UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

GABRIEL TEIXEIRA CASCHERA

CRIAÇÃO DE FERRAMENTA PARA AUXÍLIO NA VISUALIZAÇÃO E ANÁLISE EXPLORATÓRIA DE *DATASETS*

Comentado [GC1]: Mantive o título por conta do tempo para mandar a atualização para a secretaria/Sandra

São Carlos
2019

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

GABRIEL TEIXEIRA CASCHERA

CRIAÇÃO DE FERRAMENTA PARA AUXÍLIO NA VISUALIZAÇÃO E ANÁLISE EXPLORATÓRIA DE *DATASETS*

Monografia apresentada ao Departamento de Computação da Universidade Federal de São Carlos para obtenção do título de bacharel em Engenharia de Computação.

Orientação: Prof. Dr. Diego Furtado Silva

São Carlos
2019

FOLHA DE APROVAÇÃO

GABRIEL TEIXEIRA CASCHERA

CRIAÇÃO DE FERRAMENTA PARA AUXÍLIO NA VISUALIZAÇÃO E ANÁLISE EXPLORATÓRIA DE *DATASETS*

Monografia apresentada ao Departamento de Computação da Universidade Federal de São Carlos, para obtenção do título de bacharel em Engenharia de Computação. Universidade Federal de São Carlos. São Carlos, 03 de julho de 2019.

Orientador

_____
Dr. Diego Furtado Silva
Departamento de Computação - UFSCar

Examinadora

_____
Dra. Marcela Xavier Ribeiro
Departamento de Computação - UFSCar

Examinador

_____
Dr. Murilo Coelho Naldi
Departamento de Computação - UFSCar

DEDICATÓRIA

A meus pais e minhas avós, Theresa e Wilma, que sempre me incentivaram a ir mais longe.

AGRADECIMENTO

Agradeço primeiramente a Deus, pela vida, pelas oportunidades e por todos aqueles ao meu lado.

Ao meu pai e à minha mãe, que nunca mediram esforços para que eu chegasse até aqui, apoiando minhas decisões e dando amor e carinho incondicionalmente.

À minha avó, Theresa, que sempre esteve presente em minha vida e acreditou em mim. Obrigado por todo o incentivo e por todos os conselhos que recebi.

À Amanda, minha namorada, que garantiu leveza e clareza em tempos difíceis e em tempos de alegria. Obrigado pelo apoio e por todo o carinho.

Aos meus amigos, que fizeram da universidade e de São Carlos minha segunda casa. Obrigado por todos os risos e pela grande amizade que construímos.

Ao meu orientador, Diego, pela dedicação na construção deste trabalho, pelas conversas e por tudo o que me ensinou.

À Universidade Federal de São Carlos, por toda a capacitação técnica e por todos os ensinamentos e amadurecimento que trouxe à minha vida.

A todos que contribuíram, direta ou indiretamente, para minha formação, o meu mais sincero agradecimento. Obrigado.

RESUMO

Este trabalho tem como objetivo a construção de uma ferramenta para facilitar a análise exploratória e a construção de gráficos a partir dos dados presentes em *datasets*. Devido à crescente geração e disponibilidade de dados, aumentou o interesse por mecanismos automáticos para extração de informações. Com isso, surgiu também a necessidade de melhor entender e analisar conjuntos de dados, os quais são entradas para esses modelos. Assim, são analisados os gráficos do tipo barra, histograma, linha, dispersão, *box* e *violin plots*, os quais são implementados através da linguagem Python e das bibliotecas Pandas, Seaborn e Matplotlib. A ferramenta foi construída e pode ser utilizada para geração das visualizações, de modo que o usuário não interaja com a linguagem de programação. Entretanto, nem todas as visualizações e análises podem ser realizadas através de uma ferramenta, devido a particularidades que os conjuntos de dados apresentam.

Palavras-chave: Análise exploratória de dados. Geração de gráficos. Mineração de dados visual. Python.

ABSTRACT

This work has the goal to build a tool in order to facilitate the exploratory analysis and the construction of charts from the data present in a dataset. Because of the increasing generation and availability of data, the interest for automatic mechanisms to extract information has grown and also has the necessity to better understand and analyze datasets, which are the input to those models. Therefore, histograms, bar, line, scatter, box and violin plots are analyzed, and then are implemented through the Python language and Pandas, Seaborn and Matplotlib libraries. The tool has been built and can be used to generate data visualizations, in a way that the user does not have to interact with programming languages. However, not all the visualizations and analysis can be done through a tool due to particularities that the datasets present.

Keywords: Exploratory data analysis. Charts generation. Visual data mining. Python.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos 20 anos, a quantidade de dados coletados dentro das empresas, em todas as frentes (operações, campanhas de *marketing*, manufatura, entre outros), tornou-se cada vez maior. Além disso, dados externos às organizações também tornaram-se cada vez mais fáceis de serem acessados. Esses fatores, combinados, permitiram o crescimento do interesse na criação de métodos para a extração de informações úteis e conhecimento a partir desses dados (*PROVOST; FAWCETT, 2013*).

Devido a esse interesse, o número de aplicações na área de *Data Science* cresceu. Esta área tem como objetivo extrair conhecimento a partir de dados de forma automatizada, através de processos, princípios e técnicas bem definidas. *Machine learning* (em português, aprendizado de máquina), por sua vez, compreende as áreas de ciências da computação, estatística e outras, incluindo biologia e filosofia, de modo a tentar construir programas computacionais que aprendam sozinhos com experiências passadas (MITCHELL, 1997).

A correta análise e preparação dos dados é essencial para que as aplicações de aprendizado de máquina tenham bons resultados e, com o crescimento da utilização e da disponibilidade de dados, essa etapa torna-se cada vez mais importante. Grande parte do tempo de construção e teste de modelos estatísticos para ferramentas de *machine learning* é gasto justamente nesta etapa de entendimento dos dados.

Em casos de projetos cujos dados não possuem tanta qualidade, isto é, estão em desacordo com o fenômeno a ser estudado/previsto ou estão faltando informações, é de suma importância reconhecer esses detalhes logo no início do projeto. Existem modelos de aprendizado de máquina que assumem determinadas distribuições dos dados de entrada (MITCHELL, 1997). Por exemplo, ao se construir um modelo para seleção de um canal de cobrança para cada cliente de uma base de dados, o primeiro passo é entender se os dados seguem a distribuição esperada pelo modelo escolhido e se está de acordo com o comportamento futuro a ser previsto.

Atualmente, grande parte das empresas utilizam ferramentas simples para realizar análise de dados. Um exemplo são *softwares* para manipulação de planilhas, como o Microsft Office Excel (ANDERSON; SWEENEY; WILLIAMS, 2011). Apesar de ser uma ferramenta bastante poderosa para esse contexto, está longe de ser capaz de

Comentado [GC2]: Definição de machine learning e data science
Data Science for Business?
Tom Something (indicação Diego)?

Comentado [GC3R2]: "Como você tem usado bastante o termo, acho legal introduzir muuuuuito rapidamente o que é. Isso seria legal lá na intro, antes do termo aparecer pela primeira vez. Talvez depois de data Science (que tb poderia ser definido), dizendo que ML é uma parte muito importante e tal." Diego

Comentado [GC4R2]: "Esta área tem como objetivo extrair conhecimento a partir de dados de forma automatizada, através de processos, princípios e técnicas bem definidas." Esta frase tem a mesma referência acima (PROVOST; FAWCETT, 2013). É necessário recoloca-la?

Comentado [GC5]: Adicionar referência do Tom Mitchell falando sobre como alguns modelos esperam alguns tipos de entradas de dados

Comentado [GC6R5]: "Acho que aqui dá para ser mais concreto, como dizer que há modelos de aprendizado de máquina que assumem determinadas distribuições dos dados de entrada (e aí pode citar qqer livro clássico de machine learning, como o do tom mitchell)" Diego

Comentado [GC7R5]:

processar grandes quantidades de dados e construir modelagens robustas, devido às suas limitações de performance e compatibilidade.

Deste modo, é proposta uma ferramenta para facilitar o rápido diagnóstico de *datasets*, através da apresentação de dados básicos do *dataset* e da fácil criação de visualizações de informações gráficas pelo usuário. Tal ferramenta é pensada de modo a ser facilmente conectada a bases existentes, realizando a leitura de arquivos em formato *.csv*. Com isso, ela poderá somar ao entendimento da realidade dos dados, de uma forma mais simples que a utilização de linhas de código.

2 ANÁLISE EXPLORATÓRIA DE DADOS

O termo Análise Exploratória de Dados (em inglês, *Exploratory Data Analysis – EDA*) refere-se ao entendimento das características e geração de inteligência e *insights* com base na exploração dos dados disponíveis. Isto é, através de análises estatísticas e visualização das variáveis presentes nos conjuntos de dados, gerar valor e entendimento do fenômeno que gerou estes dados (TUKEY, 1977).

A realização da EDA traz uma série de benefícios em relação ao entendimento dos dados disponíveis: detecção de erros nos *datasets*, validação de hipóteses, escolha do modelo estatístico a ser aplicado posteriormente e entendimento das variáveis de *input* e *output* (*SELTMAN, 2012*). Assim, é notável a importância da realização da EDA para a aplicação de técnicas de aprendizado de máquina, já que a mesma traz o entendimento necessário para que seja possível escolher entre os diversos modelos disponíveis aquele que melhor atende ao caso com os dados disponíveis.

Segundo matéria da revista Forbes (BERNARD MARR, 2019), a quantidade de dados produzida nunca foi tão grande, chegando a 2.5 quintilhões de bytes de dados gerados todos os dias. Esse número é tão grande especialmente por conta da *Internet of Things* (IoT, internet das coisas), mas outros contribuintes são serviços como previsão do tempo e entretenimento, as redes sociais e os meios de comunicação *online*.

Deste modo, os dados coletados e disponibilizados hoje podem ser trabalhados e entendidos através da EDA, levando a melhores escolhas de modelos de *machine learning*, isto é, modelos que possam gerar bons resultados a partir dos dados de *input* fornecidos, e gerando inteligência e *insights*.

Dentro do mundo dos negócios, é citado o caso da rede Wal-Mart, um dos maiores grupos de varejo do mundo. Antes do furacão Frances atingir o estado da Florida, Estados Unidos, em 2004, segundo matéria do *The NewYork Times* (CONSTANCE L. HAYS, 2019), a CIO (*Chief Information Officer*) do Wal-Mart, Linda Dillman, solicitou à sua equipe análises para entender o que as lojas precisariam ter em estoque para que pudessem enfrentar o evento.

O resultado destes estudos mostrou que não seriam necessários itens de sobrevivência, como lanternas, por exemplo. O principal item a ser estocado seria *Pop-Tarts*, doces industrializados vendidos nos Estados Unidos, surpreendendo os executivos

da rede. Assim, é possível ver que decisões tomadas de forma orientada a dados, através da EDA e *Data Science*, podem beneficiar os resultados da empresa.

3 LEVANTAMENTO DE FERRAMENTAS PARA ANÁLISE DE DATASETS

Foi levantada uma ferramenta disponível para a realização de análise exploratória de dados: DataExplore, construída pelo Dr. Damien Farrell (2016). Além disso, é realizada uma comparação com o *software* Microsoft Office Excel, comumente utilizado no ambiente corporativo como ferramenta para realizar análises exploratórias rápidas.

A ferramenta DataExplore foi proposta devido à necessidade, segundo o autor, de uma forma mais simples de analisar os dados dentro da ciência. Atualmente, as duas formas mais utilizadas são o Excel e os *scripts* criados através de linguagens de programação, tais quais R e Python.

