Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

Ciência de dados aplicada a controle de doses de radiação em teste de controle de qualidade em aceleradores lineares

Acadêmico: Paulo Augusto Vanderlez Soares

Orientador: Prof. Dr. Anuar José Mincache

Coorientador: Prof. Dr. Gustavo Sanguino Dias

Maringá, 21 de maio de 2021

Universidade Estadual de Maringá
Centro de Ciências Exatas
Departamento de Física

Trabalho de Conclusão de Curso

Ciência de dados aplicada a controle de doses de radiação em teste de controle de qualidade em aceleradores lineares

Trabalho de conclusão de curso apresentada ao Departamento de Física da Universidade Estadual de Maringá, sob orientação do professor Dr. Anuar José Mincache, como parte dos requisitos para obtenção do título de Bacharel em Física

Acadêmico: Paulo Augusto Vanderlez Soares

Orientador: Prof. Dr. Anuar José Mincache

Coorientador: Prof. Dr. Gustavo Sanguino Dias

Maringá, 21 de maio de 2021

Sumário

Lista de Figuras

Agradecimentos

Gostaria de agradecer ao professor Anuar, por topar esse projeto, onde tudo que foi estudado e apresentado está fora do escopo das matérias estudadas na graduação em Física. Sempre com muita atenção, compreensão e paciência. Incentivando e acreditando a todo momento. Muito obrigado.

Ao departamento de física. Muito obrigado por fornecer um ensino excelente com professores extremamente capacitados, dedicados e sempre a disposição dos alunos.

Aos meus pais, obrigado por todo o amor, paciência e confiança concedidos a mim. Sempre servindo de combustível para enfrentar quaisquer dificuldade que apareça no caminho.

A minha mãe, Rosimar, gostaria de dizer o quanto a amo, agradecer por estar sempre ao meu lado em qualquer situação. Pessoa maravilhosa, gostaria de descreve-la em palavras, porém, como H.P. Lovecraft, teria que inventar infinitos adjetivos para explicar algo extraordinário.

Ao meu pai, Paulo, melhor amigo que terei em minha vida. Obrigado por tudo. É difícil saber que não irei vê-lo novamente, porém você sempre continuará vivo, em minha mente, em meu coração e em meu caráter.

A minha namorada Deyse. Entrou em minha vida quando mais precisei. Deu a perspectiva de futuro que antes não existia. Espero conseguir retribuir tudo o que tem feito a mim. Te amo.

Difícil falar de todas pessoas importantes em um texto breve, gostaria de agradecer a minha família, amigos, professores, entre todos que participaram dessa jornada.

Gostaria de agradecer ao Alisson, físico do Hospital do Câncer de Maringá, por fornecer os dados necessários para este projeto. Muito Obrigado.

Resumo

Este projeto teve como objetivo utilizar-se de uma ferramenta conhecida como ciência de dados. Esta área que faz intersecção com a ciência da computação, matemática e estatística traz uma revolução na forma de compreender e obter *insight* dos dados a serem estudados. Unindo a ciência de dados com todo esse suporte tecnológico em essência aberto, isto é, tudo de graça, é possível ir além dos *softwares* pagos, que de certa forma são rígidos na maioria das vezes dependendo do seu tipo de análise. Logo este estudo foi realizado utilizando dados fornecidos pelo Hospital do Câncer de Maringá, sendo possível fazer uma análise robusta e eficaz das doses de radiação para os fins de controle de qualidade do sistema, imposta por normas regulatórias pela comissão nacional de energia nuclear (CNEN). Mostrando que a partir dos algoritmos da linguagem de programação em *Python*, foi possível analisar padrões e fazer previsões com a ferramenta denominada aprendizado de máquina, subárea da inteligência artificial. Com esse novo conceito é possível prever níveis de radiação ionizante que estão fora das normas de segurança do equipamento.

Palavras chave: Radiação, Ciência de dados, Aprendizado de máquina.

Abstract

This project aimed to use a tool known as data science. This area crosses knowledges of computer science, math and statistics, bringing a revolution in the way of comprehending and getting insights about data yet to be studied. Combining the data science with all this technological support, which is, in your essence, open and free, it is possible to go beyond paid softwares that are, most of the time, stern in some ways, depending on its analysis. Therefore, this study was accomplished with data provided by the Cancer Hospital of Maringá, making it possible to execute a whole and effective analysis of the radiation doses for the purpose of controlling the system quality, enforced by regulatory standards by the National Nuclear Energy Commission (CNEN - portuguese acronym). Showing that from the Python programming language algorithm, it was possible to analyze patterns and predict with the tool called machine learning, a subarea of artificial intelligence. With this new concept it is possible to predict levels of ionizing radiation which are out of the equipment safety standards.

Keywords: Radiation, Data science, Machine learning.

Introdução

Após a descoberta dos raios X por Röntgen em 1895, houve uma grande popularização do uso das radiações ionizantes, tanto da população em geral quanto o mundo científico. Com o passar do tempo, descobriu-se uma grande quantidade de aplicações das radiações ionizantes. Juntamente com a gama de possibilidades veio o conhecimento da nocividade causada pela radiação a matéria biológica, de forma que normas e regras foram criadas para que as radiações ionizantes possam continuar trazendo seus benefícios com o mínimo de risco aos seres humanos e a natureza como um todo.

Hoje com avanço do poder de processamento computacional e grandes capacidades de bancos de dados, as áreas de ciência de dados e aprendizado de máquina tem crescido e amadurecido muito nos últimos anos. Ferramentas que podem ser aplicadas nos estudos de radiações ionizantes, a fim de aumentar o conhecimento da área e aprimorar ainda mais a utilização de radiação para benefícios da humanidade.

A ciência de dados é uma subárea da inteligência artificial, que utiliza de conhecimentos interdisciplinares como estatística, matemática e computação, para fazer a análise de dados. Com o foco em extrair informações ou comportamentos de alguns conjuntos de dados.

Já o aprendizado de máquina (*Machine learning*), também uma subárea da inteligência artificial, que utiliza dados com a intenção de criar modelos que compreendam, expliquem comportamentos e faça previsões, dos dados utilizados, que não seriam triviais para a análise humana.

Neste trabalho são apresentadas algumas ferramentas computacionais para trabalhar com dados. Ferramentas que fazem manipulação, processamento, visualização e análise dos dados. Os dados utilizados foram fornecidos pelo Hospital do Câncer de Maringá. Neles contêm informações sobre os testes de controle feito no acelerador linear utilizado para tratamento de radioterapia, esses testes são feitos diariamente a fim de saber se o equipamento está operando nas margens de segurança estabelecida.

O objetivo é utilizar ferramentas de código aberto e gratuitas, de ciências de dados e apren-

dizado de máquina, em linguagem *Python* para fazer a análise e criar um modelo de predição, utilizando dados dá área de física das radiações.

Capítulo 1

Radiação

Radiação é energia em movimento, transmitida de uma fonte fluindo por meios como vácuo, ar e meios materiais, na forma de partícula ou onda eletromagnética. Ela pode ser produzida de maneira natural com a instabilidade no núcleos dos átomos e de modo artificial em tubos de raios catódicos e aceleradores de partículas.

A história das radiações começou em 1895 com Wilhelm Conrad Röntgen estudando descargas elétricas em um tubo de raios catódicos. Foi utilizado um aparato semelhante ao usado por Philipp Lenard, que consiste em um tubo de Crookes modificado com uma janela de alumínio permitindo que os raios catódico continuassem seu trajeto para o exterior do equipamento, semelhante a Figura 1.1.

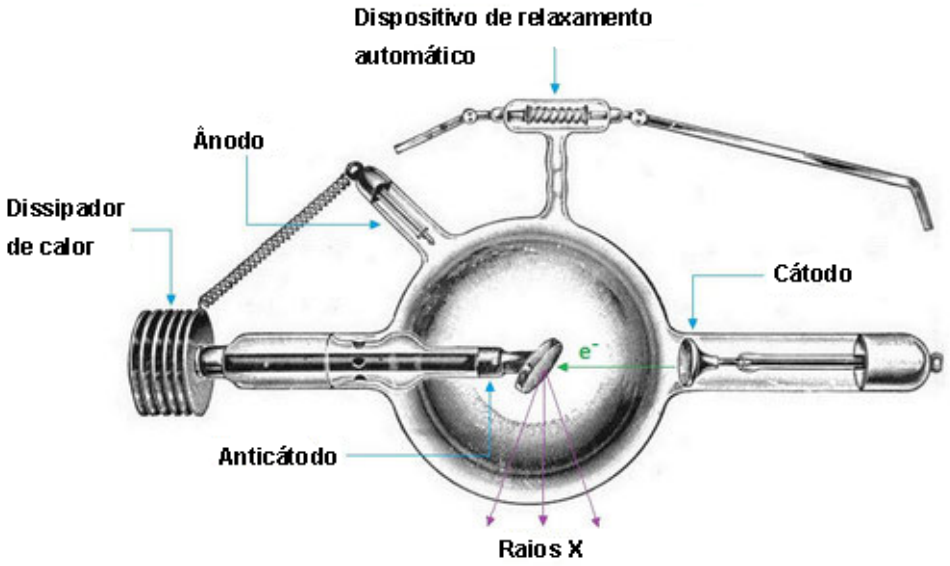


Figura 1.1: Exemplo de tubo de raios catódicos.

Fonte: https://teslaresearch.jimdofree.com/x-rays

Röntgen cobriu o tubo com um material escuro, para que a luz produzida no tubo não atrapalhasse sua visão durante a análise da luminescência produzida pelos raios catódicos, na placa de platino cianeto de bário posicionado 8 cm do tubo. Após dar início ao experimento notou-se uma fraca luminescência na placa, para uma melhor visualização ele apagou a luz e repetiu o experimento. Röntgen começou a fazer testes, ao afastar a placa verificou que a luminescência ainda continuava, colocou vários objetos entre o tubo e a placa, porém, de alguma forma os raios continuavam interagindo com a placa. Assim Röntgen percebeu que esse feito não foi produzido pelos raios catódicos e sim por outro tipo de raio com um grande poder de penetração e os nomeou de raios X.

Um segundo evento muito importante na história da radiação aconteceu em 1896, quando Antoine Henri Becquerel estava estudando os efeitos que sais de urânio realizava sobre filme fotográfico. Becquerel observou que as manchas escuras no papel fotográfico era devido a raios emanados de maneira espontânea pelo sal de urânio, e não pelo fenômeno de fluorescência que era aceita até o momento, ele continuou estudando as emissões do urânio e observou que esses raios tinham propriedades parecidas com os raios X, e que as emissões não diminuíam com o tempo. Em 1897, Marie Curie, procurou Becquerel para iniciar sua tese de doutorado que consistia em estudar a natureza dos raios de Becquerel. Porém, ela mudou a linha de pesquisa e passou a procurar outros elementos com propriedades parecidas as do urânio. Em 1898, juntamente com Pierre Curie, reportaram a descoberta de um novo elemento que foi batizado de polônio, no mesmo ano descobriram outro elemento conhecido como rádio, as propriedades desses elementos que são semelhantes ao urânio ficou conhecido como radioatividade [1] [2] [3].

1.1 Raios X

Quando elétrons são acelerados em tubos de raios X, grande parte deles vão perdendo energia cinética gradualmente durante as colisões no alvo sendo convertido em calor, por esse motivo os alvos desses tubos são feitos de materiais com alto ponto de fusão e ainda é necessário resfriamento. Os elétrons restantes de uma maneira probabilística produzem raios X de duas formas, via radiação de freamento ou raios X característicos. Esse tipo de radiação ionizante não está relacionado a instabilidade nuclear dos átomos de forma que para sua produção não há necessidade de utilizar elementos radioativos [1] [2].

- **Radiação de freamento (*Bremsstrahlung*)**

Dos elétrons acelerados até o alvo pelo tubo de raios X, uma pequena parte se aproxima dos núcleos atômicos que compõe o alvo, ao passar próximo do núcleo o elétron pode ser desacelerado bruscamente devido a atração do campo coulombiano, fazendo com que um fóton seja emitido com energia igual a energia cinética perdida pelo elétron, como pode ser visto na Figura 1.2. Esses raios X possuem espectro contínuo, podendo assumir qualquer valor de energia, dependendo apenas da diferença de potencial que acelera os elétrons e da distância que os mesmos passam pelas vizinhanças dos núcleos atômicos do alvo.

Esses raios X são chamados de radiação de freamento que vem da palavra alemã *Bremsstrahlung* [1] [4].

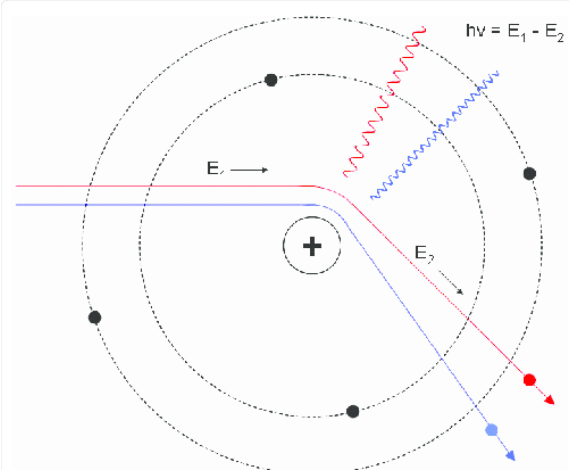


Figura 1.2: Elétrons acelerados na direção de um núcleo atômico.

Fonte: https://www.researchgate.net/figure/Bremsstrahlung-is-generated-when-fast-moving-electrons-blue-and-red-lines-closely_fig5_235960674

- **Raios X característicos**

Os raios X característicos são produzidos quando um elétron acelerado colide com algum elétron das camadas mais internas dos átomos do alvo, fazendo com que esse elétron seja removido. Quando esse evento ocorre os elétrons das camadas mais externas (mais energéticas) realizam um salto quântico para ocupar a posição deixada pelo elétron ejetado, nesse processo o elétron que realiza o salto quântico libera energia na forma de fótons no espectro de energia do raio X, como mostrado na Figura 1.3.

Diferentemente do raio X de freamento os raios X característicos possuem espectro de energia discreto que está relacionado intimamente ao material utilizado como alvo, mostrando uma assinatura particular do mesmo, já que o espectro de energia do raio X característico está ligado a energia das camadas eletrônicas dos átomos que constituem o alvo [1] [4].

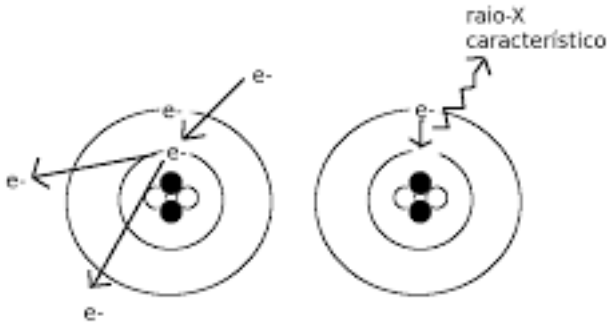


Figura 1.3: Emissão de um raio X característico via transição de um elétron de uma camada mais energética para uma menos energética.

Fonte: https://portal.ifi.unicamp.br/images/files/extensao/oficinas-fisica/xxvii/Oficina_Raios-X_agosto2011_Mario.pdf

1.2 Radiações nucleares

Núcleos atômicos são compostos por prótons e nêutrons, onde os prótons possuem carga positiva e os nêutrons carga nula. Para que um núcleo seja estável a força nuclear que é atrativa tem que ser superior a força coulombiana de repulsão entre os prótons. A função do nêutron neste sistema de forças é deixar o núcleo coeso, já que ele experimenta apenas da força nuclear atrativa. Quando o núcleo está instável ou com excesso de prótons ou nêutrons ele começa a se desintegrar até alcançar a estabilidade, quando esse fenômeno ocorre o núcleo começa a emitir espontaneamente partículas ou energia na forma de radiação eletromagnética, essas emissões são formadas pelas radiações alfa, beta e gama [1] [2] [4].

1.2.1 Radiação Alfa α

A radiação α é emitida principalmente por núcleos atômicos pesados, com número atômico maior ou igual a 83, é uma radiação corpuscular, ou seja, uma partícula acelerada que possui

dois prótons e dois nêutrons similar a um núcleo de hélio. Ela possui um baixo poder de penetração devido a sua massa e grande capacidade de ionização. Um exemplo de decaimento α é do rádio decaindo para o radônio:

$$^{226}_{88}\mathrm{Ra} \rightarrow {}^{222}_{86}\mathrm{Rn} + {}^{4}_{2}\mathrm{He}$$

após o núcleo sofrer a transmutação de rádio para o radônio, a partícula α é ejetada e vai sofrendo ionizações e perdendo energia, nessas ionizações ela captura dois elétrons tornando-se um átomo estável de hélio.

A nível biológico a radiação alfa em contato com a pele não causa danos significativos, devido a sua massa. Entretanto, se houver a ingestão, o organismo pode vir a ter inúmeras complicações [1] [3] [4].

