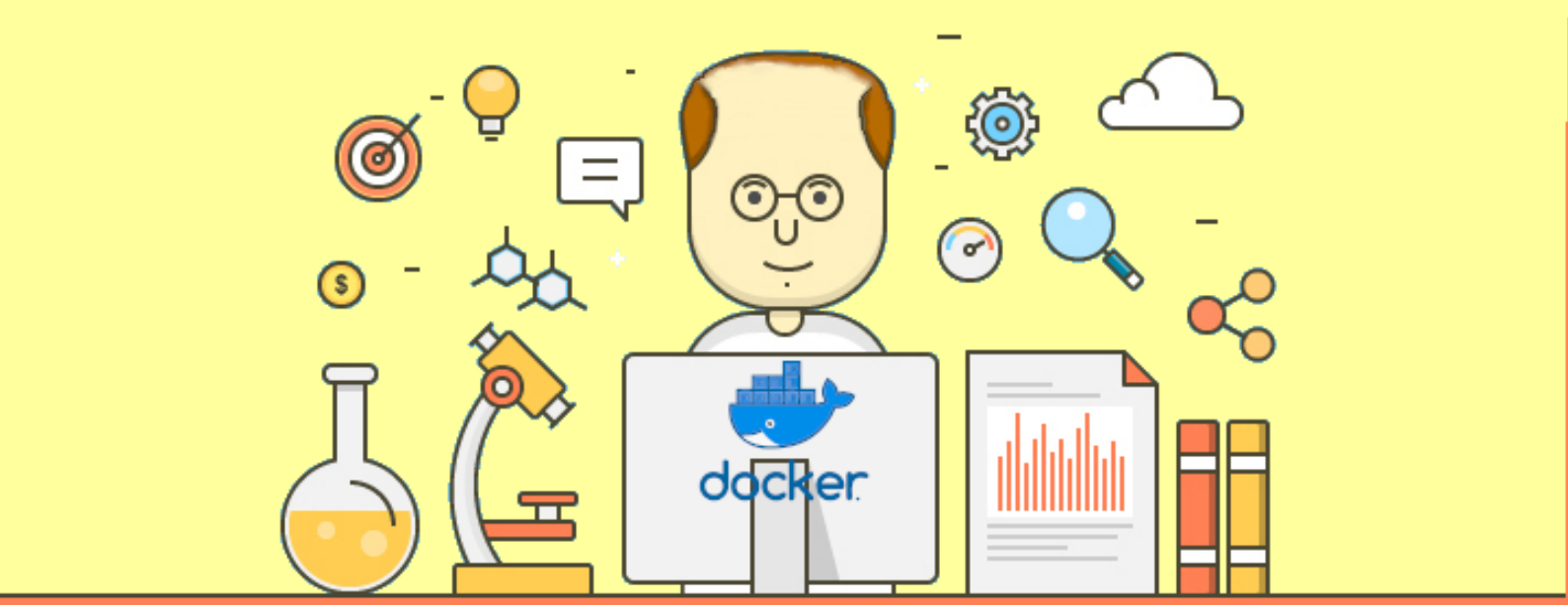
Machine Learning na Prática

Modelos em Python

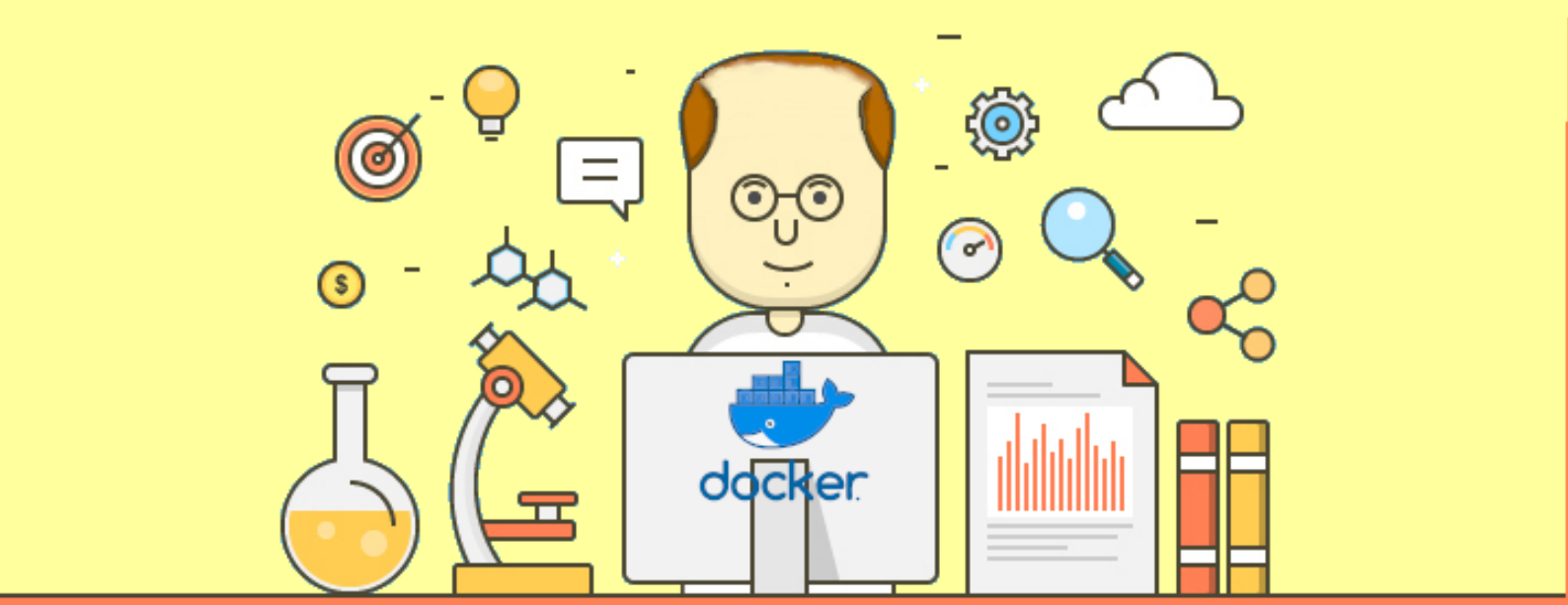
Fernando Anselmo

PUBLICAÇÃO INDEPENDENTE

http:\fernandoanselmo.orgfree.com



Sumário



1. Entendimento Geral

> Machine Learning é a habilidade de localizar padrões em Dados. (Gil Weinberg - Founding Director of Georgia Tech Center for Music Technology)

1.1 Do que trata esse livro?

Cada vez que leio um livro de **Machine Learning** tenho a sensação que o autor quer mostrar para seus leitores o quanto ele é um bom Matemático, coloca um monte de fórmulas com demonstrações (muitas delas parecem escritas em grego e usam inclusive letras gregas) é isso me faz pensar: *Será que quando indicar um livro para meus alunos vou querer que eles aprendam fórmulas ou que tenham uma base em que possam praticar?*

E graças a isso publiquei uma série de artigos sobre os mais variados modelos de *Machine Learning* na rede **Linkedin** e esses artigos foram a base para esse livro, não espere encontrar aqui muitos conceitos, apesar de ser obrigado a abordá-los ou muita coisa se perderia, mas a ideia aqui é ser totalmente prático.

Já ouvimos isso de prático tantas vezes, existe duas séries de livros especialistas denominadas: "*Hands On*"e "*Cookbook*". Aprecio e tenho muitos desses livros, porém sempre que desejo algo no primeiro caso é muito difícil encontrar bem separado e exposto da forma como queria e no segundo está fragmentado demais. O prático aqui será: Um modelo em linguagem Python, ler bases de dados, com o uso de suas possibilidades, seu treino e melhor forma de obter resultados. Sendo assim não espere encontrar nesse aulas básicas de Python.

1.2 O que é Machine Learning?

De forma bastante genérica, algorítimos de *Machine Learning* (Aprendizado de Máquina e doravante apenas **ML**) são uma mudança de paradigma da "programação tradicional" onde precisamos passar toda a heurística explicitamente para um novo conceito, onde ao invés de escrever cada ação que o algorítimos deve realizar, apenas passamos diversos exemplos e deixamos que o computador resolva (ou aprenda) quais são as "melhores" (menor custo) decisões. É também chamada de *Statistical Learning* (Autores Hastie, Tibshirani & Friedman 2009) utilizada para extrair um modelo a partir de um sistema de observações ou medidas. Sendo um campo relativamente novo da ciência composto de uma variedade de métodos (algoritmos) computacionais e estatísticos que competem entre si.

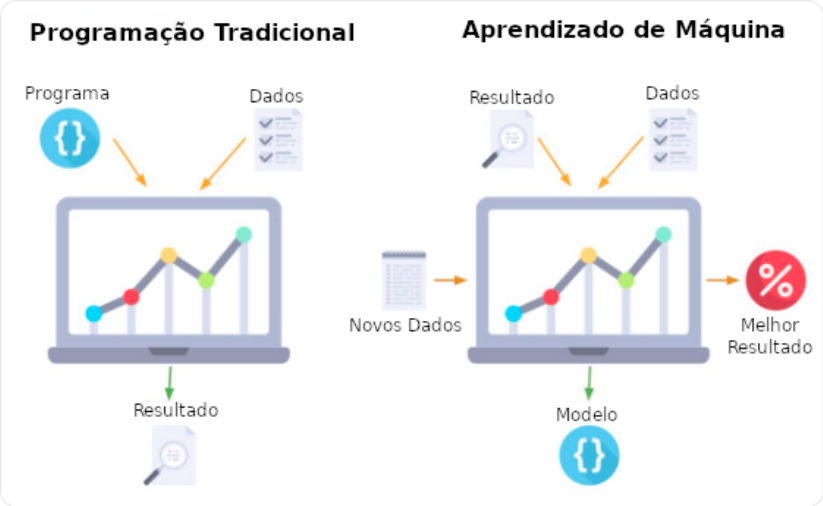


Figura 1.1: Programação Tradicional X Machine Learning

Ou seja, treinamos com dados e resultados anteriores até que possam receber **novos dados** e transforme isso em resultados sem que sejam explicitamente programáveis, isso é chamado de "Modelo de ML". Interpretar esses modelos é entender como podemos transformar algo. Em geral são classificados em:

- **Clusterização**: Encontrar uma estrutura de dados, sumarização.
- **Regressão e Estimação**: Predição de valores contínuos.
- **Classificação**: Predição de um item de um caso de classe/categoria.
- **Associação**: frequentemente ocorre entre itens e eventos.

É muito importante conhecer vários deles para decidir qual se ajusta e trabalha melhor aos dados que temos para treiná-los. Devemos ter em mente que ML pode ser bem diferente de **Estatística**. Aqui não estamos preocupados com inferência, causalidade e exogeneidade. ML está preocupada, quase que exclusivamente em melhorar as predições ou rapidamente localizar em um mar de informações aquela que buscamos.

Assim podemos pegar a mesma função, por exemplo **Regressão Logística** e analisar do ponto de vista da estatística, que interpretaria se os "betas"são significativos, se os "resíduos"têm uma distribuição normal ou analisar isso do ponto de vista de ML e descobrir como está a relação entre **Precisão** e **Recall**, ou qual a ROC ou AUC do modelo[1].

[1] As curvas ROC e AUC estão entre as métricas mais utilizadas para a avaliação de um modelo de Machine Learning.

1.3 Formas de Aprendizado

Quanto as formas de aprendizado se dividem em:

- **Não Supervisionado**, corresponde a um vetor de observações que é utilizado para observar padrões, tendências, verificar estruturas e descobrir relações.
- **Supervisionado**, além do vetor de observações, existe também uma resposta associada a cada questão.
- **Aprendizagem por Reforço**, uma ação ocorre e as consequências são observadas, assim a próxima ação considera os resultados da primeira ação. É um algoritmo dinâmico que parte do princípio "tentativa e erro".

Entender a diferença entre os dois tipos é bem simples, enviamos ao computador uma série de imagens sobre pratos de comida e não informamos absolutamente nada sobre elas, o máximo que acontecerá é a separação dessas em grupos similares. Em seguida mostremos essa imagem para o computador:

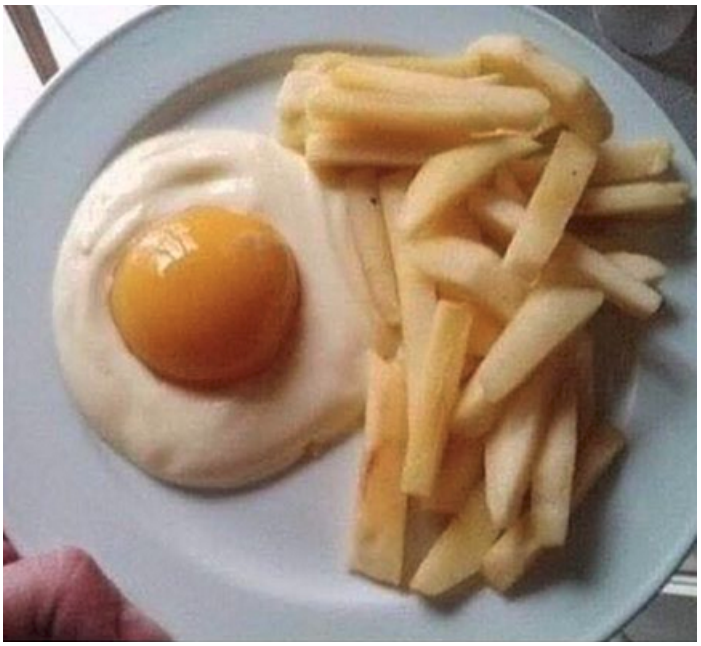


Figura 1.2: Nova informação

Qual será o prato de comida que provavelmente ele associará? Exatamente "Ovo frito e batatas fritas"[2]. No segundo tipo de aprendizado além das imagens dizemos o que cada uma vem a ser, porém ao mostrar a mesma imagem não pense que o computador saberá classificá-la corretamente pois provavelmente se confundirá mais uma vez (assim como você deve ter se confundido).

A resolução com problemas de imagem é bem complicada[3], pois conseguimos identificar a diferença devido a nossa experiência não apenas visual mas com informações sobre textura, forma e outras que o computador ainda está engatinhando. Por isso o estudo desses algorítimos é algo tão rico e como vimos anteriormente. É muito importante conhecer boa parte dos modelos para decidir qual se ajusta e trabalha melhor aos dados que temos para treiná-los.

[2] Apesar que se olhar mais de perto verá que isso é iogurte, meio pêssego e tiras de maçã
[3] Assim como análise da linguagem, chamada de NLP - *Natural Language Processing*

1.3.1 Algoritmo Não Supervisionado

Não envolvem um controle direto, aqui o ponto principal do requisito são desconhecidos e ainda precisam ser definidos. Normalmente são usados para: explorar uma estrutura de informação; extrair informações desconhecidas sobre os dados e observar a detecção de padrões.

k-means: é o mais popular, usado para segmentar em categorias exclusivas com base em uma variedade de recursos, por exemplo clientes como seus hábitos de consumo ou casas com seu preço, localidade e área.

t-Distributed Stochastic Neighbor Embedding: t-SNE é utilizado para redução de dimensionalidade que é particularmente adequada para a visualização de conjuntos que possuem dados com alta dimensão.

Principal Component Analysis: PCA é usado para enfatizar a variação e destacar padrões fortes em um conjunto de dados, de modo a facilitar a exploração e visualização de dados.

Associate Rule Mining: ARM para encontrar padrões frequentes, associações, relações e correlação, normalmente usado para fazer análises de cesta de compras. Em termos gerais, é aplicado em várias situações para encontrar associação, padrão frequente nos conjuntos de objetos nos bancos de dados.

1.3.2 Algoritmos Supervisionados

Neste tipo temos dados de valores conhecidos e do ponto de vista da máquina, esse processo é uma rotina de "conectar os pontos"ou achar similaridades entre os dados. Para "alimentar"o algoritmo determinamos que tipo de resultado é desejado ("sim/não", "verdadeiro/falso", a projeção do valor das vendas, a perda líquida de crédito ou o preço da habitação).

Regressão linear: muito utilizado para prever resultados numéricos contínuos, como preços de casas ou ações, umidade ou temperatura de um local, crescimento populacional.

Regressão logística: É um classificador popular usado especialmente no setor de crédito para prever inadimplências de empréstimos.

k-Nearest Neighbors: k-NN é um algoritmo usado para classificar dados em duas ou mais categorias e é amplamente usado para classificar casas em grupos como caras e acessíveis, com base em preço, área, quartos e toda uma gama de outros recursos.

Support Vector Machines: SVM é um classificador popular usado na detecção de imagens e faces, além de aplicativos como reconhecimento de manuscrito.

Tree-Based Algorithms: Algoritmos baseados em árvores, como florestas aleatórias ou árvores de decisão ou reforçadas, são usados para resolver problemas de classificação e regressão.

