Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de Dados e Aprendizagem Automática

Ano Letivo de 2022/2023

Relatório do Trabalho Prático

Grupo 18

PG50001 Fábio Oliveira
PG50259 Bernardo Saraiva
PG50519 José Gonçalves
PG50736 Rui Moreira

14 de fevereiro de 2023

DA A

Índice

1 Introdução

O presente projeto enquadra-se na unidade curricular de Dados e Aprendizagem Automática, na qual foi proposta a conceção e otimização de modelos de *Machine Learning*, através da linguagem *Python*, sendo ambos os *datasets* trabalhados num *Júpiter Notebook*.

O desafio apresentado visa desenvolver um projeto utilizando técnicas de aprendizagem abordados ao longo do semestre e tem por base dois *datasets* distintos.

O primeiro *dataset* foi fornecido pela equipa docente no formato de uma competição no *Kaggle*, e cabe ao grupo de trabalho explorar, analisar e preparar o mesmo, procurando extrair conhecimento relevante no contexto do problema em questão, de forma a alcançar a maior *precisão* possível.

O segundo é um *dataset* cuja escolha cabe ao grupo consultar, analisar e selecionar um *dataset* de entre as várias possibilidades presentes nas diferentes fontes, tendo sido escolhido através da plataforma *Kaggle*.

No presente relatório tem-se por objetivo efetuar a respetiva análise crítica dos resultados através da conceção de modelos de visualização, com o intuito de compreender inteiramente o *dataset* e as suas especificações.

2 Metodologia e aplicação

De modo a que o desenvolvimento de um projeto de *Machine Learning*, utilizando os modelos de aprendizagem, seja realizado com a maior qualidade possível, é importante seguir-se uma metodologia de extração de conhecimento. A metodologia seguida nos dois casos de estudo presentes, tanto na Previsão de acidentes rodoviários como na Previsão do Preço do Bilhete de Voos, foi a *CRISP-DM*, que contém seis etapas.

1 - *Business Understanding*, passou por diferentes períodos. Primeiramente, a parte de análise e seleção de um *dataset* de entre os que estão acessíveis a partir de fontes disponibilizadas. Seguidamente, passou-se para a perceção do *dataset* selecionado, onde se estudou em detalhe aquilo que são um dos vários tipos de voo (dataset de grupo), bem como as características ambientais nas estradas (dataset de competição) para que seja possível um bom entendimento do que se está a desenvolver.

2 - A etapa de *Data Understanding*, onde se analisou cada atributo percebendo-se se estes teriam ou não relevância para os temas em estudo.

3 - A etapa de *Data Preparation*, que assenta na limpeza de dados, onde se aplicou técnicas básicas e avançadas e comprovou-se a análise anterior levando à remoção de atributos sem relevância, deixando assim os dados preparados para a fase seguinte.

4 - A etapa de *Modeling*, onde se aplicaram diferentes modelos com diversos parâmetros de forma a perceber qual deles o melhor para ser aplicado aos dados em questão.

5 - Por fim, a etapa de *Evaluate* baseou-se na comparação entre os dados obtidos através da aplicação dos modelos e os objetivos dos negócios.

Uma vez que o projeto se enquadra numa unidade curricular, a etapa de *Deployment*, não é aplicada nos casos de estudo, já que o projeto foi desenvolvido em contexto académico.

3 Dataset Incidents

3.1 Domínio e objetivos

O dataset analisado no presente capítulo é referente ao tema proposto pela Equipa Docente, sendo o mesmo desenvolvido em prol de uma competição na plataforma Kaggle. Deste modo, corresponde ao tema *"Incidents"*, e tem como base a análise e **classificação** das características meteorológicas e atrasos em determinadas estradas para conseguir prever incidentes rodoviários (sendo este o *target* do *dataset*). Deste modo, considera-se o presente *dataset* um *dataset* de Classificação.

Relativamente ao presente estudo, explica-se em seguida os principais objetivos:

- A construção de um modelo capaz de efetuar uma boa previsão do atributo **incidents**, que poderá tomar uma classificação de *None*, *Low*, *Medium*, *High* ou *Very-High*;
- Analisar relações entre os atributos que possam influenciar a ocorrência de incidentes rodoviários;

Assim, ao longo do presente estudo, utilizar-se-á um *Jupiter Notebook* de *Python*, junto com as várias técnicas de preparação, tratamento e visualização como mecanismo para alcançar os objetivos apresentados, bem como gerar conclusões acerca do *dataset* em estudo.

3.2 Exploração, Visualização e Tratamento dos Dados

3.2.1 Descrição dos atributos do dataset

Inicialmente, de modo a ter uma perceção do problema em questão, começou-se por fazer a leitura dos atributos do dataset, sendo eles:

- **city_name** - nome da cidade em causa;
- **record_date** - o timestamp associado ao registo;
- **magnitude_of_delay** - magnitude do atraso provocado pelos incidentes que se verificam no record_date correspondente;
- **delay_in_seconds** - atraso, em segundos, provocado pelos incidentes que se verificam no record_date correspondente;
- **affected_roads** - estradas afectadas pelos incidentes que se verificam no record_date correspondente;
- **luminosity** - o nível de luminosidade que se verificava na cidade de Guimarães;
- **avg_temperature** - valor médio da temperatura para o record_date na cidade de Guimarães;

- **avg_atm_pressure** - valor médio da pressão atmosférica para o record_date na cidade de Guimarães;
- **avg_humidity** - valor médio de humidade para o record_date na cidade de Guimarães;
- **avg_wind_speed** - valor médio da velocidade do vento para o record_date na cidade de Guimarães;
- **avg_precipitation** - valor médio de precipitação para o record_date na cidade de Guimarães;
- **avg_rain** - avaliação qualitativa do nível de precipitação para o record_date na cidade de Guimarães;
- **incidents** - indicação acerca do nível de incidentes rodoviários que se verificam no record_date correspondente na cidade de Guimarães. Este é o atributo que se pretende prever com o presente estudo.

Diferença entre dataset training e test

O *dataset Train* foi usado para desenvolver e treinar modelos de *Machine Learning*, sendo neste *dataset* fornecidas informações sobre o número de acidentes rodoviários por registro (*incidents*).

O modelo a desenvolver deverá ter, 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 outro lado o *dataset Test* é utilizado para validar a *accuracy* do modelo de dados, tendo este ainda não sido utilizado pelo mesmo para extrair conhecimento, o que permite verificar e evitar possíveis situações de *overfitting*.

3.2.2 Exploração dos dados

Após a leitura e análise da descrição dos atributos, o primeiro passo foi ler os *datasets* de treino e de teste (*training_data.csv* e *test_data.csv*, respetivamente) com recurso à biblioteca *pandas*.

Numa fase inicial, começou-se por efetuar a **Visualização dos Dados**, com o objetivo de analisar e compreender o dataset, tomando-se como ponto de partida para o tratamento de dados (3.2.8).

Durante o tratamento de dados, foi-se recorrendo às várias funcionalidades e métodos de visualização descritos em seguida para analisar a correção e a eficácia dos mesmos, apresentando-se em seguida os resultados obtidos com as alterações efetuadas.

A função *info* foi aplicada ao dataset de teste, tendo sido útil para permitir apresentar apresentar os atributos e o respetivo tipo de dados de cada um, como se demonstra em seguida.

```
[11]  train.info()

...   <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 5000 entries, 0 to 4999
      Data columns (total 15 columns):
       #   Column            Non-Null Count  Dtype
      ---  ------            --------------  -----
       0   delay_in_seconds  5000 non-null   int64
       1   affected_roads    4915 non-null   object
       2   luminosity        5000 non-null   int64
       3   avg_temperature   5000 non-null   float64
       4   avg_atm_pressure  5000 non-null   float64
       5   avg_humidity      5000 non-null   float64
       6   avg_wind_speed    5000 non-null   float64
       7   avg_rain          5000 non-null   int64
       8   incidents         5000 non-null   object
       9   total_estradas    5000 non-null   int64
       10  Year              5000 non-null   int64
       11  Month             5000 non-null   int64
       12  Day               5000 non-null   int64
       13  Hour              5000 non-null   int64
       14  Minutes           5000 non-null   int64
      dtypes: float64(4), int64(9), object(2)
      memory usage: 586.1+ KB
```

Figura 3.1: Resultado obtido através da função *info* que descreve os atributos

3.2.3 Linear Correlation

Através da matriz de correlação, já com o tratamento de dados efetuado, foi possível inferir quais os pares de atributos poderão estar, em grande parte, a fornecer o mesmo conhecimento ao modelo ou ter uma dependência intrínseca aos atributos, tornando-se assim repetitivos para gerar conhecimento.