1.2.2 Radiação Beta β

O decaimento β ocorre quando o núcleo atômico instável emite espontaneamente um elétron ou um pósitron, respectivamente chamado de β^- e β^+. O pósitron é a antipartícula do elétron que possui as mesmas características, porém, com carga positiva, neste caso a radiação emitida também é corpuscular.

A emissão de radiação β^- foi detectada experimentalmente, seus resultados revelaram duas características que trouxeram dificuldades para compreender essa radiação na época. A primeira característica é que a radiação β^- possui espectro contínuo de energia, diferentemente da radiação α que é monoenergética. Esse problema foi solucionado propondo a existência de uma nova partícula, o neutrino, uma partícula sem carga e com massa de repouso muito pequena (ν_e). Quando ocorre a desintegração nuclear a energia cinética é separada de forma probabilística entre o neutrino e a partícula β^-, explicando o espectro contínuo de energia.

A segunda característica era que o núcleo atômico composto por prótons e nêutrons estava emitindo elétrons. Este problema foi solucionado por Enrico Fermi, supondo que no núcleo nêutrons estavam sendo convertidos em prótons e vice-versa resultando na emissão de partículas β^- e β^+. [1] [4].

- **Decaimento β^-**

O decaimento β^- ocorre quando o núcleo atômico está com excesso de nêutrons em relação ao número de prótons, de forma que acontece a já citada conversão do nêutron em próton

$$n \rightarrow p + e^- + \overline{\nu}_e$$

neste processo o nêutron(n) decaí em um próton(p) e emite um elétron(e^-) e um antineutrino do elétron($\overline{\nu}_e$). Um exemplo de decaimento β^- é o césio-137, um isótopo radioativo muito utilizado em tratamento de radioterapia

$$^{137}_{55}\mathrm{Cs} \rightarrow {}^{137\mathrm{m}}_{56}\mathrm{Ba} + {}^{0}_{-1}\beta^- + {}^{0}_{0}\overline{\nu}_{\mathrm{e}}$$

o césio decaí para o bário metaestável[1] e emite um elétron e um antineutrino [1] [4].

- **Decaimento** β^+

Este decaimento está relacionado a conversão do próton em nêutron, em núcleos onde há um excesso de prótons em relação aos nêutrons

$$p \rightarrow n + e^+ + \nu_e$$

neste caso o próton(p) decaí em um nêutron(n) e emite um pósitron(e^+) e um neutrino do elétron(ν_e), um exemplo de decaimento β^- é a do magnésio-23:

$$^{23}_{12}\mathrm{Mg} \rightarrow {}^{23}_{11}\mathrm{Na} + {}^{0}_{1}\beta^+ + {}^{0}_{0}\nu_{\mathrm{e}}$$

onde o magnésio-23 decaí em sódio estável e emite pósitron e um neutrino.

Uma particularidade deste tipo de decaimento é que para ele acontecer a massa do núcleo pai que é o núcleo que irá decair, deve ser maior que a massa do núcleo filho mais duas vezes a massa do elétron, utilizando o exemplo anterior então [1] [4]:

$$M_{^{23}_{12}\mathrm{Mg}} > M_{^{23}_{11}\mathrm{Na}} + 2M_e$$

[1] Elementos metaestáveis, são elementos que mesmo o núcleo emitindo partículas para alcançar a estabilidade continuam energéticos, no caso do bário metaestável posteriormente é emitido radiação gama (subseção 1.2.3) para alcançar a estabilidade.

- **Captura eletrônica**

Quando o núcleo atômico precisa diminuir sua carga e a não consegue emitir uma partícula β^+ o próton pode se converter em nêutron capturando um elétron de suas camadas eletrônicas sendo que a maior probabilidade é a camada k, o que resulta na emissão de um raio X característico ou emissão de elétrons Auger, que se visto posteriormente, o resultado da emissão por captura eletrônica é [1] [4]:

$$ {}^{1}_{1}\mathrm{p} + {}^{0}_{-1}\mathrm{e_k} \rightarrow {}^{1}_{0}\mathrm{n} + {}^{0}_{0}\nu_\mathrm{e} $$

onde e_k é o elétron capturado da camada k, um exemplo deste tipo de decaimento é o berílio decaindo em lítio.

$$ {}^{7}_{4}\mathrm{Be} + {}^{0}_{-1}\mathrm{e_k} \rightarrow {}^{7}_{3}\mathrm{Li} + {}^{0}_{0}\nu_\mathrm{e} $$

- **Emissão de elétrons Auger**

Este fenômeno compete na emissão de um elétron das camadas superiores, quando elétrons das camadas inferiores transitam. Utilizando o exemplo da Figura 1.4, em (a) um elétron da camada K sofre a colisão de um fóton, em (b) o elétron colidido é ejetado do átomo. Após a ejeção do elétron dois eventos podem acontecer. Em (c) um elétron da camada L transita para a camada K emitido um raio X característico, em (d) acontece o mesmo processo de (c), porém, o raio X característico colide com um elétron da camada superior M, resultando na ejeção do elétron. Esse elétron ejetado é o elétron Auger [1] [4].

- **Produção e aniquilação de pares**

A produção de pares ocorre quando um fóton de alta energia interage com o campo coulombiano do núcleo atômico. A energia do fóton é absorvida e convertida em massa de repouso e energia cinética de um par partícula-antipartícula. Para o par elétron-pósitron a energia do fóton incidente deve ser maior que 1,022 MeV, ou seja, duas vezes a massa de repouso do elétron (0,511 MeV).

Já a aniquilação de pares acontece quando uma partícula e antipartícula se encontram resultando na emissão de dois fótons, um exemplo é o pósitron com energia de 0,511 MeV, quando ele interage com um elétron os dois se aniquilam resultando em emissão de radiação gama com dois fótons sendo emitidos em direções opostas com energia de 0,511 MeV para cada [1] [2] [4].

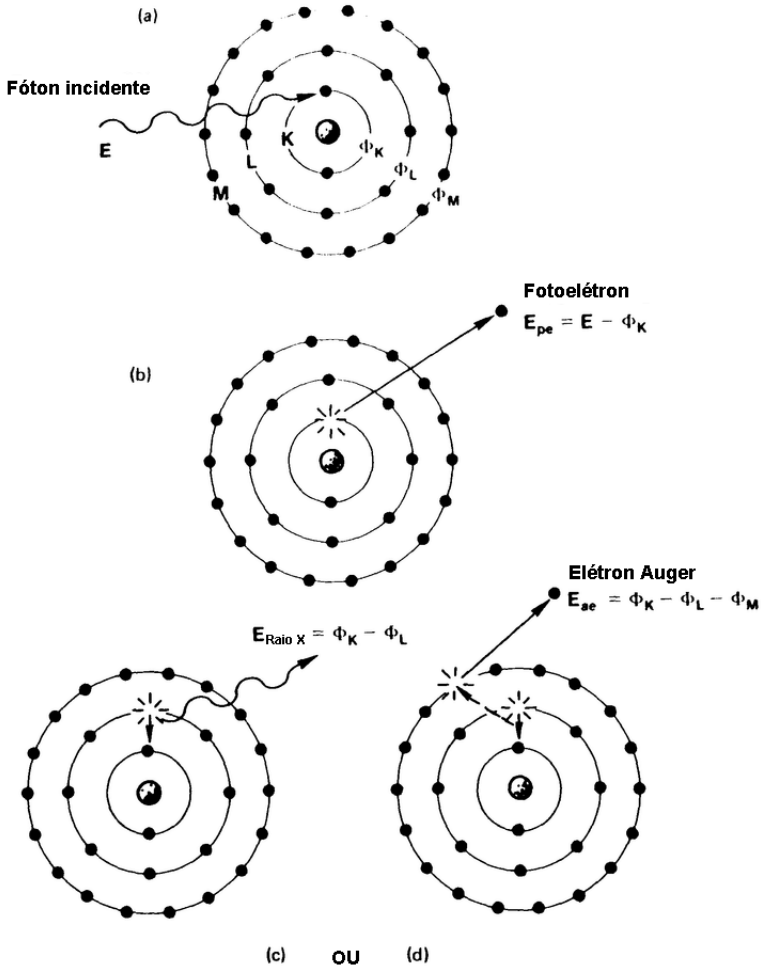


Figura 1.4: Produção do elétron Auger.

Fonte: https://www.researchgate.net/figure/Photoelectric-interaction-X-ray-fluorescence-and-Auger-Effect-After-the-photoelectric_fig8_335313954

1.2.3 Radiação Gama γ

Radiação γ é a emissão de um fóton pelo núcleo após decair com radiação alfa ou beta e ainda permanecer energético, quando isso ocorre se diz que o átomo está em um estado meta-estável e emite o fóton para diminuir sua energia para o átomo se estabilizar. Radiação γ e o raio X tem a mesma natureza, ambos são radiações eletromagnéticas e se diferem apenas de sua origem. Isto pode ser constatado na desintegração do Cs-137, tendo o bário metaestável como elemento filho. Logo em seguida da emissão de uma partícula β^-, o bário por fim emite uma radiação γ para sua estabilização nuclear [1] [3] [4].

$$^{137}_{55}\mathrm{Cs} \rightarrow {}^{137\mathrm{m}}_{56}\mathrm{Ba} + {}^{0}_{-1}\beta^- + {}^{0}_{0}\overline{\nu}_{\mathrm{e}}$$

$$^{137m}_{56}\mathrm{Ba} \rightarrow {}^{137}_{56}\mathrm{Ba} + {}^{0}_{0}\gamma$$

1.2.4 Grandezas de radiação

Após a descoberta dos raios X em 1895 e a radioatividade em 1896, houve uma grande popularização das radiações ionizantes. Fábricas de tubos de raios X começaram a ser abertas, sendo possível tirar radiografias de tudo. Após a popularização os pesquisadores viram que estavam lidando com um agente extremamente nocivo para a saúde. Demorou 30 anos para ser criada uma comissão para avaliar questões sobre radiações e também criar protocolos para medidas e os cuidados para operar esse tipo de radiação. Com isso, foi criada a comissão internacional de unidades e medidas em radiologia (ICRU - sigla inglesa). Três anos depois, criou-se o que conhecemos na atualidade, a comissão internacional de proteção radiológica (ICRP - sigla inglesa) [1] [4].

- **Exposição (X)**

A exposição, foi a primeira grandeza relacionada a radiação, definida apenas para as radiações eletromagnéticas. Essa grandeza é a capacidade de um fóton ionizar o ar medindo assim quantidade de cargas elétricas de mesmo sinal produzida por unidade de massa do ar, dado pela equação a seguir.

$$X = \frac{dQ}{dm}$$

Até 1985, era adotado o röntgen (R) como unidade de medida. Hoje a unidade nova é o C/kg (Coulomb por quilograma), na qual 1R = $2,58.10^{-4}$ C/kg [1] [5] [4].

- **Dose absorvida (D)**

A dose absorvida, é a grandeza mais importante quando se trata da interação radiação com a matéria biológica. Essa grandeza é de extrema importância na área de radioterapia, pois diferentemente da exposição, a dose absorvida pode ser calculada para qualquer tipo de radiação ionizante em qualquer meio. Está relacionada a média de energia depositada pela radiação em um volume de massa,

$$D = \frac{dE_{absorvida}}{dm}$$

a primeira unidade utilizada foi o rad, que foi definida de forma que a exposição de raios X de 1R resultasse em uma dose absorvida de 1 rad no tecido mole. 1 rad equivale a 0,01 J/kg (Joule por quilograma), atualmente a unidade de medida utilizado é o Gray (Gy) de forma que [1] [5] [4].

$$1\ Gy = 1\ J/kg = 100\ rad$$

.

- **Dose equivalente (H_T)**

A dose equivalente, é uma grandeza que relaciona o valor médio da dose absorvida causada em diferentes tipos de tecido ou orgão vezes o tipo de radiação.

$$H_T = w_R.D_{T,R}$$

Na qual w_R é o fator peso de cada radiação, $D_{T,R}$ é a dose absorvida no tecido em termos da radiação R. Sua nova unidade é o Sievert que vale [1] [5] [4].

$$1\ Sv = 1\ J/kg = 100\ rem$$

Capítulo 2

Ciência de dados

Ciência de dados é uma área que possui em torno de seus 30 anos, porém, ganhou certa notoriedade nos últimos anos devido ao avanço de capacidade computacional, a criação e popularização de grandes bancos dados. É uma área interdisciplinar voltada para o estudo e análise de dados, tanto dados estruturados que são armazenados de forma organizada, quanto dados não estruturados que não possuem nenhum tipo de organização. Possui a intenção final de obter informações e conhecimento de forma sistêmica, bem como normalizar e organizar esse conhecimento.

Essa popularização da área construiu uma grande comunidade, tal que contribuem muito fornecendo uma vasta quantidade de bibliotecas e plataformas. Esse conhecimento está disponível gratuitamente

2.1 Linguagem *Python*

Python é uma linguagem de programação com código aberto, de alto nível com programação orientada a objetos muito eficiente. Foi desenvolvida para aumentar eficiência na escrita dos códigos e leitura do mesmo, de forma que há um alto ganho no tempo do desenvolvedor, porém, com o sacrifício do tempo computacional para processar o código já que é uma linguagem interpretada [6]. Em resumo, *Python* é uma linguagem extremamente eficiente, com programas com menos quantidades de linhas, com uma escrita "limpa", fácil de ler, fácil de depurar, fácil de atualizar quando comparado com outras linguagens [7].

Hoje a linguagem é uma das mais populares do mundo com um grande número de aplicações como criação de *scripts* para automatizar tarefas em computadores, construção de *websites*

utilizando a biblioteca Django. Entre elas está a ciência de dados, aprendizado de máquina, tudo isso graças a grande comunidade que ajuda a manter e evoluir a linguagem [8].

O sucesso do *Python* na área de ciência de dados é devido a suas bibliotecas robustas para processamento e análise de dados como o *Pandas*, *Numpy*, *Matplotlib*, *Scikit learn* e etc.

2.2 Anaconda

Anaconda é um gerenciador de pacotes com ferramentas voltada para computação científica com linguagens *Python* e R. Em seu repositório pode-se encontrar mais de 7.500 pacotes voltados para ciência de dados e aprendizado de máquina. É um software multiplataforma podendo ser utilizado em sistemas Windows, Linux e macOS, e possui uma versão gratuita de código aberto [9], a Figura 2.1 mostra alguns *softwares* disponíveis para instalar dentro do navegador anaconda, já a Figura 2.2 mostra algumas das bibliotecas disponíveis no gerenciador.

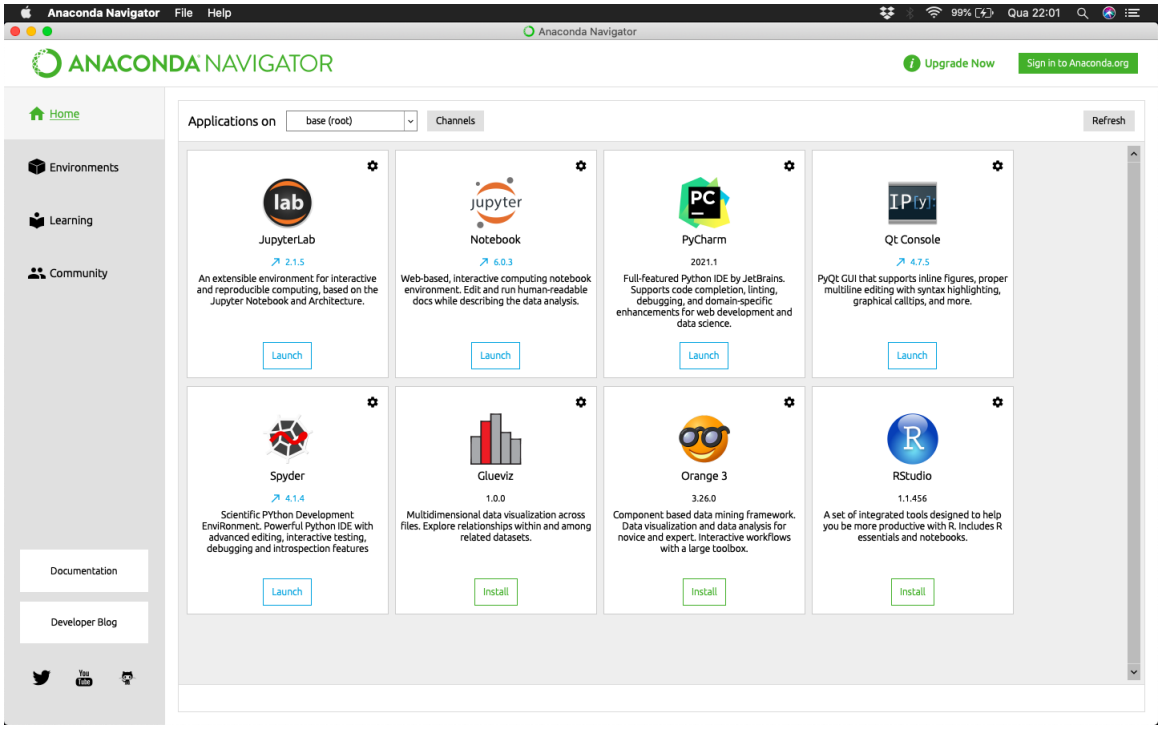


Figura 2.1: Interface do navegador anaconda.