Naive Bayes: Este classificador usa o modelo matemático de probabilidade para resolver problemas de classificação.

1.3.3 Algoritmos de Aprendizagem por Reforço

Tratam de situações onde a máquina começa a aprender por tentativa e erro ao atuar sobre um ambiente dinâmico. Desta maneira, não é necessário novos exemplos ou um modelo a respeito da tarefa a ser executada: a única fonte de aprendizado é a própria experiência do agente, cujo objetivo formal é adquirir uma política de ações que maximize seu desempenho geral.

Q-Learning: é um algoritmo de aprendizado baseado em valores. Os algoritmos baseados em valor atualizam uma função com base em uma equação (particularmente neste caso a equação de *Bellman*). Enquanto o outro tipo, baseado em políticas, estima a função de valor com uma política gananciosa obtida a partir do último aprimoramento da política. Q-learning é um aluno fora da política. Significa que aprende o valor da política ideal, independentemente das ações do agente. Por outro lado, um aprendiz de política aprende o valor da política que está sendo executada pelo agente, incluindo as etapas de exploração e encontrará uma política ideal, levando em consideração a exploração inerente à política.

Temporal Difference: TD é um agente que aprende em um ambiente por meio de episódios sem conhecimento prévio desse ambiente. Isso significa que a diferença temporal adota uma abordagem de aprendizado sem modelo ou sem supervisão. Podemos considerar isso como uma tentativa e erro.

Monte-Carlo Tree Search: MCTS é um método geralmente usado nos jogos para prever o caminho (movimentos) que a política deve seguir para alcançar a solução final vencedora. Jogos como Cubo de Rubik, Sudoku, Xadrez, Go, ou um simples Jogo da Velha têm muitas propriedades em comuns que levam ao aumento exponencial do número de possíveis ações que podem ser executadas. Esses passos aumentam exponencialmente à medida que o jogo avança. Idealmente, podemos prever todos os movimentos possíveis e seus resultados que podem ocorrer no futuro e assim aumentarmos a chance de ganhar.

Asynchronous Advantage Actor-Critic: A3C é um dos algoritmos mais recentes a serem desenvolvidos no campo de aprendizado de reforço profundo. Esse algoritmo foi desenvolvido pelo *DeepMind* do Google. Implementa treinamento no qual vários *workers*, em ambiente paralelo, atualizam independentemente uma função de valor global - portanto "assíncrona". Um dos principais benefícios de ter atores assíncronos é a exploração eficaz e eficiente do espaço de estados.

1.4 Montagem do Ambiente

Vemos atualmente uma grande revolução em torno de *Data Science* (falamos em inglês para parecer algo chique) principalmente em torno as ferramentas que tem se atualizado a uma velocidade assustadora. Porém, essas atualizações constantes muitas vezes carregam problemas que podem afetar o seu Sistema Operacional. A pergunta é: Como ficar atualizado e seguro ao mesmo tempo? A única resposta coerente que consegui encontrar foi: Usar a conteinerização para resolver o problema.

Então o ideal é partir atrás de imagens prontas, visitar sites como HubDocker (`https://hub.docker.com`) e rezar para encontrar a imagem que nos atenda ou fazer algo melhor e personalizar a imagem.

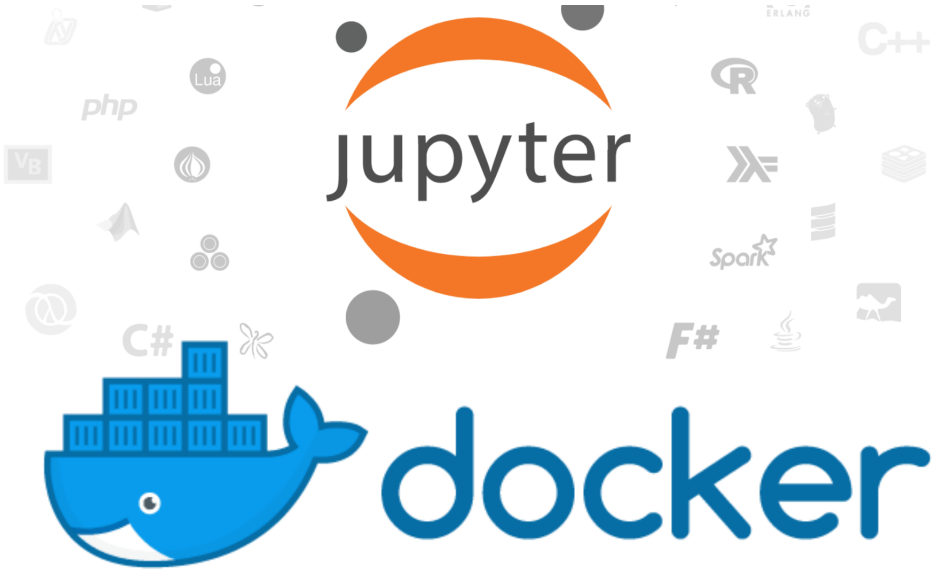


Figura 1.3: Docker e Jupyter para Data Science

Essa segunda alternativa é bem mais próxima a realidade de qualquer um que deseje trabalhar de modo efetivo com Ciência de Dados. A personalização de uma imagem no Docker não é um bicho de sete cabeças (no qual a partir do momento que cortamos uma das cabeças nasce mais duas, então sempre imaginei que seriam mais do que sete) mas algo que pode ser facilmente aprendido por qualquer um que "ao menos"saiba usar o terminal de comando do Linux.

> **Dica 1.1: Não sabe nem por onde começar com o Docker?.** Não se desespere, baixe um paper sobre o Docker gratuitamente na minha página no Academia.edu (`https://iesbpreve.academia.edu/FernandoAnselmo`).

Minha primeira personalização de Imagem[4] surgiu quando descobri que as versões **Jupyter** não me atendiam por completo e **Anaconda** era grande demais. Iremos então trabalhar em uma imagem que pussa conter um Jupyter mais adequado e uma Anaconda mas controlada. O resultado foi o seguinte **Dockerfile** (que obrigatoriamente deve ter esse nome de arquivo):

```
# Base da Imagem
FROM ubuntu:19.10

# Adiciona o metadata para a imagem com o par: chave,valor
LABEL maintainer="Fernando Anselmo <fernando.anselmo74@gmail.com>"
LABEL version="1.2"

# Variaveis de Ambiente
ENV LANG=C.UTF-8 LC_ALL=C.UTF-8 PATH=/opt/conda/bin:$PATH

# Execucoes iniciais:
# Cria a pasta de ligacao
RUN mkdir ~/GitProjects && \
# instala os pacotes necessarios
```

[4]O objetivo não é termos uma imagem pequena, mas uma PERSONALIZÁVEL, se deseja que seja pequena recomendo que acesse o tutorial disponivel em `https://jcrist.github.io/conda-docker-tips.html`.

```
apt-get update && apt-get install --no-install-recommends --yes python3 && \
apt-get install -y wget ca-certificates git-core pkg-config tree freetds-dev
    apt-utils && \
# Limpeza basica
apt-get autoclean -y && \
rm -rf /var/lib/apt/lists/* && \
# Configuracao do Jupyter
mkdir ~/.ssh && touch ~/.ssh/known_hosts && \
ssh-keygen -F github.com || ssh-keyscan github.com >> ~/.ssh/known_hosts && \
git clone https://github.com/bobbywlindsey/dotfiles.git && \
mkdir ~/.jupyter && \
mkdir -p ~/.jupyter/custom && \
mkdir -p ~/.jupyter/nbconfig && \
cp /dotfiles/jupyter/jupyter_notebook_config.py ~/.jupyter/ && \
cp /dotfiles/jupyter/custom/custom.js ~/.jupyter/custom/ && \
cp /dotfiles/jupyter/nbconfig/notebook.json ~/.jupyter/nbconfig/ && \
rm -rf /dotfiles && \
# Instalar o Anaconda
echo 'export PATH=/opt/conda/bin:$PATH' > /etc/profile.d/conda.sh && \
wget --quiet https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh -O
    ~/anaconda.sh && \
/bin/bash ~/anaconda.sh -b -p /opt/conda && \
rm ~/anaconda.sh && \
conda uninstall anaconda-navigator && \
conda update conda && \
conda update anaconda && \
conda install nodejs && \
conda update --all && \
# Limpeza basica no Anaconda
find /opt/conda/ -follow -type f -name '*.a' -delete && \
find /opt/conda/ -follow -type f -name '*.pyc' -delete && \
find /opt/conda/ -follow -type f -name '*.js.map' -delete && \
find /opt/conda/lib/python*/site-packages/bokeh/server/static -follow -type f -name
    '*.js' ! -name '*.min.js' -delete &&  \
# Instalar os temas para o Jupyter
pip install msgpack jupyterthemes && \
jt -t grade3 && \
# Instalar os pacotes
conda install scipy && \
conda install pymssql mkl=2018 && \
pip install SQLAlchemy missingno json_tricks \
gensim elasticsearch psycopg2-binary \
jupyter_contrib_nbextensions mysql-connector-python \
jupyter_nbextensions_configurator pymc3 apyori && \
# Habilitar as extensoes do Jupyter Notebook
jupyter contrib nbextension install --user && \
jupyter nbextensions_configurator enable --user && \
jupyter nbextension enable codefolding/main && \
jupyter nbextension enable collapsible_headings/main && \
# Adicionar a extensao do vim-binding
mkdir -p $(jupyter --data-dir)/nbextensions && \
git clone https://github.com/lambdalisue/jupyter-vim-binding $(jupyter
    --data-dir)/nbextensions/vim_binding && \
cd $(jupyter --data-dir)/nbextensions \
chmod -R go-w vim_binding && \
# Remover o que nao eh necessario
```

```
conda clean -afy && \
apt-get remove -y wget git-core pkg-config && \
apt-get autoremove -y && apt-get autoclean -y && \
# Adicionar o Git ao Jupyter Lab
pip install --upgrade jupyterlab-git && \
jupyter lab build

# Configurar o acesso ao Jupyter
WORKDIR /root/GitProjects
EXPOSE 8888
CMD jupyter lab --no-browser --ip=0.0.0.0 --allow-root
    --NotebookApp.token='data-science'
```

Tento manter sempre o script o mais documentado possível deste modo posso remover ou adicionar propriedades sem me incomodar muito. Para criarmos a imagem é muito simples, supondo que a localização do script esteja em uma pasta chamada docker-data-science, então na pasta anterior digitar o comando:
`$ docker build -t fernandoanselmo/docker-data-science docker-data-science`

Obviamente que "fernandoanselmo/docker-data-science"pode ser alterado para o nome que desejar, porém na essência isso cria uma imagem que contém além do sistema operacional uma versão completa do Jupyter (com a inclusão de várias funcionalidades) para a realização do nosso trabalho como Cientista de Dados.

Mas isso é muito complicado e não entendo nada disso! Sem problemas, basta saltar toda essa parte de criação e construção da imagem para o próximo tópico seguindo comando a comando, pois este foi publicado do **DockerHub** e será baixado sem problemas.

Agora o comando:
`$ docker run -d -t -i -v --privileged /dev/ttyACM0:/dev/ttyACM0 -v ~/Aplicativos/ipynb:/root/GitProjects --network=host --name meu-jupyter fernandoanselmo/docker-data-science`

Realiza a criação de um contêiner. Vamos aos detalhes, após a opção -v aparece a expressão "~/Aplicativos/ipynb", essa se refere a uma determinada pasta no sistema operacional onde serão armazenados os Notebooks produzidos. Abrir o navegador no endereço `http://localhost:8888` e obtemos o seguinte resultado:

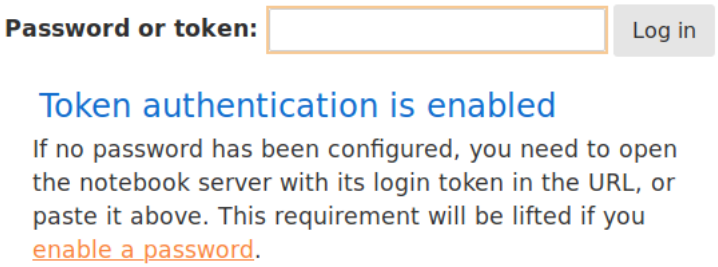


Figura 1.4: Jupyter solicitando o Token

A senha do token está definida na última linha do Script como "data-science", após informá-la o jupyter está pronto para trabalharmos:

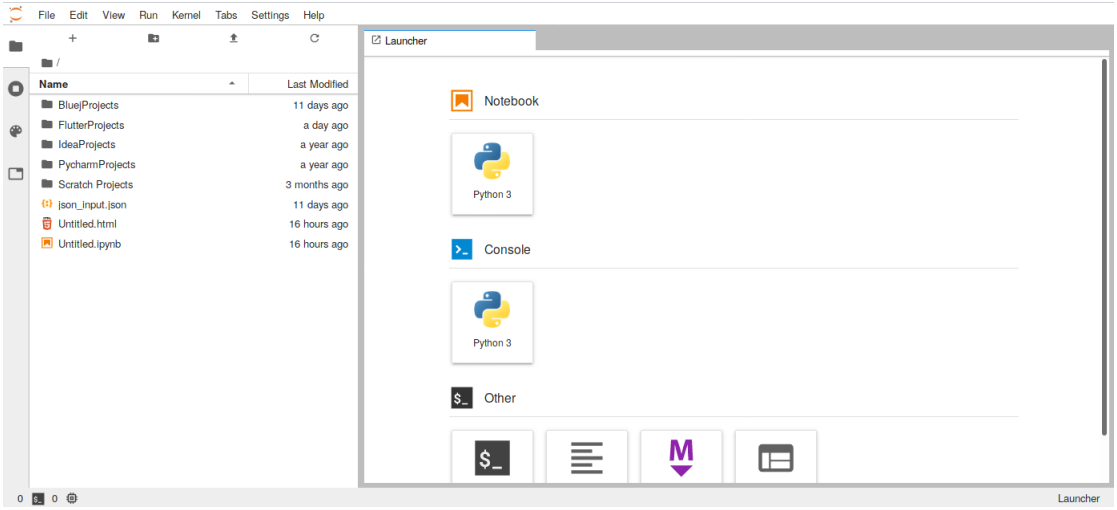


Figura 1.5: Jupyter Lab pronto

A maior vantagem que agora possuímos um ambiente com o **Jupyter Lab** completamente controlado incluindo temas e sincronização com o **Github**. Os próximos comandos são bem mais simples, tais como:

```
$ docker stop meu-jupyter
$ docker start meu-jupyter
```

Respectivamente para encerrar e iniciar o contêiner. Oh não! Agora estou preso, não posso mais fazer atualizações. Devemos entender que os contêineres são **dinâmicos**. Precisamos instalar o Keras/TensorFlow, acessar o contêiner (com ele já iniciado):

```
$ docker exec -it meu-jupyter /bin/bash
```

E instalamos normalmente, como se estivesse em uma máquina com o Ubuntu (com os poderes de superusuário):

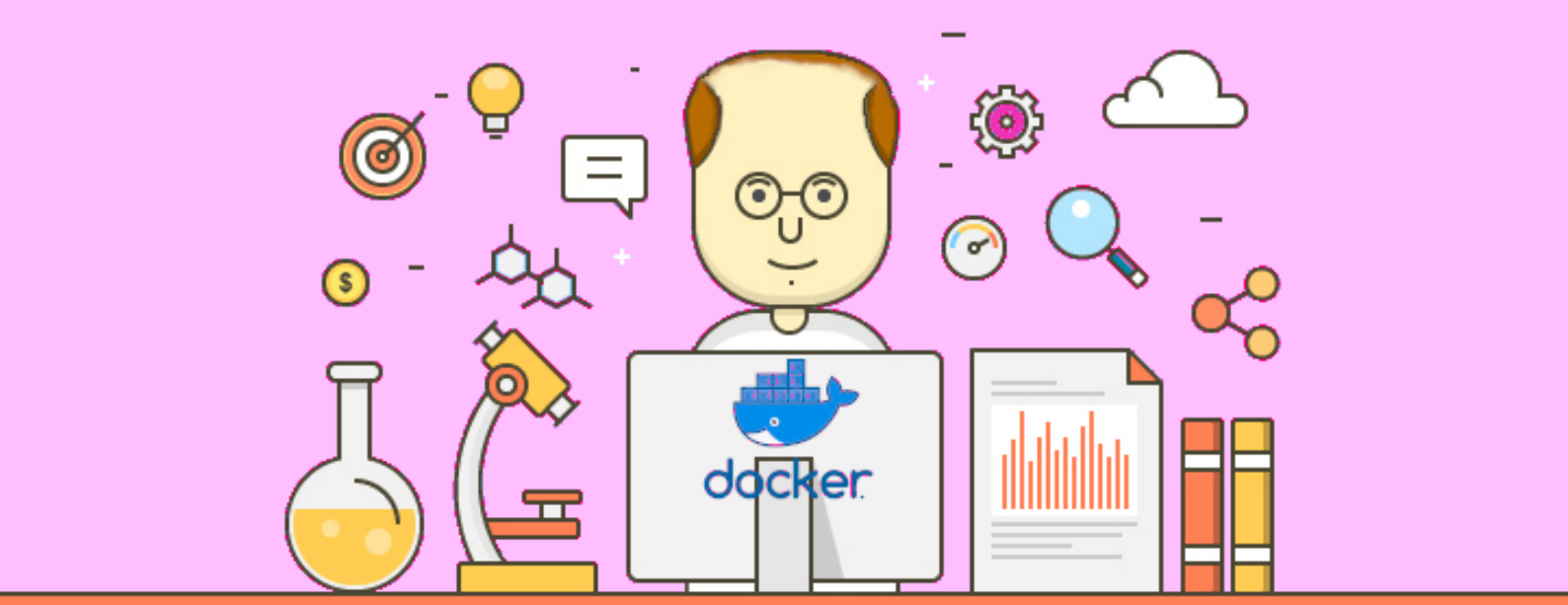
```
# conda install -c conda-forge keras
```

Pronto uma vez testado podemos optar por manter só nesse contêiner, ou então modificar o Script para ao criarmos novos contêineres e estes serão criados com essa funcionalidade embutida.

