ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

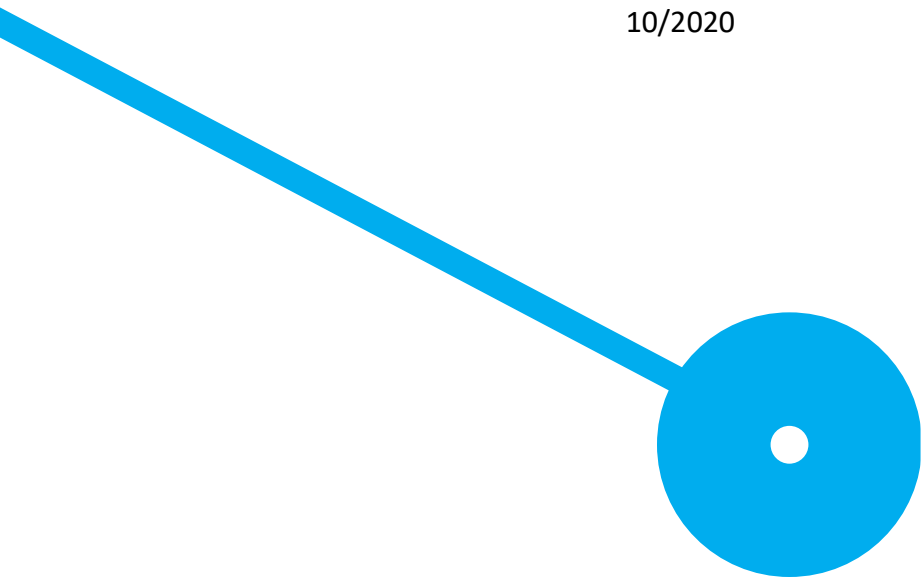
L

LICENCIATURA
ÁREA DE ESPECIALIZAÇÃO

Análise Comparativa de Algoritmos de Aprendizagem com Base em Séries Temporais

Luís Marques

10/2020



Este trabalho não inclui as críticas e sugestões feitas pelo Júri

[Página propositadamente deixada em branco.]

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO
POLITÉCNICO
DO PORTO

L

LICENCIATURA
ÁREA DE ESPECIALIZAÇÃO

Análise Comparativa de Algoritmos de Aprendizagem com Base em Séries Temporais

Luís Marques
Professor Fábio Silva

Este trabalho não inclui as críticas e sugestões feitas pelo Júri

[Página propositadamente deixada em branco.]

Biografia do Autor

Luís Marques é um estudante de Engenharia Informática na Escola Superior de Tecnologia e Gestão do Politécnico do Porto, sendo o ano atual o seu terceiro ano da Licenciatura. Nascido a 8 de junho de 1999 em Vila Nova de Famalicão, o autor nutre bastante interesse em tecnologia e gosta particularmente de desenvolvimento de software backend. Também demonstra bastante interesse na área de Inteligência Artificial e de *Machine Learning*.

Resumo

De modo a realizar uma análise comparativa entre diferentes algoritmos de aprendizagem com base em séries temporais, é necessário abordar estes dois conceitos desagregadamente e estudar cada um deles individualmente.

Este projeto representa um estudo realizado em torno de alguns modelos, entre eles o modelo *ARIMA*, *ARIMAX*, *SARIMA* e *SARIMAX*. A estrutura de cada algoritmo foi desenhada utilizando alguns recursos disponibilizados pelo Professor Fábio Silva e com a ajuda de algumas bibliotecas da linguagem *Python*, sendo as mais evidentes a biblioteca *"pandas"*, *"numpy"*, *"sklearn"*, *"statsmodels"* e *"matplotlib"*.

Depois de desenvolvida uma plataforma de seleção dos melhores modelos para determinados *datasets*, foram realizados dois casos de estudo, selecionados os melhores modelos e uilizados estes modelos para responder a algumas questões sobre os dois casos.

Feito isto, realizou-se uma adaptação do projeto para o ambiente do *Google Colab*.

Palavras Chave: *ARIMA*, Inteligência Artificial, *Machine Learning*, *Python*

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Glossário

Application Programming Interface é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar os seus serviços. x

Autoregressive Integrated Moving Average with Exogenous Variables é um modelo de aprendizagem de média móvel integrado autoregressivo com variáveis exógenas. x, 7, 23

Autoregressive Integrated Moving Average é um modelo de aprendizagem de média móvel integrado autoregressivo. x, 7, 21

classe abstrata não podem ser instanciadas, elas servem apenas para que outras classes a usem como modelo. 16

COVID-19 é uma doença infeciosa causada por um coronavírus descoberto recentemente. 40, 46

dataset (Conjunto de dados) é uma coleção de dados normalmente tabulados. Por cada elemento destacam-se várias características. Cada coluna representa uma variável particular. Cada linha corresponde a um determinado membro do conjunto de dados em questão. Cada valor é conhecido como um dado. ii, v, vi, 1, 2, 3, 5, 12, 15, 16, 20, 29, 32, 35, 36, 37, 39, 40, 41, 42, 44, 46

diagrama de Casos de Uso descreve a funcionalidade proposta para um novo sistema que será projetado, é uma excelente ferramenta para o levantamento dos requisitos funcionais do sistema. v, 11

dióxido de carbono é um composto químico constituído por dois átomos de oxigénio e um átomo de carbono. x

Google é uma empresa multinacional de serviços online e software dos Estados Unidos. viii, 13

Google Cloud é uma suíte de computação em nuvem oferecida pelo Google. viii, 34

Google Colab é um ambiente de desenvolvimento com a linguagem Python que não requer configuração e é executado utilizando a Google Cloud. ii, 1, 2, 13, 15, 34, 46

grid search é o processo de coleção de dados para configurar os parâmetros ideais para um determinado modelo. v, vi, 1, 9, 37, 42

Integrated Development Environment é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. x

Inteligência Artificial é a inteligência similar à humana exibida por mecanismos ou software, para além de também ser um campo de estudo académico. i, ii, ix, x, 34

Jupyter Notebook é uma aplicação web de código aberto que pernute criar e partilhar documentos que contêm código, equações, visualizações e texto narrativo. 34

Machine Learning é um subcampo da Engenharia e da ciência da computação que evoluiu do estudo de reconhecimento de padrões e da teoria da aprendizagem computacional em Inteligência Artificial. i, ii, x, 5, 34

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. ii, viii, 10, 12, 16, 20, 46

roadmap é um recurso visual de alto nível que mapeia a evolução do produto/projeto ao longo do tempo. v, 14

script é um programa de computador, normalmente executado com um interpretador. 1

Seasonal Autoregressive Integrated Moving Average with Exogenous Variables é um modelo de aprendizagem de média móvel integrado autoregressivo sazonal com variáveis exógenas. x, 8, 27

Seasonal Autoregressive Integrated Moving Average é um modelo de aprendizagem de média móvel integrado autoregressivo sazonal. x, 8, 25

string é uma sequência de caracteres, geralmente utilizada para representar palavras, frases ou textos de um programa. 20

validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. 20

Variáveis Exógenas são variáveis cujos valores são determinados fora do modelo e são impostas ao modelo (no caso do ARIMA serão utilizadas para ajudar a fazer as previsões). 7, 8

Abreviaturas

API Application Programming Interface. 46

ARIMA Autoregressive Integrated Moving Average. ii, v, vi, ix, 1, 5, 7, 8, 12, 21, 22, 36, 37, 38, 39, 41, 42, 43, 44, 45

ARIMAX Autoregressive Integrated Moving Average with Exogenous Variables. ii, v, vi, 1, 7, 8, 12, 23, 24, 36, 37, 38, 41, 45

CO2 dióxido de carbono. 5

IA Inteligência Artificial. i, ii, 34

IDE Integrated Development Environment. 10

MAE Erro Médio Absoluto. 13, 15, 31

ML Machine Learning. i, ii, 5, 34

MSE Erro Quadrático Médio. 13, 15, 31

POO Programação Orientada a Objetos. 15, 16

RMSE Raiz Quadrática Média do Erro. 13, 15, 31

SARIMA Seasonal Autoregressive Integrated Moving Average. ii, 1, 8, 12, 25, 26, 29, 36, 37, 41, 45

SARIMAX Seasonal Autoregressive Integrated Moving Average with Exogenous Variables. ii, 1, 8, 12, 27, 29, 36, 37, 41, 45

[Página propositadamente deixada em branco.]

Capítulo 1

Contextualização e Motivação

1.1 Contextualização

O presente documento descreve o trabalho realizado na análise comparativa de algoritmos de aprendizagem com base em séries temporais. O trabalho foi proposto pelo Professor Fábio Silva cuja motivação é desenvolver uma plataforma de seleção dos melhores algoritmos para um determinado conjunto de dados em determinadas condições. O trabalho realizado pelo autor decorre do desenvolvimento do projeto final da Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Politécnico do Porto.

1.2 Objetivos

Para a realização deste projeto, foram definidos alguns objetivos principais, sendo que parte deles contêm subobjetivos. Com a finalidade de manter uma boa coesão e garantia de qualidade do projeto, foram definidas as seguintes necessidades:

- Estudar séries temporais - visto que o projeto se trata da análise de algoritmo de aprendizagem com base em séries temporais, estes dados devem também ser estudados e compreendidos de modo obter uma melhor assimilação do comportamento dos modelos e dos resultados obtidos;
- Estudar vários algoritmos de aprendizagem, entre eles, *ARIMA*, *ARIMAX*, *SARIMA* e *SARIMAX*. Cada modelo deve ser estudado e compreendido de forma parametrizada para que a integração com qualquer conjunto de dados não cause nenhum problema;
- Desenvolver uma plataforma de testes de modelos de séries temporais com os algoritmos de aprendizagem estudados;
- Encontrar melhores modelos para os casos estudados e os respetivos *datasets* - de modo a concretizar este objetivo, é necessário o estudo de alguns métodos específicos, nomeadamente a pesquisa em grelha (*grid search*);
- Adaptar *scripts* para correr em ambiente do *Google Colab*.

1.3 Resultados

Com a realização deste projeto, espera-se o desenvolvimento de uma plataforma de testes de modelos de séries temporais. A plataforma deve conseguir testar vários algoritmos sobre um *dataset*. Deve também ser necessário realizar *grid searches*, isto é, executar todos os modelos com todas as configurações para descobrir melhores configurações para um determinado *dataset*.

Para que a plataforma funcione sem qualquer tipo de falhas, todo o código desenvolvido deverá ser adaptado para ser executado em ambiente de *Google Colab*, de modo a facilitar a manutenção do ambiente de trabalho e a fim de garantir um bom desempenho da máquina no decorrer dos testes.

1.4 Estrutura

O presente documento está dividido em 6 capítulo principais:

1. Contextualização e Motivação - este capítulo pretende apresentar de forma sucinta e objetiva as circunstâncias que fizeram surgir o projeto, contendo o contexto, objetivos e resultados do mesmo.
2. Fundamentação Teórica - neste capítulo são abrangidos, de uma forma teórica, os principais tópicos do projeto. Dá-se também a conhecer as tecnologias utilizadas.
3. Concetualização do Problema - nesta secção estão detalhados os requisitos do trabalho.
4. Metodologia de Operacionalização do Trabalho - neste capítulo é explorada e a metodologia de trabalho utilizada e a arquitetura concetual. Contém, também, como foi desenvolvido o projeto e apresenta a plataforma *Google Colab*.
5. Discussão dos Resultados - é nesta secção que serão discutidos os resultados obtidos e a explicação de possíveis controversias.
6. Conclusão - este capítulo visa realizar uma conclusão global do projeto e, por fim, uma descrição de possíveis melhoramentos futuros no trabalho desenvolvido.

Capítulo 2

Fundamentação Teórica

2.1 Série Temporal

Uma série temporal é uma sequência de dados numéricos ordenados sequencialmente ao longo do tempo[1]. A ordem dos dados é extremamente fundamental, ao contrário de modelos de regressão linear. Alguns exemplos destas séries são o registo das alturas das ondes do oceano, contagens de manchas solares, rastreio de dados meteorológicos ou medição de velocidades.

Dependendo se o interesse é entender um conjunto de dados ou fazer previsões, os objetivos adotados são diferentes. Compreender um *dataset* , chamado de análise de séries temporais, pode ajudar a fazer melhores previsões, no entanto, não é obrigatório e pode resultar num grande investimento técnico em tempo e experiência não diretamente alinhado com os resultados esperados, que é prever o futuro.

2.1.1 Análise

A análise de uma série temporal envolve o desenvolvimento de modelos que melhor capturam ou descrevem a mesma observada a fim de compreender as causas subjacentes[1]. Este campo de estudo procura o "porquê" por trás de um conjunto de dados da série. Isto, geralmente, envolve fazer suposições sobre a forma dos dados e decompô-la em componentes de constituição.

A qualidade de um modelo descritivo é determinada pelo quão bem ele descreve todos os dados disponíveis e pela interpretação que fornece para informar melhor o domínio do problema[2].

De modo a realizar esta análise, existem alguns métodos que permitem extrair estatísticas significativas, visualizar parte dos dados e exportar gráficos explicativos (e.g., gráficos lineares, histogramas, diagramas de extremos e quartis, mapas de calor, gráficos de dispersão, gráficos de autocorrelação)[3]. Na figura 2.1 pode ser analisada uma série temporal através do gráfico linear representado.