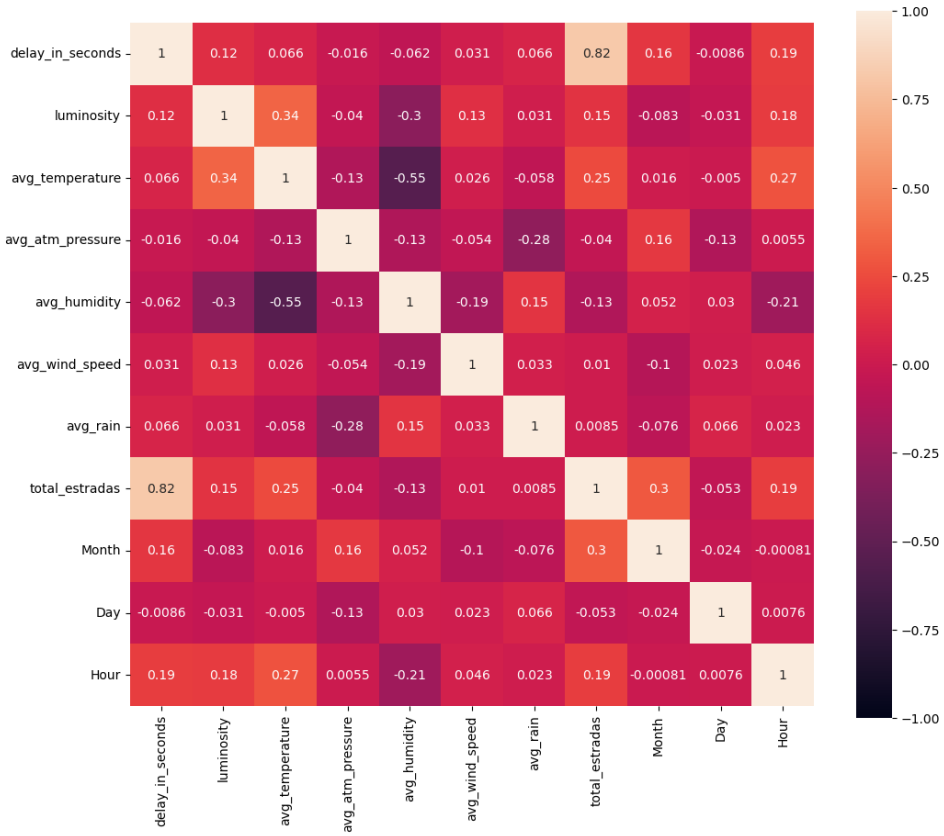

Figura 3.2: Resultado obtido na matriz de correlação do *dataset* de treino sem qualquer tipo de tratamento

Através da Figura 3.2, é possível destacar que os atributos *total_estradas* e *delay_in_seconds* têm uma correlação elevada, nomeadamente 0.87, mas não foi efetuado qualquer tratamento, uma vez que para além de estarem logicamente relacionadas (mais estradas cortadas leva a um maior atraso nas deslocações) têm diferentes propósitos e ambos são úteis para o modelo.

3.2.4 Distribuição de Incidentes por Hora

Em seguida procurou-se analisar a distribuição de incidentes por hora, tendo-se separado pelos graus de gravidade, como se observa na imagem seguinte.

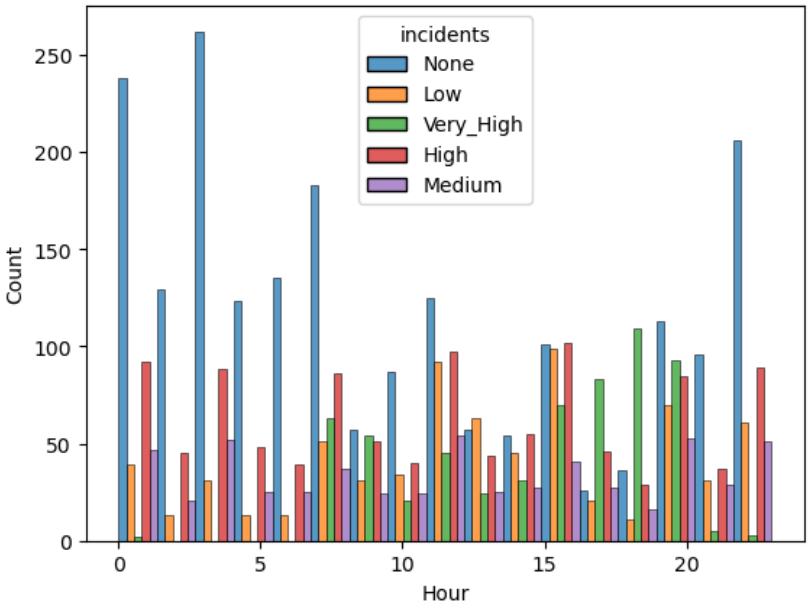

Figura 3.3: Distribuição dos incidentes por hora, com separação pelo grau de gravidade

Deste modo, foi possível verificar que:

- Durante o período noturno (entre as 20H e as 6H) é possível verificar que a ocorrência de incidentes é reduzida, ou até inexistente.
- Os incidentes tendem a ocorrer no período das 7:30H às 20H, com especial volume nos horários de ponta, podendo este resultado ser possivelmente explicado pela saída do trabalho, mas sem certeza devido à falta de indicadores.

3.2.5 Distribuição de Incidentes por Mês

Similarmente, analisou-se o número de incidentes por mês com base no seu nível, demonstrando-se o resultado obtido pela figura seguinte, onde se pode verificar que tendem a ocorrer nos meses mais chuvosos, causando consequentemente mais incidentes.

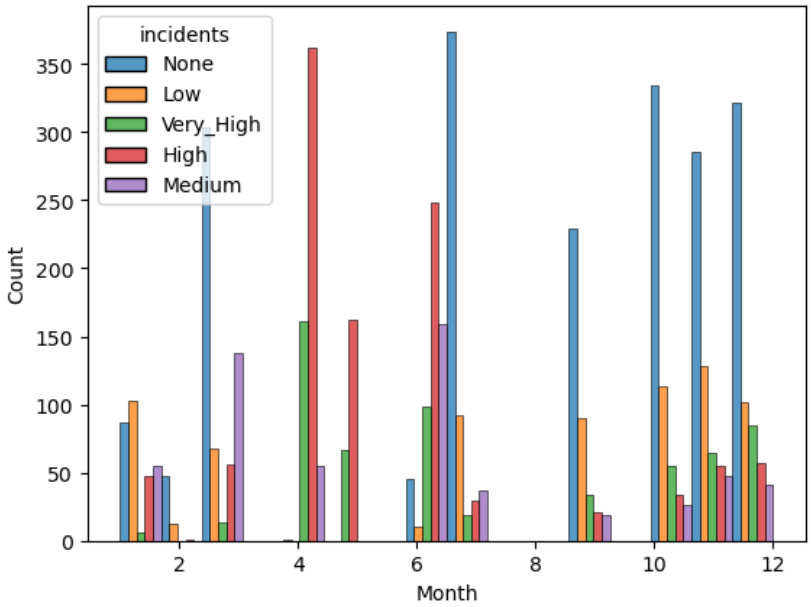

Figura 3.4: Número de incidentes por mês com base no seu nível

3.2.6 Número de estradas afetadas pelo delay

Outra das relações analisadas foi o delay com o número de estradas afetadas (sendo este um dos atributos resultantes do tratamento efetuado), como se demonstra em seguida:

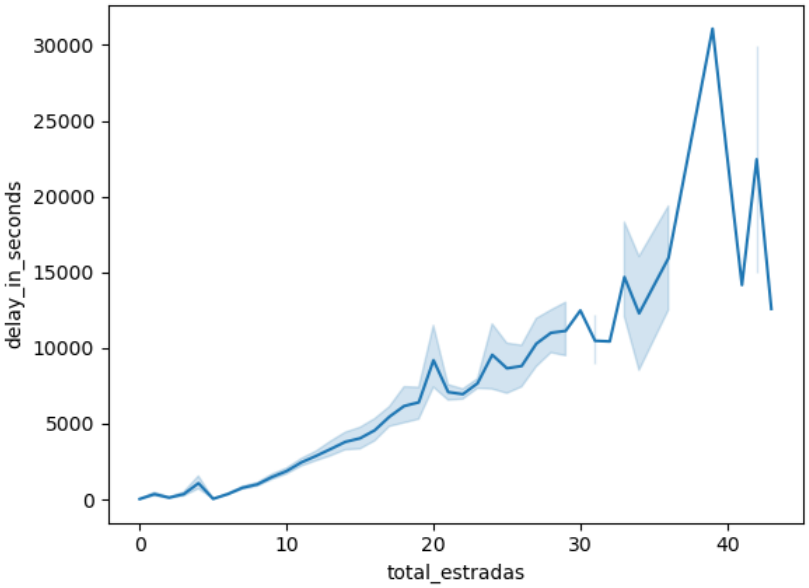

Figura 3.5: Relação entre os atributos *delay* e total de estradas cortadas

Através desta análise, verifica-se que o aumento do número de estradas afetadas relaciona-se quase linearmente com o *delay* em segundos, justificando-se pelo facto de redirecionar o trânsito todo para as mesmas estradas e causando uma maior afluência nas mesmas.

3.2.7 Relação entre número de estradas afetadas e gravidade dos incidentes

A nível da visualização, foi também analisado o número de estradas afetadas relativamente à gravidade do incidente, como é possível ver na figura a seguir.

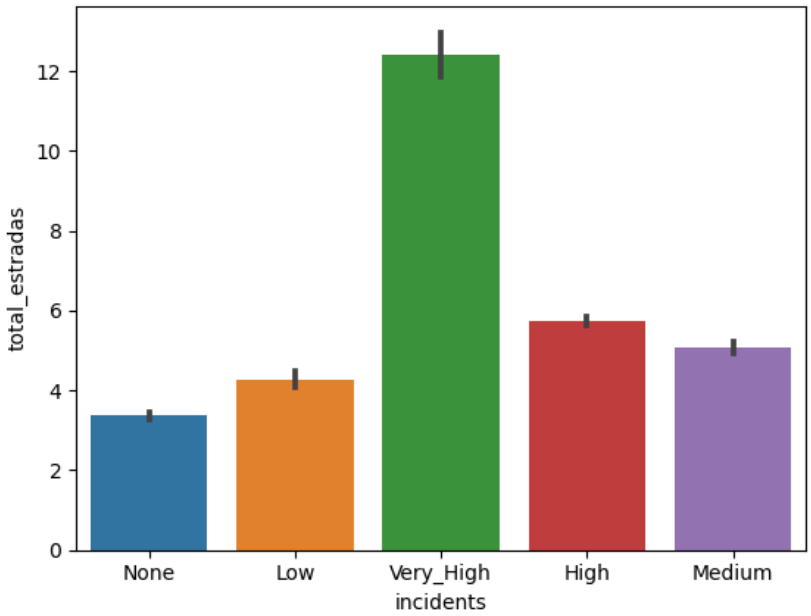

Figura 3.6: Relação entre o número de estradas afetadas e a gravidade dos incidentes

Como é possível verificar na figura acima o número de estradas influencia a gravidade do incidente de forma positiva, na medida em que quanto maior for o número de estradas afetadas, maior será a gravidade do incidente.

