

# IPBeja

INSTITUTO POLITÉCNICO  
DE BEJA

Escola Superior de Tecnologia e Gestão  
Licenciatura em Engenharia Informática

## Sistema de apoio à decisão para o turismo no Alentejo

Sistema de apoio à decisão com base em *Data Mining* para o turismo  
no Alentejo

Gonçalo Amaro

Pedro Tomás

Vítor Abreu

Beja, 16 de Fevereiro de 2022



**INSTITUTO POLITÉCNICO DE BEJA**  
**Escola Superior de Tecnologia e Gestão**  
**Licenciatura em Engenharia Informática**

## **Sistema de apoio à decisão para o turismo no Alentejo**

**Sistema de apoio à decisão com base em *Data Mining* para o turismo  
no Alentejo**

Gonçalo Amaro  
Pedro Tomás  
Vítor Abreu

Orientado por :

Doutora Isabel Brito, IPBeja

Relatório de Projecto Final, realizado na cadeira de Projecto Integrado, apresentado na  
Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja



# Resumo

*Sistema de apoio à decisão para o turismo no Alentejo*

*Sistema de apoio à decisão com base em Data Mining para o turismo no Alentejo*

Tendo em vista uma sustentável, duradoura e benéfica relação entre o turismo e todos os estabelecimentos locais, como hotéis, restaurantes e atracções, para além do património histórico-cultural, há que seguir uma estratégia. Existem várias estratégias, porém a tomada na elaboração deste trabalho foi a monitorização de fluxos de visitantes. Esta monitorização pode ser realizada através de um sistema de informação como o TripAdvisor, Zomato e Booking, que foram os escolhidos na elaboração do trabalho, o armazenamento dos dados, ou seja, o desenvolvimento de uma base de dados em SQL, o processamento de dados usando técnicas para normalizar e analisar textos, assim como a extracção das "keywords" e análise de opinião que seriam mais tarde úteis na elaboração de gráficos como medida para uma fácil visualização dos resultados acerca dos resultados obtidos, indicando muitos aspectos interessantes acerca das preferências turísticas dentro dos patrimónios.



# Abstract

*Decision support system for tourism in Alentejo*

*Decision support system based on Data Mining for tourism in Alentejo*

In view of a sustainable strategy, historical establishments and cultural establishments between tourism and all places, such as hotels, in addition to historical and cultural establishments, a strategy must be followed. Several strategies, however the elaboration in the elaboration of this workflow was monitoring. This monitoring can be carried out through an information system such as TripAdvisor, Zomato and Booking, which were chosen in the elaboration of the work, the storage of data, that is, a database in SQL, the processing of data using techniques to normalize and study, as well as the extraction of "key words" and opinion analysis that will later be useful in the elaboration of measures for an easy visualization of the results about the obtained results, often interesting about the relevant issues obtained within the heritages.



## *Agradecimentos*

O relatório de projecto final da cadeira de Projecto Integrado decorre de uma experiência que exigiu trabalho e esforço para alcançarmos os objectivos pretendidos. Como tal, agradecemos a disponibilidade, acompanhamento atento e colaboração demonstrados pela professora Isabel Brito, agradecemos também o seu apoio, incentivo, confiança e essencialmente por nos ter guiado para uma melhor execução do trabalho.



# Índice



# Índice de Figuras



# Índice de Tabelas



# Capítulo 1

## Introdução

### 1.1 Objectivo do trabalho

Este trabalho tem como principal objectivo a monitorização de fluxos de visitantes para identificar boas práticas, tendo sempre foco na criação de uma benéfica, sustentável e duradoura relação entre o turismo e todos os estabelecimentos locais, como hotéis, restaurantes e atracções, para além do património histórico-cultural. Para essa monitorização ser realizada, será necessário desenvolver um sistema de informação que recolha, armazena, processa e comunica dados e informação sobre as visitas ao património dos turistas nacionais e estrangeiros. Para que a essa monitorização seja realizada devidamente, será necessário seguir algumas etapas, das quais seriam:

1. A recolha de dados sobre as visitas dos turistas nacionais e estrangeiros ao património cultural de Beja, destacando-se dessa mesma recolha, as atracções, hotéis e restaurantes, e tendo como origens as fontes, *"TripAdvisor"*, *"Booking"*, *"Zomato"*. Para que a recolha de dados fosse realizada das devidas fontes foi necessário recorrer ao conceito de *webscraping*;
2. O armazenamento de dados numa base de dados SQL, para mais tarde facilitar o gestão de toda a informação para as seguintes etapas, nomeadamente para elaboração de gráficos temporais usando o *software PowerBI*;
3. O pré-processamento, extracção de *keywords* e *sentiment analysis*, iniciando-se com a normalização dos dados, em seguida extracção de *keywords* e depois, novamente o pré-processamento mas desta vez da *sentiment analysis*;
4. A elaboração de gráficos usando a biblioteca *Matplotlib* e o *software PowerBI*;

O presente relatório encontra-se organizado na seguinte forma: na secção 2 descreve-se a fase de investigação; na secção 3 é descrito como foi realizado o processo de *webscrapping* nos *websites* mencionados, assim como a estratégia pensada e dividida pelos elementos do grupo; na secção 4 é explicado o processo de normalização/formatação desenvolvido; na

## 1. INTRODUÇÃO

---

secção 5 mostramos os métodos escolhidos na extracção das *keywords*; na secção 6 falamos de todo o processo por detrás dos *sentiment analysis*; na secção 7 de como foram gerados os gráficos no decorrer da elaboração do projecto; na secção 8 de como e do porquê de termos reorganizado o projecto e também acerca da base de dados gerada; na secção 9 finalmente começamos a analisar os dados obtidos e por fim, na secção 10 são apresentadas as conclusões relativas à elaboração do presente trabalho. Ao auxílio da criação deste documento foram usados os relatórios de progresso criados anteriormente.

### 1.2 Descrição e motivação

O tema deste trabalho é bastante interessante do ponto de vista turístico e mais tarde financeiro, já que a partir dele é possível analisar as opiniões dos turistas (nacionais ou estrangeiros) e baseando-se nisso, ter noção de quais pontos turísticos, hotéis ou restaurantes cativam mais a atenção do público e também estudar os pontos fortes e fracos de cada um deles. É possível também verificar se determinados locais têm tendências a manter, aumentar ou diminuir o número de turistas com o decorrer dos anos. Estes valores são bastante importantes para um país como Portugal que usa o turismo como forte fonte de rendimento, e uma vez que no presente trabalho o foco é o Alentejo, que é altamente movimentado nas épocas balneares, mais importante a análise das informações recolhidas se tornam.

### 1.3 Divisão de tarefas

Uma vez que o trabalho era realizado em grupo, foi decidido previamente que existiria uma etapa onde ocorreria a separação de tarefas por cada elemento do grupo, que seria ao realizar o método *60%, 20%, 20%*, distribuindo o trabalho pelos elementos via o tempo disponível. Para além das tarefas divididas entre os elementos do grupo, foi utilizado uma ferramenta para gerir as tarefas de cada elemento respectivamente, que foi o *Trello*. O *Trello* é tão simples como uma ferramenta de gestão de projectos, é uma plataforma versátil e pode ser usada para acompanhamento de tarefas pessoais ou para organizar projectos que envolvam equipas/grupos com maior número de pessoas. Para além do *Trello* o grupo também utilizou o *GitHub* como ferramenta de gestão de versões do trabalho e repositório do mesmo, sendo a principal ferramenta no controlo de versões dos trabalhos realizados por cada elemento do grupo. Foram também utilizadas outras tecnologias no decorrer do trabalho como algumas bibliotecas específicas para algumas partes, como a *BeautifulSoup4* ou o *Yake!* ou até mesmo a ferramenta *PowerBI* que já foi mencionado, porém estas serão faladas mais adiante no decorrer do trabalho.

## 1.4 Ambientes virtuais *Python*

Neste projecto usámos ambientes virtuais *Python*. Um ambiente virtual é uma forma de ter várias instâncias paralelas do interpretador de *Python*, cada uma com diferentes conjuntos de pacotes e diferentes configurações. Cada ambiente virtual contém uma cópia do interpretador de *Python*, incluindo cópias dos seus utilitários de suporte como o *pip*. Estes contêm também uma zona para instalação de pacotes/bibliotecas localmente (dentro do ambiente virtual), sendo esta a razão principal pela qual foi decidido usá-los.



## Capítulo 2

# Identificação dos requisitos

### 2.1 Objectivo

O objectivo deste capítulo será esclarecer alguns conceitos mais teóricos associados ao trabalho realizado. Assim sendo, foram obtidos todos os dados de *posts* e comentários relacionados com *providers* de acesso, entretenimento, refeições e estadia directamente ligados ao património cultural do Alentejo. Estes a serem analisados e classificados, criando assim um modelo de possíveis sentimentos e procura que o comércio local tem o interesse em fornecer aos visitantes. O foco principal foram os *posts* e comentários em português.

### 2.2 *Websites* escolhidos

Os *websites* seleccionados assim como os métodos de obtenção de dados, e também de analisar os mesmos foi decidida previamente pelo grupo e todos os elementos decidiram usar as mesmas ferramentas para cada função que lhes fora determinado. Após exaustiva procura pelos *websites* preferenciais e discussão entre todos os elementos do grupo, para a utilização de um conjunto deles onde fosse possível se realizar uma boa análise e obtenção dos dados, com especial foco na linguagem portuguesa, reunimos alguns possíveis candidatos que serão abordados no ponto seguinte.

Foi decidido também que dentro das opções que serão referidas em seguida, teremos preferência na utilização de um em específico, uma vez que temos intenções de dar alguma prioridade às informações relacionadas com o turismo rural alentejano, sendo assim e devido a uma maior procura e número de resultados que o *website* oferecia, inicialmente iríamos utilizar o *website* *TripAdvisor* como primeira opção.

## 2.3 Sites pesquisados

Foram usados os sites:

- *TripAdvisor*
- *Booking*
- *Zomato*
- *Google Maps*

O foco principal é o *TripAdvisor* visto que este oferece a maior variedade de conteúdo (hotéis, restaurantes e outros estabelecimentos), no entanto como uma plataforma é pouco, decidimos adicionar *Booking* e *Zomato* à lista para uma maior e mais ampla rede de hotéis e restaurantes.

Foi também considerado o *Google Maps*, mas este apresentou um novo set de problemas que vão ser descritos já de seguida.

## 2.4 (Im)possibilidade de uso de APIs

Em nenhum dos *websites* testados foi observada uma facilidade na obtenção de acesso às suas *APIs*, apenas alguns (3/4) ofereceram acesso à documentação da(s) mesma(s) facilmente. A maioria requer um contacto, que foi tentado e continuou sem resposta durante quase todo o processo de elaboração do trabalho(contactos iniciados por Amaro, entre dia 26 e 29 de Outubro, e dia 11 de Novembro de 2021).

Dentro dos *websites* que oferecem documentação foi observado que todos subdividem os seus serviços de *API* em 3 ou 4 *APIs* para casos de uso específicos (reservas, dados, etc) em vez de uma com *endpoints* que oferecem solução para todos os casos.

A anomalia aqui é o *Google Maps* que é o único que facilita o acesso à *API* (mas paga, temos de ver os créditos disponíveis no *Cloud Platform*), e de elevada dificuldade em *scraping* pelo óbvio.

## 2.5 Alternativas

Sendo que é impossível o uso das *APIs* (que facilitariam o trabalho) temos de recorrer a outras técnicas para obter os dados.

### 2.5.1 Web-crawling VS web-scraping

*Web crawling*, também conhecido como Indexação, é usado para indexar as informações na página usando bots também conhecidos como *trackers*. O *tracker* é essencialmente o que os motores de busca fazem. É uma questão de visualizar uma página como um todo e

indexá-la. Quando um *Bot* rastreia um *website*, ele passa por todas as páginas e todos os *links*, até a última linha do *website*, em busca de qualquer informação.

O *web scraping*, também conhecido como extração de dados da *web*, é semelhante ao *web crawling*, pois identifica e localiza os dados de destino das páginas da *web*. A principal diferença é que, com o *web scraping*, sabemos quem identificou o conjunto de dados exactamente, por exemplo, uma estrutura de elemento *HTML* para páginas da *web* que estão a ser corrigidas, da qual os dados precisam ser extraídos.

Com isto visto, *web scraping* é o nosso alvo, visto que minimiza lixo e é direcccionado. No entanto devemos olhar para:

### Vantagens de *web scraping*

1. Mais rápido: É possível manusear grandes quantidades de dados que poderiam levar dias ou semanas a serem processados através do trabalho manual, com o uso do *scraping* podemos reduzir substancialmente o esforço e aumentar a velocidade de decisão;
2. Confiável e consistente: Ao fazer o trabalho manual é muito fácil de haver erros, por exemplo, erros tipográficos, informações esquecidas ou inserção nas colunas erradas. O uso do *web scraping* garante consistência e a qualidade dos dados;
3. Ajuda a reduzir a carga de trabalho;
4. Menor custo: Uma vez implementado o *scraping*, o custo total da extração de dados é significativamente reduzido, especialmente quando comparado ao trabalho manual;
5. Manutenção básica: Fazer o *scraping* de dados geralmente não requer muita manutenção.

### Desvantagens de *web scraping*

1. Baixa proteção: Se os dados na *web* são protegidos, o uso do *scraping* também pode se tornar um desafio e aumentar os custos;
2. Dados estruturados: Não vai ser possível fazer scraping a 1000 *websites* diferentes pois cada *website* tem uma estrutura completamente diferente. Será necessário haver alguma estrutura básica que seja diferente em determinadas situações.

#### 2.5.2 Necessidade de *Web Scraping*

Com o acima dito, é necessário recorrer a soluções de *Web Scraping*.

A comunidade internauta reparou no mesmo, visto que em reacção ao observado existem dezenas de projectos e tutoriais de *Web Scraping* das variadas plataformas de turismo e reservas. Infelizmente, as mesmas não ajudam no processo e cada uma tem um forma de actuação bastante diferente.

## 2.6 Bibliotecas de *Python* para *Web Scraping*

Para fazer *Web Scraping* vamos usar *Python*, pela sua facilidade de uso e multifaceta “*Development speed is more important than execution speed*”. Com *Python* também temos as opções de criar cadernos *Jupyter* onde o próprio código e os resultados são “encadernados” com parágrafos de texto fazendo o próprio projecto o seu pequeno relatório de progresso e resultados; como também a criação de ambientes virtuais (*containers*), onde os pacotes usados ficam registados e instalados localmente, garantindo assim a portabilidade.

Para tal linguagem existem 5 grandes bibliotecas para a resolução deste caso:

- *Requests*
- *BeautifulSoup*
- *Scrapy*
- *lxml*

Cada uma tem diferentes vantagens e desvantagens. Caso nenhuma destas tivesse resultado, teríamos usado Selenium, que é uma biblioteca mais completa e poderosa que as listadas, visto que é uma ferramenta de testes de automação. Porém tem um nível de complexidade maior e requer um *setup* inicial maior e mais trabalhoso, requer *WebDrivers* para a execução das tarefas, pode ser complicada com Firefox, sendo preferencial usar *Chromium-based Browsers* como o Chrome e o novo Edge (necessário ainda o *ChromeDriver* e o *EdgeDriver*).. Tentaremos evitar essa, a todo o custo, pela sua complexidade e extras desnecessários às nossas necessidades e custo temporal do *setup* inicial.

Para cada website pode ser necessário usar bibliotecas diferentes por necessidade ou por obtenção de informações/blog posts/etc... que facilitem ou melhorem o output desejado.

### 2.6.1 Tratamento de output

Os outputs do conteúdo *scraped* podem vir em *.xml* ou *.csv* (ou outras mas essencialmente essas duas). Tentámos ao máximo usar *.csv*, e transformar qualquer *.xml* em *.csv*, visto que um maior número de ferramentas gráficas (Excel, PowerBI, etc...) e/ou bibliotecas de *Python* (*Pandas*, *matplotlib*, etc) para a análise de dados tratam melhor ficheiros separados por vírgulas.

Foi também feita uma análise e extração de *keywords* nos textos das descrições e *reviews*. Para tal existem variados algoritmos que podemos usar, alguns "clássicos" outros até de *machine learning*.

Inicialmente todas as informações (dados e meta-dados) são dados como relevantes, após consideração e ponderação durante análises iniciais do decorrer do estudo poderemos descartar dados que não consideremos relevantes. No entanto nada nos impede de tentar prever ou imaginar quais esses serão e posteriormente avaliar o nosso julgamento para ver o que foi aprendido.

## 2.7 Análise

### 2.7.1 Algoritmos de mineração de texto

Os algoritmos de análise de texto podem ser considerados ferramentas de mineração de texto, isto é, o processo de descoberta de conhecimento potencialmente útil e inicialmente desconhecido, ou seja, a extracção de conhecimento útil utilizando bases textuais.

O processo de mineração de texto é dividido em quatro etapas bem definidas:

- Selecção;
- Pré-processamento;
- Mineração;
- Assimilação.

Na selecção, os documentos relevantes devem ser escolhidos e mais tarde processados. No pré-processamento ocorrerá a conversão dos documentos em uma estrutura compatível com minerador, bem como ocorrerá um tratamento especial do texto. Na mineração, o minerador irá detectar os padrões com base no algoritmo escolhido. E por fim, na assimilação, os utilizadores irão utilizar o conhecimento gerado para apoiar as suas decisões.

Por outras palavras este processo de mineração de texto pode também ser chamado de *webscrapping* que é o utilizado no nosso trabalho.

A etapa pré-processamento pode ser dividida em quatro tarefas:

- Remover *stopwords*;
- Compilação;
- Normalização de sinónimos;
- Indexação.

Na etapa de remoção de *stopwords* os termos com pouca ou nenhuma relevância para o documento serão removidos. São palavras auxiliares ou conectivas, ou seja, não são discriminantes para o conteúdo do documento.

Na etapa seguinte, compilação, realiza-se uma normalização morfológica, ou seja, as palavras são reduzidas ao seu radical, serão combinadas em uma única representação. A radicalização pode ser efectuada com o auxílio de algoritmos de radicalização, sendo os mais utilizados o algoritmo de *Porter (Porter Stemming Algorithm)* e *algoritmo de Orengo (Stemmer Portuguese ou RLSP)*.

Após a compilação, na etapa de normalização de sinónimos, os termos que possuem significados similares serão agrupados em um único termo, por exemplo, as palavras ruído, tumulto e barulho serão substituídas ou representadas pelo termo barulho.

## 2. IDENTIFICAÇÃO DOS REQUISITOS

---

E, por fim, na etapa indexação atribui-se uma pontuação para cada termo, garantindo uma única instância do termo no documento. No processo de atribuição de pesos devem ser considerados dois pontos:

- Quanto mais vezes um termo aparece no documento, mais relevante ele é para o documento;
- Quanto mais vezes um termo aparece na coleção de documentos, menos importante ele é para diferenciar os documentos.

### 2.7.2 Algoritmos de *machine learning*

Os algoritmos de *machine learning* são partes de código que ajudam as pessoas a explorar, analisar e localizar o significado em conjuntos de dados complexos.

Os algoritmos de *machine learning* utilizam parâmetros baseados em dados de preparação, um subconjunto de dados que representa o conjunto maior. À medida que os dados de preparação se expandem para representar o mundo de forma mais realista, o algoritmo calcula resultados mais precisos.

Algoritmos diferentes analisam os dados de diversas formas. Geralmente, são agrupados consoante as técnicas de *machine learning* para as quais são utilizados:

- Aprendizagem supervisionada;
- Aprendizagem não supervisionada;
- Aprendizagem por reforço.

#### Aprendizagem supervisionada

Na aprendizagem supervisionada, os algoritmos fazem previsões com base num conjunto de exemplos etiquetados fornecidos por si. Esta técnica é útil quando sabe como deverá ser o resultado. Por exemplo, fornece um conjunto de dados que inclui populações de cidades por ano nos últimos 100 anos e deseja saber qual será a população de uma cidade específica dentro de quatro anos. O resultado utiliza etiquetas que já existem no conjunto de dados: população, cidade e ano.

#### Aprendizagem não supervisionada

Na aprendizagem não supervisionada, os pontos de dados não são etiquetados. O algoritmo etiqueta-os ao organizar os dados ou ao descrever a sua estrutura. Esta técnica é útil quando não sabe como deverá ser o resultado. Por exemplo, fornece dados de cliente e deseja criar segmentos de clientes que gostam de produtos semelhantes. Os dados que está a fornecer não são etiquetados e as etiquetas no resultado são geradas com base nas semelhanças descobertas entre os pontos de dados.

### Aprendizagem de reforço

A aprendizagem por reforço utiliza algoritmos que aprendem com resultados e decide a acção a realizar em seguida. Após cada acção, o algoritmo recebe comentários que o ajudam a determinar se a escolha feita foi correta, neutra ou incorrecta. Por exemplo, se estivermos a criar um carro autónomo, queremos que este cumpra a lei e mantenha as pessoas seguras. À medida que o carro ganha experiência e um histórico de reforço, aprende a permanecer dentro da faixa, a não ultrapassar o limite de velocidade e a travar quando encontrar peões.

Existem muitos tipos diferentes de algoritmos de *machine learning*. Contudo, por norma, os casos de utilização destes algoritmos enquadraram-se numa destas categorias.

- Algoritmos de classificação de duas classes (binários) dividem os dados em duas categorias. São úteis para perguntas com apenas duas respostas possíveis mutuamente exclusivas, incluindo perguntas de sim/não;
- Algoritmos de classificação multi-classe (multinomial) dividem os dados em três ou mais categorias. São úteis para perguntas com três ou mais respostas possíveis mutuamente exclusivas;
- Algoritmos de detecção de anomalias identificam os pontos de dados que estão fora dos parâmetros definidos para o que é considerado “normal”
- Algoritmos de regressão prevêem o valor de um novo ponto de dados com base em dados históricos;
- Algoritmos de séries temporais mostram as alterações a um determinado valor ao longo do tempo. Com a análise e a previsão de série temporal, os dados são recolhidos a intervalos regulares ao longo do tempo e utilizados para fazer previsões e identificar tendências, sazonalidade, periodicidade e irregularidade;
- Algoritmos de *clustering* dividem os dados por vários grupos ao determinar o nível de semelhança entre os pontos de dados;
- Algoritmos de classificação utilizam cálculos de previsão para atribuir dados a categorias predefinidas.

#### 2.7.3 Decisão sobre o tipo de algoritmo

Após a grande análise dos tipos e subtipos de algoritmos de mineração de texto (análise de texto), a escolha foram os algoritmos de *machine learning* com aprendizagem não supervisionada para a execução da nossa análise; a questão está em qual serão usados visto que muitos deles para os nossos casos poderão ter de ser sujeitos a pré-processamento; o que removeria as nossas requeridas dimensões de análise textual. No entanto vai continuar a haver pré-processamento como algoritmos de redução de dimensionalidade, a diferença

## 2. IDENTIFICAÇÃO DOS REQUISITOS

---

comparado com a frase anterior é quão o pré-processamento não afectará negativamente os resultados e possíveis associações.

### Algoritmos considerados

Dos variados algoritmos vistos e disponíveis na internet ou em bibliotecas de *Python* (como *SciKitLearn*, *Tensorflow*), tomamos a decisão de considerar os seguintes algoritmos como candidatos a uso e/ou pertencentes aos grupos de algoritmos usados para comparação de resultados:

- *LDA (Latent Dirichlet Allocation)*: um modelo de distribuição gaussiana, muito usado por empresas de software sobre *feedback* e *bug reports* para associação de resultados do *QA*;
- *Naïve Bayes [mbn1]*: uma família de "classificadores probabilísticos" simples baseados na aplicação do teorema de Bayes com suposições de independência fortes (ingênuas) entre os recursos;
- *K-Means Clustering*: muito usado para fazer *clusters* de *keywords* em redes sociais;
- *KNN (K-Nearest Neighbor)*: usado para agrupar dados relacionais com os contactos, relatórios, correspondência e *emails* em empresas;
- *SVM (Support Vector Machines)*: este é usado nos mesmos lugares que regressões lineares, porém mais rápido ou poderoso, é usado para agrupar pontos como texto com imagens ou tópicos de texto em sites de vendas de 2<sup>a</sup> mão.