Fonte: Elaborada pelo autor.

Figura 2.2: Alguns pacotes *Python* fornecido pelo anaconda.

Fonte: https://www.anaconda.com

2.3 Jupyter Notebook

Jupyter notebook é uma interface gráfica de código aberto que permite a execução de códigos de linguagem de programação e também edição de texto, tudo isso feito em um ambiente no navegador de internet (Figura 2.3), o nome Jupyter é derivado da junção de três linguagens de programação (Julia, *Python* e R), porém, é possível utilizar várias outras linguagens. É uma ferramenta muito importante para o trabalho com dados em *Python*, o Jupyter em si não oferece ferramentas de processamento ou análise de dados, sua função é maximizar a produtividade para explorar os dados incentivando o fluxo execução-exploração diferentemente do fluxo padrão edição-compilação-execução. Oferece também um acesso rápido ao *shell* do sistema operacional e os arquivos do computador [8].

Alguns recursos do Jupyter são, executar o código diretamente pelo navegador exibindo a saída do código após compilação da célula, mostrar resultados de mídia diretamente no navegador como extensões HTML, LaTeX, PNG, SVG entre outros. Pode-se utilizar a linguagem *Markdown* para edição de texto e utilizar o modo matemático do *LaTeX* para editar equações e expressões matemáticas [10].

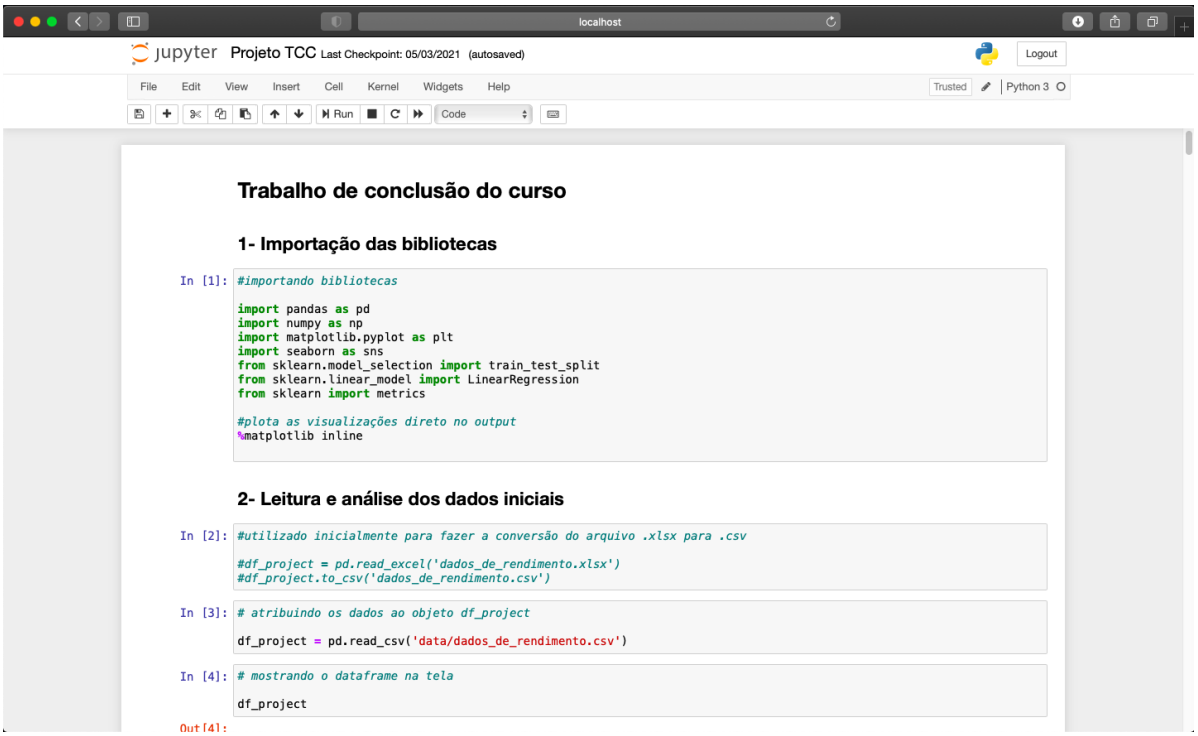


Figura 2.3: Interface do Jupyter Notebook.

Fonte: Elaborada pelo autor.

2.4 Bibliotecas

2.4.1 *Pandas*

O *Pandas* é uma biblioteca de código aberto que oferece estruturas de dados de alto nível e funções, projetadas para fazer com que trabalhar com dados estruturados ou tabulares seja rápida, fácil e expressivo. Os principais objetos dessa biblioteca são os *Dataframes*, que é uma estrutura de dados tabulares com orientação a colunas, com rótulos tanto para linhas quanto para colunas. Existe também as Séries que é um objeto *array* unidimensional com rótulo [8].

Essa biblioteca mescla o alto desempenho de *arrays Numpy* (2.4.2) com os recursos de manipulação de dados das planilhas e dos bancos de dados relacionais como SQL. Para análise de dados o *Pandas* é a primeira ferramenta a ser utilizada, para fazer o processamento, preparação, limpeza, análise e manuseio dos dados. Ela fornece os seguintes recursos, um rápido e eficiente objeto Dataframe com indexação integrada, a possibilidade de leitura de dados em diferentes extensões como CSV, *Microsoft Excel*, base de dados *SQL*, arquivos *JSON* entre outros. É uma ferramenta que oferece uma facilidade em trabalhar com dados perdidos ou faltantes, rótulos inteligentes com a capacidade de fazer *slicing* (fatiamento de dados), tem alta mutabilidade sendo

possível remover e adicionar colunas entre outros recursos.

Além do alto desempenho da biblioteca está atrelado a utilização de *arrays Numpy,* a própria biblioteca possui partes do seu código escrito em linguagem C. A área de aplicação do *Pandas* é variada tanto em área acadêmica quanto comercial, sendo utilizado na área de ciência de dados, financeira, neurociência, economia, estatística entre outros [11] [8] [12].

2.4.2 *Numpy*

Numpy (Numerical Python) é uma biblioteca de código aberto que oferece um código aglutinador para estrutura de dados com bibliotecas e algoritmos utilizado na maioria das aplicações científicas que envolvam dados numéricos em *Python* [8], pois possui um poderoso *array* N-dimensional fácil e versátil para ser utilizado de forma variada como indexação, vetorização e etc. Possui funções para efetuar operações em todos elementos do *array* e operação entre *arrays*. Oferece funções matemáticas como geradores de números aleatórios, operações de álgebra linear, transformadas de Fourier. Tem uma alta performance devido seu núcleo ser escrito em linguagem C misturando a facilidade de compreensão do *Python* com a eficiência da linguagem compilada, esse desempenho pode ser observado na Figura 2.4. A biblioteca foi utilizada em parte do software da descoberta das ondas gravitacionais pelo *LIGO (Laser Interferometer Gravitational-Wave Observatory)*, e também na primeira imagem feita de um buraco negro. [8] [13] [14].

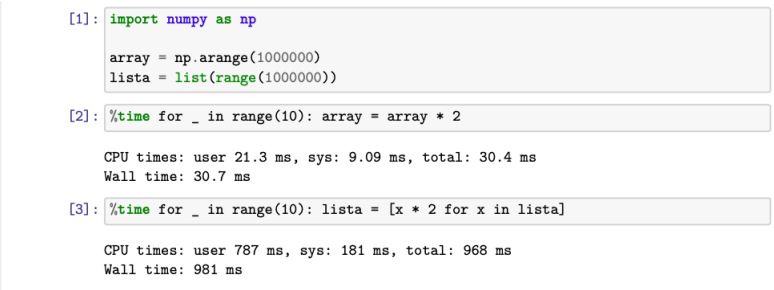


Figura 2.4: Tempo necessário para efetuar as multiplicações de um *array Numpy* e uma lista *Python* por 2, efetuada 10 vezes consecutivas.

Fonte: Elaborada pelo autor.

2.4.3 *Matplotlib*

Matplotlib é uma biblioteca para gerar gráficos baseado em *arrays Python*, ela teve origem nos comandos do *MATLAB (matrix laboratory)*, porém, escrito em *Python*, sendo assim orien-

tado a objetos. Mesmo sendo escrito em *Python* puro, essa é mais uma biblioteca que faz uso intenso da *Numpy* para obter um melhor desempenho [15] [16].

Embora exista outras bibliotecas para gerar visualizações disponíveis *Matplotlib* é a mais utilizada, sendo uma escolha segura para ser a biblioteca padrão criação de gráficos [8].

2.4.4 *Seaborn*

Seaborn é uma biblioteca de código aberto para visualizações estatísticas, ela se baseia no *Matplotlib* e possui uma integração muito próxima com as estruturas de dados do *Pandas*. Atua como uma biblioteca que de certa forma melhora as visualizações que seriam feitas no *Matplotlib* deixando com uma melhor aparência e mais atrativo [17] [18].

2.4.5 *Scikit-learn*

O *Scikit-learn* é uma biblioteca de código aberto com ferramentas para o *machine learning* (aprendizado de máquina). Oferece suporte para aprendizado supervisionado que é quando se fornece informações já rotuladas para a máquina. Aprendizado não supervisionado que é quando a própria máquina analisa os dados e faz agrupamentos baseado em similaridades e padrões. Ela utiliza como dependências apenas o *Numpy* e o *Scipy*, a biblioteca é escrita principalmente em *Python*, porém, também incorpora bibliotecas escrita em C++ [19].

Com a biblioteca é possível utilizar ferramentas de classificação com modelos *SVM (support vector machine)*, *nearst neighbors* (vizinhos próximos), *random forest* (floresta aleatória), ferramentas de regressão como regressões de *Lasso*, *Ridge*, Linear, Logística e etc. Ferramentas de *clustering* (agrupamentos por semelhança) como *k-means*, *clustering* espectral e etc. Ferramentas para redução de dimensionalidade como *PCA (Principal component analysis)*, seleção de atributos, fatoração de matrizes, ferramentas de seleção de modelos com métodos de validação cruzada. Busca em grade e algumas métricas e ferramentas de pré-processamento com extração de atributos e normalização [8] [20].

Capítulo 3

Desenvolvimento

3.1 Dados

Os dados utilizados foram fornecidos pelo Hospital do Câncer de Maringá. Estes dados foram coletados de um acelerador linear (Figura 3.1) da fabricante SIEMENS, modelo Primus Mid Energy, com energias de 6MeV (fótons) e 5, 7, 8, 10, 12, 14 MeV (elétrons), utilizado para tratamentos de câncer. Todos os dias antes da rotina de tratamento dos pacientes é feito uma checagem de rendimento da máquina, a fim de saber se ela está funcionando dentro da margem de segurança.

Figura 3.1: Acelerador linear.

Fonte: https://www.oncologysystems.com/inventory/medical-equipment-for-sale/used-linear-accelerators/siemens-primus-linear-accelerator

Os dados são obtidos a partir de condições bem específicas, o feixe utilizado é de fótons com energia média de 6 MeV, produzido por raio X de freamento. Toda medida foi realizada

em um *Phantom* d'água selado (Figura 3.2) à uma profundidade de 5 cm da superfície, abertura de campo de (10 x 10) cm², distância fonte superfície (SSD) igual a 100 cm e 100 de unidades monitoras. As medidas foram realizadas utilizando uma câmara de ionização (Figura 3.3) do tipo Farmer (Fabricante: PTW, Modelo: TN30013) .

Figura 3.2: Exemplo de *Phantom* d'água.

Fonte: https://www.iba-dosimetry.com/product/wp1d-water-phantom/

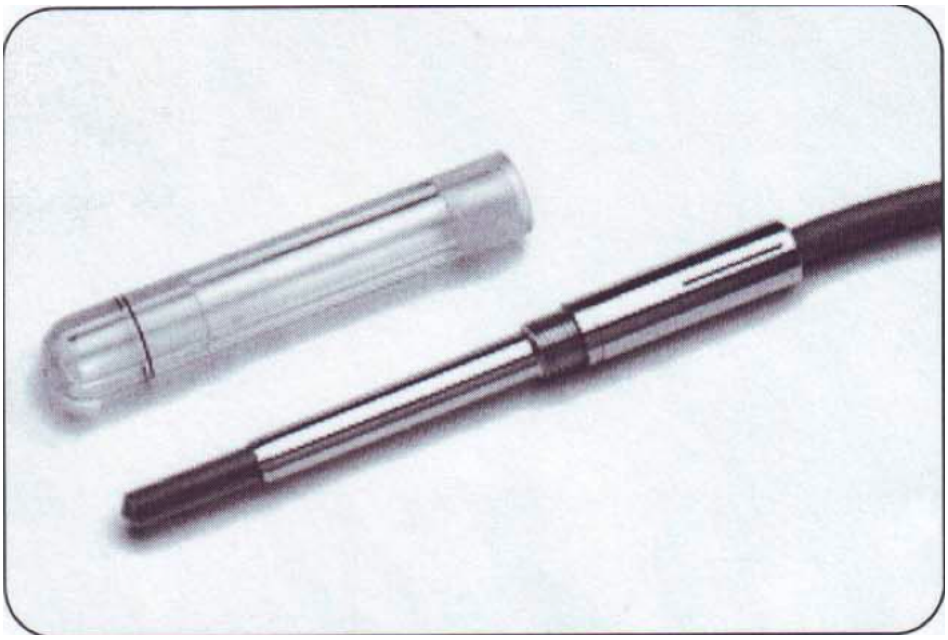


Figura 3.3: Câmara de ionização PTW, Modelo: TN30013.

Fonte: https://www.rpdinc.com/index.php?controller=attachment?id_attachment=22

O acelerador linear é um equipamento eletrônico, e conforme seu uso, vai sofrendo um

desajuste. A tendência do equipamento é que suas doses absorvidas vão aumentando com o passar do tempo. Quando o equipamento é calibrado, nessas condições específicas descritas acima, a dose absorvida deve ser ajustada para 100 cGy. A partir do dia de calibração as medidas de dose absorvida são sempre aferidas a fim de saber se o feixe está dentro da margem de segurança que é de 3% da dose ajustada no dia da calibração (chamado de Dif nos dados). Ou seja quando o equipamento atingir doses de 103 cGy, o acelerador deve se calibrado novamente.

O conjunto de dados possui as seguintes informações. As datas das medidas (no formato ano-mês-dia), doses absorvidas (em cGy), a porcentagem da dose absorvida no dia em relação a dose de referência do dia de calibração, a temperatura (em °C) e a pressão (em hPa).

3.2 Trabalhando com os dados

Após utilizar o anaconda para instalar todos os pacotes e programas necessários, o primeiro passo é abrir o Jupyter Notebook. Ele é o ambiente onde todo o trabalho será realizado.

Com o Jupyter aberto, se importa as bibliotecas (Figura 3.4) necessárias para o trabalho.

```
In [1]: #importando bibliotecas

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression
        from sklearn import metrics

        #plota as visualizações direto no output
        %matplotlib inline
```

Figura 3.4: Importação das bibliotecas.

Fonte: Elaborada pelo autor.

As bibliotecas utilizadas são as já citadas, *Pandas, Numpy, Matplotlib, Scikit-learn*. Do *Scikit-learn* foram importadas apenas as bibliotecas utilizadas no projeto.

A abertura dos dados é feito utilizando o *Pandas*. Os dados enviados estavam em formato do *Microsoft Excel* (.xlsx), então, o *Pandas* foi inicialmente usado para fazer a conversão de .xlsx para .CSV (Figura 3.5), uma extensão de arquivos mais simples que utiliza vírgulas para separação das colunas.

```
In [2]: #utilizado inicialmente para fazer a conversão do arquivo .xlsx para .csv

        df_project = pd.read_excel('dados_de_rendimento.xlsx')
        df_project.to_csv('dados_de_rendimento.csv')
```

Figura 3.5: Abrindo arquivo .xlsx e exportando como .CSV.

Fonte: Elaborada pelo autor.

Agora com o arquivo de dados já convertido, abre o arquivo .CSV como um objeto *Dataframe* (Figura 3.6).

```
In [3]: # atribuindo os dados ao objeto df_project

        df_project = pd.read_csv('data/dados_de_rendimento.csv')

In [4]: # mostrando o dataframe na tela

        df_project
```

Out[4]:

	Unnamed: 0	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12
0	0	2020-03-10	97.3560	-2.6440	18.6	956	NaN	Enviei os dados a partir de março de 2020 pois...	NaN	NaN	NaN	NaN	NaN	NaN
1	1	2020-03-11	97.4890	-2.5110	18.7	945	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2	2020-03-12	98.0775	-1.9225	18.8	949	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	3	2020-03-13	97.8950	-2.1050	19.1	950	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4	2020-03-16	98.0010	-1.9990	19.5	948	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
134	134	2020-09-29	98.8200	-1.1800	18.9	948	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
135	135	2020-09-30	98.6575	-1.3425	18.9	947	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
136	136	2020-10-01	98.8300	-1.1700	19.3	946	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
137	137	2020-10-02	98.8225	-1.1775	18.9	945	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
138	138	2020-10-05	98.8125	-1.1875	19.7	948	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

139 rows × 14 columns

Figura 3.6: Visualização dos dados.