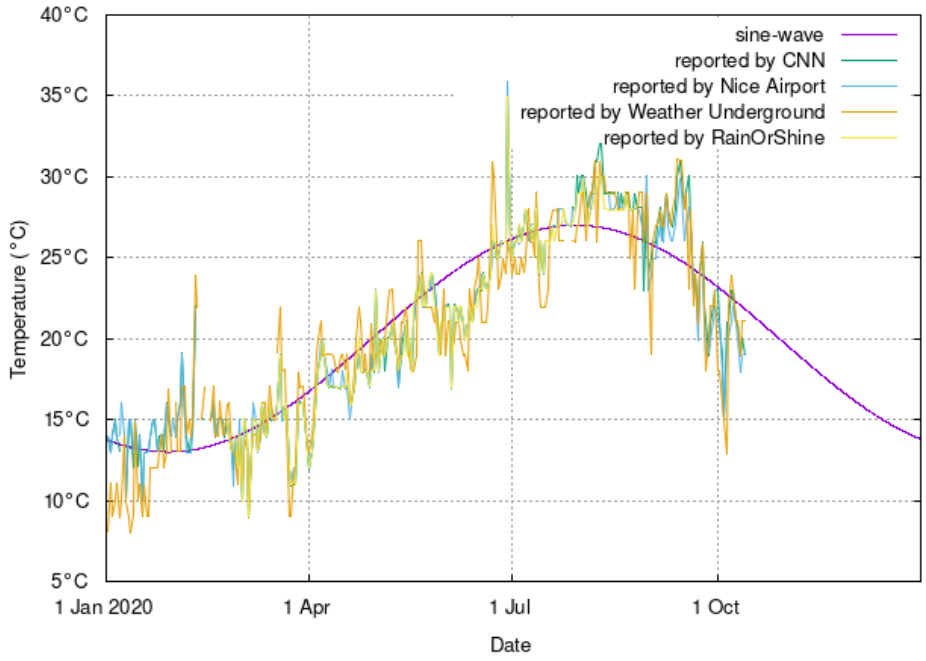


Figura 2.1: Temperaturas da cidade de Nice durante o ano atual

2.1.2 Componentes

Existem 4 componentes que uma série temporal pode ter[4]:

- Nível: valor base da série se fosse uma linha reta;
- Tendência (opcional): comportamento, normalmente linear, crescente ou decrescente ao longo do tempo;
- Sazonalidade (opcional): padrões repetitivos ou ciclos de comportamento ao longo do tempo;
- Ruído (opcional): variações nas observações que não podem ser explicadas pelo modelo.

Para concluir, todas as séries temporais têm nível e a maior parte também tem ruído. No entanto, a tendência e a sazonalidade são ocasionais[5].

2.1.3 Previsões

Fazer previsões sobre o futuro é chamado de extrapolação no tratamento estatístico clássico de dados de séries temporais. Campos mais modernos focam-se no tópico e referem-se a ele como previsão destas séries. A previsão envolve o ajuste de modelos em dados históricos e o uso destes dados para prever observações futuras. Posto isto, todos os dados de treino devem permanecer guardados e ordenados temporalmente, para que não hajam problemas com as previsões[6].

Os modelos descritivos podem ser emprestados para o futuro, ou seja, para suavizar ou remover o ruído. Um distinção importante na previsão é que o futuro está completamente indisponível e deve ser estimado apenas a partir do que já aconteceu.

A habilidade de um modelo de previsão de série temporal é determinada pelo seu desempenho em prever o futuro. Muitas vezes, isso ocorre às custas de ser capaz de explicar por que uma previsão específica foi feita, os intervalos de confiança e ainda compreender as causas subjacentes ao problema.

A previsão de séries temporais é uma área bastante importante de *Machine Learning (ML)* que é frequentemente negligenciada[7]. É muito importante, porque existem muitos problemas de previsão que envolvem um componente de tempo. Esses problemas são esquecidos com regularidade, pois é esse componente de tempo que os torna mais difíceis de resolver.

Para fazer previsões com séries temporais, é utilizado um modelo para prever valores futuros com base em valores observados anteriormente[8]. Na figura 2.2 está representado um gráfico das previsões de vendas de um champô. A azul estão os dados de treino do modelo, a verde estão os dados de teste e a vermelho as previsões realizadas pelo modelo. Este gráfico foi retirado dos exercícios realizados pelo autor aquando do estudo do modelo *ARIMA*.

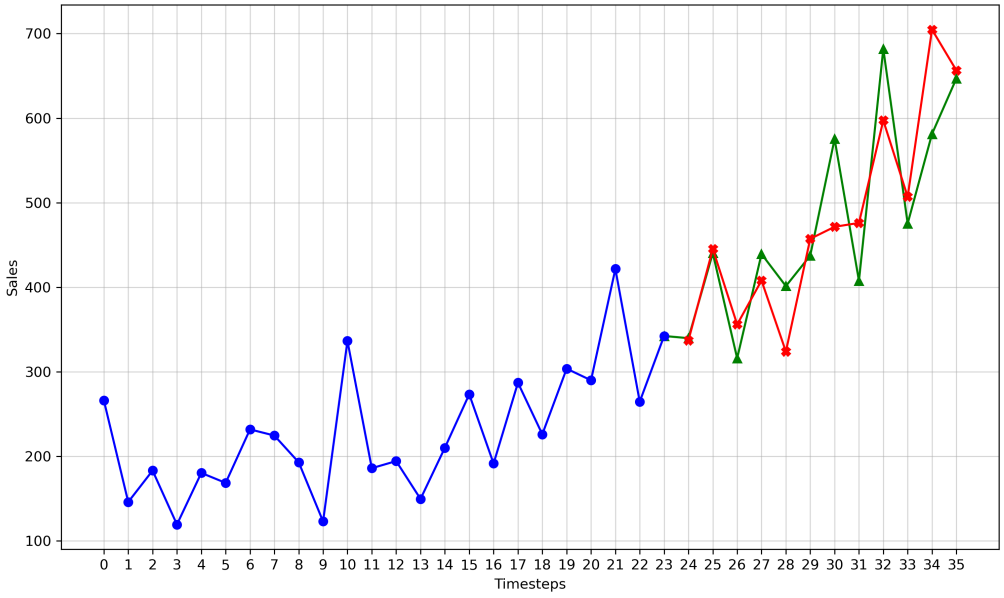


Figura 2.2: Gráfico de previsões de vendas de um champô

Como é possível ser observado na imagem, é necessário manter os dados de treino para fazer as previsões. Estes dados devem estar ordenados temporalmente, visto que estamos a trabalhar com modelos de auto regressão[9].

2.1.4 Univariada vs Multivariada

O termo "série temporal univariada" refere-se a uma série temporal que consiste em observações únicas (escalares) registadas sequencialmente em incrementos de tempo iguais. Por exemplo, as concentrações mensais de CO2 ou as medições das velocidades do vento.

Embora um *dataset* de uma série temporal univariada seja geralmente fornecido como uma única coluna de dados, o tempo é, de facto, uma variável implícita na série temporal[10].

Se os dados tiverem espaçamento igual, a variável de tempo, ou índice, não precisa de ser fornecida explicitamente. Esta variável às vezes pode ser usada explicitamente para traçar a série. No entanto, não é usada no próprio modelo. A tabela 2.1 demonstra um excerto de uma série temporal univariada para servir de exemplo.

"date"	"census"
01/01/2016	4092
02/01/2016	3807
03/01/2016	3208
04/01/2016	1057
05/01/2016	3302
06/01/2016	3761
07/01/2016	2964

Tabela 2.1: Excerto de série temporal univariada

Por outro lado, uma série temporal multivariada abrange várias variáveis que são registadas simultaneamente ao longo do tempo. Estas variáveis podem ou não contribuir para a previsão do modelo. Na figura 2.2 pode ser visualizado um excerto de uma série temporal multivariada.

"date"	"wifi status"	"temp"	"precipitation"	"census"
01/01/2016	up	-6	0	4092
02/01/2016	up	2	0	3807
03/01/2016	up	2	0	3208
04/01/2016	up	-4	0	1057
05/01/2016	up	2	0	3302
06/01/2016	up	2	0	3761
07/01/2016	up	4	0	2964

Tabela 2.2: Excerto de série temporal multivariada

2.2 Modelo ARIMA

O modelo de Média Móvel Integrada Autoregressiva[11] - *Autoregressive Integrated Moving Average (ARIMA)* - foi o modelo utilizado para realizar este trabalho, sendo que, todo o projeto foi abordado em torno deste modelo e variações do mesmo[12]. *ARIMA* significa:

- AR: "*Autoregression*"(Autoregressão) - Um modelo que usa a relação entre uma observação e um número de observações atrasadas;
- I: "*Integrated*"(Integrado) - O uso de diferenciação de observações (por exemplo, retirar uma observação de uma observação, no passo anterior) de forma a manter a série temporal estacionária;
- MA: "*Moving Average*"(Média Móvel) - Um modelo que usa a dependência entre uma observação e o erro residual de um modelo de média móvel aplicado a observações atrasadas.

O propósito de cada uma das características é fazer com que o modelo se ajuste aos dados da melhor forma possível.[13] Cada um destes componentes está especificado no modelo como um parâmetro. A notação utilizada é $ARIMA(p, d, q)$:

- p: Ordem de atraso - número de observações atrasadas incluídas no modelo;
- d: Grau de diferenciação - número de vezes que observações brutas são diferenciadas;
- q: Ordem da média móvel - tamanho da janela de media móvel.

Dado um conjungo de dados de uma série temporal X_t onde t é um índice inteiro e X_t são números reais, um modelo $ARMA(p', q)$ é dado por:

$$X_t - a_1 X_{t-1} - ... - a_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + ... + \theta_q \varepsilon_{t-q'}$$

ou equivalentemente por:

$$(1 - \sum_{i=1}^{p'} \alpha_i L^i) X_t = (1 + \sum_{i=1}^{q} \theta_i L^i) \varepsilon_t$$

2.3 Modelo ARIMAX

O modelo de Média Móvel Integrada Autoregressiva com Variáveis Exógenas - *Autoregressive Integrated Moving Average with Exogenous Variables (ARIMAX)* - é bastante semelhante ao modelo *ARIMA*. Existem algumas diferenças, embora não seja tão aparente[14].

Os componentes deste modelo são os mesmos que os do modelo *ARIMA* com o acréscimo das Variáveis Exógenas. Estas são variáveis cujos valores são determinados fora do modelo e são impostas ao modelo (neste caso serão utilizadas para ajudar a fazer as previsões).

2.4 Modelo SARIMA

O modelo de Média Móvel Integrada Autoregressiva Sazonal - *Seasonal Autoregressive Integrated Moving Average (SARIMA)* - é bastante semelhante ao modelo *ARIMA*. Existem mais diferenças que no modelo *ARIMAX*.

Os componentes deste modelo são os mesmos que os do modelo *ARIMA* com o acréscimo de um conjunto extra de parâmetros característico deste tipo de modelos[15]. São elas:

- P: Ordem de atraso sazonal;
- D: Grau de diferenciação sazonal;
- Q: Ordem da média móvel sazonal;
- S: Temporada - número de iterações por cada período sazonal (temporada).

Desta forma, o modelo será abordado com esta fórmula: $SARIMA(p,d,q)(P,D,Q,S)$. Este modelo é representado pela seguinte equação matemática[16]:

$$(1-\phi_1 B)(1-\Phi_1 B^4)(1-B)(1-B^4)y_t = (1+\theta_1 B)(1+\Theta_1 B^4)e_t$$

2.5 Modelo SARIMAX

O modelo de Média Móvel Integrada Autoregressiva Sazonal com Variáveis Exógenas - *Seasonal Autoregressive Integrated Moving Average with Exogenous Variables (SARIMAX)* - é uma junção dos modelos *SARIMA* e *ARIMAX*, visto que contém os parâmetros característicos dos modelos *SARIMA* e ainda contém Variáveis Exógenas[17].

2.6 Grid Search

Grid search (ou pesquisa em grelha) é o processo de coleção de dados para configurar os parâmetros ideais para um determinado modelo[18]. Dependendo do tipo de modelo utilizado, alguns parâmetros são necessários. Esta pesquisa não se aplica apenas a um tipo de modelo, ela pode ser aplicada ao modelo de aprendizagem para calcular os melhores parâmetros a serem usados para qualquer modelo[19]. Um exemplo desta pesquisa pode ser visualizado na figura 2.3, na qual está representado um mapa de calor resultante de uma *grid search* de parâmetros de um modelo.

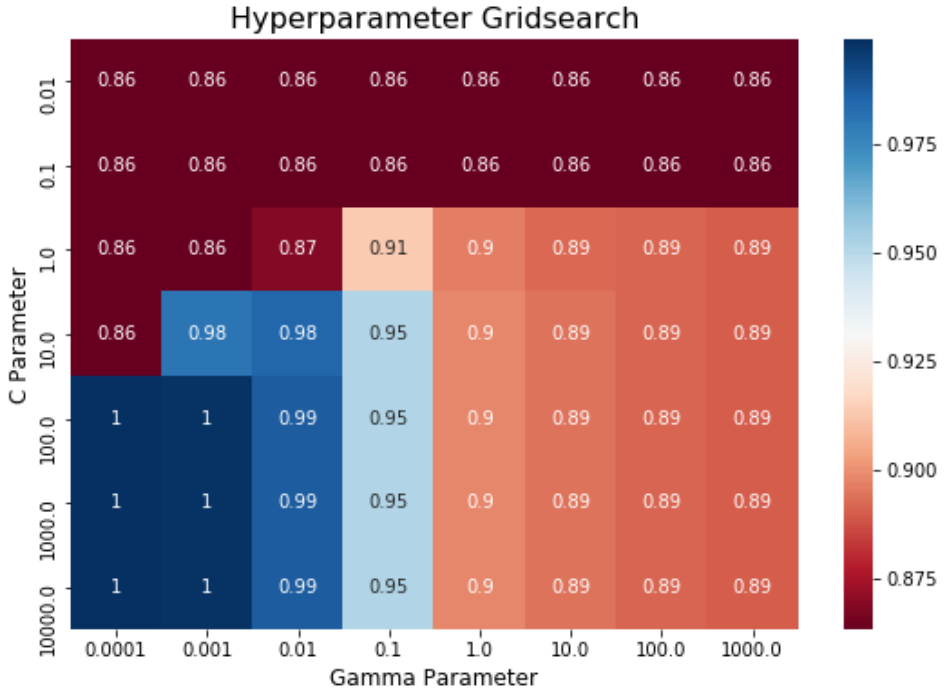


Figura 2.3: Mapa de calor representativo de uma *grid search* de parâmetros de um modelo

Um ponto importante a sublinhar é que esta pesquisa pode ser extremamente cara em termos computacionais e pode levar muito tempo até obter resultados. O *grid search* construirá um modelo em cada combinação de parâmetros possível. Ele itera por meio de cada combinação de parâmetros e armazena um modelo para cada combinação.