3.2.8 Tratamento dos dados

Remoção de atributos

O primeiro passo para o tratamento de dados foi a remoção do atributo *city*, uma vez que é um atributo nominal que não acrescenta conhecimento ao modelo. Deste modo, recorreu-se à função *drop* para remover esta coluna do *dataset*. De igual modo se procedeu para a coluna *magnitude_of_delay*, uma vez que mais de 86% destes valores são *UNDEFINED*, assim como para a coluna *avg_precipitation*, em que todos os valores são nulos.

Label Encoding

Através da análise ao dataset, foi possível verificar que os valores correspondentes aos atributos *luminosity* e *avg_rain* são nominais, mas que representam valores de uma escala, tendo então sido utilizado o Label Encoding em ambos. Através desta técnica, foi possível converter as *labels* de uma forma numérica, permitindo assim convertê-los numa forma *machine-readable*.

Deste modo, as seguintes alterações foram feitas:

- **Luminosity:** A escala que anteriormente passava por LOW_LIGHT - LIGHT - DARK passou a ser 0 - 1 - 2;

- **avg_rain:** A escala que anteriormente ia de sem_chuva - chuva forte - chuva fraca - chuva moderada - chuva forte passou a ser constituida pelos valores numéricos 0 - 1 - 2 - 3 - 4

Para efetuar o Label Encoding recorreu-se ao *package preprocessing* do *ScikitLearn*, usando a função fit_transform como se demonstra em seguida:

```
label_encoder = preprocessing.LabelEncoder()
train['luminosity'] = label_encoder.fit_transform(train['luminosity'])
train['avg_rain'] = label_encoder.fit_transform(train['avg_rain'])
test['luminosity'] = label_encoder.fit_transform(test['luminosity'])
test['avg_rain'] = label_encoder.fit_transform(test['avg_rain'])
```

Tratamento de affected roads

Um dos atributos presentes no *dataset* é o *affected roads*, que toma como valores as várias estradas afetadas por incidentes. Esta informação está representada através de uma string com as várias estradas separadas entre vírgulas. Ora, este formato não é *machine-readable*, não sendo viável inferir conhecimento através do mesmo e levando a que sofra erros ou enviesamento.

Para resolver o problema supramencionado, tomou-se como solução inicial a separação da string pela vírgula, que separa as diferentes estradas. Após este passo, decidiu-se substituir a coluna *affected_roads* por a coluna *total_estradas*, utilizando assim o número total de estradas.

De referir que se testou também uma estratégia de one-hot encoding, mas não se beneficiou com esta estratégia, pelo que se manteve para esta *feature* a contagem do nº total de estradas. Este fenómeno poderia ser explicado pelo facto de o número de estradas possível ser consideravelmente extenso, introduzindo várias variáveis novas, levando assim a um problema extenso (com muitos predictors). Outra das causas encontradas foi o problema da Multicolinearidade, que consiste em um problema comum, no qual as variáveis independentes possuem relações lineares exatas ou aproximadamente exatas, sendo assim menos informativas e baixando a accuracy. Também em alguns modelos de aprendizagem, no caso de duas colunas identicas, a sua implementação automaticamente faz *drop* a uma delas, levando novamente a um decrescimo de informação e consequente *accuracy* mas baixa.

Extração de atributos a partir da data

Um dos atributos pertencentes ao *dataset* era o record_date que corresponde à data do acidente. Estes valores não acrescentam grande informação para o modelo da forma como estão representados (como uma string), sendo que para obter as várias informações possíveis a partir da data começou-se por definir as funções que permitem separar os atributos Ano, Mês, Dia, Hora e Minutos. Em seguida, após a separação dos atributos em várias colunas, removeu-se a o atributo *record_date* do *dataset*.

O processo anterior está apresentado abaixo:

```
def separate_year(time):
    res = re.search(r'(\d+)-(\d+)-(\d+) (\d+):(\d+)', time)
    if res:
        return int(res.group(1))

def separate_month(time):
    res = re.search(r'(\d+)-(\d+)-(\d+) (\d+):(\d+)', time)
```

```
    if res:
        return int(res.group(2))

def separate_day(time):
    res = re.search(r'(\d+)-(\d+)-(\d+) (\d+):(\d+)', time)
    if res:
        return int(res.group(3))

def separate_hour(time):
    res = re.search(r'(\d+)-(\d+)-(\d+) (\d+):(\d+)', time)
    if res:
        return int(res.group(4))

def separate_minutes(time):
    res = re.search(r'(\d+)-(\d+)-(\d+) (\d+):(\d+)', time)
    if res:
        return int(res.group(5))

train['Year'] = train['record_date'].apply(separate_year)
train['Month'] = train['record_date'].apply(separate_month)
train['Day'] = train['record_date'].apply(separate_day)
train['Hour'] = train['record_date'].apply(separate_hour)
train['Minutes'] = train['record_date'].apply(separate_minutes)
test['Year'] = test['record_date'].apply(separate_year)
test['Month'] = test['record_date'].apply(separate_month)
test['Day'] = test['record_date'].apply(separate_day)
test['Hour'] = test['record_date'].apply(separate_hour)
test['Minutes'] = test['record_date'].apply(separate_minutes)

train = train.drop(['record_date'],axis=1)
test = test.drop(['record_date'],axis=1)
```

3.3 Modelos de aprendizagem

Antes de construir o modelo de *machine learning*, foi necessário efetuar uma pesquisa inicial acerca de quais as técnicas mais adequadas para o presente problema.

Para o modelo em questão, foi utilizado um modelo de validação train_test_split de 30% para teste e 70% para treino de modo a simular como o modelo se comporta com novos dados. Em todos os modelos foi usado como random seed o valor 2022, com o objetivo de permitir simular os resultados do modelo constantemente.

Deste modo, os modelos foram treinados usando os seguintes modelos:

- **Logistic Classification**
- **Random Forest**
- **BaggingClassifier**
- **CatBoostClassifier**

3.3.1 Logistic Regression

O algoritmo Logistic Regression que se baseia na análise estatística para prever o resultado de forma binária. O algoritmo prevê uma variável dependente analisado a relação entre uma ou mais variáveis independentes.

Para este algoritmo, foi efetuado um fitting com os dados de treino, com o objetivo de que o modelo pudesse aprender, sendo posteriormente efetuada a previsão com os dados de teste.

Para os parâmetros utilizados neste modelo, foi utilizado um máximo de iterações igual a 100000, sendo este valor referente ao número máximo de iterações tomado pelo solver para convergir um resultado.

Através deste modelo, o melhor resultado obtido com o tunning dos parâmetros foi de 72.87%, o que revelou que este talvez não seja o melhor algoritmo para este modelo.

Uma possível explicação para o mesmo poderá ser o facto de o *target* ter uma baixa correlação com as features, levando a um resultado menos assertivo para a previsão final.

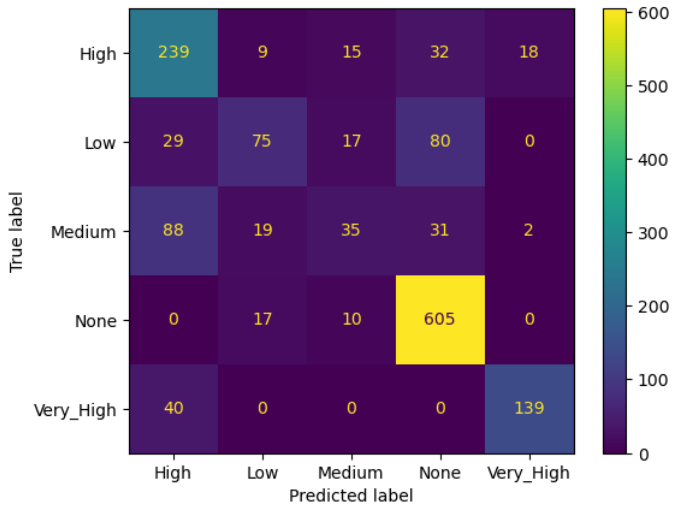

Figura 3.7: Tabela da accuracy referente ao modelo Logistic Regression

3.3.2 Random Forest

O algoritmo Random Forest é um meta-estimator que permite criar várias árvores de decisão, de maneira aleatória, formando o que se chama de uma floresta, onde cada árvore será utilizada na escolha do resultado final, em uma espécie de votação. Este método faz parte dos ensemble learners, que possuem uma característica principal que os diferenciam: a combinação de diferentes modelos para se obter um único resultado, tornando esses algoritmos mais robustos e complexos, levando a um maior custo computacional que costuma ser acompanhando de melhores resultados.

Para a previsão efetuada com este algoritmo, vários parâmetros foram testados, tendo-se usado 1000 estimadores/árvores (*n_estimators=1000*).

Com este modelo, foi possível obter resultado de 92.27%, notando-se uma grande melhoria face ao *Logistic Regression*.

