoding

Tal como vimos anteriormente, existem várias features com o tipo *object*, representadas por valores categóricos, pelo que o grupo optou por transformá-las em valores numéricos.

Para fazer isto, foi utilizada a estratégia de *Label Encoding*, em que a cada categoria é atribuída a um valor numérico único.

```
# Label Enconding: Incidents
replace_map = {'None':0,'Low':1,'Medium':2, 'High':3, 'Very_High':4}
db['incidents'] = db['incidents'].replace(replace_map)
```

Figura 3.49: Label Encoding na feature Incidents

```
# Label Enconding: Luminosity
replace_map_lum = {'DARK': 0, 'LOW_LIGHT': 1, 'LIGHT': 1}
db['luminosity'] = db['luminosity'].replace(replace_map_lum)
test['luminosity'] = test['luminosity'].replace(replace_map_lum)
```

Figura 3.50: Label Encoding na feature Luminosity

```
# Label Enconding: Average Rain
replace_map_rain = {'Sem Chuva': 0, 'chuva fraca': 1, 'chuva moderada': 2, 'chuva forte': 3}
db['avg_rain'] = db['avg_rain'].replace(replace_map_rain)
test['avg_rain'] = test['avg_rain'].replace(replace_map_rain)
```

Figura 3.51: Label Encoding na feature average_rain

Tratamento das datas

Uma vez que as datas (coluna 'record date') estavam presentes no dataset como sendo strings, foi necessário, primeiro de tudo, transformá-las num formato adequado.

```
db['record_date'] = pd.to_datetime(db['record_date'],format='%Y-%m-%d %H:%M',errors='coerce')
test['record_date'] = pd.to_datetime(test['record_date'],format='%Y-%m-%d %H:%M',errors='coerce')
```

Figura 3.52: Alteração do formato das datas

No entanto, foi ainda necessária a sua divisão em ano, mês, dia, hora e minuto, de maneira a obtermos valores númericos. Para tal, foram então criadas novas colunas com os valores correspondentes a cada data.

```
#Create new columns from 'record date'
db['year'] = db['record_date'].dt.year
db['month'] = db['record_date'].dt.month
db['day'] = db['record_date'].dt.day
db['hour'] = db['record_date'].dt.hour
db['minute'] = db['record_date'].dt.minute
```

Figura 3.53: Criação de novas colunas

Uma vez que tanto a coluna relativa ao ano, como a relativa ao minuto, têm apenas um único valor, estas foram removidas.

```
    # Unique values
    print("Unique Values")
    print("Year: " + str(db['year'].nunique()))
    print("Month: " + str(db['month'].nunique()))
    print("Day: " + str(db['day'].nunique()))
    print("Hour: " + str(db['hour'].nunique()))
    print("Minute: " + str(db['minute'].nunique()))
```

```
Unique Values
Year: 1
Month: 11
Day: 31
Hour: 24
Minute: 1
```

Figura 3.54: Valores únicos de cada coluna

```
db.drop('year',axis=1,inplace=True)
db.drop('minute',axis=1,inplace=True)
db.drop('record_date',axis=1,inplace=True)
```

Figura 3.55: Remoção das colunas

Normalização de valores

De forma a tentar melhorar o desempenho dos algoritmos de Machine Learning, o grupo decidiu fazer uso da normalização de dados, recorrendo à estratégia Min-Max, na qual os valores das variáveis são transformados para um intervalo entre 0 e 1, de modo que o valor mínimo da variável seja transformado em 0 e o valor máximo seja transformado em 1.

```
#Normalize Delay in Seconds values
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
db[['delay_in_seconds']] = min_max_scaler.fit_transform(db[['delay_in_seconds']])
test[['delay_in_seconds']] = min_max_scaler.fit_transform(test[['delay_in_seconds']])
```

Figura 3.56: Normalização dos valores da feature delay_in_seconds

A mesma estratégia foi aplicada às seguintes features: luminosity, avg_temperature, avg_atm_pressure, avg_humidity, avg_wind_speed, avg_rain e number_of_affected_roads.

3.2.3 Modelos desenvolvidos

De seguida apresentamos os modelos que implementamos para o dataset da competição.