A primeira opção permite que o usuário analise e gere visualizações gráficas, mas apresenta limitações de capacidade, como o número máximo de linhas que uma planilha pode ter (1048576 linhas na versão 16.26, 2019) e opções limitadas de funcionalidades, e dificuldades de implementação, como o cruzamento de bases, em relação à segunda. Além disso, não fornece uma forma segura de análise dos dados, já que esta tarefa ocorre no mesmo ambiente que os dados originais são armazenados, possibilitando mudanças na informação original.

Comentado [GC8]: Melhor explicado e adicionado número máximo de linhas extraído da própria ferramenta

Quanto à segunda opção citada pelo autor, as linguagens de programação são, para muitos usuários, de difícil compreensão e utilização, desfavorecendo a utilização das mesmas. Assim, por mais que sejam uma opção que oferece maleabilidade de uso, pois permitem que o usuário controle por inteiro tanto a análise quanto a geração de visualizações, acabam não sendo escolhidas por alguns usuários.

Deste modo, Dr. Damien Farrell constrói uma ferramenta *open-source* que permite o cruzamento de bases de dados, gerando novas visões das informações, e a geração de visualizações gráficas de forma simples, sem exigir que o usuário tenha habilidades de programação. Ainda, o DataExplore é capaz de armazenar como as visualizações foram geradas, para que seja possível replicar uma mesma tarefa no futuro.

Tais características do DataExplore tornam esta ferramenta muito interessante para utilização, especialmente por usuários que não conhecem tanto de programação e que estão em busca de uma forma de gerar visualizações gráficas de qualidade com facilidade.

Além dos pontos negativos do *software* Excel citados anteriormente (limitações de uso e dificuldades para implementação), esta ferramenta possui um custo de utilização (o que torna soluções *open-source*, como o DataExplore, mais atrativas). Estas características fazem com que a mesma não seja a melhor opção para análise exploratória de dados, mesmo que seja uma das mais utilizadas no mercado.

Ferramentas corporativas de visualização de dados como Tableau[1] e Power BI[2] também podem ser utilizadas para o fim de análise exploratória de dados. Elas oferecerem meios para realizar cruzamento de bases e adicionar colunas calculadas, por exemplo, tendo como *inputs datasets* em bases do SQL Server, arquivos *.xlsm* do Excel, *.accdb* do Microsoft Access, entre outros. Entretanto, devido às suas funcionalidades principais serem a criação de *dashboards online* (i.e., painéis para acompanhamento de dados em tempo real), optou-se por não considerar as mesmas como opções para comparação neste trabalho.

[1] Homepage da ferramenta disponível em https://powerbi.microsoft.com/pt-br/
[2] Homepage da ferramenta disponível em https://www.tableau.com/pt-br

4 VISUALIZAÇÕES GRÁFICAS A SEREM CONTEMPLADAS NA FERRAMENTA

Para atingir o objetivo da ferramenta de permitir realizar análise exploratória dos dados e entender melhor o comportamento das variáveis no *dataset*, foram escolhidas e revisadas visualizações gráficas e funções estatísticas.

Os dados utilizados para geração dos exemplos apresentados nas próximas seções foram obtidos através da plataforma Kaggle[3]. Nela, é possível, entre outras coisas, enxergar uma prévia dos dados, o conjunto de meta-dados e descrições das bases que são disponibilizadas, além de ser possível realizar *download* dos *datasets* em *.csv*.

A geração dos gráficos apresentados como exemplos foi realizada utilizando a linguagem de programação Python. A biblioteca para leitura e organização dos dados foi a Pandas, enquanto que aquela utilizada para construção das visualizações foi a Seaborn.

4.1 CONCEITOS ESTATÍSTICOS BÁSICOS

Para o entendimento das visualizações e das informações dadas através das mesmas, primeiro faz-se necessário entender os principais conceitos estatísticos envolvidos. Somente através dos mesmos é possível compreender o comportamento das variáveis presentes ou derivadas do conjunto de dados em análise e interpretar corretamente as visualizações gráficas.

Essencialmente, existem dois tipos de dados estatísticos: os numéricos (ou quantitativos) e os categóricos (ou qualitativos) (FREUND; SIMON, 2000). Os primeiros podem ser medidos de forma contínua, como a velocidade de um carro ao longo do tempo ou a frequência de transmissão de dados através de uma conexão de rede, ou discreta, como o número de ocorrências de tornados em uma dada região ou a quantidade de vendas realizadas em determinada plataforma de vendas.

Comentado [GC9]: Troquei os exemplos porque tinha tirado da minha cabeça Hahaha

Os categóricos representam classes e possuem um conjunto finito de possibilidades a serem assumidas. Essas categorias podem ser os estados brasileiros (“SP”, ”RJ”, “ES”, etc.) ou o canal de vendas por onde um dado cliente realizou sua

[3] *Dataset* disponível para *download* em: https://www.kaggle.com/justinas/nba-players-data

compra (“loja física”, “*e-commerce*”, “televendas”). A variável categórica pode ser binária, indicando se um dado elemento pertence ou não à classe representada, como no caso de uma variável que indica se o usuário é ou não administrador de um dado sistema (“1” ou “0”, “sim” ou “não”, “*true*” ou “*false*”). Ainda, pode ser ordinal, indicando ordem entre os elementos, como a colocação de pilotos de Fórmula 1 após uma corrida.

Para melhor entender o comportamento das variáveis numéricas e o que elas representam, são utilizadas medidas estatísticas como a média, mediana e a moda, entre outros. Cada uma delas possui uma sensibilidade diferente aos dados e fornecem informações sobre a distribuição dos mesmos (SOUKUP; DAVIDSON, 2002).

A média representa a soma de todos os valores do conjunto dividido pelo número de elementos no mesmo.

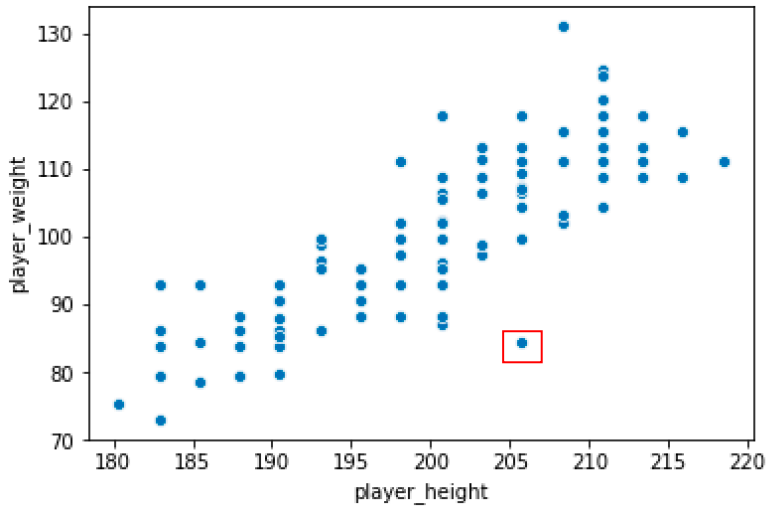
$$x = \frac{\sum_{i=1}^{N} x_i}{N} \qquad (1)$$

Esta medida é muito suscetível a *outliers*, i.e., amostras que fogem ao padrão apresentado pelo conjunto de dados. Um exemplo de *outlier* é apresentado, marcado em vermelho, no

Comentado [GC10]: Aumentei o tamanho das letras para 14. Acha que precisamos mais um?

GRÁFICO **1**. O jogador representado pelo ponto foge aos padrões apresentados pelos outros, já que seu peso é muito menor do que aquele apresentado pelos outros jogadores da sua altura. É muito importante distinguir *outliers* de ruídos nas amostras. Diferente de ruídos, que fogem ao padrão por conterem erros na obtenção dos dados (por exemplo, um mau funcionamento num dado sensor), *outliers* são medidas corretas que fogem às características padrões do grupo.

GRÁFICO 1 *– Altura x Peso de jogadores da NBA com mais de 30 anos na temporada 2016-17*



Fonte: (JUSTINAS, 2019)[4]

Outra medida comum para essas análises é a mediana (BRUCE; BRUCE, 2017). Ela consiste no valor da amostra no meio de um conjunto ordenado, isto é, amostra a qual 50% das outras amostras estão abaixo dela (num conjunto ordenado). Por exemplo, numa sala de aula com 9 (nove) alunos, com suas respectivas notas apresentadas na TABELA 1, a mediana será a nota do quinto aluno, ou seja, 7,5, pois 50% das amostras estão abaixo deste. Caso o conjunto de dados apresente número par de elementos, considera-se a média entre o último elemento da primeira metade e o primeiro elemento da segunda metade.

TABELA 1 *– Exemplo de notas de 9 alunos*

Aluno	1	2	3	4	5	6	7	8	9
Nota	5,0	5,2	6,3	7,0	7,5	7,7	8,2	8,3	8,5

A mediana não é sujeita à influência dos *outliers*, diferentemente da média, pois ela não leva em consideração o valor de todas as amostras, somente sua posição no conjunto. Utilizando o exemplo das notas dos alunos, caso o último aluno tivesse obtido

[4] Disponível em: https://www.kaggle.com/justinas/nba-players-data

uma nota 10 ao invés de 8,5 e o primeiro uma nota 0 ao invés de 5, a mediana permaneceria a mesma.

A mediana é coincidente com o segundo quartil, que representa o percentil 50%. Percentil 50% significa que 50% das amostras possuem valor abaixo daquele (e, consequentemente, que os outros 50% possuem valor acima). De maneira geral, pode-se dizer que um percentil x% representa o valor que x% das amostras estão abaixo dele e os outros (100-x%) estão abaixo. Deste modo, os primeiro, segundo e terceiro quartis são representados, respectivamente, pelos percentis 25%, 50% e 75%. Define-se a distância interquartis como sendo a diferença entre o terceiro e o primeiro quartis.

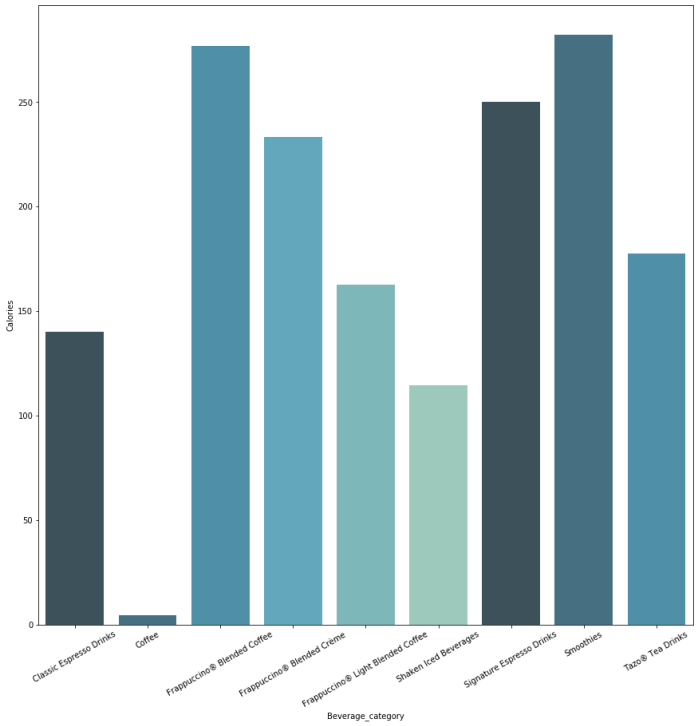
Por fim, as visualizações e análises de dados podem ser divididas em dois tipos: univariada e multivariada. O primeiro trata de atributos que variam com relação a somente um outro atributo, enquanto o segundo trata de atributos que variam conforme duas ou mais outras variáveis.