Verificar qual a versão do Python que está a nossa disposição, em uma célula do Notebook digite:

```
1 !python --version
```

Ao pressionarmos Ctrl+Enter será mostrada a versão 3.7.7. Agora podemos testar e executar qualquer ferramental sem nos preocuparmos em corromper a máquina. Talvez, no máximo, perdermos um contêiner.



2. Conceitos Introdutórios

A melhor forma de prever o futuro é cria-lo. (Peter Drucker - Escritor e Pai da Administração Moderna)

2.1 Termos da Estatística

Devemos sempre possuir um vocabulário comum, e mesmo que não se entenda absolutamente nada de estatística (e tenha um Estatístico a sua disposição), o Cientista de Dados deve conhecer os seguintes termos:

- **População**: o conjunto constituído por todos os indivíduos que representam pelo menos uma característica comum, cujo comportamento interessa analisar (inferir). Por exemplo, se em uma empresa o diretor gostaria de saber se os funcionários estão satisfeitos com os benefícios oferecidos, a população de estudo são todos os funcionários dessa empresa. O conceito de população depende do objetivo do estudo.
- **Amostra**: um subconjunto, uma parte selecionada da totalidade de observações abrangidas pela população, através da qual se faz inferência sobre as características da população. Por exemplo, uma rádio tem o interesse de saber como está sua audiência com os ouvintes no trânsito. Não é possível perguntar a todos os motoristas que ouvem rádio qual é aquela que eles preferem. Então buscamos uma parte representativa dessa população, isto significa, perguntar somente a alguns motoristas qual rádio preferem escutar enquanto dirigem. Uma amostra tem que ser representativa, sua coleta bem como seu manuseio requer cuidados especiais para que os resultados não sejam distorcidos.
- **Elemento ou Variável de Interesse**: característica a ser observada em cada indivíduo da amostra. Componentes sobre o qual serão observadas ou medidas as características. Onde cada característica

corresponde a um tipo de variável. Por exemplo, se queremos estudar o índice de massa corporal (IMC) de alunos do ensino médio de uma cidade, tomaremos uma amostra dessa população, e mediremos a altura e o peso de cada aluno, já que o IMC é calculado como uma razão entre o peso e o quadrado da altura do indivíduo. Nesse caso, peso e altura são as variáveis de interesse.

Tendo em vista as dificuldades de várias naturezas para se observar todos os elementos da população, tomaremos alguns deles para formar um grupo a ser estudado. Este subconjunto da população, em geral com dimensão menor, é denominado **amostra**.

Outros termos de interessante de conhecimento são as "Medidas Resumo", são elas:

- **Média**: É a soma das observações dividida pelo total. Este é o mais comum indicador de uma tendência central de uma variável.
- **Mediana**: Em uma lista ordenada de dados (rol), a mediana é o termo central.
- **Moda**: Refere-se ao termo que mais aparece em uma coleção. Sendo que: Amodal - rol que não tem nenhum valor que se repete; Bimodal - existem 2 valores que se repetem na mesma frequência, N-modal - existem n valores que se repetem na mesma frequência.
- **Variância**: Medida de dispersão dos dados para uma média. Quanto maior for mais distantes da média estão, e quanto menor for mais próximos estão da média.
- **Desvio Padrão (std)**: É o resultado positivo da raiz quadrada da variância. Indica como fechado estão os dados em torno da média.
- **Amplitude**: Medida de dispersão da amostra, sendo uma simples diferença entre o menor e maior valor.
- **Coeficiente de Variação** usado para analisar a dispersão em termos relativos a seu valor médio quando duas ou mais séries de valores apresentam unidades de medida diferentes. Dessa forma, podemos dizer que o coeficiente de variação é uma forma de expressar a variabilidade dos dados excluindo a influência da ordem de grandeza da variável.

Probabilidade é uma medida que nos mostra qual o grau de um evento ocorrer novamente. Muita para da ciência de dados é baseada na tentativa de medir a probabilidade de eventos, desde as chances de um anúncio ser clicado até a falha de uma determinada peça em uma linha de montagem.

> **Dica 2.1: Para saber mais.** Quer entender mais sobre Estatística, recomendo este interessante livro online e em constante evolução `http://onlinestatbook.com/2/index.html`.

2.2 Termos que devemos saber

Como toda ciência, existe um vocabulário comum que os cientistas falam e que devemos conhecer, palavras como: treinar um modelo, MSE, *Overfitting* ou bisbilhotagem fazem parte desse vocabulário. Então mesmo sendo este um livro prático, precisamos saber do que se tratam.

Label (ou rótulo) é a variável que estamos prevendo (saída), enquanto que *feature* (atributo) é a variável de entrada, normalmente é mais de uma. Um problema que pode ser trabalhado com técnicas de ML é

quando existe um padrão e não é fácil defini-lo. Fazendo uma analogia com estatística, é preciso usar um conjunto de observações (amostra) para descobrir um processo subjacente (probabilidade). Os dados são selecionados é aplicado ao modelo que possui um grau de aprendizado (acurácia). Descobrir um padrão não é memorizar *Overfitting*, pois ao obter dados novos deve ser capaz de prever o resultado.

Training (treinar) um modelo significa ensinar o modelo a determinar bons valores para todos os pesos e o viés de exemplos rotulados. *Loss* (perda) é a penalidade por uma má previsão em um único exemplo, que pode ser quantificada por métricas como *Mean Square Error* (MSE). Esse é um cálculo é conhecido como *loss function* (função de perda) sendo que representa uma medida de quão bom um modelo de previsão faz em termos de ser capaz prever o resultado correto.

Existem três princípios em ML:

- **Navalha de Occam**: o modelo mais simples que se ajusta aos dados também é o mais plausível.
- **Viés de amostragem**: se os dados são amostrados de forma tendenciosa, então a aprendizagem também produz resultados tendenciosos.
- **Bisbilhotagem de dados**: se um conjunto de dados afeta qualquer etapa do processo de aprendizado, então a capacidade do mesmo conjunto de dados avaliar o resultado foi comprometida (se usar propriedades adequadas do conjunto de dados, pode assumir relações antes de começar a escolha de modelos).

Teoria Vapnik–Chervonenkis (VC) tenta explicar o processo de aprendizagem de um ponto de vista estatístico. A dimensão VC é uma medida da complexidade de um espaço de funções que pode ser aprendido por um algoritmo de classificação estatística, sendo definido como a cardinalidade do maior conjunto de pontos que o algoritmo pode quebrar. Se tiver poucos dados, modelo mais simples funcionam melhor e complexo é um desastre.

Desigualdade de Hoeffding fornece um limite superior na probabilidade de que a soma de variáveis aleatórias independentes limitadas se desvie de seu valor esperado em mais do que uma certa quantia. Quando generalizada, modelos muito sofisticados (com grande número de hipóteses) fazem perder a ligação entre uma amostra e o total de amostras, o que implica em decorar para uma amostra em vez de fazer um aprendizado que possa ser generalizado para o conjunto todo.

Na aprendizagem supervisionada, um algoritmo de aprendizado de máquina constrói um modelo examinando muitos exemplos e tentando encontrar um modelo que minimize a perda. Uma parte dos dados aleatória dos dados deve ser separada para treinamento e outra para validação. Ainda é costume separar um conjunto de teste para estimar o erro de predição do modelo escolhido e ter certeza de que o modelo não está superajustado. Geralmente, é adotada uma razão de 70% dos dados para treinamento, 20% para validação e 10% para teste.

Gradient Descent (SGD) calcula o gradiente da curva de perda (inclinação) e indica qual o caminho para minimizar o erro (vale de um gráfico da perda em função do peso), redefinindo o peso - quando há vários pesos, o gradiente é um vetor de derivadas parciais com relação aos pesos. Esse vetor gradiente (com direção e magnitude) é multiplicado por um escalar conhecido como taxa de aprendizado (*learning rate* ou *step size*) para determinar o próximo ponto. A heurística de buscar mínimo local começa de diferentes valores para então encontrar o melhor dentre todos. A aleatoriedade permite localizar outros mínimos locais, e assim podemos determinar o melhor, que pode ser o mínimo global.

Step (etapa) é o número total de iterações do treinamento. Uma etapa calcula a perda de um lote e usa esse valor para modificar os pesos do modelo uma vez. *Batch size* (tamanho do lote) é o número total de

exemplos (escolhidos aleatoriamente) que foram usados para calcular o gradiente em uma única etapa (conjunto de dados inteiro).

Epoch (época) é um ciclo completo de treinamento e um gráfico de erro/perda indica como é a evolução do modelo. Cada *epoch* pode conter resultados piores, mas o melhor é guardado e exibido sempre *pocket algorithm*, a não ser que surja um melhor. Esse gráfico também pode ser construído destinado a um conjunto de dados para teste. Se a curva de perda em função das *epoch* cair muito para o treinamento e não acontecer o mesmo com os dados de teste, revela que o modelo está treinado em excesso para essas amostras de treino e não consegue prever um novo conjunto de amostras.

Overfitting acontece quando o ruído é ajustado também, só que ruído não tem padrão a ser descoberto. O ruído pode ser aleatório ou determinístico, relacionado a informação que existe, mas os dados, ou o conjunto de hipóteses, não conseguem promover o aprendizado dessa informação. *Weight decay* (decaimento de peso) é uma constante regularizadora, que restringe os pesos na direção da função alvo e melhora o ajuste e reduz o *overfitting*.

Normalization (normalizar) significa converter valores de recursos numéricos (por exemplo, 100 a 900) em um intervalo padrão (por exemplo, 0 a 1 ou de -1 a +1). Um exemplo desse tipo de cálculo é: $valor - min \div max - min$. Se o conjunto de dados é composto de múltiplos *features*, a normalização gera benefícios, como ajudar a descida gradiente a convergir mais rapidamente, evitar que um número torne-se NaN por exceder seu limite de precisão e ajudar o modelo a aprender os pesos apropriados para cada *features*. A normalização deve ocorrer depois de separar os dados de treinamento e teste.

Standard (padronização) é um cálculo importante para comparar medidas com diferentes unidades. Um exemplo desse tipo de cálculo é o "Z score": $(valor - media) \div std$. A padronização deve ser feita antes da normalização.

2.3 Roteiro

Devemos saber que um roteiro de projetos de Machine Learning passa pelas seguintes fases:

- Entender a questão de negócio (entendimento do negócio).
- Identificar a causa raiz.
- Coletar os dados.
- Aplicar uma limpeza nos dados.
- Entender as variáveis, possíveis valores faltantes.
- Ler os dados, se necessário renomear ou corrigir as colunas.
- Realizar uma estatística descritiva para entender as características dos dados.
- Procurar por incoerências nos dados.
- Criar novas variáveis para modelar melhor o fenômeno, se necessário.
- Levantar Hipóteses sobre o Comportamento do Negócio.
- Definir o tipo do problema: Regressão, Classificação ou Clusterização.
- Realizar uma Análise Exploratória de Dados.
- Quais hipóteses são falsas e quais são verdadeiras?
- Quais as correlações entre as variáveis e a variável alvo?
- Verificar se as variáveis possuem o mesmo peso, em termos de importância, para o modelo.
- Aplicar diferentes algoritmos de Machine Learning.
- Comparar os algoritmos sob a mesma métrica de performance, a mais apropriada.
- Garantir que seu modelo não possui *Overfit*, ou seja, memorização ao invés de aprendizado.

- Escrever os valores de previsão e seu intervalo de confiança do arquivo de teste.
- Descrever uma breve explicação do raciocínio da solução.
- Anotar as respostas encontradas.
- Detalhar as possíveis soluções para o problema.

São vários passos, devemos segui-los para garantir o sucesso das nossas análises. Algumas partes são bem preocupantes, entre elas, as que envolvem os conceitos mais básicos, a tendência é sempre saltá-los sem dar muita importância. Porém são cruciais.

2.4 Bibliotecas Utilizadas

Já temos o *Jupyter* e devemos conhecer o ferramental que iremos trabalhar. Abrir uma célula e digitar os seguintes comandos:

```
import numpy as np
import pandas as pd
import matplotlib
import pylab
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm
from numpy.random import seed
from numpy.random import randn
import matplotlib.colors
import scipy
import sklearn
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
```

Na próxima célula digitar:

```
print('numpy: {}' .format(np.__version__))
print('pandas: {}' .format(pd.__version__))
print('scipy: {}' .format(scipy.__version__))
print('matplotlib: {}' .format(matplotlib.__version__))
print('sklearn: {}' .format(sklearn.__version__))
print('seaborn: {}' .format(sns.__version__))
print('statsmodel: {}' .format(sm.__version__))
print('\nMais informações:\n\n', sklearn.show_versions())
```

E obtemos como resposta as versões das principais bibliotecas que utilizaremos:

- **NumPy** *Numerical Python*. Seu recurso mais poderoso é a matriz n-dimensional. Também contém funções básicas de álgebra linear, transformações de *Fourier*, recursos avançados de números aleatórios e ferramentas para integração com outras linguagens de baixo nível, como Fortran, C e C++.

- **Pandas** *Python and Data Analysis* para operações e manipulações de dados estruturados. Amplamente utilizada para coleta e preparação de dados.

- **SciPy** *Scientific Python*. Tem por base a NumPy. É uma das bibliotecas mais úteis para diversos módulos de ciência e engenharia de alto nível, como matrizes discretas, álgebra linear, otimização e dispersão.
- **Matplotlib** *Math Plotting Library*. para gerar uma grande variedade de gráficos, tais como histogramas, gráfico de linhas e mapa de calor.
- **SkLearn ou Scikit Learn** *SciPy Toolkit Learn*. Tem por base a NumPy, SciPy e Matplotlib. Contém muitas ferramentas para aprendizado de máquina e modelagem estatística, incluindo classificação, regressão, clusterização e redução de dimensionalidade.
- **Seaborn** *Statistical Data Visualization*. Geração de gráficos atraentes e informativos em Python. Tem por base a Matplotlib. Visa tornar a visualização uma parte central da exploração e compreensão dos dados.
- **Statsmodels** *Statistical Models*. Permite explorar dados, estimar modelos estatísticos e executar testes. Uma lista extensa de estatísticas descritivas, funções de plotagem e de resultados estão disponíveis para diferentes tipos de dados e para cada estimador.

Caso exista a necessidade de uma atualização, por exemplo da biblioteca **Scikit-learn**, abrir uma nova célula e digitar o comando:
`!pip install --upgrade scikit-learn`

> **Dica 2.2: Bibliotecas Utilizadas.** Obtenha uma boa referência sobre essas pois não serão tratadas em nível básico neste livro. Obviamente suas funções serão comentadas mas partiremos do pressuposto que sabemos lidar com estas.

2.5 Distribuição Gaussiana

A Distribuição Gaussiana (também conhecida como Curva Normal) é uma curva em forma de sino e supõe-se que, durante qualquer valor de medição, siga uma distribuição normal de um número igual de medidas acima e abaixo do valor médio. Para entendermos a distribuição normal, é importante conhecer as definições de média, mediana e moda. Se uma distribuição for normal, seus valores são os mesmos. No entanto, o valor da média, mediana e moda podem ser diferentes resultando em uma distribuição inclinada (não gaussiana).