Fonte: Elaborada pelo autor.

Com os dados já inseridos no Jupyter, é interessante fazer uma análise do tamanho do *Dataframe*, e verificar quantas colunas e linhas estão sendo utilizadas, com o atributo *.shape*. Utilizou-se também o atributo *.info()* para obter as informações necessárias, como o tipo de variável dos dados que compõe as colunas (inteiro, float, object e etc), e se as colunas possuem valores ausentes (Figura 3.7).

```
In [5]: #verificando quantidade de linhas e colunas do dataframe

        df_project.shape

Out[5]: (139, 14)

In [6]: #verificando quantas colunas tem valores ausentes

        df_project.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 139 entries, 0 to 138
        Data columns (total 14 columns):
         #   Column                Non-Null Count  Dtype
        ---  ------                --------------  -----
         0   Unnamed: 0            139 non-null    int64
         1   Data                  139 non-null    object
         2   Dose Absorvida (cGy)  139 non-null    float64
         3   Dif (%)               139 non-null    float64
         4   Temp (°C)             139 non-null    float64
         5   Pressão (hPa)         139 non-null    int64
         6   Unnamed: 5            0 non-null      float64
         7   Unnamed: 6            4 non-null      object
         8   Unnamed: 7            0 non-null      float64
         9   Unnamed: 8            0 non-null      float64
         10  Unnamed: 9            0 non-null      float64
         11  Unnamed: 10           0 non-null      float64
         12  Unnamed: 11           0 non-null      float64
         13  Unnamed: 12           0 non-null      float64
        dtypes: float64(10), int64(2), object(2)
        memory usage: 15.3+ KB
```

Figura 3.7: Tamanho e informações do *Dataframe*.

Fonte: Elaborada pelo autor.

O *Dataframe* possui 139 linhas e 14 colunas. Possuindo algumas colunas com valores ausentes. Com essas informações agora pode-se fazer a limpeza dos dados, eliminando os dados desnecessários. Como o *Dataframe* possui algumas colunas com dados nulos, é possível retirar todas com apenas um código, utilizando o método *.dropna(axis=1, how = 'any')* (Figura 3.8), onde *axis = 1* significa que o código está atuando no eixo das colunas, e *how = 'any'* que deve ser aplicado onde houver pelo menos um dado ausente.

```
In [8]: #retirando colunas com valores ausentes de uma vez

        df_project = df_project.dropna(axis=1, how='any')
        df_project
```

Out[8]:

	Unnamed: 0	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)
0	0	2020-03-10	97.3560	-2.6440	18.6	956
1	1	2020-03-11	97.4890	-2.5110	18.7	945
2	2	2020-03-12	98.0775	-1.9225	18.8	949
3	3	2020-03-13	97.8950	-2.1050	19.1	950
4	4	2020-03-16	98.0010	-1.9990	19.5	948
...	...	...	...	...	...	...
134	134	2020-09-29	98.8200	-1.1800	18.9	948
135	135	2020-09-30	98.6575	-1.3425	18.9	947
136	136	2020-10-01	98.8300	-1.1700	19.3	946
137	137	2020-10-02	98.8225	-1.1775	18.9	945
138	138	2020-10-05	98.8125	-1.1875	19.7	948

139 rows × 6 columns

Figura 3.8: Limpando colunas com ao menos um dado ausente.

Fonte: Elaborada pelo autor.

A coluna *Unnamed: 0* também pode ser retirada, pois é o índice utilizado no arquivo original

(Figura 3.9).

```
In [8]: #retirando coluna que não possuí dados necessários

        df_project = df_project.drop(['Unnamed: 0'], axis=1)
        df_project
```

Out[8]:

	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)
0	2020-03-10	97.3560	-2.6440	18.6	956
1	2020-03-11	97.4890	-2.5110	18.7	945
2	2020-03-12	98.0775	-1.9225	18.8	949
3	2020-03-13	97.8950	-2.1050	19.1	950
4	2020-03-16	98.0010	-1.9990	19.5	948
...	...	...	...	...	...
134	2020-09-29	98.8200	-1.1800	18.9	948
135	2020-09-30	98.6575	-1.3425	18.9	947
136	2020-10-01	98.8300	-1.1700	19.3	946
137	2020-10-02	98.8225	-1.1775	18.9	945
138	2020-10-05	98.8125	-1.1875	19.7	948

139 rows × 5 columns

Figura 3.9: Retirando coluna desnecessária.

Fonte: Elaborada pelo autor.

Para verificar se a limpeza foi feita com êxito, basta analisar os dados novamente com o método *.info()* (Figura 3.10).

```
In [9]: #verificando se colunas desnecessárias foram descartadas

        df_project.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 139 entries, 0 to 138
        Data columns (total 5 columns):
         #   Column                Non-Null Count  Dtype
        ---  ------                --------------  -----
         0   Data                  139 non-null    object
         1   Dose Absorvida (cGy)  139 non-null    float64
         2   Dif (%)               139 non-null    float64
         3   Temp (°C)             139 non-null    float64
         4   Pressão (hPa)         139 non-null    int64
        dtypes: float64(3), int64(1), object(1)
        memory usage: 5.6+ KB
```

Figura 3.10: Informações do *Dataframe* após limpeza.

Fonte: Elaborada pelo autor.

Com o *Dataframe* pronto para ser utilizado, o próximo passo é fazer o ajuste dos dados. Os dados enviados estão com uma defasagem no valor da dose absorvida, o primeiro valor é do dia da calibração então vale 100 cGy, nos dados está 97.3560 cGy. Essa diferença de 2.6440 cGy será utilizada para ajustar o restante dos dados. Ao alterar os dados de dose absorvida, também deve se corrigir o Dif. Para isso basta fazer uma operação por toda a coluna com os dados de dose absorvida somando o valor defasado. Na coluna do Dif basta fazer a dose absorvida no dia menos o valor do dia da calibração, e ainda fazer um ajuste para corrigir os dados para 4 casas decimais para manter coerência com os dados originais (Figura 3.11).

```
In [11]: #Corrigindo os dados

         df_project['Dose Absorvida (cGy)'] = df_project['Dose Absorvida (cGy)'] + (100 - 97.3560)
         df_project['Dif (%)'] = (df_project['Dose Absorvida (cGy)'] - 100.0)

In [12]: #Arredondando os valores do Dif para quatro casas decimais, da mesma forma que veio as informaçoes originais

         df_project = df_project.round({'Dif (%)' : 4})
         df_project
```

Out[12]:

	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)
0	2020-03-10	100.0000	0.0000	18.6	956
1	2020-03-11	100.1330	0.1330	18.7	945
2	2020-03-12	100.7215	0.7215	18.8	949
3	2020-03-13	100.5390	0.5390	19.1	950
4	2020-03-16	100.6450	0.6450	19.5	948
...	...	...	...	...	...
134	2020-09-29	101.4640	1.4640	18.9	948
135	2020-09-30	101.3015	1.3015	18.9	947
136	2020-10-01	101.4740	1.4740	19.3	946
137	2020-10-02	101.4665	1.4665	18.9	945
138	2020-10-05	101.4565	1.4565	19.7	948

139 rows × 5 columns

Figura 3.11: Ajuste dos dados.

Fonte: Elaborada pelo autor.

Para finalizar o ajuste do *Dataframe*, é adicionado uma coluna nova chamada Dias usando o *Numpy* (Figura 3.12), onde nela está a informação dos dias corridos de utilização do equipamento.

```
In [13]: #para uma melhor análise futura irei criar uma coluna chamada 'Dias'
         #semelhante ao índice do Dataframe como se fosse os dias corridos de utilização

         df_project['Dias'] = np.arange(139)
         df_project
```

Out[13]:

	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)	Dias
0	2020-03-10	100.0000	0.0000	18.6	956	0
1	2020-03-11	100.1330	0.1330	18.7	945	1
2	2020-03-12	100.7215	0.7215	18.8	949	2
3	2020-03-13	100.5390	0.5390	19.1	950	3
4	2020-03-16	100.6450	0.6450	19.5	948	4
...	...	...	...	...	...	...
134	2020-09-29	101.4640	1.4640	18.9	948	134
135	2020-09-30	101.3015	1.3015	18.9	947	135
136	2020-10-01	101.4740	1.4740	19.3	946	136
137	2020-10-02	101.4665	1.4665	18.9	945	137
138	2020-10-05	101.4565	1.4565	19.7	948	138

139 rows × 6 columns

Figura 3.12: Adicionando coluna com dias corridos de utilização do equipamento.

Fonte: Elaborada pelo autor.

Com os dados devidamente ajustados e organizados, agora se inicia a análise. Para conhecer um pouco o comportamento dos dados é utilizado o método *.describe()* (Figura 3.13). Obtendo uma estatística básica, como média, desvio padrão, valor mínimo, valor máximo e etc.

```
In [15]: #utilizando o método describe para fornecer uma estatística básica dos dados
         df_project.describe()
```

Out[15]:

	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)	Dias
count	139.000000	139.000000	139.000000	139.000000	139.000000
mean	100.930122	0.930122	18.895683	949.884892	69.000000
std	0.599258	0.599258	0.774585	3.589597	40.269923
min	99.798000	-0.202000	16.600000	940.000000	0.000000
25%	100.349750	0.349750	18.400000	947.000000	34.500000
50%	101.126500	1.126500	18.900000	950.000000	69.000000
75%	101.412750	1.412750	19.500000	952.000000	103.500000
max	102.124000	2.124000	20.700000	958.000000	138.000000

Figura 3.13: Estatística básica dos dados.

Fonte: Elaborada pelo autor.

Para analisar a correlação entre os dados fornecidos, é utilizado o método *.corr()* (Figura 3.14) para fazer a correlação de todas as colunas entre si. Por padrão o método *.corr()* utiliza a correlação de Pearson ρ, que é muito utilizado para análise de correlações lineares. A correlação de Pearson varia de -1 a 1, sendo -1 uma correlação forte inversa, 1 uma correlação forte e 0 sem correlação. Calculado pela equação:

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}.\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} = \frac{cov(X,Y)}{\sqrt{var(X).var(Y)}}$$

```
In [16]: # utilizando o método corr() para ver o quanto os dados estão correlacionados
         df_project.corr()
```

Out[16]:

	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)	Dias
Dose Absorvida (cGy)	1.000000	1.000000	0.285065	0.297430	0.789398
Dif (%)	1.000000	1.000000	0.285065	0.297430	0.789398
Temp (°C)	0.285065	0.285065	1.000000	0.109280	0.026507
Pressão (hPa)	0.297430	0.297430	0.109280	1.000000	0.316719
Dias	0.789398	0.789398	0.026507	0.316719	1.000000

Figura 3.14: Correlação entre os dados.

Fonte: Elaborada pelo autor.

Para facilitar a visualização entre as correlações, pode se utilizar os dados de correlação em um mapa de calor (Figura 3.15). Este mapa consiste em um gráfico de cores quentes que varia o espectro de cores conforme o valor das variáveis utilizadas.

```
In [19]: #utilizando os dados de correlação para criar uma vizualização de mapa de calor
         plt.figure(figsize= (10, 8))
         sns.heatmap(df_project_hm, annot=True)

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7feea95ae490>
```

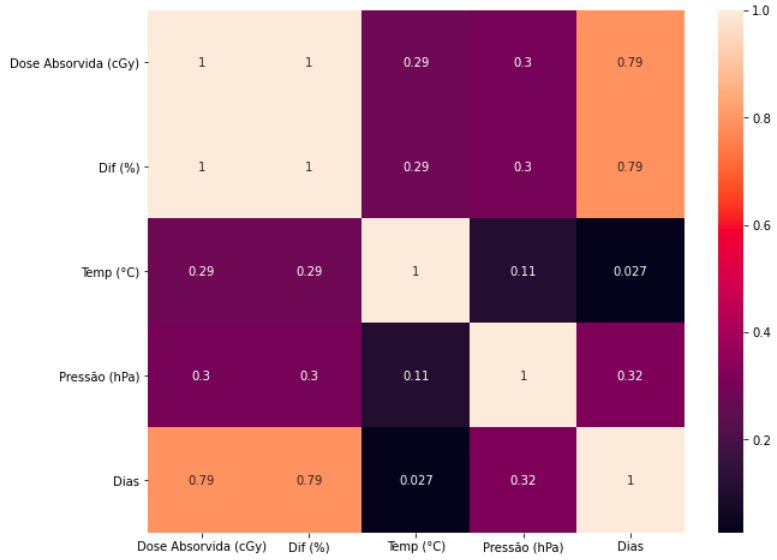


Figura 3.15: Mapa de calor dos valores de correlação.

Fonte: Elaborada pelo autor.

Analisando as informações de correlação, pode se observar uma correlação alta entre as doses absorvidas e os dias de uso do equipamento.

Além da estatística, uma forma interessante de explorar os dados, são as visualizações. Utilizando o *Pairplot* (Figura 3.16) da biblioteca *Seaborn*, que gera várias visualizações bidimensionais simples entre as grandezas do *Dataframe*, mostrando um panorama geral da relação entre eles. Os gráficos que compõe a relação de uma grandeza com ela mesma, gera um gráfico de distribuição de frequências. Para o restante são gerados gráficos de dispersão (Figura 3.17).

```
In [21]: #utilizando o Pairplot para visualizar de uma maneira geral o comportamento dos dados entre si

         sns.pairplot(df_project, kind= 'reg')

Out[21]: <seaborn.axisgrid.PairGrid at 0x7feea99534c0>
```

Figura 3.16: Código do *Pairplot*.

Fonte: Elaborada pelo autor.

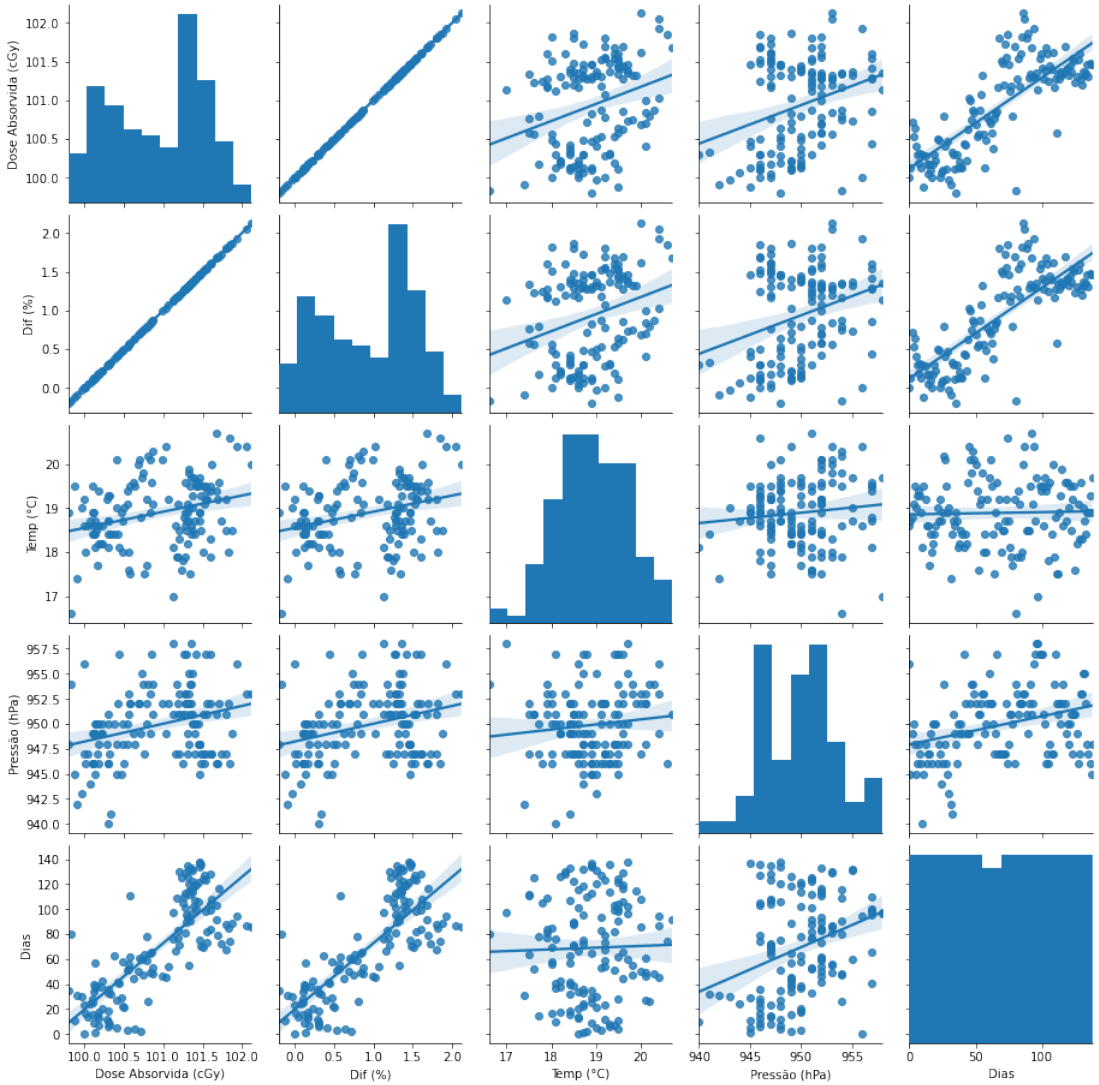


Figura 3.17: Visualização gerada pelo *Pairplot*.