Dadas estas definições, são apresentadas algumas formas de visualizações de dados ao longo das próximas subseções.

4.2 GRÁFICOS DE COLUNA E HISTOGRAMAS

As visualizações apresentadas em formato de coluna são utilizadas para comparação de dados contínuos sobre dados discretos. Desta forma, é possível comparar categorias, já que o eixo y deste tipo de gráfico apresenta o valor resultante do agrupamento das categorias presentes no eixo x. O GRÁFICO 2 mostra a quantidade média de calorias por bebida para cada um dos grupos de bebidas da rede Starbucks. Neste caso, o eixo y representa a média do agrupamento dos dados pertencentes a cada uma das nove categorias presentes no eixo x. Através desta visualização, pode-se ver que bebidas na categoria de café (*Coffee*), são as menos calóricas, enquanto bebidas mais sofisticadas, como *Smoothies*, apresentam mais calorias por porção.

GRÁFICO 2 *- Média de calorias por grupo de bebidas da rede Starbucks*



Fonte: (STARBUCKS, 2017)[5]

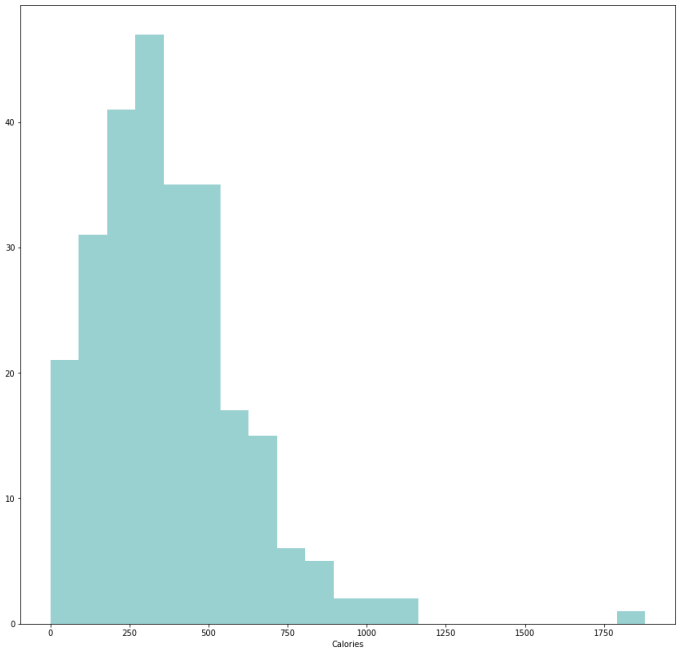
Outro uso muito comum de gráficos de barra é a construção de histogramas. Esta visualização consiste no número de ocorrências de um dado valor presente no conjunto de dados. Através dos histogramas, é possível enxergar como ocorre a distribuição de frequência das amostras acerca de uma dada característica do *dataset*. Logo, é possível entender se uma variável possui seus valores distribuídos de forma balanceada ou não, o que pode influenciar tanto as métricas calculadas no conjunto (média e mediana, por exemplo), assim como decisões tomadas a partir desses dados.

O GRÁFICO 3 representa um histograma que mostra a quantidade de itens do menu da rede americana de *fast food* McDonald's por quantidade de calorias presentes nos mesmos. Desta forma, enxerga-se a quantidade de itens para cada faixa de calorias.

[5] Disponível em: https://www.kaggle.com/starbucks/starbucks-menu/

A maior parte dos itens possui entre 250 e 375 calorias, sendo que alguns poucos possuem mais de 1000 calorias. A média e a mediana são, respectivamente, 368,27 e 340 calorias.

GRÁFICO 3 *- Quantidade de itens do menu da rede McDonald's por quantidade de calorias*

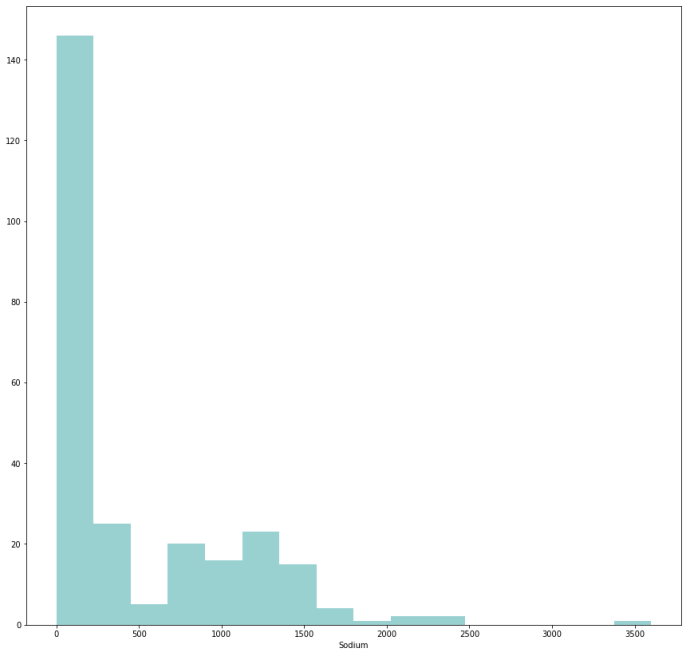


Fonte: (MCDONALD'S, 2017)[6]

Por outro lado, a distribuição da quantidade de itens por quantidade de sódio não apresenta o mesmo comportamento, como mostra o GRÁFICO 4. Existem muitos itens com menos de 250mg de sódio, mas também existem muitos itens acima desta quantidade, sendo que alguns deles apresentam valores acima de 2000. Os valores de média e mediana comprovam esta distribuição não uniforme. Seus valores são, respectivamente, 495,75 e 190. Este também é um exemplo de como a média é mais influenciada por *outliers* que a mediana.

[6] Disponível em: https://www.kaggle.com/mcdonalds/nutrition-facts/

GRÁFICO 4 *- Quantidade de itens do menu da rede McDonald's por quantidade de sódio*



Fonte: (MCDONALD'S, 2017)[7]

4.3 GRÁFICOS DE LINHA

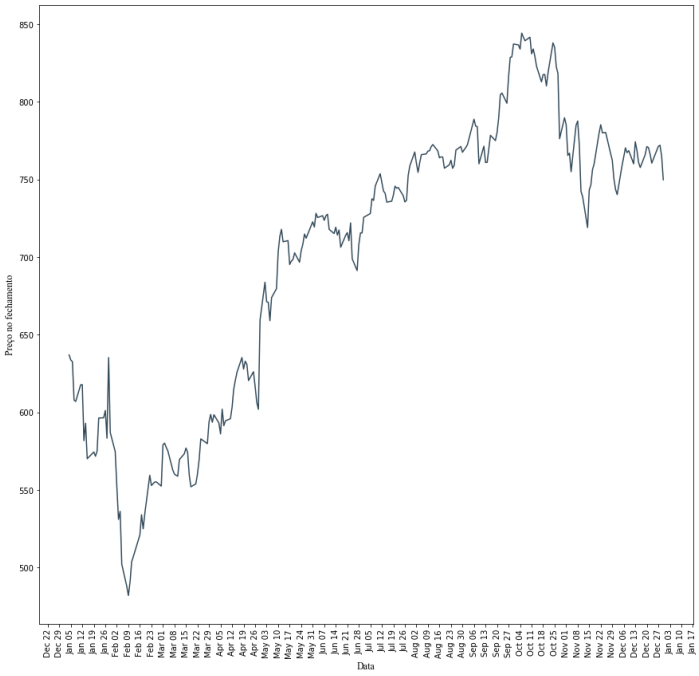
Esta visualização é composta de dois eixos: x (horizontal) e y (vertical). O eixo x pode ter seus dados compostos por valores contínuos ou discretos, enquanto o y deve pertencer ao domínio contínuo.

Situações que são comumente representadas utilizando-se gráficos de linhas são aquelas que mostram variação de valores ao longo do tempo. Por exemplo, o GRÁFICO 5 mostra o preço de fechamento (i.e., o valor da ação no horário em que a bolsa é encerrada) das ações da empresa americana Amazon (sigla AMZN) na bolsa de Nova Iorque (*NASDAQ*) durante o ano de 2016, com granularidade diária. Nesta visualização,

[7] Disponível em: https://www.kaggle.com/mcdonalds/nutrition-facts/

o eixo x representa a dimensão temporal (discreta) e o y os preços de fechamento (contínuo).

GRÁFICO 5 *- Preço de fechamento das ações da Amazon (AMZN) na bolsa de Nova Iorque durante o ano de 2016*

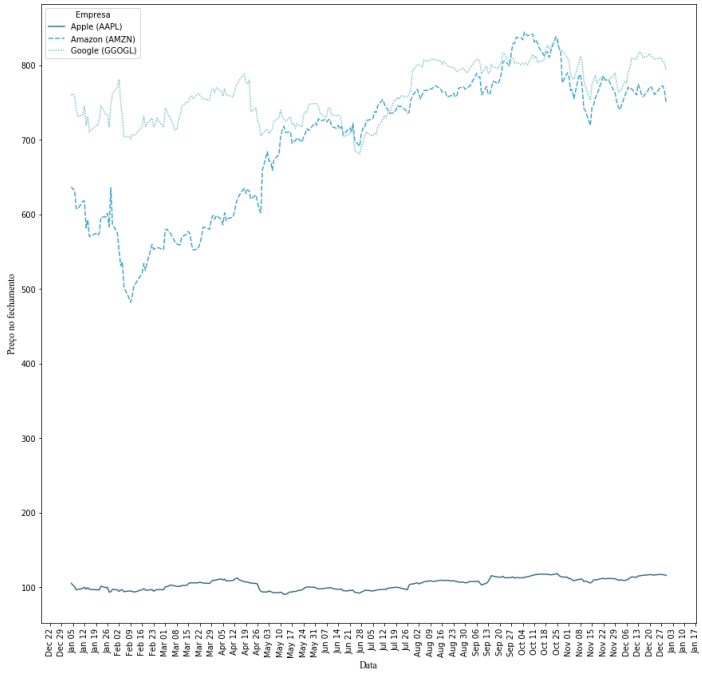


Fonte: (GAWLIK, 2017)[8]

Gráficos de linha também podem ser utilizados para fins de comparação entre diferentes curvas, plotadas juntas numa visualização comum. O GRÁFICO 6 apresenta as mesmas informações que o Gráfico 6, porém para mais duas empresas além da Amazon: Apple (AAPL) e Google (GOOGL). Deste modo, é possível realizar a comparação entre as três empresas ao longo de 2016 (em termos do preço de fechamento diário das ações das três empresas).

[8] Disponível em: https://www.kaggle.com/dgawlik/nyse/

GRÁFICO 6 *- Preço de fechamento das ações da Apple (AAPL), Amazon (AMZN) e Google (GOOGL) na bolsa de Nova Iorque durante o ano de 2016*



Fonte: (GAWLIK, 2017)[9]

4.4 GRÁFICOS DE DISPERSÃO (*SCATTER PLOTS*)

Gráficos de dispersão (no inglês, *scatter plots*) apresentam dois (x e y) ou três eixos (x, y e z), os quais cada um representa uma diferente variável do conjunto de dados. Deste modo, é possível realizar a comparação de uma variável com outra e enxergar se existe alguma correlação entre elas.

O

[9] Disponível em: https://www.kaggle.com/dgawlik/nyse/

GRÁFICO 7 apresenta um exemplo de gráfico de dispersão. Ele mostra as variáveis de receita total do estado dos Estados Unidos (em milhões de dólares americanos) contra o total de alunos matriculados em escolas no estado, em 2015. Assim, pode-se enxergar que, conforme o número de alunos matriculados cresce, a receita do estado também, indicando uma correlação positiva entre as variáveis.