Decision Tree Classifier

Um Decision Tree Classifier é um tipo de algoritmo de machine learning supervisionado usado para dados de classificação. É um tipo de árvore de decisão em que a variável de destino é categórica. O algoritmo funciona dividindo recursivamente o conjunto de dados em subconjuntos, com base nos valores dos recursos de entrada. A cada divisão, o algoritmo escolhe o recurso e o ponto de divisão que resultam na maior redução de impurezas. A saída final da árvore de decisão é um conjunto de nós folha, cada um representando um rótulo de classe.

```
x = db.drop(['incidents'], axis=1)
y = db['incidents'].to_frame()
```

Figura 3.57: Escolha da variavel dependente

A variável dependente escolhida foi o número de incidents.

```
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=0.25, random_state=RANDOM_SEED)
print("X %s; X_train %s; X_test %s" %(x.shape,X_train.shape,X_test.shape))
print("y %s; y_train %s; y_test %s" %(y.shape,y_train.shape,y_test.shape))
```

Figura 3.58: Divisão do dataset

```
clf = DecisionTreeClassifier(random_state=RANDOM_SEED)
clf.fit(X_train,np.ravel(y_train))
```

Figura 3.59: Aplicação do modelo de Decision Tree Classifier

```
pred_dtc = clf.predict(X_test)

accuracy_score(y_test,pred_dtc)

0.9032

precision_score(y_test,pred_dtc,average='micro')

0.9032

recall_score(y_test,pred_dtc,average='micro')

0.9032
```

Figura 3.60: Resultados obtidos com a aplicação do modelo Decision Tree Classifier

```
cm = confusion_matrix(y_test, pred_dtc)
cmd = ConfusionMatrixDisplay(cm, display_labels=['None', 'Low', 'Medium', 'High', 'Very_High'])
cmd.plot()

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f347cc942e8>
```

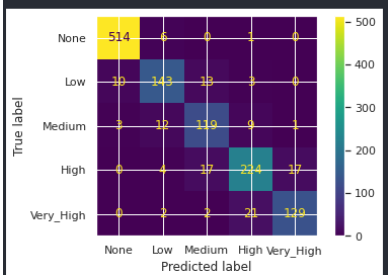

Figura 3.61: Matriz de confusão

A validação cruzada e hold-out são duas técnicas usadas para avaliar o desempenho de um modelo de machine learning e evitar o overfitting. A cross validation é uma técnica que divide os dados em vários subconjuntos chamados "dobras"e treina o modelo em diferentes subconjuntos de dados, enquanto os avalia nos subconjuntos restantes. A hold-out validation, por outro lado, é uma técnica em que os dados são divididos aleatoriamente em dois subconjuntos: um conjunto de treino e um conjunto de validação. O modelo é treinado no conjunto de treino e depois avaliado no conjunto de validação. Este método usa apenas um único conjunto de validação.

```
cross_val_model = SVC(random_state=RANDOM_SEED)
scores = cross_val_score(cross_val_model,x,np.ravel(y),cv=10)
scores
```

```
array([0.438, 0.438, 0.412, 0.43 , 0.42 , 0.412, 0.43 , 0.424, 0.42 ,
       0.436])
```

```
print("RESULT: %0.2f accuracy with a standard deviation of %0.2f \n" % (scores.mean(), scores.std()))
```

```
RESULT: 0.43 accuracy with a standard deviation of 0.01
```

Figura 3.62: Cross Validation

```
model = SVC(random_state=RANDOM_SEED)
model.fit(X_train,np.ravel(y_train))
```

```
SVC(random_state=2022)
```

```
predictions = model.predict(X_test)
```

```
print("%0.2f accuracy" % (accuracy_score(y_test, predictions)))
```

```
0.43 accuracy
```

```
plot_confusion_matrix(model, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x
```

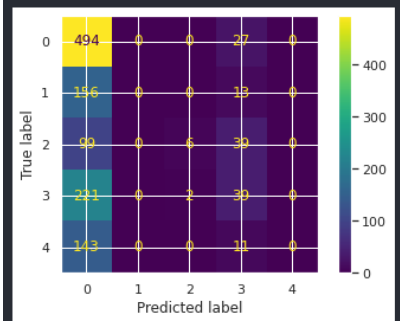

Figura 3.63: Hold-out Validation

Grid Search

A Grid Search é uma técnica usada para ajustar os hiperparâmetros de um modelo de machine learning treinando o modelo com diferentes combinações dos hiperparâmetros e avaliando seu desempenho. Hiperparâmetros são parâmetros que não são aprendidos com os dados durante o treino, mas são definidos antes do treino. Por exemplo, numa árvore de decisão de classificação, a profundidade máxima da árvore, o número mínimo

de amostras necessárias num nó folha e o critério usado para dividir os nós são alguns dos hiperparâmetros.