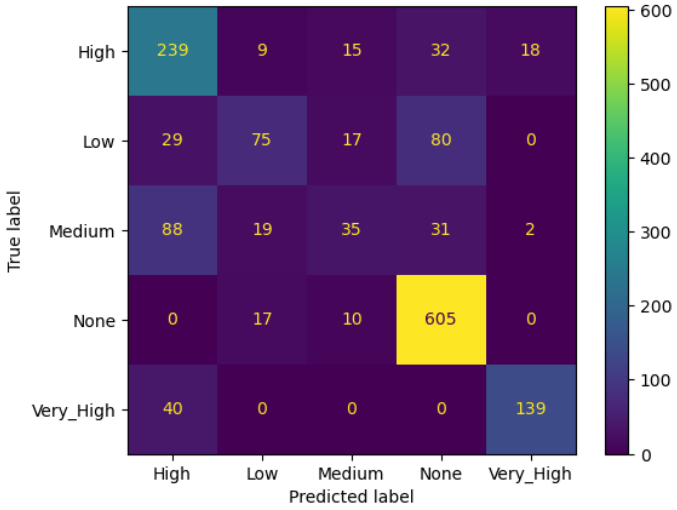

Figura 3.8: Tabela da accuracy referente ao modelo Random Forest

3.3.3 CatBoostClassifier

O algoritmo CatBoostClassifier é baseado em árvores de decisão com o aumento de gradiente. Durante o treinamento, um conjunto de árvores de decisão é construído consecutivamente. Cada árvore sucessiva é construída com uma perda reduzida em comparação com as árvores anteriores. O número de árvores é controlado pelos parâmetros iniciais.

Neste algoritmo foram utilizados e testados parâmetros, tendo-se escolhido 700 iterações (nº de trees), learning_rate=1 (não se observou overfitting) e depth=4 como profundidade das árvores.

O modelo conseguir obter um *score* de 92.93%, tendo melhorado minimamente em comparação com os anteriores.

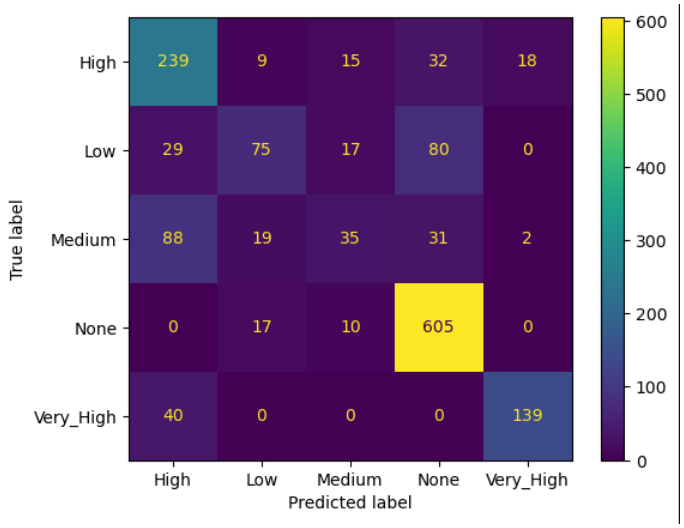

Figura 3.9: Tabela da accuracy referente ao modelo CatBoost

3.3.4 BaggingClassifier

O algoritmo BaggingClassifier é um meta-estimador que ajusta cada classificador base em subconjuntos aleatórios do dataset original e, em seguida, agrega as previsões individuais (por votação ou por média) de forma a chegar a uma previsão final.

Relativamente aos parâmetros usados para este modelo foi utilizado como estimador o *DecisionTreeClassifier*, tendo sido testado também com o *RandomForest*, com o qual se obteve um resultado pior. Foram utilizados também 1000 estimadores base e 0.7 para parâmetro *max_samples*, que define o número de amostras do estimador.

Este modelo obteve uma *accuracy* de 93.06% , tendo este sido o melhor resultado obtido entre todos os modelos testados.

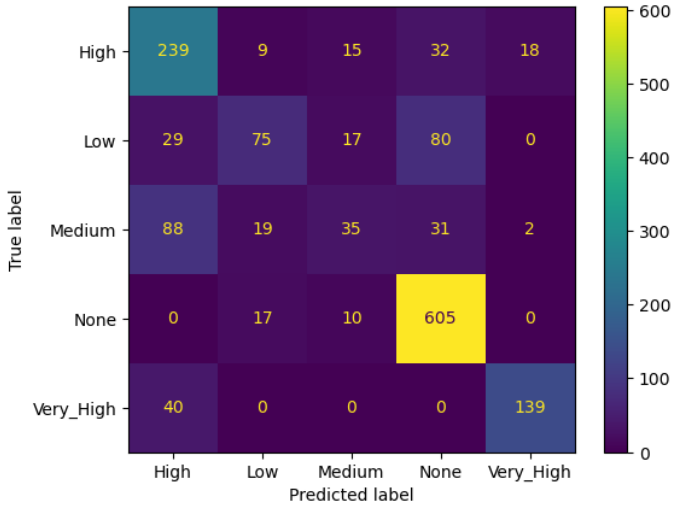

Figura 3.10: Tabela da accuracy referente ao modelo BaggingClassifier

Após se ter testado alguns outros modelos, os 4 modelos de aprendizagem referidos foram os mais eficazes.

4 Dataset Flights

4.1 Domínio e objectivos

O dataset analisado neste capítulo contém informação sobre opções de reserva de voos do website "Ease My Trip". "Ease My Trip" é uma empresa indiana de viagens online. O atributo definido como target foi o preço (do bilhete de avião).

Os objectivos principais da análise deste estudo são:

- construir um modelo capaz de fazer uma boa previsão do preço do bilhete de avião com base num conjunto de atributos.
- extrair conhecimento relevante dos dados que permitam à empresa "Ease My Trip" optimizar o seu sistema e tomar boas decisões estratégicas, e assim aumentar o sucesso da empresa.
- extrair informações valiosas que serão de enorme valor para os passageiros.

De modo a atingir estes objectivos utilizar-se-á técnicas de preparação, tratamento, exploração e visualização de dados. Para prever o preço do bilhete de avião serão utilizados modelos de *machine learning* de regressão, pois o *target* (preço) é uma variável contínua.

4.2 Exploração e Tratamento dos dados

4.2.1 Descrição dos atributos do dataset

Antes de fazer qualquer análise ao *dataset* leu-se a descrição dos atributos que foi fornecida, juntamente com os dados de treino. Entender a definição de cada um dos atributos é fundamental para a perceção do problema em questão, os valores que estão associados a cada atributo, e perceber as estratégias a aplicar na preparação do *dataset*. A descrição de cada atributo do dataset pode ser encontrada no website Kaggle.

4.2.2 Construção do dataset

Primeiramente foi necessário construir o dataset no notebook. Os dados do problema estavam inicialmente separados em dois conjuntos de dados: um dataset com os dados relativos aos bilhetes de class económica e outro com os bilhetes de class business. Deste modo cada conjunto de dados (estavam escritos num ficheiro com formato CSV) foi lido usando o comando **read_csv** .

```
business = pd.read_csv('business.csv')
economy = pd.read_csv('economy.csv')
```

Como os dados estavam organizados separadamente em *business* e *economy*, cada um dos datasets não tinha o atributo *business*. Assim, adicionou-se a cada um dos *datasets* uma nova coluna chamada *business* e foram preenchidos todos os campos com o valor *1* ou *0* dependendo do caso.

```
business['business'] = 1
economy['business'] = 0
```

Após a adição da coluna *bussiness* juntou-se os registos dos dois conjuntos de dados em apenas um *dataset* utilizando para isso o comando **concat**.

```
dataset = pd.concat([business,economy])
```

Como os nomes dos atributos não eram muito explicativos, para facilitar o reconhecimento dos dados no tratamento e preparação futura do *dataset*, renomeou-se as colunas da seguinte forma:

```
dataset.rename(columns= {'date':'date_travel','ch_code':'character_code', 'num_code':'numerical_code',
↪  'dep_time' :'departure_time', 'from':'souce_city', 'time_taken':'duration_flight', 'stop':'stops',
↪  'arr_time':'arrival_time', 'to':'destination_city'},inplace = True)
```

Após a preparação do dataset foi obtido como output o resultado apresentado na figura 4.1. Da visualização geral dos dados do *dataset* foram feitas algumas observações:

- De que tipo eram os valores dos vários atributos.
- Qual o formato de representação de cada atributo. Por exemplo, o atributo *duration_flight* está representado em horas e minutos embora este possa poder ser representado em apenas minutos.
- Quais os diferentes valores para cada atributo.

	date_travel	airline	character_code	numerical_code	departure_time	source_city	duration_flight	stops	arrival_time	destination_city	price	business
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	non-stop	20:00	Mumbai	25,612	1
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	non-stop	21:15	Mumbai	25,612	1
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\t...	20:45	Mumbai	42,220	1
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\t...	23:55	Mumbai	44,450	1
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\t...	23:55	Mumbai	46,690	1

Figura 4.1: Visualização dos primeiros 5 registos do dataset

Da figura 4.1 é possível ver os vários atributos que foram explorados:

```
data_travel                 departure_time              arrival_time
airline                     source_city                 destination_city
character_code              duration_flight             price
numerical_code              stops                       business
```

4.2.3 Redução do tamanho do *dataset*

Explorando o comando **dataset.count()** verificou-se que o *dataset* tinha 300261 registos. Este *dataset* é de dimensão relativamente grande, e nestes casos isso pode implicar um grande impacto no tempo de treino do

modelo de *machine learning*. De modo a aumentar a performance do processo de treino efectuamos a redução do tamanho do *dataset*. Primeiro aplicou-se o comando para reduzir o tamanho do dataset foi o **drop_duplicates** para remover informação repetida. Neste caso este passo não se teve grande impacto já que apenas 2 linhas foram removidas.

Visto que o comando acima não teve muito impacto, aplicou-se um **Row Sampling** para selecionar apenas 90% dos registos. Escolheu-se este valor pois era a maior percentagem de registos que conseguimos remover sem que houvesse uma redução no valor da métrica *R-Squared*. Escolheu-se a opção *Stratified Sampling* pelo atributo price para que continuar a ter a mesma proporção dos diferentes valores do atributo.