Fonte: Elaborada pelo autor.

Com a visualização do *Pairplot* é possível observar o comportamento linear da dose absorvida e os dias corridos. Como o Dif tem dependência com a dose absorvida, sua correlação é 1, e tem uma dependência linear perfeita como mostrado no gráfico.

Para analisar o comportamento da dose absorvida, é feito um gráfico de curva de probabilidades junto a um histograma (Figuras 3.18 e 3.19).

```
In [22]: #fazendo uma visualização da curva de probabilidades das doses absorvidas
         plt.figure(figsize=(10, 6))
         sns.kdeplot(df_project['Dose Absorvida (cGy)'])

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7feeab1d4bb0>
```

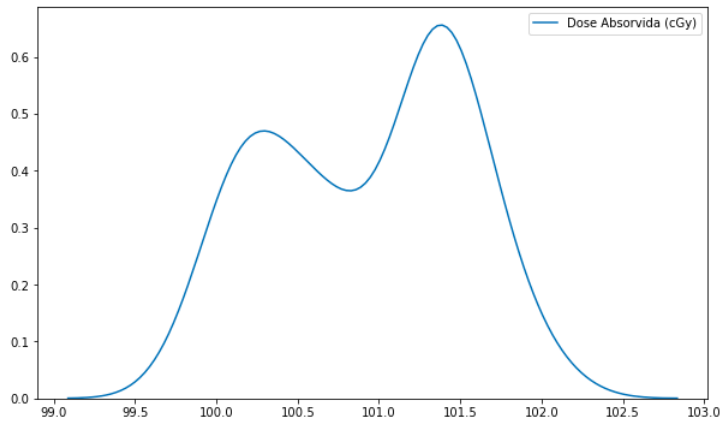


Figura 3.18: Curva de probabilidades da dose absorvida.

Fonte: Elaborada pelo autor.

```
In [24]: #mais uma visualização de probabilidades utilizando histograma

         plt.figure(figsize= (10,5))
         sns.distplot(df_project['Dose Absorvida (cGy)'], bins= 25, kde= False, rug= True)

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7feeab6d08e0>
```

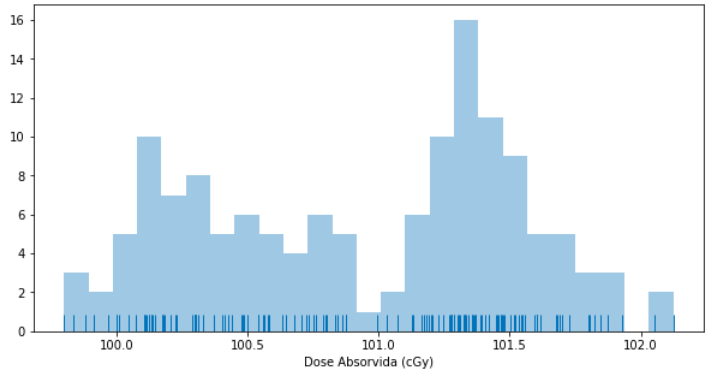


Figura 3.19: Histograma da dose absorvida.

Fonte: Elaborada pelo autor.

Agora utilizando uma visualização de distribuição de probabilidades bidimensional entre a dose absorvida e o dias de uso. O gráfico é similar a um gráfico de curva de nível, onde as regiões mais escuras são as de maior frequência (Figura 3.20).

```
In [23]: #verificar relação entre dose absorvida com os dias corridos de uso do equipamento

         sns.set_style('darkgrid')
         plt.figure(figsize=(10, 6))
         sns.jointplot(y= 'Dose Absorvida (cGy)', x= 'Dias', data= df_project, kind = 'kde')

Out[23]: <seaborn.axisgrid.JointGrid at 0x7f9259dcc670>

         <Figure size 720x432 with 0 Axes>
```

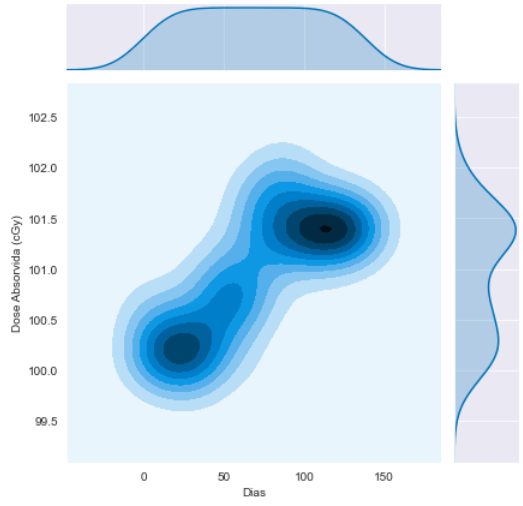


Figura 3.20: Distribuição de probabilidades bidimensional.

Fonte: Elaborada pelo autor.

Será utilizado agora visualizações de mapa de calor, que é uma ótima forma de analisar relações entre três grandezas. O primeiro mapa de calor (Figura 3.21) é relacionado as datas no eixo vertical, a temperatura no eixo horizontal e a dose absorvida representada pelas cores. O segundo mapa (Figura 3.22) é similar ao primeiro, porém, substituindo a temperatura pela pressão. E o terceiro mapa (Figura 3.23), relaciona a pressão, a temperatura e a dose absorvida.

Nos dois primeiros mapas é possível observar o aumento das doses com o passar do tempo, onde as cores (doses absorvidas) vai de um tom escuro para tons mais claros, no terceiro é uma tentativa para ver se a temperatura e a pressão estão relacionadas com as doses.

```
df_project_hm = df_project.pivot_table(index= 'Data', columns='Temp (°C)', values= 'Dose Absorvida (cGy)')
plt.figure(figsize=(15,10))
sns.heatmap(df_project_hm)
```

```
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9259b87d00>
```

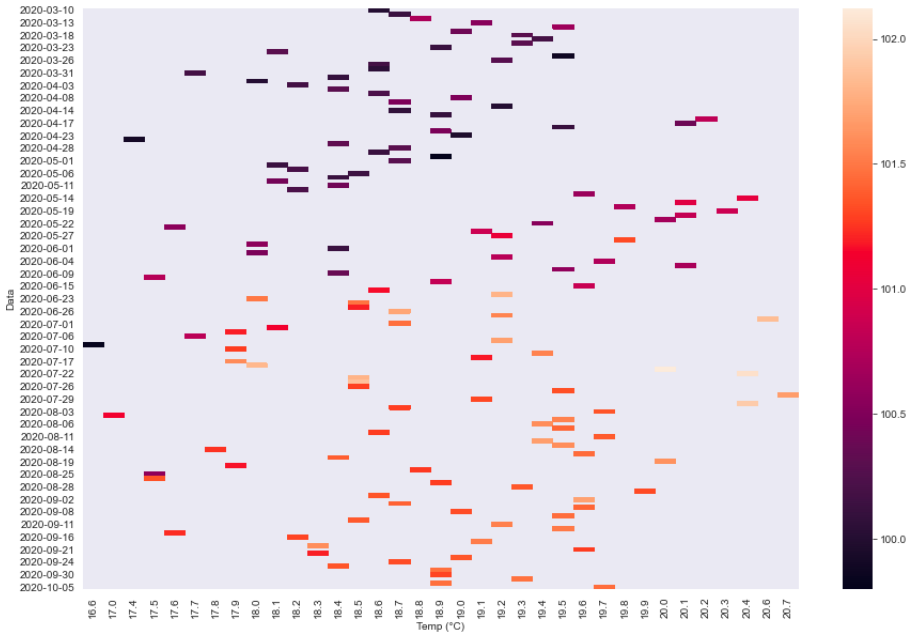


Figura 3.21: Mapa de calor data, temperatura e dose absorvida.

Fonte: Elaborada pelo autor.

```
df_project_hm = df_project.pivot_table(index= 'Data', columns='Pressão (hPa)', values= 'Dose Absorvida (cGy)')
plt.figure(figsize=(15,10))
sns.heatmap(df_project_hm)
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f925a8cd640>
```

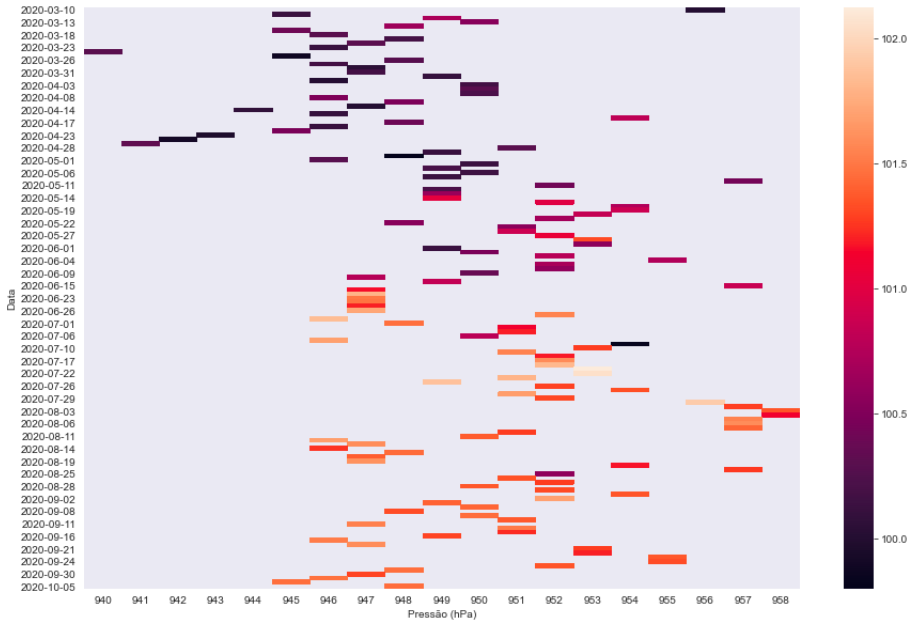


Figura 3.22: Mapa de calor data, pressão e dose absorvida.

Fonte: Elaborada pelo autor.

```
In [26]: df_project_hm = df_project.pivot_table(index= 'Temp (°C)', columns='Pressão (hPa)', values= 'Dose Absorvida (cGy)')
         plt.figure(figsize=(15,10))
         sns.heatmap(df_project_hm)

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f925a581f70>
```

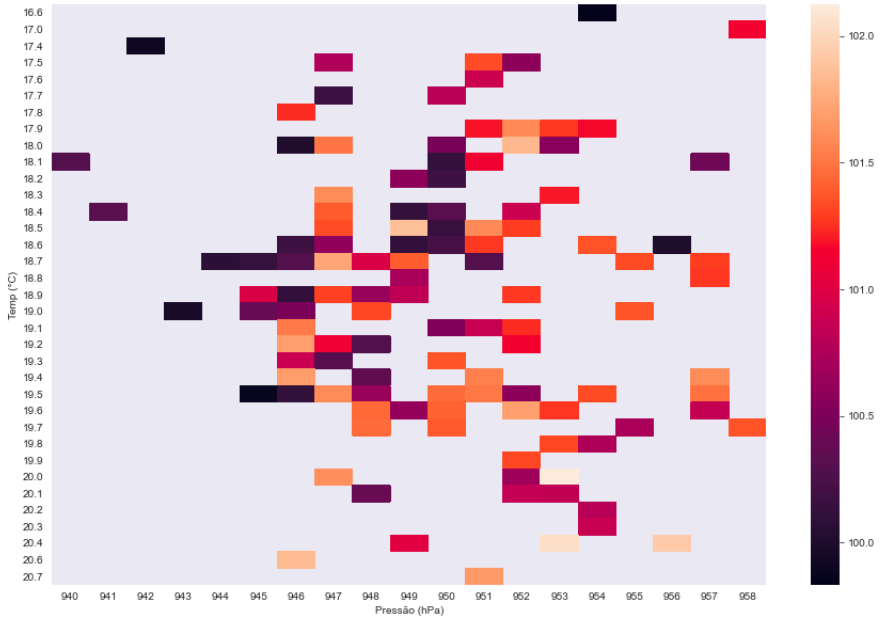


Figura 3.23: Mapa de calor temperatura, pressão e dose absorvida.

Fonte: Elaborada pelo autor.

3.3 Modelo de predição

Feito as análises de correlação e as visualizações, será utilizado o comportamento linear entre as doses absorvidas com os dias de uso do equipamento, a fim de conseguir encontrar um modelo que faça um melhor ajuste aos dados. O modelo utilizado será a regressão linear, utilizando os parâmetros de dose absorvida com os dias corridos de uso.

- **Regressão linear**

O modelo de regressão linear utiliza os valores dos dados a fim de encontrar uma reta que melhor se adéqua ao conjunto de dados.

$$y = aX + b$$

Será encontrado a equação da reta, onde X será os dados de dias corridos, e y a dose no respectivo dia. A regressão quando treinada com dados fornecerá o coeficiente angular (a) e o coeficiente linear (b). O primeiro passo é separar os dados de y e X do *Dataframe*. Para y utiliza o método *.pop( )*, de forma que a coluna dose absorvida é retirada do *Dataframe*. Para

X se utiliza o método *.drop( )*, onde é retirado as colunas desnecessárias, porém, todas ainda permanecem no *Dataframe* original, ficando apenas a coluna dias (Figura 3.24).

```
In [30]: #atribuindo os valores de X e y com os valores de dias corridos de uso do equipamento e a dose absorvida

         X = df_project.drop(['Dose Absorvida (cGy)', 'Dif (%)', 'Data','Temp (°C)', 'Pressão (hPa)'], axis=1)
         y = df_project.pop('Dose Absorvida (cGy)')
```

Figura 3.24: Seleção das grandezas y (dose absorvida) e X (dias).

Fonte: Elaborada pelo autor.

Antes de fazer o treinamento da regressão com dados, é interessante separar alguns dados para teste. Pois, se treinar o modelo com todos os dados, não sobraria dados reais para verificar se o modelo está explicando bem a realidade. Para fazer essa separação, utiliza do *Scikit-learn* a biblioteca *train_test_split* (Figura 3.25).

```
from sklearn.model_selection import train_test_split
```

Figura 3.25: Importação do *train_test_split*.

Fonte: Elaborada pelo autor.

O *train_test_split* opera da seguinte forma, se introduz de entrada os valores y e X. O código vai no conjunto y e retira de maneira randômica uma porção de dados. Essa quantidade retirada é descrita no código também, em forma de porcentagem, variando de 0.0 (0%) e 1.0 (100%). O mesmo acontece para os dados de X. Então o código retorna dois conjuntos de dados para cara variável, um conjunto para teste e outro para treino.

$$Entrada: y, X \rightarrow Saída: y\_treino, y\_teste, X\_treino, X\_teste$$

Na aplicação do modelo, foi utilizado uma proporção de 30% de dados para teste, e 70% para treino. No código a proporção é descrita como *test_size=0.3* (Figura 3.26).

```
In [31]: #da biblioteca do sklearn utilizamos a ferramenta que separa os dados para o treino da regressão e teste para
         #avaliar o quão bom está a estimativa

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3)
```

Figura 3.26: Separação dos dados para treino e teste.

Fonte: Elaborada pelo autor.

Após a separação dos dados, já é possível treinar o modelo de regressão linear. Para fazer o treinamento é importado do *Scikit-learn* a biblioteca *LinearRegression* (Figura 3.27).

```
from sklearn.linear_model import LinearRegression
```

Figura 3.27: Biblioteca importada para regressão linear.

Fonte: Elaborada pelo autor.

Com a biblioteca importada, cria-se uma variável para receber as propriedades da regressão linear (Figura 3.28).

```
In [32]: #atribuinado o objeto LinearRegression() a variável lm

         lm = LinearRegression()
```

Figura 3.28: Atribuindo o objeto *LinearRegression*.

Fonte: Elaborada pelo autor.

Para fazer o treinamento é utilizado o método *.fit( )*, que vai receber os dados de treino de y e X (Figura 3.29).

```
In [33]: #treinando o modelo

         lm.fit(X_train, y_train)

Out[33]: LinearRegression()
```

Figura 3.29: Treinando a regressão linear.

Fonte: Elaborada pelo autor.