Abrir uma nova célula e iremos gerar um gráfico com uma Distribuição Gaussiana ideal:

```
xAxis = np.arange(-3, 3, 0.001) \\
yAxis = norm.pdf(xAxis, 0, 1) \\
plt.plot(xAxis, yAxis) \\
plt.show()
```

Obtemos uma Distribuição Normal perfeita. Normalmente será muito difícil na realidade a curva ser tão perfeita assim, então podemos nos aproximar mais da realidade e gerar a partir de uma amostra de dados randômica:

```
data = 5 * randn(10000) + 50 \\
plt.hist(data, bins=100) \\
plt.show()
```

Obtemos a seguinte Distribuição Normal:

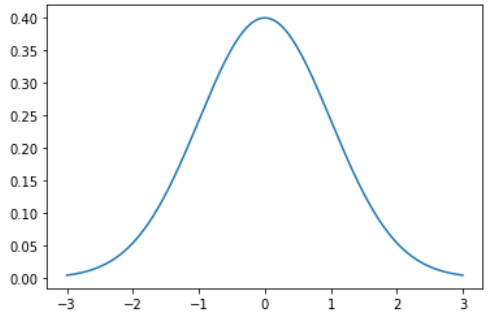


Figura 2.1: Curva da Distribuição Normal

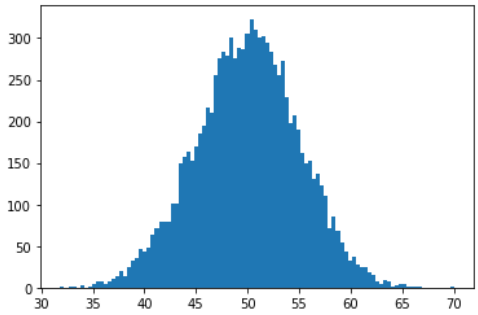


Figura 2.2: A partir de dados randômicos

Também podemos a partir dessa segunda, calcular seus valores principais:

```
print('Média: %.3f' % np.mean(data))
print('Mediana: %.3f' % np.median(data))
print('Moda:', stats.mode(data))
```

2.6 Distribuição de Poisson

Pronuncia-se *Poassom*, é uma distribuição de Probabilidade Discreta para uma variável aleatória X que satisfaça as seguintes condições:

- O experimento consistem em calcular quantas vezes (k) que um evento ocorre em um dado intervalo.
- A probabilidade do evento ocorrer é a mesma em cada intervalo.
- O experimento resulta em resultados que podem ser classificados como **sucessos** ou **falhas**.
- O número médio de sucessos (μ) que ocorre em uma região especificada é conhecido.
- A probabilidade de um sucesso ocorrer é proporcional ao tamanho da região.
- A probabilidade de um sucesso ocorrer em uma região extremamente pequena é praticamente zero.
- O número de ocorrências de um intervalo é independente.

Na prática é uma função de ponto percentual *Poisson* não existe na forma fechada simples. É computado numericamente. Essa é uma distribuição discreta, definida apenas para valores inteiros de x, a função de ponto percentual não é suave da mesma forma que a função de ponto percentual normalmente é para uma distribuição contínua.

Para vê-la na prática começamos com a importação das bibliotecas necessárias:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

A NumPy já nos provê a função necessária para vermos como é o gráfico:

```
dp = np.random.poisson(lam=3, size=(1000))
plt.hist(dp)
```

```
plt.show()
```

Geramos 1.000 números aleatórios com uma ocorrência 3 e obtemos como resultado:

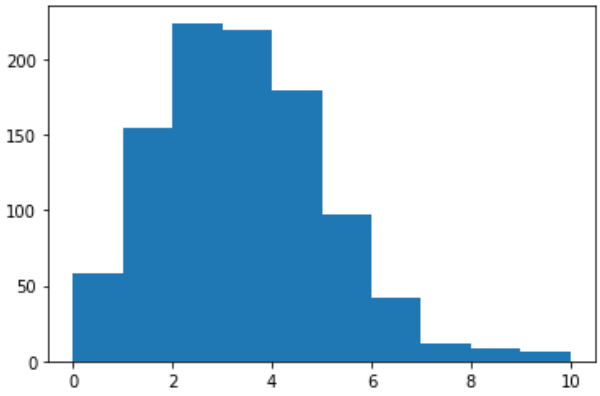


Figura 2.3: Distribuição de Poisson na MatPlotLib

Uma variável aleatória de Poisson é o número de sucessos resultantes de um experimento de Poisson. Porém esse gráfico fica bem mais apresentável com o uso da SeaBorn:

```
sns.distplot(dp)
plt.show()
```

E obtemos:

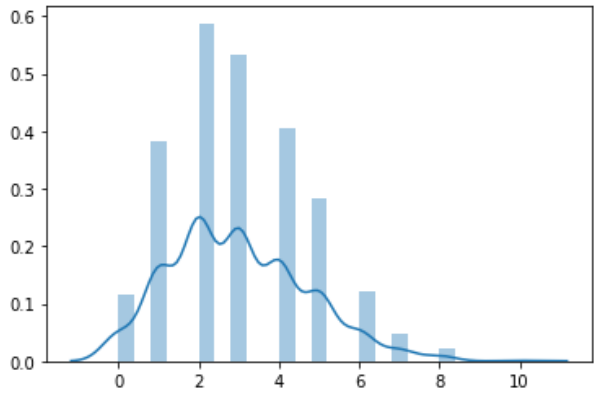


Figura 2.4: Distribuição de Poisson na Seaborn

Dica 2.3: Historicamente falando. 1946, o estatístico britânico *RD Clarke* publicou "*Uma Aplicação da Distribuição de Poisson*", com sua análise dos acertos de bombas voadoras (mísseis V-1 e V-2) em Londres durante a II Guerra Mundial. Algumas áreas foram atingidas com mais frequência do que outras. Os militares britânicos desejavam saber se os alemães estavam atacando esses distritos (os acertos indicavam grande precisão técnica) ou se a distribuição era por acaso. Se os mísseis fossem de fato apenas alvos aleatórios (dentro de uma área mais geral), os britânicos poderiam simplesmente

> dispersar instalações importantes para diminuir a probabilidade de serem atingidos.

Se acertarmos a escala, surpreendentemente, ao colocar uma Distribuição Normal e uma de Poisson sobrepostas:

```
sns.distplot(np.random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')
sns.distplot(np.random.poisson(lam=50, size=1000), hist=False, label='poisson')
plt.show()
```

Obtemos como resultado:

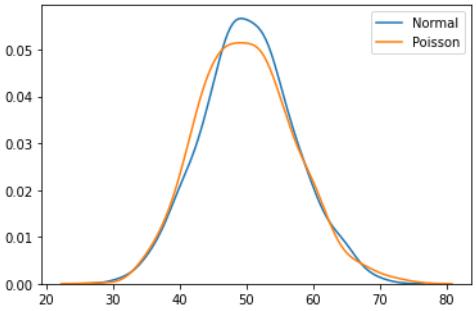


Figura 2.5: Distribuição Normal e de Poisson

E claramente percebemos a similaridade entre ambas. Um caso curioso que ocorre com a Distribuição de Poisson é a "cauda longa", sabemos que em qualquer comércio existe os produtos que mais possuem saída e aqueles outros que estão ali para "compor estoque". Por exemplo:

```
ax = sns.distplot(np.random.poisson(lam=3, size=10000), bins=27, kde=False)
ax.set(xlabel='Mercadorias', ylabel='Unidades Vendidas')
plt.show()
```

Obtemos como resultado:

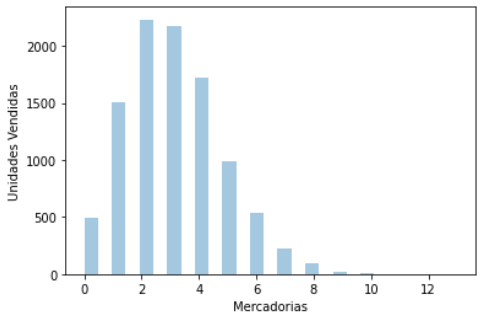


Figura 2.6: A Cauda Longa

A partir do número 6 temos um decrescimento constante, o que demonstra exatamente o que acontece com algumas mercadorias.

2.7 Distribuição Binomial

Neste tipo de distribuição apenas dois resultados são possíveis: sucesso ou fracasso, ganho ou perda, vitória ou perda e a probabilidade é exatamente a mesma para todas as tentativas. No entanto, os resultados não precisam ser igualmente prováveis, e cada estudo é independente um do outro. Podemos ver sua curva com 1.000 lançamentos de 10 possibilidades

```
sns.distplot(np.random.binomial(n=10, p=0.5, size=1000), hist=True)
plt.show()
```

Obtemos como resultado:

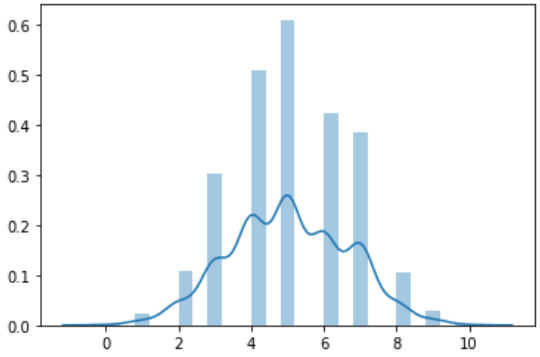


Figura 2.7: Distribuição Binomial

A principal diferença para a normal é que essa é contínua, enquanto que a binomial é discreta, mas se houver pontos de dados suficientes, serão bastante semelhantes (inclusive com a Poisson).

```
sns.distplot(np.random.normal(loc=50, scale=7, size=1000), hist=False, label='Normal')
sns.distplot(np.random.poisson(lam=50, size=1000), hist=False, label='Poisson')
sns.distplot(np.random.binomial(n=100, p=0.5, size=200), hist=False, label='Binomial')
plt.show()
```

Obtemos como resultado:

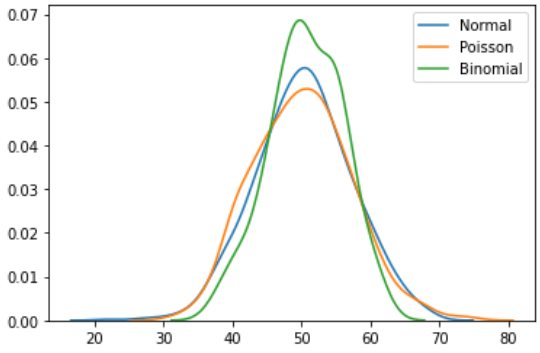


Figura 2.8: Mesmo gráfico 3 Distribuições

Mas e na prática? Imaginemos que 90% dos passageiros reservados chegam de fato para sua viagem.

Suponhamos um transporte que pode conter 45 assentos. Muitas vezes as companhias praticam "excesso de reservas"(conhecido por *Overbooking*) isso significa que vende mais passagens do que os assentos disponíveis. Isso se deve ao fato de que às vezes os passageiros não aparecem um assento vazio representa perda. No entanto, ao reservar em excesso corre o risco de ter mais passageiros do que assentos.

Com esses riscos em mente, uma companhia decide vender mais de 45 bilhetes. Supondo que desejam manter a probabilidade de ter mais de 45 passageiros para embarcar no voo abaixo de 0,4 quantas passagens podem vender?

Para resolvermos isso precisamos da SciPy, e procedemos a seguinte codificação:

```
from scipy.stats import binom_test
for i in range(45, 51):
print(i, "-", binom_test(x=45, n=i, p=0.9, alternative='greater'))
```

Obtemos como resultado:

```
45 - 0.008727963568087723
46 - 0.04800379962448249
47 - 0.13833822255419037
48 - 0.27986215181073293
49 - 0.449690866918584
50 - 0.6161230077242769
```

O parâmetro *alternative* que indica a hipótese, podemos usar:

- *greater* maior que
- *less* menor que
- *two-sided* maior e menor que (valor padrão)

E podemos vender **48 passagens**. E agora sabemos porque todas as companhias de transporte praticam *overbooking*. Para vermos isso graficamente:

```
sns.distplot(np.random.binomial(n=48, p=0.9, size=400), label='Binomial')
```

Obtemos como resultado:

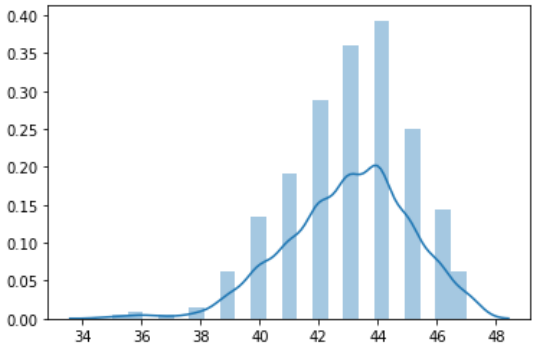


Figura 2.9: Gráfico do Overbooking

2.8 Feature Selection

Seleção de Atributos, Variáveis, Recursos ou Subconjuntos de Variáveis, são os vários nomes para o processo para a seleção de *features* (atributo) relevantes para a construção de modelos.

Label (ou rótulo) é a variável que estamos prevendo, enquanto que uma *feature* é a variável de entrada, que pode inclusive ser mais de uma. Deve ser realizada depois da etapa de pré-processamento dos dados.

O objetivo é selecionar as melhores variáveis como possíveis **preditoras**. Essa etapa ajuda a reduzir o *overfitting*, aumenta a acurácia do modelo e reduz o tempo de treinamento.

São os seguintes métodos que podemos utilizar:

- **Filter Methods**: usam medidas estatísticas para atribuir uma pontuação para cada *feature*. Estas são classificadas de modo a serem mantidas ou removidas do modelo. Normalmente usamos testes univariados que consideram a independência da *feature* com a variável alvo. Exemplo: *chi squared*, pontuações com Coeficiente de Correlação.
- **Wrapper Methods**: selecionam um conjunto de *features*, onde diferentes combinações são preparadas, avaliadas e comparadas. Um modelo preditivo é usado para avaliar a combinação de *features* a atribuir uma nota a partir a acurácia de modelo. Exemplo: RFE.
- **Embedded Methods**: aprendem quais *features* contribuem melhor para a acurácia do modelo no momento de sua construção. Exemplo: Métodos de Penalização, Algoritmos Lasso, *Elastic NEt* e *Ridge Regression*.

> **Dica 2.4: Bases de Dados.** Para TODOS os exemplos neste livro, utilizamos bases de dados que estão disponibilizadas em `https://github.com/fernandoans/machinelearning/tree/master/bases`. Salvo qualquer observação em contrário.

Importação das bibliotecas utilizadas:

```
1 import pandas as pd
2 from sklearn.feature_selection import SelectKBest
3 from sklearn.feature_selection import f_classif, mutual_info_classif
4 from sklearn.feature_selection import chi2
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.feature_selection import RFE
7 from sklearn.ensemble import RandomForestClassifier
8 %matplotlib inline
```

Os classificadores estão contidos na Scikit-Learn e utilizaremos como nossa fonte de dados o arquivo chamado **pima-indians-diabetes.csv**. Criar um DataFrame para esta:

```
1 colnames = ['gest', 'glic', 'sang', 'skin', 'insul', 'mass', 'familia', 'idade',
      'conf']
2 df = pd.read_csv('pima-indians-diabetes.csv', names=colnames)
3 df.head()
```

Esta base são dados do **Instituto Nacional de Diabetes, Doenças Digestivas e Renais** sendo de pacientes do sexo feminino. Consiste de várias variáveis preditivas médicas (independentes) e uma variável alvo

(dependente). Variáveis independentes incluem o número de gestações que a paciente teve, seu IMC, nível de insulina, idade e outras informações. Essa base possui 798 linhas sem quaisquer presença de nulos, verificar com: `df.info()`

A coluna *conf* é a nossa variável alvo se o paciente em questão teve ou não diabetes, então precisamos isolá-la:

```
X = df.drop(['conf'], axis=1)
y = df['conf']
```

Para as outras desejamos sabe: qual (ou quais) variável independente se comportar melhor para o nosso modelo?

2.8.1 Coeficientes de Coorelação

Como primeira forma aplicamos os testes estatísticos:

```
f_classif1 = SelectKBest(score_func=f_classif, k=4)
fit1 = f_classif1.fit(X,y)
```

Os tipos para o **SelectKBest**, neste caso, são:

- **f_classif**: mais adequado quando os dados são numéricos e a variável alvo é categórica.
- **mutual_info_classif**: mais adequando quando não existe uma dependência linear entre os atributos e a variável alvo.
- **f_regression**: para resolver problemas de regressão.

Visualizar as variáveis selecionadas:

```
cols = fit1.get_support(indices=True)
df.iloc[:,cols]
```

Indica que número de gestações (*gest*), concentração de glicose no plasma (*glic*), índice de massa corporal (*mass*) e *idade* seriam as melhores candidatas.

2.8.2 Chi Squared

Usada para medir a dependência entre variáveis estocásticas, o uso dessa função "elimina"os recursos com maior probabilidade de serem independentes da classe e, portanto, irrelevantes para a classificação:

```
test2 = SelectKBest(chi2, k=4)
fit2 = test2.fit(X, y)
```

Visualizar as variáveis selecionadas:

```
cols = fit2.get_support(indices=True)
df.iloc[:,cols]
```

Percebemos que aconteceu uma mudança nas variáveis selecionadas: *glic*, *mass* e *idade* continuam, porém ao invés de *gest* aparece a quantidadde adminstrada de insulina (insul).