Esta técnica funciona especificando um intervalo de valores para cada hiperparâmetro e, em seguida, treinando o modelo usando todas as combinações possíveis desses valores. O desempenho do modelo é então avaliado usando uma métrica de desempenho, como por exemplo, a precisão. O conjunto de hiperparâmetros que resulta no melhor desempenho é então escolhido como o conjunto final de hiperparâmetros para o modelo.

```
param_grid = {'C' : [0.1,1,10,100,1000], 'gamma': [1,0.1,0.01,0.001,0.0001]}
grid = GridSearchCV(SVC(random_state=RANDOM_SEED),param_grid,refit=True,verbose=3)


grid.fit(X_train, np.ravel(y_train))
```

Figura 3.64: Aplicação do modelo Grid Search

```
grid_predicitions = grid.predict(X_test)


print(classification_report(y_test, grid_predicitions))
```

	precision	recall	f1-score	support
0	0.90	0.95	0.92	521
1	0.70	0.63	0.67	169
2	0.71	0.68	0.70	144
3	0.83	0.82	0.83	262
4	0.85	0.82	0.84	154
accuracy			0.83	1250
macro avg	0.80	0.78	0.79	1250
weighted avg	0.83	0.83	0.83	1250

Figura 3.65: Resultados da aplicação do modelo Grid Search

K-means Clustering

K-means é um tipo de técnica de machine learning não supervisionada usada para clustering. É um método usado para agrupar pontos de dados semelhantes em k clusters, onde k é um parâmetro especificado pelo utilizador. O objetivo do k-means é particionar um conjunto de n observações em k clusters, onde cada observação pertence ao cluster com a média mais próxima, servindo como um protótipo do cluster.

O algoritmo k-means começa selecionando aleatoriamente k pontos de dados do conjunto de dados como centroides iniciais ou centros de cluster. Em seguida, ele executa iterativamente as seguintes etapas:

- Atribui cada ponto de dados ao cluster cujo centróide está mais próximo dele.
- Recalcula o centróide de cada cluster através da média de todos os pontos de dados no cluster.
- Repete as etapas 1 e 2 até que as atribuições de cluster não sejam mais alteradas ou um número máximo de iterações seja atingido.

Figura 3.66: Aplicação do modelo K-means Clustering

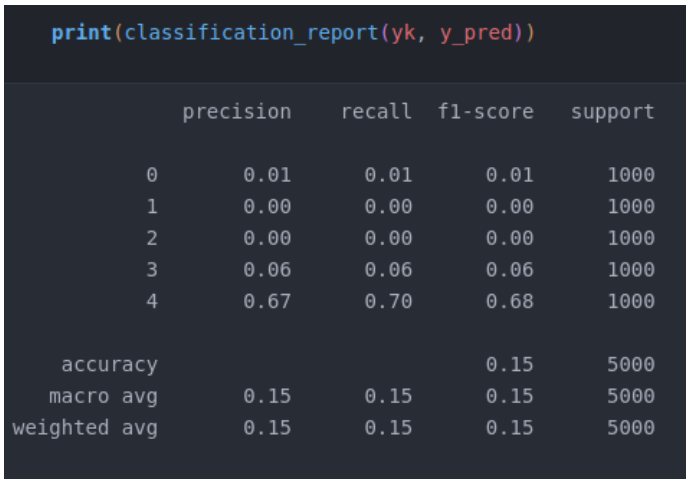

Figura 3.67: Resultados da aplicação do modelo K-means Clustering

Random Forest

Random Forest é um método de aprendizagem conjunta para classificação e regressão. É uma coleção de árvores de decisão, onde cada árvore é treinada num subconjunto aleatório (random) dos dados. A saída final do Random Forest é a média (para regressão) ou voto dominante (para classificação) das saídas de todas as árvores de decisão individuais.

A Random Forest usa uma técnica chamada bagging, para reduzir a variância do modelo calculando a média das previsões de várias árvores de decisão. Ele também usa uma

técnica chamada seleção aleatória de recursos para reduzir a correlação entre as árvores de decisão, o que, por sua vez, reduz ainda mais a variação do modelo.