Com a regressão já treinada, ou seja a equação da reta já está pronta. Utiliza-se os dados separados para teste de X, a fim de saber qual o valor de y estimado pela regressão. Para isso é utilizado o método *.predict( )*, recebendo os dados de teste X (Figura 3.30).

```
In [34]: #utilizando o método predict para obter os valores de previsão baseado na regressão ja treinada, fornecendo os
         #dados separados para o teste

         pred = lm.predict(X_test)
         pred

Out[34]: array([100.34229102, 101.5398696 , 101.59800448, 100.31903707,
                100.67947334, 101.55149658, 101.42359983, 100.353918  ,
                101.57475053, 100.52832265, 100.63296544, 100.47018777,
                101.10967147, 100.41205288, 100.66784637, 100.3074101 ,
                100.26090219, 100.90038589, 100.64459241, 100.9468938 ,
                101.71427425, 100.14463243, 101.17943333, 101.07479054,
                101.37709193, 101.40034588, 100.71435427, 101.56312355,
                100.22602126, 101.19106031, 101.65613936, 100.95852078,
                100.88875892, 101.02828264, 100.86550497, 101.62125843,
                100.56320358, 101.23756821, 101.09804449, 100.77248915,
                101.47010774, 100.45856079])
```

Figura 3.30: A variável pred, recebe os dados de previsão estipulado pela regressão, baseado nos dados de teste.

Fonte: Elaborada pelo autor.

O próximo passo é criar um novo *Dataframe*, que possui duas colunas. A primeira recebe os dados de y para o teste, que são os valores reais das doses. O segundo recebe os valores de doses estimado pelo modelo de regressão (Figura 3.31).

```
In [35]: #criando um dataframe com os dados reais separados para teste com a previsao feita pelo modelo

         dis = pd.DataFrame(y_test)
         dis['pred'] = pred

In [36]: #vendo os dez primeiros valores do dataframe

         dis.head(10)

Out[36]:
```

	Dose Absorvida (cGy)	pred
17	100.0090	100.342291
120	101.3265	101.539870
125	101.2265	101.598004
15	100.1715	100.319037
46	100.9915	100.679473
121	101.4540	101.551497
110	101.2690	101.423600
18	100.1750	100.353918
123	101.5340	101.574751
33	100.2990	100.528323

Figura 3.31: *Dataframe* com o valores de teste e os valores da previsão.

Fonte: Elaborada pelo autor.

Antes de utilizar métricas para fazer a análise dos resultados previsto pela regressão, comparado aos valores reais, é interessante fazer uma visualização para analisar graficamente a dispersão dos pontos (Figura 3.32).

```
In [37]: #visualizacão dos dados de previsão com os dados de teste
         plt.figure(figsize= (10, 8))
         sns.lmplot(x= 'Dose Absorvida (cGy)', y= 'pred', data= dis)

Out[37]: <seaborn.axisgrid.FacetGrid at 0x7f925b8293d0>

         <Figure size 720x576 with 0 Axes>
```

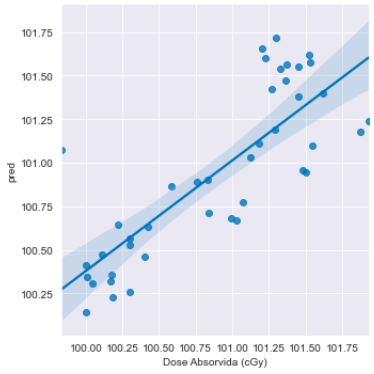


Figura 3.32: Gráfico de dispersão entre valores reais com os valores da previsão.

Fonte: Elaborada pelo autor.

A primeira métrica para avaliar é erro quadrático médio (*MSE* - sigla inglesa). Descrito na equação abaixo, onde N é o número de amostras, y_i é o valor real, e $\hat{y}_i$ é o valor predito.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Está métrica *MSE* eleva ao quadrado a diferença entre o valor real com o valor predito, possuindo uma grande sensibilidade a erros grandes. Seu valor é positivo variando de zero (nenhum erro) e não possuindo limite superior. É uma métrica interessante para fazer comparações entre modelos (Figura 3.33).

```
In [39]: #calculando o erro médio quadrado

         MSE = metrics.mean_squared_error(y_test, pred)
         MSE

Out[39]: 0.12738662019836378
```

Figura 3.33: Erro quadrático médio.

Fonte: Elaborada pelo autor.

A próxima métrica é o coeficiente de determinação (R^2) (Figura 3.34). É uma métrica que visa expressar a quantidade da variança dos dados que é explicada pelo modelo construído. Em outras palavras, essa medida calcula qual a porcentagem da variança que pôde ser prevista pelo modelo de regressão e, portanto, nos diz o quão "próximo" as medidas reais estão do nosso modelo. Seu valor varia de 0.0 (0%) até 1.0 (100%), onde 0.0 significa que o modelo não está prevendo os dados reais, e 1.0 significa que o modelo prevê totalmente os dados reais.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}$$

Onde N é o número de amostras, y_i é o valor real, $\hat{y}_i$ é o valor predito, e $\bar{y}_i$ é o valor médio das amostras.

```
In [40]: #calculando o coeficiente de determinação r2 para ver o quão bom o modelo está explicando os dados

         r2 = metrics.r2_score(y_test,pred)
         r2

Out[40]: 0.6456430602302263
```

Figura 3.34: Coeficiente de determinação.

Fonte: Elaborada pelo autor.

Isso mostra que com os dados de treino, o modelo conseguiu explicar 64.6 % dos dados.

Como o modelo de regressão linear calcula o coeficiente angular e o coeficiente linear da reta, pode se utilizar essas informações para fazer uma previsão da possível quantidade de dias corridos necessários para uma próxima calibração do equipamento. Para isso é definido uma

função que retorna quantos dias de uso a máquina teve para chegar a uma dose absorvida de interesse (Figura 3.35). Para obter o coeficiente linear é utilizado o método *.intercept_*, e para o coeficiente angular se utiliza o método *.coef_*. Agora basta inverter a equção da reta.

$$y = ax + b \rightarrow x = \frac{y - b}{a}$$

```
In [41]: #definindo uma função que retorna a possível quantidade de dias que a máquina será utilizada até a próxima
         #calibração, utilizando o coeficiente angular e o coeficiente linear do modelo

         def dia_de_calibração(dose):
             #devolve o valor possível da próxima calibração do equipamento em dias corridos de utilização

             day = (dose - lm.intercept_)/lm.coef_
             return day
```

Figura 3.35: Função que retorna o dia corrido para uma determinada dose absorvida.

Fonte: Elaborada pelo autor.

Com a função pronta, basta inserir a dose limite de 103 cGy para saber a previsão do modelo de quantos dias o equipamento será utilizado até uma próxima calibração (Figura 3.36).

```
In [42]: #dias de uso estimado até a próxima calibração

         dia_de_calibração(103.0000)

Out[42]: array([245.58126422])
```

Figura 3.36: Função que retorna o dia corrido para uma determinada dose absorvida.

Fonte: Elaborada pelo autor.

A previsão fornecida pelo modelo é de 245 dias de uso.

Sabendo que cada vez que o *train_test_split* é utilizado, os dados de treino e teste selecionados são diferentes de maneira randômica . Então, para cada vez que se roda o código as métricas variam também. A fim de saber se as métricas estão muito dispersas, é interessante testar o modelo várias vezes. Neste caso é repetido 1000 vezes, e para cada vez sempre guardando os resultados dos erros quadráticos médios, dos coeficientes de determinação e da previsão de dias de uso até a próxima calibração (Figura 3.37).

```
In [38]: #criando listas para receber os valores dos erros médios, os coeficientes de determinação r2, e os dias estimados
         #de calibração

         media_MSE = []
         media_r2 = []
         media_calibracao = []
```

```
In [39]: #como a ferramenta do split sempre pega valores aleatórios para treinar o modelo, agora testaremos 1000 vezes
         #para testar o modelo cada vez com valores diferentes

         for simu in range(1000):
             lm = 0
             MSE = 0
             pred = 0
             r2 = 0
             dia = 0
             X_train = 0
             X_test= 0
             y_train = 0
             y_test =0



             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3)
             lm = LinearRegression()
             lm.fit(X_train, y_train)
             pred = lm.predict(X_test)
             MSE = metrics.mean_squared_error(y_test, pred)
             r2 = metrics.r2_score(y_test,pred)
             dia = dia_de_calibração(103.0000).tolist()

             for x in dia:
                 a = x


             media_MSE.append(MSE)
             media_r2.append(r2)
             media_calibracao.append(a)
```

Figura 3.37: Código utilizado para rodar a simulação 1000 vezes.

Fonte: Elaborada pelo autor.

Após os 1000 resultados gerados por modelos treinados com uma seleção de diferentes para cada vez, é possível calcular a média e desvio padrão para cada uma das métricas (Figura 3.38). É possível observar que o erro quadrático médio ficou em média 0.137 com um desvio de 0.03, o coeficiente de determinação ficou em média 0.61 com desvio padrão de 0.07, e a média do dia de calibração é 245 dias com um desvio padrão de 6 dias.

```
In [42]: valores_df.describe()
```

Out[42]:

	MSE	R2	Dia de calibração
count	1000.000000	1000.000000	1000.000000
mean	0.137527	0.608210	245.126599
std	0.029396	0.070559	6.532319
min	0.068995	0.320549	225.406442
25%	0.116026	0.561548	240.759512
50%	0.135826	0.615528	245.154820
75%	0.157505	0.660846	249.439060
max	0.231654	0.786842	264.258237

Figura 3.38: Média dos resultados após 1000 repetições da simulação.

Fonte: Elaborada pelo autor.

Conclusões

O modelo de regressão linear, conseguiu explicar em média 60% do comportamento dos dados de doses absorvidas. Porém, com grandes variações, dependendo da seleção dos dados de treinamento e de teste do modelo. Uma quantidade maior de dados para treinar o modelo poderia diminuir essa variação. Para uma melhora no desempenho das previsões, deve ser considerado outros fatores que causam o desajuste do equipamento. Por se tratar de dados experimentais, deve-se considerar erros dos equipamentos e fatores externos. Neste caso por se tratar de uma área onde tudo é feito para minimizar danos, a utilização de equipamentos de alta tecnologia e correções nas medidas, fazem os erros serem muito pequenos.

As ferramentas de ciência de dados se mostraram muito eficientes. Todas as bibliotecas utilizadas operam de forma rápida e estável. A possibilidade de trabalhar em uma só plataforma, operando em forma de execução-exploração trás um grande aumento no fluxo de trabalho. Tudo isso sendo feito, de forma gratuita com programas e pacotes de código aberto.

Apêndice A

Código do projeto

Projeto TCC

April 19, 2021

1 Trabalho de conclusão do curso

1.1 1- Importação das bibliotecas

[1]:
```
#importando bibliotecas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

#plota as visualizações direto no output
%matplotlib inline
```

1.2 2- Leitura e análise dos dados iniciais

[2]:
```
#utilizado inicialmente para fazer a conversão do arquivo .xlsx para .csv

#df_project = pd.read_excel('dados_de_rendimento.xlsx')
#df_project.to_csv('dados_de_rendimento.csv')
```

[3]:
```
# atribuindo os dados ao objeto df_project

df_project = pd.read_csv('data/dados_de_rendimento.csv')
```

[4]:
```
# mostrando o dataframe na tela

df_project
```

[4]:

	Unnamed: 0	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	\
0	0	2020-03-10	97.3560	-2.6440	18.6	
1	1	2020-03-11	97.4890	-2.5110	18.7	
2	2	2020-03-12	98.0775	-1.9225	18.8	
3	3	2020-03-13	97.8950	-2.1050	19.1	

```
4             4  2020-03-16              98.0010  -1.9990       19.5
..          …         …                   …      …        …
134         134  2020-09-29              98.8200  -1.1800       18.9
135         135  2020-09-30              98.6575  -1.3425       18.9
136         136  2020-10-01              98.8300  -1.1700       19.3
137         137  2020-10-02              98.8225  -1.1775       18.9
138         138  2020-10-05              98.8125  -1.1875       19.7

     Pressão (hPa)  Unnamed: 5  \
0              956         NaN
1              945         NaN
2              949         NaN
3              950         NaN
4              948         NaN
..             …         …
134            948         NaN
135            947         NaN
136            946         NaN
137            945         NaN
138            948         NaN

                                          Unnamed: 6  Unnamed: 7  \
0    Enviei os dados a partir de março de 2020 pois…         NaN
1                                                NaN         NaN
2                                                NaN         NaN
3                                                NaN         NaN
4                                                NaN         NaN
..                                               …         …
134                                              NaN         NaN
135                                              NaN         NaN
136                                              NaN         NaN
137                                              NaN         NaN
138                                              NaN         NaN

     Unnamed: 8  Unnamed: 9  Unnamed: 10  Unnamed: 11  Unnamed: 12
0           NaN         NaN          NaN          NaN          NaN
1           NaN         NaN          NaN          NaN          NaN
2           NaN         NaN          NaN          NaN          NaN
3           NaN         NaN          NaN          NaN          NaN
4           NaN         NaN          NaN          NaN          NaN
..          …         …          …          …          …
134         NaN         NaN          NaN          NaN          NaN
135         NaN         NaN          NaN          NaN          NaN
136         NaN         NaN          NaN          NaN          NaN
137         NaN         NaN          NaN          NaN          NaN
138         NaN         NaN          NaN          NaN          NaN
```

```
[139 rows x 14 columns]
```

[5]:
```
#verificando quantidade de linhas e colunas do dataframe

df_project.shape
```

[5]:
```
(139, 14)
```

[6]:
```
#verificando quantas colunas tem valores ausentes

df_project.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 139 entries, 0 to 138
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            139 non-null    int64
 1   Data                  139 non-null    object
 2   Dose Absorvida (cGy)  139 non-null    float64
 3   Dif (%)               139 non-null    float64
 4   Temp (°C)             139 non-null    float64
 5   Pressão (hPa)         139 non-null    int64
 6   Unnamed: 5            0 non-null      float64
 7   Unnamed: 6            4 non-null      object
 8   Unnamed: 7            0 non-null      float64
 9   Unnamed: 8            0 non-null      float64
 10  Unnamed: 9            0 non-null      float64
 11  Unnamed: 10           0 non-null      float64
 12  Unnamed: 11           0 non-null      float64
 13  Unnamed: 12           0 non-null      float64
dtypes: float64(10), int64(2), object(2)
memory usage: 15.3+ KB
```

1.3 3- Limpeza e organização dos dados

[7]:
```
#retirando colunas com valores ausentes de uma vez

df_project = df_project.dropna(axis=1, how='any')
df_project
```

[7]:

	Unnamed: 0	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	\
0	0	2020-03-10	97.3560	-2.6440	18.6	
1	1	2020-03-11	97.4890	-2.5110	18.7	
2	2	2020-03-12	98.0775	-1.9225	18.8	
3	3	2020-03-13	97.8950	-2.1050	19.1	
4	4	2020-03-16	98.0010	-1.9990	19.5	

```
..          …         …                    …      …         …
134         134  2020-09-29                98.8200  -1.1800       18.9
135         135  2020-09-30                98.6575  -1.3425       18.9
136         136  2020-10-01                98.8300  -1.1700       19.3
137         137  2020-10-02                98.8225  -1.1775       18.9
138         138  2020-10-05                98.8125  -1.1875       19.7

     Pressão (hPa)
0              956
1              945
2              949
3              950
4              948
..             …
134            948
135            947
136            946
137            945
138            948

[139 rows x 6 columns]
```

[8]:
```
#retirando coluna que não possuí dados necessários

df_project = df_project.drop(['Unnamed: 0'], axis=1)
df_project
```

[8]:
```
            Data  Dose Absorvida (cGy)  Dif (%)  Temp (°C)  Pressão (hPa)
0     2020-03-10                97.3560  -2.6440       18.6            956
1     2020-03-11                97.4890  -2.5110       18.7            945
2     2020-03-12                98.0775  -1.9225       18.8            949
3     2020-03-13                97.8950  -2.1050       19.1            950
4     2020-03-16                98.0010  -1.9990       19.5            948
..           …                     …        …         …              …
134   2020-09-29                98.8200  -1.1800       18.9            948
135   2020-09-30                98.6575  -1.3425       18.9            947
136   2020-10-01                98.8300  -1.1700       19.3            946
137   2020-10-02                98.8225  -1.1775       18.9            945
138   2020-10-05                98.8125  -1.1875       19.7            948

[139 rows x 5 columns]
```

[9]:
```
#verificando se colunas desnecessárias foram descartadas

df_project.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 139 entries, 0 to 138
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Data                  139 non-null    object
 1   Dose Absorvida (cGy)  139 non-null    float64
 2   Dif (%)               139 non-null    float64
 3   Temp (°C)             139 non-null    float64
 4   Pressão (hPa)         139 non-null    int64
dtypes: float64(3), int64(1), object(1)
memory usage: 5.6+ KB
```

[10]:
```
#visualizando Dataframe após a limpeza

df_project
```

[10]:

	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)
0	2020-03-10	97.3560	-2.6440	18.6	956
1	2020-03-11	97.4890	-2.5110	18.7	945
2	2020-03-12	98.0775	-1.9225	18.8	949
3	2020-03-13	97.8950	-2.1050	19.1	950
4	2020-03-16	98.0010	-1.9990	19.5	948
..	…	…	…	…	…
134	2020-09-29	98.8200	-1.1800	18.9	948
135	2020-09-30	98.6575	-1.3425	18.9	947
136	2020-10-01	98.8300	-1.1700	19.3	946
137	2020-10-02	98.8225	-1.1775	18.9	945
138	2020-10-05	98.8125	-1.1875	19.7	948

[139 rows x 5 columns]

[11]:
```
#Corrigindo os dados

df_project['Dose Absorvida (cGy)'] = df_project['Dose Absorvida (cGy)'] + (100␣
↪- 97.3560)
df_project['Dif (%)'] = (df_project['Dose Absorvida (cGy)'] - 100.0)
```

[12]:
```
#Arredondando os valores do Dif para quatro casas decimais, da mesma forma que␣
↪veio as informaçoes originais

df_project = df_project.round({'Dif (%)' : 4})
df_project
```

[12]:

	Data	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)
0	2020-03-10	100.0000	0.0000	18.6	956
1	2020-03-11	100.1330	0.1330	18.7	945
2	2020-03-12	100.7215	0.7215	18.8	949

```
3    2020-03-13              100.5390   0.5390       19.1            950
4    2020-03-16              100.6450   0.6450       19.5            948
..          …                    …      …         …              …
134  2020-09-29              101.4640   1.4640       18.9            948
135  2020-09-30              101.3015   1.3015       18.9            947
136  2020-10-01              101.4740   1.4740       19.3            946
137  2020-10-02              101.4665   1.4665       18.9            945
138  2020-10-05              101.4565   1.4565       19.7            948

[139 rows x 5 columns]
```

[13]:
```
#para uma melhor análise futura irei criar uma coluna chamada 'Dias'
#semelhante ao índice do Dataframe como se fosse os dias corridos de utilização

df_project['Dias'] = np.arange(139)
df_project
```

[13]:
```
           Data  Dose Absorvida (cGy)  Dif (%)  Temp (°C)  Pressão (hPa)  Dias
0    2020-03-10              100.0000   0.0000       18.6            956     0
1    2020-03-11              100.1330   0.1330       18.7            945     1
2    2020-03-12              100.7215   0.7215       18.8            949     2
3    2020-03-13              100.5390   0.5390       19.1            950     3
4    2020-03-16              100.6450   0.6450       19.5            948     4
..          …                    …      …         …              …   …
134  2020-09-29              101.4640   1.4640       18.9            948   134
135  2020-09-30              101.3015   1.3015       18.9            947   135
136  2020-10-01              101.4740   1.4740       19.3            946   136
137  2020-10-02              101.4665   1.4665       18.9            945   137
138  2020-10-05              101.4565   1.4565       19.7            948   138

[139 rows x 6 columns]
```

1.4 4- Conhecendo os dados

[14]:
```
#utilizando o método describe para fornecer uma estatística básica dos dados

df_project.describe()
```

[14]:
```
       Dose Absorvida (cGy)     Dif (%)   Temp (°C)  Pressão (hPa)        Dias
count            139.000000  139.000000  139.000000     139.000000  139.000000
mean             100.930122    0.930122   18.895683     949.884892   69.000000
std                0.599258    0.599258    0.774585       3.589597   40.269923
min               99.798000   -0.202000   16.600000     940.000000    0.000000
25%              100.349750    0.349750   18.400000     947.000000   34.500000
50%              101.126500    1.126500   18.900000     950.000000   69.000000
75%              101.412750    1.412750   19.500000     952.000000  103.500000
max              102.124000    2.124000   20.700000     958.000000  138.000000
```

[15]:
```
# utilizando o método corr() para ver o quanto os dados estão correlacionados

df_project.corr()
```

[15]:

	Dose Absorvida (cGy)	Dif (%)	Temp (°C)	Pressão (hPa)	Dias
Dose Absorvida (cGy)	1.000000	1.000000	0.285065	0.297430	0.789398
Dif (%)	1.000000	1.000000	0.285065	0.297430	0.789398
Temp (°C)	0.285065	0.285065	1.000000	0.109280	0.026507
Pressão (hPa)	0.297430	0.297430	0.109280	1.000000	0.316719
Dias	0.789398	0.789398	0.026507	0.316719	1.000000

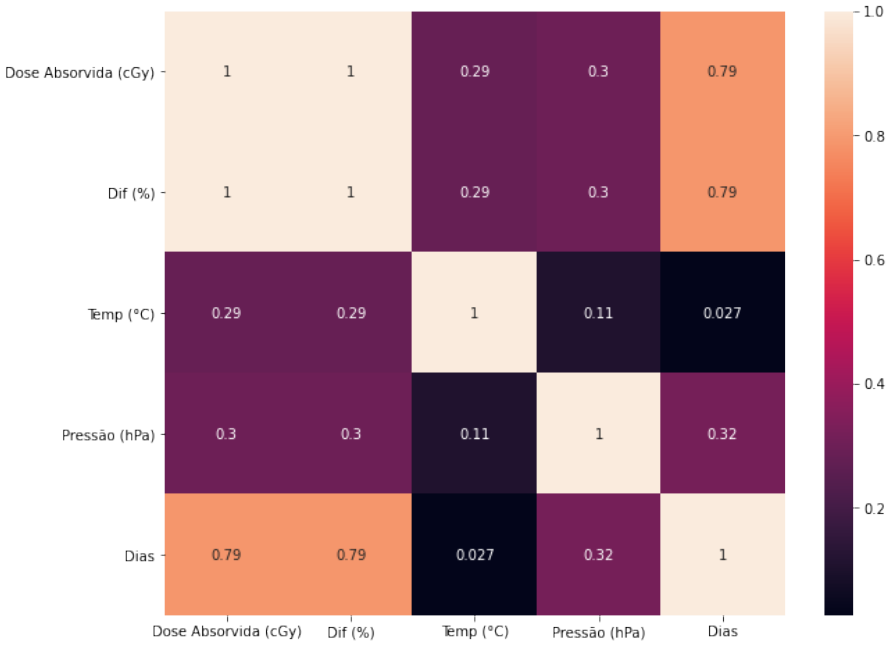
por padrão o método corr( ) utiliza o coeficiente de correlação de Pearson

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}.\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} = \frac{cov(X,Y)}{\sqrt{var(X).var(Y)}}$$

[16]:
```
df_project_hm = df_project.corr()
```

[17]:
```
#utilizando os dados de correlação para criar uma vizualização de mapa de calor
plt.figure(figsize= (10, 8))
sns.heatmap(df_project_hm, annot=True)
```

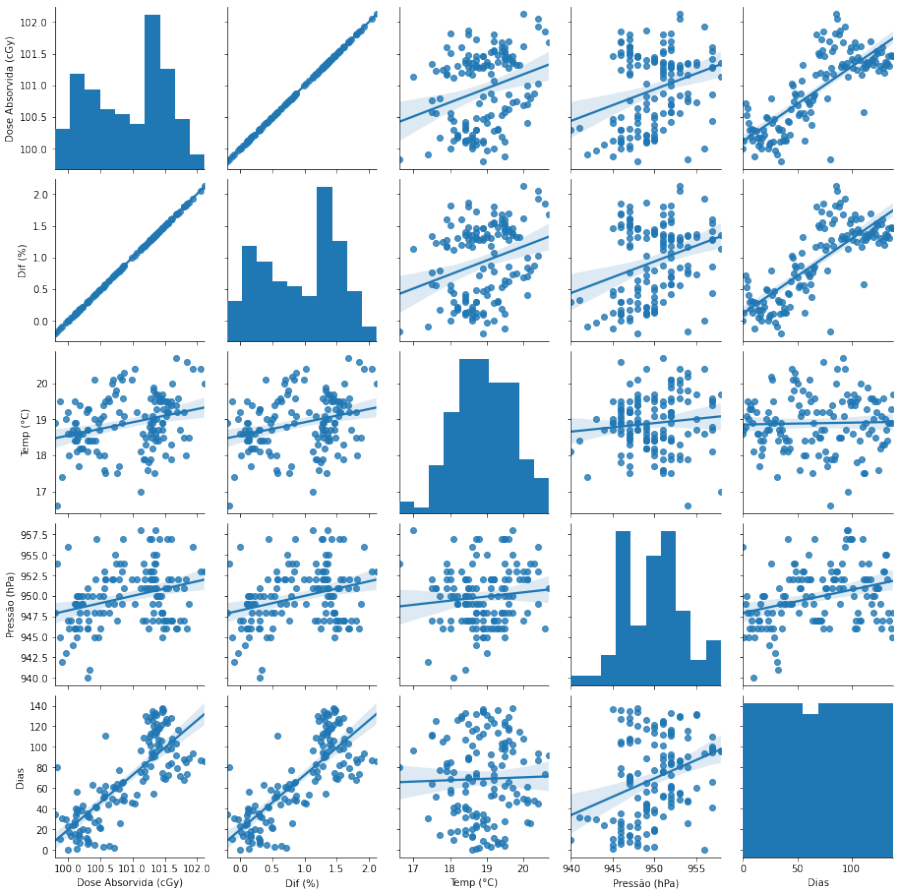
[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f812cb39f40>
```



1.5 5- Visualizações dos dados

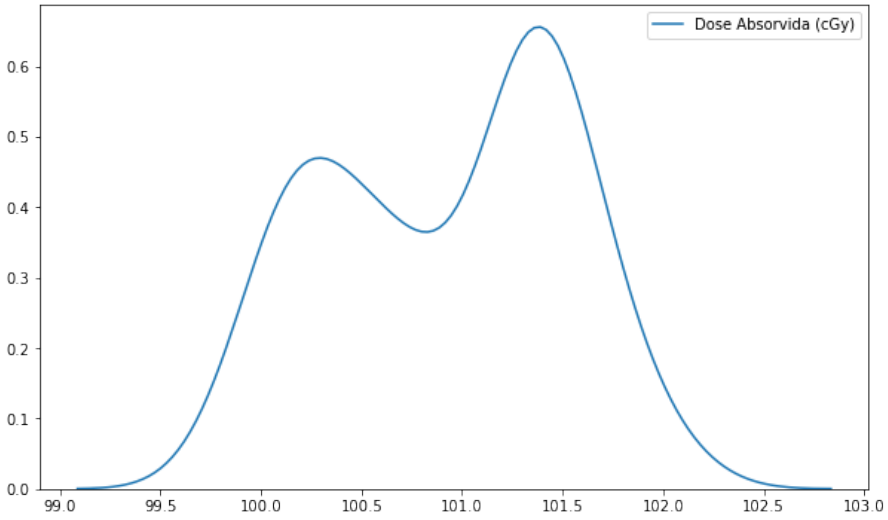
[18]:
```
#utilizando o Pairplot para visualizar de uma maneira geral o comportamento dos␣
↪dados entre si

sns.pairplot(df_project, kind= 'reg')
```

[18]: <seaborn.axisgrid.PairGrid at 0x7f812ce5ce80>



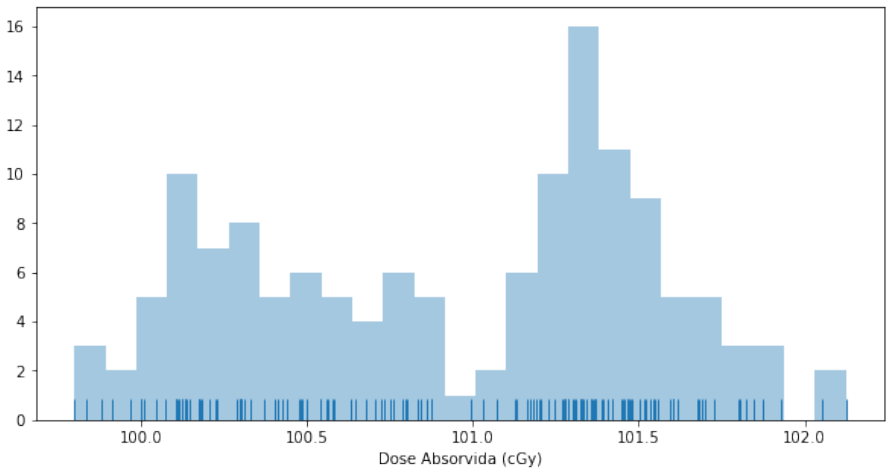
[19]:
```
#fazendo uma visualização da curva de probabilidades das doses absorvidas
plt.figure(figsize=(10, 6))
sns.kdeplot(df_project['Dose Absorvida (cGy)'])
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f812de91a90>



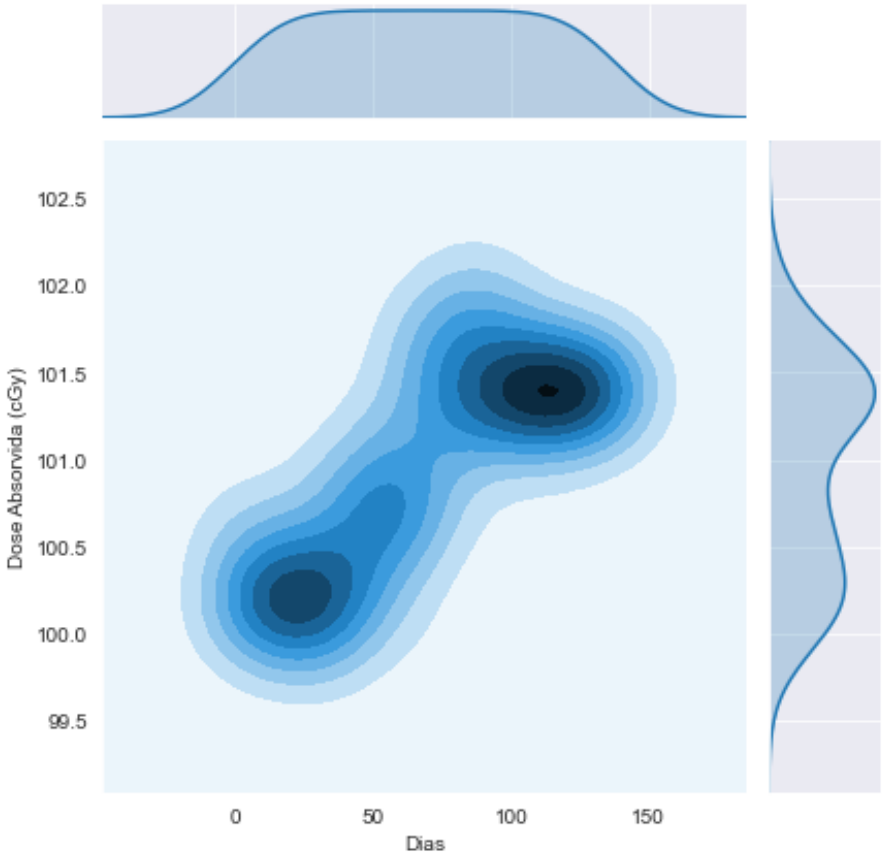
[20]:
```
#mais uma visualização de probabilidades utilizando histograma

plt.figure(figsize= (10,5))
sns.distplot(df_project['Dose Absorvida (cGy)'], bins= 25, kde= False, rug=␣
↪True)
```

[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f812e23efa0>



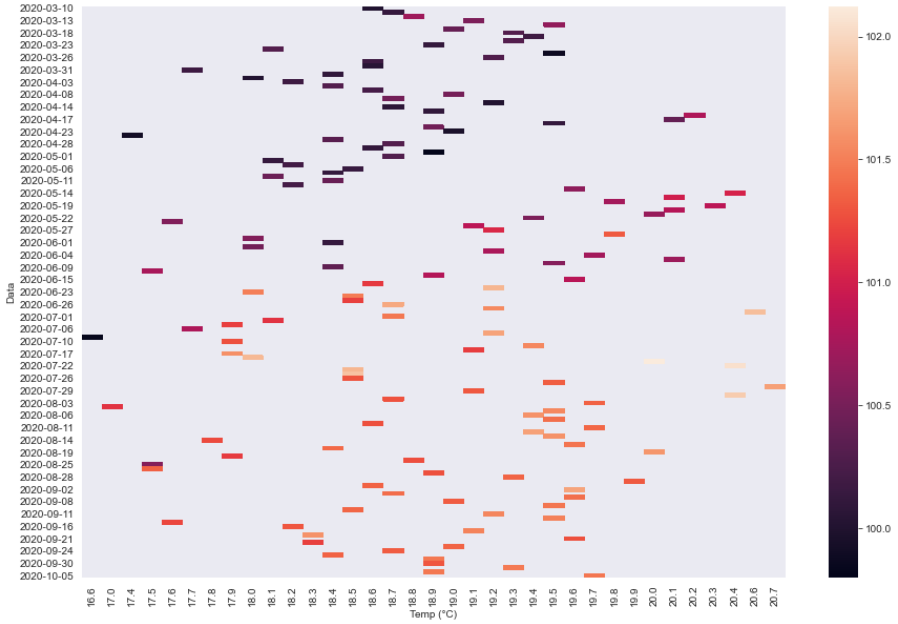
[21]:
```
#verificar relação entre dose absorvida com os dias corridos de uso do
↪equipamento

sns.set_style('darkgrid')
plt.figure(figsize=(10, 6))
sns.jointplot(y= 'Dose Absorvida (cGy)', x= 'Dias', data= df_project, kind =
↪'kde')
```