2.7 Tecnologias Utilizadas

2.7.1 Linguagens de Programação

Para a realização deste projeto foi utilizada a linguagem *Python*, visto que é uma das ou a melhor linguagem para realizar projetos de investigação de algoritmos de aprendizagem, pela enormíssima e vasta gama de bibliotecas fortíssimas, bastante úteis e adaptadas para o progresso do trabalho.

O *Python* é também uma linguagem bastante utilizada pela comunidade de desenvolvedores de software, ou seja, encontrar soluções para problemas que ocorram não será muito árduo.

2.7.2 Ambiente de Desenvolvimento

Com a finalidade de obter um bom, compacto e replicável ambiente de desenvolvimento, foi criado um ambiente de desenvolvimento virtual com *Python* e instaladas as dependências presentes no ficheiro *"requirements.txt"* presente na pasta principal do projeto.

Feito isto, foi utilizado o *IDE PyCharm* e o editor de código fonte *Visual Studio Code*. As tecnologias utilizadas no decorrer de todo o trabalho estão representadas na figura 3.3.

Figura 2.4: *Stack* tecnológica do projeto

Capítulo 3

Concetualização do Problema

3.1 Requisitos

Neste capítulo serão definidos os requisitos do projeto. Sejam eles tanto funcionais como não funcionais.

3.1.1 Funcionais

Representado na figura 3.1, está um diagrama de Casos de Uso demonstrativo da plataforma de testes desenvolvida.

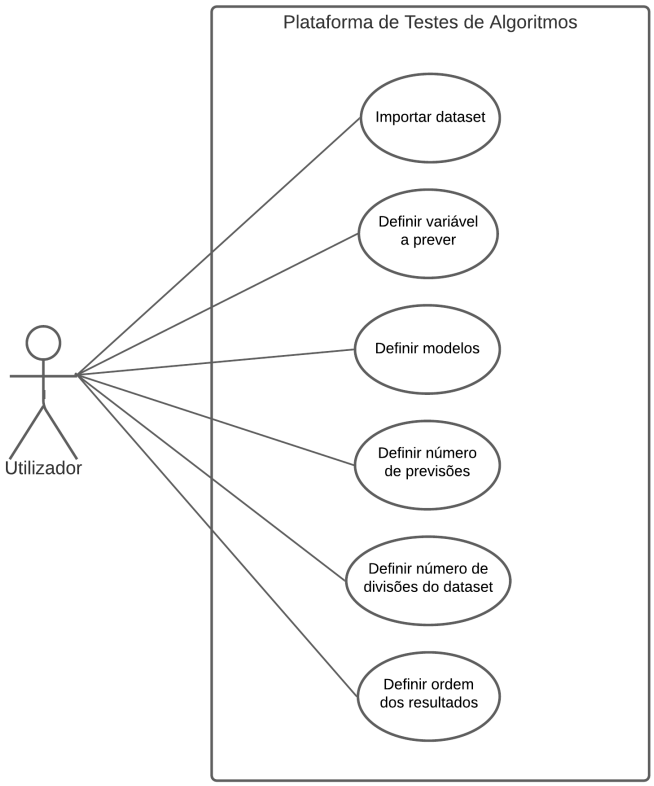


Figura 3.1: Diagrama de Casos de Uso da plataforma de testes

3.1.1.1 Carregar dados de *datasets*

Carregar os dados de um conjunto de dados é um requisito com prioridade alta. Caso seja necessário, deve ser desenvolvida uma pequena função de tratamento das datas do *dataset*. Utilizando uma biblioteca da linguagem *Python* (*pandas*), temos a capacidade de abrir um ficheiro *.csv* e carregá-lo para um objeto "*DataFrame*". Deste objeto, pode ser retirada a informação desejada do conjunto de dados ou até mesmo exportar um gráfico com os seus dados.

3.1.1.2 Dividir em *datasets* de treino e de teste

Em primeiro lugar, o utilizador deve introduzir o número de vezes que o dataset deve ser dividido, caso seja pretendido realizar validação cruzada nos testes dos modelos. Se não for introduzido nenhum valor, é subentendido que não será para utilizar esta técnica na realização dos testes.

Após carregar os dados do *dataset* os dados são separados em dados de treino e dados de teste. A quantidade de dados de teste pode ser definida pelo utilizador através de dois parâmetros (através da percentagem do dataset de teste ou através do número de previsões a realizar - caso seja introduzida uma percentagem do dataset de teste, o número de previsões é calculado a partir desse valor e é ignorado o número introduzido pelo utilizador).

3.1.1.3 Executar vários algorimos ao mesmo tempo

Este é talvez o requisito mais importante e mais meticuloso de todo o projeto. É essencial que o estudo dos modelos (*ARIMA*, *ARIMAX*, *SARIMA* e *SARIMAX*) seja bem feito e que exista uma boa compreensão dos mesmo também, caso contrário, não só a teoria do projeto fica mal entendida, como o projeto em si fica prejudicado pelo mau desenvolvimento da plataforma[20].

Para realizar a execução dos modelos, o utilizador deve indicar os algoritmos que pretende testar e os parâmetros comuns e não comuns de cada algoritmo.

3.1.1.4 Guardar dados de previsões bem sucedidas

Após a conclusão de cada modelo, caso tenha terminado com sucesso, deve ser exportado um ficheiro *.csv* com as previsões e dados de teste para dentro da pasta de cada modelo, que, por sua vez, se encontrará localizada na pasta global dos resultados.

3.1.1.5 Guardar gráficos de previsões bem sucedidas

Após a conclusão de cada modelo, caso tenha terminado com sucesso, deve ser exportado uma imagem (*.png*) com o gráfico das previsões do modelo. Tudo isto será exportado para dentro da pasta de cada modelo, que, por sua vez, se encontrará localizada na pasta global dos resultados.

3.1.1.6 Exportar ficheiro com sumário dos resultados de todos os modelos

No final, quando todos os modelos tiverem terminado, será exportado para a pasta global dos resultados um ficheiro *.csv* com uma avaliação de todos os modelos. Estes serão ordenados por um parâmetro definido pelo utilizador, podendo ser uma ordem por Tempo de Execução, Erro Médio Absoluto (Mean Absolute Error - MAE), Erro Quadrático Médio (Mean Squared Error - MSE) ou Raiz Quadrática Média do Erro (Root Mean Squared Error - RMSE).

3.1.1.7 Exportar ficheiro com o registo da execução de cada modelo

No final, quando todos os modelos tiverem terminado, será exportado para a pasta global dos resultados um ficheiro "*log.txt*" com a visão geral de cada modelo.

3.1.2 Não Funcionais

3.1.2.1 Continuar aplicação no caso da falha de algum modelo

A aplicação deve manter-se estável e sem qualquer tipo de problema caso algum dos modelos falhe. Nestes casos, a situação é reportada para o ficheiro de registos e no fim exportada com os resultados de todos os modelos.

3.1.2.2 Guardar dados de forma persistente

Os dados de cada modelo são guardados de forma persistente, isto é, mesmo que a aplicação seja encerrada manualmente, as previsões e os gráficos gerados até ao momento nunca serão perdidos. Este requisito é bastante importante neste tipo de projetos, visto que grande parte dos modelos falham ou por vezes não deixam a aplicação terminar, o que obriga a forçar o término da mesmo. No entanto, os dados dos modelos testados até ao momento permanecem intactos.

3.1.2.3 Adaptar *scripts* para correr em ambiente do *Google Colab*

Depois da plataforma de testes estar realizada, para finalizar, é necessário fazer a integração com o *Google Colab*, que é uma suíte de computação gratuita, oferecida pela *Google*. Feito isto, devem ser realizados alguns testes para comprovar a funcionalidade do projeto na plataforma desejada.

Capítulo 4

Metodologia de Operacionalização do Trabalho

4.1 Processo e Metodologia de Trabalho

Foi adotada uma abordagem iterativa incremental, com algumas reuniões de acompanhamento ao longo do projeto. No decorrer de 15 semanas, foram criadas 4 iterações pelas quais foram divididas as tarefas em cima descritas.

O *roadmap* do projeto, que se encontra representado na forma de um diagrama de Gantt, apresenta, mais detalhadamente, a atribuição das tarefas nas diferentes iterações e a duração das mesmas.

ID	Iteração	Tarefa	Duração	Início	Fim
1	1	Estudo de algumas bibliotecas de Python	17 dias	19-mai-20	5-jun-20
2		Realização de exercícios de ML c/ Python	17 dias	19-mai-20	5-jun-20
3	2	Estudo do modelo ARIMA	27 dias	5-jun-20	2-jul-20
4	3	Estudo das variantes do modelo ARIMA	35 dias	12-jun-20	17-jul-20
5		Estruturação dos scripts	14 dias	17-jul-20	31-jul-20
6		Realização dos scripts c/ Python	11 dias	15-set-20	26-set-20
7		Teste dos scripts	3 dias	26-set-20	29-set-20
8	4	Execução de grid searches	11 dias	29-set-20	10-out-20
9		Execução dos melhores modelos	11 dias	29-set-20	10-out-20
10		Estudo da plataforma Google Colab	6 dias	29-set-20	5-out-20
11		Colocar scripts na plataforma	5 dias	5-out-20	10-out-20
12		Correr alguns testes	5 dias	5-out-20	10-out-20
13		Elaboração do documento principal	14 dias	29-set-20	13-out-20

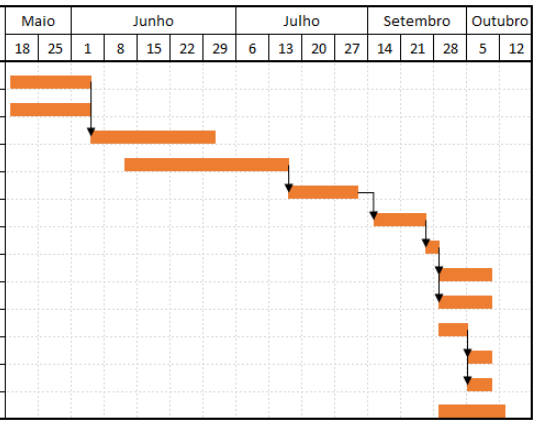


Figura 4.1: Diagrama de Gantt representativo do *roadmap* do projeto

4.2 Arquitetura Concetual

A arquitetura do projeto (representada na figura 4.2) garante três componentes principais: o conversor de *datasets* procura no diretório de *datasets*, dentro do sistem de ficheiros, e transforma o conjunto de dados selecionado num objeto *"DataFrame"* da biblioteca *pandas*; o otimizador de modelos permite testar um ou mais modelos com o *dataset* carregado anteriormente; o exportador de resultados e registos arquiva o sumários dos resultados (tempo de execução, *MAE*, *MSE*, *RMSE*) e os registos (mensagens de sucesso ou de erro) para todos os modelos testados na mesma sessão. Todo este ambiente pode ser implementado no *Google Colab*.

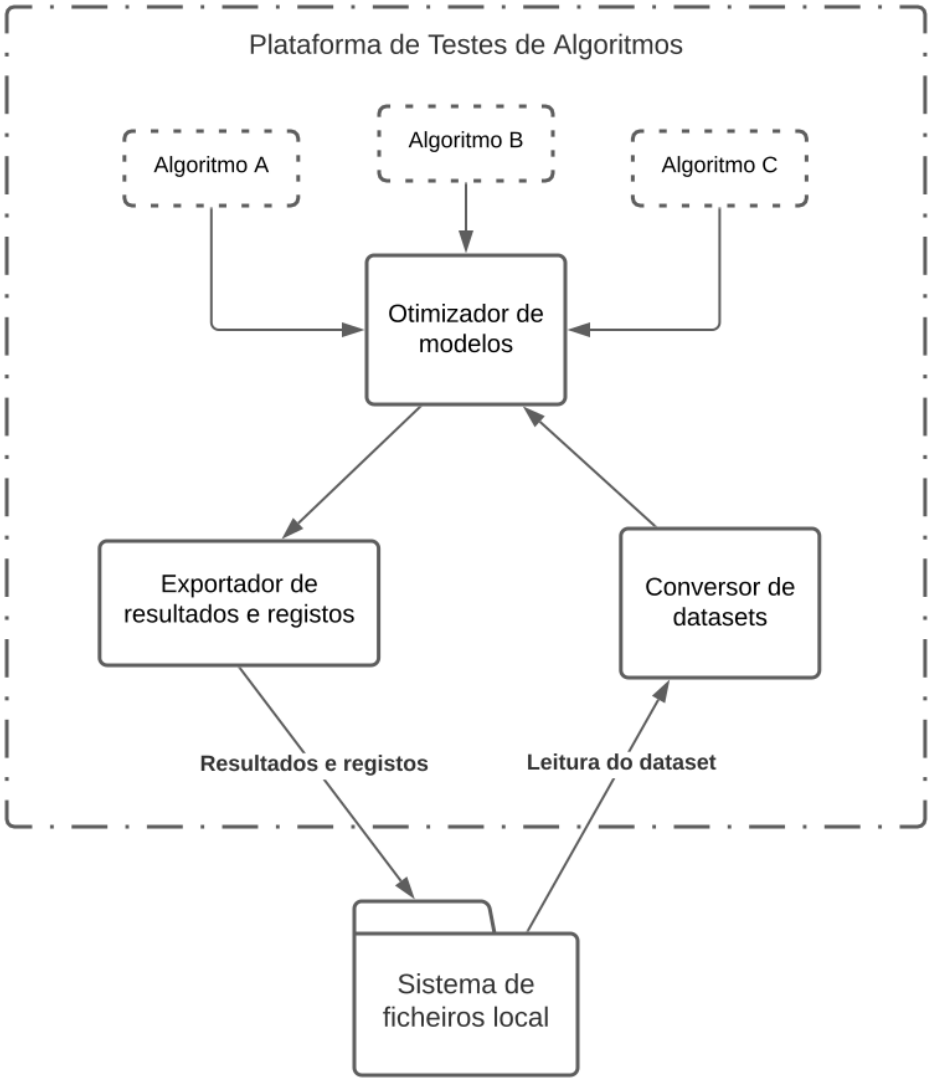


Figura 4.2: Arquitetura da aplicação

Na Programação Orientada a Objetos (POO), o encapsulamento refere-se ao agrupamento de dados com os métodos que operam nesses dados ou à restrição do acesso direto a alguns dos componentes de um objeto. [21] O encapsulamento é utilizado para ocultar os valores ou o estado de um objeto de dados estruturados dentro de uma classe, evitando o acesso direto de terceiros não autorizados a eles.