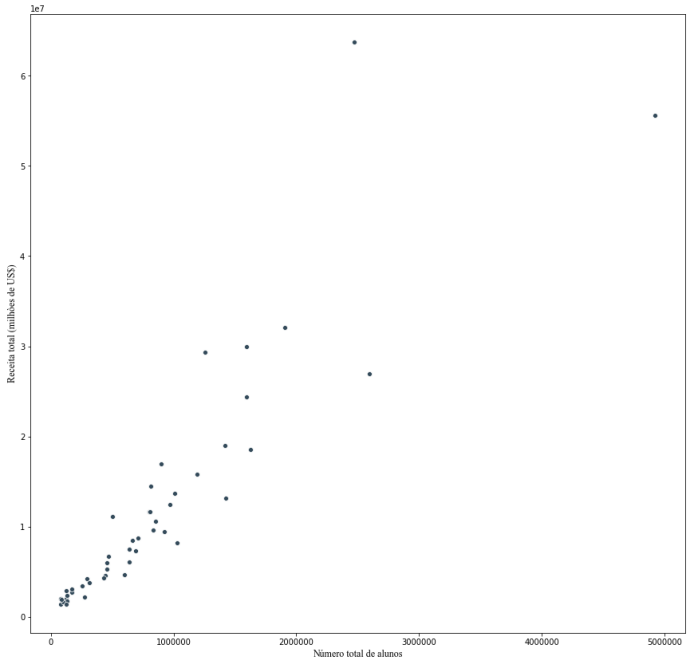
Utilizando o coeficiente de Pearson[10], definido na Equação 2 (BRUCE; BRUCE, 2017), onde os desvios em relação à média das duas variáveis são divididas pela multiplicação dos seus desvios padrões (s_x e s_y, definidos pela Equação 3), o grau de correlação entre o número de alunos matriculados e a receita total do estado é de 0,897577. Dado que a correlação de Pearson possui valores entre -1 e 1, sendo que os valores negativos indicam uma correlação negativa e os positivos uma positiva, pode-se afirmar a correlação positiva entre as variáveis em questão.

$$r = \frac{\sum_{i=1}^{N}(x_i - x)(y_i - y)}{(N-1)s_x s_y} \quad (2)$$

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - x)^2}{(N-1)}} \quad (3)$$

[10] Aplicado através da função *Pandas.DataFrame.corr()*, no Python

***GRÁFICO 7** - Receita total por número total de alunos matriculados na escola por estado dos Estados Unidos - 2015*



Fonte: (GARRARD, 2019)[11]

4.5 ***BOX PLOT***

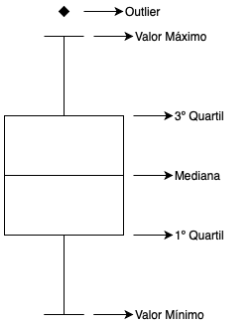
O *box plot* mostra a distribuição das amostras através de suas medidas de mediana, primeiro e terceiro quartis, além de mostrar os valores máximo e mínimo do conjunto, bem como *outliers*. A

[11] Disponível em: https://www.kaggle.com/noriuk/us-education-datasets-unification-project/

FIGURA 1 explica estes elementos de forma visual.

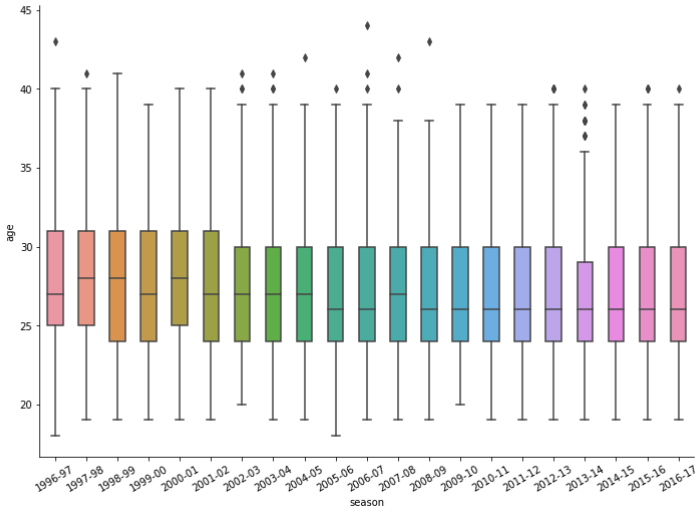
Com esta visualização, é possível enxergar a distribuição do conjunto, analisando quão compacto o mesmo se encontra e se é um conjunto simétrico, permitindo tirar conclusões sobre os dados serem ou não enviesados.

FIGURA 1 *– Características do box plot*



O GRÁFICO 8 mostra a distribuição das idades dos jogadores da NBA ao longo das temporadas utilizando *box plots*. A partir destes, pode-se ver que está é uma variável praticamente simétrica para quase todas as temporadas do esporte, com destaque para a temporada de 2007-08. Ainda, pode-se inferir que a idade máxima dos jogadores caiu ao longo das temporadas.

GRÁFICO 8 *– Distribuição das idades dos jogadores da NBA ao longo das temporadas – 1996 a 2016*



Fonte: (JUSTINAS, 2019)[12]

[12] Disponível em: https://www.kaggle.com/justinas/nba-players-data

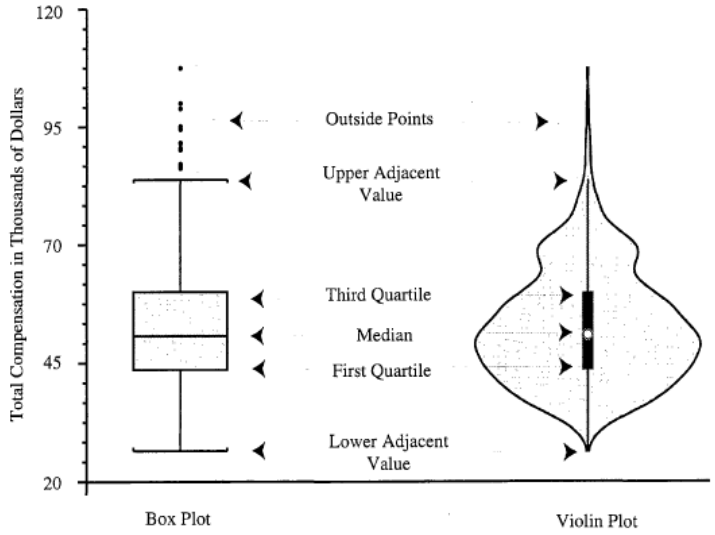
4.6 GRÁFICOS DE VIOLINO (*VIOLIN PLOTS*)

Gráficos de violino foram definidos por Hintze e Nelson (1998) e são muito próximos a *box plots*, como mostra a FIGURA 2. Entretanto, apresentam duas curvas simétricas nas suas laterais, as quais representam o traço de densidade, definido como sendo, para um dado ponto central *x* e um intervalo de tamanho *h*,

$$d(x|h) = \frac{\sum_{i=1}^{n} \delta_i}{nh} \qquad (4)$$

onde *n* representa o tamanho da amostra de dados e δ_i assume valor 1 (um) quando o i-ésimo valor do conjunto de dados está dentro do intervalo [*x – h/2, x + h/2*] e assume valor 0 (zero) quando o contrário. Intervalos muito grandes (*h>40%*), tendem a produzir curvas muito suaves, as quais podem não condizer com a realidade da amostra, enquanto intervalos muito pequenos (*h<10%*), podem ocasionar em curvas que individualizam os dados, dando a impressão de quebras.

***FIGURA 2** - Comparação do box plot com o violin plot*



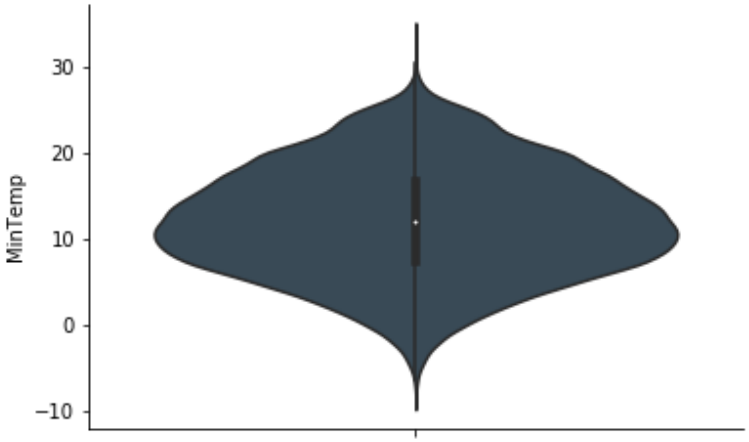
Fonte: (HINTZE; NELSON, 1998)

Vantagens deste tipo de visualização são o maior entendimento da distribuição dos dados em relação ao *box plot*, em especial para detecção de *clusters*, de picos e de vales nas amostras. Entretanto, como os *outliers* são representados na cauda longa do

traço de densidade no caso dos gráficos de violino, os *box plots* são uma melhor opção para apresentação de *outliers*.

O GRÁFICO 9 mostra a distribuição da temperatura mínima diária na Austrália, entre os anos de 2008 e 2017, através de um gráfico de violino. Deste modo, é possível observar a distribuição em torno da mediana, marcada com um círculo branco, a qual tende a uma distribuição normal (simétrica em torno da média).

GRÁFICO 9 *- Temperatura mínima diária na Austrália entre 2008 e 2017*



Fonte: (YOUNG, 2019)[13]

[13] Disponível em: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package/

5 ESCOLHA DE LINGUAGEM DE PROGRAMAÇÃO PARA DESENVOLVIMENTO

Comentado [GC11]: Não coloquei as libs nas referências, só coloquei nas notas de rodapé. Deveria colocar?

Para o desenvolvimento da ferramenta proposta, foi escolhida a linguagem de programação Python. Essa linguagem possui alta compatibilidade com fontes de dados, a exemplo do SQL Server, através da biblioteca *pyodbc*[14], de arquivos *.csv* e do Excel, ambos com funções da Pandas[15].

Segundo o índice TIOBE de maio de 2019 (TIOBE, 2019), realizado pela empresa holandesa homônima focada em qualidade de códigos, Python é a quarta linguagem de programação mais popular em 2018 e 2019, ficando atrás de Java, C e C++. Este índice é calculado com base no número de pesquisas pelas linguagens de programação em ferramentas de busca, como Google, e outras fontes de informação, como a Amazon e a Wikipédia.

Entretanto, de acordo com pesquisa realizada pelo *site* KDnuggets (2019), Gregory Piatetsky-Shapiro (Ph.D), o Python foi elegido como a linguagem mais popular para projetos de *machine learning* e *big data*. A pesquisa levou em conta a opinião de 1800 participantes, que escolheram as ferramentas que utilizavam para projetos na área de *Data Science*. As opções dadas para os participantes contemplavam, entre outras, Python, R e SQL. Um terço dos usuários que votaram em somente uma ferramenta foram removidos, de modo a prevalecer a opinião de usuários ativos dentro da área de estudo em questão.

Deste modo, devido à sua popularidade, facilidade de aprendizado (PYTHON, 2019), implementação e codificação, alta disponibilidade de bibliotecas e pacotes e compatibilidade com diversas fontes de dados e ferramentas, Python foi escolhida para a implementação deste trabalho.