O processo descrito está descrito abaixo:

```
# duplicate row filtering
dataset.drop_duplicates(inplace=True)
# row sampling
dataset.groupby('price', group_keys=False).apply(lambda x: x.sample(frac=0.9))
```

4.2.4 Conversão de tipos

O tipo do atributo stops era string tal como foi visualizado na figura 4.1. Contudo, um atributo que represente o número de paragens do avião, pode também ser representado como valor numérico ordinal (0, 1, 2, ...). O tipo do atributo stops foi convertido de string para numérico ordinal pois deste modo o modelo consegue reconhecer a relação de ordem entre os diferentes valores caso em que não acontecia com o tipo string. Para realizar a conversão o primeiro passo passou por identificar os diferentes valores possíveis para o atributo stops, ou seja, os valores únicos. Uma das formas de conseguir obter os valores únicos dos atributos nominais foi a partir dos seguintes comandos:

```
stopValues = dataset.groupby(['stops'])['stops'].count()
print(stopValues)
```

O resultado obtido apresenta-se de seguida:

```
stops
1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t                    243601
...
1-stop\n\t\t\t\t\t\t\t\t\t\t\t\tVia Vishakhapatnam\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t      8
2+-stop                                                                          13288
non-stop                                                                         36044
```

Dada a relação entre os diversos valores possíveis para o atributo stops, foram atribuídos valores inteiros que respeitassem a ordem entre eles. A ordem estabelecida foi a seguinte:

```
non-stop  |  [1-stop, 1-stop Via IXU, ...]  |  2+-stop
    0     |                1                |     2
```

A conversão foi realizada da seguinte forma:

```
dataset['stops'] = dataset['stops'].str.replace('\n', '')
dataset['stops'] = dataset['stops'].str.replace('\t', '')
dataset['stops'] = dataset['stops'].str.replace(r'1(.)+',  '1', regex=True)
dataset['stops'] = dataset['stops'].str.replace(r'no(.)+', '0', regex=True)
dataset['stops'] = dataset['stops'].str.replace(r'2(.)+',  '2',regex=True)
dataset['stops'] = dataset['stops'].apply(int)
```

Outro dos atributos em que foi necessário alterar o tipo foi o *price*. Este atributo é o *target* de um problema de regressão e portanto deverá ser convertido para tipo numérico. Este processo foi feito da seguinte forma:

```
dataset['price'] = dataset['price'].str.replace(',', '')
dataset['price'] = dataset['price'].apply(int)
```

Outra das conversões de tipo necessário foi a do atributo *duration_flight* de string para valor númerico. A razão pela qual realizamos esta conversão foi semelhante ao caso do atributo *stops*. Neste caso foi necessário criar uma função com conhecimento de expressões regulares, para converter os valores para um valor inteiro.

```
def to_minutes(time_string):
result = re.search(r'(\d+)h.*?(\d+)m', time_string)
if (result):
   minutes = int(result.group(1)) * 60 + int(result.group(2))
   return minutes
else:
    return -1
```

Neste processo 4 registos não puderam ser convertidos. Como estes registos não apresentavam características únicas do *dataset* decidimos remover essas linhas. Este processo está explicito abaixo:

```
dataset['duration_flight'] =  dataset['duration_flight'].apply(to_minutes)
# remover linhas que nao seja possivel converter para int
dataset = dataset[dataset.duration_flight ≠ -1]
```

4.2.5 Extracção de atributos a partir da data da viagem

Um dos atributos pertencentes ao dataset era o *date_travel* que corresponde à data da viagem. Os valores do *date_travel* não apresentam muita informação para o modelo da forma como estão representados (como uma *string*). Deste modo para obter as várias informações possíveis a partir da data começou-se por efectuar a conversão do formato inicial da data (dd-mm-yyyy) para outro formato (yyyy-mm-dd) de maneira a conseguir usar algumas funcionalidades da data. Por fim, transformou-se este atributo *date_travel* nos seguintes atributos:

- weekend - Dia da Semana
- year - Ano
- month - Mês
- day - Dia

O processo anterior está apresentado abaixo:

```
# atributo dia da semana
dw_mapping={
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 0,
    5: 1,
    6: 1
}
dataset['weekend']=dataset['date_travel'].dt.weekday.map(dw_mapping)
dataset['year'] = dataset['date_travel'].dt.year
dataset['month'] = dataset['date_travel'].dt.month
dataset['day'] = dataset['date_travel'].dt.day

dataset = dataset.drop(['date_travel'], axis=1)
```

4.2.6 Extracção de atributos a partir da hora de partida e hora de chegada

A hora de partida e hora de chegada não apresentam muita informação para o modelo da forma como estão representados. Deste modo separou-se cada um destes atributos em hora e minutos, resultando assim nos seguintes atributos adicionados ao dataset:

- departure_hour
- departure_min
- arrival_hour
- arrival_min

Este processo é descrito de seguida.

```
def separate_hour(time):
    res = re.search(r'(\d+):(\d+)', time)
    if res:
        return int(res.group(1))

def separate_min(time):
    res = re.search(r'(\d+):(\d+)', time)
    if res:
        return int(res.group(2))

dataset['departure_hour'] = dataset['departure_time'].apply(separate_hour)
dataset['departure_min'] = dataset['departure_time'].apply(separate_min)
dataset = dataset.drop(['departure_time'],axis=1)

dataset['arrival_hour'] = dataset['arrival_time'].apply(separate_hour)
dataset['arrival_min'] = dataset['arrival_time'].apply(separate_min)
dataset = dataset.drop(['arrival_time'],axis=1)
```

4.2.7 Label Enconding

Como foi explicado em 3.2.8, neste dataset também foi realizado label encoding dos seguintes atributos:

- airline
- character_code
- source_city
- destination_city

4.3 Visualização de dados

4.3.1 Distribuição do preço dos bilhetes

Com o objetivo de melhorar os resultados produzidos pelo modelo e também conhecer melhor o dataset selecionado, foi necessário criar uma secção no notebook para a vizualização de dados. De seguida, analisa-se detalhadamente os gráficos e tabelas obtidas, bem como o processo para a sua obtenção.

4.3.2 Preço dos bilhetes

De forma a analizar a ordem de grandeza do preço dos bilhetes, foi realizado o *binning* dos dados em 10 bins e através de um **Histplot** originou-se o gráfico de barras evidenciado de seguida. É de salientar que a unidade monetária deste dataset é a Rupia indiana.

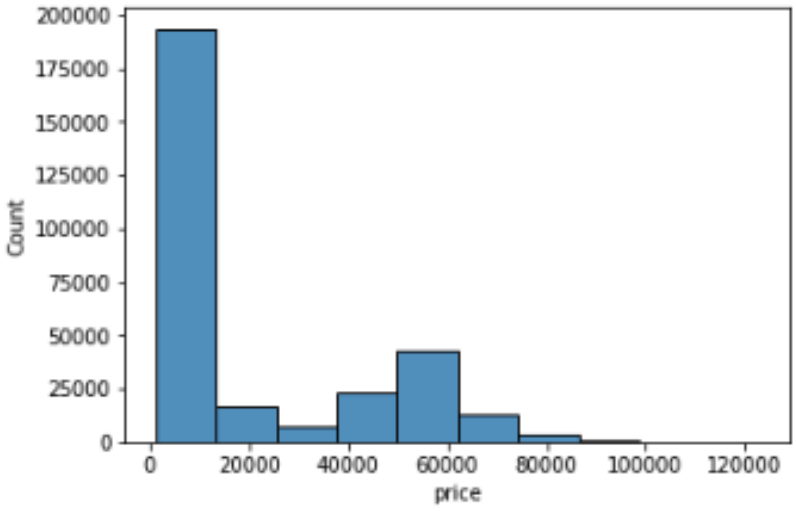

Figura 4.2: Distribuição do preço dos bilhetes

4.3.3 Duração da viagem

De modo a tratar mais eficazmente o modelo foi necessário verificar a duração das viagens referenciadas no *dataset*, com o intuito de perceber a ordem de grandeza a ser tratada. Com este fim, foi utilizado um **Box Plot** com o respetivo output de seguida. Apesar de existirem outliers representados no gráfico resultante neste caso

não se deverá aplicar nenhum tratamento, dado que como no *dataset* existem voos com diferentes números de escalas seria possível, com por exemplo a remoção de *outliers*, remover também as viagens com um número de escalas elevadas o que tornaria o modelo despreparado para prever o preço deste tipo de voo. Para além disso os restantes atributos foram analisados e em nenhum faria sentido aplicar a remoção de *outliers*.

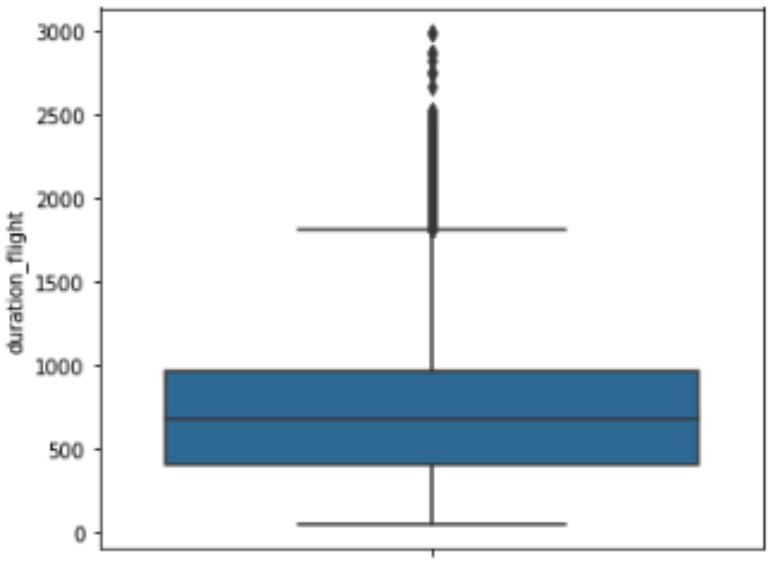

Figura 4.3: Distribuição da feature duration_flight

4.3.4 Bilhetes vendidos por airline

Para analisar o número de bilhetes vendidos por cada *airline* foi necessário recorrer a um **Bar Plot** de forma a representar também a distribuição da classe do bilhete por companhia aérea. Ao analisar os dados obtidos, é possível verificar que apenas 2 companhias possuem a classe *business*, mas também que existem algumas companhias que realizam um número reduzido de vendas o que poderá ser uma informação útil para a administração da rede de vendas criadora do dataset.