No entanto existe um algoritmo de *machine learning* semi-supervisto (aprendizagem não supervista mas ele faz a sua auto-supervisão; logo é questão de semântica). Este é o:

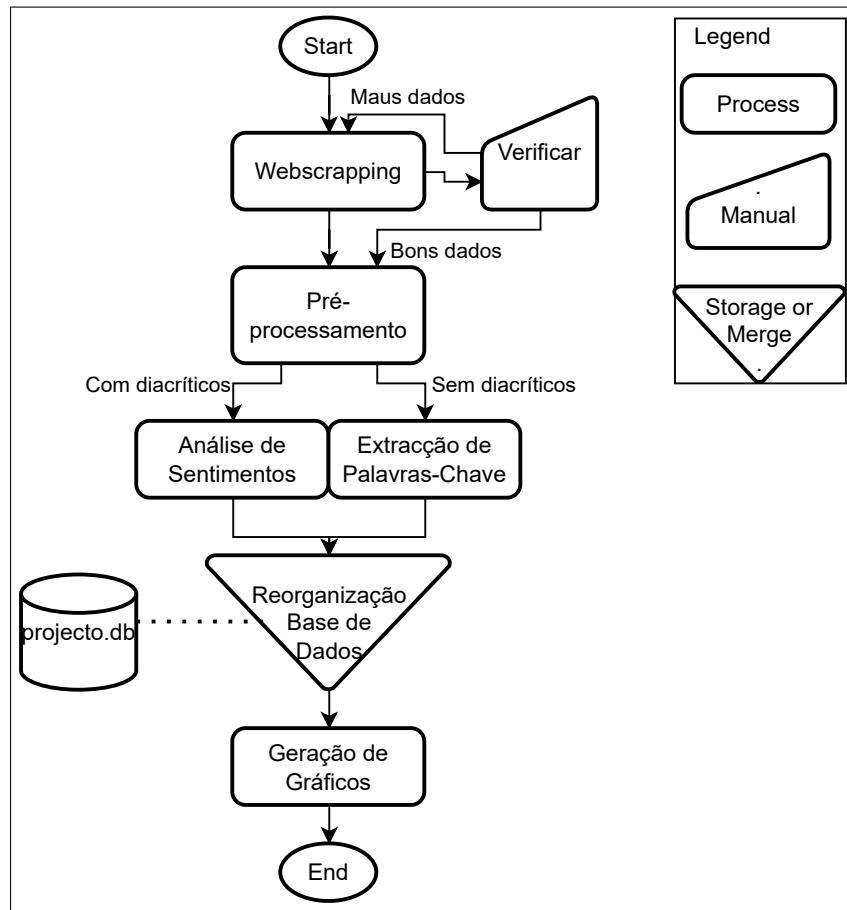
- *LSTM (Long-short term memory)*: que é um tipo de RNN (rede neural recorrente) com ou sem um *layer CNN* (um *layer* de convulsão).

Pode-se notar que aqui foram escolhidos algoritmos de *machine learning* já comuns a grupos que realizaram tarefas semelhantes e que são algoritmos simples e rápido não sendo mais um algoritmo de uma família de algoritmos (mais complexos ou não, mas que têm muita variedade), tais como redes neurais (à exceção do RNN LSTM), algoritmos genéticos ou algoritmos lineares (como regressões lineares).

Porém no decorrer do trabalho o algoritmo LSTM não funcionou da melhor maneira, e optámos por outras técnicas que serão explicadas, justificadas e demonstradas mais adiante no relatório.

## 2.8 Resumo dos Passos de Execução

A seguinte tabela descreve os passos de actuação.





# Capítulo 3

## Webscraping

### 3.1 Planeamento

#### 3.1.1 Divisão de Tarefas

A divisão de tarefas decidida foi a distribuição de cada um dos três *websites* por cada um dos elementos do grupo, pela ordem de dificuldade em congruência linear com o tempo extra-curricular disponível de cada elemento.

#### 3.1.2 Tecnologias Usadas

Na realização do *webscraping* foi desenvolvido um ambiente virtual de *Python 3* para realizar os *scripts* que iriam recolher as informações.

Como forma de organizar todos os pacotes GitHub e possíveis actualizações de bibliotecas dentro do código também foi gerado um ficheiro *requirements.txt* que actualizamos e usamos sempre que um dos elementos do grupo instale novos pacotes ao realizar o seu trabalho.

A linguagem optada para a construção dos *scripts* foi o *Python* já que é uma das mais acessíveis linguagens de programação disponíveis devido à sua simples *syntax* e também pela vasta quantidade de bibliotecas disponibilizadas, das quais temos uma dúzia que são bastante úteis para a realização deste projecto.

Para finalizar, todos os ficheiros foram guardados em formato *.csv* uma vez que é um formato que é aceite e nos facilita a manipulação de dados (ETL), a alimentação dos dados ao algoritmo de *machine learning*, e pode ser usado com ferramentas de análise e geradores de tabelas (como o PowerBI).

#### Ambientes Virtuais de *Python*

Neste projecto usamos Ambientes Virtuais de *Python*. Um ambiente virtual é uma forma de ter várias instâncias paralelas do interpretador de *Python*, cada uma com diferentes

### 3. WEBSCRAPING

---

conjuntos de pacotes e diferentes configurações.

Cada ambiente virtual contém uma cópia discreta do interpretador de *Python*, incluindo cópias dos seus utilitários de suporte como o *pip*. Estes contêm também uma zona para instalação de pacotes/bibliotecas localmente (dentro do ambiente virtual), sendo esta a razão principal pela qual foi decidido usá-los.

Tendo introduzido a razão, consegue-se perceber o óbvio: sendo este um trabalho de grupo e que posteriormente poderá ser testado pelos docentes ou futuros alunos, ao usar ambientes virtuais podemos fazer *pip freeze* para um ficheiro de texto do qual facilita a portabilidade e transmissão de requerimentos do projecto.

Para a criação destes ambientes virtuais foi instalado o *virtualenvwrapper* o qual traz uma *fork* com extensões úteis do *virtualenv* como dependência e um *set* de extensões para o mesmo.

#### Bibliotecas de *Python*

Como dito previamente, na explicação pelo qual o uso de *Python*, foi referida a grande quantidade de bibliotecas que nos são facilmente fornecidas pelo *pip*.

Dentro deste repositório existe (perto de) uma dúzia de bibliotecas que nos permitem facilmente completar as nossas tarefas deste projecto. Dessa dúzia, para esta etapa, foram usadas:

- *BeautifulSoup4*, uma biblioteca que facilita a extracção de informações de páginas da web, fornecendo expressões para iterar, pesquisar e modificar a árvore de análise;
- *lxml*, uma biblioteca *Python* que permite fácil manuseio de arquivos *XML* e *HTML*;
- *requests*, uma biblioteca *HTTP* elegante e simples para *Python*, construída de raiz para ser fácil de usar;
- *pandas*, uma ferramenta de manipulação e análise de dados de código aberto rápida, poderosa, flexível e fácil de usar;
- *jupyter*, um meta-pacote o qual traz (como dependências) o sistema *Jupyter* (em especial os cadernos), o *kernel IPython* e outros.

Com estes pacotes temos um mapa de actuação para esta etapa de projecto (de webscraping): abrir um ambiente virtual (e instalar bibliotecas), abrir um caderno de *Jupyter*, importar as bibliotecas das quais usámos *request* para ir buscar a nossa página, fazer *parsing* da página via *lxml*, criar um objecto *Soup* com o conteúdo *parsed*, fazer *scraping* e iterar pelos *scrapes* dos quais criámos *dataframes* de *pandas* e exportámos os mesmos em *.csv* para uso futuro. Nas secções seguintes será explicado com mais detalhe de acordo com o *website* em questão.

## 3.2 Booking

### 3.2.1 Estratégia

Para a realização do *webscraping* no *website* da *Booking.com*, inicialmente foi necessário a filtragem pelos hotéis apenas na localidade de Beja [yt1], uma vez ser o local que o grupo em conjunto decidiu optar para realizar todas as pesquisas num sítio em comum. Após ter o *Booking* a apresentar todos os resultados para os hotéis de Beja, foi recolhido o link que redireciona especificamente para esses resultados. Para aceder às informações específicas de cada elemento da página e mais tarde aceder aos mesmos para retirar a informação pretendida, foi usado a ferramenta de *inspeccionar a página* e assim descobrir os nomes das classes e todos os outros elementos que continham conteúdo importante para o projecto [yt1], como o nome dos hotéis, preço, classificação, número de comentários e alguns outros detalhes que pudessem ser úteis.

Em seguida foi necessário realizar o *webscraping* das *reviews* de cada hotel, a realização desta parte foi um pouco mais difícil uma vez que para as *reviews* serem bem recolhidas era fulcral que o *webscraping* fosse realizado usando outro link [yt1], ou seja, foi retirado do *website* o prefixo de um novo link que seria o *reviews* e baseando nos hotéis já retirados foi colocado o nome de cada um à frente do mesmo. Criando assim um novo link que seria usado na realização do *webscraping*. Após a criação de um novo link para cada hotel, os processos foram semelhantes aos anteriormente feitos.

Para finalizar, os resultados foram todos guardados em ficheiros *.csv* para uma mais fácil visualização.

### 3.2.2 Desenvolvimento

Aqui detalha-se o processo de desenvolvimento do *webscraping* do *website* *Booking*.

#### Hotéis

Inicialmente foi feita a filtragem de apenas os hotéis de Beja.

No código foi implementado as bibliotecas *BeautifulSoup4* para facilitar a tarefa de realizar o *webscraping*. Esse mesmo código está disponível no repositório GitHub e no apêndice 4.

A partir do *website* ao inspecionar a página era possível retirar os *headers* que eram valores necessários na realização do *webscraping*. Também é realizado o pedido *HTTP* e juntou-se a informação com a biblioteca *BeautifulSoup4*.

Foram criados diferentes *arrays* para receber as informações e posteriormente colocada a respectiva informação em cada um deles.

Devido a alguns *arrays* conterem mais informação, possivelmente devido a algum tipo de informação adicional que possa estar em algum hotel especificamente, para prevenir erros, foram reduzidos ao tamanho do *array* mais curto.

### 3. WEBSRAPING

---

Por fim todos os resultados contidos nos *arrays* foram guardados num ficheiro *.csv* denominado *listtable.csv*.

Construção dos links para realizar o *webscraping* das *reviews* de cada hotel.

Foi realizado o pedido *HTTP* e juntado á biblioteca *BeautifulSoup4* para aceder ás *reviews* de cada site e todos os valores foram salvos no formato *.csv*.

No final, temos esta tabela representativa dos hotéis *scraped* ordenada e representativa dos *scrapes* *hotelXX.csv*.

	Hotel	Classificação	Preço
0	Hotel Bejense	“8,4”	189
1	Aljana Guest House Beja	“9,3”	330
2	BejaParque Hotel	“8,1”	255
3	Pousada Convento de Beja	“8,7”	270
4	Guest House Stories	“8,7”	135
5	Hotel Melius	“8,1”	242

**Tabela 3.1:** Tabela exemplar dos hotéis do *Booking*

#### 3.2.3 Resultado

Na tabela 3.2 exemplifica-se um ficheiro *.csv*, *hotel18.csv*, que contem os *reviews* do hotel Quinta do Castelo.

	Opiniões
0	A localização é excelente assim como as condições do espaço. Local muito bem cuidado e apelativo. Fomos muito bem recebidos ...
1	Nada digno de registo.
2	A simpatia da Sra Catarina foi fantástica. A casa e as acomodações corresponderam ás expectativas e relação qualidade preço foi perfeita ...
3	Localização excelente, apartamento espaçoso, e totalmente equipado. O facto de ser uma construção antiga, cria um ambiente muito ...
4	A localização é impecável mesmo no centro histórico de Beja. Casa Limpa e organizada. Dona super prestável e e simpática.
5	De noite ouve-se o barulho da rua com muita facilidade.

**Tabela 3.2:** Tabela de resultados exemplar dos *reviews* de um hotel do *Booking*

### 3.3 TripAdvisor

O TripAdvisor é uma empresa americana de viagens *online* que opera *web* e *mobile apps* com conteúdo *user generated* e um website de comparação de preços, dos quais se pode fazer com hotéis, locais atractivos (como monumentos, parques, museus, etc..) e restauração.

Este como sendo um produto/serviço que oferece acesso a três categorias distintas (hotéis, restaurantes e atracções), foi dividido em três partes que representam as três categorias.

Este website é conhecido pelas suas tentativas de dificultar os processos de *scraping* [wws1], o qual foi observado, mas resolvido a custo de tempo. Felizmente encontramos um website chamado *Worth Web Scraping* o qual nos mostrou como fazer na página de hotéis do *TripAdvisor* o *scraping* da tabela de referência e dos *reviews* [wws2].

### 3.3.1 Estratégia

Após um *scouting* inicial às páginas das três categorias, foi observado as seguintes peculiaridades:

- as páginas das três categorias são diferentes no seu *layout* e organização;
- os nomes das classes nos *span*, *div* e outros elementos são *random generated* e mudam de acordo com a sessão aberta ou *cookie*;
- existem representações repetidas, estes são os *posts sponsored* [wws1] pelo próprio *website*;
- as subpáginas que nos retornam os *reviews*, são de comprimentos diferentes de acordo com a categoria de *listing* [wws2];
- as subpáginas que nos retornam os *reviews*, usam múltiplos de cinco ou dez na *query*;
- as subpáginas que nos retornam os *reviews* mostram por defeito os que estão na linguagem referente ao domínio (.pt, .com, etc..) sem *query parameter* para alterar,
- *links* com *query parameters* que representem uma subpágina não existente não dão erro 404 (Page not found), mas redirecionam para a primeira [wws2];
- quando tentamos extrair o total de *reviews* apenas conseguimos o total dos totais e não o total por linguagem, impedindo assim de fazer uma conta para saber qual o múltiplo de cinco ou dez que seria a ultima subpágina.

Assim sendo, a estratégia que foi usada, embora inapropriada em termos de tempo despendido e extracções redundantes, era a única que assegurava que se conseguia extrair todos os *reviews*. Essa estratégia foi:

- criar uma lista de *links* para 200 ou 400 subpáginas (de acordo com o *listing* daquela categoria com mais *reviews* em português);
- extrair incluindo os repetidos para um *array/list/arraylist*;
- usar compreensão de listas através de *sets/dicionários/tuples* que possam ser ordenados para remover repetidos e não perder ordem;
- transpor esses dados para um *dataframe* de pandas e exportá-lo para *.csv* para uso futuro.

### 3.3.2 Desenvolvimento

Aqui iremos detalhar o processo longo do *webscraping* da plataforma *TripAdvisor* e as suas três principais categorias.

## Atracções

Para o desenvolvimento do *webscraping* das Atracções de Beja, foi aberto um caderno de *Jupyter* no qual começámos por fazer o *import* das bibliotecas e desactivar o aviso da falta de certificado *SSL* (após a introdução do trabalho em Inglês).

Seguidamente, foi feita a configuração do *request* onde qual fazemos *download* da página *web* pretendida. Estes *headers* foram extraídos do *browser* do computador usado, *Microsoft Edge (Chromium)*.

Após fazer *request* e verificar o *status code* (vazio ou 200 para OK), foi criado um objecto *Soup* com o *parsing* (*via lxml*) da página *requested*.

Para a criação da tabela de referência das atracções fazemos um ciclo que nos vão fazer *scrape* aos nomes.

Sendo que agora podemos simplesmente através destes *arrays* criados fazer um *dataframe* der *pandas* via um dicionário de *Python* com os variados *pandas* referidos. Seguidamente exportamos o *dataframe* para um ficheiro *.csv*.

Agora um ciclo que retira os *HTML tag* onde contém um *href* com uma parte do *link* que nos possibilita (criar o *link* inteiro e) visitar a pagina de *reviews*.

Essas páginas têm determinadas restrições faladas nas secções anteriores e a sua solução. A qual aqui em baixo representada, cria uma enormidade de *links* por local. Dos quais *links* agora serão *scraped* (incluindo os *reviews* repetidos e excepto os que contêm *desde* e *euros*) e seguidamente tratados (remoção de repetidos) indo seguidamente para um (dicionário e transformado num) *dataframe* de *pandas*, o qual é imediatamente exportado com o número do atracção referente na tabela de referência.

## Hotéis

Para o desenvolvimento do *webscraping* dos Hotéis de Beja, foi aberto um caderno de *Jupyter* no qual começámos por fazer o *import* das bibliotecas e desactivar o aviso da falta de certificado *SSL* (após a introdução do trabalho em Inglês).

Seguidamente, foi feita a configuração do *request* onde qual fazemos *download* da página *web* pretendida. Estes *headers* foram extraídos do *browser* do computador usado, *Microsoft Edge (Chromium)*.

Após fazer *request* e verificar o *status code* (vazio ou 200 para OK), foi criado um objecto *Soup* com o *parsing* (*via lxml*) da página *requested*.

Para a criação da tabela de referência dos hotéis fazemos um grupo de ciclos que nos vão fazer *scrape* aos nomes, *ratings*, número total de *reviews* e preços. Sendo que este número de *reviews* não nos vale de muito tal como previamente referido.

Sendo que agora podemos simplesmente através destes *arrays* criados fazer um *dataframe* der *pandas* via um dicionário de *Python* com os variados *pandas* referidos. Seguidamente exportamos o *dataframe* para um ficheiro *.csv*.

O qual gerou uma tabela de hotéis como referência.

Mesmo que o número total de *reviews* não nos seja relevante o *HTML tag* onde é retirado contem um *href* com uma parte do *link* que nos possibilita (criar o *link* inteiro e) visitar a pagina de *reviews*.

Essas páginas têm determinadas restrições faladas nas secções anteriores e a sua solução. O que cria uma enormidade de *links* por local.

Dos quais *links* agora serão *scraped* (incluindo os *reviews* repetidos) e seguidamente tratados (remoção de repetidos) indo seguidamente para um (dicionário e transformado num) *dataframe* de *pandas*, o qual é imediatamente exportado com o número do hotel referente na tabela de referência.

### Restaurantes

Para o desenvolvimento do *webscraping* dos Restaurantes de Beja, foi aberto um caderno de *Jupyter* no qual começamos por fazer o *import* das bibliotecas e desactivar o aviso da falta de certificado *SSL* (após a introdução do trabalho em Inglês).

Seguidamente, foi feita a configuração do *request* onde qual fazemos *download* da página *web* pretendida. Estes *headers* foram extraídos do *browser* do computador usado, *Microsoft Edge (Chromium)*.

Após fazer *request* e verificar o *status code* (vazio ou 200 para OK), foi criado um objecto *Soup* com o *parsing* (*via lxml*) da página *requested*.

Para a criação da tabela de referência das atracções fazemos um ciclo que nos vão fazer *scrape* aos nomes e as partes de *href* contidas nos *href* para a criação dos *links* dos *reviews*.

Sendo que agora podemos simplesmente através destes *arrays* criados fazer um *dataframe* der *pandas* via um dicionário de *Python* com os variados *pandas* referidos. Seguidamente exportamos o *dataframe* para um ficheiro *.csv*. O qual gerou uma tabela de restaurantes como referência.

Agora um ciclo que vai buscar as partes de *links* onde do ciclo anterior que nos possibilita (criar o *link* inteiro e) visitar a página de *reviews*.

Essas páginas têm determinadas restrições faladas nas secções anteriores e a sua solução. A qual aqui em baixo representada, cria uma enormidade de *links* por local.

Dos quais *links* agora serão *scraped* (incluindo os *reviews* repetidos) e seguidamente tratados (remoção de repetidos) indo seguidamente para um (dicionário e transformado num) *dataframe* de *pandas*, o qual é imediatamente exportado com o número do restaurante referente na tabela de referência.

#### 3.3.3 Resultado

Aqui apresenta-se os resultados do *webscrape* do *TripAdvisor*. Os quais representam um exemplar dos dez primeiros *reviews* do primeiro hotel, atracção e restaurante, respectivamente.

### 3. WEBSRAPING

---

	Avaliações
0	Excelente hotel . Pessoal da recepção e serviços de quarto muito atenciosos e prestativos - o frigobar...
1	Património histórico ao seu melhor nível de recuperação, renovação e utilização. Magnificas salas, como...
2	Chegámos depois da meia-noite e fomos recebidos com extrema simpatia! Passámos o nosso aniversário de...
3	Gostei muito do alojamento e da estadia. Destaco a beleza, qualidade e localização da pousada, a simpatia de...
4	Na reserva tinha a descrição de um tipo de quarto e foi atribuído outro. Parque infantil insuficiente e fechado...
5	Boas instalações, uma óptima piscina, bons acessos e estacionamento. Um inexcusável acolhimento, simpatia e...

**Tabela 3.3:** Tabela representativa dos cinco primeiros *reviews* de um hotel do *TripAdvisor*

	Avaliações
0	Muito bom.
1	Castelo bem conservado, com torre de menagem activa e exuberante, pena fechar a horas proibitivas. Pode-se dar a...
3	Castelo bonito numa zona central de Beja. Não se paga para entrar nem para andar pelas muralhas. A vista é bonita....
4	
5	No entanto, apesar de existirem pessoas dedicadas, não se vende qualquer tipo de recordações.

**Tabela 3.4:** Tabela representativa dos cinco primeiros *reviews* de uma atracção do *TripAdvisor*

	Avaliações
0	Num bairro de Beja encontra-se este restaurante com esplanada e sala mediana. Carta com muitas sugestões de entradas...
1	Não se deixem intimidar pelo aspecto do restaurante. Comida ao nível de uma estrela Michelin. Bem confeccionada...
2	duas pessoas, embora sendo individuais. Serviço simpático e pronto. O espaço não é condizente com a delícia da comida...
3	Mais
4	Cozinhar divinamente, sem dúvida é uma arte! Nível de estrela Michelin e preço normal! Só retiraria a TV da sala...

**Tabela 3.5:** Tabela representativa dos cinco primeiros *reviews* de um restaurante do *TripAdvisor*

## 3.4 Zomato

A *Zomato* é um serviço de busca de restaurantes para quem quer sair para jantar, buscar comida ou pedir em casa. A *Zomato* possui duas secções: guia de restaurante e *blog*. Previamente, havia uma secção de eventos, já descontinuada.

O guia de restaurantes *Zomato* permite ao usuário buscar informações relacionadas a restaurantes, bares, cafés, *pubs* e casas noturnas. As informações fornecidas geralmente incluem o nome do estabelecimento, telefones de contacto, endereço, cardápio, fotografias, avaliações e mapas de localização.

### 3.4.1 Estratégia

As páginas da *web app* do *Zomato* usam um *parallax* de *scrolling* infinito (até não haver mais restaurantes) e as classes dos *HTML tags* mudam por sessão e/ou *rendering*, logo aqui a estratégia é literalmente fazer *download* da página *web* e fazer o *scrape* a partir do *parsing* dessa página.

### 3.4.2 Desenvolvimento

Aqui é representada uma aproximação do desenvolvimento deste *scraping*.

## Restaurantes

Primeiramente foi feito o *import* das bibliotecas.

Depois pegando no código dos colegas como *template*, adaptou-se para usar uma página previamente descarregada.

Fazemos um ciclo de *scraping* dos nomes dos locais de consumo.

E agora dois ciclos, um para as classes com nome gerado no *prerender* e outra pós *render*; vamos buscar os tipos/classes de restaurantes, e outros dois ciclos do mesmo motivo, para ir buscar os preços. Pelo mesmo motivo criámos dois ciclos; que vão buscar os *links* das páginas dos *reviews*, o qual extraímos todos os *tags* de parágrafos porque que sempre que se corria o código gerava uma classe nova.

### 3.4.3 Resultado

Os resultados deste *scrape* foram desapontantes no mínimo devido à infeliz *random generated* nome da classe, que é gerado por cada vez que se usa a página. Estes resultados vão sofrer muito *ETL* posterior.



## Capítulo 4

# Análise de Dados (*Data Mining*)

### 4.1 Pré-processamento

Esta etapa consiste em remover todos os caracteres especiais, que não são letras, números ou espaços [u1]. As quais podemos considerar diacrítico como caracteres especiais ou não. Dependendo do caso, podemos remover todos os caracteres especiais, ou apenas os que não são diacríticos [cr1].

Nos nossos casos de análise textual, houve necessidade de remover todos os caracteres especiais, mas os diacríticos podem ser úteis para a nossa análise, em especial nos passos seguintes, extracção de *keywords* e análise de sentimentos, foram descartados ou usados respectivamente, pelo motivo da precisão da análise em questão.

#### 4.1.1 Metodologia

Para remover os caracteres especiais, foram criados dois *scripts*, um para limpar os ficheiros *.csv* e outro tratar do texto em si, que consiste em remover os caracteres especiais, fazer a normalização dos caracteres via *NFKD*[nfkd1], isto é decompor os caracteres e recompor apenas pela forma canónica equivalente, e aplicar a remoção de acentos [cr1], caso seja necessário (apenas alterando o último passo de *encoding/decoding*).

Estes não requerem bibliotecas externas, podem ser executados em *Python 3*, e as suas bibliotecas *standard*.

#### Limpar ficheiros

O *script* *trimer.py* é responsável fazer *trimming* (remover espaços em branco), remover linhas em branco e colocar aspas duplas em cada *review*. Para estes foi necessário o uso de expressões regulares [u1].

## 4. ANÁLISE DE DADOS (*Data Mining*)

---

### Limpar texto dos *reviews*

O *script normalize* é responsável por remover caracteres especiais, fazer a normalização dos caracteres via *NFKD* [**nfkd1**] e aplicar a remoção de acentos [**cr1**], caso seja necessário (apenas alterando o último passo de *encoding/decoding*).