2.8.3 RFE

Recursive Feature Elimination é um método que remove o(s) recurso(s) mais fraco(s) até que o número especificado de recursos seja atingido. Sendo assim é necessário informar ao RFE o número de atributos caso contrário reduz pela metade esse valor de acordo com o número de *feature* dos dados:

```
model = LogisticRegression(max_iter=2000, solver='lbfgs')
rfe = RFE(model, n_features_to_select=4)
fit3 = rfe.fit(X, y)
```

Visualizar as variáveis selecionadas:

```
cols = fit3.get_support(indices=True)
df.iloc[:,cols]
```

Como resultado obtemos quase a mesma combinação anterior e aparece mais a variável *familia* que indica se existem casos de diabetes na família.

2.8.4 Ensembles Methods

Métodos de agrupamento[1], como o algoritmo *Random Forest*, podem ser usados para estimar a importância de cada atributo. Retorna um valor para cada atributo, quanto mais alto esse, maior é a importância desse atributo.

Aplicar o algoritmo:

```
model = RandomForestClassifier(n_estimators=10)
model.fit(X, y)
RandomForestClassifier(bootstrap=True, class_weight=None,
  criterion='gini', max_depth=None, max_features='auto',
  max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
  min_samples_leaf=1, min_samples_split=2,
  min_weight_fraction_leaf=0.0, n_estimators=10,
  n_jobs=None, oob_score=False, random_state=None,
  verbose=0, warm_start=False)
```

E podemos gerar uma visualização:

```
feature_importancia = pd.DataFrame(model.feature_importances_,
  index = X.columns, columns=['importancia']).sort_values('importancia',
    ascending=False)
feature_importancia
```

Porém é preferível vê-las através de um gráfico:

[1] São métodos que utilizam vários algoritmos de aprendizado para obter um melhor desempenho preditivo.

```
feature_importancia.plot(kind='bar')
```

Obtemos como resultado:

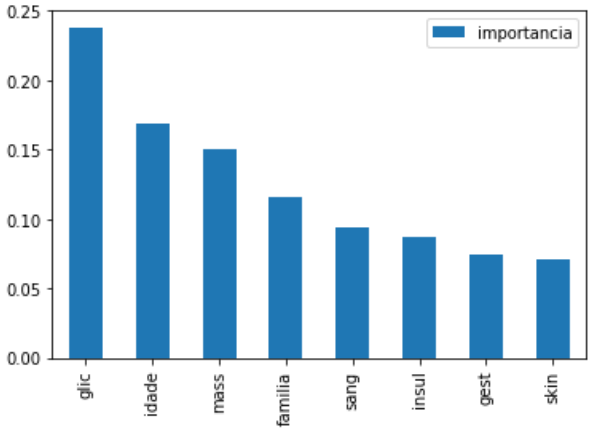


Figura 2.10: Grau de Importância dos Atributos

E afinal de contas com tudo o que vimos. Qual Método utilizar?

- Usar *RFE* caso tenha recursos computacionais para isso.
- Ao trabalhar com Classificação e as *features* forem numéricas, utilizar *f_classif* ou *mutual_info_classif*.
- Ao trabalhar com Regressão e as *features* forem numéricas, utilizar *f_regression* ou *mutual_info_regression*.
- Ao trabalhar com *features* categóricos utilizar *Chi Squared*.

2.9 K Fold Cross Validation

Vimos como encontrar as melhores *features* para trabalhar, porém verificamos um pequeno problema: qual a forma em se escolher o modelo ideal[2] para nossa base de dados? O que fazemos é testar um modelo várias vezes obtendo pedaços diferentes de treino e teste a cada vez.

Usaremos a mesma base de Casos de Diabetes com vários modelos. Começamos com a importação das bibliotecas:

```
from pandas import read_csv
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

[2] "Ideal"é apenas um conceito para designar qual terá uma melhor acurácia (não usemos como sinônimo para perfeito ou o melhor)

```
%matplotlib inline
```

Em seguida colocamos os dados em um DataFrame:

```
colnames = ['gest', 'glic', 'sang', 'skin', 'insul', 'mass', 'familia', 'idade',
    'conf']
df = read_csv('pima-indians-diabetes.data.csv', names=colnames)
df.head()
```

Para não ficar gerando códigos repetitivos, criar uma lista com todos os modelos que vamos executar:

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

Trabalharemos com Regressão Logística (LR), Análise Descriminante (LDA), K-Neighbors (KNN), Árvore de Decisão (DT), Gaussian tipo NB e SVM. E para avaliar a acurácia de cada um dos modelos:

```
acuracia = []
for sigla, modelo in models:
  kfold = KFold(n_splits=10, random_state=7, shuffle=True)
  resultado = cross_val_score(modelo, X, Y, cv=kfold, scoring='accuracy', n_jobs=-1)
  acuracia.append(resultado.mean())
  print("%s: %f (%f)" % (sigla, resultado.mean(), resultado.std()))
```

O objeto **KFold** criado executa 10 vezes (definido em *n_splits*) cada um dos modelos mantendo um estado de aleatoriedade de 7 registros. O método *cross_val_score* realiza todo o trabalho que tivemos para executar um modelo, dividir a base de dados em treino e teste obtendo o score de cada execução e obtemos o mesmo resultado final, a cada resposta podemos definir o número de vezes a executar (definido em *n_jobs*) neste caso será executado somente 1 vez (o valor é -1).

Ao mostrar a variável *resultado* obtemos uma lista de 10 valores para cada um dos modelos, porém para facilitar, trazemos a média (que também será adicionada para a lista *acuracia*) e o desvio padrão:

```
LR: 0.778640 (0.047350)
LDA: 0.766969 (0.047966)
KNN: 0.710988 (0.050792)
DT: 0.696770 (0.046741)
NB: 0.759142 (0.038960)
SVM: 0.760458 (0.034712)
```

Também podemos ter isso de forma gráfica:

```
fig = plt.figure(figsize = (8,4))
axes = fig.add_axes([0.0, 0.0, 1.0, 1.0])
axes.set_title('Comparação dos Modelos');
axes.bar([item[0] for item in models], acuracia)
plt.show()
```

E obtemos como resultado:

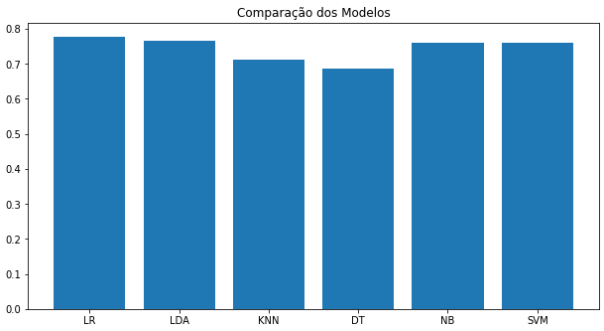


Figura 2.11: Melhor Acurácia dos Modelos

2.10 Matriz de Confusão

A Matriz de Confusão nos auxilia a medir a precisão do nosso modelo. A acurácia é uma boa medida porém um tanto falha, suponhamos que estejamos para realizar um trabalho com dados de pacientes que desejam saber a possibilidade de desenvolver uma determinada doença ou não. São quatro respostas possíveis que o nosso modelo pode prover:

- **Verdadeiro Positivo** (*TP - true positive*): no conjunto da classe real, o resultado foi correto. O paciente desenvolveu a doença e o modelo previu que iria desenvolver. (T & T)
- **Falso Positivo** (*FP - false positive*): no conjunto da classe real, o resultado foi incorreto. O paciente desenvolveu a doença e o modelo previu que não iria desenvolver. (T & F - Erro tipo 2)
- **Falso Negativo** (*FN - false negative*): no conjunto da classe real, o resultado foi incorreto. O paciente não desenvolveu a doença e o modelo previu que iria desenvolver. (F & T - Erro tipo 1)
- **Verdadeiro Negativo** (*TN - true negative*): no conjunto da classe real, o resultado foi correto. O paciente não desenvolveu a doença e o modelo previu que não iria desenvolver. (F & F)

Os casos 2 e 3 são erros no modelo, sendo que o **2** é é considerado um pior tipo. Para começar com a nossa prática importar as bibliotecas necessárias:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sn
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import confusion_matrix
7 from sklearn.linear_model import LogisticRegression
8 %matplotlib inline
```

A biblioteca *sklearn.metrics* na versão 0.22 ganhou o método **confusion_matrix**. Somente para entendermos como funciona a Matriz de Confusão, suponhamos que foi realizado um teste com base em

características de pacientes, existe uma possibilidade de desenvolver (D) ou não (S) uma determinada doença:

```
y_true = pd.Series(['D','D','D','D','D','D','S','S','S','S','S','S','S'])
y_pred = pd.Series(['D','D','S','D','D','D','D','S','D','S','S','D','S'])
```

A série contendo y_true é o resultado real (ou verdadeiro) e y_pred foi o resultado que o modelo disse que iria ocorrer. Se fossemos somente pela acurácia obtemos 13 respostas sendo 4 delas erradas, porém como saber se o modelo está realmente agindo bem e o mais importante onde está se confundindo?

```
conf = confusion_matrix(y_true, y_pred)
print(conf)
```

Ao usarmos o método *confusion_matrix* obtemos a seguinte Matriz que nos auxilia a responder essas questões:
[[5 1][3 4]]

O eixo **X** da Matriz representa o que foi predito e **y** o que realmente aconteceu pelo modelo. Nas diagonais obtemos as corretas, cinco que estão Doentes e quatro que estão sadios e o modelo acertou. Porém, três não estão doentes mas foi previsto que estariam (se pensarmos a notícia não é tão ruim - do ponto de vista para o paciente, por isso esse é o erro tipo 1) e um está doente porém prevemos que não estaria (péssima notícia, por isso esse é o erro tipo 2). Visualizar resultados assim pode ser bem complexo, então tentemos uma adaptação do Mapa de Calor da biblioteca **SeaBorn**:

```
data = {
  'Ocorreu': y_true,
  'Predito': y_pred
}
df = pd.DataFrame(data, columns=['Ocorreu','Predito'])
conf = pd.crosstab(df['Ocorreu'], df['Predito'], rownames=['Ocorreu'],
    colnames=['Predito'])
res = sn.heatmap(conf, annot=True, fmt='.0f', annot_kws={"size":12},
    cmap=plt.cm.Blues)
plt.show()
```

E obtemos como resultado:

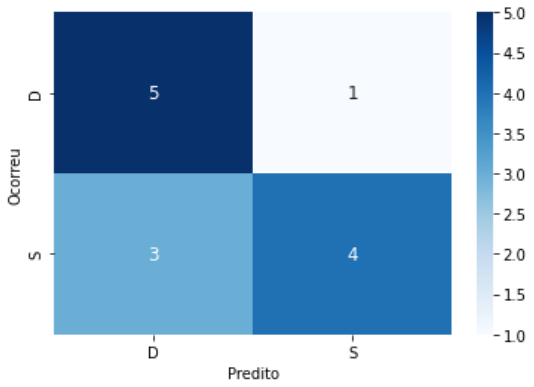


Figura 2.12: Mapa de Calor mostrando a Matriz de Confusão

Quanto mais escuro, maior a quantidade de elementos, assim podemos rapidamente avaliar como nosso modelo se comporta (o ideal é que as diagonais fiquem bem escuras enquanto que as extremidades claras).

2.10.1 Em valor ou percentual?

Um detalhe muito comum de acontecer é: devemos mostrar o valor em decimal ou percentual? Ou seja as quantidades reais das amostras ou um percentual do todo? Usar o seguinte código para gerar uma matriz de um teste realizado com imagens:

```
conf_arr = np.array([[88,14,4],[12,85,11],[5,15,91]])
sum = conf_arr.sum()
df_cm = pd.DataFrame(conf_arr,
  index = [ 'Cão', 'Gato', 'Coelho'],
  columns = ['Cão', 'Gato', 'Coelho'])
res = sn.heatmap(df_cm, annot=True, vmin=0.0, vmax=100.0, cmap=plt.cm.Blues)
plt.yticks([0.5,1.5,2.5], ['Cão', 'Gato', 'Coelho'],va='center')
plt.title('Matriz de Confusão')
plt.show()
```

E obtemos como resultado:

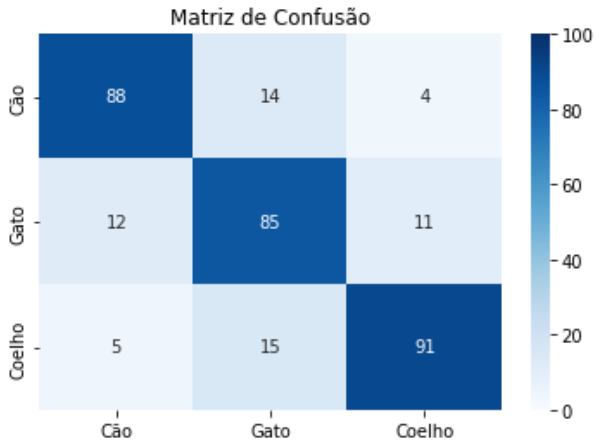


Figura 2.13: Comparativo numérico na Matriz de Confusão

E ao realizarmos um ajuste:

```
conf_arr = conf_arr * 100.0 / ( 1.0 * sum )
conf_arr /= 100
df_cm = pd.DataFrame(conf_arr,
  index = [ 'Cão', 'Gato', 'Coelho'],
  columns = ['Cão', 'Gato', 'Coelho'])
res = sn.heatmap(df_cm, annot=True, vmin=0.0, vmax=0.3, fmt='.2%', cmap=plt.cm.Blues)
plt.title('Matriz de Confusão (em %)')
plt.show()
```

Obtemos a seguinte matriz agora com o resultado em percentual:

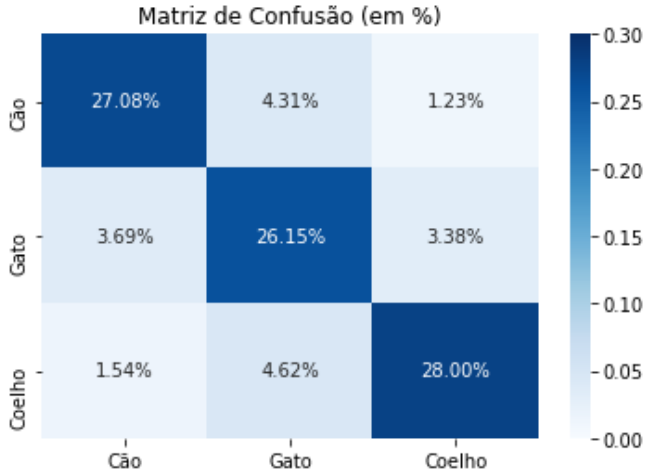


Figura 2.14: Comparativo percentual na Matriz de Confusão

Qual dos dois é melhor? A resposta seria: que fica mais fácil para que o Cientista de Dados consiga explicar o resultado para seu público com a maior clareza, não existe uma regra definida para isso.

2.10.2 Na prática

Voltamos ao nossos dados sobre Diabetes, ler os dados:

```
colnames = ['gest', 'glic', 'sang', 'skin', 'insul', 'mass', 'familia', 'idade',
    'conf']
df = pd.read_csv('pima-indians-diabetes.data.csv', names=colnames)
df.head()
```

Deixar somente as colunas que nos interessam, conforme vimos na seleção de atributos:

```
df = df.drop(columns=['sang'],axis=1)
df = df.drop(columns=['insul'],axis=1)
df = df.drop(columns=['gest'],axis=1)
df = df.drop(columns=['skin'],axis=1)
df.head()
```

> **Dica 2.5: 100% de Precisão.** Sempre que ocorrer 100% podemos ligar todas as antenas pois existe um erro, nenhum modelo possui essa previsão tão perfeita. Considere também que abaixo de 70% não é preditivo.

Agora acontece um erro clássico, a base contém *conf* que é nossa variável alvo. Sse seguimos em frente para separar as bases em teste e treino o modelo provavelmente acusa uma acurácia (errada) de 100% de precisão, é como se ele estivesse colando as respostas. Então devemos isolar essa coluna e retirá-la dos dados.

```
conf = df['conf']
df = df.drop(columns=['conf'],axis=1)
```

E assim podemos separar a base de dados em treino e teste:

```
X_train, X_test, y_train, y_test = train_test_split(df, conf, test_size = .25)
```

Usamos uma medida de 25%, ou seja 75% para treino e o restante para testar nosso modelo. Existem 768 registros não nulos no total, o resultado será: 576 registros para o modelo treinar e 192 para testar. Normalmente deixamos 25% para teste e verificação da acurácia do modelo esse número pode ser aumentado ou diminuído conforme seus dados, não existe uma regra definida.