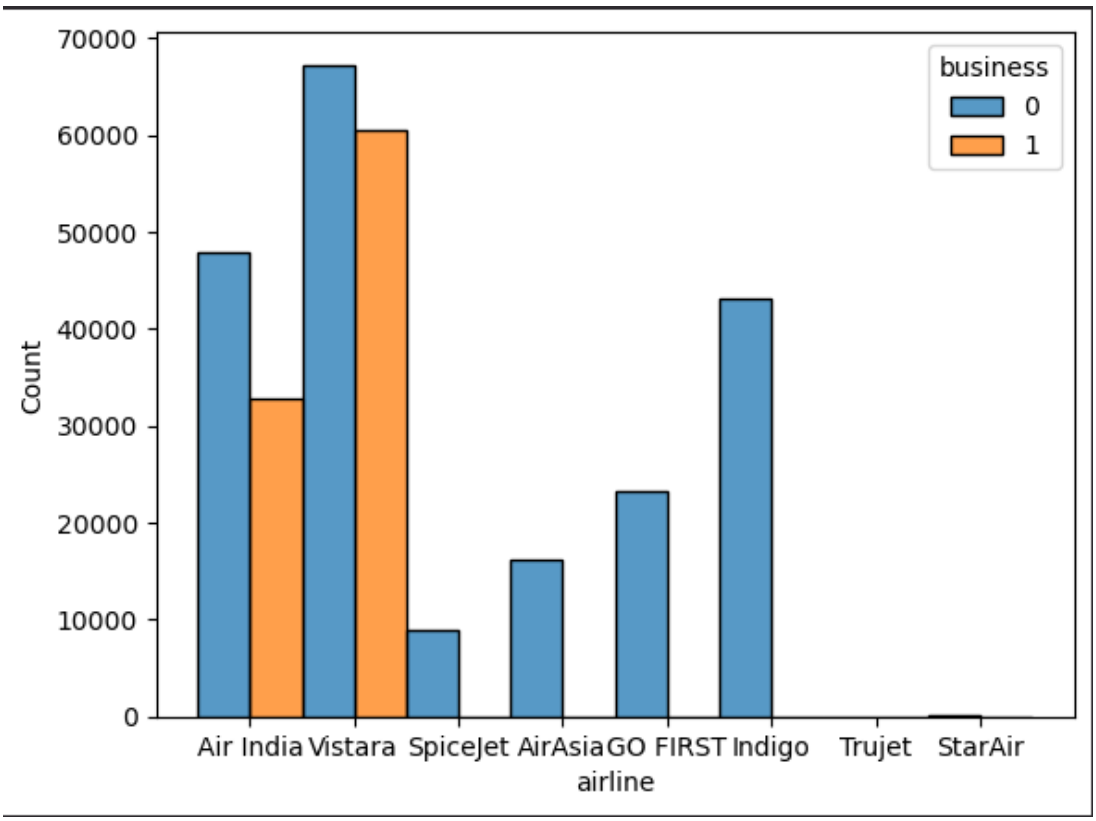

Figura 4.4: Número de Tickets por airline.

4.3.5 Preço por airline

Uma das melhores features para diferenciar a o preço dos bilhetes é a airline, deste modo foi analisada a distribuição de preço em cada uma das companhias aéreas presentes no dataset, com o auxílio de um **Box Plot** analisar a influência das várias companhias em estudo através do gráfico apresentado de seguida. Com este gráfico, é possível perceber que as empresas "Air India"e "Vistara"mais variadade no preço dos bilhetes, variedade esta que pode ser explicada por apenas estas empresas possuirem bilhetes com da classe business.

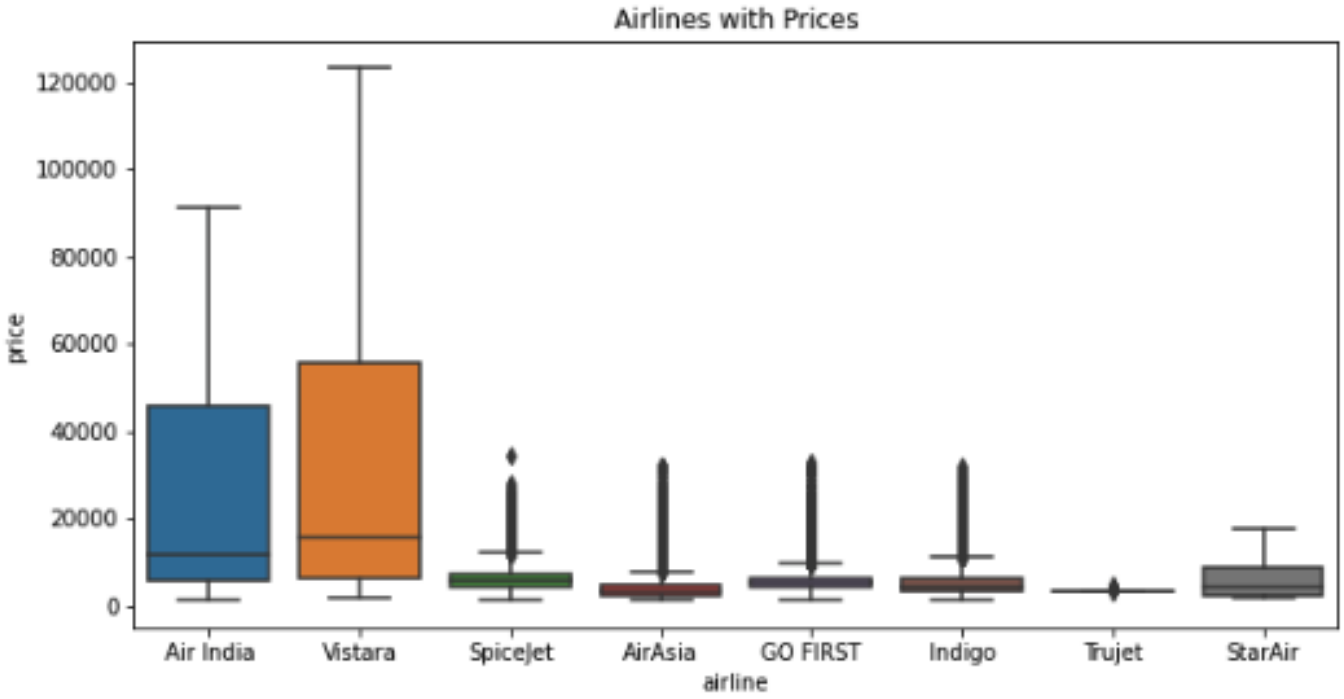

Figura 4.5: Distribuição do preço por airline

Ainda na sequência do estudo da influência das diferentes airlines no preço do bilhete, e para melhor perceber a sua relevância neste contexto, mostrou-se necessária analisar também a média dos bilhetes vendidos por companhia aérea, os resultados obtidos através de um **Barplot**, apresentam-se de seguida.

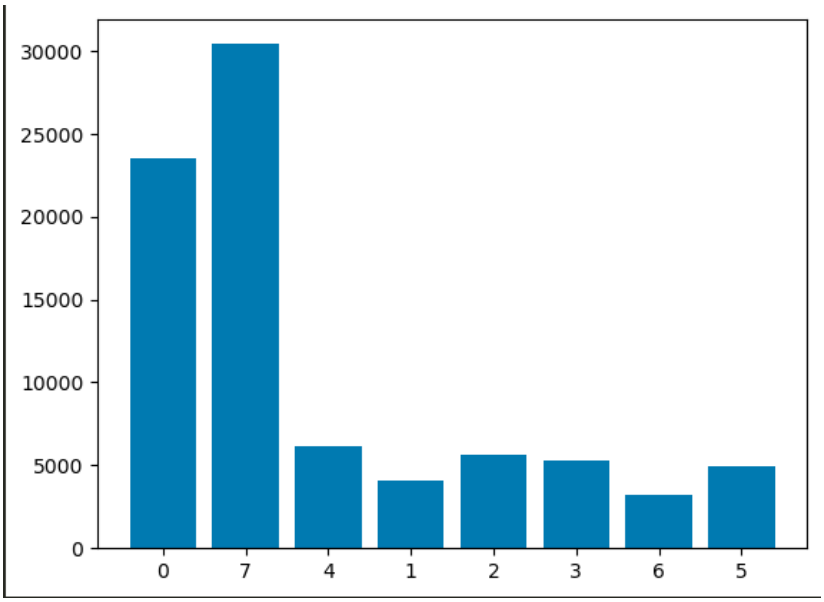

Figura 4.6: Média do preço do bilhete por airline

4.3.6 Preço do bilhete ao fim de semana e à semana

Intuitivamente, o preço dos bilhetes ao fim de semana seria maior e por isso foi aplicado tratamento de dados nesse sentido. Com o intuito de conseguir separar o dataset em entradas ao fim de semana e à semana acrescentou-se uma coluna com esta informação. Desta forma, foi possível criar o gráfico apresentado de seguida, que apresenta

a média do preço dos bilhetes em ambas as situações recorrendo a um **Bar Plot**. No entanto, não se verificou um desvio relevante nos casos em estudo para a coluna previamente criada ser aplicada no modelo.