[21]:
```
<seaborn.axisgrid.JointGrid at 0x7f812e3be4f0>

<Figure size 720x432 with 0 Axes>
```



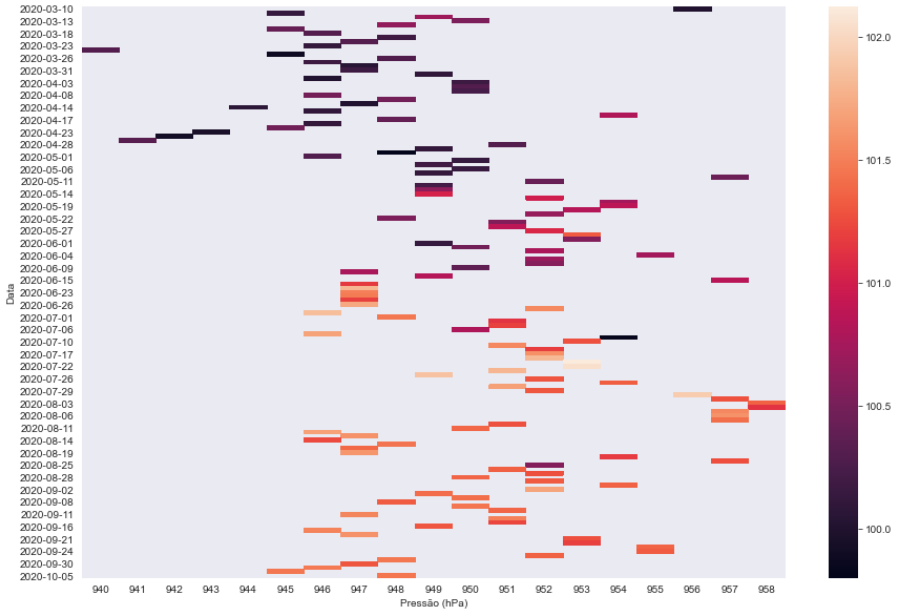
[22]:
```
#utilizando mapa de calor para analisar a relação entre dose absorvida, os dias␣
 ↪de utilização do equipamento
#e a temperatura

df_project_hm = df_project.pivot_table(index= 'Data', columns='Temp (°C)',␣
 ↪values= 'Dose Absorvida (cGy)')
plt.figure(figsize=(15,10))
sns.heatmap(df_project_hm)
```

[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f812ddce070>
```



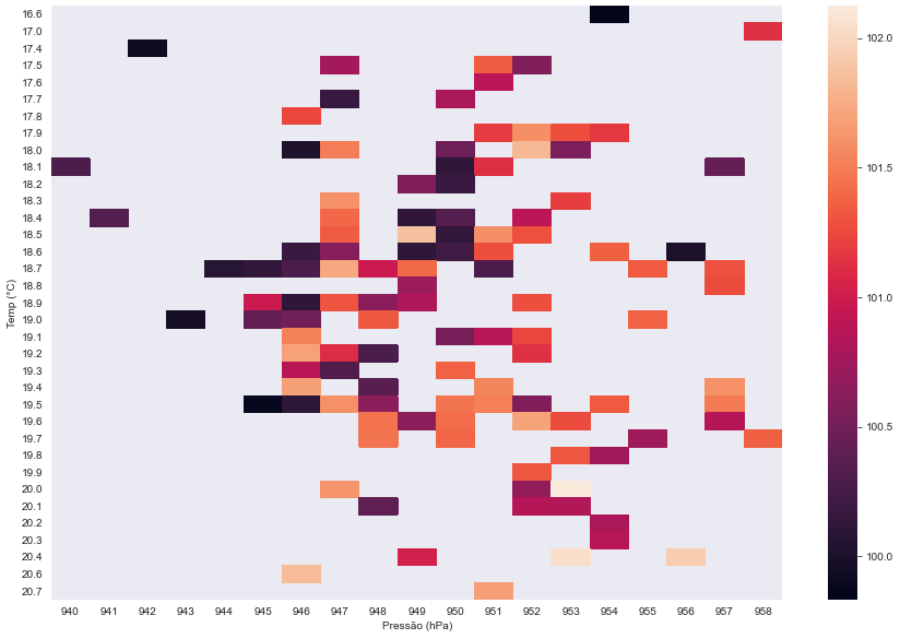
[23]:
```
#utilizando mapa de calor para analisar a relação entre dose absorvida, os dias␣
 ↪de utilização do equipamento
#e a pressão

df_project_hm = df_project.pivot_table(index= 'Data', columns='Pressão (hPa)',␣
 ↪values= 'Dose Absorvida (cGy)')
plt.figure(figsize=(15,10))
sns.heatmap(df_project_hm)
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f812d2b9970>
```



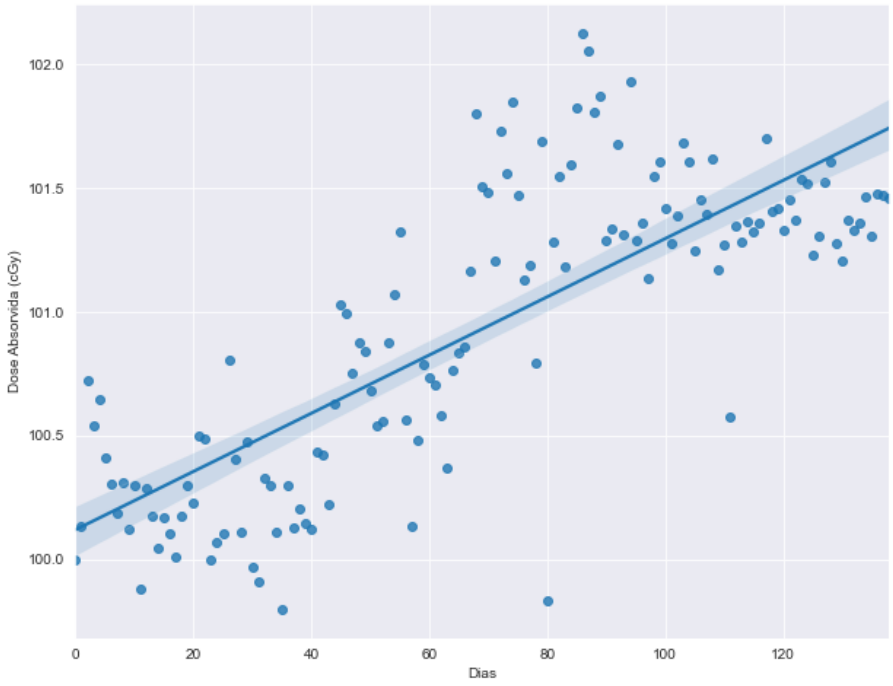
[24]:
```
df_project_hm = df_project.pivot_table(index= 'Temp (°C)', columns='Pressão␣
↪(hPa)', values= 'Dose Absorvida (cGy)')
plt.figure(figsize=(15,10))
sns.heatmap(df_project_hm)
```

[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f812cf2d190>
```



[25]:
```
#verificando o comportamento linear entre dose absorvida variando no tempo

plt.figure(figsize= (10, 8))
sns.regplot(y= 'Dose Absorvida (cGy)', x= 'Dias', data= df_project)
```

[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f812f7914f0>
```



1.6 6- Construção do modelo de predição

[26]:
```
#atribuindo os valores de X e y com os valores de dias corridos de uso do␣
↪equipamento e a dose absorvida

X = df_project.drop(['Dose Absorvida (cGy)', 'Dif (%)', 'Data','Temp (°C)',␣
↪'Pressão (hPa)'], axis=1)
y = df_project.pop('Dose Absorvida (cGy)')
```

[27]:
```
#da biblioteca do sklearn utilizamos a ferramenta que separa os dados para o␣
↪treino da regressão e teste para
#avaliar o quão bom está a estimativa

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3)
```

[28]:
```
#atribuinado o objeto LinearRegression() a variável lm

lm = LinearRegression()
```

[29]:
```
#treinando o modelo

lm.fit(X_train, y_train)
```

[29]:
```
LinearRegression()
```

[30]:
```
#utilizando o método predict para obter os valores de previsão baseado na␣
 ↪regressão já treinada, fornecendo os
#dados separados para o teste

pred = lm.predict(X_test)
pred
```

[30]:
```
array([100.93202943, 100.38096735, 100.42595201, 101.32564521,
       101.59555317, 101.42686069, 101.56181468, 101.44935302,
       100.17853638, 101.48309152, 100.66212147, 100.59464448,
       101.50558385, 101.24692205, 100.92078327, 100.65087531,
       101.37062987, 100.31349036, 100.2347672 , 100.74084463,
       100.9095371 , 101.58430701, 101.19069123, 101.53932235,
       100.54965982, 100.70710613, 101.03324492, 101.60679934,
       100.79707545, 101.71926099, 100.9432756 , 100.60589065,
       100.95452176, 100.75209079, 101.33689137, 101.31439904,
       100.56090599, 100.39221351, 101.62929167, 100.29099803,
       101.26941438, 101.17944506])
```

[31]:
```
#criando um dataframe com os dados reais separados para teste com a previsão␣
 ↪feita pelo modelo

dis = pd.DataFrame(y_test)
dis['pred'] = pred
```

[32]:
```
#vendo os dez primeiros valores do dataframe

dis.head(10)
```

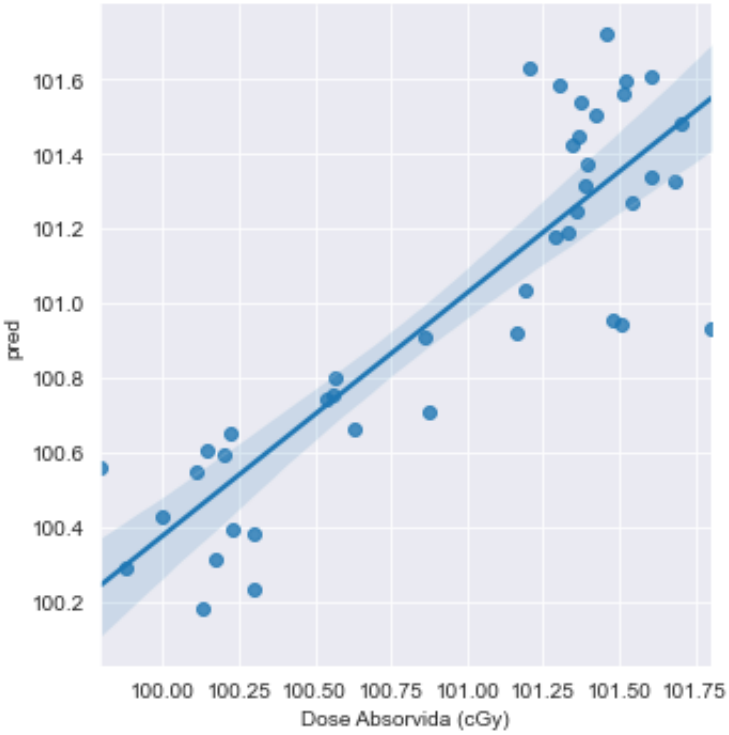
[32]:

	Dose Absorvida (cGy)	pred
68	101.8015	100.932029
19	100.2980	100.380967
23	100.0000	100.425952
103	101.6815	101.325645
127	101.5215	101.595553
112	101.3440	101.426861
124	101.5140	101.561815
114	101.3640	101.449353
1	100.1330	100.178536
117	101.6990	101.483092

[33]:
```
#visualizacão dos dados de previsão com os dados de teste
plt.figure(figsize= (10, 8))
sns.lmplot(x= 'Dose Absorvida (cGy)', y= 'pred', data= dis)
```

[33]: <seaborn.axisgrid.FacetGrid at 0x7f812fcd5cd0>

<Figure size 720x576 with 0 Axes>



[34]:
```
#calculando o erro quadrático médio

MSE = metrics.mean_squared_error(y_test, pred)
MSE
```

[34]: 0.09732219611959698

[35]:
```
#calculando o coeficiente de determinação r2 para ver o quão bom o modelo está␣
↪explicando os dados
```

```
r2 = metrics.r2_score(y_test,pred)
r2
```

[35]: 0.7343799948924876

[36]:
```
#definindo uma função que retorna a possível quantidade de dias que a máquina␣
↪será utilizada até a próxima
#calibração, utilizando o coeficiente angular e o coeficiente linear do modelo

def dia_de_calibração(dose):
    #devolve o valor possível da próxima calibração do equipamento em dias␣
↪corridos de utilização

    day = (dose - lm.intercept_)/lm.coef_
    return day
```

$$y = ax + b \rightarrow x = \frac{y - b}{a}$$

[37]:
```
#dias de uso estimado até a próxima calibração

dia_de_calibração(103.0000)
```

[37]: array([251.88228877])

[38]:
```
#criando listas para receber os valores dos erros médios, os coeficientes de␣
↪determinação r2, e os dias estimados
#de calibração

media_MSE = []
media_r2 = []
media_calibracao = []
```

[39]:
```
#como a ferramenta do split sempre pega valores aleatórios para treinar o␣
↪modelo, agora testaremos 1000 vezes
#para testar o modelo cada vez com valores diferentes

for simu in range(1000):
    lm = 0
    MSE = 0
    pred = 0
    r2 = 0
    dia = 0
    X_train = 0
    X_test= 0
```

```
    y_train = 0
    y_test =0



    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.3)
    lm = LinearRegression()
    lm.fit(X_train, y_train)
    pred = lm.predict(X_test)
    MSE = metrics.mean_squared_error(y_test, pred)
    r2 = metrics.r2_score(y_test,pred)
    dia = dia_de_calibração(103.0000).tolist()

    for x in dia:
        a = x


    media_MSE.append(MSE)
    media_r2.append(r2)
    media_calibracao.append(a)
```

[40]:
```
#criando um dicionário com os dados da simulação

valores = {'MSE': media_MSE,
           'R2': media_r2,
           'Dia de calibração': media_calibracao}
```

[41]:
```
#criando um datadrame a partir do dicionário

valores_df = pd.DataFrame(valores)
valores_df
```

[41]:

	MSE	R2	Dia de calibração
0	0.114067	0.670080	251.558807
1	0.125008	0.625252	239.187775
2	0.147358	0.535827	237.777026
3	0.091190	0.660141	244.061065
4	0.175190	0.532027	248.370154
..	…	…	…
995	0.108067	0.695320	244.208708
996	0.163441	0.605363	258.699175
997	0.130756	0.625766	248.657063
998	0.077201	0.722135	241.442447
999	0.164709	0.511049	237.905461

[1000 rows x 3 columns]

[42]:
```
#estatística sobre a simulação

valores_df.describe()
```

[42]:

	MSE	R2	Dia de calibração
count	1000.000000	1000.000000	1000.000000
mean	0.138922	0.603598	245.475801
std	0.027404	0.067038	6.256195
min	0.058920	0.346660	225.360793
25%	0.119949	0.557862	241.195932
50%	0.137767	0.605078	245.698456
75%	0.157854	0.651055	249.833567
max	0.230964	0.791318	262.403941

[ ]:

Referências Bibliográficas

[1] E. Okuno and E. M. Yoshimura, *Física das radiações*. Oficina de Textos, 2016.

[2] E. Okuno, *Radiação: efeitos, riscos e benefícios*. Oficina de Textos, 2018.

[3] M. E. C. DEYLLOT, *Física das Radiações: Fundamentos e Construção de Imagens*. Saraiva Educação S.A.

[4] A. C. M. Christovam and O. Machado, *Manual de física e proteção radiológica*. Difusão Editora, 2018.

[5] F. H. Attix, *Introduction to Radiological Physics and Radiation Dosimetry*. Wiley, 2008.

[6] Python Software Fundation. Disponível em <https://www.python.org>.

[7] E. Matthes, *Curso Intensivo de Python: Uma introdução prática e baseada em projetos à programação*. Novatec Editora, 2017.

[8] W. McKinney, *Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython*. Novatec Editora, 2019.

[9] Anaconda Software Distribution. Computer software. Vers. 2020.07. Anaconda, Nov. 2016. Web. <https://anaconda.com>.

[10] Project Jupyter. Disponível em <https://jupyter.org>.

[11] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), pp. 56 – 61, 2010.

[12] Pandas. Disponível em <https://pandas.pydata.org>.

[13] Numpy Project. Disponível em <https://numpy.org>.

[14] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, sep 2020.

[15] Matplotlib. Disponível em <https://matplotlib.org>.

[16] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[17] Seaborn. Disponível em <https://seaborn.pydata.org>.

[18] M. Waskom and the seaborn development team, "mwaskom/seaborn," Sept. 2020.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] Scikit-learn. Disponível em <https://scikit-learn.org>.