[14] Documentação da biblioteca disponível em https://github.com/mkleehammer/pyodbc/wiki
[15] Documentação da biblioteca disponível em https://pandas.pydata.org/pandas-docs/stable/

6 LEVANTAMENTO DE BIBLIOTECAS DE VISUALIZAÇÃO DE DADOS NO PYTHON

Com a escolha do Python como linguagem de programação para implementação das funções da ferramenta, é necessário encontrar um biblioteca que seja capaz de fornecer as visualizações gráficas necessárias (descritas na Seção 4). Para realização das visualizações, foram levantadas algumas bibliotecas do Python e exploradas suas principais funcionalidades e características.

A seguir, são apresentados o que foi encontrado em cada uma das bibliotecas e indicada a escolha para a construção da ferramenta.

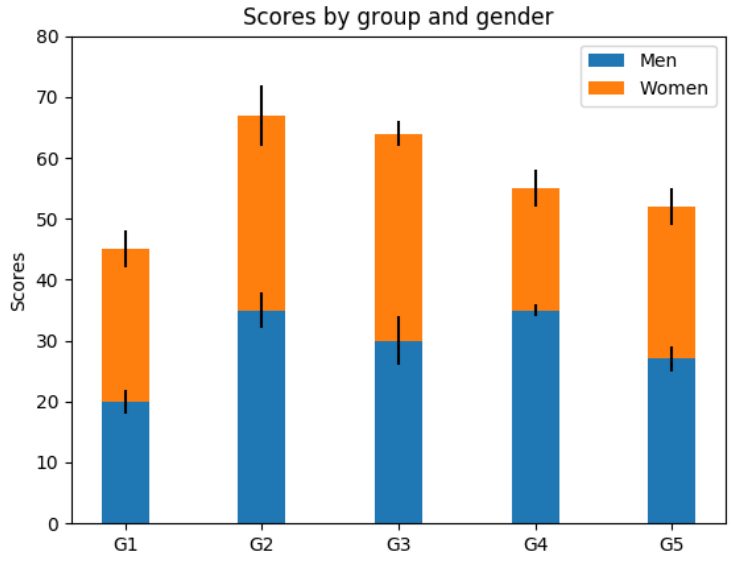
6.1 MATPLOTLIB

Primeira biblioteca dedicada à visualização de dados no Python, pode ser utilizada para a geração de vários tipos de gráficos, desde histogramas até gráficos de linha. Consegue ser utilizada para criação de gráficos com poucas linhas de código, facilitando muito o desenvolvimento de *software*. Suas visualizações não são dinâmicas, o que dificulta a simples interação do usuário com os gráficos gerados.

A

FIGURA **3** mostra um exemplo de gráfico de barras empilhadas gerado utilizando-se a Matplotlib e disponibilizado na galeria *online* desta biblioteca. Nesta fonte, também é possível encontrar o código Python para gerar essa visualização (HUNTER, 2019).

FIGURA 3 - *Exemplo de gráfico de barras empilhadas gerado pela Matplotlib*



Fonte: (HUNTER, 2019)[16]

6.2 SEABORN

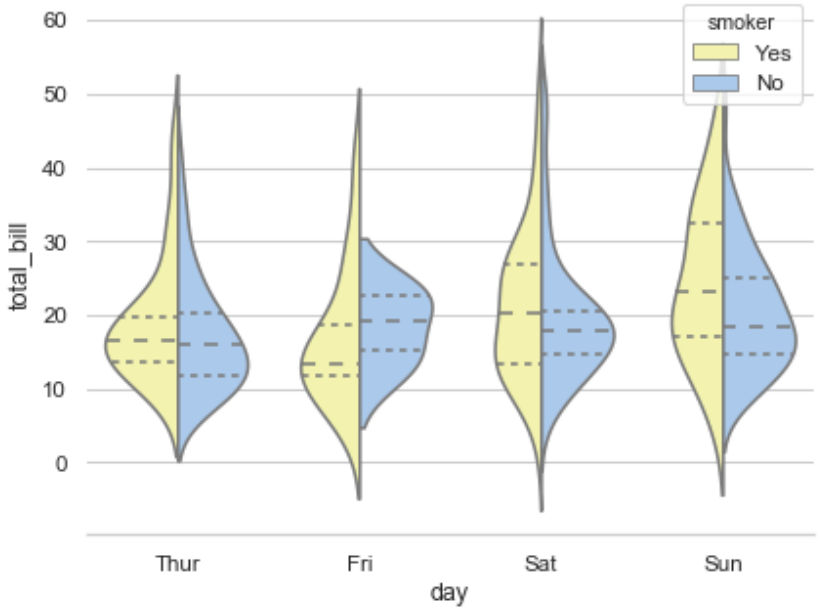
Construída em cima do Matplotlib, apresenta maior facilidade para utilização de *plots* mais complexos, como mapas de calor e gráficos violinos, por ser uma biblioteca mais alto nível. Além disso, suas cores e apresentações são consideradas mais bonitas que aquelas da biblioteca anterior. Como o Matplotlib, suas visualizações são estáticas. Ainda, esta biblioteca foi desenvolvida para funcionar muito bem com a Pandas, utilizada para manipulação dos dados em *dataframes*.

A

[16] Disponível em: https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html

FIGURA **4** apresenta exemplo de um gráfico de violino (*violin plot*), dado como exemplo na galeria *online* da Seaborn.

***FIGURA 4** - Exemplo de gráfico gerado pela biblioteca Seaborn*



Fonte: (SEABORN, 2019)[17]

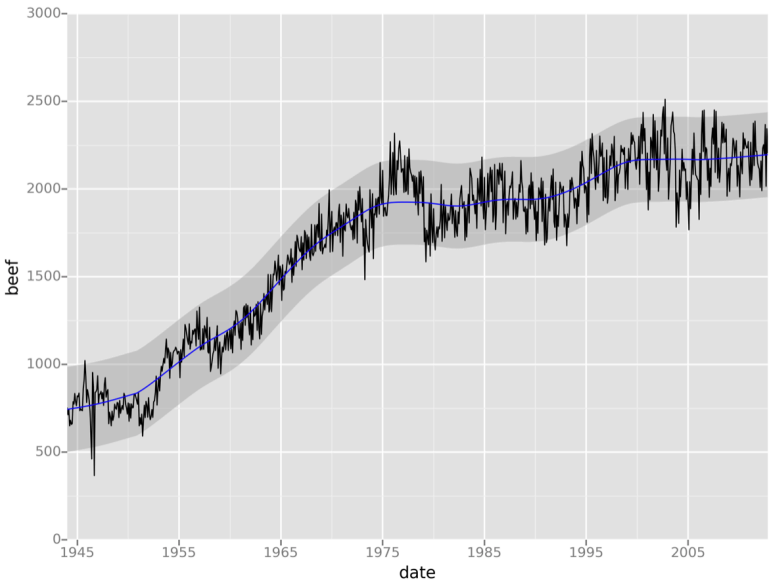
6.3 GGPLOT

Baseado nas ideias do livro *The Grammar of Graphics* e na biblioteca *ggplot2* do R, o *ggplot* permite a criação de gráficos menos personalizados, focando na reutilização de código e aceleração da geração dos gráficos.

Esta biblioteca conversa muito com a biblioteca de processamento de dados *Pandas* e seu modo de guardar os dados em *dataframes*. Sua implementação faz com que seja possível criar gráficos em camadas, adicionando *plots* um em cima do outro.

[17] Disponível em: https://seaborn.pydata.org/examples/grouped_violinplots.html

FIGURA 5 *- Exemplo de visualização da biblioteca ggplot*



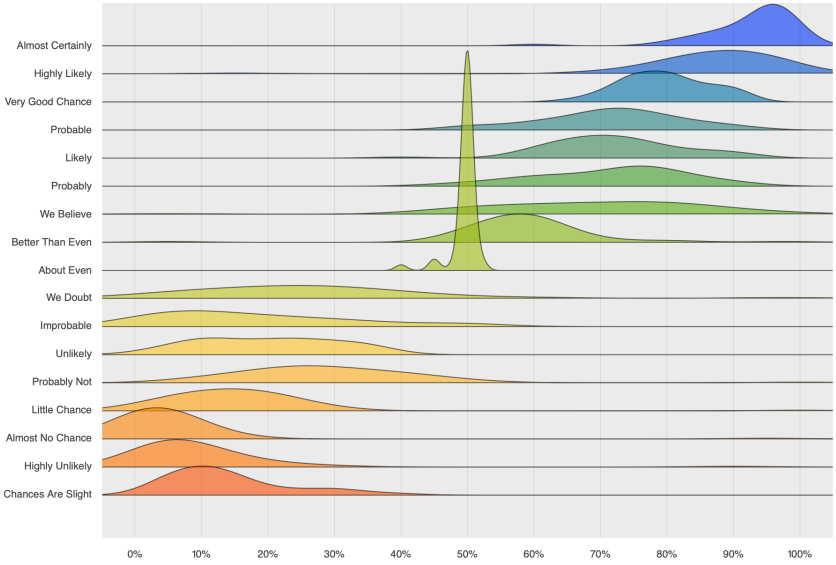
Fonte: (GGPLOT, 2019)[18]

6.4 **BOKEH**

Implementada com base no *The Grammar of Graphics*, como o *ggplot*, é uma biblioteca nativa do Python. Permite a criação de gráficos interativos, com ferramentas de *zoom*, movimentação, atualização dos dados e opção de salvar a visualização. Permite que os *plots* sejam exportados em JSON, documentos HTML ou em interfaces *web*. Os dados a serem apresentados podem ser fornecidos em listas do próprio Python ou estruturas de dados do *NumPy* e *Pandas*.

[18] Disponível em: http://ggplot.yhathq.com

***FIGURA 6** - Exemplo de visualização gerada pela biblioteca Bokeh*



Fonte: (BOKEH, 2019)[19]

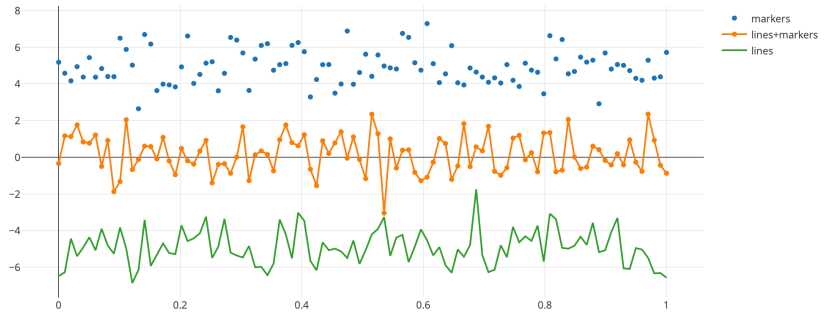
6.5 **PLOTLY**

Esta biblioteca é construída em cima da *D3.js*, que é uma biblioteca JavaScript implementada para conectar quaisquer dados a um *Document Object Model* (DOM), o qual, por sua vez, pode ser transformado conforme o necessário, gerando vários tipos de visualizações. Deste modo, ela é capaz de, com poucas linhas de código, gerar visualizações gráficas completas, com as quais o usuário pode interagir: aplicar e tirar *zoom*, navegar pelo gráfico, consultar o valor de cada ponto, entre outros.

Esta á a biblioteca utilizada para geração das visualizações gráficas de dados no Dash, um *framework* em Python para criação de *dashboards* e relatórios interativos *web* (*Javascript*).

[19] Disponível em: https://bokeh.pydata.org/en/latest/docs/gallery/ridgeplot.html

***FIGURA 7** - Exemplo de visualização gerada pela biblioteca Plotly*



Fonte: (PLOTLY, 2019)[20]

6.6 CONSIDERAÇÕES FINAIS

Para os fins deste trabalho, a biblioteca escolhida foi a Seaborn. Com a sua utilização, será possível gerar as visualizações escolhidas pelo usuário, contemplando todas aquelas descritas na Seção 4. Devido à fácil customização de seus gráficos e à habilidade de transformação das visualizações para arquivos de imagem (como o formato *.png*), suas saídas poderão ser utilizadas numa interface de usuário, seja esta construída em Python ou com *frameworks web* (e.g., React).