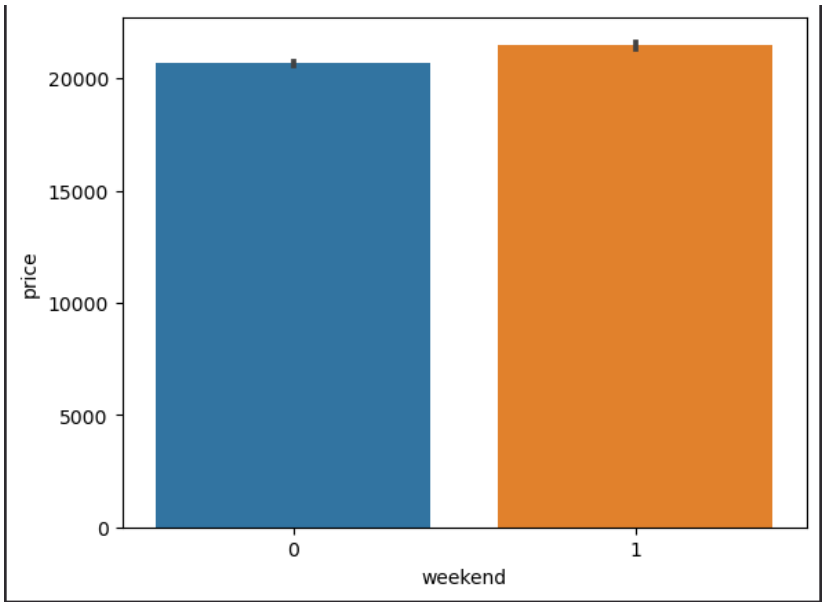

Figura 4.7: Média preço do bilhete à semana e ao fim de semana

4.3.7 Linear Correlation

Para verificar a importância das diversas features para a interpolação do target, foi feita uma visualização **Linear Correlation**, desta forma foi possível verificar ao longo do desenvolvimento do modelo as features que necessitavam de tratamento para obterem uma maior correlação, ou até mesmo de modo a retirar features irrelevantes no caso em estudo, como por exemplo a feature *character code* que por ter uma correlação muito perto de 1 com a feature *airline* se demonstrou desnecessária, visto que não acrescenta nenhuma informação (character code é a sigla de airline).

Este processo foi efectuado da seguinte maneira:

```
corr_matrix = dataset.corr()
f , ax = plt.subplots(figsize=(12, 10))
sns.heatmap(corr_matrix, vmin=-1, vmax=1, square=True, annot=True)
```

Os valores de correlação encontram-se apresentados em seguida.

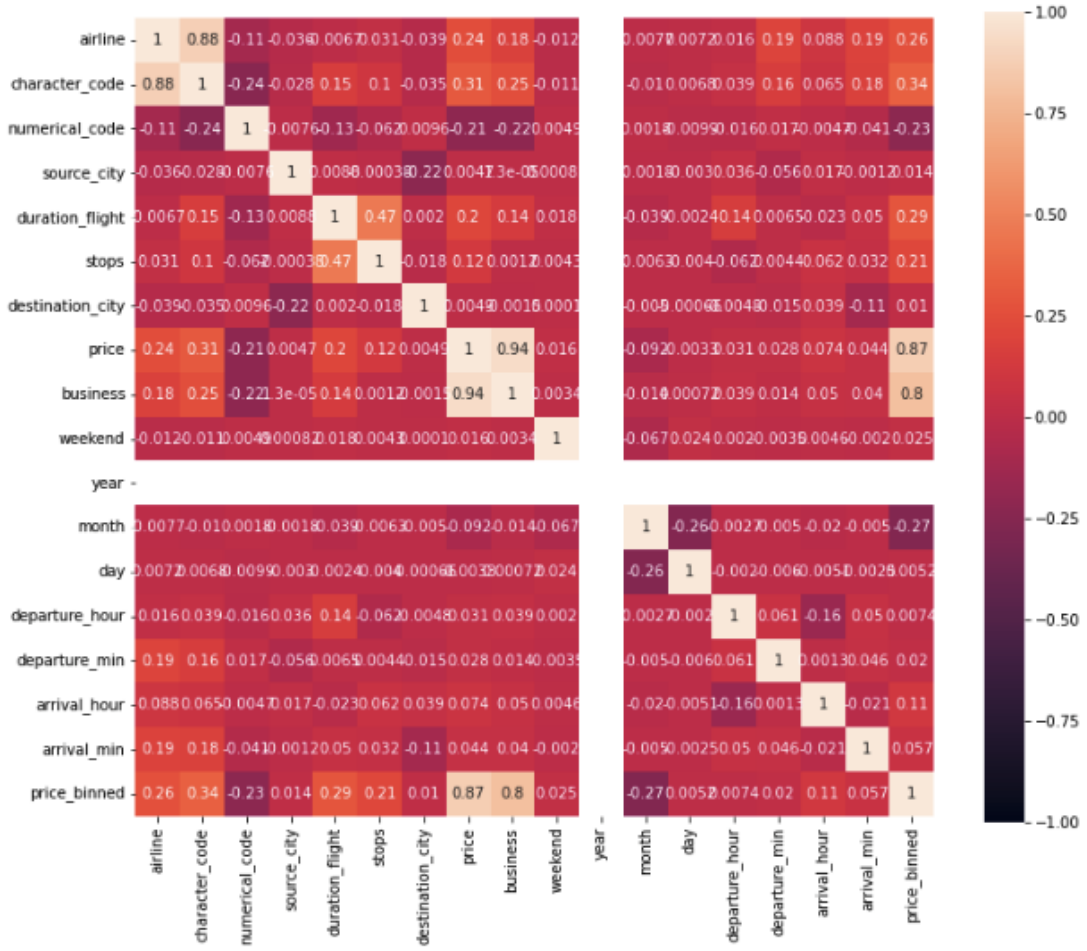

Figura 4.8: Correlação linear das features do modelo

De notar também que o year não apresenta correlação, pois só existe um valor único do year. Dado isto, decidiu-se também remover esta coluna.

```
year
2022    300255
```

4.4 Modelos

4.4.1 Técnicas de aprendizagem

Para construir o modelo de machine learning para fazer as previsões foi necessário começar por identificar quais as técnicas mais adequadas para o problema em questão. Neste sentido fez-se uma pesquisa sobre quais as técnicas de regressão mais populares para problemas de previsão do preço do voo e várias fontes referenciaram o XGBoost como o algoritmo mais comum.

Em problemas de regressão o algoritmo Random Forest também costuma apresentar bons resultados numa grande variedade de problemas, portanto também se usou inicialmente esta técnica.

Como os resultados das métricas eram melhores no caso do XGBoost do que no Random Forest decidiu-se fazer

o tratamento dos dados considerando este algoritmo. Esta parte é importante pois as técnicas de tratamento dos dados aplicadas dependem da técnica utilizada. Um exemplo disto foi o facto de não aplicar a técnica de normalização de valores pois esta técnica de aprendizagem não necessita desse tratamento.

Os modelos usados foram:

- **Decision Tree Regression**
- **Random Forest Regression**
- **XGBoost**

4.4.2 Partitioning

Em relação ao particonamento dos dados, na comunidade científica é muito comum utilizar uma porporção de 70/30 para o particionamento do dataset, no entanto este valor deve-se adequar ao modelo em questão, como tal, foram efetuados alguns testes com particionamentos semelhantes a este, sendo que as melhores métricas foram atingidas com a porporção 80/20 e portanto esta foi a utilizada no modelo.

No particionamento foi aplicada também a metodologia stratified sampling tendo em conta a coluna business, dado que se procura igualar o número de ocorrências de dados do tipo business(valor 1) e economy(valor 0), de modo a que o modelo não treine de forma desbalanceada. Apesar da utilização desta metodologia, como o dataset usado possui um número muito maior de entradas do tipo *economy* do que do tipo business foi verificada uma distribuição assimétrica deste parâmetro como se pode verificar na figura 4.9.

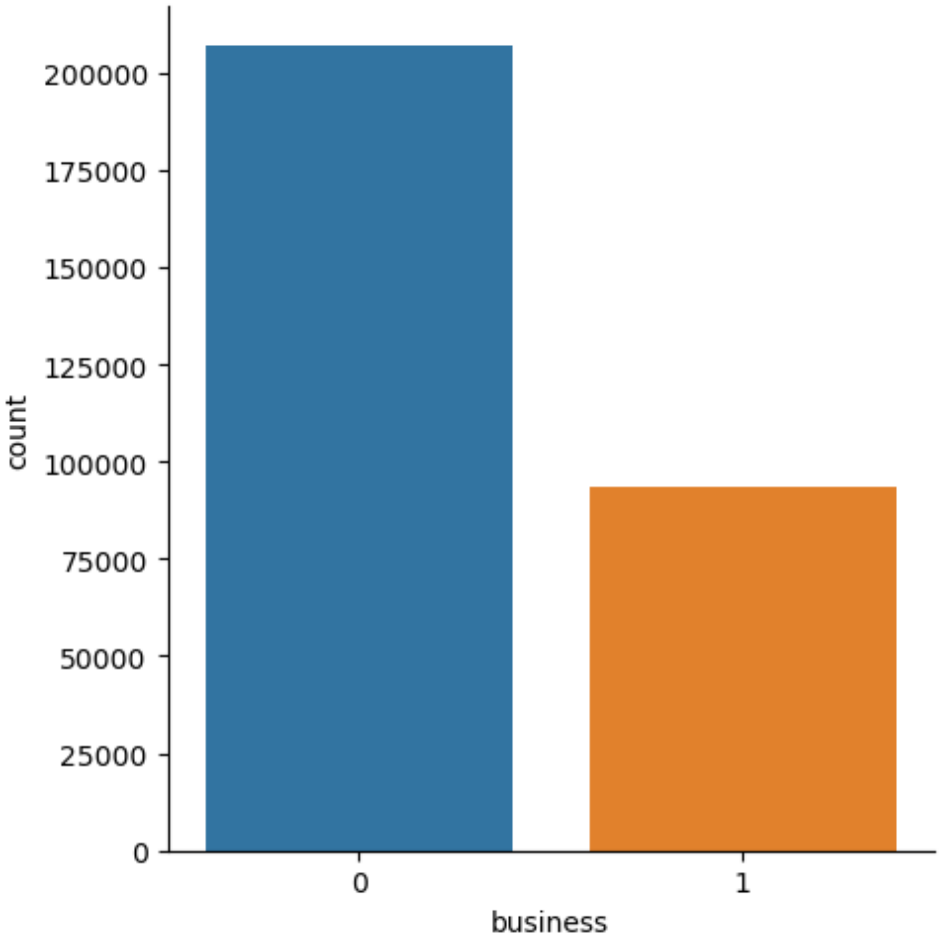

Figura 4.9: Número de ocorrências dos valores business e economy nos registos do dataset