#### 4.1.2 Execução

Executamos o *script trimer.py* para limpar os ficheiros *.csv* [**gfg1**] e o *script normalize* para limpar o texto dos *reviews*. Estes aceitam um caminho para uma pasta com os ficheiros *.csv* e itera sobre os ficheiros, executando as funções de limpeza.

Após a execução da limpeza textual e normalização sem remoção de acentos, seguidamente realizámos com remoção de acentos para que tenhamos ambas as versões. Para a versão com remoção de acentos, foi necessário usar o pacote *Unidecode* para aplicar a remoção de acentos [**cr1**].

Com estas duas versões podemos obter os resultados óptimos para a nossa análise textual.

#### 4.1.3 Resultados

Os ficheiros *.csv* foram limpos e normalizados sem remoção de acentos e com remoção de acentos.

## 4.2 Extracção de *keywords*

A extracção de *keywords* é uma técnica de extracção de informações que consiste em extraír *keywords* de um texto.

Geralmente, as *keywords* são utilizadas para identificar o conteúdo de um documento [**tamy1**], ou seja, para identificar o que o documento contém [**tamy2**]. No nosso caso, as *keywords* são utilizadas para identificar pontos fulcrais da recepção de um cliente turístico num estabelecimento turístico de Beja (hotel, atracção, restaurante, etc).

Com estas *keywords*, é possível identificar o que o cliente quer, ou seja, o que ele quer ver, o que ele quer comer, o que ele quer fazer, ou o nível de satisfação com o serviço prestado.

### 4.2.1 Metodologia

Para a extracção de *keywords*, utilizamos uma biblioteca de *Python* chamada *YAKE*. Esta é um *pipeline* de processamento de linguagem natural, que utiliza um algoritmo personalizado descendente do *Naïve Bayes* para extraír *keywords* de um texto.

O *YAKE* é um *software* livre, e pode ser obtida via *pip* ou via *Github*, que é uma ferramenta de extracção de *keywords* desenvolvida por autores portugueses (e um japonês) da

Universidade do Porto, Politécnico de Tomar, e da Universidade da Beira Interior (e da Universidade de Kyoto). A sua utilização pode ser simples, basta instalar a biblioteca e executar o seguinte comando: `yake.KeywordExtractor(lang="pt").extract_keywords(text)`

Para a extracção de *keywords* de cada *review*, utilizamos o *YAKE* iterativamente por cada *review*, alimentando-o com o seu conteúdo textual, após o pré-processamento textual de todas as *reviews* (a qual especificamente foi usada a versão sem diacríticos).

O *output* da alimentação do *YAKE* é um par de *keywords* e seus respectivos pesos, que são armazenados num dicionário [tamgh1]. O dicionário é ordenado pelo valor de seus pesos [tamyt3], e o número de *keywords* extraídas é limitado ao número de *reviews* que foram utilizadas para a extracção por cada estabelecimento.

No entanto esta pode ser mais complexa caso seja necessário optimizar a extracção de *keywords*, com os seus variados parâmetros opcionais.

Este funciona da seguinte forma: quando recebe um texto, vai testar todas as palavras do texto, com uma determinada fórmula, e guardar o peso da palavra, e a palavra em si, no final expele um dicionário com as *keywords* e seus respectivos pesos.

A fórmula falada anteriormente é:  $S(kw) = \frac{\prod_{w \in kw} S(kw)}{TF(kw) * \sum_{w \in kw} S(w)}$

Mais especificamente este módulo é uma forma mais delicada e avançada de um classificador *Naïve Bayes* [tamgh1] [tamyt1] [tamyt2] [tamyt3] o qual será mais e melhor explicado no capítulo seguinte onde procedemos ao desenvolvimento de um para efeitos de análise de sentimentos.

## Execução

Foi criado um *notebook* de *Jupyter* o qual contém o código usado para a extracção de palavras via *YAKE*. Cada bloco de código está sobreposto por um bloco de *markdown*, que é um comentário. Os comentários são usados para explicar o que cada bloco de código faz.

As rotinas de extracção de *keywords* são: iterativamente, para cada ficheiro *.csv* dentro da pasta indicada, importar via *DataFrame* de *pandas*. O qual *DataFrame* é um conjunto de dados, como uma tabela de dados, e contém uma coluna com os *reviews*. O *DataFrame* é iterado na sua coluna única, e o seu conteúdo é passado para o *YAKE*. O qual exporta o resultado para um ficheiro *.csv*, que é um ficheiro *.csv* com uma coluna com as *keywords* e outra com o seu peso.

O *notebook* e o código estão disponíveis nos apêndices ??

### 4.2.2 Resultados

Os resultados obtidos detêm um sentimento misto no grupo. Estas *keywords* por vezes não são palavras únicas, mas são expressões que são frequentes. Muitas palavras únicas aparecem lado a lado das expressões, o que dá uma noção de repetição ou confirmação de resultado. Os quais podemos observar as *keywords* “Victoria” e “Santa” e a expressão “Santa Victoria”.

## 4.3 Análise de Sentimentos

A análise de sentimentos é uma técnica de extração de informações que consiste em extrair sentimentos de um texto.

Geralmente, os sentimentos são utilizados para identificar o sentimento de um texto, ou seja, para identificar o que o texto contém. No nosso caso, os sentimentos são utilizados para identificar a satisfação do cliente com o serviço prestado em variados serviços turísticos de Beja (hotéis, atracções, restaurantes, etc).

Com base no texto, o sentimento é extraído através de um algoritmo que identifica a intensidade do sentimento. No nosso caso, a classificação é binária, ou seja, o sentimento é positivo ou negativo. Consideramos uma não-reclamação como positiva.

A percentagem de sentimentos positivos e negativos é necessária para identificar a satisfação do cliente com o serviço prestado. A satisfação do cliente é uma medida de qualidade de serviço.

### 4.3.1 Metodologia

Para fazer uma análise de sentimento textual, utilizámos um modelo de *machine learning* chamado *BERT* baseado em *Transformers*, a qual criámos um modelo de classificação de texto binário ou ternário, no nosso caso, o modelo é um classificador de sentimentos binário.

Este modelo é treinado através da alimentação de um conjunto de dados de treino e um conjunto de dados de teste, ao modelo matemático criado ou importado. Estes modelos matemáticos podem ser probabilísticos ou não. Os dados de treino são utilizados para treinar o modelo matemático. Os dados de teste são utilizados para testar o modelo matemático.

Após o treino do modelo matemático, e os resultados do teste (matriz de confusão, precisão, exactidão, etc), o modelo é utilizado para classificar um texto. O resultado da classificação é o sentimento do texto, ou seja, positivo ou negativo.

Este trabalho de treino, teste e validação, no caso do modelo final escolhido, foi feito pelas empresas/comunidades responsáveis e patrocinadoras do projecto. Porém nas varias tentativas de desenvolvimento, com outros modelos, também fizemos um conjunto destas tarefas.

Para pré-processar o texto a ser classificado (e os textos de treino e teste), utilizámos o pacote de linguagem natural do *Python*, ou seja, a biblioteca *NLTK*, para aplicar algumas transformações ao texto, como remover pontuação, remover *stopwords*, etc.

Este passo é o mais importante para o modelo matemático ser bem treinado e para o modelo ser bem avaliado.

### Pré-processamento

Para fazer o pré-processamento do texto, utilizámos as ferramentas do *NLTK*, das quais em especial os que fazem (ou determinam) as *stopwords*, fazem *stemming*, *tokenização*, etc.

Mais especificamente *stopwords* portuguesas do *corpus* de *stopwords* do *NLTK*, o *SnowballStemmer*, é utilizado para remover *stopwords* e os sufixos das palavras deixando apenas a raiz da palavra (*the stem*), e o *Vectorizer* que é utilizado para transformar o texto em um vector (ou matriz) de características (termos numéricos).

Este último é preferencial que se use o *CountVectorizer*, pois ele conta o número de vezes que um termo aparece no texto com inteiros, e não com *floats*, como o *TF-IDF* faz. Sendo o último mais preferencial para outras tarefas (não classificação).

O *Stemming* é para nós a fase mais importante do processo de pré-processamento, pois é o que removemos os sufixos das palavras. Para muitos propósitos, e em especial este, a conjugação dos verbos atrapalha a classificação e a aprendizagem do modelo (ou até a nossa). Um exemplo da sua desnecessidade é a enorme diferença entre o Inglês e o Português.

Os verbos em inglês são conjugados com substantivos, e os substantivos são conjugados com verbos. Porém, os verbos em português são conjugados com adjetivos, e os adjetivos são conjugados com verbos. Mais a enorme variação entre pessoas e tempos, em inglês variamos de três formas para seis (nove se separarmos *he/she/it*) pessoas para três tempos (um presente e dois passados, futuros e condicionais são modificações de um presente), já em português variamos de seis formas para seis pessoas (oito se separarmos *ele/ela* e *eles/elas*) para seis tempos (um presente, três passados, um futuro e um condicional).

Estas conjugações tem a mesma ação e uma enorme semelhança frásica: a raiz do verbo (*the stem*). As conjugações são desnecessárias e demasiado complexas.

Sendo assim, ao aplicar *stemming*, reduzimos a dimensão da matriz/vector, e simplificamos a linguagem, de maneira inteligente.

#### 4.3.2 Tentativas

Foram feitas três tentativas de classificação de sentimentos. A primeira foi um fracasso completo, foi tentada a criação de um modelo sequencial com *layers LSTM* com *embedding* e de convulsão. À falta de conhecimento prévio, e à falta de informações simples e palpáveis com acesso fácil na *internet*, não foi possível fazer a classificação de sentimentos com este modelo. Este modelo foi tentado com o uso da biblioteca de *TensorFlow*, que é uma biblioteca de código aberto.

A segunda tentativa foi muito mais bem sucedida que a primeira. Foi usado para o algoritmo de classificação de sentimentos o *Naïve Bayes (Multinomial)*, e como dados de treino e teste, foram utilizados os dados de treino e teste disponíveis no *Kaggle*, estes usavam *reviews* de filmes e produtos de *e-commerce* em português do *Brasil*, já que em português de Portugal não foi possível encontrar.

## 4. ANÁLISE DE DADOS (*Data Mining*)

---

### Modelo Sequencial *LSTM* com *Embedding*

Este modelo não chegou a completar qualquer fase de treinamento, pois não foi possível criar um modelo que funcionasse com o *dataset* de treino, a quantidade absurda de variáveis e modelações junto com a falta de conhecimento prévio impossibilitaram que um modelo funcional fosse criado, inclusive com *trimming* ou *truncation* dos *inputs*.

### Execução

Após a importação do pacote de *TensorFlow*, foi criado um modelo sequencial com o uso de *embeddings*, ou seja, um modelo que utiliza *embeddings* para representar os *inputs*. Os *layers LSTM* e *Dropout* foram utilizados para aumentar a capacidade de aprendizagem do modelo. E os *layers* de saída são os *softmax* e *categorical\_crossentropy*.

Cada um destes *layers* detinha parâmetros que teriam de ser ajustados para que o modelo fosse capaz de classificar os sentimentos. O qual requeria algum apoio não disponível, ou seja, o modelo não foi capaz de aprender a classificar sentimentos.

### Resultados

Não foi possível executar a tentativa de classificação de sentimentos com este modelo.

### Modelo *Naïve Bayes* do *Scikit-Learn* com *Datasets PT-BR* do *Kaggle*

Nesta tentativa de classificação de sentimentos, foi utilizado o pacote *Scikit-Learn*, e foi utilizado um par de *datasets* de *reviews* do *Kaggle*, que é um *dataset* de *reviews* de filmes e produtos de *e-commerce* em português do *Brasil*, já que em português de Portugal não foi possível encontrar.

Foi criado um objecto (que deriva da nossa classe *StemmerTokenizer*), que foi utilizado para aplicar o *Stemming* e *Tokenização* aos dados de treino e teste. Reduziu-se a quantidade de dados, e aumentou a capacidade de aprendizagem do modelo. A necessidade de aplicar *stemming* à *Tokenização* veio da necessidade de reduzir a dimensão da matriz/vector, e simplificar a linguagem, de maneira inteligente.

Sendo assim, ao aplicar *stemming*, reduzimos a dimensão da matriz/vector, e simplificamos a linguagem, de maneira inteligente.

### *Naïve Bayes*

Um classificador *Naïve Bayes* é um classificador simples e probabilístico baseado em aplicar a teoria de *Bayes* com assumpções fortes (naïve) de independência [mbn1]. *Naïve Bayes classifiers* são simples e fáceis de entender, requerendo nenhuma etapa de *prunning* para evitar o *overfitting*. Contudo, eles não são mais poderosos do que outras técnicas avançadas, como árvores de decisão ou vector de suporte, muito menos que máquinas neurónios.

### Como funciona um classificador *Naïve Bayes*?

Estes classificadores são baseados em aplicar a teoria de *Bayes* com assumpções fortes (*naïve*) de independência [mbn1]. A hipótese *naïve* diz que os *features* são independentes uns dos outros. Isso significa que os *features* são condicionais independentes a partir da classe [gfg2]. Indústria matemática pode ser usada para mostrar que os *features* são independentes a partir da classe, portanto a classe é a mais provável de saída.

A ideia principal do *Naïve Bayes* é calcular a probabilidade de cada classe, dado os *features*. A probabilidade de uma classe é calculada por multiplicar as probabilidades de cada *feature* dado a classe [mbn1].

Este é um exemplo simples de um classificador *Naïve Bayes*: recebe um vector de *features* e um rótulo de classe e retorna a probabilidade da classe dado os *features* [skl1]. No problema de classificação de texto, o rótulo de classe é a classe real do texto. Se inserirmos um vector de *features* do texto e o rótulo de classe, o classificador *Naïve Bayes* retorna a probabilidade da classe dado os *features* [gfg2].

Outros exemplos de classificadores *Naïve Bayes* são: análise de sentimentos, detecção de *spam*, e classificação de texto [mbn1]. Como para nosso caso (o problema de análise de sentimentos), temos um vector de *features* do texto e o rótulo de classe é o sentimento do texto. O conjunto de classe é um conjunto binário de positivo e negativo [gfg2]. Recebe um vector de *features* [skl1] do texto e o rótulo de classe e retorna a probabilidade de zero a um de classe, sendo o mais próximo a um o mais positivo o texto é.

### Execução

Foi criado um *notebook* para a execução da tarefa. Este *notebook*, contém o código do modelo *Naïve Bayes* do *Scikit-Learn* [skl1], e as referencias para *datasets* de treino e teste. Com blocos alternados de código e *markdown* que comentam a execução do modelo, foi possível executar a tentativa de classificação de sentimentos com este modelo.

As rotinas executadas foram: a criação de um objecto *StemmerTokenizer* [skl1], a criação de um modelo *Naïve Bayes* do *Scikit-Learn* [skl1], o treinamento do modelo, em que de fazia *drop* a inúmeras colunas, os testes e métricas de avaliação do modelo, a importação iterativa dos *.csv* que contém os *reviews* dos estabelecimentos em que iterativamente foi aplicada uma limpeza e normalização, e a execução do modelo nos *reviews*, que quando classificados, geraram um arquivo *.csv* com os resultados.

### Resultados

Este modelo foi capaz de classificar sentimentos com sucesso. A classificação foi bem sucedida, e foi possível classificar os sentimentos de um conjunto de *reviews*, e gerar um arquivo *.csv* com os resultados. Porém a classificação foi pouco precisa, e a precisão real foi baixa, apesar de ser um modelo bem sucedido e de ter sido bem treinado e avaliado. A sua *accuracy* foi de entre 91% e 87%, dependendo dos *runs*.

#### 4. ANÁLISE DE DADOS (*Data Mining*)

---

Calculámos que o problema está na diferença entre o português de Portugal e o português de *Brasil*, o qual o último é a língua dos *datasets* de treino e teste, o qual o modelo foi bastante preciso; e o primeiro a língua materna dos estabelecimentos e dos clientes, o qual o modelo foi pouco preciso.

#### **Pipelines de Transformers do HuggingFace e Modelos BERT da Google fine-tuned**

Como os resultados finais do modelo *Naïve Bayes* não foram satisfatórios, foi criado um *pipeline* de *transformers* do *HuggingFace* [yt2], com um modelo *BERT* da *Google fine-tuned* para classificar sentimentos de forma binária.

Esta biblioteca pode usar *PyTorch* ou *TensorFlow* como *backend* de computação, cada um destes tem suas vantagens e desvantagens. Por defeito ele usa *PyTorch* na maioria dos seus *pipelines*, mas pode ser configurado para usar *TensorFlow*.

Estes *pipelines* do *HuggingFace* são muito mais complexos, são muito mais eficientes, e também são muito flexíveis e fáceis de usar[yt2]. São definidos com um texto que determina qual é o *pipeline*, e os parâmetros pedidos. No nosso caso foi utilizado o *pipeline* de classificação de sentimentos em que tinha dois parâmetros, o *tokenizer* e o modelo, mais especificamente um *AutoModelForSequenceClassification* e um *AutoTokenizer*, os quais foram buscar o modelo *pretrained* e o *tokenizer* do *gchhablani/bert-base-cased-finetuned-sst2* [yt2].

#### O que é um Transformer

Um *Transformer* é um tipo de rede neural que é capaz de aprender funções complexas de dados. Ele funciona através de transformar os dados de entrada em uma nova representação, que pode então ser usada para fazer previsões em novos dados.

*Transformers* têm muitas aplicações em linguagens de processamento natural, processamento de imagens e visão computacional. Eles foram bem-sucedidos em diversos domínios, incluindo:

- Sumários de texto (*BERT*, *DistilBert*, *RoBERTa*, *XLNet*);
- Captação de imagens (*XLNet*);
- Tradução de imagens (*BERT*, *DistilBert*, *RoBERTa*);
- Respostas a perguntas (*BERT*, *DistilBert*, *RoBERTa*);
- *Chatbots* (*BERT*, *DistilBert*, *RoBERTa*);
- Classificação (*BERT*, *DistilBert*, *RoBERTa*);
- *Entre outros*.

O primeiro *Transformer* introduzido foi o *BERT* model. Foi desenvolvido pela Google. É um modelo de sequência-para-sequência que tem um *encoder* e um *decoder*. O *encoder*

mapeia uma sequência de *tokens* de entrada para uma sequência de estados ocultos. O *decoder* toma o output do *encoder* e tenta mapeá-lo de volta para a sequência original de *tokens* [yt2].

A *NVIDIA* e a *Facebook* desenvolveram *XLNet* e *DistilBert* [yt2]. São semelhantes ao *BERT*, mas tem uma arquitectura diferente e um conjunto de pesos diferentes. *XLNet* é um modelo de sequência-para-sequência que tem um *encoder* e um *decoder*. O *encoder* mapeia uma sequência de *tokens* de entrada para uma sequência de estados ocultos. O *decoder* recebe o output do *encoder* e tenta mapeá-lo de volta para a sequência original de *tokens*. *DistilBert* é similar ao *XLNet*, mas é treinado em um subconjunto muito menor do que o data.

### Como funciona um *Transformer*?

A funcionalidade de um *Transformer* é transformar uma sequência de *tokens* de entrada em uma sequência de *tokens* de saída. Os *tokens* [mtf1] [mtf2] de entrada são geralmente palavras, mas eles também podem ser outros tipos de *tokens*, como imagens de captação. Os *tokens* de saída são geralmente iguais aos *tokens* de entrada, mas eles podem ser diferentes [hf1].

Ele funciona através de transformar os *tokens* de entrada em uma nova representação, que pode então ser usada para fazer previsões em novos dados. Os *layers* de sequência são chamados de *encoder* [mtf1] e de sequência. Elas têm x *layers* de *encoder* e y *layers* de *decoder*. x e y são geralmente iguais, mas eles podem ser diferentes. Os *layers* podem ser diferentes tamanhos [hf1].

Especificamente o *Transformer* é um modelo de sequência-para-sequência. Ele usa *Embeddings* para os *tokens* de entrada e posiciona-os antes de serem enviados para o primeiro conjunto de *layers*. O primeiro conjunto de *layers* tem um *self-attention* [mtf2] mecanismo que toma os *tokens* [mtf1] de entrada e transforma-os em uma nova representação que é adicionada e normalizada antes de ser enviada para o segundo conjunto de *layers* onde os primeiros passos são iguais aos anteriores mas eles usam os *outputs* anteriores como *inputs*. Essa nova *layer* combina os *inputs* com os *outputs* antigos e os *outputs* são enviados para o próximo conjunto de *layers*. Isso é repetido até que o *output* seja o mesmo que o *input*. Então ele passa por um *layer linear regression* e um *softmax*. O *output* é o final *output* [hf1].

### Layers em específico

Os três mais importantes tipos de *layers* [hf1] são:

- A *softmax function* é uma função comum em redes neurais. Ela toma um vector e retorna um vector com o mesmo tamanho. A *softmax function* é usada para normalizar o *output* da rede. Ela é usada para garantir que o *output* é uma distribuição probabilística [mtf1];

## 4. ANÁLISE DE DADOS (*Data Mining*)

---

- Uma regressão linear é uma função que toma um vector e retorna um vector; Ela é usada para fazer previsões usando um meio probabilístico simples e comum;
- A função de *self-attention* é feita usando uma combinação de regressão linear e a *softmax*, com ou sem paralelismo (*multi-head attention*), a função de *self-attention* é usada para imprimir uma importância para o conjunto de palavras que está sendo avaliado.

### Execução

No nosso *notebook*, foi criado um *pipeline* de *transformers* do *HuggingFace*, com um modelo *BERT* da *Google fine-tuned*[[hf1](#)] para classificar sentimentos de forma binária. E foi criado um *notebook* para a execução da tarefa. Este *notebook*, contém o código do pipeline de *transformers* do *HuggingFace*, e as referências para o modelo *pretrained* e o *tokenizer* desse modelo.

Com blocos alternados de código e *markdown* que comentam a execução do pipeline, foi possível executar a classificação de sentimentos com este pipeline. Iterativamente fomos buscar os *.csv* dos *reviews* dos *estabelecimentos*, e foi aplicado o *pipeline* de *transformers* do *HuggingFace* sequencialmente a cada *review*, e ao final, foi gerado um arquivo *.csv* com os resultados do local.

Este *pipeline* foi o mais rápido das três tentativas de classificação de sentimentos, e foi o mais preciso das três tentativas de classificação de sentimentos.

### Resultados

Os resultados foram bem sucedidos, e foi possível classificar os sentimentos de um conjunto de *reviews*, e gerar um arquivo *.csv* com os resultados. A classificação foi bem sucedida e bastante precisa, e foi possível classificar os sentimentos de um conjunto de *reviews*, e gerar um arquivo *.csv* com os resultados.

De acordo com a documentação deste modelo (e do *BERT* original da *Google*), a classificação foi bem precisa, e a precisão foi de entre 97% e 98%. Dependendo da tarefa o *BERT* pode até chegar a quase 100% de precisão, neste caso desce devido à natureza binária da classificação.

Embora não sejam 100% precisos, o *BERT* foi o mais preciso das três tentativas de classificação de sentimentos e os resultados foram bastante satisfatórios.

#### 4.3.3 Ponderação dos Resultados finais

Os resultados obtidos podem não ser 100% precisos, mas ainda assim, podem ser bastante satisfatórios. O qual podemos considerar como um resultado final, e que pode ser usado como um indicador de qualidade do estabelecimento em avaliação.

# Capítulo 5

## Geração de gráficos

Para proceder à análise dos dados, é necessário que os dados sejam organizados num formato que permita a leitura e a visualização dos dados facilitada a humanos. Foram gerados os gráficos para a análise de sentimentos e *keywords* em *sets* de totais e de forma temporal.

Para os gráficos de totais, foi utilizado o pacote *matplotlib* [va1] e o *WordCloud* [gfg3] em que foram usados todos os dados disponíveis (das três plataformas). Já os gráficos de análises temporais foram gerados apenas com os dados do *TripAdvisor*, visto que não só eram os mais completos e extensos de todas as categorias de estabelecimento, como também foi possível extrair as datas de criação dos *reviews*.

Com gráficos de totais, queremos dizer que nos dados apresentados e nas análises não é considerado o desenvolvimento temporal, mas sim o desenvolvimento de um total de *reviews*, ou seja, o mês e ano são descartados.

### 5.1 Metodologia

Para a geração dos gráficos de totais, foi utilizado o pacote *matplotlib* [va1] e o *wordcloud* [gfg3] num *script Python*, que iterava sobre os dados disponíveis e gerava os gráficos para cada estabelecimento.

Os tipos de gráficos gerados são: gráficos circulares de sentimentos, gráficos de *keywords* e nuvens de *keywords*. Os quais demonstram a quantidade de sentimentos positivos e negativos, as dez *keywords* mais frequentes e as nuvens de *keywords* limitadas até cem palavras.