Este mecanismo não é exclusivo para POO. Implementação de dados abstratos, como por exemplo módulos, oferecem uma forma semelhante de encapsulamento. A semelhança foi explicada por teóricos da linguagem de programação em termos de tipos existenciais. [22]

Para a realização deste projeto foi necessário a utilização de encapsulamento, de modo a manter uma boa estrutura e organização do código.

Em relação à estrura das pesquisas em grelha, foram todas desenhadas e pensadas à medida do projeto, a fim de testar o maior número possível de modelos para um determinado *dataset*.

4.3 Desenvolvimento da Solução

O desenvolvimento dos algoritmos está representado no diagrama de classes presente na figura 4.3.

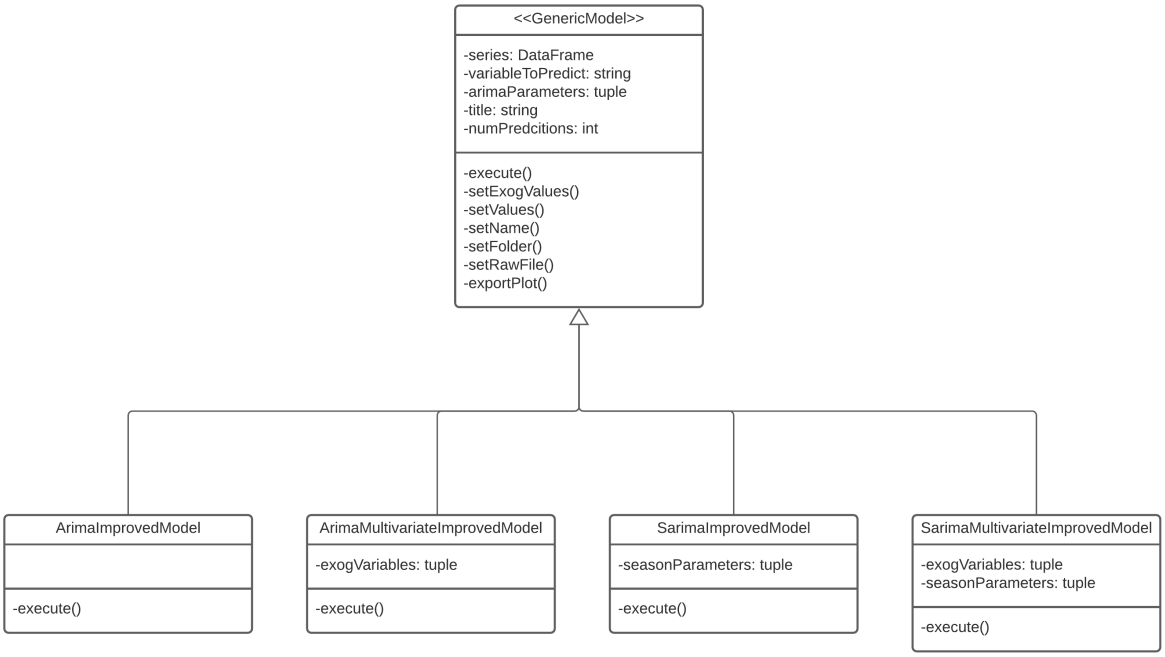


Figura 4.3: Diagrama de classes dos algoritmos implementados

4.3.1 Modelo Genérico

Em primeiro lugar, o *GenericModel* trata-se de um modelo genérico, como o próprio nome indica. Esta classe tem como objetivo ser uma classe abstrata, isto é, não deve ser instanciada, serve apenas para servir de modelo para outras classes. Neste caso em específico, como a linguagem utilizada é *Python* não existe nenhuma nomenclatura especial para este tipo de dados. Está representada no excerto de código 4.1.

```
    class GenericModel:
    """Class that represents the structure of the Generic Model for the creation
    of specific models


    Author: Luis Marques
    """

    MODEL_NAME: str = "GENERIC"

    def __init__(self, series: DataFrame,
                 variable_to_predict: str,
                 arima_parameters: tuple,
                 title: str = "",
                 num_splits: int = 0,
                 num_predictions: int = 10,
                 predictions_size: float = 0.0):
        """Creates an instance of an GenericModel.

        Args:
            series (DataFrame): series of the dataset to run the model.
            variable_to_predict (str): name of the variable to predict. It must
                be the same name of the column in the dataset.
            arima_parameters (tuple): parameters of the arima model.
            title (str): title of the model. Used to differentiate this model
                from other ones with the same parameters. Defaults to "".
            num_splits (int): number of splits to do in the dataset. Defaults
                to 0.
            num_predictions (int): number of predictions of the model. Defaults
                to 10. It will only have effect if the predictions_size is equal
                to zero.
            predictions_size (float): percentage of data to predict (from 0 to
                1). Defaults to 0.
        """

        self.series = series
        self.variable_to_predict = variable_to_predict
        self.arima_parameters = arima_parameters
        self.title = title
        if predictions_size == 0.0:
            self.num_predictions = num_predictions
        else:
            self.num_predictions = int(len(self.values) * predictions_size)
        self.data_split = 0
        self._set_values()
        self.predictions = list()
        if num_splits == 0:
            self.train = self.values[:-self.num_predictions]
            self.test = self.values[-self.num_predictions:]
            self.history = [x for x in self.train]
            self._set_name()
            self._set_folder()
```

```
            self._set_raw_file()
            self._execute()
        else:
            for train_index, test_index in TimeSeriesSplit(n_splits=num_splits).
split(self.values):
                self.train = self.values[train_index].copy()
                self.test = self.values[test_index].copy()
                self.train = [*self.train, *self.test[:-self.num_predictions]]
                self.test = self.test[-self.num_predictions:]
                self.data_split += 1
                self.history = [x.copy() for x in self.train]
                self._set_name()
                self._set_folder()
                self._set_raw_file()
                self._execute()

    def _execute(self):
        """Executes the model. This method should be implemented in each
        individual model class"""
        pass

    def _set_exog_values(self):
        """Sets the values arrays to be used based on the series and the
        variable to predict. Also sets the exog values to be used in the
        predictions."""
        if hasattr(self, "exog_variables"):
            self.exog_values = self.series.filter(items=self.exog_variables)
            self.exog_values = self.exog_values.values

    def _set_values(self):
        """Sets the values to be used based on the series and the variable to
        predict"""
        self._set_exog_values()
        self.scaler = MinMaxScaler()
        self.values = self.series[[self.variable_to_predict]]
        self.values[[self.variable_to_predict]] = self.scaler.fit_transform(self
.values[[self.variable_to_predict]])
        self.values = getattr(self.values, self.variable_to_predict).values

    def _set_name(self):
        """Sets the name of the model according to its variables"""
        self.name = ""
        self.title = "".join(self.title.split())
        if self.title != "":
            self.name += f"{self.title}_"
        self.name += f"{self.MODEL_NAME}("
        for parameter in self.arima_parameters:
            self.name += f"{str(parameter)},"
        if hasattr(self, "season_parameters"):
            self.name = self.name[:-1] + ")("
            for parameter in self.season_parameters:
                self.name += f"{str(parameter)},"
```

```
103         self.name = self.name[:-1] + ")_"
104         self.name += f"predictions_{str(self.num_predictions)}"
105         if self.data_split != 0:
106             self.name += f"_crossvalidation_{self.data_split}"
107 
108     def _set_folder(self):
109         """Creates an output folder for the model"""
110         self.folder = os.path.join(OUTPUT_FOLDER, self.name)
111         try:
112             os.makedirs(self.folder)
113         except FileNotFoundError:
114             shutil.rmtree(self.folder)
115             os.makedirs(self.folder)
116 
117     def _set_raw_file(self):
118         """Creates a raw .csv file to write the predictions of the model"""
119         file_name = f"raw_{self.name}.csv"
120         file_path = os.path.join(self.folder, file_name)
121         self.file = CSVWriter(file_path, ("Predict", self.variable_to_predict))
122 
123     def _export_plot(self):
124         """Exports the plot of the model"""
125         timesteps = numpy.arange(self.num_predictions)
126 
127         real_values = ([x for x in self.test])
128 
129         prediction_values = ([x for x in self.predictions])
130 
131         pyplot.plot(timesteps,
132                     real_values,
133                     color="green",
134                     marker="^",
135                     label="Real values")
136 
137         pyplot.plot(timesteps,
138                     prediction_values,
139                     color="red",
140                     marker="X",
141                     label="Predictions")
142 
143         pyplot.ylabel(self.variable_to_predict)
144         pyplot.xlabel("Timesteps")
145         pyplot.xticks(numpy.arange(min(timesteps), max(timesteps) + 1, 1.0))
146         pyplot.grid(which="major", alpha=0.5)
147         pyplot.gcf().canvas.set_window_title(self.name)
148         pyplot.gcf().set_size_inches(8, 5)
149         pyplot.savefig(os.path.join(self.folder, f"plot_{self.name}.png"),
150                        format="png", dpi=300)
151         pyplot.close()
```

Excerto 4.1: Classe do modelo genérico

Os parâmetros do método construtor deste modelo genérico são os seguintes:

- *series* - série temporal para testar o modelo;
- *variable_to_predict* - nome da variável do *dataset* a prever;
- *title* - título do modelo, utilizado para diferenciar o modelo dos restantes. Tem como valor por defeito uma *string*;
- *num_splits* - número de divisões a realizar no conjunto de dados antes de realizar os testes. Este parâmetro é utilizado caso seja necessário fazer uma validação cruzada. Tem zero como valor por defeito;
- *num_predictions* - número de previsões a realizar. Este parâmetro apenas será válido se não for definido nenhuma percentagem do *dataset* para testes. Tem zero como valor por defeito;
- *predictions_size* - percentagem do *dataset* para testes. Se este parâmetro for diferente de zero, o *"num_predictions"* não terá efeito. Tem zero como valor por defeito.

Em relação aos métodos, esta classe é constituída pelos seguintes:

- *_execute(self)* - este método deve ser implementado por cada classe que extenda a classe do modelo genérico;
- *_set_exog_values(self)* - define uma lista de valores das variáveis exógenas;
- *_set_values(self)* - define uma lista de valores da variável com que vão ser realizadas as previsões;
- *_set_name(self)* - define o nome do modelo, tendo em conta o seu título e as configurações;
- *_set_folder(self)* - cria um diretório para o exportar o gráfico e as previsões do modelo;
- *_set_raw_file(self)* - cria um ficheiro *.csv* para escrever as previsões do modelo;
- *_export_plot(self)* - exporta o gráfico do modelo.

Na linguagem *Python* não existe nenhuma palavra chave para declarar tipos de dados privados, existe apenas uma convenção que, geralmente, é respeitada. Esta convenção define que, de modo a declarar tipos de dados privados, deve ser colocado um *underscore* antes de definir a variável ou método a utilizar. Posto isto, todos os métodos da classe genérica estão "declarados" privados, pois não devem ser chamados fora da classe, isto é, eles são invocados no momento da instanciação de um novo modelo.