Ao treinar uma Random Forest, os dados são divididos em subconjuntos e cada árvore de decisão é treinada num subconjunto diferente dos dados. Isso significa que cada árvore fará previsões com base num subconjunto diferente de recursos e um subconjunto diferente de pontos de dados. Ao calcular a média das previsões de todas as árvores, o modelo reduz a variância causada pelo superajuste de árvores de decisão individuais.

```
rfc = RandomForestClassifier()
rfc.fit(X_train,np.ravel(y_train))
```

Figura 3.68: Aplicação do modelo Random Forest

Figura 3.69: Escolha de parâmetros

```
print (f'Train {rf_RandomGrid.score (X_train, y_train):.3f}')
print(f'Test {rf_RandomGrid.score (X_test, y_test):.3f}')

Train 0.999
Test 0.921
```

Figura 3.70: Divisão do dataset em treino e teste

```
from sklearn.model_selection import RandomizedSearchCV

rf_RandomGrid = RandomizedSearchCV(estimator = rfc, param_distributions = param_grid, cv = 3, verbose=1, n_jobs = -1, n_iter = 5, scoring = 'f1_weighted')


rf_RandomGrid.fit(X_train, np.ravel(y_train))

itting 3 folds for each of 5 candidates, totalling 15 fits

andomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=5,
                  n_jobs=-1,
                  param_distributions={'bootstrap': [True, False],
                                       'max_depth': [2, 6, 10, 20],
                                       'max_features': ['auto', 'sqrt'],
                                       'min_samples_leaf': [1, 2],
                                       'min_samples_split': [2, 5],
                                       'n_estimators': [100, 200, 300]},
                  scoring='f1_weighted', verbose=1)
```

Figura 3.71: Aplicação do Grid Search no modelo Random Forest

```
accuracy_score(y_test,pred_rfc)
```

```
0.9216
```

```
cm = confusion_matrix(y_test,pred_rfc)
cmd = ConfusionMatrixDisplay(cm,display_labels=['None','Low','Medium','High','Very_High'])
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fcf6724a9e8>
```

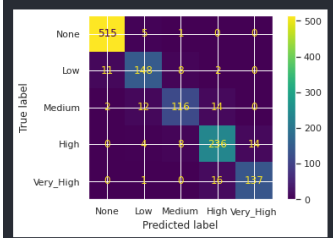

Figura 3.72: Accuracy e Matriz de confusão do modelo Random Forest

```
print(classification_report(y_test,pred_rfc))
```

	precision	recall	f1-score	support
0	0.98	0.99	0.98	521
1	0.87	0.88	0.87	169
2	0.87	0.81	0.84	144
3	0.88	0.90	0.89	262
4	0.91	0.89	0.90	154
accuracy			0.92	1250
macro avg	0.90	0.89	0.90	1250
weighted avg	0.92	0.92	0.92	1250

Figura 3.73: Resultados da aplicação do modelo Random Forest

3.2.4 Resultados e análise

Depois de se ter procedido às respetivas análises dos vários modelos implementados, determinou-se que o algoritmo Random Forest obteve os resultados mais promissores.

Após ter-se submetido no Kaggle o ficheiro com as previsões feitas pelo modelo, o grupo obteve uma accuracy de 85,6% no leaderboard público e 88,1% no leaderboard privado.

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

4 Conclusão

Com a realização deste trabalho prático, o grupo conseguiu aprofundar conhecimento relativo ao desenvolvimento de modelos de Machine Learning através da exploração de dois datasets, utilizando os modelos de aprendizagem abordados ao longo do semestre. Este projeto proporcionou ao grupo a familiarização, não só com diferentes técnicas de exploração de dados, como também com diversos modelos de extração de conhecimento.

Durante o desenvolvimento do projeto, o grupo sentiu algumas dificuldades das quais se pode realçar o facto de encontrar o melhor processo de preparação dos dados de forma benefeciar os resultados dos modelos e encontrar os melhores modelos para fazer as previsões.

O grupo verificou que técnicas mais simples não produzem necessariamente resultados inferiores, e que modelos de deep learning nem sempre são os candidatos mais ajustados. Percebeu-se que, em certos casos, é possível obter resultados mais satisfatórios recorrendo a técnicas de extração de conhecimento menos complexas e que se adequam melhor ao contexto.

Em suma, o grupo considera que fez um bom trabalho tendo implementado todas as tarefas definidas no enunciado.