#### 5.1.1 Execução

Para cada tipo de estabelecimento, de cada plataforma, foi accionada as rotinas de geração dos gráficos. As quais foram exportadas em formato de imagem. Iterando sobre os tipos de estabelecimento e as plataformas, em que itera sobre os dados disponíveis, segmentados

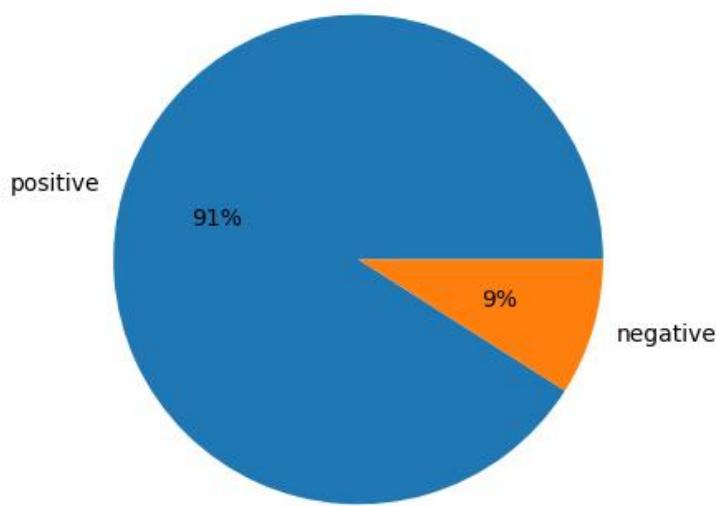
## 5. GERAÇÃO DE GRÁFICOS

---

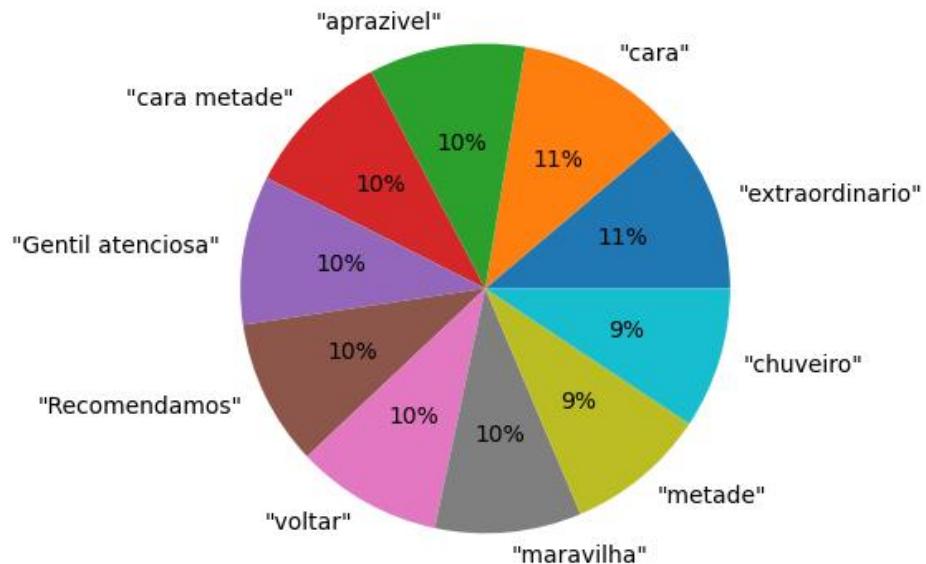
em sentimentos e *keywords*, que passam por uma e duas funções respectivamente. Gerando e exportando .jpg dos gráficos e nuvens.

### 5.1.2 Resultados

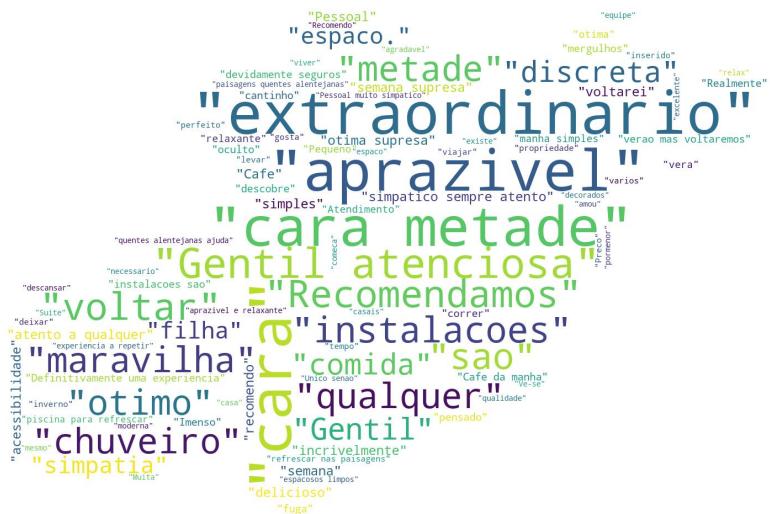
Como se podem verificar nestas três imagens seguintes (5.1, 5.2 e 5.3), os gráficos de totais apresentam um desenvolvimento de sentimentos e *keywords*, que demonstra um sentimento total positivo no nosso turismo, e *keywords* que apresentam a satisfação com o serviço ou uma característica do estabelecimento.



**Figura 5.1:** Gráfico circular gerado baseando-se nos sentimentos dados da plataforma TripAdvisor referente à Herdade das Barradas da Serra



**Figura 5.2:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Herdade das Barradas da Serra



**Figura 5.3:** Gráfico de *keywords* e nuvens de *keywords* contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Herdade das Barradas da Serra

## 5.2 Reorganização dos dados (*TripAdvisor* apenas)

Para as análises temporais e para o transporte de dados, foi necessário criar uma base de dados organizada e relacional, que garantisse a integridade dos dados e a sua coerência na importação dos mesmos para o *PowerBI*. A importação de apenas ficheiros *.csv* não nos apresenta relações entre tabelas, nem como quais possíveis relações funcionariam.

Para isso foram usados os pacotes *pandas* para o *import* dos *.csv* em *DataFrames* e *sqlite3* para a criação da base de dados *SQLite3*.

### 5.2.1 Metodologia

Inicialmente, foi criada uma base de dados *SQLite3*, que será usada para armazenar os dados. A base de dados *SQLite3* foi criada e as tabelas foram criadas, com os campos correspondentes a cada coluna do ficheiro *.csv*, dos quais os dados foram importados via *pandas*.

Os *DataFrames* de *pandas* oferecem uma maneira de aceder e manipular dados, e também fornecem uma maneira de criar e manipular tabelas. Todas as operações de manipulação de dados são feitas através de funções de *DataFrame*.

Seguidamente, é empurrado os dados dos *DataFrames* correspondentes às tabelas para a base de dados. Com esta base de dados, é possível fazer consultas e manipulações de dados, tal como também exportar os dados organizados para ficheiros *.csv* de forma a que possam ser usados em outros programas, tais como o *R*, o *Excel*, o *PowerBI*, etc.

### 5.2.2 Execução e Resultados

Foi feito um *script* para executar a criação da base de dados *SQLite3*, e para o *import* dos dados dos *DataFrames* para a base de dados. O qual foi executado com sucesso, como podemos ver na base de dados e nos ficheiros *.csv*.

## 5.3 Gráficos temporais (*TripAdvisor* apenas)

Com gráficos temporais, queremos dizer que nos dados apresentados e nas análises é considerado o desenvolvimento temporal, quer isto dizer que as *reviews* mostradas ao longo do tempo tornam possível verificar as datas em que foram escritas e a quantidade que cada hotel/ restaurante/ atracção recebeu ao longo dos anos e meses.

### 5.3.1 Metodologia

Para a geração dos gráficos temporais, foi utilizado o *software PowerBI*, que utilizava os ficheiros *.csv* gerados e organizados previamente vindos da base de dados, contendo todas as informações em ficheiros únicos. Os gráficos gerados são: gráficos circulares e de tabelas.

### 5.3. Gráficos temporais (*TripAdvisor* apenas)

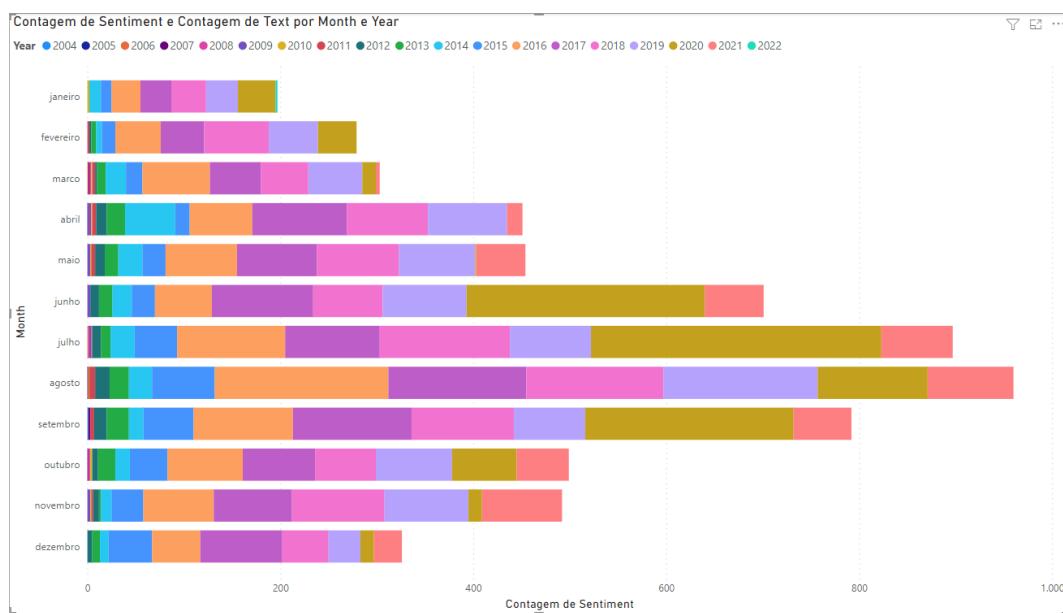
Os quais demonstram a quantidade de sentimentos e *keywords* usadas ao longo do tempo por cada hotel, divididos por anos e meses e também por cada hotel.

#### 5.3.2 Execução

Os ficheiros *.csv* que contêm as informações relativas a todas as *keywords* e sentimentos, foram importados para o *software PowerBI* e posteriormente organizados da maneira que o grupo achou mais conveniente para que os gráficos ficasse o mais apresentáveis e visivelmente mais fáceis para analisar os dados. Posteriormente os gráficos circulares e de tabelas foram exportados para *.jpg* e guardados.

#### 5.3.3 Resultados

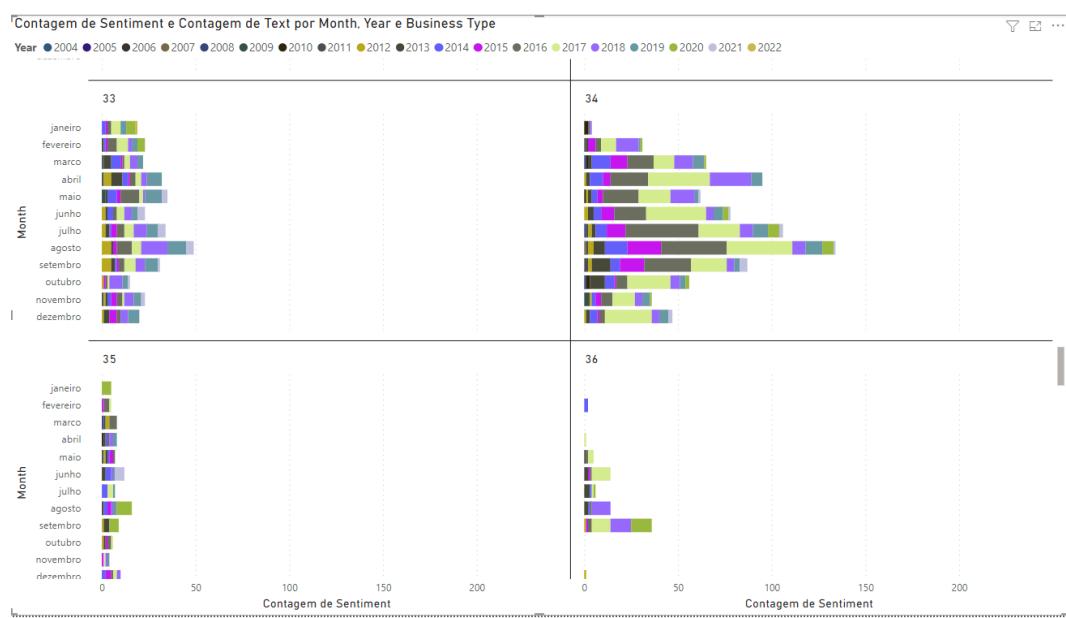
Como se podem verificar nestas três seguintes imagens (5.4, 5.5 e 5.6), os gráficos temporais apresentam um desenvolvimento de sentimentos e *keywords* ao longo do tempo bastante positivo revelando-se um óptimo ponto para o nosso turismo e *keywords* que mostram bastante agrado.



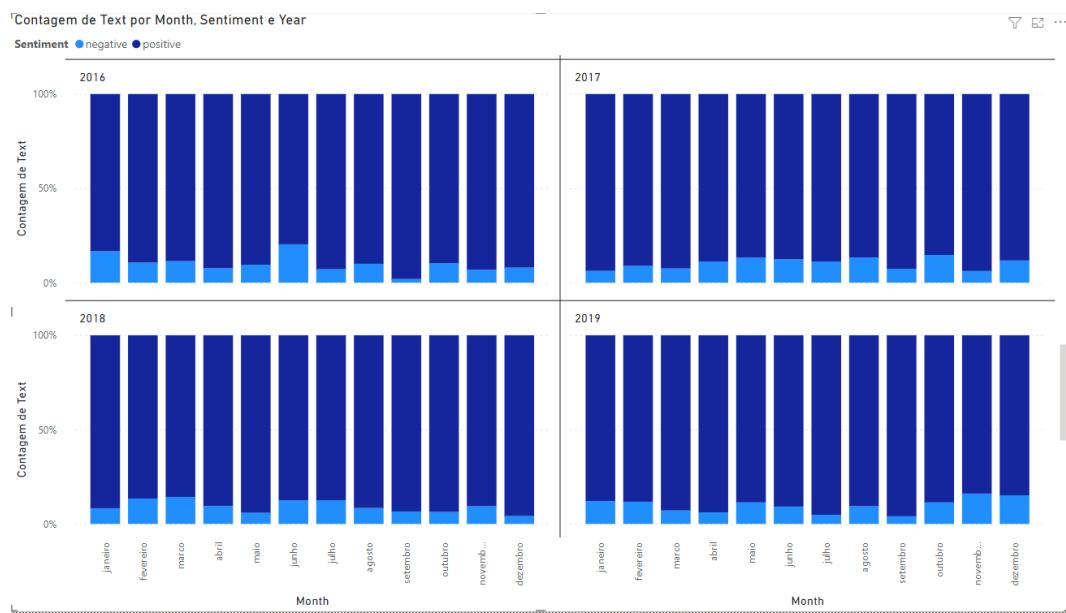
**Figura 5.4:** Gráfico de tabelas gerado baseando-se em todos os sentimentos dados da plataforma *TripAdvisor* ao longo dos anos

## 5. GERAÇÃO DE GRÁFICOS

---



**Figura 5.5:** Gráfico de tabelas gerado baseando-se em todos os sentimentos da plataforma TripAdvisor referente a cada hotel com o decorrer dos anos



**Figura 5.6:** Gráfico de tabelas gerado baseando-se na quantidade de sentimentos da plataforma TripAdvisor positivos e negativos ao longo do tempo para cada hotel

# Capítulo 6

## Análise dos dados obtidos

### 6.1 Resultados obtidos

Tendo em vista os gráficos gerados por meio das bibliotecas *MatplotLib*, *Wordcloud* [gfg3] [va1] e também os do software *PowerBI* foi possível reunir um grande conjunto de informação para realizar uma análise dos melhores pontos turísticos que todos os estabelecimentos locais, como hotéis, restaurantes e atracções, para além do património histórico-cultural podem oferecer e assim melhorar possíveis pontos negativos e positivos, ou então prever quais as atracções/hotéis/restaurantes que mais cativam os turistas.

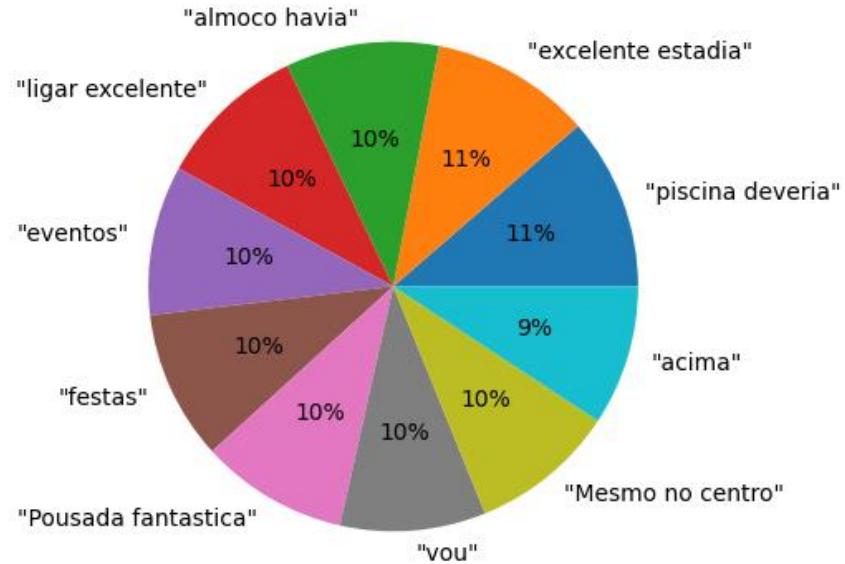
Assim sendo, foram gerados gráficos para cada hotel/ restaurante/ atracção dos websites *Tripadvisor*, *Booking* e *Zomato* usando as bibliotecas mencionadas com o intuito dos resultados serem mais facilmente visíveis com as *keywords* mais usadas para mencionar cada um dos pontos turísticos, assim como um mapa da cidade de Beja que contém todas as *keywords* e as percentagens entre sentimentos positivos e negativos.

Por fim foram gerados também gráficos com as informações ao longo do tempo dos mesmos websites referidos, acerca dos sentimentos de cada ponto turístico utilizando o software *PowerBI*.

#### 6.1.1 Resultados de totais

Nas figuras apresentadas abaixo (**6.1**, **6.2** e **6.3**) são mostrados alguns resultados gerados pelas bibliotecas mencionadas, dos quais podemos notar que existe uma maioria para a quantidade de sentimentos positivos em relação aos negativos e que a maior parte das *keywords* são também positivas. Porém, estes valores são retirados no momento em que a extracção dos dados foi realizada e não é possível verificar à medida do tempo como esses valores foram surgindo. Os valores entre restaurantes/hotéis/atracções é bastante semelhante entre si e então foi decidido que só iria ser mostrado alguns exemplos da realização desta etapa e todos os resultados ficariam mostrados nos anexos.

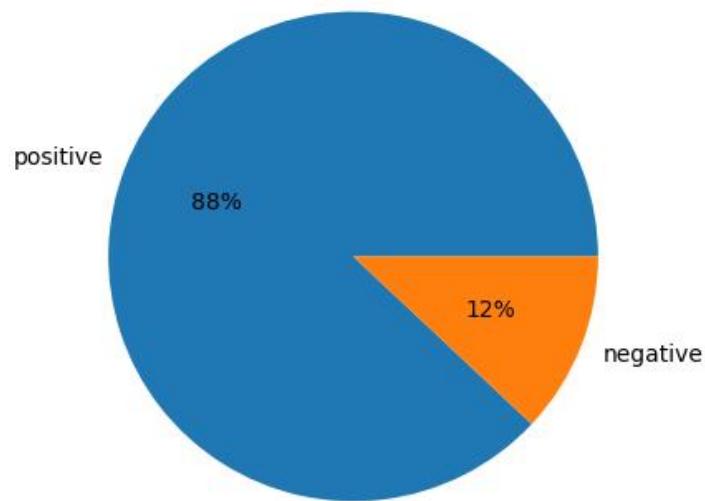
## 6. ANÁLISE DOS DADOS OBTIDOS



**Figura 6.1:** Gráfico circular com as *keywords* mais usadas para a Pousada Convento Beja



**Figura 6.2:** Mapa de Beja com as *keywords* mais usadas para a Pousada Convento Beja no seu interior



**Figura 6.3:** Gráfico circular a representar a diferença entre os sentimentos positivos e negativos na Pousada Convento Beja

## 6. ANÁLISE DOS DADOS OBTIDOS

### 6.1.2 Resultados temporais

As figuras apresentadas desta vez foram elaboradas com o valor temporal bastante visível, sendo possível verificar desta vez uma evolução com o decorrer do tempo dos valores apresentados, assim como a forte diferença entre a quantidade de sentimentos escritos em meses onde um grande número de pessoas adere aos serviços, como no verão, Páscoa ou Natal.

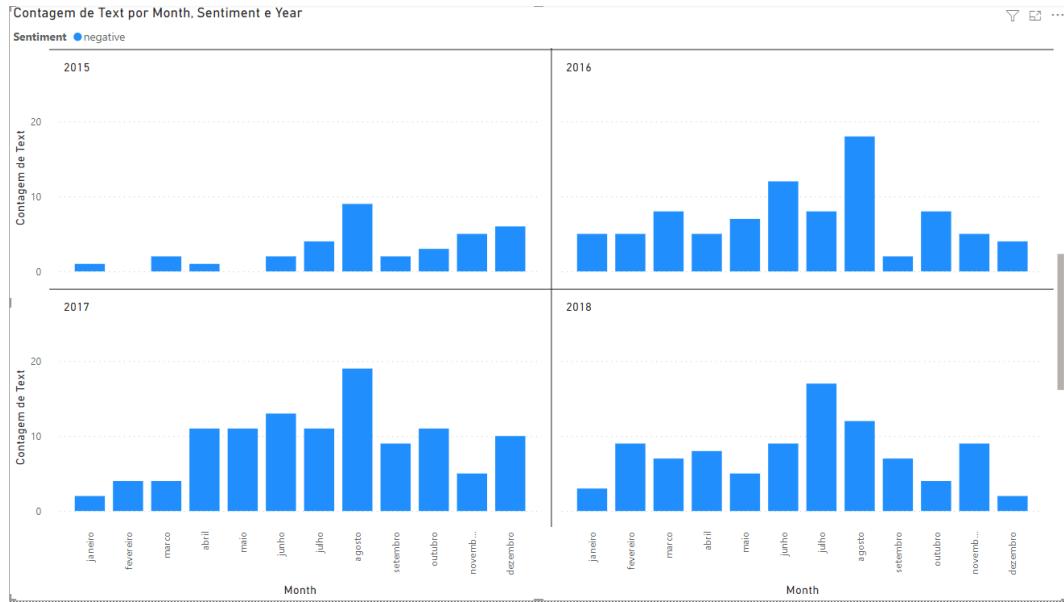


Figura 6.4: Gráfico quantitativo de sentimentos negativos ao longo do ano

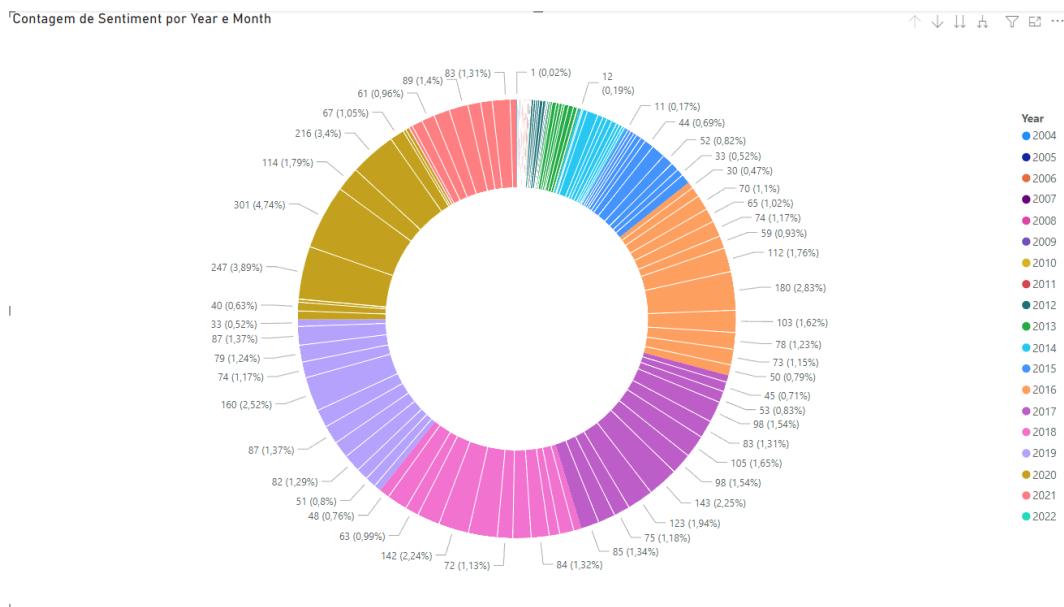
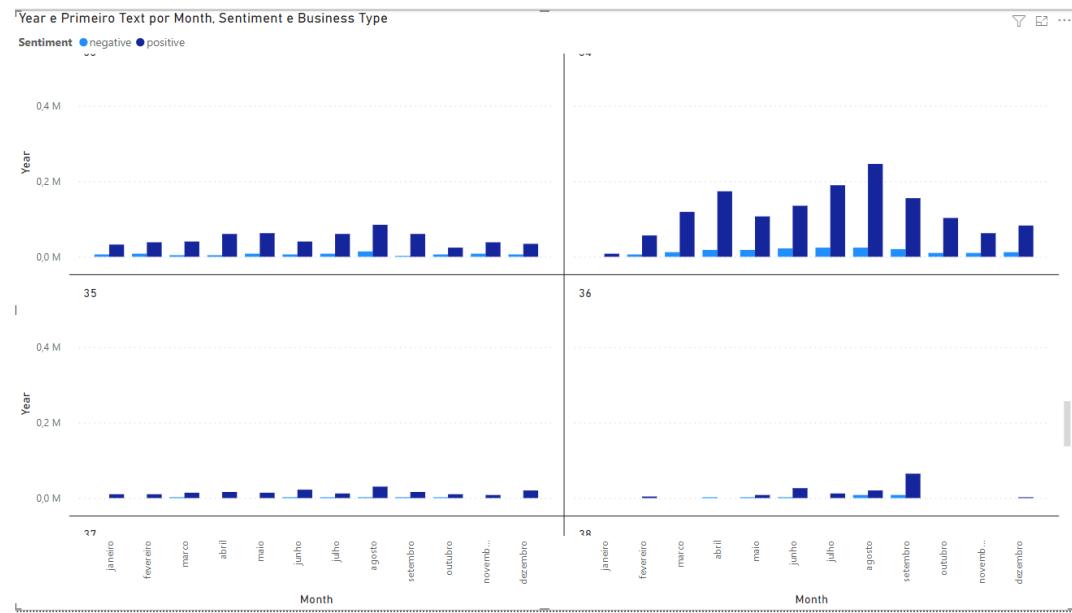