4.3.2 Classe *ARIMA*

O modelo *Autoregressive Integrated Moving Average (ARIMA)* está representado no código através da classe *ArimaImprovedModel*, demonstrada no excerto de código 4.2. Esta classe extende a classe do modelo genérico e apenas é necessário implementar o método de execução dos testes.

```
class ArimaImprovedModel(GenericModel):
    """Class that represents the structure of an ARIMA improved model

    Author: Luis Marques
    """

    MODEL_NAME: str = "ARIMA"

    def __init__(self, series: DataFrame,
                 variable_to_predict: str,
                 arima_parameters: tuple,
                 title: str = "",
                 num_splits: int = 0,
                 num_predictions: int = 10,
                 predictions_size: float = 0.0):
        """Creates an instance of an ArimaImprovedModel.

        Args:
            series (DataFrame): series of the dataset to run the model.
            variable_to_predict (str): name of the variable to predict. It must
                be the same name of the column in the dataset.
            arima_parameters (tuple): parameters of the arima model.
            title (str): title of the model. Used to differentiate this model
                from other ones with the same parameters. Defaults to "".
            num_splits (int): number of splits to do in the dataset. Defaults
                to 0.
            num_predictions (int): number of predictions of the model. Defaults
                to 10. It will only have effect if the predictions_size is equal
                to zero.
            predictions_size (float): percentage of data to predict (from 0 to
                1). Defaults to 0.
        """
        super().__init__(series, variable_to_predict, arima_parameters, title,
                         num_splits, num_predictions, predictions_size)

    def _execute(self):
        """Executes the model"""
        self.starting_time = time.time()
        try:
            predictions = list()
            for timestep in range(self.num_predictions):
                model = ARIMA(self.history, order=self.arima_parameters)
                model_fit = model.fit(disp=0)
                output = model_fit.forecast()
                prediction = output[0][0]
```

```
                predictions.append(prediction)
                obs = self.test[timestep]
                self.history.append(obs)

            self.predictions = self.scaler.inverse_transform([predictions])[0]
            self.test = self.scaler.inverse_transform([self.test])[0]

            for timestep in range(self.num_predictions):
                self.file.write_line((str(self.predictions[timestep]),
                                      str(self.test[timestep])))

            self._export_plot()

        except Exception as err:
            logs.append(f"LOG: Model {self.name} exported with an error! {type(
    err).__name__}: {err}")
            self.execution_time = -1
            self.mae = -1
            self.mse = -1
            self.rmse = -1
            # If it returns an error the model folder is removed
            self.file.close()
            shutil.rmtree(self.folder)

        else:
            logs.append(f"LOG: Model {self.name} exported with success.")
            self.execution_time = time.time() - self.starting_time
            self.mae = mean_absolute_error(self.test, self.predictions)
            self.mse = mean_squared_error(self.test, self.predictions)
            self.rmse = sqrt(self.mse)
            self.file.close()

        finally:
            results.append((f'"{self.name}"', str(self.execution_time),
                            str(self.mae), str(self.mse), str(self.rmse)))
            print(f"Model {self.name} finished.")
```

Excerto 4.2: Classe do modelo *ARIMA*

Nesta classe não existe nenhum parâmetro adicional, para além dos já definidos no modelo genérico. Quanto aos métodos, apenas o método *"_execute(self)"* foi implementado para realizar as previsões de acordo com o algoritmo *ARIMA*.

4.3.3 Classe *ARIMAX*

O modelo *Autoregressive Integrated Moving Average with Exogenous Variables (ARIMAX)* está representado no código através da classe *ArimaMultivariateImprovedModel*, demonstrada no excerto de código 4.3. Esta classe extende a classe do modelo genérico e apenas é necessário implementar o método de execução dos testes.

```
class ArimaMultivariateImprovedModel(GenericModel):
    """Class that represents the structure of an ARIMAX improved model

    Author: Luis Marques
    """

    MODEL_NAME = "ARIMAX"

    def __init__(self, series: DataFrame,
                 variable_to_predict: str,
                 exog_variables: tuple,
                 arima_parameters: tuple,
                 title: str = "",
                 num_splits: int = 0,
                 num_predictions: int = 10,
                 predictions_size: float = 0.0):
        """Creates an instance of an ArimaMultivariateImprovedModel.

        Args:
            series (DataFrame): series of the dataset to run the model.
            variable_to_predict (str): name of the variable to predict. It must
                be the same name of the column in the dataset.
            exog_variables (tuple): tuple of exogenous variables to help the
                model to predict the values.
            arima_parameters (tuple): parameters of the arima model.
            title (str): title of the model. Used to differentiate this model
                from other ones with the same parameters. Defaults to "".
            num_splits (int): number of splits to do in the dataset. Defaults
                to 0.
            num_predictions (int): number of predictions of the model. Defaults
                to 10. It will only have effect if the predictions_size is equal
                to zero.
            predictions_size (float): percentage of data to predict (from 0 to
                1). Defaults to 0.
        """
        self.exog_variables = exog_variables
        super().__init__(series, variable_to_predict, arima_parameters, title,
                         num_splits, num_predictions, predictions_size)

    def _execute(self):
        """Executes the model"""
        self.starting_time = time.time()
        try:
            predictions = list()
```

```
45              history_extra = [x.copy() for x in self.exog_values[:len(self.
    history)]]
46              for timestep in range(self.num_predictions):
47                  test_extra = tuple(self.exog_values[-self.num_predictions +
    timestep])
48                  model = ARIMA(self.history, order=self.arima_parameters,
49                                exog=history_extra)
50                  model_fit = model.fit(disp=0)
51                  output = model_fit.forecast(exog=test_extra)
52                  prediction = output[0][0]
53                  predictions.append(prediction)
54                  obs = self.test[timestep]
55                  self.history.append(obs)
56                  history_extra.append(obs)
57
58              self.predictions = self.scaler.inverse_transform([predictions])[0]
59              self.test = self.scaler.inverse_transform([self.test])[0]
60
61              for timestep in range(self.num_predictions):
62                  self.file.write_line((str(self.predictions[timestep]),
63                                        str(self.test[timestep])))
64
65              self._export_plot()
66
67          except Exception as err:
68              logs.append(f"LOG: Model {self.name} exported with an error! {type(
    err).__name__}: {err}")
69              self.execution_time = -1
70              self.mae = -1
71              self.mse = -1
72              self.rmse = -1
73              self.file.close()
74              # If it returns an error the model folder is removed
75              shutil.rmtree(self.folder)
76
77          else:
78              logs.append(f"LOG: Model {self.name} exported with success.")
79              self.execution_time = time.time() - self.starting_time
80              self.mae = mean_absolute_error(self.test, self.predictions)
81              self.mse = mean_squared_error(self.test, self.predictions)
82              self.rmse = sqrt(self.mse)
83              self.file.close()
84
85          finally:
86              results.append((f'"{self.name}"', str(self.execution_time),
87                              str(self.mae), str(self.mse), str(self.rmse)))
88              print(f"Model {self.name} finished.")
```

Excerto 4.3: Classe do modelo *ARIMAX*

Esta classe, para além dos parâmetros já definidos no modelo genérico, ainda conta com o parâmetro *"exog_variables"* que representa as variáveis exógenas utilizadas para realizar as previsões. Quanto aos métodos, apenas o método *"_execute(self)"* foi implementado para realizar as previsões de acordo com o algoritmo *ARIMAX*.

4.3.4 Classe *SARIMA*

O modelo *Seasonal Autoregressive Integrated Moving Average (SARIMA)* está representado no código através da classe *SarimaImprovedModel*, demonstrada no excerto de código 4.4. Esta classe extende a classe do modelo genérico e apenas é necessário implementar o método de execução dos testes.

```
class SarimaImprovedModel(GenericModel):
    """Class that represents the structure of a SARIMA improved model

    Author: Luis Marques
    """

    MODEL_NAME = "SARIMA"

    def __init__(self, series: DataFrame,
                 variable_to_predict: str,
                 arima_parameters: tuple,
                 season_parameters: tuple,
                 title: str = "",
                 num_splits: int = 0,
                 num_predictions: int = 10,
                 predictions_size: float = 0.0):
        """Creates an instance of an SarimaImprovedModel.


        Args:
            series (DataFrame): series of the dataset to run the model.
            variable_to_predict (str): name of the variable to predict. It must
                be the same name of the column in the dataset.
            arima_parameters (tuple): parameters of the arima model.
            season_parameters (tuple): season parameters of the sarima model.
            title (str): title of the model. Used to differentiate this model
                from other ones with the same parameters. Defaults to "".
            num_splits (int): number of splits to do in the dataset. Defaults
                to 0.
            num_predictions (int): number of predictions of the model. Defaults
                to 10. It will only have effect if the predictions_size is equal
                to zero.
            predictions_size (float): percentage of data to predict (from 0 to
                1). Defaults to 0.
        """
        self.season_parameters = season_parameters
        super().__init__(series, variable_to_predict, arima_parameters, title,
                         num_splits, num_predictions, predictions_size)

    def _execute(self):
        """Executes the model"""
        self.starting_time = time.time()
        try:
            predictions = list()
            for timestep in range(self.num_predictions):
```

```
                model = SARIMAX(self.history, order=self.arima_parameters,
                                seasonal_order=self.season_parameters,
                                enforce_stationarity=False,
                                enforce_invertibility=False)
                model_fit = model.fit(disp=0)
                output = model_fit.forecast()
                prediction = output[0]
                predictions.append(prediction)
                obs = self.test[timestep]
                self.history.append(obs)

            self.predictions = self.scaler.inverse_transform([predictions])[0]
            self.test = self.scaler.inverse_transform([self.test])[0]

            for timestep in range(self.num_predictions):
                self.file.write_line((str(self.predictions[timestep]),
                                      str(self.test[timestep])))

            self._export_plot()

        except Exception as err:
            logs.append(f"LOG: Model {self.name} exported with an error! {type(
    err).__name__}: {err}")
            self.execution_time = -1
            self.mae = -1
            self.mse = -1
            self.rmse = -1
            self.file.close()
            # If it returns an error the model folder is removed
            shutil.rmtree(self.folder)

        else:
            logs.append(f"LOG: Model {self.name} exported with success.")
            self.execution_time = time.time() - self.starting_time
            self.mae = mean_absolute_error(self.test, self.predictions)
            self.mse = mean_squared_error(self.test, self.predictions)
            self.rmse = sqrt(self.mse)
            self.file.close()

        finally:
            results.append((f'"{self.name}"', str(self.execution_time),
                            str(self.mae), str(self.mse), str(self.rmse)))
            print(f"Model {self.name} finished.")
```

Excerto 4.4: Classe do modelo *SARIMA*

Esta classe, para além dos parâmetros já definidos no modelo genérico, ainda conta com o parâmetro *"season_parameters"* que representa os parâmetros sazonais utilizados em modelos *SARIMA*. Quanto aos métodos, apenas o método *"_execute(self)"* foi implementado para realizar as previsões de acordo com o algoritmo *SARIMA*.

4.3.5 Classe *SARIMAX*

O modelo *Seasonal Autoregressive Integrated Moving Average with Exogenous Variables (SARIMAX)* está representado no código através da classe *SarimaMultivariateImprovedModel*, demonstrada no excerto de código 4.5. Esta classe extende a classe do modelo genérico e apenas é necessário implementar o método de execução dos testes.

```
class SarimaMultivariateImprovedModel(GenericModel):
    """Class that represents the structure of a SARIMAX improved model

    Author: Luis Marques
    """

    MODEL_NAME = "SARIMAX"

    def __init__(self, series: DataFrame,
                 variable_to_predict: str,
                 exog_variables: tuple,
                 arima_parameters: tuple,
                 season_parameters: tuple,
                 title: str = "",
                 num_splits: int = 0,
                 num_predictions: int = 10,
                 predictions_size: float = 0.0):
        """Creates an instance of an SarimaMultivariateImprovedModel.

        Args:
            series (DataFrame): series of the dataset to run the model.
            variable_to_predict (str): name of the variable to predict. It must
                be the same name of the column in the dataset.
            exog_variables (tuple): tuple of exogenous variables to help the
                model to predict the values.
            arima_parameters (tuple): parameters of the arima model.
            season_parameters (tuple): season parameters of the sarima model.
            title (str): title of the model. Used to differentiate this model
                from other ones with the same parameters. Defaults to "".
            num_splits (int): number of splits to do in the dataset. Defaults
                to 0.
            num_predictions (int): number of predictions of the model. Defaults
                to 10. It will only have effect if the predictions_size is equal
                to zero.
            predictions_size (float): percentage of data to predict (from 0 to
                1). Defaults to 0.
        """
        self.exog_variables = exog_variables
        self.season_parameters = season_parameters
        super().__init__(series, variable_to_predict, arima_parameters, title,
                         num_splits, num_predictions, predictions_size)

    def _execute(self):
        """Executes the model"""
        self.starting_time = time.time()
```

```
46          try:
47              predictions = list()
48              history_extra = [x.copy() for x in self.exog_values[:len(self.
    history)]]
49              for timestep in range(self.num_predictions):
50                  test_extra = tuple(self.exog_values[-len(self.test) + timestep])
51                  model = SARIMAX(self.history, exog=history_extra,
52                                  order=self.arima_parameters,
53                                  seasonal_order=self.season_parameters,
54                                  enforce_stationarity=False,
55                                  enforce_invertibility=False)
56                  model_fit = model.fit(disp=0)
57                  output = model_fit.forecast(exog=test_extra)
58                  prediction = output[0]
59                  predictions.append(prediction)
60                  obs = self.test[timestep]
61                  self.history.append(obs)
62
63              self.predictions = self.scaler.inverse_transform([predictions])[0]
64              self.test = self.scaler.inverse_transform([self.test])[0]
65
66              for timestep in range(self.num_predictions):
67                  self.file.write_line((str(self.predictions[timestep]),
68                                        str(self.test[timestep])))
69
70              self._export_plot()
71
72          except Exception as err:
73              logs.append(f"LOG: Model {self.name} exported with an error! {type(
    err).__name__}: {err}")
74              self.execution_time = -1
75              self.mae = -1
76              self.mse = -1
77              self.rmse = -1
78              self.file.close()
79              # If it returns an error the model folder is removed
80              shutil.rmtree(self.folder)
81
82          else:
83              logs.append(f"LOG: Model {self.name} exported with success.")
84              self.execution_time = time.time() - self.starting_time
85              self.mae = mean_absolute_error(self.test, self.predictions)
86              self.mse = mean_squared_error(self.test, self.predictions)
87              self.file.close()
88              self.rmse = sqrt(self.mse)
89
90          finally:
91              results.append((f'"{self.name}"', str(self.execution_time),
92                              str(self.mae), str(self.mse), str(self.rmse)))
93              print(f"Model {self.name} finished.")
```

Excerto 4.5: Classe do modelo *SARIMA*

Esta classe, para além dos parâmetros já definidos no modelo genérico, ainda conta com os parâmetros:

- *exog_variables* que representa as variáveis exógenas utilizadas para realizar as previsões;
- *season_parameters* que representa os parâmetros sazonais utilizados em modelos *SARIMA*.