[20] Disponível em: https://plot.ly/python/line-and-scatter/

7 CONSTRUÇÃO DA FERRAMENTA

De modo a facilitar a reprodução e melhoria dos resultados obtidos com este trabalho, os códigos Python para construção da ferramenta (explicados a seguir) e geração das visualizações apresentadas anteriormente foram disponibilizados num repositório do GitHub[21]. Os códigos foram disponibilizados no diretório "codigo", enquanto as visualizações mostradas nas seções anteriores (geradas por *data_visualizations_generator.py*) estão em "visualizacoes".

A ferramenta para auxílio e análise exploratória de *datasets* foi construída utilizando-se a linguagem Python e a biblioteca de visualização Seaborn. As configurações do ambiente utilizado para desenvolvimento são as que seguem:

- Sistema Operacional: MAC OS Mojave – versão 10.14.2
- Processador e RAM: Intel Core i5, 8GB RAM
- Linguagem de Programação: Python – versão 3.6.5
- Bibliotecas utilizadas: Seaborn – versão 0.9.0, Pandas – versão 0.23.0

Foi desenvolvida uma interface gráfica simples para a ferramenta, de modo a permitir os testes. Para esse fim, foi utilizada a biblioteca *tkinter*. Ela implementa uma camada orientada a objetos em cima do *Tcl/Tk*, uma linguagem e um *toolkit*, respectivamente, para desenvolvimento de aplicações gráficas *desktop* (PYTHON, 2019). De qualquer modo, a ferramenta foi pensada de forma a permitir que ela seja acoplada a uma interface gráfica futuramente, seja esta construída em Python ou utilizando *frameworks web*, como Angular, React e outros.

Após a definição da linguagem Python como aquela a ser utilizada no desenvolvimento, do levantamento das visualizações gráficas que seriam implementadas e da escolha da biblioteca Seaborn para esse fim, foram construídas as funções de impressão dos gráficos no *script* Python *data_visualization_tool.py* (APÊNDICE A). Com isso, buscou-se manter as funções responsáveis pelas visualizações isoladas daquelas para importação de dados e construção da interface com o usuário, garantindo a possibilidade de acoplamento com uma interface gráfica robusta futuramente.

[21] Acesso através do link: https://github.com/gcaschera/tcc_enc_ufscar

Foram criadas seis funções, uma para cada tipo de visualização apresentada na Seção 4 (histogramas e gráficos de barras foram divididos em duas funções distintas). Isso foi necessário devido à chamada das funções do Seaborn, as quais variam conforme o gráfico desejado. Entretanto, cada uma dessas funções implementadas chama, ao final, por uma única função, *print_plot_png()*, a qual realiza a nomeação dos eixos e do gráfico em si e exporta a figura para png. Com isso, é possível alterar facilmente o formato de exportação, caso seja necessário.

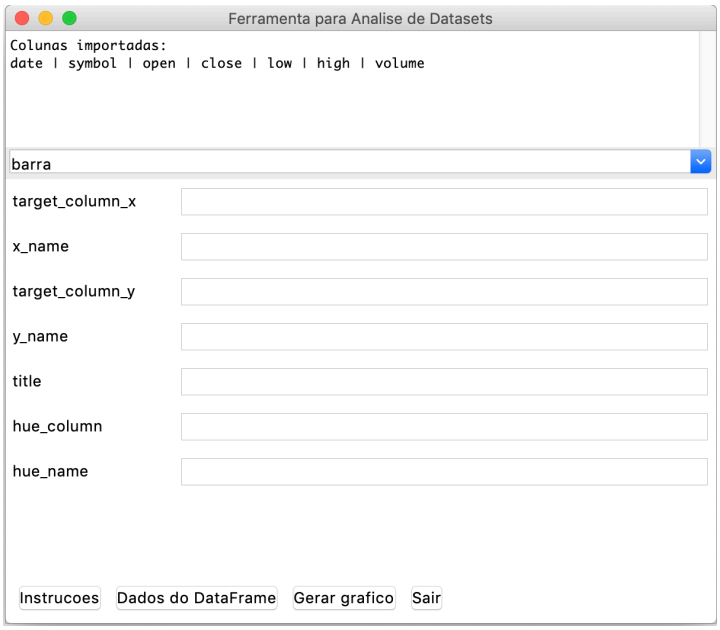
Para algumas das alterações de estilo nos gráficos construídos utilizando a biblioteca Seaborn, como nomenclaturas e formatos dos eixos, tamanhos e impressões para arquivos, foi necessário utilizar funções da biblioteca Matplotlib. Quando é chamada a função gráfica do Seaborn, a mesma retorna um objeto do tipo *axes* da biblioteca Matplotlib. Nestes casos, tais alterações são realizadas recorrendo às bibliotecas *matplotlib.pyplot* e *matplotlib.dates*, sendo a primeira para impressão de arquivos e configurações dos eixos e a segunda para alterações nos eixos que envolvem data.

Para importação dos dados e funções relacionadas diretamente à manipulação deles, foi criado o arquivo *utils.py*. Nele, encontra-se a função responsável pela importação da base de dados escolhida pelo usuário. De modo a não exigir a escrita do caminho completo do arquivo, a função foi construída de modo a abrir uma janela gráfica para escolha do arquivo *.csv* desejado e realizar a leitura dele através da biblioteca Pandas (função *read_csv()*). Com isso, a função retorna o objeto *Dataframe* importado.

Assim, um outro arquivo, *main.py* (APÊNDICE B), foi implementado como a interface gráfica simples com o usuário, para que fosse possível realizar os testes da ferramenta. Definido em *utils.py*, este *script* chama o leitor de arquivos do sistema, para que o arquivo *.csv* seja escolhido. Então, é apresentada uma tela contendo as colunas do *dataset* e as opções de gráficos a serem plotadas (FIGURA 8). Os parâmetros para os gráficos, como colunas e nomes dos eixos, são coletados por caixas de texto. O botão para geração do gráfico chama a função do *data_visualization_tool.py*, a qual cria a visualização solicitada pelo usuário e a exporta em formato *png*.

Além do botão para geração da visualização gráfica, outros três botões são definidos: “instruções”, “dados do *dataframe*” e “sair”. O primeiro apresenta as instruções de utilização, falando sobre cada possível input do usuário na tela. O segundo recorre à *data_visualization_tool.py*, chamando a função para descrição e apresentação do cabeçalho do conjunto de dados. O último botão encerra a execução da ferramenta.

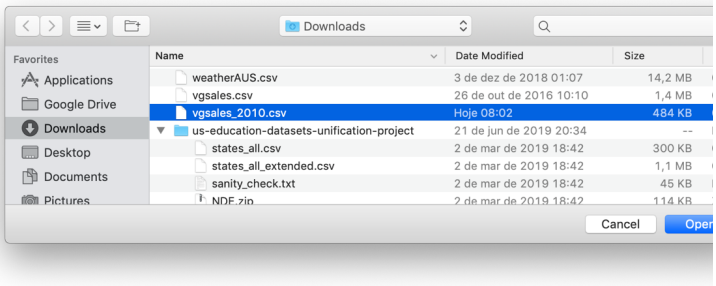
FIGURA 8 - *Interface gráfica da ferramenta desenvolvida*



8 RESULTADOS

A ferramenta funciona de forma simples para o usuário. O primeiro passo é escolher o arquivo *.csv* com os dados a serem utilizados para a construção do gráfico, como mostra a FIGURA 9. Para este exemplo de execução da ferramenta, serão utilizados o número de vendas de jogos eletrônicos com data de lançamento entre os anos de 1980 e 2016. Esse *dataset* foi disponibilizado na plataforma Kaggle por Smith (2019)[22]. Os títulos foram filtrados de modo a ser considerados apenas aqueles lançados entre 2010 e 2016, a fim de facilitar a visualização dos resultados da ferramenta.

***FIGURA 9** - Tela de importação de dados da ferramenta*

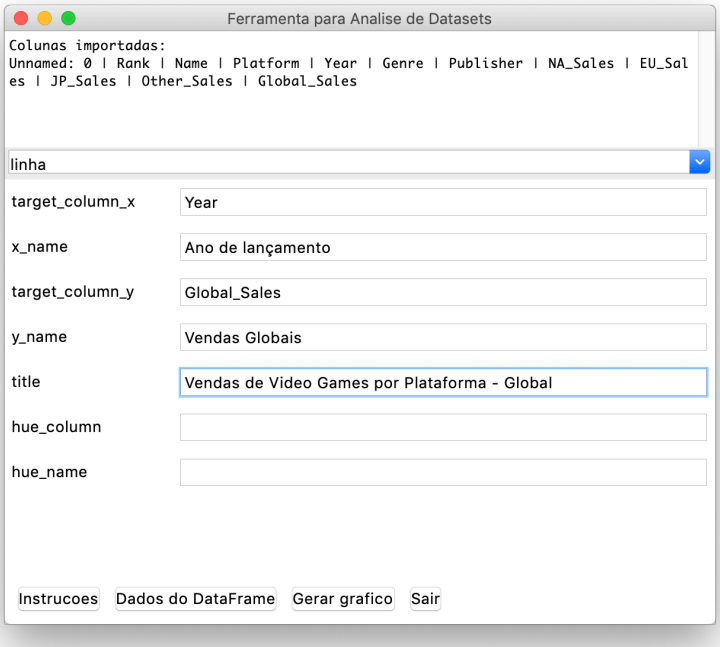


Com a escolha e confirmação do *dataset* a ser utilizado, a tela principal da ferramenta será aberta. Nela, escolheu-se utilizar um gráfico de linha para mostrar a quantidade de vendas globais ao longo dos anos. Assim, foram preenchidos os parâmetros de modo a construir essa visualização, conforme mostrado na

[22] Disponível no link: https://www.kaggle.com/gregorut/videogamesales

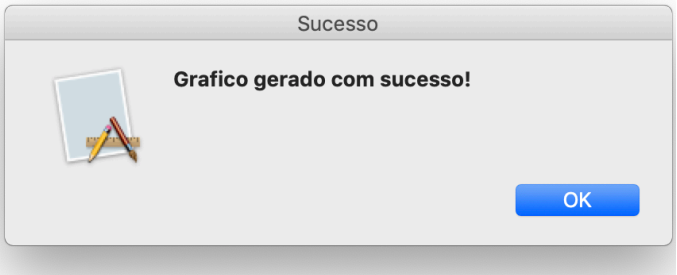
FIGURA **10**.

FIGURA 10 *- Construção do gráfico de linha*



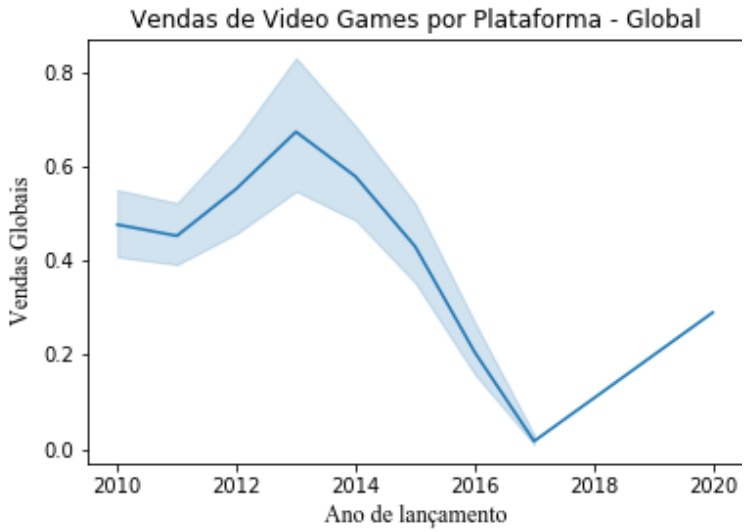
Com os parâmetros inseridos nos campos da ferramenta, basta clicar em “Gerar gráfico”. A interface irá interpretar os *inputs* e solicitar à ferramenta a visualização desejada. Caso a ferramenta consiga gerar o gráfico, é mostrada uma mensagem de sucesso (FIGURA 11).