Conforme nossa verificação do melhor modelo, vimos que devemos trabalhar com **Regressão Logística**, então executar o modelo e verificar a acurácia:

```
clf = LogisticRegression(max_iter=10000)
clf.fit(X_train, y_train)
print('Acurácia:', clf.score(X_test, y_test))
```

Esse resultado pode variar mas está em torno dos 76%, o problema é, onde esse modelo está errando? Executar a matriz de confusão para descobrir:

```
y_pred = clf.predict(X_test)
conf = confusion_matrix(y_test, y_pred)
data = {
  'Ocorreu': y_test,
  'Predito': y_pred
}
df2 = pd.DataFrame(data, columns=['Ocorreu','Predito'])
conf = pd.crosstab(df2['Ocorreu'], df2['Predito'], rownames=['Ocorreu'],
    colnames=['Predito'])
res = sn.heatmap(conf, annot=True, fmt='.0f', annot_kws={"size":12},
    cmap=plt.cm.Blues)
plt.show()
```

E obtemos como resultado:

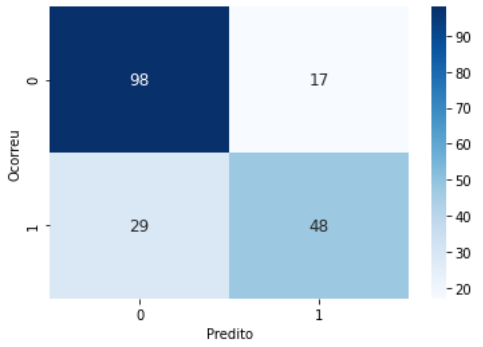


Figura 2.15: Resultado da Matriz de Confusão

Vemos que o modelo se comporta bem nos casos que o paciente não tem diabetes, acerta 85,2% e erra 14,8% das vezes. Já quando tem a doença acerta 62,3% e erra 37,7% das vezes. Ou seja, para melhorarmos a acurácia precisamos de mais dados com pacientes que desenvolveram a diabetes.

2.11 Curva ROC e Valor AUC

Entre todas as teorias vistas para Ciência de Dados este é um dos conceitos mais simples e ao mesmo tempo mais complicado (acho que é equivalente a Matriz de Confusão e inclusive depende dela).

Na teoria **ROC** (*Receiver Operating Characteristic*, algo como Característica de Operação do Receptor) é uma curva de probabilidade. É criada ao traçar a taxa de verdadeiro-positivo (TPR - true positive rate) contra a taxa de falsos-positivos (FPR - false positive rate). Que taxas são essas? Voltando ao conceito de Matriz de Confusão a TPR e FPR são calculadas pelas fórmulas:

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}$$

Figura 2.16: Fórmulas para o cálculo da ROC

Ou seja, número de vezes que o classificador acertou a predição contra o número de vezes que errou. **AUC** (*Area Under the Curve*) representa a área da ROC, considera-se como o grau ou medida de separabilidade. Quanto maior o valor, melhor é o modelo em prever ou (por exemplo) em distinguir entre pacientes com e sem uma determinada doença.

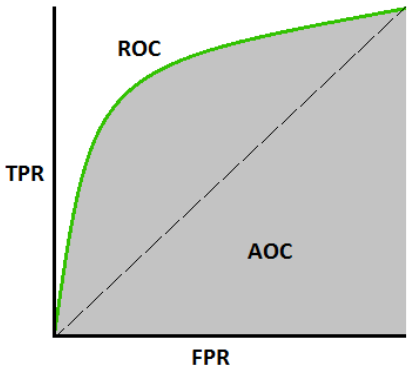


Figura 2.17: Fórmulas para o cálculo da ROC

Vejamos um simples exemplo para entendermos como esse processo funciona:

```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
%matplotlib inline
```

Apenas para entendermos como aquilo tudo que foi escrito no início da seção funciona, usar a função *make_classification* para produzir alguns dados aleatórios:

```
X, y = make_classification(n_samples = 10000, n_features=10, n_classes = 2, flip_y =
    0.5)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .25)
```

Foram geradas 10.000 amostras com 10 campos e 2 classes (traduzindo para o português isso significa valores 0 e 1), separamos 25% para testar nosso modelo. Usar o Modelo de Regressão Logística para analisar esses dados:

```
model = LogisticRegression(solver='liblinear', penalty='l2', C=0.1)
model.fit(X_train, y_train)
print('Acurácia', model.score(X_test, y_test))
```

Como os dados são randômicos o resultado pode variar, mas chegamos em torno de 70%. E agora podemos calcular o **AUROC** (*Area Under the Receiver Operation Characteristics*):

```
y_prob = model.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_prob)
auroc = float(format(metrics.roc_auc_score(y_test, y_prob), '.8f'))
print(auroc)
```

Existe 72% de área preenchida. Qual o motivo da diferença? Estamos levando em consideração as taxas TPR e FPR, não apenas acertou/errou. **AUC** resume a curva **ROC** num único valor que é o calculo da "área sob a curva", porém apresentar a informação assim não tem graça, colocar de forma gráfica:

```
plt.plot(fpr, tpr, label='Curva ROC (area = %0.2f)' % auroc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Exemplo do ROC')
plt.legend(loc="lower right")
plt.show()
```

E obtemos como resultado:

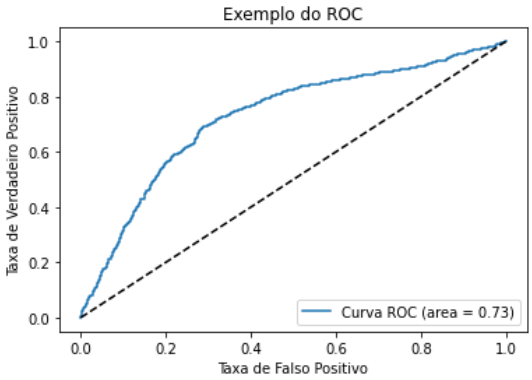


Figura 2.18: Visão da ROC

2.11.1 Na prática

Voltamos ao nossos dados sobre Diabetes, ler os dados:

```
colnames = ['gest', 'glic', 'sang', 'skin', 'insul', 'mass', 'familia', 'idade',
    'conf']
df = pd.read_csv('pima-indians-diabetes.data.csv', names=colnames)
df.head()
```

Deixar somente as colunas que nos interessam, conforme vimos na seleção de atributos:

```
df = df.drop(columns=['sang'],axis=1)
df = df.drop(columns=['insul'],axis=1)
df = df.drop(columns=['gest'],axis=1)
df = df.drop(columns=['skin'],axis=1)
df.head()
```

Isolar a coluna alvo e retirá-la dos dados.

```
conf = df['conf']
df = df.drop(columns=['conf'],axis=1)
```

Separar a base de dados em treino e teste:

```
X_train, X_test, y_train, y_test = train_test_split(df, conf, test_size = .25)
```

Iremos testar dois modelos de agrupamento para saber qual o melhor comportamento com esses dados. Para facilitar nossa vida, criar um método que retorna a acurácia de um determinado modelo:

```
def score(mdl, Xtrn, Xtst, ytrn, ytst):
  mdl.fit(Xtrn, ytrn)
  return float(format(mdl.score(Xtst, ytst), '.8f'))
```

Recebemos o modelo a ser treinado, e o conjunto de variáveis X e y de treino e teste para devolvermos a acurácia do modelo. De modo semelhante:

```
def auroc(ytst, yprob):
  fpr, tpr, _ = metrics.roc_curve(ytst, yprob)
  auc = float(format(metrics.roc_auc_score(ytst, yprob), '.8f'))
  return fpr, tpr, auc
```

Esse outro método recebe o y de teste e o de probabilidades e retorna o **FPR**, **TPR** e **AUC**.

Os modelos de agrupamento para realizarmos nosso teste são **Regressão Logística** e **Floresta Aleatória**. Para cada um desses, basicamente, são os mesmos comandos.

Regressão Logística:

```
clfRL = LogisticRegression(max_iter=1000)
print("Acurácia RL:", score(clfRL, X_train, X_test, y_train, y_test))
```

```
y_probRL = clfRL.predict_proba(X_test)[::,1]
fprRL, tprRL, aucRL = auroc(y_test,  y_probRL)
print("AUC RL", aucRL)
```

Floresta Aleatória:

```
clfRF = RandomForestClassifier(n_estimators=1000)
print("Acurácia RF:", score(clfRF, X_train, X_test, y_train, y_test))
y_probRF = clfRF.predict_proba(X_test)[::,1]
fprRF, tprRF, aucRF = auroc(y_test,  y_probRF)
print("AUC RF:", aucRF)
```

Ambos os modelos trabalham com grupos de 1.000, sendo que para a **Regressão Logística** isso é considerado de iterações realizadas enquanto que para a **Floresta Aleatória** o número de árvores de decisão utilizadas.

> **Dica 2.6: Como esses modelos trabalham?.** Não devemos nos preocupar nesse momento como esses modelos processam, em capítulos subsequentes trataremos separadamente cada um deles. Aqui veremos apenas os conceitos que serão necessários para a melhor escolha e avaliação dos modelos.

E obtemos o seguinte resultado, que pode variar de acordo com o que foi separado de treino e teste: A **Regressão Logística** atingiu um resultado melhor tanto na acurácia (73,95%) quanto na curva com quase 80% de área ocupada enquanto que a **Floresta Aleatória** ficou com quase 71% de acurácia e 77,35% de área. Às vezes o resultado de acurácia pode até ser similar, mas raramente a área ocupada será igual:

```
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fprRL,tprRL,label="RL " + str(aucRL))
plt.plot(fprRF,tprRF,label="RF " + str(aucRF))
plt.xlabel('Taxa de Falso Positivo')
plt.ylabel('Taxa de Verdadeiro Positivo')
plt.title('Detecção da Diabetes')
plt.legend(loc="lower right")
plt.show()
```

E obtemos como resultado:

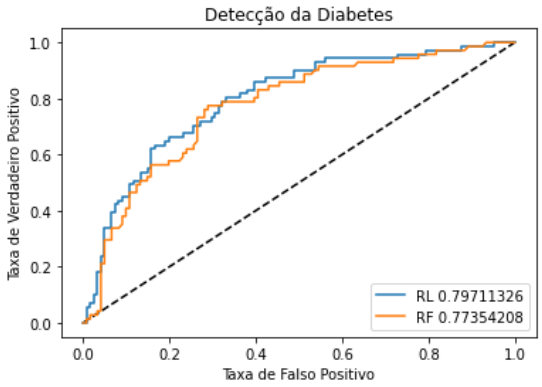


Figura 2.19: Performance dos Modelos

2.12 Terminamos?

Como conceitos sim, mas como prática permita-me deixar um exercício. A **Scikit-Learn** nos oferece uma base sobre de cistos (Câncer de Mama) encontrados em pacientes de *Wisconsin* com classificação se é malígno (212 amostras) ou benigno (357 amostras). Para usá-la importar a biblioteca:

```
from sklearn.datasets import load_breast_cancer
```

Carregamos os dados:

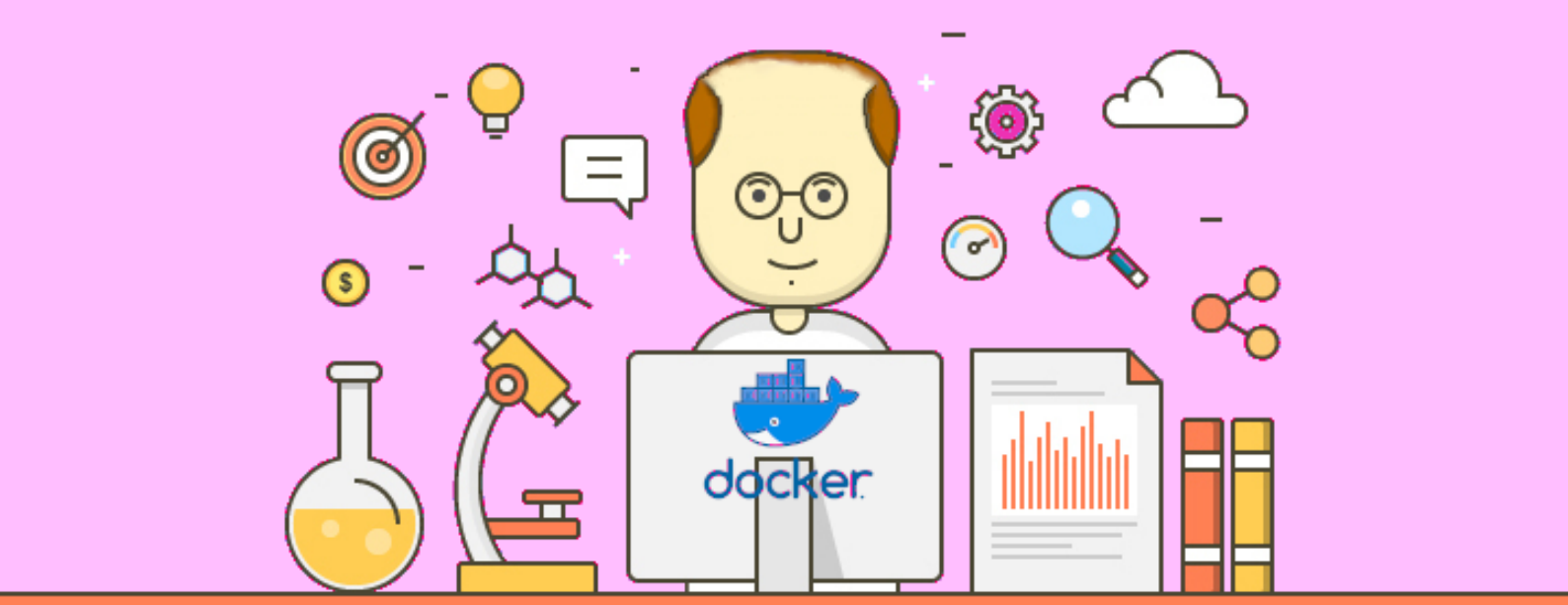
```
cancer = load_breast_cancer()
```

E obtemos um objeto *Bunch* da Scikit-Learn com os 569 casos registrados. Para criar as nossas bases de treino e teste usar o comando:

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
    test_size = .25)
```

A variável *data* contém os *features* enquanto que *target* se a paciente teve um tipo malígno ou não. Pronto, agora é com você. Aplicar os conhecimentos que vimos nesse capítulo para definir quais são as melhores *features* a usar e o modelo colhendo todos os resultados possíveis.

No endereço `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html` está disponibilizada a documentação sobre esta base.



3. EDA

Nós torturamos os dados até eles confessarem. (Ricado Cappra - Cientista de Dados)

3.1 Passos da EDA

EDA é fundamental para entender qualquer conjunto de dados. É aqui podemos obter informações e fazer descobertas. Aqui colocamos o conhecimento para trabalhar. Acrônimo para *Exploratory Data Analysis* (Análise Exploratória de Dados), desempenha um papel crítico na compreensão do quê? por que? e como? na declaração do problema.

É a primeira ação a ser realizada na ordem das operações que um Cientista de Dados deve executar ao receber uma nova fonte de dados e a declaração de problema. Tratamos EDA como uma série de técnicas utilizadas como forma de entendermos os diversos aspectos dos dados.

Figura 3.1: Passos da EDA

A preparação dos dados para análise é inevitável, e a maneira como fazemos isso define a sua qualidade. Na prática o que faremos nesse capítulo é compreendermos o que temos a nossa disposição para trabalhar.

Normalmente os dados se dividem nas variáveis **Preditoras** (Entradas) e **Alvo** (saída). Uma vez que as localizamos devemos identificar o tipo de dados e a categoria das variáveis.

3.2 Passo 1 - Entender os Dados

Nesta fase precisamos compreender o que temos a nossa disposição. Começamos o processo com a importação das bibliotecas:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Vamos trabalhar com três bibliotecas básicas, que como já mencionamos devemos conhecê-las a fundo: **Pandas** para análise dos dados, **MatPlotLib** e **SeaBorn** para mostrar os gráficos.

Agora precisamos dos dados, para isso usaremos o arquivo *StudentsPerformance.csv*:

```
df = pd.read_csv('StudentsPerformance.csv')
```

Nessa fase compreendemos melhor o que temos na nossa mão, **Pandas** é ideal para essa tarefa. Seu funcionamento é como um "Editor de Planilha", dessa forma que devemos encarar essa biblioteca, sua diferença básica é a nomenclatura de como o *DataFrame* (e não Planilha) é visualizado:

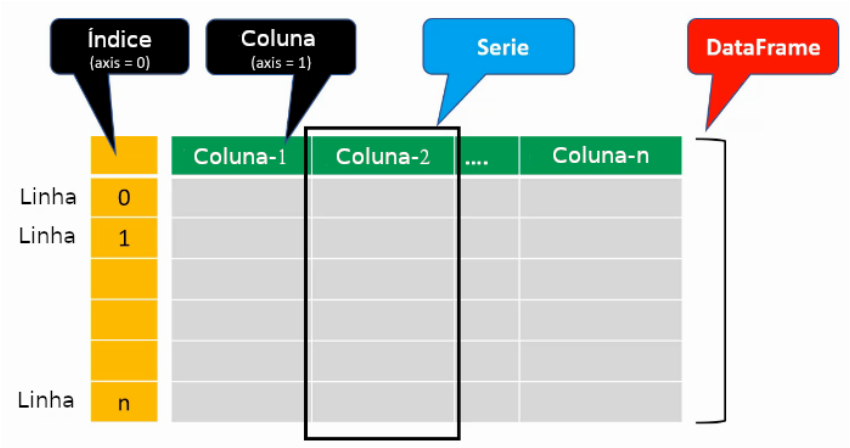


Figura 3.2: Visão Pandas

Uma coluna aqui é vista como uma *Serie* e o *index* é o que mantêm a "cola"das colunas. Dois comandos são básicos para visualizarmos o DataFrame:

```
df.head()
```

Mostra as primeiras linhas, como parâmetro podemos passar a quantidade. E:

```
df.tail()
```

Mostra as últimas linhas e também como parâmetro podemos passar a quantidade. O que temos até o momento? Sabemos é uma base sobre estudantes e as linhas são: gênero, etnicidade, nível de escolaridade dos pais, forma de alimentação, realizou um teste de preparação do curso, nota de matemática, nota de

leitura e nota de escrita.

Então só com esses dois comandos já podemos saber sobre qual assunto iremos tratar: estudantes que realizaram provas e em quais condições. Quantos registros temos a nossa disposição? Ou quais são os nomes das colunas?