Quanto aos métodos, apenas o método *"_execute(self)"* foi implementado para realizar as previsões de acordo com o algoritmo *SARIMAX*.

4.3.6 Conversor de *datasets*

Relativamente ao conversor de *datasets*, este permite fazer uma procura no diretório dos *datasets* e realiza a leitura do ficheiro, convertendo-o num objeto *"DataFrame"*. No excerto de código 4.6, pode ser analisado este componente.

```
def _dataset_to_series(filename: str, date_parser=None):
    """Searches FILES_FOLDER for a dataset and returns it into a DataFrame
    object. If it is needed to parse the dates, a function should be passed
    as the "date_parser" argument.

    Args:
        filename (str): name of the .csv file containing the dataset. This file
            should be in the FILES_FOLDER.
        date_parser (optional): function to parse the dates of the dataset if
            needed. The function should return a datetime.
    """
    FILES_FOLDER = "datasets"
    file_path = os.path.join(FILES_FOLDER, filename)
    series = DataFrame()
    try:
        series = read_csv(file_path, header=0, index_col=0, parse_dates=[0],
                          infer_datetime_format=True, date_parser=date_parser)
    except Exception as err:
        print(f"('{filename}') {type(err).__name__}: {err}")
    return series
```

Excerto 4.6: Componente de conversão de *datasets*

Depois de analisar este excerto de código, podemos concluir que este não é dos componentes mais complexos, porém, exige uma compreensão mínima da biblioteca (*pandas*), visto que caso seja necessário realizar alguma alteração nalgum parâmetro para converter o *dataset*, deve ser feito com cautela e alguma investigação do método *"read_csv()"*.

4.3.7 Exportador de resultados e registos

O excerto de código 4.7 é representativo do componente exportador de resultados e registos.

```
def _export_results(results_order: str = "name"):
    """Exports the results list into a .csv file ordered by the model order

    Args:
        results_order (str): order factor of the results list. ("name", "time",
            "mae", "mse" or "rmse"). Defaults to "name".
    """
    order = results_order.lower()
    if order == "time":
        order_num = 1
        order_name = "Execution Time (sec)"
    elif order == "mae":
        order_num = 2
        order_name = "MAE"
    elif order == "mse":
        order_num = 3
        order_name = "MSE"
    elif order == "rmse":
        order_num = 2
        order_name = "RMSE"
    else:
        order_num = 0
        order_name = "Model Name"

    results.sort(key=lambda line: float(line[order_num]))

    results_file = CSVWriter(
                        os.path.join(OUTPUT_FOLDER, "results_summary.csv"),
                        ("Model", "Execution Time (sec)", "MAE", "MSE", "RMSE")
                    )
    results_file.write_at_once(results)
    results_file.close()

    print(f"Results list file finished. Ordered by {order_name}.")


def _export_logs():
    """Exports the log list into a .txt file"""
    log_file = open(os.path.join(OUTPUT_FOLDER, "log.txt"), "w")
    for log in logs:
        log_file.write(log)
        log_file.write("\n")
    log_file.close()
    print("Log file finished.")
```

Excerto 4.7: Componente de exporte de resultados e registos

Neste componente, podemos visualizar a existência de duas funções. A primeira é destinada a exportar os resultados de todos os modelos corridos pela aplicação ordenados pelo nome, pelo *MAE*, *MSE* ou pelo *RMSE*. Todos os modelos que não tiveram sucesso na sua execução terão os seus erros todos com o valor de "-1", visto que é um valor impossível de obter se o modelo tiver terminado com sucesso.

Em relação à função de exportar os registos, trata-se de um subcomponente informativo, dado que não tem nenhuma utilidade prática para o sucesso ou insucesso dos modelos, mas apenas para informar que modelos tiveram sucesso e quais os que não tiveram e porquê.

4.3.8 Otimizador de modelos

O excerto de código 4.8 representa o componente realizado para correr os modelos.

```
def run_models(dataset_name: str, models: list, variable_to_predict: str,
                   title: str, results_order: str, num_splits: int = 0,
                   num_predictions: int = 10, predictions_size: float = 0,
                   date_parser=None):
    """Parses the dataset (.csv file) into a DataFrame object and runs ARIMA
    models with the given dataset.

    Args:
        dataset_name (str): name of the .csv file with the dataset.
        models (list): list of dictionaries with the ARIMA models to be tested.
        title (str): title to be used in the output files to distinguish the
            models.
        variable_to_predict (str): name of the variable to predict. It must be
            the same name of the column in the dataset.
        results_order (str): order factor of the results list. ("name", "time",
            "mae", "mse" or "rmse").
        num_splits (int): number of splits in case of being cross validation
            models. Defaults to 0.
        num_predictions (int): number of predictions of the model. Defaults
            to 10. It will only have effect if the predictions_size is equal
            to zero.
        predictions_size (float): percentage of data to predict (from 0 to 1).
            Defaults to 0.
        date_parser (optional): function to parse the dates of the dataset if
            needed. The function should return a datetime.
    """
    series = _dataset_to_series(dataset_name, date_parser)

    for model in models:
        for arima_parameters in model.get("arima_parameters"):
            if model.get("model") == ArimaImprovedModel:
                model.get("model")(series=series,
                                   variable_to_predict=variable_to_predict,
                                   arima_parameters=arima_parameters,
                                   num_predictions=num_predictions,
                                   predictions_size=predictions_size,
                                   title=title,
```

```
                                num_splits=num_splits)
            elif model.get("model") == ArimaMultivariateImprovedModel:
                exog_variables = model.get("exog_variables")
                model.get("model")(series=series,
                                variable_to_predict=variable_to_predict,
                                exog_variables=exog_variables,
                                arima_parameters=arima_parameters,
                                num_predictions=num_predictions,
                                predictions_size=predictions_size,
                                title=title,
                                num_splits=num_splits)
            elif model.get("model") == SarimaImprovedModel:
                for season_parameters in model.get("season_parameters"):
                    model.get("model")(series=series,
                                       variable_to_predict=variable_to_predict,
                                       arima_parameters=arima_parameters,
                                       season_parameters=season_parameters,
                                       num_predictions=num_predictions,
                                       predictions_size=predictions_size,
                                       title=title,
                                       num_splits=num_splits)
            elif model.get("model") == SarimaMultivariateImprovedModel:
                exog_variables = model.get("exog_variables")
                for season_parameters in model.get("season_parameters"):
                    model.get("model")(series=series,
                    variable_to_predict=variable_to_predict,
                                       exog_variables=exog_variables,
                                       arima_parameters=arima_parameters,
                                       season_parameters=season_parameters,
                                       num_predictions=num_predictions,
                                       predictions_size=predictions_size,
                                       title=title,
                                       num_splits=num_splits)
            else:
                logs.append(f"LOG: Model {model.get('model')} was not found!")
    _export_results(results_order)
    _export_logs()
```

Excerto 4.8: Componente para correr os modelos

Este é possivelmente o componente mais minucioso e onde mais erros podem acontecer, visto que se trata do componente *"core"*. Aqui são invocados o conversor de *datasets* e o exportador de resultados e registos. Esta função percorre a lista de modelos e recebe a série temporal, para puder tratar de fazer as previsões, guardando o resumo do desempenho de cada modelo num *array* e os registos de cada modelo noutro. À medida que cada modelo termina, os resultados e o seu registo são anexados às respetivas listas. Também se pode ir vendo os diretórios a serem criados à medida que os modelos terminam (desde que terminem com sucesso, caso contrário, como não há resultados, a pasta é apagada). No final, são criados o ficheiro de resumo dos resultados e o ficheiro de registos.

4.3.9 Exemplo de função *init()*

A função *init()* serve para testar implementações de código e fazer demonstrações do mesmo. No excerto 4.9 está representada esta mesma função, que demonstra a forma como devem ser instanciados e executados os modelos de aprendizagem.

```
    def init():
    dataset = "LinkNYC_kiosk.csv"

    def parser(x):
        return datetime.strptime(x, "%d/%m/%Y")

    variable_to_predict = "census"

    arima_parameters = [(1, 2, 0), (3, 2, 0), (2, 2, 0), (4, 2, 0)]

    sarima_parameters = [(1, 2, 0, 24), (1, 2, 0, 168)]

    exog_variables = ("temp", "heating degree", "cooling degree")

    models = [
        {
            "model": ArimaImprovedModel,
            "arima_parameters": arima_parameters
        },
        {
            "model": ArimaMultivariateImprovedModel,
            "arima_parameters": arima_parameters,
            "exog_variables": exog_variables
        },
        {
            "model": SarimaImprovedModel,
            "arima_parameters": arima_parameters,
            "season_parameters": sarima_parameters
        },
        {
            "model": SarimaMultivariateImprovedModel,
            "arima_parameters": arima_parameters,
            "exog_variables": exog_variables,
            "season_parameters": sarima_parameters
        }
    ]

    num_predictions = 20

    title = "Census"

    num_splits = 3

    results_order = "mse"

    run_models(dataset_name=dataset,
```

```
47                  variable_to_predict=variable_to_predict,
48                  date_parser=parser,
49                  models=models,
50                  num_predictions=num_predictions,
51                  title=title,
52                  num_splits=num_splits,
53                  results_order=results_order)
```

Excerto 4.9: Exemplo de função *init()*

4.4 *Google Colab*

O *Google Colaboratory*, mais conhecido por *Google Colab*, é um serviço gratuito hospedado pelo *Google Cloud* para incentivar a pesquisa sobre *Machine Learning Inteligência Artificial*. Similar ao *Jupyter Notebook*, esta plataforma é uma lista de células que podem conter textos ou códigos executáveis e os respetivos *outputs*.

Visto que o presente projeto é destinado ao estudo de algoritmos de aprendizagem, esta ferramenta é bastante adequada para treinar e recolher os resultados dos modelos em máquinas mais potentes e com mais recursos do que as nossas, de forma gratuita. Isto pode ajudar significativamente, reduzindo os tempos de execução dos modelos, ou seja, os resultados serão os mesmos, mas com um desempenho melhor. Na figura 4.4 está representado um exemplo de um projeto em ambiente de *Google Colab*.

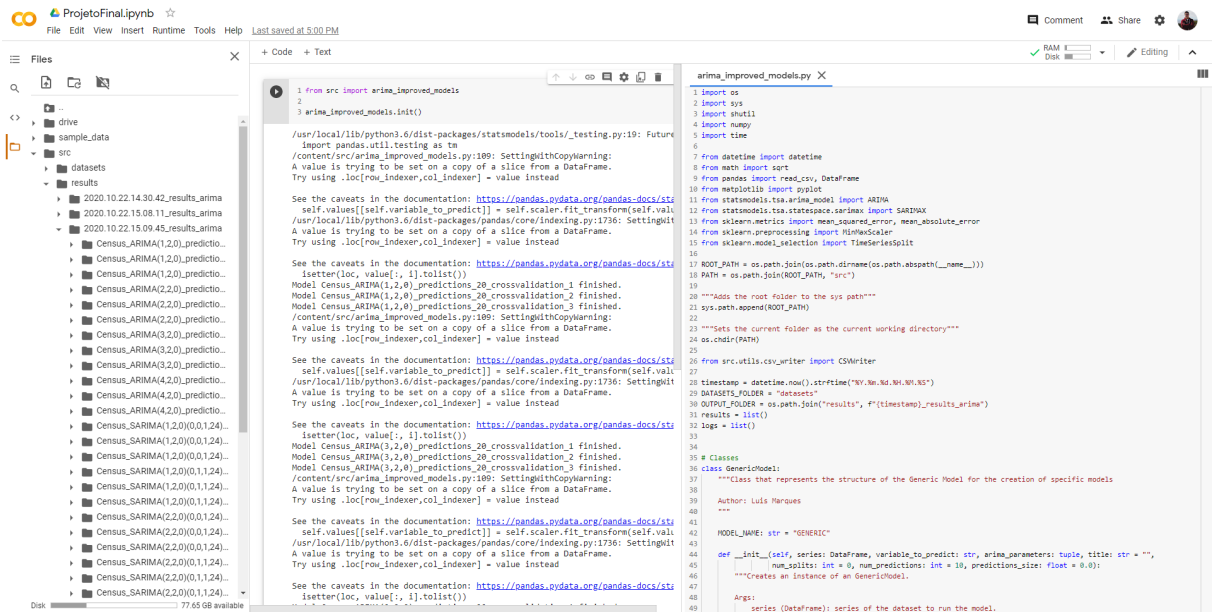


Figura 4.4: Exemplo de um projeto em ambiente de *Google Colab*

Capítulo 5

Discussão dos Resultados

5.1 Apresentação e Discussão dos Resultados

Foram utilizados dois datasets para testar os algoritmos desenvolvidos, de modo a obter resultados e contextos diferentes. O primeiro *dataset* aborda a diferença de velocidades instantâneas numa estrada, enquanto que o segundo estuda a quantidade populacional num determinado local.

5.1.1 Caso de Estudo das Velocidades Instantâneas

Este conjunto de dados é de uma rua na cidade de Braga e foi calculada a diferença de velocidades instantâneas no início e no fim da rua. Podem ser apontadas algumas vantagens no estudo deste *dataset* como: previsão das horas mais sossegadas numa estrada ou do tráfego diário num determinado local. Na tabela 5.1 está representado um excerto do conjunto de dados utilizado para realizar este caso de estudo.