FIGURA 11 *– Mensagem de sucesso na geração do gráfico*



A visualização gerada é salva na pasta do código da ferramenta, em "*results*" (esta pasta é criada automaticamente, caso não exista), em formato *.png*. O resultado do gráfico solicitado na FIGURA 10 é mostrado abaixo, na FIGURA 12. Pode ser visto um valor em 2020 (referente às vendas de um jogo da plataforma DS, da Nintendo), sendo o mesmo um erro na base.

FIGURA 12 *– Resultado do gráfico de linha gerado pela ferramenta*



De modo a validar a quantidade de títulos com ano de lançamento em 2020, é gerado um histograma através da ferramenta (FIGURA 13). Através dele (

FIGURA **14**), é possível validar a presença de um jogo com lançamento em 2020, validando o gráfico de linha mostrado na FIGURA 12.

FIGURA 13 *– Construção do histograma de ano de lançamento*

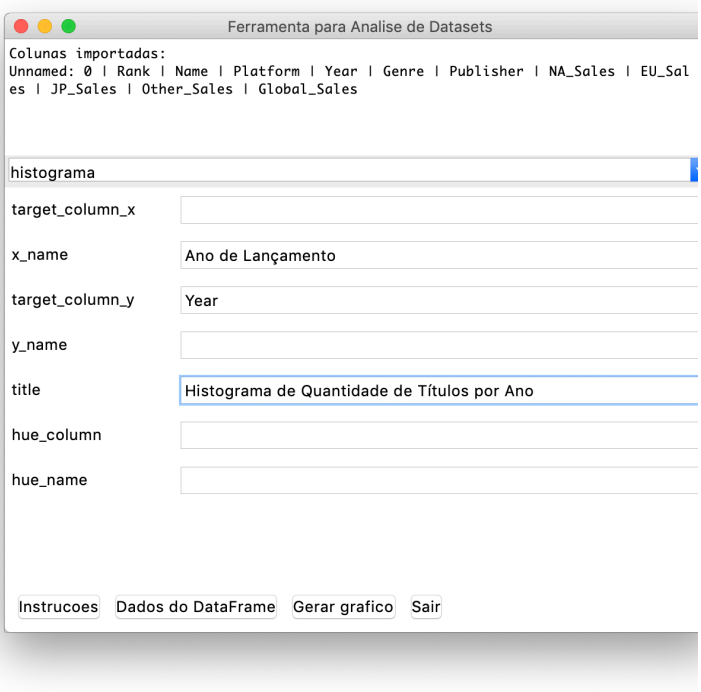
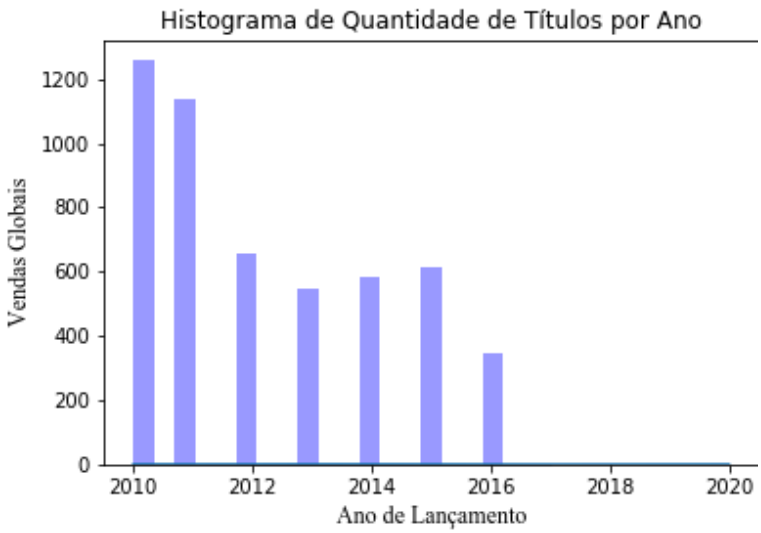


FIGURA 14 *– Resultado do histograma de ano de lançamento*



9 CONCLUSÕES

Foi possível realizar a construção de uma ferramenta que importa os *datasets* e permite realizar a análise exploratória dos dados. A ferramenta é de mais fácil utilização que soluções de manipulação de planilhas disponíveis no mercado e, também, mais segura, já que o usuário não manipula os dados puros, em sua fonte. Ainda, foram contemplados o *box* e o *violin plots*, duas visualizações muito úteis para análise de dados.

Apesar da biblioteca Seaborn ser bastante robusta e fácil de utilizar, algumas funções de estilo tiveram de ser realizadas através da Matplotlib. Ou seja, somente a biblioteca Seaborn não foi suficiente para a construção visual dos gráficos gerados pela ferramenta.

Com isso, os resultados do trabalho foram satisfatórios. Entretanto, mesmo que a ferramenta facilite o trabalho de análise exploratória e rápido diagnóstico do estado de um *dataset*, algumas visualizações exigem tratamentos específicos nos dados. Sem tais manipulações, não é possível realizar as análises no conjunto de dados ou gerar as visualizações desejadas, nem é fácil realizar alterações visuais nos gráficos. Tal liberdade é difícil de ser implementada de modo que o usuário não interaja com uma linguagem de programação como o Python ou algo parecido.

10 BIBLIOGRAFIA

Anderson, David; Sweeney, Dennis J.; Williams, Thomas A.; **Essentials of Modern Business Statistics with Microsoft Office Excel**. Estados Unidos: South-Western, Cengage Learning, 2011.

BERNARD MARR. Forbes Magazine. **How Much Data Do We Create Every Day?:** The Mind-Blowing Stats Everyone Should Read. Disponível em: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#68fdc6a460ba>. Acesso em: 28 maio 2019.

BRUCE, Peter C.; BRUCE, Andrew. **Practical Statistics for Data Scientists:** 50 Essential Concepts. Estados Unidos: O'Reilly Media, 2017.

CONSTANCE L. HAYS. The New York Times. **What Wal-Mart Knows About Customers' Habits.** Disponível em: <https://www.nytimes.com/2004/11/14/business/yourmoney/what-walmart-knows-about-customers-habits.html>. Acesso em: 22 jun. 2019.

FARRELL, Damien. **DataExplore**: An Application for General Data Analysis in Research and Education. **Journal Of Open Research Software**, [s.l.], v. 4, p.1-8, 22 mar. 2016. Ubiquity Press, Ltd.. http://dx.doi.org/10.5334/jors.94.

FREUND, John E.; SIMON, Gary A.. **Estatística Aplicada**: Economia, Administração e Contabilidade. 9a Edição, Bookman, 404 pg, ISBN 85-7307-531-7. 2002

HINTZE, Herry L.; NELSON, Ray D.. *Violin Plots: A Box Plot-Density Trace Synergism*. **The American Statistician**, Estados Unidos, v. 52, n. 2, p.181-184, maio 1998.

HUNTER, John et al. ***Matplotlib Homepage.*** *Disponível em: <https://matplotlib.org/index.html>. Acesso em: 18 maio 2019.*

KDNUGGETS. **2019 KDnuggets Poll:** What software you used for Analytics, Data Mining, Data Science, Machine Learning projects in the past 12 months?. Disponível em: <https://www.kdnuggets.com/2019/05/new-poll-software-analytics-data-science-machine-learning.html>. Acesso em: 28 maio 2019.

MITCHELL, Tom. **Machine Learning.** Estados Unidos: Mcgraw-hill, 1997.

PROVOST, Foster; FAWCETT, Tom. ***Data Science for Business.*** Estados Unidos: O'Reilly, 2013.

PYTHON. **Python's Homepage.** Disponível em: <https://www.python.org>. Acesso em: 9 jun. 2019.

RIBECCA, Severino. ***The Data Visualisation Catalogue.*** Disponível em*: <https://datavizcatalogue.com>*. Acesso em: 15 jun. 2019.

SELTMAN, Howard J. ***Experimental design and analysis.*** Estados Unidos: Carnegie Mellon University, 2018.

SOUKUP, Tom; DAVIDSON, Ian. ***Visual Data Mining:*** *Techniques and Tools for Data Visualization and Mining.* Estados Unidos: Wiley Publishing, Inc., 2002.

TIOBE. **TIOBE index for May 2019.** Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 01 jun. 2019.

Tukey, J. ***Exploratory data analysis.*** Londres: Pearson, 1977.

WASKOM, Michael. **Seaborn Homepage.** Disponível em: <https://seaborn.pydata.org/#>. Acesso em: 18 maio 2019.