Figura 6.5: Gráfico circular com a quantidade de reviews ao longo do ano

## 6.1. Resultados obtidos



**Figura 6.6:** Gráfico com a quantidade de sentimentos ao longo do ano

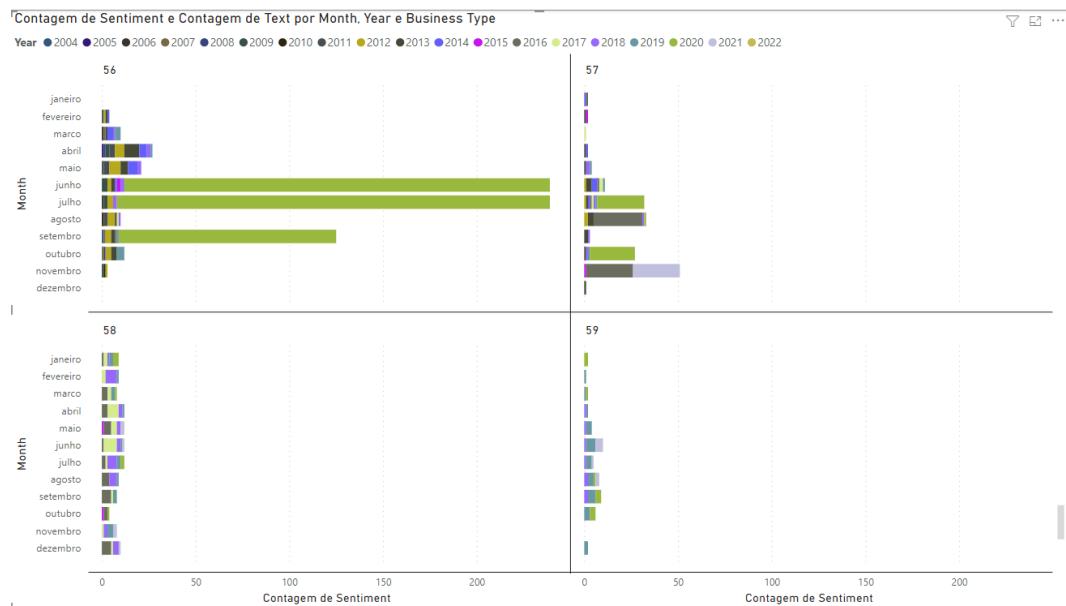
## 6. ANÁLISE DOS DADOS OBTIDOS

---

### 6.2 Análise dos resultados

Graças a estes gráficos podemos concluir que as opiniões acerca do turismo cultural na zona alentejana é bastante positiva, mostrando uma enorme maioria de comentários positivos contra uma pequena quantidade de comentários negativos **como mostra esta figura**. Podemos também notar que existem muitas mais pessoas a dar as suas opiniões em meses como Junho, Julho e Agosto, muito possivelmente devido á abertura das épocas balneares que movem grandes grupos de turistas nacionais e estrangeiros a fazerem férias pelas zonas costeiras que o Alentejo consegue fornecer com enorme facilidade graças ás magnificas praias na sua zona costeira. Por fim também é possível notar a evolução no número de opiniões com o decorrer dos anos e com a popularidade que o *website* vai conseguindo, já que no começo o número de opiniões é baixo, porém com o passar dos anos começa a subir em elevado número.

É interessante também realçar um detalhe acerca de um gráfico em específico que o grupo decidiu não passar em branco. Olhemos para o seguinte gráfico **6.7**.



**Figura 6.7:** Gráfico quantitativo de sentimentos ao longo do ano por cada estabelecimento (enumerado por **id**) com valores diferentes

O Palácio dos Maldonados (**id 56**) contém valores interessantes nos meses de Junho e Julho. São interessantes uma vez que durante o ano de 2020 o país se encontrava em confinamento devido á COVID-19, não sendo possível que ajuntamentos fossem realizados, porém foi verificado através do gráfico que o mesmo não se parece verificar já que existe uma brutal subida no número de pessoas a dar a sua opinião acerca da estadia que realizou, o que dá a entender que esse estabelecimento continuou a realizar as suas tarefas com normalidade ao contrário de outros que provavelmente seguiram as normas recomendadas.

## Capítulo 7

### Conclusão

Uma análise de dados apresenta sempre uma oportunidade de crescimento, em especial num meio de turismo rural (ou semi-rural) em que os habituais clientes podem (e vão) usar redes sociais e plataformas de comentário de prestação de serviço, para agregar dados comportamentais e percepção pública das ofertas turísticas.

Neste trabalho foi apresentado um agregado exemplar desse tipo de dados e análise o qual foi feito na melhor das formas, dado o conhecimento a nós atribuído pela escola e o tempo disponível. O qual apresentou pouco de informação nova, mas colabora com a percepção pública que se sente nos meios de comunicação verbal subjectiva em locais públicos, ou seja, existe a tendência de haver um maior número de clientes durante as típicas alturas de férias, referimos o ano novo, a Páscoa e em especial o verão.

O que foi dito acima foram as tendências observadas na análise dos dados obtidos nos gráficos gerados para análise temporal, com os dados da plataforma mais completa deste estudo, o *TripAdvisor*. O único caso mais em especial, foi a explosão de *reviews* em Junho e Julho de 2020, que coincide com a abertura da época balnear após desconfinamento (COVID-19) no dia 6 de Junho de 2020, que embora fazendo *cross-referencing* com os dados das *keywords* da mesma análise temporal, não nos dê informação relevante, a coincidência é meramente atractiva e oferece uma fácil explicação.

Sendo assim podemos concluir que foi um trabalho que corrobora as noções e conhecimento público da área, tal como os dados das análises anuais oferecidas pelo *Booking*. Graças a isto obtemos uma noção da metodologia desta área profissional e os nossos avaliados terão uma noção da qualidade de trabalho produzida por nós.

Sumarizamos assim que nesta secção de turismo rural (e semi-rural), da região de Beja, tem por norma uma boa prestação de serviço, ou pelo menos essa noção é transposta na mente popular de quem visita, e que não existem anomalias na distribuição de visitantes nem de tendências sentimentais ao longo do ano nem ao longo das décadas, mantendo-se previsível mas agradável.

**UC úteis para a elaboração deste Projecto:** *PE, BD1, LP, ES, TWAM, SI*



# Apêndices



## Apêndice I

### *Jupyter Notebook* da Extração de Palavras-Chave

Nas páginas seguintes está incluída uma renderização do *notebook* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extração de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

# Keyword Extraction

In this notebook, we will use the [Keyword Extraction](#) technique to extract keywords from text. We will use the [YAKE!](#) library to extract keywords from text.

[YAKE!](#) is a library that can be used to extract keywords from text made by portuguese authors (and a japanese) from Polytechnic Institute of Tomar, University of Beira Interior, University of Porto, INESC TEC (and Kyoto University).

## Importing the libraries

Here we import the libraries we will use.

```
In [ ]: import os
import warnings
import yake
import pandas
```

## Functions

Now we define the functions we will use. We will use the following functions:

- `extract_keywords`: to extract keywords from text.
- `extract_keywords_chunks`: to extract keywords from text using chunks.
- `import_csv`: to import a csv file.
- `import_csv_chunks`: to import a csv file using chunks.
- `get_keywords`: to get the keywords from a list of keywords.
- `get_keywords_chunks`: to get the keywords from a list of keywords using chunks.
- `get_keywords_dir`: to get the keywords from a directory of files.
- `get_keywords_dir_chunks`: to get the keywords from a directory of files using chunks.
- `get_keywords_zomato_dir`: to get the keywords from zomato folder.

### Function: extract\_keywords

This function extracts keywords from text using YAKE! library. It takes as input a text and returns a list of keywords.

```
In [ ]: def extract_keywords(df):
    keywords = []
    for i in range(0, len(df)):
        review = df["Avaliacoes"][i]
        keywords.append(yake.KeywordExtractor(lan="pt").extract_keywords(review))
    return keywords
```

### Function: import\_csv

This function imports a csv file. It takes as input a csv file and returns a dataframe.

```
In [ ]: def import_csv(path):
    df = pandas.read_csv(path, encoding="utf-8")
    return df
```

### Function: get\_keywords

This function gets all the csv files in a directory and returns a list of keywords from the dataframes.

```
In [ ]: def get_keywords(path):
    keywords = []
    for file in os.listdir(path):
        if file.endswith(".csv") and not file.startswith("list"):
            df = import_csv(path + "/" + file)
            keywords.append(extract_keywords(df))
    return keywords
```

### Function: get\_keywords\_dir

This function gets all the csv files in a directory and get a list of keywords from the dataframes, then it writes the keywords in a csv file.

```
In [ ]: def get_keywords_dir(path1, path2, name):
    current_dir = os.getcwd()
    path = current_dir + path1
    keywords = get_keywords(path)
    # print(keywords)
    i = 0
    for keyword in keywords:
        # print(keyword)
        with open(
            current_dir + path2 + name + str(i) + ".csv",
            "w",
        ) as f:
            f.write("Expressao, Frequencia\n")
            for k in keyword:
                for word in k:
                    f.write(
                        str(word)
                        .replace("(", "")
                        .replace(")", "")
                        .replace("\u2010", "-")
                        + "\n"
                    )
        i += 1
```

### Function: get\_keywords\_zomato\_dir

This function gets all the csv files from the zomato directory and get a list of keywords from the dataframes, then it writes the keywords in a csv file.

```
In [ ]: def get_keywords_zomato_dir(path1, path2, name):
    current_dir = os.getcwd()
    path = current_dir + path1
    keywords = get_keywords(path)
    # print(keywords)
    i = 0
    restaurantes = [0, 1, 2, 12, 13, 14, 15]
    for keyword in keywords:
        # print(keyword)
        with open(
            current_dir + path2 + name + str(restaurantes[i]) + ".csv",
            "w",
        ) as f:
            f.write("Expressao, Frequencia\n")
            for k in keyword:
                for word in k:
                    f.write(
                        str(word)
                        .replace("(", "")
                        .replace(")", "")
                        .replace("\u2010", "-")
                        + "\n"
                    )
        i += 1
```

## Execution

Now we execute the code. We will use the last set functions to get the keywords from the reviews.

```
In [ ]: warnings.filterwarnings("ignore")

get_keywords_dir("../scrapes/booking/hotels", "/booking/hotels/", "hotel")
get_keywords_zomato_dir(
    "../scrapes/zomato/restaurantes",
    "/zomato/restaurantes/",
    "restaurante",
)
get_keywords_dir(
    "../scrapes/tripadvisor/restaurants",
    "/tripadvisor/restaurants/",
    "restaurant",
)
get_keywords_dir(
    "../scrapes/tripadvisor/activities",
    "/tripadvisor/activities/",
    "place",
)
get_keywords_dir("../scrapes/tripadvisor/hotels", "/tripadvisor/hotels/", "hotel")
```

[ Loading IPython/Jupyter output/CommonHTML /fonts/TeXfontdata is ]



## Apêndice II

### *Jupyter Notebook da Analise de Sentimentos via Naïve Bayes Classifier*

Nas paginas seguintes está incluída uma renderização do *notebook* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extracção de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

# Sentiment Analysis

In this notebook, we will use a dataset of movie reviews to train a model to predict whether a review is positive or negative. We will use the [ultc-movies.csv](#) to train the model (this is not on the Github repo due to its size).

To train the model, we will use the `sklearn.feature_extraction.text.CountVectorizer` to create a bag of words representation of the reviews with a `nltk.stem.SnowballStemmer` to stem the words, and `sklearn.naive_bayes.MultinomialNB` as our machine learning model.

We will use the `sklearn.metrics.classification_report` to evaluate the model.

## Importing the libraries

Here we import the libraries we will use.

```
In [ ]: import os
import re
import warnings
import unicodedata
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
```

## Creating a custom tokenizer

We'll need a custom tokenizer that stems the words in our reviews. We will use the `nltk.stem.SnowballStemmer` to stem the words.

This tokenizer will be used to create a bag of words representation of the reviews.

```
In [ ]: class StemmerTokenizer:
    def __init__(self):
        self.stemmer = SnowballStemmer("portuguese")
    def __call__(self, doc):
        return [self.stemmer.stem(t) for t in word_tokenize(doc)]
```

## Functions

Now we will define the functions we will use. We will use the following functions:

- `create_dataframe`: to create a dataframe from the csv file
- `load_directory`: to load the data from the directory
- `get_training_data`: to get the training data
- `drop_useless_columns`: to drop the columns that we don't need
- `get_csv`: to get the csv file
- `filter_string`: to filter the string
- `integer`: to convert the string to an integer
- `get_count_vectorizer_with_stopwords`: to get the count vectorizer with stopwords
- `normalize`: to normalize the data
- `emotion_from_int`: to get the emotion from the integer
- `predict`: to predict the emotion

### Function: `create_dataframe`

This function will create a dataframe from the csv file.

```
In [ ]: def create_dataframe(filename):
    nan_value = float("NaN")
    df = pd.read_csv(filename)
    df.replace("", nan_value, inplace=True)
    df.dropna(how="any", inplace=True)
    return df
```

### Function: load\_directory

This function will load the data from the directory.

```
In [ ]: def load_directory(directory):
    files = []
    for filename in os.listdir(directory):
        if filename.endswith(".csv") and not filename.startswith("list"):
            files.append(filename)
    return files
```

### Function: get\_training\_data

This function will get the training data.

```
In [ ]: def get_training_data():
    df = create_dataframe(
        "models/utlc_movies.csv"
    ) # original_index, review_text, review_text_processed, review_text_tokenized, polarity, rating, kfold_polarity, kfold_rating
    df = drop_useless_columns(df)
    df = df.rename(columns={"polarity": "Class", "review_text_processed": "Data"})
    df = df.reindex(columns=["Class", "Data"])
    return df.sample(frac=1).reset_index(drop=True)
```

### Function: drop\_useless\_columns

This function will drop the columns that we don't need.

```
In [ ]: def drop_useless_columns(df):
    df.drop(
        [
            "original_index",
            "review_text",
            "review_text_tokenized",
            "rating",
            "kfold_polarity",
            "kfold_rating",
        ],
        axis=1,
        inplace=True,
    )
    return df
```

### Function: get\_csv

This function will get the csv file.

```
In [ ]: def get_csv(path):
    df = create_dataframe(path)
    return df
```

### Function: filter\_string

This function will filter the string.

```
In [ ]: def filter_string(df, column):
    ret = (
        df[column]
        .apply(lambda x: re.sub("[^a-zA-Z]", " ", x))
        .apply(lambda x: re.sub(" +", " ", x))
        .apply(lambda x: x.strip())
        .apply(lambda x: x.lower())
        .apply(lambda x: normalize(x))
        .values
    )
    return ret
```

### Function: integer

This function will convert the string to an integer.

```
In [ ]: def integer(x):
    return int(x)
```

### Function: get\_count\_vectorizer\_with\_stopwords

This function will get the count vectorizer with stopwords.

```
In [ ]: def get_count_vectorizer_with_stopwords():
    return CountVectorizer(
        tokenizer=StemmerTokenizer(),
        ngram_range=(1, 2),
        stop_words=stopwords.words("portuguese"),
    )
```

### Function: normalize

This function will normalize the data.

```
In [ ]: def normalize(text):
    text = (
        unicodedata.normalize("NFKD", text)
        .encode("ascii", "ignore")
        .decode("utf-8", "ignore")
    )
    return text
```

### Function: emotion\_from\_int

This function will get the emotion from the integer.

```
In [ ]: def emotion_from_int(x):
    if x == 0:
        return "Negative"
    elif x == 1:
        return "Positive"
    else:
        return "Unknown"
```

### Function: predict

This function will predict the emotion.

```
In [ ]: def predict(model, vec, directory):
    files = load_directory(directory)
    for filename in files:
        df_predict = get_csv(directory + "/" + filename)
        predict_data = filter_string(df_predict, "Avaliacoes")
        # print("Loaded data to predict")
        reviews = []
        sentiments = []
        for review in predict_data:
            sentiment = emotion_from_int(
                model.predict(vec.transform([review]).toarray())[0]
            )
            # print("Model predicts that: " + str(review) + " is " + str(sentiment))
            reviews.append(str(review))
            sentiments.append(str(sentiment))
        with open(
            directory.replace("scrapes", "sentimentanalysis") + "/" + filename,
            "w",
            encoding="utf-8",
        ) as f:
            f.write("Sentiment, Review\n")
            for i in range(len(reviews)):
                f.write("'" + sentiments[i] + "', '" + reviews[i] + "'\n")
```

## Execution

Now we will execute the code.

```
In [ ]: warnings.filterwarnings("ignore")
df = get_training_data()
```

```

df_test = df.sample(frac=0.1, random_state=42).reset_index(drop=True)

df = df[:15000]
df_test = df_test[:5000]

train_data, train_class = (
    filter_string(df, "Data"),
    df["Class"].apply(lambda x: integer(x)).values,
)
print("Loaded training data")

test_data, test_class = (
    filter_string(df_test, "Data"),
    df_test["Class"].apply(lambda x: integer(x)).values,
)
print("Loaded test data")

vec = get_count_vectorizer_with_stopwords()
print("Created vectorizer")

train_data = vec.fit_transform(train_data).toarray()
print("Transformed training data")

test_data = vec.transform(test_data).toarray()
print("Transformed test data")

model = MultinomialNB()
print("Created model")

model.fit(train_data, train_class)
print("Trained model")

print(
    "Tested model: " + str(model.score(test_data, test_class) * 100) + "%" + " accuracy"
)

```

Prediction of the emotion of the reviews of various establishments of various types and platforms and export the results to a csv file.

In [ ]:

```

predict(model, vec, "../scrapes/booking/hotels")
predict(model, vec, "../scrapes/zomato/restaurante")
predict(model, vec, "../scrapes/tripadvisor/hotels")
predict(model, vec, "../scrapes/tripadvisor/restaurants")
predict(model, vec, "../scrapes/tripadvisor/activities")

```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



## Apêndice III

# *Jupyter Notebook da Analise de Sentimentos via BERT Transformer Pipelines*

Nas paginas seguintes está incluída uma renderização do *notebook* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extracção de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

# Sentiment Analysis (using Transformers and PyTorch)

This notebook demonstrates how to use `transformers` to perform sentiment analysis. These transformers use the PyTorch library to perform the actual computation.

## Imports

Here we import the required packages.

```
In [ ]: import os
import warnings
import pandas
import torch
from transformers import pipeline
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
```

## Functions

Here we define the functions that we will use in this notebook. We will use these functions to perform sentiment analysis. Which are:

- `get_pipeline`: This function creates a pipeline that will be used to perform sentiment analysis.
- `get_dir`: This function returns the directory where the data is stored.
- `get_reviews`: This function returns the reviews.
- `get_sentiments`: This function returns the sentiments of the reviews.
- `export`: This function exports the sentiment analysis.
- `analyse_directory`: This function analyses the sentiment analysis of the reviews in a directory.

### Function: `get_pipeline`

This function creates a pipeline that will be used to perform sentiment analysis.

```
In [ ]: def get_pipeline():
    model = AutoModelForSequenceClassification.from_pretrained(
        "gchhablani/bert-base-cased-finetuned-sst2"
    )
    tokenizer = AutoTokenizer.from_pretrained(
        "gchhablani/bert-base-cased-finetuned-sst2", do_lower_case=False
    )
    senti_pipeline = pipeline(
        "sentiment-analysis", model=model, tokenizer=tokenizer, truncation=True
    )
    return senti_pipeline
```

### Function: `get_dir`

This function returns the directory where the data is stored.

```
In [ ]: def get_dir(path):
    files = []
    for file in os.listdir(path):
        if file.endswith(".csv") and not file.startswith("list"):
            files.append(file)
    return files
```

### Function: `get_reviews`

This function returns the reviews.

```
In [ ]: def get_reviews(df):
    reviews = []
    for i in range(0, len(df)):
        reviews.append(str(df["Avaliações"][i]))
    return reviews
```

### Function: `get_sentiments`

This function returns the sentiments of the reviews.

```
In [ ]: def get_sentiments(reviews, senti_pipeline):
    sentiments = []
    for review in reviews:
        sentiments.append(str(senti_pipeline(review)[0].get("label")))
    return sentiments
```

### Function: export

This function exports the sentiment analysis.

```
In [ ]: def export(sentiments, reviews, path, name):
    df = pandas.DataFrame({"sentiment": sentiments, "review": reviews})
    path = path.replace("../scrapes/", "/")
    df.to_csv(str(path) + str(name), index=False)
```

### Function: analyse\_directory

This function analyses the sentiment analysis of the reviews in a directory.

```
In [ ]: def analyse_directory(path, senti_pipeline):
    files = get_dir(path)
    for file in files:
        df = pandas.read_csv(path + file, encoding="utf-8")
        reviews = get_reviews(df)
        sentiments = get_sentiments(reviews, senti_pipeline)
        export(sentiments, reviews, path, file)
```

## Execution

Here we execute the notebook. First we create the pipeline.

```
In [ ]: senti_pipeline = get_pipeline()
```

Then we get the directory where the data is stored. Then we get the reviews and the sentiments. Finally we export the sentiment analysis.

```
In [ ]: current_dir = os.getcwd()
path = current_dir + "../scrapes/"
analyse_directory(path + "booking/hotels/", senti_pipeline)
analyse_directory(path + "zomato/restaurantes/", senti_pipeline)
analyse_directory(path + "tripadvisor/hotels/", senti_pipeline)
analyse_directory(path + "tripadvisor/activities/", senti_pipeline)
analyse_directory(path + "tripadvisor/restaurants/", senti_pipeline)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



## Apêndice IV

### *Jupyter Notebooks do Webscraping dos hotéis do Booking*

Nas páginas seguintes está incluída uma renderização dos *notebooks* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extração de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

O primeiro detalha a busca dos hotéis e o segundo apenas os *reviews* de cada um dos hotéis encontrados no primeiro.

Importing some libraries that we need to make the webscrapping of the booking.com

```
In [ ]: from bs4 import BeautifulSoup
import requests
import pandas as pd
```

We need to create the request to make the website send the information: To make that we use the library `requests` and the `BeautifulSoup` and the inspect tool to extract que name of the classes and the headers needed

```
In [ ]: headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4649.116 Safari/537.36'
url = "https://www.booking.com/searchresults.pt-pt.html?aid=375654&label=msn-jrwrFdUb9zKNuCHIkGmz2g-8074541083442"
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.content, 'lxml')
```

And the the arrays to receive the information from the website like the name of the hotels, the ratings... Every hotel are from Beja.