"timestep"	"temperature"	"precipitation"	"speed_diff"
2019-04-01 07:00:00	12.0	0.0	11.666666666666668
2019-04-01 08:00:00	15.0	0.0	0.0
2019-04-01 09:00:00	16.0	0.0	11.0
2019-04-01 10:00:00	17.0	0.0	19.333333333333332
2019-04-01 11:00:00	18.0	0.0	21.333333333333332
2019-04-01 12:00:00	18.0	0.0	17.666666666666668
2019-04-01 13:00:00	16.0	0.0	25.0

Tabela 5.1: Excerto do *dataset* de velocidades instantâneas

Na figura 5.1 pode ser visualizado um diagrama de extremos e quartis do *dataset* de velocidades instantâneas relativo ao mês de abril de 2019.

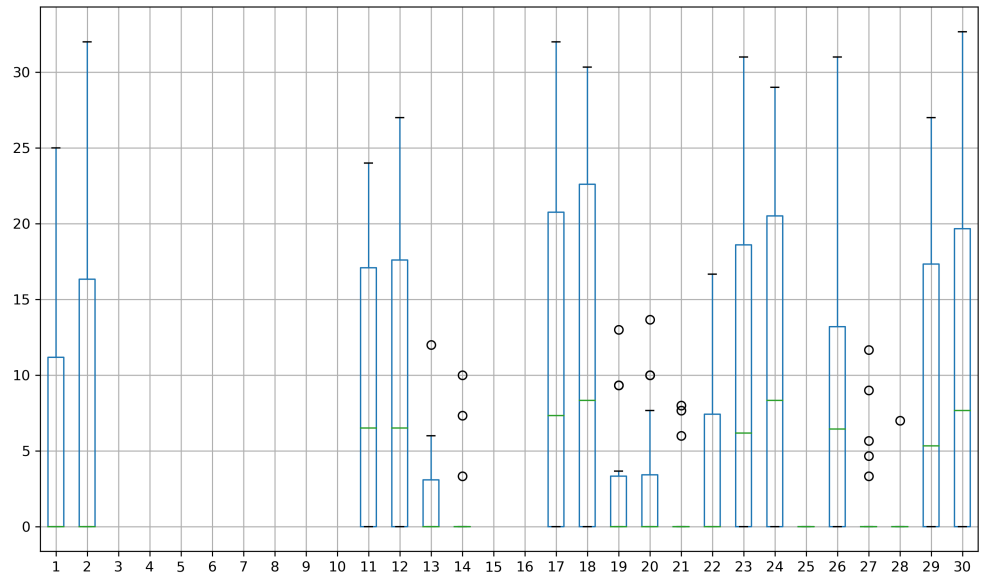


Figura 5.1: Diagrama de extremos e quartis do mês de abril de 2019 do *dataset* de velocidades instantâneas

Algumas perguntas que visam ser respondidas no final deste caso de estudo são:

- Qual a melhor hora para andar na estrada de carro?
- Qual a pior hora para andar na estrada de carro?

Para esta série temporal, foram realizadas as *grid searches* da seguinte forma:

i) Correr *ARIMA*(p, d, q) com:
   - p entre 1 e 5;
   - d entre 0 e 2;
   - q entre 0 e 2.

ii) Escolher 4 melhores configurações do *ARIMA*;

iii) Correr *SARIMA(p, d, q)(P, D, Q, S)* com as 4 melhores configurações dos primeiros parâmetros definidos no passo anterior;

iv) Escolher 2 melhores conjuntos dos parâmetros do *SARIMA*.

Escolher as variáveis exógenas dos modelos *ARIMAX* e *SARIMAX* é um pouco mais complexo e exige uma compreensão e estudo do *dataset* em si. Se não for escolhida nenuma variável o modelo não faz sentido ser executado. Por outro lado, caso seja escolhidas muitas variáveis o modelo ou demora uma eternidade a terminar ou não retorna resultados nenhuns.

Isto significa que, para escolher estas variáveis é necessário ir testanto as que dão melhores resultados e porquê e, finalmente, quando estiver a análise feita, juntá-las às configurações iniciais dos algoritmos.

Para terminar, tendo já sido escolhidas as melhores configurações dos dois tipos de parâmetros, correm-se os modelos todos com as configurações acima descritas. Deste modo, nos modelos *ARIMA* e *ARIMAX* existem 4 possibilidades de teste e nos modelos *SARIMA* e *SARIMAX* existem 8 possibilidades. Isto ajuda imenso na previsão de daodos, porque com a utilização das pesquisas em grelha, não é necessário correr todas as cofigurações para todos os modelos, ou seja, demora mesmo muito menos tempo.

"Model"	**"MAE"**	**"MSE"**	**"RMSE"**
"ARIMA(1,2,3)_15_2"	0.3863972317002109	0.17293853210778598	0.41585878866243287
"ARIMAX(1,2,3)_15_2"	0.41196228043522987	0.19971916779536353	0.44689950525298583
"ARIMA(4,2,3)_15_2"	0.67084667540561	0.5543517148947674	0.7445479936812451
"ARIMAX(1,2,3)_15_1"	3.367502360913987	23.984957736878993	4.897444000382137
"ARIMA(1,2,3)_15_1"	3.444645163427896	25.82468592020841	5.0817994765839005
"ARIMA(1,2,3)_15_3"	5.482114875372973	56.350156969636664	7.506674161680169

Tabela 5.2: Excerto sumário de resultados da *grid search* do primeiro caso de estudo

A tabela 5.2 representa o sumário de resultados da *grid search* do dataset das velocidades instantâneas.

Como pode ser constatado, os 3 primeiros modelos da tabela parecem ser os melhores, pelos erros mínimos que apresentam. No entanto, a 2ª divisão do *dataset* apanhou um conjunto de resultados todos a zero, ou seja, as previsões não foram complicadas de fazer, porém será impossível aproveitar os resultados do modelo para qualquer tipo de observação final.

Conclui-se assim que os 3 melhores modelos testados para este *dataset* foram:

ARIMAX(1, 2, 3), 1ª divisão do *dataset*, com 15 previsões. Representado na tabela 5.3 e na figura 5.2;

ARIMA(1, 2, 3), 1ª divisão do *dataset*, com 15 previsões. Representado na tabela 5.4 e na figura 5.3;

ARIMA(1, 2, 3), 3ª divisão do *dataset*, com 15 previsões. Representado na tabela 5.5 e na figura 5.4;

"Predict"	"speed_diff"
0.13817744331334328	0.0
-0.8973439961623095	7.66666666666 6 668
7.323374989923932	13.66666666666 6 664
11.038172786531765	0.0
2.103461054155741	3.0
2.091510219928224	3.0
3.208279408032984	3.33333333333 3 333
2.283585816926894	3.666666666666 6 665
3.9253583100976295	1 0.0
8.34553662378711	0.0
1.5898776295568244	4.0
2.7362308186284343	0.0
0.3293245341001239	0.0
-0.8090669648652083	0.0
0.4107963416104033	0.0

Tabela 5.3: Previsões do modelo *ARIMAX*(1, 2, 3), 1ª divisão do dataset, com 15 previsões

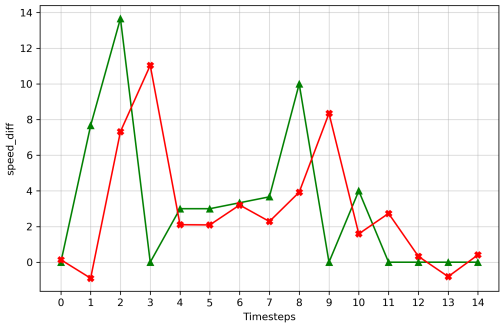


Figura 5.2: Gráfico do modelo *ARIMAX*(1, 2, 3), 1ª divisão do dataset, com 15 previsões

"Predict"	"speed_diff"
0.34030661913252236	0.0
-0.7087282570616925	7.666666666666668
6.955187335970128	13.666666666666664
12.210641102705337	0.0
2.2518274963409595	3.0
2.236290006499467	3.0
3.3883287072845674	3.333333333333333
2.597318025226484	3.6666666666666665
4.030597662527793	10.0
8.426346911548624	0.0
1.8303097185121766	4.0
2.7647857128428743	0.0
1.016524698044413	0.0
-0.7501655819003638	0.0
0.29871343930839866	0.0

Tabela 5.4: Previsões do modelo *ARIMA*(1, 2, 3), 1ª divisão do dataset, com 15 previsões

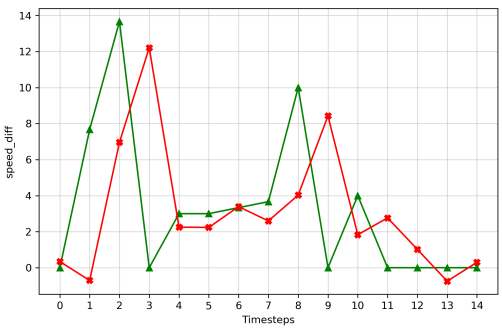


Figura 5.3: Gráfico do modelo *ARIMA*(1, 2, 3), 1ª divisão do dataset, com 15 previsões

"Predict"	"speed_diff"
17.942670071265077	18.33333333333333
17.893110238007367	12.0
13.32185938423577	22.66666666666667
20.93896774528977	18.666666666666668
19.682965507712986	23.333333333333336
22.402823802743256	16.333333333333332
18.20671584886394	25.66666666666667
24.068776185547435	32.0
31.389219555359038	32.66666666666667
32.091001235619046	21.66666666666667
23.538510942114325	3.333333333333333
5.407697532959148	0.0
0.9618211695579877	0.0
-0.5039355197295305	0.0
0.4393948308917405	0.0

Tabela 5.5: Previsões do modelo *ARIMA*(1, 2, 3), 3ª divisão do dataset, com 15 previsões

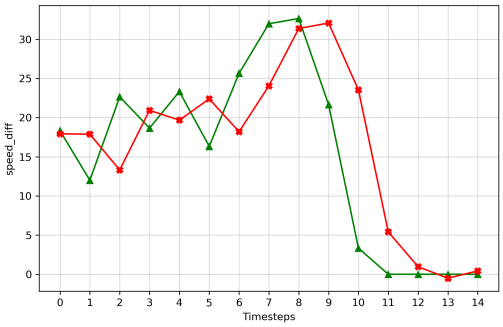


Figura 5.4: Gráfico do modelo *ARIMA*(1, 2, 3), 3ª divisão do dataset, com 15 previsões

Em suma, o modelo mais acertivo foi o modelo representado na tabela 5.1 e na figura 5.1. Posto isto, podemos concluir que a melhor hora para passar na estrada será a hora com maior diferença de velocidades instantâneas, ou seja, a hora correspondente à previsão número 1. A pior será a hora correspondente à previsão número 3, 11, 12, 13 e 14. Analisando o *dataset*, podemos afirmar que a melhor hora será às 11 horas e a pior será às 13 horas e das 20 horas até às 23 horas. No entanto, nas horas noturnas é normal não haver tanto tráfego automóvel, isto é, conclui-se que a pior hora será às 13 horas.

5.1.2 Caso de Estudo da Quantidade Populacional

Por outro lado, este conjunto te dados trata-se das medições da quantidade populacional num quiosque em Nova Iorque. Podem ser apontadas vantagens como: previsão da quantidade de pessoas num determinado local, o que pode ser uma excelente forma de prevenção da doença COVID-19. Na tabela 5.6 está representado um excerto do conjunto de dados utilizado para realizar este caso de estudo.

"date"	"wifi status"	"temp"	"precipitation"	"census"
01/01/2016	up	-6	0	4092
02/01/2016	up	2	0	3807
03/01/2016	up	2	0	3208
04/01/2016	up	-4	0	1057
05/01/2016	up	2	0	3302
06/01/2016	up	2	0	3761
07/01/2016	up	4	0	2964

Tabela 5.6: Excerto do *dataset* de quantidade populacional

Na figura 5.5 pode ser visualizado um diagrama de extremos e quartis do *dataset* de quantidade populacional relativo ao ano de 2018.

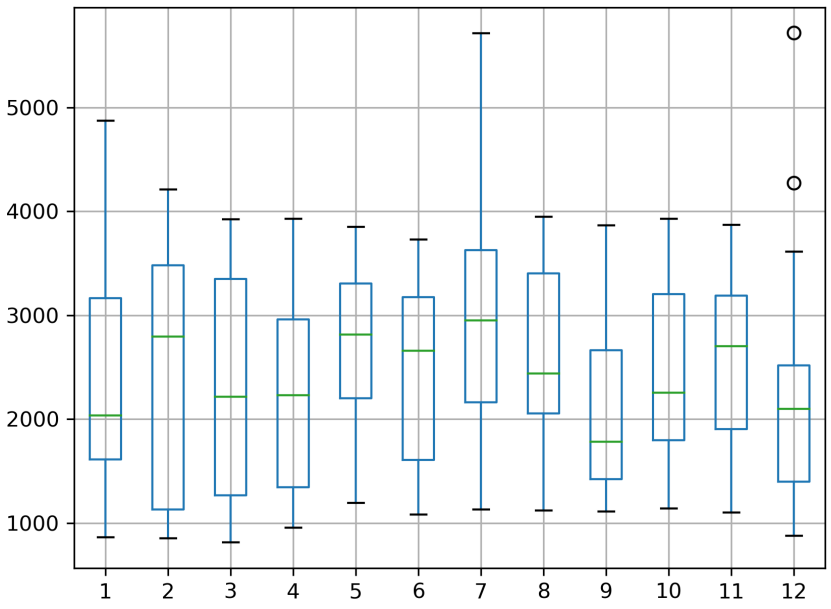


Figura 5.5: Diagrama de extremos e quartis do ano de 2018 do *dataset* de quantidade populacional

Algumas perguntas que visam ser respondidas no final deste caso de estudo são:

- Qual o melhor dia do mês para ir ao quiosque?
- Qual o pior dia do mês para ir ao quiosque?