Para contornar este desbalanceamento, foi então testado um "Equal Size Sampling"antes de efetuar o particionamento do dados. No entanto a introdução desta estratégia piorou os resultados do modelo. Decidiu-se portanto treinar o modelo sem "Equal Size Sampling".

É de notar que todo o particionamento foi aplicado recorrendo a uma random seed de 2022 com o objetivo de poder replicar os resultados a qualquer momento.

4.4.3 XGBoost

Tunning dos Hiperparâmetros

O algoritmo XGBoost tem associado múltiplos parâmetros de treino. De modo a obter os melhores resultados devemos escolher os valores dos parâmetros mais indicados. Como o número de parâmetros desta técnica é elevado começou-se por escolher aqueles que tinham mais influência. Realizamos uma pesquisa dos parâmetros mais importantes e após algumas experiências de treino do modelo escolheu-se os seguintes, que foram os que tiveram maior impacto positivo nas métricas de qualidade:

- Eta (Learning Rate)
- Maximum depth

Para optimizar estes parâmetros usou-se uma técnica de grid-search. O grid search recorre a diferentes combinações de todos os hyperparâmetros específicados e os seus valores e calcula a performance para cada um dos casos. O comando em questão é o **grid_result.best_params_** .

Os melhores valores para os parâmetros foram Eta=0.4 e Max depth=11 o que representa um valor de *R-Squared*= 0.993. Comparando com outras combinações podemos observar aumentos na casa dos 0.01, o que indica que esta técnica tem grande um grande impacto no modelo. Apesar de 0.01 não parecer muito à primeira vista, pequenos aumentos tem muito maior significado quando o valor de *R-Squared* está muito próximo de 1.

Cross Validation

Na construção de um modelo de machine learning um dos conceitos que é necessário ter em atenção é o overfitting. É importante que não ocorra overfitting para que o modelo possa generalizar bem dos dados de treino para dados que ainda não tenham sido vistos. Para evitar overfitting foi utilizada a técnica cross validation.

```
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=2022)
```

Este foi o comando usado para fazer as várias partições do dataset e escolheu-se 10 validações. O dataset tem muitos registos por isso decidiu-se não usar um valor grande de validações e para além disso este valor é recomendado como suficiente para vários tipos de problemas. Usou-se a random seed de valor 2022 e n_repeats 3.

O comando usado para agregar os resultados das previsões para as várias validações foi

```
#scores = cross_val_score(model, X, y, scoring='r2', cv=cv)
```

O cross validation foi usado com um modelo XGBoost e com os parâmetros ideais definidos na secção tunning dos hyperparâmetros. A métrica *R-Squared* manteve-se com o mesmo valor comparando com o caso sem cross

validation. Ainda assim, usamos esta técnica para dar mais robustez ao modelo na previsão de outros dados de teste.

Métricas de qualidade

Tal como já foi referido em algumas secções anteriores, a métrica definida como alvo foi a *R-Squared*. Todo o tratamento foi realizado com vista a maximização deste valor. Os resultados finais de várias métricas de qualidade podem ser vistos na figura 4.10.

```
R^2 :  0.9931153447498615
MAE : 916.18616107461
```

Figura 4.10: Métricas para o modelo XGBoost

Das métricas também é possível observar qual o erro em média na previsão (MAE): 1000. É necessário primeiro perceber o atributo price antes de entender este resultado. A moeda do atributo price é Indian Rupee e não Euros. Convertendo para Euros é possível ver que o valor médio do atributo price é 21000 Rupee corresponde a cerca de 250 Euros e o MAE é 1000 Rupee que corresponde a cerca de 12 Euros. Analisando este caso entendemos que o valor obtido para esta métrica é bastante satisfatório.

4.5 Resultados principais, análise crítica e trabalho futuro

Da exploração do *dataset* foi possível obter uma série de informações úteis não só para a empresa "Ease My Trip"como para os utilizadores da aplicação web:

- Da figura 4.2 é possível verificar que a grande maioria dos utilizadores compra bilhetes baratos (um valor entre 1105 e 13301.6 rupias indianas). A empresa pode com base nesta informação aplicar estratégias de marketing adequadas para este caso.
- Algumas das airlines contribuem muito pouco para a venda de bilhetes no site, StarAir e Trujet, enquanto outras como a Vistara e Air India representam mais de 60% dos bilhetes vendidos (ver figura 4.4). A empresa pode, com base nisto, rever se compensa manter ligação com as airlines que pouco contribuem com a venda de bilhetes.
- Os utilizadores podem usar as previsões do preço do bilhete para perceber o melhor momento para a compra do bilhete. O fornecimento desta informação aos utilizadores tem de ser considerada com cuidado pela empresa. Hipoteticamente, se a empresa disponibilizar as informações do preço dos bilhetes relativas aos anteriores, isto pode provocar uma mudança drástica de costumes no presente ano, o que poderá ter impacto nos custos e disponibilidade de recursos da airline. Para além disso disponibilizar previsões de preços que se verifiquem erradas no futuro, podem fazer com que os utilizadores se sintam enganados.

Como trabalho futuro existem uma série de aspectos de deveriam ser considerados:

- O modelo pode ficar vulnerável a eventos particulares de cada ano. Acontecimentos recentes como a guerra na Ucrânia e o covid são casos particulares de eventos que podem alterar drasticamente os preços dos bilhetes. Deve-se ter em conta estes factores quando se pretende aplicar modelos de machine learning em casos como o preço de bilhetes.
- Da exploração de dados verificou-se que apenas haviam registos relativos ao mês de Fevereiro e Março. Para fazer uma boa previsão para diferentes alturas do ano seria essencial pelo menos ter registos representativos de um ano completo. Deveríamos então contactar a empresa "Ease My Trip"e recolher mais dados.

- Uma das explorações que pretendemos fazer foi a identificação de época alta ou época baixa. Contudo não o fizemos pela razão do ponto anterior, apenas tinhamos registos relativos ao mês de Fevereiro e Março.
- Neste trabalho as métricas definidas como objectivo de minimização ou maximização foram esocolhidas pelo grupo. Num caso real os objectivos podem já estar bem definidos pelo contexto do problema ou pela empresa e portanto as métricas a analisar seriam diferentes.
- Qualquer modelo de machine learning apenas tem sucesso quando é aplicado na vida real. Neste trabalho estamos a usar dados de treino e teste estáticos. Um dos passos futuros deveria ser o *deployment* do modelo.
- Da análise do negócio percebeu-se que o intervalo de dias entre a data de compra do bilhete e a data do voo tem influência no preço. Deveríamos pedir à empresa que também fossem fornecidos estes dados com o objectivo de melhorar o modelo.
- Poder-se-ía tentar obter a data e hora de registo do bilhete no site, de modo a obter conhecimento útil como a afluência de utilizadores no site. Isto pode ter benefícios para a empresa no sentido de balancear melhor os recursos computacionais necessários para manter a aplicação web em funcionamento, consoante exista muita ou pouca afluência de utilizadores.

O modelo que revelou melhores resultados foi o XGBoost e o valor da métrica definida como objetivo de maximização, o *R-Squared*, teve um bom resultado 0.991. A métrica *MAE* também teve um resultado bastante satisfatório de 1000 Rupee, que corresponde a cerca de 12 Euros, sendo que o valor médio do preço do bilhete era de 21000 Rupee que corresponde a cerca de 250 Euros.

5 Conclusão

Durante o presente trabalho, entendeu-se que um tratamento adequado é uma parte fulcral para obter bons modelos de previsão. Não só é necessário ter em atenção esta fase, como todas as fases são importantes e estão relacionadas. Por exemplo, o entendimento da lógica de negócio (contexto, atributos, termos técnicos) é uma competência necessária para fazer um bom tratamento dos dados. Para além disto, compreendeu-se melhor sobre que tipos de tratamento se deve fazer utilizando certos algoritmos de machine learning.

Os aspectos mais desafiantes neste trabalho foram a identificação das técnicas de tratamentos dos dados que têm mais efeito no modelo, bem como a percepção relativamente a quando se deve evitar determinadas técnicas de tratamento.

Por outro lado, o presente trabalho permitiu ganhar experiência na área de machine learning, com o objectivo de aplicar os conhecimentos adquiridos durante as aulas T e TP em problemas semelhantes de regressão ou classificação. O facto da escolha do dataset de Flights ter sido de regressão permitiu ao grupo diversificar os seus conhecimentos, pelo que conferiu um impacto positivo na aprendizagem e prática dos conteúdos lecionados.