```
In [ ]: hotel = []
badge = []
title = []
reviews = []
price = []
for item in soup.select('.fb3c4512b4'):
    try:
        hotel.append(item.select('.fde44d7ef')[0].get_text().strip())
        badge.append(item.select('.9c5f726ff')[0].get_text().strip())
        title.append(item.select('.192b3a196')[0].get_text().strip())
        reviews.append(item.select('.le6021d2f')[0].get_text().strip())
        price.append(item.select('.e885fdc12')[0].get_text().strip())
    except Exception as e:
        print('')
```

Only in case any of the arrays are different in size

```
In [ ]: # bad code
length = len(price)
if length > len(reviews):
    length = len(reviews)
if length > len(hotel):
    length = len(hotel)
if length > len(badge):
    length = len(badge)
if length > len(title):
    length = len(title)
```

Saving the respective information and extracting to .csv

```
In [ ]: d1 = {'Hotel': hotel[:length], 'Classificação': badge[:length],
          'Suma': title[:length], 'Avaliações': reviews[:length], 'Preço': price[:length]}
df = pd.DataFrame.from_dict(d1)
print(df)
df.to_csv('listtable.csv')
```

The same libraries were used when webscraping was performed for hotels

```
In [ ]: from bs4 import BeautifulSoup
import requests
import pandas as pd
```

In this case it took a larger amount of headers to access the site information.

```
In [ ]: headers = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET",
    "Access-Control-Allow-Headers": "Content-Type",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "accept-language": "en-GB,en;q=0.9,en-US;q=0.8",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.122 Safari/537.36"
}
url = "https://www.booking.com/searchresults.pt.pt.html?aid=375654&label=msn-jrwrFdUb9zKNuCHIkGmz2g-8074541083442"
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.content, 'lxml')
```

To access the respective comments of each hotel it was necessary to build a new link to be performed the webscraping, for this was used a common part for all links and added the name of the hotel, to access your comments specifically

```
In [ ]: reviews_links = []
for link in soup.findAll('a', {'class': 'fb01724e5b'}):
    a = link['href']
    hotel = a.split('/')[5].split('?')[0]
    a = 'https://www.booking.com/reviews/pt/hotel/' + hotel
    reviews_links.append(a)
```

In this last part, the comments relating to each website were extracted and the .csv of each of them was created.

```
In [ ]: count = 0
allreviews = []

for link in reviews_links:
    try:
        response2 = requests.get(link, headers=headers)
        soup2 = BeautifulSoup(response2.content, 'lxml')
        for r in soup2.findAll('span', {'itemprop': 'reviewBody'}):
            try:
                rev = r.text
                allreviews.append(rev + '\n')
            except:
                pass
    except:
        pass
    count += 1
    if allreviews != []:
        seen = set()
        allreviews = [item for item in allreviews if not(
            tuple(item) in seen or seen.add(tuple(item)))]
        dfr = pd.DataFrame.from_dict({'Avaliações': allreviews})
        print(dfr)
        dfr.to_csv('hotel' + str(count) + '.csv')
        allreviews = []
```



## Apêndice V

### *Jupyter Notebook do Webscraping dos reviews de Restaurantes do Zomato*

Nas páginas seguintes está incluída uma renderização do *notebook* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extração de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

## Zomato

teste 123

```
In [ ]: from bs4 import BeautifulSoup
import requests
import pandas as pd

texto

In [ ]: headers = {
    # "Access-Control-Allow-Origin": "*",
    # "Access-Control-Allow-Methods": "GET",
    # "Access-Control-Allow-Headers": "Content-Type",
    # "accept": "/*",
    "User-Agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko)"
}

# url = "https://www.zomato.com/beja/beja-restaurants"
# response = requests.get(url, headers=headers)
# response.status_code
#soup = BeautifulSoup(response.content, 'lxml')
soup = BeautifulSoup(open(r"C:\Users\vitor\Desktop\pi2021\projeto\webscrape\scrapes\zomato\restaurantes\zomato-t

In [ ]: restaurant = []
for name in soup.findAll('h4',{'class':'sc-1hp8d8a-0'}):
    restaurant.append(name.text.strip())
print(len(restaurant))

In [ ]: type = []
for name in soup.findAll('p',{'class':'jaKOQh'}):
    print(name)
    type.append(name.text.strip())
for name in soup.findAll('p',{'class':'kegdaG'}):
    print(name)
    type.append(name.text.strip())
print(len(type))

In [ ]: price = []
for p in soup.findAll('p',{'class':'ftdqla'}):
    price.append(p.text.replace('€ para dois','').strip())
for p in soup.findAll('p',{'class':'kOONhy'}):
    price.append(p.text.replace('€ para dois','').strip())
print(len(price))

In [ ]: #bad code

length = len(price)
if length > len(type):
    length = len(type)
if length > len(restaurant):
    length = len(restaurant)

In [ ]: d1 = {'Restaurante': restaurant[:length], 'Tipo':type[:length], 'Preço':price[:length]}
df = pd.DataFrame.from_dict(d1)
print(df)
df.to_csv('listtable.csv')

In [ ]: reviews_links = []
for link in soup.findAll('a', {'class': 'ieKty'}):
    a = link['href']
    reviews_links.append(a.replace('/info', '/reviews'))
for link in soup.findAll('a', {'class': 'jjSACU'}):
    a = link['href']
    reviews_links.append(a.replace('/info', '/reviews'))
# print(reviews_links)

In [ ]: count = 0
allreviews = []
```

```
for link in reviews_links:
    try:
        response2 = requests.get(link, headers=headers)
        soup2 = BeautifulSoup(response2.content, 'xml')
        for r in soup2.findAll('p'): # sempre a mudar a class, vai sofrer ETL
            try:
                rev = r.text
                # print(rev)
                allreviews.append(rev + '\n')
            except:
                pass
    except:
        pass
    count += 1
if allreviews != []:
    seen = set()
    allreviews = [item for item in allreviews if not(
        tuple(item) in seen or seen.add(tuple(item)))]
    dfr = pd.DataFrame.from_dict({'Avaliações': allreviews})
    print(dfr)
    dfr.to_csv('restaurante' + str(count) + '.csv')
    allreviews = []
```



## Apêndice VI

### *Jupyter Notebooks do Webcrawing dos reviews dos hotéis, atracções e restaurantes do TripAdvisor*

Nas paginas seguintes está incluída uma renderização dos *notebooks* de *Jupyter* em que se detalha (em inglês) os passos das rotinas de extracção de palavras-chave, comentando de forma simples o funcionamento das funções de código usadas.

O primeiro extraiu os hotéis, o segundo as atracções e o terceiro os restaurantes (e os seus respectivos *reviews*).

# TripAdvisor

## Webscraping

### Imports

First we start with the imports. We need essentially three (or four) main libraries to work this out; these are:

- requests (to fetch the website)
- lxml (a faster html parser to speed bs4)
- bs4 (a.k.a beautiful soup, a web scraping library)
- pandas (a maths oriented data(set) manipulation library)

Since requests uses urllib3 as a dependency, we can import it first to configure it to suppress the annoying warning about the "insecure" connection (lack of SSL).

```
In [ ]: import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
import requests
from bs4 import BeautifulSoup as soup
import pandas as pd
```

### Request configuration

We need to configure our request, specially in this case, since TripAdvisor won't send us a webpage if we at least not try to emulate a real browser. First we configure our headers, ripping the main headers from our browser, as seen in the Developer tools (F12) in Chromium (we used the new Microsoft Edge).

Then we request the webpage (Hotels in Beja, Portugal) with our headers attached, a timeout to stop if it takes too long (something is wrong), and verification is disabled (SSL). If the status code is OK (200), doesn't print.

After that we create a BeautifulSoup4 scrapable object with the html content of the page, using lxml.

```
In [ ]: headers = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'GET',
    'Access-Control-Allow-Headers': 'Content-Type',
    'accept': '/*',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'en-GB,en;q=0.9,en-US;q=0.8',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.110 Safari/537.36'}
url = "https://www.tripadvisor.pt/Hotels-g189102-Beja_Beja_District_Alentejo-Hotels.html"
req = requests.get(url,headers=headers,timeout=5,verify=False)
req.status_code
bsobj = soup(req.content, 'lxml')
```

### Scraping

Now we start scraping. First we start getting hotel names.

```
In [ ]: hotel = []
for name in bsobj.findAll('div',{'class':'listing_title'}):
    hotel.append(name.text.strip())
print(len(hotel))
```

Then their ratings.

```
In [ ]: ratings = []
for rating in bsobj.findAll('a',{'class':'ui_bubble_rating'}):
    ratings.append(rating['alt'])
print(len(ratings))
```

The number of reviews (they have a big issue).

```
In [ ]: reviews = []
for review in bsobj.findAll('a',{'class':'review_count'}):
    reviews.append(review.text.strip())
print(len(reviews))
```

This number is referring to the TOTAL number of reviews. However, we can only scrape a single language, that's dependent of the domain/locale (.com .pt); there's no query parameter to change either the number here, or the sorting of reviews.

Now we get the prices.

```
In [ ]: price = []
for p in bsobj.findAll('div',{'class':'price-wrap'}):
    price.append(p.text.replace('€','').strip())
print(len(price))
```

Some of these will be empty, since the price is gotten via phone call.

Now, we get to the weird part: Reviews. These reviews are handled by the hotel page in weird ways:

- only four shown per subpage
- each subpage is counted via a multiple of five
- any multiple of five non-existent will not give 404, but redirect to the first subpage
- language selection via scraping is non-existent, no query parameters, only radio buttons with random labels, defaults chosen via domain/locale (.com .pt)

So the strategy found is:

- create a monstrous amount of subpage links (about 200)
- scrape all reviews, even repeated via the redirect to the first subpage
- later use sets, or dictionaries to remove duplicates

That was done with this awful looking but functional code.

```
In [ ]: links = []
for review in bsobj.findAll('a',{'class':'review_count'}):
    try:
        a = review['href']
        a = 'https://www.tripadvisor.pt'+ a
        c = a[:a.find('Reviews')+7] + ' -or' + a[(a.find('Reviews')+7):]
        links.append(c)
    for i in range(5,1000,5):
        b = a[:a.find('Reviews')+7] + '-or' + str(i) + a[(a.find('Reviews')+7):]
        links.append(b)
    except:
        pass
print(links)
```

Now, we get the smallest length number of the arrays regarding to the hotels, hoping we remove some of the (repeated) sponsorships.

```
In [ ]: # bad code
length = len(price)
if length > len(reviews):
    length = len(reviews)
if length > len(hotel):
    length = len(hotel)
if length > len(ratings):
    length = len(ratings)
```

And then create the ID table of the hotels with the most basic information in a pandas DataFrame, and export that one to a .csv file that we can use in Excel, PowerBI, ML libraries like Keras, Tensorflow, SciKitLearn can use.

```
In [ ]: d1 = {'Hotel':hotel[:length],'Estrelas':ratings[:length],'Avaliações':reviews[:length],'Preço':price[:length]}
df = pd.DataFrame.from_dict(d1)
print(df)
df.to_csv('listtable.csv')
```

Now the most horrible of codes presents you with the creations of various, separated .csv files with the scraped reviews, that we can use.

It iterates all the links and since there's 200 links per hotel, every 200 we use some list comprehension magic with sets to remove duplicates and export the DataFrame to a useful .csv file.

```
In [ ]: count = 0
count2 = 0
allreviews = []
for link in links:
    try:
        html2 = requests.get(link,headers=headers)
        bsobj2 = soup(html2.content,'lxml')
        for r in bsobj2.findAll('q'):
            try:
                rev = r.span.text.strip()
                allreviews.append(rev + '\n')
            except:
                pass
    except:
        pass
    count += 1
    if count == 200:
        count2 += 1
        df = pd.DataFrame(allreviews)
        df.to_csv('Reviews'+str(count2)+'.csv')
        allreviews = []
        count = 0
```

```
        except:  
            pass  
    except:  
        pass  
    count += 1  
    if count == 200:  
        seen = set()  
        allreviews = [item for item in allreviews if not(tuple(item) in seen or seen.add(tuple(item)))]  
        dfr = pd.DataFrame.from_dict({'Avaliações':allreviews})  
        print(dfr)  
        dfr.to_csv('hotel' + str(count2) + '.csv')  
        allreviews = []  
        count = 0  
        count2 += 1
```

# TripAdvisor

## Webscraping

### Imports

First we start with the imports. We need essentially three (or four) main libraries to work this out; these are:

- requests (to fetch the website)
- lxml (a faster html parser to speed bs4)
- bs4 (a.k.a beautiful soup, a web scraping library)
- pandas (a maths oriented data(set) manipulation library)

Since requests uses urllib3 as a dependency, we can import it first to configure it to suppress the annoying warning about the "insecure" connection (lack of SSL).

```
In [ ]: import urllib3
import requests
import re
from bs4 import BeautifulSoup as soup
import pandas as pd
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

### Request configuration

We need to configure our request, specially in this case, since TripAdvisor won't send us a webpage if we at least not try to emulate a real browser. First we configure our headers, ripping the main headers from our browser, as seen in the Developer tools (F12) in Chromium (we used the new Microsoft Edge).

Then we request the webpage (Attractions in Beja, Portugal) with our headers attached, a timeout to stop if it takes too long (something is wrong), and verification is disabled (SSL). If the status code is OK (200), doesn't print.

After that we create a BeautifulSoup4 scrapable object with the html content of the page, using lxml.

```
In [ ]: headers = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET",
    "Access-Control-Allow-Headers": "Content-Type",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate, br",
    "accept-language": "en-GB,en;q=0.9,en-US;q=0.8",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4649.116 Safari/537.36"
}
url = (
    "https://www.tripadvisor.pt/Attractions-g189102-Activities-Beja_Beja_District_Algentejo.html"
)
req = requests.get(url, headers=headers, timeout=5, verify=False)
req.status_code
bsobj = soup(req.content, "lxml")
```

### Scraping

Now we start scraping. First we start getting attraction names.

```
In [ ]: place = []
for name in bsobj.findAll("span", {"name": "title"}):
    place.append(re.sub(r"\b\d+\b", "", name.text.strip())[2:])
print(place)
```

Some of these will be empty, since the price is gotten via phone call.

Now, we get to the weird part: Reviews. These reviews are handled by the attraction page in weird ways:

- only ten shown per subpage
- each subpage is counted via a multiple of ten
- any multiple of five non-existent will not give 404, but redirect to the first subpage
- language selection via scraping is non-existent, no query parameters, only radio buttons with random labels, defaults chosen via domain/locale (.com .pt)

So the strategy found is:

- create a monstrous amount of subpage links (about 400)

- scrape all reviews, even repeated via the redirect to the first subpage
- later use sets, or dictionaries to remove duplicates

That was done with this awful looking but functional code.

```
In [ ]:
links = []
for review in bsoobj.findAll("a", {"href": re.compile(r'#REVIEWS')}):
    try:
        a = review["href"]
        a = "https://www.tripadvisor.pt" + a
        c = a[: (a.find("Reviews") + 7)] + "" + a[(a.find("Reviews") + 7):]
        links.append(c)
        for i in range(10, 4000, 10):
            b = (
                a[: (a.find("Reviews") + 7)]
                + "-or"
                + str(i)
                + a[(a.find("Reviews") + 7):])
            )
        links.append(b)
    except:
        pass
# print(links)
```

And then create the ID table of the attractions with the most basic information in a pandas DataFrame, and export that one to a .csv file that we can use in Excel, PowerBI, ML libraries like Keras, Tensorflow, SciKitLearn can use.

```
In [ ]:
length = len(place)
d1 = {
    "Attraction": place[:length]
}
df = pd.DataFrame.from_dict(d1)
print(df)
df.to_csv("listtable.csv")
```

Now the most terrible of codes presents you with the creations of various, separated .csv files with the scraped reviews, that we can use.

It iterates all the links and since there's 400 links per attraction, every 400 we use some list comprehension magic with sets to remove duplicates and export the DataFrame to a useful .csv file.

```
In [ ]:
count = 0
count2 = 0
allreviews = []
for link in links:
    try:
        html2 = requests.get(link, headers=headers)
        bsoobj2 = soup(html2.content, "lxml")
        for r in bsoobj2.findAll("span", {"class": "NejBf"}): # as of 7Dez, because in 6Dez it was "class": "cSoN"
            for rev in r:
                try:
                    rv = rev.text
                    if "desde" not in rv and "€" not in rv:
                        allreviews.append(rv + "\n")
                except:
                    pass
    except:
        pass
    count += 1
    if count == 400:
        seen = set()
        allreviews = [
            item
            for item in allreviews
            if not (tuple(item) in seen or seen.add(tuple(item)))]
    dfr = pd.DataFrame.from_dict({"Avaliações": allreviews})
    print(dfr)
    dfr.to_csv("place" + str(count2) + ".csv")
    allreviews = []
    count = 0
    count2 += 1
```

# TripAdvisor

## Webscraping

### Imports

First we start with the imports. We need essentially three (or four) main libraries to work this out; these are:

- requests (to fetch the website)
- lxml (a faster html parser to speed bs4)
- bs4 (a.k.a beautiful soup, a web scraping library)
- pandas (a maths oriented data(set) manipulation library)

Since requests uses urllib3 as a dependency, we can import it first to configure it to suppress the annoying warning about the "insecure" connection (lack of SSL).

```
In [ ]: import urllib3
import requests
import re
from bs4 import BeautifulSoup as soup
import pandas as pd
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

### Request configuration

We need to configure our request, specially in this case, since TripAdvisor won't send us a webpage if we at least not try to emulate a real browser. First we configure our headers, ripping the main headers from our browser, as seen in the Developer tools (F12) in Chromium (we used the new Microsoft Edge).

Then we request the webpage (Restaurants in Beja, Portugal) with our headers attached, a timeout to stop if it takes too long (something is wrong), and verification is disabled (SSL). If the status code is OK (200), doesn't print.

After that we create a BeautifulSoup4 scrapable object with the html content of the page, using lxml.

```
In [ ]: headers = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET",
    "Access-Control-Allow-Headers": "Content-Type",
    "accept": "*/*",
    "accept-encoding": "gzip, deflate, br",
    "accept-language": "en-GB,en;q=0.9,en-US;q=0.8",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.122 Safari/537.36"
}
url = "https://www.tripadvisor.pt/Restaurants-g189102-Beja_Beja_District_Algentejo.html"
req = requests.get(url, headers=headers, timeout=5, verify=False)
req.status_code
bsobj = soup(req.content, "lxml")
```

### Scraping

Now we start scraping. First we start getting restaurant names.

```
In [ ]: place = []
prelinks = []
for name in bsobj.findAll("div", {"class": "OhCyu"}):
    place.append(re.sub(r"\b\d+\b", "", name.span.text.strip())[2:])
    prelinks.append(name.span.a["href"])
```

Some of these will be empty, since the price is gotten via phone call.

Now, we get to the weird part: Reviews. These reviews are handled by the restaurant page in weird ways:

- only ten shown per subpage
- each subpage is counted via a multiple of ten
- any multiple of five non-existent will not give 404, but redirect to the first subpage
- language selection via scraping is non-existent, no query parameters, only radio buttons with random labels, defaults chosen via domain/locale (.com .pt)

So the strategy found is:

- create a monstrous amount of subpage links (about 400)
- scrape all reviews, even repeated via the redirect to the first subpage

- later use sets, or dictionaries to remove duplicates

That was done with this awful looking but functional code.

```
In [ ]:
links = []
for pre in prelinks:
    try:
        a = "https://www.tripadvisor.pt"
        c = a + "" + pre
        d = c[: (c.find("-Reviews-") + len("-Reviews-") - 1)]
        e = c[(c.find("-Reviews-") + len("-Reviews-") - 1) :]
        links.append(c)
        for i in range(10, 4000, 10):
            b = d + "-or" + str(i) + e
            links.append(b)
    except:
        pass
```

And then create the ID table of the attractions with the most basic information in a pandas DataFrame, and export that one to a .csv file that we can use in Excel, PowerBI, ML libraries like Keras, Tensorflow, SciKitLearn can use.

```
In [ ]:
length = len(place)

d1 = {"Restaurant": place[:length]}
df = pd.DataFrame.from_dict(d1)
print(df)
df.to_csv("listtable.csv")
```

Now the most terrible of codes presents you with the creations of various, separated .csv files with the scraped reviews, that we can use.

It iterates all the links and since there's 400 links per restaurant, every 400 we use some list comprehension magic with sets to remove duplicates and export the DataFrame to a useful .csv file.

```
In [ ]:
count = 0
count2 = 0
allreviews = []
for link in links:
    try:
        html2 = requests.get(link, headers=headers)
        bsobj2 = soup(html2.content, "lxml")
        for r in bsobj2.findAll("p", {"class": "partial_entry"}):
            for rev in r:
                try:
                    rv = rev.text.strip()
                    allreviews.append(rv + "\n")
                except:
                    pass
    except:
        pass
    count += 1
    if count == 400:
        seen = set()
        allreviews = [
            item
            for item in allreviews
            if not (tuple(item) in seen or seen.add(tuple(item)))]
    dfr = pd.DataFrame.from_dict({"Avaliações": allreviews})
    dfr.to_csv("restaurant" + str(count2) + ".csv")
    print(dfr)
    allreviews = []
    count = 0
    count2 += 1
```

## Apêndice VII

### Código do *script* de *Python* que fez o *trimming* dos ficheiros *.csv*

Aqui abaixo está representado o código de *Python* que fez o que fez o *trimming* dos ficheiros *.csv* *webscraped* para uso posterior sem potenciais erros.

```
1 import os
2
3
4 def main():
5     current_dir = os.getcwd()
6     open_dir(current_dir + "/scrapes/booking/hotels/")
7     open_dir(current_dir + "/scrapes/zomato/restaurantes/")
8     open_dir(current_dir + "/scrapes/tripadvisor/hotels/")
9     open_dir(current_dir + "/scrapes/tripadvisor/restaurants/")
10    open_dir(current_dir + "/scrapes/tripadvisor/activities/")
11
12
13 def open_dir(directory):
14     for file in os.listdir(directory):
15         if file.endswith(".csv") and not file.startswith("list"):
16             open_file(directory + file)
17
18
19 def open_file(file_name):
20     lines = []
21     with open(file_name, "r", encoding="utf-8", errors="ignore") as f:
22         lines = f.readlines()
23     with open(file_name, "w", encoding="utf-8", errors="ignore") as f:
```

## VII. CÓDIGO DO *script* DE Python QUE FEZ O *trimming* DOS FICHEIROS .csv

---

```
24     for line in lines:
25         f.write(fix_line(line))
26
27
28 def fix_line(line):
29     line = trim_line(line)
30     if "Avaliacoes" in line:
31         line = line.replace('""', "").strip() + str("\n")
32
33     return line
34
35
36 def trim_line(line):
37     line = line.replace('""', "").strip()
38     line = line + str(''''')
39     line = line[:-1]
40     line = line + str(''''')
41     line = line[:-1]
42     line = line + str("\n")
43
44     return line
45
46
47 if __name__ == "__main__":
48     main()
```

## Apêndice VIII

### Código do *script* de *Python* que fez a normalização NFKD dos ficheiros *.csv*

Aqui abaixo está representado o código de *Python* que fez a normalização NFKD dos ficheiros *.csv webscraped* para a sua utilização com os módulos de *keyword extraction* e análise sentimental poderem ser feitos.

```
1 import os
2 import re
3 import unicodedata
4
5
6 def main():
7     current_dir = os.getcwd()
8     normalize_files(current_dir + "/scrapes/booking/hotels/")
9     normalize_files(current_dir + "/scrapes/zomato/restaurantes/")
10    normalize_files(current_dir + "/scrapes/tripadvisor/hotels/")
11    normalize_files(current_dir + "/scrapes/tripadvisor/restaurants/")
12    normalize_files(current_dir + "/scrapes/tripadvisor/activities/")
13
14
15 def normalize_files(directory):
16     for file in os.listdir(directory):
17         if file.endswith(".csv"):
18             normalize_file(directory + file)
19
20
```

VIII. CÓDIGO DO *script* DE Python QUE FEZ A NORMALIZAÇÃO NFKD DOS FICHEIROS .csv

---

```
21 def normalize_file(file_name):
22     lines = []
23     with open(file_name, "r", encoding="utf-8", errors="ignore") as f:
24         lines = f.readlines()
25     with open(file_name, "w", encoding="utf-8", errors="ignore") as f:
26         for line in lines:
27             # f.write(normalize(remove_index(line)))
28             f.write(normalize(line))
29
30
31 def normalize(text):
32     text = remove_emoji(text)
33     text = (
34         unicodedata.normalize("NFKD", text)
35         .encode("ascii", "ignore")
36         .decode("utf-8", "ignore")
37     )
38     return text
39
40
41 # def remove_index(string):
42 #     return re.sub(r"^\[,\]*", "", string)
43 def remove_emoji(string):
44     emoji_pattern = re.compile(
45         "["
46         u"\U0001F600-\U0001F64F" # emoticons
47         u"\U0001F300-\U0001F5FF" # symbols & pictographs
48         u"\U0001F680-\U0001F6FF" # transport & map symbols
49         u"\U0001F1E0-\U0001F1FF" # flags (iOS)
50         u"\U00002500-\U00002BEF" # chinese char
51         u"\U00002702-\U000027B0"
52         u"\U00002702-\U000027B0"
53         u"\U000024C2-\U0001F251"
54         u"\U0001f926-\U0001f937"
55         u"\U00010000-\U0010ffff"
56         u"\u2640-\u2642"
57         u"\u2600-\u2B55"
58         u"\u200d"
59         u"\u23cf"
```

---

```
60         u"\u23e9"
61         u"\u231a"
62         u"\ufe0f" # dingbats
63         u"\u3030"
64         "]+",  
65         flags=re.UNICODE,  
66     )  
67     return emoji_pattern.sub(r"", string)  
68  
69  
70 if __name__ == "__main__":  
71     main()
```



## Apêndice IX

# Código do *script* de *Python* que reorganizou os dados numa base de dados relacional

Aqui abaixo está representado o código de *Python* que reorganizou os dados numa base de dados relacional, assim podendo exporta noutras *.csv* de forma mais organizada e/ou relacionar os dados garantindo uma maior e melhor integridade e coerência.