```
print("Tamanho: ", df.shape)
print("Nome das Colunas: ", df.columns)
```

Estas variáveis respondem aos questionamentos. De forma mais completa podemos usar:

```
df.info()
```

Que nos mostra inclusive qual o tipo de cada coluna e se esta contém ou não elementos nulos. Temos 3 colunas que são do tipo inteiro (int64), podemos analisá-las com o comando:

```
df.describe()
```

Nos fornece as informações estatísticas básicas como média, desvio padrão, menor valor, máximo, 1º quartil (25%), 2º quartil ou mediana (50%), 3º quartil (75%) e o maior valor. Ou seja, as informações para a montagem de um **BoxPlot**. Vamos montá-lo para melhor visualizar os dados:

```
fig, axes = plt.subplots(1, 3, figsize=(10,4))
axes[0].boxplot(df['math score'])
axes[0].set_title("Matemática")
axes[1].boxplot(df['reading score'])
axes[1].set_title("Leitura")
axes[2].boxplot(df['writing score'])
axes[2].set_title("Escrita")
plt.show()
```

E obtemos como resultado:

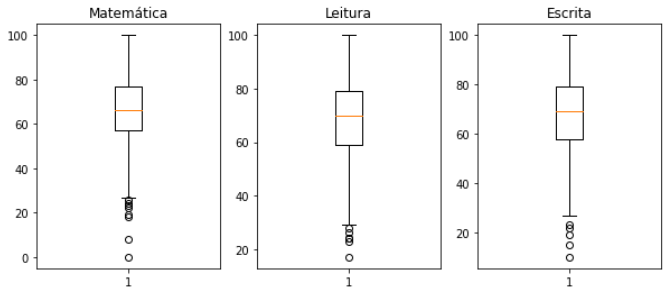


Figura 3.3: BoxPlot das Notas

Boxplot[1] é um gráfico que avalia a distribuição dos dados. É formado exatamente com os dados que mostramos na função *describe()*. Porém suas hastes (inferiores e superiores) se estendem do quartil inferior (ou superior) até o menor valor não inferior (ou superior) ao limite. São calculados da seguinte

[1] Diagrama de Caixa se prefere, foi atribuída ao matemático **John W. Tukey** (1915 –2000), curiosamente algumas literaturas chamam de "*Tukey BoxPlot*", mas se realizar uma pesquisa ninguém sabe ao certo quem criou realmente esse diagrama.

forma:

- Limite inferior: $Q_1 - 1,5 \times (Q_3 - Q_1)$.
- Limite superior: $Q_3 + 1,5 \times (Q_3 - Q_1)$.

Resumidamente é formado da seguinte maneira:

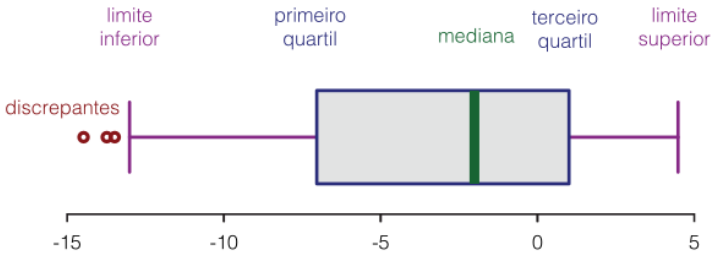


Figura 3.4: Estrutura do BoxPlot

Esses pontos "discrepantes"podem ocorrer acima ou abaixo dos limites, são chamados de *Outliers*. Não é necessariamente um erro, podemos classificá-lo como uma anomalia curiosa e que merece nossa atenção.

3.2.1 Localizar os Outliers

Para achar esses *Outliers* isolamos as três colunas numéricas:

```
X = df.iloc[:, 5:8].values
```

E criamos um novo DataFrame somente com essas com a modificação do seu nome por um número:

```
pd.options.display.float\_format = '{:.1f}'.format
xDF = pd.DataFrame(X)
```

Para quê isso serve? A função *describe()* cria um DataFrame, podemos percorrê-lo, porém fica muito mais simples se cada coluna for um numeral, pois assim podemos usar um comando *for* para isso:

```
z = xDF.describe()
for t in z:
  iqr = z[t][6] - z[t][4]
  extMenor = z[t][4] - (iqr * 1.5)
  extMaior = z[t][6] + (iqr * 1.5)
  print('Na Col. \%d devem estar abaixo de \%.2f e acima de \%.2f' \% (t, extMenor,
    extMaior))
```

E obtemos o seguinte resultado:

```
Na Col.  0 devem estar abaixo de 27.00 e acima de 107.00
Na Col.  1 devem estar abaixo de 29.00 e acima de 109.00
Na Col.  2 devem estar abaixo de 25.88 e acima de 110.88
```

Pelo BoxPlot todos os valores estão abaixos, então para localizá-los:

```
matOutliers = (X[:,0] < 27)
df[matOutliers]
```

Localizamos todas linhas que a nota de matemática (coluna 0) seja abaixo de 27 e mostramos esses registros. Podemos proceder de modo semelhante para as notas de leitura (coluna 1) e escrita (coluna 2) e assim desvendar quem são os *Outliers*.

Podemos também analisar graficamente, e visualizar a Distribuição Normal de cada variável, por exemplo para a nota de Escrita:

```
sns.kdeplot(df['writing score'], shade=True)
plt.show()
```

Obtemos como resultado:

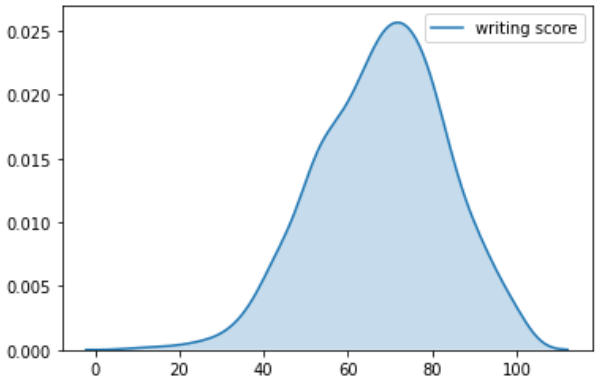


Figura 3.5: Distribuição dos Dados para Nota de Escrita

E assim verificamos como cada atributo numérico se comporta.

3.2.2 Colunas Categóricas

Sabemos que as primeiras cinco colunas do nosso Dataframe são categóricas, porém conforme a função info() o tipo delas estás *object*. É interessante mudarmos para o tipo caractere para evitarmos quaisquer problemas futuros.

```
df['gender'] = df['gender'].astype(pd.StringDtype())
df['race/ethnicity'] = df['race/ethnicity'].astype(pd.StringDtype())
df['parental level of education'] = df['parental level of
    education'].astype(pd.StringDtype())
df['lunch'] = df['lunch'].astype(pd.StringDtype())
df['test preparation course'] = df['test preparation course'].astype(pd.StringDtype())
```

E ao aplicarmos uma nova chamada a função *info()* vemos que os tipos agora estão corretos. Quantos tipos únicos existem em cada uma dessas colunas?

```
df.nunique()
```

Mostra a quantidade de valores não repetidos em cada coluna (inclusive as numéricas). E agora sabemos que temos: 2 gêneros, 5 etnicidades, 6 níveis de escolaridade dos pais, 2 formas de alimentação e 2 tipos para teste de preparação do curso. Mas quem são eles?

```
print("Gênero: ", df['gender'].unique())
print("Etinicidade: ", df['race/ethnicity'].unique())
print("Escolaridade dos Pais: ", df['parental level of education'].unique())
print("Refeição: ", df['lunch'].unique())
print("Realizou Preparatório: ", df['test preparation course'].unique())
```

3.3 Passo 2 - Limpar os Dados

A limpeza dos dados trata de muitos problemas como informação repetida, valores faltantes (que podem ser descobertos por associação) e inconsistentes. Para esse último tipo o pior caso são os nulos. (In)felizmente essa base está horrível para essa fase e assim pegamos um outro arquivo **titanic.csv**:

```
df = pd.read_csv('titanic.csv')
df.head()
```

Repetimos todo o processo da fase anterior para descobrirmos de que se tratam os dados e descobrimos que são os dados dos passageiros (sobreviventes ou não - coluna *Survived* - sendo esta a coluna alvo da base) do famoso **RMS Titanic** que foi pensado para ser o navio mais luxuoso e seguro de sua época e supostamente "inafundável". Como sabemos em sua viagem inaugural de *Southampton* para *Nova Iorque* afundou no dia 14 de abril de 1912 com mais de 1.500 pessoas a bordo. Porém esta base contém apenas 891 registros.

Ao aplicarmos a função info() percebemos que as colunas *Age* (idade), *Cabin* (número da cabine) e *Embarked* (local de Embarque) possuem dados faltantes. Que dados são esses?

```
print(df.isnull().sum())
```

Sabemos que faltam: 177 em **Age**, 687 em **Cabin** e 2 em *Embarked*. Também podemos mostrar exclusivamente os que faltam, isso é útil para quando temos muitas colunas no modelo:

```
null_value_stats = df.isnull().sum(axis=0)
null_value_stats[null_value_stats != 0]
```

Ou ainda criar uma função que nos retorna um Dataframe com a informação mais completa o possível (inclusive com o percentual):

```
def mostrarNulos(data):
  null_sum = data.isnull().sum()
  total = null_sum.sort_values(ascending=False)
  percent = (((null_sum / len(data.index))*100).round(2)).sort_values(ascending=False)
  df_NULL = pd.concat([total, percent], axis=1, keys=['Tot.Nulo', 'Perc.Nulo'])
  df_NULL = df_NULL[(df_NULL.T != 0).any()]
  return df_NULL
```

E ao chamá-la:

```
df_Age = mostrarNulos(df)
df_Age.head()
```

Obtemos como resultado:

	Tot.Nulo	Perc.Nulo
Cabin	687	77.10
Age	177	19.87
Embarked	2	0.22

Figura 3.6: Nulos e Percetual da Base Titanic

Lidar com esses tipos de nulos é complicado pois não temos como consultar e o máximo que podemos fazer é podá-los da nossa base ou atribuir um valor genérico que não afete nosso resultado (como o caso de *Embarked*). Porém **Número da Cabine** é um dado relevante? Essa é a principal pergunta que nos devemos fazer, por exemplo existe algum modelo preditivo que possa nos dizer que se estivéssemos em determinada cabine no navio sobreviveríamos ou não? Entretanto **Idade** é um dado relevante (lembra da frase: mulheres e crianças primeiro), então essa é uma característica que pode ser essencial.

Criar uma função com um gráfico para mostrar, por idade, como estão os dados:

```
def executarGrafico():
  try:
    sns.distplot([df['Age']])
    plt.show()
  except ValueError as err:
    print(err)
```

Agora a cada vez que chamarmos essa função obtemos como resultado:

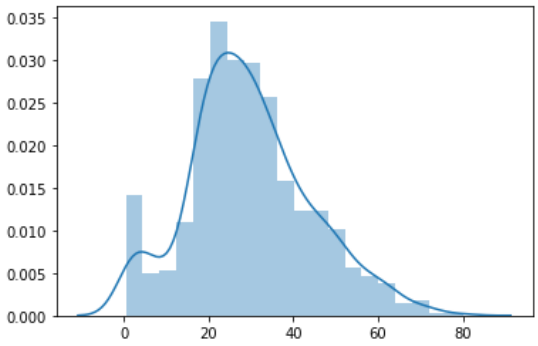


Figura 3.7: Gráfico de Idade do Titanic

> **Dica 3.1: Imputação ou retirada de valores.** Como tratamos de adicionar ou retirar dados na base a cada vez devemos ler novamente os dados do arquivo CSV.

Porém em algumas versões da SeaBorn este pode apresentar erro devido a presença dos nulos, é ideal que os retiremos do *DataFrame* para evitarmos problemas. Em muitas biografias encontramos algo do tipo: "atribuir um valor (preferencialmente *outlier*) para estes tipos". Tentaremos essa técnica com os seguintes comandos:

```
df['Age'].fillna(-25, inplace=True)
executarGrafico()
```

Obtemos como resultado:

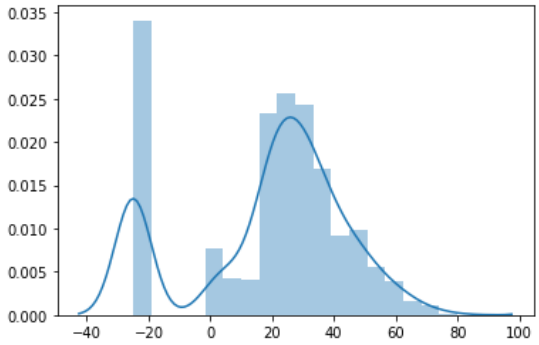


Figura 3.8: Gráfico de Idade do Titanic com Outliers

Nosso gráfico de idade agora ganhou uma coluna, que sabemos com valores não existentes, também podemos atribuir qualquer outro valor como por exemplo a média:

```
df['Age'] = df['Age'].fillna(df['Age'].mean())
executarGrafico()
```

Ou a mediana (função median()) que resultaria em um gráfico completamente esquisito. Sendo assim vamos cortar esses valores:

```
df = df.dropna(axis=0)
executarGrafico()
```

E teremos a seguinte situação:

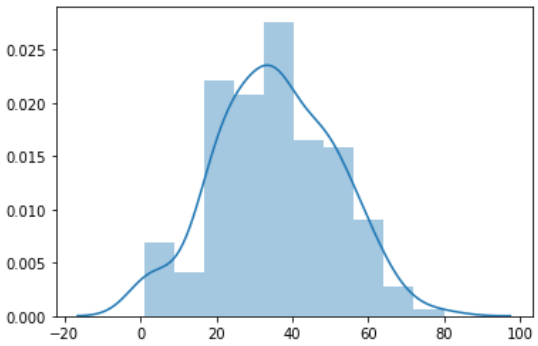


Figura 3.9: Gráfico de Idade do Titanic sem nulos

O que aconteceu? O comando executado eliminou todas as linhas que possuiam valores nulos, a coluna *Cabin* interferiu e nos resultou, conforme pode ser mostrado com a função *info()* em somente 183 registros no total. Ou seja, o corte que devemos aplicar deve ser cirúrgico e somente na coluna idade.