Para esta série temporal, foram realizadas as *grid searches* da seguinte forma:

i) Correr *ARIMA*(p, d, q) com:
   - p entre 1 e 5;
   - d entre 0 e 2;
   - q entre 0 e 2.

ii) Escolher 4 melhores configurações do *ARIMA*;

iii) Correr *SARIMA(p, d, q)(P, D, Q, S)* com as 4 melhores configurações dos primeiros parâmetros definidos no passo anterior;

iv) Escolher 2 melhores conjuntos dos parâmetros do *SARIMA*.

Escolher as variáveis exógenas dos modelos *ARIMAX* e *SARIMAX* é um pouco mais complexo e exige uma compreensão e estudo do *dataset* em si. Se não for escolhida nenuma variável o modelo não faz sentido ser executado. Por outro lado, caso seja escolhidas muitas variáveis o modelo ou demora uma eternidade a terminar ou não retorna resultados nenhuns. Isto significa que, para escolher estas variáveis é necessário ir testanto as que dão melhores resultados e porquê e, finalmente, quando estiver a análise feita, juntá-las às configurações iniciais dos algoritmos.

Para terminar, tendo já sido escolhidas as melhores configurações dos dois tipos de parâmetros, correm-se os modelos todos com as configurações acima descritas. Deste modo, nos modelos *ARIMA* e *ARIMAX* existem 4 possibilidades de teste e nos modelos *SARIMA* e *SARIMAX* existem 8 possibilidades. Isto ajuda imenso na previsão de daodos, porque com a utilização das pesquisas em grelha, não é necessário correr todas as cofigurações para todos os modelos, ou seja, demora mesmo muito menos tempo.

"Model"	"MAE"	"MSE"	"RMSE"
"ARIMA(1,2,0)_20"	431.4058281994794	294362.5316925172	542.5518700479404
"ARIMA(3,2,0)_20"	436.2527892941756	327546.80919104733	572.3170530318378
"ARIMA(2,2,0)_20"	470.73536627219164	339518.96987558715	582.6825635589134
"ARIMA(4,2,0)_20"	463.6572887631079	397280.74748587905	630.3021081083888
"ARIMA(5,2,0)_20"	481.2747052602357	463843.7919440161	681.0607843239957
"ARIMA(1,1,0)_20"	450.17960473297154	464377.9889890727	681.4528516259013

Tabela 5.7: Excerto sumário de resultados da *grid search* do segundo caso de estudo

A tabela 5.7 representa o sumário de resultados da *grid search* do dataset da quantidade populacional.

Conclui-se assim que os 3 melhores modelos testados para este *dataset* foram:

ARIMA(1, 2, 0), sem validação cruzada, com 20 previsões. Representado na tabela 5.8 e na figura 5.6;

ARIMA(3, 2, 0), sem validação cruzada, com 20 previsões. Representado na tabela 5.9 e na figura 5.7;

ARIMA(2, 2, 0), sem validação cruzada, com 20 previsões. Representado na tabela 5.10 e na figura 5.8;

"Predict"	"census"
2071.4553305007325	1000.0
991.9236886020411	877.0000000000001
322.59086790167925	1126.0
1150.7135663993647	958.0
1042.8648699235348	1128.0
1094.1752495844441	1509.0
1763.3997606287378	2474.0
3087.421245832234	1928.9999999999998
2298.294797507882	2560.0
2480.2184224343887	2126.0000000000005
2337.123922709361	2398.0
2243.3011173547725	1834.0000000000002
1776.2634968906202	2094.0
1855.7064948556406	2116.0000000000005
2282.8257530269984	2089.0
2092.242524019219	2246.0000000000005
2292.346701629985	2384.0
2534.3085052761926	3011.0
3343.248508896108	4274.0
5154.176860337541	5720.0

Tabela 5.8: Previsões do modelo *ARIMA*(1, 2, 0), com 20 previsões

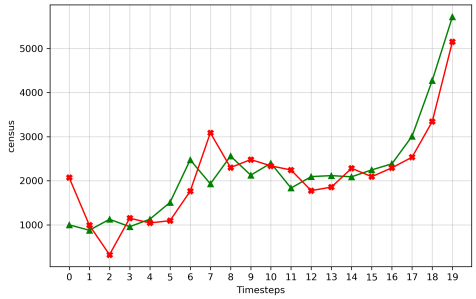


Figura 5.6: Gráfico do modelo *ARIMA*(1, 2, 0), com 20 previsões

"Predict"	"census"
2071.4553305007325	1000.0
1486.3564093774662	1000.0
1259.105158633161	877.0000000000001
936.7208987331807	1126.0
891.2391031088812	958.0
664.04646073088	1128.0
1138.8505813208287	1509.0
1536.3068503783759	2474.0
2497.963195964612	1928.9999999999998
2467.5200712884825	2560.0
2953.008515321484	2126.0000000000005
2445.992450723264	2398.0
2346.298572292104	1834.0000000000002
1998.3895774073735	2094.0
1887.2209749046133	2116.0000000000005
2026.4401798934175	2089.0
2003.6514414933647	2246.0000000000005
2335.346340795519	2384.0
2409.007736368172	3011.0
3051.686418045654	4274.0
4442.241881959838	5720.0

Tabela 5.9: Previsões do modelo *ARIMA*(3, 2, 0), com 20 previsões

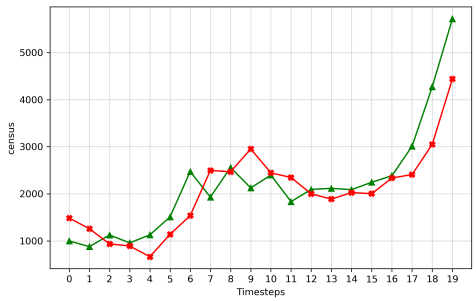


Figura 5.7: Gráfico do modelo *ARIMA*(3, 2, 0), com 20 previsões

"Predict"	"census"
1938.3248213044278	1000.0
1160.307636030289	877.0000000000001
787.2489987658323	1126.0
686.5435225844557	958.0
984.9846735946015	1128.0
1200.4527870105767	1509.0
1533.760588643199	2474.0
2808.98688952599	1928.9999999999998
2463.9565873495894	2560.0
2876.6031847102568	2126.0000000000005
2075.575964237916	2398.0
2559.965779302652	1834.0000000000002
1678.7683021383232	2094.0
2023.2650400026798	2116.0000000000005
1947.4008439849385	2089.0
2226.007489000049	2246.0000000000005
2262.5125641983063	2384.0
2450.1225894495315	3011.0
3207.1297244942393	4274.0
4722.751309975546	5720.0

Tabela 5.10: Previsões do modelo *ARIMA*(2, 2, 0), com 20 previsões

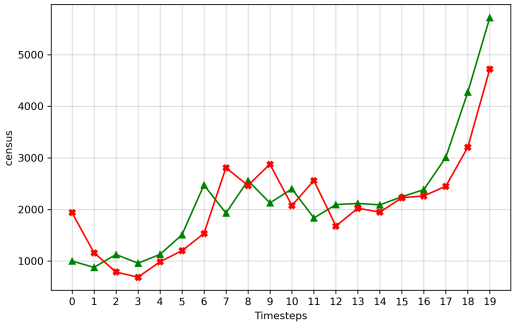


Figura 5.8: Gráfico do modelo *ARIMA*(2, 2, 0), com 20 previsões

Em suma, o modelo mais acertivo foi o modelo representado na tabela 5.4 e na figura 5.4. Posto isto, podemos concluir que o melhor dia do mês para ir ao quiosque será o dia com menor população e o pior dia será o dia com maior população. Ou seja, segundo o modelo da figura 5.4, o melhor dia é o da previsão 1 e o pior é o da previsão 20. Fazendo a consulta ao *dataset*, podemos concluir que o melhor dia para ir seria no dia 13 e o pior seria no dia 31.

5.2 Apresentação dos Impedimentos e/ou Constrangimentos

Em primeiro, é importante realçar que, na opinião do autor, se um projeto chega ao fim com sucesso a avaliação geral deve ser positiva. No entanto, podem exister precalços no decorrer do desenvolvimento do mesmo. Dito isto, os principais pontos em que o autor sentiu mais dificuldade foram no desenvolvimento do presente documento, bem como no estudo dos modelos *ARIMA*, *ARIMAX*, *SARIMA* e *SARIMAX*.

Por um lado, o desenvolvimento documento principal foi realizado com a linguagem *LaTeX*, que foi um dos desafios propostos pelo autor ao professor orientador, no entanto, como se trata da aprendizagem de algo novo, existem sempre mais complicações. Para além disto, o autor também teve enormes dificuldades em encarar os objetivos do projeto do ponto de vista teórico.

Por outro lado, os algoritmos estudados também provocaram uma certa dificuldade no desenvolvimento do trabalho. No entanto, como existe uma certa similaridade entre os modelos, a aprendizagem do primeiro facilitou bastante o entendimento dos restantes.

No geral, o trabalho foi positivo, apesar de todos os constrangimentos, visto que foi concluído com sucesso.

Capítulo 6

Conclusão

6.1 Reflexão Crítica dos Resultados

A solução desenvolvida é adequada às necessidades requeridas pelo projeto. Em relação aos resultados, existem sempre bastantes problemas com alguns modelos, visto que falham imenso por falta de convergência de alguns pontos. Na duração do projeto foi utilizada uma biblioteca, que no momento se encontra descontinuada (*"statsmodels.tsa.arima_model.ARIMA"*), todavia já não era possível fazer a migração para a nova versão pela incompatibilidade com o *Google Colab*. Foi exprimentada a nova biblioteca localmente e alguns dos modelos que anteriormente não obtinham resultados por falta de convergência de pontos, nesta biblioteca já conseguiam ter sucesso nas previsões.

Abordando os modelos de séries temporais estudadas, existem bastante utilidade nas possíveis previsões a realizar com os mesmos, visto que, é possível prever dados importantíssimos que normalmente utilizamos (ou seria vantajoso se o fizéssemos) como a velocidade média numa certa estrada ou o tráfego de uma estrada a certas horas do dia.

Debatendo sobre o segundo *dataset* estudado, as previsões a realizar nesta altura que estamos a viver com a COVID-19, seria bastante interessante ter estimativas da ocupação de certos espaços com base em dados anteriores e poderia oferecer uma ajuda enorme no assunto.

6.2 Conclusão e Trabalho Futuro

No futuro, uma implementação essêncial seria a utilização da biblioteca na sua versão mais recente, para que mais modelos possam terminar com sucesso e, desse modo, conseguir melhores previsões e mais precisas.

Em relação à incompatibilidade do *Google Colab* com esta biblioteca poderia ser contorado com o desenvolvimento de uma *API* e implementação do ambiente virtual de *Python* num servidor, com a versão da linguagem utilizada neste projeto e as dependências necessárias para a nova versão da biblioteca. Ao servir essa *API* com esse servidor, teriamos uma plataforma de testes completamente funcional e com as bibliotecas necessárias mais atuais para um maior sucesso na execução dos algoritmos.

Em suma, o trabalho foi concluído com sucesso e, na opinião do autor, as ferramentas trabalhadas podem vir a ser bastante úteis futuramente. O autor também espera concluir o trabalho futuro discutido para que o projeto fique cada vez mais único e útil.

[Página propositadamente deixada em branco.]

Referências

[1] Wikipedia, “Time series.” [Online; acedido a 2-outubro-2020].

[2] J. S. Saltz and J. M. Stanton, *An Introduction to Data Science*. Singapore, 2017.

[3] J. Brownlee, “Time series data visualization with python.” [Online; acedido a 1-outubro-2020].

[4] toppr, “Components of time series.” [Online; acedido a 15-outubro-2020].

[5] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[6] J. Brownlee, “What is time series forecasting?.” [Online; acedido a 23-setembro-2020].

[7] E. Alpaydin, *Introduction to Machine Learning*. Massachusetts Institute of Technology, 2004.

[8] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2006.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Massachusetts Institute of Technology, 2016.

[10] Elsevier, “A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in western australia.” [Online; acedido a 10-outubro-2020].

[11] J. Brownlee, “How to create an arima model for time series forecasting in python.” [Online; acedido a 23-setembro-2020].

[12] Wikipedia, “Arima.” [Online; acedido a 15-outubro-2020].

[13] W. H. Greene, *Econometric Analysis*. Prentice Hall, 1997.

[14] ElegantJBI, “What is arimax forecasting and how is it used for enterprise analysis?.” [Online; acedido a 15-outubro-2020].

[15] J. Brownlee, “A gentle introduction to sarima for time series forecasting in python.” [Online; acedido a 13-outubro-2020].

[16] S. Exchange, “Sarima model equation.” [Online; acedido a 13-outubro-2020].

[17] S. Prabhakaran, “Arima model – complete guide to time series forecasting in python.” [Online; acedido a 15-outubro-2020].

[18] E. Lutins, “Grid searching in machine learning: Quick explanation and python implementation.” [Online; acedido a 12-outubro-2020].

[19] T. Escovedo and A. Koshiyama, *Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise*. Casa do Código, 2020.

[20] J. R. Hauser, *Numerical Methods for Nonlinear Engineering Models*. Springer, 2009.

[21] W. P. Rogers, “Encapsulation is not information hiding.” [Online; acedido a 13-outubro-2020].

[22] P. 2002, “24.2 data abstraction with existentials.” [Online; acedido a 12-outubro-2020].