```
1 import os
2 import yake
3 import pandas
4 import sqlite3
5 import warnings
6
7 warnings.filterwarnings("ignore")
8
9
10 def create_db(db_name):
11     conn = sqlite3.connect(db_name)
12     return conn
13
14
15 def create_table(conn, table_name, cols):
16     c = conn.cursor()
17     c.execute("CREATE TABLE IF NOT EXISTS {} {}".format(table_name, cols))
18     conn.commit()
19
20
```

## IX. CÓDIGO DO *script* DE Python QUE REORGANIZOU OS DADOS NUMA BASE DE DADOS RELACIONAL

---

```
21 def insert_data(conn, table_name, data):
22     c = conn.cursor()
23     c.execute("INSERT INTO {} VALUES {}".format(table_name, data))
24     conn.commit()
25
26
27 def export_data(conn, table_name, file_name):
28     c = conn.cursor()
29     c.execute("SELECT * FROM {}".format(table_name))
30     data = c.fetchall()
31     df = pandas.DataFrame(data)
32     df.to_csv(file_name, index=False)
33
34
35 def keywords_from_review_by_month_year(conn, table_name):
36     months = [
37         "janeiro",
38         "fevereiro",
39         "marco",
40         "abril",
41         "maio",
42         "junho",
43         "julho",
44         "agosto",
45         "setembro",
46         "outubro",
47         "novembro",
48         "dezembro",
49     ]
50     df = pandas.DataFrame(columns=["business_id", "month", "year",
51                             "keyword", "score"])
52     ind = 0
53     for year in range(2000, 2022):
54         for month in months:
55             c = conn.cursor()
56             c.execute(
57                 "SELECT review_text, business_id FROM {} WHERE month =
58                 '{}' AND year = {}".format(
59                     table_name, month, str(year))
```

---

```
58         )
59     )
60     data = c.fetchall()
61     data = sorted(data, key=lambda x: x[1])
62     lst = []
63     for i in range(len(data)):
64         if i == 0:
65             lst.append(data[i][0])
66         elif data[i][1] == data[i - 1][1]:
67             lst.append(data[i][0])
68         else:
69             keywords =
70                 → yake.KeywordExtractor(lan="pt").extract_keywords(
71                     "".join(str(x) for x in lst)
72                 )
73             for k in keywords:
74                 df.loc[ind] = [
75                     str(data[i - 1][1])
76                     .encode("ascii", "ignore")
77                     .decode("utf-8", "ignore"),
78                     str(month)
79                     .encode("ascii", "ignore")
80                     .decode("utf-8", "ignore"),
81                     str(year)
82                     .encode("ascii", "ignore")
83                     .decode("utf-8", "ignore"),
84                     str(k[0])
85                     .encode("ascii", "ignore")
86                     .decode("utf-8", "ignore"),
87                     str(k[1])
88                     .encode("ascii", "ignore")
89                     .decode("utf-8", "ignore"),
90                 ]
91             ind += 1
92         lst = []
93         lst.append(data[i][0])
94     # df.to_sql("keywords", conn, if_exists="append", index=False)
95     for row in df.itertuples():
96         data = (
```

## IX. CÓDIGO DO *script* DE Python QUE REORGANIZOU OS DADOS NUMA BASE DE DADOS RELACIONAL

---

```
96         "(
97             + str(row.business_id).encode("ascii",
98                 ↳ "ignore").decode("utf-8", "ignore")
99             + ", "
100            + str(row.month).encode("ascii", "ignore").decode("utf-8",
101                 ↳ "ignore")
102            + ", "
103            + str(row.year).encode("ascii", "ignore").decode("utf-8",
104                 ↳ "ignore")
105            + ", "
106            + str(row.keyword).encode("ascii", "ignore").decode("utf-8",
107                 ↳ "ignore")
108            + ", "
109            + str(row.score).encode("ascii", "ignore").decode("utf-8",
110                 ↳ "ignore")
111            + ")"
112
113
114
115 def main():
116     # Create database
117     db_name = "projeto.db"
118     conn = create_db(db_name)
119
120     # Create table
121     table_name = "business_type"
122     cols = "(business_type_id INTEGER PRIMARY KEY, business_type_name
123             ↳ TEXT)"
124     create_table(conn, table_name, cols)
125
126     # Create table
127     table_name = "business"
```

---

```
127     cols = "(business_id INTEGER PRIMARY KEY, business_name TEXT,
128             ↵ business_type_id INTEGER, FOREIGN KEY (business_type_id)
129             ↵ REFERENCES business_type (business_type_id))"
130     create_table(conn, table_name, cols)
131
132     # Create table
133     table_name = "reviews"
134     cols = "(review_id INTEGER PRIMARY KEY, review_text TEXT, business_id
135             ↵ INTEGER, sentiment TEXT, month TEXT, year INTEGER, FOREIGN KEY
136             ↵ (business_id) REFERENCES business (business_id))"
137     create_table(conn, table_name, cols)
138
139     # Create table
140     table_name = "keywords"
141     cols = "(business_id INTEGER, month TEXT, year INTEGER, keyword TEXT,
142             ↵ score FLOATING POINT, FOREIGN KEY (business_id) REFERENCES
143             ↵ business (business_id))"
144     create_table(conn, table_name, cols)
145
146     # Insert data
147     table_name = "business_type"
148     data = "(1, 'Restaurant')"
149     insert_data(conn, table_name, data)
150     data = "(2, 'Attraction')"
151     insert_data(conn, table_name, data)
152     data = "(3, 'Hotel')"
153     insert_data(conn, table_name, data)
154
155     # Insert data
156     table_name = "business"
157     pd = pandas.read_csv(
158         "./data/restaurants/listtable.csv", engine="python",
159             ↵ on_bad_lines="skip"
160     )
161     pd.columns = pd.columns.str.strip()
162     lst = []
163     lst = pd["Restaurante"].tolist()
164     l = 1
165     for i in range(len(lst)):
```

---

IX. CÓDIGO DO *script* DE *Python* QUE REORGANIZOU OS DADOS NUMA BASE DE DADOS RELACIONAL

---

```
159     data = (
160         "("
161         + str(l).encode("ascii", "ignore").decode("utf-8", "ignore")
162         + ", "
163         + str(lst[i]).encode("ascii", "ignore").decode("utf-8",
164             "ignore")
165             + "' , 1)"
166     )
167     insert_data(conn, table_name, data)
168     l += 1
169     pd = pandas.read_csv(
170         "./data/activities/listtable.csv", engine="python",
171         on_bad_lines="skip"
172     )
173     pd.columns = pd.columns.str.strip()
174     lst = pd["Attraction"].tolist()
175     for i in range(len(lst)):
176         data = (
177             "("
178             + str(l).encode("ascii", "ignore").decode("utf-8", "ignore")
179             + ", "
180             + str(lst[i]).encode("ascii", "ignore").decode("utf-8",
181                 "ignore")
182                 + "' , 2)"
183     )
184     insert_data(conn, table_name, data)
185     l += 1
186     pd = pandas.read_csv(
187         "./data/hotels/listtable.csv", engine="python", on_bad_lines="skip"
188     )
189     pd.columns = pd.columns.str.strip()
190     lst = pd["Hotel"].tolist()
191     for i in range(len(lst)):
192         data = (
193             "("
194             + str(l).encode("ascii", "ignore").decode("utf-8", "ignore")
195             + ", "
196             + str(lst[i]).encode("ascii", "ignore").decode("utf-8",
197                 "ignore")
```

---

```
194             + " ", 3)"
195         )
196         insert_data(conn, table_name, data)
197         l += 1
198
199     # Insert data
200     table_name = "reviews"
201     all = 0
202     for file in os.listdir("./data/activities"):
203         all += 1
204         if file.startswith("place") and not file.endswith(".zip"):
205             print(file)
206             pd = pandas.read_csv(
207                 "./data/activities/" + file, engine="python",
208                 → on_bad_lines="skip"
209             )
210             pd.columns = pd.columns.str.strip()
211             lst = pd["review"].tolist()
212             lsts = pd["sentiment"].tolist()
213             pd = pandas.read_csv(
214                 "./dates/activities/" + file, engine="python",
215                 → on_bad_lines="skip"
216             )
217             pd.columns = pd.columns.str.strip()
218             lstm = pd[pd.columns[0]].tolist()
219             lsty = pd[pd.columns[1]].tolist()
220             # pd = pandas.read_csv(
221             #     "./keywords/activities/" + file, engine="python",
222             #     → on_bad_lines="skip"
223             # )
224             # pd.columns = pd.columns.str.strip()
225             # lstk = pd["Expressao"].tolist()
226             for i in range(len(lst)):
227                 try:
228                     insert_data(
229                         conn,
230                         table_name,
231                         "(NULL, "
232                         + str(lst[i]))
```

## IX. CÓDIGO DO *script* DE Python QUE REORGANIZOU OS DADOS NUMA BASE DE DADOS RELACIONAL

---

```
230         .encode("ascii", "ignore")
231         .decode("utf-8", "ignore")
232         + "' , "
233         + str(all).encode("ascii",
234             ← "ignore").decode("utf-8", "ignore")
235         + ", ''"
236         + str(lsts[i])
237         .encode("ascii", "ignore")
238         .decode("utf-8", "ignore")
239         + "' , ''"
240         # + str(lstk[i])
241         # .encode("ascii", "ignore")
242         # .decode("utf-8", "ignore")
243         # + "' , ''"
244         + str(lstm[i])
245         .encode("ascii", "ignore")
246         .decode("utf-8", "ignore")
247         + "' , "
248         + str(lsty[i])
249         .encode("ascii", "ignore")
250         .decode("utf-8", "ignore")
251         + ")",
252     )
253     except:
254         pass
255     for file in os.listdir("./data/hotels"):
256         all += 1
257         if file.startswith("hotel") and not file.endswith(".zip"):
258             print(file)
259             pd = pandas.read_csv(
260                 "./data/hotels/" + file, engine="python",
261                 ← on_bad_lines="skip"
262             )
263             pd.columns = pd.columns.str.strip()
264             lst = pd["review"].tolist()
265             lsts = pd["sentiment"].tolist()
266             pd = pandas.read_csv(
267                 "./dates/hotels/" + file, engine="python",
268                 ← on_bad_lines="skip"
```

---

```
266     )
267     pd.columns = pd.columns.str.strip()
268     lstm = pd[pd.columns[0]].tolist()
269     lsty = pd[pd.columns[1]].tolist()
270     # pd = pandas.read_csv(
271     #     "./keywords/hotels/" + file, engine="python",
272     #     ↳ on_bad_lines="skip"
273     # )
274     # pd.columns = pd.columns.str.strip()
275     # lstk = pd["Expressao"].tolist()
276     for i in range(len(lst)):
277         try:
278             insert_data(
279                 conn,
280                 table_name,
281                 "(NULL, ''"
282                 + str(lst[i])
283                 .encode("ascii", "ignore")
284                 .decode("utf-8", "ignore")
285                 + "'", ""
286                 + str(all).encode("ascii",
287                 ↳ "ignore").decode("utf-8", "ignore")
288                 + ", ''"
289                 + str(lstts[i])
290                 .encode("ascii", "ignore")
291                 .decode("utf-8", "ignore")
292                 + "'", ""
293                 # + str(lstk[i])
294                 # .encode("ascii", "ignore")
295                 # .decode("utf-8", "ignore")
296                 # + "'", ""
297                 + str(lstm[i])
298                 .encode("ascii", "ignore")
299                 .decode("utf-8", "ignore")
300                 + "'", ""
301                 + str(lsty[i])
302                 .encode("ascii", "ignore")
303                 .decode("utf-8", "ignore")
304                 + ")",
305
```

IX. CÓDIGO DO *script* DE *Python* QUE REORGANIZOU OS DADOS NUMA BASE DE DADOS RELACIONAL

---

```
303             )
304         except:
305             pass
306     for file in os.listdir("./data/restaurants"):
307         all += 1
308         if file.startswith("restaurant") and not file.endswith(".zip"):
309             print(file)
310             pd = pandas.read_csv(
311                 "./data/restaurants/" + file, engine="python",
312                 ↳ on_bad_lines="skip"
313             )
314             pd.columns = pd.columns.str.strip()
315             lst = pd["review"].tolist()
316             lsts = pd["sentiment"].tolist()
317             pd = pandas.read_csv(
318                 "./data/restaurants/" + file, engine="python",
319                 ↳ on_bad_lines="skip"
320             )
321             pd.columns = pd.columns.str.strip()
322             lstm = pd[pd.columns[0]].tolist()
323             lsty = pd[pd.columns[1]].tolist()
324             # pd = pandas.read_csv(
325             #     "./keywords/restaurants/" + file, engine="python",
326             #     ↳ on_bad_lines="skip"
327             # )
328             # pd.columns = pd.columns.str.strip()
329             # lstk = pd["Expressao"].tolist()
330             for i in range(len(lst)):
331                 try:
332                     insert_data(
333                         conn,
334                         table_name,
335                         "(NULL, '"
336                         + str(lst[i])
337                         .encode("ascii", "ignore")
338                         .decode("utf-8", "ignore")
339                         + "'", "
340                         + str(all).encode("ascii",
341                             ↳ "ignore").decode("utf-8", "ignore")
```

---

```
338                     + ",  "
339                     + str(lst[i])
340                     .encode("ascii", "ignore")
341                     .decode("utf-8", "ignore")
342                     + "'", "'"
343                     # + str(lstk[i])
344                     # .encode("ascii", "ignore")
345                     # .decode("utf-8", "ignore")
346                     # + "'", "'"
347                     + str(lstm[i])
348                     .encode("ascii", "ignore")
349                     .decode("utf-8", "ignore")
350                     + "'", ""
351                     + str(lsty[i])
352                     .encode("ascii", "ignore")
353                     .decode("utf-8", "ignore")
354                     + ")",
355
356             except:
357                 pass
358
359         conn.commit()
360
361     keywords_from_review_by_month_year(conn, table_name)
362     conn.commit()
363
364     # export sqlite database to csv
365     export_data(conn, "reviews", "./reviews.csv")
366     export_data(conn, "business", "./business.csv")
367     export_data(conn, "business_type", "./business_type.csv")
368     export_data(conn, "keywords", "./keywords.csv")
369
370     conn.close()
371
372
373 if __name__ == "__main__":
374     main()
375     print("Done")
```



## Apêndice X

# Código do *script* de *Python* que gerou os gráficos não temporais dos hotéis, atracções e restaurantes do *TripAdvisor*, *Booking* e *Zomato*

Aqui abaixo está representado o código de *Python* que gerou os gráficos circulares e as nuvens de palavras com de análise não temporal de todos os estabelecimentos de todos os tipos de todas as plataformas. O qual usa essencialmente “matplotlib” e “wordcloud” para a geração dos gráficos (mais uma máscara com o formato da região de Beja).

```
1 import os
2 import pandas
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from wordcloud import WordCloud
6 from PIL import Image
7
8
9 def get_keywords(path):
10     df = pandas.read_csv(path, sep=r",\s")
11     keywords = df["Expressao"].values.tolist()
12     frequency = df["Frequencia"].values.tolist()
13
14     keywords = [str(x) for x in keywords]
15     frequency = [float(x) for x in frequency]
16     frequency = [round(x * 100) for x in frequency]
17     keywords, frequency = [
```

X. CÓDIGO DO *script* DE Python QUE GEROU OS GRÁFICOS NÃO TEMPORAIS DOS HOTÉIS, ATRACÇÕES E RESTAURANTES DO *TripAdvisor*, *Booking* e *Zomato*

---

```
18     list(x)
19     for x in zip(
20         *sorted(zip(keywords, frequency), key=lambda pair: pair[1],
21             → reverse=True)
22     )
23     return keywords, frequency
24
25
26 def get_sentiments(path):
27     positive = 0
28     negative = 0
29     df = pandas.read_csv(path)
30     for sentiment in df["sentiment"]:
31         if sentiment == "positive":
32             positive += 1
33         elif sentiment == "negative":
34             negative += 1
35     return positive, negative
36
37
38 def get_sentiment_percentage(positive, negative):
39     total = positive + negative
40     positive = round(positive / total * 100)
41     negative = round(negative / total * 100)
42     return positive, negative
43
44
45 def get_sentiment_graph(positive, negative, path, name):
46     plt.pie([positive, negative], labels=["positive", "negative"],
47             → autopct=".0f%%")
48     plt.savefig(path + name.replace(".csv", "") + "_sentiments.jpeg")
49     plt.close()
50
51 # create pie chart
52 def get_keywords_graph(keywords, frequency, path, name):
53     keywords_dict = dict(zip(keywords, frequency))
```

---

```
54     keywords_dict = sorted(keywords_dict.items(), key=lambda kv: kv[1],
55                             reverse=True)
56
56     keywords = [x[0] for x in keywords_dict[:10]]
57     frequency = [x[1] for x in keywords_dict[:10]]
58
59     plt.pie(frequency, labels=keywords, autopct=".0f%%")
60     plt.savefig(path + name.replace(".csv", "") + "_keywords.jpeg")
61     plt.close()
62
63
64 def get_keywords_wordcloud(keywords, frequency, path, name):
65
66     keywords = keywords[:100]
67     frequency = frequency[:100]
68
69     mask = np.array(Image.open(path + "mask.jpeg"))
70     wc = WordCloud(
71         background_color="white",
72         mask=mask,
73         max_words=100,
74         contour_width=3,
75         contour_color="white",
76     )
77     wc.generate_from_frequencies(dict(zip(keywords, frequency)))
78     wc.to_file(path + name.replace(".csv", "") + "_keywordcloud.jpeg")
79
80
81 def get_graphs_from_directory(path):
82     current_dir = os.getcwd()
83     export_path = current_dir + "/graphs" + path
84     keyword_path = current_dir + "/keywords" + path
85     sentiment_path = current_dir + "/sentiments" + path
86
87     total_keywords, total_frequency = [], []
88     total_sentiment_positive, total_sentiment_negative = 0, 0
89
90     for file in os.listdir(keyword_path):
91         if file.endswith(".csv"):
```

X. CÓDIGO DO *script* DE *Python* QUE GEROU OS GRÁFICOS NÃO TEMPORAIS DOS HOTÉIS, ATRACÇÕES E RESTAURANTES DO *TripAdvisor*, *Booking* e *Zomato*

---

```
92     keywords, frequency = get_keywords(keyword_path + file)
93     total_keywords, total_frequency = (
94         total_keywords + keywords,
95         total_frequency + frequency,
96     )
97     get_keywords_graph(keywords, frequency, export_path, file)
98     get_keywords_wordcloud(keywords, frequency, export_path, file)
99
100    get_keywords_graph(total_keywords, total_frequency, export_path,
101        ↪ "total")
102    get_keywords_wordcloud(total_keywords, total_frequency, export_path,
103        ↪ "total")
104
105    for file in os.listdir(sentiment_path):
106        if file.endswith(".csv"):
107            positive, negative = get_sentiments(sentiment_path + file)
108            total_sentiment_positive, total_sentiment_negative = (
109                total_sentiment_positive + positive,
110                total_sentiment_negative + negative,
111            )
112            positive, negative = get_sentiment_percentage(positive,
113                ↪ negative)
114            get_sentiment_graph(positive, negative, export_path, file)
115
116            total_sentiment_positive, total_sentiment_negative =
117                ↪ get_sentiment_percentage(
118                    total_sentiment_positive, total_sentiment_negative
119                )
120            get_sentiment_graph(
121                total_sentiment_positive, total_sentiment_negative, export_path,
122                ↪ "total"
123            )
124
125    if __name__ == "__main__":
126        get_graphs_from_directory("/booking/hotels/")
127        get_graphs_from_directory("/zomato/restaurantes/")
128        get_graphs_from_directory("/tripadvisor/hotels/")
129        get_graphs_from_directory("/tripadvisor/restaurants/")
```

---

```
126     get_graphs_from_directory("/tripadvisor/activities/")
```



## Apêndice XI

# Código do *script* de *Python* que gerou os gráficos temporais dos hotéis, atracções e restaurantes do *TripAdvisor*

Aqui abaixo está representado o código de *Python* que gerou os gráficos circulares e as nuvens de palavras com de análise temporal de todos os estabelecimentos da plataforma *TripAdvisor*. O qual usa essencialmente “matplotlib” e “wordcloud” para a geração dos gráficos (mais uma máscara com o formato da região de Beja).