```
df['Age'] = df['Age'].dropna(axis=0)
executarGrafico()
```

O que nos resulta no mesmo gráfico mostrado no início desta e 891 registros. E como citamos podemos arrancar a coluna *Cabin* para que esta não interfira mais:

```
df = df.drop(['Cabin'], axis=1)
```

> **Dica 3.2: Ferramenta para Limpeza dos Dados.** Conhece o **OpenRefine?** é uma ferramenta gratuita dedicada a limpeza e tratamento dos dados baixe um paper sobre o OpenRefine gratuitamente na minha página no Academia.edu (`https://iesbpreve.academia.edu/FernandoAnselmo`).

3.4 Passo 3 - Relacionamento entre as Variáveis

Vamos retomar nossa base de **Estudantes** e verificarmos como as variáveis se relacionam:

```
df = pd.read_csv('StudentsPerformance.csv')
df.corr()
```

E temos o relacionamento o grau de como as variáveis que se relacionam, esse número varia de -1 a 1, sendo o negativo um relacionamento nulo. Porém é muito mais fácil de vermos esse resultado com um Mapa de Calor:

```
rel = df.corr()
sns.heatmap(rel, xticklabels=rel.columns, yticklabels=rel.columns, annot=True)
plt.show()
```

Obtemos como resultado:

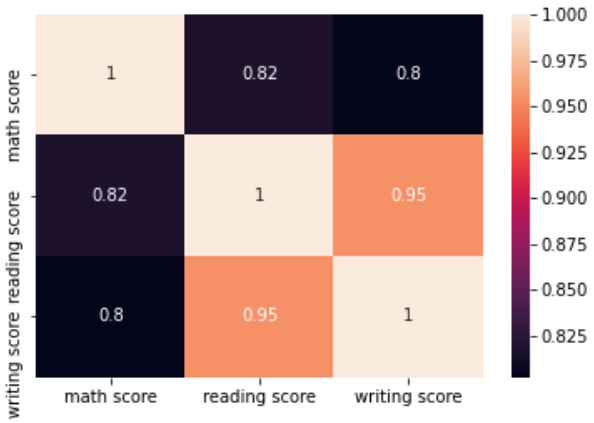


Figura 3.10: Mapa de Calor dos Relacionamentos

Vemos que as notas de Escrita e Leitura possuem um forte grau de relacionamento, como se uma fosse a responsável pela outra. Já a de matemática interfere mais na nota de leitura.

Curiosamente se aplicarmos isso na base do **Titanic** veremos que as colunas mais importantes para *Fare* (sobreviveu) que é nossa variável alvo são: *Fare* que é o valor pago pela passagem e *Parch* que se refere a quantidade de pais. Ou seja, os mais ricos e se a criança tinha ou não os pais a bordo.

Outra maneira de visualizarmos, também de forma gráfica, é através de uma dispersão de valores:

```
sns.pairplot(df)
plt.show()
```

Obtemos como resultado:

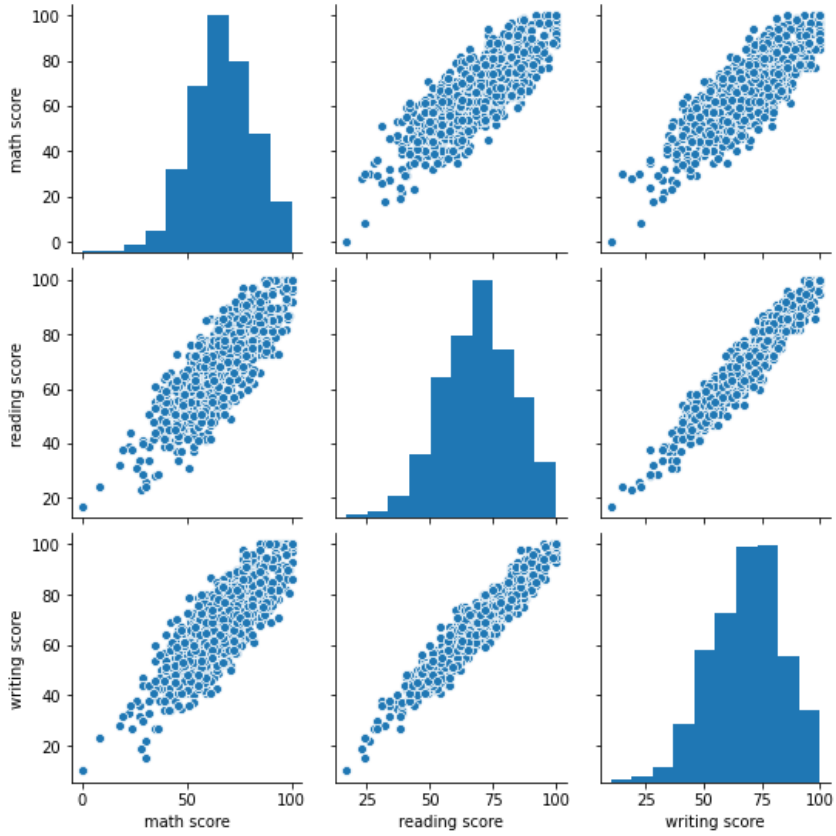


Figura 3.11: Dispersão Associada

Quanto mais juntos aparecem os pontos mais relacionadas estão. Podemos isolar as notas de Escrita e Leitura em um único gráfico, por exemplo:

```
sns.regplot(x='writing score', y='reading score', data=df)
plt.show()
```

Esta função executa um ajuste e plotagem simples com base no modelo de Regressão Linear. E obtemos como resultado:

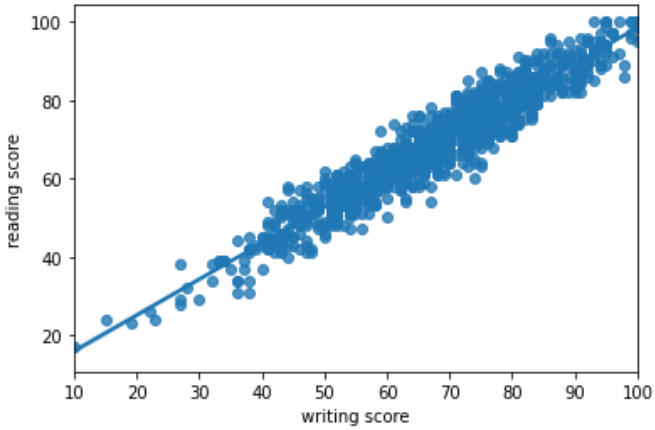


Figura 3.12: Notas de Leitura e Escrita

Porém o mais interessante é colorir os pontos de forma diferente com base em uma variável categórica que achamos que pode ser uma causa (para uma nota alta ou baixa), por exemplo o quanto a alimentação interferiu na nota:

```
sns.lmplot(x='writing score', y='reading score', hue='lunch', data=df)
plt.show()
```

A função *lmplot()* combina *regplot()* com a classe **FacetGrid**. Esta classe auxilia visualizar a distribuição de uma variável, bem como o relacionamento entre várias variáveis separadamente dentro de subconjuntos do seu conjunto de dados. E obtemos como resultado:

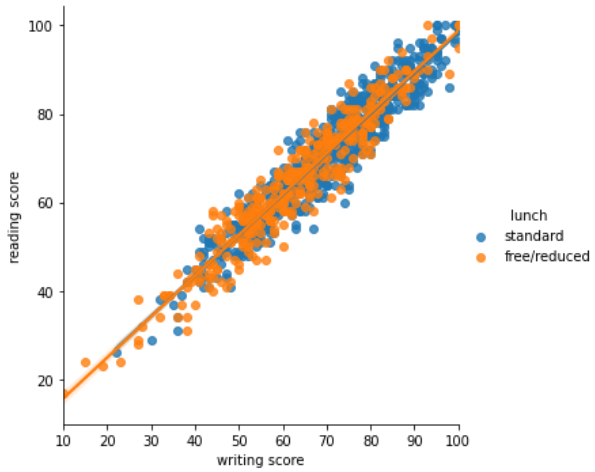


Figura 3.13: Nota associada a Alimentação - RegPlot

Uma melhor forma de visualizar é usar a função *relplot()* que fornece acesso a várias funções diferentes no nível de eixos que mostram o relacionamento entre as duas variáveis com mapeamentos semânticos de subconjuntos:

```
sns.relplot(x='writing score', y='reading score', hue='lunch', data=df)
plt.show()
```

Obtemos como resultado:

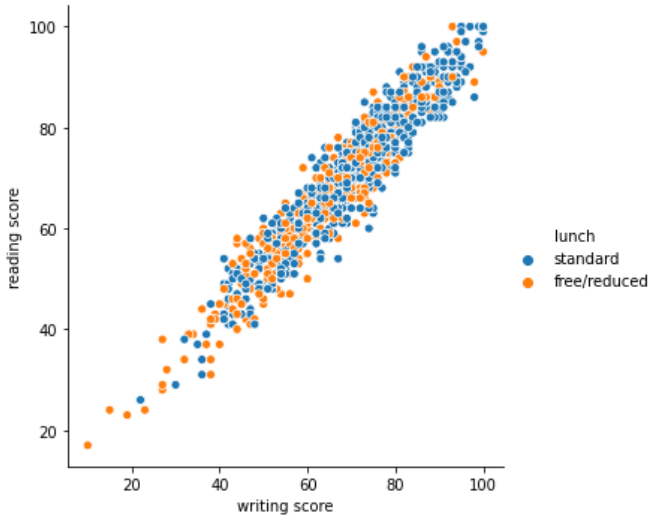


Figura 3.14: Nota associada a Alimentação - RelPlot

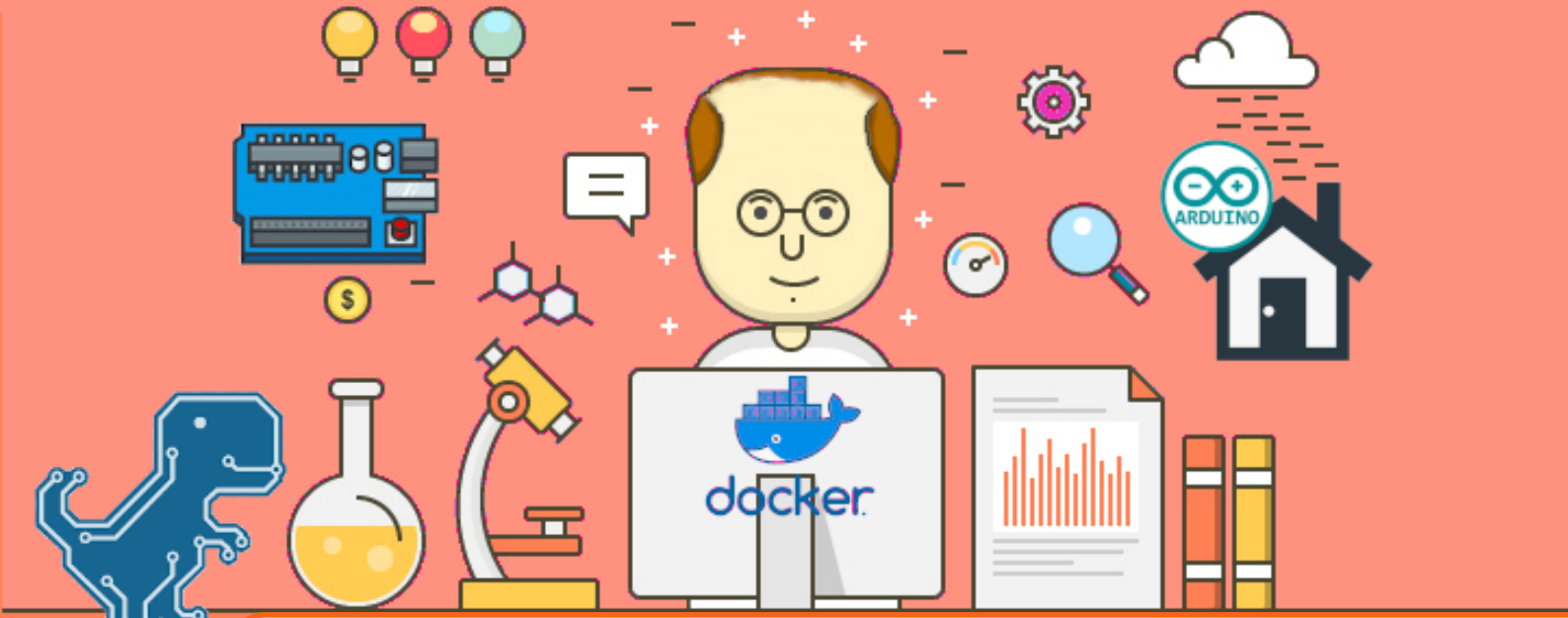
Ou seja, podemos responder várias perguntas apenas com a verificação do relacionamento entre os atributos, tente realizar o mesmo teste com as outras variáveis categóricas e descobrir como essas se comportam na nota, se existe ou não interferência.

3.5 Conclusão

Mantemos em mente que EDA e um aspecto central da *Data Science*, que às vezes é esquecido. O primeiro passo de qualquer ação que tomemos é conhecer os dados: entendê-los, familiarizar-se com eles. Quais são as respostas que estamos tentando obter dos dados? Quais são as variáveis e o que elas significam? Como é a aparência de uma perspectiva estatística? Os dados estão formatados corretamente? Tem valores ausentes? duplicados? E quanto aos *outliers*? Conhecemos eles? Ou seja, devemos responder a esses questionamentos.

É necessário muito trabalho de preparação, porque os dados no mundo real raramente são limpos e homogêneos. Costumamos dizer que 80% do tempo valioso de um cientista de dados é gasto simplesmente com a localização, limpeza e organização dados. Os 20% restantes são para realmente realizar as análises.

Agora estamos prontos para começarmos a explorar diversas bases de dados com a utilização dos modelos.



A. Considerações Finais

F Nenhum computador tem consciência do que faz. Mas, na maior parte do tempo, nós também não. (Marvin Minsky)

Os artigos deste livro foram selecionados das diversas publicações que fiz no Linkedin e encontradas em outros sites que foram nesta obra explicitamente citadas. Acredito que apenas com a prática podemos almejar o cargo de Cientista de Dados, então segue uma relação de boas bases que podemos encontrar na Internet:

- 20BN-SS-V2: `https://20bn.com/datasets/something-something`
- Actualitix: `https://pt.actualitix.com/`
- Banco Central do Brasil: `https://www3.bcb.gov.br`
- Banco Mundial: `http://data.worldbank.org`
- Censo dos EUA (População americana e mundial): `http://www.census.gov`
- Cidades Americanas: `http://datasf.org`
- Cidade de Chicago: `https://data.cityofchicago.org/`
- CIFAR-10: `https://www.cs.toronto.edu/~kriz/cifar.html`
- Cityscapes: `https://www.cityscapes-dataset.com/`
- Criptomoedas: `https://pro.coinmarketcap.com/migrate/`
- Dados da União Europeia: `http://open-data.europa.eu/en/data`
- Data 360: `http://www.data360.org`
- Datahub: `http://datahub.io/dataset`
- DBpedia: `http://wiki.dbpedia.org/`
- Diversas áreas de negócio e finanças: `https://www.quandl.com`
- Diversos assuntos: `http://www.freebase.com`
- Diversos países (incluindo o Brasil): `http://knoema.com`
- Fashion-MNIST: `https://www.kaggle.com/zalando-research/fashionmnist`
- Gapminder: `http://www.gapminder.org/data`

- Google Finance: https://www.google.com/finance
- Google Trends: https://www.google.com/trends
- Governo do Brasil: http://dados.gov.br
- Governo do Canadá (em inglês e francês): http://open.canada.ca
- Governo dos EUA: http://data.gov
- Governo do Reino Unido: https://data.gov.uk
- ImageNET: http://www.image-net.org/
- IPEA: http://www.ipeadata.gov.br
- IMDB-Wiki: https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/
- Kinetics-700: https://deepmind.com/research/open-source/kinetics
- Machine Learning Databases: https://archive.ics.uci.edu/ml/machine-learning-databases/
- MEC Microdados INEP: http://inep.gov.br/microdados
- MS coco: http://cocodataset.org/#home
- MPII Human Pose: http://human-pose.mpi-inf.mpg.de/
- Músicas: https://aws.amazon.com/datasets/million-song-dataset
- NASA: https://data.nasa.gov
- Open Data Monitor: http://opendatamonitor.eu
- Open Data Network: http://www.opendatanetwork.com
- Open Images: https://github.com/openimages/dataset
- Portal de Estatística: http://www.statista.com
- Públicos da Amazon: http://aws.amazon.com/datasets
- R-Devel: https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html
- Reconhecimento de Faces: http://www.face-rec.org/databases
- Saúde: http://www.healthdata.gov
- Statsci: http://www.statsci.org/datasets.html
- Stats4stem: http://www.stats4stem.org/data-sets.html
- Stanford Large Network Dataset Collection: http://snap.stanford.edu/data
- Vincent Rdatasets: https://vincentarelbundock.github.io/Rdatasets/datasets.html
- Vitivinicultura Embrapa: http://vitibrasil.cnpuv.embrapa.br/

Esse não é o fim de uma jornada acredito ser apenas seu começo. Espero que este livro possa lhe servir para criar algo maravilhoso e fantástico que de onde estiver estarei torcendo por você.

A.1 Sobre o Autor

Fortes conhecimentos em linguagens de programação Java e Python. Especialista formado em Gestão da Tecnologia da Informação com forte experiência em Bancos Relacionais e não Relacionais. Possui habilidades analíticas necessárias para encontrar a providencial agulha no palheiro dos dados recolhidos pela empresa. Responsável pelo desenvolvimento de dashboards com a capacidade para analise de dados e detectar tendências, autor de 15 livros e diversos artigos em revistas especializadas, palestrante em seminários sobre tecnologia. Focado em aprender e trazer mudanças para a organização com conhecimento profundo do negócio.

- Perfil no Linkedin: http://www.linkedin.com/pub/fernando-anselmo/23/236/bb4
- Endereço do Git: https://github.com/fernandoans/machinelearning

Machine Learning na Prática

ESTE LIVRO PODE E DEVE SER DISTRIBUÍDO LIVREMENTE

Fernando Anselmo