APÊNDICE A – CÓDIGO PYTHON DA FERRAMENTA DE VISUALIZAÇÃO

Esta seção traz o código construído neste trabalho para a ferramenta de visualização e análise exploratória. Nele, constam as funções para criação dos gráficos (utilizando a biblioteca Seaborn), uma função única para exportação da visualização para o formato *.png* e uma função para criação de uma pasta de resultados, caso a mesma não exista.

```
"""
Created on Sat Jun 22 18:19:29 2019

@author: gabrielcaschera
"""
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import os

def verify_results_folder():
    '''

        Funcao para verificacao da existencia da pasta; se nao existir,
cria

    '''

    if not os.path.exists(os.getcwd()+'/results'):
        try:
            os.makedirs(os.getcwd()+'/results')
            return True
        except:
            return False
    else:
        return True

def describe_df(df):
    '''

    Funcao que retorna a descricao e as informacoes do dataset

    '''

    return df.describe(), df.head(), df.info()

def print_plot_png(sns_plot, title, x_name, y_name, hue_name=None):
    '''

    Funcao que altera os nomes dos eixos e imprime o grafico para png

    '''
    # Define nome do plot com base nas variaveis
    plt_file_name = 'plt'
```

```
    if x_name:
        plt_file_name = plt_file_name + '_' + x_name
    if y_name:
        plt_file_name = plt_file_name + '_' + y_name
    # Se possuir legenda para sub grupos, a mesma e colocada
    if hue_name:
        sns_plot.legend(title=hue_name)
        plt_file_name = plt_file_name + '_' + hue_name

    # Alteracao do titulo do grafico
    if title:
        sns_plot.set_title(label=title)

     # Alteracao dos titulos do eixos
    if y_name:
        sns_plot.set_ylabel(y_name, fontsize=12, fontname='Times New
Roman')
    if x_name:
        sns_plot.set_xlabel(x_name, fontsize=12, fontname='Times New
Roman')

    # Tenta salvar a figura: se conseguir, True; senao, False
    sns_plot_fig = sns_plot.get_figure()
    try:
        sns_plot_fig.savefig(os.getcwd()+'/results/'+plt_file_name+'.png')
        print('Sucesso')
        return True
    except:
        print('Erro para geracao da imagem.')
        return False

def generate_bar(df, target_column_y, y_name, title, target_column_x =
None, x_name = None, palette='GnBu_d'):
    '''

    Funcao que imprime o grafico de barras num arquivo .png e retorna True
em caso de sucesso

    '''
    # Geracao do grafico atraves da biblioteca Seaborn
    # Se nao for dado um valor para o eixo x, e gerado o grafico somente
para o y; caso contrario, imprime o x
    if not(target_column_x):
        sns_plot = sns.barplot(y = df[target_column_y],
palette=sns.color_palette(palette))
    else:
        sns_plot = sns.barplot(x = df[target_column_x], y =
df[target_column_y], palette=sns.color_palette("GnBu_d"))
    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name)

def generate_hist(df, target_column, x_name, y_name, title,
palette='blue'):
    '''

    Funcao que imprime o histograma num arquivo .png e retorna True em
caso de sucesso

    '''
```

```
    # Geracao do grafico atraves da biblioteca Seaborn
    sns_plot = sns.distplot(df[target_column], kde=False, rug=False, color
= palette)
    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name)

def generate_line(df, target_column_x, target_column_y, x_name, y_name,
title, hue_column = None, hue_name = None, palette='GnBu_d'):
    '''

    Funcao que imprime o grafico de linhas num arquivo .png e retorna True
em caso de sucesso

    '''
    # Geracao do grafico atraves da biblioteca Seaborn
    if hue_column:
        sns_plot = sns.lineplot(x=target_column_x, y=target_column_y,
hue=hue_column, style=hue_column, data=df,
palette=sns.color_palette(palette, n_colors=df[hue_column].nunique()))
    else:
        sns_plot = sns.lineplot(x=target_column_x, y=target_column_y,
data=df, palette=sns.color_palette(palette, n_colors=2))
    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name, hue_name)

def generate_scatter(df, target_column_x, target_column_y, x_name, y_name,
title, palette='GnBu_d'):
    '''

    Funcao que imprime o grafico de dispersao num arquivo .png e retorna
True em caso de sucesso

    '''
    # Geracao do grafico atraves da biblioteca Seaborn
    sns_plot = sns.scatterplot(x=target_column_x, y=target_column_y,
data=df, palette=sns.color_palette(palette))

    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name)

def generate_boxplot(df, target_column_y, y_name, title, target_column_x =
None, x_name = None, palette='GnBu_d'):
    '''

    Funcao que imprime o box plot num arquivo .png e retorna True em caso
de sucesso

    '''
    # Geracao do grafico atraves da biblioteca Seaborn
    if not(target_column_x):
        sns_plot = sns.boxplot(y=df[target_column_y], width=0.5,
palette=sns.color_palette(palette))
    else:
        sns_plot = sns.boxplot(y=df[target_column_y],
x=df[target_column_x], width=0.5, palette=sns.color_palette(palette))

    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name)
```

```
def generate_violin(df, target_column_y, y_name, title, target_column_x =
None, x_name = None, palette='GnBu_d'):
    '''

    Funcao que imprime o grafico de linhas num arquivo .png e retorna True
em caso de sucesso

    '''
    # Geracao do grafico atraves da biblioteca Seaborn
    if not(target_column_x):
        sns_plot = sns.violinplot(y=df[target_column_y], data = df,
palette=sns.color_palette(palette, n_colors=2))
    else:
        sns_plot = sns.violinplot(y=df[target_column_y],
x=df[target_column_x], data = df, palette=sns.color_palette(palette))

    # Tenta renomear os eixos e salvar o grafico num arquivo png
    return print_plot_png(sns_plot, title, x_name, y_name)
```

APÊNDICE B – CÓDIGO PYTHON DA INTERFACE COM O USUÁRIO

A seguir, é apresentado o código utilizado para geração da interface gráfica. Foram importadas as funções da ferramenta de visualização, as quais são as responsáveis pela geração dos gráficos. Deste modo, a interface é totalmente desacoplada da ferramenta, permitindo a evolução das duas partes.

```
"""
Created on Sat Apr 20 14:23:05 2019

@author: gabrielcaschera
"""

import utils
import data_visualization_tool as datavis
import tkinter as tk
import tkinter.ttk as ttk
import tkinter.scrolledtext as scrolledtext
from tkinter import messagebox

def create_input_boxes():
    '''

        Funcao que cria as caixas de entrada de texto para input dos
parametros pelo usuario

    '''
    fields = ['target_column_x', 'x_name','target_column_y', 'y_name',
'title', 'hue_column', 'hue_name']
    entries = []
    for field in fields:
        row = tk.Frame(window)
        lab = tk.Label(row, width=15, text=field, anchor='w')
        ent = tk.Entry(row)
        row.pack(side=tk.TOP, fill=tk.X, padx=5, pady=5)
        lab.pack(side=tk.LEFT)
        ent.pack(side=tk.RIGHT, expand=tk.YES, fill=tk.X)
        entries.append((field, ent))
    return entries

def fetch(entries):
    '''

        Funcao que busca os valores dos parametros fornecidos pelo usuario

    '''
    # Resgata os valores inseridos
    for entry in entries:
        field = entry[0]
        text  = entry[1].get()
        fields_values[field] = text

    # Registra os valores no log
    for field, value in fields_values.items():
        print('%s: "%s"' % (field, value))
```

```
    # Verifica qual o grafico solicitado pelo usario e chama a respectiva
funcao da ferramenta
    if fields_values['plot_type'] == 'histograma':
        if datavis.generate_hist(df = df, target_column =
fields_values['target_column_y'], x_name = fields_values['x_name'],
                                 y_name = fields_values['y_name'], title =
fields_values['title']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

    elif fields_values['plot_type'] == 'linha':
        if datavis.generate_line(df=df, target_column_x =
fields_values['target_column_x'], target_column_y =
fields_values['target_column_y'],
                                 x_name = fields_values['x_name'], y_name
= fields_values['y_name'], title = fields_values['title'],
                                 hue_column = fields_values['hue_column'],
hue_name = fields_values['hue_name']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

    elif fields_values['plot_type'] == 'dispersao':
        if datavis.generate_scatter(df = df, target_column_x =
fields_values['target_column_x'], target_column_y =
fields_values['target_column_y'],
                                    x_name = fields_values['x_name'],
y_name = fields_values['y_name'], title = fields_values['title']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

    elif fields_values['plot_type'] == 'boxplot':
        if datavis.generate_boxplot(df = df, target_column_x =
fields_values['target_column_x'], target_column_y =
fields_values['target_column_y'],
                                    x_name = fields_values['x_name'],
y_name = fields_values['y_name'], title = fields_values['title']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

    elif fields_values['plot_type'] == 'violin':
        if datavis.generate_violin(df = df, target_column_x =
fields_values['target_column_x'], target_column_y =
fields_values['target_column_y'],
                                    x_name = fields_values['x_name'],
y_name = fields_values['y_name'], title = fields_values['title']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

    else: # barra
```

```
        if datavis.generate_bar(df = df, target_column_x =
fields_values['target_column_x'], target_column_y =
fields_values['target_column_y'],
                                x_name = fields_values['x_name'],
y_name = fields_values['y_name'], title = fields_values['title']):
            messagebox.showinfo('Sucesso', 'Grafico gerado com sucesso!')
        else:
            messagebox.showerror('Erro de execucao', 'Erro ao gerar o
grafico.')

def instructions():
    '''

        Funcao para mostrar instrucoes de utilizacao da ferramenta

    '''
    # Define mensagem
    msg = 'Utilizacao:\ntarget_column_x: coluna plotada em
x\ntarget_column_y: coluna plotada em y\n'
    msg = msg+'x_name: nome do eixo x\ny_name: nome do eixo y\ntitle:
titulo do grafico\nhue: coluna para clusters'
    msg = msg+'\nhue_name: nome da legenda dos clusters'
    # Cria a janela de mensagem
    messagebox.showinfo('Instrucoes de uso', msg)

def describe_df(df):
    '''

        Funcao para apresentar a descricao do dataframe numa nova janela
        A descricao e chamada da ferramenta

    '''
    # Coleta informacoes atraves da ferramenta
    desc, head, info = datavis.describe_df(df)
    print('Desc: \n', desc, '\nHead:\n', head, '\nInfo:\n', str(info))
    info = 'Desc: \n'+ str(desc)+ '\nHead:\n'+ str(head)+ '\nInfo:\n'+
str(info)

    # Criacao da janela de output de informacoes
    desc_win = tk.Tk()
    desc_win.title('Descricao do DataSet')
    desc_win.geometry('360x300')

    # Apresentacao das informacoes na janela de texto
    st_desc = scrolledtext.ScrolledText(desc_win, width=360,height=300)
    st_desc.insert('insert', info)
    st_desc.pack(fill='both')

    desc_win.mainloop()

def cb_update_field(index):
    '''

        Funcao para atualzar o valor do plot_type atrelado ao combobox

    '''
    fields_values['plot_type'] = combo.get()
    print('Combobox atualizado para: ', combo.get())
```

```
if __name__ == '__main__':

    # Mensagem de inicio no log
    print('Ferramenta para Analise de Datasets')

    # Cria pasta de resultados, caso a mesma nao exista
    if not datavis.verify_results_folder():
        print('Erro na criacao da pasta. Encerrar.')

    # Solicita leitura do dataframe
    df, success = utils.read_csv_file()
    if not success:
        print(utils.close_program('user'))
        messagebox.showwarning('Erro de importação',
utils.close_program('user'))
    else:

        # Estruturas de dados com informacoes basicas
        plots = ['barra', 'histograma', 'linha', 'dispersao', 'boxplot',
'violin']

        fields_values = dict([
                ('plot_type', ''),
                ('target_column_x', ''),
                ('x_name', ''),
                ('target_column_y', ''),
                ('y_name', ''),
                ('title', ''),
                ('hue_column', ''),
                ('hue_name', '')
                ])

        # Transformacao das colunas em texto separado por | para
apresentacao
        column_opts = df.columns[0]
        for col in df.columns:
            if col != df.columns[0]:
                column_opts = column_opts + ' | ' + col

        # Inicializacao e definicoes esteticas da janela
        window = tk.Tk()
        window.title('Ferramenta para Analise de Datasets')
        window.geometry('600x500')

        # Criacao da janela de apresentacao das colunas do dataset
        st = scrolledtext.ScrolledText(window, width=90,height=6)
        st.insert('insert', 'Colunas importadas:\n'+column_opts.strip())
        st.pack(fill='x')

        # Variavel para resgatar valor do ComboBox
        cb_value = tk.StringVar()
        cb_value.trace('w',cb_update_field)

        # Criacao do combobox para selecao do grafico
        combo = ttk.Combobox(window, textvar=cb_value, values=[plots[0],
plots[1], plots[2], plots[3], plots[4], plots[5]])
        combo.bind("<<ComboboxSelected>>", cb_update_field)
        combo.current(0)
        combo.pack(fill=tk.X)
```

```
        # Criacao das caixas de entrada de textos
        entries = create_input_boxes()

        # Botoes para acao
        btns_frm = tk.Frame(window)
        btns_frm.pack(side=tk.BOTTOM, fill=tk.X, padx=5, pady=5)

        # Instrucoes
        btn_help = tk.Button(btns_frm, text = 'Instrucoes',
command=instructions)
        btn_help.pack(side=tk.LEFT, padx=5, pady=5)

        # Descricao do dataframe
        btn_help = tk.Button(btns_frm, text = 'Dados do DataFrame',
command=(lambda df=df: describe_df(df)))
        btn_help.pack(side=tk.LEFT, padx=5, pady=5)

        # Geracao do grafico
        btn_plot = tk.Button(btns_frm, text = 'Gerar grafico',
command=(lambda e=entries: fetch(e)))
        btn_plot.pack(side=tk.LEFT, padx=5, pady=5)

        # Finalizar ferramenta
        btn_quit = tk.Button(btns_frm, text='Sair', command=window.quit)
        btn_quit.pack(side=tk.LEFT, padx=5, pady=5)

        window.mainloop()

    print('Utilizacao encerrada')
```