```
1 import os
2 import pandas
3 import sqlite3
4 import numpy as np
5 from wordcloud import WordCloud
6 import matplotlib.pyplot as plt
7 from PIL import Image
8
9
10 def create_wordcloud(month, year, df):
11     df = df[df["month"] == month]
12     df = df[df["year"] == year]
13     df = df.sort_values(by="score",
14                         ascending=False).groupby("business_id").head(10)
15     for business_id in df["business_id"].unique():
16         try:
17             words, freq = (
```

XI. CÓDIGO DO *script* DE *Python* QUE GEROU OS GRÁFICOS TEMPORAIS DOS HOTEÍS, ATRACÇÕES E RESTAURANTES DO *TripAdvisor*

---

```
17         df[df["business_id"] ==  
18             ↵ business_id]["keyword"].values.tolist(),  
19         df[df["business_id"] ==  
20             ↵ business_id]["score"].values.tolist(),  
21     )  
22     words, freq = [str(x) for x in words], [float(x) for x in freq]  
23     words, freq = words[:100], freq[:100]  
24     dict_words = dict(zip(words, freq))  
25     mask = np.array(Image.open("./mask.png"))  
26     wc = WordCloud(  
27         background_color="white",  
28         mask=mask,  
29         max_words=100,  
30         contour_width=3,  
31         contour_color="white",  
32     ).generate_from_frequencies(dict_words)  
33     wc.to_file(  
34         "./wordclouds/wordcloud_{0}_of_{1}_at_business{2}.png".format(  
35             month, year, business_id  
36         )  
37     )  
38     )  
39  
40 def circular_graph_keywords(month, year, df):  
41     df = df[df["month"] == month]  
42     df = df[df["year"] == year]  
43     df = df.sort_values(by="score",  
44         ascending=False).groupby("business_id").head(10)  
45     for business_id in df["business_id"].unique():  
46         try:  
47             words, freq = (  
48                 df[df["business_id"] ==  
49                     ↵ business_id]["keyword"].values.tolist(),  
50                     df[df["business_id"] ==  
51                         ↵ business_id]["score"].values.tolist(),  
52                 )  
53             # create a circular graph
```

---

```

51     words, freq = [str(x) for x in words], [float(x) for x in freq]
52     words, freq = words[:10], freq[:10]
53     plt.pie(freq, labels=words, autopct="%1.1f%%", startangle=90)
54     plt.savefig(
55         "./graphs/keywords/circular_keywords_{}_{}_at_business{}_"
56         "month, year, business_id"
57     )
58     plt.close()
59 except:
60     pass
62
63
64 def circular_graph_sentiment(month, year, df):
65     df = df[df["month"] == month]
66     df = df[df["year"] == year]
67     for business_id in df["business_id"].unique():
68         try:
69             sentiments = df[df["business_id"] == business_id][
70                 "sentiment"
71             ].values.tolist()
72             positive, negative = sentiments.count("positive"),
73             → sentiments.count(
74                 "negative"
75             )
76             total = positive + negative
77             positive = positive / total
78             negative = negative / total
79             plt.pie(
80                 [positive, negative],
81                 labels=["positive", "negative"],
82                 autopct="%1.1f%%",
83                 startangle=90,
84             )
85             plt.savefig(
86                 "./graphs/sentiments/circular_sentiment_{}_{}_at_busine{}_"
87                 "month, year, business_id"

```

XI. CÓDIGO DO *script* DE *Python* QUE GEROU OS GRÁFICOS TEMPORAIS DOS HOTEÍS, ATRACÇÕES E RESTAURANTES DO *TripAdvisor*

---

```
87         )
88     )
89     plt.close()
90 except:
91     pass
92
93
94 def import_data(db_file, table_name):
95     conn = sqlite3.connect(db_file)
96     df = pandas.read_sql_query("SELECT * FROM {}".format(table_name), conn)
97     conn.close()
98     return df
99
100
101 def main():
102
103     years = range(2000, 2022)
104     months = [
105         "janeiro",
106         "fevereiro",
107         "marco",
108         "abril",
109         "maio",
110         "junho",
111         "julho",
112         "agosto",
113         "setembro",
114         "outubro",
115         "novembro",
116         "dezembro",
117     ]
118
119     bd = "./projeto.db"
120     dr = import_data(bd, "reviews")
121     dk = import_data(bd, "keywords")
122     db = import_data(bd, "business")
123     dbt = import_data(bd, "business_type")
124
125     for year in years:
```

---

```
126     for month in months:
127         create_wordcloud(month, year, dk)
128         circular_graph_keywords(month, year, dk)
129         circular_graph_sentiment(month, year, dr)
130
131
132 if __name__ == "__main__":
133     main()
```



## **Anexos**

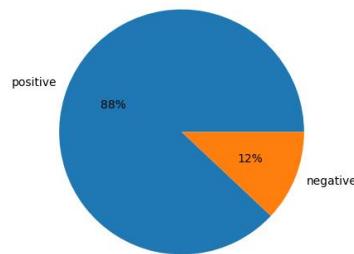


## Anexo I

### Anexo referente às imagens dos gráficos utilizados no relatório (TripAdvisor-Hóteis)

Uma vez que as imagens que retratam os gráficos elaborados são muito extensas, serão apenas mostradas algumas delas e em cada secção apontado o link que redirecciona para o GitHub do projecto onde será possível aceder a cada imagem respectivamente assim como ao ficheiro *.pbix* que contém os gráficos realizados no *PowerBI*. Inicialmente serão expostos os gráficos totais, anteriormente mostrados no capítulo 7 (Geração de gráficos) exclusivamente para o website *TripAdvisor* e acerca dos hotéis.

Acesso a todos os gráficos originados: GitHub.

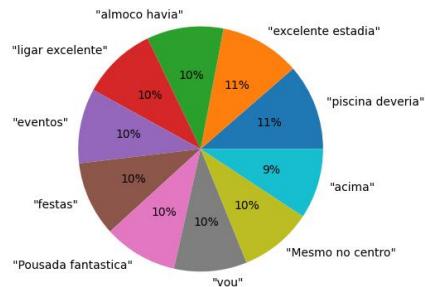


**Figura I.1:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente à Pousada Convento Beja

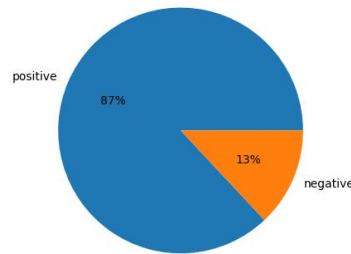
## I. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR-HÓTEIS)



**Figura I.2:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Pousada Convento Beja



**Figura I.3:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Pousada Convento Beja



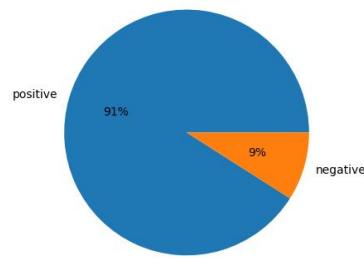
**Figura I.4:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente ao Hotel São Domingos



**Figura I.5:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente ao Hotel São Domingos



**Figura I.6:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente ao Hotel São Domingos



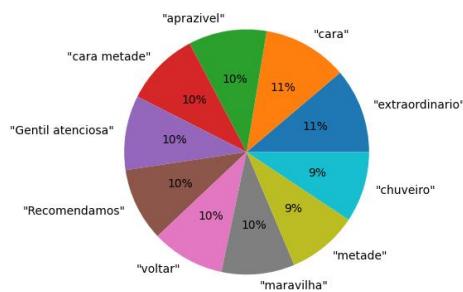
**Figura I.7:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente à Herdade das Barradas da Serra

I. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO  
(TRIPADVISOR-HÓTEIS)

---



**Figura I.8:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Herdade das Barradas da Serra

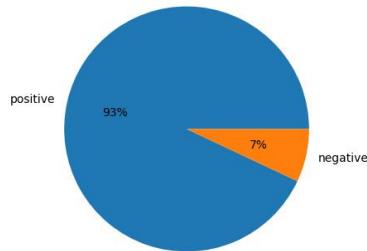


**Figura I.9:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Herdade das Barradas da Serra

## Anexo II

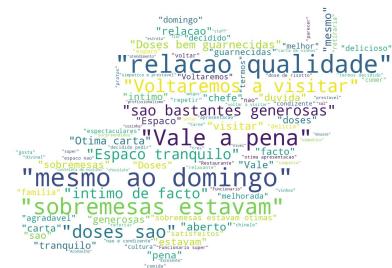
### Anexo referente às imagens dos gráficos utilizados no relatório (TripAdvisor-Restaurantes)

Acesso a todos os gráficos originados: GitHub.



**Figura II.1:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente ao restaurante Íntimo

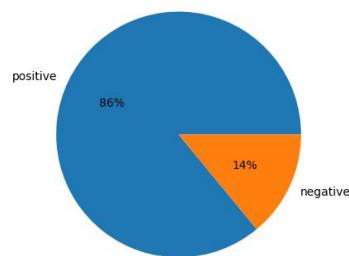
## II. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR-RESTAURANTES)



**Figura II.2:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente ao restaurante Íntimo



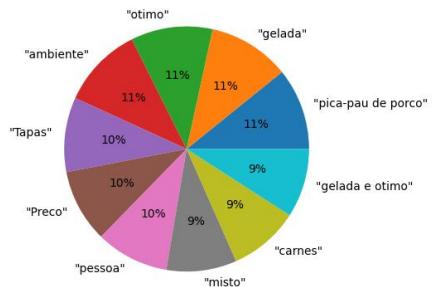
**Figura II.3:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente ao restaurante Íntimo



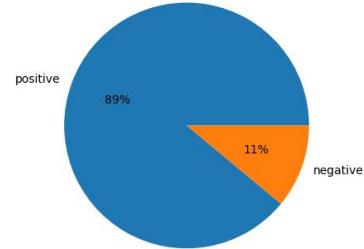
**Figura II.4:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente à Pizzaria Milano



**Figura II.5:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Pizzaria Milano



**Figura II.6:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Pizzaria Milano

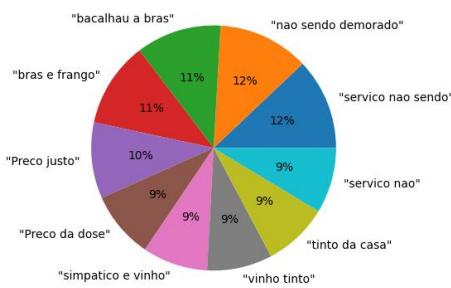


**Figura II.7:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente ao restaurante Ilha do peixe

## II. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR-RESTAURANTES)



**Figura II.8:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente ao restaurante Ilha do peixe

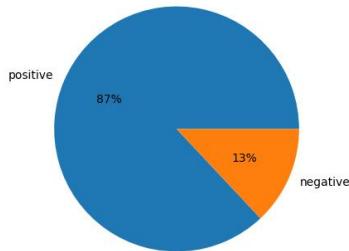


**Figura II.9:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente ao restaurante Ilha do peixe

## Anexo III

### Anexo referente às imagens dos gráficos utilizados no relatório (TripAdvisor-Actividades)

Acesso a todos os gráficos originados: GitHub.

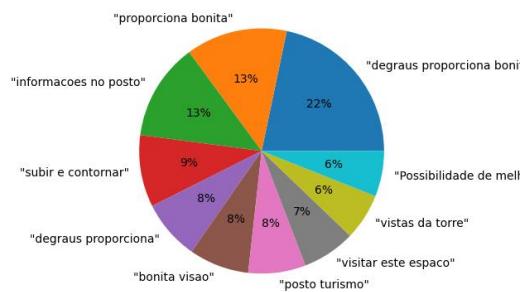


**Figura III.1:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente ao Castelo de Beja

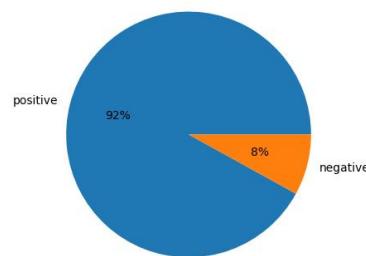
### III. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR-ACTIVIDADES)



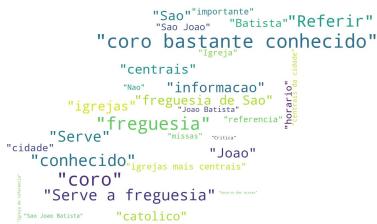
**Figura III.2:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente ao Castelo de Beja



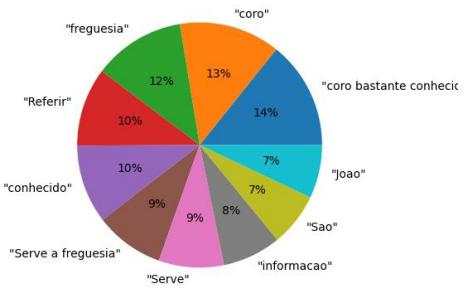
**Figura III.3:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente ao Castelo de Beja



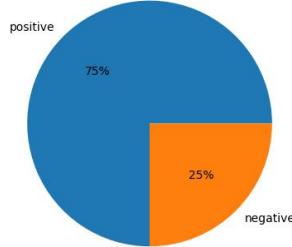
**Figura III.4:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente à Catedral de Beja



**Figura III.5:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Catedral de Beja

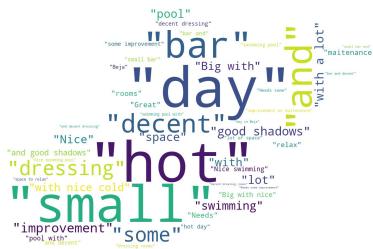


**Figura III.6:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Catedral de Beja

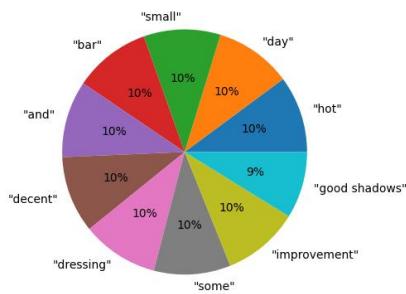


**Figura III.7:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* referente à Janela Manuelina

### III. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR-ACTIVIDADES)



**Figura III.8:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *TripAdvisor* referente à Janela Manuelina

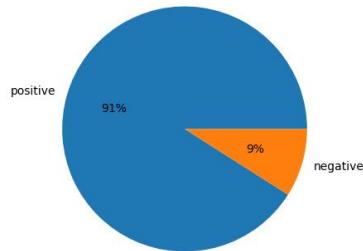


**Figura III.9:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *TripAdvisor* referente à Janela Manuelina

## Anexo IV

### Anexo referente às imagens dos gráficos utilizados no relatório (Booking)

Acesso a todos os gráficos originados: GitHub.



**Figura IV.1:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *Booking* referente ao hotel Aljana Guest House Beja

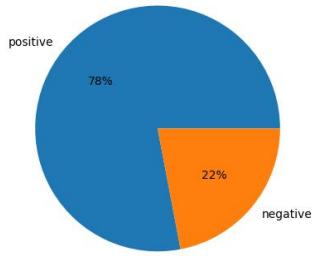
#### IV. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (BOOKING)



**Figura IV.2:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Booking* referente ao hotel Aljana Guest House Beja



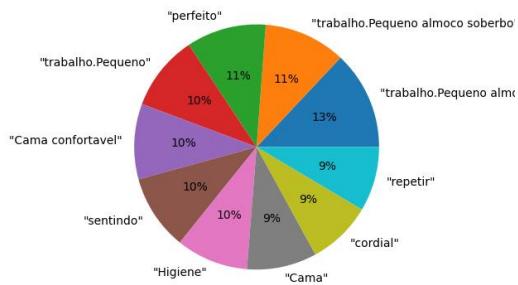
**Figura IV.3:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma Booking referente ao hotel Aljana Guest House Beja



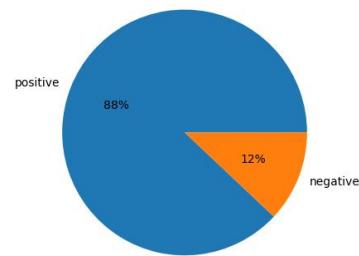
**Figura IV.4:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *Booking* referente ao Hotel Melius



**Figura IV.5:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Booking* referente ao Hotel Melius



**Figura IV.6:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *Booking* referente ao Hotel Melius



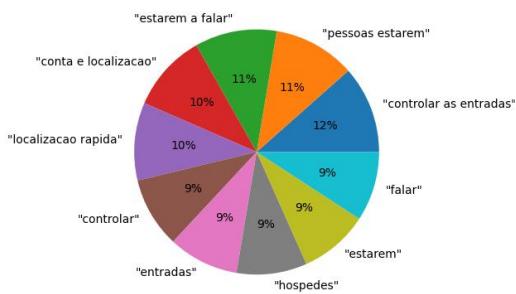
**Figura IV.7:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *Booking* referente a Herdade da Diabrória - Agroturismo

**IV. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO  
(BOOKING)**

---



**Figura IV.8:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Booking* referente a Herdade da Diabroia - Agroturismo

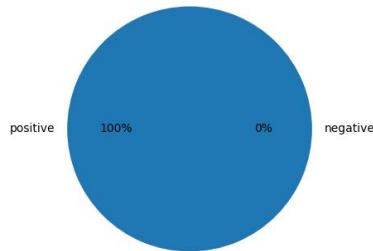


**Figura IV.9:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *Booking* referente a Herdade da Diabroia - Agroturismo

## Anexo V

### Anexo referente às imagens dos gráficos utilizados no relatório (Zomato)

Acesso a todos os gráficos originados: GitHub.



**Figura V.1:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *Zomato* referente ao restaurante Adega Típica 25 de Abril

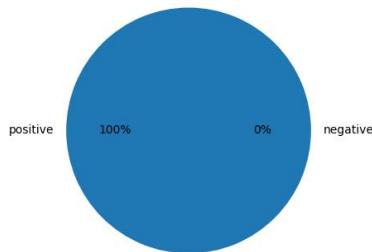
## V. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (ZOMATO)



**Figura V.2:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Zomato* referente ao restaurante Adega Típica 25 de Abril



**Figura V.3:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma Zomato referente ao restaurante Adegas Típica 25 de Abril



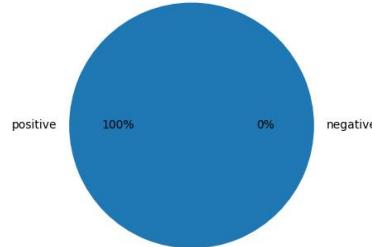
**Figura V.4:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma Zomato referente ao restaurante Sushi Alentejano



**Figura V.5:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Zomato* referente ao restaurante Sushi Alentejano



**Figura V.6:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *Zomato* referente ao restaurante Sushi Alentejano



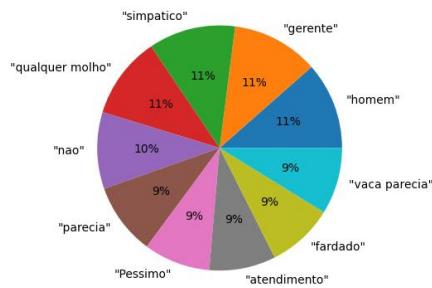
**Figura V.7:** Gráfico circular gerado baseando-se nos *sentiments* mais usados da plataforma *Zomato* referente ao restaurante Gatus Cervejaria Alentejana

V. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO  
(ZOMATO)

---



**Figura V.8:** Gráfico de palavras-chave e nuvens de palavras-chave contendo as *keywords* mais usadas da plataforma *Zomato* referente ao restaurante Gatus Cervejaria Alentejana



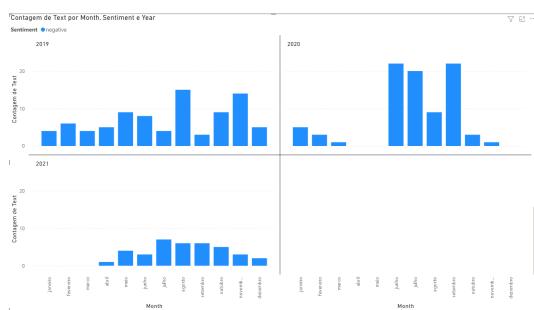
**Figura V.9:** Gráfico circular gerado baseando-se nas *keywords* mais usadas da plataforma *Zomato* referente ao restaurante Gatus Cervejaria Alentejana

## Anexo VI

# Anexo referente às imagens dos gráficos utilizados no relatório (TripAdvisor - gráficos temporais)

Acesso aos ficheiros *PowerBI* criados e as imagens retiradas do mesmo:

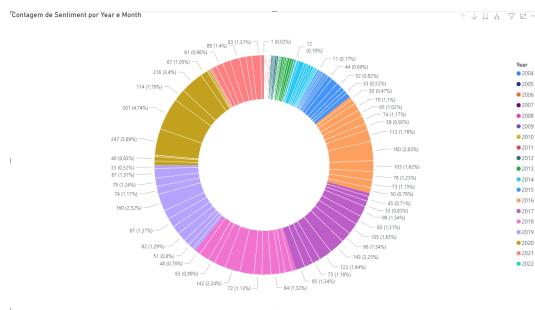
1. GitHub gráficos PowerBI 1.
2. GitHub gráficos PowerBI 2.
3. GitHub PowerBI screenshots.
4. GitHub wordclouds
5. GitHub gráficos Keywords
6. GitHub gráficos Sentimentos



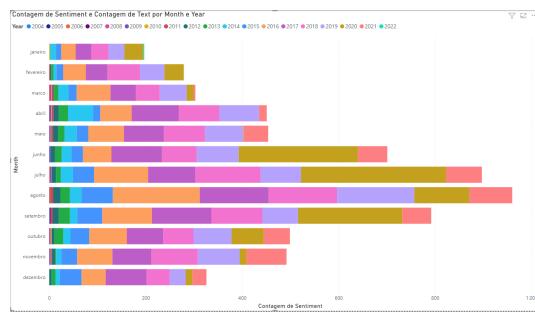
**Figura VI.1:** Gráfico de barras gerado baseando-se nos *sentiments* mais usados da plataforma *TripAdvisor* com níveis temporais sobre os *sentiments* negativos ao longo do tempo

## VI. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO (TRIPADVISOR - GRÁFICOS TEMPORAIS)

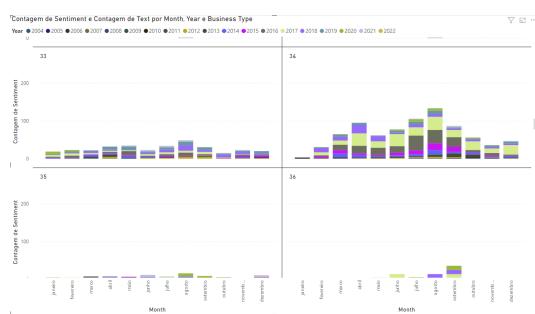
---



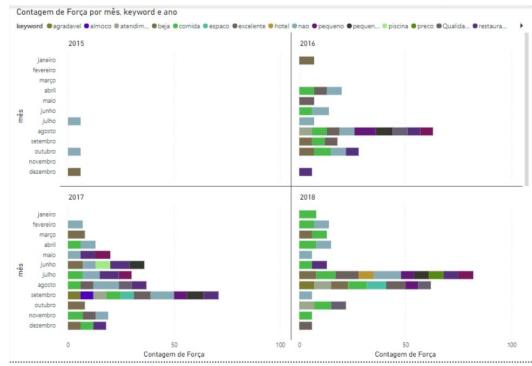
**Figura VI.2:** Gráfico circular gerado baseando-se no número de *sentiments* da plataforma *TripAdvisor* ao longo dos anos



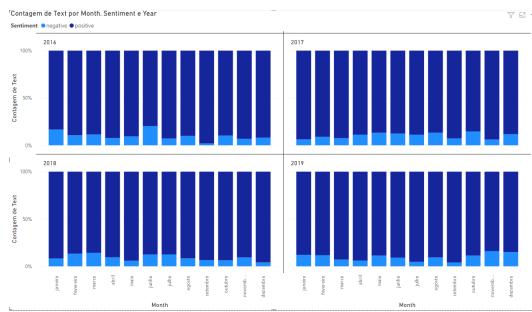
**Figura VI.3:** Gráfico de barras gerado baseando-se no número de *sentiments* da plataforma *TripAdvisor* ao longo dos anos



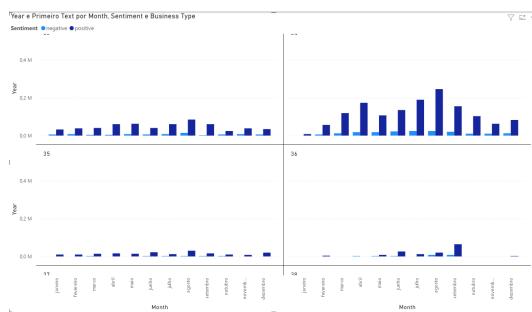
**Figura VI.4:** Gráfico de barras gerado baseando-se nos *sentiments* mais usados e em cada hotel da plataforma *TripAdvisor* ao longo dos anos



**Figura VI.5:** Gráfico de barras gerado baseando-se nas *keywords* mais usadas dos hotéis da plataforma *TripAdvisor* ao longo dos anos



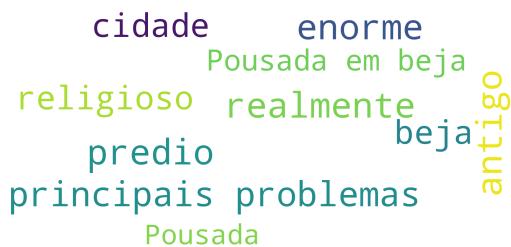
**Figura VI.6:** Gráfico de barras gerado baseando-se nos *sentiments* positivos e negativos dos hotéis da plataforma *TripAdvisor* ao longo dos anos



**Figura VI.7:** Gráfico de barras gerado baseando-se nos *sentiments* positivos e negativos da plataforma *TripAdvisor* ao longo dos anos em cada hotel

VI. ANEXO REFERENTE ÀS IMAGENS DOS GRÁFICOS UTILIZADOS NO RELATÓRIO  
(TRIPADVISOR - GRÁFICOS TEMPORAIS)

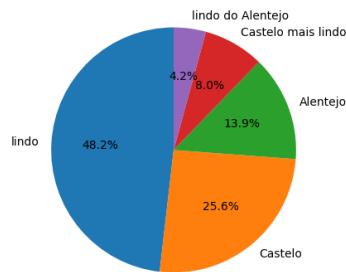
---



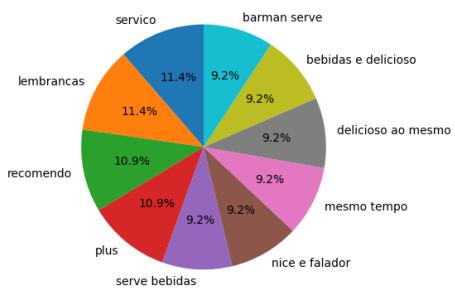
**Figura VI.8:** WordCloud gerado baseando-se nas *keywords* mais usadas no Núcleo Museológico no Mês de Abril em 2011



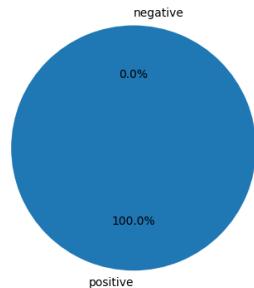
**Figura VI.9:** WordCloud gerado baseando-se nas *keywords* mais usadas no Núcleo Museológico no Mês de Agosto em 2012



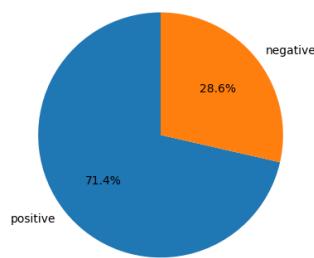
**Figura VI.10:** Gráfico circular com as percentagens das *keywords* mais usadas do Restaurante Dom Dinis no Mês de Agosto em 2012



**Figura VI.11:** Gráfico circular com as percentagens das *keywords* mais usadas na Sé Catedral Beja no Mês de Abril em 2013



**Figura VI.12:** Gráfico circular com a opinião positiva e negativa no Núcleo Museológico no Mês de Abril em 2011



**Figura VI.13:** Gráfico circular com a opinião positiva e negativa na Casa Santa Vitoria no Mês de Abril em 2014